

# SHARP®

## POCKET COMPUTER

### MODEL PC-1360

## OPERATION MANUAL

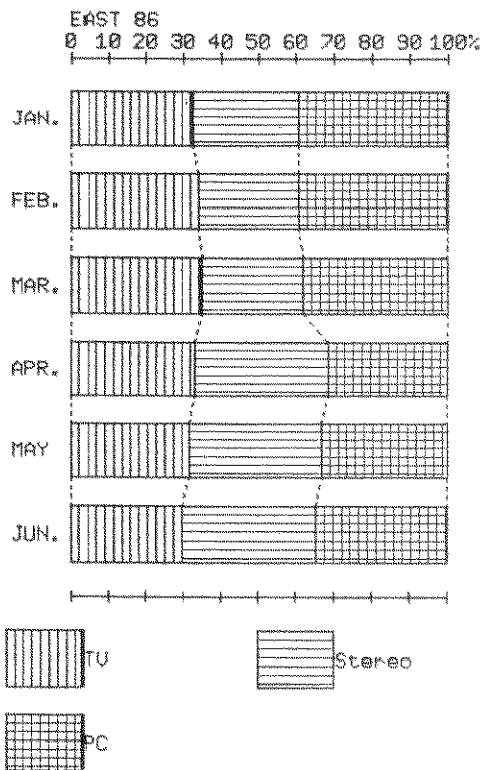




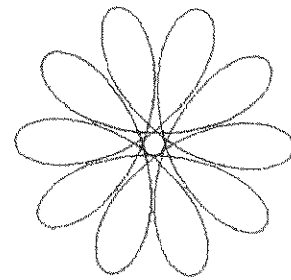
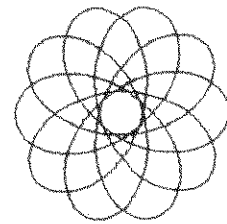
# COLOR PRINT

## ■ Comparative Belt Graph (Refer to page 315)

	TU	Stereo	PC
JAN.	30	26	36
FEB.	28	22	32
MAR.	26	20	28
APR.	34	36	32
MAY	29	32	30
JUN.	25	30	29



## ■ Round Graphic (Refer to page 330)



- 55% reduced scale from actual size

**NOTE:** These color printout examples may differ from actual colors produced by the printer.



# CONTENTS

	INTRODUCTORY NOTES	
<b>1</b>	<b>HOW TO USE THIS MANUAL</b>	<b>1</b>
<b>2</b>	<b>INTRODUCTION TO THE PC-1360</b>	<b>3</b>
	Description of Keys .....	4
	Description of Display .....	9
	ALL RESET Button .....	11
	Cell Replacement .....	12
	Installing the Cells .....	13
	Before Using the PC-1360 .....	16
	Power On .....	17
<b>3</b>	<b>USING THE PC-1360 AS A CALCULATOR</b>	<b>19</b>
	Start Up .....	19
	Shut Down .....	19
	Auto Off .....	20
	Some Helpful Hints .....	20
	Simple Calculations .....	21
	Recalling Entries .....	22
	Errors .....	26
	Serial Calculations .....	27
	Negative Numbers .....	29
	Compound Calculations and Parentheses .....	29
	Using Variables in Calculations .....	30
	Chained Calculations .....	32
	Scientific Notation .....	33
	Limits .....	34
	Last Answer Feature .....	35
	Maximum Calculation Length .....	36
	Scientific Calculations .....	37
	Priority in Manual Calculation .....	39
	Printing for Manual Calculations .....	40
<b>4</b>	<b>CONCEPTS AND TERMS OF BASIC</b>	<b>43</b>
	String Constants .....	43

Hexadecimal Numbers .....	44
Variables .....	44
Fixed Variables .....	46
Simple Variables .....	46
Array Variables .....	47
Variables in the Form of A( ) .....	51
Expressions .....	53
Numeric Operators .....	54
String Expressions .....	54
Relational Expressions .....	55
Logical Expressions .....	56
Parentheses and Operator Precedence .....	58
RUN Mode .....	59
Functions .....	59

**5 PROGRAMMING THE PC-1360 61**

Programs .....	61
BASIC Statements .....	61
Line Numbers .....	61
BASIC Verbs .....	62
BASIC Commands .....	62
Modes .....	63
Beginning to Program on the PC-1360 .....	63
Example 1—Entering and Running a Program .....	64
Example 2—Editing a Program .....	64
Example 3—Using Variables in Programming .....	67
Example 4—More Complex Programming .....	70
Storing Programs in the PC-1360's Memory .....	71
Screen Graphic Functions .....	72

**6 SHORTCUTS 79**

The DEF Key and Labelled Programs .....	79
RESERVE Mode .....	80
Template .....	82

**7 OPTIONAL PERIPHERALS 83**

CE-126P Thermal Printer/Cassette Interface .....	83
CE-140P Color Dot Printer .....	83

CE-515P/516P Color Plotter/Printers .....	84
CE-130T Level Converter .....	84
CE-124 Cassette Interface .....	85
Using the CE-126P Printer/Cassette Interface .....	85
Using the Printer .....	85
Using the Cassette Interface .....	87
Tape Notes .....	90
Using Color Printers .....	91
Serial I/O Function .....	92
<b>8 USING THE RAM CARDS</b> .....	<b>97</b>
Mounting the RAM Card .....	97
Removing the RAM Card .....	98
Using the RAM Card .....	99
Precautions When Using the RAM Card .....	100
Copying RAM Cards .....	101
Using RAM Cards from Other Computers .....	102
<b>9 BASIC REFERENCE</b> .....	<b>103</b>
Commands .....	103
Verbs .....	103
Functions .....	103
Screen Graphics Related .....	104
Plotter/Printer Graphics Related .....	104
Cassette Tape Related .....	104
Serial I/O Related .....	104
Machine Language Related .....	105
<b>10 TROUBLESHOOTING</b> .....	<b>279</b>
Machine Operation .....	279
BASIC Debugging .....	280
<b>11 MAINTENANCE OF THE PC-1360</b> .....	<b>285</b>
<b>APPENDICES</b> .....	<b>286</b>
Appendix A: Error Messages .....	286
Appendix B: Character Code Chart .....	289
Appendix C: Formatting Output .....	392

Appendix D: Expression Evaluation and Operator Priority .....	296
Appendix E: Key Functions in BASIC .....	299
Appendix F: Signals Used in the Serial I/O Terminal .....	304
Appendix G: Specifications .....	306
Appendix H: Using Programs Developed for other SHARP models on the PC-1360 .....	308

<b>PROGRAM EXAMPLES</b>	<b>311</b>
-------------------------	------------

<b>INDEX</b>	<b>339</b>
--------------	------------



# INTRODUCTORY NOTES

Welcome to the world of SHARP owners!

Few industries in the world today can match the rapid growth and technological advances being made in the field of personal computing. Computers which just a short time ago would have filled a huge room, required a Ph.D. to program, and cost thousands of dollars, now fit in the palm of your hand, are easily programmed, and cost so little that they are within the reach of nearly everyone.

Your new SHARP PC-1360 was designed to bring you all of the latest state of the art features of this computing revolution and it incorporates many advanced capabilities:

- MEMORY SAFE GUARD—the PC-1360 remembers stored programs and variables even when you turn it off.
- Battery powered operation for true portability.
- AUTO POWER OFF function which conserves the batteries by turning the power off if no activity takes place within a specified time limit.
- An expanded version of BASIC which provides formatted output, two-dimensional arrays, variable length strings, program chaining and many other advanced features.
- An optional printer/cassette interface (Model CE-126P). With the printer, you can have “hard-copies” of programs and data. The cassette interface lets you connect a cassette recorder to store programs and data on tape.
- A serial I/O interface that allows direct computer-to-computer communication and also attachment of the more versatile CE-140P color printer and CE-515P color plotter.
- Memory flexibility using the two RAM card slots.

Congratulations on entering an exciting and enjoyable new world. We are sure that you will find this purchase one of the wisest you have ever made. The SHARP PC-1360 is a powerful tool, designed to meet your specific mathematical, scientific, engineering, business and personal computing needs. With the SHARP PC-1360 you can begin NOW providing the solutions you'll need tomorrow!



# 1 How to Use This Manual

This manual is designed to introduce you to the capabilities and features of your PC-1360 and to serve as a valuable reference tool. Whether you are a “first time user” or an “old hand” with computers, you should acquaint yourself with the PC-1360 by reading and working through Chapters 2 through 6.

- Chapter 2 describes the physical features of the PC-1360.
- Chapter 3 demonstrates the use of the PC-1360 as a scientific calculator.
- Chapter 4 defines some terms and concepts which are essential for BASIC programming, and tells you about the special considerations of these concepts on the PC-1360.
- Chapter 5 introduces you to BASIC programming on the PC-1360, showing you how to enter, correct, and run programs.
- Chapter 6 discusses some shortcuts that make using your new computer easier and more enjoyable.

Experienced BASIC programmers may then read through Chapter 9 to learn the specific features of BASIC as implemented on the PC-1360. Since every dialect of BASIC is somewhat different, read through this material at least once before starting serious programming.

Chapter 9 is a reference section covering all the verbs, commands, and functions of BASIC.

If you have never programmed in BASIC before, we suggest that you buy a separate book on beginning BASIC programming or attend a BASIC class, before trying to work through these chapters. This manual is not intended to teach you how to program.

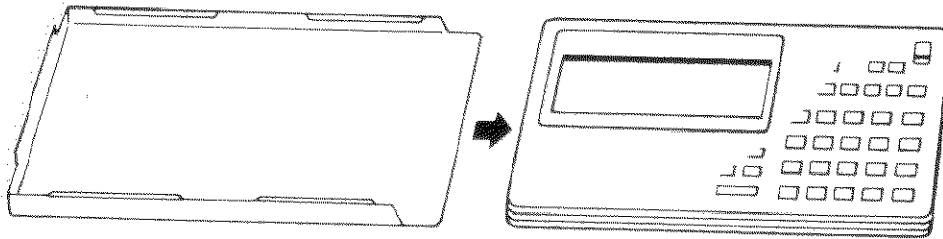
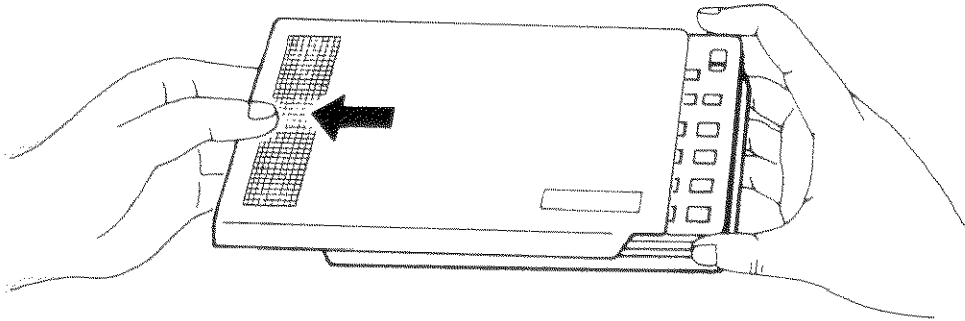
The remainder of the manual consists of:

- Chapter 7—Basic information on the use of optional peripheral units.
- Chapter 8—Description on the use of the RAM cards.
- Chapter 10—A troubleshooting guide to help you solve some operating and programming problems.
- Chapter 11—The care and maintenance of your new computer.

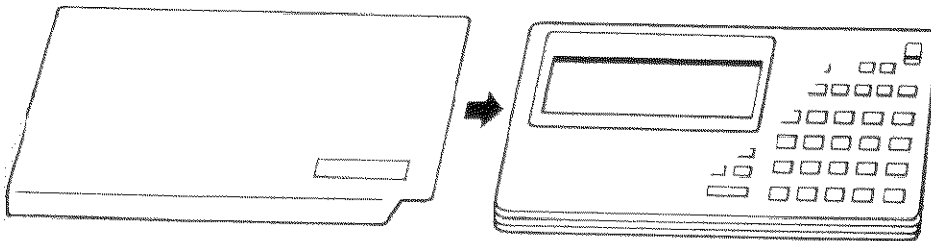
Detailed Appendices, at the end of the manual, provide you with useful charts, and special discussions concerning the use and operation of the PC-1360.

Removing and replacing the hard cover.

When using the computer slide off the cover, invert it and slide in back on again.



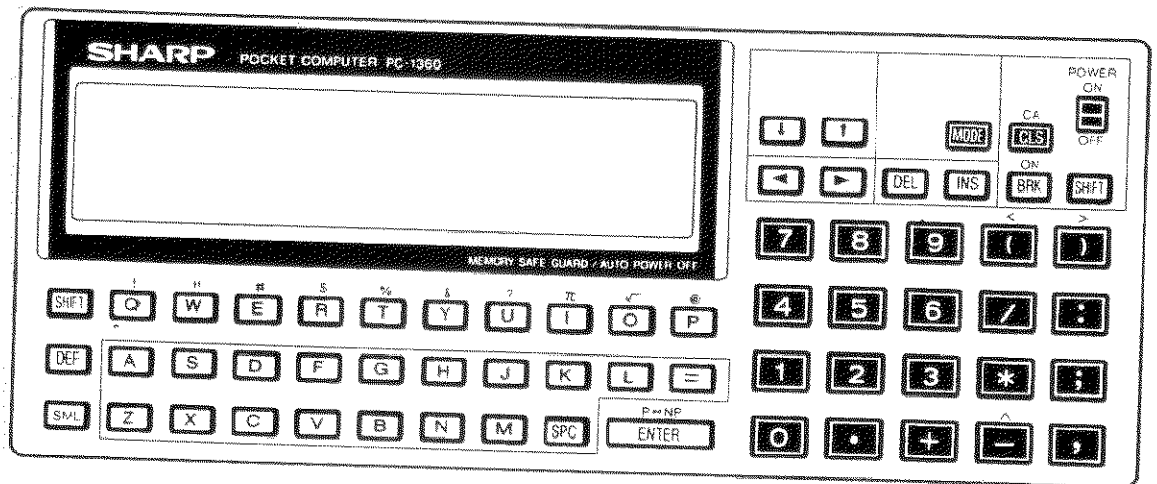
Use the cover to protect the computer when not in use.



## 2 Introduction to the PC-1360

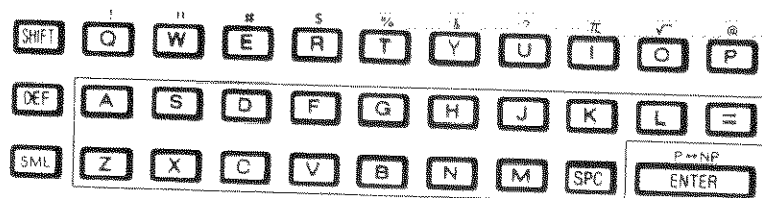
The SHARP PC-1360 system consists of:

- 62-character keyboard.
- 24-digit 4 line display or 150×32 dots graphic display.
- 8-bit CMOS processor.
- 136 KB ROM.
- Extendable RAM card RAM memory (8KB standard)
- Printer interface and serial I/O interface ports



To familiarize you with the placement and functions of parts of the PC-1360 keyboard, we will now study each section of the keyboard. For now just locate the keys and read the description of each. In Chapter 3 we will begin using your new machine.

## Description of Keys



**A ~ Z** The SHARP PC-1360 has the same 26-letter arrangement found on most typewriters. The letters are normally upper case (the opposite of most typewriters), but this is convenient because the PC-1360 only recognizes statements and commands in upper case letters. To enter the lower case letters, set the PC-1360 to the LOWER CASE mode by pressing the **SML** key.

**=** Equals key. On the PC-1360 this key is not used to indicate the end of a calculation; in BASIC programming this symbol has a special function.

**SPC** SPaCe key. Pressing this key advances the cursor leaving a blank space. Pressing **SPC** while the cursor is positioned over a character, erases that character.

**ENTER** **ENTER** key. When you type this key, whatever you previously typed is "entered" into the computer's memory. This key is similar to the Carriage Return key on a typewriter. You must press **ENTER** before the PC-1360 will act on alphanumeric input from the keyboard.

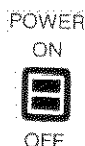
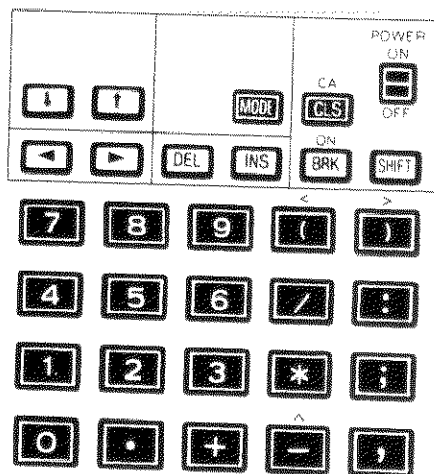
**DEF** **DEF** key. This is a special key used to execute BASIC programs.

**SHIFT** The **SHIFT** key has some important uses. First, if the **SHIFT** key is pressed before a key will a secondary function printed above it, the secondary function will be called in (e.g., pressing the **SHIFT** key before pressing the **CLS** key will call in the Clear All (CA) function). Second, if the **SHIFT** key is pressed before pressing the **MODE** key, the RESERVE mode will be selected. Ordinarily, pressing the **MODE** key will alternately select the RUN mode or the PROGRAM mode. Using the **SHIFT** key makes a third mode possible. (See **MODE** for more detail.) Third, the string stored in the RESERVE mode can be recalled by pressing **SHIFT** and the key that the string is stored under.

**SML**

The **SML** key is used when lower case letters are desired. Pressed once, this key will change the PC-1360 to the LOWER CASE mode ("SML" is displayed). Pressed again, it will return the PC-1360 to the UPPER CASE mode ("SML" disappears).

! " # These symbols are found above the top row of alphabet keys.  
 \$ % & Pressing **SHIFT** and then the alphabet key under the character  
 ? π √ desired displays these symbols.  
 @



Use this power slide switch to turn the PC-1360 ON and OFF.

**SHIFT** Shift key. See page 4.

**CA  
CLS** The red Clear **CLS** key allows you to clear the display screen. Pressing **SHIFT** before **CLS** will activate the Clear All secondary function. This will reset the computer, clearing the display and the memory stack.

**NOTE:**

The Clear All function will not erase all programs, reserve memories, and all variables. To erase all programs, type in **NEW** and press **ENTER** while in the PROGRAM mode. To erase all reserve memories, type in **NEW** and press **ENTER** while in the RESERVE mode. To clear all variables by setting them to zero (or null), type in **CLEAR** and press **ENTER**.

**MODE** When you turn the PC-1360 on, check the display for the operational mode it is in (RUN, PROGRAM, or RESERVE). Press **MODE** to change the selection from RUN to PROGRAM or from PROGRAM to RUN. Press **SHIFT** and then press **MODE** to select the RESERVE mode. To leave the RESERVE mode, press **MODE**. All manual calculations (using the computer as a calculator) are done in the RUN mode, and all programs are run in the RUN mode. All programs are written and edited in the PROGRAM mode. The RESERVE mode is used for storing frequently used functions on single keys and for storing menus to help identify these functions quickly.



**ON  
BRK**

BREAK key. The **BRK** key temporarily interrupts a program which is being executed. Pressing this key after an AUTO OFF turns the computer back on.

**INS**

Insert key. Pressing this key makes a space in front of the character indicated by the cursor. You can then INSERT a single character by pressing any key.

**DEL**

Delete key. Pressing this key will DELETE the character indicated by the cursor.



Down Arrow key. Press this key to display the next line.



Up Arrow key. Press this key to display the previous line.



Backspace key. This key allows you to move the cursor to the left without erasing previously typed characters.



Forward key. This key allows you to move the cursor to the right without erasing previously typed characters.

**0****9**

Number keys. The layout of these keys is similar to that found on a standard calculator.

**/**

Division key. Press this key to include the division operator in calculations.

**\***

Multiplication key. Press this key to include the multiplication operator in calculations.

**^****-**

Subtraction key. Press this key to include the subtraction operator in calculations.

Pressing **SHIFT** and then this key will display the "power" symbol, indicating that a number is to be raised to a specific power.

**+**

Addition key. Press this key to include the addition operator in calculations.

<



Left parenthesis key.

Pressing **SHIFT** and then this key displays the “less than” character.

>



Right parenthesis key.

Pressing **SHIFT** and then this key displays the “greater than” character.



Colon key.

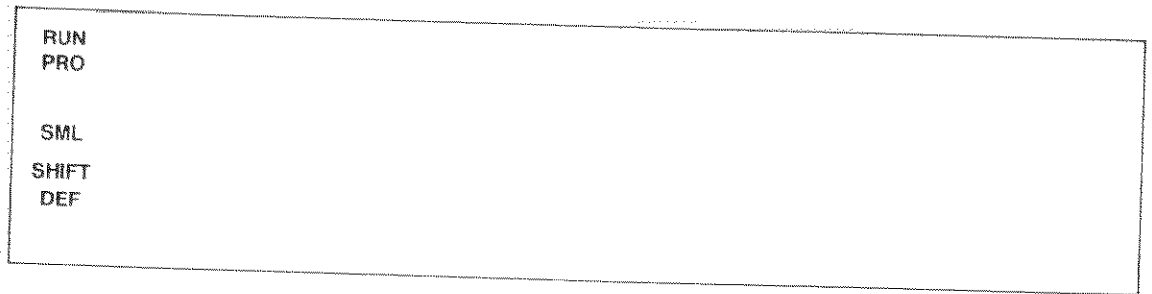


Semicolon key.



Comma key.

# Description of Display



The PC-1360 has a  $150 \times 32$  programmable, dot-matrix liquid crystal display. Character display consists of 4 lines of 24 characters each. Each character occupies a  $5 \times 7$  dot matrix.

For graphic purposes, the entire display may be utilized as a  $150 \times 32$  dot matrix. Individual dots within any of 150 columns may be energized to create graphics, figures, or special symbols.

The display consists of:



The prompt. This symbol appears when the computer is awaiting input. As you type, the prompt disappears and is replaced by the cursor.



The cursor. This symbol (the underline) tells you the location of the next character to be typed in. As you begin typing, the cursor replaces the prompt. The cursor is also used to position the computer over certain characters when using the INSert and DElete functions.

**RUN** RUN indicator. This indicator tells you the operational mode of the PC-1360 is the RUN mode.

**PRO** PROgram indicator. This indicator tells you the operational mode of the PC-1360 is the programming mode.

**Note:**

If neither RUN nor PRO indicator can be found on the display, the PC-1360 is in the reserve (RSV) mode.

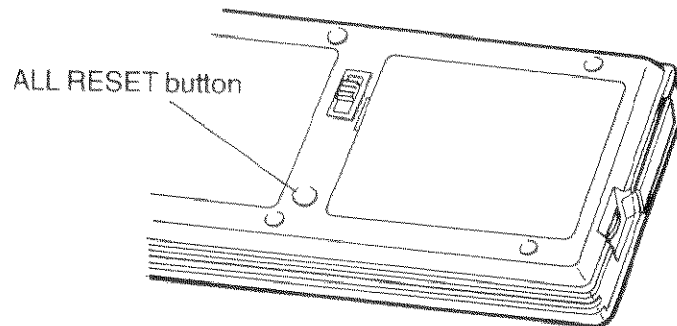
- SML** Lower case mode indicator. "SML" is displayed when **SML** is pressed. When this indicator is shown, the alphabet keys are entered as lower case characters. When **SML** is pressed while the "SML" indicator is shown, the indicator disappears and the computer returns to the upper case character mode.
- SHIFT** Shift Key Indicator. This indicator displays when the **SHIFT** key has been pressed. Remember, the **SHIFT** key must be released before pressing any other key.
- DEF** Definable Mode Indicator. This symbol lights up when you press the **DEF** key.

#### For Displays Exceeding 4 Lines

The display unit of the PC-1360 consists of 4 lines (24 characters per line). Key inputs or calculated results are displayed from the top line of the display. If the characters to be displayed exceed 4 lines, the displayed contents will be moved up by 1 line (the first displayed line will move off the top of the screen and disappear).

## All RESET Button

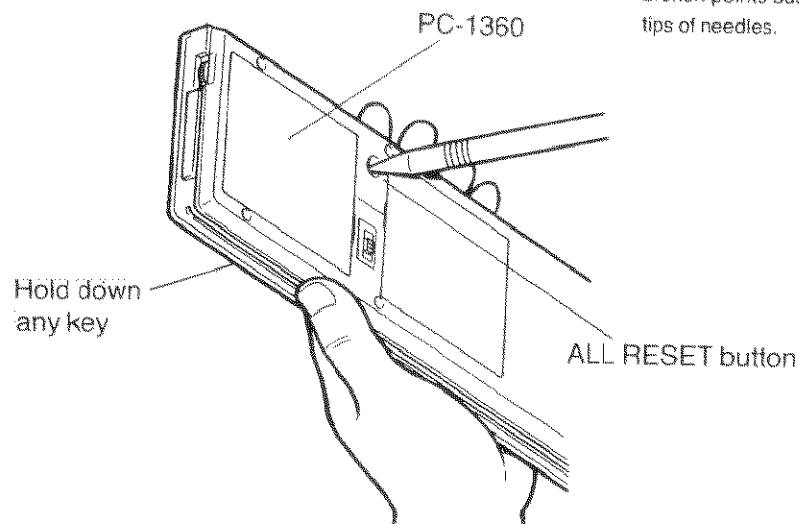
**ALL RESET:** Reset button. This button is used to reset the computer when Clear (**CLS**) or CA is not sufficient to correct the problem. The computer should be switched ON for reset operations.



### NOTE

To reset the PC-1360 hold down any key on the keyboard and simultaneously press the ALL RESET button on the back. This preserves all programs, variables, and reserve memory in RAM card memory.

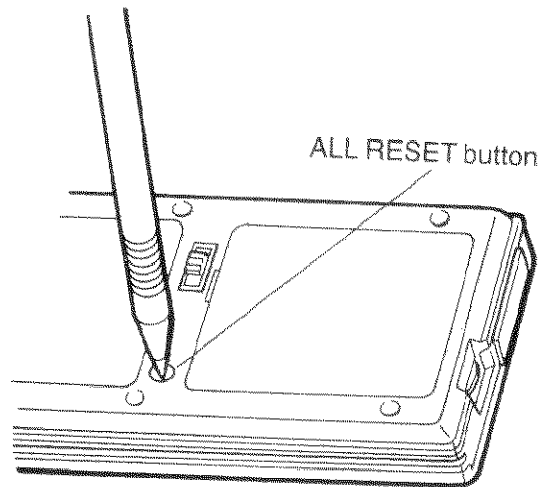
Press the ALL RESET button with any pointed object such as a ballpoint pen. Do not use easily broken points such as mechanical pencils or the tips of needles.



When you need to press the ALL RESET button, make sure that you hold it pressed for at least 3 seconds.

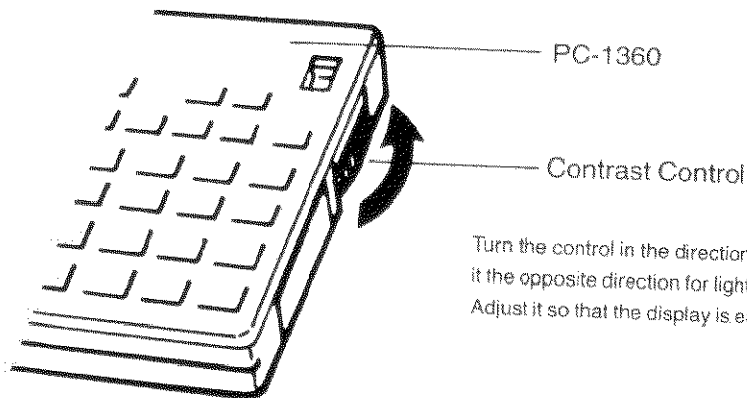
When pressing the ALL RESET button, in certain instances, the display may disappear, or a garbled display may appear. If either of these cases occurs, press the ALL RESET button again.

If the computer still does not work properly after doing this, try pressing the ALL RESET button alone. Note that this will clear all programs, variables and reserve memory in RAM card memory.



If the unit still does not operate normally, remove the lithium cells. After waiting about 10 seconds, insert the cells and press ALL RESET

### CONTRAST CONTROL



Turn the control in the direction of the arrow for darker display, and turn it the opposite direction for lighter display. Adjust it so that the display is easy to see.

### Cell Replacement

The PC-1360 operates on lithium cells alone. When connected to the optional CE-126P, the PC-1360 can also be supplied from the CE-126P when it has enough power and the lithium cell power decreases. This minimizes power consumption of the lithium cell.

When replacing the cells:

- Always replace both cells, do not mix old with new.
- Use Lithium type CR-2032 only.

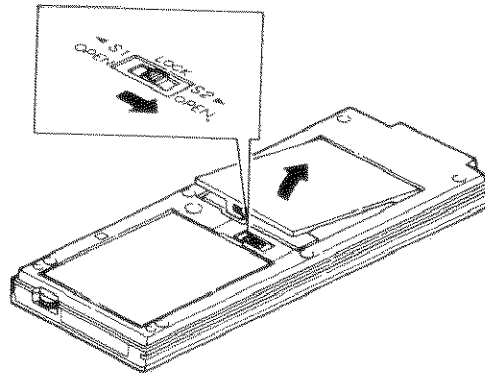
## Installing the Cells

If the display is dim, even after trying to adjust it by the contrast control, the cells need replacing.

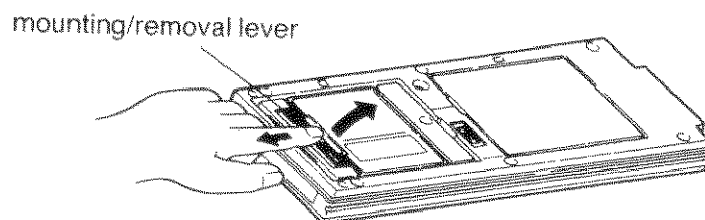
Never leave dead cells in the computer as leakage may cause damage.

**WARNING:** Keep cells away from children.

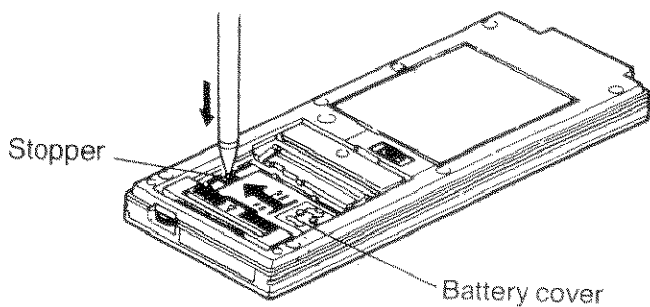
1. Turn the power switch off.
2. Turn the computer over and slide open the lock in the slot 2 direction. Take off the slot 2 cover. If there is no card in slot 2, replace the cover and lock it. If there is a RAM card, remove it in the manner described in step 4.



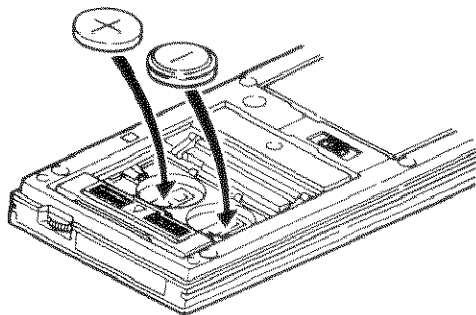
3. Slide the lock in the opposite direction and remove the cover of slot 1.
4. To remove the inserted card, push in the direction indicated by the arrow.



5. Gently press down on the stopper and slide the battery cover in the indicated direction to remove it.



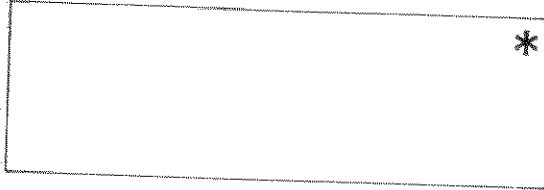
6. Take out the dead cells and replace with two fresh lithium cells. Make sure the new cells are first cleaned with a dry cloth and are inserted correctly according to their polarity.



7. Replace the battery cover in the same way that it was removed in step 5.
8. Set the lock to the unlocked side for slot 1. Do not insert the RAM cards yet. Put back the slot 1 cover and lock it in place. The RAM cards should not be inserted at this stage. If a RAM card is inserted now its memory contents will be destroyed by the ALL RESET in step 9. The back cover must be correctly locked for the computer to work. If you forget this, lock the back cover, switch power off and then on again.
9. ALL RESET BUTTON When you need to press the ALL RESET button, make sure that you hold it pressed for at least 3 seconds.



10. Check that the asterisk symbol appears at the top right of the screen. When pressing the ALL RESET button, in certain instances, the display may disappear, or a garbled display may appear. If either of these cases occurs, press the ALL RESET button again. If the trouble persists, remove the batteries, clean the contacts at either end, and reinsert them.



11. With the power off, insert the RAM card in slot 1.

If the PC-1360 fails to operate correctly, the cause may be one of the following:

1. The screen display contrast is bad; adjust the screen control.
2. If the screen continues to display just the asterisk at the top right corner, the RAM card is not in slot 1.
3. The batteries in the PC-1360 are running low.
4. The memory size is not correct. The RAM card is badly inserted in slot 1.

When the PC-1360 is not working well, reset the computer using the ALL RESET button. When the ALL RESET button is pressed alone without pressing any other key, the following screen appears:

```
MEM$ = "C"
RAM CARD S1 CLEAR O.K. ?
```

If the **Y** key is pressed in response, the RAM card is cleared and all program and data information is lost. When the computer is giving problems press any key except the **Y** key and at the same time press the ALL RESET button. The following screen will be displayed:

```
RUN MODE
```

```
>
```

## Before Using the PC-1360

Check that the batteries and the 8KB RAM card (CE-212M) are inserted correctly. Switch on the power switch on the top right of the computer.

