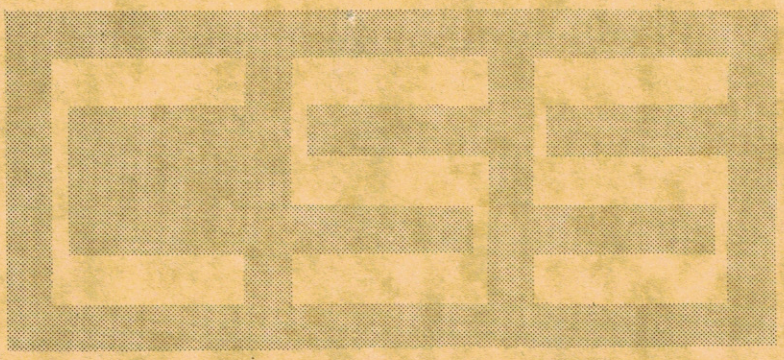


COMPUTERWARE
SOFTWARE
SERVICES

COMPUTERWARE
BASIC

A 6800
STANDARD

WE SELL CAPABILITIES . . . .

TABLE OF CONTENTS

BASIC LANGUAGE SUMMARY

INTRODUCTION:

Computerware's SUPER BASIC and RANDOM BASIC were written to conform closely to the proposed ANSI standard, thus allowing the user to run standard BASIC programs with few, if any, changes. In addition, many new commands have been added to make programming easier, and to keep your source code to a minimum. All commands, statements and functions can now be abbreviated and the line input buffer has been expanded to 128 characters (90 characters for cassette/prom) to allow the use of longer text strings and multiple statement lines. CSS BASIC supports many transcendental functions, allows programs and data to be saved on disk, and supports both RANDOM and sequential data files.

Complete documentation for input and output character routines is provided so as to allow easy adaptation for special I/O features.

LICENSE:

WARRANTEE INFORMATION:

The license to use CSS BASIC is sold AS IS without warrantee. This warrantee is in lieu of all other warrantees expressed or implied. Computerware does not warrant the suitability of BASIC for any particular user application and will not be responsible for damages incidental to its use in a user system.

REGISTRATION:

Computerware wants you, the user to be satisified with our products. To help achieve this goal, we ask that you fill out the enclosed ORIGINAL registration form. If a problem is found in the software, we can then communicate with you concerning corrections and enhancements. If you find a problem - please document it - send to Computerware the disk or tape(s) with the software in question, and we will make every attempt to resolve the problem/question. The materials you sent will be returned to you with an explanation of what we found.

MODES OF EXECUTION:

BASIC has two modes of execution - the immediate (or direct) mode and the program mode. In the program mode, BASIC executes a set of instructions that has been stored prior to execution. In the immediate mode, BASIC executes commands at the time they are entered from the terminal.

The BASIC interpreter determines whether a statement is intended for immediate execution or for storage as part of the program solely on the basis of whether or not the statement was entered with a line number. Statements having line numbers are stored for later execution; those without line numbers are executed immediately. Thus the line:

```
10 PRINT "CSS BASIC"
```

will produce no response at the terminal until the program is executed. The line:

```
PRINT "CSS BASIC"
```

however, causes the terminal to respond immediately with:

```
CSS BASIC
```

By using statements without line numbers BASIC can be used as a sophisticated calculator. For example,

```
PRINT (17*2.83)*(7/4)
```

will cause BASIC to immediately respond with:

```
89.14
```

Another use for immediate mode execution is as an aid in program developement and debugging. Through the use of direct statement execution, program variables can be read or altered, and the program flow may be directly controlled.

INSTRUCTION ABBREVIATION:

All instructions can be abbreviated by using only as many of the first letters as required to provide uniqueness and then a period (.). The most often used instructions (PRINT, LIST, RUN, FOR, NEXT, GOTO, INPUT, THEN, etc.) only require their first letter and a period. Abbreviated instructions that occur right after the line number will be expanded to their full spelling, but all others within the program will remain abbreviated and will be processed faster than the full spelling format.

PROGRAM STATEMENTS:

A BASIC program is made of a series of program lines. Each line must begin with a line number followed by one or more BASIC statements and terminated with a carriage return. The following are several rules that must be followed in writing a BASIC program:

1. Every line must have a line number ranging between 1 and 9999. Line number 0 may not be used.

2. Line numbers are used by BASIC to arrange the program lines sequentially. The program will be executed in order of increasing line number regardless of the order in which they are entered.

3. A line number may be used only once in any given program.

4. A previously entered line may be changed by simply re-entering the same line number along with the corrected line. Typing a line number followed immediately by a carriage return deletes that line.

5. Program lines need not be entered in numerical order because BASIC will automatically put them in ascending order.

6. A line cannot contain more than 128 characters including spaces.

7. Spaces are not processed by BASIC unless they are part of a character string (i.e., enclosed in quotation marks). The use of spaces is optional. The line 10 LET A = 10 is the same as the line 10LETA=10. Spaces make the line more readable, but take longer for the interperter to process and consume more memory. Numbers may not contain imbedded spaces.

8. Multiple statements on a single line are permitted and must be separated by a colon ":". The statements are processed from left to right. For example:

```
10 A=4: B=7: C=A+B: PRINT C
```

is equivalent to:

```
10 A=4
20 B=7
30 C=A+B
40 PRINT C
```


DATA FORMAT:

The range of numbers that can be represented is 1.0 E-99 to 9.99999999 E+99 where E+99 represents 10 to the power 99.

Numbers are retained to an accuracy of nine decimal digits and are internally truncated (last digits dropped) to fit this format. Numbers may be entered and displayed in three formats: integer, decimal, and exponential. For example:

1234 12.34 1234 E-2

NUMERIC VARIABLES:

Variables are represented in a statement by any single alphabetic character or any single alphabetic character followed by a number 0 through 9.

Examples: X, Y, Z, X3, Q8

STRING VARIABLES:

String variables may contain a maximum of 128 characters. A string length command is available which allows the maximum string length to be set at the beginning of the program. If the string length is not explicitly defined using the STRING command, BASIC assumes a string length of 32 characters. Refer to the STRING command description for a detailed description of its use.

Examples of string variables: X$, Y$(7), Z$(3,2)

These string variables are all distinct from numeric variables having the same name. For example, X=902, X$="POLLY", Y(5)=23, and Y$(5)="CRACKERS" are all legal and may appear in the same program.

STRING CONCATENATION:

Strings may be concatenated (joined together) using the concatenation symbol "+". For example:

```
10 X$="CSS"
20 Y$=" BASIC"
30 Z$=X$ + Y$
40 PRINT Z$
```

Will print: CSS BASIC

The total length of the strings to be concatenated may not exceed the maximum string length either set by default or by the use of the STRING command.

CONTROL FUNCTIONS:

Control characters such as CONTROL C or CONTROL X are typed by holding down the CTRL key while typing the specific letter. Control characters are not displayed on the terminal but are accepted by the computer. The control functions may be assigned different characters more suitable to the user's system. Refer to the appendices for specific details.

BREAK -

Typing CONTROL C will cause BASIC to halt its current operation and to respond with "BASIC#". BASIC will then accept additional commands. CONTROL C may be used to stop a LIST operation which is in progress before it is completed, or to halt the execution of a program. If an MP-C card is being used as the terminal interface, the user may have to hit CONTROL C several times before the terminal will respond.

LINE CANCEL -

Typing CONTROL X clears the current contents of the line buffer. If an error is made while making any entry on the terminal, either during program entry or data input during a program, this character can be used to delete the line. The user may then re-enter the line followed by a carriage return. Once a carriage return has been entered, however, the CONTROL X will no longer delete the line.

BACKSPACE -

The CONTROL H (backspace) is used as a single character back space function. When a character has been typed in error, either during program entry or data input during a program, it may be corrected by typing the CONTROL H followed by the entry of the correct character. You may backspace as many character positions as necessary.

BASIC COMMANDS:

It is possible to communicate with the computer in BASIC by typing commands directly on the keyboard of the terminal. Also, many statements can be executed directly using the direct mode of operation described earlier. In addition, there are several commands which may be used by the operator in order to list programs, run programs, save or load programs, etc. When BASIC is ready to receive commands, "BASIC#" will be displayed on the terminal. After each entry, the system will prompt the operator with a "#".

Commands are typed on the terminal without using statement numbers. After the command has been executed, "BASIC#" will be displayed indicating that BASIC is ready to receive another command from the operator.

LIST -

This command displays the lines of the current program on the terminal. The lines are listed in ascending numerical order by line number. A single line may be listed, or all lines within a given range may be listed. For example:

LIST	List the entire program.
LIST 30	List only statement 30.
LIST 30-100	List statements 30 through 100.
LIST #4	List entire program on terminal/printer connected to I/O Port #4.

RUN -

Typing RUN, followed by a carriage return, causes the program which is currently in memory to be executed starting with the lowest line numbered line. The RUN command resets all program parameters and initializes all variables to zero.

CONT -

The CONTinue command causes program execution to be resumed after a STOP statement has been executed. If a program has been interrupted using a "break" (control C) command, execution may be resumed by typing CONT followed by a carriage return. This command should not be used if a program error had been encountered or if the program has been changed. The program parameters are not changed by this command.

