Software Services

COMPUTERWARE™ BASIC

A 68XX 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.

NOTE: Computerware supports ONLY SS-50 6800 Configurations with their I/O at $8000 or $F7E0 and if Disk Based, with DOS located at either the $6000-$7FFF or $C000-$DFFF range. On the 6809, we support I/O at either $E000 or $F7E0, with SSB DOS at $C000-$DFFF and our MON09 or MON69 Monitor. We will offer no free assistance for persons with configurations other than this. Our minimum software consulting charge is $100.00 in advance.

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 have 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. (90 characters for cassette and prom Basic.)

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 an alphanumeric text string (the 1st character may NOT be numeric). BASIC considers only the first 6 positions for uniqueness.

Examples: TOTAL, Y, Z, X3, TAX3, R(5,2), ARRAY(25)

STRING VARIABLES:

String variables may contain a maximum of 126 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. String variables must contain a '$' as the last character in the variable name.

Examples of string variables: NAME$, Y$(7), TABLE$(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.

SIX CHARACTER VARABLE NAMES:

Random BASIC supports variable names ranging in length from 1 to 126 characters. Only the first six positions are used by BASIC when referencing a variable. If the variable is a numeric variable, the first position may NOT be numeric. String variables must have their '$' designator within the name, and only the portion to the left of the '$' will be considered when BASIC is evaluating the variable name. BOTH upper and lower case characters are allowed within variables and they are not considered to be equivalent - ie. A is not equal to a. The main purpose in allowing the extended variable naming is to promote better programming thru better documented programs.

* CANNOT USE MULTI CHARACTER VARIABLE NAMES IN DISK OPEN + CREATE COMMANDS

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.

REPEAT -

Typing CONTROL D will cause whatever is in BASIC'S input buffer to be again used as a line of input. This feature works in the immediate mode and its value is for the user to establish.

HALT -

With some of the operating systems (SMARTBUG/MON09), typing the rub-out character (Hex $7F) will cause processing to halt. This applies to both commands used in the immediate mode and while a program is running. To continue processing type any character but a rub-out or a CTL C.

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.

DEL - (6809 only)

The DEL command allows the user to delete portions of a BASIC program that is in the computer's memory. The format is: DEL N1 [,N2] where N1 is the starting (or only) line number and N2 is the ending line number of a range of lines. DEL only works in the immediate mode.

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. If for some reason you want to restart a program from within the program, the command RUN will close all files prior to restarting, thus allowing it's use when files are open.

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 MON09/SMARTBUG this will output a carriage return, line feed, and the "*" prompt character. If the stack pointer address 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 from DOS, type 'GOTO 103', BASIC's soft start.

AUTO -

The AUTO command provides the user with automatic generation of line numbers while writing/adding to a BASIC program. The format of the command is as follows: AUTO [X {, Y}], where 'X' is a value between 1 and 9999 and represents the beginning number to be automatically generated. 'Y' is a value between 1 and 9999 and represents the increment value between numbers. The defaults for 'X' and 'Y' are 100 and 10 respectively. To get out of the AUTO mode, use the break character (typically CTL C).

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 may NOT 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.

EDIT -

The EDIT command will allow the user to have an 'overlay' capability similar to the editor - in BASIC. The format is EDIT <line-number>, where line-number is the line that the user wants to make changes to. To change a character simply type the new character under the old one. To leave a character unchanged press the 'fill-character' (see below). CTL D will automatically include the remainder of the old line with any changes you may have made. Pressing carriage return will truncate the line at the point where CR was pressed. The line can be added to by typing beyond the end of the line and then pressing CR.

The 'fill character' is currently set to $0C (forspace character on a SOROC terminal), but may be changed to whatever control character causes a right horizontal tab on your terminal (eg. $09 for an ACT IA). To change the fill character, simply change location $0140 (CHGCHR) in the BASIC to what will work for you.

DO -

The DO command allows the user to execute most DOS commands from BASIC. It works only in the immediate mode, and it's format is as follows: DO 'DOS COMMAND'. Great CAUTION should be exercised when using the DO command - in that the DOS command requested may in fact have a memory location conflict with BASIC. Examples of commands that should NOT be used include BACKUP, FORMAT, COPY, SAVET, EDIT, ASMB, BASIC and any other command that loads into a memory location lower than $4000. If in doubt of the location a command loads into, use the FIND command to verify its location. Example: DO FIND,VIEW.$ will execute from BASIC and in fact tell you that VIEW is located in the DOS utility command area and is safe to use with DO.

NOTE: Due to the way BASIC interprets a program, the DOS commands starting with an 'S' will not work from BASIC. Simply renaming them to a name that does NOT start with 'S' will allow their use.

BASIC DISK COMMANDS AND FUNCTIONS:

The following commands and functions are unique to the DISK BASICS. 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 - (Random Basic only)
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.

0 = FILE FOUND

```
Examples:   IF FCHK 1:MASTER.FIL <> 0 THEN 2000 (FILE NOT FOUND)
   or       LET A = FCHK PAYROL.BAS
```

FREE - (Random Basic only)
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 reloaded 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).

PAGE= -

The PAGE= command allows the user to set their page length value into the page length counter. Every time a line feed character is outputted from BASIC (to any port), the page counter is decremented. When the counter is equal to zero, it is reset to the value in the page length counter. The default value is 66.

PAGE -

The PAGE function returns to the user the current value in the page counter. An example of it use:

0010 IF PAGE < 5 SKIP,PAGE - If within five lines to the bottom of the page, skip to the bottom.

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 126 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=.

NOTE: BASE can be used only once in a program, and it's occurance must be BEFORE references to any variables, DIM statements, etc.

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).