The following appears on the display because the RAM card in slot 1 is new and has not been initialized:

```
MEMS = "C"
```

```
RAM CARD S1 CLEAR O.K. ?
```

Press the **Y** key to initialize the RAM card and the following appears:

```
RUN MODE
```

```
>
```

To check that the computer has been correctly initialized, type in:

```
MEM ENTER
```

If everything is OK, you will see the number of bytes of memory available displayed on the screen as follows:

```
RUN MODE
```

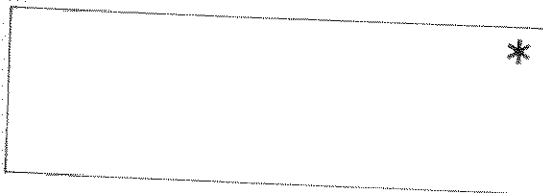
```
MEM
```

```
6558.
```

This figure will be different for a different size RAM card, but should always be 6558 for the CE-212M card supplied with the computer. Sizes for other cards are listed below:

RAM card type	Bytes available
CE-210M (2KB)	414.
CE-211M (4KB)	2462.
<b>CE-212M (8KB)</b>	<b>6558.</b>
CE-2H16M (16KB)	14750.
CE-2H32M (32KB)	31134.

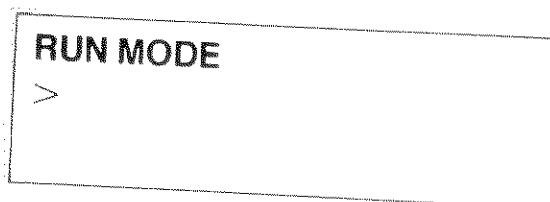
If something is wrong with the RAM card, or if there is no RAM card in slot 1, the screen will show:



In this case, press the ALL RESET button on the back of the computer and try initializing again from the start.

## Power On

After initializing the PC-1360 with its RAM card for the first time, subsequently turning on the power will display:



If a new RAM card has been inserted since the computer was last used, the initialization screen will appear again. Proceed as described in the preceding section, Before Using The PC-1360.



## 3 Using the PC-1360 as a Calculator

Now that you are familiar with the layout and components of the SHARP PC-1360, we will begin investigating the exciting capabilities of your new computer.

Because the PC-1360 allows you the full range of calculating functions, plus the increased power of BASIC programming abilities (useful in more complex calculations), it is commonly referred to as a "smart" calculator. That, of course, makes you are "smart" user!

(Before using the PC-1360, be sure that the batteries are correctly installed.)

### Start Up

To turn ON the PC-1360, slide the power switch up. For use as a calculator, the PC-1360 must be in the RUN mode. To select the mode setting, use **MODE** to display the indicator "RUN" on the left side of the display. When the machine is ON, the prompt (>) will appear on the display.

### Shut Down

To turn OFF the PC-1360, slide the power switch to the OFF position.

When you turn OFF the machine, you clear (erase) the display. However, the PC-1360 does remember all programs and reserve keys settings which were in use when the computer was turned OFF. All of these settings are still in effect when the machine is turned back ON.

When the CLOAD command is executed, stop the execution by pressing the **BRK** key and slide the power switch to the OFF position.

## Auto Off

In order to conserve on battery wear, the PC-1360 automatically powers down when no keys have been pressed for about 11 minutes. (Note: The PC-1360 will not AUTO OFF while you are executing a program.)

To restart the computer after an AUTO OFF, press the <sup>ON</sup>**BRK** key. All settings will be exactly as they were when the AUTO OFF occurred.

## Some Helpful Hints

Until you get used to your new machine, you are bound to make mistakes while entering data. Later we will discuss some simple ways to correct these mistakes. For now, if you get an Error Message, press the Clear (**CLS**) key and retype the entry. If the computer “hangs up”—you cannot get it to respond at all—press the ALL RESET button (See Chapter 2).

The PROMPT (>) tells you that the PC-1360 is awaiting input. As you enter data the prompt disappears and the CURSOR ( ) moves to the right indicating the next available location in the display.

The right  and left  arrows move the cursor.

Pressing **ENTER** informs the PC-1360 that you are finished entering data and signals the computer to perform the indicated operations. **YOU MUST PRESS ENTER AT THE END OF EACH LINE OF INPUT OR YOUR CALCULATIONS WILL NOT BE ACTED UPON BY THE COMPUTER.**

When performing numeric calculations input appears on the left of the display; the results appear on the right of the display.

When using the **SHIFT** key in conjunction with another key (to access square root for example) press **SHIFT**, release the **SHIFT**, then press the other key. **SHIFT** is active for only one key at a time.

Do not use dollar signs or commas when entering calculations into the PC-1360. These characters have special meaning in the BASIC programming language.

In this manual we use the 0 to indicate zero, so that you can distinguish between the number 0 and the letter O.

To help get you started entering data correctly, we will show each keystroke necessary to type in the example calculations. When **SHIFT** is used we will represent the desired character in the following keystroke. For example pressing **SHIFT** and **!** will produce the ! character. These keystrokes are written **SHIFT !**.

Be sure to enter CLeAr after each calculation (unless you are performing serial calculations). CLeAr erases the display and resets the error condition. It does not erase anything stored in the computer's memory.

## Simple Calculations

The PC-1360 performs calculations with ten-digit precision. Turn ON your computer and set it in the RUN mode. Now try these simple arithmetic examples.

Input

**5 0 + 5 0 ENTER**

**1 0 0 - 5 0 ENTER**

**6 0 \* 1 0 ENTER**

Display

50+50

100.

50+50

100.

100-50

50.

100-50

50.

60\*10

600.

The display unit of the PC-1360 consists of 4 lines (24 characters per line). Key inputs or calculated results are displayed from the top line of the display. If the characters to be displayed exceed 4 lines, the displayed contents will be moved up by 1 line (the first displayed line will move off the top of the screen and disappear).

Input

**3 0 0 / 5 ENTER**

Display

60 * 10	
300/5	600.
	60.

**1 0 SHIFT ^ 2 ENTER**

300/5	
10 ^ 2	60.
	100.

**2 \* SHIFT π ENTER**

10 ^ 2	
2 * π	100.
	6.283185307

**SHIFT √ 6 4 ENTER**


2 * π	
√ 64	6.283185307
	8.

**4 E 3 ENTER**

√ 64	
4E3	8.
	4000.

## Recalling Entries

Even after the PC-1360 has displayed the results of your calculation, you can edit your last entry. To edit, use the left  and right  arrows.

The left arrow  is used to position the cursor after the last character.

The right arrow  is used to position the cursor at the first character.



Remember that the left and right arrows are also used to position the cursor along a line. The right and left arrows are very helpful in editing (or modifying) entries without having to retype the entire expression.

As you read through the following examples, try them out on the machine at the same time to get familiar with using the keys.

As the head of personnel in a large marketing division, you are responsible for planning the annual sales meeting. You expect 300 people to attend the three day conference. For part of this time, the sales force will meet in small groups. You believe that groups of six would be a good size. How many groups would this be?

Input

**3 0 0 / 6 ENTER**

Display

300/6

50.



On second thought you decide that groups containing an odd number of participants might be more effective. Recall your last entry using the  arrow.

Input



Display

300/6\_

To calculate the new number of groups you must replace the six with an odd number. Five seems to make more sense than seven. Because you recalled using the  arrow, the cursor is positioned at the end of the display. Use the  to move the cursor one space to the left.

Input



Display

300/5

Notice that after you move the cursor it becomes a flashing block. Whenever you position the cursor at an existing character, it will be displayed as the flashing cursor.

Type in a 5 to replace the 6. One caution in replacing characters—once you type a new character over an existing character, the original is gone forever! You cannot recall an expression that has been typed over.

Input

**5**  
**ENTER**

Display

300/5  
60.

Sixty seems like a reasonable number of groups, so you decide that each small group will consist of five participants.


Recall is also useful to verify your last entry, especially when your results do not seem to make sense. For instance, suppose you had performed this calculation:

Input

**CLS** **3** **0** **/** **5** **ENTER**

Display

30/5  
6.



Even a tired, overworked manager like you realizes that 6 does not seem to be a reasonable result when you are dealing with hundreds of people! Recall your entry using the .

Input



Display

~~30~~/5

Because you recalled using the  the flashing cursor is now positioned over the first character in the display. To correct this entry you wish to insert an added zero. Using the , move the cursor until it is positioned over the zero. When making an INSert, you position the flashing cursor over the character before which you wish to make the insertion.

Input



Display

30/5

Use the INSert key to make space for the needed character.

Input

**INS**

Display

3  0/5

Pressing INSert moves all the characters one space to the right, and inserts a bracketed open slot. The flashing cursor is now positioned over this open space, indicating the location of the next typed input. Type in your zero. Once the entry is corrected, display your new result.

Input

**0**  
**ENTER**

Display

300/5

60.

On the other hand, suppose that you had entered this calculation:


Input

**CLS**  
**3 0 0 0 / 5 ENTER**

Display

3000/5

600.


The results seem much too large. If you only have 300 people attending the meeting, how could you have 600 "small groups"? Recall your entry using the .

Input



Display

~~3~~000/5

The flashing cursor is now positioned over the first character in the display. To correct this entry eliminate one of the zeros. Using the  move the cursor to the first zero (or any zero). When deleting a character, you position the cursor at the character to be deleted.

Input



Display

3000/5

Now use the DELete key to get rid of one of the zeros.

Input

**DEL**

Display

300/5

Pressing DELete causes all the characters to shift one space to the left. It deletes the character and the space the character occupies. The flashing cursor stays in the same position indicating the next location for input. Since you have no other changes to make, complete the calculation.

Input

**ENTER**

Display

300/5

60.

**Note:** Pressing the SPaCe key, when the cursor is positioned over a character, replaces the character leaving a blank space. DELete eliminates the character and the space it occupied.

## Errors

Recalling your last entry is essential when you get the dreaded ERROR message. Let us imagine that, unintentionally, you typed this entry into the PC-1360:


Input

**CLS**

**3 0 0 / / 5 ENTER**

Display

300//5  
ERROR 1

Naturally you are surprised when this message appears! ERROR 1 is simply the computer's way of saying, "I don't know what you want me to do here". At this time, when the  key is pressed, the flashing cursor appears at the location where the error occurred.

Input



Display

300//5

To correct this error use the DELete key.

Input

Display

**DEL** **ENTER**

300/5

60.

If, upon recalling your entry after an ERROR 1, you find that you have omitted a character, use the INSert sequence to correct it.

When using the PC-1360 as a calculator, the majority of the errors you encounter will be ERROR 1 (an error in syntax). For a complete listing of error messages, see APPENDIX A.

## Serial Calculations

The PC-1360 allows you to use the results of one calculation as part of the following calculation.

Part of your responsibility in planning this conference is to draw up a detailed budget for approval. You know that your total budget is \$150.00 for each attendant. Figure your total budget:

Input

Display

**CLS**  
**3 0 0 \* 1 5 0 ENTER**

300 \* 150

45000.

Of this amount you plan to use 15% for the final night's awards presentation. When performing serial calculations it is not necessary to retype your previous results, but DO NOT CLear between entries. What is the awards budget?

Input

Display

**\* . 1 5**

45000.\*.15\_

Notice that as you type in the second calculation (\*.15), the computer automatically displays the result of your first calculation at the left of the screen and includes it in the new calculation. In serial calculations the entry must begin with an operator. As always, you end the entry with **ENTER**.

**Note:** The  $\frac{\%}{\text{T}}$  key can not be used in the calculation. The  $\frac{\%}{\text{T}}$  key should be used as a character only.

Example: 45000 \* 15 **SHIFT** **%** → ERROR 1

Input

**ENTER**

Display

6750.

Continue allocating your budget. The hotel will cater your dinner for \$4000:

Input

**- 4 0 0 0**

**ENTER**

Display

6750.-4000\_

2750.

Decorations will be \$1225:

Input

**- 1 2 2 5 ENTER**

Display

1525.

Finally, you must allocate \$2200 for the speaker and entertainment:

Input

**- 2 2 0 0 ENTER**

Display

-675.

Obviously, you will have to change either your plans or your allocation of resources!

## Negative Numbers

Since you want the awards dinner to be really special, you decide to stay with the planned agenda and spend the additional money. However, you wonder what percentage of the total budget will be used up by this item. First, change the sign of the remaining sum:

Input

**\* - 1**

**ENTER**

Display

-675.\*-1\_

675.

Now you add this result to your original presentation budget:

Input

**+ 6 7 5 0 ENTER**

Display

7425.

Dividing by 45000 gives you the percentage of the total budget this new figure represents:

Input

**/ 4 5 0 0 0 ENTER**

Display

0.165

Fine, you decide to allocate 16.5% to the awards presentation.

## Compound Calculations and Parentheses

In performing the above calculations, you could have combined several of these operations into one step. For instance, you might have typed both these operations on one line:

$$675 + 6750/45000$$

Compound calculations, however, must be entered very carefully:

675 + 6750/45000 might be interpreted as

$$\frac{675 + 6750}{45000}$$

or

$$675 + \frac{6750}{45000}$$

When performing compound calculations, the PC-1360 has specific rules of expression evaluation and operator priority (see APPENDIX D). Be sure you get the calculation you want by using parentheses to clarify your expressions:

$$(675 + 6750)/45000$$

or

$$675 + (6750/45000)$$

To illustrate the difference that the placement of parentheses can make, try these two examples:

Input

( 6 7 5 + 6  
7 5 0 ) / 4  
5 0 0 0 ENTER

Display

0.165

6 7 5 + ( 6  
7 5 0 / 4 5 0 0  
0 ) ENTER

675.15

## Using Variables in Calculations

The PC-1360 can store up to 26 simple numeric variables under the alphabetic characters A to Z. If you are unfamiliar with the concept of variables, they are more fully explained in Chapter 4. You designate variables with an Assignment Statement:

A = 5

B = - 2



You can also assign the value of one variable (right) to another variable (left):

$$C = A + 3$$

$$D = E$$

A variable may be used in place of a number in any calculation.

Now that you have planned your awards dinner, you need to complete arrangements for your conference. You wish to allocate the rest of your budget by percentages also. First you must find out how much money is still available. Assign a variable (R) to be the amount left from the total:

Input

**R = 4 5 0 0 0  
- 7 4 2 5  
ENTER**

Display

R=45000-7425  
37575.

As you press **ENTER** the PC-1360 performs the calculation and displays the new value of R. You can display the current value of any variable by entering the alphabetic character it is stored under:

Input

**R ENTER**

Display

37575.

You can then perform calculations using your variable. The value of (R) will not change until you assign it a new value.

You wish to allocate 60% of the remaining money to room rental:

Input

**R \* . 6 0  
ENTER**

Display

R\*.60  
22545.

Similarly, you want to allocate 25% of your remaining budget to conduct management training seminars:

Input

**R \* . 2 5 ENTER**

Display

9393.75

Variables will retain their assigned values even if the machine is turned OFF or undergoes an AUTO OFF. Variables are lost only when:

- You assign a new value to the same variable.
- You type in CLEAR **ENTER** (not the Clear key).
- You clear the machine using the ALL RESET button.
- The batteries are changed.

There are certain limitations on the assignment of variables, and certain programming procedures which cause them to be changed. See Chapter 4 for a discussion of assignment. See Chapter 5 for a discussion of the use of variables in programming.

## Chained Calculations

In addition to combining several operators in one calculation, the PC-1360 also allows you to perform several calculations one after the other—without having to press **ENTER** before moving on. You must separate the equations with commas. Only the result of the final calculation is displayed. (Remember too, that the maximum line length accepted by the computer is 80 characters including **ENTER**.)

You wonder how much money would have been available for rooms if you had kept to your original allocation of 15% for the awards dinner:

Input

**R = . 8 5 \* 4 5  
0 0 0 , R \*  
. 6 0**

Display

R=.85\*45000, R\*.60

Although the computer performs all the calculations in the chain, it displays only the final result:

Input

**ENTER**

Display

R=.85 \* 45000, R\*.60

22950.

To find the value of R used in this calculation, enter R:

Input

**R ENTER**

Display

38250.

## Scientific Notation

People who need to deal with very large and very small numbers often use a special format called exponential or scientific notation. In scientific notation a number is broken down into two parts.

The first part consists of a regular decimal number between 1 and 10. The second part represents how large or small the number is in powers of 10.

As you know, the first number to the left of the decimal point in a regular decimal number shows the number of 1's, the second shows the number of 10's, the third the number of 100's, and the fourth the number of 1000's. These are simply increasing powers of 10:

$$10^0 = 1, 10^1 = 10, 10^2 = 100, 10^3 = 1000, \text{ etc.}$$

Scientific notation breaks down a decimal number into two parts: one shows what the numbers are, the second shows how far a number is to the left, or right, of the decimal point. For example:

1234 becomes 1.234 times  $10^3$  (3 places to the right)

654321 becomes 6.54321 times  $10^5$  (5 places to the right)

.000125 becomes 1.25 times  $10^{-4}$  (4 places to the left)



## Last Answer Feature

In a simple serial calculation, the result of the previous calculation can only be used in the present calculation as the first number.

For example:

Input	Display
3 <b>+</b> 4 <b>ENTER</b>	3 + 4 7.
<b>*</b> 5	7.*5_
<b>ENTER</b>	35.

However, the PC-1360 has a feature that lets you recall the result of the previous calculation and use it at any location in the current calculation. This is called the last answer feature and the previous answer can be recalled any number of times by pressing the **↓**, or **↑** key. If you entered the last example, press **CLS** then either **↓** or **↑** and you will see "35" displayed.

Let's look at an example where a previous result is used twice in the current calculation.

Note that in this example, the last answer changes and is updated with the current answer each time **ENTER** is pressed.

(Example) Use the result (6.25) of the operation,  $50 \div 8$ , to compute  $12 \times 5 \div 6.25 + 24 \times 3 \div 6.25 =$  :

Input	Display
50 <b>/</b> 8 <b>ENTER</b>	50/8 6.25 Last answer $\longrightarrow$ ↑
12 <b>*</b> 5 <b>/</b> <b>↑</b> (or <b>↓</b> )	12*5/6.25_ ↑ Last answer recalled

**+** 24 **\*** 3 **/** **↑** (or **↓**)

12\*5/6.25+24\*3/6.25\_

Last answer recalled

**ENTER**

21.12

**CLS** **↓**

21.12\_

The last answer is replaced with the result of the previous calculation by performing a manual calculation with the **ENTER** key.

The last answer is not cleared by the **CLS** or key operation.

The last answer cannot be recalled when the computer is not in the RUN mode, program execution is temporarily halted, or the Trace mode is selected.

## Maximum Calculation Length

The length of the calculation that can be input is limited to 79 key strokes before the **ENTER** key is pressed. If you try to exceed this limit, the cursor will start flashing to show that further input is invalid. If this happens, break down the calculation into two or more steps.

## Scientific Calculations

The PC-1360 is equipped with the basic functions shown below. Note that the notation of the functions in BASIC may differ from conventional mathematical notations.

Function	Conventional notation	Key operation	Remarks
Trigonometric functions	sin cos tan	SIN COS TAN	
Inverse trigonometric functions	$\sin^{-1}$ $\cos^{-1}$ $\tan^{-1}$	ASN ACS ATN	
Common logarithm	log	LOG	$\log_{10} x$ (logarithm based on 10)
Natural logarithm	ln	LN	$\log_e x$ (logarithm based on e)
Exponential function	$e^x$	EXP	$e \approx 2.718281828$
Exponential		^	$A^B$ for $A \wedge B$
Square root	$\sqrt{\quad}$	$\sqrt{\quad}$ or SQR	
Degree (decimal) $\rightarrow$ degrees (degrees, minutes, seconds) conversion		DMS	Angle conversion (Do not leave out the 0 as in DEG.5 instead of DEG 0.5)
Degrees (degrees, minutes, seconds) $\rightarrow$ degrees (decimal) conversion		DEG	
Integer		INT	In INT (x), obtains the largest integer less than or equal to x.
Absolute	X	ABS	In ABS (x), obtains the absolute value of x.
Signum		SGN	Results in 1 when $x > 0$ , -1 when $x < 0$ , and 0 when $x = 0$ for SGN(x).
Pi	$\pi$	$\pi$ or PI	$\pi \approx 3.141592654$
Hexadecimal $\rightarrow$ decimal notation		&	Converts x to a number in base 10 for &x.

Angular unit	Command	Description
Degree	DEGREE	Represents a right angle as $90[^\circ]$ .
Radian	RADIAN	Represents a right angle as $\pi/2[\text{rad}]$ .
Grade	GRAD	Represents a right angle as $100[\text{g}]$ .

These instructions are used to specify angular units in program. For practice, use these instructions to specify angular units in the following calculation examples:

(Example)  $\sin 30 =$

(Operation) DEGREE **ENTER** (Specifies "degree" for angular unit.)

SIN 30 **ENTER** | 0.5 |

(Example)  $\tan \frac{\pi}{4} =$

(Operation) RADIAN **ENTER** (Specifies "radian" for angular unit.)

TAN (PI/4) **ENTER** | 1. |

(Example)  $\cos^{-1}(-0.5) =$

(Operation) DEGREE **ENTER** (Specifies "degree" for angular unit.)

ACS -0.5 **ENTER** | 120. |

(Example)  $\log 5 + \ln 5 =$

(Operation) LOG5 + LN5 **ENTER** | 2.308407917 |

(Example)  $e^{2+3} =$

(Operation) EXP (2 + 3) **ENTER** | 148.4131591 |

(Example)  $\sqrt{4^3 + 6^4} =$

(Operation)  $\sqrt{(4^3 + 6^4)}$  **ENTER** | 36.87817783 |

(Example) Convert 30 deg. 30 mm. in sexagenary notation into decimal notation.

(Operation) DEG 30.30 **ENTER** | 30.5 |  
(30.5 degree)



(Example) Convert 30.755 deg. in decimal notation into sexagenary notation

(Operation) DMS 30.755 **ENTER** | **30.4518** |  
 (30 deg. 45 min. 18 sec.)

(Example) Convert CF8 to its decimal equivalent.

(Operation) &CF8 **ENTER** | **3320.** |

## Priority in Manual Calculation

You can type in formulas in the exact order in which they are written, including parentheses or functions. The order of priority in calculation and treatment of intermediate results will be taken care of by the computer itself.

The internal order of priority in manual calculation is as follows:

1. Recalling variables or  $\pi$ ,
2. Function (sin, cos, etc.)
3. Power ( ^ )
4. Sign (+, -)
5. Multiplication or division (\*, /)
6. Addition or subtraction (+, -)
7. Comparison of magnitude (>, >=, <, <=, <>, =)
8. Logical AND, OR

**Note:** 1. If parentheses are used in a formula, the operation given within the parentheses has the highest priority.

2. Composite functions are operated from right to left (sin cos<sup>-1</sup> 0.6).

3. Chained power (3<sup>42</sup> or 3 ^ 4 ^ 2) is operated from right to left.

4. For the above items 3) and 4), the last entry has a higher priority.

(e.g.)  $-2 \wedge 4 \rightarrow -(2^4)$

$3 \wedge -2 \rightarrow 3^{-2}$

## Printing for Manual Calculations

The calculation steps and results can be printed when an optional printer (CE-126P or CE-140P) is connected and turned on, and **SHIFT ENTER** (P ↔ NP) are pressed. (Print Mode)

If a printout is not desired, either turn off the CE-126P or CE-140P, or press **SHIFT** and **ENTER** (P ↔ NP) again. (Non-Print Mode)

## Calculation Error

The following types of errors occur in ordinary calculators, pocket computers, and personal computers.

### Errors due to least significant digit processing

Usually, the maximum number of digits that can be calculated in a computer is fixed. For example,  $4 \div 3$  results in 1.3333333333. In a computer with a maximum of 8 digits, the 8 digits are significant digits; other least significant digits are either truncated or rounded.

(example) computer with 10 significant digits

$$4/3 \text{ **ENTER** } \rightarrow \overbrace{1.3333333333}^{10 \text{ significant digits}} \dots$$

Truncated, rounded

Therefore, the calculated result differs from the true value by the amount truncated or rounded. (This difference is the error.)

In the unit, a 12-digit calculated result is obtained. This result is rounded and specially processed to minimize error in the displayed value.

(Example)  $4 \div 3 \times 3$

$4/3 * 3$	<b>ENTER</b>	→ 4.	}	Calculated in succession
$4/3$	<b>ENTER</b>	→ 1.3333333333		
$* 3$	<b>ENTER</b>	→ 3.9999999999		

Calculated independently

When calculated in succession, the result of  $4 \div 3$  is obtained internally in 12 digits and is used for calculation and then rounded.

When calculated independently, the displayed value (10 digits) is used for the calculation.

### Errors due to function determining algorithms

The computer uses a variety of algorithms to calculate the values of functions, such as power and trigonometrical functions. When calculations use such functions, an additional source of error is introduced. This error factor increases the more that functions are used in the calculation. The actual error for each function varies according to the values used and is worse around singularities and inflection points (eg, when an angle approaches 90 degrees, the tangent approaches infinity.) An example of an algorithm used by the computer is shown below for powers.

(Example)  $60^6 =$

$$60 \wedge 6 \text{ ENTER} \rightarrow 4.665599999\text{E}10$$

Although  $60^6$  is  $4.6656 \times 10^{10}$ , the unit calculates it ( $y^x$ ) based on the following power expression.

$$y^x = 10^{x \times \log y}$$

In other words,  $60^6$  is calculated as  $10^{6 \times \log 60}$ .



# 4 Concepts and Terms of BASIC

In this chapter we will examine some concepts and terms of the BASIC language.

## String Constants

In addition to numbers, there are many ways that the SHARP PC-1360 uses letters and special symbols. These letters, numbers, and special symbols are called characters.

These characters are available on the PC-1360:

```

1 2 3 4 5 6 7 8 9 0
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
! " # $ % & ( ) * + , - . / : ; < = > ? @  $\sqrt{\quad}$   $\pi$  ^

```

In BASIC, a collection of characters is called a string. In order for the PC-1360 to tell the difference between a string and other parts of a program, such as verbs or variable names, you must enclose the characters of the string in quotation marks ("").

The following are examples of string constants:

```

"HELLO"
"Goodbye"
"SHARP COMPUTER"

```

The following are not valid string constants:

```

"COMPUTER      No ending quote
"ISN"T"        Quote can't be used within a string

```

## Hexadecimal Numbers

The decimal system is only one of many different systems to represent numbers. Another which has become quite important when using computers is the hexadecimal system. The hexadecimal system is based on 16 instead of 10. To write hexadecimal numbers you use the familiar 0 ~ 9 and 6 more "digits": A, B, C, D, E, and F. These correspond to 10, 11, 12, 13, 14, and 15. When you want the PC-1360 to treat a number as hexadecimal put an ampersand '&' character in front of the numeral:

```
&A    = 10
&10   = 16
&100  = 256
&FFFF = 65535
```

## Variables

Computers are made up of many tiny memory areas called bytes. Each byte can be thought of as a single character. For instance, the word byte requires four bytes of memory because there are four characters in it. To see how many bytes are available for use, simply type in MEM **ENTER**. The number displayed is the number of bytes available for writing programs. This technique works fine for words, but is very inefficient when you try to store numbers. For this reason, numbers are stored in a coded fashion. Thanks to this coding technique, your computer can store large numbers in only eight bytes. The largest number that can be stored is + 9.999999999E + 99.

The smallest number is + 1.E - 99. This gives you quite a range to choose from. However, if the result of a calculation exceeds this range, the computer will let you know by displaying an error message on the screen. For the error message refer to Appendix A. To see it right now type in:

```
9 E 99 * 9 ENTER
```

To get the computer working properly again, just press the **CLS** key. But how do you go about storing all this information? It's really very easy. The computer likes to use names for different pieces of data. Let's store the number 556 into the computer. You may call this number by any name that you wish, but for this exercise, let's use the letter R. The statement, LET, can be used to instruct the

computer to assign a value to a variable name but only in a program statement. However, the LET command is not necessary, so we will not use it very often. Now, type in  $R = 556$  and press the **ENTER**. The computer now has the value 556 associated with the letter R. These letters that are used to store information are called variables. To see the content of the variable R, press the **CLS** key, the R key and the **ENTER** key. The computer responds by showing you the value 556 on the right of your screen. This ability can become very useful when you are writing programs and formulas.

Next, let's use the R variable in a simple formula. In this formula, the variable R stands for the radius of a circle whose area we want to find. The formula for the area of a circle is:  $A = \pi * R^2$ . Type in  $R$  **SHIFT** **^** **2** **\*** **SHIFT** **π** **ENTER**. The result is 971179.3866. This technique of using variables in equations will become more understandable as we get into writing programs.

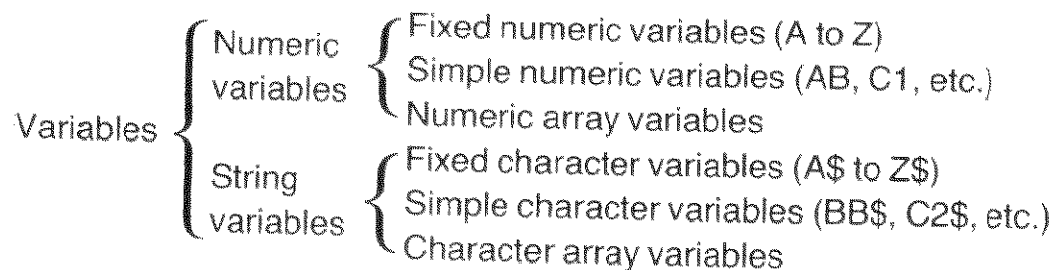
So far, we've only discussed numeric variables. What about storing alphabetic characters? Well, the idea is the same, but, so the computer will know the difference between the two kinds of variables, add a \$ to the variable name. For instance, let's store the word BYTE in the variable B\$. Notice the \$ after the B!

This tells the computer that the contents of the letter B is alphabetic, or string data.

To illustrate this, key in  $B$  **SHIFT** **\$** = **SHIFT** **"** BYTE **SHIFT** **"** **ENTER**. The value BYTE is now stored in the variable B\$. To make sure of this, press the **CLS** key and type in  $B$  **SHIFT** **\$** **ENTER**. The screen shows BYTE. This time the display is on the left side of the screen, instead of the right.

**Note:** The contents of character strings or character variables are displayed from the left edge of the first line.

Variables handled by the SHARP PC-1360 are divided into the following:



## Fixed Variables

The first section, fixed variable, is always used by the computer for storing data. It can be thought of as pre-allocated variable space. In other words, no matter how much memory your program uses up, you will always have at least 26 variables to choose from to store data in. This data can be one of two types: NUMERIC or STRING (alphabetic character). Fixed memory locations are eight bytes long and can be used for only one type of data at a time. To illustrate this, type is the following example:

```
A = 123 ENTER
A$ ENTER
```

You get the message:

```
ERROR 9
```

This means that you have put numeric data into the area of memory called A and then told the computer to show you that information again as STRING data. This confuses the computer so it says that there is an error condition. Press the **CLS** key to clear error condition. Now try the following example:

```
A$ = "ABC" ENTER
A ENTER
```

Again, the computer is confused and gives the ERROR 9 message.

## Simple Variables

Simple variable names are specified by two (or more) alphanumeric characters, such as AA or B1. Unlike fixed variables, simple variables have no dedicated storage area in the memory. The area for simple variables is automatically set aside (within the program and data area) when a simple variable is first used.

Since separate memory areas are defined for simple numeric variables and simple character variables even if they have the same name, variables such as AB and AB\$, for example, may be used at the same time.



While alphanumeric characters are usable for simple variable names (as for alphabetic characters, only upper case characters are usable), the first character of a variable name must always be an alphabetic character. If more than two characters are used to define a variable name, only the first two characters are meaningful.

- Note:**
1. The function or BASIC instruction names for the PC-1360 computer are not usable as variable names. (e.g.) PI, IF, TO, ON, SIN, etc.
  2. Each simple character variable can hold up to 16 characters or symbols.

## Array Variables

For some purposes it is useful to deal with numbers as an organized group, such as a list of scores or a tax table. In BASIC these groups are called arrays. An array can be either one-dimensional, like a list, or two-dimensional, like a table.

To define an array, the DIM (short for dimension) statement is used. Arrays must always be "declared" (defined) before they are used. (Not like the single-value variables we have been using.) The form for the numeric DIMension statement is:

DIM numeric-variable-name (size)

where:

numeric-variable-name is a variable name which conforms to the normal rules for numeric variable names previously discussed.

size is the number of storage locations and must be a number in the range 0 through 255. Note that when you specify a number for the size, you get one more location than you specified.

Examples of legal numeric DIMension statements are:

DIM X (5) DIM AA (24) DIM Q5 (0)
--

## CONCEPTS AND TERMS OF BASIC

The first statement creates an array X with 6 storage locations. The second statement creates an array AA with 25 locations. The third statement creates an array with one location and is actually rather silly since (for numbers at least), it is the same as declaring a single-value numeric variable.

It is important to know that an array-variable X and a variable X are separate and distinct to the PC-1360. The first X denotes a series of numeric storage locations, and the second a single and different location.

Now that you know how to create arrays, you might be wondering how it is that we refer to each storage location. Since the entire group has only one name, the way in which we refer to a single location (called an "element") is to follow the group name with a number in parentheses. This number is called a "subscript". Thus, for example, to store the number 8 into the fifth element of our array X (declared previously) we would write:

$$X(4) = 8$$

If the use of 4 is puzzling, remember that the numbering of elements begins at zero and continues through to the number of elements declared in the DIM statement.

The real power of arrays lies in the ability to use an expression or a variable name as a subscript.

To declare a character array a slightly different form of the DIM statement is used:

DIM character-variable-name (size) \*length

where:

character-variable-name is a variable name which conforms to the rules for normal character variables as discussed previously.

size is the number of storage locations and must be in the range 0 through 255. Note that when you specify a number, you get one more location than you specified.

\*length is optional. If used, it specifies the length of the strings that comprise the array. Length is a number in the range 1 to 80. If this clause is not used, the strings will have the default length of 16 characters.

Example of legal character array declarations are:

```
DIM X$(4)
DIM NM$(10)*10
DIM IN$(1)*80
DIM R$(0)*26
```

The first example creates an array of five strings each able to store 16 characters. The second DIM statement declares an array NM with eleven strings of 10 characters each.

Explicit definition of strings smaller than the default helps to conserve memory space. The third example declares a two element array of 80-character strings and the last example declares a single string of twenty-six characters.

Besides the simple arrays we have just studied, the PC-1360 allows "two-dimensional" arrays. By analogy, a one-dimensional array is a list of data arranged in a single column. A two-dimensional array is a table of data with rows and columns.