NEW -

This command causes the user program area and all variables and pointers to be reset. The effect of this command is to erase all traces of the previous program from memory in preparation for a new program. The CSS identification and BASIC version number will print, followed by "BASIC#".

TRACE -

The TRACE feature is a useful debugging tool. Typing TRACE causes BASIC to display to the terminal the line number of each statement as it is executed. This allows the user to follow the sequence in which the program is being executed. Typing TRACE again returns the system to its normal mode of operation. The TRACE command may be inserted anywhere in the program, or executed in the direct mode.

SIZE -

The SIZE (DISK only) command returns the following information to the control port:

```
AVAIL=(bytes of unused memory in decimal)
PROG=(bytes used for program source)
VAR=(bytes used for variable storage)
     (after program has been run)
```

MON -

This command causes the computer to return to the resident ROM monitor in the computer system. In the case of MIKBUG/SMARTBUG this will output a carriage return, line feed, and the "*" prompt character. If the stack pointer address (stored in $A008 and $A009) is not altered (ie. pressing reset), then typing "G" will restart BASIC leaving the user's BASIC program intact. The MON command may be inserted as a statement within a BASIC program.

DOS -

The DOS (DISK only) command functions identically as MON except that control is return to DOS. To re-enter BASIC, exit DOS and jump to ($0103), BASIC's soft start.

PORT -

The command PORT = N defines the I/O port which will serve as the control port. N can be a constant, a variable, or an expression. All messages, including BASIC's "BASIC#" will be sent to the port assigned by the PORT command and the BASIC program will expect all input from that port.

BEWARE

If a port without a terminal is defined as the control port, you will lose control of your program. Breaks will always be accepted from the control port.

Computerware considers the use of PORT to direct output to a printer to be a POOR programming technique. Use the PRINT #P (P = port no.) for this purpose.

BASIC DISK COMMANDS AND FUNCTIONS:

The following commands and functions are unique to RANDOM BASIC. They are in a separate catagory from the disk file handling commands and functions because they do not require data files for their use. They may also be used in conjunction with file handling, and in fact, are quite useful in that area.

FLIST -

The FLIST (file list) command allows the BASIC user to list the file names stored in the disk directory without exiting to DOS68. The format of this command is: FLIST [#<port number>][,<unit number>]. Typing FLIST alone lists the files stored on disk drive 0. FLIST 2 will list the file directory on disk 2. FLIST #4,1 will list the disk file directory for disk drive 1 on port 4. FLIST will not list the transient commands found in the disk directory.

FDEL -

The FDEL (file delete) command allows the user to delete disk files without exiting back to DOS68 to use the DELETE command. The format of the FDEL command is: FDEL <file list separated by commas>.

FREN -

The FREN (file rename) command functions just as the DOS68 RENAME command does to change the name of a disk file. The command format is: FREN <old file name>,<new file name>. Note: Drive number is allowed only on the 'old file' parameter.

FCHK -

The FCHK function allows the user to determine the absence or presence of a program or data file on the disk without invoking an Error that would stop BASIC. The format of FCHK is: FCHK <unit number:><file/prog name> - since it is a function it must be preceded by a command. The value that FCHK returns is identical to those of the STATUS function described elsewhere in the manual.

```
Examples:   IF FCHK 1:MASTER.FIL <> 0 THEN 2000
   or       LET A = FCHK PAYROL.BAS
```

FREE -

The FREE function returns to the user the number of free sectors on the diskette for the drive specified. The format is as follows: FREE X - where X is either a numeric variable or number in the range of 0 to 3, representing the drive (unit) number. To determine the number of free bytes, multiply the number of sectors by 124. Example: P. (FREE0)*124 will print # bytes free on drive 0.

HOUSEKEEPING COMMANDS:

The following three commands, LINE, DIGITS, and STRING allow the user to define the associated parameters. Once these commands are used, the values assigned remain the same until the commands are used again or BASIC is reload from the disk. LINE and DIGITS can be used more than once during a program; STRING cannot. The default values for these parameters are listed in APPENDIX J. The system returns to the default values whenever the commands NEW, LOAD or CHAIN are executed.

LINE= -

The LINE= command specifies the number of print positions in a line. For example, LINE = 40 defines a line to be 40 characters long. While printing, if the next position is within the last 25% of the line length and a space is printed, a carriage return/line feed will be issued. This is done so that a number or word will not be divided at the end of a print line. To inhibit this function, just set the line length equal to more than 125% of the actual desired line length. Setting LINE=0 disables the line command.

LINE -

The LINE (DISK only) function returns to the user the value of the line length currently in use, either from system default (64) or from the LINE= command.

DIGITS= -

This command is used to specify the number of digits to be printed to the right of the decimal point. Any digits that appear beyond the number specified will be truncated. If there are not enough digits to fill the given length, zeros will be used. DIGITS = 0 resets the system to the floating point mode.

DIGITS -

The DIGITS (DISK only) function returns to the user the value of the digits counter, which was either set by the DIGITS= command or the system default of zero (0).

STRING= -

Executing the command STRING = N will set the maximum string length to N characters. BASIC will now reserve N bytes in memory for all string variables regardless of the actual number of characters which are entered for any particular variable. A maximum of 128 characters is allowed. If the STRING command is not used, BASIC will assume the default value of 32 characters. The STRING command can be used only once during a program and, if used, must appear before reference to any string variable is made. For these reasons the user is advised to place the STRING command at the very beginning of his program in a one-time-only "housekeeping" type routine.

BASE= -

The command BASE=0 will cause array subscripts to begin with the number 0. The command BASE=1 will cause array subscripts to begin with the number 1 which is the default value. Random record numbers (RECNO) also are relative to the BASE= command. To conform to the proposed ANSI standard, the BASE command may be entered in the format: OPTION BASE=.

HOME -

The HOME command will send the home-up and clear-to-end-of-frame sequence to the output device. Appendix F contains the location of where this string is located to allow you to change this to be compatable with your system.

SKIP -

The SKIP command is used to skip X print lines. SKIP X eliminates the need to use multiple PRINT statements. 'X' must be a decimal value between 1 and 255. This command sends BASIC's carriage return line feed sequence to the output device. SKIP may be directed to any I/O port, using the following format: SKIP #P,X where P is the port number and X is the number of lines.

WAIT -

The WAIT command provides the user with an easy way to program wait loops. 'X' is a decimal value between 1 and 255. The length of time represented by the value 1 is dependent upon the the speed of the user's processor (usually between .5 and .9 seconds). A WAIT loop can be interrupted by the BREAK command.

RJUST= -

The value of 'X' in the command RJUST=X is the number of print positions to the left of the decimal point when printing a number. Leading zeros in the field are printed as blanks. To reset RJUST for left justification, set RJUST=0.

RJUST -

The RJUST (DISK only) function returns to the user the value of the right justification counter, which was either set by the RJUST= command or the system default of zero (0).

SAVING AND LOADING BASIC PROGRAMS:

Computerware's BASICS were written to allow programs to be saved and loaded using either disk, Kansas City Standard cassette, or paper tape. The Commands SAVE, LOAD, APPEND, REPLACE, and CHAIN are RANDOM DISK BASIC only commands, while TSAVE, TLOAD, and TPEND are in all of the Basics.

SAVE -

This command is used to save programs to DISK. To save a file, the user can type either SAVE filename or SAVE followed by a carriage return. BASIC will prompt you for the file name if you did not enter it. To save a program on disk unit #1 or #2 simply prefix the name with 1: or 2:.

LOAD -

This command is used to load a program from DISK. The format of this command is the same as for SAVE.

APPEND -

This command also loads programs into memory as does LOAD except that the current contents of memory are not cleared out. The program which is loaded is "appended" (added) to the program already in memory according to the line numbers on the appended program. Variable storage will be affected by the APPEND command - thus it should not be used as a command in a BASIC program but only in the immediate mode.

REPLACE -

The REPLACE command provides the dual function of FDEL and SAVE. The format is the same as SAVE but BASIC will first delete the program file from disk and then save the current memory contents on the disk, using the same program file name. CAUTION: if the program file name is not on the disk an error will occur. Also, be sure that you really want to delete the existing file on disk prior to using REPLACE.

CHAIN -

The CHAIN command allows one BASIC program to call another BASIC program. The called program will automatically begin execution. The format of the CHAIN command is the same as SAVE and LOAD. A practical example of the use of the CHAIN command would be to have a master program call various selected programs which chain back to the master program after execution. Even though variable storage is cleared by the CHAIN command, you may use a variable as the file name (ie. CHAIN A$).

TSAVE -

The TSAVE command allows the user to dump the current BASIC program to cassette tape. The TSAVE command is similiar to the P command of MIKBUG - punch on/off commands are automatically sent to the recording device. The cassette interface can be used in either a manual or automatic motor control mode. If in the manual mode, the recorder should be turned on prior to pressing carriage return, after typing the TSAVE command. TSAVE will output the entire BASIC source buffer onto the recording device. The source buffer in memory is unchanged by the TSAVE command.

TSAVE allows file names to be entered in the following format: TSAVE "FILE NAME" or TSAVE #3 "FILE NAME". The name will be output to the tape ahead of the source program.

