PTBASIC

Version 1.0

User's Manual

Peripheral Technology
1250 E. Piedmont Rd.
Marietta, GA 30062

Table of Contents

Table of Contents (Continued)

Appendix

Introduction

This manual assumes that the user has an understanding of BASIC and of the REX operating system. This manual is not a textbook on BASIC. The user should purchase a book on programming in BASIC if he/she does not have a good understanding of BASIC.

ABOUT BASIC

Program Lines

Program lines may be up to 128 characters in length. Attempting to type a line longer than 128 characters will result in an input buffer overflow error.

Program line numbers may range from 1 to 65535. All line numbers are unique. There can be only one "300" in a program. Typing another line with "300" will replace the old line with the new line. A line number followed by a return will delete the line containing that number.

Multiple statements may be entered on a line. Multiple statements must be separated with a colon, ":". It is better to enter multiple statements on separate lines from a speed standpoint, since PTBASIC processes separate statements faster than multiple statements per line.

Numbers

PTBASIC provides 14 digits of precision for numeric variables and arithmetic expressions. Scientific functions, LOG,EXP,TAN,SIN etc. are accurate to 13 digits of precision. All math is performed using BCD math. While BCD math is not as fast as binary math there are not round-off errors that occur when using binary math. The maximum range of number is + or - 9.9999999999999E99.

Strings

Maximum string length is 126 and is set by the STRING command. PTBASIC defaults to 32 characters. The result of string expressions must also fit in the length as defined by the STRING statement.

Variables

Numeric variables may be a single letter or a letter followed by an additional letter or number. For example: A, A9, AB are all legal numeral variable names. If numeric arrays are used [eg. A(10)] you cannot use a single variable of the same name. Example: you cannot use A and A(10) in the same program.

String variables consist of a single letter and/or a second letter or number followed by a "$"

character. Examples of legal string variable names are A$, A4$, AZ$. If a string array is used you may not use a simple string variable of the same name. Example: you may not use A$ and A$(15) in the same program.

Precedence of Expressions

The following table lists the arithmetic operators recognized by PTBASIC. They appear in order of precedence. Operations within parentheses are performed first. Inside the parentheses the usual order of precedence is maintained.

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

Relational Operators

Relational operators let you compare two values. The result of the comparison is either true or false. This result can then be used to make a decision regarding program flow.

Relational Operators

Operator	Relation Tested	Expression
=	Equal	A=B
<>	Not equal	A<>B
<	Less than	A<B
>	Greater than	A>B
<=	Less than or equal	A<=B
>=	Greater than or equal	A>=B

Input and Output

PT BASIC uses the I/O of REX for input and output. Input from the keyboard is through the INCH2 vector and output to the screen is through the OUTCH2 vector. Channel #0 is through the INCH and OUTCH vectors. Channel #1 will output to the POUT vector and channel #2 to the POUT2 vector; neither of these channels allows input. Channels 3 thru 9 will allow input or output with disk files through the FMS. PT BASIC commands that can use channel numbers are OPEN, CLOSE, LIST, INPUT, LINPUT and PRINT.

ABS Function

Purpose:

The ABS function returns the absolute value of the expression as its parameter. This means that numbers greater than or equal to zero are returned unchanged, while numbers less than zero are returned positive.

Syntax:

ABS(n)

Comments:

n must be a numeric expression.

Example:

```
PRINT ABS(6*(-4))
24
```

Prints 24 as the result of the action.

ASC Function

Purpose:

To return a numeric value that is the ASCII code for the first character of the string n$.

Syntax:

ASC(n$)

Comments:

The CHR$ function performs the inverse of the ASC function. See Appendix A for a list of ASCII codes.

Example:

```
100 N$="TOGETHER"
110 PRINT ASC(N$)
RUN
84
```

ATN Function

Purpose:

To return the arctangent of x, when x is expressed in radians.

Syntax:

ATN(x)

Example:

```
100 X=1.243
110 PRINT ATAN(X)
RUN
0.893314324377
```

BEEP Statement

Purpose:

To sound the speaker for about one-half second.

Syntax:

BEEP

Example:

```
100 IF X>20 THEN BEEP : PRINT "ERROR"
110 PRINT "HELLO WORLD"
```

CHAIN Statement

Purpose:

This command combines the functions of the LOAD and RUN commands. After loading the program, the new program is executed.

Syntax:

CHAIN filename

Comments:

filename is a legal REX file name.

Example:

```
20 PRINT "1 = PAYROLL"
30 PRINT "2 = GENERAL LEDGER"
40 INPUT "ENTER CHOICE ? ",A
50 ON A GOTO 70,80
60 GOTO 20
70 CHAIN "PAYROLL"
80 CHAIN "GL"
```

CHR$ Function

Purpose:

To convert an ASCII code to its equivalent character.

