C&C Computers and Communications

N82-BASIC REFERENCE MANUAL

PC-8300

PC-8300-RM

NEC

NEC

Printed in Japan
78118292

NEC
PC-8300

N82-BASIC
REFERENCE MANUAL

The policy of NEC being that of continuous product improvement, the contents of this manual are subject to change, from time to time, without notice.

All efforts have been made to ensure that the contents of this manual are correct; however, should any errors be detected, NEC would greatly appreciate being informed.

NEC can assume no responsibility for errors in this manual or their consequences.

TABLE OF CONTENTS

CHAPTER 3 EXPRESSIONS AND OPERATIONS

CHAPTER 4 N82-BASIC INSTRUCTIONS

APPENDICES

INTRODUCTION

The N82-BASIC Refererce Manual is a guide to the programming language used for the PC-8300 personal computer. Microsoft's N82-BASIC language, developed specifically for the PC-8300, offers a wide range of commands and functions, making it very useful and versatile.

This Reference Manual was designed for people of all levels, from beginners to professional programmers. It is intended to be used in conjunction with the PC-8300 User's Guide.

This Manual is divided into nine chapters:

Chapter 1 is an overview of the N82-BASIC language. You will learn about the special features unique to N82-BASIC and its operating modes. This chapter also gets you started using N82-BASIC.

Chapter 2 includes all the general information about the BASIC language that you will need to know, such as definitions of statements and symbos used for programming. A description of the PC-8300 LCD screen display is included.

Chapter 3 explains how programming expressions are formed specificaly for the N82-BASIC language.

Chapter 4 includes complete explanations of the purpose and use of system commands, statements, and functions available with N82-BASIC.

Chapter 5 outlines information needed for proper file handling.

Chapter 6 describes machine language programming.

Chapter 7 is a guide to actual programming problems that may be encountered, especially with beginning programmers. Programming hints and solutions to programming problems are included.

Chapter 8 explains the causes of errors and what action should be taken when error messages are displayed.

Chapter 9 contains a variety of sample programs written in the N82-BASIC language.

Appendix includes quick reference tables and guides, memory maps, etc.

The PC-8300 is a very special personal computer. It has its own specialized built-in BASIC language, along with more easy-to-read special function keys than any other portable computer available. Another unique feature of the PC-8300 is its full screen editing capability which is extremely powerful for a compact portable computer.

In order to fully utilize the capabilities of the PC-8300, you should become familiar with the N82-BASIC language outlined in this Reference Manual.

It is best for beginning programmers to read this manual thoroughly, and actually input the sample programs into their PC-8300. More advanced programmers can use this manual as a reference.

The system commands, statements, and functions in chapter 4 are presented in alphabetical order for easy reference. The explanations are all written in the following format:

FUNCTION: Gives a brief description of the use of a command or function.

FORMAT: Describes how the instruction is actually written. The following points apply to the format description of all of the commands and functions:

1. All capitalized words are BASIC Reserved Words.

2. All lower case words contained within angle bracket < > symbols are parameters, which must be supplied by you.

   The three types of parameters:
   a. Line numbers: only whole numbers are allowed.
   b. Strings: enclosed by quotation marks. Any combinations of letters, numbers or other symbols are allowed.
   c. Varables: constants, numerical values, or mathematical formulas are allowed.

3. Parentheses ( ) must be typed in as shown in the format.

4. Braces { } indicate that the enclosed clause is optional, so you may choose to omit it.

5. Brackets [ ] denote that one of the enclosed items must be used.

6. Punctuation marks such as commas, periods, semicolons, etc., must be included in the format exactly as written.

7. Items enclosed by a pair of the "..." symbols can be repeated any number of times as long as they do not go over the maximum length of a line, which is 255 characters.

8. Placement of spaces between reserved words or parameters within the format of a command or function is not necessary

SAMPLE STATEMENT: This is a sample of the correct format of system commands, statements, and functions.

DESCRIPTION: Explains important points about the use of system commands, statements, and functions.

NOTE: Describes situations in which problems may arise if you do not fully understand the uses of a command or function.

SEE ALSO: Consists of other items relevant to the command or function being described.

SAMPLE PROGRAM: When included, a sample program demonstrates the actual use in a program of the system commands, statements, or functions described.

KEYS

Keys that must be pressed together require special notation. Here is an example:

Press [CTRL] [A]. This means "While holding down [CTRL], press A."

FUNCTION KEYS: If you see a written direction in text: "Press [f.'] ," all you have to do is press the key marked [f.1] .
Press [f.6] this means "Hold down [SHIFT] then press [f.1] ."

Remember the following:

[f.6] is the shift of [f.1]
[f.7] is the shift of [f.2]
[f.8] is the shift of [f.3]
[f.9] is the shift of [f.4]
[f.10] is the shift of [f.5]

USER INPUT NOTATION

Whatever the user should type is shown in bold print.
Here's an example:
Type **CAS:CALC.BA**
Press [↵] .

SYMBOLS USED IN THIS REFERENCE MANUAL

Remember this tip to avoid errors and problems!

CHAPTER 1

N82-BASIC OVERVIEW

N82-BASIC has been designed to fully utilize the many features of the PC-8300 personal computer. The language that is used is similar to many other forms of BASIC. In certain ways, it differs since the hardware features of the PC-8300 are different from those of other computers.

All of the hardware and software features of the PC-8300 are related to the N82-BASIC language:

1.1 INTERNAL AND EXTERNAL FEATURES

The LCD screen of the PC-8300 can handle high resolution graphics of 240 x 64 pixels (dots); amazing for a computer this size.

You can easily create and modify (edit) BASIC programs using the PC-8300's powerful screen editor. You also have the option to write and edit programs while in the TEXT mode, and to load them into the BASIC mode of the PC-8300. This TEXT feature transforms your PC-8300 into a powerful and versatile word processor.

Direct computer-to-computer communication from your PC-8300 to other types of computers is simple, convenient and is accomplished effectively through the RS-232C interface. This is done by using the TELCOM software feature, along with BASIC operation instructions, such as ON COM GOSUB, etc.

Large BASIC programs may be written on the PC-8300, because of its memory size. The PC-8300 comes equipped with 64K bytes of RAM installed, with two memory banks (#1 & #2) available for use. A third memory bank of 32K bytes (bank #3) is available if an additional RAM Cartridge is installed in the unit.

The PC-8300 can store up to 21 different files in each memory bank. This allows for 18 of your own customized files, along with the three primary files of BASIC, TEXT, and TELCOM. These files can be accessed faster and easier than with a disk drive on other computers.

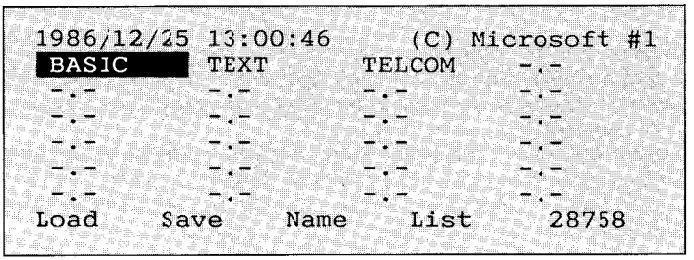
Battery power of the PC-8300 is conserved as efficiently as possible because of the automatic shut-off feature. This feature is operated by the POWER instruction, which can be written into the PC-8300 programs.

Data stored within the RAM of the PC-8300 is protected from loss by a back-up power system. This means that a minimal amount of battery power is used even when the power switch is turned off, allowing the files and programs stored in the RAM to remain intact, as long as the battery power lasts.

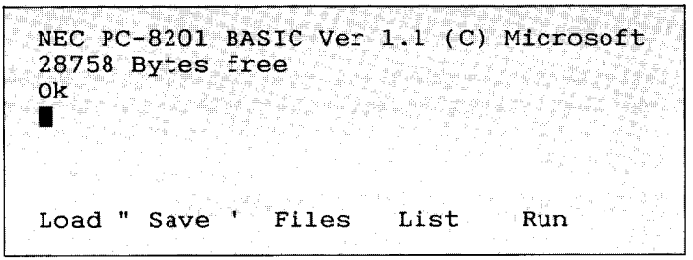
1.2 TWO OPERATING MODES, DIRECT AND PROGRAM

The N82-BASIC of the PC-8300 has two operating modes, the Direct Mode and the Program Mode, and they are both used in the BASIC mode.

As described in the PC-8300 User's Guide, the BASIC mode is entered by moving the directory cursor onto the word BASIC on the screen on the start-up menu called MENU, which is the first thing to appear on the screen, after turning on the PC-8300.



After pressing [↵], the message "Ok" will be displayed:



You can now utilize either the Direct Mode or the Program Mode of BASIC. The lines must conform to syntax requirements of the N82-BASIC language. The statements used in the Program mode must conform to command format requirements of N82-BASIC.

1.2.1 Direct Mode

The Direct Mode of BASIC allows an individual program statement, written in the N82-BASIC language, to be executed immediately. This is done by typing in the statement and then pressing [↵]. The statements used in the Direct Mode do not have a line number. The Direct Mode is useful for testing a particular statement. You can then see if the statement gives the result you expect it to, or if it performs a function correctly, without you having to write and run an entire program or set of statements.

Variables used in the Direct Mode are "held" in the memory temporarily, while you are working with them. They may be erased from the memory by typing NEW or CLEAR and then pressing [↵]. Statements written in the Direct Mode cannot be saved in RAM or onto external storage devices.

1.2.2 Program Mode

The Program Mode is entered by starting each statement with a line number, such as 10, 20, or 30.

The line number and the statement can then be saved and stored in the RAM, or on external storage devices such as RAM Cartridges, disk drives, and data recorders. Such numbered statements are "held" in the working memory but are not executed until a RUN command is keyed in, followed by [↵]. This way, multiple statements can be written to create a program. This differs from the Direct Mode because those unnumbered statements cannot be "SAVED" in the RAM or on external devices (disk drives or a data recorder). Line numbers used in the Program Mode can range from 0 to 65529.

Once a program has been created, it can be executed by a RUN command. The PC-8300 returns to the Direct Mode after a program has ended. This means that it switches back to the Direct Mode if a program finishes running normally, if a program terminates abnormally due to an error, or if [STOP] or [CTRL] [C] is pressed while a program is running.

The PC-8300 is device independent, allowing you to program on the PC-8300 without any peripheral devices attached. You have the option of attaching a data recorder, a RAM Cartridge, or a disk drive for the purpose of saving your programs, but it is certainly not necessary.

1.3 GETTING STARTED WITH N82-BASIC

To begin using the N82-BASIC language, put the PC-8300 into the BASIC Mode. Your screen should appear as illustrated:

```
NEC PC-8201 BASIC Ver 1.1 (C) Microsoft
28758 Bytes free
Ok
■


Load " Save " Files  List  Run
```

The "Ok" message with the flashing cursor appearing on the next line indicates that the PC-8300 is ready for use and is waiting for instructions from you. The PC-8300 is now in the Direct Mode, meaning that you can enter system commands or statements

When in the Direct Mode, commands and statements are always executed as soon as [↵] is pressed after they are typed. See Chapter 4 for a complete list of system commands.

Statements can be entered using either the Direct or the Program mode.

1.4 USING THE DIRECT MODE

The Direct Mode of N82-BASIC allows an individual statement to be executed immediately. Statements used n the Direct Mode are typed without line numbers, ard [↵] is then pressed to mmedately execute them.

An example of using the Direct Mode:

Type in: `INPUT "Racius of circle";R` [↵]

This statement causes the question: "Radius of circle?" to be printed on the screen, waiting for your answer to be input. Input the number for the radius and [↵]. (For example type `5`, then [↵].)

Type in: `PRINT "Diameter = ";2*R` [↵]

This statement calculates the diameter of a circle with the radius you have already input and prints (for example):

```
Diameter= 10
```

on the screen.

Type in: `PRINT "Area = ";3.14159*R^2` [↵]

3.14159 is the value of π.

This statement calculates the area of a circle, and prints:

```
Area= 78.5397
```

on the screen.

Type in: `PRINT "Circumference = ";2*3.14159*R` [↵]

This statement calculates the circumference of the circle and prints:

```
Circumference= 31.4159
```

on the screen.

While in the Direct Mode, the PC-8300 prints an "Ok" message on the screen each time [↵] is pressed at the end of a statement.

The Direct Mode is useful for testing particular statements, or for performing simple calculations. Most program statements can be entered in the Direct Mode, but not all can be executed. This is because some statements need to be executed in conjunction with other statements.

The PC-8300 retains the value of Radius (R) by holding it in a temporary working area of the memory. Values will remain until a CLEAR or NEW command is used, the power switch is turned off, another program is executed, or until the value is redefined.

Whenever you execute NEW or CLEAR, the value you had assigned to radius is cleared to zero, as are the values of all variables.

1.5 USING THE PROGRAM MODE

Assume that you wanted to know the diameter, area and circumference of a circle with a different radius, using the Direct Mode, you would have to repeat the whole process described. This is where the Program Mode comes in handy.

Type in the following:

```
10 INPUT "Radius of circle";R [↵]
20 PRINT "Diameter = ";2*R [↵]
30 PRINT "Area = ";3.14159*R^2 [↵]
40 PRINT "Circumference = ";2*3.14159*R [↵]
50 END [↵]
```

Now type **RUN** and press [↵]

If you typed the program correctly, the question "Radius of circle?" will appear on your screen. Type in a radius value and press [↵] (for example: 12).

Now you see the answers:

```
Diameter= 24
Area= 452.389
Circumference= 75.3982
```

Congratulations, you have written your first program. Now SAVE it in the RAM. Press [f.2] and then respond to the prompt **Save "** by typing:

RADIUS.BA" and press [↵].

Press [f.10] (hold [SHIFT] down and press [f.5]) to go to the MENU and you will see your program name among the other file names in the directory.

By pressing [SHIFT], you change [f.1], [f.2], [f.3], [f.4], [f.5], to [f.6], [f.7], [f.8], [f.9], and [f.10] respectively.

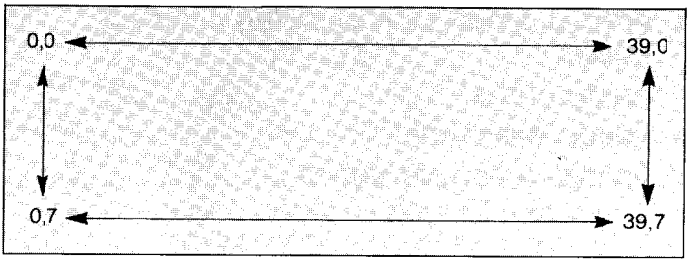
CHAPTER 2

GENERAL INFORMATION

2.1 SCREEN DISPLAY

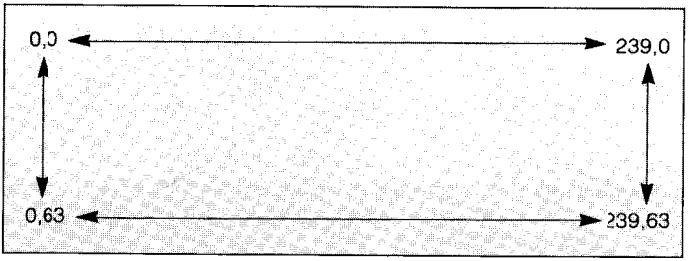
The Liquid Crystal Display (LCD) screen can display 8 lines of 40 characters per line. The first 7 lines are usually available for your use, depending on the mode in which the PC-8300 is in. The last line usually displays the names of the functions currently assigned to the five function keys, marked [f.1] to [f.5], which are just above the main keyboard.

The columns on the screen are numbered horizontally 0 through 39 and vertically 0 through 7 from top to bottom:



Each of the positions is addressable by using the LOCATE (x,y) statement, in which x gives the horizontal position (0-39) and y the vertical position (0-7).

Dot graphics may be displayed on the screen of the PC-8300. The display consists of 240 pixels (dots) across from left to right, numbered 0 through 239. There are 64 pixels from top to bottom on the screen, numbered 0 through 63:



Each dot is addressable using the PSET statement.

2.2 STATEMENTS AND LINE NUMBERS

BASIC programs consist of numbered statements, which give the PC-8300 instructions in ascending order of line numbers, starting at the lowest. These statements can perform arithmetic operations, assign values, input data, output data transfer the sequence of execution of certain program functions, test certain conditions within a program, etc.

A program line consists of one or more statements. If there are more statements than one in a line, the group of statements is called a compound statement. Statements within compound statements must be separated by a colon (:).
For example:

```
10 X=5:PRINT X
```

"X=5" is a statement, and PRINT X is another statement, so they must be separated by a colon.

Each program line begins with a line number, which indicates the sequence in which it is to be executed and stored in the memory. Program execution starts with the lowest numbered line and then continues in sequence. Line numbers can range from 0 to 65529. No program line can exceed 254 characters in length.

EXAMPLE OF A PROGRAM LINE FORMAT:

```
20 LET A=1:LET B=2:LET C=3
```

The above program line is a compound statement, starting with the no 20, and with the three individual statements separated by colons.

```
LET A=1
LET B=2
LET C=3
```

2.3 SPECIAL SYMBOLS

In addition to the arithmetic symbols, such as + (addition), – (subtraction), * (multiplication), and / (division), N82-BASIC reserves several symbols for special purposes:

- Period (.) is used to reference the last program line input. It is also used to point to the line in which an error has occured during program execution.
- Hyphen (–) indicates a range, in place of the word "to" such as 1–19. The hyphen is the same character as the minus sign.
- Comma (,) separates variables or data within a PRINT command into separate columns called Space Zones.
- Colon (:) is used to separate statements within one program line, which saves memory space.

- Semicolon (;) is usually used in the PRINT or INPUT statement as a non-spacing separator. Two items in a PRINT statement separated by a (;) will be printed without a space between them.
- Apostrophe (') is used to precede non-executable lines such as remarks or comments.
- Double quotation mark (") is used to enclose character strings. The strings cannot be more than 255 characters long.
- Question mark (?) is the abbreviation for the PRINT command.
- Blank spaces are generally ignored by the PC-8300, unless enclosed by " "

2.4 SPECIAL SYMBOLS FOLLOWING VARIABLE NAMES

Symbol	Format	Variable
Percent (%)	(variable)%	Integer
Exclamation (!)	(variable)!	Single Precision Real Number
Sharp (#)	(variable)#	Double Precision Real Number
Dollar ($)	(variable)$	Character String

2.5 CHARACTERS

The characters recognized by N82-BASIC include:

Upper case alphabet characters	A to Z
Lower case alphabet characters	a to z
Numeric characters	0 to 9
Special symbols	! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { }
Graphics characters	 and up to a total of 125 programmable graphics characters

2.6 ERROR MESSAGES

If an error occurs during program execution, the PC-8300 will terminate the program, return to the Direct Mode, and display an error message.

The error message is displayed on the screen if the PC-8300 is in the Direct Mode of BASIC. While in the Program Mode, the line number where the error occurred is displayed along with the error message. See Chapter 7 for the list of error messages and their explanations.

2.7 PROGRAM EDITING MODES

The two editing modes featured by the PC-8300 are the Direct Mode in BASIC and the TEXT mode. You can edit your programs in either mode, depending upon your preference.

2.7.1 Screen Editing of Programs

Editing programs in the BASIC mode is done by modifying program lines. When you edit in this manner, [↵] must be pressed after your changes have been made in order for them to be entered into the memory. Remember that a program line cannot exceed 254 characters long, which is more than 6 full lines on the screen. It is recommended that lines have less than 200 characters, so they may be LISTed and edited.

The following operations are used to edit (modify) program lines. First list the line by typing LIST and then the line number following by [↵].

INSERT:

1. Using the cursor keys, move the cursor to the place where one or more characters are to be inserted.
2. Press [PAST / INS].
3. Type the character(s) to be inserted.
4. If other insertions are needed on the same program line, move the cursor to the desired positions, again using the cursor keys, then press [↵] and insert the character(s).
5. Press [↵] to enter your changes into the memory.
6. Keep in mind that when INSERTion editing in the Direct Mode of BASIC is used, the INSERT is active until [↵] or a cursor key is pressed.

DELETE:

To delete characters that precede the cursor in a program line, LIST the line by typing LIST followed by the last number, then press [↵].

1. Move the cursor to the right of the character to be deleted.
2. Press [DEL / BS].
3. Press [DEL / BS] once for each character to the left to be deleted.
4. Press [↵] to store the changes.

To delete characters that follow the cursor in a program line, LIST the line by typing LIST followed by the last number, then press [↵].

1. Move the cursor onto the first character to be deleted.

2. Press and hold [SHIFT] and then press [DEL / BS].

3. Repeat the same process once for each character to be deleted. This converts the cursor position to a deletion site, deleting the character in the cursor position and pulls the other characters ock to it.

4. Press [↵] to store the changes.

To delete an entire line:

1. Type only the line number of the line to be deleted.

2. Press [↵].

Another way to delete an entire line is to LIST the line then:

1. Move the cursor to the space between the line number and the body of the statement.

2. Press [CTRL] [E], then press [↵].

ADD:

A new line can be added at any point in the program.

The PC-8300 executes programs in ascending order of line number, staring at the lowest line number, regardless of what order the lines were typed in.

To rewrite a line, just type the old line number followed by the new contents; this new line will automatically replace the old one. As stated above, the PC-8300 will put the lines in order when the program is LISTed.

2.7.2 Other Keys Used for Screen Editing

[TAB] Moves the cursor directory to columns, 8, 16, 24, and 32 of the line in which the cursor is positioned.

[STOP] Termirates the EDIT mode. [CTRL] [C] Same as [STOP]

[CTRL] [E] Erases characters from the position directly to the right of the cursor, all the way to the end of the program line.

[CTRL] [H] Same as [DEL / BS]

[CTRL] [I] Same as [TAB]

CTRL K	Moves the cursor to the "home" position, in the upper left corner of the screen.
CTRL L	Clears the screen and moves the cursor to "home" position, the upper left corner of the screen.
CTRL M	Same as ↵ .
CTRL O	Continues the scrolling of a program listing on the screen by the LIST instruction.
CTRL S	Interrupts the scrolling of a program listing on the screen by the LIST instruction.
CTRL R	Same as PAST NS
CTRL U	Erases a line displayed on the screen. The internal memory is not altered.

2.7.3 Editing Programs Using the TEXT Mode

Programs can be edited in the TEXT mode by entering EDIT and then pressing ↵ . To exit the TEXT editing mode, press ESC twice or f.10 (SHIFT f5).

In this mode, any characters typed are inserted at the location of the cursor. Unlike editing in the Direct Mode, every modification that you make in a program line is entered into the memory of the PC-8300 immediately, even before you press ↵ .

Use of TAB while in the TEXT editing mode will indent the line being typed. ↵ must be used to end a program line being typed or modified in this mode, or else the line will not appear in the program in the correct sequence.

The PC-8300 will check a newly input program line in the TEXT editing mode. If only a line number or a line which does not contain a line number is input, the PC-8300 will not store it in the memory. When this type of line is input the message 'Text ill-formed" will be displayed on the screen and a "BEEP" sound will be generated. You will have to type in a correct program line or delete the line number from the screen to avoid this error message.

2.7.4 Copying a Section of a Program Using the TEXT Mode

The TEXT editing mode is most useful if you want to copy a section of a program into another program by using the PASTE buffer. The string searching function of the FIND command is also very helpful in locating certain words, strings, etc., when you are editing programs. PASTE and FIND are described fully in the TEXT Manual.

CHAPTER 3

EXPRESSIONS AND OPERATIONS

3.1 VARIABLES

Variables are distinct quantities for different types of elements within your N82-BASIC programs that are represented by unique names. The two types of variables used are numeric and string variables.

An example of a numeric variable is when you want to use the element AGE within a program, and 40 items are needed. You can then assign the name "AGE" to represent the quantity of 40 items of that variable.

When you assign variable names, try to use names that are meaningful to you, and related to the element that they represent. The N82-BASIC language utilizes only the first two characters of the variabe name to distinguish between variables. A variable type specified character placed at the end of the variable name, indicates whether a variable is string or numeric.

Variable names may be any length up to 255 characters; however keep in mind that the longer the variable names the less RAM available for your subsequent use. The recommended characters to use for a variable names are letters and numbers.

The first character for the variable must be a letter. There are also certain words that are reserved for use within N82-BASIC that are not available for your use, such as all the N82-BASIC reserved words.

3.1.1 Examples of Reserved Variables

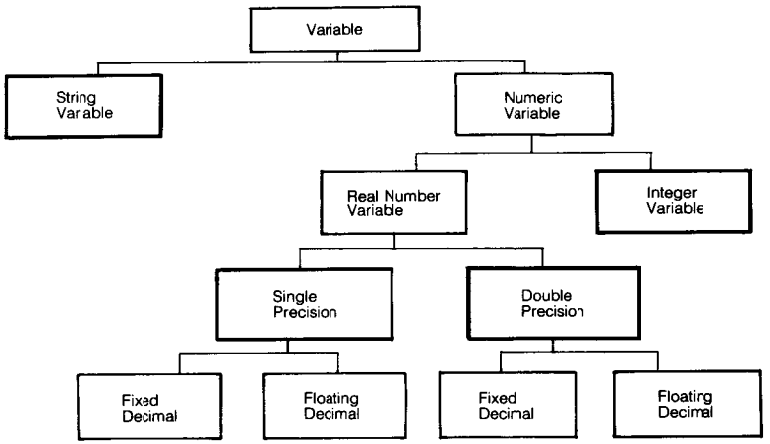
TIME$	-This variable holds the time in hours, minutes, and seconds **(HH:MM:SS)**
DATE$	-This variable holds the year, month and date **(YY/MM/DD)**.
ERL	-This variable holds the line number where an error occured during program execution.
ERR	-This variable holds the error code which caused the interruption.

See Appendix A.1 for a complete listing of N82-BASIC Reserved Words.

3.1.2 Types of Variables

The last character of a variable name determines the type of variable. The four types of variables are: integers, single precision real numbers, double precision real numbers, and string variables. If the variable type is omitted, it is assigned single precision (!) by default.

Following is a table of the different types of variables:



Variable type can be designated by using declaration statements.

Examples of different types of variable designations:

A$ String variable

A! or **A** Single precision real number variable (default)

A# Double precison real number variable

A% Integer variabe.

As you can see in the above example the variable name "A" in conjuction with special characters represent four different types of variables.

Please refer to DEFINT, DEFSNG, DEFDBL and DEFSTR statements in Chapter 4.

3.1.3 String Variables

String variables are a colection of cha'acters with a non-numeric value. They are composed of letters (both upper and lower case letters), numbers, or special symbols. If double quotations are needed within the string, CHR$(34) should be used to enter these double quotations. The maximum length of a string variable is 255 characters. They cannct be used in an arithmetic operation.

3.1.4 Numeric Variables

Numeric variables are integers or real numbers, represented by a numeric variable name such as age, salary, etc.

3.1.5 Integer Variable

In N82-BASIC, integers are numbers that have the following characteristics:

- Numbers with no decimal point.
- Numbers ranging from −32768 to +32767.
- Numbers followed by % (percentage sign).

EXAMPLES: `NUMBER%=1234`
`NUMBER%=123%.`

3.1.6 Real Number Variables

Real numbers are subdivided into single precision format and double precision format. Both single and double precision can have the numbers expressed in ether fixed decimal form or floating decimal form.

A fixed decimal form number may or may not show a decmal point (a decima point is assumed at the end of the number if it is not specified).

A floating decimal number is displayed in scientific notation.

3.1.6.1 Single Precision Format

A floating decimal single precision number has two parts, the magnitude and the exponent.

The magnitude is stored in seven significant (high order) digits internally. When displaying the numeric value, the seventh digit is rounded off and trailing zeros are deleted to show six digits or less on the screen.

The exponent portion is attached to the magnitude. It consists of the leter E, a sign, and a two digit number. Valid exponent numbers range from 01 to 38.

Single precision numbers have the folowing characteristics:

- Real numbers of up to 7 digits.
- Real numbers followed by an optional exclamation mark (!).
- Real numbers range from $-1.70141E+38$ to $1.70141E+38$.
- The exponent indicated by the letter E.

EXAMPLES: Fixed decimal:
```
NUMBER=1.23
NUMBER!=3.14!
```

Floating decimal:
```
NUMBER=-7.06E+06
NUMBER!=1.23E+10!
```

3.1.6.2 Double Precision Format

A double precision floating decimal number consists of two parts, the magnitude and the exponent as in the sngle precision format.

The magnitude is stored with a precison of 17 significant digits and can be displayed by up to 16 digits, with the 17th digit rounded off. The exponent is displayed starting with the letter D, followed by a sign and a two digit number. Valid exponent numbers range from 01 to 38.

Double precision numbers have the following characteristics:

- Numbers containing from 8 to 16 digits.
- Exponent indicated by the letter D
- Numbers followed by a sharp sign (#).

EXAMPLES: Fixed decimal:
```
NUMBER#=123456789012345
NUMBER#=0657036.1543976
```

Floating decimal:
```
NUMBER#=-1.09432D|06
NUMBER#=0.3141592653D+01.
```

3.2 ARRAYS

A group of logicaly related varables designated by the same variable name is callec an array. The items of an array are called elements. Each element is assigned a unique number called the subscript, to distinguish it.

Each array value is indexed by a subscript value. More than one subscript may be designated, thus specifying the dimension of the array. A single dimension array has one subscript index:

Subscripts: 0 1 2 3 4 5

Values: 11 91 36 12 19 50

When the elements of an array are designated with two subscripts then the array has two dimensions. This is explained by the following example. Let the array "ITEMS%" be two-dimensional, consisting of 4 rows by 8 columns. To reserve memory space for the array, the statement `DIM ITEM%(3,7)` would be used. Following is the layout of the location of each element of an array ITEM%:

Columns

Rows	—	0	1	2	3	4	5	6	7
	0	8	12	99	0	70	88	123	9
	1	23	88	56	91	87	72	192	23
	2	43	71	92	3	9	62	11	10
	3	51	82	95	64	93	57	26	4

As shown in the table, in order to access the fourth element of the second row you will have to use the name ITEM3%(1,3), this element contains the value 91.