TLOAD -

The TLOAD command allows for the entering of previously recorded BASIC programs from cassette tape. The TLOAD command is similiar to the L command of MIKBUG - reader on/off commands are automatically sent, and either manual or automatic motor control can be used on the cassette interface. Typing TLOAD, followed by a carriage return, will transfer the source program from tape to the BASIC source buffer. The buffer is automatically cleared at the beginning of a TLOAD command.

If TLOAD is used with the filename option (TLOAD "FILE NAME"), only a source program with that file name will be loaded. If a file name was not specified, the first source program encountered will be loaded.

TPEND -

The TPEND command is identical to the TLOAD command except that the current BASIC buffer is not cleared. The information provided for the APPEND command is also applicable to TPEND.

The TSAVE, TLOAD, and TPEND commands can all be used to work with any port. If, for example, your cassette recording device is on the ACIA port two, a TSAVE #2 command would be used. If a port number is not specified, the control port is assumed.

NOTE: If your cassette interface does not have automatic motor control, you will have to manually turn the motor on and off when using the above commands.

ARITHMETIC OPERATORS:

BASIC performs addition, subtraction, multiplication, division, and exponentiation. Mathematical expressions are evaluated from left to right using the following operator precedence. Parentheses may be used to override this normal precedence of operators.

1) Exponentiation
2) Negation
3) Multiplication and division
4) Addition and subtraction

The mathematical operators are symbolized as follows:

^ . . Exponentiation (up arrow character)
- . . Negate (unary minus)
* . . Multiplication
/ . . Division
+ . . Addition
- . . Subtraction

No two mathematical operators may appear in sequence, and no operator is ever assumed. For example:

```
10 C = A++B
20 (A+2)(B-3)        are not valid.
```

RELATIONAL OPERATORS:

The following relational operators are used to compare two values. They may be used to compare arithmetic expressions or strings in an IF--THEN statement.

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

Examples:

```
10 IF X = Y THEN 320
20 IF Q > R THEN PRINT Q
30 IF A >= Z THEN GOSUB 100 : GOTO 200
```

FUNCTIONS:

Functions are not to be confused with commands. Functions may be used as expressions or as parts of expressions. Function arguments must be enclosed between parentheses.

PEEK -

PEEK(X) returns the decimal value contained in the memory location specified by the decimal number X. For example, the statement Z=PEEK(194) will assign to Z the value of the contents of memory location 194 (hex C2).

PI -

PI returns the decimal value 3.14159265. It may be used in any arithmetic expression. The PI function has no arguments.

RND -

RND(X) generates a set of uniformly distributed random numbers. There are two ways in which RND can be used. 1) If X=0, then a different random number between 0 and 1 will be returned each time RND(X) is used. 2) If X is not 0, then the same random number will be returned each time RND(X) is used. If no argument is used then X=0 is assumed. To yield random numbers within a range other than 0 to 1 use the following:

```
10 PRINT ((J-I+1)*RND(0)+I)
```

where the range of numbers is to be I through J.

TAB -

TAB(X) moves the print position to the Xth position to the right of the left margin. If the print position is already to the right of the Xth position then a carriage return - line feed will be first printed and the TAB will then position to the print location specified by 'X'. This conforms to the proposed ANSI standard. The left-hand margin is print position #1. For example, to print A$ starting in column 25:

```
220 PRINT TAB(25); A$
```

The following function illustrates how a table of values can be printed with the right-hand column aligned:

```
100 DEF FNA(J) = LEN(STR$(INT(J)))
200 PRINT TAB(25-FNA(J));J
```

INT -

INT(X) returns the greatest integer value which is less than X. For example:

```
INT(8.9) returns 8
INT(-7.2) returns -8
```

ABS -

ABS(X) returns the absolute value of the expression X. For example:

```
ABS(6.27)  returns  6.27
ABS(-6.27) returns  6.27
```

SGN -

SGN(X) returns the sign (+ or -) of X. Examples:

```
SGN(2.3)   returns  1
SGN(-2.3)  returns  -1
SGN(0)     returns  0
SGN(-0)    returns  0
```

POS -

POS returns the present column number of the print head. In fact, POS is the inverse of the TAB function. For example:

```
10 PRINT TAB(1); X;
20 IF POS = 71 THEN PRINT A$
```

LEN -

LEN(X$) returns the number of characters currently in the string represented by X$. Example:

```
LEN("EXAMPLE")  returns  7
```

ASC -

ASC(X$) returns the decimal ASCII numeric value of the first ASCII character within the string. For example:

```
ASC("?")    returns  63
ASC("A")    returns  65
ASC("ABC")  returns  65
```

CHR$ -

CHR$(X) returns a single character string equivalent to the decimal ASCII numeric value of X. CHR$ is the inverse of the ASC function. Example:

```
CHR$(63)  returns  ?
CHR$(65)  returns  A
```

VAL -

VAL(X$) returns the numeric constant equivalent to the string X$. VAL is the inverse of the STR$ function. An error will occur if X$ is non-numeric. Examples:

```
VAL("5E4")  returns  5000
VAL("17.8") returns  17.8
```

STR$ -

STR$(X) returns the string value of a numeric value. STR$ is the inverse of the VAL function. Example:

```
10 LET G = 4918 + 2
20 LET M$ = STR$(G)  -  Now M$ = "4920"
```

LEFT$ -

LEFT(X$,N) returns the string of characters from the left most to the Nth character of X$. For example:

```
10 LET W$ = "BIG BROWN COW"
20 LET A$ = LEFT$(W$,5)  -  Now A$ = "BIG B"
```

RIGHT$ -

RIGHT$(X$,N) returns a string of characters from the Nth position to the left of the right most character through the right most character. For example:

```
10 LET P$ = "BIG BROWN COW"
20 LET A$ = RIGHT$(P$,5)  -  Now A$ = "N COW"
```

MID$ -

MID$(A$,X,Y) returns a string of characters from the string variable A$ beginning with the Xth character from the left and continuing for Y characters from that point. Y is optional. If Y is not specified, then the string returned is from the Xth character to the right of the beginning of the string through the end of the string. For example:

```
10 LET P$ = "RED,BLUE,GREEN"
20 LET A$ = MID$(P$,3,10)
```

The variable A$ now contains the string "D,BLUE,GRE"

IMOD -

IMOD(X,Y) returns the integer remainder of dividing X by Y. (DISK only)

TRANSCENDENTAL FUNCTIONS:

Accuracy for the following mathematical functions is retained to seven significant digits. The accuracy of the seventh digit is not guaranteed. The arguments of SIN, COS, and TAN are in radians rather than degrees.

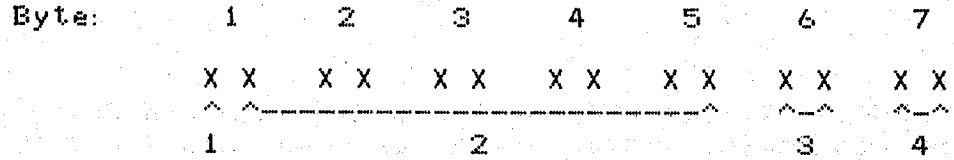
FUNCTION	EXPLANATION
SIN(X)	Returns the sine of X
COS(X)	Returns the cosine of X
TAN(X)	Returns the tangent of X
ATAN(X)	Returns the arctangent of X
LOG(X)	Returns the natural logarithm of X
EXP(X)	Returns 2.718282 (e) to the Xth power. The inverse of LOG(X).
SQR(X)	Returns the square root of X

USER -

The USER function is provided to allow the programmer to jump to a user defined subroutine from a BASIC program. The statement LET A = USER (X) transfers program control to a user written machine language program. Program control branches to the memory location pointed to by memory locations $28 and $29. X is a numeric expression which is then stored in a 7-byte series beginning at a memory location pointed to by $30 and $31. This value may be modified by the user written machine language program to act as a data output from the program or as an indicator that something must be done. The user routine must terminate with a $39 (RTS), thereby transferring control back to the BASIC interperter. Additionally, X is now set equal to the value stored in the seven byte series stored in memory locations pointed to by $30 and $31.

When BASIC is loaded, memory locations $28 and $29 point a location containing an RTS ($39) so that if the user function is called it simply returns control to the BASIC interperter. You must modify memory locations $28 and $29 using POKE or MON command in order to take advantage of the USER function.

Format of the seven byte numeric area:



1 = sign : 2 = BCD number : 3 = not used : 4 = exponent

+ = 0	Standard BCD	Standard BCD
- = 9	10's Compliment	Standard BCD

STATEMENTS:

This section describes statements which can be used in BASIC programs.

POKE -

POKE(X,Y) stores the value of Y at location X (both X and Y are decimal values). This allows the user to modify specific memory locations during program execution. Extreme caution should be taken while using this statement. It is very easy to accidentally modify the BASIC interpreter itself which would cause the program, the data, and the interpreter to be destroyed!

DIM -

The DIM (dimension) statement allows the user to explicitly define the size of an array. An array is a collection of subscripted variables or strings. One and two dimensional arrays are allowed with a maximum size of 256 X 256 elements (when using BASE=0). The DIM statement initializes all elements within the array to zero.

An array can only be "dimensioned" once in a program, but need not be dimensioned at all. If a subscripted variable is encountered prior to dimensioning, a default of 10 elements is established for the array. Only the variables A - Z followed a $ may be dimensioned for string arrays.

