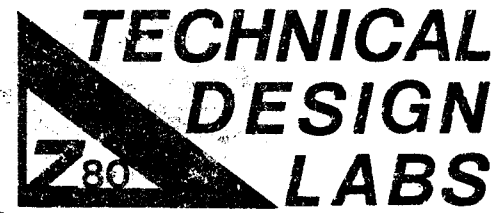
ZAPPLE™

8K BASIC

USER'S MANUAL



ZAPPLE 8K BASIC

USER'S MANUAL

TECHNICAL DESIGN LABS INC.
RESEARCH PARK, BLDG. H
1101 STATE ROAD
PRINCETON, NEW JERSEY 08540

1/1/77

BASIC ERRATA
All Versions

There have been some reports of trouble with the TAB(X), SPC(X), and Insert (editor mode). These have been traced to USER written I/O routines that are not properly setting the A register to the output character. This has been most often with a V.D.M. driver routine. The fix is to place the character which has just been output (from the 'C' reg.) into the 'A' register BEFORE returning. This will fix the problems. (See the RULES regarding the I/O conventions in the ZAP/ZAPPLE Monitor documentation.)

Thanks.

T.D.L.

ZAPPLE BASIC - INTRODUCTION

Welcome to ZAPPLE BASIC. This program functions as a Basic Interpreter occupying 8K of core, and provides some of the most advanced software features of any commercially available Basic.

Zapple Basic offers many unique features, including tremendously powerful I/O handling capability, relocatability, ROMability, unique and powerful commands and a large measure of hardware independence.

All the above are covered in this manual. This is NOT a "How to write Basic Programs" manual. Many excellent texts on this subject have been produced. Your local Computer Store can recommend many such texts.

I/O HANDLING

Oftentimes the lament of the programmer is the lack of source documentation for a Basic Interpreter. This is usually due to the fact that internal I/O routines must be modified to suit the exact configuration of your hardware. The TDL method of I/O handling eliminates this problem in that the source code for the Monitor IS provided, and once modified to your hardware configuration, ALL other TDL software automatically interfaces to your system.

Zapple Basic has this feature of hardware independence. All of its I/O drivers are contained in the Monitor, so interfacing it to your hardware is simple.

To get the most out of BASIC, we highly recommend getting the 2K ZAPPLE Monitor.

LOADING BASIC

Loading of Zapple Basic is very straight-forward. It is loaded using the "R" command of either the Zap or Zapple Monitors. It is provided on paper tape in TDL's relocatable hex file format. It occupies almost exactly 8K of core.

Zapple Basic has been assembled on TDL's relocating macro-assembler. Because of this, Zapple Basic is completly relocatable. It is not necessary to load and run this program at one address only. Within limits, which will be mentioned here, it may be loaded and run at any convenient address by the user.

The procedure for loading the program is very simple. Place the tape in the reader device on the nulls between the serial number and the start of the data. Type on the

console: "R,(x)"(cr): and start the reader.

EXAMPLE: R,200(cr)

will load Basic at address 200H. For the exact details on the operation of the "R" command, see either the Zap or Zapple Monitor Manuals.

After loading, Zapple Basic will NOT sign on. You must begin execution at the address given in the relocation parameter above by typing: G200(cr). Basic will then ask: "Highest Memory?" asking the user to type the upper limit Basic will be allowed to use. A carriage return (cr) will assign all available memory to basic save for a small amount at the top which is reserved for use by the monitor.

The limits on its practical relocatability are goverened by two factors: the buffer storage area required, and the address at which the monitor will be located.

For the first, Zapple Basic requires a buffer space of approximatly 255 bytes. Regardless of the loading address of Basic, this 255 byte buffer resides from address 100H to 1FFH. Thus, the minimum loading address for Basic is 200H. From this it should be evident that this Basic at no time uses any memory below address 100H.

THE MINIMUM LOADING ADDRESS FOR BASIC IS 200H.

As to the second, the Monitors are also relocatable, but we do recommend that they be placed up near the top of memory at F000 (up "out of the way"). Thus the 2K monitor would reside from F000 to F7FF (hex) allowing F800 to FFFF for monitor extension routines. (Such as a VDM driver, Tarbell driver. Etc.). Thus, since the Basic occupies 8K of core, the maximum practical loading address is D000 (hex).

In addition to the factor of relocatability, Zapple Basic is ROMABLE. That is, it does not change any of its internal structure while operating, and therefore may be resident in ROM, PROM, or protected memory. It does not have to be moved down to RAM to then function.

ZAPPLE BASIC JUMP TABLE

All I/O handling for Zapple Basic is done through either the ZAP (1K) or ZAPPLE (2K) monitors. The I/O interfacing is done in the beginning of the Basic Program.