Syntax:

CHR$(n)

Comments:

n is a value from 0 to 255.

CHR$ is commonly used to send a special character to a terminal or printer. For example, CHR$(7) will beep the speaker. Other uses would be to print box/line characters.

See the ASCII chart in the appendix for a list of ASCII codes.

Example:

```
PRINT CHR$(65);
A
```

This example prints the ASCII character code 65, which is the uppercase letter A.

Close Statement

Purpose:

To terminate input or output to a disk file or device.

Syntax:

CLOSE [#n]

Comments:

Without any channel number, CLOSE will close all open files in DOS with the FMSCLS routine. It will also clear the channel vectors and restore the I/O to INCH2 and OUTCH2. It will not reset the OUTCH and INCH vectors in REX.

If n=0 the RSTRIO routine in REX will be invoked and the OUTCH and INCH vectors restored to OUTCH2 and INCH2.

If n=1 or 2. It is not necessary to close channels 1 and 2.

If n=3 to n=9, the open file will be closed, and the channel vector and I/O will be set through INCH2 and OUTCH2.

Example:

CLOSE	will close all open files
CLOSE #3	will close disk file number 3

CLS Statement

Purpose:

To clear the screen.

Syntax:

CLS

Comments:

This command clears a PC type screen. An RS232 terminal may or may not be cleared depending on the control codes required by the terminal. The CHR$ function could be used to send the appropriate clear screen codes to an RS232 terminal in the event the CLS statement dos not work.

COLOR Statement

Purpose:

To select colors for text modes.

Syntax:

COLOR foreground,background

Comments:

0 - Black	4 - Red8 - Gray		12 - Light Red
1 - Blue	5 - Magenta	9 - Light Blue	13 - Light Magenta
2 - Green	6 - Brown	10 - Light Green	14 - Yellow
3 - Cyan	7 - White	11 - Light Cyan	15 - High Intensity White

Example:

COLOR 7,1

Sets a blue background with white characters.

CONT Command

Purpose:

To continue program execution after a break.

Syntax:

CONT

Comments:

To continue program execution after a control-break, STOP, or END command. After using a STOP command you may examine or modify variables and then continue program execution. Do not use the CONT command if you modify the program.

COS Function

Purpose:

To return to the cosine of x, when x is expressed in radians.

Syntax:

COS(x)

Example:

```
PRINT COS(0.39)
0.924909059859
```

CSRLIN Variable

Purpose:

To return the current row position of the cursor.

Syntax:

y=CSRLIN

Comments:

y is a numeric variable. The value returned is between 1 and 25. The POS function will return the column position of the cursor.

DATA Statement

Purpose:

To store numeric and string constants that are accessed by the program READ statement.

Syntax:

DATA constants

Comments:

constants are numeric constants or string data separated by commas. String data must be enclosed with quote marks. DATA statements may be placed anywhere in the program. Placing DATA statements near the beginning will speed up program execution. If a READ statement attempts to read the wrong type of data (string data in a numeric variable) an error will result. See the RESTORE command for more information.

Example:

```
10 FOR Z=1 TO 5
20 READ I(Z)
30 NEXT Z
40 DATA 3,4,5,6,7
```

DATE$ Variable

Purpose:

To retrieve the current date.

Syntax:

string exp = DATE$

Comments:

The expression returned is in the form of mm-dd-yyyy.
mm is in the range of 0 to 12.
dd is in the range of 0 to 31.
yyyy is the year.

Example:

```
PRINT DATE$
04-04-1992
```

DEF FN Statement

Purpose:

To define and name a user written function.

Syntax:

DEF FNname[arguments]expression

Comments:

name is a single letter. The argument whether used or not must be defined. This argument is a dummy in that it is replaced during the function call by the value of the user's parameter. Functions must be defined on a single line and return a numeric result. A DEF FN statement must be executed before the function can be used.

Example:

```
190 INPUT "ENTER A NUMBER",B
200 DEF FNA(Z)=Z/3.14
220 B=FNB(1.3)
```

DIGITS Statement

Purpose:

To set the number of digits printed to the right of the decimal point.

Syntax:

DIGITS=numeric expression

Comments:

This command is supplied for compatibility with older versions of BASIC. Digits default to 13 when starting BASIC. DIGITS=0 will disable this function. Normally PRINT USING would be used for this type of function.

Example:

```
10 A=1/3
20 PRINT A
30 DIGITS=2
40 PRINT A
RUN
0.3333333333333
0.33
```

DIM Statement

Purpose:

To specify the maximum values for array subscripts and allocate memory for their use.

Syntax:

DIM variable(subscripts)[,variable(subscripts)]...

Comments:

If an array variable name is used without a DIM statement, it will default to a maximum size of 10. The minimum subscript is 0. Arrays can be double dimensioned. [eg. A(20,20)]. The DIM statement should be executed before using the array variable.

