4.6 Technico Super BASIC

4.6.1 Loading

Super BASIC is supplied in several forms - tape, paper tape, disk, etc. Instructions supplied with your object program will describe how to load Super BASIC.

4.6.2 Operation

To begin Super BASIC, branch to the start address using the monitor's GO command. The start address is included with the loading instructions. When you branch to Super BASIC, it will print the version identification and then prompt for configuration parameters as follows:

> FILES? - enter the maximum number of I/O file buffers required followed by a carriage return. A typical response is two or three. If only a carriage return is entered, BASIC will use the default of two.
>
> BUFFER SIZE? - enter the maximum number of bytes for each file buffer followed by a carriage return. BASIC will reserve this amount of RAM memory for each file. A typical response is 132. If you SAVE a program in source form, the buffer must be large enough for the longest line of source text. If not, an error is generated during the SAVE. Therefore, a response of less than 132 may limit the SAVE command. If only a carriage return is entered, BASIC will use the default buffer length of 80.
>
> SYMBOLS? - enter the maximum number of variable names your program will require followed by a carriage return. BASIC will create a symbol table of sufficient size to hold this number of symbols. Each table entry is four bytes. A typical response is 50. If only a carriage return is entered, BASIC will use the default of 50.

After you have entered the configuration information, BASIC will print the BREAK message and then prompt for a command by printing a decimal point. You are now in the command mode and you can either enter a program or direct execution statement.

To restart Super BASIC without erasing the program in memory, and retaining the current configuration, you can branch to the restart address.

To interrupt program execution, depress BREAK on the terminal. After completing the current BASIC line, BASIC will stop execution and print a prompt. To restart execution, type a CONT command. If BASIC is printing and you wish to BREAK, press and hold BREAK until the printing stops. If your terminal does not have a BREAK, then put the system RAM switch in the RAM position and reset the system. BASIC sets the restart vector to simulate a break.

4.6.3 Errors

During execution, BASIC may print an error message indicating its displeasure. The line number of the errant statement is also printed (RUN mode only). Certain disk file errors, however, will return control to the disk/tape handler (distinguished by the ">" prompt). You can then execute any valid disk command (e.g. CD,DF). To return control to BASIC just press BREAK on the terminal or reset the system as described in paragraph 4.6.2. The error numbers for the BASIC errors are described below.

00 Unrecognized line or a syntax error (improper line construction).

01 Improper BASIC statement - issued by the input routine. Similar to error 00, except that the line contains non BASIC characters.

02 Symbol table overflow - program has more variables than than indicated during program start (see paragraph 4.6.2).

03 Range of a number exceeds BASIC's capacity.

04 Illegal line number (e.g. 12345). Line numbers must be between 1 and 9999.

05 Illegal construction of a number (e.g. 1.7.2).

06 Print Using Error - A list of expressions was encountered but no previous print specifications (e.g. PRINT using "XYZ",A).

07 Print Using Error - numeric specification but string data (e.g. PRINT Using "###", A$).

08 Print Using Error - Negative number with "$" or "*" format and no trailing "-".

09 Print Using Error - number exceeds specification (e.g. 124 with ##).

10 Print Using Error - string specification, but numeric data (e.g. PRINT Using "LLLL", 12).

11 Continue (CONT) not allowed at this point. If the program was stopped by BREAK, entry of an indirect statement prevents a future CONTinue.

12 Command Error. Illegal command or the command

parameters are invalid.

13 Resequence Error - text cannot be resequenced as specified without creating invalid line numbers (e.g., RESEQ 1000,1000 for 300 lines of text).

14 Text Buffer full. A "CLEAR" will return all available memory to text buffer. If the program still overflows it is too large for the existing memory.