Examples:

```
10 DIM X(30)        assigns 30 memory spaces to array X
                    (room for 30 numeric variables)
20 DIM Z(12,3)      dimensions a 12 X 3 array for Z
30 DIM A$(155)      defines a 155 element string array
                    (room for 155 strings each of
                    maximum string length)
```

REM -

The REMark statement is a nonexecutable statement which gives the user the ability to document his program. By including remark statements with the program source the listing becomes more readable. The abbreviation for REM is colon (:).

DEF -

DEF FNA(X) allows the user to define his own functions. The letter "A" can be any letter of the alphabet and the variable X must be a non-subscripted variable. Once defined, the function FNA(X) can be used anywhere in the program just like any other BASIC functions. A function must be defined before a reference is made to it.

DATA AND READ STATEMENTS -

Data and read statements are used together to assign values to variables within a program. Every time a data statement is encountered, the values in the argument field are assigned sequentially to the next available position of a data buffer. All data statements, no matter where they occur in a program, are combined into a continuous list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement. They start with the first data element from the first data statement, then the next element, and so on to the end of the first data statement, and then to the first element of the second data statement, etc. Each time a READ command is encountered, it reads the next data value that has not been assigned to a variable. If a READ is executed and the data statements are out of data, an error is generated.

Numeric and string data may be intermixed. However, they must be used in the appropriate order to assign the data to the appropriate variables. DATA and READ statements may be placed anywhere within the program.

String data need not be enclosed in quotes since the comma acts as the delimiter. However, if the string contains a comma, then it must be enclosed in quotes. For example:

```
10 DATA JANUARY, 17, 1973
20 DATA "SMITH, BOBBY", 5
30 READ M$, D, Y, N$
40 READ A
```

The statements shown above are equivalent to the following:

```
10 LET M$ = "JANUARY"
20 LET D = 17
30 LET Y = 1973
40 LET N$ = "SMITH, BOBBY"
50 LET A = 5
```

RESTORE -

The RESTORE statement causes the data buffer pointer (which is advanced by execution of READ statements) to be reset to the beginning of the data buffer. For example:

```
10 DATA ALVIN, 17, KAREN, 22
20 READ A$, A, B$, B
30 RESTORE
40 READ C$, C
```

is equivalent to:

```
10 LET A$ = "ALVIN" : A = 17
20 LET B$ = "KAREN" : B = 22
30 LET C$ = "ALVIN" : C = 17
```

LET -

The LET statement is used to assign a value to a variable. The use of LET is optional unless the statement is being executed in the immediate mode. In the immediate (or direct) mode, the LET is required. For example, the statement LET B=100 is the same as the statement B=100.

The equal sign does not mean equivalence as in mathematics, but rather the replacement operator. It means: replace the value of the variable name on the left with the value of the expression on the right side of the equal sign. The expression on the right can be a simple numeric value or an expression composed of numerical values, variables, mathematical operators, or functions.

FOR --- NEXT STATEMENTS -

The following is the format of the FOR - NEXT group of statements:

```
10 FOR I = ... TO ... STEP ...
20
30
40 NEXT I
```

The FOR - 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 before exiting from the loop. The variable in the FOR statement (shown above as "I") is initially set to the value of the first expression. Subsequently, the statements following the FOR are executed.

When the NEXT statement is encountered the STEP value is added to the variable and program execution is resumed at the statement following the FOR - TO statement. If the addition of the STEP results in a sum greater than the expression that follows TO, the NEXT instruction executed will be the one following the NEXT statement. If no STEP is specified, the value of 1 is assumed. If the TO value is less than the initial value, the FOR - NEXT loop will be executed only once. For example:

```
10 FOR K=1 TO 3 STEP .5
20 PRINT K;
30 NEXT K
40 PRINT "DONE"
```

This example will print: 1 1.5 2 2.5 3 DONE

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). The same index variable cannot be used in two different loops if the loops are nested together. When the statement after the NEXT statement is executed, the variable is equal to the last value assigned, i.e. the value which caused the loop to stop (generally one greater than the TO value).

STOP -

The STOP statement causes the program to halt execution. BASIC returns to the command mode and prints "BASIC#". The STOP statement differs from the END statement in that it causes BASIC to display the statement number where the program stopped. The program can be restarted by executing a GOTO or a CONT command. The message displayed is STOP XXXX where XXXX is the line number where the program stopped. STOP is often used as a debugging aid. STOP automatically closes all files.

END -

The END statement causes the program to stop executing. BASIC returns to the command mode and prints "BASIC#". END may be used more than once and need not be used at all. When executed, END closes all files.

GOTO -

The GOTO statement is an unconditional branch which directs the program flow to the statement number specified. Note that the statement number may specified as being the contents of a numeric variable or expression.
Examples of GOTO:

```
100 GOTO 10
200 LET L=500 : GOTO L
GOTO 1000 (direct mode execution)
GOTO I*100
```

GOSUB AND RETURN -

The GOSUB statement causes the program to branch to a specified statement number. It is assumed that this statement number is the start of a subroutine. The sequence of statements which make up the subroutine must be terminated with a RETURN statement in order to send the program back to the statement following the original GOSUB statement. Like GOTO, the destination of a GOSUB may be represented as a numeric variable or expression.

A subroutine is a sequence of instructions which need to be executed more than once in a BASIC program. To use such a sequence, a GOSUB instruction is employed. Upon completion of the subroutine, control is returned statement following the GOSUB by execution of the RETURN statement.

A subroutine may use a GOSUB to call another subroutine which in turn may call another subroutine, and so on. This process is referred to as "nesting". Subroutine nesting is limited to eight levels.

Example of the use of GOSUB and RETURN:

```
10 T = 0
20 P = 3.50: GOSUB 100: PRINT C
30 P = 5.00: GOSUB 100: PRINT C
40 PRINT "TOTAL ",T
50 END
100 C = P * 1.06
110 T = T + C
120 RETURN
```

This program would output:

```
3.71
5.30
TOTAL 9.01
```

ON N GOTO OR ON N GOSUB -

This statement causes the program to branch to a specified statement number depending upon the value of N. N may be an integer value or may be an expression. If it is an expression, the expression will be evaluated, rounded to an integer, and the program will then branch to the Nth statement number. For example:

```
220 ON N GOTO 700,350,490,450
```

This means:

```
If N = 0 Fall through to next line
If N = 1 GOTO 700
If N = 2 GOTO 350
If N = 3 GOTO 490
If N = 4 GOTO 450
If N > 4 an error will result
```

IF --- THEN -

The IF statement is used to control program execution depending upon specified conditions. If the relational expression after the IF is true, then the program performs the statement after the THEN. If the conditional after the IF is false, program execution continues with the statement on the next line after the IF --- THEN statement. The statement after THEN may be just a line number, which will cause program execution to GOTO the line specified. All multiple statements on the same line as an IF -- THEN will be executed if the relationship tested true.

For example:

```
10 IF A=5 THEN GOSUB 100: GOTO 230
```

This statement will perform the GOSUB and then will GOTO 230 when A is equal to 5.

The logical operators "AND" and "OR" are not supported in this version of BASIC but may be easily handled using the IF -- THEN statement.

To perform:

```
IF A=B AND C=D THEN 100
```

use the following:

```
IF A=B IF C=D THEN 100
```

To perform the function:

```
IF A=B OR C=D THEN 100
```

use the following:

```
IF A=B THEN 100
IF C=D THEN 100
```

INPUT/OUTPUT STATEMENTS:

Any INPUT or PRINT statement may be followed with an #N where N is the I/O port number (0-7). N may be a constant, variable, or an expression. If no port number is specified, the control port (port #1, or port #2 for SMARTBUG) is assumed. If any expression follows the port number, it must be separated by a comma. For example:

```
730 INPUT #2, A$
220 PRINT #4, X, Y, Z
```

INPUT -

The INPUT statement allows the user to enter either numerical or string data on the terminal during program execution. For example, statement 10 INPUT X allows one numeric value to be entered. The statement 10 INPUT X$ allows one string value, with a length up to that specified by the STRING command, to be entered. The values inputed are assigned to the variables specified in the INPUT statement.

Multiple inputs can be entered by separating them with commas. If the expected number of values are not entered, a " ?" will be generated. The statement 10 INPUT "ENTER VALUE",X will print the message inside the quotes, then prompt with a " ?".

When the program comes to an INPUT statement, a " ?" is displayed on the terminal. The program then waits for the user to respond by entering the requested data followed by a carriage return. If insufficient data is entered, the system then prompts the user with another " ?". If a non-numeric character is entered when a numeric variable is required, the system will prompt the user with "RE-ENTER".

PRINT -

The PRINT statement directs BASIC to print either the value of the expression, literal values, string values, or text strings on the terminal. The various forms of print requests may be combined on a single statement and separated by commas or by semi-colons. If the statement is terminated with a semi-colon or comma, the line feed / carriage return sequence (which is normally issued by BASIC automatically at the end of each print statement) will be suppressed and the next print statement will resume printing on the same line where the last print left off.

