HAND HELD PERSONAL COMPUTER

ORDINATEUR PORTABLE

INSTRUCTION MANUAL

MANUEL DE FONCTIONNEMENT

CONTENTS

I. BASIC LANGUAGE SPECIFICATION

COMMANDS

APPEND, CHAIN, CLEAR, CLOAD, CONSOLE, CONT, CSAVE, DELETE, LIST, LLIST, LOCK, NEW, RENUM, RUN, UNLOCK.

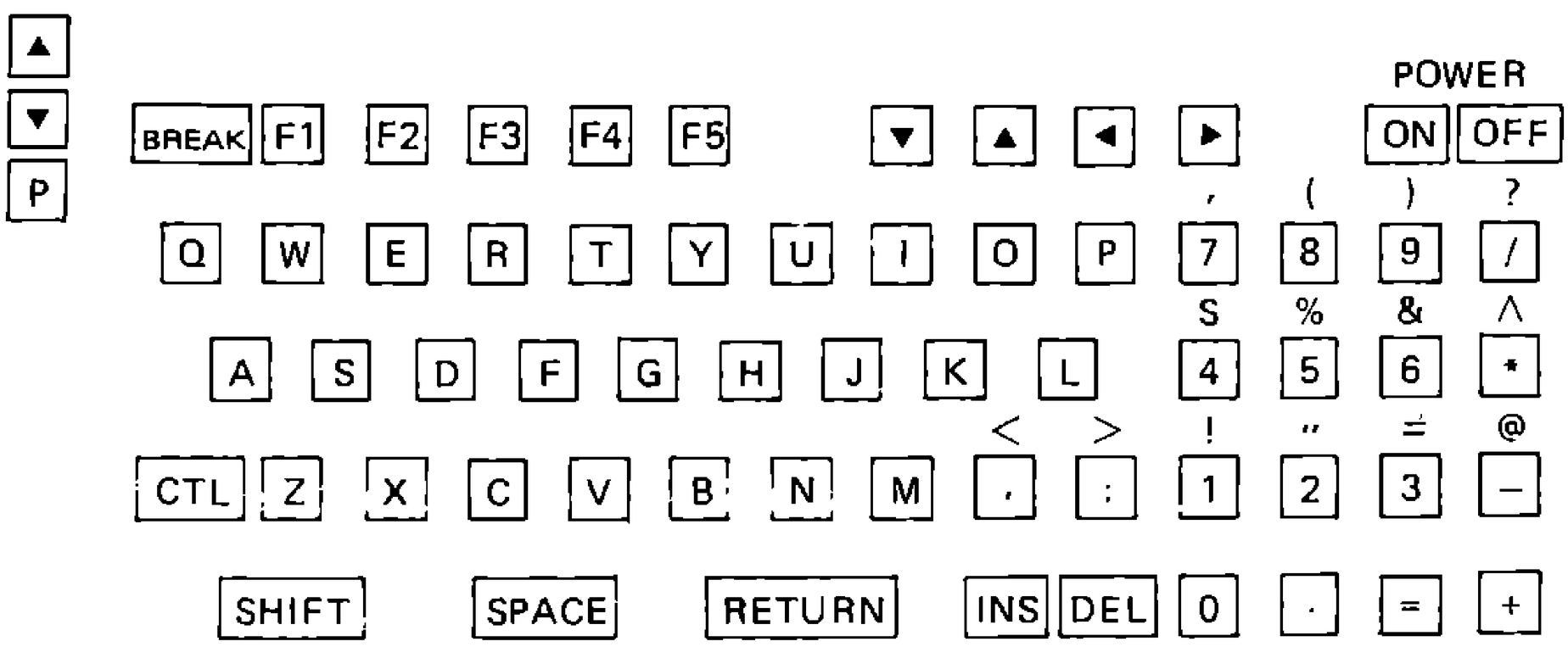
STATEMENTS

BEEP, BOX, COLOR, DATA, DEFFN, DEFINT/SNG/STR, DIM, END, ERASE, FOR –NEXT, GOSUB – RETURN, GOTO, IF – THEN [– ELSE], LET, LINE, MOVE, MOTOR, ON ERROR GOTO, ON – GOSUB, ON – GOTO, OPTION BASE, ORIGIN, POKE, RANDOMIZE, READ, RLINE, REM, RESTORE, RESET, RESUME, ROTATE, STOP, SCALE, TRON/TROFF, INPUT, INPUT#, LOCATE, LPRINT, PRINT, PRINT#, PRINT USING, PAUSE, WAIT.

FUNCTIONS

KEY, KEYLIST, ABS, ATN, COS, EXP, FIX, INT, LOG, RND, SGN, SIN, SQR, TAN, TAB, ASC, CHR$, HEX$, LEFT$, LEN, MID$, OCT$, RIGHT$, STR$, VAL, ERR, ERL, FRE, PEEK, INKEY$.

II. KEYBOARD LAYOUT



III. SPECIFICATIONS

Program Language		BASIC	
Capacity	: CPU	:	CMOS 8 bit
	System ROM	:	20K Bytes
	Memory Capacity RAM	:	6K Bytes
	System area	:	1.7K Bytes
	User area	:	4.3K Bytes
Editing Function	: Cursor Shifting (◄, ►)		
	Insertion (INS)		
	Deletion (DEL)		
	Line up and down (▲, ▼)		
Memory Protection	CMOS battery back-up (program, data and reserve memories are protected)		
Display	Liquid Crystal 48 characters (24 characters each in 2 lines)		
Keys	62 keys including Alphabetic, Numeric, User-definable Function, Pre-programmed		
Power Supply	6.0 V DC : 4 pcs Dry Batteries (Type UM-3) AC : AC Adaptor more than 10 MA		
Power Consumption	: 0.02 W		
Operating Temperature	: 0°C – 40°C (32°F – 104°F)		
Dimensions	199(W) x 96(D) x 26(H) mm		
Weight	: 410 g (with batteries)		
Options	: a. Printer and Cassette Interface unit b. Expansion memory module (Plug-in Type: 4K byte RAM and 8K byte RAM)		
Others	Can be connected directly to any terminal printers (10", 15", etc.) in Centronics parallel type, using with Printer/Cassette Interface unit.		

IV. KEYBOARD AND DISPLAY

A. KEYBOARD

[ON] AND [OFF] KEYS

These keys turn the power on and off. Machine, to conserve power, will automatically shut off if nothing is keyed in for a period of about seven minutes, unless a program is being executed.

ALPHABETIC KEYS ([A] ~ [Z])

The Alphabetic keys allow the computer user (you) to give instructions and enter data. In addition, these keys may be used to designate "storage areas" within the computer's memory into and from which you will be able to save or retrieve data.

NUMERIC ([.], [0] ~ [9]) and ARITHMETIC OPERATION ([+], [−], [*], [/], [=]) KEYS

With these you enter numbers for calculation. The [+], [−], [*] and [/] keys tell machine to add, subtract, multiply and divide respectively.

[SHIFT] KEY

This key delivers the secondary functions inscribed above the non-alphabetic keys. For instance, to type a semi-colon (;), press [SHIFT] and then the [+] key.

FUNCTION KEYS ([F1] ~ [F5])

You can assign the commands frequently typed or other operations to these keys. You can store 10 Functions in total.

Example:

a) To write the word "PRINT" into F1:
   K E Y 1 , " P R I N T " [RETURN]

b) To read the above, just press [F1] key.

To read [F6] to [F10] keys, press [SHIFT] key and [F1] to [F5] keys. For example, to call contents of [F6] key, press [SHIFT] key and [F1] key.

[P] KEY (in the Printer/Cassette Interface unit)

Press for getting the proper position of pen replacement.

[BREAK] KEY

This key stops program execution. Continue pressing this key until "BREAK IN nn" is displayed.

CURSOR SHIFT KEYS

[◄] Moves the cursor one position to the left.

[►] Moves the cursor one position to the right.

CONTROL (CTL) CHARACTERS

CTL + D	Deletes the character in the cursor position.
CTL + E	Deletes all the characters of the right of the cursor.
CTL + ▶	Displays on the screen the characters exceeded when more than 48 characters entered.
CTL + ◀	Displays on the screen first 48 characters when more than 48 characters entered.
CTL + L	Clears screen.

SHIFT CHARACTERS

SHIFT + ▶	Shifts the data displayed in PRINT to the left.
SHIFT + ◀	Shifts the data displayed in PRINT to the right.
DEL	Deletes the character left of the cursor.
INS	Shifts all characters right of the cursor in one space and leaves the cursor position blank.
▲	Displays the contents in the previous line number. (In Printer/Cassette Interface unit, paper UP-feed key)
▼	Displays the contents in the latter line number. (In Printer/Cassette Interface unit, paper DOWN-feed key)

RETURN KEY

As you type into the computer, the letters or numbers appear on the display. Machine will take NO ACTION, however, until you signal that you have finished typing. This is done by pressing the **RETURN** key after your other keystrokes. At this point, the computer will scan the characters you have typed for correct form. Certain errors, but by no means all errors, will cause your input to be rejected. Press the **RETURN** key each time you wish to enter an instruction or item of data into the machine.

B. DISPLAY

Press ON. The "glass window" part of the computer is called the "display". It looks something like this:

```
>
10: INPUT "LIST SIZE?";A
```

On the display you should see an > (called "prompt"), several words or abbreviations.

The CURSOR and the PROMPT

At the far left of the display, find the prompt symbol (>); it prompts you to talk to machine. When the "prompt" appears it means machine has no immediate plans and awaits your bidding. Type a letter of your choice. It replaces > at the left of the display, while to the right of your letter appears a _ (underline symbol). This is a cursor. As you press each key, the cursor inches it way across the display, indicating where the next symbol you type will appear.

C. EDITING FUNCTIONS

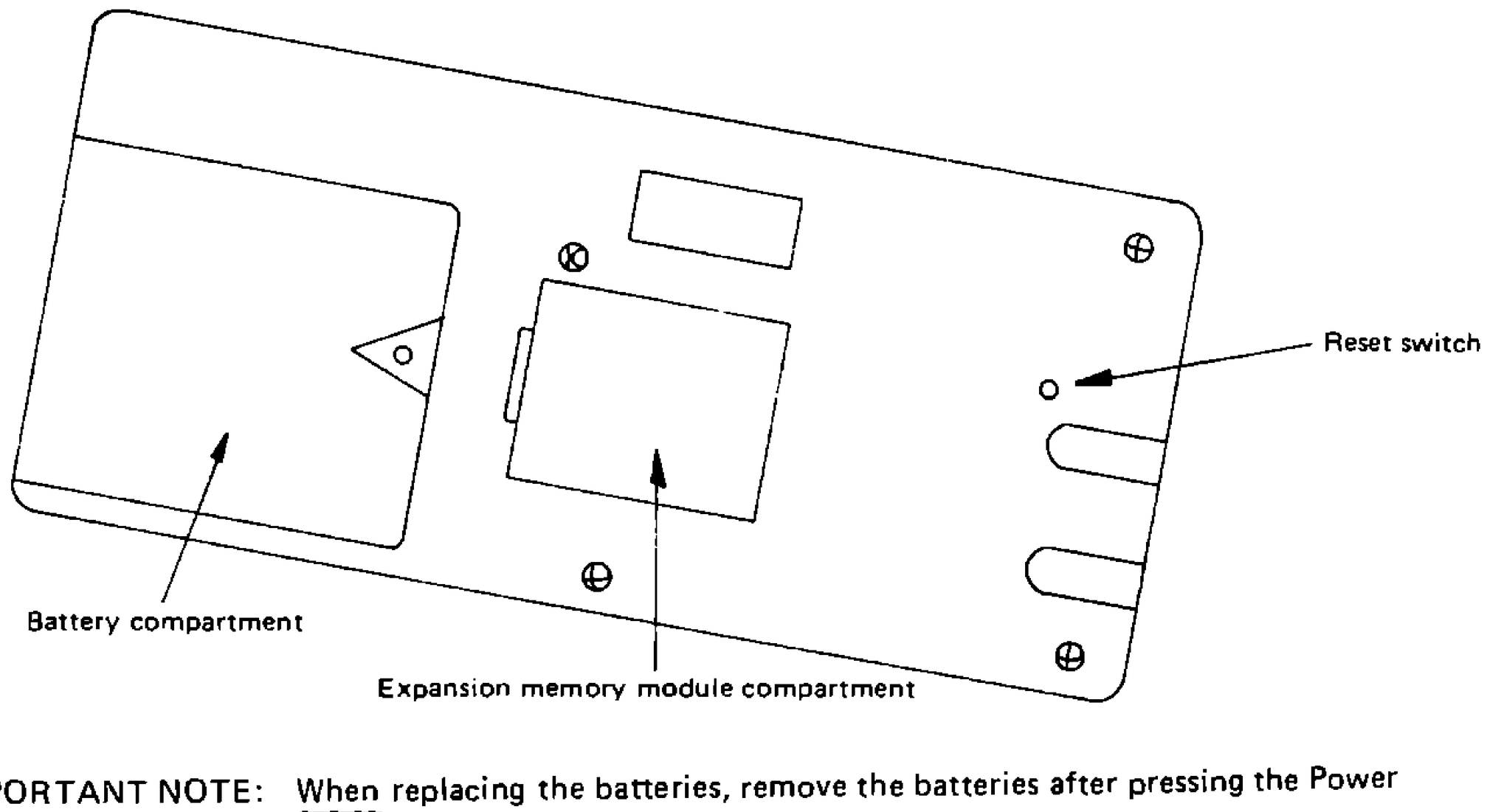
Editing keys ([◄], [►], [INS], [DEL]) can be used as correction keys when made mis-type on the computer.

1. You wrote as: I HAVT A PEN where you should write I HAVE A PEN.
   a. If the cursor (_) is under the letter N, press [◄] key repeatedly (or hold i down) until the cursor is positioned under "T" and write "E".
   b. If the cursor is under the letter I, press [►] key repeatedly until it is positioned under "T" and write "E".
2. If you wrote as: I HAE A PEN and the cursor is under the letter N, press [◄] key until the cursor is positioned under "E" and press [INS] key then write "V".
3. If you wrote as: I HAVBE A PEN and the cursor is under the letter I, press [►] key until it is positioned under "E" and press [DEL] key.

D. RESET SWITCH

Press RESET switch on the back of the computer if display disappears or keys become inoperative due to wrong operations or external impact during operation.

BACK OF THE COMPUTER



IMPORTANT NOTE: When replacing the batteries, remove the batteries after pressing the Power [OFF] key otherwise the programs memorized are lost.