15 Arithmetic overflow. Integer exceeds two bytes or floating point exponent too large or small.

16 Line number expected but not found.

17 Variable name expected but not found.

18 Trace parameter error.

19 Integer expected but not found.

20 Reference to undefined line number.

21 Improper GOSUB/RETURN sequence.

22 Memory overflow during variable allocation.

23 Improper FOR/NEXT sequence.

24 Receptacle String too small (e.g. DIM X$(5): X$="TOOBIG")

25 Receptacle list in error.

26 Data exhausted during READ.

32 Expression stack overflow - expression too complex.

33 Expression improperly formed.

34 Illegal operand mix (e.g., X MOD Y for X,Y single precision).

35 Arithmetic overflow during expression evaluation.

36 Type error. An array expected but not found (e.g., 1(3)), or a variable was referenced (e.g. PRINT I) before it was assigned a value (e.g. I=23).

37 String error.

38 Allocation error during expression evaluation. User space was exhausted.

39 Subscript error (e.g., A(-1,2)).

40 Function call error (e.g., SQR('X').

41 Illegal file number or device or the specified file is already active.

42 File name improperly formed

43 Illegal device code

44 Disk handler error. This can be caused by SAVEing a CHAIN file using a filename that is already on the disk. CHAIN files must be unique files not already on the disk. Source files can duplicate files on the disk in which case that file is overwritten.

45 Chain error

46 Buffer overflow on output or input disk error

4.6.4 Direct/Indirect Execution

If a BASIC statement is preceded by a line number it is an indirect statment and is stored in the text buffer. Any old lines with the same line number will be erased. For example:

```
100 PRINT 'THIS'
```

If not preceded by a line number it is called a direct statement and is immediately executed. An example of direct execution is:

```
FOR I=1,10: PRINT I: NEXT
```

If any indirect statement is used after a BREAK, Super BASIC will not allow a future CONTinue.

4.6.5 EDIT COMMANDS

When Super BASIC is in the command mode you can enter any of the following Edit commands (direct execute statements). Any word in all caps (e.g. DELETE) is typed as a keyword. Words within <-> (e.g. <L1>) indicate the type of item required at that point. Any item within [-] (e.g. [<L1>]) is optional and may be omitted.

DELETE<L1>[,<L2>]

Delete lines <L1> through <L2>, or just <L1>, if <L2> is not entered.

NULL<N>

Set the number of nulls (RUBOUTS) after a carriage return to <n>. If a source program is to be saved on paper tape, set <n> to 10 or more. This command allows a delay after carriage return which is required by some terminals (e.g. TI700 series).

WIDTH<N>

Set the terminal page width to <n>. <n> must be between 40 and 132. During startup, BASIC will set the width to the default value of 72.

AUTO[<L1>[,<L2>]]

Initiate prompting with a line number. <L1> is the first line number used and <L2> is the increment between lines. If <L1> is omitted, the previous <L1> parameter is used. If there is no previous <L1> paramter, 100 is used for <L1>. If <L2> is omitted, the previous <L2> paramter is used. If there is no previous <L2> paramter, then 10 is used for <L2>. To exit the AUTO line number mode, type a carriage return immediately following the line number prompt. This command removes the burden of entering line numbers during program entry thus speeding the entry time.

RENAME<V1>,<V2>

Replace all occurrences of variable name <V1> with variable name <V2>. If the program is now LISTed, you will see that all occurrences of <V1> are listed as <V2>.

NEW

Clear the text and variable buffer and prepare for a new program.

CLEAR

Clear all variable or array definitions, but leave the text buffer undisturbed.

RUN[<L1>]

Begin program execution at the specified line or at the beginning of the program if no line is specified. BASIC will CLEAR before beginning.

SAVE<FILE>[,CHAIN][<,device>]

Store the current program in the text buffer on the specified device. If CHAIN is specified the program is stored in absolute. Refer to the description of CHAIN for more details. The <file> must be a string representation of a valid disk file (e.g. "XYZ", "ABC.SRC/2"). If the file is on the disk already (non CHAIN mode) it will be rewritten or an error message issued (CHAIN mode). If not already on the disk, a new file is allocated. The <device> is an integer which specifies the save device (0=disk, 1=tape). The <device> can be ommitted in which case <device>=0.