Examples:

DIM A(20),A$(30),B(50,50),A9(30,4)

END Statement

Purpose:

To terminate program execution and return to the command mode.

Syntax:

END

Comments:

END statements may be placed anywhere in the program.

Example:

10 IF A>20 THEN END

EXEC Command

Purpose:

To execute a REX command from a BASIC program.

Syntax:

EXEC command

Comments:

command can be a string variable or a text string. Since only a small space is available for execution of REX commands only the smaller utilities will execute such as DIR, DELETE, LIST. Attempting to run large programs such as ASMK will crash the system.

Examples:

```
10 EXEC "DIR"                    This will print the directory from BASIC.
20 D$="DELETE JUNK.BAK"
30 EXEC D$
```

EXP Function

Purpose:

To return the base of natural logarithm to the power x.

Syntax:

EXP(x)

Example:

```
PRINT EXP(2)
7.389056098934
```

FOR/NEXT Statement

Purpose:

To execute a series of instructions a specified number of times in a loop.

Syntax:

FOR variable=a TO b [STEP c]
.
NEXT variable

Comments:

variable is used as a counter.
a,b and c are numeric expressions.

a is the initial value the first time through the loop. The loop increments by c, or by 1 if c is not specified, each time through the loop. Looping will continue until a is equal or greater than the value of b. The loop will also execute at least one time even if the initial value of a is greater than the value contained in b. You may branch out of a loop with a GOTO, but it is not permissible to branch into the middle of a FOR/NEXT loop.

Example:

```
10 FOR A=1 TO 3
20 PRINT A;" TIME(S) THROUGH LOOP"
30 NEXT A
RUN
1 TIME(S) THROUGH LOOP
2 TIME(S) THROUGH LOOP
3 TIME(S) THROUGH LOOP
```

FRE Function

Purpose:

To return the amount of free program memory.

Syntax:

a=FRE(x)

Comments:

The argument (x) is a dummy argument and is not used. This is for compatibility with other BASIC's.

Example:

PRINT FRE(0) This statement prints the amount of program memory available.

GOSUB Statement

Purpose:

To branch to a subroutine and return to the next line.

Syntax:

GOSUB line number

Comments:

Control is passed to the line number named in the GOSUB statement. Program execution will continue until a RETURN statement is executed. When the RETURN is executed control is passed to a statement following the GOSUB statement. Use of subroutines makes programs smaller and easier to write and understand.

Example:

```
10 GOSUB 50
20 PRINT "RETURN FROM SUBROUTINE"
30 END
50 PRINT "START OF SUBROUTINE"
60 RETURN
```

GOTO Statement

Purpose:

To branch unconditionally out of the normal program sequence to a specified line number.

Syntax:

GOTO line number

Comments:

line number is any number between 1 and 65535.

Example:

```
10 INPUT "ENTER A NUMBER ? ",A
20 PRINT A*A
30 GOTO 10
RUN
ENTER A NUMBER ? 2
4
ENTER A NUMBER ? 4
16
ENTER A NUMBER ? 5
25
(This will continue forever unless a Control C is pressed)
```

HEX$ Function

Purpose:

To return a string that represents the hexadecimal value of the numeric argument.

Syntax:

v$=HEX$(x)

Comments:

HEX$ converts decimal values within the range of 0 to 65535 into a hexadecimal string expression within the range of 0000 to FFFF.

Example:

```
PRINT HEX$(32)
0020
```

HEX8$ Function

Purpose:

To return a string that represents the hexadecimal value of the numeric argument.

Syntax:

v$=HEX8$(x)

Comments:

HEX8$ converts decimal values within the range of 0 to 4,294,967,295 into a hexadecimal string in the range of 00000000 to FFFFFFFF.

Example

```
PRINT HEX8$(65536)
00010000
```

IF Statement

Purpose:

To make a decision regarding program flow based on the result returned by an expression.

Syntax:

IF expression THEN line number
IF expression THEN statement

Comments:

If the result of expression is true, control is passed to line number or statement is executed.

Example:

```
10 INPUT A
20 IF A<0 THEN PRINT "YOU MUST ENTER POSITIVE NUMBERS"
```

INKEY$ Statement

Purpose:

To return a single character from the keyboard.

Syntax:

a$=INKEY$

Comments:

If no character is ready, a null string is returned. If more than one character is in the buffer only the first is returned. The character returned is not displayed on the screen.

Example:

```
100 PRINT "PRESS ANY KEY TO CONTINUE"
100 A$=INKEY$
110 IF A$="" THEN 100
120 PRINT "IT'S ABOUT TIME, SLOW POKE"
```

INPUT Statement

Purpose:

To input data from the terminal and write it to a list of variables.