The two-dimensional array is declared by the statement:

DIM numeric-variable-name (rows, columns)

or

DIM character-variable-name (rows, columns)\*length

where:

rows specifies the number of rows in the array. This must be a number in the range 0 through 255. Note that when you specify the number of rows you get one more row than the specification.

columns specifies the number of columns in the array. This must be a number in the range 0 through 255. Note that when you specify the number of columns you get one more column than the specification.

The following diagram illustrates the storage locations that result from the declaration DIM T (2, 3) and the subscripts (now composed of two numbers) which pertain to each storage location:

	column 0	column 1	column 2	column 3
row 0	T (0, 0)	T (0, 1)	T (0, 2)	T (0, 3)
row 1	T (1, 0)	T (1, 1)	T (1, 2)	T (1, 3)
row 2	T (2, 0)	T (2, 1)	T (2, 2)	T (2, 3)

**Note:** Two-dimensional arrays can rapidly eat up storage space. For example, an array with 25 rows and 35 columns uses 875 storage locations!

Arrays are very powerful programming tools.

The following table shows the number of bytes used to define each variable and the number used by each program statement.

Variable	Variable name	Data	
Numeric variable	7 bytes	8 bytes	
String variable	7 bytes	Array variable	Specified number*
		Simple variable (two-character variable)	16 bytes

\* For example, if DIM Z\$(2, 3)\*10 is specified, 12 variables, each capable of storing 10 characters, are reserved. This requires 7 bytes (variable name) + 10 bytes (number of characters) × 12 = 127 bytes.

Element	Line number	Statement & function	<b>ENTER</b> , and others
Number of bytes used	3 bytes	2 bytes	1 byte

## Variables in the Form of A ( )

While a data area on the computer's memory is set aside for fixed variables, it may also be used to define subscripted variables which have the same form as array variables.

There are 26 fixed variable names available: i.e. A through Z(A\$ through Z\$). Each of these names can be subscripted with the numbers 1 through 26, such as A(1)—A(26) or A\$(1)—A\$(26). This means that variable A(1) may be used in place of variable A, A(2) in place of B, A(3) in place of C, and so forth.

A	=	A\$	=	A(1)	=	A\$(1)
B	=	B\$	=	A(2)	=	A\$(2)
C	=	C\$	=	A(3)	=	A\$(3)
D	=	D\$	=	A(4)	=	A\$(4)
E	=	E\$	=	A(5)	=	A\$(5)
F	=	F\$	=	A(6)	=	A\$(6)
G	=	G\$	=	A(7)	=	A\$(7)
H	=	H\$	=	A(8)	=	A\$(8)
I	=	I\$	=	A(9)	=	A\$(9)
J	=	J\$	=	A(10)	=	A\$(10)
K	=	K\$	=	A(11)	=	A\$(11)
L	=	L\$	=	A(12)	=	A\$(12)
M	=	M\$	=	A(13)	=	A\$(13)
N	=	N\$	=	A(14)	=	A\$(14)
O	=	O\$	=	A(15)	=	A\$(15)
P	=	P\$	=	A(16)	=	A\$(16)
Q	=	Q\$	=	A(17)	=	A\$(17)
R	=	R\$	=	A(18)	=	A\$(18)
S	=	S\$	=	A(19)	=	A\$(19)
T	=	T\$	=	A(20)	=	A\$(20)
U	=	U\$	=	A(21)	=	A\$(21)
V	=	V\$	=	A(22)	=	A\$(22)
W	=	W\$	=	A(23)	=	A\$(23)
X	=	X\$	=	A(24)	=	A\$(24)
Y	=	Y\$	=	A(25)	=	A\$(25)
Z	=	Z\$	=	A(26)	=	A\$(26)

However, if an array named A or A\$ has already been defined by the DIM statement, subscripted variables named A cannot be defined. For example, if an array A is defined by DIM A(5) the location for A(0) through A(5) are set aside in the program/data area. So if you specify variable A(2), it does not refer to the fixed variable B, but refers to the array variable A(2) defined in the program/data area. If you specify A(9), it will cause an error since A(9) is outside the range of the dimension specified by the DIM A(5) statement.

On the other hand, if subscripted variables are already defined in the form of A( ), it is not possible to define arrays A or A\$ by using the DIM statement, unless the definition for the subscripted variables is cleared with the CLEAR statement.

When using the PC-1360 with two RAM cards with a total memory capacity of more than 32KB, and MEM\$ indicates storage structure "B", the one-dimensional array A\$( ) must be declared with a DIM statement before it can be used.

If subscripts greater than 26 are used for subscripted variables A( ) when array A is not defined by a DIM statement, the corresponding locations in the program/data area are set aside for these A( ) variables. For instance, if you execute A(35) = 5, locations for variables A(27) to A(35) will be reserved in the program/data area.

While variables subscripted in excess of 26 are treated as array variables, they are subject to the following special restrictions:

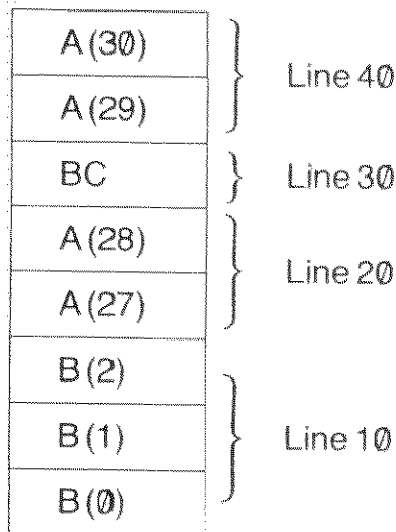
1. Locations for an array with the same name must be contiguous in the program/data area. Otherwise, an error will occur.

```

10: DIM B(2)
20: A(28)=5
30: BC=12
40: A(30)=9
    
```

In this program is executed, the array named "A" is not defined in two consecutive segments in the program data area, and an error will result at line 40.

[Program/data area]



2. Numeric array variables and character array variables with the same subscript cannot be defined at the same time. For example, A(30) and A\$(30) cannot be defined at the same time, since they use the same location in the program/data area.
3. Two dimensional arrays cannot be defined, nor is possible to specify the length of character strings to be held in character array variables. For example, the length of a character string which can be held in the character array variable A\$( ) is limited to seven characters or less.
4. Variables subscripted with zero (0) cannot be defined. If A(0) or A\$(0) is defined, an error will result.
5. When A(27), or A\$(27) and higher is first used, 7 bytes are used for the variable names and 8 bytes are used for each variable.

## Expressions

An expression is some combination of variables, constants, and operators which can be evaluated to a single value. The calculations which you entered in Chapter 3 were examples of expressions. Expressions are an intrinsic part of BASIC programs. For example, an expression might be a formula that computes an answer to some equation, a test to determine the relationship between two quantities, or a means to format a set of strings.

## Numeric Operators

The PC-1360 has five numeric operators. These are the arithmetic operators which you used when exploring the use of the PC-1360 as a calculator in Chapter 3:

- + Addition
- Subtraction
- \* Multiplication
- / Division
- ^ Power

A numeric expression is constructed in the same way that you entered compound calculator operations. Numeric expressions can contain any meaningful combination of numeric constants, numeric variables, and the numeric operators:

$(A * B) \wedge 2$   
 $A(2, 3) + A(3, 4) + 5.0 - C$   
 $(A/B) * (C + D)$

## String Expressions

String expressions are similar to numeric expressions except that there is only one string operator—concatenation (+). This is the same symbol used for plus. When used with a pair of strings, the + attaches the second string to the end of the first string and makes one longer string. You should take care in making more complex string concatenations and other string operations because the work space used by the PC-1360 for string calculations is limited to only 80 characters.

**Note:** String quantities and numeric quantities cannot be combined in the same expression unless one uses one of the functions which convert a string value into a numeric value or vice versa:

"15" + 10            is illegal  
 "15" + "10"        is "1510", not "25"



# Relational Expressions

A relational expression compares two expressions and determines whether the stated relationship is true or false. The relational operators are:

>	Greater Than
> =	Greater Than or Equal To
=	Equals
< >	Not Equal To
< =	Less Than or Equal To
<	Less Than

The following are valid relational expressions:

```
A < B
C(1, 2) > = 5
D(3) < > 8
```

If A was equal to 10, B equal to 12, C (1, 2) equal to 6, and D (3) equal to 9, all of these relational expressions would be true.

Character strings can also be compared in relational expressions. The two strings are compared character by character according to their ASCII value starting at the first character (see Appendix B for ASCII values). If one string is shorter than the other, a  $\emptyset$  or NULL will be used for any missing positions. All of the following relational expressions are true:

```
"ABCDEF" = "ABCDEF"
"ABCDEF" < > "ABCDE"
"ABCDEF" > "ABCDE"
```

Relational expressions evaluate to either true or false. The PC-1360 represents true by a 1; false if represented by a  $\emptyset$ . In any logical test an expression which evaluates to 1 or more will be regarded as true while one which evaluates to  $\emptyset$  or less will be considered false. Good programming practice, however, dictates the use of an explicit relational expression instead of relying on this coincidence.

# Logical Expressions

Logical expressions are relational expressions which use the operators AND, OR, and NOT. AND and OR are used to connect two relational expressions; the value of the combined expression is shown in the following tables:

A AND B

Value of A

		True	False
Value of B	True	True	False
	False	False	False

A OR B

Value of A

		True	False
Value of B	True	True	True
	False	True	False

(Note: Value of A and B must be 0 or 1)

Decimal numbers can be expressed in the binary notation of 16 bits as follows:

DECIMAL NOTATION      BINARY NOTATION OF 16-BITS

32767	0111111111111111
3	0000000000000011
2	0000000000000010
1	0000000000000001
0	0000000000000000
-1	1111111111111111
-2	1111111111111110
-3	1111111111111101
-32768	1000000000000000

The negative (NOT) of a binary number 0000000000000001 is as follows:

NOT            0000000000000001  
 (Negative) → 1111111111111110

Thus, 1 is inverted to 0, and 0 to 1 for each bit, which is called “to take negative (NOT)”.

Then, the following will result when 1 and NOT 1 are added together:

```

    0000000000000001 (1)
+ 1111111111111110 (NOT 1)
-----
    1111111111111111 (-1)
    
```

Thus, all bits become 1. According to the above number list, the bits become -1 in decimal notation, that is  $1 + \text{NOT } 1 = -1$ .

The relationship between numerical value X and its negative (NOT X) is:

$$X + \text{NOT } X = -1$$

This results in an equation of  $\text{NOT } X = -X - 1$

i.e.  $\text{NOT } X = -(X + 1)$

From the equation the following are found to result.

$$\text{NOT } 0 = -1$$

$$\text{NOT } -1 = 0$$

$$\text{NOT } -2 = 1$$

More than two relational expressions can be combined with these operators. You should take care to use parentheses to make the intended comparison clear.

$$(A < 9) \text{ AND } (B > 5)$$

$$(C = 5) \text{ OR } (C = 6) \text{ OR } (C = 7)$$

The PC-1360 implements logical operators as “bitwise” logical functions on 16 bit quantities. (See note on relational expressions and true and false). In normal operations this is not significant because the simple 1 and 0 (true and false) which result from a relational expression uses only a single bit. If you apply a logical operator to a value other than 0 or 1, it works on each bit independently. For example if A is 17, and B is 22, (A OR B) is 23:

## CONCEPTS AND TERMS OF BASIC

17 in binary notation is 10001

22 in binary notation is 10110

17 OR 22 is 10111 (1 if 1 in either number, otherwise 0)

10111 is 23 in decimal.

If you are a proficient programmer, there are certain applications where this type of operation can be very useful. Beginning programmers should stick to clear, simple true or false relational expressions.

## Parentheses and Operator Precedence

When evaluating complex expressions the PC-1360 follows a predefined set of priorities which determine the sequence in which operators are evaluated. This can be quite significant:

$5 + 2 * 3$  could be

$5 + 2 = 7$       or       $2 * 3 = 6$

$7 * 3 = 21$        $6 + 5 = 11$

The exact rules of "operator precedence" are given in Appendix D.

To avoid having to remember all these rules and to make your program clearer, always use parentheses to determine the sequence of evaluation. The above example is clarified by writing either:

$(5 + 2) * 3$       or       $5 + (2 * 3)$

## RUN Mode

In general, any of the expressions described before can be used in the RUN mode as well as in programming a BASIC statement. In the RUN mode an expression is computed and displayed immediately. For example:

<u>Input</u>	<u>Display</u>
(5 > 3) AND (2 < 6) <b>ENTER</b>	1.

The 1 means that the expression is true.

## Functions

Functions are special components of the BASIC language which take one value and transform it into another value. Functions act like variables whose value is determined by the value of other variables or expressions. ABS is a function which produces the absolute value of its argument:

ABS(-5)    is    5  
 ABS(6)     is    6

LOG is a function which computes the log to the base 10 of its argument.

LOG(100)    is    2  
 LOG(1000)  is    3

A function can be used any place that a variable can be used. Many functions do not require the use of parentheses:

LOG 100     is the same as    LOG(100)

You must use parentheses for functions which have more than one argument. Using parentheses always makes programs clearer.

See Chapter 9 for the functions available on the PC-1360.



# 5 Programming the PC-1360

In the previous chapter we examined some of the concepts and terms of the BASIC programming language. In this chapter you will use these elements to create programs on the PC-1360. Let us reiterate however, this is not a manual on how to program in BASIC. What this chapter will do is familiarize you with the use of BASIC on your PC-1360.

## Programs

A program consists of a set of instructions to the computer. Remember the PC-1360 is only a machine. It will perform the exact operations that you specify. You, the programmer, are responsible for issuing the correct instructions.

## BASIC Statements

The PC-1360 interprets instructions according to a predetermined format. This format is called a statement. You always enter BASIC statements in the same pattern. Statements must start with a line number:

```
10: INPUT A  
20: PRINT A * A  
30: END
```

## Line Numbers

Each line of a program must have a unique line number—any integer between 1 and 65279. Line numbers are the reference for the computer. They tell the PC-1360 the order in which to perform the program. You need not enter lines in sequential order (although if you are a beginning programmer, it is probably less confusing for you to do so). The computer always begins execution with the lowest line number and moves sequentially through the lines of a program in ascending order.

When programming it is wise to allow increments in your line numbering (10, 20, 30, ... 10, 30, 50 etc). This enables you to insert additional lines if necessary.

**CAUTION:** Do not use the same line numbers in different programs you plan to merge.

If you use the same line number, the oldest line with that number is deleted when you enter the new line.

## BASIC Verbs

All BASIC statements must contain verbs. Verbs tell the computer what action to perform. A verb is always contained within a program, and as such is not acted upon immediately.

Some statements require or allow an operand:

```
10: DATA "HELLO"  
20: READ B$  
30: PRINT B$  
40: END
```

Operands provide information to the computer telling it what data the verb will act upon. Some verbs require operands, with other verbs they are optional. Certain verbs do not allow operands. (See Chapter 9 for BASIC verbs and their use on the PC-1360.)

**Note:** Verbs, commands and functions must be typed in the upper case character mode.

## BASIC Commands

Commands are instructions to the computer which are entered outside of a program. Commands instruct the computer to perform some action with your program or to set modes which affect how you programs are executed.

Unlike verbs, commands have immediate effects—as soon as you complete entering the command (by pressing the **ENTER** key), the command will be executed. Commands are not preceded by a line number:



RUN  
NEW  
RADIAN

Some verbs can also be used as commands. (See Chapter 9 for BASIC commands and their use on the PC-1360).

## Modes

You will remember that when using the PC-1360 as a calculator, it is set in the RUN mode.

The RUN mode is also used to execute the programs you create.

The PROgram mode is used to enter and edit your programs.

The RESERVE mode enables you to designate and store predefined string variables and is used in more advanced programming (See Chapter 6).

## Beginning to Program on the PC-1360

After all your practice in using the PC-1360 as a calculator you are probably quite at home with the keyboard. From now on, when we show an entry, we will not show every keystroke. Remember to use **SHIFT** to access characters above the keys and end every line by pressing the **ENTER** key.

Now you are ready to program!

To enter program statements into the computer, the computer must first be placed in the PROGRAM mode using the **MODE** key. The display will appear as in the following illustration.

```
PRO  PROGRAM MODE
    >
```

Enter the NEW command.

InputNEW **ENTER**DisplayNEW  
>

The NEW command clears the PC-1360 memory of all existing programs and data. The prompt appears after you press **ENTER**, indicating that the computer is awaiting input.

## Example 1—Entering and Running a Program

Make sure the PC-1360 is in the PRO mode and enter the following program:

Input

10 PRINT "HELLO"

Display

10: PRINT "HELLO"

Notice that when you push **ENTER** the PC-1360 displays your input, automatically inserting a colon (:) between the line number and the verb. Verify that the statement is in the correct format.

Now change the mode to the RUN:

Input**CLS**  
RUNDisplay








HELLO




Since this is the only line of the program, the computer will stop executing at this point. Press **ENTER** to get out of the program and reenter RUN if you wish to execute the program again.


## Example 2—Editing a Program

Suppose you wanted to change the message that your program was displaying, that is, you wanted to edit your program. With a single line program you could just retype the entry, but as you develop more complex programs editing becomes a very important component of your programming. Let's edit the program you have just written.

Are you still in the RUN mode? If so switch back to the PROgram mode.

You need to recall your program in order to edit it. Use the Up Arrow  to recall your program. If your program was completely executed, the  will recall the last line of the program. If there was an error in the program, or if you used the BREAK ( **BRK** ) key to stop execution, the  will recall the line in which the error or BREAK occurred. To make changes in your program use the  to move up in your program (recall the previous line) and the  to move down in your program (display the next line). If held down, the  and the  will scroll vertically, that is, they will display each line moving up or down in your program.

You will remember that to move the cursor within the program line displayed at the top line of the display you use the  (right arrow) and  (left arrow). Using the  position the cursor over the first character you wish to change:

**Note:** Even if several lines of a program are displayed on the display unit, the cursor can be moved only within the first displayed line. To edit a lower line, move the line to the top using the  key and then edit.

Input



Display

```
10 PRINT "HELLO"
```

```
10 PRINT "HELLO"
```

Notice that the cursor is now in the flashing block form indicating that it is on top of an existing character. Type in:

Input

GOODBYE"!

Display

```
10 PRINT "GOODBYE"!_
```

Don't forget to press **ENTER** at the end of the line. Change to the RUN mode.

Input

RUN **ENTER**

Display

```
ERROR 1 IN 10
```

This is a new kind of error message. Not only is the error type identified (our old friend the syntax error) but the line number in which the error occurs is also indicated.

Press the **CLS** to clear the error condition.

And return to the PRO mode. You must be in the PROgram mode to make changes in a program. Using the **↑** (or **↓**), recall the line in which the error occurred.

Input

**↑** (or **↓**)

Display

10 PRINT "GOODBYE!"

The flashing cursor is positioned over the problem area. In Chapter 4 you learned that when entering string constants in BASIC all characters must be contained within quotation marks. Use the DELEte key to eliminate the " !":

Input

**DEL**

Display

10 PRINT "GOODBYE" \_

Now let's put the ! in the correct location. When editing programs, DELEte and INSert are used in exactly the same way as they are in editing calculations (See Chapter 3). Using the **◀** position the cursor on top of the character which will be the first character following the insertion.

Input

**◀**

Display

10 PRINT GOODBYE!

Press the INSert key. A **□** will indicate the spot where the new data will be entered:

Input

**INS**

Display

10 PRINT GOODBYE □ "

Type in the !. The display looks like this:



Input

!

Display

10 PRINT GOODBYE!"

Remember to press **ENTER** so the correction will be entered into the program.

- Note:** 1. If you wish to DELEte an entire line from your program just type in the line number and the original line will be eliminated. The DELETE command can be used to delete more than one line at a time.
2. In the program mode, if keys are input when the cursor is not displayed, their corresponding characters are usually displayed on the leftmost column of the top lines of the display unit. However, if the  or  key is pressed when the prompt symbol is displayed, the contents of successive key inputs are displayed starting from the prompt position.

### Example 3—Using Variables in Programming

If you are unfamiliar with the use of numeric and string variables in BASIC, reread these sections in Chapter 4.

Using variables in programming allows much more sophisticated use of the PC-1360's computing abilities.

Remember, you assign simple numeric variables using any letter from A to Z:

```
A=5
```

To assign string variables you also use a letter, followed by a dollar sign. Do not use the same letter in designating a numeric and a string variable. You cannot designate A and A\$ in the same program.

Remember that simple string variables cannot exceed 7 characters in length:

```
A$="TOTAL"
```

The values assigned to a variable can change during the execution of a program, taking on the values typed in or computed during the program. One way to assign a variable is to use the INPUT verb. In the following program the value of A\$ will change in response to the data typed in answering the inquiry "WORD?". Enter this program:

```

10: INPUT "WORD?";A$
20: B=LEN (A$)
30: PRINT "WORD IS ";B;" LETTERS"
40: END

```

↑   ↑   ↑  
\_\_\_\_\_ means space

Since line 30 of this program is longer than 24 columns, the remaining part is displayed in the next line.

The second new element in this program is the use of the END statement to signal the completion of a program. END tells the computer that the program is completed. It is always good programming practice to use an END statement.

As your programs get more complex you may wish to review them before you begin execution. To look at your program, use the LIST command. LIST, which can only be used in the PROgram mode, displays programs beginning with the lowest line number.

Try listing this program:

Input

LIST **ENTER**

Display

```

10: INPUT "WORD?"; A$
20: B = LEN (A$)
30: PRINT "WORD IS ";B;"
    LETTERS"

```

Use the **↑** and **↓** arrows to move through your program until you have reviewed the entire program. To review a line which contains more characters than can be seen at one time, move the cursor to the extreme right of the display and the additional characters will appear on the screen. After checking your program, run it:

Input**CLS**

RUN

HELP

**ENTER**DisplayRUN  
WORD?\_RUN  
WORD? HELP\_RUN  
WORD? HELP  
WORD IS 4. LETTERS

This is the end of your program. Of course you may begin it again by entering RUN. However, this program would be a bit more entertaining if it presented more than one opportunity for input. We will now modify the program so it will keep running without entering RUN after each answer.


Return to the PRO mode and use the up or down arrow (or LIST) to reach line 40. Press the up or down arrow key until the Line 40 comes to the top of the screen or type

Input

LIST 40

**ENTER**Display

40:END

You may type 40 to Delete the entire line or use the  to position the cursor over the E in End. Change line 40 so that it reads:

40: GOTO 10

Now RUN the modified program.

The GOTO statement causes the program to loop (keep repeating the same operation). Since you put no limit on the loop it will keep going forever (an "infinite" loop). To stop this program hit the BREAK (**BRK**) key.

When you have stopped a program using the **BRK** key, you can restart it using the CONT command. CONT stands for CONTInue. With the CONT command the program will restart on the line which was being executed when the **BRK** key was pressed.

## Example 4—More Complex Programming

The following program computes N Factorial (N!). The program begins with 1 and computes N! up to the limit which you enter. Enter this program.

```

100: F=1: WAIT 118
110: INPUT "LIMIT?"; L
120: FOR N=1 TO L
130: F=F*N
140: PRINT N, F
150: NEXT N
160: END

```

Several new features are contained in this program. The WAIT verb in line 100 controls the length of time that displays are held before the program continues. The numbers and their factorials are displayed as they are computed. The time they appear on the display is set by the WAIT statement to approximately 2 seconds, instead of waiting for you to press **ENTER**.

Also in line 100, notice that there are two statements on the same line separated by a colon(:). You may put as many statements as you wish on one line, separating each by a colon, up to the 80 character maximum including **ENTER**. Multiple statement lines can make a program hard to read and modify, however, so it is good programming practice to use them only where the statements are very simple or there is some special reason to want the statements on one line.

Also in this program we have used the FOR verb in line 120 and the NEXT verb in line 150 to create a loop. In Example 3 you created an "infinite" loop which kept repeating the statements inside the loop until you pressed the **BRK** key. With this FOR/NEXT loop the PC-1360 adds 1 to N each time execution reaches the NEXT verb. It then tests to see if N is larger than the limit L. If N is less than or equal to L, execution returns to the top of the loop and the statements are executed again. If N is greater than L, execution continues with line 160 and the program stops.



You may use any numeric variable in a FOR/NEXT loop. You also do not have to start counting at 1 and you can add any amount at each step. See Chapter 9 for details.

We have labelled this program with line numbers starting with 100. Labelling programs with different line numbers allows you to have several programs in memory at one time. To RUN this program instead of the one at line 10 enter:

```
CLS  
RUN 100
```

In addition to executing different programs by giving their starting line number, you can give programs a letter name and start them with the **DEF** key (see Chapter 6).

## Storing Programs in the PC-1360's Memory

You will remember that settings, ReSerVe keys, and functions remain in the computer even after it is turned OFF. Programs also remain in memory when you turn off the PC-1360, or it undergoes an AUTO OFF. Even if you use the **BRK**, CLear or CA keys the programs will remain.

Programs are lost from memory only when you:

- Enter NEW before beginning programming.
- Initialize the computer using the ALL RESET button.
- Create a new program using the SAME LINE NUMBERS as a program already in memory.
- Change the batteries.

This brief introduction to programming on the PC-1360 should serve to illustrate the exciting programming possibilities of your new computer. For more practice in programming exercises, please see the Program Examples.

# Screen Graphic Functions

The display unit (screen) of the PC-1360 is composed on 150 horizontal and 32 vertical dots (points). Simple pictures can be drawn on the screen using these dots. The following 6 commands are available for drawing pictures,

- GPRINT:** Graphic PRINT. This command is used to produce patterns with 8 vertical dots per unit.
- GCURSOR:** Graphic CURSOR. This command is used to specify the position when drawing a pattern with GPRINT.
- PSET:** Point SET. This command is used to light up or reverse a specified point (dot).
- PRESET:** Point RESET. This command clears the specified point (dot).
- LINE:** This command is used to draw a line or a square between 2 specified points.
- POINT:** This function is used to determine whether the specified dot is lit or not.

The basics of picture drawing will be described here. For details on the function of each command, see the description for each command.

Basically, there are 2 ways to draw pictures.

The first is to draw pictures by combining predetermined patterns. The other is to draw pictures by specifying and lighting up each dot (point) as necessary.

To display a combination of characters and numeric values along with graphics, use the graphic command after displaying the characters and numeric values.

## 1. Drawing a Picture Using Predetermined Patterns

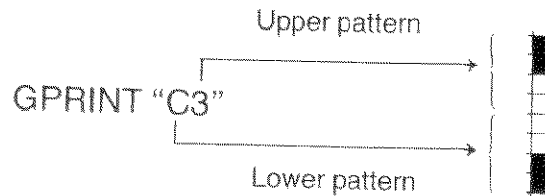
A picture is drawn by combining the 16 available patterns shown in the table below and using the GPRINT command.

Hexadecimal character	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Pattern																

The patterns in this table all use 4 dots (points).

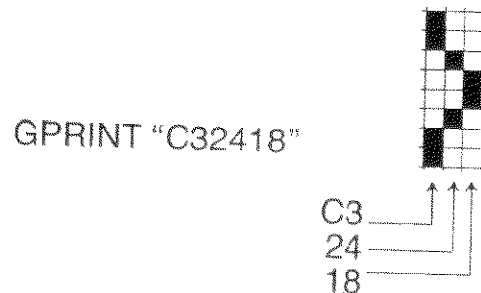
However, in the GPRINT command, two patterns are combined vertically for an 8-dot pattern as shown in the example below. A picture is drawn by combining and lining up a number of 8-dot patterns.

(Example)



A pair of hexadecimal numbers are used with the first specifying the lower pattern and the second specifying the upper pattern.

(Example)

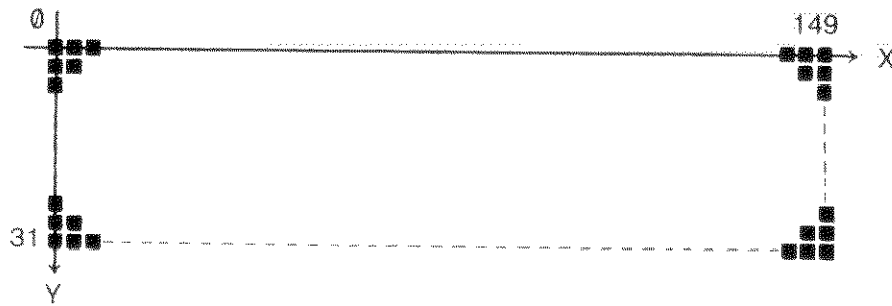


The 8-dot patterns specified by the hexadecimal number pairs are arranged to draw the picture.

### Specifying a Location on the Screen

As described, patterns are specified using the GPRINT command. The location where the pattern is to be displayed can be specified using the GCURSOR command. The display unit (graphic screen) of the PC-1360 is composed of  $150 \times 32$  dots (points).

Each dot, like a point on an X-Y coordinate system, can be specified in the form of (x, y), where "x" is the horizontal direction and "y" is the vertical direction.



Note, however, that in normal X-Y coordinate systems, the point is higher with larger values of "y", while in the PC-1360, the point is lower on the screen.

The coordinates of the dots on the screen range 0—149 for "x" and 0—31 for "y".

**Note:**

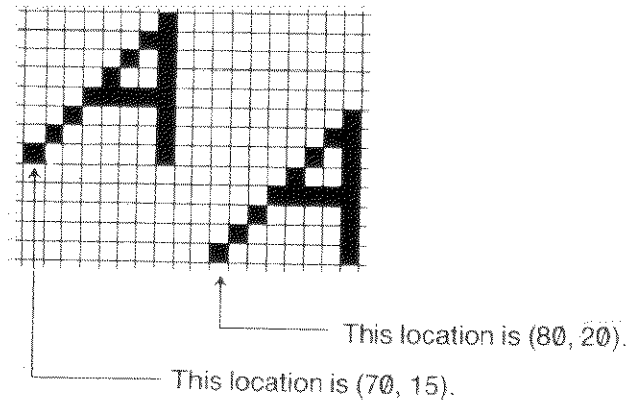
The computer will actually accept "x" and "y" values in the range -32768 to +32767. Any specified value that lies within this range but outside the screen range will be ignored and will not affect the screen. This applies to all screen graphics commands including PSET, PRESET and LINE.

The GPRINT command draws pictures by lining up groups of 8-dot patterns. If the location (dot) where the picture is to be displayed was specified in the GCURSOR command, the first 8-dot pattern is displayed on the 8 dots above and including the dot at the specified location. The rest of the pattern is drawn in sequence.

(Example)

```
5:CLS
10:AA$="80402010181412FF"
20:GCURSOR(70,15)
30:GPRINT AA$
40:GCURSOR(80,20)
50:GPRINT AA$
```

Executing the program will display the following pattern near the center of the screen.



## 2. Drawing a Picture by Specifying Dots One-by-One

A picture is drawn by lighting up or clearing the specified dot using the PSET or PRESET command.

Each dot is specified in the same way as in the GCURSOR command.

(Example)

PSET (75, 15)	Lights up the dot at (75, 15)
PSET (75, 15), X	Lights up the dot at (75, 15) if cleared, and clears it if lit. (Specifying "X" reverses the dot.)
PRESET (75, 15)	Clears the dot at (75, 15)

(Example)

```

100: WAIT 0 : DEGREE
110: FOR A=180 TO -180
      STEP -2
120: B=SIN A*12
130: C=COS A*12
140: X=B+80
150: Y=C+14
160: PSET (X, Y)
170: NEXT A
180: WAIT : GPRINT

```

} \*1  
} \*2  
← \*3  
← \*4

- \*1 Determines the coordinates of the perimeter of a circle with radius 12. The center is at (0, 0).
- \*2 Moves the center of the circle to (80, 14).
- \*3 Lights up the dot specified by (x, y).
- \*4 Continues to display the drawn circle.

Executing this program draws a circle with radius 12 and center at (80, 14).

The angle is changed in 1 units from  $+180^\circ$  to  $-180^\circ$  in the FOR-NEXT loop. At each angle, the coordinates on the circle's perimeter are determined and the corresponding dot is lit.

### Drawing Lines and Squares

Although lines and squares can also be drawn using the PSET command, they can be easily drawn simply by specifying 2 dots in the LINE command.

(Example)

```
LINE (0, 0)—(149, 31)
```

Draws a line from (0, 0) to (149, 31).

```
LINE (30, 0)—(80, 31), B
```

Draws a square with its diagonal from (30, 0) to (80, 31). If B is changed to BF, the inside of the square will be filled.

```
LINE (30, 0)—(80, 31), X, BF
```

A filled box is drawn. However, if a dot within the square is already lit, it is cleared.

(Specifying "X" reverses the dots.)

(Example)

```
200: "A" : WAIT 0
```

```
210: LINE (60, 0)—(100, 31)
      , X, BF
```

```
220: GOTO 210
```

Executing this program draws a square with its diagonal from (60, 0) to (100, 31) and fills and clears it. With the picture previously drawn using the GPRINT and PSET commands still displayed, try executing the program by pressing **DEF** and **A**. The picture within the square will reverse.

- Note:** 1. The graphic screen is 6 dots wider on the left side than the usual character and numeric display.

The PC-1360 is designed to retain the drawn image unless cleared. Consequently, if the program ends or is stopped, the graphic image may remain on the left side or top of the screen.

(The display is cleared by Pressing **CLS**.)

2. Characters or numerals keyed in when the program is stopped by the graphic command may not be displayed. First clear the display (screen) using the **CLS** key and then key in.





## 6 Shortcuts

The PC-1360 includes several features which make programming more convenient by reducing the number of keystrokes required to enter repetitive material.

One such feature is in the availability of abbreviations for verbs and commands (See Chapter 9).

This chapter discusses two additional features which can eliminate unnecessary typing—the **DEF** key and the Reserve mode.

### The DEF Key and Labelled Programs

Often you will want to store several different programs in the PC-1360 memory at one time. (Remember that each must have unique line numbers). Normally, to start a program with a RUN or GOTO command, you need to remember the beginning line number of each program (see Chapter 9). But, there is an easier way! You can label each program with a letter and execute the program using only two keystrokes using **DEF**:

Put a label on the first line of each program that you want to reference. The label consists of a single character in quotes, followed by a colon:

```
10: "A": PRINT "FIRST"
20: END
80: "B": PRINT "SECOND"
90: END
```

Any one of the following characters can be used: A, S, D, F, G, H, J, K, L, =, Z, X, C, V, B, N, M, and SPC. Notice that these are the keys in the last two rows enclosed with the white line to make it easier for you to remember.

