31336 VIA COLINAS, WESTLAKE VILLAGE, CA 91361

SMOKE SIGNAL BROADCASTING

DFB-68
DISK FILE BASIC

SSB BASIC TABLE OF CONTENTS

INTRODUCTION

SSB's Disk File Handling BASIC, DFB-68, was 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. SSB BASIC supports many transcendental functions, allows programs and data to be saved on disc, and implements file handling capability.

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

BASIC is distributed in three versions. These are named on the disk as follows:

```
BASIC.$    for use with DOS68 in the $6000-$7FFF range
BASICA.$   for use with DOS68 in the $A000-$BFFF range
BASICC.$   for use with DOS68 in the $C000-$DFFF range
```

To run BASIC, simply put the disk containing BASIC into drive 0 and type BASIC followed by a carriage return. (This assumes you are using DOS68 at $6000-$7FFF.)

LICENSE

The Smoke Signal BASIC, in all machine readable formats, and written documentation accompanying them are copyrighted. The purchase of SSB BASIC, or the purchase of a disc system with which SSB BASIC is distributed without additional charge, conveys to the purchaser a license to copy BASIC for his/her own use on any disc system manufactured by Smoke Signal Broadcasting, and not for sale or free distribution to others. No other license, expressed or implied is granted.

WARRANTEE INFORMATION

The license to use SSB BASIC is sold AS IS without warrantee. This warrantee is in lieu of all other warrantees expressed or implied. Smoke Signal Broadcasting 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.

MODES OF EXECUTION

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

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

```
10 PRINT "SSB BASIC"
```

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

```
PRINT "SSB BASIC"
```

however, causes the terminal to respond immediately with:

```
SSB 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.

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 Ø 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 where 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 8Ø characters including spaces.

7. Spaces are not processed by BASIC unless they are part of a character string (i.e., enclosed in quotation marks). The use of spaces is optional. The line 1Ø LET A = 1Ø is the same as the line 1ØLETA=1Ø. 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:

```
1Ø A=4: B=7: C=A+B: PRINT C
```

is equivalent to:

```
1Ø A=4
2Ø B=7
3Ø C=A+B
4Ø PRINT C
```

DATA FORMAT

The range of numbers that can be represented is 1.Ø E-99 to 9.99999999 E+99 where E+99 represents 1Ø to the power 99.

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

```
1234      12.34      1234 E-2
```

NUMERIC VARIABLES

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

Examples: X, Y, Z, X3, Q8

STRING VARIABLES

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

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

These string variables are all distinct from numeric variables having the same name. For example, X=9Ø2, X$="POLLY", Y(5)=23, and Y$(5)="CRACKERS" are all legal and may appear in the same program.

STRING CONCATENATION

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

For example:

```
1Ø X$="SSB"
2Ø Y$=" BASIC"
3Ø Z$=X$ + Y$
4Ø PRINT Z$
```

Will print: SSB 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), 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 recieve commands, "BASIC#" will be displayed on the terminal. After each entry, the system will prompt the operator with a "#".

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

LIST

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

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

RUN

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

CONT

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

AVAIL=(size of available memory in decimal)
PROG=(size of program in decimal)
VAR=(size of variable storage area)

MON

This command causes the computer to return to the resident ROM monitor in the computer system. In the case of MIKBUG this will output a carriage return, line feed, and the "*" prompt character. If the program counter address (stored in $A048 and $A049) is not altered, 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 command functions identically as MON except that control is return to DOS68.

PORT

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

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

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 Ø. FLIST 2 will list the file directory on disc 2. FLIST #4,1 will list the disc 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 disc files without exiting back to DOS68 to use the DELETE command. The format of the FDEL command is: FDEL,<file list>.

FREN

The FREN (file rename) command functions just as the DOS68 RENAME command does to change the name of a disc file. The command format is: FREN,<old file name>,<new file name>.

HOUSEKEEPING COMMANDS

The following three commands, LINE, DIGITS, and STRING allow the user to define the associated parameters. Once these commands are used, the values assigned remain the same until the commands are used again or BASIC is reload from the disc. LINE and DIGITS can be used more than once during a program; STRING cannot. Although these parameters have default values, the default values are not asserted after each program. Once these commands are used, the values remain in affect until they are explicity changed.

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 the line length to zero also disables this function.

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.

STRING

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

BASE

The command BASE=Ø will cause array subscripts to begin with the number Ø. The command BASE=1 will cause array subscripts to begin with the number 1 which is the default value. To conform to the proposed ANSI standard, the BASE command may be entered in the format: OPTION BASE=.

HOME

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

SKIP

The SKIP command is used to skip X print lines. SKIP 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. Appendix F gives the location of this sequence for your reference and modification.

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.

SAVING AND LOADING BASIC PROGRAMS

SSB BASIC was written to allow for convenient use of the BFD-68 disc system. This section describes how to save and load programs with the disc system. Also, cassette commands are included to provide an easy transition to disc files.

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.

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 end of the program already in memory.

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.

TSAVE

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

TSAVE allows a single letter file name to be entered in the following format: TSAVE D or TSAVE #3 D. This letter will be output to the tape ahead of the source program.

TLOAD

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

the BASIC source buffer. The buffer is automatically cleared at the beginning of a TLOAD command.

If TLOAD is used with the filename option (TLOAD D or TLOAD #3 D) only a source program with that file name will be loaded. If a file name was not specified, the first source program encountered will be loaded.

TPEND

The TPEND command is identical to the TLOAD command except that the current BASIC buffer is not cleared.

The 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.

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 upredicatable result.

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 arguements.

RND

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