Zapple Basic has, in addition, a recovery address, from which recovery of the program can often be made following a "blow-up".

Zapple Basic also has a "USR" command which allows one's own assembly language routines to be called as part of a Basic program. A jump vector is provided allowing the user

great latitude in this application.

Here the source code of the first part of Zapple Basic is presented. It contains all of the I/O vectors which are necessary for complete user versatility. Note that as part of the code, addresses marked with an apostrophe (') are those addresses which are relocatable. Those without an apostrophe (') are considered absolute, in that they are vectoring to addresses outside of Basic, where they expect to find specific I/O routines.

The specific I/O routines in question are those of the monitor. Although both the monitor and Zapple Basic are relocatable, we recommend placing the monitor (either the Zap or Zapple) as high as possible - usually F000H. Thus Basic expects to find the monitor at that address. The source code of these "jumps to the monitor" are presented so that in the event that you do not wish to, or are not able to have the monitor reside at this address, you may make the necessary, although simple modifications to the program.

Note that the source code below is in TDL's relocating assembler format, in that address information is presented in the "High byte first, low byte second" format. For example, at address 000C' there is a jump to F006 (Hex). With some assemblers this might be construed to be a jump to address 06F0. Also note that any modifications you might make when using the monitor at an address other than F000 would entail changing only the high byte of the stated jump address.

```
0000' C3 XXXX'    BASIC:   JMP   INIT    ;"INITIALIZE"
                                          ENTRY POINT
0003' C3 XXXX'    REST:    JMP   RECOVER;RECOVERY ENTRY
                                          POINT
0006' C3 XXXX'    USR:     JMP   ERROR   ;USER DEF.
0009' C3 F003     CI:      JMP   CIN     ;CONSOLE INPUT
000C' C3 F006     RI:      JMP   RIV     ;READER INPUT
000F' C3 F009     CO:      JMP   CON     ;CONSOLE OUTPUT
0012' C3 F00C     PO:      JMP   WRTV    ;PUNCH OUTPUT
0015' C3 F00F     LO:      JMP   LISTX   ;LIST OUTPUT
0018' C3 F012     CSTS:    JMP   CSTSX   ;CONSOLE
                                          STATUS CHECK
001B' C3 F015     IOCHK:   JMP   IOCHX   ;I/O CONFIG.
                                          CHECK
001E' C3 F018     IOSET:   JMP   IOSTX   ;I/O MODIFCTN.
0021' C3 F01B     MEMSIZ:  JMP   MEMCK   ;MEMORY SIZE CK
0024' C3 F01E     TRAP:    JMP   TRAPX   ;BREAKPOINT ENTRY
```

"X"s are inserted into some jump notations above because the values may change in future versions of Basic, and thus could cause confusion. These addresses are modified in the course of various applications, and all that is needed is the recognition that they lie at the starting

Specifics on making use of the USR command portion of the above are covered in the section of the manual which deals with the USR command.

ERROR MESSAGES

The following is a list of error messages returned by Basic when the particular error has been detected. They are given here without explanation. Within the context of a Basic Program they indicate clearly what is amiss, and are of great use in program debugging.

NEXT W/O FOR
SYNTAX ERROR
RETURN W/O GOSUB
OUT OF DATA
ILLEGAL FUNCTION
ARITHMETIC OVERFLOW
OUT OF MEMORY
UNDEFINED STATEMENT
SUBSCRIPT OUT OF RANGE
RE-DIMENSIONED ARRAY
CAN'T /0
ILLEGAL DIRECT
TYPE MIS-MATCH
NO STRING SPACE
STRING TOO LONG
TOO COMPLEX
CAN'T CONTINUE
UNDEFINED USER CALL
FILE NOT FOUND
ILLEGAL EOF
FILES DIFFERENT
RECOVERED
*INVALID INPUT
*EXTRA LOST