To execute the program, instead of typing RUN 80 or GOTO 10, you need only press the **DEF** key and then the letter used as a label. In the above example, pressing **DEF** and then 'B' would cause 'SECOND' to appear on the display.

When **DEF** is used to execute a program, variables and mode settings are affected in the same way as when GOTO is used. See Chapter 9 for details.

## RESERVE Mode

Another timesaving feature of the PC-1360 is the RESERVE mode.

Within the memory of the PC-1360, 144 characters are designated for "Reserve Memory". You can use this memory to store frequently used expressions, which are then recalled by a simple two keystroke operation.

Try this example of storing and recalling a reserved string.

Change the PC-1360 into RESERVE mode by pressing the **SHIFT** and **MODE** keys. Notice that the mode indicator "RUN" and "PRO" disappear and the message "RESERVE MODE" is displayed when the reserve mode is set.

Type NEW followed by the **ENTER** key. This will clear out any previously stored characters in the same way NEW clears out stored programs in the PROgram mode.

Type **SHIFT** followed by 'A':

Input

**SHIFT** A

Display

A: \_

Notice that the 'A' appears in the display at the left followed by a colon.

Enter the word 'PRINT' and press the **ENTER** key:

Input

PRINT **ENTER**

Display

A: PRINT

A space appears after the colon signalling you that 'PRINT' is now stored in the reserve memory under the letter A.

Switch the PC-1360 to PROgram mode. Type NEW followed by **ENTER** to clear the program memory. Type '10' as a line number and then press **SHIFT** and the 'A' key:

Input10 **SHIFT** A**ENTER**Display

10 PRINT \_

10: PRINT

Immediately the word 'PRINT' will appear in the display after the line number.

Any character sequence can be stored in reserve Memory. The stored strings can be recalled at any time in either the PROgram or the RUN mode by typing **SHIFT** and the key that the string is stored under. The keys available are the same as those used with DEF, i.e., those in the area enclosed with the white line.

The maximum length of the reserve string for one key is 80 key strokes, including the key name and the **ENTER** to terminate the definition. Each BASIC command, verb, or function occupies two bytes of the 144 bytes of reserve memory available.

To edit a stored character sequence, switch to the reserve mode and press **SHIFT** and the key under which the sequence is stored. You can then edit using the Left Arrow, Right Arrow, DEL, and INS keys in the same way as in other modes.

When the last character in a stored sequence is a '@' character, it is interpreted as **ENTER** when the sequence is recalled. For example, if you store the string "GOTO 100@" under the 'G' key, typing **SHIFT** and 'G' in the RUN mode immediately starts execution of the program at line 100. Without the '@' character, you must press **ENTER** after the **SHIFT** and 'G' to begin execution.

**To delete reserve programs:**

1. As you know, typing NEW followed by **ENTER** clears all reserve memories. Please note that the above key operation must be done in the RESERVE mode.
2. To delete a reserve memory, use the **SPC** or **DEL** key as described below:

Example: Clear A\*A which is reserved in the key **S**.

SHORTCUTS

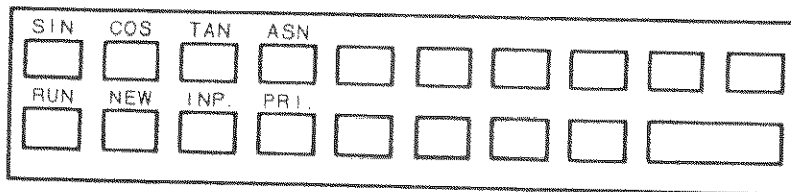
Input	Display	Remarks
<b>SHIFT</b> S	S:_	Reserve mode
A*A <b>ENTER</b>	S:A*A	
<b>CLS</b>	>	
<b>SHIFT</b> S	S:A*A	
◀ or ▶	S:A*A	
<b>DEL DEL DEL</b>	S:_	Delete A*A
<b>ENTER</b>	>	

## Template

A template is provided with the PC-1360. You can use the template to help you remember frequently used ReSeRve sequences or **DEF** key assignments. After you have labelled the programs or created the sequences, mark the template so you know what is associated with each key. You can then execute programs or recall sequences using the two-keystroke operation.

For example, if you have one group of programs which you often use at the same time, label the programs with letters and mark the template so that you can easily begin execution of any of the programs with two keystrokes. You might also store frequently used BASIC commands and verbs in the Reserve Memory and mark a template to speed up entering BASIC programs:

Example:



# 7 Optional Peripherals

The PC-1360 runs with the following optional SHARP peripherals.

MODEL	DESCRIPTION	CONNECTOR
CE-126P	Printer/Cassette interface	printer interface
CE-140P	7 color dot printer	serial I/O interface
CE-515P/516P	4 color plotter/printers	serial I/O interface
CE-124	Cassette interface	printer interface
CE-152	Cassette recorder	via CE-126P or CE-124
CE-130T	Level converter	serial I/O interface

A brief description of each unit is given below. For detailed information refer to individual instruction manuals. See the chapter RAM cards for information on what RAM cards can be used with the PC-1360.

## CE-126P Thermal Printer/Cassette Interface

This optional CE-126P Printer/Cassette Interface allows you to add a printer and to connect a cassette recorder to your SHARP PC-1360 Computer.

The CE-126P features:

- 24 character wide thermal printer.
- Convenient paper feed and tear bar.
- Simultaneous printing of calculations as desired.
- Easy control of display or printer output in BASIC.
- Built-in cassette interface with remote function.
- Manual and program control of recorder for storing programs, data.
- Dry battery operation for portability.

For connecting the PC-1360 to the CE-126P, refer to the instruction manual which is supplied with the CE-126P.

## CE-140P Color Dot Printer

The CE-140P is a multi-color rechargeable dot printer. It is physically much larger than the CE-126P and has a recess into which the computer fits to make an integral unit. A four ink color cartridge offers 7 printing colors. Paper width is 114 mm with up to 80 characters on a line. AC adaptor EA-150 is used to recharge the unit. Features include a text mode for printing characters, numbers

and symbols of varying sizes, and a graph mode for drawing figures. Print head location and direction can be controlled and continuous or a variety of dotted lines drawn. The printer also uses a printing window to ignore printing off the page without generating errors.

The PC-1360 features a number of special graphics commands designed to be used with the CE-140P (see the chapter on computer statements.)

Though the CE-140P connects to the serial I/O interface, the PC-1360 recognizes it as a printer and treats it as if connected to the printer interface like the CE-126P. Listing and printing do not require the serial I/O channel to be opened (see LLIST and LPRINT commands).

### **CE-515P/516P Color Plotter/Printers**

These units are four color plotter/printers. Unlike the CE-140P dot printer, they plot using ink pens and are well suited to producing high quality graphs and diagrams. These plotter/printers are stand alone units that connect to the computer's serial I/O interface via a cable.

Many of the features of the CE-140P dot printer are included such as a text mode for characters, numbers and symbols of varying size and a graph mode for plotting and drawing. A plotting window ignores plotting off the page without generating errors. Both absolute and relative pen movements are possible. The CE-516P is an improved version of the CE-515P offering extended character and symbol sets, 63 character sizes instead of 15, and such plotting functions as arc, circle, box, ellipse and solid shading.

The PC-1360 features a number of special graphics commands designed to be used with the CE-515/516P (see the chapter on computer statements.)

### **CE-130T Level Converter**

The CE-130T is a small adaptor used to adjust input and output levels of the serial I/O interface to conform with RS-232C interface standards. The purpose of this is to let the PC-1360 link up with a wide range of other RS-232C compatible peripherals and computers on the market. For example, a computer to computer link can be used for direct transfer of programs. The unit includes a Ni-Cd battery that is recharged using AC adaptor EA-11E.

## CE-124 Cassette Interface

This unit is required when interfacing the computer to a cassette recorder to save and load programs on tape. (unless the built-in cassette interface is used on the CE-126P printer.) It plugs into the printer interface on the computer side and into the MICrophone and EARphone jack inputs of the recorder. The recommended cassette recorder is SHARP's CE-152 model though others may be used if they meet the following specification:

Item	Requirements
1. Recorder Type	Any standard cassette or micro-cassette recorder
2. Input Jack	"MIC" mini input jack (never the "AUX" jack)
3. Input Impedance	Low input jack impedance (200–1000ohms)
4. Minimum Input Level	Below 3mV or –50dB
5. Output Jack	EXTernal, MONITOR, or EARphone mini output jack, or equivalent
6. Output Impedance	Below 10 ohms
7. Output Level	Above 1V (maximum practical output above 100mW)
8. Distortion	Within 15% (range: 2kHz to 4kHz)
9. Wow and Flutter	0.3% maximum (WRMS)
10. Other	Stable speed recorder motor

## Using the CE-126P Printer/Cassette Interface

### Using the Printer

If you are using the PC-1360 for manual calculation, you may use the CE-126P to simultaneously print your calculations.

This is easily accomplished by pressing the **SHIFT** key and then the **ENTER** key (P ↔ NP) while in the RUN mode.

## OPTIONAL PERIPHERALS

After this, when you press the **ENTER** at the end of a calculation, the contents of the display will be printed on one line and the results will be printed on the next. For example:

Input

300/50 **ENTER**

Paper

300/50  
6.

You may print output on the printer from within BASIC programs by using the LPRINT statement (see Chapter 9 for details). LPRINT can be used in the same form as the PRINT statement.

Programs which have been written with PRINT can be converted to work with the printer by including a PRINT = LPRINT statement in the program (see Chapter 9 for details). All PRINT statements following this statement will act as if they were LPRINT statements. PRINT = PRINT will reset this condition to its normal state. This structure may also be included in a program in an IF statement allowing a choice of output at the time the program is used.

You may also list your programs on the printer with the LLIST command (see Chapter 9 for details). If used without line numbers LLIST will list all program lines currently in memory in their numerical order by line number. A line number range may also be given with LLIST to limit the lines which will be printed. When program lines are longer than 24 characters, two or more lines may be used to print one program line. The second and succeeding line will be indented four or six characters so that the line number will clearly identify each separate program line. (Line number, 1 to 999: four; over 999: six)

### CAUTION:

- In case an error (ERROR code 8) occurs due to a paper misfeed, tear off the paper tape, and pull the remaining part of the paper tape completely out of the printer. Then press the **CLS** key to clear the error condition.
- When the printer is exposed to strong external electrical noise, it may print numbers at random. If this happens, press the **BRK** key to stop the printing. Turn the CE-126P power off and on, and then press the **CLS** key. Pressing the **CLS** key will return the printer to its normal condition.



- When the printer causes a paper misfeed or is exposed to strong external electrical noise while printing, it may not operate normally. If this occurs, depress the **BRK** key to stop printing. (Release the paper misfeed.) Turn the CE-126P power off and on, and then press the **CLS** key.
- When the CE-126P is not in use, turn off the printer switch to save the battery life.

## Using the Cassette Interface

### Recording (saving) onto magnetic tape

See Tape Notes.

1. Turn off the REMOTE switch on the CE-126P.
2. Enter a program or data into the Computer.
3. Load the tape in the recorder. Advance the tape to the position where you want the program to be recorded being careful to avoid the clear tape leader (non-magnetic mylar material) and any programs previously recorded.
4. Connect the Interface's red plug to the tape recorder's MIC jack and the black plug to the REM jack.
5. Turn on the REMOTE switch.
6. Simultaneously press record and play buttons on the tape recorder (to put it in record mode).
7. Enter recording instructions (CSAVE statement, PRINT # statement), and press the **ENTER** key for execution.

First set the unit to "RUN" or "PRO" mode. Next push the following keys:

**C S A V E** **SHIFT** **"** file name **SHIFT** **"** **ENTER**

(To write the contents of data memory onto tape, use the following method:

**P R I N T** **SHIFT** **#** Variable **ENTER**.)

E.g., **C S A V E** **SHIFT** " **A A** **SHIFT** " **ENTER**

When you press the **ENTER** key, tape motion will begin, leaving about an 8-second non-signal blank. (Beep tone is recorded.) After that, the file name and its contents are recorded.

8. When the recording is complete, the PROMPT symbol (>) will be displayed and the tape recorder will automatically stop. Now you have your program on tape (it still is in the PC-1360 also).

When data is to be automatically recorded by program execution (PRINT # statement, not manual operation), set up steps 1 thru 6 before executing the program.

To aid you in locating programs on tapes, use the tape counter on the recorder.

## Verifying a Saved Program

After loading or transferring a program to or from tape, you can verify that the program on tape and program in the PC-1360 are identical (and thus be sure that everything is OK before continuing your programming or execution of programs).

1. Turn off the REMOTE switch.
2. Position the tape to a point just before the file that you want to check.
3. Connect the gray plug to the EARphone and the black plug to the REMote jack sockets.
4. Turn on the REMOTE switch.
5. Press the PLAY button.
6. Input a CLOAD? statement and start execution with **ENTER** key. Do this as follows: Set unit to "RUN" or "PRO" mode. Enter the following key sequence—

The file name which you used previously.

**C L O A D   S H I F T   ?   S H I F T   "   A   A   S H I F T   "   E N T E R**

The PC-1360 will automatically search for the specified file name and will compare the contents on tape with the contents in memory.

When the specified file name is found on the tape, an asterisk "\*" is automatically appended to the line typed in on the screen. This indicates that the actual checking has started.

If the programs are verified as being identical, a PROMPT symbol (>) will be displayed.

If the programs differ, execution will be interrupted and an Error code 8 will be displayed. If this occurs, try again.

### **Loading from a magnetic tape**

See Tape Notes.

To load, transfer, or read out programs and data from magnetic tape into the PC-1360, use the following procedure.

1. Turn off the REMOTE switch.
2. Load tape in the tape recorder. Position tape just before the portion to be read out.
3. Connect the gray plug to the EARphone and the black plug to the REMote jack sockets (if the recorder has no REMote socket, use the PAUSE button to control tape advancement manually.)
4. Turn on the REMOTE switch.
5. Push the PLAY button on the tape recorder (playback mode).

Set the VOLUME control to middle or maximum.  
Set Tone to maximum treble.

6. Input transfer instructions (CLOAD statement, INPUT # statement), and press **ENTER** key for execution.

Put the unit into "RUN" or "PRO" mode. Then push the following keys:

**C L O A D** **SHIFT** " file name **SHIFT** " **ENTER**.

(To load the contents of the data memory, push as follows: **I N P U T** **SHIFT** # variable **ENTER**.)

E.g., **C L O A D** **SHIFT** " **A A** **SHIFT** " **ENTER**

The specified file name will be automatically searched for and its contents will be transferred into the PC-1360.

Once the specified file is found on tape, loading begins. This is indicated by an asterisk "\*" that is automatically appended to the line typed in on the screen.

7. When the program has been transferred the computer will automatically stop the tape motion and display the PROMPT (>) symbol.

To transfer data (INPUT # statement) in the course of a program, set up steps 1 thru 5 prior to executing the program.

If an error code 8 is generated, try running the load again from the beginning. If it occurs again repeat the process after adjusting the volume up or down a little. If no error code is displayed but the tape does not stop, something is wrong. Press the **BRK** key to stop the tape and try again.

## Tape Notes

1. Always use the same recorder for checking or loading that was used for saving the program. Using a different model may generate errors.
2. Use high quality cassette tapes only. Standard audio tapes should not be used.
3. Keep tape heads and mechanism clean. If errors in reading, checking or loading programs persist, the problem may be a dirty head. Try cleaning and demagnetizing the head then repeating the process.

4. Set the volume to between middle and maximum. An incorrect setting is also sometimes the cause of errors. Experiment by adjusting the level each time until no errors are generated.
5. Keep all leads and connections clean and correctly attached.
6. Sometimes using the AC adaptor for the CE-126P interface can cause hum to affect the recording signal. If this happens, switch to battery usage.
7. The tone control should always be set to maximum.
8. When reusing an old tape for recording programs, erase all old programs on the tape first before recording.

## Using Color Printers

Connection of the optional CE-140P Color Dot Printer or CE-515P/CE-516P Color Plotter/Printer to the **COMPUTER** allows you to have hard-copy outputs of programs and calculation results as well as graphic printouts in multiple colors. The CE-140P can draw a figure in seven different colors using four color cartridge inks, whereas the CE-515P draws a chart or diagram with four different color pens. The CE-140P printer can be connected directly to the serial I/O interface of the computer. However, the CE-515P requires cable (CE-516L) for connection to the serial I/O interface.

### Color Printer Notes

The BASIC commands used in the **COMPUTER** include various printer commands applicable to a color dot printer or color plotter/printer, as well as the LLIST and LPRINT commands for printing programs and calculation results.

The CE-515P color plotter/printer can be used only when the serial input/output interface of the **COMPUTER** is in the "Open" state. Therefore, be sure to execute the OPEN command before executing any printer command to the plotter/printer.

## Drawing Range and Coordinates of Graphics

When printing data in graphic form will the CE-140P or CE-515P, you may use a printer function called scissoring, whereby that portion of a figure or diagram that should fall on the printing paper is actually drawn and that portion that should fall outside the printing paper is imaginarily drawn. This function is very convenient when you wish to draw only a part of a figure, but it is easier for you to prepare a program for drawing the entire figure, or when you draw a large diagram by dividing it into some parts according to the size of the printing paper.

However, if your program is incorrect, the figure intended to be drawn on the printing paper may be drawn in the imaginary area. So be sure to program correctly when you use this function.

The directions and positions required for drawing a line in graphics printing are expressed by X and Y coordinates. With regard to coordinate X for horizontal direction, “-” indicates the leftward direction and “+”, the rightward direction. With regard to coordinate Y for vertical direction, “+” indicates the upward direction and “-”, the downward direction.

When using any of the optional color printers, refer to the operation manual supplied with each printer for detailed information.

## Serial I/O Function

The **COMPUTER** is equipped with a serial I/O interface. This interface function can be used to connect the optional CE-140P or CE-515P color printer to the **COMPUTER** for graphic printing in multiple colors and to allow data transfer between the **COMPUTER** and the host computer.

**Note:** Exercise care since applying to the I/O terminal a voltage exceeding the allowable range of the **COMPUTER** may damage the internal parts.

### Basics on Use of the Serial I/O Interface

The circuit of the serial I/O interface is usually closed. If closed, data from the serial I/O terminal cannot be sent and the received data cannot be read.

Therefore, you must open the circuit using the OPEN command. (An attempt to execute this command when already open will result in ERROR 8.)

Further, the conditions to perform data I/O with the host computer connected to the **COMPUTER** must be satisfied. In other words, the conditions for I/O signals must be the same for both the **COMPUTER** and the connected host. If the conditions are different, the I/O signals (data) cannot be read correctly and this will result in data errors. The OPEN command can be used to set or modify the I/O conditions.

After the conditions for both sending and receiving sides are satisfied and the circuit opened, the following commands are used to perform data or program I/O.

LPRINT, LLIST, SAVE, LOAD, PRINT #1, INPUT #1

At the end of the data (or program) transfer, the circuit of the serial I/O interface is closed. Although the CLOSE command is used to close the circuit, the circuit is also closed when the program ends (such as when the END command is executed) or when the RUN command is executed.

When writing a program which uses the serial I/O interface, the circuit must be opened, the I/O operation performed, and then the circuit must be closed as described above.

**Note:** The **COMPUTER** is not equipped with a timer function to interrupt communication with the connected equipment by measuring the waiting time for each I/O command to the serial I/O interface.

Therefore, if the connected equipment is not ready to communicate (such as when the power is off) while a command is being executed or if communication at the connected equipment is interrupted, the **COMPUTER** cannot terminate its command execution.

If this is the case, press the **BRK** key and stop the command execution.

### Serial I/O Function during CE-140P Use

Connection of the **COMPUTER** with the optional CE-140P color dot matrix printer enables on-the-spot printout with a single compact unit.

A serial I/O interface is provided on the side of the CE-140P printer to enable connection of the computer and printer to another personal computer.

- The CE-140P switch should normally be set to SIO. The printer can be used as follows depending on the system configuration.

CE-140P switch position	Serial I/O interface	CE-140P only connected	CE-140P and CE-126P connected*1
SIO (Set here for normal use.)	OPEN	I/O performed from CE-140P I/O interface	Same as left
	CLOSED	Output printed on CE-140P	Output printed on CE-126P*3
PRINTER	OPEN*2	Output printed on CE-140P	Same as left
	CLOSED	Output printed on CE-140P	Output printed on CE-126P*3

\*1 In this case, the CE-126P printer is connected to the connector on the left side of the **COMPUTER** together with the CE-140P printer.

\*2 When using the CE-140P in this setting, specify serial I/O conditions with the OPEN command as described below (Initialized: refer to the OPEN command.)  
 OPEN "1200, N, 8, 1, A, C, &1A"  
 If the I/O condition setting is not correct, the CE-140P printer will not operate properly.

\*3 Commands related to graphics are output to the CE-140P.

- Notes:**
1. Before connecting the CE-140P to the computer, be sure to turn off the power of the computer.
  2. While the LOW BATTERY indicator of the CE-140P is illuminating, an attempt to produce any outputs on the CE-140P by commands from the computer will result in an error (Error code 8).
  3. When the CE-140P is in the ink replaceable state, the power of the computer cannot be turned off even if you slide the power switch to the OFF position. Press the **INK** key of the CE-140P and then turn off the power of the computer.

### When using the serial I/O commands

When several programs are stored within the PC-1360 with the MERGE command, a program cannot be sent through the serial I/O interface with the SAVE or LOAD command. This is because the MERGE command of the PC-1360 differs from the MERGE command of other personal computers.



**Note:**

1. Applying voltages exceeding allowable PC-1360 ranges to serial I/O terminal pins may damage internal parts.
2. The PC-1360 serial interface cannot detect if the other computer is ready to receive data or is busy. The PC-1360 will continue execution of I/O commands and cannot be interrupted by a request from the receiving side. If the other computer stops receiving data, use the **BRK** key on the PC-1360 to stop transmission.



# 8 Using the RAM Cards

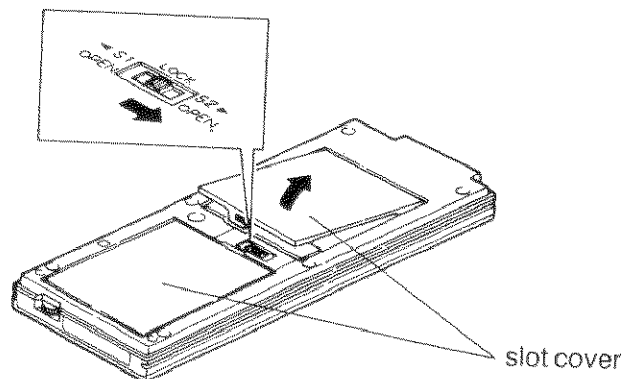
The PC-1360 has two slots accessed from underneath the computer that can hold a variety of RAM cards. Programs and data are held in the cards according to the memory structure selected by using the SET MEM command. The PC-1360 does not have any internal RAM of its own and so it cannot be operated as a computer unless there is at least a RAM card in slot 1. RAM cards feature build in batteries that protect the memory on removal from the computer. In this way programs can be stored and saved on RAM card as an alternative to cassette tape. Memory capacity varies from 2 KBytes to 32 KBytes of RAM.

RAM card name	Capacity
CE-210M	2 KB
CE-211M	4 KB
CE-212M	8 KB (supplied as standard)
CE-2H16M	16 KB
CE-2H32M	32 KB

## Mounting the RAM Card

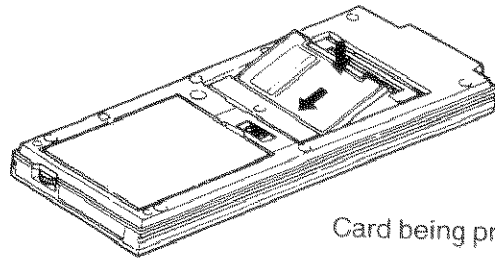
Always mount one RAM card at a time. The lock on the underneath of the machine has three positions. The middle position locks both slot covers. Moving the lock in opposite directions will open one of the two slots.

1. First check that the RAM card contains its internal battery.
2. Turn off the PC-1360
3. Open one of the slots by moving the lock and remove the cover. Be careful not to touch any of the pin terminals of the RAM card.



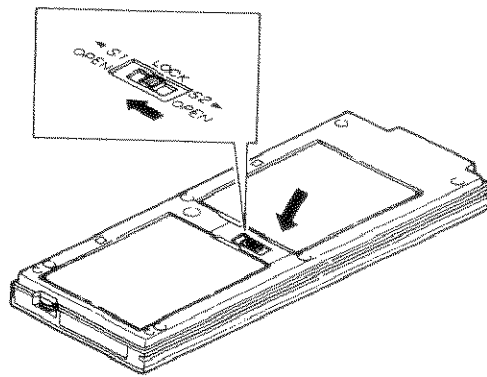
## USING THE RAM CARDS

4. Insert the terminal end of the RAM card according to the arrow. Be careful not to insert the card backwards.
5. Gently press the RAM card and the card will snap into place.



Card being pressed down

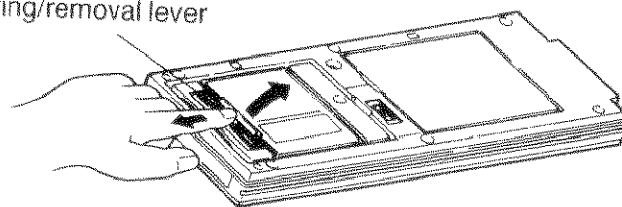
6. Replace and lock the slot cover. Note that the PC-1360 will not operate unless the cover is locked. If you forget this, lock the cover, turn off the power and turn on again.



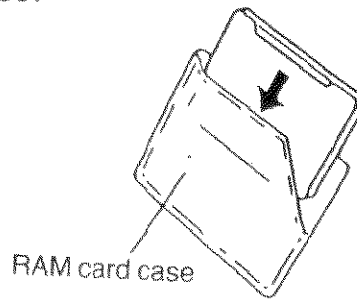
## Removing the RAM Card

1. Turn the PC-1360 off.
2. Remove the slot cover of the PC-1360 as described in the mounting instructions. Pull the card mounting/removal lever in the direction of the arrow as shown in the figure. This releases the RAM card from the tab so that it can be removed.

Mounting/removal lever



Take out another RAM card from its case, and keep the replaced RAM card in the RAM card case.



3. Attach the slot cover to the PC-1360 and slide the lock to the "LOCK" position.

## Using the RAM Card

The structure of memory usage depends on the SET MEM setting. There are three options. Refer to the SET MEM command for details.

The program written the RAM card is retained (by the battery inserted into the RAM card) even if the RAM card is removed from the PC-1360. Therefore, the same program can be used by mounting the RAM card again into the PC-1360.

The PC-1360 can hold two RAM cards in its two RAM slots. When using two RAM cards, always be careful to reinsert them in the same slot that they were in before. This is because the function of each card differs according to which slot it is in. Usually, the program is in the card in slot 1 and variables are held in the card in slot 2. See the SET MEM command for options on using the slots. If you confuse the cards and reinsert them in the wrong slots, the initialization message appears. To proceed with initialization, press the **Y** key.

Possible slot usage combinations are shown below:

1. Slot 1 = MEM\$ "C" card  
Slot 2 = anything
2. Slot 1 = MEM\$ "B" program side  
Slot 2 = MEM\$ "B" data side
3. Slot 1 = MEM\$ "D" program side  
Slot 2 = MEM\$ "D" data side

## USING THE RAM CARDS

With a MEM\$“D” card, the accompanying card in the other slot need not be formatted first.

Initialization messages appear when the following combinations are used:

1. Slot 1 = MEM\$“D” program side card  
Slot 2 = new card

```
MEM$ = "D"  
RAM CARD S2 CLEAR O.K. ?
```

2. Slot 1 = new card  
Slot 2 = MEM\$“D” data side card

```
MEM$ = "D"  
RAM CARD S1 CLEAR O.K. ?
```

When using the cards together like this and pressing any key except the **Y** key, the initialization message appears:

```
MEM$ = "C"  
RAM CARD S1 CLEAR O.K. ?
```

## Precautions When Using the RAM Card

The stored program and data are cleared when the battery in the RAM card is replaced (or removed). Therefore, if a valuable program is stored in the RAM card, it is recommended that the program be recorded beforehand on tape.

The program and data stored within the RAM card are cleared if the ALL RESET button is carelessly pressed. Refer to chapter 2 for initializing the PC-1360 while protecting RAM card contents.

# Copying RAM Cards

RAM cards that do not contain password protected programs can be copied. The card to be copied from is placed in slot 1 and the card to be copied to is placed in slot 2. Both cards must have the same memory size.

1. Switch off power.
2. Insert the card to be copied from in slot 1.
3. Insert the card to be copied to in slot 2.
4. While pressing the **C** key, turn on power. The following screen appears:

**RAM CARD COPY?**

Pressing any other key except the **Y** key aborts the copy and returns you to the normal operation display:

**RUN MODE**

>

Pressing the **Y** key generates another prompt:

**S1:SOURCE, S2:DEST O.K. ?**

Pressing any other key except the **Y** key generates the following screen and power should be turned off:

\*

Pressing the **Y** key start the copying:

**COPYING...**

On completion, a message is displayed:

```
COPY COMPLETED.
```

## Using RAM cards from other computers

PC-1450 and PC-1460 programs held on the card are automatically converted for the PC-1360. After conversion, if the system work area (about 1.5 KB on the RAM card) could not be obtained, the following message appears:

```
MEM$ = "C"  
RAM CARD S1 CLEAR O.K. ?
```

Also, if any line exceeds 80 bytes, the excess part is ignored and after conversion a message is displayed:

```
WARNING:  
SOME LINE EXCEEDED 79 B  
HIT ANY KEY.
```

If conversion executed without errors, the normal operation screen is displayed:

```
RUN MODE
```

```
>
```

Note that all variables etc., are cleared in this process.



# 9 BASIC Reference

This chapter lists alphabetically all the computer statements that run on the PC-1360. Where it exists, the shortest recognizable abbreviation for a statement name is given. Statements are classified into three main types:

## Commands

Commands are used outside a program to change the working environment, perform utilities, or to control the program. They are typed in without a line number and execute on pressing the **ENTER** key.

## Verbs

Verbs are used within a program. Each one forms a single BASIC statement. The verb is declared with a line number on the left of the equals sign of the BASIC statement (except when more than one verb is on a single program line.)

## Functions

A function is a special operator used in a BASIC program to change one variable into another. It appears somewhere on the right of the BASIC statement equals sign except when directly entered using the computer as a calculator. Some functions take an argument. Those that do not are used in the same way as simple program variables and are sometimes called pseudovariables.

Numeric functions take a single numeric value and return a numeric value. They include trigonometric functions, logarithmic functions, and functions which operate on the integer and sign parts of a number. Unlike some forms of BASIC, if the argument part is a number or single variable, the argument need not be enclosed in parentheses.

String functions are used to manipulate text strings. Some take a string argument and return a numeric value. Some take a string argument and return a string. Some take a numeric value and return a string. Some take a string argument and one or two numeric arguments and return a string. Unlike some forms of BASIC, if the function takes a single argument and the argument is not an expression, parentheses may be omitted.

Some instructions can be used as both a command and a verb. All instructions must be typed in upper case. There are five special groups of statements:

## **Screen Graphics Related**

These control graphic output to the screen of the PC-1360.

## **Plotter/Printer Graphics Related**

These are related to the graphic features of the CE-140P color dot printer and the CE-515P/516P color plotter/printer. Graphic data can be sent to the plotter/printer using graphic codes with the LPRINT command. However, these special commands let you do the job more easily and simplify your program. All parameter values specified for plotter/printer graphics related commands (except the ratio given with CIRCLE) are truncated to whole numbers on execution.

## **Cassette Tape Related**

These control the reading and writing of data to a cassette tape.

## **Serial I/O Related**

These concern data input and output via the serial I/O interface.

## Machine Language Related

These are used to call up and execute machine language programs, and to access and read to individual bytes of memory.

These abbreviations are used to classify the type of each statement in the descriptions that follow.

- C=command
- V=verb
- F=function
- P=printer
- Gs=screen graphic related
- Gp=plotter/printer graphic related
- T=cassette tape related
- S=serial I/O related
- M=machine language related

The following conventions have been adopted in compiling this dictionary.

- [ ] The parameter in square brackets is optional. The brackets themselves are not part of the command entry.
- ( ) Used to enclose parameter values in certain commands. They should be entered as part of the command.
- { } Braces indicate that you should select one of the enclosed parameters. The braces are not part of the command entry.
- “ ” Used to enclose parameters in certain commands. They should be entered as part of the command.

# ABS

C,V,F

**FORMAT:** 1. ABS(X)

**Abbreviation:** AB.

**See Also:**

**PURPOSE:**

Returns the absolute value of the number X.

**REMARKS:**

The number X may be any numeric expression.

**EXAMPLE:**

```
5: WAIT 70
10: PRINT "      NUMBER      ABSOLUTE"
20: FOR I=-2 TO 2
30: PRINT I, ABS (I)
40: NEXT I
50: END
```

RUN

NUMBER	ABSOLUTE
-2.	2.
-1.	1.
0.	0.
1.	1.
2.	2.

>

---

# ACS

**C,V,F**

---

**FORMAT:** 1. ACS(X)

**Abbreviation:** AC.

**See Also:** ASN, ATN, COS

---

**PURPOSE:**

Returns the arc cosine of X.

**REMARKS:**

This function returns the inverse cosine of the expression X in degrees, radians, or as a gradient value depending on which mode the computer is set to with the DEGREE, RADIAN, or GRAD command. The value of X is limited to  $-1 \leq X \leq 1$ .

**EXAMPLE:**

```
10: DEGREE
20: PRINT "ARC COS OF 0.5 IS ";ACS(0.5)
30: PRINT "ARC COS OF 0 IS ";ACS(0)
40: END
```

RUN

ARC COS OF 0.5 IS 60.

ARC COS OF 0 IS 90.

>

# AREAD

V

**FORMAT:** 1. AREAD variable name

**Abbreviation:** AR.

**See Also:** INPUT verb and discussion of the use of the **DEF** key in Chapter 6

**PURPOSE:**

The AREAD verb is used to read in a single value to a program which is started using the **DEF** key.

**REMARKS:**

When a program is labelled with a letter, so that it can be started using the **DEF** key, the AREAD verb can be used to enter a single starting value without the use of the INPUT verb. The AREAD verb must appear on the first line of the program following the table. If it appears elsewhere in the program, it will be ignored. Either a numeric or string variable may be used, but only one can be used per program.

