PC-8201A

N82-BASIC

REFERENCE MANUAL

NEC

PC-8201A-RM
PTS-211

N82-BASIC

REFERENCE MANUAL

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION

The N82-BASIC Reference manual is a guide to the programming language used for the PC-8201 personal computer. Microsoft TM 's N82-BASIC language, developed specifically for the PC-8201 offers a wide range of commands and functions, making it very useful and versatile.

This Reference Manual was designed for anyone, from beginning to professional programmers. It is intended to be used in conjunction with the PC-8201 User's Guide.

This Manual is divided into ten 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 symbols used for programming. A description of the PC-8201 LCD screen display is included.

Chapter 3 explains how programming expressions are formed specifically 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 contains the causes and what action should be taken when error messages occur.

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

Chapter 10 includes the Appendices, offering quick reference tables and guides, memory maps, etc.

The PC-8201 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-8201 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-8201, you should become familiar with the N82-BASIC language outlined in this Reference Manual.

It is best for beginning programmers to review this manual thoroughly and actually input sample programs with the PC-8201. More advanced programmers can use this manual as a reference.

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

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

FORMAT: Describes how an instruction is written. The following points apply to the format description for 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.

3. Parentheses ( ) are required to be typed in as shown in format.

   The three types of parameters:

   a. A line number – whose numbers are allowed.

   b. A string – enclosed by quotation marks. Combinations of letters and numbers are allowed.

   c. A variable – constants, numerical values, or numerical formulas are allowed.

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

5. Brackets [ ] denote that any one of the enclosed words must be chosen for use.

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

7. Items preceding the " . . . " symbol can be repeated any number of times as long as they do not go over the 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 essential.

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

DESCRIPTION: Explains important points for the method of use for 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 shared by the command or function being described.

SAMPLE PROGRAM: When included, this is a sample program for system commands, statements, and functions described.

SHIFT **+⟨ Character ⟩:**

Indicates that you should press and hold the SHIFT Key, then type the specified character. The + sign is not to be typed in.

CTRL **+⟨ Character ⟩:**

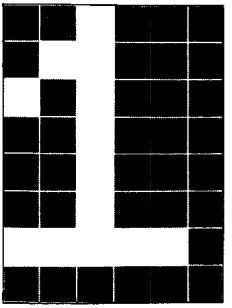
Indicates that you should press and hold the CTRL Key, then type the specified character. The + sign is not to be typed in.

Symbols used in this Reference Manual:

NOTES to be remembered.

REFERENCE is made to another chapter, to the PC-8201 User's Guide.

CAUTION is required when utilizing certain features of the N82-BASIC language.



N_{82}-BASIC Overview

CHAPTER 1

N82-BASIC Overview

N82-BASIC has been designed to fully utilize the many features of the PC-8201 personal computer. The language that is used is similar to many other forms of BASIC language. In certain ways, it differs since the hardware features of the PC-8201 are different than those on other computers.

All of the hardware and software features of the PC-8201 are related to the N82-BASIC language:

Internal and External Features

- Programmable Function Keys
- Real-time Clock
- Sound Generator
- Automatic Power Shut off
- Cassette recorder connector
- RS-232C interface connector
- Dot Matrix Printer connector
- Letter Quality Printer connector (same as above)
- Bar Code Reader connector
- SIO connectors
- Modem capability
- RAM Cartridges

For a computer of its size, the LCD screen of the PC-8201 can handle extremely high resolution graphics of 240 x 64 pixels (dots). Graphics capability is utilized through programs written in N82-BASIC.

You can easily create and modify (edit) BASIC programs using the PC-8201's powerful screen editor. You also have the option to write and edit programs while in the TEXT mode, and then load them into the BASIC mode of the PC-8201. This TEXT feature is quite powerful and versatile.

Use of the PC-8201 for computer-to-computer communication through a telephone modem is accomplished effectively. This is done by using the TELCOM software feature, along with BASIC operation instructions, such as ON COM GO SUB.

Large BASIC programs may be written with the PC-8201, since the memory is expandable to 96K bytes. The PC-8201 comes equipped with 16K bytes of RAM installed, with one memory bank available for use. Two other memory banks of 32K bytes each may be utilized if additional RAM chips or cartridges are installed in the unit.

The PC-8201 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.

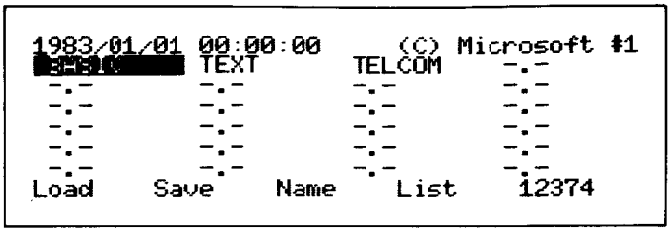
Battery power of the PC-8201 is conserved as efficiently as possible due to the Automatic Shut off feature. This feature is operated by the POWER instruction, which is proprammed into the PC-8201.

Data stored within the RAM of the PC-8201 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.

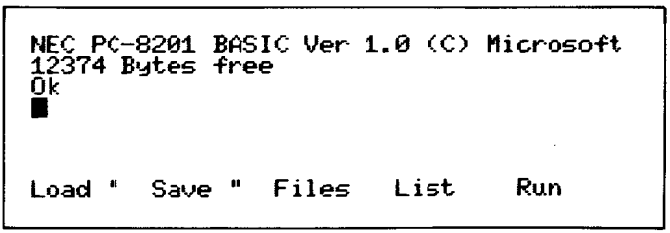
OPERATING MODES

The BASIC software feature of the PC-8201 has two operating modes, the Direct Mode and the Program Mode. These operating modes are used when you are in the BASIC mode of the PC-8201.

As described in the PC-8201 User's Guide, the BASIC mode is entered by moving the cursor onto the word BASIC on the LCD screen:



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



You can now utilize either the Direct Mode or the Program Mode of the BASIC feature.

DIRECT MODE

The Direct Mode of BASIC allows an individual program statement, written in the N82-BASIC language, to be executed. This is done by typing in the statement and then pressing the Key. The statements used in the Direct Mode do not have a line number, and they must conform to syntax requirements of the N82-BASIC language. The Direct Mode is useful for testing a particular statement. You can then see if the statement acts as you expect it to, or if it performs a function correctly, without running an entire program or set of statements.

The variable of a statement used in the Direct Mode is "held" in the memory temporarily, while you are working with them. They may be erased from the memory by typing NEW and then pressing the Key. These statements cannot by "SAVED" in the RAM or external devices for future use.

PROGRAM MODE

Statements used in the Program Mode must conform to command format requirements of N82-BASIC. The Program Mode is entered simply by placing a line number, such as 10, 20, or 30, directly to the left of a program statement.

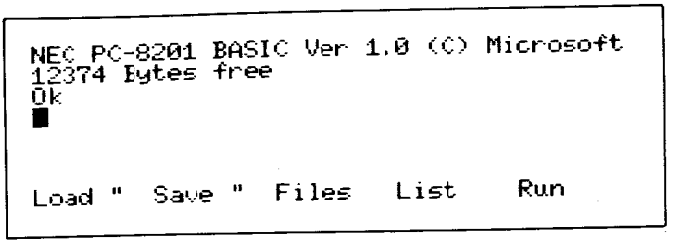
The line number and the statement can then be stored in the RAM. This means that the numbered statement is "held" in the working memory. This way, multiple statements can be written to create a program. This differs from statements in the Direct Mode because those unnumbered statements cannot be "SAVED" in the RAM or on external devices, such as 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 using a RUN command. The PC-8201 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 the STOP Key is pressed while a program is running.

The PC-8201 is device independent, allowing all of your programming on the PC-8201 to be done without any peripheral devices attached. All programs can be written, edited, run, and saved within the unit itself. You have the option of attaching a Data Recorder for the purpose of saving your programs, but it is certainly not necessary. You are not even required to attach a printer since the LCD screen displays your program for editing and modification.

Getting Started with N82-BASIC

To begin using the N82-BASIC language, get the PC-8201 into the BASIC Mode. Your screen should appear as illustrated:



The "Ok" message with the flashing cursor appearing on the next line indicates that the PC-8201 is ready for use and is waiting for instructions from you. The PC-8201 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 they are typed and the ↵ Key is pressed.

See Chapter 4 for a complete list of system commands.

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

Using the Direct Mode

The Direct Mode of N82-BASIC allows an individual statement to be executed. Statements used in the Direct Mode are typed without line numbers, and the ↵ Key is then pressed to execute the statement.

An example of using the Direct Mode:

Type in: INPUT "Radius 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 your choice and press the ↵ Key. (For example press 5 then ↵ Key).

Type in: PRINT "Diameter = ";2*R ↵

This statement calculates the diameter of the circle and prints:

Diameter = (result)

on the screen. (For our example the result will be 10)

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

3.14159 is the value of π.

This statement calculates the area of the circle and prints:

Area = (result)

on the screen. (For our example 78.5397 will be the result.)

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

This statement calculates the circumference of the circle and prints:

Circumference = (result)

on the screen. (The direct mode will give 31.4159 as the result of our example.)

While in the Direct Mode, the PC-8201 prints an "Ok" message on the screen each time the ↵ Key is input at the end of the 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-8201 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 the value is redefined.

Notice whenever you type NEW or CLEAR, the radius loses its value.

Using the Program Mode

Assume that you wanted to know the diameter, area and circumference of a circle with a different radius, then you would have to repeat the whole process described for the Direct Mode. 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 the ↵ Key.

If you type the program correctly the question "Radius of circle?" will appear on your screen. Type in a radius value and press the ↵ Key.

Now you see the answers:

Diameter = (result)

Area = (result)

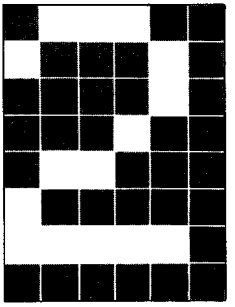
Circumference = (result)

Congratulations, you have written your first program. Now SAVE it in the RAM. Press the f.2 Function Key and then type:

"RADIUS.BA" and press the ↵ Key

Press the f.10 Function Key (hold SHIFT down and press f.5) to go to the MENU and you will see your program name among the other files.

By pressing the SHIFT key, you change function keys f.1, f.2, f.3, f.4, f.5, to f.6, f.7, f.8, f.9, and f.10 respectively. So, by holding down SHIFT and pressing f.5 you have entered the f.10 Function Key (MENU).



General Information

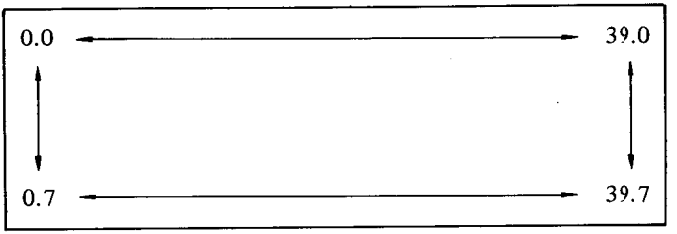
CHAPTER 2

General Information

Screen Display

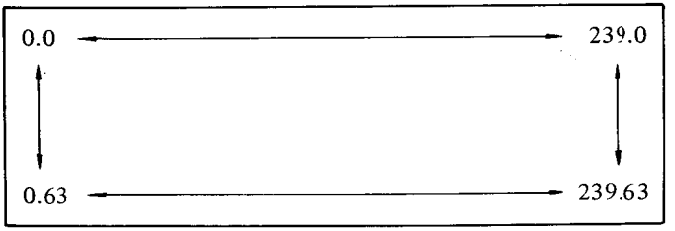
The Liquid Crystal Display screen can display 8 lines of 40 characters per line. The first 7 lines are usually available for your use, depending on the mode of the PC-8201. The last line usually displays the names of the functions corresponding to the Function Keys on the keyboard.

The character positions on the screen are numbered 0 through 39 columns from left to right, and 0 through 7 lines from top to bottom:



Each position is addressable by using the LOCATE statement.

Dot graphics may be displayed on the screen of the PC-8201. The screen consists of 240 pixels (dots) across from left to right, with the columns numbered 0 through 239. There are 64 pixels from top to bottom on the screen, with the lines numbered 0 through 63:



Each dot is addressed using the PSET statement.

Statements and Line Numbers

BASIC programs consist of statements, which give the PC-8201 instructions. 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 is more than one statement in a line, the group of statements are called compound statements. Compound statements must be separated by a colon (:).

Each program line begins with a line number, which indicates the sequence in which they are to be executed and stored in the memory. Program execution starts with the lowest numbered line and then continues in programmed sequence. Acceptable line numbers can range from 0 to 65529. Each program line cannot exceed 255 characters.

EXAMPLE OF A PROGRAM LINE FORMAT:

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

The above program line is a compound statement with the individual statements separated by a colon, and a line number of 20.

Special Symbols

In addition to regular arithmetic symbols, such as +, −, *, and /, 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 occurs 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  unit widths called Space Zones.

- Colon (:) is used to separate compound statements within one program line, which saves memory space.

- Semicolon (;) is usually used in the PRINT or INPUT statement. It directs the cursor to the position immediately following the last printed character on the same line.

- A Single quotation mark ' is used to precede remarks or comments in a statement. These remarks are not executed when the program is run.

- Double quotation marks (" ") are used to enclose character strings. The strings cannot be longer than 255 characters.

- Question mark (?) is the abbreviation for the PRINT command.

- Blank spaces are generally ignored by the PC-8201.

Special Symbols following Variable Names:

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

Control Characters

The characters recognized by N82-BASIC include:

Upper case alphabet	A - Z
Lower case alphabet	a - z
Numbers	0 - 9
Special symbols	. – , : ; ' " ? % ! # $ & = ( ) [ ] \ / @ + ^ _ etc.
Graphics characters	◀ ↵ ▒ , and up to a total of 125 programmable graphics characters

Error Messages

If an error occus during program execution, the PC-8201 will terminate the program and return to the Direct Mode.

The error message is displayed on the screen if the PC-8201 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 and explanations of error messages.

Program Editing

The two editing modes featured by the PC-8201 are the Direct Mode in BASIC and the TEXT mode. You can edit your programs in either mode, depending upon your preference.

Screen Editing of Programs

Editing programs in the BASIC mode is done by modifying program lines. When you edit in this manner, the ↵ Key must be pressed after your changes have been made in order to be entered into the memory. Remember that a program line cannot be over 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 the ↵ Key.

INSERT:

1. Move the cursor to the place where the character is to be inserted using the Cursor Movement Keys.

2. Press the INS Key.

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 Movement Keys, then press INS Key and insert the character(s).

5. Press the ↵ Key to enter your insertions into the memory.

6. Keep in mind that when INSERTion editing in the Direct Mode of BASIC is used, the INSERT is active until a ↵ Key is pressed, or a cursor movement key is entered.

DELETE:

To delete characters that precede the cursor in a program line, LIST the line, then:

1. Move the cursor to the right of the character to be deleted.

2. Press the DEL Key.

3. Press the same key as many times as needed to delete characters to the left of the cursor.

4. Press the ↵ Key to store the changes.

To delete characters that follow the cursor in a program line, LIST the line, then:

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

2. Press and hold the SHIFT Key and then input the DEL key.

3. Repeat the same process as many times as needed.

4. Press the ↵ Key to store the changes.

To delete an entire line:

1. Type the line number to be deleted, with no characters following it.

2. Press the ↵ Key.

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 and hold the CTRL Key and input the E Key, then press the ↵ Key.

This procedure of holding the CTRL Key down while inputting a character will appear in this manual as CTRL + ⟨ character ⟩. Do not input the + sign, because it just signifies that the two keys are being entered simultaneously.

ADD:

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

The program is executed following the sequential order of line numbers. The PC-8201 will put the line numbers in increasing order, regardless of what order the lines were typed in.

To rewrite a line just type the old line number followed by the contents of the new line, even if you are at the end of the program. As stated above, the PC-8201 will put the lines in order when the program is LISTed.

Other Keys Used for Screen Editing

- TAB — Moves the cursor directly to columns 8, 16, 24, and 32 of the line in which the cursor is positioned.
- STOP — Terminates the EDIT mode.
- CTRL + C — Same as the STOP Key.
- 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 the DEL BS Key.
- CTRL + I — Same as the TAB Key.
- CTRL + K — Moves the cursor to the cursor "home" position, in the upper left corner of the screen.
- CTRL + L — Clears the screen and moves the cursor to the home position.
- CTRL + M — Same as the RETURN Key.
- CTRL + Q — Continues the scrolling of a program listing on the screen after the LIST instruction has been given and the listing was interrupted. See CTRL + S.
- CTRL + S — Interrupts the scrolling of a program listing on the screen after the LIST instruction has been used.
- CTRL + R — Same as the PAUSE Key.
- CTRL + U — Erases a line displayed on the screen. The internal memory is not altered.

Editing Programs Using the TEXT Mode

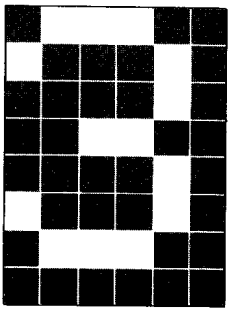
Programs can be edited in the TEXT mode by entering EDIT and then pressing the Key. To exit the TEXT editing mode, press the ESC Key twice or the f.10 Function Key (SHIFT Key and f.5).

In this mode, any character typed is inserted one at a time, 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-8201 immediately, before you press the Key.

Use of the TAB Key while in the TEXT editing mode will indent the line being typed. The Key must be used to end a program line being typed or modified in this mode, or else the line will appear in the program out of sequence.

The PC-8201 will check a newly input program line in the TEXT editing mode. If a line with only a line number and no characters following it or if a line which does not contain a line number is input by you, the PC-8201 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.

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 pattern 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 User's Guide.



Expressions and Operations

CHAPTER 3

Expressions and Operations

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 ard string variables.

An example of a numeric variable is when you want to use the element CHARACTERS within a program, and 40 characters are needed. You can then assign the name "CHARACTERS" 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 variable name to distinguish between variables. A variable type specified character placed at the end of the variable name, ndicates 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 sub-sequent use. The recommended characters to use for a variable name 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 BASIC Reserved Words

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 occurs during program execution.

ERR · This variable holds the error code which causes the interrupt.

See Appendix A1 for a complete listing of Reserved Words.

