MICROPERIPHERALS
icom
A Division of Pertec Computer Corporation
6741 VARIEL AVENUE - CANOGA PARK - CALIFORNIA 91303 - PHONE: (213) 348-1391
TELEX/TWX 910-494-2788

OPERATOR'S MANUAL

BASIC-M

iCOM BASIC-M

BASIC-M has been designed for use with the iCOM Floppy Disk System and FDOS-II, or FDOS-M. BASIC-M is a useful high level language due to its ease of implementation and its English language-like features, enabling the programmer to arrive at quick solutions to programming problems.

The text in this manual is provided as a guide and reference only; it is not intended to instruct in BASIC programming techniques.

The first section of this manual describes the features of BASIC-M, its operation and preparation for programming.

Section II lists and describes BASIC-M commands. These commands are simple to follow and remember because of their conversational nature.

Section III describes the four types of statements in BASIC-M and how they are used.

Subprograms for BASIC-M are defined in Section IV.

Section V contains error messages, followed by appendix for statement and command summary.

Throughout this manual, when discussing statements, commands and use of BASIC-M, parenthetical statements in lower case letters that appear in examples have the following meanings:

(statement n)	Statement Number
(var)	Variable Name
(exp)	Mathematical Expression
(rel exp)	Relational Expression
"textstring"	Concatenation of literal alpha-numeric characters surrounded by quotation marks.
(cr)	Carriage Return

iCOM BASIC-M

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

SECTION I

OPERATION OF BASIC-M

A. BASIC-M FEATURES

1. Multiple statements may be entered on a line.
2. All mathematical operations are performed in BCD arithmetic.
3. Data output may be formatted.
4. Programs may be saved and recovered from diskette.
5. *FUNCTION* subprograms may be implemented.
6. Program statements may be executed in the direct mode for immediate calculations and debugging.
7. Programs generally require no more than 8K bytes of memory.
8. PASS and CALL functions aid in programming in conjunction with other 8080 machine language programs.

B. OPERATION

Using FDOS-M or FDOS-II, BASIC-M may be used in one of four ways:

1. To simply input a program, or to use BASIC-M in the conversational mode without requiring that it be saved,

   Type: RUNGO, BASIC(cr)

2. To bring in and run a program without requiring that it be saved, or written out to another file,

   Type: RUNGO, BASIC, "ifile"(cr)

3. To write out to file, code which has been entered as a program,

   Type: RUNGO,BASIC,,"ofile"(cr)

4. To bring up a BASIC program which is to be written back out to another file,

   Type: RUNGO,BASIC,"ifile","ofile"(cr)

When one of the above commands has been entered, BASIC-M will be loaded and executed. The console will then display the work READY.

To load a program file, type: DLOAD(cr)

When loading is completed, READY will again be displayed on the console.

To see a program listing, type: LIST(cr)

To execute a program, type: RUN(cr)

To save a BASIC program after input, type: DSAVE
The program will then be written to the output file.

C. PREPARING TO PROGRAM

When BASIC-M has been loaded and the word READY appears on the console, the system is ready to accept commands or statements.

1. Data Format
   The range of numbers which may be represented in BASIC-M is .E-127 to .999999E+127. BASIC-M has 6-digit precision, and numbers will automatically be rounded to fit. Numbers may be entered and displayed in the following formats:

FORMAT	EXAMPLE
Integer	153
Decimal	34.52
Exponential	136E-2

2. Variable Names
   A variable may be specified by one or two characters, but may not exceed two characters. The first character in the variable must be an alphabetic letter. The second may be either alphabetic or numeric.

   EXAMPLE: A, B5, X, D1

3. REM Statement

This is a remark or comment statement which is not executed as part of the program, but is used to clarify programming statements. A REM statement may not be terminated by a semicolon, but must be terminated each time with a carriage return. In the example below, the LET statement for line 255 will not be recognized or executed.

```
EXAMPLE:    255 REM ACCOUNT SUBROUTINE(cr)

Illegal
Example:    255 REM ACCOUNT SUBROUTINE; LET Z1=1Ø
```

The LET statement must be assigned its own line number in order to execute.

4. Automatic Sequence

Regardless of how statements are written, they will automatically be numerically sequenced.

```
PROGRAM     12Ø REM "ACCOUNTING PROGRAM"
EXAMPLE:    13Ø OPENR "FILE1"
            14Ø DSKIN X
            15Ø Y=X+3/2
            16Ø PRINT Y
            17Ø STOP
```

If a statement is needed between two existing statements, type a statement number which would fall numerically between the two existing statement numbers, followed by the statement.

```
EXAMPLE:    155 IF Y>1ØØ THEN GOTO 17Ø
```

The new line is automatically sequenced between 150 and 160.