Syntax:

INPUT [prompt string;] list of variables

Comments:

A prompt may be specified by enclosing the message to be displayed in quotes, preceding the variable list. The variable list may be any combination of numeric and string variables. If not enough data is entered, PT BASIC will prompt with a '?' for more data. If non-numeric data is entered for a numeric variable a 'RE-ENTER' message will be displayed. Multiple entries of data must be separated with commas, and string data should be enclosed with quote marks "" .

Examples:

```
10 INPUT "ENTER THREE NUMBERS ? ";A,B,C
RUN
ENTER THREE NUMBERS ? 2,4,6
```

INPUT # Statement

Purpose:

To read data items from a file or device opened with an OPEN statement.

Syntax:

INPUT #n,list of variables

Comments:

n cannot be 1 or 2 since these are output only devices (PRINTER)
If n=0 and a serial device is opened, the program will wait on the INPUT statement until the appropriate data is entered.

Examples:

```
10 OPEN #3,"DATA.DAT"
20 INPUT #3,A,B,C
```

INT Function

Purpose:

To truncate an expression to a whole number.

Syntax:

INT(x)

Examples:

```
PRINT INT(69.69)
69

PRINT INT(-13.09)
-14
```

LEFT$ Function

Purpose:

To return a string that comprises the leftmost n characters of X$.

Syntax:

LEFT$(x$,n)

Comments:

n must be between 0 and 126. If n is greater than the length of the string, the entire string is returned. If n=0, the null string is returned.

Related Functions:

MID$, RIGHT$

LEN Function

Purpose:

To return the number of characters in x$.

Syntax:

LEN(x$)

Comments:

Nonprinting character, control codes, and blanks are included in the count.

Example:

```
10 X$="This is It."
RUN
11
```

LET Statement

Purpose:

To assign the value of an expression to a variable.

Syntax:

[LET] variable=expression

Comments:

The word **LET** is optional. LET A=5 and A=5 are equivalent statements. LET is seldom used today and is included for compatibility with older versions of BASIC that require the **LET** statement.

Example:

```
100 LET A=13
110 LET B=12+7
120 LET C=11^2
```

LINE INPUT Statement

Purpose:

To input an entire line (up to 126 characters) from the keyboard into a string variable, ignoring delimiters.

Syntax:

LINE INPUT [prompt string,]string variable

Comments:

prompt string is a string literal, displayed on the screen, that allows user input during program execution.

A question mark is not printed unless it is part of the prompt string.

The number of characters that can actually be put into the string is determined by the **STRING=** command.

Example:

```
10 LINE INPUT "Enter a Line ? ",X$
20 PRINT X$
RUN
Enter a Line ? A,B,C,D,E,F COMMAS ARE ALLOWED
A,B,C,D,E,F COMMAS ARE ALLOWED
```

LIST Command

Purpose:

To list all or a part of a program to the screen, line printer, or file.

Syntax:

LIST [#N,][line begin][,line end]

Comments:

line begin and line end must be between 1 and 65535.

Examples:

LIST #1	Lists all lines in the program to printer.
LIST 1,100	Lists lines from 1 through 100.

LOAD Command

Purpose:

To load a file from disk/diskette into memory.

Syntax:

LOAD filename

Comments:

filename refers to a legal REX DOS filename. Names must begin with a letter and can have up to eight characters. Numbers can be used for positions other than the first letter. The file extension defaults to .BAS and it is not necessary to enter the extension. Different drive numbers can be specified by a number and a period in front of the filename.

Examples:

LOAD "STARTREK"
LOAD "2.PAYROLL" Load a program named payroll from drive 2.

LOCATE Statement

Purpose:

To move the cursor to the specified position.

Syntax:

LOCATE row,col

Comments:

row is the screen line number and must be between 1 and 25.

col is the screen column number and must be between 1 and 80.

Example:

```
LOCATE 1,1
PRINT "HELLO"
LOCATE 12,1
PRINT "THIS IS LINE 12"
```

LOG Function

Purpose:

To return the natural logarithm of x.

Syntax:

LOG(x)

Comments:

x must be a number greater than zero.

Example:

```
PRINT LOG(2)
0.6931471

PRINT LOG(1)
0
```

MID$ Function

Purpose:

To return a string of m characters from v$, beginning with the nth.

Syntax:

MID$(x$,n[,m])

Comments:

n must be between 1 and 126.
m must be between 0 and 126.
If m is omitted, or if there are fewer than m characters to the right of n, all rightmost characters beginning with n are returned.

Example:

```
10 A$="Good Day America"
20 B$=MID$(A$,6,3)
30 PRINT B$
RUN
Day
```

MON Statement

Purpose:

To enter the MONK machine language monitor.

Syntax:

MON

Comments:

This command is not normally needed but may be of use in special debugging cases. After entering the MONK monitor an "S" will single step the program and a "G" will continue the program execution. See the MONK manual for more information on use of MONK.