Examples:

10 PRINT	Skip a line
20 PRINT A,B,C	Print variables A, B, and C automatically tabbed into 16 character fields
30 PRINT A; B; C	Print A, B, and C with only one space separating them
40 PRINT "FOR SALE"	Print a message
50 PRINT "TOTAL=";A	Print the message followed by the value of variable A
60 PRINT #4,X	Print the value of X on I/O port 4

RANDOM AND SEQUENTIAL DISK FILE HANDLING:

The following section describes the disk file capabilities of Computerware's RANDOM DISK BASIC. Both the random and sequential modes allow the user to create true 'records' in that string and numeric variables may be intermixed in the logical fashion that the data normally appears when being processed by the user.

Sequential files are denoted by the use of file numbers 0 - 9 and Random files use file numbers 10 - 19. The use of random access file commands and functions on sequential files will result in a disk error and termination of the Basic program.

Random access records are fixed length records so that they may be updated in place. This is accomplished by using a 6 byte BCD format for the numeric variables and forcing all strings to the size specified by the STRING= command. Thus it is important for the user to PLAN the record definition for a random file, because once created, all programs accessing the file must have the same string size as the create program had.

OPEN -

The command OPEN #(FLN),(FILE SPEC) is used to open a disk file. The file number, (FLN), is an expression that must evaluate to the range 0 through 19. The file specification, (FILE SPEC), must be string variable or string literal which supplies the file name in standard DOS68 format.

For sequential files, the type of file access (read or write) will be determined by the first usage of the file after opening. Random files are OPENed for both input and output. Before a BASIC program can read input or write output to a file, the file must have previously been opened by the OPEN statement.

Before a RANDOM file can be opened, it must first be created, using the CREATE statement.

Multiple files may be opened with the same OPEN statement by using variables for (FLN) and (FILE SPEC) and repeating the series of statements. For example:

```
10 INPUT "NUMBER OF FILES",F
20 FOR I = 1 TO F
30 INPUT "FILE NAME",F$
40 OPEN #I,F$
50 NEXT I
```

CREATE -

Format: CREATE #(FLN),(FILE SPEC),(# OF RECS),(#BYTES/REC)
The CREATE statement is used for the creation of new random access files. The file number must be a random access file number (10 - 19) and file spec, as described above. The number of records is either a numeric variable or constant specifying the size, in records, that the user wants allocated for the random file. The number of bytes / record is calculated as follows:

Numeric variables - six (6) bytes per variable
String variables - string length + 1

Example:

CREATE#12,CREDIT.HST,200,55

Where STRING=24 and the record is one string (24+1) and five numeric variables (6 * 5) for a record size of 55 and an allocation of a file for 200 records.

A newly created file will be initialized to a 'zero' data state - nulls for strings and zeros for numerics.

CLOSE -

The command CLOSE #(FLN),#(FLN),... is for closing open files. The file number may also be an expression as with OPEN. The specified file number must have previously been opened by an OPEN statement. Example:

100 CLOSE #1,#2

READ -

The statement READ #(FLN),(VARIABLE LIST) is provided to request data be read from a disk file. Input from a file is taken an item at a time - as the program needs it. (VARIABLE LIST) consists of one or more variables, either string or numeric, separated by commas. If the receiving element is a string variable, it will receive the data from the file up to the maximum string length of 128 characters. The line input buffer for a single item from a file is 128 characters.

A string item over 128 characters will be truncated, and if more then 128 characters are contained in a single item of input, the buffer input processing will be terminated.

If the receiving element is a numeric variable, the input is scanned for a break character (a comma or a null) and that portion of the input - up to the break character - is then processed by a validation routine which verifies the number as being a valid numeric value. If the number is invalid, Error #3 (ILLEGAL CHARACTER) will occur.

Random Access Notes: A complete record must be read on a single read statement - this applies to all four random access statements: READ, WRITE, GET and PUT. Auto increment of the record number (RECNO) takes place after the READ is completed. String size must be equal to that specified in the program which created the file and those writing to it.

GET -

GET is a random access statement identical to READ except that auto increment of the record number (RECNO) does NOT occur.

RESTORE -

The statement RESTORE #(FLN),#(FLN),... causes the files associated with the list of file numbers to be repositioned to the beginning of the file. Thus, the data in the file may be reread. Note that this statement functions for files just as the RESTORE dscribed earlier functions for DATA statements. The file number may be that of file which is open for reading (input) or writing (output). On a random file, RESTORE resets the record number (RECNO) to the first record (BASE=0 is 0 - BASE=1 is 1).

```
10 OPEN #1,"PART.MST"    (Quotes are not
20 LET C = 5               required)
40 READ #1,B
50 IF STATUS#1 <> 0 THEN 80
60 PRINT B
70 GOTO 40
80 RESTORE #1
90 LET C=C-1
100 IF C <> 0 THEN 40
110 CLOSE #1
120 END
```

The above program causes File #1 named "PART.MST" to be opened. A counter (C) is set to 5 to keep track of the number of times we go through the file. Data is then read and printed until the status is non-zero (generally 6 for EOF). The file is then restored (rewound). The count, C, is decremented and if the result is not 0 the process is repeated until the count does become 0 in which case the file is closed and the program ended.

This example could be adapted to listing a file just created. The RESTORE after the write sequence will close the file, rewind the file, and open the file for reading.

SCRATCH FILE -

The SCRATCH statement is used to remove an existing file from the current disk directory and then re-enter it into the directory. After the file is re-entered into the directory it is opened for output (writing). The old file is lost from the disk and a new file with the same name is prepared to receive data. A file that has been opened for input (read) cannot be scratched until it is closed and then reopened.

SCRATCH is an ILLEGAL statement for a random file since it is always opened for both input and output - to delete a random file permanently from the disk, use the FDEL command.

```
10 OPEN #1,FILE.RND
20 SCRATCH #1
30 FOR I=1 TO 10
40 WRITE #1,RND(0)
50 NEXT I
60 RESTORE #1
70 READ #1,I
80 IF STATUS #1 = 6 THEN 110
90 PRINT I
100 GOTO 70
110 CLOSE #1
120 END
```

This program opens a file called "FILE.RND" and clears out all existing data with the scratch command. Then it writes ten random numbers to the file, closes it, and then re-opens the file for reading with the RESTORE statement. The random numbers are read and printed until the end of file is encountered (STATUS = 6) at which time the file is closed and the program ends.

WRITE -

The statement WRITE #(FLN),(VARIABLE LIST) allows the writing of the data indicated in the variable list to a disk file. The variable list may contain either string, or numeric variables, separated by commas. An error will occur on the first execution of the WRITE command if the file specified currently exists on the disk. To insure the availability of the file write access, use the SCRATCH command which will prepare the file to receive the output.

For random access files, a complete record must be on a single WRITE statement. The record number (RECNO) will be auto incremented after the completion of the WRITE statement. As mentioned previously, the string size must be the same as that specified in the program which created the random file.

PUT -

PUT is a random access statement identical to WRITE except that auto increment of the record number (RECNO) does NOT occur.

STATUS -

STATUS #(FLN) is a function allowing for monitoring of error status of any specified file number. The status most often used is the end-of-file (EOF) which has a value of 6. The status number is that number returned by the disk file management system. Refer to the BFD-68 SYSTEM MANUAL for other values. It is a good idea to check the file status after a READ at least for end of file.

Random access files do not have an end-of-file status - the functions RSIZE and RNEXT have been provided so that both file size and the record number (RECNO) positioning can be monitored by the user. NOTE: After the last record in a random file has been read or written to, the record number (RECNO) will remain positioned on the last record - it obviously can not be auto-incremented.

RSIZE -

RSIZE is a function that returns to the user the size of a RANDOM file in records. This should be used during random file processing to insure that the record number (RECNO) requested is within the boundaries of the file.

Example: IF RECNO#10 > RSIZE#10 THEN (error message - etc.)

RNEXT -

RNEXT is a function that returns to the user the value of the highest record written in the random file plus one. This is typically the next available record 'slot' to be written into. If you are using a method of loading the random data base other than sequential, the RNEXT function may return a value that would not reflect the next available 'slot'. Assuming a sequential load, the following example shows how to add records to an existing random file without over-writing any existing records:

```
0010 SET RECNO#13 = RNEXT#13     set RECNO to next available
0020 WRITE#13, A$,B$,C$,X,Y,Z    WRITE record/auto increment
```

SET -

SET is used to assign a value to the RECNO fummand (both function and command).

RECNO -

The RECNO fummand serves the dual purpose of informing the user the current position of the record pointer, and allowing the user to move the record pointer to any location within the random file. To assign a value to RECNO, the SET verb is used as follows:

```
SET RECNO#12 = <any numeric expression>
```

RECNO can also be interrogated, printed, etc. as a function:

```
SET RECNO#12 = RECNO#12 + 4
IF RECNO#13 > 55 THEN 2000
PRINT RECNO#17 / PI
```

CASSETTE FILE HANDLING STATEMENTS:

OPEN FILE -

For INPUT - OPENI "FILE NAME"

For OUTPUT - OPENO "FILE NAME"

The FILE NAME may be any name and may have a length up to 60 characters. It may also be represented by a string variable. When opening for output, the file name is written into a special header record at the front of the file.

When opening for input, BASIC begins reading the input tape, looking for a header record with the requested file name in it. BASIC will print on the control terminal the names of any files that it passes while looking for the requested file. NOTE: The BREAK feature of BASIC IS NOT FUNCTIONAL WHILE SEARCHING FOR AN INPUT FILE NAME. If you have given BASIC an incorrect file name, the only way to regain control of the computer system is to press 'RESET' and then re-enter BASIC at HEX ADDRESS $0103.