5. Line Replacement

To replace a statement, type the new statement using the same line number. The new statement automatically replaces the original "Y" statement in the example below.

```
EXAMPLE:    15Ø LET Q=SIN X(cr)
```

6. Line Termination

To terminate a line entered, hit carriage return (cr). Line feed is automatic.

7. Erase

A character may be erased by using SHIFT/RUB key. An entire line may be erased by using the "@" key prior to hitting carriage return.

8. Program Execution

To execute the program, enter RUN(cr).

SECTION II

BASIC-M COMMANDS

Direct communication with BASIC-M is accomplished with the use of commands and *DIRECT EXECUTION* statements. When either mode of communication is employed, it is immediately executed. Neither commands, nor direct execution statements, are preceeded by a statement line number. Upon execution, more information may be required of the user; if not, READY is displayed and BASIC-M is ready to receive input. Further discussion of the direct mode of execution appears in Part B of this section.

A. BASIC-M commands are:

```
CLEAR
LIST
RESTART
NULL
RUN
NEW
DLOAD
DSAVE
@
SHIFT/RUB
, (comma)
```

1. CLEAR

All variables are set to zero. The READ pointer is reset and the program is initialized to be run from the beginning. CLEAR may also be used as a statement in subprograms, and statements that exit FOR-TO loops. (See *CONTROL STATEMENTS*, Section III, D)

2. LIST

This command may be used in one of two ways. All statements will be displayed on the console with lines listed numerically beginning from the start of the current program, by typing: LIST(cr)

A specified number of lines may be listed by using the following format:

```
LIST (n),(n)(cr)
```

The first number in the LIST command specifies where the list will begin, and the second number will be the last line listed.

If no line is specified, the entire program will be listed.

3. RESTART

Clears all error conditions without altering the current program. Upon execution, control is returned to the command mode.

4. NULL

Null character codes are transmitted to the console device after a carriage return.

5. RUN

Execution of the program begins at the beginning, for the entire program. The data pointer is then reset and performs a CLEAR.

6. NEW

Removes all program data from memory and resets all pointers, variables and working storage to begin again. In other words, the entire current program is erased.

7. DLOAD

Loads BASIC-M program from disk specified by "ifile" in the RUNGO statement.

8. DSAVE

Stores, or writes, the program to disk file specified by "ofile" in the RUNGO statement.

9. @

Deletes an entire line. To erase a line the "@" must be used prior to hitting carriage return.

10. SHIFT/RUBOUT

Deletes a single character. The number of times these two keys are struck simultaneously will determine the number of corresponding characters to be deleted, following which the correct characters may be typed.

11. , (Comma)

More than one argument in a command is established when separated by commas.

```
EXAMPLE:  PRINT Y, "NEXT", X
```

B. DIRECT EXECUTION COMMANDS

Statements may be used as commands which allow the user to calculate mathematical problems immediately. The computer executes immediately upon input while in the command mode. These commands are also useful in program development and as debugging tools.

The direct execution statements listed below, which are also used as program statements, are *not* preceeded by statement numbers.

STATEMENT & FORMAT

```
DIM (var) (exp)

GOTO (n)

IF (rel exp) THEN (n)

LET (var=exp)

PRINT
```

EXAMPLE:

```
DIM AR(1Ø)
GOTO 250
IF A>B THEN 1ØØØ
LET PI=3.14
PRINT "AREA IS",PI*R*R
```

SECTION III

BASIC-M STATEMENTS

All program statements must begin with a statement number, followed by the statement body, and terminated by a carriage return. Termination using a semicolon (;) is employed when making multiple statements. The semicolon may not, however, be used to terminate a line.

The four types of statements in BASIC-M are:

Declarations

Assignments

Input/Output

Control

These four statements are described in detail further on in this section.