3.2.1 DIM Statement

The subscripts are always enclosed in parentheses and they have a numeric integer value greater than or equal to zero. Numeric variables that follow the above rules can also be used when designating subscripts.

N82-BASIC requires information such as the maximum number of elements within each dimension of an array, so storage space can be allocated for the entire array. This is possible through the use of a DIM statement.

Sample format: `DIM ITEMS%(I,T)`

In this example, "I" represents the ROWS and "T" represents the COLUMNS. Notice that although there are 4 rows and 8 columns for each row, DIM(3,7) was specified. This is because counting of ROWS and columns start at 0 for the DIM statement. If we started with row 1 and column 1, memory space would have been wasted.

The layout for the array with dimensions (3,7) is addressed by subscripts according to the following table:

Columns

Rows	—	0	1	2	3	4	5	6	7
	0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
	1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
	2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
	3	(3,0)	(3,1)	(3,2)	(3,3)	(3.4)	(3.5)	(3.6)	(3,7)

An array can be expanded to include over 100 dimensions, (sub-elements of each element). The number of elements of an array is limited by the amount of memory space available.

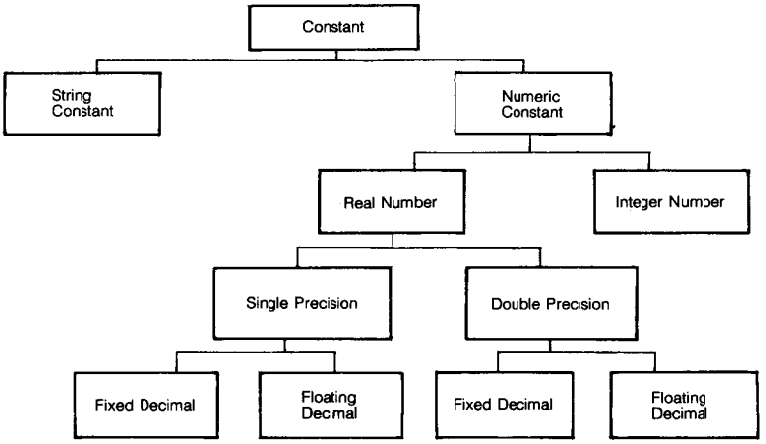
3.2.2 Rules for Elements of an Array

The array names, like the four different variable names, could represent the same types of information. The same rules as in the variables govern the different types of arrays. In addition to those rules, all the elements of an array can be of only one type. Also, if the array is a character array, no element should be longer than 255 characters.

3.3 CONSTANTS

Constants are values that you assign to variable names for use throughout your program or while in the Direct Mode. Constants are elements that do not and cannot change during the execution of a program.

Constants could represent the same types of information as variables. The same rules regarding designation of variables apply to constants. The following table illustrates types of constants used in BASIC:



3.3.1 Numeric Constants

A numeric constant has between 1 and 16 digits, either positive or negative. Numeric constants cannot contain any spaces. When numeric constants of more than 16 characters are used, the least significant digits are rounded off by N82-BASIC, and the number is displayed in floating decimal format. The following numeric constants are valid:

```
25.                          234567

-1234.01                     32760

1112345678901.23             .1234567890123

3.14159

.0000002
```

It is possible to enter numeric constants which are longer than 16 characters using the following format:

(+ or -)x.xxxxxxxxxxxxxxxD(+ or -)nn

where:

(+ or −)	is the sign of the number. The minus sign is required with negative numbers. If no sign is used, + is assumed (default).
x	is the number with up to 16 significant digits.
D	represents the Exponent (the power of 10).
nn	is the exponential value ranging from −38 to +37.

The Exponent in this format can be any number, including 0, but cannot be blank. The following are valid numeric constants in the D format:

```
1.2568D10                    8.254681325257D-30

-1.2345678901230-12          2358.25624798D2

1235D-30                     1.2D20
```

3.3.2 Integer Constants

An integer constant is a special type of numeric constant that is a whole number written without a decimal point, ranging from −32768 to +32767. For example, the following numbers are all integer constants:

```
1          0          -1234

25         -15        100

32767      -32767     10000
```

3.3.3 Character String Constants

A character string constant is one or more alphanumeric and/or special characters, enclosed in double quotation marks ("). Include both the starting and ending delimiters (quotation marks) when typing a character string constant in a program. A character can be a letter, a number, a space, or any ASCII character except a control character or quotation marks. (In such cases, use the CHR$ function and concatenate (connect) them into the string with the + sign).

The following is an example of acceptable character string constants:

Character String Constant

```
"Another"+CHR$(34)+"Constant"+CHR$(34)
```

Internal Representation

```
Another"Constant"
```

3.4 TYPE CONVERSION

Numeric variables can be converted from one type to another in N82-BASIC. Character string constants can be converted into numeric types and vice versa. The following are rules for type conversions:

1. When assigning variables, the type of numeric value being transferred depends upon the type of receiving variable.

   EXAMPLE:

Statement	Variable	Value
`ABC%=1.234`	`ABC%`	1
`ABC=1.234`	`ABC`	`1.234`

2. Numeric types are arranged in the following order of precedence:

   Integer

   Single Precision

   Double Precision

   As shown above, the integer is the lowest degree of precision. Arithmetic operations are performed in numeric values with the same degree of precision. If different types of numeric values are involved in an operation, the lower ordered values are converted into the higher ordered format first, before the operation is performed.

   EXAMPLE:

   `10#/3` is first converted to `10#/3#`

3. All numeric values used in logical operations are converted into integers. The result of the operation is in integer form.

   EXAMPLE:

Statement	**Variable**	**Content**
A#=12.34	A#	12.34000015258789
B=NOT A#	B	-13

4. Digits after a decimal point are omitted when real numbers are converted to integers. In such case, the number is rounded down to the nearest whole number. Numbers converted outside the valid range for integers (−32768 to +32767) would cause an overflow error.

   EXAMPLE:

Statement	**Variable**	**Content**
A%=34.4	A%	34
B%=34.5	B%	34

5. Values of Double Precision real numbers are rounded to 7 significant digits when converting to Single Precision numbers. An overflow error could occur if rounded values exceed the valid Single Precision range of −1.7014E+38 to +1.70141E+38.

   EXAMPLE:

Statement	**Variable**	**Content**
A#=1.23456789#	A#	1.23456789
B!=A#	B!	1.234567

6. Numbers within strings can be converted to numeric variables by using the VAL function.

   EXAMPLE:

Statement	**Variable**	**Content**
A#=12.34	A#	12.34000015258789 error factor
A!=12.34	A!	12.34
A#=VAL(STR$(A!))	A#	12.34 no error factor

7. Numeric variables can be converted into strings by using the STR$ function.

   EXAMPLE:

Statement	Variable	Content
A!=1.234	A!	1.234
A$=STR$(A!)	A$	"1.234"

3.5 LOGICAL EXPRESSIONS

A Logical Expression is the specification of a series of operations to be performed on variables, constants and functions, resulting in one value. The types of logical expressions used in N82-EASIC are:

- Arithmetic expressions
- Relational expressions
- Logical expressions
- String expressions

3.5.1 Arithmetic Expressions

Priority Number	Operator	Function
1 (high)	^	Exponentiation
2	-	Negative sign
3	*	Multiplication
3	/	Division
4	\	Integer division
5	MOD	Module division (Remainder)
6	+	Addition
6 (low)	-	Subtraction

An arithmetic expression is defined as:

< arithmetic term > < arthmetic operator >< arithmetic term >.

The following are examples of valid arithmetic expressions:

`NOT A%`	Integer result
`A%+23`	Integer result
`SUB+CURRENT*PRICE`	Single precision
`O%*THREE`	Single precision
`+1/-4`	Single precision
`3.14159*RADIUS^+2`	Single precision
`3*4/(PI#*R^2)`	Double precision

Rules for arithmetic expressions:

1. When there are different operators with the same priority, calculation is performed from left to right.

2. All arithmetic expressions are calculated from left to right with the highest priority (the lower priority number) operations being calculated first, followed by the lower order ones.

3. Lower priority expressions enclosed in parentheses in an arthmetic expression are performed before the higher pricrity expressions (outside the parentheses).

4. Priority order is also in effect inside parentheses.

5. Any division by zero will cause an error. Also, if zero is raised to a power of a negative number (for example `0^-6`) will cause an error.

6. An overflow error occurs whenever the results of an operation exceed the assigned variable type limits.

Example:

Statement	**Meaning**	**Result**
`Z*X+Y`	$ZX+Y$	(Z multiplied by X) plus X
`X/Y+2`	$\frac{X}{Y}+2$	(X divided by Y) plus 2
`(X+Y)/2`	$\frac{X+Y}{2}$	(X plus Y) divided by 2
`X^2+2*X+1`	X^2+2X+1	(X multiplied by X) plus X pus X plus 1
`X^(Y^2)`	$X^{(Y^2)}$	X multiplied by Y multiplied by Y
`X^Y^2`	$(X^Y)^2$	(X multiplied by X Y-Times) and the result multiplied by itself

X*(-X)	Y (−X)	Y multiplied by minus X
2/0	$\frac{2}{0}$	?/0 Error caused by division by zero
0/-1	$\frac{0}{-1}$	0
10\3	INT ($\frac{10}{3}$)	3
15MOD4	15 − 4 (INT ($\frac{15}{4}$))	3

3.5.2 Relational Expressions

A Relational Expression is defined as:

< arithmetic term > < relational operator > < arithmetic term >

or

< string term > < relational operator > < string term >

The following are all acceptable Relational Expressions:

CHARACTER$$>"HELLO"	String relation
NUM1<=NUM2	Numeric relation
NUMBER%<>225*(5-TWO)	Numeric relation with arithmetic sub-expression
539=TWO	Numeric relation

3.5.3 Logical Expressions

A Logical Expression operates on integer values and produces an integer value.
A logical expression is defined as:

< arithmetic term > < logical operator > < arithmetic term >

A logical operator is any of the following:

Operator	Function
NOT	Invert bits (ON to OFF; OFF to ON) in one term
AND	Tests for bit ON in both terms
OR	Tests for bit ON in either term
XOR	Tests for bit ON in either but not both terms
IMP	Tests both terms, it returns bit OFF if the first term bit is ON and the second term bit is OFF
EQV	Tests for equality, it returns bit ON only if both bits are ON or both OFF

The binary representation of ON is – 1, and 0 is the binary representation of OFF

Logical Expressions are comparisons between the corresponding "bits" of the two terms of the expression. A bit is a binary (either ON or OFF) piece of information. An integer value is composed of sixteen bits. A decimal integer is expressed in bits by converting the number to base 2 notation and adding any leading binary zeros, if necessary. The following is a list of some equivalent values in decimal and binary:

Decimal	Binary Bits
0	0000000000000000
1	0000000000000001
5	0000000000000101
23	0000000000010111
100	0000000001100100
-1	1111111111111111

Note that a decimal zero (0) has all zero bits and a decimal -1 has all one bits. This relationship between decimal and binary is used in the result of relational expressions. Logical expressions are valid wherever arithmetic expressions are allowed, however, both terms must be integers. The following tables are called truth tables. They show graphically the results of the logical operations for every possible combination of two bits

NOT	
A%	Not A%
0	−1
−1	0

OR		
A%	B%	A% or B%
0	0	0
0	−1	−1
−1	0	−1
−1	−1	−1

AND		
A%	B%	A% AND B%
0	0	0
0	−1	0
−1	0	0
−1	−1	−1

XOR		
A%	B%	A% XOR B%
0	0	0
0	−1	−1
−1	0	−1
−1	−1	0

IMP		
A%	B%	A% IMP B%
0	0	−1
0	−1	−1
−1	0	0
−1	−1	−1

EQV		
A%	B%	A% EQV B%
0	0	−1
0	−1	0
−1	0	0
−1	−1	−1

Examples of logical expressions

```
NUM1% OR NUM2%

I% AND 23

I% AND (A XOR B) IMP C%

(A AND B) OR (A AND C)

CHARACTER$>="A" AND CHARACTER$<="Z"
```

Logical expressions are normally used to evaluate terms that are the result of relational expressions (bits all ON or all OFF). However, since the logical expression compares all sixteen bits of each of the terms there are many other uses for logical expressions. One of the more common of these other uses is binary coded information, or "bit switches".

Some examples will illustrate how the logical operators work on non-relational values:

15 AND 14		0000 0000 0000 1111	(15)	
	AND	0000 0000 0000 1110	(14)	
		0000 0000 0000 1110	(14)	(TRUE)
10 OR 23		0000 0000 0000 1010	(10)	
	OR	0000 0000 0001 0111	(23)	
		0000 0000 0001 1111	(31)	(TRUE)
NOT 153	NOT	0000 0000 0001 1001	(153)	
		1111 1111 1110 0110	(−154)	(TRUE)
25 XOR 13		0000 0000 0001 1001	(25)	
	XOR	0000 0000 0000 1101	(13)	
		0000 0000 0001 0100	(20)	(TRUE)
234 EQV 3429		0000 0000 1110 1010	(234)	
	EQV	0001 1101 0110 0101	(34299)	
		1111 0010 0111 0000	(−3472)	(TRUE)
56 IMP 720		0000 0000 0011 1000	(56)	
	IMP	0000 0010 1101 0000	(720)	
		1111 1111 1101 0111	(−41)	(TRUE)

As you can see, there does not appear to be a relationship between the decimal terms and the decimal result of the expression. However, using the binary representations of the integers, there is a definite Boolean relationship. This can be utilized to make an integer value contain sixteen binary (ON/OFF) switches. When using binary switches the logical expressions can be utilized to set or mask the number to expose the bit switch desired.

3.5.4 String Expressions

Character strings can be joined together, broken down into shorter strings, and sorted into order.

3.5.4.1 Concatenating (Connecting Strings)

A string can be concatenated (connected end to end) with another string by the "+" operator. The resulting string cannot be more than 255 characters long

EXAMPLE:

Statement	Variable	Content
A$="NEC"	A$	NEC
B$=CHR$(34)+"PORTABLE"	B$	"PORTABLE
C$="COMPUTER"+CHR$(34)	C$	COMPUTER"
D$=A$+B$+C$	D$	NEC"PORTABLECOMPUTER"

3.5.4.2 Comparing Strings

When sorting strings, relational operators are used for the comparision of letters and numbers. Strings are compared one character at a time, starting from the beginning until there are no more related conditions.

Two strings are ecual if they have all the same characters in the respective positions, and both strings have the same number of characters. Otherwise, they are not equal.

EXAMPLES:

Relational Testing	Result
"AA"<"AB"	TRUE
"BASIC"="BASIC"	TRUE
"PENX"<"PEN"	FALSE
"cm"="CM"	FALSE
"cm">"CM"	TRUE
"DESK"<"DESKS"	TRUE

3.6 MATHEMATICAL FUNCTIONS

Mathematical functions are designated by enclosing the numeric value or numeric variable in parentheses and placing the value or variable after the function name. Most functions do calculations in single precision format. For integer functions all real numbers are converted into integers before function operation is performed.

EXAMPLES:

```
A=SIN(3.14)+COS(3.14)

PRINT2,2*2,SQR(2)
```

See Chapter 4 for a complete listing of functions available with N82-BASIC.

Mathematical formulas are a combination of numbers and variables related by arithmetic operators.

EXAMPLES:

```
"N82"+"BASIC"
3.14159*2
10+3/5
A+B/C-D
TAN(DO)+COS(DO)
10\3/2
13MOD2
```

3.7 HIERARCHY OF OPERATIONS

N82-BASIC operations are performed in the following order:

Precedence	
1	Expressions enclosed by parentheses
2	Functions
3	Exponential arithmetic (^)
4	Negative sign (-)
5	Multiplication and division (*, /)
6	Integer division (\)
7	Modulus arithmetic (**MOD**)
8	Addition and subtraction (+, -)
9	Relational operators =, <, >, <>, <=, =>, etc)
10	Logical operator **NOT**
11	Logical operator **AND**
12	Logical operator **OR**
13	Logical operator **XOR**
14	Logical operator **IMP**
15	Logical operator **EQV**

CHAPTER 4

N82-BASIC INSTRUCTIONS

ABS

FUNCTION: This function returns the absolute value of a number.

FORMAT: ABS (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT ABS(-8+7.9)
```

DESCRIPTION: An ABS function is used to determine the absolute value of a <numeric expression>, e.g. without a "+" or "−" sign.

SAMPLE PROGRAM:

```
10 FOR X=-3 TO 3
20 PRINT "THE ABSOLUTE VALUE OF ";X;" IS
 ";ABS(X)
30 NEXT X
40 END
```

AND

FUNCTION: This logical operator is used to test multiple relational expressions.

FORMAT: < operand 1> AND <operand 2>

SAMPLE STATEMENT:

```
IF A=5 AND B=6 THEN 30
```

DESCRIPTION: And is a logical operator that performs tests on mutiple relational expressions, bit manipulation, or Boolean operations. It returns either a non-zero (true) or zero (false) value.

For the conditional operation to be true, both <operands> must be true. If one or both is false, then the conditional operation is false. The table below indicates the evaluation process:

```
-1 AND -1→ -1  (TRUE AND TRUE →TRUE)

-1 AND  0→  0  (TRUE AND FALSE →FALSE)

 0 AND -1→  0  (FALSE AND TRUE→FALSE)

 0 AND  0→  0  (FALSE AND FALSE→FALSE)
```

For more details on logical operations and relational expressions see Chapter 3.

NOTE: Logical operators work by converting their <operands> to sixteen-bit binary integers. Therefore, the <operands> must be within the range of −32768 to +32767. If operands are not within this range, an "?OV Error"(Overflow) message will appear on the screen.

SEE ALSO: NOT OR, XOR, EQV, IMP, and Chapter 3.

EXAMPLE:

Integer	Binary bits
15	0000 0000 0000 1111
14	0000 0000 0000 1110

After you input the statement PRINT 15 AND 14, the integer 14 appears on the screen, whose binary representation is 0000 0000 0000 1110. By looking at the above table in the DESCRIPTION section, notice that the computation is correct.

SAMPLE PROGRAM:

```
10 A=5:B=6:C=7
20 IF A=5 AND B=6 THEN 40
30 PRINT "A IS NOT 5, OR B IS NOT 6"
40 IF A=5 AND C>6 THEN 70
50 PRINT "A IS NOT 5, OR C IS NOT GREATE
R THAN 6"
60 END
70 PRINT "A IS 5, B IS 6, AND C IS GREAT
ER THAN 6"
80 END
```

ASC

FUNCTION: This function provides the ASCII value of a character.

FORMAT: ASC (<string>)

SAMPLE STATEMENT:

```
PRINT ASC("AB")
```

DESCRIPTION: An ASC function determines the ASCII code of a character, or the ASCII code of the first character in the specified <string>. If the <string> is null (an empty string) the "?FC Error" (Illegal function call) message will be displayed on the screen.

SEE ALSO: CHR$ and Appendix A.4 Character Codes.

SAMPLE PROGRAM:

```
10 PRINT "THE ASCII VALUE OF D IS ";ASC(
"D")
20 PRINT "THE ASCII VALUE OF DAY IS ALSO
 ";ASC("DAY")
30 PRINT "PRESS ANY KEY TO CONTINUE . .
. "
40 IF INKEY$="" THEN 40
50 FOR X=32 TO 122
60 PRINT "THE ASCII VALUE OF ";CHR$(X);"
 IS ";ASC(CHR$(X))
70 NEXT X
80 END
```

ATN

FUNCTION: This function provides the inverse tangent of an angle.

FORMAT: ATN (<numeric expression>).

SAMPLE STATEMENT:

```
PRINT ATN(.05)
```

DESCRIPTION: An ATN function, used in trigonometric application, computes the inverse tangent (arc tangent) of an angle. The <numeric expression> is the angle expressed in radians, not in degrees.

The value obtained is within a range from $-\pi/2$ to $\pi/2$ (−90 to +90 degrees).

NOTE: To convert values from degrees to radians multiply the degrees by .0174533. To convert values from radians to degrees multiply the radians by 57.29578.

SEE ALSO: TAN, COS, and SIN.

SAMPLE PROGRAM:

```
10 FOR I=1 TO 5
20 PRINT "ENTER THE TANGENT OF AN ANGLE'
30 INPUT R
40 PRINT "THE ANGLE IS ";ATN(R);"RADIANS
, WHICH IS ";ATN(R)*57.2958;" DEGREES"
50 NEXT
60 END
```

BEEP

FUNCTION: This instruction is used to generate a "BEEP" sound from the PC-8300.

FORMAT: BEEP

SAMPLE STATEMENT:
```
BEEP
```

DESCRIPTION: The duration of the beep is approximately 0.12 second.

NOTE: The BEEP has no parameter.
PRINT CHR$(7) has the same function as the BEEP instruction.

SEE ALSO: SOUND.

SAMPLE PROGRAM:
```
10 FOR I=0 TO 6
20 READ W:BEEP
30 FOR J=0 TO W:NEXT J
40 NEXT I
50 DATA 10,100,10,10,100,300,100,100
60 END
```

BLOAD

FUNCTION: This instruction is used to load a machine language file into the memory.

FORMAT: BLOAD "{<external device name>:}<file name>"

SAMPLE STATEMENTS:
```
BLOAD "MACHLG"
BLOAD "CAS:HEXCAL"
```

DESCRIPTION: The BLOAD instruction loads a machine language program file specified by <file name> into the memory. The PC-8300 loads a machine language file from RAM if <external device name> is omitted.

Loading is not possible if a file in RAM is stored via the BSAVE instruction without specifying the file type. However, file type may be omitted when the actual loading process is executed.

If an execute start address is designated when a ".CO" file is created, this ".CO" file is executed as a subroutine immediately after it is loaded. Therefore, an additional EXEC instruction is not required after a ".CO" file is loaded.

The PC-8300 returns to BASIC from the subroutine by using a RET machine language instruction. It loads from a data recorded if "CAS:" is designated for <external device name>. The PC-8300 loads the first file it locates if <file name> is omitted.

SHIFT STOP interrupts the execution of a BLOAD "CAS:" instruction.

NOTE: Be certain to clear a memory region by using a CLEAR statement before a BLOAD instruction is executed. Otherwise, an "?OM Error" (Out of Memory) message is displayed and the PC-8300 reverts to the Direct Mode.

SEE ALSO: BSAVE, CLEAR, BLOAD?, EXEC, Chapter 5 Files, and Chapter 6 Machine Language Programming.

BLOAD?

FUNCTION: This instruction is used to compare/verify a machine language program currently in the memory with another program saved on cassette tape.

FORMAT: BLOAD? "[<external device name>:]<file name>"

SAMPLE STATEMENT:

```
BLOAD?"CAS:MACHLG"
```

DESCRIPTION: A machine language program in the memory and another machine language program on cassette tape can be compared and verified. This process is used to determine if a program file has been saved properly.

Execute a BLOAD?"<CAS:file name>" instruction only when a data recorder is connected to the PC-8300. If the contents of both programs are identical, the PC-8300 displays an "Ok" message. Otherwise, if any error has occured during the loading process, the PC-8300 will display the message "BAD" and execution is terminated.

The BLOAD? instruction should be used immediately after the BSAVE instruction has been executed.

SHIFT STOP interrupts the execution of a BLOAD?"CAS:" instruction.

BSAVE

FUNCTION: This instruction is used to save a machine language program from the memory into a designated file.

FORMAT: BSAVE "[<external device name>:]<file name>", <start address>, <length> [,<execute start location>]

SAMPLE STATEMENTS:
```
BSAVE"MACHLG",61000,256
BSAVE"CAS:MACHLG",61000,256
```

DESCRIPTION: The BSAVE instruction saves a machine language program or the contents of memory to a file designated by <file name>. The number of bytes specified by <length> is saved as the machine language program beginning at <start address>. BSAVE and BLOAD may be used for such a program only if it can be executed from <start address> (execution entry point).

The PC-8300 saves a machine language file to RAM if <external device name> is omitted. To specify a data recorder for the device name, "CAS:" is used.

If an <execute start location> option is designated, the contents can be stored as a ".CO" file. It is executed as a machine language subroutine when it is loaded by using the BLOAD insturction.

The <file name> cannot be omitted. In the sample statement, the contents are saved from memory location 61000 to 61255.

SHIFT STOP interrupts the execution of a BSAVE "CAS:" instruction.

SEE ALSO: BLOAD, Chapter 5 Files, and Chapter 6 Machine Language Programming.

CDBL

FUNCTION: This function converts integers or single precison real numbers to double precision real numbers.

FORMAT: CDBL(<numeric expression>)

SAMPLE STATEMENT:
```
PRINT CDBL(454.67)
```

DESCRIPTION: The CDBL function converts a < numeric expression > to a Double Precision real number without changing the effective number of digits.

NOTE: Refer to Type Conversion in Chapter 3.

SEE ALSO: CINT and CSNG.

SAMPLE PROGRAM:

```
10 DEFDBL D
20 A%=875
30 B1=45.3442
40 D1=CDBL(A%)
50 D2=CDBL(B1)
60 PRINT A%;TAB(20);D1
70 PRINT B1;TAB(20);D2
80 END
```

CHR$

FUNCTION: This function allows the PC-8300 to change a single value ASCII code to its matching character.

FORMAT: CHF$ (<numeric expression>)

SAMPLE STATEMENT:

```
A$=CHR$(65)
```

DESCRIPTION: This function returns a character specified by <numeric expression>. The ASCII character code represented by <numeric expression> can correspond to a letter, number, or any special character. The value of the <numeric expression> must be between 0 and 255, or an "?FC Error" (Illegal Function Call) message will be displayed.

Real numbers may be included in the <numeric expression> but the value is rounded off at the decimal point, to change them to integers.

SEE ALSO: ASC, and Appendix A.4 Character Codes.

SAMPLE PROGRAM:

```
10 FOR I=0 TO 28
20 READ C:PRINT C;" = ";CHR$(C):NEXT
30 DATA 36,32,130,68,79,94,100,125
40 DATA 95,63,129,64,85,80,102,126
50 DATA 33,122,111,125,99,81,38,55,96
60 DATA 117,37,63,77
70 END
```

CINT

FUNCTION: This function converts single or double precision real numbers to integers.

FORMAT: CINT (<numeric expression>)

SAMPLE STATEMENT:
```
CINT(4578)
```

DESCRIPTION: The CNT function rounds off (truncates) the value of a <numeric expression> at the decimal point, and returns an integer.

An "?OV Error" (overflow) message is displayed if the <numeric expression> is not between −32768 and +32767.

SEE ALSO: CDBL, CSNG, FIX, and INT.

CLEAR

FUNCTION: This instruction is used to reset all variables to null or zero, to establish the size of a string region, and to set the memory boundary.

FORMAT: CLEAR [<string area size>][,<maximum memory area used in BASIC>]

SAMPLE STATEMENT:
```
CLEAR 300,60000
```

DESCRIPTION: This instruction initializes all numeric variables to zero and all string variables to null strings. If parameters are omitted, the previous value is unchanged.

If large character string arrays are used, or a large number of character string operations are performed, designate only the first parameter.

The second parameter sets the maximum memory area used for BASIC and maintains the memory capacity used for machine language programs.

In the sample statement given above, the maximum memory area specified is 59999, thus a machine language program can be placed within the area ranging from 60000 to 62335. The locations beyond 62337 cannot be designated because they are reserved for internal functions of the PC-8300.

NOTE: When both parameters are omitted, only the initialization of the variables is executed, but the establishment of memory location remains unchanged. If the first parameter is specified, the string region is altered. The establishment of a region in the memory is not atered until a new CLEAR instruction is executed. Therefore, f a large string region is not secured in the program, an "?OS Error' (Out of String space) error message may occur during execution. When a CLEAR instruction is executed, al data in the PASTE buffer will be erased.

SEE ALSO: BLOAD, EXEC and DIM.

SAMPLE PROGRAM:

```
10 A$="ATW":B=486:C=7111
20 PRINT "A$ = ";A$;"   B = ";B;"   C =
";C
30 PRINT "CLEAR  !":BEEP
40 CLEAR
50 PRINT "A$ = ";A$;"   B = ";B;"   C =
";C
60 END
```

CLOAD

FUNCTION: This nstruction is used to load a recorded program from cassette tape into the memory.

FORMAT: CLOAD "<file name>"

SAMPLE STATEMENT:

```
CLOAD "DEMO"
```

DESCRIPTION: If a <file name> is specified, the PC-8300 will retrieve that program file from the cassette tape and load it into memory. However, when a <file name> is not specified, the PC-8300 loads the first program encountered from the cassete tape. A maximum of six characters can be used for a <file name>.

When a specific file is being sought, the system outputs a 'Skip: *filename*" message during the searching process. The PC-8300 will continue to scan the cassette tape until it finds the specific file, at which time t outputs a "Found: *filename*" message. An "Ok" message is displayed when the loading process is completed.

If the remote lead of the cassette cable is properly connected to the cata recorder, the PC-8300 can automatically turn the recorder ON and OFF during the LOAD process.

NOTE: IF <file name> exceeds 6 characters (not including the file type extension), or if a file of the specified <file name> does not exist on the tape the CLOAD instruction will search for the file name until the end of the tape is reached.

Even after an "Ok" message has appeared, it is possible that this loaded program may not operate properly, usually due to incorrect settings of the data recorder.

The CLOAD process can be interrupted by pressing SHIFT | STOP.

SEE ALSO: CSAVE, BLOAD, BLOAD?, BSAVE, NEW, LCAD, CLOAD?, and SAVE.

CLOAD?

FUNCTION: This instruction is used to compare/verify the program currently in memory with another program saved on cassette tape.

FORMAT: CLOAD? "<file name>"

SAMPLE STATEMENT:
```
CLOAD?"DEMO"
```

DESCRIPTION: The CLOAD? command is used to verify whether a previously CSAVED program (saved onto a cassette tape) matches the program currently residing in the memory. The <file name> refers to the program recorded on tape. If the contents of both programs are the same the system displays "Ok", but if the programs are not identical, the system displays "Bad" and execution is terminated.

This verification is useful to check that the program in the memory has been recorded correctly to tape. The CLOAD? instruction is normally used immediately after the CSAVE instruction.

SEE ALSO: CSAVE, CLOAD, BLOAD, BLOAD?, BSAVE, NEW, LOAD, and SAVE.

CLOSE

FUNCTION: This instruction is used to close files.

FORMAT: CLOSE [[#]<file number>], [[#],<file number>]...

SAMPLE STATEMENTS:

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

DESCRIPTION: This instruction is used to terminate input/output between a BASIC program and the data file(s). It closes the file corresponding to <file number>. If more than one <file number> is specified, as in the format explained above, the files are closed simultaneously. All currently opened files are closed if <file number> is omitted.

Before performing an input/output operation on a closed file, it must first be opened.

The CLOSE instruction writes out all data remaining in the file buffer. These files must be closed properly in order to correctly terminate file output.

SEE ALSO: OPEN, END, and NEW.

CLS

FUNCTION: This instruction erases the display from the screen.

FORMAT: CLS

SAMPLE STATEMENT:

```
CLS
```

DESCRIPTION: The CLS instruction clears all alphanumeric characters and graphics characters from the screen. If the currently assigned function key is on display, it will not be cleared by this instruction.

NOTE: This instruction has no parameter.

SAMPLE PROGRAM:

```
10 FOR I = 0 TO 40
20 X=RND(1)*35:Y=RND(1)*7
30 XP=RND(1)*240:YP=RND(1)*64
40 PSET(XP,YP)
50 LOCATE X,Y:PRINT "GARBAGE";
60 NEXT
70 LOCATE 0,0:PRINT "    PRESS RETURN TO
 CLEAR THE SCREEN    "
80 IF INKEY$<>CHR$(13) THEN 80
90 CLS
100 END
```

COM ON/OFF/STOP

FUNCTION: This instruction establishes, prohibits, or gives information about the interruption by a data transmission circuit.

FORMAT:

COM { ON / OFF / STOP }

SAMPLE STATEMENT:

```
COM ON
```

DESCRIPTION: The COM instruction informs BASIC that data may be input from an external cevice through the communication port (the RS-232C circuit).

The COM ON instruction establishes the possibility of a BASIC program being interrupted by data, from a data transmission circuit. Interruption by such communication from a data transmission circuit may then occur after this instruction is executed. The BASIC programming flow will then be diverted as a process routine designated by an ON COM GOSUB nstruction.

The COM OFF prohibits interruption of a BASIC program by communications input.

The COM STOP signals BASIC to warn about the occurrence of data, from a data transmission circuit. No diversion to any process routine will occur after this command is executed though the signal of the occurrence of the transmission is retained. After a subsequent COM ON instruction, diversion occurs to the ON COM GOSUB process routine.

SEE ALSO: ON COM GOSUB.

CONT

FUNCTION: The CONT instruction restarts the execution of a program that was interrupted, either by the STOP instruction, or by pressing [STOP].

FORMAT: CONT

SAMPLE STATEMENT:

```
CONT
```

DESCRIPTION: This instruction is normally used in conjunction with [STOP] (or [CTRL] [C]) to debug a program.

The CONT instruction is used to re-start the program after variable values, statements, etc. have been investigated in the Direct Mode. A complete program can also be listed on the screen when execution is interrupted.

By input of the CONT instruction or by pressing [f7] ([SHIFT] [f.2]), the program will resume execution from where the stop occurred. If the program has been altered while execution was stopped, then execution cannot be continued using this instruction and the error message "?CN Error" will be displayed.

COS

FUNCTION: This function returns the cosine of an angle.

FORMAT: COS (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT COS(3.14159)
```

DESCRIPTION: The COS function is used in trigonometric applications. It computes the cosine of a given angle. The <numeric expression> for the angle size is expressed in radians.