FPSW= - (6809 only)

The FPSW command allows the user to set the value of the Floating Point Switch to a number between 1 and 9. The Floating Point Switch indicates the number of positions to the right of the decimal point that will be shown in the floating point notation rather than exponential notation. The total number of positions printed to the right of the decimal point is still controlled by the DIGITS command. Example: Try to get .005 to print without FPSW equal to 3 or more. The FPSW function returns to the user the value of the Floating Point Switch.

ION - (6809 only)

The ION command enables the IRQ interrupt (clear the IRQ mask bit).

IOFF - (6809 only)

The IOFF command disables the IRQ interrupt (set the IRQ mask bit).

FION - (6809 only)

The FION command enables the FIRQ interrupt (clear the FIRQ mask bit).

FIOFF - (6809 only)

The FIOFF command disables the NIRQ interrupt (set the FIRQ mask bit).

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 CSS DISK BASIC only commands, while TSAVE, TLOAD, and TPEND are in cassette and sequential disk BASIC (Not in RANDOM).

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$) - but NOT a Subscripted Variable (ie. CHAIN A$(1)).

TSAVE - (Not in RANDOM)

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 - (Not in RANDOM)

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 - (Not in RANDOM)

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.
```

NOTE: Exponentiation with negative numbers will give unpredictable results.

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. (Not in RANDOM BUSINESS BASIC)

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
```

TAB(0) is a null tab command. Thus if you are using TAB with a numeric variable and sometimes do not want to tab, assign a value of zero to the variable prior to executing the TAB statement.

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
```

ESC -

ESC(X) when used in conjuction with the PRINT statement, will send an Escape Sequence to the output device. 'X' represents the numeric value in decimal ASCII, of the character that is to be sent following the Escape ($1B) control character.

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
```

NVAL -

NVAL allows the user to 'test' a string variable for numerics prior to using the VAL function. NVAL can be used in conjunction with the MID function to test any position within a string variable. The format for NVAL is:

```
IF NVAL (A$) = 0     The FIRST position of A$ is NUMERIC
IF NVAL (A$) <> 0    The FIRST position of A$ is NOT NUMERIC
                         and an ERROR 27 will occur if you
                         try to take the VAL(A$).
```

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)

MSIZE -

The MSIZE function returns the decimal value of unused memory space in the computer. It is identical to the available space value as shown in the SIZE command.

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

NOTE: The above Functions are NOT in RANDOM BUSINESS BASIC.

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 interpreter. 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 interpreter. 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:

```
Byte:     1     2     3     4     5     6     7
          X X   X X   X X   X X   X X   X X   X X
          ^ ^---------------------------^   ^-^   ^-^
          1               2                 3     4
```

1 = sign	2 = BCD number	3 = not used	4 = exponent
+ = 0	Standard BCD		Standard BCD
- = 9	10's Compliment		Standard BCD

BASIC STATEMENTS:

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 a one or two dimension array. An array is a collection of subscripted variables or strings. The DIM statement initializes String arrays to nulls, and numeric arrays to zeros.

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. The maximum dimension size is '255' - which will provide an array that has 256 variable positions when BASE=0 and 255 positions when BASE=1. When processing under BASE=0, there is always one (1) more position in the array than the DIM size. Some 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)

CALL -

CALL allows the user to 'call' an assembly language sub-routine from BASIC and pass a parameter to that sub-routine. The format of CALL is simpler and more flexible than the USER function. But, just as with USER, CALL is a very DANGEROUS command in that if not properly used, it is entirely possible to DESTROY your program, your disk and your sanity. The format of CALL is identical to that of POKE (destructive commands hang around together):

CALL(X,Y)

Where X is the DECIMAL address of your sub-routine and Y is a DECIMAL value between 0 and 255 which will be placed in the 'A' accumulator for use by your sub-routine. You MUST end your sub-routine with an RTS ($39) if you wish to return to BASIC. There is no provision for passing data back to BASIC. Both USER and the INPUT command can be used for this.

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. (Not in RANDOM BUSINESS BASIC)

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.

FOR - NEXT contd.

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
```

ON ERROR -

The command ON ERROR can be followed by any BASIC statement. More than one ON ERROR command can be used in the same program; the last one encountered will be the one executed if an error occurs. If there is an error in the ON ERROR routine, an infinite loop will result.

For example: 0010 ON ERROR GOTO 9000

ERLINE -

The ERLINE function returns to the user the line number on which the error occurred. For Example:

```
9000 PRINT "ERROR FOUND IN LINE: ";ERLINE
```

ERCODE -

The ERCODE function returns to the user the error number that occurred. The meaning of this value is described in Appendix D. For Example:

```
9010 PRINT "ERROR CODE WAS: ";ERCODE
```

or

```
9010 IF ERCODE = 14 PRINT "BUY MORE MEMORY - TURKEY!!"
```

IF --- THEN - [:ELSE]

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:

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

use the following:

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

:ELSE -

The :ELSE operator allows for the 'not true' side of a compare to be acted upon on the same line as the 'true' side. It allows for more efficient program flow in that without it, the 'not true' case had to be processed on a separate line and then if program flow was to again converge, a GOTO over the 'not true' was required by the 'true' side. The format for :ELSE is as follows: it must appear as the next operator after the normal [IF (expr) THEN (operation)], and must be preceeded by a colon. ie. IF A=4 THEN B=6 : ELSE B=8.

Processing can continue after the :ELSE operator, and it is executed by both the 'true' and 'not true' sides of the compare (just as if it were a new line).

PACK -

The PACK statement allows multiple string variables to be 'packed' into one string variable. The format is as follows: PACK A$,X$,Y$..... where A$ is the receiving variable and X$,Y$, etc. are the variables to the 'packed'. Each of the input string variables is separated from the next by the PDLM character. Standard string truncation will take place if the total of the input strings is greater than the size 'STRING' is set to.

UNPACK -

The UNPACK statement functions in the reverse of the PACK command. It's format is as follows: UNPACK A$,X$,Y$.... The string A$ will be moved into X$ until the PDLM character is found - then into Y$, etc. Each time a PDLM is found, a new output string is used. If there are more segments in the string being unpacked than receiving strings, the excess is ignored. If there are not enough segments, the remaining output strings are filled with nulls.

PDLM -

The PDLM statement sets the Pack Delimiter to the value of the left most character in the argument on the right of the equal sign. Example: PDLM = S$ (PDLM now equals the first character in S$) or PDLM = "JUNK" (PDLM now equals the character 'J').

NOTE: The default value for PDLM is the tilde '~'.

PDLM -

The PDLM function returns the current value of the PDLM character. Format: LET P$ = PDLM or IF PDLM = "?" THEN END.

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 inputted 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".

If a carriage return is pressed without any number when inputting a numeric variable, a zero will be placed into the numeric variable.

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.

PRINT contd.

Examples:

```
10 PRINT                 Skip a line
20 PRINT A,B,C           Print variables A, B, and C auto-
                         matically 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
```

DLM=ON - DLM=OFF

DLM=ON and DLM=OFF allow the user to manipulate strings with commas and semi-colons etc. in them. DLM=ON is the 'normal' mode of BASIC in that comma is used as a delimiter between both string and numeric variables. The default upon loading BASIC or executing NEW is DLM=ON. DLM=OFF allows the user to input or read from a disk file a string variable that has imbedded commas, along with any other valid ASCII characters other than carriage return. This feature also applies to printing or writing to a disk file. Another way to describe DLM=OFF is that it provides a 'line input' capability: one line = one string variable. Only one (1) variable can be referenced at a time on either the INPUT, PRINT, READ0, or WRITE0 commands. The delimiter commands are supported for both sequential and random files.

Even though the DLM= command can be used at any time in a BASIC program, Computerware does not recommend this as it is very easy to get the variable pointers in BASIC confused through accidental use and thus garbage all your data.

PRINT USING -

The PRINT USING statement allows both alphanumeric and numeric variables to be printed in a pre-defined format. The format is:

PRINT USING (mask),(variable list)

The mask is a string expression that must match the variable list with respect to the data type (alphanumeric or numeric) and number of elements. If there is not a one for one correspondence between the mask and the variable list, then an ERROR 48 will result and none of the 'USING' elements will be printed. The following list describes the control characters that PRINT USING recognizes:

PRINT USING contd.

'	print alphanumeric character
#	print numeric digit
$	print 'floating' dollar sign
*	print asterisk fill from leftmost position to the most significant digit

Any text that is imbedded in the mask other than the control characters will be printed in its own position on the output device. The mask for any numeric field must be followed by a space to reserve a print position for the minus sign. If a numeric value requires more print positions than provided in the mask, the field will be filled with asterisks to indicate an overflow condition. When a control character of "*" is used, an error will result if the field is negative. The following is a sample of valid masks:

VARIABLE	MASK	RESULTS
"ABCDEFGHIJ"	'''	ABC
"ABC"	'''''	ABC^^
1.5	###	1
1.5	###.##	1.50
1.5	$###.##	$1.50
1.5	*###.##	***1.50
1.5	$*###.##	**$1.50
-1.5	$###.##	$1.50-
-1.5	###	1-

$######.## WAS DUE ON '''''''' $173.86 WAS DUE ON 05/31/79

The PRINT USING element list may be terminated by a semicolon, colon or carriage return. If terminated by the semicolon, the normal end-of-print-line sequence is suppressed and the print mechanism will be left positioned after printing the last element. Any non-control character will serve as a delimiter between string fields, or at the start of a numeric field. The following are some valid examples:

```
PRINT #P,TAB(10);USING Q,,A0;L$
PRINT #P,TAB(90);"TODAY IS ";D$;TAB(35);"TOTAL = ";USING M$,A1
PRINT #P,TAB(15);USING M$,A2;
PRINT #P,TAB(25);USING Z$,A3;TAB(35);A$;TAB(50);A5
```

The PRINT USING does not affect the values of DIGITS or RJUST. The commands DIGITS and RJUST have no effect on the PRINT USING command. They do apply to the non-USING portion of the PRINT statement. Conventional PRINT and PRINT USING can be intermixed in the same statement.

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 rmsult in a disk error and termination of the Basic program.

Sequential files are Variable Length Record files and are in ASCII format. Only the actual data in a variable will be written out, regardless of string lentgh. Commas, Semi-colons and Carriage Returns are used as delimiters.

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 statemmnt{. 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 - 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 between (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,"1: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 - and will be OPEN.

EXPAND -

EXPAND Random Data Bases allows a previously CREATED file to be increased in record count. It's format is as follows: EXPAND #(FLN),(FILE SPEC),(NEW # OF RECS). The new # of records is the total of the previous size plus the amount to be expanded by.

Example: EXPAND#12,"1:CREDIT.HST",300

NOTE: EXPAND and CREATE should never be used in multiple statement lines.

CLOSE -

The command CLOSE #(FLN),#(FLN),... is for closing open files. The file number may be an expression. The specified file number must have previously been OPENed.

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, it will receive the data from the file up to the maximum string length of 126 characters. The line input buffer for a single item from a file is 128 characters.

A string item over 126 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 READing a numeric variable, the input is scanned for 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: The commands GET, READ, PUT, WRITE have the ability to be CONTINUED on additional lines. To continue one of these commands, a colon (:) must be placed on the end of the line to be continued (right after the last variable on that line). The next line (the continuation) must start with a colon, followed immediately by the remaining variables in the record. Example:

```
0100 GET #10, NAME$,ADDR$,CITY$,ZIP$,CURBAL:
0110 : PREBAL,DATE1,DATE2,DATE3:
0120 : AMTDUE,PSTDUE
```

GET -

GET is a random access statement identical to READ except that automatic 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 described earlier functions for DATA statements. The file number may be that of a 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 #1,2: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. Thg 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.

The record number (RECNO) will be automatically 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 automatic 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 for SSB and 8 for FLEX. The status number is that number returned by the disk file management system. Refer to your DOS MANUAL for other values. It is a good idea to check the file status after a READ at least for end of file.

If STATUS of a file is checked after opening, but prior to reading or writing, the absence or presence of the file may be established without getting a Basic Error (if the file were not there). A STATUS of zero (0) means the file is there and non-zero means not there.

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 automatically incremented.

FSIZE -

The FSIZE function returns to the user the size of a file in terms of a decimal count of sectors. To establish the file size in characters, single density files should be multiplied by 124 and double density files by 252.

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#15 > 55 THEN 2000
PRINT RECNO#17 / PI
```

CASSETTE FILE HANDLING STATEMENTS:

---> CASSETTE BASIC ONLY <---

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 image 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.....

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	56	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 A

QUICK REFERENCE OF INSTRUCTIONS

COMMANDS

APPEND	RJUST				
CHAIN	RUN				
CONT	SAVE				
DIGITS	SKIP				
HOME	STRING				
LINE	TLOAD				
LIST	TPEND				
AUTO	DOS				
LOAD	TRACE				
MON	TSAVE				
NEW	WAIT				
PORT	BASE				
SIZE	PAGE				
FLIST	FREN				
DO	ON-ERROR				
FDEL	REPLACE				
EDIT	DEL				
ION	IOFF				
FION	FIOFF				
FPSW					

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	FREE				
ESC	FCHK				
ERCODE	ERLINE				
STATUS	FSIZE				
MSIZE	RSIZE				
NVAL	RNEXT				
RECNO					

STATEMENTS

DATA	ON-GOSUB				
DIM	ON-GOTO				
END	POKE				
FOR-NEXT	PRINT				
GOSUB	READ				
GOTO	REM				
IF-THEN	RESTORE				
INPUT	RETURN				
LET	STOP				
SCRATCH	READ				
CLOSE	WRITE				
PDLM	OPEN				
PACK	UNPACK				
SET	ELSE				
DLM=OFF	DLM=ON				
GET	PUT				
CREATE	CALL				
EXPAND					
PRINT USING					

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 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 - 010D	Maximum memory size available for BASIC to use.
010E - 010F	Address of home / clear end-of-frame string
0110 - 0111	Address of carriage return/line feed string
0112 - 0113	Address of ERROR routine (Error # in ACCB)
0114 - 011B	Disk FCB sizes and location of non file FCB
0137 - 013A	RESERVED for future jump addresses
013B	Line delete control character (CTL X)
013C	Character delete control character (CTL H)
013D	Character delete ECHO character (NULL)
013E	BREAK control character (CTL C)

NOTE: The last 256 bytes of memory available are used as a string expression buffer and for the machine stack. ALSO, some of the above addresses may vary between the different versions of BASIC (cassette or disk etc.). The partial Source Listing that comes with your Basic will give you accurate addresses.