Example:

```
10 A=4
20 MON
30 PRINT "END OF TEST"
RUN
PC=00007922  US=00030000  SR=0400 = ..4....
MONK:G
END OF TEST
```

NEW Command

Purpose:

To delete the program in memory and clear all variables.

Syntax:

NEW

Comments:

If NEW is encountered during the execution of a program, the program and all variables will be cleared.

Examples:

NEW

ON...GOSUB ON...GOTO Statements

Purpose:

To branch to one of several specified line numbers, depending on the value returned when the expression is evaluated.

Syntax:

ON expression GOTO line numbers
ON expression GOSUB line numbers

Comments:

The value of the expression determines which line number will be used for branching. A value of one will branch to the first line number, a value of 2 will branch to the second line number and so on. If the value of the expression is greater than the number of line numbers specified, program execution will continue on the next line.

Example:

```
5 INPUT "ENTER CHOICE (1-2) ? ",A
10 ON A GOTO 100,200
100 PRINT "CHOICE 1 " : STOP
200 PRINT "CHOICE 2 " : STOP
```

OPEN Statement

Purpose:

To establish input/output to a file or device.

Syntax:

OPEN [r/w] #n filename

Comments:

The r or w is optional and will default to read.

If channel #n = 0 then the filename should be the name of a routine that initializes the port, has the input and/or output routines, and sets the INCH and/or OUTCH vectors.

If n=2 or n=3, the REX printer initialization routine is called, output is set to the printer and the filename is ignored.

If n=3 to n=9, the filename is the drive #, filename, and extension of the file for reading or writing. The default extension is **.DAT**.

Example:

```
100 OPEN #3,"1.DATA.DAT"
110 OPEN #1,"PRINTER"
120 PRINT #1,"THIS PRINTS ON THE PRINTER"
```

PEEK Function

Purpose:

To read a byte from a specified memory location.

Syntax:

PEEK(a)

Comments:

This function returns a byte (decimal integer within the range of 0 to 255) from the specified memory location a. a must be between 0 and 16,777,215.

Example:

```
10 Z = PEEK(60000)
```
The value stored in memory location 60000 will be stored in the Z variable.

PII Constant

Purpose:

To return the value of pie (3.14159265359)

Syntax:

a=PII

Comments:

PII may be used in expressions just as any other variable. PII may not be on the left side of an equal mark.

Example:

```
100 A=PII/180
110 PRINT PII
RUN
3.14159265359
```

POKE Statement

Purpose:

To write a byte of data into a memory location.

Syntax:

POKE a,b

Comments:

a and b are integer expressions.
The data that is to be poked is expression b.
b must be an integer between 0 and 255.
a is an address that must be within the range of 0 to 16,777,215.

The **PEEK** statement will read a byte from memory.

Extreme care must be used with the **POKE** command. Poking to memory locations used by the operating system or BASIC could crash the computer.

Example:

Poke 60000,234
This statement will place a 234 decimal in memory location 60000.

POS Function

Purpose:

To return the current cursor position.

Syntax:

POS(c)

Comments:

c is a dummy argument. Position 1 is the leftmost position.

Example:

```
100 INPUT A$
110 PRINT A$;" THIS IS PART OF THE LINE ";
120 IF POS(A)>30 THEN PRINT
130 GOTO 100
```

PRINT Statement

Purpose:

To output to the display screen.

Syntax:

PRINT [list of expressions][;]
?[list of expressions][;]

Comments:

If the list of expressions is omitted, a blank line is displayed.

If a list of expressions is included, the values of the expressions are displayed. Expressions in the list may be numeric and/or string expressions, separated by commas, spaces, or semicolons. String constants must be enclosed in quote marks.

A question mark (?) may be used in place of the word PRINT.

The type of punctuation used after a variable or text string determines how the items are spaced.

If a semi-colon (;) is used, the next item will be printed immediately after the last. If a comma (,) is used, the item will print on the next 16 position zone. If PRINT statement is used with no list of expressions, a blank line will be printed. SPC and TAB may also be used in PRINT statements to control spacing.

Examples:

```
90 A=3
100 PRINT "There are";A;"cats present"
110 PRINT TAB(10);"LINE 1"
120 PRINT TAB(20);"LINE 2"
130 PRINT "BEGINNING";SPC(10);"END"
RUN
There are 3 cats present
        LINE 1
        LINE 2
BEGINNING         END
```

PRINT USING Statement

Purpose:

To print numbers using a specified format.

Syntax:

PRINT [#n] USING mask, output list

Comments:

The characters in the mask may perform the following functions:

- A digit may occupy this position.
$ - If this is the 1st or 2nd character in a mask, it will appear before the first significant digit of a number.
. - The position of a decimal point in a number.
* - If 1st character in mask, leading zeros will be replaced by '*'.
- - Will print the minus sign for negative numbers and a space for positive numbers at the (-) location.