LOAD<file>[,CHAIN][,<device>]

Load the program <file> into memory. If it was saved in CHAIN mode, CHAIN must be specified. Refer to the description of CHAIN for more information regarding CHAIN files. <device> is as defined above. A non chain or source LOAD will not erase the program in memory. This feature can be used to merge to programs together. For example, the following sequence merges the programs on files "PRG1" and PRG2" and saves the resulting program on file "PRG3". Note that the text was resequenced to insure that the program line numbers did not overlap.

```
.NEW
.LOAD "PRG1"
.RESEQ 1000,10
.SAVE "PRG1"
.NEW
.LOAD "PRG2"
.RESEQ 2000,10
.LOAD "PRG1"
.SAVE "PRG3"
```

LIST[<L1>[,<L2>]]

List program lines <L1> to <L2>. If <L1> is omitted, use <L1>=1. If <L2> is omitted, use <L2>=<L1>. BREAK can be used to interrupt the print.

RESEQ[<L1>[,<L2>]]

Resequence the lines of text currently in the buffer. If <L1> is omitted, <L1>=100. If <L2> is omitted, <L2>=10. Notice that all GOTO or GOSUB references are also changed to correspond to the new line numbers.

RESEQ 9999

This special form of resequence will resequence the program for high speed execution. Line numbers are replaced by memory pointers. Use of this statement will significantly reduce program execution time. Prior to editing the program, restore normal line numbers with a resequence like RESEQ 100,10. If you list a program in this form, the line numbers will be meaningless.

CONT

Continue program execution that was interrupted by BREAK. A CONT will result in an error if any indirect statements are entered before the CONT command (e.g. 100 I=1).

4.6.6 BASIC STATEMENTS

The allowed Super BASIC statements are summarized below. It is intended as reference material only and it is assumed that the user is already familiar with the concepts of BASIC. Any word in all caps (e.g. GOTO) is typed as a keyword. Lower case words (e.g., <variable>) refer to the type of item required at that point. Items within [-] are optional and may be omitted.

GOTO <line-number>

Branch to the specified line number.

GOSUB <line-number>

Perform a subroutine call to the specified line number.

NOTE: GOTO and GOSUB are executed substantially faster when you RESEQ 9999 prior to RUN. But do not RESEQ 9999 on an untried program!

TRACE <level>

Trace execution according to specified <level>:
0 - No trace
1 - Trace GOSUB/RETURN statements
2 - Trace GOTO/GOSUB/RETURN statements
3 - Trace all statements

END
STOP

Terminate program execution and return to command mode.

DATA <value>, ... ,<value>

Define values for a later READ. If <value> does not begin with +, -, ., or a digit it is assumed to be a string. When the program is listed, BASIC will always put such strings in quotes.

RESTORE

Restore DATA statements. The Next READ will reuse the first DATA statement.

REM <anything>

Insert a comment. <anything> can include any character except carriage return.

FOR <index>=<exp-1> TO <exp-2> [STEP <exp-3>]

FOR is used with NEXT to create program loops. If the step expression is not specified, <exp-3> is one. The FOR is equivalent to the following if the STEP expression is positive.

```
        <index>=<exp-1>
 <top>  .
        .
        .
        <index>=<index>+<exp-3>
        IF <index> <= <exp-1> THEN <top>
```

If the STEP expession is negative, the comparison is reversed. That is, <= becomes >=.

NEXT [<variable>]

End of FOR/NEXT loop. If <variable> is not specified end the inner most loop only.