A. USE OF STATEMENTS

1. Each statement is assigned a number between 1 and 65000.

2. BASIC-M sequentially orders the program according to statement number.

3. A statement number may be used only once per program.

4. Each statement may contain up to, but not in excess of, 72 characters and spaces.

5. Use of blank spaces is optional in BASIC-M. Spaces are not executed unless found within a character string and surrounded by quotation marks.

```
EXAMPLE:  LET E = M*C*C
would
execute
the same
as:       LETE=M*C*C
```

6. Multiple statements, separated by semicolons, may be used on a line having only one statement number.

```
EXAMPLE:  150 LET A=1; B=3.2; C=2.5E+1Ø(cr)
```

7. Commas (,) may be used to separate multiple arguments within a command or statement.

```
          EXAMPLE:  1ØØ PRINT X, "NEXT", Z
```

B. DECLARATION STATEMENTS

The declaration statements are:

DATA and READ

RESTORE

DIM

1. DATA and READ

These two statements are used together to assign a value to a variable. As each DATA statement is encountered, the value in the argument field is assigned sequentially to the next available position of the data buffer. Regardless of where DATA statements occur in a program, they are combined sequentially into a single data list.

The READ statements causes the values in the data buffer to be accessed sequentially and assigned to the variable designated.

```
          FORMAT:   DATA (n), (n), (n)(cr)
                    READ (var),(var),(var)
          EXAMPLE:
                    DATA 1Ø,-6,25
                    READ A,B,C
```

2. RESTORE

Causes the data buffer pointer to be reset to the beginning of the buffer. The buffer pointer has been advanced as a result of the READ statement having been executed.

```
          EXAMPLE:  11Ø DATA 50, 75, 125
                    12Ø READ A; REM (A NOW = 50)
                    13Ø RESTORE
                    14Ø READ B,C ;REM (B NOW = 50)
                    15Ø REM (C NOW = 75)
```

3. DIM

Memory space is allocated for an array, using single dimension arrays only. Maximum array size is 10,000, and all elements are set to zero by the DIM statement. An array that is not specifically designated by a DIM statement is assumed to be an array of 10 elements from the first reference in the program. An array may be dimensioned only once either dynamically or statically, within a program.

DIM may also be used as a direct execution command

FORMAT: DIM (var)(exp)

C. ASSIGNMENT STATEMENTS

The LET statement, with the use of mathematical operators, assign values to variables.

1. LET

Use of the word LET is optional, but the statement format is:

```
LET (var)=(exp)

EXAMPLE:  1ØØ LET B=827
May also  11Ø LET B5=87E2
be shown: 12Ø R=(X*X)/2*b
```

The equal sign "=" is the replacement operator which instructs the program to replace the value of the variable with the value of the expression. The expression may consist of a single numerical value, or an expression of several numerical values, variables, mathematical operators and functions.

The LET statement may also be used as a direct execution command.

2. MATHEMATICAL OPERATORS

Mathematical operators, used to form expressions, are evaluated by the computer in specific order, and from left to right if the same operator is shown more than once. Those operators and the order of evaluation are:

-(unary)	Negate
* and /	Multiply and divide
+ and -	Add and subtract

Negation requires only one operand. The sequence of evaluation may be controlled with the use of parentheses pairs, evaluating the inside pair first, then the outside pair.

EXAMPLE FOR STATEMENT EVALUATION:

```
LET R=A+B-C/2*3
```

Evaluated as:

```
Temp. 1 = C/2
Temp. 2 = Temp. 1*3
R = A+B - Temp. 2
```

EXAMPLE FOR USE OF PARENTHESES:

```
LET R = ((A+B)-C)/(2*3)
```

Evaluated as:

```
Temp. 1 = A+B
Temp. 2 = Temp. 1 - C
Temp. 3 = 2*3
R = Temp. 2/Temp. 3
```

D. CONTROL STATEMENTS

Control statements, which direct the sequential progression of program statement execution, transfer control to other sections of the program, terminate execution, or set up loops. Those statements are:

FOR
NEXT
GOTO
IF
STOP
END

1. FOR and NEXT

These two statements, used together, create program loops which cause execution of one or more statements to be repeated a specified number of times.

```
FORMAT:   FOR (var)=(expl) TO (exp2) STEP (exp3)
                    .
                 statement body
                    .
          NEXT (var)
```

A variable is set to the value of the first expression for which statements following the FOR(var) are executed. When the NEXT statement is encountered, the specified variable is added to the value specified by the STEP expression, and execution resumes at the statement following the FOR-TO. If adding the STEP value creates a sum greater than the TO expression (exp2), the next instruction executed will be that following the NEXT statement. If no STEP is specified, a value of one is assumed. If the TO value is less than the initial value, the FOR-NEXT loop will still be executed, once.

```
          EXAMPLE:  1ØØ FOR J = 1 TO 9
                    12Ø PRINT J, 1/J, SQR(J)
                    13Ø NEXT J
                    14Ø END
```