Before utilizing a variable within your program you should initialize it to some type of a value. As an example we will initialize CHARACTERS with the following statement:

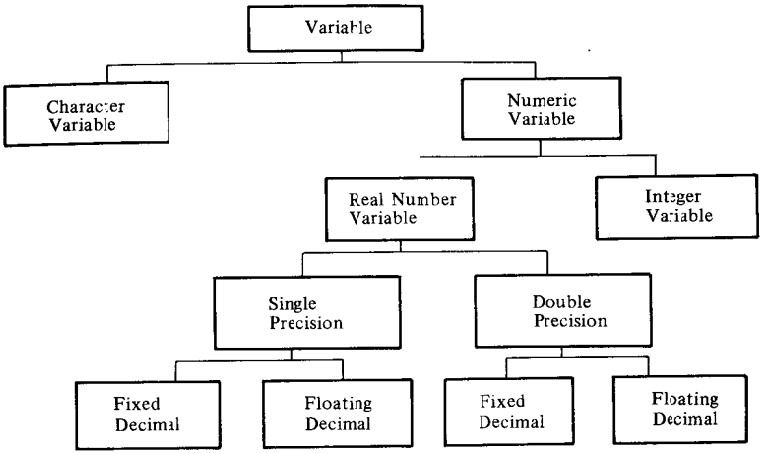
```
CHARACTERS=40
```

If you do not initialize your variables, then the numeric variables are automatically initialized to zero, the character variables are initialized to empty (null) string (" ").

Types of Variables

The last character of a variable name determines the type of variable. The 4 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 variables:

A$	String variable
A! or A	Single precision real number variable (default)
A#	Double precision real number variable
A%	Integer variable

As you can see in the above example the variable name "A" in conjuction with special characters represent 4 different types of variables.

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

String Variables

String variables are a collection of characters with a non-numeric value. String Variables are composed of letters (both upper and lower case letters), numbers or special symbols. If double quotations are used inside the character variable, CHR$(34) should be used to enter the double quotations. The maximum length of a String Variable is 255 characters, and it should not be used in an arithmetical operation.

Numeric Variables

Numeric variables are integers or real numbers, represented by a numeric variable name.

Integer Variable

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

- Numbers with no decimal point.
- Numbers in the range from –32768 to +32767.
- Numbers followed by % (percentage sign).

EXAMPLES: NUMBER% = 1234
NUMBER% = 123%

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 either fixed decimal form or floating decimal form.

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

A floating decimal form number represents its value in scientific notation with an exponent.

Single Precision Format

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

The magnitude is stored in 7 significant (high order) digits internally. When displaying the numeric value, the seventh digit is rounded off and trailing zeroes are deleted to show 6 digits or less on the screen.

The exponent portion is attached to the magnitude. It consists of the letter E, a sign, and a two digit number. The valid exponent number is from 01 to 38.

Single precision numbers have the following characteristics:

- Real numbers of less than 7 digits.
- Real numbers followed by an exclamation mark (!). The exclamation mark is optional.
- Real numbers range from −1.70141E+38 to 1.70141E+38.
- Exponent is indicated by E.

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

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

Double Precision Format

The double precision floating decimal number consists of magnitude and exponent as in the single precision format.

The magnitude is stored with a precision of 17 significant digits and can be displayed in up to 16 digits, the 17th digit is rounded off. The exponent is indicated by the letter D, followed by a sign and a two digit number. The valid exponent range is 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 pound sign (#).

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

Floating decimal: NUMBER# = −1.09432D+06
NUMBER# = 0.3141592653D+01

Array

A group of logically related variables designated by the same variable name is called an Array. The items of an array are called elements. Each element is assigned a unique number called the subscript, to distinguish each of them.

Array values are indexed by 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 with the following example. Let the array "ITEMS%" be two-dimensional to a size of 4 rows by 8 columns. To reserve memory space for the array, the statement DIM ITEMS%(3,7) would be used. Following is the layout of the location of each element of an array ITEMS% :

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 use the name ITEMS%(1,3), this element contains the value 91.

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 the DIM statement starts reserve space beginning with element 0. We could have started with row1 and column 1, but 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)

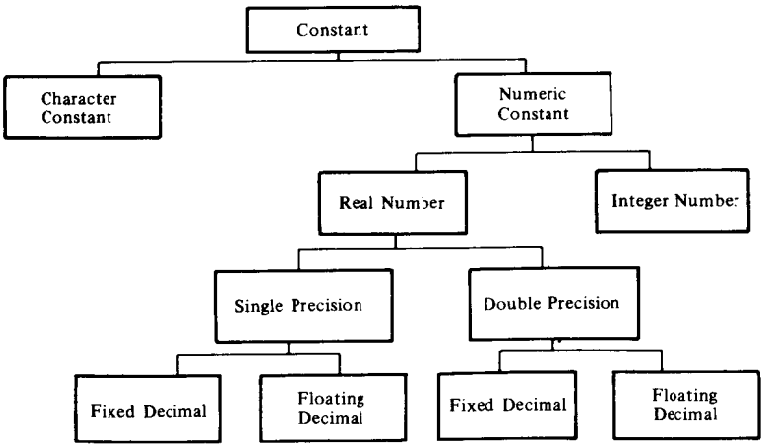
An array can be expanded to include over 100 dimensions, (subelements of each element). The number of elements of an array is limited by the amount of memory space available.

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 only of one type. Also, if the array is a character array, no element should be longer than 255 characters.

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:



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 will be displayed in floating format. The following numeric constants are valid:

25.	234567
−1234.01	32760
12345678901.23	.1234567890123
3.14159	
.0000002	

It is possible to enter numeric constants 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.
x	is the number with up to 16 significant digits.
D	represents the Exponent (the power of 10)
nn	is the exponential value in the range of −38 to +37.

The Exponent in this format can be 0 but never blank. The following are valid numeric constants in D format:

1.2568D10	8.254681325257D−30
−1.234567890123D−12	2358.25624798D2
1235D−30	1.2D20

Integer Constants

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

1	0	−1234
25	−15	100
32767	−32767	10000

Character Constants

A character 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 constant in a program. Each character can be a letter, a number, a space, or any ASCII character except a control character and quotation marks. In such cases use CHR$ function and concatenate (connect) them into the string with the + sign.

The following is an example of acceptable character constants:

Character Constant

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

Internal Representation

Another "Constant"

Type Conversion

Numeric variables can be converted from one type to another in N82-BASIC. Character 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 order of precedence:

   Integer

   Single Precision

   Double Precision

   Integer, as shown abive, is the lowest degree of precision. Arithmetic operations are performed in rumeric 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. Integers are returned as the result of the operation.

   EXAMPLE:

Statement	Variable	Content
A#=12.34	A#	12.34000015258789
B=NCT A#	B	−13

4. Digits after a decimal point are omitted when real numbers are converted to integers. 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#	12.340000152587 89 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"

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-BASIC are:

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

Arithmetic Expressions:

Priority	Operator	Function
1	^	Exponentiation
2	–	Negative sign
3	*	Multiplication
3	/	Division
4	\	Integer division
5	MOD	Modulo division (Remainder)
6	+	Addition
6	–	Subtraction

An arithmetic expression is defined as:

⟨ arithmetic term ⟩ [⟨ arithmetic operator ⟩ ⟨ arithmetic term ⟩]

The follwing are examples of valid arithmetic expressions:

NOT A%	Integer result
A%+23	Integer result
SUB.TOTAL+CURRENT*UNIT.PRICE	Single precisior
ONE%*THREE	Single precisior
+1/–4	Single precisior
3.14159*RADIUS^+2	Single precisior
3*4/(PI#*R^2)	Double precisicn

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 parertheses in an arithmetic expression are performed before the higher priority ones (outside the parentheses).

4. Priority order is in effect inside parentheses.

5. Any division with zeroes will cause an error. This is also the case if a zero is raised to a power of a negative number for example (0^–6).

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

Example:

Statement	Meaning	Result
Z*X+Y	$ZX+Y$	
X/Y+2	$\frac{X}{Y}+2$	
(X+Y)/2	$\frac{X+Y}{2}$	
X^2+2*X+1	X^2+2X+1	
X^(Y^2)	$X^{(Y^2)}$	
X^Y^2	$(X^Y)^2$	
X*(-X)	$Y(-X)$	
2/0	$\frac{2}{0}$	?/0 ERROR
0/-1	$\frac{0}{-1}$	
10\3	$INT(\frac{10}{3})$	3
15 MOD 4	$15-4(INT(\frac{15}{4}))$	3

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:

STRING$ ⟩ "HELLO"	String relation
NUM1 ⟨ = NUM2	Numeric relation
NUMBER% ⟨ ⟩ 225*(5–ONE)	Numeric relation with arithmetic sub-expression
539 = ONE	Numeric relation

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 has all zero bits and a decimal minus one 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

The following are examples of logical expressions:

NUM1% OR NUM2%

I% AND 23

I% AND (NUMBER XOR TOTAL) IMP TEST%

(A AND B) OR (A AND C)

STRING$ > = "A" AND STRING$ < = "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		0000000000001111	(15)	
	AND	0000000000001110	(14)	
		0000000000001110	(14)	(TRUE)
10 OR 23		0000000000001010	(10)	
	OR	0000000000010111	(23)	
		0000000000011111	(31)	(TRUE)
NOT 153	NOT	0000000000011001	(153)	
		1111111111100110	(−154)	(TRUE)
25 XOR 13		0000000000011001	(25)	
	XOR	0000000000001101	(13)	
		0000000000010100	(20)	(TRUE)
234 EQV 3429		0000000011101010	(234)	
	EQV	0001110101100101	(34299)	
		1111001001110000	(−3472)	(TRUE)
56 IMP 720		0000000000111000	(56)	
	IMP	0000001011010000	(720)	
		1111111111010111	(−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.

String Expressions

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

Connecting Strings

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

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 "PORTABLE COMPUTER"

Comparing Strings:

When sorting strings, relational operators are used for the comparison 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 equal if they have the same character in the respective position, 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

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)

PRINT 2, 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 with arithmetic operators.

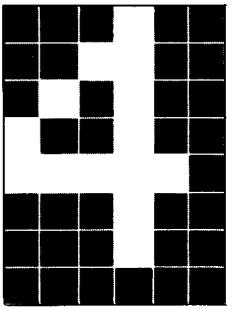
EXAMPLES:

```
"N82"+"BASIC"

3.14159*2

10+3/5

A+B/C-D

TAN(DO)+COS(DO)

10\3/2

13 MOD 2
```

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	Modulo division (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



N_{82}-BASIC Instructions

CHAPTER 4

N82-BASIC Instructions

ABS

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

FORMAT: ABS(⟨ numeric expression ⟩)

SAMPLE STATEMENT: PRINT ABS(−8.+7.9)

DESCRIPTION: The 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 VALLE 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 multiple 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 bits binary integers. Therefore, the ⟨ operands ⟩ must range from −32768 to +32767. If operands are not within this range, an "?OV Error" (Overflow) message will appear on the screen.

SEE ALSO: Functions 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
40 IF A=5 AND C=6 THEN 70
50 PRINT " A IS NOT 5, OR B IS NOT 6"
70 IF A=5 AND C>6 THEN 90
80 PRINT "A IS NOT 5, OR C IS NOT GREATER
   THAN 6"
90 PRINT  "A IS 5,  B IS  6,  AND  C  IS
   GREATER THAN 6"
100 END
```

ASC

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

FORMAT: ASC(⟨ string ⟩)

SAMPLE STATEMENT: PRINT ASC("AB")

DESCRIPTION: The 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.

For more detail on ASCII codes see the Table of Character Codes.

SEE ALSO: The CHR$ function and Table of 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$="" THEN 40
60 FOR X=32 TO 122
70 PRINT "THE ASCII VALUE OF ";CHR$(X);"
   IS";ASC(CHR$(X))
80 NEXT X
```

ATN

FUNCTION: This function provides the inverse tangent.

FORMAT: ATN(⟨ numeric expression ⟩)

SAMPLE STATEMENT: PRINT ATN(.05)

DESCRIPTION: The ATN function, used in trigonometric applications, 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 functions.

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
```

BEEP

FUNCTION: This command is used to generate a "BEEP" sound from the PC-8201.

FORMAT: BEEP

SAMPLE STATEMENT: BEEP

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

NOTE: The BEEP has no parameter.

The statement PRINT CHR$(7) has the same function as the BEEP command.

SEE ALSO: The SOUND command.

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
```

BLOAD

FUNCTION: This system command 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 command loads a Machine Language program file specified by ⟨ fi e name ⟩ nto the memory. The PC-8201 loads a Machine Language file from RAM if ⟨ external device name ⟩ is om tted.

Loading is not possible if a file in RAM is written via the BSAVE command without the file type. However, file type may be omitted when the actual load ng process s 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 statement is not required after a ".CO" file is oaded.

The PC-8201 returns to BASIC from the subroutine by using a RET Machine Language instruct on. It loads from a data recorded if "CAS:" is designated for ⟨ external device name ⟩ . The PC-8201 loads the first file it locates if ⟨ file name ⟩ is omitted.

The [SHIFT] and [STOP] Keys can be pressed simultaneously to interrupt the execution of a BLOAD "CAS:" command.

BLOAD?

FUNCTION: This system command 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⟩" command only when a data recorder is connected to the PC-8201. If the content of both programs are identical, the PC-8201 displays an "Ok" message. Otherwise, if any error has occurred during the load process, the PC-8201 will output the message "BAD" and execution is terminated.

The BLOAD? command should be used immediately after the BSAVE command is executed.

The [SHIFT] and [STOP] Keys can be pressed at the same time to interrupt the execution of a BLOAD? "CAS:" command.

BSAVE

FUNCTION: This command is used to save a Machine Language program from the memory in 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 command 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⟩. This program may use BSAVE and BLOAD only if it can be executed from ⟨start address⟩ (execution entry point).

The PC-8201 saves a Machine Language file from RAM if ⟨external device name⟩ is omitted. When device name is specified, "CAS:" is designated for data recorder.

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 via the BLOAD statement.

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

The SHIFT and STOP Keys can be pressed simultaneously to interrupt the execution of a BSAVE "CAS:" command.

SEE ALSO: The BLOAD commands and the chapters or Files and Machine Language programming.

CDBL

FUNCTION: This function converts integers or Single Precision real numbers to Double Precision real rumbers.

FORMAT: CDBL(⟨ numeric expression ⟩)

SAMPLE STATEMENT: PRINT CDBL(454.67)

DESCRIPTION: The CDBL function converts the ⟨ 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: The CINT and CSNG functions.

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-8201 to change a single value ASCII code to its matching character.

FORMAT: CHR$(⟨ 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, rumber, or any special character. The value of the ⟨ numeric expressior ⟩ must be within a range 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 to the decimal point.

SEE ALSO: The ASC function, and the Table of 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
```

CINT

FUNCTION: This function corverts Single or Double Precision real numbers to integers.

FORMAT: CINT(⟨ numeric expression ⟩)

SAMPLE STATEMENT: CINT (4578)

DESCRIPTION: The CINT function rounds off (truncates) the value of the ⟨ numeric expression ⟩ and returns an integer.

An "?OV Error" (Overflow) message is displayed if the ⟨ numeric expression ⟩ is not between −32768 and +32767.

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

CLEAR

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

FORMAT: CLEAR { ⟨ string area size ⟩ } {,⟨ maximum memory used in BASIC ⟩ }

SAMPLE STATEMENT: CLEAR 300,60000

DESCRIPTION: This statement initializes all numeric variables to zero and string variables to null string. If designated parameters are omitted, the previous value is preserved.

Designate only the first parameter if large character string arrays are used, or a large number of character string operations are performed. The second parameter sets the maximum memory used for BASIC and maintains memory chapacity used for Machine Language programs.

In the sample statement given above, the maximum memory specified is 59999, thus a Machine Language program can be placed between the area from 60000 to 62335. The locations beyond 62337 cannot be designated because they are reserved for the PC-8201.

NOTE: When both parameters are omitted, only the initialization of the variables is executed and the establishment of memory location remains unchanged. The string region is altered if the first parameter is specified. The establishment of a region in the memory is not altered until a new CLEAR statement is executed. Therefore, if a large string region is not secured in the program, an "?OS Error" (Out of String space) error message can occur during execution.

When a CLEAR statement is executed, any data in the PASTE buffer will be erased.

SEE ALSO: The BLOAD, EXEC, and DIM commands.

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
```

CLOAD

FUNCTION: This system command 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-8201 will retrieve that program file from the cassette tape and load it into memory. However, when a ⟨ file name ⟩ is not specified, the PC-8201 loads the first program encountered from the cassette tape. A maximum of six characters can be used for the ⟨ file name ⟩.

When a specific file is being searched, the system outputs a "SKIP: ⟨ file name ⟩" message during the searching process. The PC-8201 will continue to scan the cassette tape until it finds the specific file, at which time it outputs a "FOUND" ⟨ file name ⟩" 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 Data Recorder, the PC-8201 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 name ⟩ does not exist on tape, the CLOAD command 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, and may be due to improper set up of the Data Recorder.

The CLOAD process can be interrupted by pressing both the SHIFT and STOP Keys simultaneously.

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

CLOAD?

FUNCTION: This system command 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 matches with the program currently residing in the memory. The ⟨ file name ⟩ refers to the program recorded on tape. If the content of both programs is 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? command is normally used immediately after the CSAVE command.

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

CLOSE

FUNCTION: This statement is used to close files.

FORMAT: CLOSE { [#] ⟨ file number ⟩ } { ,{{#},file number ⟩ } . . .

SAMPLE STATEMENTS:
CLOSE
CLOSE #1,#2

DESCRIPTION: This statement is used to terminate input/output between a BASIC program and the data file(s). It closes the file corresponding to ⟨ file number ⟩. These files are closed simultaneously if more than one ⟨ file number ⟩ is specified. All currently opened files are closed if ⟨ file number ⟩ is omitted.

Input/Output for a closed file is again possible if it is reopened by a specified file number.

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

SEE ALSO: The OPEN, END, and NEW commands.

CLS

FUNCTION: This statement erases the display screen.

FORMAT: CLS

SAMPLE STATEMENT: CLS

DESCRIPTION: The CLS statement clears all alphanumeric characters and graphics characters from the display screen. However, when the second parameter (Function key display switch) in the SCREEN statement is "1" (means it is ON), only the contents of the Function keys will remain on display.