EXIT [<variable>]

EXIT provides a vehicle to exit a FOR/NEXT loop prior to completion. If you GOTO from within a FOR/NEXT loop, the stack is left set incorrectly since the loop was not terminated. EXIT is like NEXT, but forces the loop to complete. An example of its usage is:

```
100 FOR I=1 TO 10
110 INPUT A(I)
120 IF A(I)<>0 THEN NEXT ELSE EXIT
```

If the EXIT is not at the end of the loop as in the above example, it should be followed by a GOTO to transfer out of the loop.

DEF FN<variable> (<parameters>)=<expression>

Define a user function The variable names used for parameters are global so you must use unique names. The definition must be executed prior to any reference. BASIC will interpret the function like a macro so the output type is determined by the

input parameters. As an example, the following program will first print "AA", and then "2". Thus, the function is both a string and a floating point function.

```
100 DEF FNA(I1)=I1+I1
110 PRINT FNA("A")
120 PRINT FNA(1)
130 END
```

DIM <variable list>

Define dimensioned variables (e.g. A(10,5)). An array may have one or two dimensions only. Dimensioned strings are interpreted as follows: one dimension specified the string length (e.g. A$(75)). Two dimensions specifies a group of strings of a specific length (e.g. A$(5,75) specifies 6 strings (0 to 5) of 76 characters (0 to 75) each.

IF <relation> THEN <stmt> or <line no.>
IF <relation> ELSE <stmt> or <line no.>
IF <relation> THEN <stmt> or <line no.> ELSE <stmt> or <line no.>

Conditional statement. Examples illustrate various possibilities.

```
100 IF X=Y THEN 17
110  IF  (X=Y)  AND  Z=1  THEN  X=2   ELSE  PRINT  1
120 IF X=Y THEN Z=2 THEN K=1
130 IF X=Y THEN END ELSE Z=1 THEN T=2
```

NOTE: THEN can be used to combine multiple statements to an IF or ELSE. The following are <u>not</u> the same:

```
100 IF X=1 THEN Y=Z: Z=3
110 IF X=1 THEN Y=Z THEN Z=3
```

The first will always set Z=3. The second will set Z=3 only when X=1.

ON <expression> GOTO <line-number>, ... ,<line-number>

Branch to the <line-number> selected by the <expression>. A value of one will select the first <line-number>, two the second and so on. If the <expression> is out of range an error is issued.

LET <receptacle>=<expression>

<receptacle>=<expression>

Store the value of the <expression> in the specified <receptacle>. For example: X(3)=1+5 or LET X(3)=I+5. BASIC will accept the functions CRU and LOC as valid receptacles. Refer to the description of these functions for further details.

INPUT[<string>]<receptacle>, ... ,<receptacle>
INPUT[:<file number>]<receptacle>, ... ,<receptacle>

If a string is specified, print it on a new line. Then, print a prompt ("?") and await user inputs. To stop the program, type a carriage return in response to "?". To begin the input line over, type CONTROL-S. If a <file number> is present data will be taken from that file. Refer to file I/O for further details. The input must agree in type with the receptacle. For example, a response of ?1.2 to INPUT A$ will produce an error.

PRINT[USING<string>]<expression>, ... ,<expression>
PRINT:<file number> <expression>, ... ,<expression>

Print the expressions. If a <file number> is specified, the data is transferred to the specified file. Refer to the description of file I/O for further details. If ";" is typed instead of "," no space is left between entries. The USING option allows the user to format the output to his unique requirements. Format characters are:

$	numeric with leading "$" prior to first non-zero digit.
#	numeric digit
*	leading asterisk
.	decimal point
,	comma notation
EEEE	exponent form
!L...L	left justify string
!R...R	right justify string
!C...C	center string
!E...E	left justify string, but extend field to handle the string, if required.
-	minus sign

Examples: If the number is 101.27 it can be printed in several forms as follows:

$$$$.##	$101.27
####.#	101.3 - rounded

```
##.#EEEE    1.0E+02
```

The string "ABCD" can also be printed in several ways:

```
!RRRRRRR       ABCD
!LLLLLLL  ABCD
!CCCCCCC     ABCD
```