V. PRINTER AND CASSETTE INTERFACE

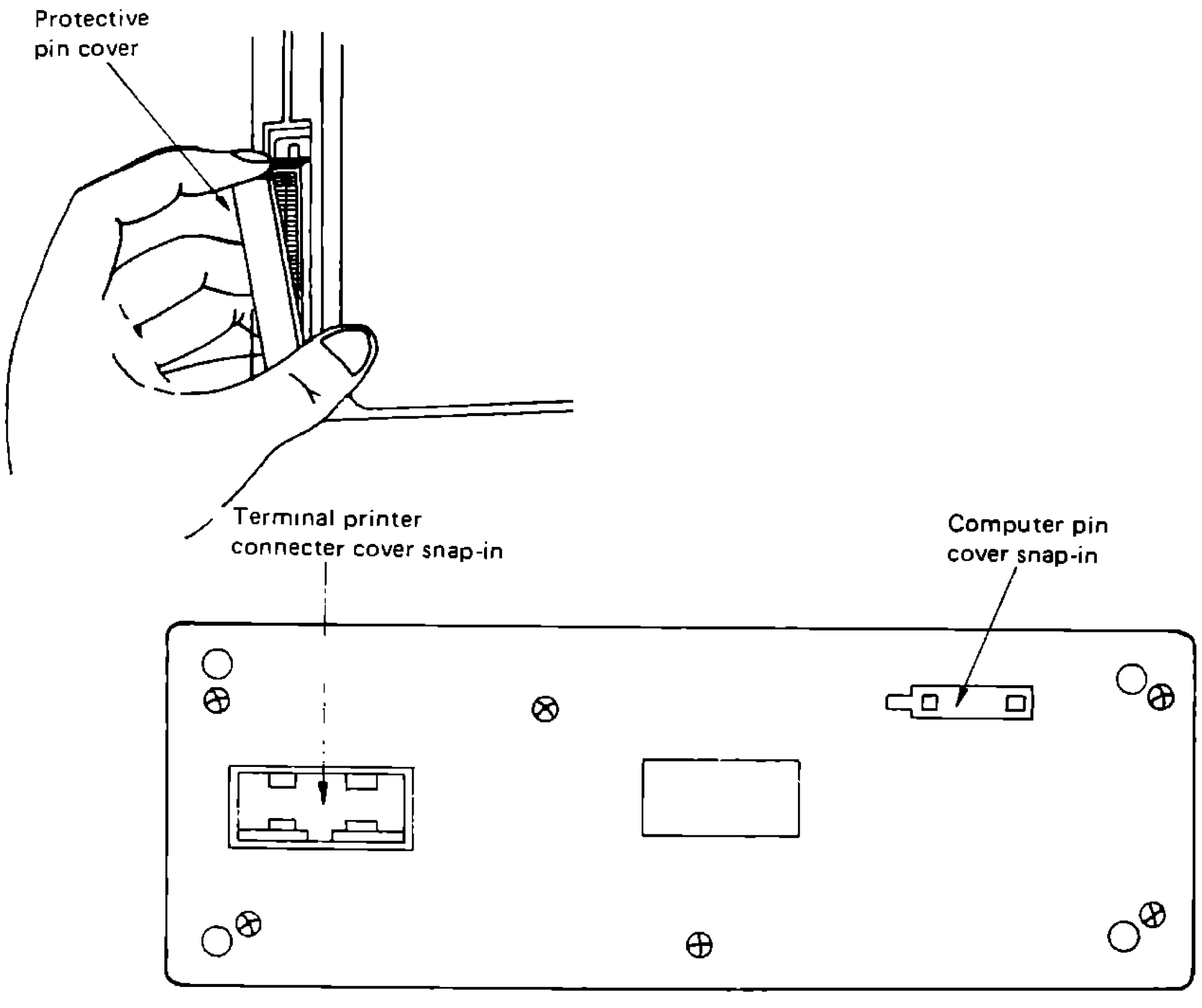
The Printer/Cassette Interface unit is an option for the Computer. With this unit (Printer), you can print, in four colors, copies of your programs and data and even plot graphic displays.

This unit (Cassette Interface) can also be connected to Cassette Tape Recorder. The Tape Recorder can be used to store programs and data on standard cassette. The programs on the tape can be loaded back into the computer for use at a later date, saving your trouble of typing them again.

To **Connect the Computer to the Printer/Cassette Interface Unit**

Connect the Printer/Cassette Interface unit and the Computer in the following procedures:

1. Turn the computer power OFF
2. Remove the protective pin cover from the left side of the computer and snap it into place on the back of the Printer/Cassette Interface unit.



3. Then slide the computer into the cradle of the Printer/Cassette Interface unit to connect the connectors of both units.

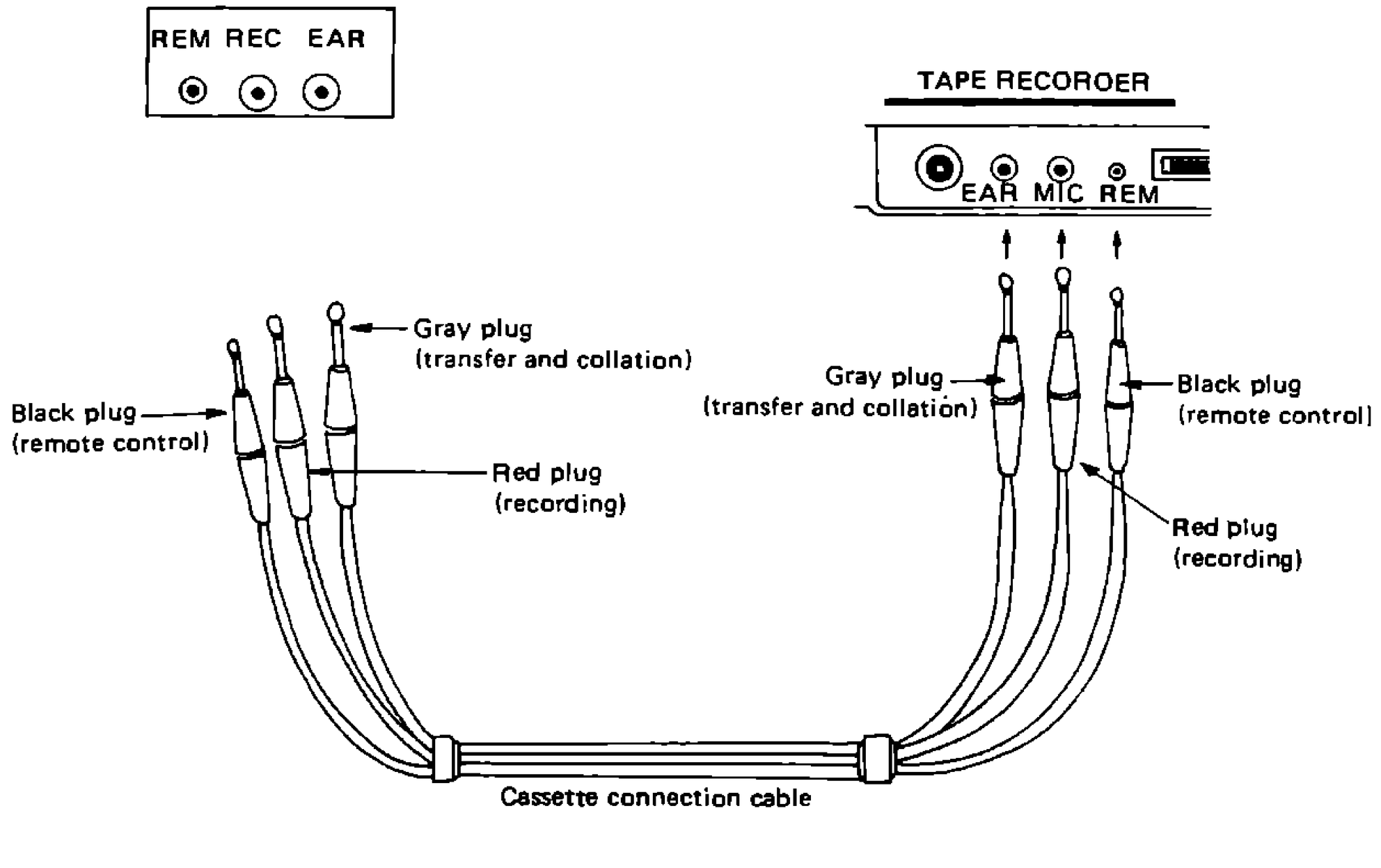
A. POWER

The Printer and Cassette Interface unit works with 4 pcs. of rechargeable NI-CAD battery which are built in the machine. Therefore it is necessary to recharge the batteries for about ten minutes after unpacking the machine, and the machine starts working. It will take about 15 hours for the battery to become fully charged. For best results, fully charge the battery before use.

B. CONNECTING A TAPE RECORDER TO THE INTERFACE

First, connect the Printer/Cassette Interface unit and Computer then connect the Tape Recorder with the Interfece unit as shown in the following:

PRINTER/CASSETTE INTERFACE UNIT



Cassette connection cable

C. TAPE RECORDER SPECIFICATIONS

The following is a description of the minimum tape recorder specifications necessary for interfacing with the Interface unit.

Item		Requirements
1. Recorder Type	:	Any tape recorder, cassette, micro-cassette, or open reel recorders may be used in accordance with the requirements outlined below.
2. Input Jack	:	The recorder should have a mini-jack input labelled "MIC". Never use the "AUX" jack.
3. Input Impedance	:	The input jack should be a low impedance input (200 – 1,000 OHM).
4. Minimum Input Level	:	Below 3 mV or –50 dB.
5. Output Impedance	:	Should be below 8 OHM.
6. Output jack	:	Should be a minijack labelled "EXT. (EXTernal speaker)", "MONITOR", "EAR. (EARphone)" or equivalent.
7. Output Level	:	Should be more than 1 V (practical maximum output over 100 MW).

In case the miniplug provided with the unit is not compatible with the input/output jacks of your tape recorder, special line conversion plugs are available on the market.

NOTE: Some of tape recorders may reject connection due to different specifications. Or those tape recorders having distortion, increased noise, and power deterioration after long years of use may not show satisfactory results owing to change in their electrical characteristics.

Precautionary Instructions for Tape Recorder

(1) For any transfer or collation, use the tape recorder that was used for recording. If the tape recorder for transfer or collation is different from that used for recording, no transfer or collation may be possible.

(2) The head of a tape recorder, if stained, increases distortion or decreases the recording level. Therefore, keep the head clean.

(3) Use a tape that is free of extremely low frequency response, scratches and creases.

D. USING A CASSETTE RECORDER

1. The CSAVE Command

With CSAVE command which saves your program, you must give the program a "filename". This is for reference purposes. Your filename can not be longer than 6 characters. To save the program with a filename, type:

CSAVE "filename"

Your program will be saved with the filename you put. You can assign any name you desire, whatever is easiest for you to keep track of. If the name is longer than 6 characters, the excess is ignored. More than one program can be sorted on each cassette and the computer will skip over them until it comes to the named program.
(see 2.1.7: CSAVE)

2. The CLOAD Command

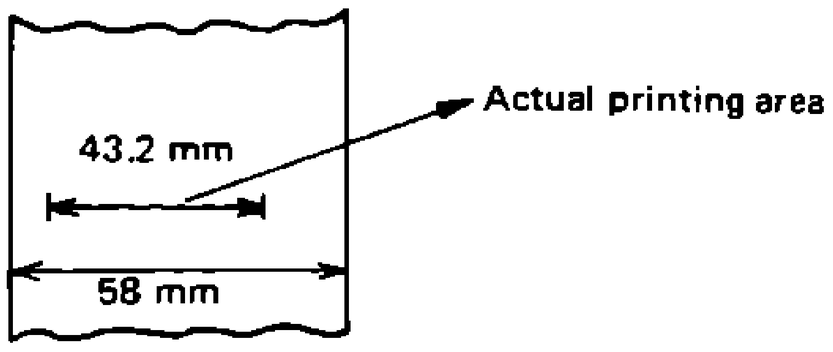
With CLOAD command which loads your program, you must give a "filename" designated when saved program on the cassette. Enter as follows:

CLOAD "filename"

When "filename" is found out on the tape, "FOUND filename" is displayed and when loading is completed, prompt sign ">" is displayed.
(see 2.1.4: CLOAD)

E. PRINTER SPECIFICATIONS

Printing	:	Four, water-color pens (black, blue, green and red)
Printing System		X- and Y-axis plotter system
Printing Command Area	:	$-999 \leqq X, Y \leqq 999$ (actual printing area on the paper is 43.2 mm in X-axis)

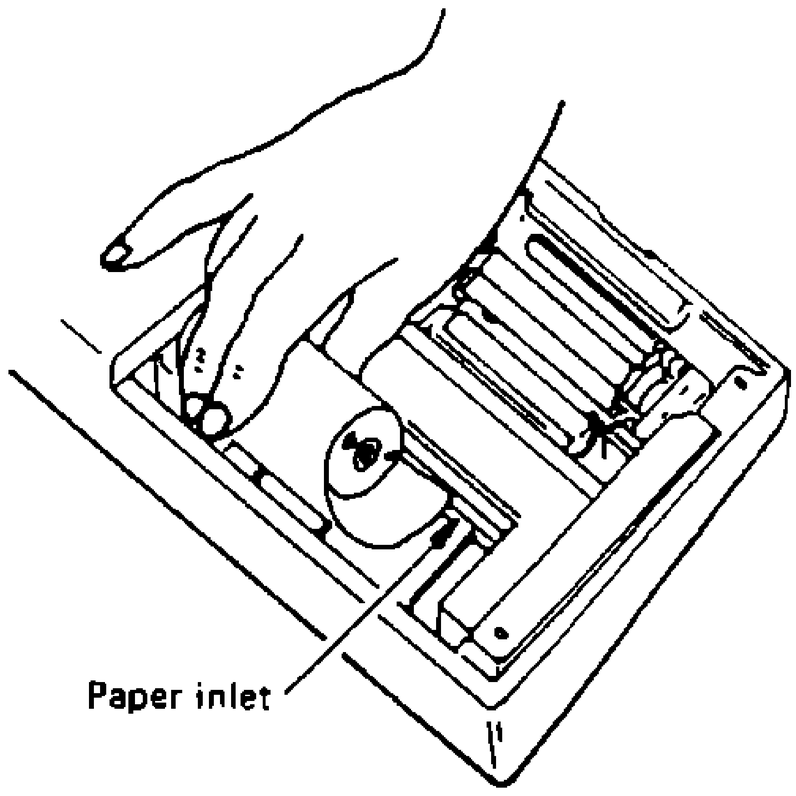


Printing Speed	:	10 characters max./second
Print Characters	:	ASCII set
Character Size	:	16 sizes of 0 (smallest: 1.05(W) x 1.45(H) mm) to 15 (largest: 12.78(W) x 19.45(H) mm)
Paper		58 mm in width

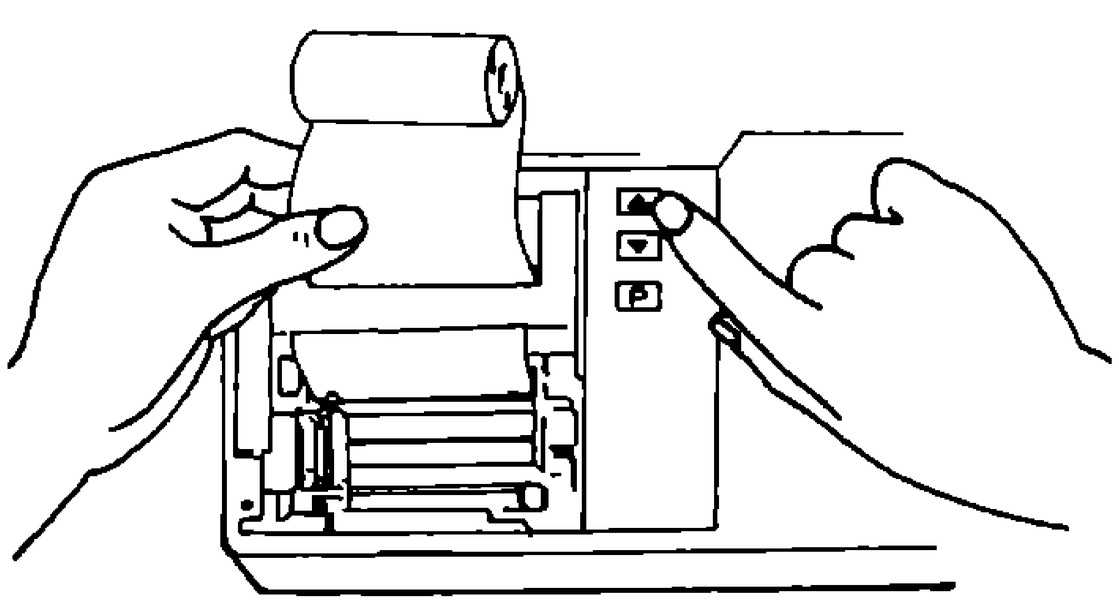
F. LOADING THE PAPER

(1) To remove the printer cover, lift the top portion of the cover.