Expressions are used for the first, final and STEP values in the FOR statement and are evaluated the first time the loop is entered only. If a variable in the NEXT statement is not named, the program will automatically add the STEP value to the variable in the last FOR statement.

```
          EXAMPLE:  1ØØ FOR J=1 TO 9
                              .
                              .
                     (statement body)
                              .
                              .
                    11Ø FOR X=25 TO 1ØØ
                              .
                     (statement body)
                              .
                    13Ø NEXT
                    135 NEXT
                    14Ø (statement)
```

The NEXT at 130 will STEP the FOR loop beginning at statement 110. The NEXT at statement 135 will step the FOR loop beginning at 100.

The same variable may not be used in two loops which are nested. Nesting FOR-NEXT may be five deep.

When the statement following the last NEXT has been executed, the variable becomes equal to the value which caused the loop to terminate, and not the TO value itself. In the example above, J would be equal to 10 when that loop terminated.

2. GOTO

This is an unconditional jump statement which instructs the program to go to the specified line number and resume execution at that point.

```
          FORMAT:   GOTO (statement n)
```

The GOTO statement is also used as a direct execution command.

3. IF and THEN

Since the GOTO statement produces an unconditional jump, or a non-stop loop, the IF statement introduces a condition, which when satisfied requires the program to proceed to the next statement.

FORMAT: IF (rel exp) THEN (statement n)

EXAMPLE: 11Ø IF A>B+3 THEN 16Ø

The IF statement controls the sequence of program statements to be executed, depending on specific conditions. If the expression A>B+3 is true, then control is given to statement line 160. If the relational expression is false, program execution proceeds to the next statement.

A statement may be given after the THEN statement.

FORMAT: IF (rel exp) THEN (BASIC statement)

If the relational expression is true, the BASIC statement will be executed and the program will proceed to the next statement. In this case, the IF statement may also be used as a direct execution command.

EXAMPLE: IF A>B+3 THEN PRINT A,B

3a. RELATIONAL OPERATORS

When evaluating relational expressions, arithmetic operations are evaluated in the specific order described in Item 2 of this section, followed by the relational operators.

CHARACTER	MEANING
=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
=>	Greater Than or Equal

Relational operators have a value of -1 if they are true, and zero if they are false.

```
EXAMPLE:  10+2*3<25*2       False statement

          (12>10) = -1      True Statement

          (A<>A) = Ø        False Statement
```

4. STOP

This statement causes the program to stop executing and returns program control to the command mode. The program will display the statement number where the program halted. The program is resumed by using the GOTO statement. The STOP message is displayed on the consoled as follows:

```
"STOP IN LINE(n)"
```

5. END

The END statement is assigned the last number in the program and causes the program to stop executing. Control is given to the command mode.

```
FORMAT:  5ØØ END
```

E. INPUT/OUTPUT STATEMENTS

When using input or output statements, note that only one file of each type of I/O at a time may be used.

```
EXAMPLE:    11Ø OPENR "file1"      ;REM (Open file for read)
            12Ø OPENW "file2"      ;REM (open file for write)
            13Ø DSKIN P            ;REM (read a value)
            14Ø DSKOUT P           ;REM (write to file2)
            15Ø CLOSE              ;REM (close file 2)
            16Ø STOP               ;REM (done)
```

INPUT/OUTPUT STATEMENTS

DSKIN
DSKOUT
OPENR
OPENW
PRINT
CLOSE

1. DSKIN

This statement causes data to be brought in from a disk file to the console.

2. DSKOUT

Data is output to the disk file.

3. OPENR

Used prior to a disk READ of data to open a specific file.

```
        FORMAT:  OPENR "file-name"(cr)
```

4. OPENW

Causes a file to be open to write data.

```
        FORMAT:  OPENW "file-name"(cr)
```

5. PRINT

Directs program to print out to the user console device. Values of expressions, literal values, variables and text strings may be printed out, and may be combined in the print list by separating each form with a comma, ending in carriage return.

```
        FORMAT:  PRINT (var)
                 PRINT "string"
                 PRINT (exp)
                 PRINT %(Z)(E)(F)(N)

        EXAMPLE: 11Ø PRINT X,Y,5(cr)
                 12Ø PRINT (cr)      ;REM (creates blank line)
                 13Ø PRINT "VALUE=-B", X3, "YZ2=",A2(cr)
                 14Ø PRINT A,B(cr)
```

A PRINT statement containing no argument causes a line to be skipped, as shown for line 120 in the above example. Values printed on a single line should have intervening blank spaces for easy comprehension. If the next position to be printed is greater than, or equal to, position 56, a carriage return is given prior to printing the next value.