APPENDIX D

ERROR MESSAGES

The following is printed when an error occurs:

```
ERROR # -----
LINE NO.  -  DATA ON LINE
                  ^          (DISK BASIC ONLY)
                  ----  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	Not used in Logical IrO Basic
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	Cheap memory found in system
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)
48	Print Using Error - best bet is to re-read manual and go back and check your code.

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 error number. 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
```

```
                    0261 *
0100                0262          ORG    $100
                    0263 *
0100 BD028B         0264 BEGIN    JSR    START        COLD START
0103 BD0C02         0265          JSR    RSTART       SOFT START
0106 7E0CD7         0266          JMP    RUN           AUTO RUN
                    0267 *
0109 3140           0268 BUFBAS   FDB    SRCBEG       END OF BASIC & WORK AREAS
010B 02             0269 CNTPRT   FCB    2            CONTROL PORT
010C C000           0270 MEMMAX   FDB    $C000        MEM LIMIT - CHGD TO YMEMAX
010E 0B88           0271 HOMSTR   FDB    HOMLIS       ADDR OF HOME/CLEAR EOF STRING
0110 03DF           0272 CRLFST   FDB    CRLF2        ADDR OF CR/LF STRING
0112 0D76           0273 ERRPNT   FDB    ERROR        ADDR OF ERROR RTN (# IN ACCB)
0114 00A6           0274 SEQFSZ   FDB    166          SEQUENTIAL FILE FCB SIZE
0116 0140           0275 RNDFSZ   FDB    320          RANDOM FILE FCB SIZE
0118 0240           0276 DRNDFZ   FDB    576          RANDOM DOUBLE DENSITY
011A C880           0277 CMDFCB   FDB    DOSFCB        COMMAND FCB ADDRESS
                    0278 *
011C 43             0279 NOTICE   FCC    'COPYRIGHT 1979 COMPUTERWARE'
                    0280 *
0137 FFFF           0281          FDB    $FFFF,$FFFF JUMP ADDRESSES
                    0282 *
013B 18             0283 DELINE   FCB    $18          CTL X
013C 08             0284 DELCHR   FCB    $08          CTL H
013D 00             0285 BSECHO   FCB    $00          NULL
013E 03             0286 BRKCHR   FCB    $3           CTL C
013F 2C             0287 DELIM    FCB    COMMA        - VARIABLE DELIMITER
0140 0C             0288 CHGCHR   FCB    $0C          CHG FILL CHAR
                    0289
                    0290 * I/O DEFINITION TABLE  -  CONFIGURATION BYTE
                    0291 *
                    0292 *BIT# |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
                    0293 *     |     |     |     |     |     |     |     |     |
                    0294 *HEX  | 80  |  40 |  20 |  10 |  08 |  04 |  02 |  01 |
                    0295
                    0296 *      CTL    RES   SPR   PIA   PIA   STD   X64   X16
                    0297 *      TERM   CAS          IN    OUT   PIA   SER   SER
                    0298
                    0299 * CONFG = 0 -> NON-STD I/O (IE. VIDEO BOARD - GRAPHICS)
                    0300
                    0301 * LOGICAL UNIT TABLE ENTRY
                    0302
                    0303 *   0   -> CONFIGURATION BYTE
                    0304 *  1-2  -> ADDRESS OF I/O DEVICE
                    0305 *  3-5  -> JUMP CHAR OUTPUT
                    0306 *  6-8  -> JUMP CHAR INPUT
                    0307 *  9-11 -> JUMP PORT INITIALIZATION
                    0308
0141                0309 LUTBLE EQU    *
                    0310 *
                    0311 * LOGICAL UNIT #0
                    0312 *
0141 01             0313 LU0      FCB    SERCLK
0142 E000           0314          FDB    0*4+IO
0144 7E01F0         0315          JMP    CHROUT
0147 7E021E         0316          JMP    CHRIN
014A 7E01A1         0317          JMP    IOINIT
```

```
                    0319 *
                    0320 * LOGICAL UNIT #1
                    0321 *
014D 01             0322 LU1     FCB   SERCLK
014E E004           0323         FDB   1*4+IO
0150 7E01F0         0324         JMP   CHROUT
0153 7E021E         0325         JMP   CHRIN
0156 7E01A1         0326         JMP   IOINIT
                    0327 *
                    0328 * LOGICAL UNIT #2
                    0329 *
0159 81             0330 LU2     FCB   $80+SERCLK
015A E008           0331 CTLADR  FDB   2*4+IO
015C 7E01F0         0332 CTLOUT  JMP   CHROUT
015F 7E021E         0333 CTLIN   JMP   CHRIN
0162 7E01A1         0334         JMP   IOINIT
                    0335 *
                    0336 * LOGICAL UNIT #3
                    0337 *
0165 01             0338 LU3     FCB   SERCLK
0166 E00C           0339         FDB   3*4+IO
0168 7E01F0         0340         JMP   CHROUT
016B 7E021E         0341         JMP   CHRIN
016E 7E01A1         0342         JMP   IOINIT
                    0343 *
                    0344 * LOGICAL UNIT #4
                    0345 *
0171 04             0346 LU4     FCB   $04
0172 E010           0347         FDB   4*4+IO
0174 7E01F0         0348         JMP   CHROUT
0177 7E021E         0349         JMP   CHRIN
017A 7E01A1         0350         JMP   IOINIT
                    0351 *
                    0352 * LOGICAL UNIT #5
                    0353 *
017D 04             0354 LU5     FCB   $04
017E E014           0355         FDB   5*4+IO
0180 7E01F0         0356         JMP   CHROUT
0183 7E021E         0357         JMP   CHRIN
0186 7E01A1         0358         JMP   IOINIT
                    0359 *
                    0360 * LOGICAL UNIT #6
                    0361 *
0189 04             0362 LU6     FCB   $04
018A E018           0363         FDB   6*4+IO
018C 7E01F0         0364         JMP   CHROUT
018F 7E021E         0365         JMP   CHRIN
0192 7E01A1         0366         JMP   IOINIT
                    0367 *
                    0368 * LOGICAL UNIT #7
                    0369 *
0195 04             0370 LU7     FCB   $04
0196 E01C           0371         FDB   7*4+IO
0198 7E01F0         0372         JMP   CHROUT
019B 7E021E         0373         JMP   CHRIN
019E 7E01A1         0374         JMP   IOINIT
                    0375
                    0376 ********************************************************
```

```
                        0378
                        0379 * INITIALIZE I/O PORT AS SPECIFIED
                        0380
01A1 9E56               0381 IOINIT LDX   LUPTRX
01A3 E684               0382        LDB   CONFG,X
01A5 AE01               0383        LDX   OPORT,X
01A7 C49F               0384        ANDB  #$9F          CLEAR NU BITS
01A9 2B13               0385        BMI   IOINTR        CONTROL TERM
01AB 2711               0386        BEQ   IOINTR        OTHER TYPE IO
01AD C51C               0387        BITB  #%11100       PIA TYPE
01AF 260E               0388        BNE   IOPIA         YES
01B1 86FF               0389        LDA   #$FF          DELAY
01B3 4A                 0390 IOI1   DECA                  FOR OUTPUT
01B4 8103               0391        CMPA  #3            COMPLETION
01B6 26FB               0392        BNE   IOI1
01B8 A784               0393        STA   0,X           MASTER CLEAR ACIA
01BA CA14               0394        ORB   #$14          COMBINE WORD & DIVIDE SELECT
01BC E784               0395        STB   0,X
01BE 39                 0396 IOINTR RTS
01BF C504               0397 IOPIA  BITB  #%00000100    STD PIA
01C1 2713               0398        BEQ   IOPIA1        NO
01C3 6F01               0399        CLR   1,X           INIT A AND B SIDES
01C5 6F03               0400        CLR   3,X
01C7 6F84               0401        CLR   0,X
01C9 6F02               0402        CLR   2,X
01CB 6384               0403        COM   0,X
01CD 863E               0404        LDA   #$3E
01CF A701               0405        STA   1,X
01D1 862E               0406        LDA   #$2E
01D3 A703               0407        STA   3,X
01D5 39                 0408        RTS
01D6 C508               0409 IOPIA1 BITB  #%00001000    OUTPUT PIA
01D8 270B               0410        BEQ   IOPIA2        NO
01DA 6F01               0411        CLR   1,X           SET PIA A FOR OUTPUT
01DC 86FF               0412        LDA   #-1