(2) Cut the tip of roll paper straight, and insert the paper correctly into the paper inlet. (Any curve or crease at the paper tip may prevent paper insertion.)

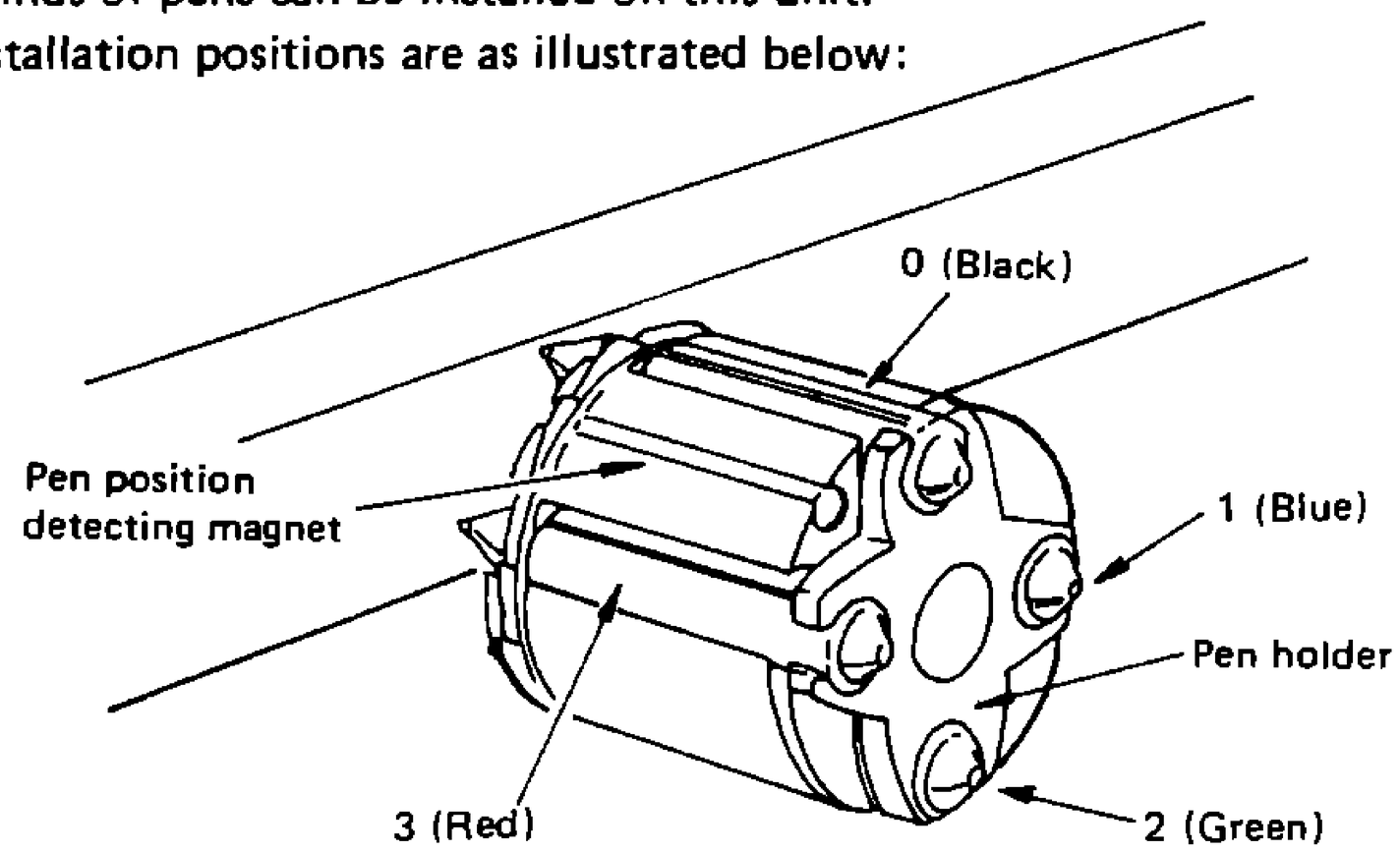


(3) Press [▲] key to feed paper and feed the paper so that the paper tip may be 3 to 5 cm from the printer.



G. REPLACING THE PENS

Four kinds of pens can be installed on this unit.
Pen installation positions are as illustrated below:



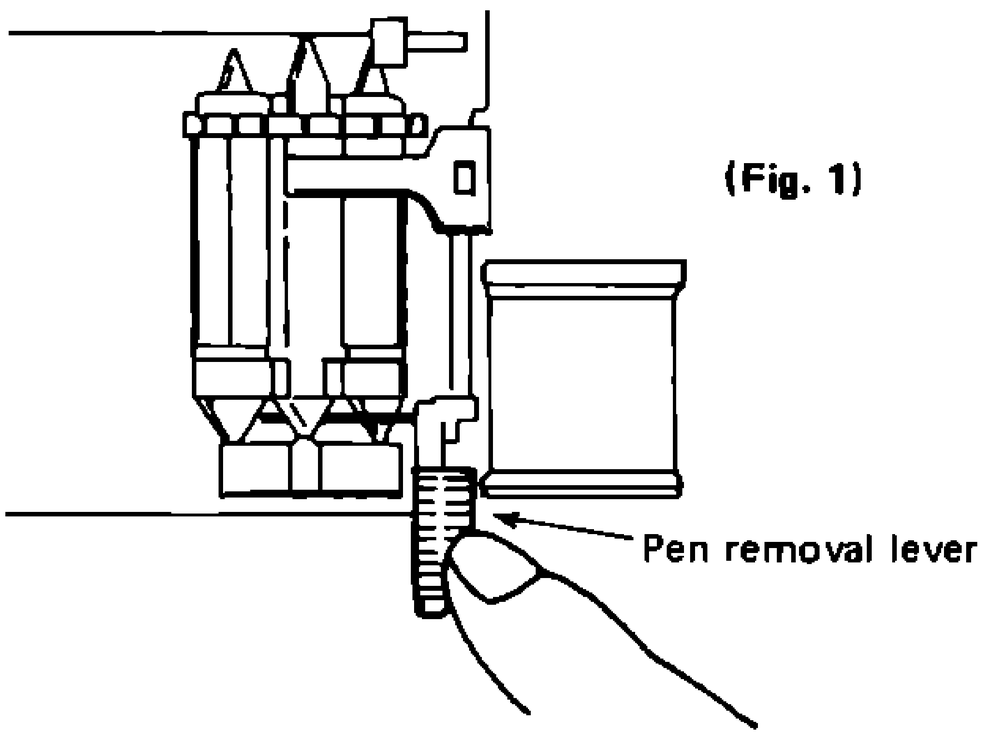
When designated by the COLOR instruction, the pen positions number 0, 1, 2 and 3 clockwise from the pen position detecting magnet as illustrated above. (The pen holder rotates counter-clockwise so that the selected pen comes stop).

For pen installation or replacement, follow the procedure below:

(1) Press the [P] key. This allows the printer to be in its pen replacement state.

(2) To remove the pen, press the pen removal lever. This causes the pen on top to come off.

NOTE: When removing the pen, hold it lightly to prevent it from hopping up into the printer. (Fig. 1)

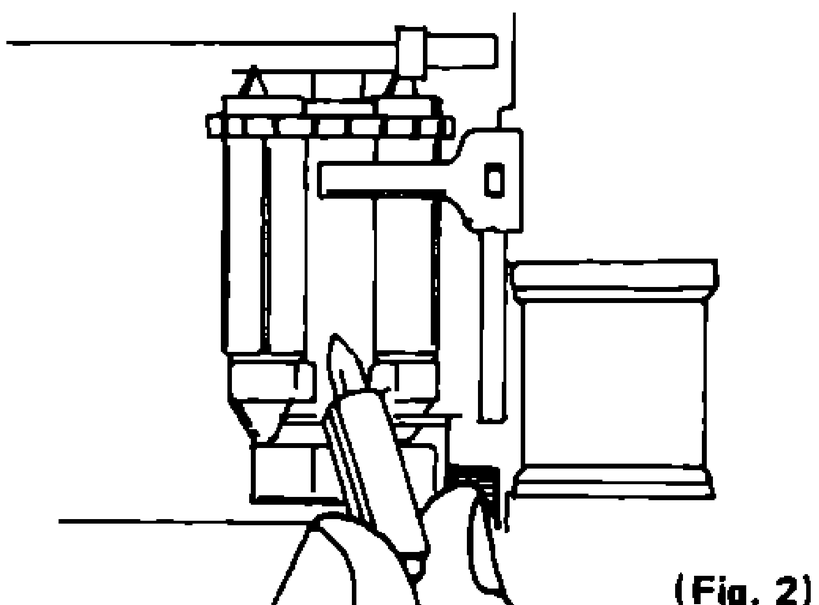


(Fig. 1)

(3) Install a new pen. (Fig. 2)

(4) Press [▲] key and enter "COLOR 1" [RETURN] to install or remove the next pen. The pen holder returns to the left, rotates so that the next pen comes stop and shifts to the right again.
Remove the pen and replace it with a new one.

(5) After the pen replacement or installation, press [▲] key. This causes the printer to be released from its pen replacement state, and the pen to return to the left.



(Fig. 2)

NOTE: To use this unit, install the four pens on the pen holder.
Operation with a lack of even one pen may cause color changes to malfunction.

Handling the Pens:

Remove the pens from the printer after use. Cap the pens and place them in their refill for storage. Leaving the pens installed on the printer for a long time or placing them on a desk may cause the ink to dry.

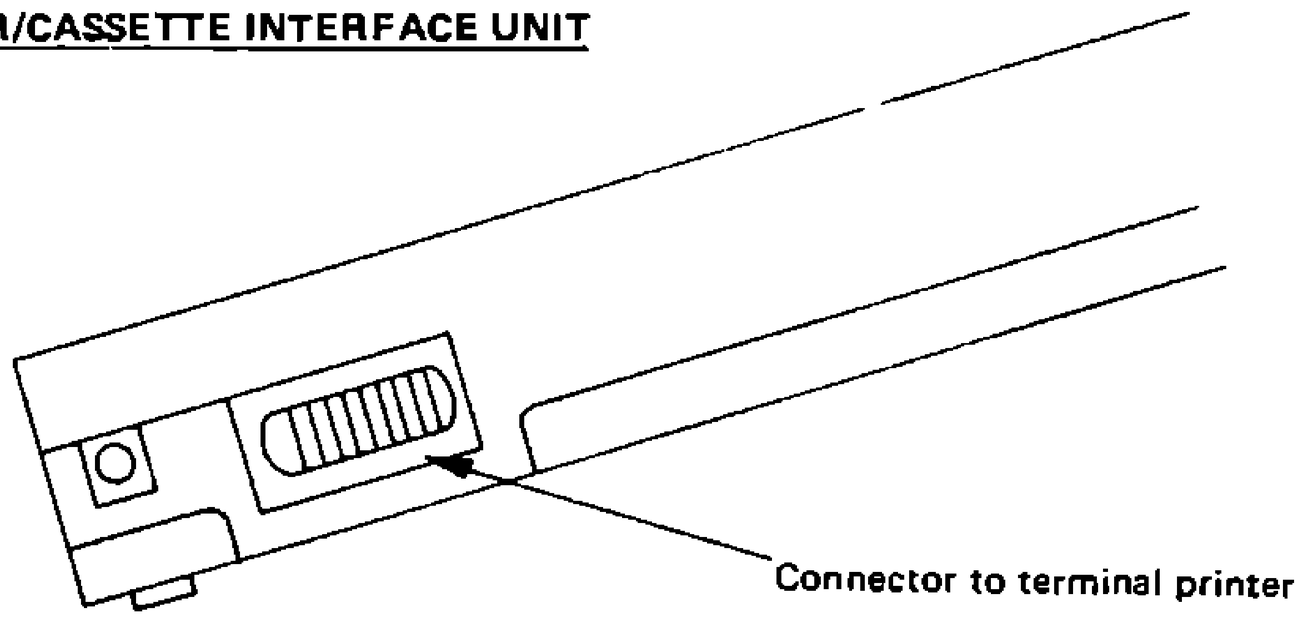
H. CONNECTION TO TERMINAL PRINTER

The Printer/Cassette Interface unit can be connected directly to any terminal printers (10", 15", etc) of Centronics Parallel type.
The connector is equipped in the Printer/Cassette Interface unit in the following wiring.

Pin No. 1	$\overline{\text{STRB}}$ (STROBE)
2	DATA 1
3	DATA 2
4	DATA 3
5	DATA 4
6	DATA 5
7	DATA 6
8	DATA 7
9	DATA 8
11	BUSY
13 – 23	GND (GROUND)

Connect the Terminal Printer and the Printer/Cassette Interface unit using the connecting cable and also connect Pin No. 24 of the connector of Printer/Cassette Interface unit to GND (Pin No. 13 – 23).

PRINTER/CASSETTE INTERFACE UNIT



I. PRINTER/CASSETTE INTERFACE SPECIFICATIONS

Power Consumption	:	4.5 W
Power Supply	:	4 pcs Rechargeable NI-CAD Battery (AA size) (AC Adaptor 9 V 300 MA)
Weight	:	890 g
Dimensions		117 (D) x 329(W) x 51(H) mm

VI. CHAPTER 1: GENERAL INFORMATION

1.1 MODES OF OPERATION

When the machine is turned on, " > " is displayed. This " > " means that BASIC is in command level and the machine is prepared to accept commands. Then, the machine can be used in the following two modes: direct and indirect.

In direct mode, statements and commands are executed as entered. The results are displayed immediately, but the instruction is lost. The direct mode is useful for debugging and using the computer as a calculator for quick computations that do not require a complete program.

You use the indirect mode to create programs. Program statements are preceded by line numbers that are stored in memory and later executed by RUN command.

Checking on battery is made automatically and the message of "NO BATTERY" is displayed when battery runs out in the Printer/Cassette Interface unit.

1.1.1 Arithmetic Calculations

To perform arithmetic operations in direct mode, write on the Computer "PRINT" first and enter the numbers and operators (+, –, *, /, etc.) for calculations then press [RETURN] key. You do not need [=] key to generate the answer. You can write a question mark (?) instead of "PRINT".