PRINT may also be used as a direct execution command.

5a. TAB

This function causes data to be printed in specified format. The TAB argument may be an expression, as follows:

```
        EXAMPLE: 1ØØ PRINT TAB(2),B,TAB(2*R),C
```

The result of this PRINT statement, when the program is run, will be two aligned columns of figures.

5b. FORMATTED PRINT

The format of the print output may be specified in one of four ways: free format, exponential format, trailing zeros, degree of accuracy in numbers of digits to right of decimal point.

If format is not specified, six digits of precision will automatically be printed, with the low order digit rounded, and trailing zeros suppressed. Automatic selection will be made between the decimal, integer and exponential formats, based on the size of the stored value.

5c. AUTOMATIC FORMAT OVERRIDE

Formatting may be overridden if format specification is included in the output list. Specification is designated by two percent signs (%%) between which are the code characters, as shown in the example below.

FORMAT CODE: F - Free format (automatic)

Z - Print trailing zeros

E - Print in exponential format

N - Print N (N=1-6)places to right of decimal point

Once a format is specified, it will repeat until new format specification is given. The program may be returned to normal default format by specifying %(exp)%.

```
EXAMPLE: 11Ø PRINT %6E%
         2ØØ PRINT %Z2%,A,B; PRINT %Z3%,3D,%%
```

5d. : (colon)

Use of the colon may be substituted for PRINT.

```
EXAMPLE: 1Ø PRINT X,Z,3

the same
as:      1Ø : X,Z,3
```

6. CLOSE

This statement must be written when all data has been completely written to a file. CLOSE ensures that the last buffer of data is truly written to the file.

SECTION IV

BASIC-M SUBPROGRAMS

A subprogram is a set of instructions that will be executed more than once in the BASIC program. Two types of subprograms used are *subroutines* and *functions*. Use of subroutines helps to simplify program writing and debugging, and limits the need for excessive use of GOTO statements.

A. GOSUB

Access to subroutines is accomplished by using the statement pair GOSUB-RETURN.

```
FORMAT:     GOSUB (statement n)
                        .
               statement body
                        .
                        .
            RETURN
```

When the subroutine has completed execution, control is returned to the statement following the subroutine, or upon execution of RETURN.

1. NESTING

Subroutines may call other subroutines, which may in turn call other subroutines. A subroutine may not however, call itself. Subroutines may be nested up to six deep. Avoid the use of variables in a subroutine which have been used elsewhere in the program.

```
EXAMPLE: 1ØØ X=15
         11Ø GOSUB 2ØØ
         12Ø PRINT X
         13Ø X=4.9
         135 GOSUB 2ØØ
         14Ø PRINT X
         15Ø STOP
         2ØØ X=RND(X)+ X*X
         21Ø RETURN
         22Ø END
```

B. FUNCTIONS

Program control is given to a function, or unit of the statement, when the GOSUB statement is executed. A numerical value is computed for the function name in the expression, and upon execution, passes control back to the GOSUB statement.

FUNCTION	DEFINITION
ABS (exp)	Absolute value of expression
INT (exp)	Largest integer less than, or equal to argument
RND (exp)	Generates pseudo-random numbers ranging from 0.0 and 1.0. The argument does not alter the function, but is necessary for syntax. Execution is reset with the use of the CLEAR command.
SGN (exp)	If an argument is greater than, or equal to zero, it is given a value of +1. If the argument is negative, a value of -1 is given.
SQR (exp)	Square root of the argument
SIN (exp)	Sine of the argument when given in radians
COS (arg)	Cosine of the argument when given in radians
TAB (exp)	Positions output characters in PRINT statement
PASS (exp) and CALL (exp)	Used together to link to assembly language program. May be used as command for direct execution.

1. PASS and CALL

A PASS argument is evaluated as a sixteen bit integer and is temporarily stored in the monitor.

```
EXAMPLE: B=PASS(X2)
```

If linkage to an 8080 assembly language program segment is made via the CALL function, previously stored 16 bits will be passed to the assembly language code in the D,E register pair.