Below are sample input and output routines using file handling:

```
0010 INPUT "INPUT FILE NAME ",F$
0020 OPENI F$
0030 TREAD A$ : IF EOF <> 0 THEN END
0040 PRINT A$ : GOTO 30
```

```
0010 INPUT "INPUT FILE NAME ",F$
0020 OPENO F$
0030 FOR X = 1 TO 15
0040 TWRITE X
0050 NEXT X
0060 CLOSE
```

CLOSE FILE -

The CLOSE statement is for OUTPUT FILES ONLY. CLOSE will cause the last file control block to be written out to cassette and set an end of file indicator in the file.

EOF -

The EOF function is used to determine whether an input file has more data to be read or is at the 'End Of File'. When EOF is equal to ZERO (0) the file still has data - when EOF is NON-ZERO (<>0) all of the input file has been read.

READ FILE -

TREAD (VARIABLE LIST)

This statement specifies that data is to be 'read' from the cassette file. Input from a file is taken an item at a time - as the program needs it. (VARIABLE LIST) consists of one or more variables, either string or numeric, separated by commas.

If the receiving element is a string variable, it will receive the data from the file up to the maximum string variable length of 90 characters. The line input buffer for a single item from a file is 90 characters.

A string item over 90 characters will be truncated, and if more than 90 characters are contained in a single item of input, the buffer input processing will be terminated.

If the receiving element is a numeric variable, the input is scanned for a break character (a comma or a null) and that portion of the input - up to the break character - is then processed by a validation routine which verifies the number as being a valid numeric variable. If the number is invalid, Error 3 (ILLEGAL CHARACTER) will occur.

WRITE TO FILE -

TWRITE (VARIABLE LIST)

The TWRITE statement allows writing data contained in the variable list. This list may contain either string or numeric variables, separated by commas. String literals (text enclosed in quotes) may also be written out with the TWRITE statement.

ADDITIONAL NOTES:

The TWRITE statement will display the data being written out on the control terminal - on the same line (line feed not given). TREAD will not display on the control terminal (the software echo is turned off). Motor control is essential for proper operation of file handling statements.

The input routine must be a reverse mirror immage of the output routine used to save the data initially. For example, if you write out three variables with a single TWRITE statement (TWRITE L,M,N), you must read back those variables with a single TREAD statement (TREAD A, B,C). The most flexible method is to only write or read a single variable with any TREAD or TWRITE statement.

Although the file handling statements will not give an error message when used in the immediate mode, it is not recommended to to use this mode of operation - nor is it guaranteed to work.....

AUTO RUN:

It is possible for the user to save a copy of BASIC along with the source of a BASIC program and RUN or call, as a transient command, the saved file and have it immediately begin execution. The transfer address for AUTO RUN is $0106, the first loaction to save is $0000, and to determine the last address to save, examine locations $0022 and $0023.

Thus, the steps to make an auto-run file are:

1) Load BASIC and create (or load in) the file to be auto-run.

2) Type DOS to exit to the operating system.

3) Use the resident ROM Monitor (MIKBUG, SMARTBUG, ...)
   to examine locations $0022 and $0023.

4) Restart DOS68 by the warm start entry point

5) Type:
   SAVE,<auto-run file name>,0,<contents of $22, and $23>,106
   (To make the file a transient file, add ",$" following 106).

6) The auto-run file is now ready to run
   RUN,<auto-run file name>
   or
   <auto-run file name> if the ",$" was used

If you append the auto-run file onto a copy of DOS68 that does not have a transfer address (use find to get beginning and ending addresses - then save one without a transfer address), then you can 'LINK' to this new appended file and, upon a cold start ($8020), DOS68; BASIC and your program will load and the program will automatically begin executing. (Whew!)

APPENDIX A

QUICK REFERENCE OF INSTRUCTIONS

COMMANDS
(used in direct mode)

##APPEND	*RJUST
##CHAIN	RUN
*CONT	##SAVE
*DIGITS	*SKIP
*HOME	*STRING
*LINE	TLOAD
*LIST	TPEND
*LJUST	*TRACE OFF
##LOAD	*TRACE ON
*MON	TSAVE
NEW	*WAIT
*PORT	*BASE
##SIZE	

FUNCTIONS

ABS	PEEK
ASC	PI
ATAN	POS
CHR$	RIGHT$
COS	RND
DEF	SGN
EXP	SIN
INT	SQR
LEFT$	STR$
LEN	TAB
LOG	TAN
MID$	VAL
##IMOD	

STATEMENTS
(used in programs)

DATA	#ON GOSUB
#DIM	#ON GOTO
END	#POKE
FOR-NEXT	#PRINT
#GOSUB	READ
#GOTO	REM
#IF-THEN	RESTORE
INPUT	RETURN
#LET	STOP

* commands that can be used within a program

Disk only

\# statements that can be used in the direct mode of execution

RANDOM BASIC COMMANDS						
OPEN	CLOSE	RESTORE	SCRATCH	FLIST	DOS	
READ	WRITE	STATUS	FDEL	FREN	FREP	
FREE	FCHK	SET	RECNO	GET	PUT	
RSIZE	RNEXT	CREATE				

CASSETTE FILE COMMANDS: OPENI OPENO CLOSE TREAD TWRITE EOF

NOTE: All instructions can be abbreviated by using as many of the first letters as required to provide uniqueness and then a period (.). ie. FLIST = FL. - PRINT = P. - etc.

MATHEMATICAL OPERATORS

^	Exponentiation
-	Negate
*	Multiplication
/	Division
+	Addition
-	Subtraction

RELATIONAL OPERATORS

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

PRECEDENCE OF OPERATORS

(1) Exponentiation
(2) Negation
(3) Multiplication or Division
(4) Addition or Subtraction

APPENDIX B

CHARACTER CONVERSION TABLE

ASCII	CNTL	HEX	DEC
NUL	@	00	00
SOH	A	01	01
STX	B	02	02
ETX	C	03	03
EOT	D	04	04
ENQ	E	05	05
ACK	F	06	06
BEL	G	07	07
BS	H	08	08
HT	I	09	09
LF	J	0A	10
VT	K	0B	11
FF	L	0C	12
CR	M	0D	13
SO	N	0E	14
SI	O	0F	15
DLE	P	10	16
DC1	Q	11	17
DC2	R	12	18
DC3	S	13	19
DC4	T	14	20
NAK	U	15	21
SYN	V	16	22
ETB	W	17	23
CAN	X	18	24
EM	Y	19	25
SUB	Z	1A	26
ESC	[	1B	27
FS	\	1C	28
GS	]	1D	29
RS	^	1E	30
US	_	1F	31
SP		20	32
	!	21	33
	"	22	34
	#	23	35
	$	24	36
	%	25	37
	&	26	38
	'	27	39
	(	28	40
	)	29	41
	*	2A	42
	+	2B	43

ASCII	HEX	DEC
,	2C	44
-	2D	45
.	2E	46
/	2F	47
0	30	48
1	31	49
2	32	50
3	33	51
4	34	52
5	35	53
6	36	54
7	37	55
8	38	56
9	39	57
:	3A	58
;	3B	59
<	3C	60
=	3D	61
>	3E	62
?	3F	63
@	40	64
A	41	65
B	42	66
C	43	67
D	44	68
E	45	69
F	46	70
G	47	71
H	48	72
I	49	73
J	4A	74
K	4B	75
L	4C	76
M	4D	77
N	4E	78
O	4F	79
P	50	80
Q	51	81
R	52	82
S	53	83
T	54	84
U	55	85
V	55	85
W	57	87

ASCII	HEX	DEC
X	58	88
Y	59	89
Z	5A	90
[	5B	91
\	5C	92
]	5D	93
^	5E	94
_	5F	95
`	60	96
a	61	97
b	62	98
c	63	99
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121
z	7A	122
	7B	123
	7C	124
	7D	125
	7E	126
DEL	7F	127

APPENDIX C

MEMORY LOCATIONS USED BY BASIC

0020 - 0021	Contains the start of BASIC program (source)
0022 - 0023	Contains the next available memory location after the BASIC program (source)
0024 - 0025	Contains the next available memory location after the BASIC source program and any defined variables
0026 - 0027	Memory limit
0028 - 0029	Contains the pointer for USER
0030 - 0031	Contains the address of the present arithmetic value in use during a USER call
0100	Cold start address
0103	Warm start address
0106	Auto-run address
0109 - 010A	Size of the BASIC interpreter
010B	Number of the control port
010C	I/O definition byte for PROM/CASSETTE BASIC
010D - 010E	Maximum memory size available for BASIC to use.
010F - 0110	Starting address for I/O ports
0111 - 0112	Address of home / clear end-of-frame string
0113 - 0114	Address of ERROR routine (Error # in ACCB)
0115 - 0120	RESERVED for future jump addresses
0121	Line delete control character (CTL X)
0122	Character delete control character (CTL H)
0123	Character delete ECHO character (NULL)
0124	BREAK control character (CTL C)
0125 - 016C	I/O port jump tables

NOTE: The last 256 bytes of memory available are used as a string expression buffer and for the machine stack.