Example:

30 + 40 x 20 ÷ (100 – 80) = 70

OPERATION: a. Write "PRINT" or "?"

b. 30 [+] 40 [*] 20 [/] [(] 100 [–] 80 [)]

c. [RETURN]

NOTE: a. Press [SHIFT] key then [8] key to write "(".
(see page 7 : SHIFT KEY).

b. See 1.8.4: ORDER OF MATHEMATICAL OPERATIONS for priority of each calculations.

1.2 LINE FORMAT

A program line always begins with a line number and ends with a carriage return (by pressing [RETURN] key). A program line can contain a maximum of 255 characters and follows this statement:

nnn (line number) BASIC statement [: BASIC statement . . .] < Carriage return (RETURN key)>

More than one statement to a line can be specified by separating each program statement with a colon.

Example:

100 SCALE 4 : MOVE (20,0) [RETURN]

100	SCALE 4	MOVE (20,0)	[RETURN]
line number	statement	statement	carriage return

1.3 LINE NUMBERS

Each program begins with line number. Line number indicates the sequence of storing the line of program into the memory and is used as indicator for branching or editing. Line numbers range from integers 0 to 65529. A period (.) can be used to indicate (or delete) the current line when using LIST and DELETE commands.

1.4 CHARACTER SET

The character set is composed of alphabetic, numeric, and other special characters. Alphabetic characters are capital letters and numerical characters are composed of 0 to 9. There are also special characters.

1.5 CONSTANTS

The two types of constants used are string and numeric.

1.5.1 String Constants

A string constant is a sequence of up to 255 alphanumeric characters enclosed in quotation marks.

"HELLO"

"$25,000.00"

"NUMBER OF EMPLOYEES"

1.5.2 Numeric Constants

Numeric constants can be positive or negative and can be one of the types listed below.

Table Numeric Constants

TYPE	DESCRIPTION
Integer constant	Whole numbers between −32768 and +32767. Example: A%=32767
Fixed point	Positive or negative real numbers. Example: A=35.54
Floating point	Positive or negative numbers represented in exponential form. A floating-point constant consists of a mantissa followed by the letter E and the exponent. The exponent must be in the range of −38 to +38. Example: 235.988E −7 is equivalent to .0000235988 2359E 6 is equivalent to 2359000000
Hex	Hexadecimal numbers are prefixed by "&H". Hex numbers entered in this format will be output in decimal. Example: 10 X=&H76 20 PRINT X RUN [RETURN] . . . Key operation 118 . . . Result to be displayed

Octal — Octal numbers are prefixed by "&O" or "&".
Octal numbers are also displayed in decimal.

Example:
```
10 X=&O347
20 PRINT X
RUN [RETURN]
231
```

1.5.3 Single Precision

Single precision constants consist of any numeric value that has seven or fewer digits, exponential form, or a trailing exclamation point (!). Six of the seven significant digits are displayed.

Example:

Single Precision
```
   46.8
   -0.9E-06
3489.0
   22.5!
```

1.6 VARIABLE

1.6.1 Variables and Declaration Characters

Variable names can be any length; however, only the first two characters are significant. The first character of a name must be an alphabetic character; the remaining characters can be alphanumeric.

A variable name cannot be a reserve word nor contain a reserve word. For example, BFOR is illegal because it contains the reserve word FOR. Reserve words include all commands, statements, and functions.

Variables can represent numeric or string values. String variable names are written with a dollar sign ($) as the last character (A$="Name"). The dollar sign is a variable type declaration character; it declares that the variable represents a string. You use any one of the two declaration characters to declare the following variable types.

%	Integer Variable
!	Single Precision Variable

If you omit a declaration character, the variable is assumed to represent a single precision value.

Example:

MINIMUM!	Declares a single precision value
LIMIT%	Declares an integer value
N$	Declares a string value
ABC	Represents a single precision value

1.6.2 Arrays

Variables can also be declared as arrays by subscripting. For example, A(10) refer to a one-dimensional array of ten elements, any elements of which can be referenced by A(1) through A(10). A(5,5) declares a two-dimensional array that contains 25 elements in 5 rows and 5 columns. Elements are referenced by A(1,1) to A(5,5).

1.7 TYPE CONVERSION

When necessary, Basic converts a numeric constant from one type to another. If you attempt to convert a string variable to numeric or numeric to string, a "Type mismatch" error occurs.

Observe the following rules when converting numeric constants.

a. If you set constants of different types, the constant is stored as the type declared in the variable name.

Example:

```
10 A%=23.42
20 PRINT A%
RUN RETURN
23
```

b. During expression evaluation, all of the operands in an arithmetic or relational operation are converted, and their results returned, to the same degree of precision of the most precise operand.

Example:

```
10 D=6 / 7
20 PRINT D
RUN RETURN
.857143
```

c. Logical operation convert their operands to integers and return an integer result. Dperands must be in the range −32768 to 32767 or an "Overflow" error occurs.

Example:

```
10 PRINT 8.123 DR 24
RUN RETURN
24
```

d. When a floating-point value is converted to an integer, the fractional portion is truncated.

Example:

```
10 C%=55.88
20 PRINT C%
RUN RETURN
55
```

1.8 EXPRESSIONS AND OPERATORS

An expression is a string or numeric constant that when combined produce a single value. Dperators used in performing these mathematical or logical operations are divided into four categories: arithmetic, relation, logical, and functional.

Example:

```
3.14
"PI"
5 − 8
A + B
M$
X < (T−1)
```

1.8.1 Arithmetic Operators

Arithmetic operators, in the order of their precedence, are listed in the table below. The order of operations follows this precedence.

OPERATOR	OPERATION	SAMPLE EXPRESSION
∧	Exponentiation	2 ∧ 4
–	Negative sign	–2
*, /	Multiplication and Division	2*4 4/2.5
MOD	Modulus integer division	5 MOD 2
+, –	Addition and subtraction	4+2, 4–2

Use parentheses to change the order of operations. Operations within parentheses are performed first. Inside the parentheses, the normal order of operation is maintained.

Examples:

a) Algebraic Expression	b) Basic Expression
$X + 2Y$	X+2*Y
$X - \frac{Y}{X}$	X–Y/X
$\frac{X + Y}{Z}$	(X + Y)/Z
$(X^2)Y$	(X∧2)*Y
XYZ	X*Y*Z
X(–Y)	X*(–Y)

1.8.2 Relational Operators

Relational operators compare two values and make decisions regarding program flow. The result of the comparison is true (–1) or false (0). The operators available in Basic are listed in the table below.

Table Available Operators

OPERATOR		EXPRESSION
=	Equality	X=Y
<> or ><	Inequality	X <> Y
<	Less than	X < Y
>	Greater than	X > Y
<= or =<	Less than or equal to	X<=Y
>= or =>	Greater than or equal to	X>=Y

Example:

```
IF SIN (X) 0 < GOTO 100
IF I/J <> 0 THEN K=K + 1
```

NOTE: If arithmatic and relational operators are combined in one expression, the arithmetic is always performed first.
For example, calculation of "10/5 > 5" displays "0" as a result. It means that the result of comparison is false as the value of "10/5 (i.e. 10 ÷ 5)" is smaller than 5. If the calculation is "10/5 < 5", "−1" is displayed, which means that comparison is true.

1.8.3 Logical Operators

Basic provides logical operators for performing bit manipulation, Boolean operations, or tests on multipla relations. As with relational operators, the logical operator returns a true (1) or false (0) value. Logical operations are performed after arithmetic and relational operations. Examples showshowing the outcome of logical operations follow. The operators are listed in the order of their preference.

Example:

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

One use of logical operators is to connect two or more ralational operators to make decisions on the direction of program flow. For example:

```
IF D < 200 AND          Both conditions would have to be
F < 4 THEN 80           true to branch to 80.

IF I > 10 OR K < 0      Unless both relational conditions are false,
THEN 50                 program control branches to line 50.
```

Logical operators convert their oparands to 16 bit, signed, twos complement integers in the range of −32768 to +32767. The given operation is performed on these integers in a bit-by-bit fashion; thus, it is possible to use logical operators to test bytes for a particular bit pattern. In this vane, you can use the AND operator to mask all but one of the bits of a status byte at a machine I/O port, or you can use the OR operator to merge two bytes to create particular binary value.

Examples:

63 AND 16=16

63 equals binary 111111, 16 equals binary 10000, so a bit by bit AND operation yields 10000.

10 OR 10=10

10 equals binary 1010, so 1010 OR 1010=1010.

1.8.4 Order of Mathematical Operations

Following is a summary of priority of operations:

1. Values enclosed in parentheses ( ).
2. Trigonometric functions (such as SIN, COS, TAN, etc.)
3. Exponentiation (^)
4. Negative sign (−)
5. Multiplication (*) and Division (/)
6. Addition (+) and Subtraction (−)
7. Relational operators (<, >, =, ≧, ≦, <>)
8. Logical operators (AND, OR, NOT, XOR)

* Operations of equal precedence are evaluated left to right.

1.8.5 String Operations

Strings can be compared using the same relational operators which are used with numbers. String comparisons are made by taking each character from the left and compare with the ASCII codes one by one. If the ASCII codes are the same, the string is considered equal. If the codes differ, the lower codes are considered smaller in numerical value than the higher ones. A shorter string is oon-sidered smaller. Leading and trailing blanks are significant.

Examples:

"AAN" < "AB" "FILENAME" = "FILENAME"
"X&" > "X#" "CL (space)" > "CL"
"SMYTH" < "SMYTHE"
B$ < "9/12/80" (where B$ = "8/12/80")

Strings can also be concatenated using the "+" (plus) sign. For example:

PROGRAM	KEY OPERATION	DISPLAY
10 A$="FILE" : B$="NAME"	RUN RETURN	FILENAME
20 PRINT A$ + B$	RETURN	NEWFILENAME
30 PRINT "NEW" + A$ + B$		

1.9 SCREEN EDITOR

Characters entered from the keyboard are first received by the screen editor of Basic, then displayed at the current position of the cursor. Input from the keyboard is not interpreted by the interpreter until you press RETURN key. When the RETURN key pressed, the lines that begin with a proper line number are stored in memory as program lines, and input without line numbers is interpreted in direct mode and operations are performed immediately.

1.10 ERROR MESSAGES

When Basic detects the error which interrupts execution of program, error messages (or error codes) are displayed. The format for error message in direct statement is "ERROR #nn (error number)". The format in statement of program is "ERROR #nn (error number) IN nn (line number)".

See IX. : ERROR MESSAGES for details of each error.

VII. CHAPTER 2. COMMANDS AND STATEMENTS

In the syntax in the following commands and statements, the below mentioned remarks are applied.

a. Enter the item written in capital alphabetic characters as described.

b. The item bracketed by < > has to be designated.

c. The item bracketed by [ ] is optional. When omitted, default values or previously designated values will be applied. Default values mean ones previously designated in Basic.

d. In addition to brackets: (< >) and ([ ]), the symbols such as comma (,), brackets (( )), semi-colon (;), hypen (-), etc. have to be entered correctly in the designated positions.

e. The item having abbreviation symbol (. . .) can be repeated in the limit of one program line.

f. One of the items chosen if some items are parenthesized by { }.

2.1 COMMANDS

2.1.1 APPEND

APPEND loads a program stored on cassette tape into memory and combines with a program already stored in memory. APPEND differs from CLOAD in the point where in APPEND loading from cassette tape can be made without destroying the program already in memory (refer 2.1.4: CLOAD). When APPEND command is completed, Basic returns to common level. This APPEND is used to combine more than two programs. However, the line numbers of the program in cassette tape, which will be added into the memory by APPEND command have to be bigger than the maximum line number of the program already in memory. In case of being smaller, the result of execution cannot be guaranteed.

Syntax:

APPEND <"file name">

Example:

APPEND "ABCD"

NOTE: In case where PDWER [OFF] or [BREAK] is executed during APPEND command, the program in process of loading shall become invalid.

2.1.2 CHAIN

CHAIN loads the designated file from the cassette tape into the computer's memory and begins execution from the designated line number. In case where line number is not designated, execution is made from the smallest line number. The program stored in memory before execution of CHAIN is destroyed. However, variables are not destroyed. In case where tape read error occurs during program loading in CHAIN command, program and data stored in memory become invalid.

Syntax:

CHAIN < "filename" > [, < line number >]

Example:

CHAIN "ABCD" , 100

NOTE: In case where tape read error occurs during program loading, program and data stored in memory becomes invalid.

2.1.3 CLEAR

CLEAR sets all numeric variables to 0 and string variables to the null string. Its optional variables reserve string space.

Syntax:

CLEAR [< string space >]

Examples:

CLEAR	Sets all numeric and string variables to null.
CLEAR 500	Sets all numeric and string variables to null and reserves 500 bytes of memory for string space.

2.1.4 CLOAD

CLOAD loads a program stored on cassette tape into memory. When CLOAD is executed, the system searches to tape for the specified program. Write the program name exactly as it is saved. The prompt and SKIP program appear, displaying all the programs skipped in the search. "FOUND filename" is displayed when the program is located and the prompt (>) appears when completed loading. If another program for example, program named "AA" are saved on the tape before the program to be called out, "SKIP AA" is displayed before finding the specified program.

OPERATION:

1. Connect the Cassette Interface and the Tape Recorder with Cassette Connection Cable according to the drawing in page 11 (V. b. CONNECTING A TAPE RECORDER TO THE INTERFACE).
2. Press PLAY button of Tape Recorder.
3. Write on the computer:
   CLOAD "ABCD (filename)"
4. Press **RETURN** key.

CLOAD? compares a program currently in memory with a program of the same name on tape. If they are the same, ">" appears. If not, error is displayed. The programs are not altered by this command.

Syntax:

CLOAD < "FILE NAME" >
CLOAD? < "FILE NAME" >

Examples:

CLOAD "TEST1"	The program TEST1 is searched for on tape and loaded into memory.
CLOAD? "TEST1"	Compares the program TEST1 on tape with the program currently in memory.

2.1.5 CONSOLE

CONSOLE displays the contents stored in Function keys (F1 to F10). Contents of Function keys are displayed by entering "1" (or any number of 1 to 9) and disappear by "0". The computer is set with 10 Functions of PRINT in F1, INPUT (F2), GOTO (F3), LIST (F4), RUN **RETURN** (F5), CSAVE (F6), KEY (F7), CLOAD (F8), CONSOLE (F9) and CONT **RETURN** (F10) unless new functions are stored through keyboard. When power key ON (in reset state), contents of F1 to F5 are displayed in the lower line of display, and F6 to F10 are displayed when **SHIFT** key is pressed.

Example:

CONSOLE , 1 **RETURN** displays

```
>
PRIN    INPU    GOTO    LIST    RUN
```

2.1.6 CONT

CONT resumes execution of a program after entering STOP or execution of STOP or END statement. Execution resumes at the point the break occurred.

Syntax:

CONT

Example:

	KEY OPERATION	DISPLAY
10 FOR I=1 TO 100	**RUN** [RETURN]	1
20 PRINT I	[RETURN]	BREAK IN 30
30 STOP	CONT [RETURN]	2
40 NEXT I	[RETURN]	BREAK IN 30
	CONT [RETURN]	3

* In the above KEY OPERATION, you can press [F5] key instead of "**RUN** [RETURN] " and [F10] (actually is [SHIFT] [F5]) key for "CONT [RETURN] ". (see 2.1.5: COSOLE).