To use the AREAD verb type the desired value in the RUN mode, press the **DEF** key, followed by the letter which identifies the program. If a string variable is being used, it is not necessary to enclose the entered string in quotes.

When the display indicates PROMPT (" > ") at the start of program execution, the designated variable is cleared.

Note how text is stored in the following example that uses PRINT at the start of the program.

```
10: "A": PRINT "ABC", "DEFG"
```

```
20: "S": AREAD A$: PRINT A$
```

RUN mode

```
DEF A → ABC
```

DEFG

```
DEF S → DEFG
```

When the display indicates PRINT Numeric expression, Numeric expression, Numeric expression..., or PRINT "String", "String", "String" ..., the contents displayed last are stored.

When the display indicates PRINT Numeric expression; Numeric expression; Numeric expression ..., the contents displayed first (on the extreme left) are stored.

When the display indicates PRINT "String"; "String"; "String" ..., the contents of the "String" designated last (on the extreme right) are stored.

**EXAMPLE:**

```
10: "X":AREAD N  
20: PRINT N ^ 2  
30: END
```

Entering "7 **DEF** X" will produce a display of "49".

---

# ARUN

---

V

**FORMAT:** 1. ARUN [(expression)]  
2. ARUN [(character string)]

**Abbreviation:** ARU.

**See Also:** AUTOGOTO, RUN

---

**PURPOSE:**

Sets the computer to start program execution automatically on power on.

**REMARKS:**

When ARUN is included as the first program statement (i.e., with the lowest line number in the program) at power on, the program will execute as if a RUN command had been entered from the keyboard.

This command is similar to AUTOGOTO except that all variables and arrays are cleared before program execution.

After an AUTO POWER OFF statement has been executed to turn the PC-1360 off, when power is turned on again, the AUTO RUN function is initially inoperative.

**EXAMPLE:**

```
5: ARUN
10: CLS : WAIT 50
20: PRINT "WELCOME TO THE WORLD OF"
30: PRINT "THE COMPUTER"
40: PRINT "YOU HAVE "; MEM ; " BYTES FREE"
50: END
```

The program runs automatically on POWER ON.



# ASC

**C,V,F**

**FORMAT:** 1. ASC (string variable)  
2. ASC ("string")

**Abbreviation:** A.

**See Also:** CHR\$

**PURPOSE:**

Returns the ASCII code value for the first character in the specified string.

**REMARKS:**

The string can be specified as the contents of a string variable in the form X\$ or as an actual string enclosed in quotes, "XXXX". Only the value of the first character in the string is returned regardless of the length. For character code tables, refer to Appendix B.

**EXAMPLE:**

```
10: INPUT "ENTER A CHARACTER ";A$
20: N= ASC (A$)
30: PRINT "THE ASCII CODE IS ";N
40: GOTO 10
```

[10] The user hits a key to enter any character.

[20] ASC finds the code number for this character.

[30] Prints out the answer.

[40] Repeats until the user halts the program by hitting the **BRK** key.

---

# ASN

C,V,F

---

**FORMAT:** 1. ASN(X)

**Abbreviation:**

**See Also:** ACS, ATN, SIN

---

**PURPOSE:**

Returns the arc sine of X.

**REMARKS:**

This function returns the inverse sine of the expression X in degrees, radians, or as a gradient value depending on which mode the computer is set to with the DEGREE, RADIAN, or GRAD command. The value of X is limited to:  $-1 \leq X \leq 1$ .

**EXAMPLE:**

```
5: WAIT 60
10: CLS
20: DEGREE
30: PRINT "ARC SIN", "ANGLE"
40: FOR H=0 TO 10
50: X=H/10
60: DX= ASN X
70: PRINT X,DX
80: NEXT H
```

---

# ATN

---

**C,V,F**

**FORMAT:** 1. ATN(X)

**Abbreviation:** AT.

**See Also:** ACS, ASN, TAN

---

**PURPOSE:**

Returns the arc tangent of X.

**REMARKS:**

This function returns the inverse tangent of the expression X in degrees, radians, or as a gradient value depending on which mode the computer is set to with the DEGREE, RADIAN, or GRAD command.

**EXAMPLE:**

```
10: CLS
20: RADIAN
25: PRINT "TANGENT", "ANGLE"
30: FOR T=0 TO 20
40: RT= ATN (T)
50: PRINT T,RT
60: NEXT T
```

---

# AUTOGOTO

---

V

**FORMAT:** 1. AUTOGOTO (expression)  
2. AUTOGOTO (character string)

**Abbreviation:** AU.

**See Also:** ARUN, GOTO

---

**PURPOSE:**

Sets the computer to start program execution automatically on power on.

**REMARKS:**

When AUTOGOTO is included as the first program statement (i.e., with the lowest line number in the program) at power on, the program will execute as if a GOTO command had been entered from the keyboard. This command is similar to ARUN but does not clear all variables and arrays before execution.

**EXAMPLE:**

```
10: AUTOGOTO 60  
:  
:  
:
```

The program starts executing from line 60 on POWER ON.

---

# BASIC

---

**C**

**FORMAT:** 1. BASIC

**Abbreviation:** BA.

**See Also:** TEXT

---

**PURPOSE:**

Clears the text mode.  
(valid only in program mode)

**REMARKS:**

Executing this command clears the text mode and returns the mode to BASIC. As the mode returns to BASIC, the prompt symbol changes from "<" to ">".

Changing from the text mode to the BASIC mode usually changes the text in the PC-1360 to a program (internal code).

Note that abbreviations such as "P." and "I." are not converted to their respective commands. In such cases, move the cursor to the right line and press the **ENTER** key to convert the abbreviated code into a PC-1360 recognized command name. Any commands or formats used in text mode but not available in the PC-1360 will not be executable.

During program conversion, the " \*\* " mark is displayed at the end of the fourth line.

If a password has been set, executing the BASIC command results in an error (ERROR 1).

# BEEP

C,V

- FORMAT:**
1. BEEP number
  2. BEEP number [, tone [, duration]]

**Abbreviation:** B.

**See Also:**

---

**PURPOSE:**

Generates beeps of the specified tone and duration through the computer's internal speaker.

**REMARKS:**

number specifies the number of times the beep will sound. The value must be a positive number or expression up to 9.999999999E99.

The tone option specifies the rising tone (frequency) of the beep in the range 255 to 0.255 corresponds to a frequency of approximately 230 Hz, and 0 corresponds to a frequency of approximately 2.8 kHz.

The duration option specifies the duration of the beep in the range 0-65535. The beep duration setting varies with the tone parameter. The same duration value will appear relatively longer for lower frequencies.

Specifying just the number uses the command the same way as for the PC-1350 computer, though because of hardware differences the sound may differ slightly. BEEP execution can be halted by pressing the **BRK** key.

The chart below gives details on combinations of parameters used.

Note	No.	Ideal Frequency	Actual Frequency	Tolerance (%)	Tone	Note	No.	Ideal Frequency	Actual Frequency	Tolerance (%)	Tone
do	3	261.6	261.8	0.08	222	do	27	1046.5	1040.7	-0.56	39
do#	4	277.2	277.7	0.18	208	do#	28	1108.7	1113.0	0.39	35
re	5	293.7	294.3	0.20	195	re	29	1174.7	1174.3	-0.03	32
re#	6	311.1	311.4	0.10	183	re#	30	1244.5	1242.7	-0.14	29
mi	7	329.6	329.0	-0.18	172	mi	31	1318.5	1319.6	0.08	26
fa	8	349.2	348.8	-0.11	161	fa	32	1396.9	1406.6	0.69	23
fa#	9	370.0	371.0	0.27	150	fa#	33	1480.0	1471.2	-0.60	21
so	10	392.0	391.4	-0.20	141	so	34	1568.0	1580.2	0.77	18
so#	11	415.3	414.2	-0.27	132	so#	35	1661.2	1662.3	0.07	16
la	12	440.0	439.9	-0.02	123	la	36	1760.0	1753.4	-0.38	14
la#	13	466.2	465.5	-0.15	115	la#	37	1864.7	1855.1	-0.52	12
ti	14	493.9	494.2	0.06	107	ti	38	1975.5	1969.2	-0.32	10
do	15	523.3	522.4	-0.17	100	do	39	2093.0	2098.4	0.26	8
do#	16	554.4	554.1	-0.05	93						
re	17	587.3	589.9	0.44	86						
re#	18	622.3	624.4	0.34	80						
mi	19	659.3	656.4	-0.44	75						
fa	20	698.5	699.4	0.13	69						
fa#	21	740.0	739.9	-0.01	64						
so	22	784.0	785.3	0.17	59						
so#	23	830.6	825.8	-0.58	55						
la	24	880.0	882.8	0.32	50						
la#	25	932.3	934.3	0.21	46						
ti	26	987.8	992.2	0.44	42						

MIN. 230.6 Hz (255)

MAX. 2844.4 Hz (ø)

$$T = (90 + 4n) \times 3.9 \text{ (}\mu\text{s)}$$

$$f = \frac{1}{T} = \frac{1}{(90 + 4n) \times 3.9} \text{ (MHz)}$$

### EXAMPLE:

```

10: FOR I=1 TO 3
20: FOR J=5 TO 25 STEP 5
30: BEEP I,J,150
40: NEXT J
50: NEXT I
60: END

```

[10] This outer loop is used to change the number of beeps from 1 to 3.

[20] The inner loop counter is used to change the tone.

[30] The BEEP statement is executed 15 times.

# CALL

**C,V,M**

---

**FORMAT:** 1. CALL memory address

**Abbreviation:** CA.

**See Also:** PEEK, POKE

---

**PURPOSE:**

Calls up and executes a machine language subroutine.

**REMARKS:**

Calls up a machine language subroutine starting at the specified memory address and begins execution. The address is a value in the range of 0-65535 (&0-&FFFF in hexadecimal.) Normally only bank 0 can be accessed regardless of which RAM card is being used.



---

# CHAIN

---

T,V

- FORMAT:**
1. CHAIN
  2. CHAIN expression
  3. CHAIN "filename"
  4. CHAIN "filename", expression

**Abbreviation:** CHA.

**See Also:** CLOAD, CSAVE, RUN

---

**PURPOSE:**

The CHAIN verb is used to start execution of a program which has been stored on cassette tape. It can only be used in connection with the reading a cassette tape.

**REMARKS:**

To use the CHAIN verb one or more programs must be stored on a cassette. Then, when the CHAIN verb is encountered in a running program, a program is loaded from the cassette and executed.

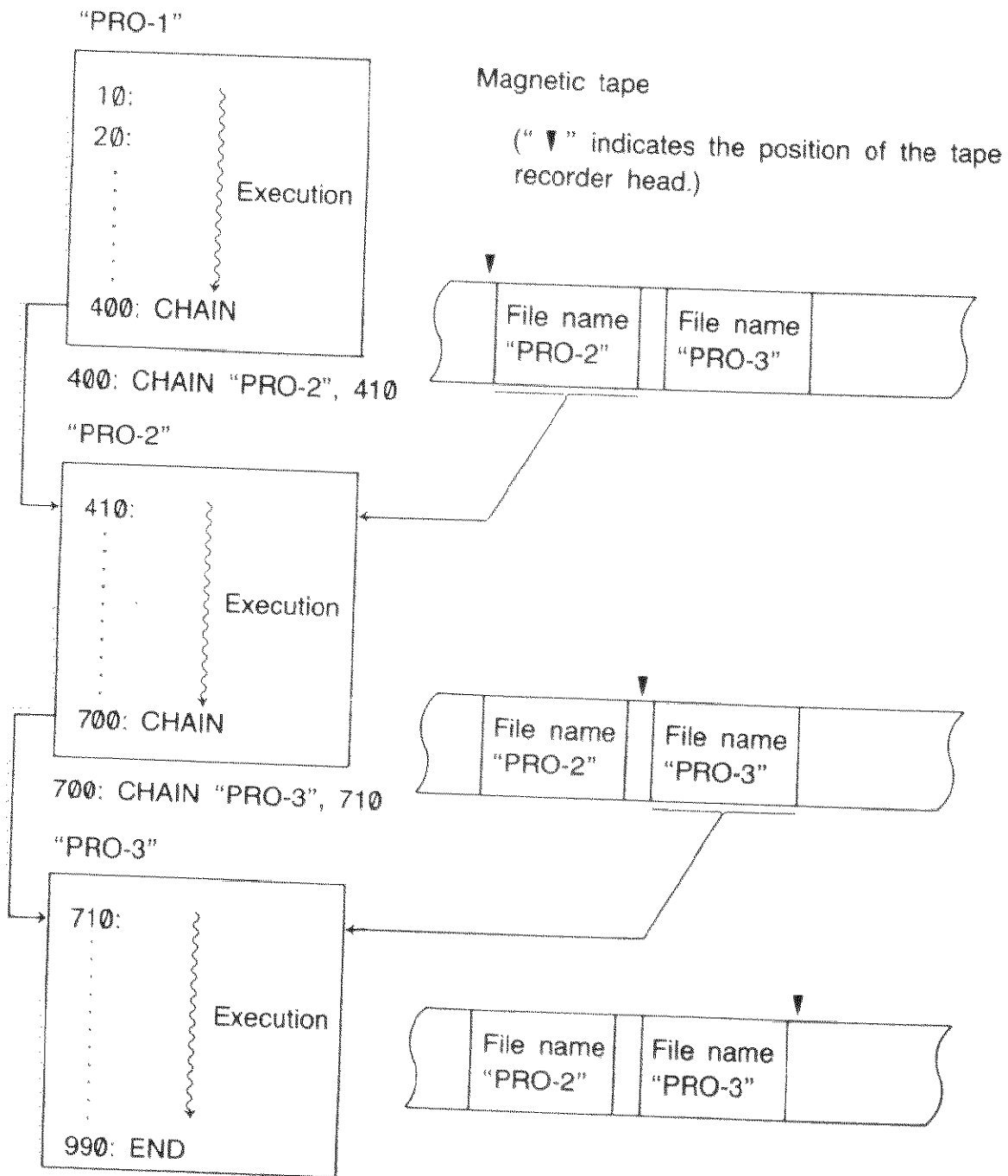
The first form of CHAIN loads the first program stored on the tape and begins execution with the lowest line number in the program. The effect is the same as having entered CLOAD and RUN when in the RUN mode.

The second form of CHAIN loads the first program stored on the tape and begins execution with the line number specified by the expression.

The third form of CHAIN searches the tape for the program whose name is indicated by "filename", loads the program, and begins execution with the lowest line number.

The fourth form of CHAIN will search the tape for the program whose name is indicated by filename, load the program, and begin execution with the line number indicated by the expression.

For example, let's assume you have three program sections named PRO-1, PRO-2, PRO-3. Each of these sections ends with a CHAIN statement.



During execution, when the computer encounters the CHAIN statement, the next section is called into memory and executed. In this manner, all of the sections are eventually run.

Note:

When a program containing a CHAIN command is loaded from the tape by the MERGE command, check to be sure that the CHAIN command is correct.

**EXAMPLE:**

10: CHAIN

20: CHAIN "PRO-2", 480

[10] Loads the first program from the tape and begins execution with the lowest line number.

[20] Searches the tape for a program named PRO-2, loads it, and begins execution with line number 480.

---

# CHR\$

---

C,V,F

**FORMAT:** 1. CHR\$ numeric expression

**Abbreviation:** CH.

**See Also:** ASC

---

**PURPOSE:**

CHR\$ is a string function which returns the character which corresponds to the numeric character code of its argument.

**REMARKS:**

The chart of character codes and their relationship to characters is given in Appendix B. CHR\$ 65 is "A".

If character code 13 is specified when manually executing the CHR\$ command, the specified contents that follow it will not be displayed.

**EXAMPLE:**

CHR\$70+CHR\$71+CHR\$13+CHR\$75+CHR\$76

**ENTER** → FG

Characters K and L for codes 75 and 76 are not displayed.

---

# CIRCLE

**C,V,Gp**

---

**FORMAT:** 1. CIRCLE (expression 1, expression 2), expression 3  
[,expression 4][,expression 5][,expression 6]  
[,expression 7][,expression 8][,expression 9]

**Abbreviation:** Cl.

**See Also:** PAINT, COLOR

---

**PURPOSE:**

The CIRCLE verb is used to draw a circle.

**REMARKS:**

This verb is effective only in the Graphics mode and is used to draw a circle, arc, sector, or ellipse in solid line.

Expressions 1 and 2 are used to specify coordinates X and Y, respectively, at the center point of a circle. The values of expressions 1 and 2 must be within the range of -999 to 999.

Expression 3 is used to specify the radius of a circle. The value of expression 3 must be within the range of 1 to 499 ( $1 \leq \text{expression 3} \leq 499$ ).

Expression 4 is used to specify the color of line. The value of expression 4 may be specified within the range of 0 to 7 in the extended Color mode and 0 to 3 in other than the extended Color mode. (See the COLOR verb for the colors specified by the respective values.) If expression 4 is omitted, the previous value (i.e., the color previously specified) is assumed.

Expressions 5 and 6 are used to specify the starting angle and ending angle, respectively, of an arc or sector. The respective values of expressions 5 and 6 must be within the range of -2047 to 2047. If the value is 0, the right side of the central coordinates is specified. If a negative value is given, the counterclockwise direction is specified. If a positive value is given, the clockwise direction is specified. The default value of expression 5 is 0 degrees, and that of expression 6 is 360 degrees.

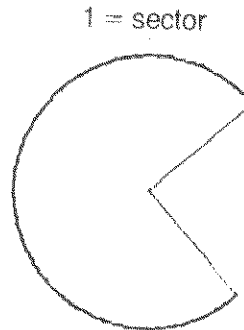
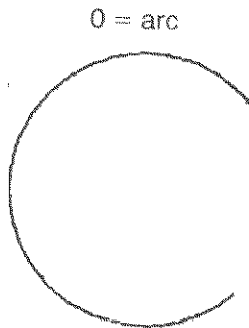
Expression 7 is used to specify the following ratio.

$$\text{Ratio} = \frac{r_y (\text{radius in Y-axis direction})}{r_x (\text{radius in X-axis direction})}$$

If the value of expression 7 is 1, a circle is drawn. If the given value is other than 1, an ellipse is drawn. The default value of expression 7 is 1.

Expression 8 is used to specify a pitch angle. The printer starts drawing a circle (arc or sector) or ellipse by dividing it in units of pitches from the starting angle to the ending angle. The value of expression 8 must be within the range of 1 to 2047. The default value of expression 8 is 1.

Expression 9 is used to specify an arc or a sector. If the value of expression 9 is 0, an arc is drawn. If the value given is 1, a sector is drawn. The default value of expression 9 is 0.



Note:

The respective values of expressions 5, 6, and 8 are specified in units of degrees irrespective of the specified angular mode.

**EXAMPLE:**

```
5: OPEN
10: GRAPH
20: CIRCLE (240, -100), 100, 0, 0, 360, 1/2, 10, 0
30: LTEXT
40: LPRINT
50: END
```



[5] This verb is required for CE-515P/CE-516P.

[20] Ratio = 0.5

[30] These two verbs return the printer to the Text mode and move the print head back to its leftmost position.

5: OPEN

10: GRAPH

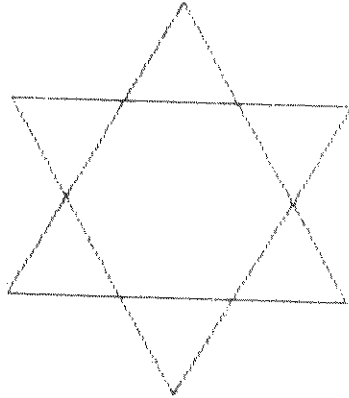
20: CIRCLE (240, -100), 100, 2, 90, 450, 1, 120, 0

30: CIRCLE (240, -100), 100, 3, -90, 270, 1, 120, 0

40: LTEXT

50: LPRINT

60: END



[20] Pitch angle = 120

[30] Pitch angle = 120

The figure is actually printed in color.

# CLEAR

C,V

- 
- FORMAT:**
1. CLEAR
  2. CLEAR variable 1, variable 2,..variable n

**Abbreviation:** CL.

**See Also:** DIM,ERASE

---

**PURPOSE:**

CLEAR is used to erase variables which have been used in the program and to reset all preallocated variables to zero or null.

**REMARKS:**

CLEAR recovers space which is being used to store variables. This might be done when the variables used in the first part of a program are not required in the second part and available space is limited. CLEAR may also be used at the beginning of a program when several programs are resident in memory and you want to clear out the space used by execution of prior programs.

CLEAR does not free up the space used by the variables A—Z, A\$—Z\$, or A(1)—A(26) (without DIM declaration) since they are permanently assigned (see Chapter 4). Format 1 clears all variables resetting the permanent variables to zero or null and leaving 2-character variables undefined. Format 2 allows specific variables to be cleared. Array names cannot be specified in format 2. See ERASE for clearing specified arrays.

**EXAMPLE:**

```
10: A = 5: DIM C(5)
20: CLEAR
```

[20] Frees up the space assigned to C( ) and resets A to zero.



---

# CLOAD

---

**C,T**

**FORMAT:** 1. CLOAD  
2. CLOAD "filename"

**Abbreviation:** CLO.

**See Also:** CLOAD?, CSAVE, MERGE, PASS

---

**PURPOSE:**

The CLOAD command is used to load a program saved on cassette tape.

**REMARKS:**

The first form of the CLOAD command Clears the memory of existing programs and loads the first program stored on the tape, starting at the current position.

The second form of the CLOAD command clears the memory, searches the tape for the program whose name is given by "filename", and loads the program.

If the PC-1360 is in PROgram or RUN mode, program memory is loaded from the tape. When the PC-1360 is in the reserve mode, reserve memory is loaded. Care should be taken not to load programs into reserve memory or reserve characters into program memory.

The computer does not distinguish between a program and a reserve program. If a program or reserve program is loaded into the wrong memory in the computer, the system may hang. If this happens, press the ALL RESET button on the back of the computer to clear the condition.

Break execution of the command using **BRK** if the search is unsuccessful.

If any errors occur during execution, the loaded program will be invalid.

See the section on tape operation in the peripheral chapter.

BASIC REFERENCE

**EXAMPLE:**

CLOAD\*

CLOAD "PRO3"\*\*\*

\*Loads the first program from the tape.

\*\*Searches the tape for the program named 'PRO3' and loads it.

---

# CLOAD?

---

C,T

**FORMAT:** 1. CLOAD?  
2. CLOAD? "filename"

**Abbreviation:** CLO.?

**See Also:** CLOAD, CSAVE, MERGE, PASS

---

**PURPOSE:**

The CLOAD? command is used to compare a program saved on cassette tape with one stored in memory.

**REMARKS:**

To verify that a program was saved correctly, rewind the cassette tape to the beginning of the program and use the CLOAD? command.

The first form of the CLOAD? command compares the program stored in memory with the first program stored on the tape, starting at the current position.

The second form of the CLOAD? command searches the tape for the program whose name is given by "filename" and then compares it to the program stored in memory.

See the section on tape operation in the chapter on peripherals.

Note that if a program is read in from tape having been saved from a PC-1350 computer, program code is automatically converted to be compatible with the PC-1360 and CLOAD? cannot be used for comparison.

**EXAMPLE:**

CLOAD?\*

CLOAD? "PRO3"\*\*\*

\*Compares the first program from the tape with the one in memory.

\*\*Searches the tape for the file name "PRO3" and compares it to the one stored in memory.

# CLOAD M

C,T,M

---

- FORMAT:**
1. CLOAD M [load address]
  2. CLOAD M "filename" [;load address]

**Abbreviation:** CLO.M

**See Also:** CLOAD, CSAVE M

---

**PURPOSE:**

CLOAD M is used to load a machine program from tape.

**REMARKS:**

CLOAD M loads a machine program saved on cassette tape to memory.

BASIC programs already in memory are not overwritten. If load address is omitted, the machine program is loaded to the address specified when CSAVE M was used to save it. If specified, load address causes the machine program to be loaded beginning from this address. If file name is omitted, the tape rewinds and the first program saved on the tape is loaded.

---

# CLOSE

---

**C,V,S**

**FORMAT:** 1. CLOSE [#1]

**Abbreviation:** CLOS.

**See Also:** OPEN

---

**PURPOSE:**

Close the circuit of the serial I/O interface.

**REMARKS:**

This command closes the circuit (in the software sense) of the serial I/O interface which was opened by the OPEN command.

Therefore, after this command is executed, any output to the serial I/O terminal or input from the same terminal cannot be performed.

The #1 may be omitted.

# CLS

C,V

---

**FORMAT:** 1. CLS

**Abbreviation:**

**See Also:** CURSOR

---

**PURPOSE:**

The CLS command clears the display.

**REMARKS:**

Clears the display and returns the display start position to 0.

**EXAMPLE:**

```
10: WAIT 3
20: INPUT A$
30: FOR B=0 TO 23
40: CLS
50: CURSOR B,1
60: PRINT A$
70: NEXT B
80: CLS
90: END
```

This program displays the entry while moving it from left to right on the display unit (from the upper line to the lower line). Each time the FOR-NEXT loop of lines 30-70 is executed, the display is cleared with the CLS command, display start position is shifted with the CURSOR command, and the contents of A\$ are displayed with the PRINT command. By writing and clearing the display in this manner, the display can be made to appear to move. (Delete line 40 and execute. Note the difference.)

# COLOR

**C,V,Gp**

**FORMAT:** 1. COLOR expression  
2. COLOR expression, 7

**Abbreviation:** COL.

**See Also:** LLINE, RLINE, CIRCLE, PAINT

## PURPOSE:

The COLOR verb is used to specify the color of characters or lines to be printed in normal or extended Color mode.

## REMARKS:

Using format 1 (COLOR expression), you can specify four different colors by giving 0 to 3 as the value of expression.

Using format 2 (COLOR expression, 7), you can specify eight different colors by giving 0 to 7 as the value of expression.

With the CE-140P color dot printer, both the formats can be used to specify colors. However, the CE-515P color plotter/printer can only use format 1. The color specifications applicable to the respective printers are as follows:

### CE-140P

Value of expression	0	1	2	3	4	5	6	7
Format 1	Black	Purple	Green	Red				
Format 2	Black	Purple	Red	Magenta	Green	Cyan	Yellow	White

The white specified by value "7" refers to the color of a printing paper. When this value is given, both the print head and the printing paper move, but no printing is executed.

### CE-515P/CE-516P

Value of expression	0	1	2	3
Format 1	Black	Blue	Green	Red

#### BASIC REFERENCE

When the COLOR verb is executed with format 2, the printer is automatically put in the extended Color mode. In this mode, you can select a desired color from the above eight colors when using the LLINE, RLINE, CIRCLE, or PAINT verb.

The printer is released from the extended Color mode when the power switch of the computer is turned off.

#### **EXAMPLE:**

10: GRAPH

20: COLOR 1,7

[20] Sets the extended Color mode and at the same time, specifies "Purple" as the color to be used. (CE-140P must be used in this case.)



---

# CONSOLE

---

**C,V,S**

**FORMAT:** 1. CONSOLE expression

**Abbreviation:** CONS.

**See Also:** OPEN, LPRINT, LLIST

---

**PURPOSE:**

Sets the number of columns per line for data sending.

**REMARKS:**

This command sets the number of columns per line for data sent from the serial I/O interface (terminal) using the LPRINT or LLIST command.

The PC-1360 sends an end code (CR, LF, or CR + LF) after sending the preset line of data.

Valid values of the expression are integer in the range of 1-160. If the value of the expression exceeds 160, 160 columns per line will be set. An error (ERROR 3) results if the value is 0 or negative. A value given in the range 1-7, defaults to 7.

If an expression is not specified, the command is ignored and the number of columns previously set is retained.

The number of columns becomes 39 when the batteries are replaced or when the ALL RESET button is pressed.

---

# CONT

---

C

**FORMAT:** 1. CONT

**Abbreviation:** C.

**See Also:** RUN, STOP

---

**PURPOSE:**

The CONT command is used to continue a program which has been temporarily halted.

**REMARKS:**

When the STOP verb is used to halt a program during execution, the program can be continued by entering CONT in response to the prompt.

When a program is halted using the **BRK** key, the program can be continued by entering CONT in response to the prompt.

CONT can also be used to continue a program interrupted by PRINT or GPRINT.

**EXAMPLE:**

CONT

Continues interrupted program execution.

---

# COS

**C,V,F**

---

**FORMAT:** 1. COS (X)

**Abbreviation:**

**See Also:** ACS, SIN, TAN

---

**PURPOSE:**

Returns the cosine of the angle X.

**REMARKS:**

This function returns the cosine of the angle X, where X is expressed in degrees, radians or as a gradient value, depending on which mode the computer is set to with the DEGREE, RADIAN or GRAD command.

**EXAMPLE:**

```
10: DEGREE
20: PRINT "COS OF 60 IS "; COS (60)
30: PRINT "COS OF 90 IS "; COS (90)
40: END
```

```
RUN
COS OF 60 IS 0.5
COS OF 90 IS 0.
```

>

# CROTATE

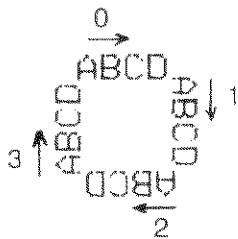
**C,V,Gp****FORMAT:** 1. CROTATE expression**Abbreviation:** CR.**See Also:****PURPOSE:**

CROTATE is used to specify the orientation and printing direction of characters to be printed.

**REMARKS:**

This verb is effective only when the printer is in the Graphic mode. By changing the value of expression, you may change the printing direction and orientation of characters to be printed.

If you give a value in the range of 0 to 3 to the expression, one of the following four orientations and printing directions of characters is specified, causing the printer to perform printing in the direction indicated by the arrow.

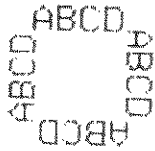


When the LTEXT verb is executed, the orientation and printing direction of characters to be printed change automatically to normal as when the value 0 is given.

**EXAMPLE:**

```
5: OPEN  
10: GRAPH  
20: GLCURSOR (200, -30)  
30: FOR Z=0 TO 3  
40: CROTATE Z  
50: LPRINT "PABCD"  
60: NEXT Z  
70: LTEXT  
80: LPRINT  
90: END
```

[5] This command is required for the CE-515P/CE-516P.



ABCD  
D  
C  
B  
A  
ABCD

# CSAVE

C,V,T

- FORMAT:**
1. CSAVE
  2. CSAVE "filename"
  3. CSAVE, "password"
  4. CSAVE "filename", "password"

**Abbreviation:** CS.

**See Also:** CLOAD, CLOAD?, MERGE, PASS

**PURPOSE:**

The CSAVE command is used to save a program to cassette tape.

**REMARKS:**

The first form of the CSAVE command writes all of the programs in memory on to the cassette tape without a specified file name.

The second form of the CSAVE command writes all of the programs in memory on to the cassette tape and assigns the indicated file name.

The third form of the CSAVE command writes all of the programs in memory on to the cassette tape without a specified file name and assigns the indicated password. Programs saved with a password may be loaded by anyone, but only someone who knows the password can list or modify the programs. (See PASS command.)

The fourth form of the CSAVE command writes all of the programs in memory on to the cassette tape and assigns them the indicated file name and password.

If the PC-1360 is in PROgram or RUN mode, program memory is loaded to the tape. When the PC-1360 is in the Reserve mode, reserve memory is loaded.

See the section on tape operation in the chapter on peripherals.

**EXAMPLE:**

```
CSAVE "PRO3", "SECRET"
```

Saves the programs now in memory on to the tape under the name 'PRO3', protected with the password 'SECRET'.

---

# CSAVE M

**C,V,T,M**

---

**FORMAT:** 1. CSAVE M ["file name";] start address, end address

**Abbreviation:** CS. M

**See Also:** CLOAD, CLOAD M

---

**PURPOSE:**

CSAVE M is used to save a machine program to tape.

**REMARKS:**

CSAVE M saves a machine program held in memory to cassette tape in binary format.

If file name is not specified, the machine program is written to tape without giving it a file name. The program is written to tape starting from the specified start address up to the specified end address.

# CSIZE

**C,V,Gp**

**FORMAT:** 1. CSIZE expression

**Abbreviation:** CSI.

**See Also:**

**PURPOSE:**

CSIZE is used to specify the size of characters to be printed.

**REMARKS:**

The size of characters to be printed can be specified by giving a value within the following range to the expression.

CE-140P  
1 to 63

CE-515P  
1 to 15

Assuming that the value of expression given as "1" is the minimum size of characters, the size of characters specified by 2, 3, 4, ... will be two, three, four, ... times the minimum character size as well as the character pitch and line spacing. When the power switch of the printer is turned on, the printer is in the state to print characters by CSIZE 2.

**EXAMPLE:**

CSIZE 1

Character size: 0.8mm (W) × 1.2mm (H) (4 × 6 steps)

Character pitch: 1.2mm (6 steps)

Line spacing: 2.4mm (12 steps)



# CURSOR

C,V

**FORMAT:** 1. CURSOR expression 1, expression 2  
2. CURSOR expression  
3. CURSOR

**Abbreviation:** CU.

**See Also:** GCURSOR, CLS, INPUT, PRINT, PAUSE

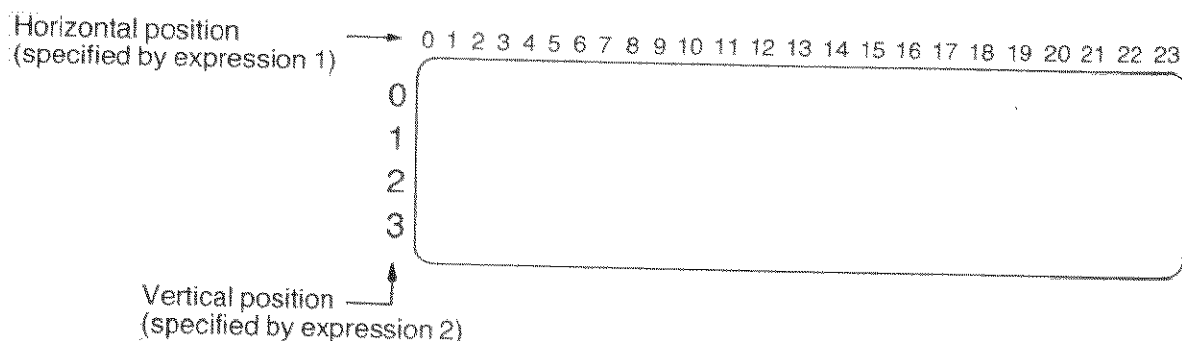
## PURPOSE:

Specifies the display start position in column units.