```
10 PRINT ((J-I+1)*RND(Ø)+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 no action is taken. The left-hand margin is print position #1. For example, to print A$ starting in column 25:

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

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

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

INT

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

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

ABS

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

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

SGN

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

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

POS

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

```
1Ø PRINT TAB(1); X;
2Ø IF POS = 71 THEN PRINT A$
```

LEN

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

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

ASC

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

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

CHR$

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

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

VAL

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

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

STR$

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

```
10 LET G = 4918 + 2
20 LET M$ = STR$(G)
```

The variable M$ would contain "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)
```

The variable A$ now contains the string "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)
```

The variable S$ now holds the string "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.

TRANSCENDENTAL FUNCTIONS

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

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

USER

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

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

STATEMENTS

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

POKE

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

DIM

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

An array can only be "dimensioned" once in a program, but need not be dimensioned at all. If a subscripted variable is encountered prior to dimensioning, a default of 10 elements is established for the array. Only the variables A - Z followed a $ may be dimensioned for string arrays. The maximum dimension size is 255 - which will provide an array that has 256 variable proisitons 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.

Examples:

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

REM

The REMark statement is a nonexecutable statement which gives the user the ability to document his program. By including remark statements with the program source the listing becomes more readable.

DEF

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

DATA AND READ STATEMENTS

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

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

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

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

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

The statements shown above are equivalent to the following:

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

RESTORE

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

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

is equivalent to:

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

LET

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

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

FOR --- NEXT STATEMENTS

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

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

The FOR - NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times before exiting from the loop. The variable in the FOR statement (shown above as "I") is initially set to the value of the first expression. Subsequently, the statements following the FOR are executed.

When the NEXT statement is encountered the STEP value is added to the variable and program execution is resumed at the statement following the FOR - TO statement. If the addition of the STEP results in a sum greater than the expression that follows TO, the NEXT instruction executed will be the one following the NEXT

statement. If no STEP is specified, the value of 1 is assumed. If the TO value is less than the initial value, the FOR - NEXT loop will be executed only once. For example:

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

This example will print: 1 1.5 2 2.5 3 DONE

Although expressions are permitted for the initial, final, and step values in the FOR statement, they will be evaluated only once (the first time the loop is entered). The same index variable cannot be used in two different loops if the loops are nested together. When the statement after the NEXT statement is executed, the variable is equal to the last value assigned, i.e. the value which caused the loop to stop.

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.

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 used at all.

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.

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, truncated 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 = 1 GOTO 700
If N = 2 GOTO 350
If N = 3 GOTO 490
If N = 4 GOTO 450
If N > 4 an error will result
```

IF --- THEN

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

For example:

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

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

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

To perform:

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

use the following:

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

To perform the function:

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

use the following:

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

INPUT/OUTPUT STATEMENTS

Any INPUT or PRINT statement may be followed with an #N where N is the I/O port number (0-7). N may be a constant, variable, or an expression. If no port number is specified, the control port (port #1) is assumed. If any instruction 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, having up to the maximum number of characters as specified by the string length command, to be entered. The values input are assigned to the variables specified in the INPUT statement.

Multiple inputs can be entered by separating them by 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 "?", and wait for the input of the variable requested.

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 no data is entered, or a non-numeric character is entered when a numeric variable is required, the system will prompt the user with "RE-ENTER".

PRINT

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

Examples:

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

OPEN

The command OPEN #(FLN),(FILE SPEC) is used to open a disc file. The file number, (FLN), is an expression that must evaluate to the range 0 through 9. The file specification, (FILE SPEC), must be string variable or string literal which supplies the file name in standard DOS68 format.

The type of file access (read or write) will be determined by the first usage of the file after opening. Before a BASIC program can read input or write output to a file, the file must have previously been opened by the OPEN statement.

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

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

CLOSE

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