NOTE: This statement 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:INPUT"HIT RETURN TO CLEAR
   THE DISPLAY"; C$
80 CLS
```

COM ON/OFF/STOP

FUNCTION: This command establishes, prohibits, or informs of interruption by a data transmission circuit.

FORMAT:

```
COM [ ON   ]
    [ OFF  ]
    [ STOP ]
```

SAMPLE STATEMENT: COM ON

DESCRIPTION: The COM command informs BASIC that data that is being input from an external device through the communication port (the RS-232C circuit) may occur.

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

The COM OFF prohibits a BASIC program from being interrupted by communications input.

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

SEE ALSO: The ON COM GOSUB command.

CONT

FUNCTION: The CONT command restarts the execution of a program that was interrupted, either by the STOP statement, or the pressing of the STOP Key.

FORMAT: CONT

SAMPLE STATEMENT: CONT

DESCRIPTION: This command is normally used in conjunction with the STOP Key (or the CTRL + C Keys) to debug a program. The CONT command 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 command or pressing the f.7 Function Key (SHIFT and f.2), the program will resume execution where the half occurred. If the program has been altered while execution is stopped, then execution cannot be continued using this command.

COS

FUNCTION: This function provides 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 an angle. The unit of the ⟨ numeric expression ⟩ is the angle expressed in radians.

NOTE: To convert an angle from degrees to radians multiply the degrees by .0174533.

SEE ALSO: SIN, TAN, and ATN functions.

SAMPLE PROGRAM:

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

CSAVE

FUNCTION: This system command is used to save a copy of the program on cassette tape.

FORMAT: CSAVE "〈file name〉"

SAMPLE STATEMENT: CSAVE "DEMO"

DESCRIPTION: This command saves a program currently in the memory onto cassette tape. The file name is specified using 6 characters or less. The PC-8201 will return to Direct Mode after the CSAVE command has been executed.

NOTE: Please refer to BSAVE and SAVE commands in regard to saving ".CO" and ".DO" files (ASCII code format) respectively.

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

If interruption is necessary during the execution of a CSAVE command, press the SHIFT Key and the STOP Key at the same time.

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

CSRLIN

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

FORMAT: CSRLIN

SAMPLE STATEMENT: PRINT CSRLIN

DESCRIPTION: The CSRLIN (cursor line) function returns the line 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 from 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: The POS function.

SAMPLE PROGRAM:

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

DATA

FUNCTION: This statement holds the constants which are loaded into the variables with a READ statement.

FORMAT: DATA ⟨ constant ⟩ {, ⟨ constant ⟩ } ...

SAMPLE STATEMENT: DATA 1, CBA,1465

DESCRIPTION: The DATA statement is used to define information to the READ statement, and it can be inserted anywhere in the program. A program can have as many DATA statements as needed with ro more than 255 characters on each data line.

READ statements nput constants from DATA statements, starting from the DATA statement with the smallest line number. However, the order can be revised with the RESTORE statement.

Arithmetic expressions used for reading in numeric constants are not permitted in DATA statement. Constants are separated by commas on the data line. Their types should match the corresponding variable types in the READ statement. 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 in double quotation marks

SEE ALSO: See the READ and RESTORE commands.

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 function provides the data from the internal real-time clock of the PC-8201.

FORMAT: DATE$="⟨ year ⟩/⟨ month ⟩/⟨ day ⟩"

SAMPLE STATEMENTS:
```
DATE$="83/05/05"
PRINT DATE$
```

DESCRIPTION: The DATE$ function 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 again is not necessary, unless a Cold Start has been performed.

NOTE: The ⟨ year ⟩ value must be re-designated when the year advances because the timer repeats the same year again.

SEE ALSO: The TIME$ function.

DEFINT/SNG/DBL/STR

FUNCTION: This command 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 statement, 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 statement is used, in Double Precision real number format a DEFDBL statement is used, or in string format a DEFSTR statement is used.

Only one character may be used to specify each variable name, with its range designated in character range . The range is indicated by joining the 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 statement. All variables starting with characters which have not been type designated by a DEF statement are assumed to be Single 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) statement is used to allocate memory space for storing an array.

FORMAT: DIM ⟨ variable name ⟩ (⟨ max subscript value ⟩ { , ⟨ max subscript value ⟩ . . . })

SAMPLE STATEMENT: DIM A(12,2)

DESCRIPTION: This statement 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 statement is not defined, the maximum subscript value is set at 10. Any reference to an array beyond the allocated size will display a "?BS Error" (subscript out of range) message.

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

The minimum subscript values is set at 0. For instance, if the array A is dimensioned A(3), four elements are in the array with subscripts of 0, 1, 2, and 3.

SEE ALSO: Array variables.

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
```

EDIT

FUNCTION: This command shifts the PC-8201 from BASIC mode into TEXT mode.

FORMAT: EDIT {⟨Line in which to start eidting⟩}
{—⟨line in which to stop editing⟩}

SAMPLE STATEMENT: EDIT 20—80

DESCRIPTION: The command shifts into TEXT mode and allows program editing. If parameter is not designated for editing, the entire program text is open for editing. Other combinations are also allowed.

Parameter Specified	Line(s) Edited
No parameter specified	All
First parameter only	Only that line
First parameter and hyphen	That line and all following
Hyphen and second parameter	First line to the second line specified by that parameter
First parameter hyphen, and second parameter	The range of the two parameters

SEE ALSO: Program Editing.

END

FUNCTION: The END statement is used to terminate program execution.

FORMAT: END

SAMPLE STATEMENT: END

DESCRIPTION: This command terminates program execution, closes all files, and returns the PC-8201 to Direct Mode.

The END statement is inserted into the program at the location(s) at which it termirates program execution. The final END statement may be omitted in a program, but files are not closed.

SEE ALSO: The STOP and CLOSE commands.

SAMPLE PROGRAM:

```
10 PRINT "HIT 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
   EXECUTED."
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 is reached.

FORMAT: EOF(⟨file number⟩)

SAMPLE STATEMENT: IF EOF(3) THEN CLOSE #1 ELSE GOTO 100

DESCRIPTION: The EOF (End Of file) function determines if an end of a sequential file, designated by the ⟨ file number ⟩, is reached.

The function returns a non-zero (true) value if the end is reached, and 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 zero (false) value.

For the operation to be true both ⟨ operand 1 ⟩ and ⟨ operand 2 ⟩ must be true, 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)

For more details on logical operators see Chapter 3.

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

SEE ALSO: Functions AND, IMP, NOT, OR, XOR, 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 is 1111 0010 0111 0000. By looking at the table under DESCRIPTION notice that the computation was done correctly.

ERL

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

FORMAT: ERL

SAMPLE STATEMENT: A=ERL

DESCRIPTION: The ERL function is a Reserved variable used in the error processing routine. 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 the values cannot be assigned.

NOTE: The ERL function has no parameters.

SEE ALSO: See the ON ERROR GOTO and ERROR statements.

ERR

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

FORMAT: ERR

SAMPLE STATEMENT: B=ERR

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

The ERR function is a Reserved variable which contains the error code when an error is detected. The content of ERR can be accessed but the values cannot be designated. The PC-8201 assigns ERR when an error occurs.

NOTE: The ERR function has no parameter.

SEE ALSO: See the ON ERROR GOTO and ERROR statements.

ERROR

FUNCTION: The ERROR statement 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 statement simulates the occurrence of that error and prints the corresponding message.

The ERROR statement may be used as a user-defined or undefined error code. When under particular conditions, the program branches to an error routine specified with the ON ERROR GOTO statement.

SEE ALSO: The ON ERROR GOTO and the ERL/ERR functions, and the Table of Error Codes.

SAMPLE PROGRAMS:

```
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
    LINE NUMBER";
530 IF ERR=1 THEN PRINT"NEXT WITHOUT FOR
    ";
540 IF ERR=5 THEN PRINT"AN ILLEGAL
    FUNCTION CALL";
550 IF ERR=255 THEN PRINT"AN UNDEFINED";
560 PRINT " ERROR HAS OCCURED.":PRINT
570 RESUME NEXT
```

EXEC

FUNCTION: This statement executes a Machine Language subroutine.

FORMAT: EXEC ⟨ initial location ⟩

SAMPLE STATEMENT: EXEC 61000

DESCRIPTION: The EXEC statement transfers control to a Machine Language subroutine in the memory. The ⟨ initial location ⟩ is designated by integers 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-8201 can return to BASIC from a Machine Language subroutine via the RET command.

NOTE: Select ⟨ initial location ⟩ carefully to avoid erratic operation.

SEE ALSO: The BLOAD, PEEK, and POKE commands.

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) message will result if the power raised is greater than 87.33655.

FILES

FUNCTION: This command displays all the files in the RAM.

FORMAT: FILES

SAMPLE STATEMENT: FILES

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

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

SEE ALSO: Chapter 5, Files.

SAMPLE PROGRAM:

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

FIX

FUNCTION: This function 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 function does not round off the number.

SEE ALSO: INT and CINT functions

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
```

FOR . . . TO . . . STEP ~ NEXT

FUNCTION: This statement repeats a series of instructions for 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 statement executes a series of statements a given number of times (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 statement is encountered, the ⟨ variable name ⟩ increases or decreases 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 statement.

The ⟨ variable name ⟩ in the NEXT statement may be omitted. NEXT always terminates the last unmatched FOR statement. 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 of ⟨ 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 statement.

If ⟨ increment ⟩ is zero then the loop is executed continuously (infinite loop). Press SHIFT and STOP Keys for interruption.

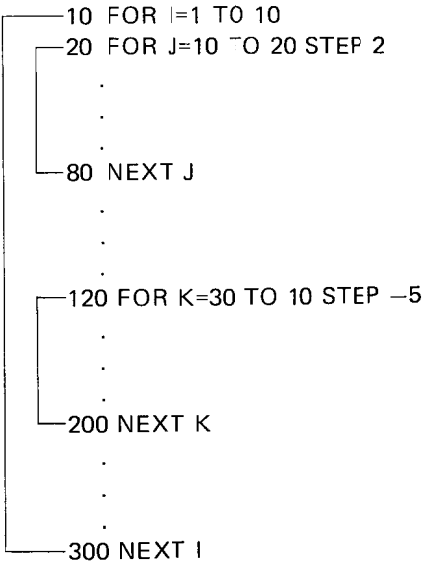
FOR ~ NEXT loops may be nested to any depth. In such case different ⟨ variable names ⟩ must be used, and the second loop must be

completely located within the first loop. An "?NF Error" (Next without For) occurs if there is an illegal form of nesting.

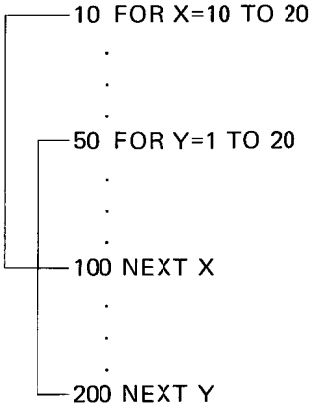
The loop may be exited with a GOTO statement. The loop will remain open until another loop is executed using the same ⟨ variable name ⟩, or when the loop is re-entered.

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 statements. 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
```

FRE

FUNCTION: This function reports the amount of unused 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 amount of unused bytes.

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 program space available.

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

GOSUB ~ RETURN

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

FORMAT: GOSUB ⟨ line number ⟩

SAMPLE STATEMENT: GOSUB 1000

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

Subroutines are called by the GOSUB statement 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 statement GOSUB is used to call the subroutines. When a RETURN statement is reached in the subroutine, the program will resume execution at the statement following the GOSUB statement.

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

SEE ALSO: The RETURN statement.

SAMPLE PROGRAM:

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

GOTO

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

FORMAT: [GOTO / GO TO] ⟨ line number ⟩

SAMPLE STATEMENTS:
GOTO 500
GO TO 500

DESCRIPTION: This command unconditionally branches to a specified ⟨ line number ⟩ in the program.

NOTE: This statement may be written either as "GOTO" or "GO TO". If two or more blanks are entered, N82-BASIC does not interpret it as the GOTO statement.

SEE ALSO: The IF and GOSUB statements.

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 statements 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 functions 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 statements are allowed. When nesting occurs the ELSE option will match the most previous unmatched IF statement.

The ⟨ ther clause ⟩ can be made up from multiple statements, separated by a colon(:).

The complexity of a multiple arrangement is limited within the range of one line, which is 255 characters long, or before the Key is pressed.

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 only a programming aid in the structure of the code.

SAMPLE PROGRAM:

```
10 M=10000:CLS
20 PRINT"YOU HAVE $";M;"."
30 PRINT"$";M;".";"HOW MACH DO YOU
   WANT TO BET ON THIS DIE":
   INPUT K
40 K=INT(K):PRINT
50 REM ** This is the nesting of the
   type of IF statement of line 70
60 REM *** when the input is not the
   right input...
70 IF K>M THEN PRINT"IMPOSSIBLE WITH
   ONLY ";  M  :BEEP:GOTO 30 ELSE IF
   K<0 THEN PRINT"SNEAKY!":BEEP:GOTO
   30 ELSE IF K>M/2 THEN PRINT
   "GENEROUS!" ELSE IF K<M/100 THEN
   PRINT"CHEAPSKATE!"
80 INPUT"  NOW WHAT DO YOU THINK WILL
   COME 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:
    PRINT"YOU   WERE  SUCCESSFUL!" ELSE
    PRINT"YOU LOST THIS TIME!":SOUND
    16000,10:M=M-K
140 IF M<1 THEN PRINT"YOU'RE BANKRUPT
    NOW!" ELSE IF M>1E+06 THEN PRINT
    "YOU ARE A MILLIONAIRE!" ELSE 30
```

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 operators 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 range from −32768 to

+32767. If not, an "?OV Error" (Overflow) message will be displayed.

SEE ALSO: Functions 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 is 1010 1111 0100 1111. By looking at the table in the DESCRIPTION section, notice that the computation is correct.

INKEY$

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

FORMAT: INKEY$

SAMPLE STATEMENT: A$=INKEY$

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

SEE ALSO: The Table of Character Codes.

SAMPLE PROGRAM:

```
10 SCREEN 0,0:CLS:X=20:Y=3
20 PRINT" TRY TO MOVE THE CURSOR IN
   DIFFERENT DIRECTIONS"
30 PRINT" U=UP,D=DOWN,R=RIGHT,L=LEFT"
40 PRINT" HIT ANY OF THE ABOVE KEYS"
50 A$=INKEY$:IF A$="" THEN 50
60 LOCATE X,Y:PRINT" ";
70 IF A$="U" AND Y>0 THEN Y=Y-1
80 IF A$="D" AND Y<7 THEN Y=Y+1
90 IF A$="R" AND X<39 THEN X=X+1
100 IF A$="L" AND X>0 THEN X=X-1
110 LOCATE X,Y:PRINT "X";
120 GOTO 50
```

INP

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

FORMAT: INP(⟨ port number ⟩)

SAMPLE STATEMENT: A=INP(15)

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 statement.

INPUT

FUNCTION: The INPUT statement 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 statement is used to display a prompting message and then accepts one or more fields of data through keyboard input.

When the INPUT statement is executed, the ⟨ prompt statement ⟩ is displayed with a question mark following it, and the PC-8201 waits for data to be entered through the keyboard. If the prompt statement is omitted, the question mark alone will be displayed.

The input ⟨ variable(s) ⟩ are separated by commars, 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 statement.

If the number of data elements is less than the number of variables indicated, a double question mark (??) is displayed. This asks for additional 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 statement, disregarding the extra data. The message "?Extra ignored" is ther displayed.

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

It is optional to enclose the character string in double quotation marks. However, if blank spaces (leading or trailing the string) or commas are entered irto a string variable, they must be enclosed in double quotation marks ("). These double quotation marks in this case are not considered part of the character string.

Successive input of commas in the INPUT statement (with more than 2 variables such as 12,,3) indicate the omission of input data. The corresponding variable is assigned " " (null string) if it is a string type, and 0 if it is a numeric type.

SEE ALSO: The LINE INPUT and INPUT # commands.

SAMPLE PROGRAM:

```
10 INPUT "ENTER NAME          :";N$
20 PRINT "*** USE COMMA TO SEPERATE
   VARIABLES ***"
30 INPUT "ENTER 2 NUMBERS       :";A%,B%
40 C%=A%+B%
50 PRINT N$;",THE SUM OF";A%;"AND";B%;
   "IS";C%
```

INPUT$

FUNCTION: This function reads a character string of a specified length, either from a designated file in the function statement or from the keyboard.

FORMAT:

INPUT$ ([⟨ integer constant ⟩ / ⟨ integer variable ⟩] [, {#} ⟨ file number ⟩ })

SAMPLE STATEMENTS:

A$=INPUTS (10)
B$=INPUT$ (I%,#3)

DESCRIPTION: The integer in the first parameter is the character string to be input from the file. The maximum length of the string is 255 characters. The optional ⟨ file number ⟩ is assigned to the file by the OPEN statement.

The string is input from the keyboard if the ⟨ file number ⟩ is omitted in the statement. Keys entered are not displayed on the screen when input through the keyboard. The PC-8201 waits for more input if the number of characters entered is less than the specified string length.

The STOP Key or CTRL + C can be used to interrupt the INPUT$ function. All other keys are treated as part of the input string. The input buffer is cleared whenever the INPUT$ function is executed.

SEE ALSO: OPEN command.

SAMPLE PROGRAM:

```
10 REM*INPUT$*
20 CLS:INPUT"DESIGNATE A PASSWORD";PW$
30 WL=LEN(PW$)
40 REM* THEN PROGRAM  STARTS FROM HERE *
50 CLS:PRINT "ENTER PASSWORD:";
60 N$=INPUT$(WL)
70 IF N$=PW$ THEN PRINT"WELCOME USER!":
   SOUND 3000,20:GOTO 20
80 LOCATE 0,3:PRINT"INVALID PASSWORD!"
90 PRINT "PLEASE TRY AGAIN"
100 SOUND 5000,4:SOUND 1000,4
110 CLS:GOTO 50
```

INPUT

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

FORMAT: INPUT # ⟨ file number ⟩, ⟨ variable 1 ⟩ { , ⟨ variable 2 ⟩ } . . .

SAMPLE STATEMENTS:
```
INPUT#1,A
INPUT#1,B,C$
```

DESCRIPTION: This statement inputs data from a designated file (in RAM, cassette tape, etc.) and functions similar to the INPUT statement 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# statement. The ⟨ file number ⟩ is the number designated in the OPEN statement. The file should be opened for the input mode.

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 # statement is the number of data elements used each time the statement is executed. Each time the INPUT# statement 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 # statement is reached and insufficient 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 the EOF function.

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: The INSTR function locates a substring in a string and returns its position. ⟨ Character string 1 ⟩ is the original string which is searched for a match with ⟨ character string 2 ⟩ substring.

The ⟨ numeric expression ⟩ is designated by an integer, that specifies the position in ⟨ character string 1 ⟩, where the search begins. If the ⟨ numeric expression ⟩ is omitted the searching begins at position 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 position of the first character in ⟨ character string 1 ⟩ where the match begins.

When the null string (empty string) is designated for ⟨ character string 2 ⟩:

1. If the ⟨ numeric expression ⟩ is omitted then "1" is returned.

2. If ⟨ numeric expression ⟩ is less than or equal to the length 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. If not, an "?FC Error" (Illegal function call) message is displayed on the screen. When the number is read just its integer portion is taken as the beginning position.

The length of ⟨ character string 2 ⟩ must be less than or equal to (⟨ =) ⟨ character string 1 ⟩ or 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: The INT function rounds the ⟨ numeric expression ⟩ to its integer (whole) value. If the ⟨ numeric expression ⟩ is positive, INT truncates it (drops decimal digits).

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 that is returned is always less than or equal to the ⟨ numeric expression ⟩.

SEE ALSO: The FIX and CINT functions.

SAMPLE PROGRAM:
```
10 PRINT "     I          INT       FIX"
20 FOR I=-1.5 TO 1.5 STEP .2
30 PRINT USING "###.##    #####    #####";
   I,INT(I),FIX(I)
40 NEXT
```

KEY

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

FORMAT: KEY ⟨ key number ⟩, "⟨ function ⟩"

SAMPLE STATEMENT: KEY1, "LOAD"

DESCRIPTION: Up to ten programmable functions can be defined by using the five function keys (five on the keyboard, with five more in SHIFT mode). The function keys are numbered from 1 to 5, and 6 to 10 are used in the SHIFT mode. Each function key can be assigned with a character string or a control statement of 15 or less characters. Characters that cannot be input from the keyboard are entered by using the plus sign "+" and the CHR$ function.

SEE ALSO: See the Table of 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
```

KILL

FUNCTION: This command is used to erase a cesignated file.