## REMARKS:

Format (1) and format (2) specify the display start position in units of a character position for the contents displayed by the PRINT command, PAUSE command, etc.

The display position is specified as follows using format (1).



A position on the display unit is specified by its horizontal and vertical positions. The values of expression 1 and expression 2 specify the horizontal and vertical position, respectively. The range of expression 1 is 0-23, and the range of expression 2 is 0-3.

Format (3) clears the display start position.

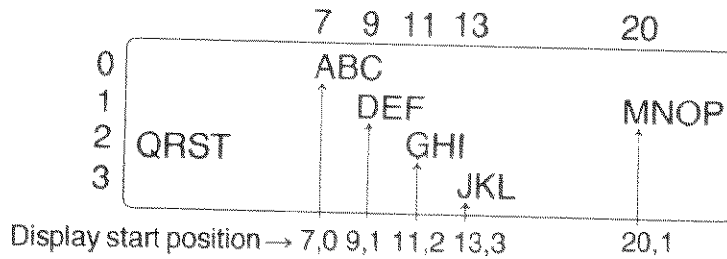
Using the CURSOR command allows text to be written to any part of the screen without affecting existing screen text except where characters are directly overwritten. To clear the whole screen use the CLS command.

If the displayed characters overflow the screen, the screen is moved up (scrolled) to display all the characters even though the display start position was specified with the CURSOR command.

**EXAMPLE:**

```
5: CLS
10: CURSOR 7,0: PRINT "ABC"
20: CURSOR 9,1: PRINT "DEF"
30: CURSOR 11,2: PRINT "GHI"
40: CURSOR 13,3: PRINT "JKL"
50: CURSOR 20,1: PRINT "MNOPQRST"
```

Executing the program will display the following.



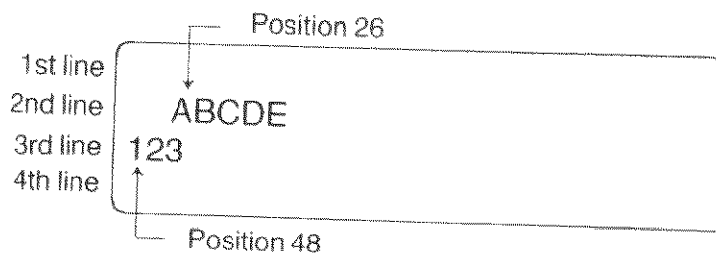
The display position is specified as follows using format (2).

1st line	0 1 2 3 . . . . . 22 23
2nd line	24 25 26 . . . . . 46 47
3rd line	48 49 50 . . . . . 70 71
4th line	72 73 74 . . . . . 95

As shown, the positions on the display are assigned numbers from 0 to 95, starting from the top left of the display unit to the bottom right. The value of the expression in format (2) specifies the number of the position for the display start position. Therefore, be sure that the value of the expression in format (2) is within the range of 0—95. Exceeding this range results in an error (ERROR 3).

**EXAMPLE:**

```
5: CLS
10: CURSOR 48: PRINT 123
20: CURSOR 26: PRINT "ABCDE"
```



CURSOR can also be used to position the screen cursor ready for an INPUT command. However, after INPUT has been executed, the cursor position remains unchanged at the position specified by the CURSOR command so that any PRINT will overwrite the INPUT text.

**EXAMPLE:**

```
10: CLS  
20: CURSOR 0,2  
30: INPUT "DATA = ";A  
40: PRINT A/2
```

[30,40] The position specified in line 20 applies for lines 30 and 40.

```
10: CLS  
20: CURSOR 0,2  
30: INPUT "DATA = ";A  
35: CURSOR  
40: PRINT A/2
```

[35] The position specified in line 20 is cleared.

# DATA

V

**FORMAT:** 1. DATA expression list

Where: expression list    is: expression  
                                 or: expression, expression list

**Abbreviation:** DA.

**See Also:** READ, RESTORE

---

**PURPOSE:**

DATA is used to provide values for use by READ.

**REMARKS:**

When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR... NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the sequential order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

DATA statements have no effect if encountered in the course of regular execution of the program, so they can be inserted wherever it seems appropriate. Many programmers like to include them immediately following the READ which uses them. If desired, the values in a DATA statement can be read a second time by using the RESTORE statement.

**EXAMPLE:**

```
10: DIM B(10)
20: WAIT 128
30: FOR I = 1 TO 10
40: READ B(I)
50: PRINT B(I)
60: NEXT I
70: DATA 10, 20, 30, 40, 50, 60
80: DATA 70, 80, 90, 100
90: END
```

[10] Sets up an array.

[40] Loads the values from the DATA statement into B(1) will be 10, B(2) will be 20, B(3) will be 30, etc.

---

# DEG

**C,V,F**

---

**FORMAT:** 1. DEG dd.mmssrr

**Abbreviation:**

**See Also:** DMS

---

**PURPOSE:**

For an angle specified in degrees, minutes and seconds, converts the value to decimal degrees.

**REMARKS:**

The angle must be expressed in the format dd.mmssrr where dd is degrees (range not limited), mm is minutes in the range 00 to 59, ss is seconds in the range 00 to 59, and rr is the remainder. Note that the degree part is separated from the rest by a decimal point. A number must always be specified before the decimal point, for example specify .5 as 0.5.

**EXAMPLE:**

```
10: X = DEG 50.300000  
20: PRINT X  
30: END
```

RUN

50.5

---

# DEGREE

---

C,V

**FORMAT:** 1. DEGREE

**Abbreviation:** DE.

**See Also:** GRAD, RADIAN

---

**PURPOSE:**

DEGREE is used to change the form of angular values to decimal degrees.

**REMARKS:**

The PC-1360 has three forms for representing angular values —decimal degrees, radians and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The DEGREE function changes the form for all angular values to decimal degree form until GRAD or RADIAN is used. The DMS and DEG functions can be used to convert decimal degrees to degree, minute, second form and vice versa.

**EXAMPLE:**

```
10: DEGREE  
20: X = ASN 1  
30: PRINT X
```

[20] X now has a value of 90, i.e. 90 degrees, the Arcsine of 1.

---

# DELETE

---

C

- FORMAT:**
1. DELETE line #
  2. DELETE line #,
  3. DELETE line #, line #
  4. DELETE , line #

**Abbreviation:** DEL.

**See Also:** NEW

---

**PURPOSE:**

Deletes specified program lines in memory.

**REMARKS:**

DELETE line # deletes only the specified program line. DELETE line #, deletes program lines from the line number specified up to the highest program line in memory. DELETE line #, line # deletes all program lines between the first specified line number (lower value) and the second specified line number (higher value). DELETE ,line # deletes program lines from the lowest line number in memory up to the specified line number.

Using DELETE in run mode generates ERROR 9. If a password has been used, nothing happens and the prompt is displayed. If there is more than one program in memory because of using the MERGE command, DELETE operates on the program last merged. Only the digit 0–9 are accepted for the line numbers. Specifying a line that does not exist generates an error. Specifying a start line number that is greater than the end line number also generates an error.

To delete the whole program, use the NEW command.

**EXAMPLE:**

```
DELETE 150*  
DELETE 200,**  
DELETE 50, 150***  
DELETE ,35****
```

## BASIC REFERENCE

- \* Deletes line 150 only.
- \*\* Deletes from line 200 to the highest line number.
- \*\*\* Deletes all lines between line 50 and line 150.
- \*\*\*\* Deletes from lowest line number up to line 35.



# DIM

**C,V**

<b>FORMAT:</b>	1. DIM dim list	
	Where: dim list	is: dimension spec. or: dimension spec., dim list
	and: dimension spec.	is: numeric dim spec. or: string dim spec.
	and: numeric dim spec.	is: numeric name (size)
	and: string dim spec.	is: string name (dims) or: string name (dims) * len
	and: numeric name	is: valid numeric variable name
	and: string name	is: valid string variable name
	and: dims	is: size or: size, size
	and: size	is: number of elements
	and: len	is: length of each string in a string array

**Abbreviation:** D.**See Also:** ERASE,CLEAR**PURPOSE:**

DIM is used to reserve space for numeric and string array variables.

**REMARKS:**

Except for the array of the form; A( ), A\$( ), two-character( ), and two-character\$( ), DIM must be used to reserve space for any array variable.

The maximum number of dimensions in any array is two; the maximum size of any one dimension is 255. In addition to the number of elements specified in the dimension statement, one additional "zeroeth" element is reserved. For example, DIM B(3) reserves B(0), B(1), B(2), and B(3). In two dimensional arrays there is an extra "zeroeth" row and column.

In string arrays one specifies the size of each string element in addition to the number of elements. For example, DIM B\$(3) \* 12 reserves space for 4 strings which are each a maximum of 12 characters long. If the length is not specified each string can contain a maximum of 16 characters.

When a numeric array is dimensioned, all values are initially set to zero; in a string array the values are set to null.

For the array A and A\$ DIM declaration, refer to the paragraph discussing variables.

Array variables can be cleared (or set undefined) with the CLEAR or ERASE command. When the program is started using the RUN command, array variables are automatically cleared.

An array can only be declared once and any attempt made to redeclare within the program, without first CLEARing or ERASEing, will generate errors. Be careful when executing a program using the GOTO command or DEF key that the same DIM statement will not be executed again unless CLEAR has been used first.

**EXAMPLE:**

```
10: DIM B(10)
20: DIM C$(4, 4)*10
```

[10] Reserves space for a numeric array with 11 elements.

[20] Reserves space for a two dimensional string array with 5 rows and 5 columns; each string will be a maximum of 10 characters.

---

# DMS

**C,V,F**

---

**FORMAT:** 1. DMS angle

**Abbreviation:** DM.

**See Also:** DEG

---

**PURPOSE:**

For an angle specified in decimal degrees, converts the value to degrees, minutes and seconds.

**REMARKS:**

The angle is returned in the format dd.mmssrr where dd is degrees (range not limited), mm is minutes in the range 00 to 59, ss is seconds in the range 00 to 59, and rr is the remainder. Note that the degree part is separated from the rest by a decimal point. The angle must be entered in degrees as a decimal dd. with a decimal point.

**EXAMPLE:**

```
10: X = DMS 50.5
20: PRINT X
30: END
```

RUN

50.3

# END

V

---

**FORMAT:** 1. END

**Abbreviation:** E.

**See Also:**

---

**PURPOSE:**

The END verb is used to signal the end of a program.

**REMARKS:**

When multiple programs are loaded into memory at the same time a mark must be included to indicate where each program ends so that execution does not continue from one program to another. This is done by including an END verb as the last statement in the program.

**EXAMPLE:**

```
10: PRINT "HELLO"  
20: END  
30: PRINT "GOODBYE"  
40: END
```

With these programs in memory a 'RUN 10' prints 'HELLO', but not 'GOODBYE'. 'RUN 30' prints 'GOODBYE'.

---

# ERASE

---

**C,V**

**FORMAT:** 1. ERASE array 1, array 2,... array n

**Abbreviation:** ER.

**See Also:** CLEAR

---

**PURPOSE:**

Erases specified arrays.

**REMARKS:**

ERASE erases specified arrays. The fixed array area A(26) cannot be erased. Individual array elements cannot be erased, the whole array is cleared and its memory area freed up. To redefine an array size, first ERASE it and then respecify it in a DIM statement.

**EXAMPLE:**

```
10: DIM AA(10)
:
:
200: ERASE AA
```

**EXP****C,V,F**

---

**FORMAT:** 1. EXP(X)**Abbreviation:** EX.**See Also:** LN

---

**PURPOSE:**

Returns the value of the exponential function  $e$  raised to the power of  $X$ .

**REMARKS:**

The expression  $X$  must be in the range  $+230.2585092$  to  $-227.9559242$ . For values below this range,  $0$  is returned. To raise another number base to a power, use the “ $\wedge$ ” function. The computer holds the value of  $e$  as  $2.718281828$ .

**EXAMPLE:**

```
EXP(10)
22026.46579
```

Prints out the value of  $e$  raised to the power  $10$ .

---

# FOR

---

V

- FORMAT:**
1. FOR numeric variable = expression 1 TO expression 2
  2. FOR numeric variable = expression 1 TO expression 2  
STEP expression 3

**Abbreviation:** F.; STE.

**See Also:** NEXT

---

**PURPOSE:**

FOR is used in combination with NEXT to repeat a series of operations a specified number of times.

**REMARKS:**

FOR and NEXT are used in pairs to enclose a group of statements which are to be repeated. The first time this group of statements is executed the loop variable (the variable named immediately following the FOR) has the value of expression 1.

When execution reaches the NEXT verb the loop variable is increased by the step size and then this value is tested against expression 2. If the value of the loop variable is less than or equal to expression 2, the enclosed group of statements is executed again, starting with the statement following the FOR. In the first form the step size is 1; in the second form the step size is given by expression 3. If the value of the loop variable is greater than expression 2, execution continues with the statement which immediately follows the NEXT. Because the comparison is made at the end, the statements within a FOR/NEXT pair are always executed at least once.

Expression 1, expression 2, and expression 3 must be in the range of  $-9.999999999E99$  to  $9.999999999E99$ . When expression 3 is zero FOR/NEXT will execute in an infinite loop.

The loop variable may be used within the group of statements, for example as an index to an array, but care should be taken in changing the value of the loop variable.

Programs should be written so that they never jump from outside a FOR/NEXT pair by jumping out. This is because loop execution does not terminate even on jumping out. Always exit a FOR/NEXT loop via the NEXT statement. To do this, set the loop variable to a value higher than expression 2.

The group of statements enclosed by a FOR/NEXT pair can include another pair of FOR/NEXT statements which use a different loop variable as long as the enclosed pair is completely enclosed; i.e., if a FOR statement is included in the group, the matching NEXT must also be included. FOR/NEXT pairs may be "nested" up to five levels deep. Illegally jumping out of an inner loop will generate ERROR 5, a nesting error.

**EXAMPLE:**

```

10:FOR I=1 TO 5
20:PRINT I
30:NEXT I
40:FOR N = 10 TO 0 STEP -1
50:PRINT N
60:NEXT N
70:FOR N = 1 TO 10
80:X = 1
90:FOR F = 1 TO N
100:X = X*F
110:NEXT F
120:PRINT X
130:NEXT N

```

[10–30] This group of statements prints the numbers 1, 2, 3, 4, 5.

[40–60] This group of statements counts down 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0.

[70–130] This group of statements computes and prints N factorial for the numbers from 1 to 10.



# GCURSOR

**C,V,Gs**

**FORMAT:** 1. GCURSOR (expression 1, expression 2)

**Abbreviation:** GC.

**See Also:** CURSOR, GPRINT

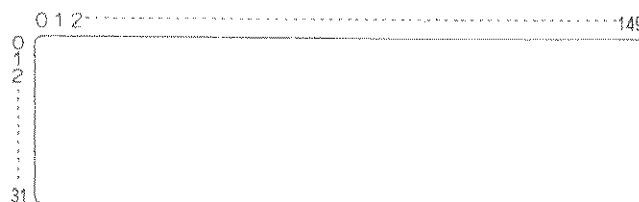
**PURPOSE:**

Specifies the display start position in dot (point) units.

**REMARKS:**

The GCURSOR command specifies the start position of the display for the GPRINT command. (Moves the graphic cursor to the specified position.)

The display unit is composed of 150 horizontal and 32 vertical dots (points). Each dot is assigned a number ranging 0—149 in the X direction and 0—31 in the Y direction. The display start position is specified by specifying the number in the X direction with expression 1 in the format above and the number in the Y direction with expression 2.



The values of expression 1 and expression 2 can be specified in the range of -32768 to +32767. However, if the values for expression 1 and expression 2 exceed the ranges of 0—149 and 0—31, respectively, the specified position will be outside the boundaries of the display unit and a position which does not actually exist will be specified for the display start position.

The display start position of (0, 7) will be specified when the RUN command or CLS command is executed or when **SHIFT** and **CA** are pressed.

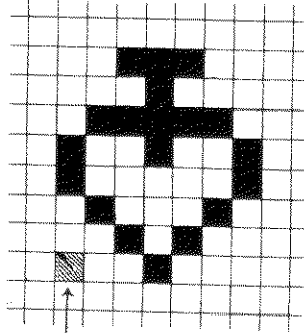
When the program is started using the GOTO command or defined key, the value for the y-direction is retained. The x-direction returns to 0.

**EXAMPLE:**

```
10: GCURSOR (71, 20)
```

```
20: GPRINT "1824458F452418"
```

Executing this program displays the following near the center of the display unit.  
(The shaded part is not displayed.)



In the program above, this shaded position is the display start position (71, 20).

# GLCURSOR

C,V,Gp

**FORMAT:** 1. GLCURSOR (expression 1, expression 2)

**Abbreviation:** GL.

**See Also:**

**PURPOSE:**

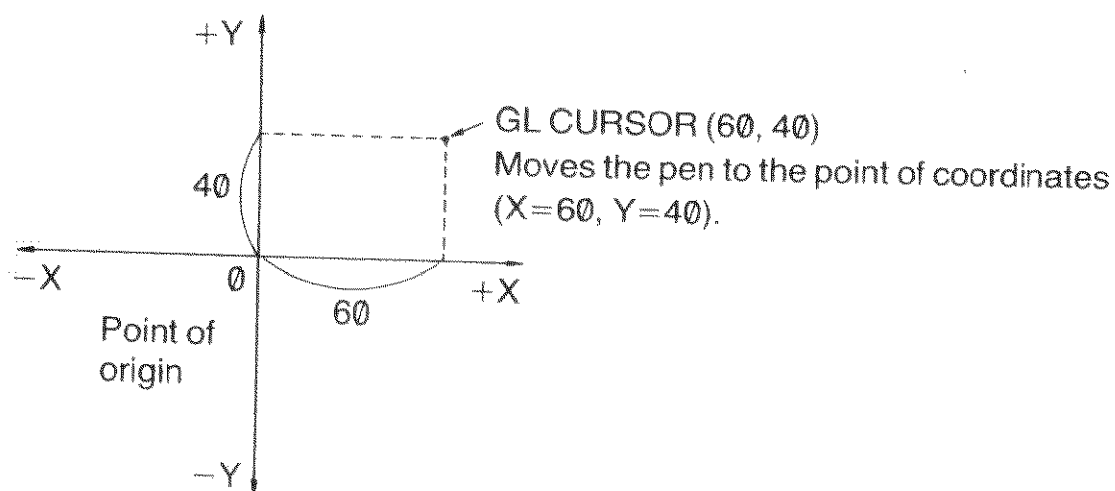
GLCURSOR is used to move the pen.

**REMARKS:**

This verb is effective only in the Graphics mode and is used to move the pen in the X-axis or Y-axis direction from the origin of coordinates.

The pen moves to the point of coordinates specified by expression 1 (X coordinate) and expression 2 (Y coordinate).

The value of each expression must be within the range of  $-999$  to  $999$ . The minimum amount of movement is  $0.2$  mm in either direction.



**Note:**

Moving the pen in the  $+Y$  direction means that the paper feeding must be effected in reverse direction.

Refer to the operation manual of the printer for the scissoring area.

BASIC REFERENCE

**EXAMPLE:**

10: GRAPH

20: GLCURSOR (60, 40)

[20] Moves the pen to the point of coordinates (X=60, Y=40).

---

# GOSUB

---

V

**FORMAT:** 1. GOSUB expression

**Abbreviation:** GOS.

**See Also:** GOTO, ON... GOSUB, ON... GOTO, RETURN

---

**PURPOSE:**

GOSUB is used to execute a BASIC subroutine.

**REMARKS:**

When you wish to execute the same group of statements several times in the course of a program or use a previously written set of statements in several programs, it is convenient to use the BASIC capability for subroutines using GOSUB and RETURN.

The group of statements is included in the program at some location where they are not reached in the normal sequence of execution. A frequent location is following the END statement which marks the end of the main program. At those locations in the main body of the program—where subroutines are to be executed, include a GOSUB statement with an expression which indicates the starting line number of the subroutine. The last line of the subroutine must be a RETURN. When GOSUB is executed, the PC-1360 transfers control to the indicated line number and processes the statements until a RETURN is reached. Control is then transferred back to the statement following the GOSUB.

A subroutine may include a GOSUB. Subroutines may be “nested” in this fashion up to 10 levels deep.

Since there is an ON ... GOSUB structure for choosing different subroutines at given locations in the program, the expression usually consists of just the desired line number. When a numeric expression is used it must evaluate to a valid line number, i.e., 1 to 65279, or an ERROR 4 will occur.

## BASIC REFERENCE

### **EXAMPLE:**

```
10: GOSUB 100
```

```
20: END
```

```
100: PRINT "HELLO"
```

```
110: RETURN
```

When this program is run it prints the word 'HELLO' one time.

# GOTO

WHEN USED AS A COMMAND:

**C**

GOTO can be used as both a command and a verb.

**FORMAT:** 1. GOTO expression

**Abbreviation:** G.

**See Also:** RUN

---

**PURPOSE:**

The GOTO command is used to start execution of a program.

**REMARKS:**

The GOTO command can be used in place of the RUN command to start program execution at the line number specified by the expression.

GOTO differs from RUN in eight respects:

- 1) The value of the interval for WAIT is not reset.
- 2) The display format established by USING statements is not cleared.
- 3) Variables and arrays are preserved.
- 4) PRINT=LPRINT status is not reset.
- 5) The pointer for READ is not reset.
- 6) The cursor specification is maintained.
- 7) The horizontal direction of the graphic cursor is specified with 0. The setting for the vertical direction is maintained.
- 8) The serial I/O circuit is not closed.

Execution of a program with GOTO is identical to execution with the **DEF** key.

**EXAMPLE:**

```
GOTO 100
```

Begins execution of the program at line 100.

# GOTO

WHEN USED AS A VERB:

V

**FORMAT:** 1. GOTO expression

**Abbreviation:** G.

**See Also:** GOSUB, ON ... GOSUB, ON ... GOTO

**PURPOSE:**

GOTO is used to transfer control to a specified line number.

**REMARKS:**

GOTO transfers control from one location in a BASIC program to another location. Unlike GOSUB, GOTO does not "remember" the location from which the transfer occurred.

Since there is an ON ... GOTO structure for choosing different destinations at given locations in the program, the expression usually consists of just the desired line number. When a numeric expression is used, it must evaluate to a valid line number, i.e., 1 to 65279, or an ERROR 4 will occur.

Well designed programs usually flow simply from beginning to end, except for subroutines executed during the program. Therefore, the principal use of the GOTO verb is as a part of an IF ... THEN statement.

**EXAMPLE:**

```
10: INPUT A$  
20: IF A$="Y" THEN GOTO 50  
30: PRINT "NO"  
40: GOTO 60  
50: PRINT "YES"  
60: END
```

This program prints 'YES' if a 'Y' is entered and prints 'NO' if anything else is entered.



# GPRINT

**C,V,Gs**

- FORMAT:**
1. GPRINT string
  2. GPRINT expression; expression; expression;...
  3. GPRINT

**Abbreviation:** GP.

**See Also:** GCURSOR, PRINT

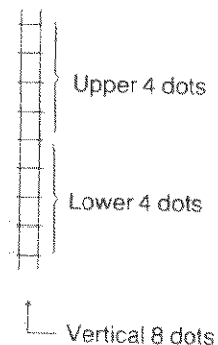
**PURPOSE:**

Displays the specified dot pattern.

**REMARKS:**

The GPRINT command displays the specified dot pattern. A vertical line of 8 dots is specified as one dot pattern.

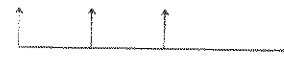
In format (1), the 8-dot pattern, divided into a lower 4 dots and an upper 4 dots, is specified by a string enclosed within " ", where the 4-dot patterns are represented by hexadecimal numbers.



BASIC REFERENCE

Hexadecimal number	Dot pattern	Hexadecimal number	Dot pattern	Hexadecimal number	Dot pattern	Hexadecimal number	Dot pattern
0		4		8		C	
1		5		9		D	
2		6		A		E	
3		7		B		F	

GPRINT "○○ ○○ ○○ ..."



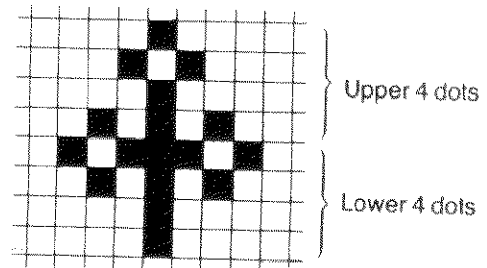
A single group (pair) of 2 numerals specifies one vertical (8-dot) pattern. Any numeral remaining at the end of the string which is not in a pair will be ignored.

**EXAMPLE:**

GPRINT "102812FD122810"

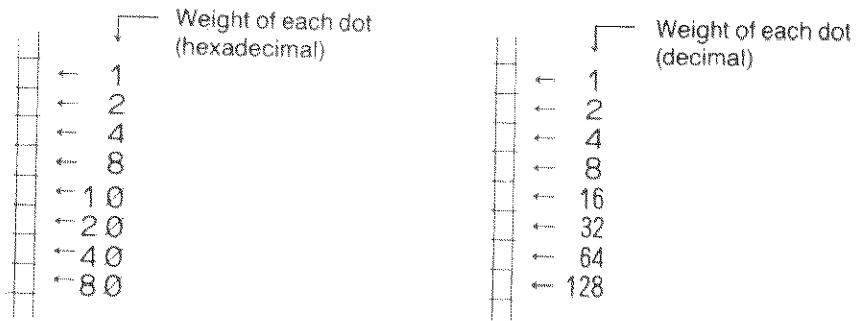
————— Represents the upper (4-dot) pattern.

————— Represents the lower (4-dot) pattern.



0 8 2 D 2 8 0 ← Upper (4-dot) pattern represented in hexadecimal.  
 1 2 1 F 1 2 1 ← Lower (4-dot) pattern represented in hexadecimal.

In format (2), a vertical 8-dot pattern is specified as a hexadecimal or decimal value. A "weight" is assigned to each dot in the vertical 8-dot pattern as shown below.



Specify the dot pattern with a numeric value with the sum of the weights of the dots to be lit on the display unit.

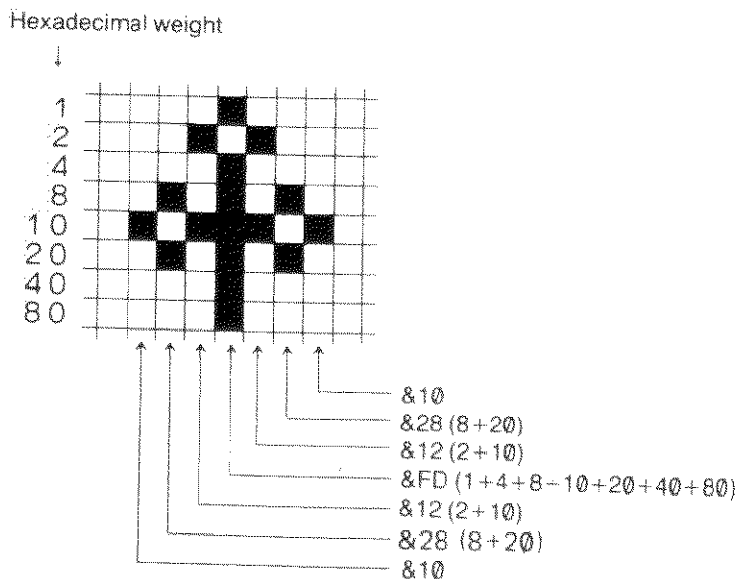
**EXAMPLE:**

Specifying dot patterns in hexadecimal.

GPRINT &10; &28; &12; &FD; &12; &28; &10

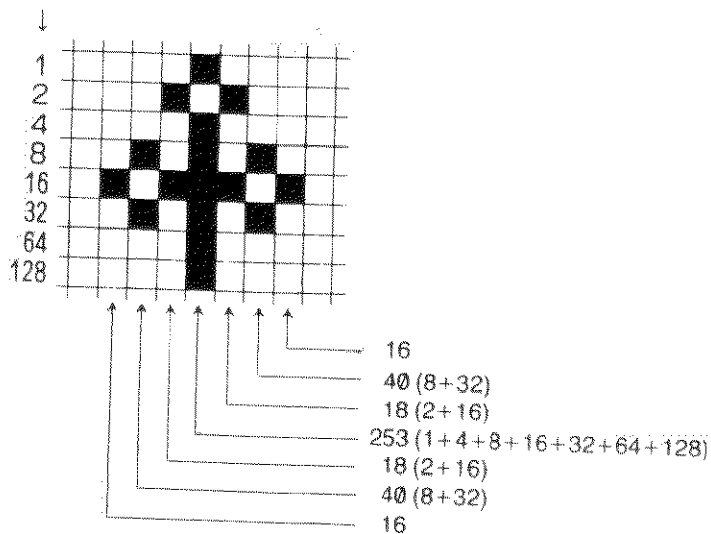
Specifying dot patterns in decimal.

GPRINT 16; 40; 18; 253; 18; 40; 16



## BASIC REFERENCE

Decimal weight

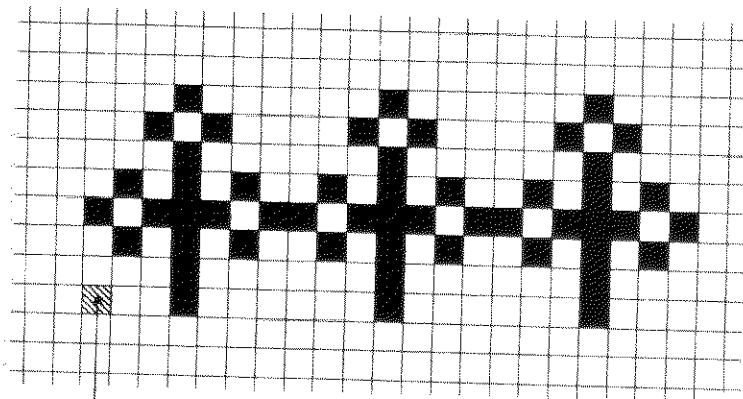


In format (3), the previous graphic display continues to be displayed.

The GCURSOR command is used to position the screen cursor before sending graphics to the screen with GPRINT. The cursor is at the bottom left corner of the the graphic pattern output by GPRINT, ie: the first dot from the bottom of the first 8 dot vertical pattern specified.

### EXAMPLE:

```
10: AA$="102812FD122810"
20: GCURSOR (60, 20)
30: GPRINT AA$;AA$;AA$
```



Specified position of the GCURSOR command (60, 20). the display of the pattern starts from this position using the upper 8 dots.

If GPRINT ends with a ";" the cursor is moved horizontally to the dot beginning in the column immediately to the right of the pattern just displayed. If GPRINT ends with ":" or nothing, the cursor position is not updated and the following GPRINT command will output starting from the same position.

---

# GRAD

---

C,V

**FORMAT:** 1. GRAD

**Abbreviation:** GR.

**See Also:** DEGREE, RADIAN

---

**PURPOSE:**

GRAD is used to change the form of angular values to gradient form.

**REMARKS:**

The PC-1360 has three forms for representing angular values—decimal degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The GRAD function changes the form for all angular values to gradient form until DEGREE or RADIAN is used. Gradient form represents angular measurement in terms of percent gradient, i.e., a 45° angle is a 50 gradient.

**EXAMPLE:**

```
10: GRAD
20: X= ASN 1
30: PRINT X
```

X now has a value of 100, i.e., a 100 gradient, the Arcsine of 1.

# GRAPH

**C,V,Gp****FORMAT:** 1. GRAPH**Abbreviation:** GRAP.**See Also:** LTEXT**PURPOSE:**

GRAPH is used to set the printer in the Graphics mode.

**REMARKS:**

When this verb is executed, the printer is released from the Text mode and put in the Graphics mode for drawing a graph.

The printer is automatically released from the Graphics mode and returns to the Text mode after the execution of the LLIST command or after printing out a manual calculation with the CE-140P connected.

To print characters in the Graphics mode, the LPRINT verb is used in either of the following two formats:

- (1) LPRINT "P character string"
- (2) LPRINT "P" { + } string variable

If you interrupt the printer operation by pressing the **BRK** key while the printer is drawing a figure in the Graphics mode, be sure to enter:

LPRINT " " **ENTER**

If you fail to do this, subsequent commands to the printer may not be executed properly.

To return the print head to its leftmost position, operate:

LTEXT **ENTER**

LPRINT **ENTER**

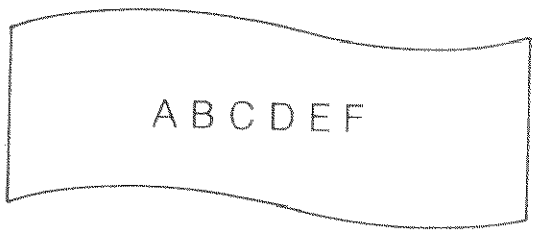
In this case, however, the printer will be released from the Graphics mode.

When any of the printer-related commands or verbs effective only in the Graphics mode (CIRCLE, CROTATE, GLCURSOR, etc.) are used in the Text mode, the printer will print characters according to the statement character.

**EXAMPLE:**

```
5: OPEN  
10: GRAPH  
20: GLCURSOR (200, -30)  
30: LPRINT "PABCDEF"  
40: LTEXT  
50: LPRINT  
60: END
```

[5] This command is required for the CE-515P.





# IF ... THEN

V

**FORMAT:** 1. IF condition THEN statement  
2. IF condition statement

**Abbreviation:** IF; T.

## **PURPOSE:**

The IF ... THEN pair is used to execute or not to execute a statement depending on conditions at the time the program is run.

## **REMARKS:**

In the normal running of BASIC programs, statements are executed in the sequence in which they occur. The IF ... THEN verb pair allows decisions to be made during execution so that a given statement is executed only when desired. When the condition part of the IF statement is true, the statement is executed; when it is false, the statement is skipped.

The condition part of the IF statement can be any relational expression as described on page 55. It is also possible to use a numeric expression as a condition, although the intent of the statement will be less clear. Any expression which evaluates to zero or a negative number is considered false; any which evaluates to a positive number is considered true.

The statement which follows the THEN may be any BASIC statement, including another IF ... THEN. If it is a LET statement, the LET verb itself must appear.