```
100 CLOSE #1,#2
```

READ

The statement READ #(FLN),(VARIABLE LIST) is provided to request data be read from a disk file. Input from a file is taken an item at a time - as the program needs it. (VARIABLE LIST) consists of one or more variables, either string or numeric, separated by commas. If the receiving element is a string variable, it will receive the data from the file up to the maximum string length of 80 characters. The line input buffer for a single item from a file is 80 characters.

A string item over 80 characters will be truncated, and if more then 80 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.

RESTORE

The statement RESTORE #(FLN),#(FLN),... causes the files associated with the list of file numbers to be repositioned to the beginning of the file. Thus, the data in the file may be reread. Note that this statement functions for files just as the RESTORE dscribed earlier functions for DATA statements. The file number may be that of file which is open for reading (input) or writing (output).

```
10 OPEN #1,"PART.MST"   (Quotes are not required)
20 LET C = 5
30 READ #1,A
40 FOR I = 1 TO A
50 READ #1,B
60 PRINT B
70 NEXT I
```

```
80 RESTORE #1
90 LET C=C-1
100 IF C <> 0 THEN 30
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. The first item is read in from the the file and in expected to hold the number of items to follow. Data is read and printed until the item limit is reached. 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 disc 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.

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

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

WRITE

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

STATUS

STATUS #(FLN) is a function allowing for monitoring of error status of any specified file number. The status most often used is the end-of-file (EOF) which has a value of 6. The status number is that number returned by the disc file management system. Refer to the BFD-68 SYSTEM MANUAL for other values. It is a good idea to check the file status after a READ at least for end of file. 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 (Ø) means the file is there and non-zero means not there.

AUTO RUN

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

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

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

2) Type DOS to exit to the operating system.

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

4) Restart DOS68 by the warm start entry point

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

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

If you append the auto-run file onto a copy of DOS68, then you can have Basic come up running as part of your DOS BOOT.

APPENDIX A: QUICK BASIC REFERENCE

COMMANDS (used in direct mode)

APPEND	PORT
CONT	RUN
DOS	SAVE
DIGITS	*STRING
FLIST	TLOAD
*LINE	TPEND
*LIST	*TRACE OFF
LOAD	*TRACE ON
*MON	TSAVE
NEW	

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

STATEMENTS (used in programs)

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

* commands that can be used within a program

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

DISK FILE COMMANDS:

OPEN CLOSE RESTORE SCRATCH READ WRITE STATUS

MATHEMATICAL OPERATORS

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

RELATIONAL OPERATORS

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

PRECEDENCE OF OPERATORS

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

APPENDIX B: CHARACTER CONVERSION TABLE

ASCII	CNTL	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC
NUL	@	00	00	,	2C	44	X	58	88
SOH	A	01	01	-	2D	45	Y	59	89
STX	B	02	02	.	2E	46	Z	5A	90
ETX	C	03	03	/	2F	47	[	5B	91
EOT	D	04	04	0	30	48	\	5C	92
ENQ	E	05	05	1	31	49	]	5D	93
ACK	F	06	06	2	32	50	^	5E	94
BEL	G	07	07	3	33	51	_	5F	95
BS	H	08	08	4	34	52	`	60	96
HT	I	09	09	5	35	53	a	61	97
LF	J	0A	10	6	36	54	b	62	98
VT	K	0B	11	7	37	55	c	63	99
FF	L	0C	12	8	38	56	d	64	100
CR	M	0D	13	9	39	57	e	65	101
SO	N	0E	14	:	3A	58	f	66	102
SI	O	0F	15	;	3B	59	g	67	103
DLE	P	10	16	<	3C	60	h	68	104
DC1	Q	11	17	=	3D	61	i	69	105
DC2	R	12	18	>	3E	62	j	6A	106
DC3	S	13	19	?	3F	63	k	6B	107
DC4	T	14	20	@	40	64	l	6C	108
NAK	U	15	21	A	41	65	m	6D	109
SYN	V	16	22	B	42	66	n	6E	110
ETB	W	17	23	C	43	67	o	6F	111
CAN	X	18	24	D	44	68	p	70	112
EM	Y	19	25	E	45	69	q	71	113
SUB	Z	1A	26	F	46	70	r	72	114
ESC	[	1B	27	G	47	71	s	73	115
FS	\	1C	28	H	48	72	t	74	116
GS	]	1D	29	I	49	73	u	75	117
RS	^	1E	30	J	4A	74	v	76	118
US	_	1F	31	K	4B	75	w	77	119
SP		20	32	L	4C	76	x	78	120
!		21	33	M	4D	77	y	79	121
"		22	34	N	4E	78	z	7A	122
#		23	35	O	4F	79		7B	123
$		24	36	P	50	80		7C	124
%		25	37	Q	51	81		7D	125
&		26	38	R	52	82		7E	126
'		27	39	S	53	83	DEL	7F	127
(		28	40	T	54	84			
)		29	41	U	55	85			
*		2A	42	V	55	85			
+		2B	43	W	57	87			