01DE A784               0413        STA   0,X
01E0 863E               0414        LDA   #$3E
01E2 A701               0415        STA   1,X
01E4 39                 0416        RTS
01E5 6F01               0417 IOPIA2 CLR   1,X           SET PIA A FOR INPUT
01E7 8600               0418        LDA   #0
01E9 A784               0419        STA   0,X
01EB 862E               0420        LDA   #$2E
01ED A701               0421        STA   1,X
01EF 39                 0422        RTS
                        0423
                        0424 ********************************************************
```

```
                    0426 * DO CHARACTER OUTPUT
                    0427
01F0 E684           0428 CHROUT LDB   CONFG,X
01F2 2712           0429        BEQ   CHROR        NO OUTPUT REQ'D
01F4 AE01           0430        LDX   OPORT,X
01F6 C50C           0431        BITB  #%00001100   PIA OUTPUT
01F8 260D           0432        BNE   OUTPIA       YES
01FA C503           0433        BITB  #%00000011   ACIA OUTPUT
01FC 2708           0434        BEQ   CHROR        NO
01FE E684           0435 CHRO1  LDB   0,X          WAIT FOR TDRE
0200 C502           0436        BITB  #2
0202 27FA           0437        BEQ   CHRO1
0204 A701           0438        STA   1,X          OUT CHAR
0206 39             0439 CHROR  RTS
0207 1A50           0440 OUTPIA ORCC  #$50         LOCK INTERRUPTS
0209 A784           0441        STA   0,X          OUTPUT CHAR
020B C636           0442        LDB   #$36
020D E701           0443        STB   1,X
020F C63E           0444        LDB   #$3E
0211 E701           0445        STB   1,X
0213 965C           0446        LDA   INTRP        RESTORE INTERRUPT STATE
0215 1F8A           0447        TFR   A,CC
0217 E601           0448 OUTPI1 LDB   1,X          WAIT FOR CHAR TRANSFER
0219 2AFC           0449        BPL   OUTPI1
021B A684           0450        LDA   0,X          CLEAR INTR FLAG
021D 39             0451        RTS
                    0452
                    0453 ********************************************************
                    0454
                    0455 * DO CHARACTER INPUT
                    0456
021E E684           0457 CHRIN  LDB   CONFG,X
0220 272D           0458        BEQ   CHRINR
0222 AE01           0459        LDX   OPORT,X
0224 C514           0460        BITB  #%00010100   PIA INPUT
0226 2615           0461        BNE   INPIA        YES
0228 C503           0462        BITB  #%00000011   ACIA INPUT
022A 2723           0463        BEQ   CHRINR       NO
022C E684           0464 INACIA LDB   0,X          WAIT FOR RDRF
022E 57             0465        ASRB
022F 24FB           0466        BCC   INACIA
0231 A601           0467        LDA   1,X          GET CHAR
0233 847F           0468        ANDA  #$7F         MASK PARITY
0235 817F           0469        CMPA  #$7F         DEL CHAR
0237 27F3           0470        BEQ   INACIA       YES,GET NEXT
0239 9E56           0471 INECHO LDX   LUPTRX       ECHO VIA CHAR OUTPUT
023B 6E03           0472        JMP   OFFOUT,X
023D C504           0473 INPIA  BITB  #%00000100   STD PIA
023F 2708           0474        BEQ   INPIA2       NO
0241 E603           0475 INPI1  LDB   3,X          WAIT FOR INTR ON B
0243 2AFC           0476        BPL   INPI1
0245 A602           0477        LDA   2,X          READ B SIDE
0247 20F0           0478        BRA   INECHO       ECHO CHAR
0249 E601           0479 INPIA2 LDB   1,X          WAIT FOR INTR ON A
024B 2AFC           0480        BPL   INPIA2
024D A684           0481        LDA   0,X          READ A SIDE
024F 39             0482 CHRINR RTS                NO ECHO, INPUT ONLY
                    0483
                    0484 ********************************************************
```

```
                        0486
                        0487 * PROCESS CHECK FOR BREAK
                        0488
0250 3416               0489 BREAK   PSHS  D,X
0252 9E58               0490         LDX   LUBRKX       USE DEFAULT PORT ONLY
0254 E684               0491         LDB   CONFG,X
0256 AE01               0492         LDX   OPORT,X
0258 C517               0493         BITB  #%00010111   ACIA,PIA INPUT
025A 2718               0494         BEQ   BREAK2       NO
025C C503               0495         BITB  #%00000011   ACIA
025E 2616               0496         BNE   BREAK3       YES
0260 C514               0497         BITB  #%00010100   PIA INPUT
0262 2619               0498         BNE   BREAK4       YES
0264 200E               0499         BRA   BREAK2       WRONG, NO CONFIGURATION TO BR
0266 9E58               0500 BREAK0  LDX   LUBRKX       GOTO INPUT PROCESSOR
0268 AD06               0501         JSR   OFFIN,X
026A B1013E             0502 BREAK1  CMPA  BRKCHR       BREAK CHAR?
026D 2605               0503         BNE   BREAK2       NO
026F 0F81               0504         CLR   AUTFLG
0271 7E0C08             0505         JMP   READY        STOP CYCLING
0274 3596               0506 BREAK2  PULS  D,X,PC
0276 E684               0507 BREAK3  LDB   0,X          ACIA PORT
0278 57                 0508         ASRB
0279 24F9               0509         BCC   BREAK2
027B 20E9               0510         BRA   BREAK0
027D C504               0511 BREAK4  BITB  #%00000100   STD PIA
027F 2706               0512         BEQ   BREAK6       NO
0281 E603               0513         LDB   3,X          CHECK B SIDE
0283 2AEF               0514 BREAK5  BPL   BREAK2
0285 20DF               0515         BRA   BREAK0
0287 E601               0516 BREAK6  LDB   1,X          PIA INPUT ONLY
0289 20F8               0517         BRA   BREAK5       CHECK SIDE A
                        0518
                        0519 ********************************************************
```

```
                        0521
                        0522 * BASIC'S START-UP ROUTINE
                        0523
028B 86FF               0524 START   LDA    #$FF
028D 976D               0525         STA    AUTRFG      TURN ON TEST FOR AUTO FILE LOA
028F B6010B             0526         LDA    CNTPRT      CONTROL PORT#
0292 975B               0527         STA    CONSOL
0294 1FA8               0528         TFR    CC,A
0296 84D0               0529         ANDA   #$D0        KEEP INTERRUPT STUFF
0298 975C               0530         STA    INTRP       SAVE INTERRUPT MASKS
029A 8E0C08             0531         LDX    #READY      BREAK RETURN ADDR
029D BFD30D             0532         STX    YABRTV      STORE IN DOS BREAK ADDR
02A0 BED317             0533         LDX    YCPORT      PARAMETER TABLE CTL PORT ADDR
02A3 2703               0534         BEQ    START0
02A5 BF015A             0535         STX    CTLADR      SAVE IN PORT#2 JUMP TABLE
02A8 BED302             0536 START0  LDX    YMEMAX      PARAMETER TBL MEM LIMIT
02AB 2705               0537         BEQ    START1
02AD BF010C             0538         STX    MEMMAX
02B0 2017               0539         BRA    START4
02B2 BE0109             0540 START1  LDX    BUFBAS
02B5 86AA               0541         LDA    #$AA
02B7 A784               0542 START2  STA    0,X         SIZE THE MEMORY
02B9 A184               0543         CMPA   0,X
02BB 2607               0544         BNE    START3      DIDN'T STORE OK
02BD 6F80               0545         CLR    X+          CLEAR OUT THE $AA
02BF BC010C             0546         CMPX   MEMMAX      DON'T OVERRUN THE TOP
02C2 25F3               0547         BLO    START2      NOT DONE YET
02C4 BF010C             0548 START3  STX    MEMMAX
02C7 2005               0549         BRA    START5
02C9 8C3800             0550 START4  CMPX   #$3800      MIN MEMORY REQD FOR BASIC
02CC 25E4               0551         BLO    START1      YMEMAX TOO SMALL; SIZE MEM
02CE 7F010D             0552 START5  CLR    MEMMAX+1    FORCE BOUNDARY
02D1 BD1865             0553         JSR    PORTCN      INITIALIZE CTL PORT
02D4 9E56               0554         LDX    LUPTRX
02D6 9F58               0555         STX    LUBRKX
02D8 7E0B80             0556         JMP    NEW
                        0557
                        0558 ****************************************************
```