Examples:

```
100 PRINT USING "## DOLLARS ## CENTS,A
RUN
12 DOLLARS 59 CENTS
```

READ Statement

Purpose:

To read values from a DATA statement and assign them to variables.

Syntax:

READ list of variables

Comments:

A READ statement must always be used with a DATA statement.

READ statements assign variables to DATA statement values on a one-to-one basis.

READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree an error is reported.

To reread DATA statements from the start use the RESTORE command.

REM Statement

Purpose:

To allow explanatory comments to be inserted in a program.

Syntax:

REM[comment]

Comments:

REM statements are not executed but are printed out as they were typed in when the program is listed. REM statements can make a program easier to understand; but their use can slow the program down, especially if a REM is used inside a FOR/NEXT loop.

Example:

```
100 REM PROGRAM NAME - STARTREK GAME
110 REM INIT SOME VARIABLES
120 A=4 : B=6
130 REM INITIALIZE THE GALEXY ARRAY
135 DIM G(40)
140 FOR Z=1 TO 40 : G(Z)=4 : NEXT Z
```

RESTORE Statement

Purpose:

To allow DATA statements to be reread from aspecified line.

Syntax:

RESTORE [line number]

Comments:

If line number is specified, the next READ statement accesses the first item in the specified DATA statement.

If line numberis omitted, the next READ statement accesses the first item in the first DATA statement.

Examples:

```
10 READ F,G,H
20 RESTORE
30 READ I,J,K
40 DATA 69,70,71
```

F AND I IS ASSIGNED 69, G AND J IS ASSIGNED 70, H AND K IS ASSIGNED 71.

RETURN Statement

Purpose:

To return from a subroutine.

Syntax:

RETURN

Comments:

The RETURN statement causes BASIC to branch back to the statement following the most recent GOSUB statement.

RIGHT$ Function

Purpose:

To return the rightmost **i** characters of string x$.

Syntax:

RIGHT$(x$,i)

Comments:

If i is equal to or greater than the length of x$, RIGHT$ returns all of x$. If i equals zero, the null string is returned.

Example:

```
10 A$="COPPER KETTLE"
20 PRINT RIGHT$(A$,6)
RUN
KETTLE
```

RND Function

Purpose:

To return a random number between 0 and 1.

Syntax:

RND[(x)]

Comments:

If x is 0, or if x is omitted, the next random number in the sequence is generated.

If x is greater than 0, the random number is reseeded from x; and the sequence starts from the seed. The same random number sequence can be repeated by reseeding the random number generator with the same number.

Example:

```
10 FOR A=1 TO 3
20 PRINT INT(RND*50)
30 NEXT A
RUN
3
35
17
```

Produces 3 random numbers between 0 and 50.

RUN Command

Purpose:

To execute a program in memory.

Syntax:

RUN

SAVE Command

Purpose:

To save a program on disk/diskette.

Syntax:

SAVE filename

Comments:

filename refers to a legal REX DOS filename. Names must begin with a letter and can have up to eight characters. Numbers can be used for positions other than the first letter. The file extension defaults to .BAS, and it is not necessary to enter the extension. Different drive numbers can be specified by a number and a period in front of the filename.

SGN Function

Purpose:

To return the sign of x.

Syntax:

SGN(x)

Comments:

x is any numeric expression.
If x is positive, SGN(x) returns 1.
If x is zero, SGN(x) returns 0.
If x is negative, SGN(x) returns -1.

SIN Function

Purpose:

To calculate the trigonometric sine of x, in radians.

Syntax:

SIN(i)

Example:

PRINT SIN(1.2)
0.932039085967

SOUND Statement

Purpose:

To generate sound through the speaker.

Syntax:

SOUND frequency,duration

Comments:

frequency is the desired frequency in Hz. (cycles per second.) frequency is a numeric expression and should be between 19 and 20,000. duration is measured in clock ticks. Clock ticks are 0.02 seconds. A duration of 0 will result in a continuous sound until another sound statement is executed.

Examples:

```
SOUND 600,1
SOUND RND*2000+29,8
```

SPACE$ Function

Purpose:

To return a string of x spaces.

Syntax:

SPACE$(x)

Comments:

x must be in the range of 0 to 126.

Example:

```
100 A$="BEGIN"
110 B$="END"
120 C$=SPACE$(10)
130 PRINT A$;B$;C$
RUN
BEGIN         END
```

SPC Function

Purpose:

To skip a specified number of spaces in a PRINT statement.

Syntax:

SPC(n)

Comments:

n must be between 0 and 255.

Example:

```
100 PRINT "BEGIN";SPC(5);"END"
RUN
BEGIN     END
```

SQR Function

Purpose:

To return the square root of x.