FORMAT: KILL " (file name.file type) "

SAMPLE STATEMENT: KILL "SAMPLE.BA"

DESCRIPTION: The KILL command deletes a specific file designated by a file name and/or device name. The file to be deleted must be closed. Any opened file is indicated by an asterisk (*) when the FILES command is executed. Only one file may be deleted with each KILL command.

The file name must always include its file type extension (".BA", ".DO", and ".CO") when the KILL command is executed. The PC-8201 returns to the Direct Mode after the execution.

SEE ALSO: The LOAD and SAVE commands 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 STA
STATEMENT: B$=LEFT$(A$,4)

DESCRIPTION: A ⟨ character string ⟩ can be a string constant or a string variable. The value of a ⟨ numeric expession ⟩ must be in a range from 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: The RIGHT$ and MID$ functions.

SAMPLE PROGRAM:

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

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 can not be printed (control codes 1-31).

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

SAMPLE PROGRAM:

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

LET

FUNCTION: This statement 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 can 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
```

LINE INPUT

FUNCTION: The LINE INPUT statement 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) up to the carriage return, is substituted for the ⟨ string variable ⟩.

Any punctuation marks and symbols can be input in the ⟨ string variable ⟩. The CTRL + C Keys and the STOP Key can be pressed to interrupt the LINE INPUT statement. This will stop program execution and return the PC-8201 to the Direct Mode. The LINE INPUT statement can be continued by executing the CONT command.

SEE ALSO: The INPUT statement.

SAMPLE PROGRAM:

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

LIST/LLIST

FUNCTION: These commands are used to list either a portion or an entire program currently in the memory.

FORMT: [LIST / LLIST] { ⟨ line number 1 ⟩ } { − ⟨ line number 2 ⟩ }

SAMPLE STATEMENTS:
LIST 70–120
LLIST 70–120

DESCRIPTION: The LIST command is used to list a program on the screen; the LLIST command outputs the listing to the printer. The PC-8201 returns to the Direct Mode after the LIST or LLIST command is executed.

When both ⟨ line number ⟩s are omitted, the entire program is listed. The STOP Key may be pressed at any time to interrupt listing on the screen. The SHIFT Key and the STOP Key are pressed simultaneously to interrupt 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 within the range of both ⟨ line number ⟩s inclusive will be listed. The ⟨ line number 2 ⟩ must be greater than or equal to ⟨ line number 1 ⟩.

The LLIST command is identical to the LIST command with the exception that it outputs to a printer.

LOAD

FUNCTION: This command s 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 command loads the program specified by ⟨ file name ⟩ and optional ⟨ external device name ⟩ into the memory. When executed, the LOAD command 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-8201 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. The SHIFT and STOP Keys can be pressed simultaneously to interrupt the execution of the LOAD "CAS:" command.

The intended device is the RS-232C interface when "COM:" is designated for ⟨ external device name ⟩. Data transmission format can be indicated but ⟨ file name ⟩ cannot be used. (Please refer to the OPEN "COM:" command for details on this specific application).

The file must be a ".BA" or ".DO" file. File type extension can be omitted during loading. If the "R" (Run) option is specified, the program is 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-8201 returns to Direct Mode when the load process has been completed.

NOTE: A NEW command is executed before the actual execution of a LOAD command occurs, so that all existing variables and programs can be cleared.

SEE ALSO: The BLOAD, CLOAD, and SAVE commands. See Chapter 5, Files.

LOCATE

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

FORMAT: LOCATE ⟨ horizontal coordinate ⟩, ⟨vertical coordinate ⟩

SAMPLE STATEMENT: LOCATE 20,5

DESCRIPTION: This command 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. Home position is considered to be at coordinate (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 statement designates character coordinates and has absolutely no connection to 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:PRINT'HOP';
60  FOR I=0 TO 300:NEXT
70  LOCATE X,Y:PRINT'    ';
80  GOTO 20
```

LOG

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

FORMAT: LOG(⟨ numeric expression ⟩)

SAMPLE STATEMENT: PRINT LOG(2.7182818)

DESCRIPTION: The 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 printer head column.

FORMAT: LPOS(⟨ numeric expression ⟩)

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

DESCRIPTION: The 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 function.

MAXFILES

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

FORMAT: MAXFILES= ⟨number of file(s)⟩

SAMPLE STATEMENT: MAXFILES=3

DESCRIPTION: The number of files that can be opened is set to 1 when a Cold Start is conducted. The maximum number of files that can be opened at one time is designated by a MAXFILES statement. The range of ⟨ number of file(s) ⟩ is from 0 through 15. Once this type of value has been designated, it will be protected until it is redesignated or when a Cold Start is again conducted.

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

MENU

FUNCTION: This command returns to MENU display.

FORMAT: MENU

SAMPLE STATEMENT: MENU

DESCRIPTION: The MENU command clears all variables and returns to MENU mode. Files in access mode (indicated by an asterisk when the FILES command is executed), are closed when the MENU command 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 command 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-8201 returns to Direct Mode after the MERGE command is executed.

RAM is selected when the 〈 external device name 〉 is not specified, but 〈 file name 〉 cannot be omitted. 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 circuit) is designated, the file name cannot be used but the designation of data transmission format is possible. (Refer to the OPEN command for more detail in this specific situation.) The MERGE command will close all files after execution.

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, the line(s) in the memory are overwritten by the line from the designated file.

SEE ALSO: The SAVE and RENUM commands.

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 specified length from a desired position within the ⟨ character string ⟩.

The ⟨ numeric expression 1 ⟩ specifies 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 expression 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 function provides the remainder of an arithmetic expression.

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 as the result is returned.

In addition, 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"IT IS TOO LAARGE.":FOR I=0 TO
    1000:NEXT:GOTO 10
50  CLS:LOCATE 6,2:PRINT"THE DECIMAL
    NUMBER";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
```

MOTOR

FUNCTION: This command controls the ON and OFF functions of the motor that drives the cassette recorder.

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
    MUSIC THAT YOU LIKE"
30  PRINT"PLUG ONE END OF THE CABLE INTO
    THE PC-8201 AND INSERT THE BLACK PLUG
    INTO THE REMOTE CONNECTOR."
40  PRINT"SET RECORDER TO ON"
50  PRINT"HIT 1 TO START"
60  IF INKEY$="" THEN 60
70  MOTOR 1
80  PRINT"HIT 0 TO STOP "
90  IF INKEY$="" THEN 90
100 MOTOR 0:GOTO 50
```

NAME

FUNCTION: This command 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 command 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. The file type for both file names must be identical.

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

1. A non-existing file name is designated as an ⟨ old file name ⟩.

2. An existing file name is used as a ⟨ new file name ⟩.

3. File types for both files are not identical.

SEE ALSO Chapter 5, Files.

NEW

FUNCTION: This command erases any program or data currently in the BASIC area ard clears all variables.

FORMAT: NEW

SAMPLE STATEMENT: NEW

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

This command does not use ary parameter and it returns to Direct Mode after execution is completed.

SAMPLE PROGRAM:

```
10  REM This program will self-destruct
    when you run it.
20  PRINT"YOU HAVE DESTROYED THE
    PROGRAAM!"
30  BEEP:BEEP
40  NEW
```

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) or it returns 0 if the bit is −1.

The following table shows the negated calculations:

NOT −1 → 0 (NOT TRUE → FALSE)

NOT 0 → −1 (NOT FALSE → TRUE)

For more details on logical operators see Chapter 3.

NOTE: Because of the ⟨ operand ⟩ conversion to sixteen bit binary, the ⟨ operand ⟩ must range from −32768 to +34767. If not, an "?OV Error" (Overflow) message is displayed.

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

EXAMPLE:

INTEGER	BINARY BITS
153	0000 0000 1001 1001
−154	1111 1111 0110 0110

To negate it just replace 0 with 1 and vice versa. If you input the statement PRINT NOT 153, the PC-8201 responds −154, whose binary is 1111 1111 0110 0110, which is the correct result, according to the table above in the DESCRIPTION section.

ON . . . GOTO/ON . . . GOSUB

FUNCTION: These statements 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 statements branch to a specific ⟨ line number ⟩ based on the evaluation of the ⟨ numeric variable ⟩ .

After the ⟨ numeric variable ⟩ is evaluated its integer part is taken, and it is then used to select the first ⟨ line number ⟩ if the value is 1, the second ⟨ line number ⟩ if the value is 2, 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 branches to the next logical line (following the ON . . . GOTO/GOSUB statement).

The ⟨ line number ⟩ following the GOTO or GOSUB must be separated by commas, or else an "?SN Error" (Syntax) message is displayed on the screen. There may be any number of ⟨ line numbers ⟩ in a list (up to 255 characters per line).

When ON . . . GOSUB is used and control is transferred to the subroutine, a RETURN statement is needed. After the RETURN statement is executed, control returns to the line following the ON . . . GOSUB statement.

For more information refer to GOSUB and RETURN statements.

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 ⎫
                      ⎬ ON L GOSUB 150, 80
IF L=2 THEN GOSUB 80  ⎪   200, . . .
                      ⎪
IF L=3 THEN GOSUB 200 ⎭
```

.

.

.

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

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
```

ON COM GOSUB

FUNCTION: This statement establishes initial line of a branch process when interruption occurs from a RS-232C communications port.

FORMAT: ON COM GOSUB ⟨ line number ⟩

SAMPLE STATEMENT: ON COM GOSUB 2000

DESCRIPTION: This statement designates ⟨ line number ⟩, which branches to the first line of a routine usec to perform communication process when an RS-232C interrupt occurs.

A return from the process routine is conducted the same as normal subroutine.

A return from ON COM GOSUB routine is exactly the same as other normal routine, by using the RETURN statement. When specified, the program is restarted from where program execution was suspended. When ⟨ line number ⟩ is specified, the program is restarted from the specified line.

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

ON ERROR GOTO ~ RESUME

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

The ON ERROR GOTO statement is used to instruct the PC-8201 that an error processing subroutine is in effect. In situations when an error occurs, ⟨ line number ⟩ indicated is to receive control, 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 statement is used when an error trap 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 statement for error processing routines so that any failure in the routines can be trapped.

SEE ALSO: The RESUME and ERROR statements.

OPEN

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

FORMAT: OPEN" { ⟨externa 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 statement 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 a file number.

Three different ⟨ modes ⟩ are used to specify their 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, and "APPEND" specifies addition to a RAM file.

The PC-8201 opens a file from RAM if ⟨ external device name ⟩ is omitted, but the file name must be supplied. When device name is specified, "CAS:" is designated for data recorder. If file name is omitted in this context, the PC-8201 opens the first tape file it detects if in input mode, and creates a new tape file if in the output mode but without a file name. The SHIFT and STOP Keys are pressed to interrupt the execution of an OPEN "CAS:" command.

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 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.

NOTE: Please refer to OPEN "COM" for details on its subject.

SEE ALSO: The CLOSE and OPEN "COM" statements, and Files in Chapter 5.

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
    ARRANGED."
110 PRINT "FILES":FILES
```

OPEN "COM"

FUNCTION: This statement opens up the RS-232C 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 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.

Please refer to OPEN statement for more details.

The designated parameter that follows the COM: requires six characters to establish a data transmission circuit format. Respective designation 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 communication format.

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

VALUE		Communication Speed (Baud Rate) (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

CONTROL ACCORDING TO "X" PARAMETER

X : Affects Control

N : Does not Affect Control

The "X" parameter controls communication 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 statement (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.

Please refer to the TELCOM command in the PC-8201 User's Guide for specific precautions.

SEE ALSO: The OPEN and COM ON/OFF/STOP statements, and TELCOM section of the PC-8201 User's Guide.

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 operation. 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 table indicates the evaluation process:

–1 OR –1→–1 (TRUE OR TRUE→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)

For more details on logical operators see Chapter 3.

NOTE: Logical ⟨ 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: Functions 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 integer 32509 appears, whose binary is 0111 1110 1111 1101. By looking at the above table in DESCRIPTION, notice that the computation is correct.

OUT

FUNCTION: This statement sends data to a specific port.

FORMAT: OUT ⟨ port number ⟩, ⟨ data ⟩

SAMPLE STATEMENT: OUT 1,32

DESCRIPTION: The OUT statement sends data to a designated output port. The ⟨ port number ⟩ must be an integer ranging from 0 to 255, while ⟨ data ⟩ is the data that is output through the port.

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

PEEK

FUNCTION: This function loads the content of a designated location in the memory.

FORMAT: PEEK (⟨address⟩)

SAMPLE STATEMENT: A=PEEK (61400)

DESCRIPTION: The PEEK function returns the memory content 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.

SEE ALSO: The POKE command.

POKE

FUNCTION: This command writes data to a designated memory address.

FORMAT: POKE ⟨ address ⟩, ⟨ data ⟩

SAMPLE STATEMENT: POKE 61400,201

DESCRIPTION: This command 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 statement is used in conjunction with the PEEK statement to perform the inverse operation. It is used when the numeric values of a Machine Language subroutine are to be accessed.

NOTE: The POKE command rewrites the current memory content. 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-8201 operates abnormally after the POKE statement is used, the Reset Switch may be pressed to restore normal operation.

SEE ALSO: The PEEK statement and Machine Language Programming.

POS

FUNCTION: This function determines the current cursor column.

FORMAT: POS(⟨ expression ⟩)

SAMPLE STATEMENT: PRINT"123456" ;POS(0)

DESCRIPTION: The POS (position) function determines the current column position (horizontal position) of the cursor 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 columns on the screen, the returned value is always between 0 through 39.

SEE ALSO: The CSRLIN function.

SAMPLE PROGRAM:

```
10  CLS
20  PRINT:PRINT"PC-8201";
30  PRINT POS(X)
40  LOCATE 2,2
50  PRINT POS(X)
60  LOCATE 4,4
70 PRINT POS(X)
```

POWER

FUNCTION: This statement automatically turns OFF the electrical power of the PC-8201.

FORMAT:

```
POWER [ ⟨ timer ⟩ ]
      [ OFF       ]   ,RESUME
      [ CONT      ]
```

SAMPLE STATEMENTS:

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

DESCRIPTION: The designated value for ⟨ timer ⟩ can be ranging 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-8201 is promptly turned OFF when a POWER OFF command is executed. It returns to the MENU mode when the power switch is turned ON again. If optional parameter ",RESUME" is also appended, the PC-8201 is reinstated in the configuration when it was automatically turned OFF. The contents of the variables is also reinstated.

After a POWER CONT (Continuous Power) command is executed, the automatic power shut off function is deactivated until the POWER ⟨ timer ⟩ command is input again.

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

In the sample statement, the POWER 200 statement will cause the PC-8201 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 * 6 seconds (per unit) = 1,200 seconds or 20 minutes

PRESET

FUNCTION: This statement 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 statement resets dots on the screen at the designated coordinates. The ⟨ vertical ⟩ and ⟨ horizontal ⟩ coordinates or the function code must be within the 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 generally treated as 239, and if the ⟨ vertical coordinate ⟩ is greater than 63 it is generally treated as 63.

When the ⟨ function code ⟩ is an even number, the PRESET command reverses, and operates exactly the same way as the PSET command.

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

SEE ALSO: PSET statements.

SAMPLE PROGRAM:

```
10  PRINT"      THESE 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
```

PRINT/LPRINT

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

FORMAT: [PRINT / LPRINT] { " } { ⟨ expression ⟩ ... } { " }

SAMPLE STATEMENTS:
PRINT "ABC"
LPRINT "PC-8201"

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

A PRINT statement 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 every 14 spaces, which are called print zones.

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

NOTE: A comma (,), semicolon (;), or blank space can be omitted, except for punctuation within a string (where a variable is enclosed by quotation marks). In this case, the operation is identical to using a semicolon for punctuation.

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

SAMPLE PROGRAM:

```
10  PRINT"IF YOU DO NOT WANT AN
    INDENTATION,";
20  PRINT"THEN",
30  PRINT"USE A SEMICOLON."
```

PRINT USING/LPRINT USING

FUNCTION: This statement outputs formatted data to the display screen or to a printer.

FORMAT:

```
[PRINT ] USING ⟨ formatting string ⟩;
[LPRINT]       ⟨ numeric expression ⟩
               { [,] ⟨ numeric expression list ⟩
                 [;] }
```

SAMPLE STATEMENT: PRINT USING "## ####";2.3;4567

DESCRIPTION: The PRINT USING statement 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.
- Exponential 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 , which reserves space for one digit and indicates that leading zeros are to be suppressed. For example:

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

   results:

   __3
   %333

   The underscore (_) denotes a blank space.

2. The ".", (decimal point), which specifies the number of digits to the left and right of the decimal point. The digits to the left of "." will always 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 "." 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:

   __2.50
   __2.56
   __2.3%45.0

3. The "," symbol (comma), which is used anywhere within the ⟨ formatting string ⟩, after the first character and before the decimal point. It punctuates the printed number with "," 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:

```
2,222.200
%123,456.00
_1235.,
```

4. The "+" symbol (plus sign), which is used at the beginning or at the end of the ⟨ formatting string ⟩, and 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:

```
_+2.00
34.5+
_−3.00
_34.5−
12,245.6+
```

5. The "−" symbol (minus sign), which is used only at the end of the ⟨ formatting string ⟩, and specifies the sign (+ or −) of the ⟨ numeric expression ⟩. For example:

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

will result:

```
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 "###.###^^^^";123456
PRINT USING "#.###^^^^";1234567
PRINT USING "#.###^^^^";-1234567
```

will result:

```
_12.346E+04
0.123E+07
-.123E+07
```

7. The "**" (asterisks), which are used at the beginning of the ⟨ formatting string ⟩, and 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:

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

NOTE: When characters that are not described above are used, they wil be printed before or after any numeric values.

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

SAMPLE PROGRAM:

```
10  PRINT"LET'S CREATE TWO HUNDRED
    RANDOM NUMBERS OF FOUR COLUMNS
    EACH."
20  FOR I=0 TO 24
30  FOR J=0 TO 7
40  R=RND(1)*10000
50  PRINT USING"####";R;
60  NEXTJ,I
```

PSET

FUNCTION: This statement sets a desired dot pattern on the LCD screen of the PC-8201.

FORMAT: PSET (〈horizontal coordinate〉, 〈vertical coordinate〉 { ,〈function code〉 })

SAMPLE STATEMENT: PSET (80,32)

DESCRIPTION: The PSET statement sets dots on the screen at the designated coordinates. The 〈 vertical 〉 and 〈 horizontal 〉 coordinates of the 〈 function code 〉 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 generally treated as 239, and if the 〈 vertical coordinate 〉 is greater than 63 it is generally treated as 63.

When the 〈 function code 〉 is an even number, the PSET command reverses, and operates exactly the same way as the PRESET command. If the 〈 function code 〉 is an odd number, the command operates the same as if it was omitted.

SEE ALSO: PRESET statements.

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
```

READ

FUNCTION: This statement is used to read a value from a DATA statement and assign data to a variable.