2.1.7 CSAVE

CSAVE stores the file currently in memory on cassette tape. You can use it in direct mode or in a program statement.

Syntax:

CSAVE < "file name" >

OPERATION:

1. Connect the Cassette Interface and the Tape Recorder using Cassette Connection Cable according to the drewing in page 11 (V. b. CONNECTING A TAPE RECORDER TO THE INTERFACE).
2. Press RECORD button of the Tape Recorder.
3. Write on the computer:

   CSAVE "ABCD (filename)"
4. Press [RETURN] key.

NOTE: No display appears when saving program and ">" (prompt) displays when saving completed.

2.1.8 DELETE

DELETE erases specified program lines.
A period (.) can be used to delete the current line.

Syntax:

DELETE < Line number > [- < line number >]

Example:

DELETE 40	Deletes line 40
DELETE 40 – 100	Deletes lines 40 through 100
DELETE –40	Deletes all lines up to and including 40
DELETE 40–	Deletes all lines bigger than line 40 (including line 40)

2.1.9 LIST

LIST displays a program of the designated line.
If not designate the line, program of the smallest line number is displayed. For example: LIST 100 displays programmed contents of line 100. A period (.) can be used to indicate current line.

2.1.10 LLIST

LLIST prints all or part of the program currently in memory to the printer. Press BREAK key to stop printing a listing.

Syntax:

LLIST [< line number > – < line number >]

Example:

LLIST	Prints the entire program in memory
LLIST 500	Prints line 500
LLIST 150 –	Prints all lines from line 150 to the end of the program
LLIST – 150	Prints all lines from the beginning to line 150
LLIST 50 – 100	Prints all lines from line 50 to line 100

2.1.11 LOCK

LOCK retains a program currently in memory.
When LOCK command is executed, commands related to changing program i.e. NEW, CLEAR, ERASE and CLOAD become invalid. If the commands to change program are entered, "Wrong Program Change" error will occur. Program destruction by misoperation of key can be avoided. To release LOCK command, enter "UNLOCK" command. (see 2.2.15: UNLOCK)

2.1.12 NEW

NEW deletes program and clear all variables.

Syntax:

NEW

NOTE: The NEW command is executed before new program is loaded when in command level. When the command NEW is executed, Basic returns to the command level.

2.1.13 RENUM

RENUM renumbers program lines and changes all line number referred in, following GOTO, GOSUB, THEN, ON . . . . GOTO, ON . . . . GOSUB AND ERL.

Syntax:

RENUM [new number [, old number] [, increment]]

Examples:

a) RENUM — Renumber the entire program starting at line 10 at increments of 10.

BEFORE	AFTER
1 PRINT 1	10 PRINT 1
2 PRINT 2	20 PRINT 2
3 PRINT 3	30 PRINT 3
4 PRINT 4	40 PRINT 4

b) RENUM 2

Renumber the entire program beginning at line 2 at 10-line increments.

BEFORE	AFTER
10 PRINT 1	2 PRINT 1
20 PRINT 2	12 PRINT 2
30 PRINT 3	22 PRINT 3
40 PRINT 4	32 PRINT 4

c) RENUM 100,,50

Renumber the entire program beginning at line 100 at increments of 50.

BEFORE	AFTER
2 PRINT 1	100 PRINT 1
12 PRINT 2	150 PRINT 2
22 PRINT 3	300 PRINT 3
32 PRINT 4	250 PRINT 4

d) RENUM 300, 150, 50

Change line 150 to 300 and all following lines at increments of 50.
(line numbers smaller than 150 are not changed)

BEFORE	AFTER
100 PRINT 1	100 PRINT 1
150 PRINT 2	300 PRINT 2
200 PRINT 3	350 PRINT 3
250 PRINT 4	400 PRINT 4

2.1.14 RUN

RUN executes a program currently in memory.
If designated the line number, program execution starts from the line.

Syntax:

RUN [< line number >]

Example:

RUN 100

Begins execution of the program currently in memory at line 100.

2.1.15 UNLOCK

UNLOCK releases the program locked, and enables editing of program.

Syntax:

UNLOCK

When UNLOCK command is executed, commands of NEW, CLEAR, ERASE, CLOAD and program editing by line numbers become enabled.

2.2 GENERAL STATEMENTS

2.2.1 BEEP

BEEP briefly sounds the buzzer.

Syntax:

BEEP < Integer > , < Integer >

Example:

BEEP 3,10 **RETURN**

NOTE: First number represents musical scale and second number does the length of sound. Number of musical scales is 32 and the length can be designated by positive integer up to 255. The relation between integer and musical scales are described below. The length is integer ÷ 10 seconds and default value is 4. When length is 0, "Illegal Function Call" error occurrs.

INTEGRAL NUMBER	MUSICAL SCALE	FREQUENCY	INTEGRAL NUMBER	MUSICAL SCALE	FREQUENCY
0	NOTHING		16	So#	833 HZ
1	Fa	349 HZ	17	La	BB0
2	Fa#	370	18	La#	933
3	So	393	19	Si	992
4	So#	414	20	Do	1042
5	La	440	21	Do	1116
6	La#	466	22	Re	1179
7	Si	492	23	Re#	1250
8	Do	525	24	Mi	1330
9	Do#	553	25	Fa	1389
10	Re	584	26	Fa#	14BB
11	Re#	625	27	So	1563
12	Mi	658	2B	So#	1645
13	Fa	702	29	La	1736
14	Fa#	735	30	La#	1B3B
15	So	781	31	Si	1953

2.2.2 BOX

BOX prints the box in designated line and color.
The kind of line desired can be selected by entering a number in the range of 0 to 15. 0 produces a solid line while 15 produces a dotted line. When a selection is omitted, the previous designation is chosen. Kind of color ranges 0 to 3. When designation omitted, the present color of pen is chosen. (For color designation, see 2.2.4: COLOR).

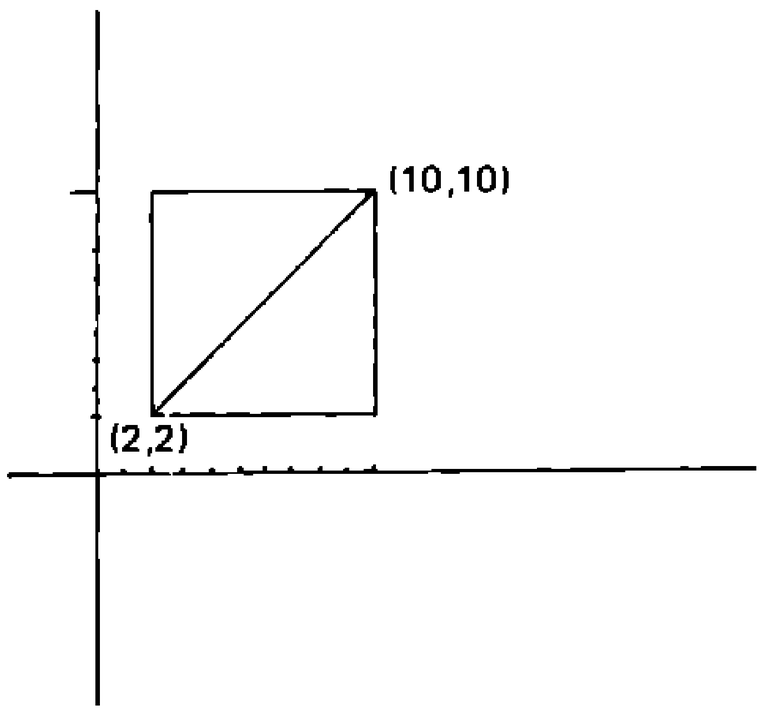
Syntax:

BOX [(X1, Y1)] < – (X2, Y2) > , [Kind of line] , [Kind of color]

This prints the box having a diagonal line between co-ordinates of (X1,Y1) and (X2,Y2) when (X1,Y1) is the original point.

Example:

BOX (2,2) – (10,10), 0, 3 **RETURN**



Left box printed in the solid line in red color

2.2.3 CIRCLE

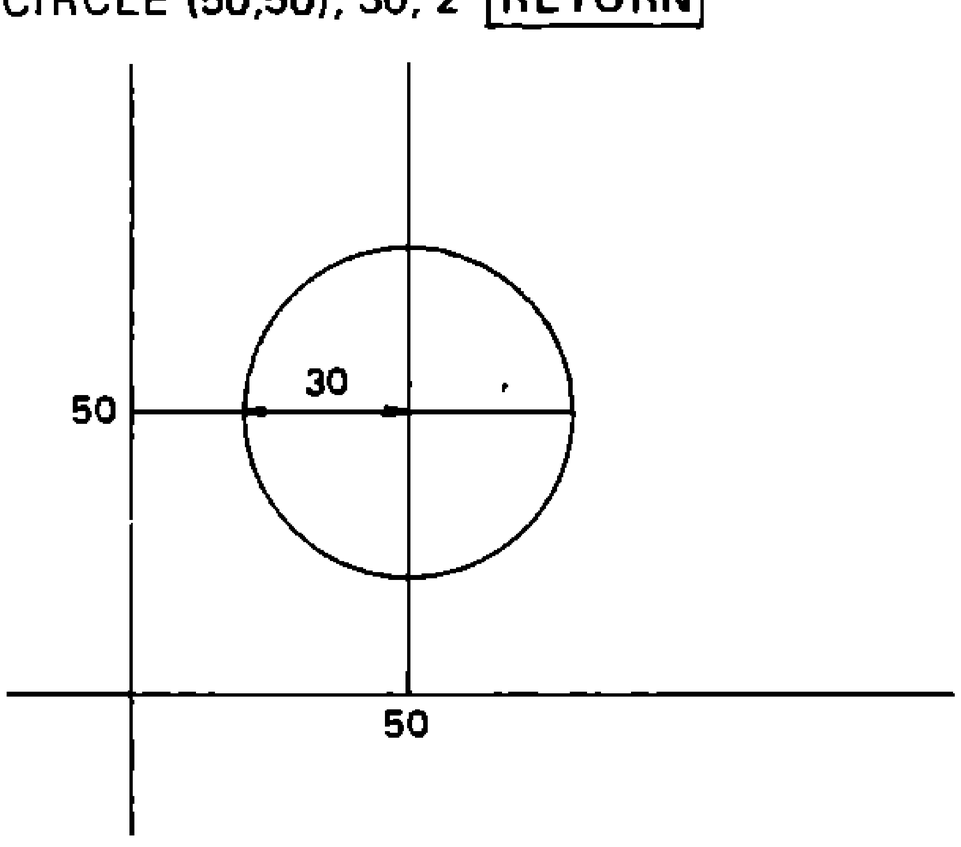
CIRCLE command prints a circle with a center at co-ordinates (X,Y) and a designated radius. Optionally, the color may also be specified.

Syntax:

CIRCLE < (X,Y) > , < Radius > [, < kind of color >]

Example:

CIRCLE (50,50), 30, 2 **RETURN**



Left circle is printed in green.

2.2.4 COLOR

COLOR changes the default color code on the printer. There are four possible color selection (0 to 3). The color numbers are: 0 – black, 1 – blue, 2 – green and 3 – red. If COLOR TEST are entered, numbers of 0 to 3 are printed in respective color of the number.

Syntax:

COLOR n (color number 0 to 3)

Example:

COLOR TEST

0 (in black) 1 (in blue) 2 (in green) 3 (in red)

2.2.5 DATA

DATA supplies data of numeric and string constants to a READ statement. The READ statements read the DATA statements in order of line number. The items contained in the DATA statements can be considered one continuous list. A DATA statement contain as many constants as one program line can contain and can be placed anywhere in a program.

Syntax:

DATA < constant > [, <constant >] . . .

Example:

```
10 REM PROGRAM TO READ LIST
20 REM OF VARIABLES AND PRINT
30 REM THEM OUT
40 FOR I=1 TO 3
50 READ A,B,C
60 PRINT A;B;C
70 NEXT I
80 DATA 1,2,3
90 DATA 4,5,6
100 DATA 7,8,9
110 END
```

RUN RETURN

```
1 2 3
```

RETURN

```
4 5 6
```

RETURN

```
7 8 9
```

2.2.6 DEFFN

The DEFFN function is used to difine new functions. The function name must be preceded by FN, and the variables in its argument list must be separated by comma (,). Functions to be defined can be numeric or string, but their values must match with their argument values. Execute a DEFFN statement to define a function before it is called.

Syntax:

DEFFN < function name > [<(argument list)>] = < function definition >

Example:

* Numeric Function Definition

```
10 DEFFNB (X,Y)=X/Y*100
20 I=20:J=5
30 T=FNB (I,J)
40 PRINT T
50 END
```

RUN RETURN

```
400
```

2.2.7 DEF + INT/SNG/STR

DEF + INT/SNG/STR declare type of a variable or a range of variables as DEFINT for integer, DEFSNG for single precision or DEFSTR for string.

Syntax:

DEF < Type > < Character range > [< character range >]

Type means INT, SNG AND STR

Example:

```
10 DEFINT   L-P     All variables beginning with L,M,N,O and P
                    become integers.
20 DEFSTR   A       All variables beginning with letter A become
                    string variables.
```

2.2.8 DIM

The DIM statement allocates storage for arrays and matrices. The system default for all subscripted variables used without the DIM statement is 10. If the value bigger than maximum designated is used, "Subscript out of Range" error occurs. In DIM statement, all values in array are set to initial value of 0.

Syntax:

DIM < variable > (< minimum value > [, < maximum value > ···])

Examples:

```
100 DIM A (20)
```

```
10 DIM A(5)
20 FOR I=1 TO 5
30 A(I)=I
40 PRINT A;
50 NEXT I
RUN [RETURN]
0 0 0 0 0
```

2.2.9 END

END terminates program execution and returns to command level after closing all files. END statement can appear anywhere and often in a program. The END statement is optional at the end of a program.

Syntax:

END

Example:

```
520 IF K > 1000 THEN END ELSE GOTO 20
```

2.2.10 ERASE

ERASE eliminates previously dimensioned arrays. After you erase an array, you can redimension it and the freed array space in memory can be used for other purposes.

Syntax:

ERASE < array variable > [, < array variable > ···]

Examples:

```
10 DIM A(10)
20 FOR I=1 TO 10
30 PRINT A(I);
40 NEXT I
50 ERASE A
60 DIM A(5)
70 FOR I=1 TO 5
80 PRINT A(I);
90 NEXT I
RUN [RETURN]
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
```

2.2.11 FOR . . . NEXT

The FOR . . . NEXT command loops through the instructions between them a given number of times. FOR opens the loop and must set name of variable, first value and last value and increment step (system default 1). NEXT closes the loop and sends control back to its paired FOR statement. This process continues until the end value of the loop is reached.

Syntax:

FOR < variable name > = x (start value) TO y (end value) STEP z (increment)
NEXT < variable name >

Example:

```
10 REM SIMPLE  ITERATION
20 FOR I=1 TO 5
30 PRINT I;
40 NEXT I
RUN [RETURN]
1 2 3 4 5
```

2.2.12 GOSUB/RETURN

GOSUB orders to skip program execution to a designated subroutine. Subroutine is used when same contents of program process and used in one program. Once program statements of the subroutine are executed, RETURN statement controls back to the line next to GOSUB statement. There can be more than one RETURN statement which correspond to one GOSUB statement. Other subroutine can be used in process of executing a sabroutine (nesting). Such nesting is limited only by capacity of memory.