Syntax:

SQR(x)

Comments:

x must be greater than 0.

Example:

```
PRINT SQR(2)
1.414213562372
```

STOP Statement

Purpose:

To terminate the program and return to the command level.

Syntax:

STOP

Comments:

STOP may be placed anywhere in the program. Normally STOP is used during program debugging when you wish to pause the program, examine and/or change the variables, and then continue with program execution. The program can be continued with the CONT command, which would be entered from the command level.

Example:

```
10   IF A>10 THEN STOP
```
The statement above could be used during program debugging to allow you to examine other variables when A is greater than 10.

STR$ Function

Purpose:

To return a string representing the value of x.

Syntax:

STR$(x)

Example:

```
10 INPUT "Enter a number ? ",A
20 PRINT "THE NUMBER IS ";STR$(A)
RUN
Enter a number ? 6
THE NUMBER IS 6
```

STRING Statement

Purpose:

To set the maximum string length.

Syntax:

STRING=n

Comments:

n must be between 6 and 126. This statement determines the maximum string size.

Example:

```
10 REM SET MAXIMUM STRING LENGTH TO 40
20 STRING=40
30 REM THE NEXT STRING HAS 40 CHARACTERS
40 A$="1234567890123456789012345678901234567890"
```

SYSTEM Command

Purpose:

To return to REX.

Syntax:

SYSTEM

Comments:

SYSTEM, REX and DOS perform the same function.

TAB Function

Purpose:

Spaces to position n on the screen.

Syntax:

TAB(n)

Comments:

If the current print position is beyond n, TAB has no effect.

TAB may only be used with PRINT or PRINT #.

Example:

```
10 PRINT "NAME";TAB(25);"SS#"
20 PRINT "JONES";TAB(25);"258-37-4567"
RUN
NAME                    SS#
JONES                   258-37-4567
```

TAN Function

Purpose:

To calculate the trigonometric tangent of x, in radians.

Syntax:

TAN(x)

Examples:

```
PRINT TAN(1.3)
2.57215622124
```

TIME$ Variable

Purpose:

To retrieve the current time.

Syntax:

string exp = TIME$

Comments:

The expression returned is in the form of hh:mm:ss.
hh is in the range of 0 to 23.
mm is in the range of 0 to 59.
ss is in the range of 0 to 59.

Example:

```
PRINT TIME$
09:12:34
```

TIMER Function

Purpose:

To return the number of elapsed number of seconds, since BASIC was started.

Syntax:

v=TIMER

Comments:

The resolution of the timer is 0.02 seconds, and the timer resets to 0 after 21 minutes.

Example:

```
10 A=TIMER
20 FOR Z=1 TO 1000 : NEXT Z
30 PRINT "ELASED TIME ";TIMER-A
RUN
ELASED TIME 0.42
```

TRON/TROFF Commands

Purpose:

To trace the execution of program statements.

Syntax:

TRON
TROFF

Comments:

The trace function prints each line in brackets as the statement is executed. The trace is used to trace program flow when debugging programming. The TRON/TROFF can be executed in either a program line or in direct mode. Tracing is turned off when a NEW command is executed.

Example:

```
10 A=5
20 B=6
30 PRINT "LINE 30"
40 Z=6
RUN
[10]
[20]
[30]
LINE 30
[40]
```

USR Function

Purpose:

To call an assembly language subroutine.

Syntax:

USR(n)

Comments:

This statement is intended for the experienced machine language programmer. The long address (32 bits) of the machine language routine must be loaded at HEX addresses 6056,6057,6058 and 6059 . The address of the user program may be placed in BASIC by use of the POKE command.

VAL Function

Purpose:

Return the numerical value of string x$.

Syntax:

VAL(x$)

Comments:

If the first character is not numeric, the function will return a zero.

Example:

```
10 X$="40 days and nights"
20 PRINT VAL(X$)
RUN
40
```

VARPTR Function

Purpose:

To return the memory address of a variable.

Syntax:

VARPTR(variable name)

Comments:

This is an advanced programming function, used to pass the address of a variable to machine language subroutines.