The two forms of the IF statement are identical in action, but the first form is clearer.

## **EXAMPLE:**

```
10: INPUT "CONTINUE?"; A$  
20: IF A$="YES" THEN GOTO 10  
30: IF A$="NO" THEN GOTO 60  
40: PRINT "YES OR NO, PLEASE"  
50: GOTO 10  
60: END
```

This program continues to ask 'CONTINUE?' as long as 'YES' is entered; it stops if 'NO' is entered, and complains otherwise.

---

# INKEY\$

---

V,F

**FORMAT:** 1. INKEY\$

**Abbreviation:** INK.

**See Also:**

---

**PURPOSE:**

INKEY\$ is a string pseudovariable which gives the specified variable the value of the key pressed while the INKEY\$ function is executed.

**REMARKS:**

INKEY\$ is used to respond to the pressing of individual keys without waiting for the **ENTER** key to end the input.

For key code values, see the key code chart in appendix B.

**EXAMPLE:**

```
10: A$= INKEY$
20: B= ASC A$
30: IF B=0 THEN GOTO 10
40: IF B ...
```

Lines 40 and beyond contain tests for the key and the actions to be taken.  
(For example: 40: PRINT A\$)

# INPUT

V

**FORMAT:** 1. INPUT input list

Where: input list	is: input group
	or: input group, input list
and: input group	is: var list
	or: prompt, var list
	or: prompt; var list
and: var list	is: variable
	or: variable, var list
and: prompt	is: any string constant

**Abbreviation:** 1.

**See also:** INPUT #, READ, CURSOR, PRINT

**PURPOSE:**

INPUT is used to enter one or more values from the keyboard.

**REMARKS:**

When you want to enter different values each time a program is run, use INPUT to enter these values from the keyboard.



In its simplest form the INPUT statement does not include a prompt string; instead a question mark is displayed on the left edge of the display. A value is then entered, followed by the **ENTER** key. This value is assigned to the first variable in the list. If other variables are included in the same INPUT statement, this process is repeated until the list is exhausted.

If a prompt is included in the INPUT statement, the process is exactly the same except that, instead of the question mark, the prompt string is displayed at the left edge of the display. If the prompt string is followed by a semicolon, when the statement is executed the prompt will be displayed with the cursor positioned immediately following the prompt. If the prompt string is followed by a comma, when the statement is executed, the prompt will be displayed, and then when data is entered, the data entered will appear on the screen from the position of the first character in the prompt, overwriting the prompt.

When a prompt is specified and there is more than one variable in the list following it, the second and succeeding variables are prompted with the question mark. If a second prompt is included in the list, it is displayed for the variable which immediately follows it.

When the display starting position has been specified using the CURSOR command before executing the INPUT command, the input prompt or "?" will be displayed from that position.

If the **ENTER** key is pressed and no input is provided, the variable retains the value it had before the INPUT statement.

If you make an error on keying in your value, press the CLS key. This will clear the input and move the cursor back to the start again. The cursor move keys  and  can be used too but they do not delete the bad entry first so they are more difficult to use.

### EXAMPLE:

```
10: INPUT A
20: INPUT "A=";A
30: INPUT "A=" ,A
40: INPUT "X=?";X,"Y=?";Y
```

[10] Clears the display and puts a question mark at the left edge.

[20] Displays 'A=' and waits for input data

[30] Displays 'A='. When data is input 'A=' disappears and the data is displayed starting at left edge.

[40] Displays 'X=?' and waits for first input. After **ENTER** is pressed, display is cleared and 'Y=?' is displayed at left edge.

# INPUT #

**C,V,T**

- FORMAT:** 1. INPUT # var list  
2. INPUT # "filename"; var list

Where: var list

is: variable

or: variable, var list

**Abbreviation:** I. #**See Also:** INPUT, PRINT #, READ**PURPOSE:**

INPUT # is used to enter values from the cassette tape.

**REMARKS**

The following variable types can be specified in the INPUT # statement: (1) Fixed variables—A, B, C, A(7), D\*, A(20)\*, etc., (2) Simple variables—AA, B3, CP\$, etc., (3) Array variables—S(\*), K\$(\*), etc.

## 1) Transferring data to fixed variables

To transfer data from tape to fixed variables, specify the variable names in the INPUT # statement.

```
INPUT # "DATA 1" ; A, B, X, Y
```

This statement transfers data from the cassette file named "DATA 1" to the variables, A, B, X, and Y in that order.

To fill all the available fixed variables and, if defined, extended variables (A(27) and beyond) with data transferred from tape, specify the first variable with an asterisk (\*) subscripted to it.

```
INPUT # "D-2"; D*
```

This statement transfers the contents of the tape file "D-2" to variables D through Z and to A(27) and beyond.

**INPUT # A(10)\* (without DIM declaration)**

This statement transfers the data of the first file found after the tape was started, to the variables A(10) and beyond (to J through Z and A(27) and beyond).

If an array named A is already defined by the DIM statement, it is not possible to define subscripted fixed variables in the form of A( ).

Data transfer to fixed variables and extended variables (A(27) and beyond) will continue until the end of the source data file on the tape is reached, but if the computer's memory becomes full, an error (ERROR 6) results.

**2) Data transfer to simple variables**

Data in a tape file can be transferred to simple variables by specifying the desired simple variable names in the INPUT # statement.

```
INPUT # "DM-1"; AB, Y1, XY$
```

This statement transfers data from the tape file named "DM-1" to simple variables AB, Y1 and XY\$.

Numeric data must be transferred to numeric simple variables, and character data must be simple character variables. Cross-transfer is not allowed.

Locations for simple variables must be set aside in the program data area before the INPUT # statement is executed. If not, an error will result. Use assignment statements to reserve the locations for simple variables.

```
AA=0 ENTER  
B1$="A" ENTER  
INPUT # AA, B1$ ENTER
```

Use appropriate numeric values or characters in assignment statements to reserve locations for variables.

**3) Data transfer to array variables**

To transfer data from a tape file to array variables, specify the array name in the INPUT # statement in the form of array name(\*).

```
50 DIM B(5)
60 INPUT # "DS-4"; B(*)
```

This statement transfers data from the tape file named "DS-4" to the variables (B(0) through B(5)) in array B.

Numeric data must be transferred to numeric array variables with the same length as that of the data, character data must be transferred to character array variables with the same length as that of the data. If this rule is not observed, an error will result.

Locations for array variables must be set aside in the program data area before the INPUT # statement is executed. If not, an error will result. Use the DIM statement to define the array in advance.

#### —CAUTION—

If the number of variables specified in the INPUT # statement does not agree with the amount of data recorded on the tape, the following will happen:

- If the number of pieces of data recorded on the tape file (to be transferred) is greater than the number of specified variables, data transfer will be performed to the last variable, and the remaining data will be ignored.
- If the number of pieces of data recorded in the tape file (to be transferred) is smaller than the number of specified variables, all the file data will be transferred to the variables to the end of the file, and the remaining variables will maintain their previous contents. In this case, however, the computer will continue to wait for data transfer from the tape. To halt this state, you should operate the **BRK** key.
- If the INPUT # statement is executed with no variable name specified, an error (ERROR 1) will result.

# INPUT#1

**C,V,S**

**FORMAT:** 1. INPUT#1 variable, variable, variable...

**Abbreviation:** I.#1

**See Also:** OPEN, PRINT#1

**PURPOSE:**

Assigns data, input through the serial I/O interface (terminal), to the specified variables.

**REMARKS:**

This command is valid only when the circuit of the serial I/O interface is open (after the OPEN command is executed) and is ignored otherwise.

INPUT#1 assigns data to variables in the same way as the PRINT#1 command. See the PRINT#1 command for details.

**EXAMPLE:**

INPUT#1A, AB, C\$, E(\*)

Data input through the I/O interface is assigned to variables A, AB, and C\$, and array variable E( ).

Be sure that the type of both the specified variables and the input data match (i.e. character or numeric types).

In the ASCII code system, if a character is assigned to a numeric variable, its value becomes 0. If a number is assigned to a character variable, its contents become a character string. Therefore, if the type of both the specified variable and the input data do not match, unexpected values may result.

For example, "SIN 30" assigned to a numeric variable will store 0 in the variable, and "10+40" will be stored as just 10 since anything after the operator symbol is ignored. CR (control code &0D) or NULL (&00) cause any following input data to be ignored.



Simple variables and array variables must be allocated in the program/data area before executing an INPUT#1 command. An error will result if these variables are not allocated.

# INT

C,V,F

**FORMAT:** 1. INT(X)

**Abbreviation:**

**See Also:**

**PURPOSE:**

Truncates the decimal part of a real number and returns an integer.

**REMARKS:**

The value X is rounded down to an integer. The integer result is always less than or equal to X regardless of whether X is positive or negative. For negative values, the absolute integer result will thus be greater than or equal to the absolute value.

**EXAMPLE:**

```
5: WAIT 60
10: A=-3.3: PRINT A, INT (A)
20: B=-1.9: PRINT B, INT (B)
30: C=-0.5: PRINT C, INT (C)
40: D=0.2: PRINT D, INT (D)
50: E=1.6: PRINT E, INT (E)
60: F=3: PRINT F, INT (F)
```

RUN

-3.3	-4.
-1.9	-2.
-0.5	-1.
0.2	0.
1.6	1.
3.	3.

---

# LEFT\$

---

**C,V,F**

**FORMAT:** 1. LEFT\$(X\$,N)  
2. LEFT\$("string", N)

**Abbreviation:** LEF.

**See Also:** MID\$, RIGHTS\$

---

**PURPOSE:**

Returns N characters from the left end of any string, X\$.

**REMARKS:**

The value of N must be in the range 0 to 80. Fractions will be rounded down (truncated). If N is less than 1, a null string is returned. If N is greater than the number of characters in X\$, the whole string is returned.

**EXAMPLE:**

```
10: X$="SHARP"  
20: FOR N=1 TO 6  
30: LET S$= LEFT$ (X$,N)  
40: PRINT S$  
50: NEXT N
```

```
RUN  
S  
SH  
SHA  
SHAR  
SHARP  
SHARP
```

---

# LEN

**C,V,F**

---

**FORMAT:** 1. LEN(X\$)  
2. LEN("string")

**Abbreviation:**

**See Also:**

---

**PURPOSE:**

Returns the number of characters in string X\$.

**REMARKS:**

The number of characters in the string includes any blanks or non-printing characters such as control codes or carriage returns.

**EXAMPLE:**

```
10: INPUT "ENTER A WORD ";A$
20: N= LEN A$
30: PRINT "THE WORD LENGTH IS ";N
40: END
```

```
RUN
ENTER A WORD CHERRY
THE WORD LENGTH IS 6.
```

[10] Inputs a word. In this example, the user enters "CHERRY".  
[20] Finds the length of the word.  
[30] Prints out the answer.

# LET

V

**FORMAT:** 1. LET variable=expression  
2. variable=expression

**Abbreviation:** LE.

**See Also:** IF ~ THEN

**PURPOSE:**

LET is used to assign a value to a variable.

**REMARKS:**

LET assigns the value of the expression to the designated variable. The type of the expression must match that of the variable, i.e. only numeric expressions can be assigned to numeric variables and only string expressions can be assigned to string variables. In order to convert from one type to the other, one of the explicit type conversion functions, STR\$ or VAL, must be used.

The LET verb may be omitted in all LET statements except those which appear in the THEN clause of an IF ... THEN statement.

**EXAMPLE:**

```
10: I=10  
20: A=5*I  
30: X$= STR$(A)  
40: IF I>=10 THEN LET Y$=X$+".00"
```

[10] Assigns the value 10 to I.  
[20] Assigns the value 50 to A.  
[30] Assigns the value '50' to X\$  
[40] Assigns the value '50.00' to Y\$.

**LF****C,V,Gp**

---

**FORMAT:** 1. LF  
2. LF expression

**Abbreviation:**

**See Also:**

---

**PURPOSE:**

LF is used to feed the printing paper.

**REMARKS:**

This verb is effective only in the Text mode. With format 1, the printer feeds the paper by one line. With format 2, the printer feeds the paper the paper by the specified number of lines. The value of the expression must be within the range of -999 to 999.

If the value of the expression is a positive value, the paper is fed in the forward direction. If a negative value is given, the paper is fed in the reverse direction.

The line spacing when the LF verb is executed will be the same as that specified by CSIZE.

**EXAMPLE:**

LF 10

Feeds the paper by 10 lines.

# LINE

**C,V,Gs**

**FORMAT:** 1. LINE (expression 1, expression 2)–(expression 3, expression 4),  

$$\left. \begin{array}{c} \text{S} \\ \text{R} \\ \text{X} \end{array} \right\}, \text{expression 5}, \left. \begin{array}{c} \text{B} \\ \text{BF} \end{array} \right\}$$
(A) (B)  
(C) (D) (E)

**Abbreviation:** LIN.

**See Also:** GCURSOR, PSET

**PURPOSE:**

Draws a line between 2 specified points.

**REMARKS:**

A line is drawn between the 2 points specified by (expression 1, expression 2) and (expression 3, expression 4).

**EXAMPLE:**

LINE (0, 0)–(149, 31)

A line from the top left to the bottom right of the display (screen) is drawn.

The values of expression 1–expression 4 in terms (A) and (B) can be specified within the range of –32768 to +32767, though to be specified within the screen, expression 1 and expression 3 must be within the range of 0–149, and expression 2 and expression 4, the range of 0–31.

No errors are generated if values are within the machine limits of –32768 and 32767, though only those portions of the line crossing the screen will be displayed. Specifying beyond machine limits generates error 3.

Terms (A) (expression 1, expression 2) can be omitted. If omitted, the line is drawn from either position (0, 0) or the position specified by term (B) (expression 3, expression 4) in a LINE command executed directly before.

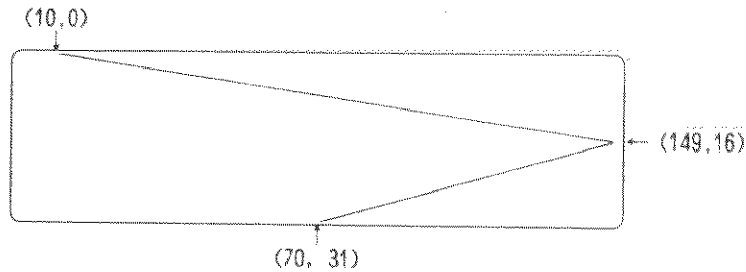
**EXAMPLE:**

```

5: CLS : WAIT 0
10: LINE (10, 0)–(149, 16)
20: WAIT : LINE –(70, 31)

```

## BASIC REFERENCE



As each line has to be broken down to dots on the screen, diagonal lines may not appear straight.

Term (C) determines the dot type:

S: Draws the line with the dots lighted. (Sets the dots.)

R: Draws the line with the dots cleared. This is used to draw a line in an area where the surrounding dots are lighted or to clear an existing line. (Resets the dots.)

X: Draws the line and clears the dots if already lit or lights the dots if not lit. (Reverses the dots.)

If none of S, R, or X is specified, S is assumed.

Term (D) specifies the nature of the line, whether continuous or dashed, by defining a bit pattern sequence.

For example, when the value of expression 5 is 26214 (&6666), the following type of line will be drawn.



16 dots

A line is drawn by repeating the pattern shown on the left.

The number 26214 (&6666) can be expressed as a binary number as follows:

0110011001100110

If the 16 dots of the line shown in the figure above and the binary number are compared, it can be seen that the dots corresponding to the 1's are lit and the dots corresponding to the 0's are cleared. In this manner, the type of line is specified by the 0's and 1's after converting the value of expression 5 into a 16 digit binary number. Therefore, the line does not appear on the screen when the value of expression 5 is 0 and a solid line appears when the value is 65535 (&FFFF). A solid line is also displayed if expression 5 is omitted. However, if R is specified in term (C), the opposite occurs, and if X is specified, the dots corresponding to the 1's are reversed.



The value of expression 5 can be specified in the range of 0-65535 (&FFFF).

Term (E) draws a square, the diagonal of which is a line connecting 2 points specified by terms (A) and (B).

B: Draws a square.

BF: Draws a square filled in with lines.

**EXAMPLE:**

```
10: CLS : WAIT 0
20: AA$="102812FD122810"
30: GCURSOR (64, 20)
40: GPRINT AA$; AA$; AA$
50: LINE (24, 0)-(124, 31), &F18F, B
60: LINE (34, 3)-(114, 28), X, BF
70: GOTO 60
```

# LIST

**C**

---

- FORMAT:**
1. LIST
  2. LIST line number
  3. LIST "label"

**Abbreviation:** L.

**See Also:** LLIST

---

**PURPOSE:**

The LIST command is used to display a program.

**REMARKS:**

The LIST command may only be used in the PROgram mode. With format (1), the program is displayed from its first line until the display is full. With format (2), the program is displayed from the line of the specified line number until the display is full.

If the line for the specified number does not exist, the program will be displayed from the line with the next largest number which does exist.

With format (3), the program is displayed from the line written with the specified label until the display is full.

When programs are merged with the MERGE command, the LIST command functions for the last program.

However, if the label specified in format (3) does not exist in the last program, it is searched for in squence from the first program. If the specified label is found, the line containing it is displayed. If a password has been set the LIST command is ignored.

**EXAMPLE:**

```
LIST 100
```

Displays line number 100.

# LLINE

**C,V,Gp**

**FORMAT:** 1. LLINE [(expression 1, expression 2)]-(expression 3, expression 4) [,expression 5][,expression 6][,B]

**Abbreviation:** LLIN.

**See Also:** RLINE, PAINT, COLOR

**PURPOSE:**

The LLINE verb is used to draw a line between two specified points.

**REMARKS:**

This verb is effective only in the Graphics mode and is used to draw a line from the point of coordinates specified by (expression 1, expression 2) to the point of coordinates specified by (expression 3, expression 4).

(expression 1, expression 2) may be omitted. If omitted, a line is drawn from the current position of the pen to the point specified by (expression 3, expression 4).

Expression 5 is used to specify one of the following types of lines by giving a value (0 to 15) to the expression. The default value of expression 5 is 0.

0	_____
1	.....
2	.....
3	.....
4	.....
5	.....
6	.....
7	.....
8	.....
9	.....
10	.....
11	.....
12	.....
13	.....
14	.....
15	_____

## BASIC REFERENCE

Expression 6 is used to specify the color of the line to be drawn. The value of expression 6 may be within the range of 0 to 7 in the extended Color mode and 0 to 3 in the normal Color mode. (Refer to the COLOR verb for the color specified by each value.) Expression 6 may be omitted. If omitted, the previous value is assumed.

If B is specified at the end of the LLINE verb, a rectangle is drawn using the point of coordinates specified by (expression 1, expression 2) and the point of coordinates specified by (expression 3, expression 4) as the end points of a diagonal line.

Example: LLINE (10,20)–(200,–20), 2,0,B

### Note:

(expression 1, expression 2) cannot be omitted when drawing a rectangle.

When LLINE or RLINE without B is executed after the execution of these commands with B, following command should be executed immediately after the execution of these commands without B;

```
POKE &FB20,0
```

If this command is not executed the computer may draw rectangle by LLINE or RLINE without B;

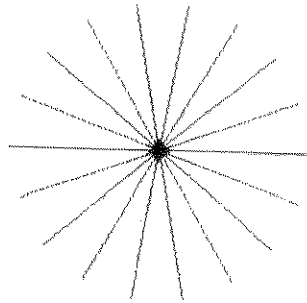
The LLINE verb in the following format allows the printer to draw lines continuously.

```
LLINE (expression 1, expression 2) – (expression 3, expression 4)  
      – (expression 3, expression 4) – (expression 3, expression 4)  
      – (expression 3, expression 4) – (expression 3, expression 4),  
      expression 5, expression 6
```

With the above format, a line connecting the point specified by (expression 1, expression 2) to the point specified by (expression 3, expression 4), a line connecting the point specified by (expression 3, expression 4), to the point specified by the next pair ... can be drawn continuously. A maximum of six pairs can be specified.

**EXAMPLE:**

```
5: OPEN
10: GRAPH : RANDOM
20: GLCURSOR (240, -120)
25: SORGN
30: FOR J=0 TO 340 STEP 20
40: A=107* COS J
50: B=107* SIN J
60: R= RND 4-1
70: LLINE (0, 0)-(A,B), 0,R
80: NEXT J
90: LTEXT
100: LPRINT
110: END
```



- [5] This command is required for the CE-515P.
- [25] Moves the pen to to about the center of the paper and designates it as the origin of coordinates for drawing a figure.
- [60] Random number 0 to 3 is assigned to R.
- [70] Color is specified by the value of R.
- [90] The printer returns to the Text mode and moves the print head to its leftmost position.

# LLIST

OUTPUT VIA THE PRINTER INTERFACE: **C,P**

---

LLIST can be used to output program lines directly to the printer via the printer interface, or output via the serial I/O interface.

- FORMAT:**
1. LLIST
  2. LLIST expression
  3. LLIST expression 1, expression 2
  4. LLIST expression,
  5. LLIST, expression

**Abbreviation:** LL.

**See Also:** LIST

---

**PURPOSE:**

The LLIST command is used for printing a program on the optional CE-126P or CE-140P.

**REMARKS:**

Note that although the CE-140P connects to the serial I/O interface, it operates as if connected to the printer interface.

When the serial I/O interface is open due to the OPEN command, the LLIST command outputs the program at the serial I/O interface terminal. To return the program printing command to the printer, execute the CLOSE command.

The LLIST command may be used in the PROgram or RUN mode.

The first form prints all of the programs in memory.

The second form prints only the program line whose line number is given by the expression.

The third form prints the statements from the line number with the nearest line equal to or greater than the value of expression 1 to the nearest line equal to or greater than the value of expression 2. There must be at least two lines between the two numbers.

The forth form prints all program lines beginning with the line whose number is given by the expression.

The fifth form prints all program lines up to, and including, the line whose number is given by the expression.

When programs are merged with the MERGE command, the LLIST command functions for the last program. To list a program stored earlier, execute.

```
LLIST "label",
```

If a password has been set the LLIST command is ignored.

**EXAMPLE:**

```
LLIST 100,200
```

Lists the statements between line numbers 100 and 200.

# LLIST

OUTPUT VIA THE SERIAL I/O INTERFACE: **C,V,S**

**FORMAT:** 1. LLIST

2. LLIST { expression }  
          { "label" }

3. LLIST expression 1, expression 2

**Abbreviation:** LL

**See Also:** OPEN, CONSOLE

---

**PURPOSE:**

Sends the program contents out of the serial I/O interface (terminal).

**REMARKS:**

The LLIST command is valid under manual operation in the PRO or RUN mode.

When the circuit of the serial I/O interface is open due to the OPEN command, the program is sent out in ASCII code.

When the circuit is closed, the program is printed on the printer. This is true for the CE-140P as well as the CE-126P printer. Note that although connected to the serial I/O interface, the CE-140P operates as if connected to the printer interface.

In format (1), all programs in the PC-1360 are sent out.

For example, when the program below is in the PC-1360, pressing

LLIST **ENTER**

sends out the program in the form shown below.



```

10: OPEN
100: REM  **ABC-12**
65279: END

```

Space	1	0	:	O	P	E	N	Space	CR
1	0	0	:	R	E	M	Space	Space	*
*	A	B	C	-	1	2	*	*	CR
6	5	2	7	9	:	E	N	D	Space
CR									

Note: CR is an end code. It is either LF or CR + LF depending on the setting of the OPEN command.

In format (2), the line indicated by the value of the expression or the line with the specified label is sent out.

In format (3), the program, from the line indicated by the value of expression 1 to the line indicated by the value of expression 2, is sent out. (Labels can also be used for expression 1 and expression 2.)

Expression 1 or expression 2 can be omitted in format (3).

If expression 1 is omitted, the program, from the first line to the line indicated by the value of expression 2, is sent out.

If expression 2 is omitted, the program, from the line indicated by the value of expression 1 to the last line, is sent out.

If a line corresponding to the value of expression, expression 1 or expression 2 does not exist, the line with the next largest number which does exist will be specified.

An error results (ERROR 1) if the lines specified in expression 1 and expression 2 are the same.

The LLIST command is ignored if a password has been set.

If programs have been merged using the MERGE command, the LLIST command functions only for the last merged program.

## BASIC REFERENCE

To list the previously stored programs, execute

```
LLIST "label",
```

The number of print columns per line is set by the `CONSOLE` command. If set to 23 columns or less, executing the `LLIST` command results in an error (ERROR 3).

---

# LN

**C,V,F**

---

**FORMAT:** 1. LN(X)

**Abbreviation:**

**See Also:** EXP, LOG

---

**PURPOSE:**

Returns the natural logarithm (to base e) of the expression X.

**REMARKS:**

The value specified for X must be greater than zero. To find the antilog of a number, use the EXP function.

**EXAMPLE:**

```
10: CLS
20: INPUT "TYPE IN A NUMBER ",X
30: PRINT "THE NATURAL LOGARITHM OF ";X;" IS "; LN(X)
40: INPUT "USE AGAIN? Y/N ",A$
50: IF A$="Y" THEN 20
60: END
```

# LOAD

C

**FORMAT:** 1. LOAD

**Abbreviation:** LOA.

**See Also:** OPEN, CLOAD, SAVE

**PURPOSE:**

Loads the data sent from the serial I/O interface (terminal) into the program/data area.

**REMARKS:**

The LOAD command is valid when the circuit of the serial I/O interface has been opened by the OPEN command. It is ignored when the circuit is closed.

Data through the serial I/O interface is read until the end code is reached. Each section of data up to the end code is treated as a single program line.

The PC-1360 converts the data into a form which can be stored as a program and then transfers (writes) it to the program/data area. This process repeats for each program line until the text end code is read (see the OPEN command.)

Up to 256 bytes of data can be read at a time. Exceeding this limit generates an error. Errors are also generated if a single program line exceeds 80 bytes or does not begin with a numeric value for the line number.

Lines are not rearranged according to line number.

Execution of the LOAD command ends when the text end code is read (from the sending side).

Even if the sending side sends out the entire program, the PC-1360 does not end execution as long as the text and code is not read. In this case, end execution as follows:

- (1) After the sending side sends the program, have it also send only the text end code.

(2) Or, press the **BRK** key

The reserved contents cannot be read from the serial I/O interface.

---

# LOG

---

C,V,F

**FORMAT:** 1. LOG(X)

**Abbreviation:** LO.

**See Also:** LN

---

**PURPOSE:**

Returns the common logarithm (to base 10) of the expression X.

**REMARKS:**

To obtain a logarithm to a base other than 10, use the following conversion formula: Log to base B of X = LOG(X)/LOG(B).

To obtain the antilog of a common logarithm, raise 10 to the power of the logarithm using the “^” operator.

**EXAMPLE:**

LOG(2)

3.010299957E-01

---

# LPRINT

 OUTPUT VIA THE PRINTER INTERFACE: **C,V,P**

---

LPRINT can be used to output program lines directly to the printer via the printer interface, or output via the serial I/O interface.

- FORMAT:**
1. LPRINT print expr
  2. LPRINT print expr, print expr, ..., print expr
  3. LPRINT print list
  4. LPRINT print list;
  5. LPRINT

Where: print list	is: print expr
	or: print expr; print list
and: print expr	is: expression
	or: USING clause; expression

The USING clause is described separately under USING

**Abbreviations:** LP.

**See also:** PAUSE, PRINT, USING, WAIT

---

**PURPOSE:**

LPRINT is used to print out information to an attached printer (CE-126P or CE-140P).

**REMARKS:**

The default condition is output to the attached printer. If the OPEN command has opened the serial I/O port, output is via the serial I/O interface instead. Using CLOSE to close the output channel resets printer output to the attached printer.

Note that although the CE-140P printer connects to the serial I/O interface, it operates as if connected to the printer interface (so that the output channel should be closed). If both CE-126P and CE-140P printers are connected at the same time, printing executes on the CE-126P while graphics is sent to the CE-140P.

The LPRINT verb is used to print prompting information, results of calculations, etc. The first form of the LPRINT statement prints a single value. If the expression is numeric, the value will be printed at the far right edge of the paper. If it is a string expression, the print is made starting at the far left.

In format (2), the number of print columns is delimited into groups of 12 columns. The specified values are printed in sequence. In other words, the first specified value is printed on the left side of the first line, the second specified value on the right side of the first line, the third specified value on the left side of the second line, and the fourth specified value on the right side of the second line.

Numeric values are printed right justified and text is printed left justified in the columns. If the output length exceeds the column width available, least significant digits are truncated for numeric output and trailing characters for text output are truncated.

The number of the values (items) specified in format (2) must be within 2—8.

Even if the display starting position has been specified in format (2) with format (4) or the CURSOR command, the specification will be cleared and printing will be performed in the form as shown above. The values are printed from the left edge of the paper.

In format (3), the values are printed from the left edge of the paper. If the value to be printed exceeds 24 columns, a newline is automatically performed. Up to a maximum of 96 characters can be printed. An error occurs if the 96th column is in the middle of a numeric value.

In format (4), at the specified printing value, the value specified in the LPRINT command to be executed next will be printed in succession. However, due to the structure of the printer, printing occurs in the following instances:

- (1) when the value to be printed exceeds 24 columns.
- (2) when an LPRINT command not ending with a ";" has been executed.
- (3) when the program execution ends.

In format (5), no printing occurs but the paper is fed one line.



**EXAMPLE:**

```
10: A=10:B=20:X$="ABCDE":Y$="XYZ"  
20: LPRINT A  
30: LPRINT X$  
40: LPRINT A,B,X$,Y$  
50: LPRINT X$;A;B  
60: LPRINT  
70: LPRINT A*B;  
80: LPRINT Y$
```

# LPRINT

OUTPUT VIA THE SERIAL I/O INTERFACE: **C,V,S**

- FORMAT:**
1. LPRINT  $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\}$
  2. LPRINT  $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\}, \left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\} \dots,$   
 $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\}$
  3. LPRINT  $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\}; \left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\} \dots;$   
 $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\}$
  4. LPRINT  $\left\{ \begin{array}{l} \text{expression} \\ \text{character string} \end{array} \right\};$
- (Format where a ";" is added to the end of 1 and 3 above.)
5. LPRINT

**Abbreviation:** LP.

**See also:** OPEN, CONSOLE, USING

## **PURPOSE:**

Sends the specified information out through the serial I/O interface (terminal).

## **REMARKS:**

When the circuit of the serial I/O interface is opened by the OPEN command, the specified information is sent out through the serial I/O terminal in ASCII code.

When the circuit is closed, LPRINT prints information on the printer CE-126P or CE-140P.

Although the CE-140P printer connects to the serial I/O interface, it operates as if connected to the printer interface.

In format (1), the value of the expression or the character string is sent from its beginning.

If the value of the expression is negative a “-” sign is sent before the value. If positive, a space is sent.

In format (2), 12 column (digit) divisions are automatically set. The value of a single expression or a character string is sent within a range of 12 columns (digits).

**EXAMPLE:**

```
10: OPEN "1200, N, 8, 1, A, C"
20: CONSOLE 36
30: LPRINT 12345, "ABCDE",
    -7/5, 1.23456789E12
```

Executing this program sends information in the following form.

Sending direction ←

```
12345. ABCDE1.4CR 1.23456E12CR
```

← 12 columns → ← 12 columns → ← 12 columns → ← 12 columns →

After all the data is sent, the end code is sent.

After the data corresponding to the number of columns specified in the CONSOLE command is sent, the end code is sent.

If the specified character string exceeds 12 columns in this format, only the first 12 characters are sent. Also, if the value of the expression exceeds 12 digits (in exponential display), it is sent after the low order digits of the fractional part are truncated.

If the value of the expression is negative, a “-” signal is sent before the value. If positive, a space is sent.

In format (3), the specified values or strings are sent in the specified sequence. In this format, a space is not sent before a positive number.

Example:

```
50 LPRINT -123; "ABC"; 567.89
```

← Sending direction

```
-123.ABC567.89CR
```

#### BASIC REFERENCE

End code (LF or CR + LF is sent depending on how the OPEN command is specified.)

In format (4), the end code which indicates the end of the data is not sent. Instead, the end code is sent after the data corresponding to the number of columns specified by the CONSOLE command has been sent.

In format (5), only the end code is sent.

When the format has been specified in the USING command, formats (1)—(4) send data accordingly.

Executing PRINT=LPRINT causes the PRINT command to function as the LPRINT command. Specifying PRINT = LPRINT is valid only if it is executed when the printer is connected or when the circuit of the serial I/O interface has been opened by the OPEN command.

To send characters or control codes which cannot be directly entered through the keyboard, specify them using the CHR\$ command as shown below.

To send [ ]:

```
(1) 50 LPRINT CHR$ &5B, CHR$ &5D
```

```
(2) 50 A$= CHR$ &5B: B$=CHR$ &5D  
60 LPRINT A$, B$
```

NULL (&00) is valid only in (1) and will be ignored in (2).

---

# LTEXT

**C,V,Gp**

---

**FORMAT:** 1. LTEXT

**Abbreviation:** LT.

**See Also:** GRAPH

---

**PURPOSE:**

LTEXT is used to set the Text mode.

**REMARKS:**

This verb is used to put the printer in the Text mode for printing alphabetic and numeric characters.

The printer is automatically put in the Text mode after the execution of the LLIST verb or after the printing by manual operation with the CE-140P connected. Be sure to set the operation mode with either the GRAPH or LTEXT verb when the **BRK** key is pressed after the automatic power off of the computer or when the power switch of either the computer or the printer is turned on again from the OFF position.

**EXAMPLE:**

LTEXT

Sets the Text mode.

# MDF

**C,V,F**

**FORMAT:** 1. MDF (expression)  
2. MDF (expression, threshold number)

**Abbreviation:** MD.

**See Also:** USING

**PURPOSE:**

The MDF verb is used to round up the value of an expression.

**REMARKS:**

The MDF is a function used to round the value of an expression to the number of decimal places specified by the USING command.

This verb is effective only when the number of decimal places is specified for a value by the USING command.

Format 1 uses the standard default threshold number of 4. This means that if the first digit of the truncated part is more than 4, one is carried to the non-truncated part. This threshold number can be specified in format 2.