APPENDIX D

ERROR MESSAGES

The following is printed when an error occurs:

```
ERROR # ------
LINE NO.   -  DATA ON LINE
                    ^
                    -----  Pointer to location just past
                           part of line that was processed
                           prior to the error condition.
```

BASIC ERRORS

ERROR	MEANING
01	Maximum variable length exceeded (over 255)
02	Input error
03	Illegal character or variable
04	Missing ending " in print literal
05	Dimension error
06	Illegal arithmetic
07	Line number not found
08	Attempt to divide by 0
09	Maximum subroutine nesting exceeded (over 8 levels)
10	RETURN statement executed without a prior GOSUB
11	Illegal variable
12	Unrecognizable statement - also common disc command error
13	Parenthesis error
14	Memory full
15	Subscript error
16	Too many FOR-NEXT loops active (maximum is 8)
17	NEXT X statement without prior FOR X=...
18	Nesting error in FOR-NEXT
19	Error in READ statement
20	Error in ON statement
21	Input overflow (more than 128 characters on input line)
22	Syntax error in DEF statement
23	FN function error. Either syntax error or FN is not defined.
24	Error in STRING usage, or mixing of numeric and strings
25	String buffer overflow, or string extract too long
26	I/O operation to a port which does not contain an I/O card
27	VAL function error - string starts with a non-numeric
28	Cannot take LOG of a negative number
29	Error message error

DISK FILE ERRORS

ERROR	MEANING
30	File number is not in range of 0 through 19
31	Unable to open file for write
32	Attempt to write to file not open for write
33	Unable to open file for read
34	Attempt to read from a file not open for read
35	Attempt to read data beyond end of file
36	Specified file failed to close
37	Specified file failed to delete
38	Disk Directory error
39	Disk Unit (drive) number error
40	Disk Rename (FREN) error
41	Unused at this time
42	RANDOM file failed to create (disk space - already exists)
43	RANDOM file failed to open
44	RANDOM file failed to position (out of boundaries)
45	Attempt to read from random file a numeric variable that was not numeric (numerics on disk in BCD format)
46	Attempt to read from a random file a string variable that was not string data
47	Insufficient record size to hold data (total # of variables greater than record size or string size has been changed since file was created)

Error numbers 60-69 (sequential) and 70-79 (random) indicate that DFM (the disk file handler) has detected an error in handling the file number corresponding to the least significant digit of the number 40-49. DFM's own error code will also be displayed (see the BFD-68 system manual for value of the DFM error codes).

CASSETTE FILE ERRORS

ERROR	MEANING
32	Attempt to write to a file not opened for write
34	Attempt to read from a file not opened for read or past EOF

APPENDIX E

EXAMPLE OF THE USER FUNCTION

```
                    *
                    *
                    *  THE FOLLOWING EXAMPLE OF HOW TO USE USER
                    *  MULTIPLIES THE NUMBER 'X' GIVEN  USER(X)
                    *  BY TEN AND THEN RETURNS TO BASIC.  NOTE HOW
                    *  THE 'X' IS REFERENCED IN THIS PROGRAM -
                    *  THIS IS THE MOST COMMON MISUNDERSTANDING
                    *  ON HOW USER WORKS.
                    *
                    *
0030                UPOINT EQU  $30          ADDR OF USER DATA
                    *
0028                       ORG  $0028        ADDR OF POINTER TO USER PGM
0028 C0 00                 FDB  USTART       SET UP POINTER TO USER PGM
                    *
C000                       ORG  $C000
                    *
C000 7E C005        USTART JMP  BEGIN
                    *
                    * SAVE AREAS FOR USER PGM
                    *
C003 00 00          ISAVE  FDB  0
                    *
C005 FF C003        BEGIN  STX  ISAVE        SAVE THE INDEX REGISTER
C008 DE 30                 LDX  UPOINT       LOAD X W/ADDR OF USER DATA
C00A 6C 06                 INC  6,X          INC THE EXPONENT BY 1
C00C FE C003               LDX  ISAVE        RESTORE THE INDEX REGISTER
C00F 39                    RTS               RETURN TO BASIC
                    *
                           END
```

APPENDIX F

PARTIAL SOURCE LISTING

```
                        0247  *
                        0248  *
                        0249  *
                        0250  * START OF BASIC
                        0251  *
0100                    0252          ORG    $100
                        0253  *
0100 BD 0AC9            0254  ENTRY   JSR    START
0103 BD 0B70            0255          JSR    RSTART
0106 7E 0BB4            0256          JMP    RUN
                        0257  *
0109 2A FE              0258  BUFBAS  FDB    SRCBEG
010B 01                 0259  CNTPRT  FCB    1
                        0260  *
                        0261  * CASSETTE/PROM BASIC 'BUG BYTE'
                        0262  *
010C 00                 0263  IODEF   FCB    0
                        0264  *
                        0265  * 0=MIKBUG/SWTBUG-PIA (1)
                        0266  * 1=SWTBUG-ACIA (1)
                        0267  * 2=SWTBUG-ACIA (1)
                        0268  *          PIA  (0)
                        0269  * 3=SMTBUG-ACIA (2)
                        0270  *
                        0271  * NOTE: ABOVE IS CASSETTE/PROM ONLY
                        0272  *
010D 68 00              0273  MEMMAX  FDB    $6800
010F 80 00              0274  IOSTRT  FDB    $8000          START OF I/O PORTS
0111 0A FC              0275  HOMSTR  FDB    HOMLIS         HOME / CLEAR END-OF-FRAME
0113 0C 4E              0276  ERRPNT  FDB    ERROR          ADDRESS OF ERROR RTN (# IN ACCE
                        0277  *
0115 00 00              0278          FDB    $0000,$0000          RESERVED FOR FUTURE
0119 00 00              0279          FDB    $0000,$0000          JUMP ADDRESSES
011D 00 00              0280          FDB    $0000,$0000
                        0281  *
0121 18                 0282  DELINE  FCB    $18            CTL X
0122 08                 0283  DELCHR  FCB    $08            CTL H
0123 00                 0284  BSECHO  FCB    $00            NULL
0124 03                 0285  BRKCHR  FCB    $3             CTL C
                        0286  *
                        0287  * I/O PORT JUMP TABLE
                        0288  *
                        0289  * PORT 0
                        0290  *
0125 7E 0188            0291  JMPTAB  JMP    OUTACI
0128 7E 0178            0292          JMP    INACIA
012B 7E 016D            0293          JMP    ACIINZ
                        0294  *
                        0295  * PORT 1
012E 7E E1D1            0296          JMP    OUTEEE
0131 7E E1AC            0297          JMP    INEEE
0134 7E 0177            0298          JMP    DUMRTS
                        0299  *
                        0300  * PORT 2
0137 7E 0188            0301          JMP    OUTACI
013A 7E 0178            0302          JMP    INACIA
013D 7E 016D            0303          JMP    ACIINZ
                        0304  *
                        0305  * PORT 3
0140 7E 0188            0306          JMP    OUTACI
0143 7E 0178            0307          JMP    INACIA
0146 7E 016D            0308          JMP    ACIINZ
                        0309  *
                        0310  * PORT 4
0149 7E 01B3            0311          JMP    OUTPIA
014C 7E 01A8            0312          JMP    INPIA
014F 7E 0193            0313          JMP    PIAINZ
                        0314  *
                        0315  * PORT 5
0152 7E 01B3            0316          JMP    OUTPIA
0155 7E 01A8            0317          JMP    INPIA
0158 7E 0193            0318          JMP    PIAINZ
                        0319  *
```

```
                    0320  * PORT 6
015B 7E 01B3        0321          JMP    OUTPIA
015E 7E 01A8        0322          JMP    INPIA
0161 7E 0193        0323          JMP    PIAINZ
                    0324  *
                    0325  * PORT 7
0164 7E 01B3        0326          JMP    OUTPIA
0167 7E 01A8        0327          JMP    INPIA
016A 7E 0193        0328          JMP    PIAINZ
                    0329  *
                    0330  * SERIAL I/O INIT
                    0331  *
016D DE 85          0332  ACIINZ  LDX    PORTAD
016F 86 03          0333          LDA A  #$03
0171 A7 00          0334          STA A  0,X
0173 86 15          0335          LDA A  #$15
0175 A7 00          0336          STA A  0,X
0177 39             0337  DUMRTS  RTS
                    0338  *
                    0339  * SERIAL I/O INPUT
                    0340  *
0178 DE 85          0341  INACIA  LDX    PORTAD
017A A6 00          0342          LDA A  0,X
017C 47             0343          ASR A
017D 24 F9          0344          BCC    INACIA
017F A6 01          0345          LDA A  1,X
0181 84 7F          0346          AND A  #$7F
0183 81 7F          0347          CMP A  #$7F
0185 27 F1          0348          BEQ    INACIA
0187 01             0349          NOP                  RTS FOR NO ECHO
                    0350  *
                    0351  * SERIAL I/O OUTPUT
                    0352  *
0188 DE 85          0353  OUTACI  LDX    PORTAD
018A E6 00          0354          LDA B  0,X
018C C5 02          0355          BIT B  #2
018E 27 F8          0356          BEQ    OUTACI
0190 A7 01          0357          STA A  1,X
0192 39             0358          RTS
                    0359  *
                    0360  * PARALLEL I/O INIT
                    0361  *
0193 DE 85          0362  PIAINZ  LDX    PORTAD
0195 6F 01          0363          CLR    1,X
0197 6F 03          0364          CLR    3,X
0199 6F 00          0365          CLR    0,X
019B 6F 02          0366          CLR    2,X
019D 63 00          0367          COM    0,X
019F 86 3E          0368          LDA A  #$3E
01A1 A7 01          0369          STA A  1,X
01A3 86 2E          0370          LDA A  #$2E
01A5 A7 03          0371          STA A  3,X
01A7 39             0372          RTS
                    0373  *
                    0374  * PARALLEL I/O INPUT
                    0375  *
01A8 DE 85          0376  INPIA   LDX    PORTAD
01AA E6 03          0377          LDA B  3,X
01AC 2A FA          0378          BPL    INPIA
01AE A6 02          0379          LDA A  2,X
01B0 A7 02          0380          STA A  2,X
01B2 39             0381          RTS
                    0382  *
                    0383  * PARALLEL I/O OUTPUT
                    0384  *
01B3 DE 85          0385  OUTPIA  LDX    PORTAD
01B5 81 7F          0386          CMP A  #$7F
01B7 27 14          0387          BEQ    OPIA4
01B9 0F             0388          SEI
01BA A7 00          0389          STA A  0,X
01BC C6 36          0390          LDA B  #$36
01BE E7 01          0391          STA B  1,X
01C0 C6 3E          0392          LDA B  #$3E
01C2 E7 01          0393          STA B  1,X
01C4 96 D0          0394          LDA A  INTRP
01C6 06             0395          TAP
01C7 E6 01          0396  OPIA2   LDA B  1,X
01C9 2A FC          0397          BPL    OPIA2
01CB A6 00          0398          LDA A  0,X
01CD 39             0399  OPIA4   RTS
```

```
                    0400  *
                    0401  *
                    0402  * BREAK ROUTINE - CHECKS FOR CTL C
                    0403  *
01CE 37             0404  BREAK   PSH B
01CF 36             0405          PSH A
01D0 DF 7F          0406          STX    IOSAVX
01D2 FE 010F        0407          LDX    IOSTRT
01D5 96 CE          0408          LDA A  IOTYPE
01D7 2B 10          0409          BMI    BREAK2          MPC1
01D9 48             0410          ASL A
01DA 26 04          0411          BNE    BREAK0
01DC A6 04          0412          LDA A  ACIA1,X         MPS1
01DE 20 04          0413          BRA    BREAK1
                    0414  *
01E0 2A 07          0415  BREAK0  BPL    BREAK2          MPC1
01E2 A6 08          0416          LDA A  ACIA2,X         MPS2
01E4 47             0417  BREAK1  ASR A
01E5 24 11          0418          BCC    BREAK8
01E7 20 04          0419          BRA    BREAK3
                    0420  *
01E9 A6 04          0421  BREAK2  LDA A  PIAD,X
01EB 2B 0B          0422          BMI    BREAK8
01ED BD E1AC        0423  BREAK3  JSR    INEEE
01F0 B1 0124        0424  BREAK4  CMP A  BRKCHR
01F3 26 03          0425          BNE    BREAK8
01F5 7E 0B73        0426          JMP    READY
                    0427  *
01F8 DE 7F          0428  BREAK8  LDX    IOSAVX
01FA 32             0429          PUL A
01FB 33             0430          PUL B
01FC 39             0431          RTS
                    0432  *
                    0433  * OUT HEX RTN
                    0434  *
01FD A6 00          0435  OUTH    LDA A  0,X
01FF 8D 0D          0436          BSR    OUTHL
0201 A6 00          0437          LDA A  0,X
0203 08             0438          INX
0204 20 0C          0439          BRA    OUTHR
                    0440  *
                    0441  * OUT TWO HEX RTN
                    0442  *
0206 8D F5          0443  OUT2HS  BSR    OUTH
0208 8D F3          0444          BSR    OUTH
020A 86 20          0445  OUTSP   LDA A  #SPACE
020C 20 0E          0446          BRA    OUTCH
                    0447  *
020E 44             0448  OUTHL   LSR A
020F 44             0449          LSR A
0210 44             0450          LSR A
0211 44             0451          LSR A
0212 84 0F          0452  OUTHR   AND A  #$F
0214 8B 30          0453          ADD A  #'0
0216 81 39          0454          CMP A  #'9
0218 23 02          0455          BLS    OUTCH
021A 8B 07          0456          ADD A  #7
                    0457  *
                    0458  * OUTPUT CHARACTER RTN
                    0459  *
021C 37             0460  OUTCH   PSH B
021D 36             0461          PSH A
021E 8D AE          0462          BSR    BREAK
0220 DF 7F          0463          STX    IOSAVX
0222 DE 81          0464          LDX    OUTPTR
0224 AD 00          0465          JSR    0,X
0226 20 D0          0466          BRA    BREAK8
                    0467  *
                    0468  * INPUT CHARACTER RTN
                    0469  *
0228 DF 7F          0470  INCH    STX    IOSAVX
022A 37             0471          PSH B
022B DE 83          0472          LDX    INPTR
022D AD 00          0473          JSR    0,X
022F 36             0474          PSH A
0230 20 BE          0475          BRA    BREAK4
                    0476  *
                    0477  *
                    0478  *
                    0479  *
```