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 returnline feed string
0112 - 0113	Address of ERROR routine (Error in ACCB)
0114 - 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)
013F	Jump Table

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

A04A - A07F	Part of this area is used by DOS68. See the DOS68 reference manual for the actual addresses used.

APPENDIX D: ERROR MESSAGES

The following is printed when an error occurs:

ERROR # ----- IN LINE # -----

The line number will be 0000 for errors in direct mode execution.

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 80 characters on input line)
22	Syntax error in DEF statement
23	FN function error. Either syntax error in FN or FN is not define
24	Error in STRING usage, or mixing of numeric and string variables
25	String buffer overflow, or string extract too long
26	Not used in Logical I/O Basic
27	VAL function error - string starts with a non-numeric character
28	Cannot take LOG of a negative number
29	Error message error
30	File number is not in range of 0 through 9
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	Diskname (FREN) error
41	Memory error

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

APPENDIX E: EXAMPLE FOR USER FUNCTION

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

APPENDIX F: PARTIAL SOURCE LISTING

```
                        0175 *
0100                    0176        ORG    $100
                        0177 *
0100 BD 02A6            0178 BEGIN  JSR    START       COLD START
0103 BD 0C31            0179        JSR    RSTART      SOFT START
0106 7E 0C8D            0180        JMP    AUTO        AUTO RUN
                        0181 *
0109 28 73              0182 BUFBAS FDB    SRCBEG      END OF BASIC & WORK AREAS
010B 02                 0183 CNTPRT FCB    2           CONTROL PORT
010C 60 00              0184 MEMMAX FDB    $6000       MEMORY LIMIT - YMEMAX FOR DOS68.50
010E 0B BD              0185 HOMSTR FDB    HOMLIS      HOME/CLEAR EOF
0110 03 BA              0186 CRLFST FDB    CRLF2       CR/LF STRING
0112 0D 2A              0187 ERRPNT FDB    ERROR       ADDR OF ERROR RTN (# IN ACCB)
                        0188 *
0114 43                 0189 NOTICE FCC    'COPYRIGHT 1979 SMOKE SIGNAL BROADCASTING'
                        0190 *
013C 00 00              0191        FDB    $0000,$0000 RESERVED FOR FUTURE
0140 00 00              0192        FDB    $0000,$0000 JUMP ADDRESSES
0144 00 00              0193        FDB    $0000,$0000
                        0194 *
0148 18                 0195 DELINE FCB    $18         CTL X
0149 08                 0196 DELCHR FCB    $08         CTL H
014A 00                 0197 BSECHO FCB    $00         NULL
014B 03                 0198 BRKCHR FCB    $3          CTL C
                        0199
                        0200
                        0201 * I/O DEFINITION TABLE  -  CONFIGURATION BYTE
                        0202 *
                        0203 *BIT#|  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
                        0204 *    |     |     |     |     |     |     |     |     |
                        0205 *HEX | 80  | 40  | 20  | 10  |  08 |  04 |  02 |  01 |
                        0206
                        0207 *      CTL    RES   MPC   PIA   PIA   STD   X64   X16
                        0208 *      TERM   CAS   PIA    IN   OUT   PIA   SER   SER
                        0209
                        0210 * CONFG = 0 -> NON-STD I/O (ie. Video Board - Graphics)
                        0211
                        0212 * LOGICAL UNIT TABLE ENTRY
                        0213
                        0214 *   0    -> CONFIGURATION BYTE
                        0215 *  1-2   -> ADDRESS OF I/O DEVICE
                        0216 *  3-5   -> JUMP CHAR OUTPUT
                        0217 *  6-8   -> JUMP CHAR INPUT
                        0218 *  9-11  -> JUMP PORT INITIALIZATION
                        0219
                        0220
014C                    0221 LUTBLE EQU    *
                        0222 *
                        0223 * LOGICAL UNIT #0
                        0224 *
014C 01                 0225 LU0    FCB    $01
014D F7 E0              0226        FDB    $F7E0
014F 7E 01FC            0227        JMP    CHROUT