ZAPPLE BASIC COMMAND SET

COMMAND	PURPOSE
ABS	Absolute Value Function
AND	Logical AND operator
ASC	Convert character to numeric value function
ATN	Arctangent function
CHR$	Convert numeric value to character function
CLEAR	Delete all variables and set string space
CONT	Continue program execution from program breakpoint
COS	Cosine function
DATA	Defines constants
DEF	Defines User functions
DELETE	Deletes range of line numbers
DIM	Reserves storage for matrices
EDIT	Invokes the line editor
ELSE	What to do if relational is not true
END	End of program; return to command mode
EXP	Function to return "E" raised to a power
FN	Class of user defined functions
FOR	Sets up a loop
FRE	Function to determine amount of unused memory
GOSUB	Invokes a subroutine
GOTO	Transfer control to another part of the program
IF	Relational test
INP	Input directly from an I/O port
INPUT	Input data from the keyboard
INT	Function returns the integer portion of a number
LEFT$	Returns the left portion of a string
LEN	Returns the length of a string
LET	Logical Assignment
LIST	Lists the program on the console
LLIST	Lists the program on the list device
LLVAR	Lists the program variables on the list device
LNULL	Sets the nulls for the printer
LOAD	Loads a program from the reader
LOG	Returns the natural logarithm of a number
LPOS	Function to return the current position of the list device
LPRINT	Directs the output to the list device
LVAR	Prints the variables on the console
LWIDTH	Sets the width of the list device
MID$	Returns the middle of a string
NEW	Clears all program statements and variables
NEXT	Returns to the beginning of a loop
NOT	Logical "NOT" operator
NULL	Sets the nulls for the console
ON	Indexed transfer of control
OR	Logical "OR" operator
OUT	Output directly to an I/O port
PEEK	Function to return data from a memory location
POKE	Insert data into a memory location

POS	Function to return the correct print head position of the console
PRINT	Directs output to console
RANDOMIZE	Changes the seed used by the psuedo-random number generator
READ	Move data from a DATA statement to a variable
REM	Remarks
RENUMBER	Renumber the program and change line number references
RESTORE	Returns pointer to the beginning of the data statements
RETURN	Return control back from a subroutine
RIGHT$	Function to return the right portion of a string
RND	Function to return a psuedo-random number
RUN	Clear variables and start execution of program
SAVE	Dump a copy of the program to the currently assigned punch device
SGN	Function to return the sign of a variable
SIN	Sine function
SPC	Used in PRINT statement to print spaces
SQR	Function to return the square root of a number
STEP	Used in FOR statement for increment of loop control
STOP	Used to terminate program execution
STR$	Function to convert a value to a character string
SWITCH	Used to change the console assignment
TAB	Used in a PRINT statement to tab to a position
TAN	Tangent function
THEN	What to do if relational IF is true
TO	Used in FOR statement to specify limit
TRACE	Used to turn On/Off line number TRACE
USR	May be patched to user provided routine
VAL	Function to return the numeric value of a string expression
WAIT	Used to loop on a status port
WIDTH	Set the width of the console
?	Same as PRINT
↑	Exponentiation operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
+	Addition operator
<	Less than operator
>	Greater than operator
=	Equals operator
CONTROL U	Delete input line
RUBOUT	Delete previous character
CONTROL C	Abort execution of program
CONTROL X	Return to Monitor
CONTROL O	Suppress console output
,	Move to next TAB position or delimiter
;	Don't move
:	Used for multiple statements per line

GROUP 1 GENERAL PURPOSE UTILITY COMMANDS

CLEAR

Deletes all variables in storage. The command CLEAR followed by an argument, such as, "CLEAR 400" will set the string space to that value.

The CLEAR command may be placed in a program, for example:

15 CLEAR 250

to set the string space to the exact amount needed by that program. If the argument is omitted, the string space is not changed.

CONTINUE

If your program is stopped by typing a control C or by executing a stop statement, then you may resume execution of your program by typing CONTinue. Between the stopping and restarting of the program you may display the value of the variables, (see PRINT and LVAR) or change the value of the variables, (see LET). However, you may not modify the program, or continue after an error.

DELETE

Deletes a range of line numbers. The DELETE command is followed by two line numbers seperated by a dash "-". For example, DELETE 115-135 would delete from your program the line numbers 115 up to and including 135. DELETE 25 would delete only line 25.

LOAD

Loads a program from the reader device. The LOAD command is followed by a one character program name. A NEW operation is performed (see NEW), then the reader device is searched for a program under that name, if found, the program is then loaded. Example : LOAD P

A "bell" will sound on the console when the file starts to load. Additionally, a saved file may be verified by reloading the file using the following format:

LOAD?P

If an error occurs a message will be generated, otherwise, you will return to the command mode.

NEW

This command deletes all program and any stored variables. The slate is wiped clean, so to speak.

RENUMBER

The RENUMBER command causes the lines of a program to be renumbered and all the internal line number references such as 123 GOTO 547 to be properly adjusted. This command has two parameters separated by a comma. The first parameter specifies the starting point for the line numbers, the second parameter specifies the increment between numbers. If either parameter is omitted it defaults to 10. .

```
example        RENUMBER
               RENUMBER 10
               RENUMBER ,10
               RENUMBER 10,10
All start at 10 and increment by 10.
```

RUN

RUN clears all variables and starts the execution of the program starting with the first program statement. RUN followed by a line number will clear all variables and start execution at that line number.

Example RUN 105

SAVE

This command causes a copy of the program to be output to the punch device using Basic's internal compressed format. The SAVE command has one parameter, a one character program name which is used in reloading the program with the LOAD command.