APPENDIX G

HOW TO REDUCE EXECUTION TIME

1. Subscripted variables require considerable time; use non-subscripted variables whenever possible.

2. The number of calculations involved in the transcendental functions (SIN, COS, TAN, ATAN, EXP, and LOG) make them slow. Use these functions only when necessary.

3. BASIC searches for functions and subroutines in the source file. Placing often called routines at the start of the program will reduce BASIC's search time.

4. Variables are entered into the symbol table as they are referenced. BASIC then searches the symbol table each time a variable is used. Therefore, reference a frequently called variable early in the source program so that it comes near the front of the table.

5. Numeric constants are converted each time they are encountered. If a constant is used often, it should be assigned to a variable and the variable name used instead.

6. Multiple statements per line take longer to process than the same instructions on single statement lines. This is an example of the classical trade off between space and speed.

APPENDIX H

MEMORY USAGE IN BASIC

1. REM statements use space, so use them wisely.

2. Each non-subscripted numeric variable uses 8 bytes.

3. Each numeric array uses 6 bytes + 6 bytes for each element.

4. If the default string length of 32 characters is used, each non-subscripted string variable uses 34 bytes and each string array uses 6 bytes + 32 bytes for each element. Use the STRING command to explicitly allocate the size you need.

5. An implicitly dimensioned variable creates a 10 X 10 array. If you do not intend to use all 10 elements use the DIM statement to explicitly allocate only the space you need.

6. Each BASIC line uses 2 bytes for the line number, 2 bytes for the encoded key word, 1 byte for the line length, 1 byte for the end of line terminator, plus 1 byte for each character following the key word. Reduce memory space by using as few spaces as possible.

7. Each Sequential file opened takes 177 bytes - Random files take 326 bytes per file opened. Reusing the same file number (after the file closing) in subsequent OPEN statements will save allocation of new space when the old space is no longer required.

APPENDIX I

LOADING & USING CASSETTE BASIC

LOADING BASIC - USING: Mikbug "L" and a
Binary Load Program

The following is typically how your terminal will look after you have loaded CSS BASIC.

```
*                                   * is the prompt
*L                                    in Mikbug
*G 03 E0 E12B 0100 A042               and Smartbug
*G

COMPUTERWARE BASIC                  Basic's ID and
VERSION: CASS - 3.0                 Version Number

BASIC
#                                   # is the prompt
                                      in BASIC
```

SEQUENCE OF EVENTS:

Mikbug '*' on the terminal - you type 'L' - and turn the AC-30 to the READ status 'locked on' position (this is the far right position). After the 'G' (which is on the tape) is displayed on the terminal put the AC-30 read status switch back ito the center position. When the Binary Load is completed, the following will be displayed on the terminal:

G 03 E0 E12B 0100 A042 (some of these values may differ)

Again the Mikbug '*' will appear on the terminal. You type 'G'. This exectues the BASIC interpreter which will display the Basic's ID, version number and prompt. You are now ready to use BASIC.

The same sequence of events follows for SWTBUG as shown above for MIKBUG. The reverse side of the CSS SUPER BASIC cassette has a binary and Mikbug load for the SMARTBUG operating system, which also works as described above. Regardless of which operating system you are using, if you have a problem with the binary loader, you should try the MIKBUG load before concluding that the cassette is in error.

USING BASIC WITH THE DIFFERENT SYSTEMS:

Appendix F shows the location and possible values for the 'BUG BYTE' or I/O Definition Indicator. The MIKBUG/SWTBUG side of the cassette is set for option #1, which is an ACIA control port in port #1. The SMARTBUG side of the cassette is set to option #3 - ACIA in port #2. If neither of these settings is compatible with your system, you may either change the 'BUG BYTE' each time you load or, resave BASIC with your modifications included.

APPENDIX J

SYSTEM DEFAULT VALUES

The following table lists the default values of system parameters that are set on system initialization or on executing the NEW, LOAD, or CHAIN commands.

TRACE is turned off.

DIGITS is set to floating point mode.

RJUST is set to floating point mode.

STRING is set to 32.

BASE is set to 1.

LINE is set to 64.