0152 7E 022C            0228        JMP    CHRIN
0155 7E 01AC            0229        JMP    IOINIT
                        0230 *
                        0231 * LOGICAL UNIT #1
                        0232 *
0158 01                 0233 LU1    FCB    $01
```

```
0159 F7 E4   0234         FDB   $F7E4
015B 7E 01FC 0235         JMP   CHROUT
015E 7E 022C 0236         JMP   CHRIN
0161 7E 01AC 0237         JMP   IOINIT
             0238 *
             0239 * LOGICAL UNIT #2
             0240 *
0164 81      0241 LU2     FCB   $81
0165 F7 E8   0242 CTLADR  FDB   $F7E8       CHGD TO YCPORT IF DOS68.50
0167 7E 72C1 0243 CTLOUT  JMP   ZPUTCH      *** SEE NOTE BELOW
016A 7E 72C4 0244 CTLIN   JMP   ZGETCH      *** SEE NOTE BELOW
016D 7E 01AC 0245         JMP   IOINIT
             0246 *
             0247 * NOTE: TO HAVE BASIC'S CONTROL PORT NOT USE
             0248 *       THE WIDTH, DEPTH, ETC. PARAMETERS THAT
             0249 *       YOU HAVE SET FOR DOS - CHANGE THE JUMP
             0250 *       AT CTLOUT TO GOTO 'CHROUT' IN BASIC &
             0251 *       THE JUMP AT CTLIN TO GOTO 'CHRIN'
             0252
             0253
             0254 * LOGICAL UNIT #3
             0255 *
0170 01      0256 LU3     FCB   $01
0171 F7 EC   0257         FDB   $F7EC
0173 7E 01FC 0258         JMP   CHROUT
0176 7E 022C 0259         JMP   CHRIN
0179 7E 01AC 0260         JMP   IOINIT
             0261 *
             0262 * LOGICAL UNIT #4
             0263 *
017C 04      0264 LU4     FCB   $04
017D F7 F0   0265         FDB   $F7F0
017F 7E 01FC 0266         JMP   CHROUT
0182 7E 022C 0267         JMP   CHRIN
0185 7E 01AC 0268         JMP   IOINIT
             0269 *
             0270 * LOGICAL UNIT #5
             0271 *
0188 04      0272 LU5     FCB   $04
0189 F7 F4   0273         FDB   $F7F4
018B 7E 01FC 0274         JMP   CHROUT
018E 7E 022C 0275         JMP   CHRIN
0191 7E 01AC 0276         JMP   IOINIT
             0277 *
             0278 * LOGICAL UNIT #6
             0279 *
0194 04      0280 LU6     FCB   $04
0195 F7 F8   0281         FDB   $F7F8
0197 7E 01FC 0282         JMP   CHROUT
019A 7E 022C 0283         JMP   CHRIN
019D 7E 01AC 0284         JMP   IOINIT
             0285 *
             0286 * LOGICAL UNIT #7
             0287 *
01A0 04      0288 LU7     FCB   $04
01A1 F7 FC   0289         FDB   $F7FC
01A3 7E 01FC 0290         JMP   CHROUT
01A6 7E 022C 0291         JMP   CHRIN
01A9 7E 01AC 0292         JMP   IOINIT
             0293 ----------------------------------------
```

```
                    0294
                    0295 * INITIALIZE I/O PORT AS SPECIFIED
                    0296
01AC DE 01          0297 IOINIT LDX   LUPTRX
01AE E6 00          0298        LDA B CONFG,X
01B0 EE 01          0299        LDX   OPORT,X
01B2 C4 9F          0300        AND B #$9F          CLEAR NU BITS
01B4 2B 14          0301        BMI   IOINTR        CONTROL TERM
01B6 27 12          0302        BEQ   IOINTR        OTHER TYPE IO
01B8 C5 1C          0303        BIT B #%11100       PIA TYPE
01BA 26 0F          0304        BNE   IOPIA         YES
01BC 37             0305        PSH B
01BD BD 0F22        0306        JSR   DELAY1
01C0 33             0307        PUL B
01C1 86 03          0308        LDA A #3
01C3 A7 00          0309        STA A 0,X           MASTER CLEAR ACIA
01C5 86 14          0310        LDA A #$14          WORD SELECT BITS
01C7 1B             0311        ABA                 ADD DIVIDE SELECT
01C8 A7 00          0312        STA A 0,X
01CA 39             0313 IOINTR RTS
01CB C5 04          0314 IOPIA  BIT B #%100         STD PIA
01CD 27 13          0315        BEQ   IOPIA1        NO
01CF 6F 01          0316        CLR   1,X           INIT A AND B SIDES
01D1 6F 03          0317        CLR   3,X
01D3 6F 00          0318        CLR   0,X
01D5 6F 02          0319        CLR   2,X
01D7 63 00          0320        COM   0,X
01D9 86 3E          0321        LDA A #$3E
01DB A7 01          0322        STA A 1,X
01DD 86 2E          0323        LDA A #$2E
01DF A7 03          0324        STA A 3,X
01E1 39             0325        RTS
01E2 C5 08          0326 IOPIA1 BIT B #%1000        OUTPUT PIA
01E4 27 0B          0327        BEQ   IOPIA2        NO
01E6 6F 01          0328        CLR   1,X           SET PIA A FOR OUTPUT