FORMAT: READ ⟨ variable list ⟩

SAMPLE STATEMENT: READ A,Z,H$

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

A sirgle READ statement may access one or more DATA statements (accessed in order). In addition, multiple READ statements may access a single DATA statement. If the number of cata items in the DATA statement is less than the variables specified in the ⟨ variable list ⟩, an "?OD Error" (out of data) message is displayed.

When designated variables in the ⟨ variable list ⟩ are less than the amount of data in a DATA statement, the next READ statement accesses data not read previously. If no more READ statements are coded in the program, any extra data is ignored.

If repeat utilization of the same data in a program is necessary, the RESTORE statement can make this possible by recycling through the complete or partial set of DATA statements.

SEE ALSO: The RESTORE and DATA statements.

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-8201), am, reading,
    data.
```

REM

FUNCTION: The REMARK statement is used to put non-executable remarks or comments in a program.

FORMAT: [REM] ⟨ remark ⟩

SAMPLE STATEMENTS:
```
REM THIS IS A TEST PROGRAM
' THIS IS A TEST PROGRAM
```

DESCRIPTION: The REM statement is used to input 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 statements are output unchanged. REM statements may be used in multi-statement lines only as the last statement. This is because all statements that follow the REM statement in the multi-statement line are treated as the ⟨ remark ⟩, and they will 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 subsrituted for
    the keyword "REM" in a REM statement.
40  REM The PC-8201 disregards anything
    in a REM statement that follows the
    keyword "REM".
50  REM Any commands that follow a REM
    statement in the same line will also
    be disregarded.
60  PRINT"HOWEVER, THE REVERSE WITH A
    REM STATEMENT AFTER ANOTHER
    STATEMENT IN A LINE IS POSSIBLE.:
    REM This is useless."
```

RENUM

FUNCTION: This command is used to reorganize the lire numbers of a program.

FORMAT: RENUM { ⟨ new line number ⟩ } { ,⟨old lire 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 current program. Optional ⟨ increment ⟩ is the amount that each subsequent line number is to be incremented, with the default value being 10.

The RENUM command can renumber lines used in conjunction with the GOTO, GOSUB, ON. .GOTO, ON. .GOSUB, THEN RESTORE statements, and ERL function. 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 a case, an erroneous line number (*llll*) cannot be modified via the RENUM command, but line number (yyyy) can be altered.

The PC-8201 returns to Direct Mode after the RENUM command is executed.

NOTE: The RENUM command 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 cannct be written in excess of 65529, or else an "?FC Error" (Illegal Function Call) message will occur.

RESTORE

FUNCTION: The RESTORE statement is used to manipulate the data list pointer, and thus re-use data elements from the DATA statement.

FORMAT: RESTORE { ⟨ line number ⟩ }

SAMPLE STATEMENT: RESTORE 80

DESCRIPTION: The RESTORE statement 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 statement in the program is accessed by the next READ statement.

IF ⟨ line number ⟩ is specified, the first item of the DATA statement (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."
```

RESUME

FUNCTION: This statement 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 statement terminates an error handling routine and the parameter specifies NEXT action when program execution continues. This statement functions in a manner similar to the RETURN statement, but may only be used in an error routine, and then returns control to BASIC after an error processing routine has been performed.

Depending 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 to be transferred to the line specified.

SEE ALSO: The ON ERROR GOTO statement.

RETURN

FUNCTION: The RETURN statement terminates execution in a subroutine and returns control to the statement following the GOSUB (call) statement.

FORMAT: RETURN { ⟨ line number ⟩ }

SAMPLE STATEMENTS:
RETURN
RETURN 200

DESCRIPTION: The RETURN statement from the subroutine transfers control to the first statement which follows the GOSUB statement.

If an optional ⟨ line number ⟩ is included with the RETURN statement, program execution transfers to the line number specified, and the statement following the GOSUB call is discarded.

A GOSUB statement is used wher performing (call ng) subroutines. If a GOSUB is not executed first, and a FETURN is encountered an "?RG Error" (Return without gosub) message will be displayed.

A subroutine can have more than one RETURN statement. Orly one RETURN statement is executed each time a subroutine is called.

NOTE: If a CLEAR command 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) message results when the RETURN statement is reached.

SEE ALSO: See the CLEAR, GOSUB . . . RETURN, and ON . . . GOSUB statements.

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$ statement returns a null string when the ⟨ numeric expression ⟩ is 0.

SEE ALSO: The LEFT$ and MID$ functions.

SAMPLE PROGRAM:

```
10  A$="CONTEST"
20  B$=RIGHT$(A$,4)
30  PRINT"THE ";RIGHT$("ALRIGHT",5);
    "$ FUNCTION PASSED THIS ";B$;"."
40  END
```

RND

FUNCTION: The RND function generates a uniformly distributed random number between 0 and 1.

FORMAT: RND (⟨ numeric expression ⟩)

SAMPLE STATEMENT: PRINT RND (9.9)

DESCRIPTION: The RND (Random) function is used whenever you want the PC-8201 to pick a number, flip a coin, draw a card, etc.

The random number that is furnished 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 same number as the most recent one designated is generated repeatedly.
- 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
```

RUN

FUNCTION: This statement 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 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 statement, 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-8201 will display an "Ok" message and execution is not 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 the data recorder is loaded and executed. If option "R" is included, it will open all data files.

When a RUN statement is executed all open files are closed, and the contents of the BASIC area is cleared when the program is loaded.

The PC-8201 reverts back to Direct Mode after program execution is completed.

NOTE: The loading for RUN "CAS:" can be interrupted by pressing both the SHIFT Key and STOP Key at the same time.

SAMPLE PROGRAMS:

```
5   'SAVE  THIS PROGRAM UNDER  THE  NAME
    RUN 1
10  REM ** RUN 1 **
20  REM It's not easy to use a "RUN"
    command within an actual program.
30  PRINT"IF IT RUNS, THE PROGRAM WILL
    NOT STOP."
40  PRINT
50  PRINT"PRESS THE STOP KEY!"
60  PRINT
70  RUN "RUN 2"


5   'SAVE  THIS PROGRAM UNDER  THE  NAME
    "RUN 2"
10  REM ** RUN 2**
20  PRINT"NOW, RUN 2 IS BEING EXECUTED."
30  PRINT
40  PRINT"NEXT, LET'S RETURN TO RUN 1."
50  PRINT
60  RUN "RUN 1"
```

SAVE

FUNCTION: This command 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 command saves a program currently in the memory into RAM or onto external devices. The designated ⟨ file name ⟩ can be six characters or less. When an identical ⟨ file name ⟩ is specified, (compared to an existing file name) the original file content will be overwritten. After the command is executed, the PC-8201 returns to Direct Mode.

The PC-8201 saves a program file from the RAM if ⟨ external device name ⟩ is omitted. When ⟨ external device name ⟩ is specified, "CAS:" is designated for data recorder, "COM:" is designated for an RS-232C circuit, and "LPT:" is used to designate a printer.

For more details, please refer to the CSAVE command for "CAS:", the OPEN command for "COM:", and the LLIST command for "LPT:".

File type ".BA" is automatically selected if none is specified. If file type ".DO" is 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 an identical file name, until a KILL command is executed, or when a Cold Start is performed.

An "?FC Error" (Illegal function call) message will be displayed if 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 command.

The LIST command can be executed before the SAVE command. This is to display the program content before saving, and any required changes can then be made.

If screen editing is performed while a program is in access mode (indicated by an asterisk when the FILES command is executed), the original statement(s) is rewritten 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 creating a ".DO" (ASCII format) file on cassette. The SHIFT and STOP Keys can be pressed simultaneously to interrupt the SAVE "CAS:" command.

SEE ALSO: The CSAVE, LOAD, LLIST, BSAVE, and OPEN "COM:" commands, and Chapter 5, Files.

SCREEN

FUNCTION: This statement establishes the display mode.

FORMAT: SCREEN 0, ⟨ function key display switch ⟩

SAMPLE STATEMENT: SCREEN 0,0

DESCRIPTION: The SCREEN statement establishes the display mode.

When the ⟨ function key display switch ⟩ is 0, the function key is not indicated and display is 8 lines long.

The first parameter is dammy and can be omitted, and the comma is always needed. For example:

SCREEN 0, 1 (function key display enable)

SCREEN 0, 0 (function key display disenable)

The ⟨ function key device switch ⟩ must be in the range from 0 to 255, or else an error occurs.

SEE ALSO: The CLS statement.

SAMPLE PROGRAM:

```
10  FOR I=0 TO 21
20  SCREEN 0,I MOD 2
30  NEXT
```

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 positve, 0 if the ⟨ numeric expression ⟩ is 0, and −1 is returned 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 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 ⟩ determines the angle expressed in radians.

NOTE: To convert ar angle from degrees to radians, multiply it by .0174533.

SEE ALSO: The ATN, COS, and TAN functions.

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
```

SOUND

FUNCTION: This command produces a designated sound.

FORMAT: SOUND ⟨tone⟩,⟨length⟩

SAMPLE STATEMENT: SOUND 5586,50

DESCRIPTION: This command is designated by tone and length , which produce a sound. The integers for the tones range from 0 through 16383, where higher numbers produce a higher pitch tone. Length is comprised of integers within a range of 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.

MUSCIAL SCALE TABLE:

		OCTAVE					
		1	2	3	4	5	6
CODE	C	–	9394	4697	2348	1171	587
	C#	–	8866	4433	2216	1103	554
	D	–	8368	4184	2092	1045	523
	D#	15800	7900	3950	1975	987	493
	E	14912	7456	3728	1864	932	466
	F	14064	7032	3516	1758	879	439
	F#	13284	6642	3321	1660	830	415

	1	2	3	4	5	6
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: The BEEP statement.

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
```

SPACES

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: The TAB function.

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
```

STOP

FUNCTION: The STOP statement is used to halt program execution and return to Direct Mode.

FORMAT: STOP

SAMPLE STATEMENT: STOP

DESCRIPTION: When a STOP statement is executed, the PC-8201 halts the execution of a program. The following message s displayec on the screen. "Break in *llll*" is displayed, with "*llll*" representing the line number that the STOP statement has executed.

A STOP statement differs from an END statement because STOP does not close the file. This statement is useful for debugging programs. The execution o: the program can be resumed by using the CONT command, unless the program has been altered while stopped.

SEE ALSO: The CONT command

SAMPLE PROGRAM:

```
10  PRINT"Use a STOP command for
    debugging."
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
```

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. This function 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: The VAL and STRING$ functions.

SAMPLE PROGRAM:

```
10  PRINT"ENTER A 1 OR 2 DIGIT NUMBER"
20  INPUT"NOW, WHAT HOUR IS IT";H:H$=
    MID$(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$=MID$
    (STR$(S),2)
70  IF LEN(S$)=1 THEN S$="0"+S$
80  TIME$=H$+":"+M$+":"+S$
90  PRINT"THE TIME HAS NOW BEEN SET AT"
    ;TIME$;"."
```

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 by the ⟨ numeric expression ⟩.

The ⟨ numeric expression ⟩ must be in the range of 0 to 250. If it is not within this range, a "?TM Error" (Type Mismatch) message is displayec. The ⟨ ASCII code ⟩ is converted to its equivalent character code and then it is returned by the function.

If the ⟨ character string ⟩ is more than one character, only the first character is returned.

SEE ALSO: The STR$ function.

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'
```

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 and "?FC Error" (Illegal function call) message will be displayed on the screen.

The cursor position does not move backward, so if the position specified by the ⟨ numeric expression ⟩ is left of the cursor, the TAB function will start displaying from the right side of the cursor.

The TAB function is only used with the PRINT and LPRINT statements.

NOTE: You can use more than one TAB function on the same line.

SEE ALSO: The SPACE$ function.

SAMPLE PROGRAM:

```
10  FOR I=1 TO 21 STEP 4
20  PRINT STRING$(I,"#");TAB(22-I);'*"
30  NEXT
```

TAN

FUNCTION: This function provides 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. The unit of the ⟨ numeric expression ⟩ is the angle expressed in radians.

NOTE: To convert an angle from degrees to radians, multiply the degrees by .0174533.

SEE ALSO: The ATN, COS, and SIN functions.

SAMPLE PROGRAM:

```
10  INPUT"ENTER AN ANGLE IN DEGREES";D
20  PRINT"THE ";D;" DEGREES ANGLE IS";
    D*.0174533;"RADIAND AND ITS TANGENT
    IS";TAN(D*.0174533)
30  END
```

TIME$

FUNCTION: This function provides the time from the interral real-time clock of the PC-8201.

FORMAT: TIME$="⟨ hour ⟩ : ⟨ minute ⟩ : ⟨ second ⟩"

SAMPLE STATEMENTS:
```
TIME$="15:30:20"
PRINT TIME$
```

DESCRIPTION: The TIME$ function 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. Reset is not necessary once the time has been set, unless a Cold Start is performed.

SEE ALSO: The DATE$ function.

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 frst 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 ⟩, anythirg after that character is ignored. For example:

VAL("123a4") = 123
VAL("ab") = 0

SEE ALSO: The STR$ and CHR$ functions.

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 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)

For more details on logical operators, see Chapter 3.

NOTE: The XOR function performes exactly opposite from the EQV function.

Logical operators work by converting their ⟨ operands ⟩ to sixteen bits binary integers. Therefore, the ⟨ operands ⟩ must be in the

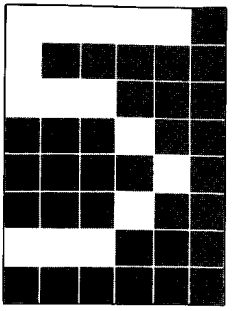
range from −32768 to +32767. If the ⟨ operands ⟩ are not within this range, an "?OV Error" (Overflow) message will be displayed on the screen.

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 is 0000 0000 0001 0100. By looking at the table in the DESCRIPTION section above, notice that the computation is correct.

SEE ALSO: The AND, EQV, IMP, NOT, and OR functions.



Files

CHAPTER 5

Files

A file is a collection of records in the RAM of the PC-8201 or external 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-8201 uses the record unit to read or write into a file, and each file is designated a distinct file name when the file is created.

File Names

A file name consists of three parts:

- The main name, which must be no more than 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.

The file name can consist of any combination of characters, however the use of letters instead of numbers or symbols is recommended. You run the risk of getting the error message "?NM Error" (Name Error) 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:

PC8201.BA

The ".BA" is the extension added by the PC-8201 when the file was saved.

The file name may be input in either upper or lower 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 input by you.

The extensions represent specific file types:

- ".BA" BASIC file. BASIC programs are in Binary format.
- ".DO" TEXT file. TEXT and BASIC programs are in ASCII format.
- ".CO" Machine Language file. Programs and data are in Machine Language format.

The file type extension can be input by you, or the PC-8201 will assign one according to the mode you are using. For the BASIC mode, the file type extension assigned by the PC-8201 would be ".BA".

The file names are displayed on the MENU screen in the following order:

Machine Language files

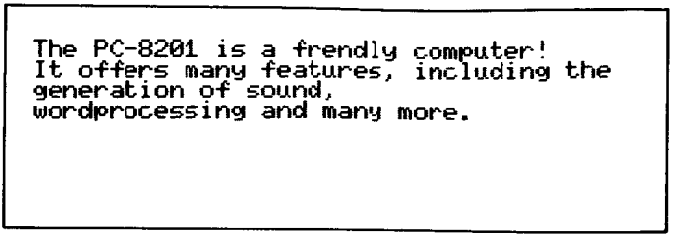
TEXT files

BASIC files

You can also display the file names within the specific bank when in the BASIC mode by using the "FILES" command. It is possible to execute BASIC programs from the MENU mode.

EXAMPLE:

Move the cursor onto the word "PC8201.BA" and then press the Key. The PC-8201 is now in the EASIC mode and the previously created BASIC program "PC8201.BA" is executed. The screen will appear as shown:



Buffers

Buffer memory is reserved RAM area that is used by the PC-8201 to store transmitted and received data. Each time you OPEN a file thru BASIC you reserve a buffer area. The maximum number of OPEN files that are open at the same time is 15. This means that the maximum number of buffers that you can reserve is also 15.

File Handling

In order to read or write to a file you will have to prepare the file for this. This is done by the use of the OPEN command. The OPEN command utilizes the file number in conjunction with the file descriptor to assign a specific buffer area to that file.