NOTE: To convert an angle from degrees to radians multipy the degrees by .0174533, or divide by 57.3, since one radian = 57.3 degrees.

SEE ALSO: SIN, TAN, and ATN.

SAMPLE PROGRAM:

```
10 INPUT "ENTER AN ANGLE EXPRESSED IN DE
GREES";D
20 PRINT "THE ANGLE EXPRESSED IN RADIANS
 IS ";
23 PRINT D*.0174533;" AND ITS COSINE IS
"COS(D*.0174533)
30 END
```

CSAVE

FUNCTION: This instruction is used to save a copy of the program which is in the memory to a cassette tape, using a data recorder.

FORMAT: CSAVE "<file name>"

SAMPLE STATEMENT: `CSAVE "DEMO"`

DESCRIPTION: This instruction saves a program currently in the memory orto cassette tape. The file name is specified using up to 6 characters. The PC-830C will return to the Direct Mode after the CSAVE instruction has been executed.

NOTE: A program file cannot be SAVEd to RAM once it has been shifted to the BASIC area by using a LCAD instruction. This is due to the fact that any modifications to the MENU-displayed program that is LOADed into BASIC automatically updates the program shown in the MENU. The LIST instruction should be used for final inspection before a CSAVE (to cassette tape) instruction is executed.

If interruption is necessary during the execution of a CSAVE instruction, press [SHIFT] [STOP].

SEE ALSO: CLOAD, SAVE, LOAD, BSAVE, and BLOAD.

CSRLIN

FUNCTION: The CSRLIN function determines the line number of the current cursor position, and returns this line number.

FORMAT: CSRLIN

SAMPLE STATEMENT: `PRINT CSRLIN`

DESCRIPTION: The CSRLIN (cursor line) function returns the line number of the current cursor position (vertical position).

The top line of the screen is always "0". Therefore, the value that is returned will be within the range of 0 to the number of lines of the screen minus 1. The number of the lines of the screen is either 7 or 8, depending on the mode. If the cursor is on the last line of the screen the CSRLIN function will return 6 or 7 as the result, depending on the mode.

SEE ALSO: POS.

SAMPLE PROGRAM:

```
10 CLS
20 PRINT "LINE 1 IS USED AS THE CURSOR L
INE:";CSRLIN
30 LOCATE 1,1:PRINT "LINE 2 IS USED AS T
HE CURSOR LINE:";CSRLIN
40 LOCATE 2,2:PRINT "LINE 3 IS USED AS T
HE CURSOR LINE:";CSRLIN
50 LOCATE 3,3:PRINT "LINE 4 IS USED AS T
HE CURSOR LINE:";CSRLIN
60 LOCATE 4,4:PRINT "LINE 5 IS USED AS T
HE CURSOR LINE:";CSRLIN
70 LOCATE 5,5:PRINT "LINE 6 IS USED AS T
HE CURSOR LINE:";CSRLIN
80 LOCATE 6,6:PRINT "LINE 7 IS USED AS T
HE CURSOR LINE:";CSRLIN
90 LOCATE 7,7:PRINT "LINE 8 IS USED AS T
HE CURSOR LINE:";CSRLIN
100 LOCATE 8,8:PRINT "LINE 9 IS USED AS
THE CURSOR LINE:";CSRLIN
110 END
```

DATA

FUNCTION: This instruction holds the constants which are loaded into the variables by a READ instruction.

FORMAT: DATA <constant> {,<constant>}...

SAMPLE STATEMENT:

```
DATA 1,CBA,1465
```

DESCRIPTION: A DATA instruction is used to define information to the READ instruction, and it can be inserted anywhere in the program. A program can have as many DATA instructions as needed with a maximum of 255 characters in each data statement.

READ instructions input constants from DATA instructions, starting from the DATA instruction with the smallest line number. However, READing will start again at the first data item after execution of the RESTORE instruction.

Arithmetic expressions used for reading in numeric constants are not permitted in DATA instruction. Constants are separated by commas on the data line. Their types should match the corresponding variable types in the READ instruction. Numeric constant type is converted into numeric variable type if the numeric types do not match. String constants are not type converted, so they must be read into a string variable.

When a string data element includes significant spaces (leading or trailing), or embedded commas, it must be enclosed by double quotation marks.

SEE ALSO: READ and RESTORE.

SAMPLE PROGRAM:

```
10 CLEAR 256:DIM A$(5),A(5):CLS
20 FOR I=0 TO 5
30 READ A$(I),A(I)
40 NEXT I
50 FOR I = 0 TO 5
60 LOCATE A(I),I:PRINT A$(I)
70 BEEP:NEXT I
80 LOCATE 0,0
90 DATA THIS,5,IS,11,HOW,16,TO,21,USE,25
,DATA,30
100 END
```

DATE$

FUNCTION: This reserved variable provides the data on the date for the internal real-time clock and calendar of the PC-8300.

FORMAT: DATE$ = "<year>/<month>/<day>"

SAMPLE STATEMENTS:

```
DATE$="83/03/18"
PRINT DATE$
```

DESCRIPTION: Using a double-digit format YY/MM/DD, the DATE$ is used to set year, month, and day. The values for <year>, <month>, and <day> are designated for the current date, or any desired date.

Once the date has been set correctly, reset of the date is not necessary, unless a cold start has been performed.

NOTE: The <year> value must be re-designated when the year advances because otherwise the timer repeats the same year.

SEE ALSO: TIME$.

DEF/INT/SNG/DBL/STR

FUNCTION: This instructon defines the format of a variable.

FORMAT:

DEF { INT / SNG / DBL / STR } <character range>

SAMPLE STATEMENT: `DEFINT A,I-K`

DESCRIPTION: By using the DEFINT instruction, a variable name that begins with a character designated by a <character range> can be designated as integer type.

In single precision real number format a DEFSNG instruction is used, in double precision real number format a DEFDBL instruction is used, while in string format a DEFSTR instruction is used.

Only one character may be used to specify each variable name, with its range designated by joining characters with a hyphen if contiguous characters are to be specified. (i.e., DEFINT X Y, Z can be entered as DEFINT X-Z).

Variable names followed by type declaration characters are given priority over variable names type-designated by the DEF instruction. All variables starting with characters which have not been type designated by a DEF instruction are assumed to be singe precision type.

SAMPLE PROGRAM:

```
10 DEFINT A-J,L:DEFSNG N-T
20 DEFDBL U-W:DEFSTR S,X-Z
30 A=53.9314558#:T=53.9314558#
40 W=53.9314558#:SE="  END"
50 PRINT A,T,W,SE
```

DIM

FUNCTION: The DIM (DIMension) instruction is used to allocate memory space for storing an array.

FORMAT: DIM <variable name> (<maximum subcript value> [,<maximum subscript value >...])

SAMPLE STATEMENT: `DIM A(12,2)`

DESCRIPTION: This instruction allocates memory space for the array area and sets the maximum subscript values for array variables. When an array variable is used and the DIM instruction is not defined, the maximum subscript value is set at the default 10. Any reference to an array beyond the allocated size will display a "?BS Error" (Bad Subscript, out of range) error message.

If the same array is defined more than once, a "?DD Error" (Duplicate Definition) error message will be displayed. By executing the CLEAR instruction this problem can be eliminated.

SEE ALSO: Chapter 3.2 Arrays.

SAMPLE PROGRAM:

```
10 PRINT "RND(1) 20 TIMES AND SORT THESE
 NUMBERS"
20 DIM R(19)
30 FOR I=0 TO 19:R(I)=RND(1):NEXT I
40 FOR I=0 TO 18:L=R(I):N=I
50 FOR J=I+1 TO 19
60 IF R(J)<L THEN L=R(J):N=J
70 NEXT J:T=R(I):R(I)=L:R(N)=T
80 NEXT I
90 FOR I=0 TO 19
100 PRINT USING "#.######";R(I);
110 NEXT I
120 END
```

EDIT

FUNCTION: This instruction changes the PC-8300 from the BASIC mode into the TEXT mode.

FORMAT: EDIT {<line on which to start editing>} {– <line on which to stop eciting>}

SAMPLE STATEMENT:

```
EDIT 20-80
```

DESCRIPTION: The instruction changes the PC-8300 into the TEXT mode and formats program editing. If a parameter, such as a line number or range, is not designated for editing, the entire program text is open for editing. Other combinations are also permitted, as explained below:

Parameter Specified	Line(s) Available For Editing
No parameter specifed	All
First parameter only (line no)	Only that line
First parameter and hyphen (start line no. & hyphen)	That line and all following lines
Hyphen and second parameter (hyphen and end line no.)	For the first line to the line specified by that parameter

First parameter, hyphen, and second parameter (first line nc. – end line no.)	The range consisting of all the lines between and including the two specified

SEE ALSO: Section 2.7.3.

END

FUNCTION: The END command is used on the last line of a program to terminate program execution.

FORMAT: END

SAMPLE STATEMENT: `END`

DESCRIPTION: This instruction terminates program execution, closes all files, and returns the PC-8300 to the Direct Mode.

The END instruction is inserted into the program at the location(s) at which it should terminate program execution. The final END instruction may be omitted from a program, but if so, the files are not closed.

SEE ALSO: STOP and CLOSE.

SAMPLE PROGRAM:

```
10 PRINT "PRESS ANY KEY"
20 IF INKEY$ = "" THEN 20
30 CLS:LOCATE 1,3
40 FOR I=0 TO 10:READ S,L,P$
50 PRINT P$;"    ";:SOUND S,L:NEXT
60 END
70 PRINT "THIS SECTION CANNOT BE EXECUTE
D."
80 DATA  11172,16,THIS,11172,32,IS,11172
,16,THE,11172
90 DATA 64,END,0,32,.,9394,32,MY,9952,32
,ONLY,12538
100 DATA 32,FRIEND,11172,48,.,9394,16,.,
11172,64,.
```

EOF

FUNCTION: This function determines if the end of a sequential file has been reached.

FORMAT: EOF(<file number>)

SAMPLE STATEMENT:

```
IF EOF(3) THEN CLOSE >1 ELSE GOTO 100
```

DESCRIPTION: An EOF (End Of File) function determines if the end of a sequertial file (designated by the <file number>), has been reached.

The function returns a non-zero (true) value if the end has been reached, but it returns a zero (false) value if the end has not been reached yet.

SAMPLE PROGRAM:

```
20 OPEN "TSTEOF" FOR OUTPUT AS #1
30 INPUT "HOW MANY TIMES DO YOU WANT TO
WRITE IN DATA?";N
40 FOR I=1 TO N
50 PRINT #1,I;
60 NEXT
70 CLOSE
80 OPEN 'TSTEOF" FOR INPUT AS #1
90 IF EOF(1) THEN PRINT "END OF FILE HAS
 BEEN REACHED":END
100 INPUT #1,N
110 GOTO 90
```

EQV

FUNCTION: This logical operator tests multiple relations.

FORMAT: < operand 1 > EQV < operand 2 >

SAMPLE STATEMENT:

```
PRINT 5 EQV 6
```

DESCRIPTION: The EQV (Equivalence) logical operator performs tests on multiple relations, Boolean operations, and bit manipulation. It returns either a non-zero (true) value, or a zero (false) value.

For the operation to be true, both <operand 1 > and < operand 2 > must be true(−1), or both of them must be false. But if one of them is true and the other is false then a zero (false) value is returned.

The following table indicates the evaluation process:

−1 EQV −1 → −1 (TRUE EQV TRUE → TRUE)

−1 EQV 0 → 0 (TRUE EQV FALSE → FALSE)

0 EQV −1 → 0 (FALSE EQV TRUE → FALSE)

0 EQV 0 → −1 (FALSE EQV FALSE → TRUE)

NOTE: EQV performs exactly the opposite to XOR. Logical operators convert their < operands > to sixteen bit binary integers. Therefore, each < operand > must be within the range of −32768 to +32767. If they are not within this range, an "?OV Error" (Overflow) error message will be displayed.

SEE ALSO: The AND, IMP, NOT, OR, XOR commands and Chapter 3.

EXAMPLE:

Integer	Binary bits
234	0000 0000 1110 1010
3429	0000 1101 0110 0101

After you input the statement PRINT 234 EQV 3429 the integer −3472 is returned, whose binary value is 1111 0010 0111 0000. By looking at the table under DESCRIPTION notice that the computation was done correctly.

ERL

FUNCTION: The ERL reserved variable provides the line number where an error occurs.

FORMAT: ERL

SAMPLE STATEMENT: `A=ERL`

DESCRIPTION: The ERL is a reserved variable used in error processing routines. It is used for displaying the line location of an error. It has the value of 65535 if an error occurs in the Direct Mode.

The content of ERL changes each time an error occurs during program execution. The value of ERL can be accessed, but cannot be assigned. The PC-8300 assigns the value to ERL when an error occurs.

NOTE: The ERL reserved variable has no parameters.

SEE ALSO: ON ERROR GOTO and ERROR.

ERR

FUNCTION: The ERR reserved variable provides the error code when an error occurs.

FORMAT: ERR

SAMPLE STATEMENT: `B=ERR`

DESCRIPTION: When an error occurs in the Direct Mode or during program execution, a message is displayed to indicate the cause of the error. Each error message is associated with a different error code.

The ERR is a reserved variable which contains the error code of the error detected. The content of ERR can be accessed but the values cannot be designated. The PC-8300 assigns a value to ERR when an error occurs.

NOTE: The ERR has no parameter.

SEE ALSO: ON ERROR GOTO and ERROR.

ERROR

FUNCTION: The ERROR instruction is used to simulate the occurrence of an existing error.

FORMAT: ERROR < integer >

SAMPLE STATEMENT: `ERROR 200`

DESCRIPTION: The value designated for < integer > must be between 0 and 255. When a specified value has been defined as a BASIC error code, the ERROR instruction simulates the ocurrence of that error and prints the corresponding message.

The ERROR instruction may be used as a user-defined or undefined error code. Under certain conditions, a program will branch to an error routine specified with the ON ERROR GOTO instruction.

SEE ALSO: ON ERROR GOTO, the ERL/ERR, and Appendix A.2 Error Codes.

SAMPLE PROGRAM:

```
20 ON ERROR GOTO 500
30 A=1/0
40 GOTO 0
50 NEXT
60 PRINT SQR(-2)
70 ERROR 255
80 END
500 PRINT "ERROR" ERR " IN LINE NUMBER "
ERL
510 IF ERR=11 THEN PRINT "A DIVISION BY
ZERO";
520 IF ERR=8 THEN PRINT "AN UNDEFINED LI
NE NUMBER";
530 IF ERR=1 THEN PRINT "NEXT WITHOUT FO
R";
540 IF ERR=5 THEN PRINT "AN ILLEGAL FUNC
TION CALL";
550 IF ERR=255 THEN PRINT "AN UNDEFINED"
;
560 PRINT" ERROR HAS OCCURRED.":PRINT
570 RESUME NEXT
```

EXEC

FUNCTION: This instruction executes a machine language subroutine.

FORMAT: EXEC <initial location>

SAMPLE STATEMENT:

```
EXEC 61000
```

DESCRIPTION: The EXEC instruction tranfers control to a machine language subroutine in the memory. The <initial location> is designated by integers ranging from 33468 to 65535. A negative number, if used for <initial location> should be subtracted from 65536 (thus a negative 1 is 65536 −1, or 65535).

If values are POKEd into the following locations, they can be transferred to the A, L, and H registers, respectively. After the system returns to BASIC from the subroutine, it is possible to obtain results by investigating the same locations using the PEEK function.

A Register Location 63911
L Register Location 63912
H Register Location 63913

The PC-8300 can return to BASIC from a machine language subroutine via the RETurn instruction.

NOTE: Select <initial location> carefully to avoid erratic operation.

SEE ALSO: BLOAD, PEEK, and POKE.

EXP

FUNCTION: This function calculates the value of "e"(base value of natural logarithm = 2.71828) raised to the power specified in the parameter.

FORMAT: EXP ([<arithmetic expression> / <numeric constant> / <numeric variable>])

SAMPLE STATEMENT:
```
A=EXP(1)
```

DESCRIPTION: This function returns the value of "e" raised to the specified power in single precision format. An "?OV Error" (Overflow) error message will result if the power is greater than 87.33655.

FILES

FUNCTION: This instruction displays all the names of the files in the RAM.

FORMAT: FILES

SAMPLE STATEMENT:
```
FILES
```

DESCRIPTION: This instruction displays all of the file names (including file type extensions) stored in the RAM.

The file type "BA" denotes a BASIC program file, ".DO" denotes a TEXT file, and ".CO" denotes a machine language program. When an asterisk (*) is displayed directly after the file type extension ".BA", this means that it is currently accessible.

SEE ALSO: Chapter 5, Files.

SAMPLE PROGRAM:

```
10 REM THIS PROGRAM MAY BE DESTROYED UPO
N EXECUTION,
20 REM SO SAVE IT BEFORE RUNNING IT
30 ON ERROR GOTO 170
40 PRINT "TO USE IN ONE OF THE FOLLOWING
 PROCESSES--LOAD, OPEN, BLOAD--"
50 FILES
60 INPUT "WHICH FILE NAME + FILE TYPE DO
 YOU WANT TO SELECT";N$
70 K$=RIGHT$(N$,3)
80 IF K$=".BA" THEN 120
90 IF K$=".DO" THEN 130
100 IF K$=".CO" THEN 140
110 PRINT "THE FILE NAME THAT YOU DESIGN
ATE DOES NOT EXIST!":BEEP:GOTO 40
120 LOAD N$
130 OPEN N$ FOR INPUT AS #1:GOTO 150
140 BLOAD N$
150 INPUT #1,A$:PRINTA$:IF NOT (EOF(1))
THEN 150
160 END
170 RESUME 110
```

FIX

FUNCTION: This functon returns the integer portion of a number.

FORMAT: FIX (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT FIX(9.9)
```

DESCRIPTION: The FIX function returns the integer portion of the <numeric expression>. It will omit the digits after the decimal point

NOTE: This functon does not round off the number.

SEE ALSO: INT and CINT.

SAMPLE PROGRAM:

```
10 PRINT "      I      FIX      INT"
20 FOR I=-2 TO 2 STEP .5
30 PRINT USING "###.##    #####    #####";
I,FIX(I),INT(I)
40 NEXT
50 END
```

FOR...TO...STEP ~ NEXT

FUNCTION: This instruction repeats a series of instructions a designated number of times.

FORMAT:

```
FOR <variable name> = <initial value> TO
    <final value> [STEP<increment>]
      :
      :
NEXT [<variable name>[,<variable name list>]]
```

where:

<initial value>: <numeric expression>

<final value>: <numeric expression>

<increment>: <numeric expression>

SAMPLE STATEMENT:

```
FOR J=0 TO 100 STEP 10
  :
  :
NEXT J
```

DESCRIPTION: The FOR...TO...STEP ~ NEXT instruction executes a series of statements a given number of times. This is one type of loop.

The <variable name> is used as a counter, which at the beginning is set to the <initial value>. Each time the sequence is completed and the NEXT instruction is encountered, the <variable name> increases or decreases as specified by the <increment> in the STEP parameter.

The value of the <variable name> is compared with the <final value>, and the loop will stop executing when the terminating condition is met or exceeded. Once the value of the <variable name> exceeds the <final value>, program control is passed to the statement following the NEXT instruction.

The <variable name> in the NEXT instruction may be omitted. NEXT always terminates the last unmatched FOR instruction. If a <variable name list> is used and the variable list is not in proper sequence, the nested loops will not terminate correctly.

If the STEP parameter is omitted the default value <increment> is +1. A negative value may also be specified as an <increment>.

The loop is executed only once in the following cases:

- When <increment> is positive, and <initial value> is greater than <final value>.
- When <increment> is negative, and <initial value> is less than <final value>.
- When <initial value> is equal to <final value> no matter what the <increment> is.
- When there is not a matching NEXT instruction.

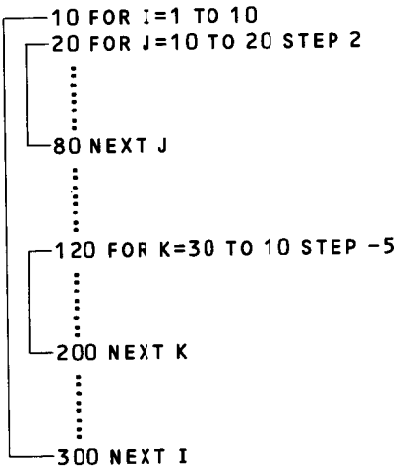
If <increment> is zero then the loop is executed continuously (infinite loop), press [STOP] for interruption.

FOR-NEXT loops may be nested to any depth. In such cases different <variable names> must be used, and the second loop must be completely located within the first loop. An "?NF Error" (Next without For) occurs if there is an illegal form of nesting.

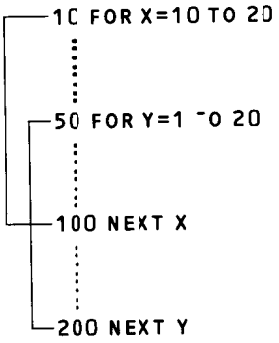
The loop may be exited with a GOTO instruction. The loop will remain open until another loop is executed using the same <variable name>, or when the loop is re-entered.

After a loop is terminated the <variable name> has the value of the <final value> +1.

NOTE: A common practice to determine whether or not the nested loops are legal is to draw lines between the matching FOR and NEXT instructions. If the lines cross each other, then the nesting is illegal. For example:



The above is an example of legal nesting.



The above is an example of illegal nesting.

SAMPLE PROGRAM:

```
10 FOR I=1 TO 5
20 FOR J=16000 TO 1000 STEP -1000
30 SOUND J,I
40 NEXT J,I
50 END
```

FRE

FUNCTION: This function reports the amount of unused or free memory area.

FORMAT: FRE (<expression>)

where:

<expression> = [<character string> | <character variable> | <numeric expression> | <numeric variable>]

SAMPLE STATEMENTS:

```
PRINT FRE(A)
PRINT FRE(A$)
```

DESCRIPTION: The FRE function calculates the amount of free string memory or the amount of free program memory. The value returned is the number of unused, available bytes of memory.

If the <expression> is a <character string> or a <character variable>, the FRE function returns the amount of string space available.

If the <expression> is a <numeric expression> or <numeric variable>, the FRE function returns the amount of memory space available.

SAMPLE PROGRAM:

```
10 PRINT "INITIAL AMOUNT = ";FRE(0)
20 PRINT "STRING AREA = ";FRE(A$)
30 CLEAR 500
40 PRINT "AMOUNT OF PROGRAM NOW = ";FRE(
0)
50 PRINT "STRING SPACE NOW = ";FRE(A$)
60 END
```

GOSUB ~ RETURN

FUNCTION: The GOSUB instruction transfers control to a specified line number (beginning of the subroutine). The RETURN branches back to the next statement after the GOSUB instruction in the main program when the execution of the subroutine is completed.

FORMAT: GOSUB <line number>

SAMPLE STATEMENT:

```
GOSUB 1000
```

DESCRIPTION: The GOSUB instruction is used to eliminate repeating frequently used routines. The subroutine is a portion of the main program, and it starts with a specific line number and terminates with a RETURN instruction. However, a subroutine can have more than one RETURN instruction, depending on the specific subroutine.

Subroutines are called by the GOSUB instruction, in order to perform the same sequence of instructions at different points of the program.

Subroutines usually reside at the end of a BASIC program, and the instruction GOSUB is used to call the subroutines. When a RETURN instruction is reached in the subroutine, control is passed to the main program and execution continues from the first statement following the GOSUB instruction.

The procedure of one subroutine calling another subroutine is called "subroutine nesting". Such a procedure can take place as long as the memory stack is not overflowed. (Seven stack bytes are used for each GOSUB. The RETURN will put the stack back to normal).

SEE ALSO: RETURN.

SAMPLE PROGRAM:

```
10 GOSUB 30:GOSUB 50:GOSUB 70
20 END
30 FOR I=0 TO 9:PRINT "FIRST ROUTINE":NE
XT I
40 BEEP:RETURN
50 FOR I=0 TO 9:PRINT "SECOND ROUTINE":N
EXT I
60 BEEP:RETURN
70 FOR I=0 TO 9:PRINT "THIRD ROUTINE":NE
XT I
80 BEEP:RETURN
```

GOTO

FUNCTION: This instruction branches the program execution to a designated line number.

FORMAT: [GOTO / GO TO] <line number>

SAMPLE STATEMENTS:
```
GOTO 500
GO TO 500
```

DESCRIPTION: This instruction unconditionally branches to the specified <line number> in the program.

NOTE: This instruction may be written either as "GOTO" or "GO TO". If two or more blank spaces are entered between the words "GO" and "TO", then N82-BASIC does not interpret it as the GOTO instruction.

SEE ALSO: IF and GOSUB.

SAMPLE PROGRAM:
```
20 GOTO 60
30 PRINT " SPAGHETTI.":GOTO 70
40 PRINT " CALLED";: GOTO 30
50 PRINT " NOT MAKE";: GOTO 90
60 PRINT "THIS IS";:GOTO 40
70 PRINT:PRINT " DO";:GOTO 50
80 PRINT " PROGRAM.";:GOTO 100
90 PRINT " THIS KIND OF A";:GOTO 80
100 END
```

IF...THEN...ELSE
IF...GOTO...ELSE

FUNCTION: These instructions are used to evaluate a logical expression and then perform a conditional process.

FORMAT:

IF <expression> [THEN <then clause> / GOTO <goto clause>]
{ELSE <else clause>}

where:

<expression> = [<arithmetic expression> / <logical expression> / <relational expression>]

<then clause> = [<statement> / <multiple statement> / <line number>]

<goto clause> = <line number>

<else clause> = [<statement> / <line number>]

SAMPLE STATEMENTS:

```
IF A$="Y" THEN BEEP ELSE 120
IF A+B=C AND A>E GOTO 200 ELSE PRINT A;B
```

DESCRIPTION:

The IF...THEN...ELSE/IF...GOTO...ELSE instructions control the program execution based on conditions established by the evaluation of the <expression>. If the evaluation of the <expression> is non-zero (true) the <then clause> or <goto clause> is processed. If the evaluation is zero (false) the <else clause> is processed.

When the ELSE option is omitted, and the evaluation of the <expression> is zero (false), the next line following the IF statement is processed.

Multiple nested IF instructions are allowed. When nesting is used the ELSE option will be matched with the most recent unmatched IF instruction.

The <then clause> can be made up of multiple statements, separated by a colon(:).

The complexity of a multiple statement is limited to the maximum length line, which is 255 characters long.

For more details on the evaluation of a <logical expression>, <relational expression> or <arithmetic expression> see Chapter 3

NOTE: Tabs are not considered in matching IF, THEN, GOTO or ELSE clauses; they are used only as programming aids in the structure of the code.

SAMPLE PROGRAM:

```
10 M=10000:CLS
20 PRINT " YOU HAVE $";M;"."
30 PRINT "$";M;".";"HOW MUCH DO YOU WANT
 TO BET ON THIS DIE?":INPUT K
40 K=INT(K):PRINT
50 REM ** This is the  nesting of the ty
pe of IF statement of line 70
60 REM ** when the input is not the righ
t input ...
70 IF K>M THEN PRINT "IMPOSSIBLE WITH ON
LY';M:BEEP:GOTO 30 ELSE IF K<0 THEN PRIN
T 'SNEAKY!":BEEP:GOTO 30 ELSE IF K>M/2 T
HEN PRINT "GENEROUS!" ELSE IF K<M/100 TH
EN PRINT"CHEAPSKATE!"
80 INPUT " NOW WHAT DO YOU THINK WILL CO
ME UP ON THE DIE(1-6)";N
90 N=INT(N):PRINT
100 IF N<1 OR N>6 THEN PRINT"IMPOSSIBLE
WITH AN ORDINARY DIE.":BEEP:GOTO 80
110 SOUND 3000,20:R=INT(RND(1)*6)+1
120 PRINT:PRINT "SO,";R;"SPOT(S) CAME UP
 ON THE DIE.":PRINT
130 IF N=R THEN SOUND 4000, 10:M=M+K*6:P
RINT"YOU WERE SUCCESSFUL!" ELSE PRINT "Y
OU LOST THIS TIME!":SOUND 16000,10:M=M-K
140 IF M<1 THEN PRINT "YOU'RE BANKRUPT N
OW!" ELSE IF M>1E+06 THEN PRINT "YOU ARE
 A MILLIONAIRE!" ELSE 30
150 END
```

IMP

FUNCTION: This logical operator is used to test multiple relations.

FORMAT: <operand 1> IMP <operand 2>

SAMPLE STATEMENT:

```
PRINT 2 IMP 2
```

DESCRIPTION: The logical operation IMP (Implication) performs tests on multiple relations, Boolean operations, and bit manipulation. It returns either a non-zero (false) value or a non-zero (true) value.

The operation returns zero (false) whenever <operand 1> is true and <operand 2> is false. Otherwise it returns a DELETE zero (true) value.

The following table indicates the evaluation process:

-1 IMP -1 → -1 (TRUE IMP TRUE→TRUE

-1 IMP 0 → 0 (TRUE IMP FALSE→FALSE)

0 IMP -1 → -1 (FALSE IMP TRUE→TRUE)

0 IMP 0 → -1 (FALSE IMP FALSE→TRUE)

For more details on logical operations see Chapter 3.

NOTE: IMP performs the same way as NOT (<operand 1>) OR (<operand 2>). A IMP B is the same as NOT (A) OR B.

Logical operators convert their operands to sixteen bits binary integers. Therefore, <operand 1> and <operand 2> must be within the range of 32768 to +32767. If not, an overflow error will occur, and the error message "?OV Error" will be dispayed.

SEE ALSO: AND, EQV, NOT, OR, XOR, and Chapter 3.

EXAMPLE:

Integer	Binary bits
23280	0101 1010 1111 0000
11853	0010 1110 0100 1101

After you input the statement PRINT 23280 IMP 11853, the integer -20657 appears, whose binary code is 1010 1111 0100 1111. BY looking at the table in the DESCRIPTION section, notice that the computation is correct.

INKEY$

FUNCTION: INKEY$ function is used to check if a character has been entered through the keyboard.

FORMAT: INKEY$

SAMPLE STATEMENT: `A$=INKEY$`

DESCRIPTION: INKEY$ returns a null string if the keyboard buffer is empty. When the keyboard buffer contains any characters, the first character in the buffer is returned. Any key that is not included in the Character Codes Table will be ignored.

SEE ALSO: Appendix A.3 Character Codes.

SAMPLE PROGRAM:

```
10 SCREEN 0,0:CLS:X=20:Y=5
20 PRINT " TRY TO MOVE THE CURSOR IN DIF
FERENT DIRECTIONS"
30 PRINT " U=UP, D=DOWN, R=RIGHT, L=LEFT
"
40 PRINT " HIT ANY OF THE ABOVE KEYS"
50 GOTO 120
60 A$=INKEY$:IF A$="" THEN 60
70 LOCATE X,Y:PRINT " ";
80 IF A$="U" AND Y>0 THEN Y=Y-1
90 IF A$="D" AND Y<7 THEN Y=Y+1
100 IF A$="R" AND X<39 THEN X=X+1
110 IF A$="L" AND X>0 THEN X=X-1
120 LOCATE X,Y:PRINT "";
130 GOTO 60
140 END
```

INP

FUNCTION: This function obtains a value from an input port.

FORMAT: INP (<port number>)

SAMPLE STATEMENT: `A=INP(&5)`

DESCRIPTION: The INP (Input from a Port) function reads a byte from the input port specified by the <port number>, and it returns that byte as the function value.

The <port number> must be an integer ranging from 0 to 255.

SEE ALSO: OUT.

INPUT

FUNCTION: The INPUT instruction allows data to be entered through the keyboard during program execution.

FORMAT: INPUT {"<prompt statement>";}<variable 1>{,<variable 2>}...

SAMPLE STATEMENT:

```
INPUT "NAME,NO.";N$,A$
```

DESCRIPTION The INPUT instruction is used to display a prompt message and then to accept one or more fields of data input from the keyboard.

When the INPUT instruction is executed, the <prompt statement> is displayed with a question mark following it, and the PC-8300 waits for data to be entered through the keyboard. If no prompt statement is used, the question mark alone will be displayed.

The input <variable(s)> are separated by commas, containing a mixture of variable types (integer, string, numeric, array), and may be as long as the line allows. Data elements entered are also separated by commas, and each data element corresponds to a variable in the INPUT instruction

If the number of data elements is less than the number of variables indicated, a double question mark (??) is displayed. This asks for more input until there is sufficient data for the variables.

On the other hand, if data entered is more than needed, program execution continues with the next statement following the INPUT instruction, disregarding the extra data. The message "?Extra ignored" is then displayed.

The type of data input should match the corresponding variable type. The screen displays "?Redo from start" if, for example a character string is input to a numeric variable. Data must then be input again, starting from the first variable.

SAMPLE PROGRAM:

```
10 CLS:INPUT "DESIGNATE A PASSWORD";PW$
20 WL=LEN(PW$)
30 REM PROGRAM STARTS FROM HERE
40 CLS:PRINT "ENTER PASSWORD:";
50 N$=INPUT$(WL)
60 IF N$=PW$ THEN PRINT "WELCOME USER!":
SOUND 3000,20:GOTO 10
70 LOCATE 0,3:PRINT "INVALID PASSWORD!"
80 PRINT "PLEASE TRY AGAIN"
90 SOUND 5000,8:SOUND 1000,8
100 CLS:GOTO 40
110 END
```

INPUT#

FUNCTION: This instruction is used to read data from an opened input file into variable(s) contained in the instruction.

FORMAT: INPUT #<file number>,<variable 1> [<variable 2>]...

SAMPLE STATEMENTS:

```
INPUT #1,A
INPUT #1,B,C$
```

DESCRIPTION: This instruction inputs data from a designated file (in RAM, cassette tape, etc.) and functions similar to the INPUT instruction except that a question mark (?) is not displayed.

The contents of the specified data file (file type ".DO") are read into the variables in the INPUT# instruction. The <file number> is the number designated in the OPEN instruction. The file should be opened for input.

The <variable(s)> are assigned from left to right, starting from the beginning of the input file. The number of <variable(s)> in the INPUT# instruction is the number of data elements used each time the instruction is executed. Each time the INPUT# instruction of the same file number is executed, it starts reading in data from where it terminated previously.

Data in the input file should be the appropriate type for the corresponding variable. The message "?EF Error"(End of File) will be displayed when an INPUT# instruction is reached and insufficent data is available. The EOF function is used to test for end of file condition before an INPUT# statement is executed.

SEE ALSO: PRINT#, INPUT, LINE INPUT#, and EOF.

INSTR

FUNCTION: This function searches for a character string within a string and returns its position.

FORMAT: INSTR [<numeric expression>,]<character string 1>, <character string 2>

SAMPLE STATEMENT:

```
PRINT INSTR(6,"THIS IS A TEST","TEST")
```

DESCRIPTION: INSTR locates a substring in a string and returns its position. <Character string 1> is the original string which is searched through to find a match for <character string 2> substring.

The <numeric expression> is designated by an integer, that specifics the position in <character string 1>, where the search begins.

If the <numeric expression> s omitted the searching begins at postion 1. The INSTR function returns the position where the match occurred. It returns zero if <character string 1> does not contain <character string 2> (no match).

If <character string 2> contains more than one character and a perfect match is made, the INSTR function returns only the postion of the first character in <character string 1> where the match begins.

When a null string (empty string) is designated for <character string 2>:

1. f the <numeric expression> is omitted then "-" is returned.

2. f <numeric expression> is less than or equal to the ength of <character string 1> then the <numeric expression> is returned.

or else 0 is returned if <numeric expression is larger than the length of <character string 1>.

NOTE: The <numeric expression> must be an integer from 1 to 255. f not, an "FC Error" (Illegal Function Call) message is displayed on the screen. When a number is read, ust its integer portion is used.

The length of <character string 2> must be less than or equal to (< =)<character string 1>; or else a zero will be returned.

INT

FUNCTION: This function rounds numbers to their integer value.

FORMAT: INT (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT INT(9.9)
PRINT INT(-9.9)
```

DESCRIPTION: INT rounds the <numeric expression> to its integer (whole number) value. If the <numeric expression> is positive, INT truncates it (drops its decimal part).

If the <numeric expression> is negative, INT returns the next smallest whole number. For example:

```
INT(-3.1)=-4
INT(-3.9)=-4
```

NOTE: The value of a negative that is returned is always less than or equal to (<=) <numeric expression>.

SEE ALSO: FIX and CINT.

SAMPLE PROGRAM:

```
10 PRINT "      I      FIX      INT"
20 FOR I=-2 TO 2 STEP .5
30 PRINT USING "###.##    #####    #####";
I,FIX(I),INT(I)
40 NEXT
50 END
```

KEY

FUNCTION: This function is used to assign functions to the programmable function keys.

FORMAT: KEY <key number>,"<character string>"

SAMPLE STATEMENT:

```
KEY1,"LOAD"
```

DESCRIPTION: Up to ten programmmable functions can be defined by using the five function keys (five accessible by pressing the five function keys labelled [f.1] to [f.5] on the keyboard, with another five being accessible by holding [SHIFT] while pressing these five function keys. The function keys are numbered from [f.1] to [f.5], but the additional five functons 6 to 10 are accessible by the use of [SHIFT] as explained above.) Each function key can be assigned with a character string or a control statement of up to 15 characters. Characters that cannot be input from the keyboard are entered by using the plus sign "+" and the CHR$ function.

SEE ALSO: See the Appendix A.3 Character Codes for use with the CHR$ function.

SAMPLE PROGRAM:

```
10 A$(0)=""
20 A$(1)="LOAD "+CHR$(34)
30 A$(2)="SAVE "+CHR$(34)
40 A$(3)="FILES "+CHR$(13)
50 A$(4)="LIST "
60 A$(5)="RUN "+CHR$(13)
70 FOR I=1 TO 59
80 KEY (I MOD 5)+1,A$(I MOD 6)
90 NEXT
100 END
```

KILL

FUNCTION: This instruction is used to erase a designated file.

FORMAT: KILL "<file name.file type>"

SAMPLE STATEMENT:

```
KILL "SAMPLE.BA"
```

DESCRIPTION: The KILL instruction deletes a specific file designated by a file name with or without a device name. The file to be deleted must be closed. Any opened file is indicated by an astersk (*) when the FILES instruction is executed. Only one file may be deleted with each KILL instruction.

The file name must always include its file type extension (".BA", ".DO", or ".CO") when the KILL instruction is executed. The PC-8300 returns to the Direct Mode after the execution.

SEE ALSO: LOAD, SAVE and Chapter 5, Files.

LEFT$

FUNCTION: This function is used to designate a specific number of characters from a string, starting from the left-most position of a string.

FORMAT: LEFT$ (<character string>,<numeric expression>)

SAMPLE STATEMENT:

```
B$=LEFT$(A$,4)
```

DESCRIPTION: A <character string> can be a string constant or a string variable. The value of the <numeric expression> must be within the range of 0 to 255, which specifies the number of characters to be read, beginning from the left-most character.

The full <character string> is returned when the <numeric expression> is greater than or equal to the total number of characters in the <character string>. LEFT$ returns a null string when the <numeric expression> is 0.

SEE ALSO: RIGHT$ and MID$.

SAMPLE PROGRAM:

```
10 A$="++++++++++++++++++++++++++++++++++
++++++"
20 PRINT " INPUT DATA FOR EACH LINE."
30 FOR I=1 TO 5:PRINT I;
40 INPUT "INPUT THE BAR LENGTH (0 TO 39)
";A(I)
50 IF A(I)<0 OR A(I)>39 THEN BEEP:PRINT
 "ILLEGAL NUMBER";:PRINT I:GOTO 40
60 PRINT LEFT$(A$,A(I))
70 NEXT I
80 END
```

LEN

FUNCTION: This function returns the number of characters that are contained in a string.

FORMAT: LEN(<character string> | <character variable>)

SAMPLE STATEMENT:

```
PRINT LEN("123456789")
```

DESCRIPTION: The LEN (LENgth) function returns the length of a <character string> or <character variable>. It counts all characters including the ones that cannot be printed (control codes 1-31).

NOTE: To determine the length of a number, double quotation marks must be placed around it.

SAMPLE PROGRAM:

```
10 INPUT "INPUT ANY COMBINATION OF LESS
THAN 36   CHARACTERS";N$
20 CLS:L=LEN(N$):GOSUB 50
30 PRINT "+ ";N$" +"
40 GOSUB 50: END
50 FOR I=1 TO L+4
60 PRINT "+";:NEXT
70 PRINT:RETURN
80 END
```

LET

FUNCTION: This instruction is used to assign values to variable names.

FORMAT: {LET} <variable name> = <value>

SAMPLE STATEMENT:

```
LET A=10+5
```

DESCRIPTION: The BASIC Reserved Word (keyword) LET is optional, so the statement: `LET A=10+5` could also be entered as `A=10+5`. The <variable name> is assigned the evaluated <value> which may be a number, a string, an equation, or a function.

SAMPLE PROGRAM:

```
10 BE=26:IT=810
20 LET IT=BE
30 PRINT IT,BE
40 END
```

LINE INPUT

FUNCTION: The LINE INPUT instruction is used to allow the input of an entire line of data

FORMAT: LINE INPUT {"<prompt string>";} <string variable>

SAMPLE STATEMENT:

```
LINE INPUT "WHAT?";A$
```

DESCRIPTION: A <prompt string> is a sentence that displays a query for a specific input. A maximum of 255 characters, including delimiters (quotation marks, comma, etc.), can be entered and assigned to a <string variable>. All input from the keyboard (after the prompt string) and up to the carriage return, is substituted for the <string variable>.

Any punctuation marks and symbols can be input in the <string variable>. [CTRL] [C] or [STOP] can be pressed to interrupt the LINE INPUT instruction. This will stop program execution and return the PC-8300 to the Direct Mode. The LINE INPUT instruction can be continued by executing the CONT instruction.

SEE ALSO: INPUT.

SAMPLE PROGRAM:

```
10 PRINT "INPUT (ANYTHING UP TO 255 CHAR
ACTERS IN ALL,";
20 PRINT " INCLUDING A COMMA OR QUOTATIO
N MARKS)"
30 LINE INPUT A$
40 FOR I=1 TO LEN (A$)
50 PRINT MID$(A$,I,1);
60 FOR T=0 TO 200:NEXT
70 NEXT I
80 END
```

LIST/LLIST

FUNCTION: These instructions are used to list either a portion of a program or the entire program which is currently in the memory.

FORMAT: [LIST / LLIST] { <line number 1> } [– <line number 2> }

SAMPLE STATEMENTS:

```
LIST 70-120
LLIST 70-120
```

DESCRIPTION: The LIST instruction is used to list a program on the screen. the LLIST instruction outputs the listing to the printer. The PC-8300 returns to the Direct Mode after the LIST or LLIST instruction is executed.

When both <line number>s are omitted, the entire program is listed. [STOP] may be pressed at any time to interrupt listing on the screen. [SHIFT] [STOP] interrupts listing to the printer.

If only <line number 1> is designated only that specific line is listed (if it exists). If <line number 1> and a hyphen (-) are specified, all lines starting from <line number 1> are listed. When a hyphen is followed by a designated <line number 2>, the listing starts from the beginning and continues up to and including <line number 2>. When a hyphen is used between both <line number 1> and <line number 2>, all lines included between these <line number>s, inclusive will be listed. The <line number 2> must be greater than or equal to <line number 1>.

The LLIST instruction is identical to the LIST instruction with the exception that it outputs the list to a printer as "hard copy" printed on paper.

LOAD

FUNCTION: This instruction is used to load a program file into memory

FORMAT: LOAD "{<external device name>:}<file name>"[,R]

SAMPLE STATEMENT:
```
LOAD "CAS:SAMPLE.BA",R
```

DESCRIPTION: This instruction loads the program specified by <file name> and optional <external device name> into the memory When executed, the LOAD instruction closes all open files and deletes variables.

RAM is selected if the <external device name> is omitted, but <file name> must be specified. The PC-8300 loads from cassette tape if "CAS:" is designated for <external device name>. If file name is omitted, the first program file that it detects on the cassette tape is loaded. SHIFT STOP interrupts the execution of the LOAD 'CAS:" command.

The intended device is the RS-232C serial interface when 'COM:' is designated for <external device name>. Data transmission format can be indicated but <file name> cannot be usec. (Please refer to the OPEN "COM:" instruction for details on this specific application.)

The file must be a ".BA" or ".CO" file. File type extension can be omitted during loading. If the "R" (Run) option is specified the program is run or executed immediately after loading.

The program currently in the memory is preserved until the specified file is found and the program loading has begun. The PC-8300 returns to the Direct Mode when the loading process has been completed.

NOTE: A NEW instruction should be executed before the actual execution of a LOAD instruction occurs, so that all existing variables and programs can be cleared. Otherwise you may end up with a garbage mixture of part of your new program mixed with parts of the program that was already in the memory.

SEE ALSO: BLOAD, CLOAD, SAVE, and Chapter 5, Files

LOCATE

FUNCTION: This instruction designates the location of the screen cursor.

FORMAT: LOCATE <horizontal coordinate>,<vertical coordinate>

SAMPLE STATEMENT:
```
LOCATE 20,5
```

DESCRIPTION: This instruction moves the cursor to a designated location on the display screen. The range of the <horizontal coordinate> is 0 through 39, and for the <vertical coordinate> the range is 0 through 7. The home position is at the coordinates (0,0).

Any number greater than 39 will be set as 39 for the <horizontal coordinate>, while any number larger than 7 will be set as 7 for the <vertical coordinate> (or 6 when the Function Keys are displayed on the bottom line of the screen).

NOTE: A LOCATE instruction designates character coordinates and has absolutely no connection with the dot matrix structure of the screen itself.

SAMPLE PROGRAM:

```
10 SCREEN 0,0:CLS
20 LOCATE 10,7:PRINT "X=";X;
30 LOCATE 20,7:PRINT "Y=";Y;
40 X=INT(RND(1)*39):Y=INT(RND(1)*7)
50 LOCATE X,Y:SOUND 200,5:PRINT "HOP";
60 FOR I=0 TO 300:NEXT
70 LOCATE X,Y:PRINT "   ";
80 GOTO 20
90 END
```

LOG

FUNCTION: This function returns the natural logarithm of a number.

FORMAT: LOG (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT LOG(2.7182818)
```

DESCRIPTION: A LOG function is useful in trigonometric applications, and it returns the natural logarithm of a number based on "e" (exponent).

The <numeric expression> must be greater than zero. If it is zero or less an "?FC Error" (Illegal Function Call) message is displayed on the screen.

SAMPLE PROGRAM:

```
10 READ I
20 IF I=999 THEN END
30 X=LOG(I)
40 PRINT I,X
50 GOTO 10
60 DATA 34,1,06,44,8976,146,35,677,999
70 END
```

LPOS

FUNCTION: This function determines the current column position of the printer.

FORMAT: LPOS (<rumeric expression>)

SAMPLE STATEMENT:
```
LPRINT "ABCDE"; LPOS(0)
```

DESCRIPTION: A LPOS function determines the current column position of the printer head within the buffer. It keeps track of the number of characters printed until a carriage return appears, which resets it to zero.

The value of the <expression> is only used as a dummy expression, used for the value that is returned by the LPOS function.

SEE ALSO: POS.

MAXFILES

FUNCTION: This instruction establishes the maximum number of files that can be opened.

FORMAT: MAXFILES = <number of file(s)>

SAMPLE STATEMENT:
```
MAXFILES=3
```

DESCRIPTION: The maximum number of files that can be opened s set to 1 when a Cold Start is performed. The maximum number of files that can be opened at one time is designated by a MAXFILES instruction. The range of <number of file(s)> is from 0 through 15. Once this value has been designated, it will be protected until it is redesignated, or until the next Cold Start.

SEE ALSO: OPEN, CLOSE, and Chapter 5 Files.

MENU

FUNCTION: This instruction returns the MENU.

FORMAT: MENU

SAMPLE STATEMENT:
```
MENU
```

DESCRIPTION: The MENU instruction clears all variables and returns the screen to the MENU mode. Files in access mode (indicated by an asterisk when the FILES instruction is executed), are closed when the MENU instruction is executed. The program is maintained in the BASIC area and execution is possible by entering the BASIC mode.

NOTE: MENU does not use any parameters.

MERGE

FUNCTION: This instruction is used to merge two programs together.

FORMAT: MERGE "[<external device name>:]<file name>"

SAMPLE STATEMENT:
```
MERGE "CAS:DEMO.DO"
```

DESCRIPTION: A program file within the RAM or from an external device can be merged with a program currently in the memory. The PC-8300 returns to the Direct Mode after the MERGE instruction has been executed.

RAM is selected when the <external device name> is not specified. When "CAS:" (cassette tape) is designated for external device and the <file name> is omitted, the first program detected is used in the merging process. When "COM:" (the RS-232C port) is designated the file name cannot be used but the designation of data transmission format is possible. (Refer to the OPEN instruction for more details.) The MERGE instruction will close all files after it has been completed.

In all cases, the designated program must have been saved in ASCII code (must be a ".DO" file). If it is not, an error occurs.

NOTE: Use with caution, because if the two programs have identical line numbers, you will end up with a meaningless mixture of two programs; before running MERGE, the line numbers of the two programs to be merged should be renumbered using the RETURN instruction, so that they will end up in the correct position relative to each other in the final merged program.

SEE ALSO: SAVE and RENUM.

MID$

FUNCTION: This function returns a specified number of characters from a desired position within a string.

FORMAT: MID$(<character string>,<numeric expression 1>[,<numeric expression 2>])

SAMPLE STATEMENT:

```
PRINT MID$("ABCD",2,2)
```

DESCRIPTION: The MID$ (MIDdle) function returns a substring of a specfied length from a desired position within the <character string>.

The <numeric expression 1> specifes the position within the <character string>, while <numeric expression 2> determines the length of the substring.

When <numeric expression 2> is omitted, or when the number of characters to the right of the <numeric expresson 1> position within the <character string> is less than <numeric expression 2>, all characters to the right of the <numeric expression 1> position are returned.

If <numeric expression 1> is greater than the length of the <character string> a null string is returned.

NOTE: <Numeric expression 2> must be an integer from 0 to 255, while <numeric expression 1> must be an integer from 1 to 255. If not, an "?FC Error" (Illegal Function Call) message is displayed.

SAMPLE PROGRAM:

```
10 A$="JANUARY XX, 19"
20 D$="1234567890"
30 P$=MID$(A$,1,8)+MID$(D$,1,1)+MID$(D$,
10,1)+MID$(A$,11)+MID$(D$,9,2)
40 PRINT P$
50 END
```

MOD

FUNCTION: This operator provides the remainder of an arithmetic division.

FORMAT: <numeric expression 1> MOD <numeric expression 2>

SAMPLE STATEMENT:

```
PRINT A MOD 7
```

DESCRIPTION: Values for both numeric expressions can be positive integers that are less than 32767.

When a negative value is used for <numeric expression 2>, it will be processed as an absolute value. If a negative value is specified for <numeric expression 1>, a negative value is returned as the result.

Also note, a zero cannot be used in <numeric expression 2>. Any decimal fraction included is rounded to the decimal point.

SAMPLE PROGRAM:

```
10 SCREEN 0,0:CLS
20 LOCATE 5,0:BEEP:INPUT" A NUMBER"; A:A
=INT(A)
30 IF A<32768! THEN 50
40 PRINT"IF IS TOO LARGE.":FOR I=0 TO 10
00:NEXT:GOTO 10
50 CLS:LOCATE 6,2:PRINT "THE DECIMAL NUM
BER";A;" WILL BE "
60 LOCATE 6,4:PRINT "IN BINARY"
70 N=0
80 LOCATE 30-N*2,6
90 PRINT A MOD 2:A=INT(A/2):N=N+1
100 IF A<>0 THEN 80
110 GOTO 20
120 END
```

MOTOR

FUNCTION: This instruction turns the motor that drives the cassette recorder on and off.

FORMAT: MOTOR <switch>

SAMPLE STATEMENT:

```
MOTOR 0
```

DESCRIPTION: The cassette recorder motor is turned off when the <switch> value is set to 0. Any numeric value ranging from 1 to 255 turns the motor on. An error occurs if a value greater than 255 is designated to turn the motor on.

SAMPLE PROGRAM:

```
10 MOTOR 0
20 PRINT "SELECT CASSETTE TAPE WITH MUSI
C THAT YOU LIKE"
30 PRINT "PLUG ONE END OF THE CABLE INTO
 THE PC-E300 AND ";
40 PRINT "INSERT THE BLACK PLUG INTO THE
 REMOTE CONNECTOR  ";
50 PRINT "SET RECORDER TO ON"
60 PRINT "PRESS ANY KEY TO START"
70 IF INKEY$="" THEN 70
80 MOTOR 1
90 PRINT "PRESS ANY KEY TO STOP"
100 IF INKEY$="" THEN 100
110 MOTOR 0:GOTO 60
120 END
```

NAME

FUNCTION: This instruction is used to reNAME files in the RAM.

FORMAT: NAME "<old file name>" AS "<new file name>

SAMPLE STATEMENT:

```
NAME "OLD.BA" AS "NEW.BA"
```

DESCRIPTION: The NAME (rename) instruction renames the RAM file <old file name> as <new file name>. The designated file name must include the file-type extension for both the old and the new file names, and both of these must be identical.

An error message appears on the screen if one of the following occurs:

1. A non-existent file name is designated as an <old filename>.
2. A file name which is already currently in use is designated as a <new filename>.
3. File-types for the two files are different.

SEE ALSO: Chapter 5 Files.

NEW

FUNCTION: This instruction erases any program or data currently in the BASIC memory area and clears all variables. It is normally used before writing a new program.

FORMAT: NEW

SAMPLE STATEMENT:

```
NEW
```

DESCRIPTION: The NEW instruction is used in the Direct Mode prior to the input of a new program. When executed, it closes all opened files. Furthermore, a file which is in access mode (indicated by an asterisk when the FILES instruction is executed) will be terminated.

This instruction does not use any parameter and it returns control to the Direct Mode after execution is completed.

SAMPLE PROGRAM:

```
10 REM This program will self-destruct w
hen you run it.
20 PRINT"YOU HAVE DESTROYED THE PROGRAM!
"
30 BEEP:BEEP
40 NEW
50 END
```

NOT

FUNCTION: This logical operator is used to test multiple relations, bit manipulation, and Boolean operations.

FORMAT: NOT <operand>

SAMPLE STATEMENT:
```
PRINT NOT 5
```

DESCRIPTION: The logical operator NOT converts its <operand> to a sixteen bit binary integer, and then it inverts (negates) each bit of the <operand>. It returns -1 (true) if the bit is 0 (false), and vice versa.

The following table shows the negated calculations:

NOT -1 → 0 (NOT TRUE → FALSE)

NOT 0 → -1 (NOT FALSE → TRUE)

NOTE: Because of the <operand> conversion to sixteen bit binary, the <operand> must range from −32768 to +34767. Otherwise, an Overflow message "?OV Error" is displayed.

SEE ALSO: AND, EQV, IMP, OR, XOR, and Chapter 3.

EXAMPLE:

Integer	Binary bits
153	0000 0000 1001 1001
-154	1111 1111 0110 0110

To negate, just replace 0 with 1, and vice versa. If you input the statement PRINT NOT 153, the PC-8300 responds −154, whose binary is 1111 1111 0110 0110, which is the correct result, according to the table in the DESCRIPTION section.

ON...GOTO/ON...GOSUB

FUNCTION: These instructions transfer control (branch) to one of several specified lines/subroutines based on the evaluation of the statement.

FORMAT: ON <numeric variable> {GOTO / GOSUB} <line number>, <line number list>

SAMPLE STATEMENT:
```
ON A GOTO 100, 140, 200, 400
```

DESCRIPTION: The ON ... GOTO/ON ... GOSUB instructions branch to a specific <line number> based on the evaluation of the <numeric variable>.

After the <numeric variable> is evaluated it is converted to an integer; if this is 1, then the first <line number> is selected; if it is 2, then the second <line number> is selected, etc.

An "?FC Error in line" occurs if the value of the <numeric variable> is negative. But if it is zero or greater than the number of <line number> then control continues on to the next logical line (following the ON...GOTO/GOSUB instruction).

The <line number>s following the GOTO or GOSUB must be separated by commas, or else an syntax error message "?SN Error" is displayed on the screen. There may be any number of <line numbers> in a list, as long as the line does not go over the maximum line length (up to 255 characters per line).

When ON ... GOSUB is used and control is transferred to the subroutine, a RETURN instruction is needed as the last statement in the subroutine, in order to return control to the main program when the subroutine execution is completed. Control returns to the next statement following the ON ... GOSUB instrucion in the main program (containing the GOSUB command).

NOTE: These statements save time and program lines when they are used in place of the IF ... THEN statement. For example:

```
IF L=1 THEN GOSUB 150
IF L=2 THEN GOSUB 80     } ON L GOSUB 150, 80, 200 . . .
IF L=3 THEN GOSUB 200
.
.
```

SEE ALSO: ON ERROR, GOTO, GOSUB and RETURN.

SAMPLE PROGRAM:

```
10 INPUT "ENTER A NUMBER FROM 0 TO 5";A
20 ON (A AND 1) +1 GOSUB 120,130
30 PRINT "YOUR NUMBER IS ";
40 ON A+1 GOTO 60,70,80,90,100,110
50 PRINT "OUT OF RANGE.":GOTO 10
60 PRINT "ZERO":END
70 PRINT "ONE":END
80 PRINT "TWO":END
90 PRINT "THREE":END
100 PRINT "FOUR":END
110 PRINT "FIVE":END
120 PRINT A "IS AN EVEN NUMBER":RETURN
130 PRINT A "IS AN ODD NUMBER":RETURN
140 END
```

ON COM GOSUB

FUNCTION: This instruction establishes the initial line of the subroutine to branch to when an interruption occurs from the RS-232C communications port.

FORMAT: ON COM GOSUB <line number>

SAMPLE STATEMENT: `ON COM GOSUB 2000`

DESCRIPTION: This instruction designates <line number>, which branches control to the first line of a subroutine used to perform some communications process when an RS-232C interrupt occurs.

A return from the process routine is conducted by a RETURN instruction, just the same as for normal subroutines.

When <line number> is specified, the program is restarted from the specified line.

SEE ALSO: COM ON/OFF/STOP, OPEN and RETURN.

ON ERROR GOTO ~ RESUME

FUNCTION: The ON ERROR GOTO instruction is used to specify an error subroutine used for trappable errors.

FORMAT: ON ERROR GOTO [<line number> / <0>]

SAMPLE STATEMENTS:

```
ON ERROR GOTO 100
ON ERROR GOTO 0
```

DESCRIPTION: The ON ERROR GOTO ~ RESUME instruction creates an error handling routine, which takes control from N82-BASIC if an error is detected during program execution.

The ON ERROR GOTO instruction is used to instruct the PC-8300 that an error processing subroutine is in effect. When an error occurs, control will be transferred to the <line number> indicated, which should be the beginning of the error handling routine. If a line specified in <line number> does not exist, a "?UL Error" (Undefined Line number) message will be displayed.

The ON ERROR GOTO 0 command is used when an error trapping function is not possible, which signals BASIC to handle all errors. BASIC proceeds with normal system error handling by displaying error messages and stopping program execution. It is advisable to execute an ON ERROR GOTO 0 instruction for error processing routines so that any failure in the routines can be trapped.

SEE ALSO: RESUME and ERROR.

OPEN

FUNCTION: This instruction is used to open a file for input or output.

FORMAT: OPEN "{<external device name>:}<file name>"

FOR { INPUT / OUTPUT / APPEND } AS {#} <file number>

SAMPLE STATEMENT:

```
OPEN "SESAME" FOR OUTPUT AS #1
OPEN "CAS.SESAME" FOR OUTPUT AS #2
```

DESCRIPTION: The OPEN instruction opens a file specified by <file name> for use with the buffer number <file number>. A range from 1 through 15 can be designated for <file number>. A <file number> previously used to open a file cannot be subsequently used to open another (a second) file. Input and output of an opened file are conducted by subsequently specifying its file number.

Three different <modes> are used to specify the access methods to a file. "INPUT" assigns sequential input from a device or an existing file "OUTPUT" designates sequential output to a device or a file. "APPEND" specifies addition to a RAM file.

The PC-8300 opens a file from RAM if <external device name> is omitted, but in the case of a RAM file, the file name must be supplied. When device name is specified, "CAS:" is designated for a data recorder. File name may be omitted when using a data recorder with cassette tapes as the storage medium, and if so, the PC-8300 opens the first tape file it detects if it is in input. SHIFT STOP is pressed to interrupt the execution of an OPEN "CAS:" instrucion.

OPEN reserves the buffer space required for input/output and uses it only for the specified file while it is open.

Any file name designated in the output mode means that a new file is being created. If an existing file name is used for output, its content is erased when the file is open. Care should be exercised when selecting a file name for OPEN OUTPUT.

SEE ALSO: CLOSE, OPEN "COM", Chapter 5 Files.

SAMPLE PROGRAM:

```
20 OPEN "SESAME" FOR OUTPUT AS #1
30 PRINT #1,"OPEN SESAME!"
40 PRINT #1,"CLOSE SESAME!"
50 CLOSE
60 OPEN "SESAME" FOR INPUT AS #1
70 INPUT #1,A$:PRINT A$:SOUND 2000,20
80 INPUT #1,A$:PRINT A$:SOUND 5000,20
90 CLOSE
100 PRINT "THE SESAME FILE IS NOW ARRANG
ED."
110 PRINT "FILES":FILES
120 END
```

OPEN "COM"

FUNCTION: This instruction opens the RS-232C serial circuit.

FORMAT: OPEN "COM [<CPBSXS>]" FOR { INPUT / OUTPUT }

AS [#] <file number>

SAMPLE STATEMENT:

```
OPEN "COM:9N82XN" FOR INPUT AS #1
```

DESCRIPTION: This command establishes the RS-232C serial circuit data transmission format and opens it as a file. <Mode> and <file number> perform the same way as in the OPEN statement. However, appended output mode cannot be designated.

The designated parameter that follows the COM: requires six characters to establish a data transmission circuit format. The designations are as follows.

"COM:<CPBSXS>"

where CPBSXS stands for:

C Communications speed (BAUD RATE)

P Parity

B Word length

S Stop bit

X Control according to "X" parameter

S Control according to SHIFT IN/OUT sequence.

Each different character of the parameter is controlled by a different feature of the communications format.

The following are the values for each different feature of the communications format:

VALUE

Communication Speed (Bits per second)

1	:	75 bps
2	:	110 bps
3	:	300 bps
4	:	600 bps
5	:	1200 bps
6	:	2400 bps
7	:	4800 bps
8	:	9600 bps
9	:	19200 bps

Parity

N	:	No parity
E	:	Even parity
O	:	Odd parity
I	:	parity bit Ignored

Word length

6	:	6 word length bits
7	:	7 word length bits
8	:	8 word length bits

Stop bit

1	:	1 stop bit
2	:	2 stop bits

X : affects control
N : does Not affect control

The "X" parameter controls communications transmission by using CTRL S to start and CTRL Q to stop transmission.

Control according to SHIFT IN/OUT sequence

S : affects control
N : does Not affect control

If the value of <CPBSXS> is omitted, then the previously established value is in effect.

When the RS-232C circuit is used in BASIC, two separate files must be opened to send transmitted data. The OPEN instruction (at either end of the transmission) that was established last is used to set the data transmission format.

The CTRL S and CTRL Q functions can be transmitted although only the input/output of a file is opened.

NOTE: The RS-232C circuit cannot be used while the data recorder is in use.

SEE ALSO: OPEN, COM ON/OFF/STOP and the TELCOM Manual.

OR

FUNCTION: This logical operator is used to test multiple relations.

FORMAT: <operand 1> OR <operand 2>

SAMPLE STATEMENT:

```
IF A=5 OR B=5 THEN 200
```

DESCRIPTION: The logical operator OR performs tests on multiple relations, bit manipulation, and Boolean operations. It returns either a non-zero (true) or zero (false) value.

For the operation to return a non-zero (true) value, the condition of at least one <operand> has to be true, or else the operation returns zero (false).

The following truth table indicates the evaluation process:

-1 OR -1 → -1 (TRUE OR FALSE → TRUE)

-1 OR 0 → -1 (TRUE OR FALSE → TRUE)

0 OR -1 → -1 (FALSE OR TRUE → TRUE)

0 OR 0 → 0 (FALSE OR FALSE → FALSE)

NOTE Logica <operators> work by converting their <operands> to sixteen bit binary integers. Therefore, <operand 1> and <operand 2> must range from −32768 to +32767. If not, an "?OV Error" (Overflow) message will be displayed.

SEE ALSO: AND, EQV, IMP, NOT, XOR, and Chapter 3.

EXAMPLE:

Integer	Binary bits
23280	0101 1010 1111 0000
11853	0010 1110 0100 1101

After you input the statement PRINT 23280 OR 11853, the irteger 32509 appears, whose binary is 0111 1110 1111 1101. By looking at the table in DESCRIPTION, you will see that the computation is correct.

OUT

FUNCTION: This instruction sends data to a specific port.

FORMAT: OUT <port number>,<data>

SAMPLE STATEMENT:

```
OUT 1,32
```

DESCRIPTION: The OUT instruction sends data to a designated output port. The <port number> must be an integer rarging from 0 to 255, while <data> is the data that is output through the port.

NOTE: If the OUT instruction is not used correctly, BASIC might not operate normally.

PEEK

FUNCTION: This function loads the contents of a designated location ir the memory.

FORMAT: PEEK (<address>)

SAMPLE STATEMENT:
```
A = PEEK(61400)
```

DESCRIPTION: The PEEK unction returns the memory contents of a designated <address>. Any value from 0 through 65535 may be designated for <address>.

Any numbers (specified for <address>) that contain decimal fractions are rounded off to integers.

SEE ALSO: POKE.

POKE

FUNCTION: This instruction writes the specified data to a designated memory address.

FORMAT: POKE <address>, <data>

SAMPLE STATEMENT:
```
POKE 61400,201
```

DESCRIPTION: This instruction is used to write one byte (8 bits) of data into a designated location in the memory. The <address> is designated with 2 byte integers between 0 and 65535. The <data> is designated by one byte integers between 0 and 255. The POKE instruction is used in conjunction with the PEEK function to perform the inverse operation. It is used when the numeric values of a machine language subroutine are to be accessed.

NOTE: The POKE instruction changes the current contents of the specified address in memory. Therefore, it should only be used after checking the memory to ensure that data in the BASIC work area is not destroyed. It is quite easy to destroy programs and files if you do not adequately understand machine language. If the PC-8300 operates abnormally after the POKE instruction is used, the Reset Switch may be pressed to restore normal operaton.

SEE ALSO: PEEK, and Chapter 6 Machine Language Programming.

POS

FUNCTION: This function determines the current cursor horizontal position

FORMAT: POS (<expression>)

SAMPLE STATEMENT:

```
PRINT "123456";POS(0)
```

DESCRIPTION: The POS (POSition) function determines the current horizontal position on the screen.

The <expression> is only used for the value that is returned by the POS function. Therefore, it does not make any difference what value is used for the <expression>.

NOTE: Since there are 40 possible horizontal positions on the screen, the returned value is always between 0 through 39.

SEE ALSO: CSRLIN.

SAMPLE PROGRAM:

```
10 CLS
20 PRINT:PRINT"PC-8300";
30 PRINT POS(X)
40 LOCATE 2,2
50 PRINT POS(X)
60 LOCATE 4,4
70 PRINT POS(X)
80 END
```

POWER

FUNCTION: This instruction automatically turns off the electrical power of the PC-8300 immediately, or after the designated time perod.

FORMAT:

```
POWER [ <timer> ]
      [ OFF     ] ,RESUME
      [ CONT    ]
```

SAMPLE STATEMENTS:

```
POWER 200
POWER OFF
POWER CONT
```

DESCRIPTION: The designated value for <timer> can be from 10 through 255, at increments of approximately 6 seconds per unit. Keyboard input is not accepted once the designated <timer> is reached and the electrical power is automatically turned off. Once the value for the <timer> has been established, it remains at that value until it is reset or modified.

The electrical power of the PC-8300 is promptly turned off when a POWER OFF instruction is executed. It returns to the MENU mode when the power switch is turned on again. If the optional ",RESUME" is also appended, when the PC-8300 is turned on again, the configuration in which it was automatically turned off is reinstated. The contents of the variables are also maintained.

After a POWER CONT (Continuous Power) instruction is executed, the automatic power shut off function is deactivated until the POWER <timer> instruction is input again.

It is not recommended to execute the POWER CONT instruction unless an AC Adapter is used, otherwise the batteries may be severely drained.

In the sample statement, the POWER 200 instruction will cause the PC-8300 to shut off in 20 minutes, if nothing is input or output during that time. The calculation of time for the sample statement is as follows:

200 units x 6 seconds (per unit) = 1,200 seconds or 20 minutes.

PRESET

FUNCTION: This instruction resets the desired dot pattern on the LCD screen.

FORMAT: PRESET (<horizontal coordinate>, <vertical coordinate> [,<function code>])

SAMPLE STATEMENT:

```
PRESET (80,32)
```

DESCRIPTION: The PRESET instruction resets dots on the screen at the designated coordinates. The <vertical> and <horizontal> coordinates must range from 0 to 255, or else an error occurs.

The system for the dot coordinates for the LCD display is 239 x 63. If the <horizontal coordinate> is greater than 239, it is converted to 63.

When the <function code> is an even number, the PRESET instruction reverses, and operates exactly the same way as the PSET instruction.

If the <function code> is an odd number the command operates the same way as when it is omitted.

SEE ALSO: PSET.

SAMPLE PROGRAM:

```
10 PRINT"      THIS SENTENCES WILL"
20 PRINT
30 PRINT"      DISAPPEAR SLOWLY"
40 PRINT
50 PRINT"      BY THE EFFECTS OF"
60 PRINT
70 PRINT"      PRESET.";
80 FOR Y=0 TO 55:FOR X=30 TO 160
90 PRESET(X,Y):NEXT X,Y
100 END
```

PRINT/LPRINT

FUNCTION: These instructions output information to the display screen or to a printer.

FORMAT: [PRINT / LPRINT] [<expression>...]

SAMPLE STATEMENTS:

```
PRINT "ABC"
LPRINT "PC-8300"
```

DESCRIPTION: The PRINT instruction outputs the value of a designated expression or a string to the display screen, while the LPRINT instruction outputs to a printer.

A PRINT instruction by itself (without expression), will cause a line feed carriage return to be executed. If a comma is used to separate each individual item, it causes these items to be printed in columns 14 spaces apart called print zones.

A question mark (?) can be used as the abbreviated form of the PRINT instruction.

NOTE: A comma (,), semicolon (;), or blank space can be omitted, except for punctuation within a string (where a variable is enclosed by quotation marks).

Single precision numbers can be displayed without loss of precision in six columns (excluding exponential format). Double precision numbers can be displayed without loss of precision (excluding exponential format) in sixteen columns.

SAMPLE PROGRAM:

```
10 PRINT "IF YOU DO NOT WANT AN INDENTAT
ION,";
20 PRINT "THEN",
30 PRINT "USE A SEMICOLON."
40 END
```

PRINT USING/LPRINT USING

FUNCTION: These instructions output formatted data to the display screen or to a printer.

FORMAT: [PRINT / LPRINT] USING <formatting string>; <numeric expression>
{ [, / ;] <numeric expression list> }

SAMPLE STATEMENT:

```
PRINT USING "## ####";2.3;4567
```

DESCRIPTION: The PRINT USING instruction outputs numeric data in a designated format. It formats numbers in several ways, making it easier to read and interpret the output on the screen. LPRINT USING outputs data to a printer in the same manner.

PRINT USING/LPRINT USING allows you to specify

- Number of significant digits.
- Location of decimal point.
- Exponental format.
- Inclusion of symbols (asterisk, dollar sign, comma, leading zeros).
- Indicate negative values.

The output of a <numeric expression> field will always be the same length as the length of the <formatting string>, unless there is insufficient space and an error occurs.

If the field specified by the <formatting string> is not large enough for the <numeric expression>, the number that is printed includes a '%" symbol at the beginning.

The <formatting string> may include the following:

1. The "#' <symbol> pound sign reserves space for one digit and indicates that leading zeros are to be suppressed. For example:

   ```
   PRINT USING "###";3
   PRINT USING "###";3333
   ```

   will resut in:

   ```
     3
   %3333
   ```

2. The ".", (decimal poirt) specifies the number of digits to the left and right of the decimal point. The digits to the left of " " will aways be printed, even if zeros are required.

Rounding will occur when the number of specified spaces to the right of "." is less than the <numeric expression>. Only one "." may be specified. A second "." will cause a ?SN Error. If a comma is used, or in the third sample below, it indicates the end of the old format field and the beginning of a new one. For example:

```
PRINT USING"###.##";2.5
PRINT USING"###.##";2.555
PRINT USING"###.#.#";2.34,45
```

will result in:

```
 2.50
 2.56
 2.3%45.0
```

3. The "," symbol (comma), used anywhere within the <formatting string>, after the first character and before the decmal point, punctuates the printed number with a comma "," appearing every third digit, starting from the decimal point and heading left. For example:

```
PRINT USING"#,###.###";2222.2
PRINT USING"#,####.##";123456
PRINT USING"#####.,#";1234.5
```

will result in:

```
2,222.200
%123,456.00
 1235.,
```

4. The "+" symbol (plus sign) used at the beginning or at the end of the <formatting string> specifies the sign (+ or −) of the <numeric expression>. For example:

```
PRINT USING"+##.##";2
PRINT USING"##.#+";34.5
PRINT USING"+##.##";-3
PRINT USING"###.#+";-34.5
PRINT USING"#,####.#+";12345.6
```

will result in:

```
 +2.00
34.5+
 -3.00
 34.5-
12,345.6+
```

5. The "−" symbol (minus sign), used only at the end of the <formatting string>, indicates the sign (+ or −) of the <numeric expression> after the number itself. For example:

```
PRINT USING"###.#-";-123
PRINT USING"##.#-";12.3
PRINT USING"#,####.#-";-12345.6
```

will result in:

```
123.0-
12.3
12,345.6-
```

6. The "^" symbol (exponent), which is used at the end of the <formatting string>, and outputs the exponential format of a <numeric expression>
For example:

```
PRINT USING "**###.##-";-2.2
PRINT USING "**######+";-123
PRINT USING "**##.####.#-";-12345.6
```

will result in:

```
 12.3456E+04
0.123E+07
-.123E+07
```

7. The "**" symbols (asterisks) used at the beginning of the <formatting string> provide the number with leading asterisks instead of with leading zeros.
For example:

```
PRINT USING "**###.##-";-2.2
PRINT USING "**#####+;-123
PRINT USING "**##,####.#-";-12345.6
```

will result in:

```
****2.20-
****123-
***12,345.6-
```

NOTE When characters that are not described above are used, they will be printed before or after the numeric values.

SEE ALSO: PRINT/LPRINT, PRINT#, and PRINT# USING.

SAMPLE PROGRAM:

```
10 PRINT "LET'S CREATE TWO HUNDRED RANDO
M NUMBERS OF FOUR COLUMNS EACH."
20 FOR I=1 TO 200
30 R=RND(1)*10000
40 PRINT USING"####";R
50 NEXT I
60 END
```

PSET

FUNCTION: This instruction sets a desired dot pattern on the LCD screen of the PC-3300.

FORMAT: PSET (<horizontal coordinate>, <vertical coordinate> [,<function code>])

SAMPLE STATEMENT:

```
PSET (80,32)
```

DESCRIPTION: The PSET instruction sets dots on the screen at the designated coordinates. The <vertical> and <horizontal> coordinates of the <function coce> must be within the range from 0 to 255, or else an error occurs.

The LCD display has 240 dots horizontally and 64 dots vertically. If the <horizontal coordinate> is greater than 239 it is converted to 239, and, smilarly, if the <vertical coordinate> is greater than 63 it is converted to 63.

When the <function code> is an even number, the PSET instruction reverses, and operates exactly the same way as the PRESET instruction. IF the <function code> is an odd number, the instruction operates the same as if it was omitted.

SEE ALSO: PRESET.

SAMPLE PROGRAM:

```
10 SCREEN 0,0:CLS
20 A=150:B=.05:C=11
30 FOR T=-15 TO 72 STEP .13
40 X=EXP(-T*B)*COS(160*3.14*T/180-A)
50 Y=EXP(-T*B)*COS(160*3.14*T/180-C)
60 X=X*120+120:Y=Y*32+32
70 IF X>=0 AND X<256 AND Y>=0 THEN PSET(
X,Y)
80 NEXT
90 BEEP
100 END
```

READ

FUNCTION: This nstruction is used tc read a value from a DATA instruction and assign data tc a variable.

FORMAT: READ <variable list>

SAMPLE STATEMENT:
```
READ A,2,H$
```

DESCRIPTION: The READ instruction is always used in conjuction with the DATA instruction. The READ instruction is used to accept data from the DATA instructions and assigns corresponding data to a variable. Numeric or string variables may be contained in the READ instruction.

A sirgle READ instruction may access one or more DATA instructions (accessed in order). In addition, multiple READ instructions may access a single DATA instruction. If the number of data items in the DATA instruction is less than the variables specified in the <variable list> an "?OD Error" (Out of Data) message is displayed.

When the number of designated variables in the <variable list> is less than the number of data items in a DATA instruction, the next READ instruction accesses data not read previously. If no more READ instructions are coded in the program, any unused data is ignored.

The RESTORE instruction sets the counter which controls the reading of data items back to the start of the data.

SEE ALSO: RESTORE and DATA.

SAMPLE PROGRAM:
```
10 CLS:LOCATE 8,3
20 FOR I=0 TO 8
30 READ R$
40 PRINT R$;"  ";
50 NEXT
60 END
70 DATA Please, read, this, manual.
80 DATA I, (PC-8300), am, reading, data.
```

REM

FUNCTION: A REMark instruction is used to put nonexecutable lines such as remarks or comments in a program.

FORMAT: [REM / '] { <remark> }

SAMPLE STATEMENTS:

```
REM THIS IS A TEST PROGRAM
'THIS IS A TEST PROGRAM
```

DESCRIPTION: A REM instruction is used to include explanatory remarks or comments in a program. It is not an executable statement.

There is a single quotation mark on the keyboard, used as an apostrophe. An apostrophe (') can be used as a substitute for the keyword "REM" in a remark statement.

When the program is listed, all the REM instructions are output unchanged. REM instructions may be used in multi-statement lines only as the last statement. This is because all statements that follow the REM instruction in the multi-statement line would be treated as the <remark>, and they would not be executed.

SAMPLE PROGRAM:

```
10 REM ** REM **
20 REM A REMARK statement is included as
 an explanation in a program.
30 'An apostrophe can be substituted for
 the keyword "REM" in a REM statement.
40 REM The PC-8300 disregards anything i
n a REM statement that follows the keywo
rd "REM".
50 REM Any commands that follow a REM st
atement in the same line will also be di
sregarded.
60 PRINT "HOWEVER, THE REVERSE WITH A RE
M STATEMENT AFTER ANOTHER STATEMENT IN A
 LINE IS POSSIBLE.: REM This is useless.
"
70 END
```

RENUM

FUNCTION: This instruction is used to reorganize the line numbers of a program.

FORMAT: RENUM { <new line number> }{,<old line number>} {,<increment>}

SAMPLE STATEMENTS:

```
RENUM
RENUM 101,50
RENUM ,,6
```

DESCRIPTION: The <new line number> is the line number replacing the <old line number> when renumbering, with a default value of 10. The <old line number> is the first line to be renumbered as <new line number>, with its default value being the first line number of the currert program. The optional <increment> is the difference between the line numbers, with the default value being 10.

The RENUM instruction can renumber lines used in conjunction with GOTO, GOSUB, ON...GOTO, ON...GOSUB, THEN, RESTORE and ERL instructions. If a non-existent line is designated by one of these statements, an "Undefined line *(llll)* in YYYY" error message appears on the screen. In such cases, the erroneous line number *(lll)* cannot be modified via the RENUM instruction, but line number (yyyy) can be altered.

The PC-8300 returns to the Direct Mode after the RENUM instruction is executed.

The RENUM instruction cannot be used to change the sequence of program lines, for example, using RENUM 15,30 with three lines numbered 10,20, and 30 in a program.

Line numbers greater than 65529 cannot be used or else an "?FC Error" (Illegal Function Call) message will occur.

RESTORE

FUNCTION: The RESTORE instruction is used to reset the data list pointer back to the start of the data, and thus prepare the data for re-use.

FORMAT: RESTORE [<line number>]

SAMPLE STATEMENT: `RESTORE 80`

DESCRIPTION: The RESTORE instruction is used when the same data elements (from the DATA statement) are needed to be utilized more than once.

If <line number> is omitted, the first DATA instruction in the program is accessed by the next READ instruction.

If <line number> is specified, the first item of the DATA instruction (designated by <line number>) is the next item to be accessed.

SAMPLE PROGRAM:

```
10 FOR I=0 TO 19
20 READ A$:PRINT A$"  ";
30 RESTORE 70
40 NEXT I
50 RESTORE 80
60 READ A$:PRINT A$
70 DATA Anything
80 DATA "can be read as data."
90 END
```

RESUME

FUNCTION: This instruction is used to continue program execution after performing an error processing routine.

FORMAT: RESUME [<0> | <NEXT> | <line number>]

SAMPLE STATEMENTS:

```
RESUME
RESUME NEXT
RESUME 100
```

DESCRIPTION: The RESUME instruction terminates an error handling routine and the parameter specifies NEXT action when program execution continues. This instruction functions in a manner similar to the RETURN statement, but may only be used in an error processing routine, and then returns control to BASIC after an error processing routine has been performed.

Depencing on the location where program execution is to continue after an error processing routine, one of the following three formats is selected:

1. RESUME or RESUME0 - continues execution at the statement that caused the error.

2. RESUME NEXT - continues execution at the statement immediately after the statement where the error occurred.

3. RESUME <line number> - continues execution but control is transferred to the line specified

SEE ALSO: ON ERROR GOTO.

RETURN

FUNCTION: The RETURN instruction terminates execution of a subroutine and returns cortrol to the statement following the GOSUB (call) instruction.

FORMAT: RETURN {<line number>}

SAMPLE STATEMENTS:

```
RETURN
RETURN 200
```

DESCRIPTION: A RETURN instruction at the end of the subroutine transfers control to the first statement which follows the GOSUB instruction in the main program.

If an optional <line number> is included in the RETURN instruction, program execution control is transferred to the line number specified, instead of the statement following the GOSUB instruction.

A GOSUB instruction is used to transfer control to a subroutine. If RETURN is encountered without first being sent to a subroutine by a GOSUB command, a (RETURN without GOSUB) error message "?RG Error" will be displayed.

A subroutine can have more than one RETURN instruction, but only the first RETURN instruction is executed each time the subroutine is called.

NOTE: If a CLEAR instruction is executed in a subroutine, the line number to which the subroutine is to return is removed from the memory. An "?RG Error" (RETURN without GOSUB) error message results when the RETURN instruction is reached.

SEE ALSO: CLEAR, GOSUB...RETURN, and ON...GOSUB.

SAMPLE PROGRAM:

```
10 GOSUB 200
20 A%=A%+1: PRINT A%;
30 IF A%<6 THEN GOSUB 200
40 END
200 IF A%<5 THEN RETURN 20
210 RETURN
```

RIGHT$

FUNCTION: This function is used to access a specific number of characters from a string, starting from the right-most position of the string.

FORMAT: RIGHT$ (<character string>, <numeric expression>)

SAMPLE STATEMENT:
```
B$=RIGHT$(A$,4)
```

DESCRIPTION: The <character string> can be a string constant or a string variable. The <numeric expression> is a value ranging from 0 to 255, which specifies the number of characters to be read, beginning from the right-most character.

The full <character string> is returned when the <numeric expression> is greater than or equal to the total number of characters in the <character string>. The RIGHT$ function returns a null string when the <numeric expression> is 0.

SEE ALSO: LEFT$ and MID$.

SAMPLE PROGRAM:
```
10 A$="CONTEST"
20 B$=RIGHT$(A$,4)
30 PRINT "THE ";RIGHT$("ALRIGHT",5);"$ F
UNCTION PASSED THIS ";B$;"."
40 END
```

RND

FUNCTION: The RND function generates a random number between 0 and 1.

FORMAT: RND (<numeric expression>)

SAMPLE STATEMENT:
```
PRINT RND(9.9)
```

DESCRIPTION: The RND (random) function is used to pick a number, flip a coin, draw a card, etc.

The random number that is returned by the RND function is a floating point (real number) between 0 and 1, and it depends upon the <numeric expression>. The following cases apply to the RND function:

- If the <numeric expression> is positive, an ordinary random number is generated.
- If the <numeric expression> is zero, the most recently generated random number is repeated.
- If the <numeric expression> is less than zero (negative number), a new random series is established by changing the random seed.

SAMPLE PROGRAM:

```
10 X=120:Y=32
20 SCREEN 0,0:CLS
30 X=X+INT (RND(1)*3)-1
40 IF X<0 OR X>255 THEN X=120
50 Y=Y+INT (RND(1)*3)-1
60 IF Y<0 OR Y>63 THEN Y=32
70 PSET (X,Y)
80 GOTO 30
90 END
```

RUN

FUNCTION: This instruction is used to execute a program already in memory, or to load a program and execute it.

FORMAT: RUN {<line number>}
RUN "{<device name>:}<program name>"{,R}

SAMPLE STATEMENTS:

```
RUN 100
RUN "GAME"
```

DESCRIPTION: The format of RUN [<line number>] is used to execute a program (which is already in memory) from a designated <line number>. Program execution starts from the first line if the <line number> is not specified.

When a parameter is not specified with the RUN instruction, the program currently in the memory is executed starting from the first statement of that program. If a program does not exist in the memory, the PC-8300 will display an "Ok" message and no execution is performed.

The format RUN "[<device name>:]<program name>"[,R] loads a program file from the RAM if <device name> is omitted. When "CAS:" is designated, a program file from a cassette tape in a data recorder is loaded and executed. If option "R" is included, it will open all data files.

When a RUN instruction is executed all open files are closed, and the contents of the BASIC area is cleared when the program is loaded.

The PC-8300 reverts back to the Direct Mode after program execution is completed.

NOTE: The loading of a program in response to RUN "CAS:" can be interrupted by pressing [SHIFT] [STOP].

SAMPLE PROGRAMS:

```
10 ' SAVE THIS PROGRAM UNDER THE NAME "R
UN 1"
20 ' ** RUN 1 **
30 ' It's tricky to use a "RUN" command
from within a program.
40 PRINT "WHEN IT RUNS, THE PROGRAM WILL
 NOT STOP"
50 PRINT
60 PRINT "PRESS THE STOP KEY"
70 PRINT
80 RUN "RUN 2"
90 END
```

```
10 ' SAVE THIS PROGRAM AS "RUN 2"
20 ' ** RUN 2 **
30 PRINT "NOW, RUN 2 IS RUNNING."
40 PRINT
50 PRINT "NEXT, LET'S RETURN TO RUN 1."
60 PRINT
70 RUN "RUN 1"
80 END
```

SAVE

FUNCTION: This instruction is used to save a program on a designated device.

FORMAT: SAVE "[<external device name>:]<file name>"[,A]

SAMPLE STATEMENTS:

```
SAVE "ENERGY",A
SAVE "CAS:ENERGY",A
```

DESCRIPTION: This instruction saves a program currently in the memory into the RAM or onto an external device. The designated <file name> can be up to six characters long. When a <file name> specified is the same as an existing filename the original file's contents wil be overwritten by the new file's contents. After the command is executed, the PC-8300 returns to the Direct Mode.

The PC-8300 saves a program file to the RAM if <external device name> is omitted. "CAS:" is used as <external device name> for a cassette data recorder, while "COM" is used for an RS-232C device, and "LPT:" is used to designate a printer.

For more details, please refer to the CSAVE instruction for "CAS:" the OPEN instruction for "COM:", and the LLIST instruction for "LPT:"

File type ".BA" is automatically selected if no file-type is specifiec. If file type ".DO" s designated for a ".BA" file, or if option "A" is assigned, then a ",DO" file in ASCII format is created.

Once a program file is saved, it is maintained as a file unless another program is saved with the same filename, or until a KILL instruction specifying it as the file to be killed or erased is executed, or when a cold start is performed.

An '?FC Error" (Illegal Function Call) message will be displayed f a program is saved twice with the same file name.

NOTE: A program file in the RAM cannot be saved if it is retrieved into the BASIC area by a LOAD instruction.

The LIST instruction can be executed before the SAVE instruction. This is to display the program contents before saving, and any required changes can then be made.

If screen editing is performed while a program is in the access mode (indicated by an asterisk when the FILES instruction is executed), the original statement(s) is overwritten by the newly input statement(s).

A program should be saved as a ".DO' file if adequate memory capacity is available. If this is not possible, try saving the program on cassette tape as a ".BA" file. Use the option "A" when saving a ".DO' (ASCII format) file on cassette. [SHIFT][STOP] can be pressed to interrupt the SAVE "CAS:" instruction.

SEE ALSO: CSAVE, LOAD, LLIST, BSAVE, OPEN 'COM:', and Chapter 5, Files.

SCREEN

FUNCTION: This instruction establishes the display mode.

FORMAT: SCREEN 0, <function key display switch>

SAMPLE STATEMENT:
```
SCREEN 0,0
```

DESCRIPTION: The SCREEN instruction establishes the display mode. When the <function key display switch> is 0, the function key assignments are not displayed, and the 8 lines of the screen are available to you.

The frst parameter is dummy data and can be omitted, but a comma is always needed. For example:

SCREEN 0,1 (function key assignment displayed)
SCREEN 0,0 (function key assignments hidden)

The <function key device switch> must range between 0 to 255, or else an error occurs.

SEE ALSO: CLS.

SAMPLE PROGRAM:
```
10 FOR I=0 TO 21
20 SCREEN 0,I MOD 2
30 NEXT
40 END
```

SGN

FUNCTION: This function determines whether a number has a negative or positive sign.

FORMAT: SGN (<numeric expression>)

SAMPLE STATEMENT:
```
PRINT SGN (-245)
```

DESCRIPTION: The SGN function returns 1 if the <numeric expression> is positive, 0 if the <numeric expression> is 0, but −1 if the <numeric expression> is negative.

SAMPLE PROGRAM:
```
10 READ X
20 IF X=999 THEN END
30 PRINT X,SGN(X)
40 GOTO 10
50 DATA 55,2,0,-3,4,18,5,999
60 END
```

SIN

FUNCTION: This function provides the sine of an angle expressed by a numeric expression.

FORMAT: SIN (<numeric expression>)

SAMPLE STATEMENT:
```
PRINT SIN(3.14159/2)
```

DESCRIPTION: The SIN function has many practical uses such as trigonometric applications. The <numeric expression> is an angle expressed in radians.

NOTE: To convert an angle from degrees to radians, multiply it by .0174533.

SEE ALSO: ATN COS, and TAN.

SAMPLE PROGRAM:
```
10 SCREEN 0,0:CLS
20 X=0:N=0:F=1
30 Y=SIN(N/25)*32+33
40 PSET(X,Y)
50 IF X<1 THEN F=1
60 IF X>239 THEN F=-1
70 X=X+F:N=N+1
80 GOTO 30
90 END
```

SOUND

FUNCTION: This instruction produces a designated sound.

FORMAT: SOUND <tone> , <length>

SAMPLE STATEMENT:

```
SOUND 5586,50
```

DESCRIPTION: This instruction designates a tone and length, and produces a sound. The integer for the tone ranges from 0 through 16383, with the higher numbers producing a higher pitch tone. Length is comprised of integers between 0 through 250, where the length of a single unit is 0.02 seconds.

The designation of 5586 in the example produces a sound of 440 Hz.

MUSICAL SCALE TABLE:

		OCTAVE					
	—	1	2	3	4	5	6
CODE	C	—	9394	4697	2348	1171	587
	C#	—	8866	4433	2216	1103	554C
	D	—	8368	4184	2092	1045	523O
	D#	15800	7900	3950	1975	987	493D
	E	14912	7456	3728	1864	932	466E
	F	14064	7032	3516	1758	879	439
	F#	13284	6642	3321	1660	830	415
	G	12538	6269	3134	1567	783	—
	G#	11836	5918	2954	1479	733	—
	A	11172	5586	2793	1396	693	—
	A#	10544	5272	2636	1316	653	—
	B	9952	4968	2486	1244	622	—

SEE ALSO: BEEP.

SAMPLE PROGRAM:

```
10 DIM S(17):Z#=4697
20 FOR I=1 TO 17
30 S(I)=Z#
40 Z#=Z#/1.0594639#
50 NEXT
60 FOR I=1 TO 16
70 SOUND S(15),32/I:SOUND S(17),32/I
80 SOUND S(13),32/I:SOUND S(1),32/I
90 SOUND S(8),48/I:SOUND S(0),16/I
100 NEXT I
110 END
```

SPACE$

FUNCTION: This function provides spaces (blanks) of a desired length.

FORMAT: SPACE$ (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT "A"+"B"+SPACE$(5)+"C"
```

DESCRIPTION: The SPACE$ function is used in spacing output for reports and forms. It will provide a string of spaces determined by the designated <numeric expression>. The value of the <numeric expression> must range from 0 to 250.

SEE ALSO: TAB.

SAMPLE PROGRAM:

```
10 FOR Z=1 TO 12
20 PRINT "*"+SPACE$(Z)+"*"
30 NEXT Z
40 END
```

SQR

FUNCTION: This function provides the square root of a number.

FORMAT: SQR (<numeric expression>)

SAMPLE STATEMENT:
```
PRINT SQR (16)
```

DESCRIPTION: The SQR function is used to compute the square root of a positive <numeric expression>. If the <numeric expression> is negative, the message "?FC Error" (Illegal Function Call) will be displayed.

SAMPLE PROGRAM:
```
10 INPUT "WHAT'S YOUR NUMBER";X
20 IF X=0 THEN END
30 PRINT "THE SQUARE ROOT IS";SQR(X)
40 GOTO 10
50 END
```

STOP

FUNCTION: The STOP instruction is used to halt program execution and return to the Direct Mode.

FORMAT: STOP

SAMPLE STATEMENT:
```
STOP
```

DESCRIPTION: When a STOP instruction is executed, the PC-8300 halts the execution of a program, and the following message is displayed on the screen. "Break in llll" with "llll" representing the line number at which the STOP command stopped execution.

A STOP instruction differs from an END instruction because STOP does not close the file. This instruction is useful for debugging programs. The execution of the program can be resumed by using the CONT (CONTinue) instruction, unless the program has been altered while stopped.

SEE ALSO: CONT.

SAMPLE PROGRAM:
```
10 PRINT "Use a STOP command for debuggi
ng."
20 PRINT "Use a CONT command to continue
 the execution of the program."
30 STOP 'USE CONT TO CONTINUE
40 I=1:PRINT I;"Resume execution."
50 GOTO 20
60 END
```

STR$

FUNCTION: This function converts a numeric value to a numeric string.

FORMAT: STR$ (<numeric expression>)

SAMPLE STATEMENT:

```
A$"STR$(123)
```

DESCRIPTION: The STR$ function converts the value of the <numeric expression> to a string. It is useful for programming a sort routine that includes both numbers and characters. If the <numeric expression> contains a non-numeric character, then a 0 will be returned.

SEE ALSO: VAL and STRING$.

SAMPLE PROGRAM:

```
10 PRINT "ENTER A 1 OR 2 DIGIT NUMBER"
20 INPUT "NOW, WHAT HOUR IS IT" ;H:H$=MI
D$(STR$(H),2)
30 IF LEN(H$)=1 THEN H$="0"+H$
40 INPUT "HOW MANY MINUTES" ;M:M$=MID$ (
STR$(M),2)
50 IF LEN(M$)=1 THEN M$="0"+M$
60 INPUT "HOW MANY SECONDS";S:S$=MID$(ST
R$(S),2)
70 IF LEN(S$)=1 THEN S$="0"+S$
80 TIME$=H$+":"+M$+":"+S$
90 PRINT "THE TIME HAS NOW BEEN SET AT "
;TIME$;"."
100 END
```

STRING$

FUNCTION: This function provides a string which contains the specified character, repeated a designated number of times.

FORMAT: STRING$ (<numeric expression>), [<character string> / <ASCII code>]

SAMPLE STATEMENTS:

```
PRINT STRING$(10,"*")
PRINT STRING$(10,45)
```

DESCRIPTION: The STRING$ function returns a string which contains the desired <character string> or <ASCII code>, repeated the number of times designated by <numeric expression>.

The <numeric expression> must be in the range from 0 to 250. If it is not within this range, a "?TM Error" (Type Mismatch) message is displayed. The <ASCII code> is converted to its equivalent character code and then it is returned by this command.

If the <character string> is more than one character, only the first character is returned.

SEE ALSO: STR$.

SAMPLE PROGRAM:

```
10 PRINT STRING$(20,"*");"HEADING";STRING$(10,"*"
20 PRINT
30 PRINT STRING$(20,"-");"LINE ONE"
40 PRINT STRING$(20,"*");"LINE TWO"
50 PRINT STRING$(20,45);"LINE THREE"
60 END
```

TAB

FUNCTION: The TAB function is used to space out or separate data to be printed or displayed on a line.

FORMAT: TAB (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT "A";TAB(10);"B"
```

DESCRIPTION: This function is useful for printing reports, tables, and forms, and to organize the screen display for maximum readability.

It spaces out or separates data to be printed or displayed on the current line. Before the printing begins, the cursor or the print-head skips to the position specified by the <numeric expression>. The <numeric expression> must be in the range of 0 to 255, or else an "?FC Error" (illegal function call) message will be displayed on the screen.

Under these conditions, the cursor cannot move backward, so if the position specified by the <numeric expression> is to the left of the cursor, the TAB function will start displaying from the right side of the cursor.

The TAB function is only used with PRINT and LPRINT statements.

NOTE: You can use more than one TAB function on the same line.

SEE ALSO: SPACE$.

SAMPLE PROGRAM:

```
10 FOR I=1 TO 21 STEP 4
20 PRINT STRING$(I,"#");TAB(22-I);"*"
30 NEXT
40 END
```

TAN

FUNCTION: This function returns the tangent of an angle.

FORMAT: TAN (<numeric expression>)

SAMPLE STATEMENT:

```
PRINT TAN(3.14159/4)
```

DESCRIPTION: The TAN function is used in trigonometric applications. It computes the tangent of an angle expressed in radians.

NOTE: To convert an angle from degrees to radians, multply it by .0174533.

SEE ALSO: ATN, COS, and SIN.

SAMPLE PROGRAM:

```
10 INPUT "ENTER AN ANGLE IN DEGREES";D
20 PRINT "THE ";D;" DEGREES ANGLE IS";D*
.0174533;"RADIAN AND ITS TANGENT IS";TAN
(D*.0174533)
30 END
```

TIME$

FUNCTION: This reserved variable provides the time from the internal real-time clock of the PC-8300.

FORMAT: TIME$ "<hour>:<minute>:<second>"

SAMPLE STATEMENTS:

```
TIME$="15:30:20"
PRINT TIME$
```

DESCRIPTION: The TIME$ is used to set the current time. The <hour> is a number between 00 and 23. Both the <minute> and <second> values are numbers ranging from 00 through 59, used when the time is set. Resetting the time is not necessary, unless a Cold Start has been performed.

SEE ALSO: DATE$.

VAL

FUNCTION: This function returns the numeric value of a numeric string.

FORMAT: VAL (<numeric string>)

SAMPLE STATEMENT:

```
PRINT VAL("123")
```

DESCRIPTION: The VAL function returns the numeric value of a numeric string. The "+" or "−" sign can be used as the first character of the <numeric string>.
For example:

```
VAL("-1234.567") → -1234.567
```

Any spaces in the <numeric string> are disregarded.
For example:

```
VAL("12 12") → 1212
```

If any other character not mentioned above is used within the <numeric string>, anything after that character is ignored. For example

```
VAL("123a4") → 123
VAL("ab") → 0
```

SEE ALSO: STR$ and CHR$.

SAMPLE PROGRAM:

```
10 A$="123":B$="456,7":C$="-8.9"
20 X=VAL(A$):Y=VAL(B$):Z=VAL(C$)
30 D$=A$+B$+C$
40 N=X+Y+Z
50 PRINT A$, B$, C$, D$
60 PRINT X,Y,Z,N
70 END
```

XOR

FUNCTION: This logical operator is used to test multiple relations.

FORMAT: <operand 1> XOR <operand 2>

SAMPLE STATEMENTS:

```
IF A=5 XOR B=5 THEN 200
PRINT 5+3 XOR 4+4
```

DESCRIPTION: The logical operator XOR (exclusive OR) performs tests on multiple relations, bit manipulation, and Boolean operations. It returns either a non-zero (true) or zero (false) value.

For the operation to return a non-zero (true) value, one of them has to be true and the other must be false. Otherwise, if both of them are true, or both are false, the operation returns a zero (false) value.

The following truth table indicates the evaluation process:

-1 XOR -1 → 0 (TRUE XOR TRUE → FALSE)

-1 XOR 0 → -1 (TRUE XOR FALSE → TRUE)

0 XOR -1 → -1 (FALSE XOR TRUE → TRUE)

0 XOR 0 → 0 (FALSE XOR FALSE → FALSE)

NOTE: The XOR command performs in exactly the opposite way compared to the EQV command.

Logical operators work by converting their <operands> to sixteen bits binary integers. Therefore, the <operands> must be in the range from −32768 to +32767. If the <operands> are not within this range, an "?OV Error" (Overflow) message will be displayed on the screen.

EXAMPLE:

Integer	Binary bits
25	0000 0000 0001 1001
13	0000 0000 0000 1101

After inputting the statement PRINT 25 XOR 13, the integer 20 appears on the screen, whose binary form is 0000 0000 0001 0100. By looking at the table in the DESCRIPTION section notice that the computation is correct.

SEE ALSO: AND, EQV, IMP, NOT, and OR.

CHAPTER 5

FILES

A file is a collection of records in the RAM of the PC-8300 or external storage devices, such as a data recorder. Each record consists of a group of logically related characters. For example, an N82-BASIC program line is one record. The PC-8300 uses the record unit to read or write into a file, and each file is assigned a distinct file name when it is created.

5.1 FILE NAMES

A file name consists of three parts:

- The main name, up to 6 characters in length.
- A period, used as a connector in the middle of the file name.
- The file-type extension, added to the end of the file name, which is 2 characters long.

A file name can consist of any combination of characters, but the use of letters instead of numbers or symbols is recommended. You run the risk of getting the error message "?NM Error" (File Name Misnuch) when using characters other than ordinary letters. A legal file name must be entered if this error message is displayed.

An example of a legal file name with a file type extension:

```
PC8300.BA
```

".BA" is the extension which was added by the PC-8300 when the file was saved, in accordance with the correct mode.

The file name may be input in either lower or upper case characters, and will be saved and displayed on the screen exactly as typed. The extension will always be displayed as upper case characters, so it does not matter which way it is typed, if it is input by you.

The file-type extensions represent specific file types:

- ".BA" BASIC files. BASIC programs in Binary format.
- ".DO" TEXT files. TEXT and BASIC programs in ASCII format.
- ".CO" Machine Language files. Programs and data in machine language format.

The file-type extension can be input by you, or the PC-8300 will assign one according to the mode you are using. For the BASIC mode, the file type extension assigned by the PC-8300 would be ".BA".

The file names are displayed on the MENU screen in the following order:

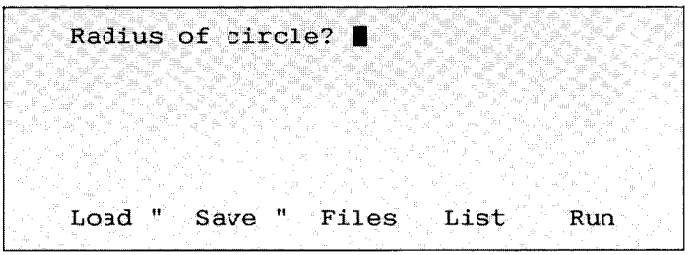
Machine language files
TEXT files
BASIC files

You can also display the file names within the specific bank when in the BASIC mode by using the "FILES" instruction. It is possible to execute BASIC programs from the MENU mode.

EXAMPLE:

The program you created earlier in section 1.5 (page 10) is now saved in the RAM. From the BASIC mode, press [SHIFT] and [f.5] (Menu) to return to the MENU. You will then see the file name RADIUS.BA.

Move the cursor onto the file name "RADIUS.BA" and then press [↵]. The PC-8300 is now in the BASIC mode and the previously created BASIC program 'RADIUS.BA" is executed. The screen will appear as shown:



5.2 BUFFERS

Buffer memory is reserved RAM area that is used by the PC-8300 to store transmitted and received data. Each time you OPEN a file in BASIC you reserve a buffer area. The maximum number of OPEN files that can be open at the same time is 15. This means that the maximum number of buffers that you can reserve is also 15.

5.3 FILE HANDLING

In order to read or write to a file you will first have to open the file. This is done by the use of the OPEN instruction, which utlizes the file number in conjunction with the file descriptor to assign a specific buffer area to that file.

After a file has been OPENed you can use the READ instruction to read records from it and the PRINT instruction to write records to it. When you have completed your processing you will have to close the file by the use of the CLOSE instruction.

CHAPTER 6

MACHINE LANGUAGE PROGRAMMING

5.4 PRECAUTIONS FOR FILE CREATION

When accessing files within the RAM of the PC-8300, the extensions are checked during the process. This means that you can use identical file names for different files if the extensions of those names are different. The PC-8300 will recognize the difference between each of these files during loading and saving, because it will check for an external device descriptor and file-type extension, as well as for the file name.

The maximum number of files that can be stored in each of the three memory banks is 21, depending on the size of the individual files. If an attempt is made to store more than the maximum allowable in a bank, an error will occur, and the message "?FL Error" (Filing Limit) is displayed.

When a machine language file is saved using the BASIC language command "BSAVE", it can then be run directly from the MENU. However, when a file created does not have a designated execute address, the machine language file is loaded into the memory, but will not run directly.

Machine language programming is a collection of meaningful coded instructions that the PC-8300 can execute. All other programming languages must be compiled or translated into machine language before they can be executed. Machine language is also known as assembler language or code.

Machine language programs execute much faster than any other programs, such as BASIC. They take ess memory, and there is virtually no limit to what they can be programmed to dc.

With machine language programs you have the ability to get into any memory location of the PC-3300. If you alter vital memory locations such as the programs that operate the PC-8300, you could get the PC-8300 "hung-up", meaning that it does not respond, no matter what you input. Thus, it is necessary to save important programs or files on external devices, such as a data recorder, because a simple mistake can easily wipe out files in RAM. In the case of such a problem, you will have to perform a cold start, after which only the primary programs of BASIC, TEXT and TELCOM are displayed on the screen. The rest of the files are destroyed. That is why it is so important to save your other files before attempting to run your machine language programs.

See the User's Guide for more details on how to perform a cold start.

6.1 CREATING MACHINE LANGUAGE PROGRAMS

In order to write machine language programs you will have to know the 8085 Assembler Language. An assembler program can be written in the TEXT mode and then the optional assembler language compiler can be used to create machine language code, or the POKE instruction can be used to actually create a machine language routine in the PC-8300 RAM.

Since creating a machine language program is tedious work, make sure you save it using the BSAVE instruction before attempting to test it, which avoids the loss of effort. When debugging (testing) your machine language programs you can use the PEEK function to check the value stored at a specific memory location.

See Chapter 4 for how to use the BSAVE, POKE, and PEEK instructions.

Once the machine language routine has been tested and saved, the BLOAD instruction can be used to load your program into the PC-8300 RAM. The EXEC instruction is then used from within BASIC mode to run it. Before loading a machine language routine, enough space must be reserved within the RAM for it.

For more details on BLOAD and EXEC instructions, please refer to Chapter 4.

All machine language programs should include a RET instruction at the end of the routine, so control can be returned to the BASIC mode.

CHAPTER 7

N82-BASIC PROGRAMMING AIDS

This chapter is designed to provide enough information to make programming easier for beginning programmers. It will aid in the creation of your own programs, as well as help to resolve problems within those programs.

7.1 RECOVERY FROM CRITICAL SITUATIONS

7.1.1 Word Wraparound and Screen Scrolling

Situation:
Scrolling occurs whenever characters are input on the bottom line of the screen, or the space between characters is not what is expected.

Explanation:
The cursor in the BASIC mode is described as a flashing box ; its position is very important when you input or print on the screen display.

Wraparound is a process when characters continue on to the next line of the screen. When characters are input past the 39th position of the current line, they are moved onto the first position of the next line.

- Wraparound is executed when a field longer than 40 characters is printed, or the semicolon ";" is used when printing more than one field on the same line with the total length over 40.

- When you print a field with less than 40 characters in length and the semicolon ";" is not used, the cursor skips to the beginning of the next line when the operation is completed.

Scrolling up means all of the lines of the screen display move up one line, with the top line moving off the screen and a new line appearing at the bottom. Scrolling up occurs if the cursor is on the last line and wraparound occurs.

7.1.2 Spontaneous Program Execution Errors

Situation:
A program started to operate incorrectly but executed previously without any difficulty.

Explanation:
In this situation, the program was somehow modified. This primarily happens when a ".BA" file has been loaded and modified. When programs are loaded into the temporary working area of the PC-8300, they can be modified and stored in the RAM or on external devices, such as a data recorder.

When a program s loaded from the RAM and needs modfication, this program shoud be saved again in the RAM and not on external devices. If a program has been loaded from a cassette tape, do not save it in the RAM unless it is free of errors and operates the way t shouid.

When files loaded from tape are modified and SAVEd in the RAM, the display of the file name includes an asterisk (*) after the file-type extension, when the FILES instruction is used When attempting to LOAD the original file from tape, it is important to recognize that these modified programs may contain potential errors and the bad file could mistakerly be loaded.

7.1.3 Logical Errors

Situation:
When the program result is different from what was expected.

Explanation:
This type of situation is hard to resolve, because it is difficult to determine all the underlying causes. You will have to go through your program statement by statement, and determine the operation of each statement. By doing so, the logical flow of your program may be established.

You have to be persistent, because even if the program initially appears to be in order, it may actually have a problem at some point. Keep in mind that the PC-8300 is executing your commands to the letter, exactly as they were input, and it will do exactly what you ask of it.

EXAMPLE:
Assume that you have the following program:

```
20 DATA 10,13,2,5,6,33
30 FOR I=0 TO 5
40 READ A(I)
50 NEXT
60 FOR I=1 TO 6
70 B=B+A(I)
80 NEXT
90 PRINT B
```

In this program we want to add the numbers 10, 13 2, 5, 6, and 33, and print the result of this calculation. If you RUN the program, the result printed is 59, which is incorrect. The logical error must be found, which is actually in statement 60. Statement 20 defines values for six different numbers, with statement 30 reading the values of the numbers into statements 40 and 50. The array is A, so A (0) will have the value of 10, A(1) a value of 13, A(2) a value of 2, etc. Statements 60, 70, and 80 will add the values of A(1) through A(6) into B, and then statement 90 will print the value of B.

The logical error occurs in statement 60 because we added elements 1 to 6 insteac of 0 to 5. We do not add element 0 which has the value of 10, instead we add element 6 which has not been initialized and therefore it has the value of zero. In order to demonstrate this, change statement 60 to read:

```
60 FOR I=0 TO 5
```

Type RUN and press [↵] and you will see that the correct result, 60, is now returned.

7.1.4 Loss of Program Control

Situation:
STOP is ineffective and you have no control over a program.

Explanation:
In this situation you may have temporarily overlaid vital routines through the use of a POKE instruction or through your own machine language programs. These vital routines include the information that the PC-8300 utilizes for its operation.

The only option you have at this point is to turn the power switch OFF. Files stored in the RAM are erased when this situation is encountered. When the power is turned ON again, no files are displayed on the MENU screen except the primary ROM-resident files called BASIC, TEXT, and TELCOM.

If the PC-8300 still does not operate correctly in some way, perform a cold start. To do this, hold [CTRL] and [SHIFT], and then press and release the RESET swtch on the back of the PC-8300. If necessary, refer to the User's Guide.

7.1.5 Return to BASIC from TEXT is Impossible

Situation:
When editing a BASIC program within the TEXT mode, it may be impossibe to exit from this mode.

Explanation:
In this situation, the message "Text ill-formed" is displayed on the screen whenever you try to exit and return to the BASIC or MENU mode. This happens because a statement within the program is longer than 255 characters, or the statement format is illegal.

The PC-8300 locks you out, and pressing [STOP] or [f.10] have no effect except to display the error message. To resolve this problem, it is necessary to find the long statement and make it shorter, or re-format the statement. Exit from the TEXT mode should then be possible.

7.2 PROGRAMMING HINTS

7.2.1 Hints for Detecting Errors:

1. A flowchart (a chart depictirg the course of program operations) should be carefully constructed. This is especially useful when beginning programmers are suddenly confrorted with a major error in the middle of a program.

2. The PC-8300 User's Guide and this N82-BASIC Reference Manual should be carefuly read and you should understand and try out the commands and functions utilized by the PC- 8300.

3. A record of the variables you have assigned should be kept to avoid any duplication in the names of variables.

4. Make it a point to use REM statements to write remarks for your own use within a program, and avoid multiple statements as much as possible; make the program easy to understand.

5. If a particular ine does not work at all, isolate it by writing REM at its beginning, (to make the computer ignore it) rather than eliminating it. You can easily modify it later.

6. Use a STOP instruction to confirm any changes in the value of a variable. A CONT instruction can be used to resume after STOPping.

7.2.2 Hints for Speeding Up Program Execution:

1. Spaces and REM instructions should be eliminated.

2. Integer variabes should be used whenever possible.

3. Omit a contro variable designation within NEXT statements when possible.

4. Multiple statements should be used as much as possible.

5. Use the format A = 0 at the beginning of a program for any frequently used variables.

6. Frequently used subroutines should be placed at the beginning of a program.

7. Make sure that the region reserved for string use is adequate.

8. Try to simplify frequently used loops.

7.2.3 Hints for Saving Memory Space:

1. Use multiple statements whenever possible.

2. Remove spaces and REM statements from the program

3. Constants should be held as variables, no matter how many times a constant appears within a program.

4. Utilize old variables no longer being used within a program, instead of defining new variables.

5. When there are numerous situations where the same process is usec, write it as a subroutine.

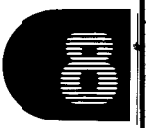
6. Any array variable used should be declared. If it has not been declared, it is automatically declared to 10.

7. Integer variables should be used whenever possible.

8. Keep the memory area reserved for strings to a minimum.

CHAPTER 8

ERROR MESSAGES



This chapter outlines causes and what action you should take when error messages are displayed on your screen. There are 43 messages programmed into the PC-8300. Many more error messages could be defined by you, using a BASIC program.

If an incorrect system command, statement, or function is encountered while a BASIC program is running, the program will terminate abnormally and an error message will be displayed.

N82-BASIC has a built-in error trap function. To simplify the process of determining the source of errors within a program, the explanations of error messages listed are in alphabetical order.

MESSAGE: **?AO Error** File is Already Open

POSSIBLE CAUSES:

1. The execution of an OPEN instruction for a file already open.
2. The execution of a KILL instruction for an open file.

USER ACTION: Close the file using the CLOSE instruction before trying to KILL it or OPEN it again.

MESSAGE: **?BN Error** Bad file number is used.

POSSIBLE CAUSES:

1. When a PRINT instruction is used with a file number not previously designated by an OPEN instruction.
2. When an OPEN instruction is used to assign a file number larger than the maximum number designated by a MAXFILES instruction.

USER ACTION:

1. OPEN the file.
2. Use the MAXFILES instruction to assign the desired number of files.

MESSAGE: **?BO Error** Bufer is Overflowed.

POSSIBLE CAUSE: An attempt is made to input more characters than the buffer can hold.

USER ACTION: Adjust the program that creates the file to shorten the length of the records.

MESSAGE: **?BS Error** Bad subscript

POSSIBLE CAUSES:

1. When the subcript of an element of an array is incorrect.
2. When the subscript of an element of an array is outside the dimensions of the array.

USER ACTION:

1. Correct the number of elements specified for arrays within the program.
2. Increase the size of the array dimensions if necessary.

MESSAGE: **?CF Error** Closed File

POSSIBLE CAUSE: An attempt is made to access an unopened file.

USER ACTION: Open the file properly before trying to access it.

MESSAGE: **?CN Error** Continue Not possible

POSSIBLE CAUSES:

1. When a CONT instruction is used after a break occurs in program execution and the program is then ecited.
2. When a CONT instruction is written as a statement within a program.
3. When a CONT instruction is used after a break occurs in program execution, following a CLEAR instructon.

USER ACTION:

1. Rerun the program by using a RUN instruction.
2. Eliminate the CONT instruction from the program.
3. Rerun the program from the beginning.

MESSAGE: **?DD Error** Duplicate Definition

POSSIBLE CAUSE: An attempt is made to redefine an array previously designated by use of the DIM instruction.

USER ACTION: Use the CLEAR command within the program to clear all arrays so that they can be redefined. When the NEW or RUN instruction is used, all arrays will be cleared automatically.

MESSAGE: **?DS Error** Direct Statement in File

POSSIBLE CAUSE: When loading a file using the LOAD instruction with a file-type extension ".DO", the file contains a statement without a line number.

USER ACTION: Enter the ".DO" file while in the TEXT mode and add line numbers to all lines within the file.

MESSAGE: **?DV Error** Device Unavailable

POSSIBLE CAUSE: When there is something unusual or incorrect for a device designation.

NOTE: An "?FC Error" (Illegal Function Call) occurs if no external devices are connected to the PC-8300.

MESSAGE: **?EF Error** End of File

POSSIBLE CAUSE: When using the INPUT instruction or LINE INPUT instruction beyond the end of the file.

USER ACTION: Use the EOF function in conjunction with INPUT or LINE INPUT instructions to detect the end of the file and avoid going past it.

MESSAGE: **?FC Error** Illegal Function Call

POSSIBLE CAUSES: A parameter that is out of range is passed to a math or string function. May also occur as the result of:

1. A negative or unreasonably large subscript.
2. A negative or zero argument for LOG
3. A negative argument for SQR or CLEAR
4. When ".BA" files are combined with a MERGE instruction.
5. When a RENUM instruction is used incorrectly and line sequence is changed.
6. When a device is used that is not connected or is incorrectly connected to the PC-8300.
7. When parameter values are not within the proper range for a CLOSE, EFR, LOCATE, MOTOR, GOTO, GOSUB, OUT, POKE, POWER, PRESET, SCREEN, CHR, EOF, INP, INPUT, INST, LEFT$, MID$, RIGHT$, SPACE, STRING, TAB, KEY, MAXFILES, or SOUND.

USER ACTION:

1. Be sure all peripheral devices used with the PC-8300 are attached correctly.

2. Correct all parameter designations entered into the program incorrectly.

See Chapter 4 for legal parameter designations of system commands, statements, and functions.

MESSAGE: **?FF Error** File Not Found

POSSIBLE CAUSES:

1. When a file used with a LOAD, KILL, or OPEN instructon is not on a designated device. If the device designated is a data recorder, the PC-8300 will continue searching for the file until the end of the tape is reached.

2. When a file with a type extension other than ".CO" is loaded using the BLOAD instruction.

USER ACTION:

1. Be sure all files loaded with the BLOAD instruction are ".CO" files.

2. Use [SHIFT][STOP] to interrupt the searching and try the command with the correct name.

MESSAGE: **?FL Error** Filing Limit

POSSIBLE CAUSE: When the MENU directory is filled with file names, and no space is available for display of a new filename although some memory may still be available.

USER ACTION: Move some files to external devices and KILL unwanted files, to create space for more directory entries

MESSAGE: **?IE Error** Internal Error

POSSIBLE CAUSE: An error occurs within BASIC itself.

USER ACTION: Consult your Authorized NEC Dealer.

MESSAGE: **?IO Error** Input-Output Error

POSSIBLE CAUSES:

1. When SHIFT STOP are pressed to forcibly stop input or output to/from an external device.
2. When peripheral equipment is in need of maintenance.

USER ACTION: Check equipment if error occurred spontaneously. It may need maintenance such as cleaning of recording heads.

MESSAGE: **?LS Error** Long String

POSSIBLE CAUSE: An attempt is made to designate a string longer than 255 characters.

USER ACTION: Use multiple variables to break down string length to avoid exceeding the limit of 255 characters. If the string was made too long in error, simply change the length designated in the program.

MESSAGE: **?OM Error** Missing Operand

POSSIBLE CAUSE: A necessary operand is missing.

USER ACTION: Check the program and insert the omitted operand.

See Chapter 4 for full explanations of statement format.

MESSAGE: **?NE Error** NEXT without FOR

POSSIBLE CAUSES:

1. A program attempts to execute a NEXT instruction without the previous execution of a corresponding FOR.
2. When a GOTO or GOSUB subroutine causes a program to jump into a FOR NEXT loop.
3. When a FOR NEXT loop is incorrectly nested.

USER ACTION:

1. Check that the program has the same number of NEXT and FOR statements.
2. Check the GOTO and GOSUB subroutine operations included in the program, and correct them if necessary.
3. Correct incorrect nesting of FOR NEXT loops.

See Chapter 4 for rules regarding nested loops.

MESSAGE: **?NM Error** File Name Mismatch

POSSIBLE CAUSES:

1. File name conventions described in Chapter 5 were not followed.
2. An attempt is made to access ".CO" files using commands other than BLOAD or BSAVE.

USER ACTION:

1. Use correct file name and follow the rules exacty.
2. Be sure that the appropriate commands for loading and saving of files are used for different file types.

MESSAGE: **?NR Error** No Resume

POSSIBLE CAUSE: When an error processing subroutine has no RESUME statement

USER ACTION: Add RESUME, END, or ON Error GOTO to error processing subroutines.

MESSAGE: **?OD Error** Out of Data

POSSIBLE CAUSES:

1. The elements read by using the READ instruction do not correspond to the number of elements within the DATA instructions.
2. When a RESTORE instruction is not used at all, or is incorrectly used.

USER ACTION:

1. Check the program to be sure the number of elements designated for READ and DATA instructions correspond.
2. Be sure the program includes a RESTORE instruction in the appropriate place, before trying to read DATA elements that have been previously read.

See Chapter 4 for correct use of the RESTORE instruction.

MESSAGE: **?OM Error** Out of Memory

POSSIBLE CAUSES:

1. When a program is too long to be stored in the memory.
2. When sufficient memory is available for storage of a program but there is not enough available to run it.
3. When an array is too large for the available memory.
4. When a string is too large for the available memory space.
5. When nesting becomes excessively deep with FOR or GOSUB instructions.
6. When you are creating or expanding a file and there is no memory available.
7. When memory area required for a machine language application is too small.

USER ACTION: Move files to external devices, such as a data recorder, or KILL unwanted files to create memory space.

MESSAGE: **?OS Error** Out of String Space

POSSIBLE CAUSE: Sufficient working memory area for string handling has not been created.

USER ACTION: Utilize the CLEAR instruction to reserve enough RAM space for string operations. The default value for the working area is 255 characters. You can use combined (concatenated) strings totaling 255 characters in length. If more area is needed, you will have to use the CLEAR instruction to reserve more space.

MESSAGE: **?OV Error** OVerflow

POSSIBLE CAUSES:

1. When results of an integer operation or substitution are not within the range of −32768 through +32767.
2. When the results of a real number operation are not between −1.70141E+38 and 1.70141E+38.
3. When parameters used with POKE, OUT, and DIM instructions are not within the proper range.

USER ACTION: Rearrange operations within the program so that they are within the legal ranges.

See Chapter 4 for descriptions of legal ranges for statements, and Chapter 3 for ranges of integer and real number operations.

MESSAGE: **?PC Error** PC-8001 format

POSSIBLE CAUSE: When a BASIC program, which cannot be executed in N82-BASIC (a program for PC-8001), is loaded into the PC-8300.

USER ACTION: The program will need to be written and modified into an N82-BASIC program. This error will usually not occur because an "?SN Error' or "?FC Error" will occur first.

MESSAGE: **?RG Error** RETURN without GOSUB.

POSSIBLE CAUSE: An attempt is made to execute a RETURN instruction without a corresponding GOSUB instruction.

USER ACTION:

1. Make sure you are not using a GOTO to execute a subroutine.
2. Make sure to use an END statement, so the program does not fall through any possible subsequent subroutines.

MESSAGE: **?RW Error** Resume Without Error

POSSIBLE CAUSE: A RESUME instruction is encountered before an error trapping routine is entered.

USER ACTION:

1. Check for any other GOTO or GOSUB instructions which send control to error trapping routines, without using the ON ERROR instruction.
2. Check that you have used an END instruction, so that at the end the program does not fall through any possible subsequent error trapping routines.

See Chapter 4 for more information about the ON ERROR instruction.

MESSAGE: **?SN Error** SyNtax Error

POSSIBLE CAUSES:

1. When a statement does not agree with the grammer of N82-BASIC.
2. When there is only a function or mathematical expression on the left side of a substitution formula (although it can normaly be used alone in a statement).
3. When the name of a variable does not begin with a letter, when a reserved word is included, etc.
4. When a colon is missing as a punctuation mark between multiple statements.
5. When line numbers are not within the range from 0 to 65529.
6. When a variable is used to designate a line number.
7. When an ELSE instruction is used without a THEN in an IF...THEN...ELSE instruction.
8. When the number of dummy variables in a function or the parameters of a command are insufficient or in excess.
9. When two lines become joined together during the screen editing process.

USER ACTION:

1. Use the LIST instruction. In most cases, the number of the line in which the error has occurred will be displayed, after [f.9] is pressed.
2. If two lines are joined together, split this excessively long line in the TEXT mode.
3. Check for an accidental substitution, (1 for l, a period for a comma, a colon for a semicolon etc.).
4. Check names of variables that might contain a reserved word (a keyword), for instance, **COS**T, SH**IF**T, etc.
5. Check for compound numeric formulas that are not properly enclosed by punctuation marks.

MESSAGE: **?ST Error** String formula is Too complex

POSSIBLE CAUSE: When an expression is too long or too complex.

USER ACTION: The expression should be broken into smaller expressions.

MESSAGE: **?TM Error** Type Mismatch

POSSIBLE CAUSES:

1. When a string variable name is assigned a numeric value or vice versa.
2. When a function that requires a numerical argument is given a string argument, or vice versa.
3. When a double precision real number is used as the control variable in a FOR instruction.

USER ACTION: Correct the incorrectly assigned value.

MESSAGE: **?UF Error** Undefined User Function

POSSIBLE CAUSE: When an undefined user function has been called up.

USER ACTION: This error cannot occur in N82-BASIC.

MESSAGE: **?UL Error** Undefined Line number

POSSIBLE CAUSES:

1. When a reference is made to a nonexistent line number.
2. When no line exists with the line number designated by a RESTORE or RUN instruction.
3. When a program has no line with the line number designated in a GOTO or GOSUB instruction.

USER ACTION: Correct program references for ine numbers.

MESSAGE: **?/0 Error** Division by zero

POSSIBLE CAUSES:

1. When division is performed with an undefined variable, (and its initial value has been set at zero).
2. When the variable used as the divisor of an operatior is zero.
3. When the dummy variable of a TAN function is $\pi/2$.
4. When multiplication is performed on zero by a negative exporent.

USER ACTION: Have the value of the variable displayed by the PRINT instruction. Investigate to find where the illegal use occurred.

CHAPTER 9

SAMPLE PROGRAMS

9.1 PSET ROUTINE

The PSET routine is used to draw lines and functions. It specifically draws boxes and circles. You should feel free to use required segments from this program by themselves to function as subroutines when creating new programs.

```
10 '                          LINE BOX CIRCLE
20 SCREEN 0,0:CLS
30 PRINT
40 PRINT " PSET PRACTICE"
50 PRINT
60 PRINT " 1 LINE"
70 PRINT " 2 BOX"
80 PRINT " 3 CIRCLE"
90 PRINT
100 INPUT " WHAT DO YOU WANT TO DRAW";A$
110 ON VAL(A$) GOTO 130,210,290
120 BEEP:GOTO 20
130 '                    LINE COODINATES
140 CLS:PRINT
150 INPUT "COORDINATE FOR POINT X";X0:IF
 X0<0 OR X0>239 THEN BEEP:GOTO 150
160 INPUT "COORDINATE FOR POINT Y";Y0: I
F Y0<0 OR Y0>63 THEN BEEP:GOTO 160
170 INPUT "COORDINATE FOR ENDPOIKNT X";X
1:IF X1<0 OR X1>239 THEN BEEP:GOTO 170
180 INPUT "COORDINATE FOR ENDPOINT Y";Y1
:IF Y1<0 OR Y1>63 THEN BEEP:GOTO 180
190 CLS:GOSUB 370
200 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
210 '                    BOX COODINATES
220 CLS:PRINT
230 INPUT "X COORDINATE";X0:IF X0<0 OR X
0>239 THEN BEEP:GOTO 230
240 INPUT "Y COORDINATE";Y0:IF Y0<0 OR Y
0>63 THEN BEEP:GOTO 240
250 INPUT "SECOND X COORDINATE";X1:IF X1
<0 OR X1>239 THEN BEEP:GOTO 250
260 INPUT "SECOND Y COORDINATE";Y1:IF Y1
<0 OR Y1>63 THEN BEEP:GOTO 260
270 CLS:GOSUB 510
280 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
290 '                    CIRCLE COODINATES
300 CLS:PRINT
310 PRINT "CENTER COORDINATES:"
320 INPUT "X COORDINATE";X0:IF X0<0 OR X
0>239 THEN BEEP:GOTO 320
330 INPUT "Y COORDINATE";Y0:IF Y0<0 OR Y
0>63 THEN BEEP:GOTO 330
340 INPUT "RADIUS";R:IF R<0 THEN BEEP:GO
TO 340
```

```
350 CLS:GOSUB 590
360 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
370 '                  SUB LINE
380 XD=ABS(X1-X0):YD=ABS(Y1-Y0)
390 XS=SGN(X1-X0):YS=SGN(Y1-Y0)
400 IF XD>YD THEN 450
410 F=-1:T=X0:X0=Y0:Y0=T
420 T=X1:X1=Y1:Y1=T
430 T=XD:XD=YD:YD=T
440 T=XS:XS=YS:YS=T
450 R=XD/2
460 IF F THEN PSET(Y0,X0) ELSE PSET(X0,Y
0)
470 IF X0=X1 THEN RETURN
480 X0=X0+XS:R=R+YD
490 IF R>=XD THEN R=R-XD:Y0=Y0+YS
500 GOTO 460
510 '                  SUB BOX
520 FOR I=X0 TO X1 STEP SGN(X1-X0)
530 PSET(I,Y0):PSET(I,Y1)
540 NEXT
550 FOR I=Y0 TO Y1 STEP SGN(Y1-Y0)
560 PSET(X0,I):PSET(X1,I)
570 NEXT
580 RETURN
590 '                    SUB CIRCLE
600 FOR I=0 TO 1 STEP 1/(R*2)
610 II=I*I
620 X=R*I*2/(II+1)
630 Y=R*(1-II)/(II+1)
640 X2=X0-X:IF X2<0 THEN X2=0
650 Y2=Y0-Y:IF Y2<0 THEN Y2=0
660 X1=X0+X:Y1=Y0+Y
670 PSET(X1,Y1):PSET(X1,Y2)
680 PSET(X2,Y1):PSET(X2,Y2)
690 NEXT
700 RETURN
```

9.2 CHARACTER DEFINITION PROGRAM

There are many characters that can be defined by you through the character definition function. When you type in the following program such composition is greatly simplified because up to 125 individual graphic characters can be created at one time using the screen editing process. A group of characters that have been defined at one time as a character set can be loaded one set after another by means of a BLOAD instruction, to give you a hundred or even a thousand graphic characters to work with,if so desired.

Since characters can be skipped over when [ESC] and the "E" key are used, you can even replace individual characters in a given set without erasing or altering others that you wish to retain (and if nothing new is defined, it is also possible to eliminate all).

The newly defined characters are stored into a machine language program. The value of the character corresponds to the ASCII character code represented on the keyboard. The graphic characters are accessed by holding [GRPH] and pressing some other key at the same time.

```
10 REM COPYRIGHT (C) NEC 1983
11 REM Updated for the PC-8201A
12 REM NEC Home Electronics, 11/4/1983
13 REM NEC Corp. Tokyo       11/25/1983
100 REM CHARACTER GENERATOR
110 REM USING ADDRESS F091-F37F
120 CLEAR 256,61584!:DIM M(5,7):DEFINTE-
Z
130 REM ***** INITIALIZE *****
140 SCREEN 0,0:CLS
150 POKE 65215!,145:POKE 65216!,240
160 H=131:C=0:AD=61585!
170 REM ***** MAIN LOOP1 *****
180 LOCATE 15,0:PRINT "   KEY
FUNCTION"
190 LOCATE 15,2:PRINT " SPACE = Mode"
200 LOCATE 15,3:PRINT "CURSOR = Move"
210 LOCATE 15,4:PRINT " 'ESC' = Next"
220 LOCATE 15,5:PRINT "       = Define
Character"
230 LOCATE 15,6:PRINT "     E = End"
240 LOCATE 10,7:PRINT "CHR$(";
250 PRINT MID$(STR$(H),2);")BEING
DEFINED";
260 X=0:Y=0:MX=0:MY=0:H=H+1
270 FOR Y1=0 TO 63:PSET(36,Y1):NEXT:
280 REM ***** MAIN LOOP2 *****
290 IF T=0 THEN C$="ERASE" ELSE
C$="WRITE"
```

```
300 LOCATE 29,2:PRINT "<"C$">"
310 LOCATE X,Y:I$=INPUT$(1)
320 IF I$=CHR$(27) THEN 450
330 IF I$=CHR$(28) THEN X=X+1:IF X=6
THEN X=5 ELSE MX=MX+1
340 IF I$=CHR$(29) THEN X=X-1:IF X=-1
THEN X=0 ELSE MX=MX-1
350 IF I$=CHR$(30) THEN Y=Y-1:IF Y=-1
THEN Y=0 ELSE MY=MY-1
360 IF I$=CHR$(31) THEN Y=Y+1:IF Y=8
THEN Y=7 ELSE MY=MY+1
370 IF I$=CHR$(32) THEN T=NOT T
380 IF I$="E" OR I$="e" THEN 600
390 IF I$=CHR$(13) THEN GOSUB 490:GOTO
450
400 M(MX,MY)=-T:LOCATE X,Y
410 IF T THEN PRINT"#"; ELSE PRINT" ';
420 PSET(MX+40,MY+30,-T)
430 GOTO 290
440 REM ***** END OF LOOP*****
450 IF H=256 THEN 600
460 C=C+1:CLS
470 GOTO 180
480 REM ***** DATA POKE  *****
490 FOR X=0 TO 5
500 FOR Y=0 TO 7
510 M=M+M(X,Y)*2^Y
520 NEXT Y
530 POKE AD+C*6+X,M
540 M=0
550 NEXT X
560 FOR Q=0 TO 5:FOR R=0 TO 7:M(Q,R)=0
570 NEXT R,Q
580 RETURN
590 REM ***** LISTING *****
600 CLS:PRINT "DEFINED CHARACTER(131-
255)"
610 FOR I=131 TO 255
650 PRINT CHR$(I);" ";:NEXT
660 PRINT"
BSAVE(Y/N)?";:Y$=INPUT$(1):PRINT Y$;
670 IF Y$="Y" OR Y$="y" THEN LOCATE
15,8:INPUT" FILE NAME";N$ ELSE END
680 REM ***** FILE SAVE *****
690 BSAVE N$,61585!,750
700 END
```

9.3 MUSIC PROGRAM

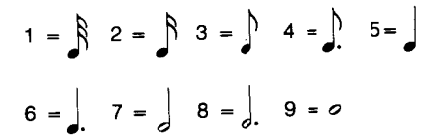
The SOUND instruction in N82-BASIC can be used to create sophisticated music compositions consisting of simple half notes. The number 1 parameter determines the precise musical step. The SOUND instruction will also work qute effectively n programs where a composition is to be performed. The program that follows is exclusively for musical composition.

The keyboard of the PC-8300 is turned into an actual keyboard of a musica instrument in terms of input. This keyboard input is organized in the following order:

a) Length of note (the "L" key + a length designation between 1 and 9 with a default of "5");

b) Octave (the "O" key + an octave designation between 1 and 4 with a default of "2");

c) Note the keys "Z", "X", "C", "V", "B", "N", and "M" on the keyboard correspond to the whole notes "do", "re", "mi", "fa", "so", "la", and "ti" in the key of C, while keys "S", "D", "G", "H" and "J" located obliquely above the first group on the keyboard correspond to half notes. The designated length of a note consists of the following. A rest is input by the space bar.



The length of a note and the octave can be omitted if these are not to be modified because they will automatically be set at the default values indicated above. A single note at a time can be modifed by using escape sequences.

It is useful practice to press the "E" key after every 20 or so notes have been input because this will cause an immediate review of those input notes and will define that series of notes as a 'Part' before a prompt is displayed inquiring whether you want to redo or save that series of notes.

If you dislike what you heard during the playback review, the entire series can be discarded and you can begin again. The input will be displayed on the screen as capital letters "A" through "G" the sharps displayed as lower case letters that correspond to "1a" through "so" (n the key of C). The input process can be stopped at any time by pressng the "Q" key.

The program can be executed after the data has been input at any time that you desire, once this data has been converted into a file. Tempo and transposition functions are also available during playback. You simply have to follow carefully the instructions in the program.

If you desire to compose longer compositions, useful modifications can be made to the input and editing methods by manipulating the data as string arrays (the original data) and numerical arrays (data for the performance of a composition). In addition, the structure of the original data itself can be directly rewritten while that data is open to editing in the TEXT mode.

```
10 REM COPYRIGHT(C)  1983 NEC
20 REM The explanations and prompts are
changed on 6. 5. '86 (lines 390, 580,
1380, 1420, 1470, 1480, 1490, 1500,
1520, 1530, 1540, 1560, 1590, 1610,
1620).
100 REM *** MUSIC ***
110 CLEAR 2000!:MAXFILES=1
120 DEFINT A-T:DEFSNG U-Y:DEFDBL Z
130 DIM A(48),M$(49),S(999),L(999)
140 SCREEN 0,0:Z=9394#
150 FOR I=0 TO 47
160 A(I)=Z:Z=Z/1.0594639#
170 NEXT I
180 FOR I=1 TO 9:READ LN(I):NEXT
190 DATA 4,8,16,24,32,48,64,96,128
200 REM *** MENU ***
210 CLS:PRINT" *** MUSIC ***
220 PRINT:PRINT" --- Play or Input ---"
230 PRINT:INPUT"(P/I)";Y$
240 IF Y$="P" OR Y$="p" THEN 280
250 IF Y$="I" OR Y$="i" THEN 790
260 PRINT"????":BEEP:BEEP:GOTO 210
270 REM *** PLAY ***
280 CLS:PRINT" --- PLAYER ---"
290 PRINT:PRINT"Type music data."
300 INPUT"File name ";N$
310 OPEN N$ FOR INPUT AS #1
320 S=0:E=0
330 IF EOF(1) THEN 360
340 LINEINPUT #1,M$(E)
350 E=E+1:GOTO 330
360 CLOSE:PRINT"End of load."
370 PRINT"Data conversion."
380 PRINT"You may transpose for music
from O1G to O4F."
390 PRINT"You may change the tempo (but
L1=4)."
400 INPUT"Do you want to change the key
(Y/N)";I$
410 IF I$="Y" OR I$="y" THEN GOTO 430
420 IF I$="N" OR I$="n" THEN GOTO 430
ELSE BEEP: CLS:GOTO 360
```

```
430 INPUT"Do you want to change the
tempo (Y/N)";Y$
440 IF Y$='Y" OR Y$="y" THEN GOTO 460
450 IF Y$='N" OR Y$="n" THEN GOTO 460
ELSE BEEP: CLS:GOTO 360
460 IF I$<>"Y" AND I$<>"y" THEN 500
470 INPUT"Change transposition from -7
to 7 ";D:IF D<-7 OR D>7 THEN 470
480 IF D>0 THEN FOR I=0 TO
41:A(I)=A(I+D):NEXT:GOTO 500
490 FOR I=47 TO 7 STEP -
1:A(I)=A(I+D):NEXT
500 IF Y$="Y" OR Y$="y" THEN INPUT"V
(From .25 to 2)";V ELSE V=1
510 PRINT" --- Wait for a moment please
---"
520 C=0:FOR I=0 TO E-1
530 T$=M$(I):GOSUB 690
540 NEXT I
550 BEEP:CLS
560 PRINT N$;" End of change data."
570 LOCATE 10,3:PRINT N$:LOCATE 10,4
580 PRINT"Press any key."
590 IF INKEY$<>"" THEN 590
600 IF INKEY$="" THEN 600
610 LOCATE 10,4:PRINT SPACE$(14)
620 FOR I=0 TO C-1:SOUND
S(I),L(I)*V:NEXT I
630 INPUT"Once more (Y/N)";Y$
640 IF Y$="Y" OR Y$="y" THEN 570
650 IF Y$="N" OR Y$="n" THEN GOTO 650
ELSE BEEP: CLS: GOTO 560
660 IF I$="Y" OR I$="y" THEN PRINT"I
must reinitialize.":RUN
670 GOTO 210
680 REM *** DATA COMPILER ***
690 FOR T=1 TO LEN(T$)
700 N=INSTR("CcDdEFfGgAaB
L0",MID$(T$,T,1))
710 IF N>13 THEN GOSUB 750:GOTO 700
720 M=N+M:S(C)=A(M-1):L(C)=L:M=M-N
730 IF N=13 THEN S(C)=0
740 C=C+1:NEXT T:RETURN
750 IF N=15 THEN
M=12*(VAL(MID$(T$,T+1,1))-
1):T=T+2:RETURN
760 L=VAL(MID$(T$,T+1,1)):L=LN(L)
770 T=T+2:RETURN
780 REM *** INPUT ***
790 CLS:PRINT" --- INPUT ---"
800 S=0:E=0:C=0
```

```
810 INPUT"Append or New data (A/N)";Y$
820 IF Y$="N" OR Y$="n" THEN GOTO 840
830 IF Y$="A" OR Y$="a" THEN GOTO 840
ELSE  BEEP: CLS: GOTO 800
840 INPUT"File name ";N$
850 IF Y$="A" OR Y$="a" THEN OPEN N$ FOR
APPEND AS #1 ELSE 880
860 PRINT"Please input to continue
data.":GOTO 900
870 REM *** NEW DATA ***
880 OPEN N$ FOR OUTPUT AS #1
890 PRINT"Please input new music data."
900 INPUT"Do you want to see the input
explanation (Y/N)";Y$
910 IF Y$="Y" OR Y$="y" THEN GOSUB 1460
920 IF Y$="N" OR Y$="n" THEN GOTO 930
ELSE BEEP : CLS: GOTO 890
930 REM
940
CLS:L$="L5":O$="O2":S=C:M$(E)="":B=0:T$=
"":F=1:L=32
950 LOCATE 0,0:PRINT L$
960 LOCATE 3,0:PRINT O$
970 LOCATE 6,0:I$=INPUT$(1)
980 P=INSTR("ZSXDCVGBHNJM
LOE"+CHR$(27)+"Q",I$)
990 IF P=0 THEN 970
1000 I$=MID$("CcDdEFfGgAaB ",P,1)
1010 IF F=1 THEN T$=L$+O$+I$
1020 IF F=2 THEN T$=O$+I$
1030 IF F=3 THEN T$=L$+I$
1040 IF F=0 THEN T$=I$
1050 IF B=0 THEN T$=L$+O$+I$
1060 IF P=17 THEN IF F<>0 OR B=0 THEN
950 ELSE B=0:GOTO 1290
1070 IF P=18 THEN IF S=C THEN E=E-1:GOTO
1320 ELSE 1320
1080 IF P>13 THEN 1140
1090 X$=T$:B=1
1100 PRINT I$;:M$(E)=M$(E)+T$
1110 LOCATE 0,5:PRINT M$(E)+SPACE$(10);
1120 GOSUB 690:SOUND S(C-1),L(C-1):F=0
1130 GOTO 950
1140 ON P-13 GOTO 1150,1180,1210
1150 IF S=C THEN F=1 ELSE IF F=2 THEN
F=1 ELSE F=3
1160 LOCATE
0,0:Y$=INPUT$(1):P=INSTR("123456789",Y$)
:IF P=0 THEN 1150
1170 L$="L"+Y$:GOTO 950
```

```
1180 LOCATE
3,0:Y$=INPUT$(1):P=INSTR("1234",Y$):IF
P=0 THEN 1180
1190 IF S=C THEN F=1 ELSE IF F=3 THEN
F=1 ELSE F=2
1200 O$="O"+Y$:GOTO 950
1210 LOCATE 0,3:PRINT "End of part";E;
1220 FOR I=S TO C-1:SOUND S(I),L(I):NEXT
1230 INPUT"Ok(Y/N)";Y$:IF Y$="Y" THEN
1270
1240 IF Y$="N" OR Y$="n" THEN GOTO 1260
ELSE BEEP: CLS: GOTO 1210
1250 IF Y$="Y" OR Y$="y" THEN GOTO 1260
ELSE BEEP: CLS :GOTO 1210
1260 C=S:PRINT"Try again.":BEEP:GOTO 940
1270 S=C:IF E<49 THEN
E=E+1:M$(E)="":F=1:B=0:CLS:GOTO 950
1280 BEEP:PRINT"Out of data space.":GOTO
1350
1290 M$(E)=LEFT$(M$(E),LEN(M$(E))-
LEN(X$))
1300 C=C-1:BEEP:LOCATE 0,3:PRINT"1 step
back.":BEEP
1310 LOCATE 0,3:PRINT SPACE$(12);:GOTO
950
1320 PRINT:PRINT"End of music."
1330 C=C+1
1340 REM *** END ***
1350 PRINT"Your music.":FOR I=0 TO
200:NEXT
1360 FOR I=0 TO C-2:SOUND S(I),L(I):NEXT
1370 CLS:PRINT"Save to start."
1380 PRINT"File name
";N$;".":PRINT"Press any key."
1390 IF INKEY$="" THEN 1390
1400 FOR I=0 TO E:PRINT #1,M$(I):NEXT I
1410 CLOSE:BEEP
1420 PRINT"End of save. Press any key."
1430 IF INKEY$="" THEN 1430
1440 GOTO 210
1450 REM *** EXPLAIN ***
1460 PRINT "   EXPLANATIONS "
1470 PRINT"1 Please press 'CAPS' key."
1480 PRINT"2 'ZSXDCVGBHNJM' keys are
music keyboard."
1490 PRINT"3 'ZSXDCVGBHNJM' keys changed
'CcDdEFfGgAaB'keys."
1500 LOCATE 0,7:PRINT"  Press any key.";
1510 IF INKEY$="" THEN 1510
```

```
1520 PRINT:PRINT"4 Press 'E' key to end
one block."
1530 PRINT"5 Pressing 'Q' key ends data
input."
1540 PRINT"6 Press 'ESC' key to reenter
last note."
1550 PRINT"7 Space is a rest."
1560 LOCATE 0,7:PRINT"  Press any key.";
1570 IF INKEY$="" THEN 1570
1580 PRINT:PRINT"8 L=LENGTH(1-
9),O=OCTAVE(1-4)"
1590 PRINT"9 Press 'E' key after
inputting about 20 keys to go to next
step."
1600 PRINT"10 Maximum 49 parts to be
input."
1610 PRINT"11 You can change 'L' and 'O'
keys as many time as you want, if you do
not press 'ESC' key."
1620 LOCATE 0,7:PRINT"  Press any key.";
1630 IF INKEY$="" THEN 1630
1640 RETURN 930
```

9.4 RANDOM DISPLAY PRINTING PROGRAM

Data that is placed in an array can be easily used for calculation or for display. If data is properly combined with the RND function the results are very interesting. It is even possible to intergate this type of process with the Character Definition program irtroduced previously.

Please use any alphabetical or numerical characers when you run the program.