01E8 86 FF          0329        LDA A #-1
01EA A7 00          0330        STA A 0,X
01EC 86 3E          0331        LDA A #$3E
01EE A7 01          0332        STA A 1,X
01F0 39             0333        RTS
01F1 6F 01          0334 IOPIA2 CLR   1,X           SET PIA A FOR INPUT
01F3 86 00          0335        LDA A #0
01F5 A7 00          0336        STA A 0,X
01F7 86 2E          0337        LDA A #$2E
01F9 A7 01          0338        STA A 1,X
01FB 39             0339        RTS
                    0340 ----------------------------------------
                    0341
                    0342 * DO CHARACTER OUTPUT
                    0343
01FC 81 7F          0344 CHROUT CMP A #$7F
01FE 27 16          0345        BEQ   CHROR
0200 E6 00          0346        LDA B CONFG,X
0202 27 12          0347        BEQ   CHROR         NO OUTPUT REQ'D
0204 EE 01          0348        LDX   OPORT,X
0206 C5 0C          0349        BIT B #%1100        PIA OUTPUT
0208 26 0D          0350        BNE   OUTPIA        YES
020A C5 03          0351        BIT B #%11          ACIA OUTPUT
020C 27 08          0352        BEQ   CHROR         NO
020E E6 00          0353        LDA B 0,X           WAIT FOR TDRE
```

```
0210 C5 02   0354         BIT B #2
0212 27 FA   0355         BEQ   *-4
0214 A7 01   0356         STA A 1,X          OUT CHAR
0216 39      0357 CHROR   RTS
0217 0F      0358 OUTPIA  SEI                LOCK INTERRUPTS
0218 A7 00   0359         STA A 0,X          OUTPUT CHAR
021A C6 36   0360         LDA B #$36
021C E7 01   0361         STA B 1,X
021E C6 3E   0362         LDA B #$3E
0220 E7 01   0363         STA B 1,X
0222 96 DF   0364         LDA A INTRP        RESTORE INTERRUPT STATE
0224 06      0365         TAP
0225 E6 01   0366         LDA B 1,X          WAIT FOR CHAR TRANSFER
0227 2A FC   0367         BPL   *-2
0229 A6 00   0368         LDA A 0,X          CLEAR INTR FLAG
022B 39      0369         RTS
             0370 ----------------------------------------
             0371
             0372 * DO CHARACTER INPUT
             0373
022C E6 00   0374 CHRIN   LDA B CONFG,X
022E 27 2D   0375         BEQ   CHRINR
0230 EE 01   0376         LDX   OPORT,X
0232 C5 14   0377         BIT B #%10100      PIA INPUT
0234 26 15   0378         BNE   INPIA        YES
0236 C5 03   0379         BIT B #%11         ACIA INPUT
0238 27 23   0380         BEQ   CHRINR       NO
023A E6 00   0381 INACIA  LDA B 0,X          WAIT FOR RDRF
023C 57      0382         ASR B
023D 24 FB   0383         BCC   INACIA
023F A6 01   0384         LDA A 1,X          GET CHAR
0241 84 7F   0385         AND A #$7F         MASK PARITY
0243 81 7F   0386         CMP A #$7F         DEL CHAR
0245 27 F3   0387         BEQ   INACIA       YES,GET NEXT
0247 DE 81   0388 INECHO  LDX   LUPTRX       ECHO CHAR OUT - RTS FOR NO ECHO
0249 6E 03   0389         JMP   OFFOUT,X
024B C5 04   0390 INPIA   BIT B #%100        STD PIA
024D 27 08   0391         BEQ   INPIA2       NO
024F E6 03   0392         LDA B 3,X          WAIT FOR INTR ON B
0251 2A FC   0393         BPL   *-2
0253 A6 02   0394         LDA A 2,X          READ B SIDE
0255 20 F0   0395         BRA   INECHO       ECHO CHAR
0257 E6 01   0396 INPIA2  LDA B 1,X          WAIT FOR INTR ON A
0259 2A FC   0397         BPL   INPIA2
025B A6 00   0398         LDA A 0,X          READ A SIDE
025D 39      0399 CHRINR  RTS                NO ECHO, INPUT ONLY
             0400 ----------------------------------------
             0401
```

```
                    0403 * PROCESS CHECK FOR BREAK
                    0404
025E 37             0405 BREAK   PSH B
025F 36             0406         PSH A
0260 DF 7F          0407         STX    IOSAVX
0262 DE 83          0408         LDX    LUBRKX       USE DEFAULT PORT ONLY
0264 E6 00          0409         LDA B  CONFG,X
0266 EE 01          0410         LDX    OPORT,X
0268 C5 37          0411         BIT B  #%110111     ACIA,PIA INPUT,MPC
026A 27 1A          0412         BEQ    BREAK2       NO
026C C5 03          0413         BIT B  #%11         ACIA
026E 26 1B          0414         BNE    BREAK3       YES
0270 C5 14          0415         BIT B  #%10100      PIA INPUT
0272 26 1E          0416         BNE    BREAK4       YES
0274 C5 20          0417         BIT B  #%100000     MPC INPUT