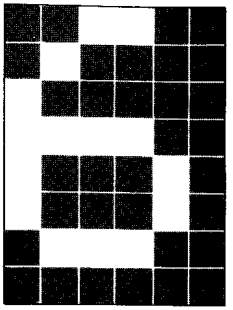
After a file has been OPENed you can use the READ command to read records and the PRINT command to write records. When you have completed your processing you will have to close the file by the use of the CLOSE command.

Precautions for File Creation

When accessing files within the RAM of the PC-8201, 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-8201 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" is displayed.

When a Machine Language file is saved using the BASIC language "BSAVE" command, 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 the file does not run.



Machine Language Programming

CHAPTER 6

Machine Language Programming

Machine Language Programming is a collection of meaningful coced instructions that the PC-8201 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 less memory, and they have virtually no limit to the things they can be programmed to do.

With Machine Language programs you have the ability to get into any memory location of the PC-8201. 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.

If you alter vital memoty locations, such as the programs that operate the PC-8201, you could get the PC-8201 into a "hung up" situation, meaning that it does not respond, no matter what you input.

In the case of such a problem, you will have to perform a Cold Start. After a Cold Start only the primary programs of BASIC, TEXT and TELCOM are displayed on the screen. The rest of the files are destroyed. This is why it is so important to save your files before attempting to run your Machine Language program.

See the User's Guide for more detail on how to execute a Cold Start.

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 use the optional Assembler Language compiler to create Machine Language code, or use the POKE command to actually create a Machine Language routine in the PC-8201 RAM.

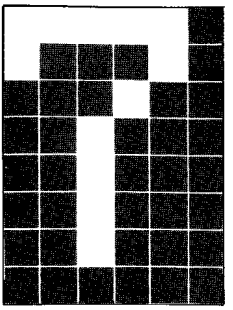
Since creating a Machine Language program is tedious work, make sure you save it using the BSAVE command before attempting to test it, which avoids the loss of effort. When debugging (testing) your Machine Language programs you can use the PEEK command to check the value of a specific memory location.

See Chapter 4 for an explanation on how to use the BSAVE, POKE, and PEEK commands.

Once the Machine Language routine has been tested and saved, the BLOAD command can be used to load your program into the PC-8201 RAM. The EXEC command 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 the routine.

For more details on BLOAD and EXEC commands, please refer to Chapter 4.

The Machine Language program should include a RET command at the end of the routine, so control can be returned to BASIC mode.



N82-BASIC Programming

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 helping to resolve problems within those programs.

Recovery from Different Critical Situations

Wrap Around 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.

Wrap around 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.

- Wrap around occurs 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 is the process when 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 occurs if the cursor is at the last line and a wrap aroud is encountered.

Spontaneous Program Execution Errors

SITUATION:

A program started to operate ircorrectly 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-8201, they can be modified and stored in the RAM or on external devices, such as a Data Recorder.

When a program is loaded from the RAM and needs modification, this program should be saved again in the RAM and not on external devices. If a program is loaded from a cassette tape, do not save it in the RAM unless it is free of errors and operates the way it should.

When loaded files from tape are modified and then SAVEd in the RAM, the display of the file name includes an asterisk (*) after the file type extension, when the FILES command is used. It is important to recognize that these modified programs may contain potential errors when attempting to LOAD the original file from tape, and the bad file can mistakenly be loaded.

Logical Errors

SITUATION:

When the program result is different than 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-8201 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 NEXY
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 6 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 add elements 1 to 6 instead 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 the ↵ Key and you will see that the result of 69 is now correct.

Loss of Program Control

SITUATION:

The [STOP] Key is ineffective and you have no control over a program.

EXPLANATION:

In this situation you may have temporarily overlayed vital routines through the use of a POKE command or through your own Machine Language programs. These vital routines include the information that the PC-8201 utilizes for its operation.

Files stored in the RAM are erased when this situation is encountered. The only option you have at this point is to turn the power switch OFF. When the power is turn ON again, no files are displayed on the MENU screen except the primary files of BASIC, TEXT, and TELCOM.

If the PC-8201 still does not operate correctly in some way, conducting a Cold Start is necessary. To do this, press the [CTRL] Key and the [SHIFT] Key simultaneously, while the Reset Switch on the back of the PC-8201 is pressed. If necessary, refer to the User's Guide.

Return to BASIC from TEXT is Impossible

SITUATION:

When editing a BASIC program within the TEXT Mode, it may be impossible 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-8201 locks you out and pressing the STOP Key or the f.10 Function Key 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.

Programming Hints

Hints for Detecting Errors:

1. A flowchart (a chart depicting the course of program operations) should be carefully constructed. This is especially useful when beginning programmers are suddenly confronted with a major error in the middle of a program.

2. The PC-8201 User's Guide and this N82-BASIC Reference Manual should be carefully read and you should understand and try out the commands and functions utilized by the PC-8201.

3. A chart 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 extensive REM statements and avoid multiple statements as much as possible, which makes the program easy to understand when searching for errors.

5. If a particular line does not work at all, isolate it by means of a REM statement rather than eliminating it. You can then easily modify it later.

6. Use a STOP statement to confirm any changes in the value of a variable. A CONT command can be used during this process.

Hints for Speeding Up Program Execution:

1. Spaces and REM statements should be eliminated.

2. Integer variables should be used whenever possible.

3. Omit a control 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 for string use is adequate.

8. Try to simplify the process of frequently used loops.

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 with a variable, no matter how many times a constant appears within a program.

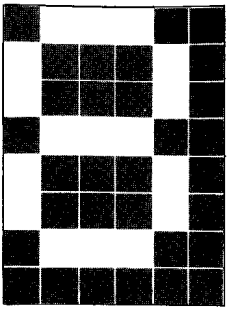
4. Utilize old variables no longer being used within a program, instead of defining new variables.

5. When there are numerous situations where the same process is being conducted, consider ordering these by directing them through a single 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.



Error Messages

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-8201. 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.

Error Messages

MESSAGE: **?AO ERROR** File s Already Open.

POSSIBLE CAUSES:

1. The execution of an OPEN statement for a file already opened.
2. The execution of a KILL statement for an open file.

USER ACTION: Close the file using the CLOSE commanc before trying to OPEN it or to KILL it.

MESSAGE: **?BN ERROR** Bad file Number is used.

POSSIBLE CAUSES:

1. When a PRINT statement is used with a file number nor previously designated by an OPEN statement.

2. When an OPEN statement is used to assign a file number larger than the maximum number designated by a MAXFILES command.

USER ACTION:

1. OPEN the file.

2. Use the MAXFILES command to assign the desired number of files.

MESSAGE: **?BO ERROR** Buffer 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 subscript 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 array dimensions if necessary.

MESSAGE: **?CE 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 statement is used after a break occurs in program execution and the program is then edited.
2. When a CONT command is written as a statement within a program.
3. When a CONT statement is used after a break occurs in program execution, following a CLEAR statement.

USER ACTION:

1. Return the program by using a RUN command.
2. Eliminate the CONT statement from the program content.
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 command.

USER ACTION: Use the CLEAR command within the program to clear all arrays so that they can be re-defined. When using the NEW or RUN command all arrays will be cleared.

MESSAGE: **?DS ERROR** Direct Statement in File

POSSIBLE CAUSE: When loading a file using the LOAD command with a file type extension of ".DO", and 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: **?DU 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-8201.

MESSAGE: **?EF ERROR** End of File

POSSIBLE CAUSE: When using the INPUT statement or LINEINPUT statement beyond the end of the file.

USER ACTION: Use the EOF command in conjuction with INPUT or LINEINPUT commands 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 with LOG
3. A negative argument to SQR or CLEAR
4. When ".BA" files are combined with a MERGE command.
5. When a RENUM statement is used improperly and line sequence is changed.
6. When a device is used that is not connected or is incorrectly connected to the PC-8201.
7. When parameter values are not within the proper range for CLOSE, ERROR, LOCATE, MOTOR, GOTO, GOSUB, OUT, POKE, POWER, PRESET, SCREEN, CHR, EOF, INP, INPUT, INST, LEFT, MID, RIGHT, SPACE, STRING, TAB, KEY, MAXFILES, and SOUND statements.

USER ACTION:

1. Be sure all peripheral devices used with the PC-8201 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 command is not on a designated device. If the device designated is a Data Recorder, the PC-8201 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 command.

USER ACTION:

1. Be sure all files loaded with the BLOAD command are ".CO" files.
2. Use the SHIFT Key and the STOP Key simultaneously to interrupt the searching and try the command with the correct name.

MESSAGE: **?FL ERROR** Filing Limit

POSSIBLE CAUSE:

When the MENU director is filled with file names, and no space is avilable for display of a new file name. Memory bytes may still be free.

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 the SHIFT Key and STOP Key are pressed to forcibly stop input or output to an external device.
2. When peripheral equipment is in need of maintenance.

USER ACTION: Check ecuipment if error occurred spontaneously. May neec maintenance such as cleaning of Data Recorder 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 limit of 255 characters. If the

string was made too long in error, simply change the length designated in the program.

MESSAGE: **?MO ERROR** Missing Operand

POSSIBLE CAUSE: A necessary operand is missing.

USER ACTION: Check the program and insert the omitted parameter.

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 statement without the previous execution of a corresponding FOR.
2. When a GOTO or GOSUB subroutine causes a p program to jump into a FOR NEXT loop.
3. When a FOR NEXT loop is improperly 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 if necessary.
3. Correct improper nesting of FOR NEXT loops.

See Chapter 4 for rules for the use of 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. Correct file name to follow conventions exactly.
2. Be sure that correct 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 statement do not correspond to the number of elements within the DATA statements.
2. When a RESTORE statement is not used at all, or is improperly used.

USER ACTION:

1. Check the program to be sure the number of elements designated for READ and DATA statements correspond.

2. Be sure the program includes a RESTORE statement in the appropriate place, before trying to read DATA elements that have been previously read.

See Chapter 4 for correct use of the RESTORE statement.

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 statements.

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 becomes 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: A sufficient working memory area for string handling has not been maintained.

USER ACTION: Utilize the CLEAR command 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 command 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 statements are not within the proper range.

USER ACTION: Rearrange operations within the program so that they flow 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

POSSIBLE CAUSE: When an N-BASIC program, which cannot be executed in N82-BASIC, is loaded into the PC-8201.

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 statement without a corresponding GOSUB statement.

USER ACTION:

1. Make sure you are ont using a GOTO to execute a subroutine.

2. Make sure to use an END statement, so the program does not fall through ary possible subsequent subroutines.

MESSAGE: **?RW ERROR** Resume Without error

POSSIBLE CAUSE: A RESUME statement is encountered before an error trapping routine is entered.

USER ACTION:

1. Check for any other GOTO's or GOSUB's to error trapping routines, except by using the ON ERROR command.

2. Check for END statement, so 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 command.

MESSAGE: **?SN ERROR** Syntax Error

POSSIBLE CAUSES:

1. When a statement or a command does not agree with the grammar of BASIC.

2. When there is only a function or mathematical expression on the left side of a substitution formula (although it can normally 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 is used without a THEN in terms of an IF statement.

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 command. In most cases, the number of the line in which the error has occurred will be displayed, after the f.9 Function Key is pressed.

2. If two lines are joined together, edit this excessively long line in the TEXT mode.

3. Check for an accidental substitution, (1 and l, a period and a comma, a colon and a semicolon, etc.).

4. Check names of variables that might contain a Reserved Word (a keyword), for instance, **COS**T, SHI**F**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: Expression should be broken into smaler 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 expects 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 statement.

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 number exists but one has been designated by a RESTORE or RUN statement.
3. When a program has nonexistent line for GOTO or GOSUB.

USER ACTION: Correct program references for line numbers.

MESSAGE: **?/0 ERROR** Division by zero

POSSIBLE CAUSES:

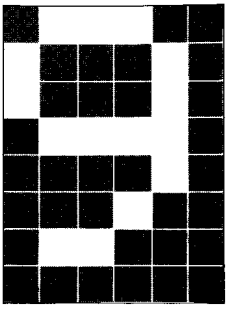
1. When division is performed with an undefined variable, (and its initial value has been set at zero).

2. When the variable that comprises the resultant divisor of an operation is zero.

3. When the dummy variable of a TAN function is $\pi/2$.

4. When multiplication is performed on zero by a negative exponent.

USER ACTION: Have the value of the variable displayed by the PRINT statement. Attempt to investigate the portion where the operation has been run that has used that variable within the program in terms of zero.



Sample Programs

CHAPTER 9

Sample programs

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 130,260,400
120 BEEP: GOTO 20
130 '                LINE
140 CLS:PRINT
150 INPUT"COORDINATE FOR POINT X";X0: IF X0<0
    OR X0>239 THEN BEEP: GOTO 150
170 INPUT"COORDINATE FOR POINT Y";Y0: IF Y0<0
    OR Y0>63 THEN BEEP: GOTO 170
190 INPUT"COORDINATE FOR ENDPOINT X";X1:IF X1<0
    OR X1>239 THEN BEEP:GOTO 190
200 INPUT"COORDINATE FOR ENDPOINT Y";Y1:IF Y1<0
    OR Y1>63 THEN BEEP:GOTO 210
230 CLS:GOSUB 520
240 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
260 '                BOX
270 CLS:PRINT
290 INPUT"X COORDINATE";X0:IF X0<0 OR X0>239
    THEN BEEP:GOTO 290
310 INPUT"Y COORDINATE";Y0:IF Y0<0 OR Y0>63
    THEN BEEP:GOTO 310
330 INPUT"SECOND X COORDINATE";X1:IF X1<0 OR
    X1>239 THEN BEEP:GOTO 330
```

```
350 INPUT"SECOND Y COORDINATE";Y1:IF Y1<0 OR
    Y1>63 THEN BEEP:GOTO 350
370 CLS:GOSUB 660
380 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
400 '                CIRCLE
410 CLS:PRINT
420 PRINT"CENTER COORDINATES:"
430 INPUT"X COORDINATE";X0:IF X0<0 OR X0>239
    THEN BEEP:GOTO 430
450 INPUT"Y COORDINATE";Y0:IF Y0<0 OR Y0>63
    THEN BEEP:GOTO 450
470 INPUT"RADIUS";R:IF R<0 THEN BEEP:GOTO 470
490 CLS:GOSUB 740
500 FOR I=0 TO 1000:NEXT:BEEP:GOTO 20
520 '                SUB LINE
530 XD=ABS(X1-X0):YD=ABS(Y1-Y0)
540 XS=SGN(X1-X0):YS=SGN(Y1-Y0)
550 IF XD>YD THEN 600
560 F=-1:T=X0:X0=Y0:Y0=T
570 T=X1:X1=Y1:Y1=T
580 T=XD:XD=YD:YD=T
590 T=XS:XS=YS:YS=T
600 R=XD/2
610 IF F THEN PSET(Y0,X0) ELSE PSET(X0,Y0)
620 IF X0=X1 THEN RETURN
630 X0=X0+XS:R=R+YD
640 IF R>=XD THEN R=R-XD:Y0=Y0+YS
650 GOTO 610
660 '                SUB BOX
670 FOR I=X0 TO X1 STEP SGN(X1-X0)
680 PSET(I,Y0):PSET(I,Y1)
690 NEXT
700 FOR I=Y0 TO Y1 STEP SGN(Y1-Y0)
710 PSET(X0,I):PSET(X1,I)
720 NEXT
730 RETURN
740 '                SUB CIRCLE
750 FOR I=0 TO 1 STEP 1/(R*2)
760 II=I*I
770 X=R*I*2/(II+1)
780 Y=R*(1-II)/(II+1)
790 X2=X0-X:IF X2<0 THEN X2=0
800 Y2=Y0-Y:IF Y2<0 THEN Y2=0
810 X1=X0+X:Y1=Y0+Y
820 PSET(X1,Y1):PSET(X1,Y2)
830 PSET(X2,Y1):PSET(X2,Y2)
840 NEXT
850 RETURN
```

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 graphics 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 command, to bring out a hundred or even a thousand graphics characters to work with if so desired.

Since characters can be skipped over when the ESC Key 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 is newly defined, it is also possible to eliminate all if so desired).

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 pressing the GRAPH Key and any other key at the same time.