Syntax:

GOSUB < line number >

Example:

```
10 PRINT "BEFORE SUBROUTINE J=";J
20 GOSUB 60
30 PRINT
40 PRINT "AFTER SUBROUTINE J=";J
50 END
60 J=J + 5
70 RETURN
RUN [RETURN]
BEFORE SUBROUTINE J=0
[RETURN]
[RETURN]
AFTER SUBROUTINE J=5
```

2.2.13 GOTO

GOTO unconditionally branches program execution to a specified line number.

Syntax:

GOTO < line number >

Example:

```
10 READ R
20 PRINT "R= ";R;
30 A=3.14*R^2
40 PRINT "AREA= ";A
50 GOTO 10
60 DATA 5,7,12
RUN [RETURN]
R=5    AREA = 78.5
[RETURN]
R=7    AREA = 153.86
[RETURN]
R=12   AREA = 452.16
[RETURN]
ERROR #4 IN 10
```

2.2.14 IF . . . THEN . . . [~ ELSE]

IF chooses a particular route for program execution, based on conditions established in a logical expression. Use AND if more than one condition is required to meet a desired result.

Syntax:

IF < logical expression > [< AND logical expression >] { GOTO < line number > / THEN < expression > / < line number > }

ELSE < expression > / < line number >

Example:

```
10 INPUT "AGE AND INCOME" ; AGE, INC
20 IF AGE 65 AND INC < 7000 THEN PER = .05 : ELSE PER = .1
30 DIS = 300 * PER
40 PRINT
50 PRINT "YOUR DISCOUNT IS"
60 PRINT USING "$##.##"; DIS
```

KEY OPERATION	DISPLAY
RUN [RETURN]	AGE AND INCOME?
64, 8000	AGE AND INCOME? 64,8000
[RETURN] [RETURN]	YOUR DISCOUNT IS
[RETURN]	$30.00

2.2.15 LET

LET assigns a value or the value of an expression to a variable. Its use is optional.

Syntax:

[LET] < variable > = < value >

Example:

The following examples have same meaning.

```
10 LET D$="HELLO"  ↔  10 D$="HELLO"
20 LET A=100       ↔  20 A=100
30 LET B=8^2       ↔  30 B=B^2
40 LET SUM=A + B   ↔  40 SUM=A + B
```

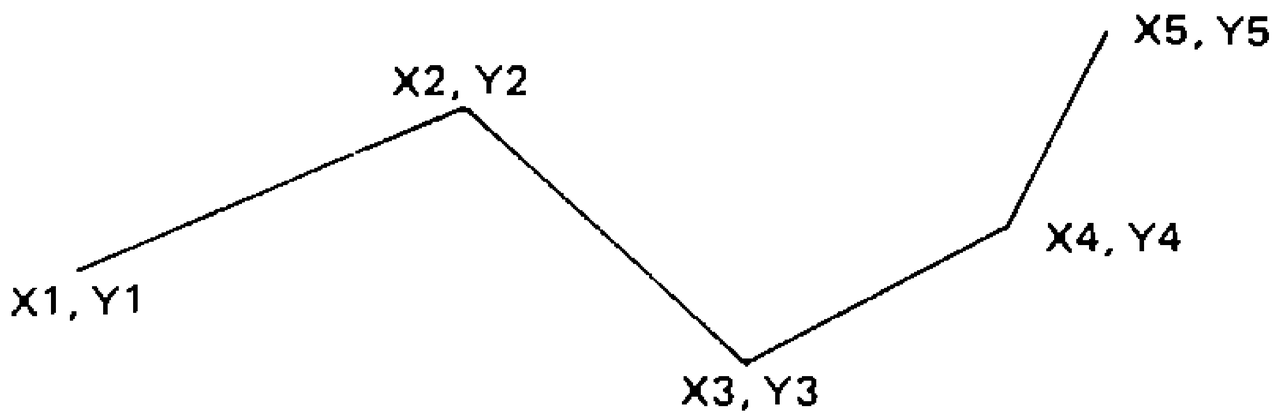
2.2.16 LINE

The LINE command prints a line joining the specified points. Optionally, the color can be specified.

Syntax:

LINE [(X1, Y1)] – (X2, Y2) [. . . – (X7, Y7)] ,
< kind of line > [, kind of color]

Example:



NOTE: When (X1,Y1) is omitted, the co-ordinate is the original point. Kinds of line range from 0 to 15. 0 designates the solid line and 15 is the dotted line. X and Y can be designated up to 7 points. Kind of color is 0 to 3.

2.2.17 MOVE

MOVE moves the pen to the designated coordinate with the pen being lifted.

Syntax:

MOVE [(X,Y)]

Example:

MOVE (100, 100)

NOTE: When (X,Y) is omitted, movement is made to the original point.

2.2.18 MOTOR

MOTOR controls the motor of a cassette tape recorder. For MOTOR to work properly, hook up the wires of the cassette correctly and set its play button to ON. Connect the Cassette Connection Cable as follows.

COLOR	POSITION
Black	Remote Jack
Grey	Earphone Jack
Red	Microphone Jack

Specify a 0 for the switch option to turn the motor off. Any value greater than 0 turns the cassette motor on.

MOTOR used without the switch option turns the cassette motor on or off depending on its last state.

Syntax:

MOTOR [< switch >]

Examples:

MOTOR 1 Turns the cassette motor on.

MOTOR 0 Turns the cassette motor off.

2.2.19 ON ERROR GOTO

ON ERROR GOTO sets up an error trap for any errors that may occur during program execution. When an error is encountered, program control passes to the error routine specified by the GOTO statement.

ON ERROR GOTO 0 disables the error trap routine. Subsequent errors are displayed in their normal fashion. If an ON ERROR GOTO 0 statement is executed in an error trap routine, the appropriate system error message is displayed and control returns to direct mode.

Syntax:

ON ERROR GOTO < line number >

Example:

```
10 ON ERROR GOTO 100
20 FOO I=1 TO 3
30 PRINT I
40 NEXT I
45 STOP
100 PRINT "YOU MADE A MISTAKE"
110 END
RUN [RETURN]
YOU MADE A MISTAKE
```

2.2.20 ON GOSUB/ON GOTO

ON GOSUB/ON GOTO branches program flow to one of several specified line numbers. The branch is contingent on the value of the variable expression. For example, if the value of the expression is 3, control branches to the line number specified by the third value in the line list.

Syntax:

ON < expression > { GOSUB / GOTO } < line > [, line] . . .

Example:

```
10 PRINT "ENTER A 1 – IF YOU WISH TO WITHDRAW 2 – TO DEPOSIT 3 – TO
   WITHDRAW FROM CHECKING ACC. 4 – TO DEPOSIT CHECKING ACC.": INPUT A
20 ON A GOTO 50, 70, 90, 100
 ⋮
```

A 2 entered for the variable A branches control to line 70. A 3 branches control to line 90. A negative number for the variable A cuases the error message "Illegal Function Call" in line 20. A value greater than the amount of line numbers in the list, in this example a value greater than 4, branches control to the next logical line.

2.2.21 OPTION BASE

OPTION BASE announces minimum value of array.

Syntax:

OPTION BASE n (n is 1 or 0)

NOTE: If OPTION BASE is omitted, the minimum value of array is 0. When OPTION 1 is executed, minimum value of array is 1. This command can be used only once before DIM.

2.2.22 ORIGIN

ORIGIN sets the origin in commands of BOX, CIRCLE, LINE, etc.

Syntax:

ORIGIN

Example:

```
20 ORIGIN
30 CIRCLE (10, 20), 10, 3
```

2.2.23 POKE

POKE writes a byte of information into memory. The integer "I" is the memory address which must be in the range of 0 to 65535. A specified J is the data to be written into memory which must be in the range of 0 to 255. I and J can be hex, octal or decimal values. You use PEEK to read a particular byte of information memory. The range of the integers for PEEK is the same that for POKE.
(see 3.3.3: PEEK)

Syntax:

```
POKE I,J
PEEK (I)
```

Example:

```
10 POKE &H5A00, &HFF
20 PRINT PEEK (&H5A00)
RUN RETURN
 62
```

2.2.24 RANDOMIZE

RANDOMIZE generates a series of random numbers.

Syntax:

RANDOMIZE [Formula]

Example:

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
50 PRINT
```

KEY OPERATION	DISPLAY				
RUN RETURN	(0–65529)?				
3	(0–65529)? 3				
RETURN	.88598	.484668	.586328	.119452	.709225
RUN RETURN	(0–65529)?				
4	(0–65529)? 4				
RETURN	.803506	.162462	.929364	.292443	.322921

If formula is omitted then execution is interrupted and the user is prompted for a seed number 0 and 65529 is displayed.

2.2.25 READ

READ assigns the data items of a DATA statement to variables on a one to one basis. The READ statement can be of numeric or string variebles, but must agree with the data type of the items in the DATA statement.

One READ statement can access several DATA statements and several READ statements can access one DATA statement.

If the numger of variables in a READ statement exceeds the number of data items, the "Out of Data" error is displayed. If the numberr of DATA items exceeds the variables of a READ statement, they are ignored.

Syntax:

READ < variable, variable, . . . >

Examples:

```
10 FOR I=1 TO 5
20 READ A
30 PRINT A;
40 NEXT I
50 DATA 1,2,3,4,5,7
RUN [RETURN]
1 2 3 4 5
```

2.2.26 RLINE (RELATIVE LINE)

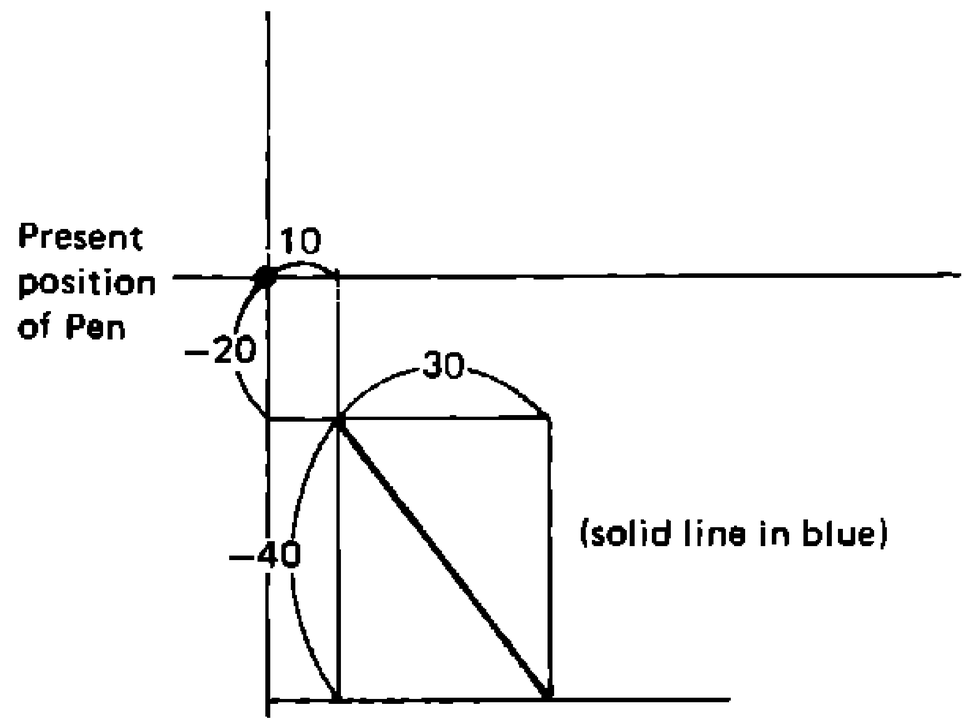
RLINE draws the line on the printer in designated type and color.

Syntax:

RLINE [(X1, Y1)] – (X2, Y2) [. . . – (X7, Y7)] [, kind of line, kind of color]]

Example:

10 RLINE (10, –20) – (30, –40), 0, 1 **RUN** **[RETURN]**



NOTE: In LINE, the default starting position is the designated origin, but in RLINE, the default is the present position of the pen.

2.2.27 REM or " ' "

REM is a nonexecutable statement used for explanatory remarks concerning a program. Remarks can follow a program line if you precede the remark with an apostrophe. You can also use the apostrophe ('), as an abbreviation of REM.

Syntax:

REM < remark >

Example:

```
120 REM CALCULATE AVERAGE VELOCITY
130 FOR I=1 TO 20
140 SUM=SUM + V (I)
or
120 FOR I=1 TO 20 'CALCULATE AVERAGE VELOCITY
130 SUM=SUM + V (I)
140 NEXT I
```

2.2.28 RESTORE

RESTORE resets the items of a DATA statement so they can be re-read by a READ statement. RESTORE instructs following READ statement to read data in DATA statement from first. If specified line number, the READ statement reads data in DATA statement of the designated line.

Syntax:

RESTORE [< line number >]

Example:

```
10 READ A, B, C
20 RESTORE
30 READ D, E, F
40 RESTORE 90
50 READ G, H, I
60 PRINT A; B; C; D; E; F; G; H; I
70 DATA 1, 2, 3, 4, 5
80 DATA 6, 7, B, 9, 10
90 DATA 11, 12, 13, 14, 15
RUN [RETURN]
1  2  3  1  2  3  11  12  13
```

2.2.29 RESET

RESET is used to reset the position of the writing pen in the printer. When executed the pen will be shifted to the extreme left of the printable area without drawing. The printer then returns to character mode and resets the co-ordinates of the origin to the new position.

Syntax:

RESET

Example:

```
10 COLOR 1
20 MOVE (10, 20)
30 RLINE (10, –20) – (30, 10), 0
40 RESET
50 PRINT "TITLE:" ; X$
```

2.2.30 RESUME

Use RESUME to continue program execution after performing an error recovery procedure.
Its options are as follows.

RESUME [0] – resumes execution at the statement that caused the arror.

RESUME NEXT – resumes program execution at the line immediately following the faulty line.

RESUME line number – resumes program execution at the specified line number.

Syntax:

```
RESUME [ 0 ]
RESUME NEXT
RESUME < LINE NUMBER >
```

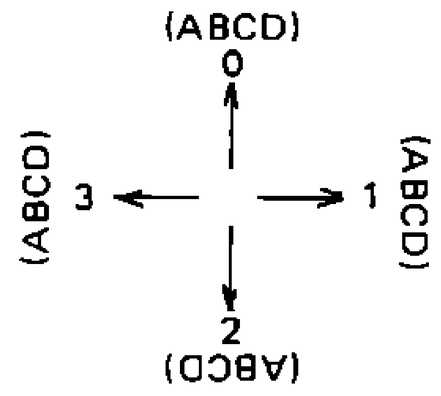
Example:

```
10 ON ERROR GOTO 250
  :
250 IF ERR=230 THEN PRINT "TRY AGAIN"
260 IF ERL=90 THEN RESUME 80
```

2.2.31 ROTATE

ROTATE < Integer >

ROTATE designates 0 – 3 for the inclination of characters.



Example:

```
10 MOVE (100,100)
20 ROTATE 1
30 LPRINT "ABCD"
RUN [RETURN]
```

2.2.32 STOP

STOP terminates program execution and returns control to direct mode. You can use STOP at any location in a program. When STOP is encountered, the message "BREAK IN LINE nnn" is displayed where nnn is the line number of the statement.

To resume execution of the program after a break, simply type in "CONT" in direct mode.

Syntax:

```
STOP
```

Example:

```
10 INPUT A, B, C
20 K=A^2*5.3:L=B^3/.26
30 STOP
40 M=C*K+100: PRINT M
```

KEY OPERATION	DISPLAY
RUN [RETURN]	?
1,2,3 [RETURN]	BREAK IN 30
PRINT [L] [RETURN]	30.7692
[RETURN]	>
CONT [RETURN]	115.9

2.2.33 SCALE

The SCALE command is used to set the size of characters. The SCALE factor is set to 1 when the machine is first turned on. The scaling size can range from 0 to which is the smallest, to 15 which is the largest.

Syntax:

SCALE < Integer >

Example:

```
10 SCALE 0
20 LPRINT "ABCD"
```

2.2.34 TRON/TROFF

TRON traces the execution of program. When TRON is executed, either in direct or indirect mode, line numbers of programs which are executed are displayed or printed. TROFF (or NEW) resets the program tracing. The trace feature is useful for debugging programs. It allows the user to single step through the program line by line.

Example:

```
10 I=1
20 J=2
30 K=3
40 PRINT I;J;K
50 END
```

KEY OPERATION	DISPLAY
TRON RETURN	>
RUN RETURN	< 10 >
RETURN	< 20 >
RETURN	< 30 >
RETURN	< 40 >
RETURN	1 2 3
RETURN	< 50 >

2.3 INPUT/OUTPUT STATEMENTS

2.3.1 INPUT

INPUT accepts data from the keyboard during program execution. When encountered, the system prompts you to enter data by displaying a question mark.

Data items must be separated by commas and coincide with the number of variables listed after the INPUT statement. If too few data items are supplied, the system displays two question marks "??" and waits for further input. If too many data items are entered, the message "EXTRA IGNORED" is displayed and execution continues. Data input must correspond to the data types of the variables.

You also have the option of including a prompt string with the INPUT statement to aid proper data input. When using a prompt string, the system displays the prompt immediately followed by a question mark.

Syntax:

INPUT ["prompt string";] variable, variable . . .

Examples:

```
10 INPUT X
20 PRINT X "SQUARED IS"; X^2
30 END
```

KEY OPERATION	DISPLAY
RUN RETURN	?
5 RETURN	5 SQUARED IS 25

```
10 PI=3.14
20 INPUT "WHAT IS THE RADIUS" ;R
30 A=PI*R^2
40 PRINT "THE AREA OF CIRCLE IS" ;  A
50 PRINT
60 GOTO 20
```

KEY OPERATION	DISPLAY
RUN RETURN	WHAT IS RADIUS?
7.4 RETURN	THE AREA OF CIRCLE IS 171.946

2.3.2 INPUT#

INPUT# is used to input a record from a previously created sequential file. It initializes its variable list to the appropriate record items. When a file number of "–1" is used, the computer will fetch a record from a cassette file.
Variables in an INPUT# statement must correspond to each output variable of PRINT# statement.

Syntax:

INPUT# file number, < variable, variable, ... >

Example:

```
100 INPUT# – 1, A$, X
```

2.3.3 LOCATE

LOCATE moves the cursor at a specified location on the screen.

Syntax:

LDCATE < Integer >

Example:

```
10 LOCATE 10
20 PRINT "HELLO"
```

NOTE: 0 – 255 can be specified as integers. When integers bigger than 48 (function key display), the cursor is set to 0.

2.3.4 LPRINT

LPRINT is same to PRINT except that output is printed on the paper.

Syntax:

PRINT " < prompt > " [, < item >] . . .

NOTE: After graphic commands (BOX, CIRCLE, COLOR, RLINE, LINE, MOVE, ORIGIN, ROTATE, SCALE), RESET must be entered for text printing.

2.3.5 PRINT

PRINT displays the values of numeric and string variables, expressions, and user-defined prompts. The place those items are displayed, is determined by the punctuation used to separate the list of items to be printed.

One display line in the Basic has 1B spaces. If each item is punctuated by comma (,) in a statement, next printing is made in new line. If punctuated by semicolon (;), next printing is made following the previous item (Example 2). A PRINT statement not terminated by a comma or a semicolon is followed by a carriage return (Example 1).

A question mark (?) can be used as an abbreviation of PRINT.

Syntax:

PRINT "prompt" ; < item > [, (or ;) item] [, (or ;) item] . . .

Example 1:

```
10 FOR I=1 TO 6
20 PRINT I
30 NEXT I
```

KEY OPERATION	DISPLAY
RUN [RETURN]	1
[RETURN]	2
[RETURN]	3
[RETURN]	4
⋮	

Example 2:

```
10 FOR I=1 TO 6
20 PRINT I;I+1;
30 NEXT I
```

KEY OPERATION	DISPLAY
RUN [RETURN]	1 2 2 3 3 4 4 5 5 6 6 7

NOTE: A PRINT statement used alone skips a line.

2.3.6 PRINT#

The PRINT# command is used to write a record to a sequential file. A file number of "–1" should be used to write to a cassette file.

Syntax:

PRINT# file number, < variable > . . .

Example:

```
100 A%=10 : B$="XYZ"
110 PRINT#–1, A%, B$
```

2.3.7 PRINT USING

The PRINT USING command allows the use to control the formating of the output. The format code, which consists of special symbols, is enclosed in quotation marks and is used to tell the computer how to print out the desired information.

2.3.7.1 String Data

There are two formatting characters that can be used to print string data to a field:

!	specifies that only the first character of a given string is to be printed.
"& (spaces) &"	displays a specified number of characters in a given string, determined by two plus the number of spaces between the delimiter "&".

Syntax 1:

PRINT USING "format symbol"; string variable; string variable; . . .

Example:

```
10 A$="GOOD"
```

```
20 B$="KARMA"
30 PRINT USING " ! ";A$;B$
40 PRINT USING " !& &";A$;B$
50 PRINT USING "& (3 spaces) &";A$;B$;"!!"
```

KEY OPERATION	DISPLAY
RUN [RETURN]	GK
[RETURN]	GKAR
[RETURN]	GOOD KARMA!!

2.3.7.2 Numeric Data

There are several formatting characters which can be used to print numeric data in a field.

Syntax:

PRINT USING "format string" ; variable , variable . . .

Example:

Represents each digit of a field. Decimal point can be used at any position of the field. Zeros are placed in all unused positions in the right of the decimal point and unused positions in the left of the decimal are filled with balnks. Fractions are displayed with at least one zero preceded by blanks left of the decimal. Data to be placed in a field must contain its own decimal point for proper results. Numbers are rounded when necessary.

PRINT USING "###.##" ; 1. , .2 , 3.456 displays 1.00 0.20 3.46

+ Prints the sign of plus (+) or minus (—) of the value when placed at the beginning or the last of a format string.

— Displays negative values with a trailing minus sign when at the end of a format string.

PRINT USING "+###.##" ; 111.22 , 222.33 , —444.44 displays
+111.22 +222.33 —444.44

PRINT USING "###.##—" ; 111.22 , 222.33 , —444.44 displays
111.22 222.33 444.44—

****** Fills all leading spaces with asterisks when at the beginning of a format field. They are also considered two digit positions.

PRINT USING "**##.##" ; 12.39 ; —0.9 , 765.1 , 1.0 displays
12.39—0.90*765.10***1.00

$$ Same as in ** but prints a $ only though taking two digit positions.

****$** Makes both functions in above ** and $$.

, Places a comma every third digit in the left of the decimal when used immediately left of a decimal point. When placed at the end of a format string, the comma is displayed as part of the defined filed.

PRINT USING "#####,.##" ; 12345.5 displays 12,345.50

PRINT USING "#####.##," ; 12345.5 displays 12345.50,

^^^^ Specifies exponential notation when placed at the end of a format string. Those marks reserve space for E+nn to be displayed.

PRINT USING "###.## ^^^^ " ; 123.56 displays 12.36E+01

% Is displayed by the system to warn that a value is greater than a specified field.

PRINT USING "##.##" ; 123.22 displays %123.22

2.3.8 PAUSE

The PAUSE command is similar to the PRINT command. PAUSE is used to display the expression for the time specified by WAIT statement.

Syntax:

PAUSE [USING < format symbol > ;] expression

Example:

```
10 A=5 : B$="123"
20 WAIT A*6
30 PAUSE B$
RUN [RETURN]
123
```

In the above example, 123 is displayed for three seconds.
(Length of display time (second) is number designated in WAIT statement ÷ 10)

2.3.9 WAIT

WAIT being used together with PAUSE, specifies the displaying time of the expression in PAUSE statement. (See 2.3.7: PAUSE)

Syntax:

WAIT < Integer >

2.4 FUNCTION STATEMENT

2.4.1 KEY

KEY stores maximum 15 characters of string or control character into Function keys. The characters which cannot be entered from keyboard are specified using the function CHR$(n) and appended to the string using a plus (+) sign though characters of CHR$(13) are not displayed.
(See page 7: FUNCTION KEYS.)

Syntax:

KEY < key number > , < "string" >

Example:

KEY 5, "RETURN" + CHR$(13)

2.4.2 KEY LIST

KEY LIST displays the contents stored in Function keys.

VIII. CHAPTER 3. FUNCTIONS

The functions can be called from any program without further definition.

Arguments to functions are always enclosed in parentheses and are abbreviated as follows.

X and Y — represent numeric expressions

I and J — represent integer expressions

A$ and B$ — represent string expressions

If you supply a floating-point value where an integer is required, the integer is used, truncating the fractional portion.

3.1 NUMERIC FUNCTIONS

3.1.1 ABS

ABS returns the absolute value of the expression X.

Syntax:

```
ABS (X)
```

Example:

```
PRINT ABS ( 7*(–5) )
RETURN
35
```

3.1.2 ATN

ATN returns the arctangent of X in radians. The result is in the range $-\pi/2$ to $\pi/2$. The expression X can be any numeric type, but the evaluation of ATN is always performed in single precision.

Syntax:

```
ATN(X)
```

Example:

```
10 INPUT X
20 PRINT ATN (X)
```

KEY OPERATION	DISPLAY
RUN RETURN	?
3 RETURN	1.24905

3.1.3 COS

COS returns the cosine of X in radians. The calculation of COS(X) is performed in single precision.

Syntax:

```
COS (X)
```

Example:

```
10 X=2*COS (.4)
20 PRINT X
RUN RETURN
1.B4212
```

3.1.4 EXP

EXP returns "e" to the power of X, which must be less than 87.3366. If EXP overflows, the "Overflow" error message is displayed.

Syntax:

EXP (X)

Example:

```
10 A=5
20 PRINT EXP (A-1)
RUN [RETURN]
54.5982
```

3.1.5 FIX

FIX returns the integer part of X. FIX (X) is equivalent to SGN(X)*INT (ABS(X)). The major difference between FIX and INT is that FIX returns bigger integer than X when X is negative value.

Syntax:

FIX (X)

Example:

```
PRINT FIX (58.75)          PRINT FIX (-58.75)
[RETURN]                   [RETURN]
58                         -58
```

3.1.6 INT

INT returns the largest integer which does not exceed X.

Syntax:

INT (X)

Example:

```
PRINT INT (99.89)          PRINT INT (-12.11)
[RETURN]                   [RETURN]
99                         -13
```

3.1.7 LOG

LOG returns the natural logarithm of X. X must be larger than zero.

Syntax:

LOG (X)

Example:

```
PRINT LOG (45/7)
[RETURN]
1.86075
```

3.1.8 RND

RND returns a random number between 0 and 1. The value of I will vary the random number returned as follows:

I=0	generates the same number for a given value I.
I>0	generates random numbers between 0 and I.

Syntax:

RND (I)

Example:

```
10 FOR I=1 TO 5
20 PRINT INT (RND (1)*100);
30 NEXT I
RUN [RETURN]
24 30 31 51 5
```

3.1.9 SGN

SGN returns the sign of the value X.

Syntax:

SGN (X)

If X>0, SGN (X) returns 1
If X=0, SGN (X) returns 0
If X<0, SGN (X) returns −1

Example:

ON SGN (X) + 2 GOTO 100, 200, 300

Control branches to 100 if X is negative, to 200 if X is 0, and to 300 if X is positive.

3.1.10 SIN

SIN returns the sine of X in radians. SIN (X) is calculated in single precision.

Syntax:

SIN (X)

Example:

```
PRINT SIN (9)
[RETURN]
.412118
```

3.1.11 SQR

SQR returns the square root of X.

Syntax:

SQR (X)

Example:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X, SQR (X)
30 NEXT X
```

KEY OPERATION	DISPLAY	
RUN [RETURN]	10	3.16228
[RETURN]	15	3.87298
[RETURN]	20	4.47214
[RETURN]	25	5

3.1.12 TAN

TAN returns the tangent of X in radians. TAN (X) is calculated in single precision. If TAN overflows, the "Overflow" error message is displayed and execution halts.

Syntax:

TAX (X)

Example:

```
10 G=5
20 Y=G*TAN (6)/2
30 PRINT Y
RUN [RETURN]
-.727516
```

3.1.13 TAB

TAB spaces to position I. If the current print position is already beyond space I, TAB has no effect. Space 0 is the leftmost position. I must be in the range 0 to 255. TAB can only be used in PRINT, LPRINT, and PAUSE statements.

Syntax:

TAB (I)

Example:

```
10 PRINT "NAME" TAB (15) "AMOUNT" : PRINT
RUN [RETURN]
```

3.2 CHARACTER FUNCTIONS

3.2.1 ASC

ASC returns a ASCII code of the first character of the string X$.

Syntax:

ASC (X$)

Example:

```
10 X$="TEST"
20 PRINT ASC (X$)
RUN [RETURN]
84
```

3.2.2 CHR$

CHR$ returns the character whose ASCII code is I. You normally use this statement to output special characters.

Syntax:

CHR$(I)

Example:

```
PRINT CHR$ (66)
[RETURN]
B
```

3.2.3 HEX$

HEX$ returns a string that represents the hexadecimal value of the decimal argument. X is truncated to an integer before HEX$ (X) is evaluated.

Syntax:

HEX$ (X)

Example:

```
10 INPUT X                          RUN [RETURN]          ?
20 A$=HEX$ (X)                      32 [RETURN]           32 DECIMAL IS 20 HEXADECIMAL
30 PRINT X "DECIMAL IS"
   A$ "HEXADECIMAL"
```

KEY OPERATION | DISPLAY

3.2.4 LEFT$

LEFT$ returns a string of I characters from leftmost of X$. I must be in the range 0 to 255. If I is greater than LEN (X$), the entire string (X$) is returned. If I=0, the null string (length zero) is returned.

Syntax:

LEFT$ (X$,I)

Example:

```
10 A$="CMT BASIC"
20 B$=LEFT$ (A$,5)
30 PRINT B$
RUN [RETURN]
CMT B
```

3.2.5 LEN

LEN returns the number of character in X$. Non-printing characters and blanks are counted.