Appendix A ASCII Character Codes

Dec	HEX	Chr	Dec	HEX	Chr	Dec	HEX	Chr
0	00		48	30	0	96	60	`
1	01	☺	49	31	1	97	61	a
2	02	☺	50	32	2	98	62	b
3	03	♥	51	33	3	99	63	c
4	04	♦	52	34	4	100	64	d
5	05	♣	53	35	5	101	65	e
6	06	♠	54	36	6	102	66	f
7	07	•	55	37	7	103	67	g
8	08		56	38	8	104	68	h
9	09	❍	57	39	9	105	69	i
10	0A	_	58	3A	:	106	6A	j
11	0B	_	59	3B	;	107	6B	k
12	0C	_	60	3C	<	108	6C	l
13	0D	_	61	3D	=	109	6D	m
14	0E	_	62	3E	>	110	6E	n
15	0F	☼	63	3F	ú	111	6F	o
16	10	➢	64	40	@	112	70	p
17	11	➢	65	41	A	113	71	q
18	12	⇳	66	42	B	114	72	r
19	13	!	67	43	C	115	73	s
20	14	¶	68	44	D	116	74	t
21	15	§	69	45	E	117	75	u
22	16	■	70	46	F	118	76	v
23	17	↑	71	47	G	119	77	w
24	18	↑	72	48	H	120	78	x
25	19	↓	73	49	I	121	79	y
26	1A	→	74	4A	J	122	7A	z
27	1B	←	75	4B	K	123	7B	{
28	1C	_	76	4C	L	124	7C	|
29	1D	↔	77	4D	M	125	7D	}
30	1E	➢	78	4E	N	126	7E	~
31	1F	➢	79	4F	O	127	7F	_
32	20		80	50	P	128	80	Ç
33	21	!	81	51	Q	129	81	ü
34	22	"	82	52	R	130	82	é
35	23	#	83	53	S	131	83	â
36	24	$	84	54	T	132	84	ä
37	25	%	85	55	U	133	85	à
38	26	&	86	56	V	134	86	å
39	27	'	87	57	W	135	87	ç
40	28	(	88	58	X	136	88	ê
41	29	)	89	59	Y	137	89	ë
42	2A	*	90	5A	Z	138	8A	è
43	2B	+	91	5B	[	139	8B	ï
44	2C	,	92	5C	\	140	8C	î
45	2D	-	93	5D	]	141	8D	ì
46	2E	.	94	5E	^	142	8E	Ä
47	2F	/	95	5F	_	143	8F	Å

Appendix A (*continued)*

144	90	É	192	C0	└	240	F0	≡
145	91	æ	193	C1	┴	241	F1	±
146	92	Æ	194	C2	┬	242	F2	≥
147	93	ô	195	C3	├	243	F3	≤
148	94	ö	196	C4	─	244	F4	⌠
149	95	ò	197	C5	┼	245	F5	⌡
150	96	û	198	C6	╞	246	F6	÷
151	97	ù	199	C7	╟	247	F7	≈
152	98	¯	200	C8	╚	248	F8	°
153	99	Ö	201	C9	╔	249	F9	∙
154	9A	Ü	202	CA	╩	250	FA	·
155	9B	¢	203	CB	╦	251	FB	√
156	9C	£	204	CC	╠	252	FC	⁻
157	9D	¥	205	CD	═	253	FD	²
158	9E	₋	206	CE	╬	254	FE	■
159	9F	*f*	207	CF	╧	255	FF	
160	A0	á	208	D0	╨			
161	A1	í	209	D1	╤			
162	A2	ó	210	D2	╥			
163	A3	ú	211	D3	╙			
164	A4	ñ	212	D4	╘			
165	A5	Ñ	213	D5	╒			
166	A6	ª	214	D6	╓			
167	A7	º	215	D7	╫			
168	A8	¿	216	D8	╪			
169	A9	⌐	217	D9	┘			
170	AA	¬	218	DA	┌			
171	AB	½	219	DB	█			
172	AC	¼	220	DC	▄			
173	AD	¡	221	DD	▌			
174	AE	«	222	DE	▐			
175	AF	»	223	DF	▀			
176	B0	░	224	E0	α			
177	B1	▒	225	E1	╤			
178	B2	▓	226	E2	Γ			
179	B3	│	227	E3	π			
180	B4	╣	228	E4	Σ			
181	B5	╡	229	E5	σ			
182	B6	╢	230	E6	µ			
183	B7	╖	231	E7	τ			
184	B8	╕	232	E8	Φ			
185	B9	╣	233	E9	Θ			
186	BA	║	234	EA	Ω			
187	BB	╗	235	EB	δ			
188	BC	╝	236	EC	∞			
189	BD	╜	237	ED	φ			
190	BE	╛	238	EE	ε			
191	BF	┐	239	ED	∩			

Appendix B

ASCII Table

	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		➢		0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
1	)	➢	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	ß	±
2	☺	⇳	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	♥	!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	♠	■	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	•	↑	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8		↑	(	8	H	X	h	x	ê	_	¿	╕	╚	╪	Φ	°
9	❍	↑	)	9	I	Y	i	y	ë	Ö	_	╣	╔	┘	Θ	·
A	_	↓	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
B	_	→	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	_	←	,	<	L	\	l	|	î	£	¼	╝	╠	▄	∞	_
D	_	_	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
E	_	↔	.	>	N	^	n	~	Ä	_	«	╛	╬	▐	ε	■
F	☼	⮟	/	?	O	_	o	_	Å	*ƒ*	»	┐	╧	▀	∩	