READ<receptacle>, ... ,<receptacle>

Read data from the DATA statement into specified receptacles. For example, the following will with I=1, J=2, A$="THIS", and K=1

```
100  DATA 1
110  DATA THIS,2
120  READ I,A$
130  READ J
140  RESTORE
150  READ K
```

CHAIN <file>[,<device>]

Load and begin execution of the specified file on the specified device. If the <device> is not specified, <device>=0 or disk. The <file> must be a file that was saved by SAVE <file>,CHAIN. Also, the configuration of BASIC during the CHAIN must be the same as the configuration of BASIC during the SAVE. If not, errors will result and execution is unpredictable. Refer to paragraph 4.6.2 for instructions regarding the specification of the BASIC configuration.

4.6.8 EXPRESSIONS

4.6.8.1 Variable Names

The format of a variable name is:

<letter>[<digit>][<type>]

The <letter> or <letter><digit> specify the name of the variable. <type> is optional and specifies one of the following types:

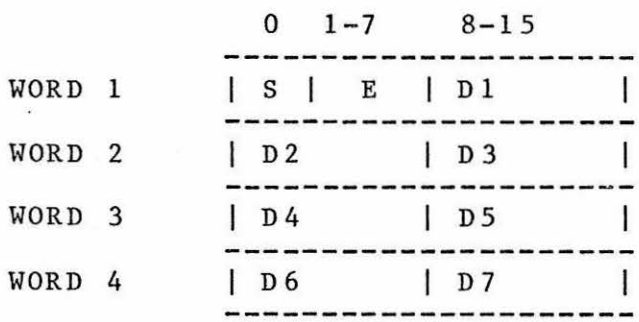
$ - string
# - double precision
% - integer

4.6.8.2 Storage Allocation

Floating point values require 4 bytes, integer values require 2 bytes, double precision values require 8 bytes and strings require 4 bytes plus the length of the string.

Arrays require 4 bytes plus the storage for the elements. That is, X(4,5) would require 4+20*4 for single precision, 4+20*8 for double precision, and 4+20*2 for integer.

Floating point values are strored as follows (word 3 and word 4 are for double precision values).



S=sign of number (0=+, 1=-) E=hexadecimal exponent "e" + 64 D1-D7=hexadecimal mantissa with radix point at left of D1. Each of D1 to D7 are two hex digits.

The value of the number is (16**(E-64))*(.D1 D2 D3 D4 D5 D6 D7). therefore >4110 0000 = (16**(>41->40))*(>.10 0000)

4.6.8.3 Constants

Constants in BASIC can be integer, floating point, or string. An integer constant is a number between -32,768 and +32,767 which does not contain a decimal point or exponent. Sample integer constants are:

```
123
-5678
2
32760
```

A special form of integer constant is a hexadecimal integer. A hexadecimal integer is indicated by preceeding the value by the symbol &. In this case, BASIC will assume that the following digits are in hexadecimal. For example:

```
&123
&A
&FFFE
```

A floating point constant is any number which exceeds the range allowed for integer or contains a decimal or exponent. Typical floating point values are:

```
1.
2.3E-3
222222222
.00004
```

It is important that you fully understand the difference between integer and floating point constants, because BASIC will apply integer rules to any operation like divide. Therefore, 1/2 is zero and 1./2 or 1/2. or 1./2. is .5. As you see, entering a decimal will force BASIC to use floating point rules.

String constants are any sequence of characters except carriage return that are enclosed between double quotes. String constants are most often used with the PRINT statement to label the output. Sample string constants are:

```
"THIS IS A BIG ONE"
"A"
"123"
```

4.6.8.4 Operators

BASIC provides several expression operators described below. The allowed operand types are shown in parens preceeding the operator. For example, (SFI) means the

operator allows string, floating point or integer operands.