Example SAVE P

saves the program under the name "P"

GROUP 2 THE EDIT COMMAND

EDIT

The EDIT command followed by a line number invokes the line editor to process that line. The editor moves a copy of the line to be edited into its edit buffer. At the end of the editing process, the user has the option of replacing the line in the program with the contents of the edit buffer, or throwing away the changes (say that you decide that you really don't want to make those changes).

Example: EDIT 55

The editor will then print the line number and then wait for single letter commands. ALL commands are NOT ECHOED. Illegal commands will echo as a bell. Some commands may be preceeded

by a numerical instruction to repeat itself. These are shown by a lower case "n" and may range from 1 to 255.

EDIT COMMAND	-FUNCTION
A	Reload the Edit Buffer from the program line. This is used after making a mistake.
nD	DELETE "n" characters
E	END EDIT - don't print line and replace program line with the contents of the edit buffer.
nFx	FIND the "n"th character X in the edit buffer and stop with the pointer just before the character.
H	DELETE everything to the right of the pointer and go to the insert mode.
I	INSERT all following characters from the keyboard, STOP INSERTING on a carriage return or Escape.
nKx	KILL or DELETE characters from where the pointer is now to the "n"th character X, but don't delete that character.
L	Print the line and return to the beginning of the line.
Q	QUIT. Leave the edit mode - without replacing the program line.
nR	REPLACE the "n" following characters with characters from the keyboard.
X	MOVE the pointer to the end of the line, and go to the insert mode.
SPACE	MOVE the pointer to the right.
RUBOUT	MOVE the pointer to the left.
CARRIAGE RETURN	END EDITING, print the line, replace the program line. This may also be used to terminate the insert mode.
ESCAPE	END insert mode or cancel pending commands.

console print head or cursor. This line:

```
55 PRINT A,B;"DOLLARS"
```

will be used in all the examples.

```
USER TYPES:     MACHINE RESPONDS:

EDIT 55         55!
```

To list out the line (command not echoed)

```
L               55 PRINT A,B;"DOLLARS"
                55!
```

To move the pointer forward:

```
(space)     55 P!
(space)     55 PR!
10(space)   55 PRINT A,B;"D!
```

etc.

To move the pointer backward, the system echos characters that are passed by the pointer as it is going backwards.

```
(rubout)    55 PRINT A,B;"DD!
2(rubout)   55 PRINT A,B;"DD";!
L           55 PRINT A,B;"DD";;"DOLLARS"
            55 !
```

Although the example of what happens when using the rubout command may be confusing when shown as above, in actual use, the response of the machine when you type the rubout command is quite easy to get used to.

To Delete a character:

```
D           55 \P\!
L           55 \P\RINT A,B;"DOLLARS"
            55 !
L           55 RINT A,B;"DOLLARS"
            55 !
(space)     55 R!
2D          55 R\IN\T A,B;"DOLLARS"
L           55 R\IN\T A,B;"DOLLARS"
            55 !
L           55 RT A,B;"DOLLARS"
            55 !
```

To recover from a mistake:

At this point we deceide we didn't really want to change the word "PRINT", so we can reload the edit buffer with the "A" command.

```
A           55 !
L           55 PRINT A,B;"DOLLARS"
            55 !
```

The Insert command inserts characters into the line at the present position of the pointer. The Insert command is terminated by either an escape character or a carriage return.

```
L           55 PRINT A,B;"DOLLARS"
            55 !
18(space)   55 PRINT A,B;"DOLLARS!
IXX(escape) 55 PRINT A,B;"DOLLARSXX!
L           55 PRINT A,B;"DOLLARSXX"
            55 !
L           55 PRINT A,B;"DOLLARSXX"
            55 !
```

The "X" command moves the pointer to the end of the line and goes into the insert mode.

```
XYY(escape) 55 PRINT A,B;"DOLLARSXX"YY!
L           55 PRINT A,B;"DOLLARSXX"YY
            55 !
L           55 PRINT A,B;"DOLLARSXX"YY
            55 !
```

The "H" command deletes all the line to the right of the pointer and goes to the insert mode.

```
11(space)    55 PRINT A,B;"!
HCENTS"(esc)55 PRINT A,B;"CENTS"!
L            55 !
L            55 PRINT A,B;"CENTS"
             55 !
```

GROUP 3 COMMANDS INVOLVING THE CONSOLE

LIST Causes the program to be typed on the console device. List may have one or two parameters separated by a dash.

EXAMPLE: LIST 20-30

will print lines 20 thru 30 on the console. LIST 20 would only print line 20.

LVAR Causes the variable storage area to be typed on the console. LVAR has no parameters but may

be placed in a program to get a snapshot of the variables during execution.

NULL

Sets the number of nulls to be output to the console after a carriage return/line feed sequence. This may be used to give additional time after a carriage return for terminals which require additional time to return the print head.