A CALL function is executed by coding it into a BASIC statement and the CALL argument will be evaluated as a 16 bit address. BASIC automatically transfers control to the specified address.

```
EXAMPLE: Y6=CALL(5.2*A4)
```

The machine language code will load register pair H,L with any specified information, which is passed back into the BASIC program as the value of CALL.

```
        EXAMPLE: 11Ø REM "LINK TO ASSY LANG PROGRAM"
                 12Ø LET X=1Ø; Y5=4280
                 13Ø L=PASS(X/6)
                 14Ø LET K=CALL(Y5)
                 15Ø PRINT K
                 16Ø END
```

In the example above, L is assigned the value of the PASS argument, linkage is made to the assembly language program at address 4280, and L is set to the data returned in the H,L pair.

SECTION V

ERROR MESSAGES

AE Floating point arithmetic error. Occurs when attempt is made to divide by zero, or when calculation results in a number too large for representation in BASIC number format. Underflow results in zero and no error is indicated.

CE Command error. Attempt to give BASIC a command that cannot be processed in direct execution mode.

DE Dimension error. Attempt to DIM more than once in a program.

FE File reference error.

IA Illegal argument

IE Input error. Response to INPUT or DISKIN statement error when a number has been given incorrectly.

IS Illegal syntax

LE Limit error. Allowable limited is exceeded for array index, TAB value or other integer.

LO Line Overflow. 72 character limit has been exceeded.

NE Nesting error. Stack exceeds 6 deep in FOR-NEXT, FOR lacks corresponding NEXT, or GOSUB-RETURN exceeds nest limit of 5 deep.

NS Negative square root function argument.

OV Overflow of storage. Insufficient room in memory for text, symbol table, array space, or program data.

RE Read error. Data buffer is full. READ statements have exceeded number of DATA values given.

UL Undefined line. Line number reference error in GOTO, GOSUB, or IF statement.

APPENDIX A

INDEX

SUMMARY OF COMMANDS AND STATEMENTS

MNEMONIC	DEFINITION	PAGE
IA	Error message, illegal argument	19
IE	Error message; input error	19
IF	Control statement; used with THEN to control execution sequence	6, 12
IS	Error message; illegal syntax	19
INT	Function; largest integer less than or equal to argument	17
LE	Error message; Limit error	19
LET	Assignment statement; assigns value to variable	6, 8
LIST	Command; displays current program	4
LO	Error message; line overflow	19
NE	Error message; nesting error	19
NEW	Command; erases program from memory	5
NEXT	Control statement; used with FOR to STEP loops	10
NS	Error message; negative square root argument	19
NULL	Command; transfers nulls to console device	5
OPENR	I/O statement; opens file prior to disk read	14
OPENW	I/O statement; open file to write data	14
OV	Error message; memory overflow	19
PASS	Function; used with CALL to link to assembly language program	17
PRINT	I/O statement; directs program to print on console device	6, 14
RE	Error message, read error	19
READ	Declaration statement; used with DATA to access values in buffer	8

MNEMONIC	DEFINITION	PAGE
REM	Statement; remarks, non-executable	3
RESTART	Command; clears error conditions	5
RESTORE	Declaration statement; resets buffer pointer to beginning after DATA-READ	8
RETURN	Subroutine statement; used with GOSUB, returns control to program	16
RND	Function; random number generator	17
RUN	Command; program is executed	3,5
RUNGO	Command; initializes BASIC-M	1
SIN	Function; sine	17
SGN	Function; gives value relative to zero	17
SQR	Function; Square root	17
STEP	Control statement; value in FOR-NEXT statement	10
STOP	Control statement; program stops executing	13
TAB	I/O Statement; used in PRINT to format list	14
TAN	Function; tangent	17
THEN	Control statement; used within IF statement	12
UL	Error message; undefined line	19

APPENDIX B

SUMMARY OF CHARACTER SET

CODE	DEFINITION	PAGE
@	Clears current line buffer	3,5
SHIFT/RUB	Clears single character	3,5
;	(Semicolon) Permits multiple statements on a line	7
,	(Comma) Permits multiple arguments in a command	5,8
:	(Colon) May be used in place of the word PRINT	15
%	Returns specified format to normal default format in PRINT-TAB statement	15
*	Multiply	9
/	Divide	9
+	Add	9
-	Subtract	9
" "	Causes literal alpha-numeric characters to print in output	7
-()	Negate	9
( )	(Parentheses) Controls negation sequence	9
=	Equal; relational operator	12
<>	Not Equal; " "	12
<	Less Than " "	12
>	Greater than" "	12
<=	Less than or equal "	12
=>	Greater than or equal	12