```
10 '                    DEMO
20 SCREEN 0,0:CLS
30 DIM C%(39,7),X%(319,1):C=0
40 PRINT "READING DATA"
50 FOR X=0 TO 39
60 FOR Y=0 TO 7
70 X%(Y*40+X,0)=X:X%(Y*40+X,1)=Y
80 READ C%(X,Y)
90 NEXT Y,X
100 '                  MAKE DATA
110 SCREEN 0,0:CLS:PRINT
120 PRINT "DATA SCRAMBLING"
130 FOR I=0 TO 200
140 R=RND(1)*319
150 R1=RND(1)*319
160 N=X%(R,0):X%(R,0)=X%(R1,0):X%(R1,0)=
N
170 N=X%(R,1):X%(R,1)=X%(R1,1):X%(R1,1)=
N
180 NEXT
190 '                    PRINT
200 BEEP:CLS:PRINT CHR$(27)+"V"
210 PRINT "TYPE ANY KEY";:A$=INPUT$(1)
220 PRINT A$:PRINT
230 PRINT "TYPE ANOTHER KEY";:B$=INPUT$(
1)
240 PRINT B$:CLS
250 FOR N=0 TO 319
260 X=X%(N,0):Y=X%(N,1)
270 SOUND X*200+200,3
280 LOCATE X,Y
290 IF C%(X,Y)=1 THEN PRINT A$; ELSE PRI
NT B$;
300 NEXT
310 BEEP: LOCATE 0,0:PRINT A$; ELSE PRIN
T B$;
320 FOR I=0 TO 500:NEXT
330 LOCATE 0,0:GOTO 120
```

```
340 DATA 0,0,0,0,0,0,0,0,0,1,1,1,1,1,1
,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0
350 DATA 0,1,0,0,0,1,0,0,0,0,1,0,1,0,0,0
,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
360 DATA 0,0,0,1,1,1,1,0,0,0,0,1,0,0,0,1,0
,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1
370 DATA 0,1,0,0,0,0,0,1,0,0,1,0,0,0,1,0
,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
380 DATA 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
,0,0,1,1,0,1,1,0,0,1,0,0,1,0,0,1
390 DATA 0,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1
,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0
400 DATA 0,0,1,0,0,0,1,1,0,1,0,0,0,1,0,1
,0,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1
410 DATA 0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0
,0,0,1,1,1,1,1,0,0,1,0,0,0,0,0,1
420 DATA 0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1
,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0
430 DATA 0,0,1,0,0,0,0,1,0,1,1,1,1,1,1,1
,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
```

9.5 GAME PROGRAM

The missle base is moved by using the left and right cursor movement keys. Pressing the space bar shoots a missile. As presently set, the game will end after one minute, but play can easily be extenced by simply modifying the TIME$ function in line 130.

```
10 '       GAME
20 DEFINT A-Z
30 SCREEN 0,0:CLS
40 TIME$="00:00:00"
50 SC=0
60 '       START
70 X=RND(1)*35+1
80 LOCATE X,0:PRINT " >O< ";
90 I$=INKEY$
100 IF I$=CHR$(28) THEN M=M+1
110 IF I$=CHR$(29) THEN M=M-1
120 IF I$=" " THEN GOSUB 230
130 IF TIME$>"00:01:00" THEN 460
140 IF M<0 THEN M=37:LOCATE 0,6:PRINT "
" ;
150 IF M>38 THEN M=1:LOCATE 38,6:PRINT "
 ";
160 LOCATE M,6:PRINT " M " ;
170 LOCATE 2,7:PRINT TIME$;
180 LOCATE 18,7:PRINT SC;"POINTS";
190 P=INT(RND(1)*3)-1:X=X+P
200 IF X<1 THEN X=1
210 IF X>35 THEN X=35
220 GOTO 80
230 '          MISSILE SUB
240 FOR Y=6 TO 0 STEP -1
250 LOCATE M+1, Y:PRINT "!";
260 SOUND Y*1000+1000,1
270 LOCATE M+1,Y:PRINT " " ;
280 NEXT
290 IF M=X OR M=X+2 THEN SC=SC+1:BEEP:GO
SUB 330:RETURN 70
300 IF M=X+1 THEN GOSUB 390
310 RETURN
320 '          MISS
330 FOR I=0 TO 10
340 LOCATE X,0:PRINT "OOPS!"
350 FOR J=0 TO 20:NEXT:LOCATE X,0:PRINT
" ";
360 SOUND 16000,1:NEXT
370 RETURN
380 '       SOLID HIT
390 SC=SC+5:SOUND 440,10
```

```
400 FOR I=0 TO 10
410 LOCATE X-1,0:PRINT "HOLIDAY!"
420 SOUND 1760,1
430 NEXT I
440 LOCATE X-1,0:PRINT "         "
450 RETURN
460 LOCATE 10,4:PRINT "END OF GAME":END
```

9.6 SCORE RANKING PROGRAM

This program uses the sequential file management function of N82-BASIC, in order to manipulate results, scores, ranks, etc. It can be used in a variety of applications if the kinds of items and number of items are appropriately adjusted to suit the specific requirements.

```
10 SCREEN 0,0:CLS
20 PRINT "*** RANKING SCORES ***"
30 PRINT
40 PRINT "PLEASE INPUT SCORE TITLE "
50 PRINT,":";
60 LINE INPUT TI$
70 PRINT
80 INPUT "NUMBER OF ITEM    ";NC
90 INPUT "NUMBER OF PERSONS";NR
100 DIM D(NC,NR),IT$(NC),NA$(NR),RSUM(NR),RMEAN(NR)
, SUM(NC),SSM(NC),MEAN(NC),SD(NC)
110 CLS
120 PRINT "NAME OF ITEMS:"
130 FOR I=1 TO NC
140   LOCATE 0,2:PRINT SPACE$(40)
150   LOCATE 0,2:PRINT "NAME OF ITEM";I;
160   INPUT ITM$(I)
170 NEXT
180 CLS
190 PRINT "INPUT THE DATA "
200 FOR J=1 TO NR
210   LOCATE 0,2:PRINT SPACE$(20)
220   LOCATE 0,2:PRINT "NO.";J;"NAME";
230   INPUT NA$(J)
240   FOR I=1 TO NC
250     LOCATE 0,4:PRINT SPACE$(40)
260     LOCATE 0,4:PRINT ITM$(I);" POINTS";
270     INPUT DA
280     D(I,J)=DA:RSUM(J)=RSUM(J)+DA
290     SUM(I)=SUM(I)+DA
300     SSM(I)=SSM(I)+DA^2
310   NEXT I
320   LOCATE 0,4:PRINT SPACE$(40)
330   RMEAN(J)=RSUM(J)/NC
340 NEXT J
350 FOR I=1 TO NC
360   MEAN(I)=SUM(I)/NR
370   SD(I)=SSM(I)/NR-MEAN(I)^2
380 NEXT I
390 '               OUTPUT
400 PRINT "PLEASE PRESS THE SPACE BAR TO HOLT SCROL
LING."
```

```
410 OPEN "LCD:" FOR OUTPUT AS #1
420 FOR I=0 TO 1000:NEXT:BEEP:CLS
430 TT=200:GOSUB 600
440 CLOSE #1:PRINT
450 PRINT "DO YOU WANT TO CREATE A FILE (Y/N)";
460 Y$=INPUT$(1):PRINT Y$:IF Y$<>"Y" AND Y$<>"y" TH
EN 540
470 ON ERROR GOTO 540
480 INPUT "NAME OF FILE";A$
490 OPEN A$ FOR OUTPUT AS #1
500 ON ERROR GOTO 0
510 TT=0:GOSUB 600
520 CLOSE #1
530 PRINT
540 PRINT "DO YOU WANT TO PRINT IT (Y/N)";
550 Y$=INPUT$(1):PRINT Y$:IF Y$<>"Y" AND Y$<>"y" TH
EN END
560 OPEN "LPT:" FOR OUTPUT AS #1
570 TT=0:GOSUB 600
580 CLOSE #1:END
590 RESUME 480
600 '                OUTPUT SUBROUTINE
610 PRINT #1,SPACE$(12);LEFT$(TI$,30)
620 PRINT #1,
630 PRINT #1,SPACE$(9);
640 FOR I=1 TO NC
650   PRINT #1,LEFT$(ITEM$(I)+SPACE$(12)
,12);
660 NEXT I
670 PRINT #1,"TOTAL    MEAN"
680 FOR J=1 TO NR
690   PRINT #1,LEFT$(NA$(J)+SPACE$(10),1
0);
700   FOR I=1 TO NC
710     PRINT #1,USING"#####        ";D(I
,J);
720   NEXT I
730   PRINT #1,USING"#### ####.#";RSUM(J
);RMEAN(J)
740   IF TT<>0 THEN IF INKEY$=" " THEN A
$=INPUT$(1)
750   FOR T=0 TO TI:NEXT
760 NEXT J
770 PRINT #1,
780 PRINT #1,"TOTAL"
790 PRINT #1,"POINTS  ";
800 FOR I=1 TO NC
810   PRINT #1,USING "#######      ";SUM(
I);
820 NEXT
830 PRINT #1,
```

```
840 PRINT #1,"MEAN     ";
850 FOR I=1 TO NC
860   PRINT #1,USING "#######      ";MEAN
(I);
870 NEXT
880 PRINT #1,
890 PRINT #1,"DEVIATION";
900 FOR I=1 TO NC
910   PRINT #1,USING "######.#     ";SQR(
SD(I));
920 NEXT
930 PRINT #1,
940 RETURN
```

Student Achievement

	Mathematics	English	History	TOTAL	MEAN
John	71	78	73	222	74.0
Tom	53	78	80	211	70.3
Mike	83	62	80	225	75.0
Ken	78	91	54	223	74.3
Bob	73	46	43	162	54.0
Ed	100	90	65	255	85.0
Frank	68	89	65	222	74.0
Ann	90	78	75	243	81.0
TOTAL POINTS	616	612	535		
MEAN	77	77	67		
DEVIATION	13.4	14.6	12.2		

APPENDICES

TABLE OF CONTENTS

A TABLES

A.1 RESERVED WORDS

ABS
AND
ASC
ATN
BEEP
BLOAD
BLOAD?
BSAVE
CDBL
CHR$
CINT
CLEAR
CLOAD
CLOAD?
CLOSE
CLS
COM
CONT
COS
CSAVE
CSNG
CSRLIN
DATA
DATE$
DEFINT
DEFDBL
DEFSNG
DEFSTR
DIM
EDIT
ELSE
END
EOF
EQV
ERL
ERR
ERROR
EXEC
EXP
FILES
FIX
FOR
FRE
GOSUB
GOTO
IF
IMP
INKEY$
INP
INPUT
INPUT$
INPUT#
INSTR
INT
KEY
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOCATE
LOG
LPOS
LPRINT
MAXFILES
MENU
MERGE
MID$
MOD
MOTOR
NAME
NEW
NEXT
NOT
OFF
ON
OPEN
OR
OUT
PEEK
POKE
POS
POWER
PRESET

PRINT
PSET
READ
REM
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
RUN
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SQR
STEP
STOP
STR$
STRING$
TAB
TAN
THEN
TIME$
TO
USING
VAL
XOR

A.2 ERROR CODES

Error Message	Code	N-BASIC Message	Meaning
?AO Error	53	File Already Open	That file is already open
?BN Error	51	Bad file Number	The file number is incorrect.
?BO Error	23	Communication buffer overflow (Buffer Overflow)	The input buffer has overflowed.
?BS Error	9	Subscript out of range (Bad Subscript)	Array subscript is incorrect.
?CF Error	58	File not open (Closed File)	The file is not yet open.
?CN Error	17	Can't ContiNue	Program execution cannot be resumed by means of a CONT command.
?DD Error	10	Duplicate Definition	The same array has been declared twice.
?DS Error	56	Direct Statement in file	An ASCII format won't load.
?DU Error	25	Device Unavailable	The designated device is not accessible.
?EF Error	54	Input past end (End of File)	No more data in the file.
?FC Error	5	Illegal Function Call	Attempts to use Commands or Functions are incorrect.
?FF Error	52	File not Found	The designated file cannot be located.
?FL Error	57	Filing Limit	There are too many files
?ID Error	12	Illegal Direct	The specified command cannot be used in the direct mode.
?IE Error	50	Internal Error	An error within BASIC.
?IO Error	24	I/O error	An error during input or output.
?LS Error	15	String too long (Long String)	Over 255 characters in a string variable.

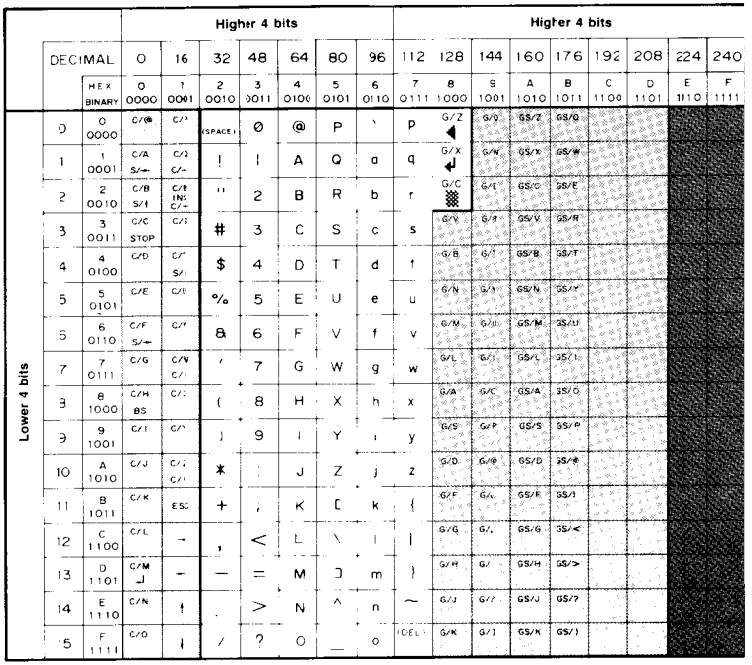
Error Message	Code	N-BASIC Message	Meaning
?MO Error	22	Missing Operand	A required parameter is missing
?NF Error	1	NEXT without FOR	There is no FOR statement to match the NEXT statement.
?NM Error	55	Bad file name (File Name Mismatch)	The name of the file is inappropriate for the operation attempted.
?NR Error	19	No RESUME	There is no RESUME statement present in an error processing routine.
?OD Error	4	Out of Data	There is no more data.
?OM Error	7	Out of Memory	There is not enough memory.
?OS Error	14	Out of String space	The memory region available for string storage is inadequate.
?OV Error	6	Overflow	A numeric value is too big.
?PC Error	59	PC-8001A Command	This command is for use only on the PC-8001A.
?RG Error	3	RETURN without GOSUB	A RETURN statement is present without a matching GOSUB statement.
?RW Error	20	RESUME without Error	A RESUME is met before an error processing routine is entered.
?SN Error	2	SyNtax error	The grammar of a statement is incorrect.
?ST Error	16	String formula Too complex	The string formula is too complex.
?TM Error	13	Type Mismatch	The types of variables and integers are inconsistent.
?UE Error	21	Unprintable Error	An error that has not been designated in a message occurred.
?UF Error	18	Undefined user Function	An undefined user function has been read.

Error Message	Code	N-BASIC Message	Meaning
?UL Error	8	Undefined Line number	A designated line has not been defined.
?/0 Error	11	Division by Zero	A division by 0 is attempted.

A.3 CONTROL CODES

OPERATION	CHARACTER CODE	FUNCTION
CTRL C or STOP	3	Interrupts program execution
CTRL E	5	Deletes after the cursor position to the end of the file
CTRL G	7	Sounds bell
CTRL H or DEL BS	8	Deletes one character to the left of the cursor
CTRL I or TAB	9	Moves the cursor to the next tab setting
CTRL K	11	Moves the cursor to the home position
CTRL L	12	Clears the screen
CTRL M or ↵	13	Moves the cursor to the biginning of a new line
CTRL N	14	Shift OUT
CTRL O	15	Shift IN
CTRL Q	17	Authorizes reopening on transmissions (XON)
CTRL S	19	Requests an interrupt of transmissions (XOFF)
ESC	27	Begins an ESCape sequence
◁	28	Moves the cursor one character to the right
▷	29	Moves the cursor one character to the left
▽	30	Moves the cursor up one line
△	31	Moves the cursor down one line

A.4 CHARACTER CODES



Lower 4 bits	DECIMAL	HEX / BINARY	Higher 4 bits: 0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
			0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0	0	0 0000	C/@	C/^	(SPACE)	0	@	P	`	p	G/Z ◀	G/0	GS/Z	GS/@				
1	1	1 0001	C/A S/←	C/a C/←	!	1	A	Q	a	q	G/X ↵	G/W	GS/X	GS/W				
2	2	2 0010	C/B S/↑	C/b INS C/→	"	2	B	R	b	r	G/C ▩	G/E	GS/C	GS/E				
3	3	3 0011	C/C STOP	C/c	#	3	C	S	c	s	G/V	G/R	GS/V	GS/R				
4	4	4 0100	C/D	C/d S/↓	$	4	D	T	d	t	G/B	G/T	GS/B	GS/T				
5	5	5 0101	C/E	C/e	%	5	E	U	e	u	G/N	G/Y	GS/N	GS/Y				
6	6	6 0110	C/F S/→	C/f	&	6	F	V	f	v	G/M	G/U	GS/M	GS/U				
7	7	7 0111	C/G	C/N C/↓	'	7	G	W	g	w	G/L	G/I	GS/L	GS/I				
8	8	8 1000	C/H BS	C/↑	(	8	H	X	h	x	G/A	G/C	GS/A	GS/O				
9	9	9 1001	C/I	C/^	)	9	I	Y	i	y	G/S	G/P	GS/S	GS/P				
10	A	A 1010	C/J	C/; C/↓	*	:	J	Z	j	z	G/D	G/@	GS/D	GS/@				
11	B	B 1011	C/K	ESC	+	;	K	[	k	{	G/F	G/,	GS/F	GS/1				
12	C	C 1100	C/L	→	,	<	L	\	l	|	G/G	G/,	GS/G	GS/<				
13	D	D 1101	C/M ↵	←	−	=	M	]	m	}	G/H	G/.	GS/H	GS/>				
14	E	E 1110	C/N	↑	.	>	N	^	n	~	G/J	G/;	GS/J	GS/?				
15	F	F 1111	C/O	↓	/	?	O	_	o	DEL	G/K	G/1	GS/K	GS/1				

Higher 4 bits **Decimal** **Hexadecimal**
Binary **Lower 4 bits**

Notes: 1. C/r means hold CTRL while pressing "r"
2. S/r means hold SHIFT while pressing "r"
3. G/r means hold GRPH while pressing "r"
4. GS/r means hold SHIFT and GRPH while pressing "r"

Example: To find the character code of "A", add 1 to DECimal 64, or to HEX 40.
Thus the character code of "A" is 65 DEC or 41 HEX (01000001 BINARY).

Code:

- 00H — 1FH: Unique code that cannot be output as characters (See p.175 Control Codes)
- 83H — DFH: User-defined characters (Can be input from the keyboard)
- E0H — FFH: User-defined characters (Can be output by using the CHR$ function)

B MEMORY MAPS

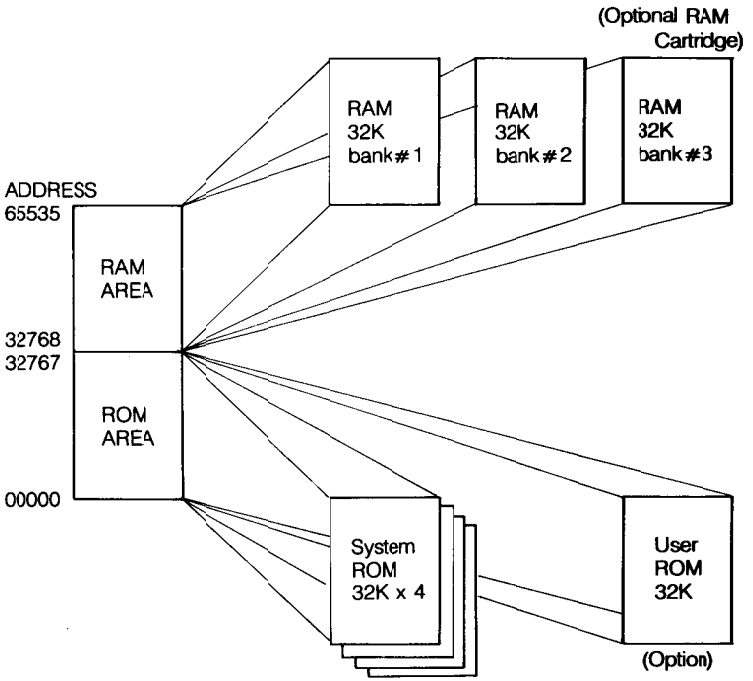
The PC-8300's 8-bit CPU (80C85) can access 64K bytes of memory at a time. The address is allocated from address 0 to address 65535 (FFFF hexadecimal).

The lower half 32K bytes (address 0 to 32767) are assigned to the ROM area, and the higher half 32K bytes (address 32768 to 65535) are for RAM.

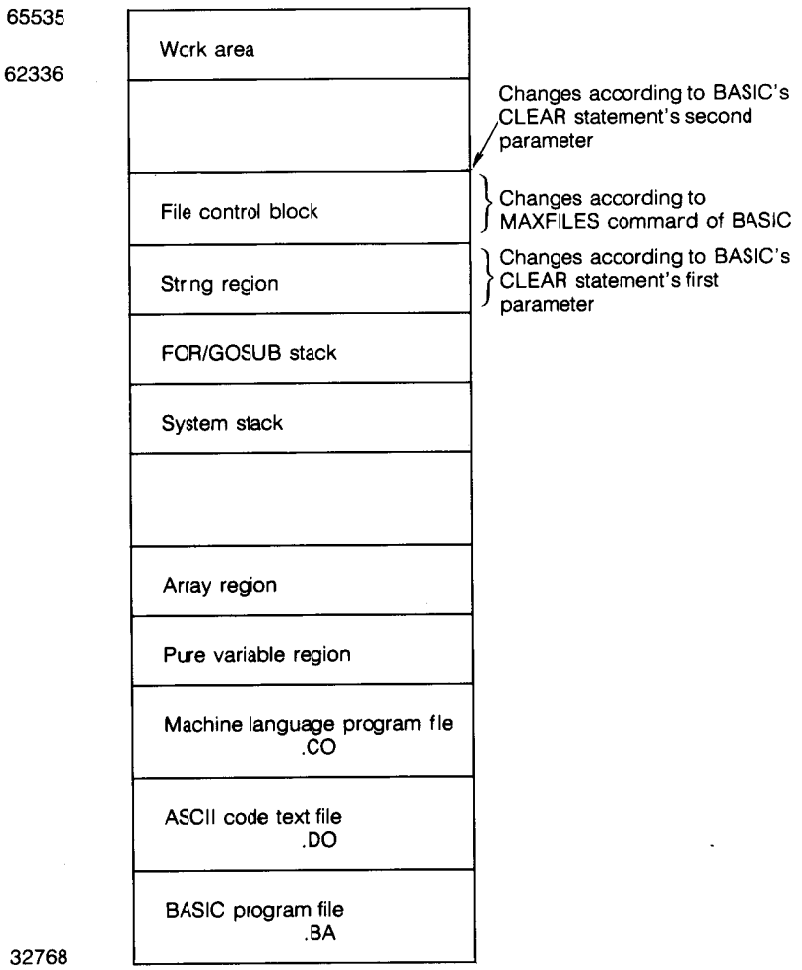
You have 32K × 4 byte ROM in the standard PC-8300. The PC-8300's software (MENU, BASIC, TEXT, and TELCOM) is in two of these 32K byte ROM areas, and remainder is reserved for the PC-8300 system. The software automatically switches between these two banks of ROM.

You can also install optional user ROM into the PC-8300.

The standard PC-8300 has two banks of RAM, selectable by the MENU's BANK command. By using optional RAM Cartridge, a third RAM bank is available.



Each RAM bank is as explained in the figure below:



C ESCAPE SEQUENCES

ESC +	CHARACTER CODE	FUNCTION
A	27,65	Moves the cursor one line up
B	27,66	Moves the cursor one one line down
C	27,67	Moves the cursor one character (one column) to the right
D	27,68	Moves the cursor one character (one column) to the left
E	27,29	Clears screen and moves the cursor to the top left corner of the screen (the home position)
J	27,74	Erases characters from the cursor position to the end of the display
K	27,75	Erases characters from the cursor position to the right end of the current line
L	27,76	Insert a line
M	27,77	Deletes the line where the cursor is located
T	27,84	Displays Function Keys
U	27,85	Erases Function Keys display
V	27,86	Inhibits scrolling (freezes the display)
W	27,87	Scrolling is permitted
Y<y><x>	*	Moves the cursor to the designated location
j	27,106	Clears the screen
p	27,112	Changes the screen to reverse display
q	27,113	Restores display to normal (back from reverse display)

* ESC + Y<y><x>
The cursor position is cesignated by the vertical and horizontal coordinates (y and x) respectively.

Characters and capital letters beginning at ASCII code 32 are used in the coordinate designation. A space corresponds to 0, the excaimation point (!) corresponds to 1, etc. Refer to an ASCII code chart for the complete list.

To move the cursor to the home position (coordinate 0,0), for example, you would input the following string:

ESC Y SPACE SPACE

D GLOSSARY

ABSOLUTE VALUE The unsigned form of any given number.

ARRAY A set of values arranged in a regular pattern such as in single file or in two dimensions.

ASCII American Standard Code for Information Interchange.

BASIC Beginner's All-purpose Symbolic Instruction Code. Easy-to-understand programming language.

BOOLEAN A type of algebra which deals with binary mathematics.

CONDITIONAL A statement that requires a test to be made. An IF statement is a conditional statement since the computer will take one of two paths, depending on the outcome of the IF statement.

COSINE In a right triangle, the value obtained when the side adjacent to an angle is divided by the hypotenuse.

DATA Information such as numbers, names, etc., that a computer must have in order to solve a given problem, or do a given job.

DELIMIT Separate.

DIMENSION The number of elements in an array and their configuration (one or more dimensions).

EXPONENTIATION Raising a number to some power.

EXPRESSION In an assignment statement, the value to the right of the equal sign.

FILE A collection of data to be used with a computer program. The program itself is often called a file.

INCREMENT To increase the value of a counter.

INITIALIZATION Giving first values to a data name. In loops, counters are normally initialized to 1.

INPUT The values that a program must have in order to solve a given problem.

INTEGER A whole number.

LINE NUMBER An identifying number that is placed at the start of each BASIC statement in a program.

LOG (NATURAL) The number to which "e" must be raised in order to obtain a given value.

LOOP A set of statements that is executed over and over.

MEMORY A computer can store electronically several million characters of information at any given moment. In back-up devices, computers can store up to several trillion characters for relatively immediate use.

NULL Empty set or empty string: {}

OUTPUT The answers given by a computer program.

PROGRAM A set of instructions telling a computer how to solve a given problem or do a job. The instructions are given in a programming language such as BASIC.

READ To obtain data from a DATA statement.

RELATIONAL SYMBOLS The symbols >, =, and < that may be used to indicate whether one value is larger, smaller, equal, or not equal to another. Relational symbols are used in IF statements.

RAM Random Access Memory. The type of memory that can be altered, by means of saving files or new programs, or by running programs.

RETURN KEY A key on your terminal's keyboard that is used to enter a BASIC statement.

RESERVED WORD In BASIC, words that have special meanings and uses, such as commands.

ROM Read Only Memory. The type of memory that stays intact even when the PC-8300's power is turned off.

SEARCH The finding of a particular value in an array table.

SINE In a right triangle, the value obtained when the side opposite the angle is divided by the hypotenuse.

SQUARE ROOT The number which, when multiplied by itself gives the specified value. Thus, the square root of 64 is 8.

STATEMENT A sirgle instruction to the computer such as: 10 LET P = 7

SUBSCRIPT A number, name, or expression that tells which one element of an array is specified.

SYSTEM COMMAND A command directly to the computer telling it to do something with a program you have created or wish to create. Some system commands are SAVE, LIST, RUN, NEW.

TANGENT In a right triangle, the value obtained when the side opposte the angle is divded by the side adjacent to the angle.

TEST To check such as the value of a counter, a condition, a program, etc.

TRUNCATE Drop the decimal digits of a number (round off).

ZONE An Area.

INDEX

F

G

H

I

K

L