The NULL command may be followed by 1 or 2 parameters separated by a comma. The first parameter specifies the number of nulls to send after the carriage return line feed sequence and the second, as a decimal number, specifies what character is to be used. (initializes to zero or ASCII null).

EXAMPLE: NULL 3,255

to send 3 rubout characters after a carriage return line feed.

POS

This function is used to return the present position of the print head or cursor of the console device. The first position is considered to be zero (0). The function requires a dummy argument.

EXAMPLE: 40 A=POS(B)

would take the present position of the print head as a number from 0 to 131 and place it in variable A.

PRINT

is used to direct printed output to the console device. The PRINT command is followed by a list of variables, constants, or literals to be printed, separated by commas or semicolons.

EXAMPLE: 40 PRINT 123,A,"IS THE ANSWER"

If the separators are commas, then the items are printed in columns spaced every 14 positions across the console.

If the separators are semicolons, the items are printed with 2 spaces in between. Additionally, if the last item in the print list is followed by a semicolon, then unless executing the command would overprint the last print position on the console, BASIC will not carriage return but would stack successive PRINT commands across the console on the same line.

```
EXAMPLE:   10 PRINT A,B;
           20 PRINT C
```

would print A,B and C across the same line on the console. There would be 14 spaces between the beginnings of A and B and 2 spaces between the end of B and the beginning of C.

SPC

This is a function-like command that is used to print a number of spaces on the console. It is only used in a PRINT or LPRINT statement and is called function-like because it looks like a function but CANNOT be used in a LET statement.

```
EXAMPLE:   35 PRINT A;SPC(5);B
```

may be used to place an additional 5 spaces between A and B over and above the two that would normally be printed due to the semi-colon.

SWITCH

is used to change the console assignment. SWITCH used with no variable will always switch between the teleprinter and the user console. SWITCH with an argument of 0-3 (zero to three) will assign the console to that value.I.E.

0=TTY
1=CRT
2=BATCH MODE
3=USER DEFINED

A value greater than 3 will generate an error message. These are further discussed in the ZAPPLE Monitor Manual.

TAB

Is a function-like command that is used only with a PRINT or LPRINT statement and is used to Tab directly to a particular position. If the printhead or cursor is on or after the specified TAB position then BASIC will ignore the TAB command.

```
EXAMPLE:   25 PRINT A;TAB(25);B
```

TRACE

is a one parameter command that turns on or

brackets, e.g. <25>. The parameter may be an expression and if that expression is evaluated to be non-zero, then the TRACE is turned on. If the expression is evaluated to be zero, then the TRACE is turned off. (NOTE: The TRACE and LTRACE are completely independent functions and may be separately manipulated at will.)

EXAMPLE: 25 TRACE A-B

TRACE 1 (TURN ON)

If A-B is equal to zero then the trace is turned off, if equal to non-zero then TRACE is turned on.

WIDTH

Basic keeps track of the number of characters and spaces printed on the console and will generate an automatic carriage return line feed to prevent over-printing at the end of a line. the WIDTH command may be used to change the sign-on default of 72 spaces.

EXAMPLE: WIDTH 80

will cause an automatic carriage return line feed sequence after 80 characters. The minimum value is 15, and the maximum value is 255.

GROUP 4 COMMANDS INVOLVING THE LINE PRINTER

Most of the commands of Group 3, which effect the console, have a counterpart in Group 4, which effects the line printer. The general rule is to add the letter L in front - thus PRINT becomes LPRINT, and LVAR becomes LLVAR, etc. This section simply lists the commands and directs your attention back to Group 3 for information on how the commands function otherwise.

LLIST	see	LIST
LLVAR	see	LVAR
LNULL	see	NULL
LPRINT	see	PRINT
LTRACE	see	TRACE
LWIDTH	see	WIDTH
LPOS	see	POS
SPC	use	in LPRINT statement
TAB	use	in LPRINT statement

GROUP 5 COMMANDS AND FUNCTIONS THAT INVOLVE THE MOVEMENT OF DATA FROM ONE PLACE TO ANOTHER.

LET(=) This is the assignment command. It causes the evaluation of an expression on the right side of the equals sign (=) and the assignment of the resultant value to the variable on the left side of the equals sign.

EXAMPLE: 10 LET A=B+2

would cause BASIC to get variable B, add 2 to it, and place the result in A. In TDL Basic, the command LET is optional. for example, the previous statement could also be written as:

```
10  A=B+2
```

DIM Reserves storage for matrices. The storage area is first assumed to be zero. Matrices may have from one to 255 dimensions, but is limited by the available remaining workspace (memory).

```
257 DIM A(72),B(4)
258 DIM C(72,66)
259 DIM D(J)
```

Matrices may also be dimensioned during the execution of the program after the storage space is calculated, however, remember that the DIM command zeros the storage area, and a previously dimensioned array may not be re-dimensioned.