(SFI)	+	Add
(FI)	-	Subtract
(FI)	*	Multiply
(FI)	/	Divide (int/int uses int. rules!)
(FI)	**	Exponentiation(int**int uses integer rules!)
(SFI)	<>	Not Equal
(SFI)	<=	Less than or equal
(SFI)	>=	Greater than or equal
(SFI)	<	Less than
(SFI)	>	Greater than
(I)	AND	Logical bit by bit AND
(I)	OR	Logical bit by bit OR
(I)	XOR	Logical bit by bit Exclusive-OR
(I)	MOD	Modulo
(I)	NOT	Ones complement

Examples:

Expression	Result
1+2	3
(1=2) OR (3=3)	-1
(1=2) OR (3=4)	0
6 AND 10	2
13 MOD 3	1
"A"+"B"	"AB"
NOT(3=3)	0

NOTE: integer/integer or integer**integer use integer rules. Therefore, 1/2=0 and 10**10 is to large for an integer answer so an error is given. To eliminate the integer rules, enter the numbers as 1./2 and 10.**10. The decimal in the numbers will force a floating point representation.

4.6.8.5 Intrinsic Functions

The intrinsic functions can be used in any arithmetic expression. The allowed functions are described below.

CRU(I[,J])

Perform 9900 CRU I/O. If used in an expression, return J bits at CRU bit I. If J not present J=1. If used on left of replace (e.g. CRU (10,3) = 7) output J bits starting at CRU bit I.

LOC (I)

If used in an expression, return the contents of memory word I. I should be an even address. If used on left of replace (e.g. LOC (10) = 5), store into the specified location.

SEG$(S,I)

Return the first I characters of string S. The SEG$ can also select any subsequence of a string. If we want characters 14 to 17 (a total of 4 characters) of A$, just write:

```
SEG$(A$(14),4)
```

Since A$(14) refers to the substring of A$ which starts at character 14 (not character 14 alone as you might expect). To insert the letter "J" between character 14 and 15 just code the following (remember the characters start with zero).

```
A$=SEG$(A$,15)+"J"+A$(15)
```

SEG$(A$,15) selects characters 0 to 14, A$(15) selects characters 15 to the end.

INSTR([I,]<string-1>,<string-2>)

Return the index to <string-1> of the first occurance of <string-2>. If I is specified, begin the search at that character position. If I is not specified, begin the search at character zero of <string-1>. If <string-2> is not contained within <string-1> return -1. Examples:

```
0=INSTR("ABCABC","ABC")
3=INSTR(1,"ABCABC","ABC")
-1=INSTR("ABCABC","X")
4=INSTR(A$,B$), assuming A$="THISTHAT" and B$="THAT"
```

LINE$(I)

Return a single line of text from the specified file I. If the file number is zero, assume that the input is from the terminal. All characters typed or read, including the carriage return, are returned as a string. This function can be used to perform unformatted input. As mentioned earlier, a response of ?12 to the statement INPUT A$ will result in an error message. But, the same response to LINE$(0) is acceptable. To insure maximum

flexibility, the LINE$ function does not issue an input prompt or a carriage return. It is the user program's responsibility to prompt for input.

USR(I,[parameter list])

Call a user routine written in assembly language. I is the address of a workspace pointer, program counter pair in memory. When BASIC encounters the USR call, it will evaluate all of the expressions in the parameter list and will then perform a BLWP to address I. The parameters are passed to the user routine as a sequence of five word packets. Word one of each packet determine what is contained in the packet as described below. To return a result, the user routine stores its answer in a similar five word packet. The address of the first parameter packet is in BASIC's R0 (or user *R13). The address of the result packet is in BASIC's R1 (or user @2(R13)). The number of parameter packets is in BASIC's R2 (or user @4(R13)). The assembly language routine should perform all necessary computations and then return via an RTWP. It cannot destroy R13, R14, R15 or the RTWP will not function properly. All other user registers are available. The user routine must be stored outside of BASIC memory. To reserve memory for a user routine, simply modify the beginning or ending of memory pointers to reserve a block of memory from BASIC. The address of the beginning or ending of memory pointers are indicated in the loading instructions for BASIC.