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.

7. Speed up your processor.

APPENDIX H

MEMORY USAGE IN BASIC

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

2. Each non-subscripted numeric variable uses 13 bytes.

3. Each numeric array uses 11 bytes + 6 bytes for each element.

4. If the default string length of 32 characters is used, each non-subscripted string variable uses 39 bytes and each string array uses 11 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 Single Density Sequential file opened takes 172 bytes - Single Density Random files take 326 bytes per file opened. Double Density file sizes are 326 (Sequential) and 582 (Random). 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: Command "L" and a Binary Load Program

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

```
$ or *                                  * = Mikbug and
$ or *L                                   Smartbug prompt
$ or *G 03 E0 E12B 0100 A042            $ = Swtbug prompt
$ or *G

    COMPUTERWARE BASIC                  Basic's ID and
    VERSION: CASS - 4.3                 Version Number

    BASIC
    #                                   # is the prompt
                                          in BASIC
```

SEQUENCE OF EVENTS:

System prompt 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 system prompt 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.

Side #1 of the Basic tape is configured for Port #1 as control terminal, using an ACIA type I/O. First the Binary Load - then a MIKBUG type load. Side #2 is configured for the control port being #2 - otherwise it's the same as side #1. Refer to Appendix K for more information on how the Logical I/O Basic works. 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:

Refer to Appendix F for Source Listing information, and to Appendix K for Logical I/O information.

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.

PAGE is set to 66.

PDLM is set to '~' (tilde).

APPENDIX K

MODIFYING LOGICAL I/O BASIC'S I/O

The new logical I/O drivers in CSS Basic makes the Basic essentially self contained. Cassette/Prom Basic use the upper scratch pad area -- all the Basics reference $E0E3 (Control) and $A008 (Stack Pointer) and the Disk versions reference the DOS -- these are the only external references.

The philosophy behind using Logical I/O is that the user may easily modify the Basic to interface to virtually any I/O device or to special machine language subroutines for features that are not included in the Basic. THE EASIEST WAY TO GET DATA INTO OR OUT OF BASIC IS THRU THE I/O ROUTINES. This way, all of Basic's edits are performed and data is normalized so that when referenced later by the program there will not be problems associated with invalid data (maybe bad data - but at least syntactially correct!)

THE CONFIGURATION BYTE

The following is a description of the available configurations in Basic:

$8X - Control terminal 'x' may be 0,1,2, or 4. The Control Terminal is not initialized. The 'X' will determine the type of I/O that will be performed.