DATA Specifies constants that may be retrieved by the READ statement.

EXAMPLE: 10 DATA 5,4,3,2,1.5

READ Retrieves the constants that are specified in the DATA statement.

EXAMPLE: 20 READ A

The first time line 20 is executed the value 5 from the previous example will be placed into variable A. If at some later time statement 20 is executed again, or another READ statement is executed, then the value 4 would be retrieved etc. DATA statements are considered to be chained together and appear to be one big data statement. If at any time all the data has been read and another READ statement is executed then the program is terminated and

RESTORE

This command restores the internal pointer back to the beginning of the data so that it may be read again.

INPUT

Allows the operator to type data into one or more variables.

```
EXAMPLE:   35 INPUT A,B
```

would cause the printing of a question mark on the console as a prompt to the operator to input two numbers separated by a comma. If the operator doesn't type enough data then BASIC responds with 2 question marks.

```
EXAMPLE:   10 INPUT A,B,C
           RUN
           ? 5
           ?? 7,5
           READY
```

would input the value 5 to the variable "A" and when the operator typed carriage return, Basic wanted more data and so responded with 2 question marks.

The input statement may be written so that a descriptive prompt is printed to tell the user what to type.

```
EXAMPLE:   10 INPUT "TYPE A,B,C";A,B,C
           RUN
           TYPE A,B,C?  (ans)5,6,7
           READY
```

This causes the message placed between the quotes to be typed before the question mark. Note the semicolon must be placed after the last quote.

PRINT
LPRINT

PRINT and LPRINT are the converse of INPUT in that they print out data on the console and line printer. For an explanation of these commands see groups 3 and 4.

INP

Basic has the ability to directly read an input port. The INP function takes as its argument the number of the port to be read, and the result may be assigned to a variable or printed directly.

```
EXAMPLE:   10 A=INP(0)
           20 PRINT INP(0)
```

would in both cases input from port zero. In line 10 the value input from the port is

placed in variable "A" and in line 20 it is directly printed.

OUT

This command causes Basic to output data directly to any output port. The OUT command has two parameters separated by a comma. The first parameter is the port number and the second parameter is the data to be output.

```
EXAMPLE:   10 A=1
           20 B=7
           30 OUT A,B
           RUN
```

would cause a seven to be output to port ONE, and if your console data port is port one the bell would ring since a 7 is a BELL in ASCII code.

WAIT

If you write a program to INP or OUT directly to the console for a purpose such as reading a paper tape from the teleprinter tape reader, Basic itself will interface with inputing the data because Basic is looking at the console keyboard to see if a Control C is typed to abort execution or Control X is typed to return to the Monitor. The WAIT statement will place Basic in a loop, looking at a specified status port, until a specified condition occurs. Then and only then will the next statement be executed.

(NOTE: Be careful using this because it is possible to put Basic in a loop waiting for a condition that will never occur. Should this happen, your only recourse is to reset the machine, or examine the memory location 3 higher than the address the program was loaded at, and hit RUN again. Basic will then recover without destroying your program.)

```
EXAMPLE:   100 WAIT A,B,C
           110 D=INP(A+1)
```

Basic will then input port "A", exclusive or the value with "C", and then AND the result with B. If a non-zero result occurs, then the process is repeated until a zero result occurs. Basic, in this example, will input from the next higher port and place the data in "D". The fact that at Line 110 Basic looked at the console port to see if the data was a

that data is available on port 1. Then let A=0 for port Zero and One. Let B=4 to isolate Bit 2, and let C=255 so that a complement of the status occurs to follow the rule that data available is indicated by a zero result. If parameter C is ommitted, then Basic defaults to zero for the value.

PEEK

This function allows the direct retrieval of data anywhere in memory.

EXAMPLE: 50 B=PEEK(A)

causes the value of the byte at address "A" to be assigned to the variable "B". Address "A" may range from 0 to 65535 (decimal).

POKE

Has two parameters.

EXAMPLE: 57 POKE A,B

in which the first parameter specifies an address in which to insert the data specified by the second parameter. The address may range from 0 to 65535 and the Data may range from 0 to 255.

GROUP 6

TRANSFER OF CONTROL AND RELATIONAL TESTS

GOTO

This statement followed by a valid existing line number will cause Basic to transfer control directly to that statement.

EXAMPLE: 55 GOTO 100

will, if line 100 exists, cause execution of the program to resume at line 100.

GOSUB

acts in a manner similar to that of GOTO except that the location of the next statement is saved so that a RETURN can be performed to return control.