```
 10 REM COPYRIGHT (C) NEC 1983
100 REM CHARACTER GENERATOR
110 REM USING ADRESS E960-EACF
120 CLEAR 256,59743!:DIM M(5,7):DEFINTB-Z
130 REM ***** INITIALIZE *****
140 SCREEN 0,0:CLS
150 POKE 65215!,96:POKE 65216!,233
160 H=131:C=0:AD=59744!
170 REM ***** MAIN LOOP1 *****
180 LOCATE 20,0:PRINT " USING KEY"
190 LOCATE 15,1:PRINT "SPACE = MODE"
200 LOCATE 15,2:PRINT "CURSOR = MOVE"
210 LOCATE 15,3:PRINT "'ESC' = NEXT"
220 LOCATE 15,4:PRINT " = DEFINE CHARACTER"
230 LOCATE 15,5:PRINT "E = END"
240 LOCATE 10,7:PRINT "CHR$(";
250 PRINT MID$(STR$(H),2);")BEING DEFINED";
260 X=0:Y=0:MX=0:MY=0:H=H+1:IF H=160 THEN H=224
270 FOR Y1=0 TO 63:PSET(36,Y1):NEXT:
280 REM ***** MAIN LOOP2 *****
290 IF T=0 THEN C$="ERASE" ELSE C$="WRITE"
```

```
300 LOCATE 10,0: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(131-159)"
610 FOR I=131 TO 159
620 PRINT CHR$(I);" ";:NEXT:PRINT
630 PRINT "AND(224-255)"
640 FOR I=224 TO 255
650 PRINT CHR$(I);" ";:NEXT:PRINT
660 INPUT"BSAVE (Y/N)";Y$
670 IF Y$="Y" OR Y$="y" THEN INPUT"FILE NAME";N$
    ELSE END
680 REM ***** FILE SAVE *****
690 BSAVE N$,59744!,366
700 END
```

Music Program

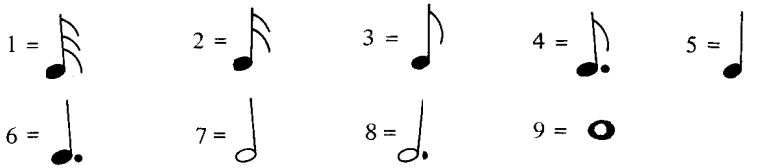
The SOUND command in N82-BASIC can be used to create sophisticated music compositions consisting of simple half notes. The number 1 parameter determines the precise musical step. The SOUND command will also work quite effectively in programs where a composition is to be performed. The program that follows is exclusively for musical composition.

The keyboard of the PC-8201 is turned into an actual keyboard of a musical instrument in terms of input. This keyboard input is organized in the following order of input:

a) Length of note (the 'L' Key + a length designation between 1 and 9 with an initial automatic designation of '5');

b) Octave (the 'O' Key + an octave designation between 1 and 4 with an initial automatic designation 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 the 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 values indicated above. A single note at a time can be modified by using the ESC Key.

It is a 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 capical letters "A" through "G" the sharps displayed as lower case letters that correspond to "1a" through "so" (in the key of C). The input process can be stopped at any time by the "Q" Key.

The data can be performed after it 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 *** MUSIC ***
 30 CLEAR 2000!:MAXFILES=1
 40 DEFINT A-T:DEFSNG U-Y:DEFDBL Z
 50 DIM A(48),M$(49),S(500),L(500)
 60 SCREEN 0,0:Z=9394#
 70 FOR I=0 TO 47
 80 A(I)=Z:Z=Z/1.0594639#
 90 NEXT I
100 FOR I=1 TO 9:READ LN(I):NEXT
110 DATA 4,8,16,24,32,48,64,96,128
120 REM *** MENU ***
130 CLS:PRINT" *** MUSIC ***
140 PRINT:PRINT" --- Play or Input ---"
150 PRINT:INPUT"(P/I)";Y$
160 IF Y$="P" OR Y$="p" THEN 200
170 IF Y$="I" OR Y$="i" THEN 710
180 PRINT"????":BEEP:BEEP:GOTO 130
```

```
190 REM *** PLAY ***
200 CLS:PRINT" --- PLAYER ---"
210 PRINT:PRINT"Type music data"
220 INPUT"file name.";N$
230 OPEN N$ FOR INPUT AS #1
240 S=0:E=0
250 IF EOF(1) THEN 280
260 LINEINPUT #1,M$(E)
270 E=E+1:GOTO 250
280 CLOSE:PRINT"End of road."
290 PRINT"Data conversion."
300 PRINT"You may transpose for music from 01G to
    04G."
310 PRINT"You may change to tempo.(but L1=4)"
320 INPUT"Are you change transpose?(Y/N)";I$
330 IF I$="Y" OR I$="y" THEN GOTO 350
340 IF I$="N" OR I$="n" THEN GOTO 350 ELSE BEEP: CLS:
    GOTO 280
350 INPUT"Are you change tempo?(Y/N)";Y$
360 IF Y$="Y" OR Y$="y" THEN GOTO 380
370 IF Y$="N" OR Y$="n" THEN GOTO 380 ELSE BEEP: CLS:
    GOTO 280
380 IF I$<>"Y" AND I$<>"y" THEN 420
390 INPUT" Change transpose of a unit.(from -7 to
    7)";D:IF D<-7 OR D>7 THEN 390
400 IF D>0 THEN FOR I=0 TO 41:A(I)=A(I+D):NEXT:GOTO
    420
410 FOR I=47 TO 7 STEP -1:A(I)=A(I+D):NEXT
420 IF Y$="Y" OR Y$="y" THEN INPUT"V (From .25 to
    2)";V ELSE V=1
430 PRINT" ---Just moment please---"
440 C=0:FOR I=0 TO E-1
450 T$=M$(I):GOSUB 610
460 NEXT I
470 BEEP:CLS
480 PRINT N$;" End of change data."
490 LOCATE 10,3:PRINT N$:LOCATE 10,4
500 PRINT" Hit any key.!"
510 IF INKEY$<>"" THEN 510
520 IF INKEY$="" THEN 520
530 LOCATE 10,4:PRINT SPACE$(14)
540 FOR I=0 TO C-1:SOUND S(I),L(I)*V:NEXT I
550 INPUT"Onece more (Y/N)";Y$
560 IF Y$="Y" OR Y$="y" THEN 490
570 IF Y$="N" OR Y$="n" THEN GOTO 580 ELSE BEEP: CLS:
    GOTO 480
580 IF I$="Y" OR I$="y" THEN PRINT"I must do
    initialize over again.":RUN
590 GOTO 130
600 REM *** DATA COMPILER ***
```

```
610 FOR T=1 TO LEN(T$)
620 N=INSTR("CcDdEFfGgAaB LO",MID$(T$,T,1))
630 IF N>13 THEN GOSUB 670:GOTO 620
640 M=N+M:S(C)=A(M-1):L(C)=L:M=M-N
650 IF N=13 THEN S(C)=0
660 C=C+1:NEXT T:RETURN
670 IF N=15 THEN M=12*(VAL(MID$(T$,T+1,1))-1):T=T+2:
    RETURN
680 L=VAL(MID$(T$,T+1,1)):L=LN(L)
690 T=T+2:RETURN
700 REM *** INPUT ***
710 CLS:PRINT" --- INPUT ---"
720 S=0:E=0:C=0
730 INPUT" Append or New data (A/N)";Y$
740 IF Y$="N" OR Y$="n" THEN GOTO 760
750 IF Y$="A" OR Y$="a" THEN GOTO 760 ELSE  BEEP: CLS:
     GOTO 720
760 INPUT"File name.";N$
770 IF Y$="A" OR Y$="a" THEN OPEN N$ FOR APPEND AS #1
    ELSE 800
780 PRINT"Please input continue":GOTO 820
790 REM *** NEW DATA ***
800 OPEN N$ FOR OUTPUT AS #1
810 PRINT"Please input new music "
820 PRINT"Data."
830 INPUT"Are you want explanation for input?(Y/N)";Y$
840 IF Y$="Y" OR Y$="y" THEN GOSUB 1390
850 IF Y$="N" OR Y$="n" THEN GOTO 860 ELSE BEEP : CLS:
     GOTO 810
860 REM *** KEY INPUT ***
870 CLS:L$="L5":O$="O2":S=C:M$(E)="":B=0:T$="":F=1:
    L=32
880 LOCATE 0,0:PRINT L$
890 LOCATE 3,0:PRINT O$
900 LOCATE 6,0:I$=INPUT$(1)
910 P=INSTR("ZSXDCVGBHNJM LOE"+CHR$(27)+"Q",I$)
920 IF P=0 THEN 900
930 I$=MID$("CcDdEFfGgAaB ",P,1)
940 IF F=1 THEN T$=L$+O$+I$
950 IF F=2 THEN T$=O$+I$
960 IF F=3 THEN T$=L$+I$
970 IF F=0 THEN T$=I$
980 IF B=0 THEN T$=L$+O$+I$
990 IF P=17 THEN IF F<>0 OR B=0 THEN 880 ELSE B=0:
    GOTO 1220
1000 IF P=18 THEN IF S=C THEN E=E-1:GOTO 1250 ELSE
    1250
1010 IF P>13 THEN 1070
1020 X$=T$:B=1
1030 PRINT I$;:M$(E)=M$(E)+T$
```

```
1040 LOCATE 0,5:PRINT M$(E)+SPACE$(10);
1050 GOSUB 610:SOUND S(C-1),L(C-1):F=0
1060 GOTO 880
1070 ON P-13 GOTO 1080,1110,1140
1080 IF S=C THEN F=1 ELSE IF F=2 THEN F=1 ELSE
     F=3
1090 LOCATE 0,0:Y$=INPUT$(1):P=INSTR("123456789",Y$):
     IF P=0 THEN 1080
1100 L$="L"+Y$:GOTO 880
1110 LOCATE 3,0:Y$=INPUT$(1):P=INSTR("1234",Y$):IF P=0
     THEN 1110
1120 IF S=C THEN F=1 ELSE IF F=3 THEN F=1 ELSE
     F=2
1130 O$="O"+Y$:GOTO 880
1140 LOCATE 0,3:PRINT "END OF PART";E;
1150 FOR I=S TO C-1:SOUND S(I),L(I):NEXT
1160 INPUT" OK(Y/N)";Y$:IF Y$="Y" THEN 1200
1170 IF Y$="N" OR Y$="n" THEN GOTO 1190 ELSE BEEP: CLS:
      GOTO 1140
1180 IF Y$="Y" OR Y$="y" THEN GOTO 1190 ELSE BEEP: CLS
     :GOTO 1140
1190 C=S:PRINT"Try again.":BEEP:GOTO 870
1200 S=C:IF E<49 THEN E=E+1:M$(E)="":F=1:B=0:CLS:GOTO
     880
1210 BEEP:PRINT"OUT OF DATA SPACE":GOTO 1280
1220 M$(E)=LEFT$(M$(E),LEN(M$(E))-LEN(X$))
1230 C=C-1:BEEP:LOCATE 0,3:PRINT"1 STEP BACK":
     BEEP
1240 LOCATE 0,3:PRINT SPACE$(12);:GOTO 880
1250 PRINT:PRINT"END OF MUSIC"
1260 C=C+1
1270 REM *** END ***
1280 PRINT"Your music.":FOR I=0 TO 200:NEXT
1290 FOR I=0 TO C-2:SOUND S(I),L(I):NEXT
1300 CLS:PRINT"Save to start."
1310 PRINT"File name.";N$:PRINT"Hit any key."
1320 IF INKEY$="" THEN 1320
1330 FOR I=0 TO E:PRINT #1,M$(I):NEXT I
1340 CLOSE:BEEP
1350 PRINT"End of save. Hit any key."
1360 IF INKEY$="" THEN 1360
1370 GOTO 130
1380 REM *** EXPLAIN ***
1390 PRINT "   EXPLANATIONS."
1400 PRINT"1 Please push 'CAPS' key!."
1410 PRINT"2 'ZSXDCVGBHNJM'keys are music keybord."
1420 PRINT"3 'ZSXDCVGBHNJM'keys changed 'CcDdEFfGgAaB'
     keys."
1430 LOCATE 0,7:PRINT" Hit any key.";
```

```
1440 IF INKEY$="" THEN 1440
1450 PRINT:PRINT"4 Push 'E' key end to one brock."
1460 PRINT"5 Push 'Q' key end of input."
1470 PRINT"6 Push 'ESC' key return one music
     brock."
1480 PRINT"7 Space is a rest."
1490 LOCATE 0,7:PRINT" Hit any key.";
1500 IF INKEY$="" THEN 1500
1510 PRINT:PRINT"8 L=LENGTH(1-9),O=OCTAVE(1-4)"
1520 PRINT"9  input about 20 keys,push 'E' key goto
     next step.!"
1530 PRINT"10 End to part 49."
1540 PRINT"11 'L' and 'O' keys could change many
     times,if you not push 'ESC' key."
1550 LOCATE 0,7:PRINT" Hit any key.";
1560 IF INKEY$="" THEN 1560
1570 RETURN 860
```

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 INTEGRATE this type of process with the Character Definition program introduced previously.

Please use any alphabetical or numerical characters when you run the program.

```
 10 '                    DEMO
 20 CLEAR 256,62336!
 30 SCREEN 0,0:CLS
 40 DIM C%(39,7),X%(319,1):C=0
 50 PRINT "READING DATA"
 60 FOR X=0 TO 39
 70 FOR Y=0 TO 7
 80 X%(Y*40+X,0)=X:X%(Y*40+X,1)=Y
 90 READ C%(X,Y)
100 NEXT Y,X
110 '                   MAKE DATA
120 SCREEN 0,0:CLS:PRINT
130 PRINT "DATA SCRAMBLING"
140 FOR I=0 TO 200
150 R=RND(1)*319
160 R1=RND(1)*319
170 N=X%(R,0):X%(R,0)=X%(R1,0):X%(R1,0)=N
180 N=X%(R,1):X%(R,1)=X%(R1,1):X%(R1,1)=N
190 NEXT
200 '                   PRINT
210 BEEP:CLS:PRINT CHR$(27)+"V"
220 PRINT "HIT ANY KEY";:A$=INPUT$(1)
230 PRINT A$:PRINT
240 PRINT "HIT ANOTHER KEY";:B$=INPUT$(1)
250 PRINT B$:CLS
260 FOR N=0 TO 319
270 X=X%(N,0):Y=X%(N,1)
280 SOUND X*200+200,3
290 LOCATE X,Y
300 IF C%(X,Y)=1 THEN PRINT A$; ELSE PRINT B$;
310 NEXT
320 BEEP:LOCATE 0,0:PRINT A$; ELSE PRINT B$;
330 FOR I=0 TO 500:NEXT
340 LOCATE 0,0:GOTO 130
```

```
350 DATA 0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,
    0,0,0,1,0,0,0,1,0,0,0,1,0,0
360 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
370 DATA 0,0,0,1,1,1,0,0,0,0,1,0,0,0,1,0,0,1,
    0,0,0,0,0,1,0,1,0,0,0,0,0,1
380 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
390 DATA 0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
    1,1,0,1,1,0,0,1,0,0,1,0,0,1
400 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
410 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
420 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
430 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
440 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
```

Game Program

The missile base is moved by using the left and right Cursor Movement keys, while pressing the Space bar shoots a missile. As presently set, the game will end after one minute but play can easily be extended 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:GOSUB
    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 "HOORAY!"
420 SOUND 1760,1
430 NEXT I
440 LOCATE X-1,0:PRINT "        "
450 RETURN
460 LOCATE 10,4:PRINT "END OF GAME":END
```

Score Ranking Program

This program uses the sequential file management function which N82-BASIC contains, 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 specific requirements.

```
 10 SCREEN 0,0:CLS
 20 PRINT "*** RANKING SCORES ***"
 30 PRINT
 40 PRINT"P_EASE 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$(40):BEEP
220   LOCATE 0,2:PRINT "NO.";J;"NAME";
230   INPUT NA$(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 FINISH."
410 OPEN "SCRN:" 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$<>"y"
    THEN 540
470 ON ERROR GOTO 540
480 INPUT "NAME OF FILE";A$
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 TO PRINT IT (Y/N)";
550 Y$=INPUT$(1):PRINT Y$;IF Y$<>"Y" AND Y$<>"y"
    THEN END
560 OPEN "LPT:" FOR OUTPUT AS #1
570 TT=0:GOSUB 600
580 CLOSE#1:END
590 RESUME 480
600 '              OUTPUT SUBROUTINE
610 PRINT#1,SPACE$(12);LEFT$(TI$,30)
620 PRINT#1,
630 PRINT#1,SPACE$(9);
640 FOR I=1 TO NC
650   PRINT#1,LEFT$(ITM$(I)+SPACE$(12),12);
660 NEXT I
670 PRINT#1,"TOTAL  MEAN"
680 FOR J=1 TO NR
690   PRINT#1,LEFT$(NA$(J)+SPACE$(10),10);
700   FOR I=1 TO NC
710     PRINT#1, USING"######      ";D(I,J);
720   NEXT I
730   PRINT#1, USING"#### ####.#";RSUM(J);RMEAN(J)
740   IF TT<>0 THEN IF INKEY$=" " 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);:NEXT
820 PRINT#1,
830 PRINT#1,"MEAN       ";
840 FOR I=1 TO NC
850   PRINT#1, USING"######       ";MEAN(I);:
    NEXT
860 PRINT#1,
```