$40 - SWTBUG optional port for cassette - Cassette Basic only - if used, it must be for Port #0 and the SWTBUG command 'OE' needs to be executed prior to entering Basic. The scratch pad location $A00B is referenced.

$20 - If the user is still using an MP-C type of I/O card with either SWTBUG or MIKBUG the Config byte should be set to $A0 (ctl port & MP-C) for Logical Unit #1 and the jumps to CHROUT and CHRIN changed to OUTEEE and INEEE.

$10 - Input only from a parallel device (either side of an MPL-A)

$08 - Output only to a parallel device (either side of an MPL-A)

$04 - Input (side B) and Output (side A) on a parallel device. This is the way Basic used the parallel I/O ports in the past.

$02 - Serial I/O with X64 clock - this is not standard - but we understand that some 6800 Computers (SSB) will be using a times 64 clock, rather than a times 16. Consult your dealer or the Manufacturer for more specifics.

$01 - Garden Variety Serial I/O - typically your CTL I/O unit will be $81.

$00 - Other - see Table description for more information.

ADDRESS OF I/O DEVICE

Basic doesn't really care what address you assign to an I/O device. There is no longer an ERROR #26 in Basic - if there isn't an I/O device at the address you specify, you may lose control of Basic - so be very careful when modifying the I/O addresses. For the standard I/O types, Basic will assume a two byte location - ie. Basic supports Dual serial I/O cards and also both sides of a parallel card as individual I/O locations.

LOGICAL I/O JUMP TABLE

This is where you may use the 'ports' of Basic for your own routines If you specify $00 for the configurator byte - Basic will do nothing in it's I/O routines, but all of the I/O jumps in the jump table are made. Thus if you modify the jump table to go to your own routines, you will be able to pass data to Basic using INPUT or PRINT, etc. or just go execute some code that you want executed.

THE CONTROL PORT

With DOS68.50, the control port address will be picked up from the Parameter Table and will always be logical port #2. If you aren't using DOS68.50 it should be assigned to the port your operating system talks thru. For SWTBUG, this is #1 and for SMARTBUG it is #2. There are two locations that MUST be modified when changing the location of the control port. The first is CNTPRT which is located at $010B - this is equal to the number of the port ie. 1,2,3,4 etc. The other location is configuration byte of the logical unit corresponding to the number that was put into CNTPRT.

MODIFY AND SAVING BASIC

Prior to getting too involved in modifying Basic, the user should establish the size of Basic. The starting address for all versions is $0100 and the transfer (execution beginning) address is $0100. The easiest way of getting a 'safe' ending address is to memory examine locations $0109 and $010A, which gives you the ending address of Basic and work areas. This is slightly greater than what really needs to be saved, but not by much. To get the exact location, SSB Disk users can 'FIND' the Basic PRIOR to loading it into memory - and Cassette users can display the MIKBUG load of the cassette in the local mode on their terminal and take note of the address where the load ends.

After knowing the starting, ending and transfer addresses, make your modifications and then re-save the Basic.

APPENDIX L

LOADING AND USING RANDOM DISK BASIC

There are four (4) copies of RANDOM BASIC on the 6800 disk: 1) RBAS90.$ - BASIC for $6000 - $7FFF DOS with I/O located at $8000. 2) DBAS90.$ - same BASIC for $C000 - $DFFF DOS. 3). CH6BAS.$ - BASIC for $6000 - $7FFF DOS with I/O located at $F7E0. 4). CHCBAS.$ - same BASIC for $C000 - $DFFF DOS. There are two (2) versions of BASIC on the 6809 disk: 1). DBAS20.$ - BASIC for $C000 - $DFFF DOS and I/O located at $E000. 2). CHBS20.$ - same BASIC with I/O located at $F7E0.

After you've selected the version on Random Basic that suits your needs (and is compatible with your system) simply type the name of the Basic and it will load and begin execution. If this does not happen, check the following:

1) Does the version of BASIC match the version of DOS you are using - if yes goto 2.

   a) correct situation and try again!!

2) If the COMPUTERWARE logo prints but then BASIC goes away, it found your I/O port but doesn't like your memory.

   a) Check MEMAX
   b) Swap memory boards around.

The above covers most problems people have in loading Basic. If after trying these you are still unable to load Basic, make a disk with your DOS, the version of Basic you are trying to use, your overlays, and any other relevant material including a description of your system (operating system, amt. of memory, type of I/O used, etc.) and send it to Computerware - Box 668 - Encinitas, Calif. 92024. We will research the problem and try to help you resolve it. The above proceedure also applies to any type of problem that you encounter in using the Basic - also a sample program must be enclosed that demonstrates the problem. NOTE: If you want us to help you - follow the above... Material sent to us that does not include these items will be returned.

RANDOM BASIC'S MEMORY LIMIT WITH DOS68.5X OR HIGHER

Random Basic checks to confirm that it is running with DOS68.5X or higher. If it is, the value that is in MEMAX (Parameter Table) is used for Basic's memory limit. Since Basic must start it's stack on a page boundary, only the high order address byte is used, the low order being forced to zero.

BASIC'S AUTO LOAD AND RUN FEATURE

BASIC NOW allows you to enter the name of the program that you want executed on the DOS command line at the time you call in BASIC. Example: DBAS90,START.BAS will load BASIC, which will then load the program START.BAS and begin execution. CAUTION: If you use this technique in your START.UP file, SET MEMAX to a value that protects the version of EXEC that you are using.

06A7 : 81 55

#23

5#6 = 5^6

03A7 A7 01 => A701 [illegible] backspace before 'EDIT' 00

PORT #	ASSIGNED
0	DOS SYSTEM PRINT DRIVER
1	Console port #1
2	#F5D4 Serial port
3	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]

[illegible] = Parallel port #F418

[illegible] = Default [illegible] Stack