RETURN

This statement is used to "return" control back to the statement following the most previous GOSUB that control came from.

ON x GOTO
ON x GOSUB

The ON statement causes control to be transferred to the "x"th line number in the

EXAMPLE: 10 ON A GOTO 100,125,150

If A was equal to 1, control would be transferred to Line 100. If A=2, GOTO Line 125, etc. If A is equal to zero, or larger than the number of line numbers in the list, control will be given to the statement after the ON x GOTO. The value of x may range from 0 to 255.

FOR,TO,STEP,NEXT

These key words are used to set-up and control loops.

```
EXAMPLE:    10 FOR A=B TO C STEP D
            20 PRINT AA
            30 NEXT A
```

If B=0, C=10 and D=2, the statement at line 20 will be executed 6 times. The values of A that will be printed will be 0,2,4,6,8,10. A represents the name of the index or loop counter. The value of "B" is the starting value for the index, the value of "D" is the value to be added to the index. If D is omitted then the value defaults to 1. The "NEXT" keyword causes the value of "D" to be added to the index and then the index is tested against the value of C, the limit. If the index is less than or equal to the limit, Control will be transferred back to the statement after the "FOR" statement. The index may be omitted from the "NEXT" statement, and if omitted the "NEXT" statement effects the most recent "FOR". This may be of concern in the case of nested "FOR-NEXT" statements.

```
EXAMPLE:    10 DIM A(3,3)
            20 FOR B=1 to 3
            30 PRINT "PLEASE TYPE LINE";B
            40 FOR C= 1 TO 3
            50 INPUT A(B,C)
            60 NEXT C
            70 PRINT "THANK YOU.";
            80 NEXT B
```

IF THEN ELSE The "IF" keyword sets up a conditional test.

EXAMPLE: 25 IF A=75 THEN 30 ELSE 40

Upon execution of line 25 if A is equal to 75 then control is transferred to line 30. Else

EXAMPLE: 25 IF A=75 GOTO 30 ELSE 40

The THEN and ELSE clauses may contain imperative statements.

EXAMPLE: 30 IF A=75 THEN A=0 ELSE A=A+1

The ELSE clause may be omitted in which case control passes to the next statement.

EXAMPLE: 40 IF A=75 THEN A=0

Relational operators used in IF statements:

=	EQUAL
<>	NOT EQUAL
<	LESS THAN
>	GREATER THAN
<=	LESS THAN OR EQUAL
>=	GREATER THAN OR EQUAL

The logical operators may also be used:

NOT	Logical Negation
AND	Logical And
OR	Logical Or

EXAMPLE: 20 IF(A=0) OR NOT(B=4) THEN C=5

GROUP 7 TRIG FUNCTIONS

ATN

Function to return the ARCTANGENT of a value. The result is expressed in radians.

EXAMPLE: 10 B = ATN(.45)

returns the angle, expressed in radians, whose tangent is equal to .45.

COS

Function to return the cosine of an angle, expressed in radians.

EXAMPLE: 20 C=COS(A)

SIN

Function to return the sine of an angle, expressed in radians.

EXAMPLE: 30 D =SIN(A)

TAN

Function to return the TANGENT of an angle, expressed in radians.

EXAMPLE: 40 T=TAN(A+B)

GROUP 8 MISCELLANEOUS FUNCTIONS

ABS

Function to return the absolute value of a value.

EXAMPLE: 10 A=ABS(B+C)

If the result of the expression is positive then ABS returns that value. If the result is less than zero, then ABS returns the positive equivalent.

DEF,FN

These commands allow the user to define his own functions. A function defined in this way must have a name that begins with the letters "FN" followed by a valid variable name. For example "FNA", or "FNZ9". The function name is then followed by one parameter enclosed in parentheses. This parameter is a dummy argument and is included in the expression to the right of the equals sign.

EXAMPLE: 159 DEF FNQ(X)=X*A

In this case A is a variable within the program and X is an argument that may be replaced with a constant or another variable when the function is used.

```
EXAMPLE:   15 DEF FNQ(x)=X*A
             .
             .
           121 A=3
           122 B=4
           123 C=FNQ(B)+5
```

Thus Basic would take the argument B and multiply it by the value of A, add 5 to the product and place the result in variable "C".

EXP

This function returns the base of the natural log system "e" or 2.71828 raised to a power.

EXAMPLE: 20 B=EXP(A)

If A is equal to 1 the result is "e" or 2.71828.

FRE

This function, when used with a dummy

EXAMPLES: 30 A=FRE(X)
40 B=FRE(X$)

INT — Returns the integer portion of a number. This is essentially a "round-down" operation. For negative arguments the result would be the next more negative integer.

EXAMPLE: 50 C=INT(D)

If "D" has a value of 5.25 then "C" will have a result of 5.0. If "D" has a value of -3.4, then "C" will be set to -4.0.