Syntax:

LEN (X$)

Example:

```
10 X$ = "CMT BASIC"
20 PRINT LEN (X$)
RUN [RETURN]
9
```

3.2.6 MID$

MID$ returns a string of J characters in X$, beginning with the Ith character. I and J must be in the range of 0 to 255. If you omit J or if there are fewer than J characters to the right of the Ith character, all right characters beginning with the Ith character are returned. If I is less than LEN (X$), MID$ returns a null string.

Syntax:

MID$ (X$, I [,J])

Example:

```
10 A$="GOOD"
20 B$=";;MORNING EVENING AFTERNOON"
30 PRINT A$; MID$ (B$,8,8)
RUN [RETURN]
GOOD EVENING
```

3.2.7 OCT$

OCT$ returns a string that represents the octal value of the decimal argument. X is rounded to an integer before OCT$ (X) is evaluated.

Syntax:

```
OCT$ (X)
```

Example:

```
PRINT OCT$ (24)
[RETURN]
30
```

3.2.8 RIGHT$

RIGHT$ returns a string of I characters from the rightmost of X$.
(See 3.2.4: LEFT$).

3.2.9 STR$

STR$ returns a string which represents numeric value of X.

Syntax:

```
STR$ (X)
```

Example:

```
PRINT "$" + STR$ (15 + 3)
[RETURN]
$18
```

3.2.10 VAL

VAL returns the numerical value of string X$. If the first character of X$ is not +, –, &, or a figure, VAL (X$) is 0.

Syntax:

```
VAL (X$)
```

Example:

	KEY OPERATION	DISPLAY
10 INPUT A$	RUN [RETURN]	?
20 PRINT VAL (A$) + 32	25 [RETURN]	57

3.3 GENERAL FUNCTION

3.3.1 ERR, ERL

ERR and ERL are used to handle errors in a error trap routine.
When an error trap is entered, the error code is stored in the variable ERR and the line number where the error occurred is stored in the variable ERL. You normally use ERR and ERL in IF . . . THEN statements to control the flow in the error handling routine.

Syntax:

IF ERR = error code THEN.....
IF ERL = line number THEN . . .

Example:

```
10 ON ERROR GOTO 500
.
.
500 REM error handling  routine
510 IF ERR=4 THEN RESUME 100
520 IF ERL=150 THEN RESUME 150
530 X=15
540 RESUME 0
```

3.3.2 FRE

Arguments to FRE are dummy arguments. If the argument is 0 (numeric), FRE returns the of bytes in memory not being used. If the argument is a string, FRE return the number of free bytes in string space.

Syntax:

FRE (0)
FRE (X$)

Example:

```
PRINT FRE (0)
RETURN
4034
```

3.3.3 PEEK

PEEK returns the byte (decimal integer in the range of 0 to 255) read from memory location I, which must be in range 0 to 65536. PEEK is the complementary function to the POKE statement.

Syntax:

PEEK (I)

Example:

```
PRINT PEEK (741)
RETURN
52
```

3.4 INPUT/OUTPUT FUNCTION

3.4.1 INKEY$

In INKEY$, a character is entered when key is depressed. When key is not pressed, null string is entered.

IX. ERROR MESSAGES

Error Code	Error Message	Meaning
1	**NEXT without FOR**	FOR and NEXT statements do not correspond properly (too many NEXT statements).
2	**Syntax error**	A line has been encountered with an incorrect sequence of characters (such as misspelled statement). In INPUT#-1, the data in cassette tape has an error.
3	**RETURN without GOSUB**	RETURN statements and GOSUB statements do not correspond properly.
4	**Out of Data**	There are no DATA statements which are read with READ statements.
5	**Illegal function call**	A parameter that is out of range has been passed to a math or string function.
6	**Overflow**	The result of calculation is too large to be represented in Basic.
7	**Out of memory**	A program is too large or has too large array variables.
8	**Undefined line number**	A reference has been made to a line number that does not exist.
9	**Subscript out of range**	An array element has been referenced with a subscript that is outside the dimension of the array.
10	**Redimensioned array**	Two DIM statements have been given for the same array.
11	**Division by zero**	Division by zero has been encountered in an expression.
12	**Illegal direct**	A statement that is illegal in direct mode has been entered as a direct mode command
13	**Type mismatch**	A derived value type does not match. (as in numeric value and string)
14	**Out of string space**	String variables exceed the amount of allocated string space.
15	**String too long**	A string is too long (more than 255 characters).

Error Code	Error Message	Meaning
16	**String formula too complex**	A string expression is too long or too complex. (An illegal nesting level of parentheses, etc.)
17	**Can't continue**	A nonexecutable CONT command has been encountered. (The pointer is destroyed, etc.)
18	**Undefined User Function**	A user function has been called before the function definition (DEF statement is given).
19	**No RESUME**	An error trapping routine with no RESUME statement has been encountered.
20	**RESUME without error**	A RESUME statement has been encountered before an error trapping routine is entered.
21	**Unprintable error**	An error message is not available for the error condition that exists.
22	**Missing Operand**	An expression contains an operator with no operand following it.
24	**Tape read ERROR**	An error has been found in input from a cassette tape.
25	**Bad File Data**	File data on tape is improperly formatted.
26	**Wrong Program Change**	An attempt has been made to execute program change with memory "LOCK" being made.

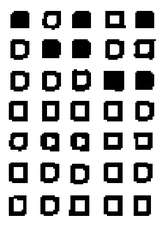
X. USER DEFINING CHARACTERS

Data of 10 characters (50 bytes) from address FCAE are available for the user, and corresponds to the ASCII codes or E0–E9 or F0–F9.

When 5 bytes data is stored by POKE command after address FCAE, desired graphic characters can be displayed on LCD by PRINT and CHR$ (&HEQ). 7 bits in ASCII code are used and 8th bit is ignored.

Example:

```
10 POKE &HFCAE, 1 : POKE &HFCAF, 2 : POKE &HFCB0, 3:POKE &HFCB1,
   4 : POKE &HFCB2,5
20 PRINT CHR$ (&HE0)
RUN
```



Left character is displayed on display.

• SETTING OF PRINTER WIDTH

Initial value : 18
Address = FC34

In the case where printer width is other than the above, change can be made by POKE command. When data is output continuously by " , " in LPRINT, printing can be made in accordance to paper width.

XI. ASCII CHARACTER CODE CHART

Low Bit Positions b4, b3, b2, b1 ↓ / Upper Bit Positions → b7, b6, b5	Hexa decimal	000	001	010	011	100	101
0000	0			SPACE	0	@	P
0001	1			!	1	A	Q
0010	2			"	2	B	R
0011	3			#	3	C	S
0100	4			$	4	D	T
0101	5			%	5	E	U
0110	6			&	6	F	V
0111	7			'	7	G	W
1000	8			(	8	H	X
1001	9			)	9	I	Y
1010	A			*	:	J	Z
1011	B			+	;	K	[
1100	C			,	<	L	¥
1101	D			—	=	M	]
1110	E			.	>	N	^
1111	F			/	?	O	_

XII. PROGRAMMING EXAMPLES

Example 1:

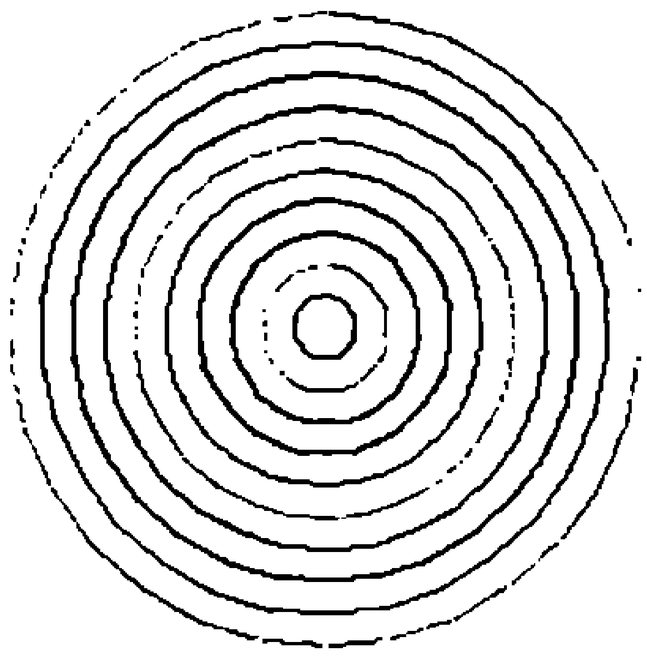
```
50* 123 = 6150
50* 123 = 6150
50* 123 = 6150
50* 123 = 6150
```

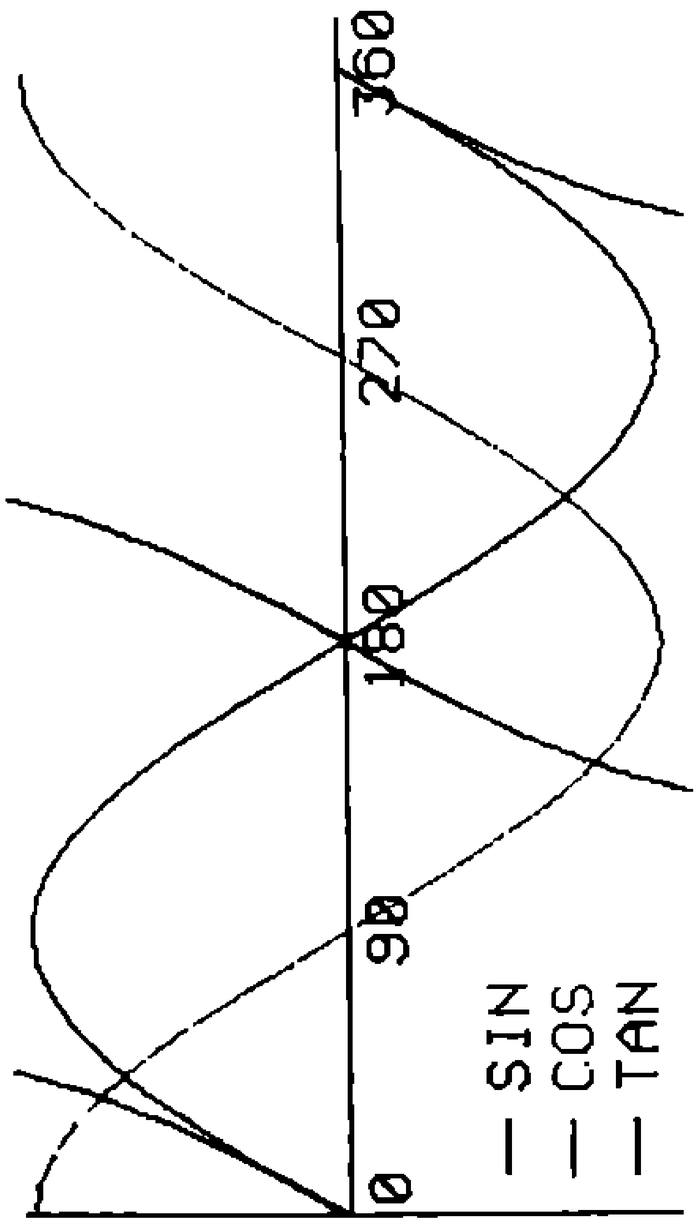
Program for tha above exampla

```
10 FOR C=0 TO 3
20 COLOR C:RESET
30 INPUT S
40 K=50*S
50 LPRINT "50*";S"
=";K
60 NEXT C
```

Example 2:

```
DEMONSTRATION
TPC-8300
```





Program for the left example

```
10 SCALE 1:MOVE(20
,0)
20 COLOR 1
30 LPRINT "DEMONST
RATION"
40 SCALE 2:MOVE(0,
-50)
50 COLOR 2
60 LPRINT "TPC-830
0"
70 RESET
80 ' ***CIRCLE***
90 MOVE (107,-120)
:ORIGIN
100 FOR R=10 TO 10
0 STEP 10
110 GOSUB 160
120 CIRCLE(0,0),R,
C
130 NEXT
140 MOVE(-107,-110
):RESET
150 GOTO 180
160 C=C+1:IF C>3 T
HEN C=0
170 RETURN
180 '*SIN*COS*TAN*

190 MOVE(0,-360):R
ESET:SCALE 1
200 ROTATE 3:MOVE(
107,0):ORIGIN
210 LINE(-105,0)-(
105,0),0,0:LINE(0,
0)-(0,375)
220 GOSUB 460
230 FOR D=1 TO 3
240 K=57.2958:X1=0
:Y1=0
```

```
250 FOR Y2=0 TO 36
0 STEP 10
260 ON D GOTO 340,
390,360
270 LINE(-X1,Y1)-(
-X2,Y2),0,D
280 X1=X2:Y1=Y2
290 NEXT Y2
300 NEXT D
310 GOSUB 520
320 MOVE(0,0): RES
ET
330 END
340 RD=Y2/K:X2=SIN
(RD)*100:GOTO 270
350 RD=Y2/K:X2=COS
(RD)*100:GOTO 270
360 IF Y2=60 GOTO
410
370 IF Y2=240 GOTO
 440
380 RD=Y2/K:X2=TAN
(RD)*100:GOTO 270
390 IF Y2=0 THEN X
1=100
400 GOTO 350
410 Y2=120:MOVE(12
0,120)
420 RD=Y2/K:X2=TAN
(RD)*100
430 GOTO 280
440 Y2=300:MOVE(12
0,300)
450 GOTO 420
460 MOVE(20,5):LPR
INT "0"
470 MOVE(20,80):LP
RINT "90"
480 MOVE(20,165):L
PRINT "180"
490 MOVE(20,255):L
PRINT "270"
500 MOVE(20,345):L
PRINT "360"
510 RETURN
520 LINE(50,10)-(5
0,30),0,1:MOVE(58,
40):LPRINT "SIN"
530 LINE(70,10)-(7
0,30),0,2:MOVE(78,
40):LPRINT "COS"
540 LINE(90,10)-(9
0,30),0,3:MOVE(98,
40):LPRINT "TAN"
550 RETURN
```

Example 3:

Program for a music

```
1000 BEEP 19,2
1010 BEEP 19,2
1020 BEEP 19,2
1030 BEEP 19,2
1040 BEEP 17,2
1050 BEEP 15,2
1060 BEEP 15,2
1070 BEEP 14,2
1080 BEEP 12,2
1090 BEEP 12,2
1100 BEEP 15,2
1110 BEEP 19,2
1120 BEEP 24,2
1130 BEEP 24,2
1140 BEEP 24,2
1150 BEEP 24,2
1160 BEEP 22,2
1170 BEEP 20,2
1180 BEEP 20,2
1190 BEEP 19,2
1200 BEEP 17,2
1210 BEEP 17,2
1220 BEEP 19,2
1230 BEEP 20,2
1240 BEEP 19,2
1250 BEEP 20,2
1260 BEEP 19,2
1270 BEEP 23,2
1280 BEEP 20,2
1290 BEEP 19,2
1300 BEEP 19,2
1310 BEEP 17,2
1320 BEEP 15,2
1330 BEEP 15,2
1340 BEEP 14,2
1350 BEEP 12,2
1360 BEEP 14,2
1370 BEEP 14,2
1380 BEEP 14,2
1390 BEEP 14,2
1400 BEEP 15,2
1410 BEEP 14,2
1420 BEEP 12,2
1430 BEEP 12,2
1440 BEEP 12,2
1450 BEEP 12,2
```

IMPRIME AU JAPON
PRINTED IN JAPAN 8300-1