PARAMETER PACKETS:

Word 1=0 single precision floating point parameter
Word 2-3=value in floating point notation
Word 4-5=unused

Word 1=1 integer parameter
Word 2=value in integer form
Word 3-5=unused

Word 1=2 string parameter
Word 2=length of the string in bytes
Word 3=pointer to the first character of string
Word 4-5=normally unused, but may be the string itself if Word 3 points to this location,

and the length is four or less.

Word 1=3 double precision parameter
Word2-5=value in double precision floating
point representation.

ABS(I)

Return the absolute value of I.

ASC(S)

Return the ASCII numeric equivalent of the first character of the string S. For example, ASC ("A") is 65 or 41 (hexadecimal)

ATN(I)

Return the ArcTangent of I.

COT(I)

Return the CoTangent of I.

COS(I)

Return the CoSine of I.

EXP(I)

Return E**I.

INT(I)

Return the integer part of I.

FRE(dummy)

Return the number of bytes of free space.

CHR$(I)

Return a one character string containing the the character whose ASCII code is the integer I. For example "A"=CHR$(65). This can be used to create strings with embedded control codes.

Return the actual length of string S.

LOG10(I)

Return the base 10 or common logarithm of I.

LOG(I)

Return the base e or natural logarithm of I.

HEX$(I)

Return a four character string containing the hexadecimal equivalent of integer I. For example: "1234" = HEX$(4660)

HEX(S)

Return the integer value of the hexadecimal string S. The string S is not checked for valid hexadecimal digits. For example: 4660 = HEX ("1234").

POS(I)

Returns the current position of the terminal print head.

RND(I)

Return a random number 0<=X<1 according to I as follows:
I<0 Reset random sequence
I=0 Return last random number
I>0 Return a new random number

SHIFT(I,N)

Return the shifted value of integer I. I is shifted left N bit positions for positive N and right N bit positions for negative N. For example: 4=SHIFT (1,2) 1=SHIFT (4,-2)

SIN(I)

Return the Sine of I.

SGN(I)

Return an integer value representing the sign of I as follows:
I<0 Return -1

I=0 Return 0
I>0 Return +1

SQR(I)

Return the square root of I.

STR$(I)

Return a character string which represents the value of I. For example "-1" = STR$(-1)

TAB(I)

Tab (move over) to column I on the terminal. TAB should be used only with PRINT statement. An error will be issued if past column I.

TAN(I)

Return the tangent of I.

VAL(S)

Return the numeric value of string S. For example: -1 = VAL ("-1").

4.6.9 File I/O

Two statements are used to control the file I/O in BASIC - OPEN and CLOSE. Prior to an INPUT:<file> or PRINT:<file>, the <file> must be OPENed. The syntax of each is described below.

OPEN <file no>,<name>[,<device>[,<rec len>[,<tot len>]]]

This statement will open the <file no> whose name is <name> on the device specified by the integer value <device>. If the <device> is not specified, <device>=0. <rec len> is the number of characters per record. If not specified, BASIC will assume that each record will be the length of the file buffer. If the records are variable length, this is the maximum length required. <tot len> is the total number of records to be allocated for this file. If the file is not already on the disk, BASIC will allocate enough space for this number of records. If the file is already on the disk, the <tot len> is ignored. OPEN will also rewind the file to its logical beginning. It is the users responsibility not to read beyond the end of file. Typically this is avoided by writing some marker character on the end of the file.

CLOSE [<file no>, ... ,<file no>]

Close all of the specified files. If no files are specified, all user files are closed. To rewind a file, simply CLOSE then reOPEN it.