LOG — Returns the natural logarithm of the expression used as an argument.

EXAMPLE: 60 E=LOG(F+G) will return the log to the base "e" of the expression "F+G".

SGN — Will return the value +1 if the argument is greater than zero, zero if the argument is zero, and -1 if the argument is less than zero.

EXAMPLE: 70 H=SGN(I)

SQR — Returns the square root of the argument. The argument may not be less than zero.

EXAMPLE: 80 J=SQR(K)

RND — Returns a psuedo-random number in the range between 0 and 1. The RND function uses a dummy argument to perform the following functions. An argument less than zero is used to initialize the psuedo-random number sequence. An argument of zero will return the previous random number. An argument of more than zero sill return the next psuedo-random number in the sequence.

EXAMPLE: 90 R=RND(1)

RANDOMIZE — Although this is not a function, it is discussed here because of its relation to RND. The RANDOMIZE command may be used to generate a truly random starting point for the psuedo-random number sequence. (RND)

GROUP 9 STRING RELATED FUNCTIONS

ASC

Returns the decimal number that represents the first ASCII character of the string expression used as an argument.

EXAMPLE: 10 A=ASC(A$)

The decmial value 65 represents an ASCII "A". If the character "A" was the left-most character of string "A$" then variable "A" would be set to 65.

CHR$

Returns the ASCII character represented by the decimal value of the argument.

EXAMPLE: 20 PRINT CHR$(7)

would ring the bell on the teleprinter connected to the console. A "7" is an ASCII bell.

LEFT$

Uses two arguments, the first is the string expression and the second is the number of characters to return from the left end of that string. The second parameter may range from 1 to 255.

EXAMPLE: 30 B$ =LEFT$(A$,5)

would set B$ equal to the first 5 characters of string "A$".

LEN

Returns the length of a string expression in bytes.

EXAMPLE: 40 X=LEN(S$)

would set "X" to the number of bytes contained in the string "S$".

MID$

May have 3 parameters:
1) The string expression.
2) The position to start extracting characters.
3) The number of characters to extract. This value defaults to 1 if omitted.

EXAMPLE: 50 A$=MID(B$,5,6)

RIGHT$ — See LEFT$ except works on the righthand end of the string.

STR$ — Returns a string whose characters represent the numeric value of the argument.

EXAMPLE: 70 A$=STR$(2.2)

would return the characters "2.2" preceded by a space as the result.

VAL — Opposite of STR$. Returns the numeric value represented by a character string.

EXAMPLE: 80 A=VAL("4.5")

would return the numeric result 4.5.

GROUP 10 MISCELLANEOUS COMMANDS

END — Stops the execution of the program. The END command may be placed anywhere in the program.

EXAMPLE: 65520 END

REM — Denotes that this line is a remark and is not processed.

EXAMPLE: 10 REM This is a remark

STOP — Similar to END except that the message BREAK @ LINE (x) is printed, where "x" is the line number of the STOP command.

USR — The USR command allows Basic to exit to a user provided assembly language routine, evaluate a value, and return with the result.

In use, Basic must be told where to go for the assembly language routine. When the USR function is referenced, Basic will call the USR transfer vector. Normally, this vector points to an error routine within Basic. In order to link to an assembly language routine, you must patch the address (start of Basic +6H) with a jump to your assembly language routine.

In your assembly language routine, in order to get the passed value, call 0027'H (start of Basic + 27H). Basic will return with the passed value in registers D&E.

To return the result back to Basic, place the low byte of information in register B, and the high byte in register A, and call 002A'H (start of Basic plus 2AH).

To give control back to Basic, execute a RET instruction.

Having done the above, the Basic program and your routine can be made to interact at will by use of the USR function.

```
EXAMPLE:   10 X=USR(Y)
           20 PRINT X
```

will pass the value "Y" to your assembly language routine. The returned value would be assigned to "X", and then printed on the console.

OPERATORS LISTED IN THE ORDER OF EVALUATION

A) Any expression enclosed in parentheses is evaluated from the innermost parenthesis first to the outermost parenthesis last.

B) ↑ Exponentiation

C) - Negation. I.E. A minus sign placed so as to NOT indicate subtraction. For example: A=-B or C=-(2*D)

D) * Symbol for multiplication. Used in the form 2*2(cr) yields an answer by Basic of "4".

E) / Division. Used in same manner as multiplication symbol.

F) RELATIONAL OPERATORS:

=	Equals
<>	NOT EQUAL
<	LESS THAN
>	GREATER THAN
<=	LESS THAN OR EQUAL TO
>=	GREATER THAN OR EQUAL TO

G) NOT Logical negation - such as A=NOT B

H) AND Logical AND