```
870 PRINT#1,"DEVIATION ";
880 FOR I=1 TO NC
890   PRINT#1, USING"######.#     ";SQR(SD(I));
    :NEXT
900 PRINT#1,
910 RETURN
```

STUDENT ACHIEVEMENT BY SUBJECT

	ENGLISH	MATHEMATICS	HISTORY	TOTAL	MEAN
JOHN	71	78	73	222	74.0
JAMES	53	78	80	211	70.3
MARY	83	62	48	193	64.3
ANN	78	91	45	214	71.3
BOB	73	46	43	162	54.0
HELEN	43	75	72	190	63.3
DORIS	80	71	72	223	74.3
ALEX	78	64	69	211	70.3
LOIS	68	82	70	220	73.3
ADAM	60	58	93	211	70.3
TOTAL POINTS	687	705	665		
MEAN	69	71	67		
DEVIATION	12.3	12.5	15.4		

APPENDICES

APPENDIX A1

Reserved Words

ABS
AND
ASC
ATN
BEEP
BLOAD
BLOAD?
BSAVE
CDBL
CHR$
CINT
CLEAR
CLOAD
CLOAD?
CLOSE
CLS
COM ON/OFF/STOP
CONT
COS
CSAVE
CSNG
CSRLIN
DATA
DATE$
DEFINT/SNG/DBL/STR
DIM
EDIT
END
EOF
EQV
ERL
ERR
ERROR
EXEC
EXP
FILES
FIX
FOR . . . TO . . . STEP ~ NEXT
FRE
GOSUB ~ RETURN
GOTO
IF . . . THEN . . . ELSE
IMP
INKEY$
INP
INPUT
INPUT$
INPUT#
INSTR
INT
KEY
KILL
LEFT$
LEN
LET
LINE INPUT
LINE INPUT#
LIST/LLIST
LOAD
LOCATE
LOG
LPOS
MAXFILES
MENU
MERGE
MID$
MOD
MOTOR
NAME
NEW

NOT
ON COM GOSUB
ON ERROR GOTO ~ RESUME
ON . . . GOTO/GOSUB
OPEN
OPEN "COM"
OR
OUT
PEEK
POKE
POS
POWER
PRESET
PRINT/LPRINT
PRINT USING/LPRINT USING
PSET
READ
REM
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
RUN
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SQR
STOP
STR$
STRING$
TAB
TAN
TIME$
VAL
XOR

APPENDIX A2

Error Codes

Error Message	Code	N82-BASIC Message	Meaning
?AO Error	53	File Already Open	The same file has been opened before.
?BN Error	51	Bad file Number	The number of file is inappropriate.
?BO Error	23	Buffer Overflow	The input buffer has overflowed.
?BS Error	9	Bad Subscript (out of range)	The subscript of the array is inappropriate
?OF Error	58	File not open	The file has not yet been opende.
?ON Error	17	Continuation is Not possible	The execution of the program connot be resumed by means of a CONT command.
?DD Error	10	Duplicate Definition	The same array is declared twice.
?DS Error	56	Direct Statement in file	An ASCII format file does not load.
?DU Error	25	Device Unavailable	A designated device is not being used.
?EF Error	54	End of File	No more data in the file.
?FC Error	5	Illegal Function Call	Commands or Functions are used incorrectly.

Error Message	Code	N82 BASIC Message	Meaning
?FF Error	52	File not Found	The designated name of file can not be located.
?FL Error	57	Fi ing L mit	There are too many files.
?ID Error	12	Illegal Direct	The specified command cannot be used in the dircet mode.
?IE Error	50	Internal Error	An error has occured within BASIC itself.
?IO Error	24	I/O Error	An error occurs during input or output.
?LS Error	15	Long String	The contents of a string variable are in excess of 255 characters.
?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	File Name Mismatch	The name of the file is inappropriate.
?NR Error	19	No RESUME	There is no RESUME command present in an error reutine.
?OD Error	4	Out of Data	The data required to be read is insufficient.

Error Message	Dode	N82 BASIC Message	Meaning
?OM Error	7	Out of Memory	There is insufficient memory.
?OS Error	14	Out of String space	The memory region available for string storage is inadecuate.
?OV Error	6	Overflow	A numerical value is excessive.
?PC Error	59	PC-8001 Command	This command is used on the PC-3001.
?RG Error	3	RETURN without GOSUB	A RETURN statement is present without GOSUB statement.
?RW Error	20	RESUME Without existence of an Error	A RESUME is enccuntered before an error routine is entered.
?SN Error	2	Syntac error	The grammar of a statement is erroneous.
?ST Error	16	String formula Too complex	The string formula is complicated.
?TM Error	13	Type Mismatch	The types of variables and integers are inconsistent with one another.
?UE Error	21	Unprintable Error	An error that has not been designated in a message.
?UF Error	18	Undefined Function	An undefined user function has been read.

Error Message	Code	N82-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 performed.

APPENDIX A3

Control Codes

The PC-8201 uses ASCII character codes from 1 through 31 as control codes, and has a function for display operations such as cursor movement control.

The following control codes are effective in the TELCOM mode:

OPERATION	CHARACTER CODE	FUNCTION
CTRL + C	3	Interrupts command input (effective during keyboard input) the same as the ket
CTRL + G	7	Bell to sound the beeper
CTRL + H	8	Back Space (the same as DEL BS)
CTRL + I	6	TAB
CTRL + J	10	Line Feed
CTRL + K	11	Home Position
CTRL + L	12	Clear the Screen
CTRL + M	13	Carriage Return (same as ↵ Key)
CTRL + N	14	Shift OUT (effective only with a contrcl designation, applies to RS-232C)
CTRL + O	15	Shift IN (effective only with a control deisgnation)

OPERATION	CHARACTER CODE	FUNCTION
CTRL + Q	17	Request Interrupt during transmission (effective only with a control deisgnatiɔn)
CTRL + S	19	Atuhorizes Reopening of transmission (effective only with a control designation)
ESC	27	Begins the Escape Sequence
◁	28	Moves the cursor one character to the right
▷	29	Moves the curso· one character to the left
▽	30	Moves the curso· up one line
△	31	Moves the cursor down one line

APPENDIX A4

Character Codes

Decimal	Character	Decimal	Character
0	Control Code Table Comparison (Unique code that cannot be output as characters)	20	Control Code Table Comparison (Unique code that cannot be output as characters)
1		21	
2		22	
3		23	
4		24	
5		25	
6		26	
7		27	
8		28	
9		29	
10		30	
11		31	
12		32	(space)
13		33	!
14		34	"
15		35	#
16		36	$
17		37	%
18		38	&
19		39	'

Decimal	Character	Decimal	Character
40	(	65	A
41	)	66	B
42	*	67	C
43	+	68	D
44	,	69	E
45	—	70	F
46	.	71	G
47	/	72	H
48	0	73	I
49	1	74	J
50	2	75	K
51	3	76	L
52	4	77	M
53	5	78	N
54	6	79	O
55	7	80	P
56	8	81	Q
57	9	82	R
58	:	83	S
59	;	84	T
60	<	85	U
61	=	86	V
62	>	87	W
63	?	88	X
64	@	89	Y

Decimal	Character	Decimal	Character
90	Z	115	s
91	[	116	t
92	\	117	u
93	]	118	v
94	^	119	w
95	_	120	x
96		121	y
97	a	122	z
98	b	123	{
99	c	124	|
100	d	125	}
101	e	126	~
102	f	127	
103	g	128	◀
104	h	129	◀┘
105	i	130	▒
106	j	131	User-defined characters (Potential to be Input from the Keyboard)
107	k	132	
108	l	133	
109	m	134	
110	n	135	
111	o	136	
112	p	137	
113	q	138	
114	r	139	

Decimal	Character	Decimal	Character
140	User-defined characters (Potential to be Input from the keyboard)	165	User-defined characters (Potential to be Input from the keyboard)
141		166	
142		167	
143		168	
144		169	
145		170	
146		171	
147		172	
148		173	
149		174	
150		175	
151		176	
152		177	
154		178	
154		179	
155		180	
156	CLUBS	181	
157	DIAMONDS	182	
158	HEARTS	183	
159	SPADES	184	
160		185	
161		186	
162		187	
163		188	
164		189	

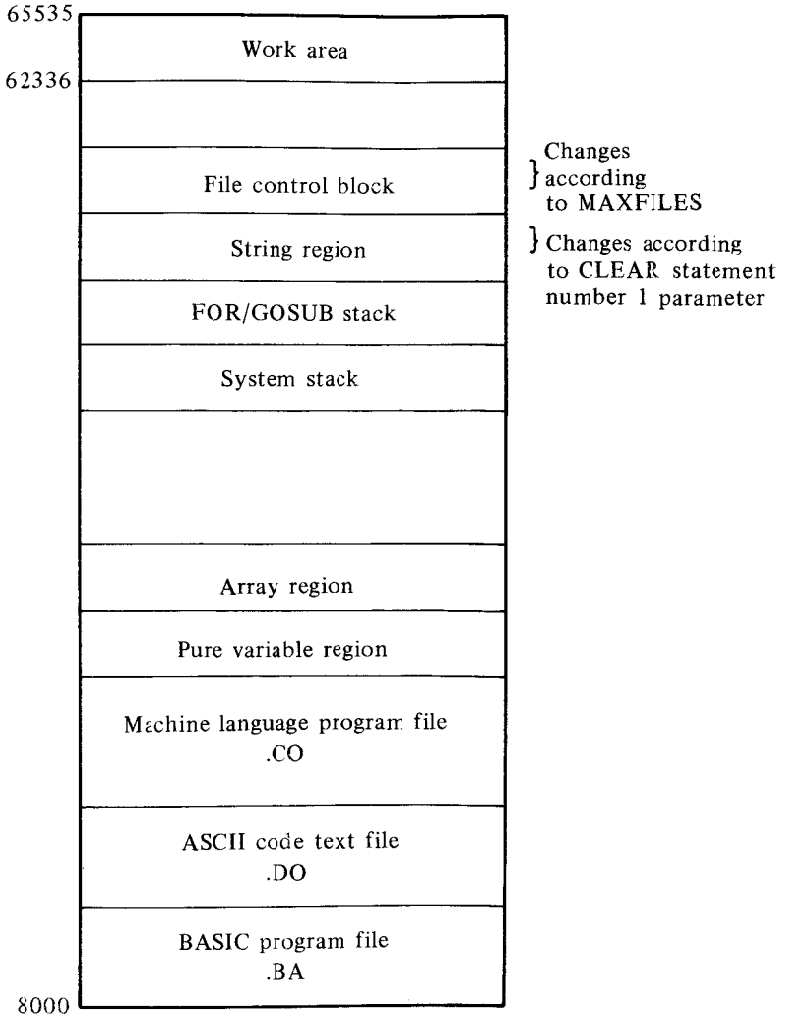
Decimal	Character	Decimal	Character
190	User-defined characters (Potential to be Input from the Keyboard)	215	User-defined characters (Potential to be Input from the keyboard)
191		216	
192		217	
193		218	
194		219	
195		220	
196		221	
197		222	
198		223	
199		224	User-defined characters (Output by using the CHS$ function)
200		225	
201		226	
202		227	
203		228	
204		229	
205		230	
206		231	
207		232	
208		233	
209		234	
210		235	
211		236	
212		237	
213		238	
214		239	

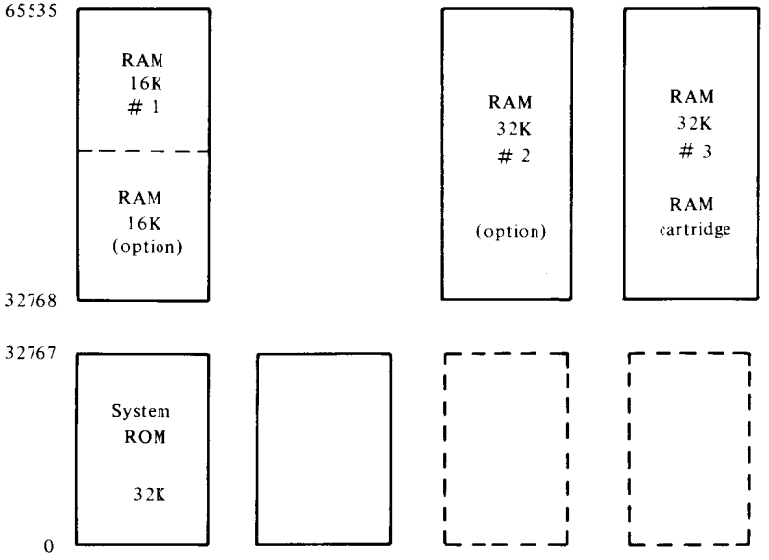
Decimal	Character
240	User-defined characters (Output by using the CHS$ function)
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	

APPENDIX B

Memory Maps

1





The addresses for RAM # 2 and RAM # 3 can be designated as either 0 through 32767 or 32768 through 65535.

Each block can affect a bank conversion in 32K byte segments.

APPENDIX C

Escape Sequences

An Escape Sequence involves the performance of a designated function according to any array of letters which follow the Escape code (ESC:27). It is input by pressing the ESC Key and pressing a letter key. The methods of using the CTRL and SHIFT Keys are entirely different, so do not confuse these special methods with normal functions of the CTRL and SHFT Keys.

An Escape Sequence is also effective in BASIC.

The following Escape Sequences can be used with the PC-8201:

ESC +	CHARACTER CODE	FUNCTION
E	27, 69	Clears Screen and moves the cursor to the top left corner of the screen (the home position)
J	27, 106	Clear Screen
K	27, 75	Erases characters from cursor position to the end of line
J	27, 74	Erases characters from cursor position up to the end of the display
I	27, 108	Erases characters on the line where the cursor is located
L	27, 76	Inserts a Line
M	27, 77	Deletes the line where the cursor is located

ESC +	CHARACTER CODE	FUNCTION
Y ⟨y⟩ ⟨x⟩		Moves the cursor to a designated location, the y x offset by the space character ASCII (decimal 32).
A	27, 65	Moves the cursor one line up
B	27, 66	Moves the cursor 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
P	27, 112	Changes the screen into reverse display
q	27, 113	Restores characters to normal (switches from reverse display)
T	27, 84	Displays Function Keys
U	27, 85	Erases the display of Function Keys
V	27, 86	Inhibits scrolling (freezes the display)
W	27, 87	Permits Scrolling

ESC + Y ⟨y⟩ ⟨x⟩

The cursor position is designated vertically and horizontally by two characters which are subsequent to ESC + Y.

Capital letters from character code 32 are used in the designation. A blank (space) corresponds to the location 0, and (!) corresponds to 1, while (") corresponds to 2. for instance, to move the cursor to home position, input the following string:

ESC, "Y", " ", " "

This means 27, 89, 32, 32 in character code.

In TERM mode, when the [J] Key is input, only the carriage code (13) is transmitted while the change line code (10) is not transmitted. In the case where the carriage return code is received, the line is not changed. Though this does not cause a problem in communication with a host computer, when communicating with other computers the user must input [CTRL] + J in order to actively perform the change of lines.

No change line code will be transmitted when the UPLOAD command is executed. This is something to be fully aware of when a program is being created at the receiving end of the data transmission.

APPENDIX D

Glossary

ABSOLUTE VALUE — The positive 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 — Deals with on-off circuit elements, and 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.

COSINE — In a right triangle, the value obtained when the side adjacent to an angle is divided by the hypoteruse.

DATA — The input values that a computer must have in order to solve a given problem.

DELIMIT — Separate.

DIMENSION — The number of elements in an array and their configuration (one or two 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 ahead 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 within its mechanism several million characters of information at any given moment. In back up devced, 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. 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 running programs.

RETURN KEY A key on your terminal's keyboard that is used to enter a BASIC statement.

RESERVED WORD In BASIC, the first word of a statement that identifies the type of statement.

ROM Read Only Memory. The type of memory that stays intact even when the PC-8201'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 multipled by itself gives a specified value. Thus, the square root of 64 is 8.

STATEMENT A single instruction to the computer such as: 10 LET P=7

SUBSCRIPT A number, name, or expressior that tells which one of an array element is to be worked with.

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 opposite the angle is divided by the side adjacent to the angle.

TEST To check out, such as the value of a counter, the state of a condition, a program, etc.

TRUNCATE Drop the decimal digits of a number. (Rounded off).

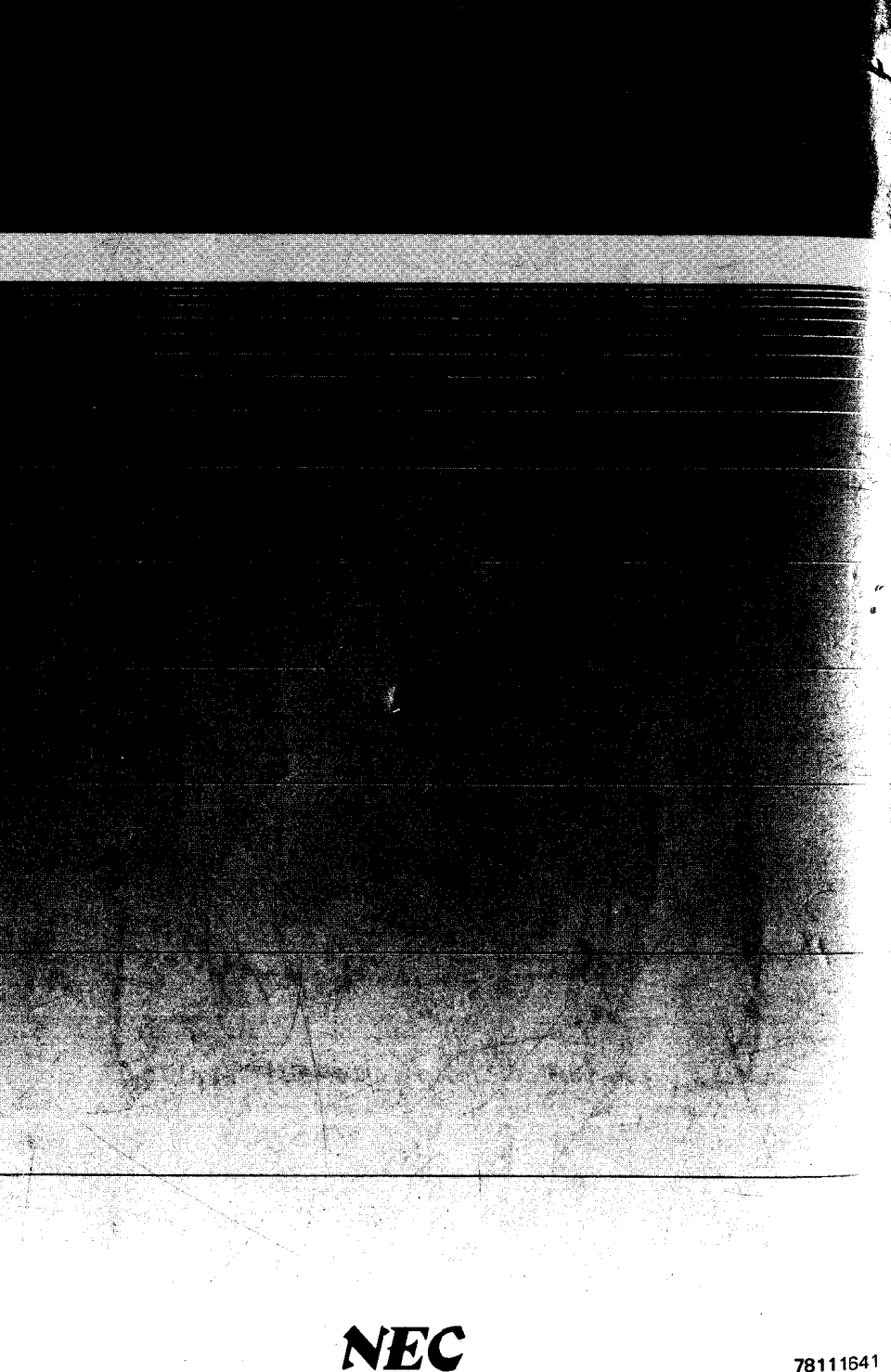
ZONE One of the two areas of the screen where an answer may be displayed.

INDEX

Model 100 Command List

8200 User Monthly - 3/85

NEC	100		NEC	100	
ABS	SAME	ABS$	LOCATE	SIMILAR	PRINT@
AND	SAME	AND$	LOF	SAME	LOF$
ASC	SAME	ASC$	---	NO EQV.	MAXRAM
BLOAD	SIMILAR	LOADM	---	NO EQV.	MDM ON
BLOAD?	NO EQV.	---	---	NO EQV.	MDM OFF
BSAVE	SIMILAR	CSAVEM	---	NO EQV.	MDM STOP
---	NO EQV.	DAYS$	MOD	SAME	MOD$
DSKI$	SAME	DSKI$$	NOT	SAME	NOT$
DSKO$	SAME	DSKO$$	---	NO EQV.	ON KEY GOSUB
EQV	SAME	EQV$	---	NO EQV.	ON MDM GOSUB
EXEC	SIMILAR	CALL	---	NO EQV.	ON TIME$ GOSUB
---	NO EQV.	HIMEM	OR	SAME	OR$
IMP	SAME	IMP$	---	NO EQV.	POWER CONT
---	NO EQV.	IPL	---	NO EQV.	POWER OFF
---	NO EQV.	KEY LIST	---	NO EQV.	RUNM
---	NO EQV.	KEY ON	---	NO EQV.	SOUND ON
---	NO EQV.	KEY OFF	---	NO EQV.	SOUND OFF
---	NO EQV.	KEY STOP	---	NO EQV.	TIME$ ON
---	NO EQV.	LCOPY	---	NO EQV.	TIME$ OFF
---	NO EQV.	LINE	---	NO EQV.	TIME$ STOP
---	NO EQV.	LINE INPUT$	---	NO EQV.	VARPTR
---	NO EQV.	LINE INPUT#	XOR	SAME	XOR$



NEC

7811641