0276 26 28          0418         BNE    BREAK7       YES
0278 20 0C          0419         BRA    BREAK2       WRONG, CONFIGURATION TO BREAK
027A DE 83          0420 BREAK0  LDX    LUBRKX       GOTO INPUT PROCESSOR
027C AD 06          0421         JSR    OFFIN,X
027E B1 014B        0422 BREAK1  CMP A  BRKCHR       BREAK CHAR?
0281 26 03          0423         BNE    BREAK2       NO
0283 7E 0C34        0424         JMP    READY        STOP CYCLING
0286 DE 7F          0425 BREAK2  LDX    IOSAVX       RESTORE AND CONT
0288 32             0426         PUL A
0289 33             0427         PUL B
028A 39             0428         RTS
028B E6 00          0429 BREAK3  LDA B  0,X          ACIA PORT
028D 57             0430         ASR B
028E 24 F6          0431         BCC    BREAK2
0290 20 E8          0432         BRA    BREAK0
0292 C5 04          0433 BREAK4  BIT B  #%100        STD PIA
0294 27 06          0434         BEQ    BREAK6       NO
0296 E6 03          0435         LDA B  3,X          CHECK B SIDE
0298 2A EC          0436 BREAK5  BPL    BREAK2
029A 20 DE          0437         BRA    BREAK0
029C E6 01          0438 BREAK6  LDA B  1,X          PIA INPUT ONLY
029E 20 F8          0439         BRA    BREAK5       CHECK SIDE A
02A0 E6 00          0440 BREAK7  LDA B  0,X          MPC PIA
02A2 2B E2          0441         BMI    BREAK2
02A4 20 D4          0442         BRA    BREAK0       GO INPUT, HOPE MIKBUG INEEE
                    0443 ----------------------------------------
                    0444
                    0445 * BASIC'S START-UP ROUTINE
                    0446
02A6 B6 010B        0447 START   LDA A  CNTPRT       CONTROL PORT#
02A9 97 8B          0448         STA A  CONSOL
02AB 07             0449         TPA
02AC 97 DF          0450         STA A  INTRP        SAVE PROC STATUS
02AE B6 7300        0451         LDA A  YMONV        DOS VERSION
02B1 81 50          0452         CMP A  #$50         IS IT DOS68.50
02B3 26 19          0453         BNE    START4       NO
02B5 CE 0C34        0454         LDX    #READY       BREAK RETURN ADDR
02B8 FF 730D        0455         STX    YABRTV       STORE IN DOS BREAK ADDR
02BB FE 7317        0456         LDX    YCPORT       PARAMETER TABLE CTL PORT ADDR
02BE FF 0165        0457         STX    CTLADR       SAVE IN PORT#2 JUMP TABLE
02C1 B6 7302        0458         LDA A  YMEMAX       PARAMETER TBL MEM LIMIT
02C4 81 38          0459         CMP A  #$38         MIN MEMORY REQD FOR BASIC
02C6 25 06          0460         BLO    START4
02C8 B7 010C        0461         STA A  MEMMAX       SAVE IN BASIC'S MEMMAX
```

```
02CB 7F 010D 0462         CLR   MEMMAX+1     FORCE BOUNDARY
02CE BD 167F 0463 START4  JSR   PORTCN       INITIALIZE CTL PORT
02D1 DE 81   0464         LDX   LUPTRX
02D3 DF 83   0465         STX   LUBRKX
02D5 7E 0BB5 0466         JMP   NEW
             0467 *

             0469 *
             0470 * OUT HEX RTN
             0471
02D8 A6 00   0472 OUTH    LDA A 0,X
02DA 8D 0D   0473         BSR   OUTHL
02DC A6 00   0474         LDA A 0,X
02DE 08      0475         INX
02DF 20 0C   0476         BRA   OUTHR
             0477 *
02E1 8D F5   0478 OUT2HS  BSR   OUTH
02E3 8D F3   0479         BSR   OUTH
02E5 86 20   0480 OUTSP   LDA A #SPACE
02E7 20 0E   0481         BRA   OUTCH
             0482 *
02E9 44      0483 OUTHL   LSR A
02EA 44      0484         LSR A
02EB 44      0485         LSR A
02EC 44      0486         LSR A
02ED 84 0F   0487 OUTHR   AND A #$F
02EF 8B 30   0488         ADD A #'0
02F1 81 39   0489         CMP A #'9
02F3 23 02   0490         BLS   OUTCH
02F5 8B 07   0491         ADD A #7
             0492 *
             0493 * OUTPUT CHAR RTN
             0494 *
02F7 37      0495 OUTCH   PSH B
02F8 36      0496         PSH A
02F9 BD 025E 0497         JSR   BREAK
02FC DF 7F   0498         STX   IOSAVX
02FE DE 81   0499         LDX   LUPTRX       10/26
0300 AD 03   0500         JSR   OFFOUT,X
0302 20 82   0501         BRA   BREAK2
             0502 *
             0503 * INPUT CHAR RTN
             0504 *
0304 DF 7F   0505 INCH    STX   IOSAVX
0306 37      0506         PSH B
0307 DE 81   0507         LDX   LUPTRX       10/26
0309 AD 06   0508         JSR   OFFIN,X
030B 36      0509         PSH A
030C 7E 027E 0510         JMP   BREAK1
             0511 *
```

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.

APPENDIX H: MEMORY USAGE IN BASIC

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

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

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

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

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

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

7. Each file number opened takes 177 bytes. 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.

8 SSB BASIC checks to find out if it is running with DOS68 Version 5. If it is, the value that is in MEMMAX (DOS parameter table) is used for BASIC's memory limit. Since BASIC must start its stack on a page boundary, only the high order address byte is used. The low order byte in MEMMAX will be ignored and BASIC will always use 00 as the low order byte.

APPENDIX I: DEFAULT VALUES ON SYSTEM INITIALIZATION

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.

APPENDIX J: MODIFYING LOGICAL I/O BASIC`S I/O

The new LOGICAL I/O drivers in SSB BASIC makes the basic essentially self contained. All the BASICS reference $E0E3 (control) and $A008 (stack pointer) and the disk versions reference the DOS -- these are the only external references.

The philosohy 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 syntactically 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 determined the type of I/O that will perfromed.

$40 - UNUSED

$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 change 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.
basic used the parallel I/O ports in the past.

$02 - Serial I/O with X64 clock - this will select 110, 134.5 300, 4800, 9600 baud on the CHIEFTAIN microcomputers.

$01 - Serial I/O with a X16 clock as used in the SWTPC 6800 computer, and for the highest baud rates on the CHIEFTAIN.

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

ADDRESS OF I/O DEVICE

Basic doesn't really care what address you assign to and 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 speecify $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 exectued.

THE CONTROL PORT

With DOS68 Version 5, the control port address will be picked up from the parameter table and will always be logical port 2. If you are not using DOS68 Version 5, the control port 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 i.e., 1, 2, 3, 4 etc. The other location is configuration byte of the logical unit corresponding to the number that was put into CNTPRT.

MODIFYING AND SAVING BASIC

Prior to getting too involved in modifying Basic, the user should establish the size of Basic. The starting address for all verisons 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 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.