**EXAMPLE:**

	<u>Display</u>
USING "###.###" MDF (0.5/9)	0.056
USING "###.###" MDF (0.5/9,5)	0.055

```

10: USING "###.###"
20: A = MDF (5/9)
30: PRINT A
40: USING
50: PRINT A, 5/9
60: END

```

BASIC REFERENCE

RUN **ENTER**

**0.556**

**ENTER**

**0.556 5.55555E-01**

**MEM****C,V,F**

---

**FORMAT:** 1. MEM**Abbreviation:** M.**See Also:** MEM\$, SET MEM

---

**PURPOSE:**

Returns the byte count of the available area for program and data storage in memory.

**REMARKS:**

MEM returns the available programming area in memory. This value will depend on the use of RAM cards in slots 1 and 2 (see MEM\$ and SET MEM.)

Note that in the case of "D" being set by SET MEM, the whole area indicated for both RAM cards may not be fully available for the program and data. This is because program and variable storage areas are restricted to the respective RAM cards. If either memory is exhausted, the remaining free area in the other card cannot be used.



---

# MEM\$

**C,V,F**

---

**FORMAT:** 1. MEM\$

**Abbreviation:** M.\$

**See Also:** MEM, SET MEM

---

**PURPOSE:**

Indicates which RAM slots are currently being used.

**REMARKS:**

MEM\$ is entered without any parameters and returns the value "B", "C" or "D". "C" indicates that only the RAM card in slot 1 is in use. "B" and "D" indicate that both RAM cards, and hence both slots 1 and 2, are in use. "B" and "D" indicate different ways of using the two slots together (see SET MEM.)

# MERGE

C,T

**FORMAT:** 1. MERGE  
2. MERGE "filename"  
(effective in PROgram or RUN mode)

**Abbreviation:** MER.

**See Also:** CLOAD

---

**PURPOSE:**

The MERGE command is used to load a program saved on cassette tape and merge it with the program existing in memory.

**REMARKS:**

The MERGE command retains the program already stored in the PC-1360 and then loads a program recorded on the tape. Therefore, several different programs can be stored in the PC-1360 at the same time. Merged programs can have the same line numbers.

If there are 2 or more programs in memory, the RUN and GOTO commands will always execute the last program to be merged unless program labels are used. A previously merged program is thus executed by using the RUN or GOTO command, or the DEF key, together with the label. Notice that there is no way to edit previously merged programs so that labels should be assigned before programs are merged. Should program labels be the same, the latest program to be merged is executed.

**Merging password protected programs**

When loading programs with passwords (password protected programs) using the MERGE command, the handling of the programs differ as follows depending on whether the programs within the computer are protected.

When protected:

Password protected programs cannot be loaded.

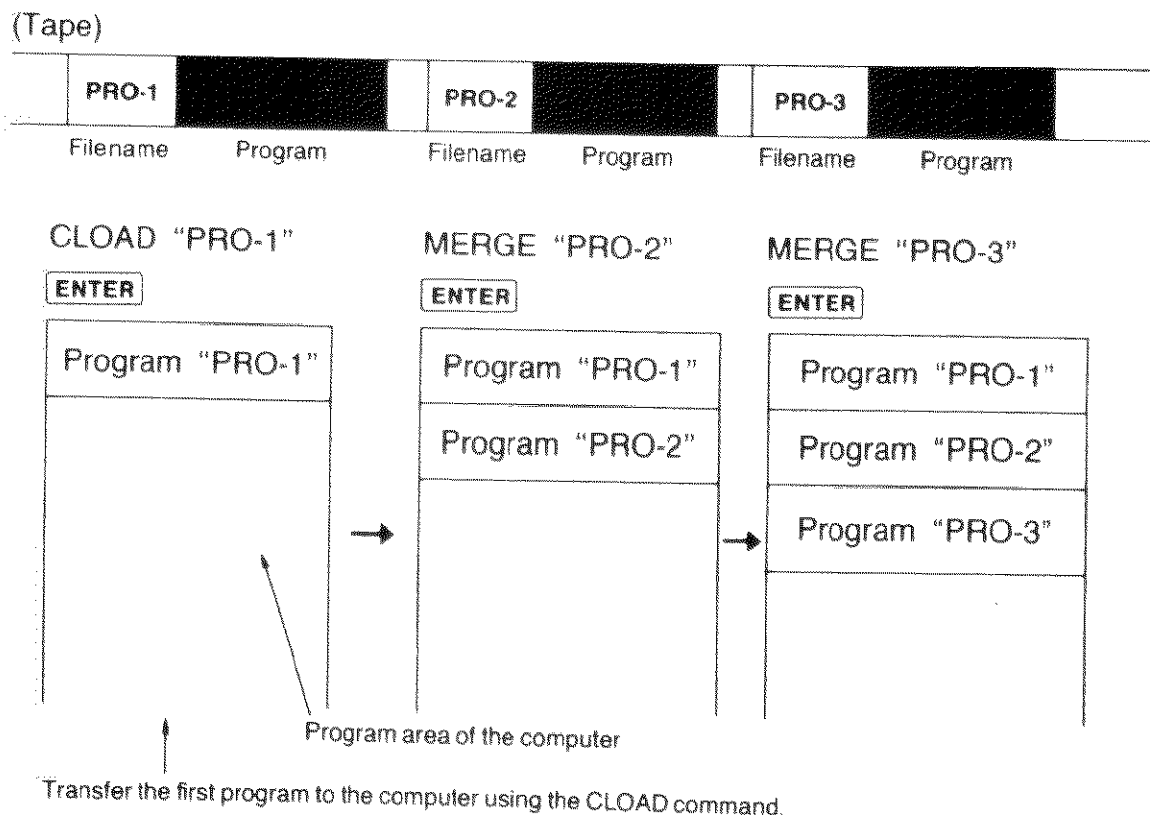
When not protected:

If password protected programs are loaded using the MERGE command, all programs within the computer become protected.

When the programs within the computer are protected, even programs without passwords become password protected when loaded using the MERGE command.

**EXAMPLE:**

When programs with file names PRO-1, PRO-2 and PRO-3 are to be stored, PRO-1 is stored using the CLOAD command, and PRO-2 and PRO-3 are transferred to the computer using the MERGE command. The state of the storage is as follows.



Programs loaded using the MERGE command are stored as in the example. The programs are handled as follows by their line numbers.

If the first line number of the program loaded using the MERGE command is larger than the last line number of the previously loaded program, the two programs are considered to be a single program.

If the first line number of the program loaded using the MERGE command is smaller than the last line number of the previously loaded program, the two programs are considered separate.

In the example above, where the line numbers for programs PRO-1, PRO-2, and PRO-3 are 10-200, 50-150, and 160-300 respectively, PRO-1 and PRO-2 are considered separate. PRO-2 and PRO-3 are considered to be a single program with line numbers 50-300.

### Executing merged programs

"A" PRO-1
"B" PRO-2
"C" PRO-3

The figure shows the memory when PRO-1 is loaded after which PRO-2 and PRO-3 are loaded using the MERGE command. If a program is started using RUN or GOTO (RUN expression or GOTO expression), PRO-3 will be executed. On the other hand, if started using RUN "label", GOTO "label", or a defined key, the specified label is searched for from the beginning of PRO-3 within the computer.

If not found in PRO-3, the search continues in the next most recently merged program, PRO-2.

---

# MID\$

**C,V,F**

---

- FORMAT:**
1. MID\$(X\$,N,M)
  2. MID\$("string",N,M)

**Abbreviation:** MI.

**See Also:** LEFT\$, RIGHT\$

---

**PURPOSE:**

Returns a string of M characters from inside string X\$ starting from the Nth character in string X\$.

**REMARKS:**

If N is less than 1 or greater than the number of characters in X\$, a null string is returned. M must be in the range 0 to 80 and N in the range 1 to 80. Fractions will be rounded down.

**EXAMPLE:**

```
10: Z$="ABCDEFGG"  
20: LET Y$= MID$ (Z$,3,4)  
30: PRINT Y$
```

```
RUN  
CDEF
```

---

# NEXT

---

V

**FORMAT:** 1. NEXT numeric variable

**Abbreviation:** N.

**See Also:** FOR

---

**PURPOSE:**

NEXT is used to mark the end of a group of statements which are being repeated in a FOR/NEXT loop.

**REMARKS:**

The use of NEXT is described under FOR. The numeric variable in a NEXT statement must match the numeric variable in the corresponding FOR.

**EXAMPLE:**

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

Print the numbers from 1 to 10 each time the **ENTER** is pressed.

---

# NEW

---

**C**

**FORMAT:** 1. NEW

**Abbreviation:**

**See Also:**

---

**PURPOSE:**

NEW is used to clear existing program or reserve memory.

**REMARKS:**

When used in the PROgram mode the NEW command clears all programs and data which are currently in memory. (programs with password cannot be cleared.)

When used in Reserve mode, NEW clears all existing reserve memory.

NEW will generate ERROR 9 if used in RUN mode.

**EXAMPLE:**

NEW

Clears program or reserve memory

---

# ON ... GOSUB

---

V

**FORMAT:** 1. ON expression GOSUB expression list  
Where: expression list is: expression  
or: expression, expression list

**Abbreviations:** O.; GOS.

**See Also:** GOSUB, GOTO, ON ... GOTO

---

**PURPOSE:**

ON ... GOSUB is used to execute one of a set of subroutines depending on the value of a control expression.

**REMARKS:**

When ON ... GOSUB is executed the expression between ON and GOSUB is evaluated and reduced to an integer. If the value of the integer is 1, the first subroutine in the list is executed as in a normal GOSUB. If the expression is 2, the second subroutine in the list is executed, and so forth. After the RETURN from the subroutine execution proceeds with the statement which follows the ON ... GOSUB.

If the expression is zero, negative, or larger than the number of subroutines provided in the list, no subroutine is executed and execution proceeds with the next line of the program.

Commas may not be used in the expressions following the GOSUB. The PC-1360 cannot distinguish between commas in expressions and commas between expressions.

**EXAMPLE**

```
10: INPUT A
20: ON A GOSUB 100, 200, 300
30: END
100: PRINT "FIRST"
110: RETURN
200: PRINT "SECOND"
210: RETURN
300: PRINT "THIRD"
310: RETURN
```

An input of 1 prints 'FIRST'; 2 prints 'SECOND'; 3 prints 'THIRD'. Any other input does not produce any print.



---

# ON ... GOTO

---

V

**FORMAT:** 1. ON expression GOTO expression list  
Where: expression list is: expression  
or: expression, expression list

**Abbreviations:** O.; G.

**See Also:** GOSUB, GOTO, ON ... GOSUB

---

**PURPOSE:**

ON ... GOTO is used to transfer control to one of a set of locations depending on the value of a control expression.

**REMARKS:**

When ON ... GOTO is executed the expression between ON and GOTO is evaluated and reduced to an integer. If the value of the integer is 1, control is transferred to the first location in the list. If the expression is 2, control is transferred to the second location in the list; and so forth.

If the expression is zero, negative, or larger than the number of locations provided in the list, execution proceeds with the next line of the program.

Commas may not be used in the expression following the GOTO. The PC-1360 cannot distinguish between commas in expressions and commas between expressions.

**EXAMPLE:**

```
10: INPUT A
20: ON A GOTO 100,200,300
30: GOTO 900
100: PRINT "FIRST"
110: GOTO 900
200: PRINT "SECOND"
210: GOTO 900
300: PRINT "THIRD"
310: GOTO 900
900: END
```

An input of 1 prints 'FIRST'; 2 prints 'SECOND'; 3 prints 'THIRD'. Any other input does not produce any print.

# OPEN

**C,V,S**

- FORMAT:** 1. OPEN "baud rate, parity, word length, stop bit, type of code, end code, text end code"  
2. OPEN

**Abbreviation:** OP.

**See Also:** CLOSE

**PURPOSE:**

Allows data to be transferred through the I/O interface. Also sets the I/O conditions.

**REMARKS:**

Format (1) enables data to be transferred through the I/O interface (serial I/O terminal). It also sets the conditions for the data transfer with the connected equipment. The conditions are specified in the following form:  
"baud rate, parity, word length, stop bit, type of code, end code, text end code"

Baud Rate:	300, 600, 1200 Specifies the modulation rate (transfer rate). For the PC-1360, 300 baud, 600 baud, or 1200 baud can be selected. (1 baud = 1 bit/sec)
Parity:	N, E, O Specifies the type of parity by a character. N: No parity bit is transmitted nor received. E: Specifies even parity. O: Specifies odd parity.
Word Length:	7, 8 Specifies how many bits to be transmitted or received per character. Either 7 or 8 bits can be specified.
Number of Stop Bits:	1, 2
Type of Code:	A Only ASCII codes can be transmitted or received. Therefore, A is always specified.

End Code:	C, F, L Specifies the type of end code to indicate the end of data (delimiting), end of a program line, etc. C: Specifies the CR (carriage return) code. F: Specifies the LF (line feed) code. L: Specifies the CR code + LF code.
Text End Code:	&00—&FF Specifies the text end code to indicate the end of the program, etc. (May be required when using the SAVE or LOAD commands.)

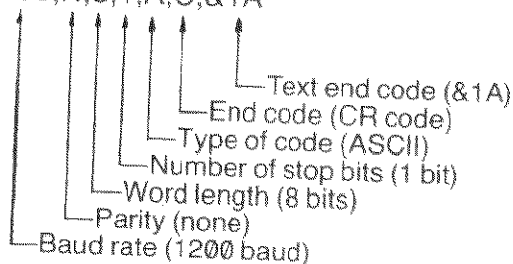
Any condition specified in the OPEN command can be omitted. If omitted, the current condition remains unchanged.

In format (2), all conditions set previously are retained. This format enables data to be transferred through the I/O interface.

Executing the OPEN command while the circuit of the I/O interface is open and ready for data transfer (due to prior execution of the OPEN command) results in an error (ERROR 8). Execute the CLOSE command to close the circuit. (The circuit also closes when the RUN command is executed, when the program ends, or when the power is switched off.) The previously set conditions are retained even after the CLOSE command is executed.

**EXAMPLE:**

OPEN "1200,N,8,1,A,C,&1A"



The conditions in the example above are set after the batteries are replaced or after the ALL RESET button is pressed.

OPEN " ,,,2"

Only the number of stop bits is changed.

# OPENS\$

C,V,F,S

---

**FORMAT:** 1. OPENS\$

**Abbreviation:** OP.\$

**See Also:** OPEN

---

**PURPOSE:**

Obtains the currently set I/O conditions.

**REMARKS:**

The currently set I/O conditions are obtained as a character string.

**EXAMPLE:**

OPENS\$ **ENTER** 1200, N, 8, 1, A, C, &1A

# PAINT

**C,V,Gp**

**FORMAT:** 1. PAINT expression 1 [,expression 2]

**Abbreviation:** PAI.

**See Also:** LLINE, RLINE, CIRCLE, COLOR

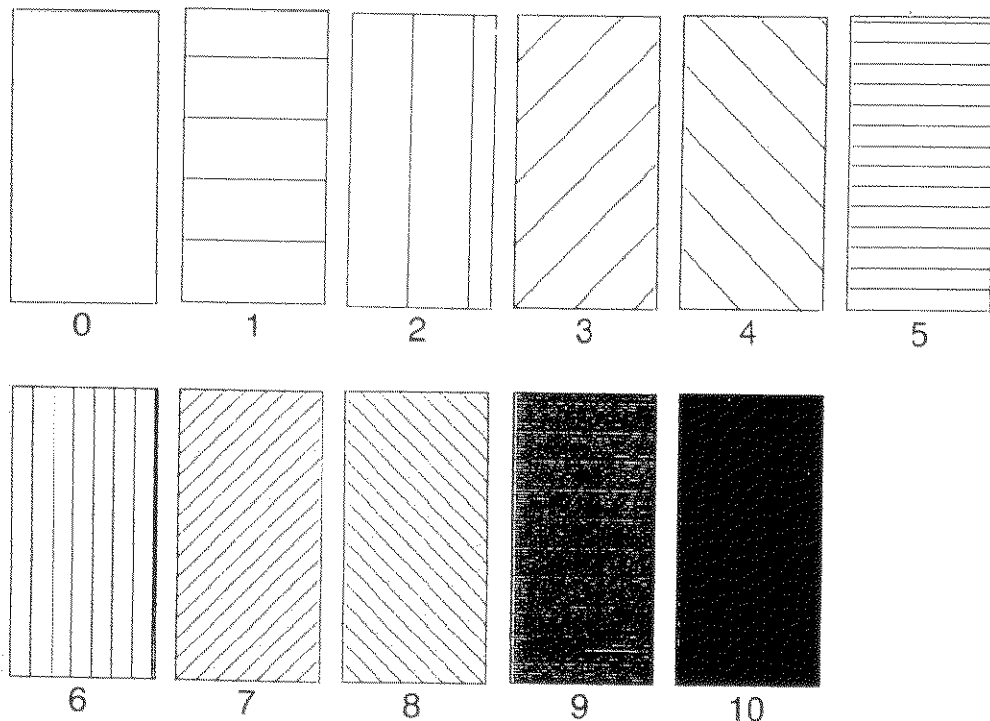
**PURPOSE:**

PAINT is used to hatch the inside of a rectangle or circle (or sector).

**REMARKS:**

This verb is effective only in the Graphics mode. When a rectangle or circle (or sector) has been drawn by the verb executed immediately before the PAINT verb, the printer hatches the inside of the rectangle or circle (or sector).

The following types of patterns can be specified for the rectangle drawn by the "B" specification of the LLINE or RLINE verb and for the circle (or sector) drawn by the CIRCLE verb by specifying a value for expression 1. For rectangles, 0 to 10 may be specified while for circles (or sectors) 1 to 6, 9 or 10 can be specified.



#### BASIC REFERENCE

Repeated use of PAINT together with LLINE allows the printer to draw a special pattern as shown below.

Expression 2 is used to specify the color of hatching. The value of expression 2 must be within the range of 0 to 7 in the extended Color mode and 0 to 3 in the normal Color mode. (Refer to COLOR for the color specified by each value.)

To execute PAINT for the circle (or sector) drawn by CIRCLE, ratio = 1 and pitch angle = 1 must have been specified as the values of expressions 7 and 8 of the CIRCLE verb.

#### **EXAMPLE:**

```
LLINE (0,0) - (200, -200), 0, 0, B  
PAINT 5, 0  
PAINT 6, 0
```

# PASS

C

**FORMAT:** 1. PASS "character string"

**Abbreviation:** PA.

**See Also:** CSAVE, CLOAD, NEW

**PURPOSE:**

The PASS command is used to set and cancel passwords.

**REMARKS:**

Passwords are used to protect programs from inspection or modification by other users. A password consists of a character string which is no more than seven characters long. The seven characters must be alphabetic or one of the following special symbols: ! # \$ % & ( ) \* + - / , . : ; < = > ? @  $\sqrt{\quad}$   $\pi$   $\wedge$

Once a PASS command has been given, the programs in memory are protected. A password protected program cannot be examined or modified in memory. It cannot be sent to tape or listed with LIST or LLIST, nor is it possible to add or delete program lines. If several programs are in memory and PASS is entered, all programs in memory are protected. The only way to remove this protection is to execute another PASS command with the same password.

When a password with 7 or more characters is declared, only the first 7 characters are valid and are used to set and remove protection.

Press **ENTER** right after the password.

Writing characters or symbols after the password results in an error and the password cannot be cancelled.

(example) PASS"ABCDEFGG":A = 123 **ENTER** → Error

**EXAMPLE:**

PASS "SECRET"

Establishes the password 'SECRET' for all programs in memory.

# PAUSE

V

- FORMAT:**
1. PAUSE print expr
  2. PAUSE print expr, print expr, ..., print expr
  3. PAUSE print list
  4. PAUSE print list;
  5. PAUSE

Where: print list is: print expr  
          or: print expr; print list  
and: print expr is: expression  
      or: USING clause; expression

The USING clause is described separately under USING

**Abbreviation:** PAU.

**See Also:** LPRINT, PRINT, CURSOR, USING, WAIT

---

**PURPOSE:**

PAUSE is used to print information on the display for a short period.

**REMARKS:**

PAUSE is used to display prompting information, results of calculations, etc. The operation of PAUSE is identical to PRINT except that after PAUSE the PC-1360 waits for short preset interval of about .85 seconds and then continues execution of the program without waiting for the ENTER key or the WAIT interval.

The first form of the PAUSE statement displays a single value. If the expression is numeric, the value is printed at the far right end of the display. If it is a string expression, the display is made starting at the far left.

However, when the display starting position is specified using format (4) or the CURSOR command, the display starts from that position.

In format (2), the display unit is divided into groups of 12 columns. The values are displayed, in sequence, from the first specified value. In this case too, within a range of 12 columns, the numeric value of an expression is displayed from the right end of the display and characters are displayed from the left side.



The number of the values (items) specified in format (2) must be within 2—8. If the specified value exceeds 12 columns, the following is performed.

- 1) When the numeric value exceeds 12 digits (when the decimal fraction in the exponential display is 8 digits or more), the least significant digits are truncated.
- 2) When the characters exceed 12 columns, only the first 12 characters (from the left) are displayed.

In format (3), the specified value is displayed continuously from the left side of the display. However, if the display starting position has been specified using format (4) or the CURSOR command, the display starts from that position.

If the value to be displayed in format (3) exceeds 96 columns, the excess portion is not displayed. When the value to be displayed exceeds 96 columns, an error (ERROR 6) occurs if the 96 column is in the middle of a numeric value.

In format (4), the specified value is displayed from the left side of the display. The column following the end of this displayed value is specified as the display starting position for display commands such as for the next PRINT command. In format (5), the previously displayed value is displayed as is.

**EXAMPLE:**

```
10: A=10:B=20:X$="ABCDEF":
   Y$="XYZ"
```

	<u>Display</u>
20: PAUSE A	<b>10.</b>
30: PAUSE X\$	<b>ABCDEF</b>
40: PAUSE X\$, Y\$, A, B	<b>ABCDEF                      XYZ</b> <span style="display: block; text-align: right;">10.    20.</span>
50: PAUSE Y\$;X\$;	<b>XYZABCDEF</b>
60: PAUSE A*B	<b>XYZABCDEF200.</b>

# PEEK

**C,V,M**

---

**FORMAT:** 1. PEEK memory address

**Abbreviation:** PE.

**See Also:** CALL, POKE

---

**PURPOSE:**

Returns the memory contents at the specified location.

**REMARKS:**

PEEK returns one byte of memory. The returned value represents the 8 bits of data as a value from 0 to 255 (&0-&FF hexadecimal) Memory address is the range between 0 to 65535 (&0 ~ &FFFF).

---

**PI****C,V,F**

---

**FORMAT:** 1. PI

**Abbreviation:**

**See Also:**

---

**PURPOSE:**

PI is a numeric pseudovariable which has the value of PI.

**REMARKS:**

It is identical to the use of the special PI character ( $\pi$ ) on the keyboard. Like other numbers the value of PI is kept to 10 digit accuracy (3.141592654).

# POINT

**C,V,Gs****FORMAT:** 1. POINT (expression 1, expression2)**Abbreviation:** POI.**See Also:** GCURSOR, PSET, PRESET**PURPOSE:**

Reads the state of the specified dot.

**REMARKS:**

If the dot specified by (expression 1, expression 2) is lit, a "1" is returned, and if cleared, a "0" is returned. If the specified dot lies beyond the boundaries of the screen, a "-1" is returned.

The values of expression 1 and expression 2 can be specified within the range of -32768 to +32767. However, dots on the screen are in the range of 0-149 for expression 1 and 0-31 for expression 2.

**EXAMPLE:**

10: CLS: WAIT 0:A=75	
20: LINE (50,0)-(50,31)	↳ Draws 2 vertical lines.
30: LINE (100,0)-(100,31)	← Lights up a dot (point) between the 2 lines.
40: PSET (A,16)	← Checks to see whether the next dot on the right is lit.
50: B= POINT (A+1,16)	← If lit, jump to line 150.
60: IF B THEN 150	← If cleared, light it.
70: PSET (A+1,16)	← Then, clear the dot lit up earlier.
80: PRESET (A,16)	← Move one dot to the right.
90: A=A+1	← Go back to line 50.
100: GOTO 50	← Checks to see whether the dot to the left of the lit dot is lit.
150: B= POINT (A-1,16)	← If lit, go to line 50.
160: IF B THEN 50	← If cleared, light it.
170: PSET (A-1,16)	← Then, clear the dot lit up earlier.
180: PRESET (A,16)	← Move one dot to the left.
190: A=A-1	← Go back to line 150.
200: GOTO 150	

Executing this program moves a dot back and forth between 2 vertical lines drawn on the screen.

---

# POKE

**C,V,M**

---

**FORMAT:** 1. POKE memory address, data [,data ...]

**Abbreviation:** PO.

**See Also:** CALL, PEEK

---

**PURPOSE:**

Writes data to a specified memory location.

**REMARKS:**

POKE writes one or more bytes of data starting at the specified memory address. Each data must be specified as a byte in the range 0-65535 (&0-&FFFF hexadecimal.) If more than one byte is specified, the following bytes are written to consecutive memory addresses. Attempting to write beyond memory address limits will generate errors.

---

# PRESET

**C, V, Gs**

---

**FORMAT:** 1. PRESET (expression 1, expression 2)

**Abbreviation:** PRE.

**See Also:** PSET, GCURSOR, POINT

---

**PURPOSE:**

Clears (resets) the specified dot on the screen.

**REMARKS:**

PRESET clears the dot specified by (expression 1, expression 2). The values of expression 1 and expression 2 can be specified within the range of  $-32768$  to  $+32767$ . However, dots on the screen are in the range of  $0-149$  for expression 1 and  $0-31$  for expression 2.

**EXAMPLE:**

```
10: CLS : WAIT 0
20: LINE (20, 0)-(130,31), BF
30: FOR X=-25 TO 25 STEP 0.5
40: Y=-1* SQR ABS (25*25-X*X)
50: PRESET (X+75, Y+31)
60: NEXT X
70: WAIT : GPRINT
```

Executing this program draws a semicircle inside a filled square.

---

# PRINT

---

V

- FORMAT:**
1. PRINT print expr
  2. PRINT print expr, print expr, print expr, print expr
  3. PRINT print list
  4. PRINT print list;
  5. PRINT
  6. PRINT = LPRINT
  7. PRINT = PRINT

Where: print list is: print expr  
          or: print expr; print list  
and: print expr is: expression  
          or: USING clause; expression

The USING clause is described separately under USING

**Abbreviation:** P.

**See Also:** LPRINT, PAUSE, CURSOR, USING, WAIT

---

**PURPOSE:**

PRINT is used to print information on the display or on the printer.

**REMARKS:**

PRINT is used to display prompting information, results of calculations, etc. The first form of PRINT displays a single value. If the expression is numeric, the value is printed at the far right end of the display. If it is a string expression, the display is made starting at the far left.

However, when the display starting position is specified using format (4) or the CURSOR command, the display starts from that position. In format (2), the display unit is divided into groups of 12 columns. The values are displayed, in sequence, from the first specified value. In this case too, within a range of 12 columns, the value of an expression is displayed from the right end of the display and characters are displayed from the left side.

BASIC REFERENCE

The number of the values (items) specified in format (2) must be within 2—8. If the specified value exceeds 12 columns, the following is performed.

- 1) When the numeric value exceeds 12 digits (when the decimal fraction in the exponential display is 8 or more), the least significant digits are truncated.
- 2) When the characters exceed 12 columns, only the first 12 characters (from the left) are displayed.

In format (3), the specified value is displayed continuously from the left side of the display. However, if the display starting position has been specified using format (4) or the CURSOR command, the display starts from that position.

If the value to be displayed in format (3) exceeds 96 columns, the excess portion is not displayed. When the value to be displayed exceeds 96 columns, an error (ERROR 6) occurs if the 96 column is in the middle of a numeric value.

In format (4), the specified value is displayed from the left side of the display. The column following the end of this displayed value is specified as the display starting position of display commands such as for the next PRINT command.

Do not combine display commands (PRINT, etc.) with serial I/O commands (LPRINT, etc. for the serial I/O interface). Combining them may clear the display start position specified in format (4).

In format (5), the previously displayed value is displayed as is.

The sixth and seventh forms of the PRINT statement do no printing. The sixth form causes all PRINT statements which follow it in the program to be treated as if they were LPRINT statements. The seventh form resets this condition so that the PRINT statements will again work with the display.

**EXAMPLE:**

	<u>Display</u>
10: A=123:B=5/9:X\$="ABCDEF": Y\$="VWXYZ"	ABCDEF                    5.55555E—01
20: PRINT X\$,B	



30: PRINT A;B

123.5.555555556E-01

40: PRINT X\$;A;

ABCDEF123.

50: PRINT Y\$;B

ABCDEF123.VWXYZ5.555555  
56E-01

# PRINT #

**C,V,T**

- FORMAT:**
1. PRINT # "var list"
  2. PRINT # "filename" ; var list

Where: var list is: variable  
 or: variable, var list

**Abbreviation:** P. #

**See Also:** INPUT #, PRINT, READ

**PURPOSE:**

PRINT # is used to store values on the cassette tape.

**REMARKS:**

The following variable types can be used for variable names:

- (1) Fixed variables—A, B, X, A(26), C\*, A(10)\*, etc.
- (2) Simple variables—AA, B2, XY\$, etc.
- (3) Array variables—B(\*), CD(\*), N\$(\*), etc.

## 1) Saving fixed variable contents onto tape

The contents of fixed variables can be saved onto tape by specifying the desired variable names (separated by commas) in the PRINT # statement.

```
PRINT # "DATA 1"; A, B, X, Y
```

This statement saves contents of variables A, B, X, and Y into tape file named "DATA 1".

If you wish to save the contents of the specified fixed variable and all the subsequent fixed variables, subscript that variable name with an asterisk\*.

```
PRINT # "D-2"; D*
```

This statement saves the contents of fixed variables D through Z (and of extended variables A(27) and beyond, if defined) into the tape file named "D-2".

```
PRINT # E,X$,A(30)*
```

This statement saves the contents of the fixed variables E and X\$ and of the extended variables A(30) and all the remaining variables, onto the tape without filename.

Subscripted fixed variable names A(1) through A(26) can be specified in the PRINT # statement in much the same way as A through Z (or A\$ through Z\$). However, if array A is already defined by the DIM statement, A( ) cannot be used to define subscripted fixed variables.

## 2) Saving simple variable (two-character variable) contents

The contents of simple variables can be saved onto tape by specifying the variable names.

```
PRINT # "DM-1"; AB, Y1, XY$
```

This statement saves the contents of the simple variables AB, Y1, and XY\$ in the tape file named "DM-1".

## 3) Saving array variable contents

The contents of all variables of a specific array can be saved onto tape by specifying the array name subscripted by an asterisk enclosed in parentheses (\*).

```
PRINT # "DS-2"; X(*), Y$(*)
```

This statement saves the contents of all the elements (X(0), X(1),...) of the array X, and of all the elements (Y\$(0), Y\$(1),...) of the array Y\$, in the tape file name "DS-2".

It is not possible to save the contents of only one or more specific elements of an array. While fixed variables or subscripted fixed variables allow you to save only specific parts of them, an array (such as A), defined by the DIM statement does not allow you to save only a specific part of it.

If the PRINT # statement is executed with no variable names specified, an error (ERROR 1) will result.

**—CAUTION—**

The locations for extended variables such as A(27) and beyond, simple variables, and/or array variables must be set aside in the program/data area before the PRINT # statement is executed. Otherwise, execution of the PRINT # statement for underfined variables will result in an error.

# PRINT #1

**C,V,S**

**FORMAT:** PRINT #1 variable, variable, variable...

**Abbreviation:** P.#1

**See Also:** OPEN, INPUT#1

---

**PURPOSE:**

Sends the contents of the specified variables through the serial I/O interface (terminal).

**REMARKS:**

This command is valid only when the circuit of the serial I/O interface is open (due to the OPEN command). It is ignored otherwise.

\* Variables are specified as follows.

Fixed Variables: Specify each variable name.

[Example] A, B, C\$

Note: Fixed variables cannot be specified in the form of A\*

Simple Variables: Specify each variable name.

[Example] AA, B1\$, C2

Array Variables: Specify in the form of array name (\*).

[Example] B(\*), C\$(\*)

Specified in this manner, the contents of all elements in the array are sent. (Array elements cannot be specified individually).

[Example] 50 PRINT #1A, AB, C\$, E(\*)

When data is sent, the end code is added to the end of the contents of each variable. The end code is added to the end of the contents of each element for array variables also.

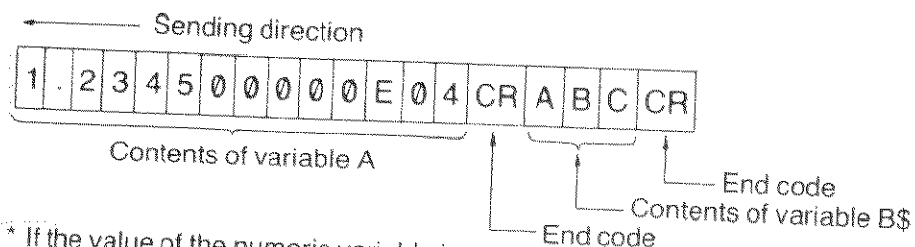
For example, when

A = 12345 and B\$ = "ABC"

executing

PRINT #1A,B\$

Sends A and B\$ in the following form. (Provided the end code is CR.)



\* If the value of the numeric variable is negative, a " - " sign is sent before the value.

The elements of an array are sent in the following sequence.

1 dimensional array:            [Example] for B (3)

$B(0) \rightarrow B(1) \rightarrow B(2) \rightarrow B(3)$

2 dimensional array:            [Example] for C (2, 3)

$C(0, 0) \rightarrow C(0, 1) \rightarrow C(0, 2) \dots$

The locations for extended variables such as A(27) and beyond, simple variables, and/or array variables must be allocated in the program/data area before the PRINT #1 command is executed. An error results if an attempt is made to send the contents of a variable which has not been allocated.

An error also results if the type of the specified variable (number or character) and that of within the PC-1360 do not match.

If symbols such as " $\pi$ " and " $\sqrt{\quad}$ " are included in the data to be sent, they are converted to "PI" and "SQR" respectively, and then sent.

# PSET

**C, V, Gs**

- FORMAT:**
1. PSET (expression 1, expression 2)
  2. PSET (expression 1, expression 2), X

**Abbreviation:** PS.

**See Also:** PRESET, GCURSOR, POINT

**PURPOSE:**

Lights up or reverses the specified dot on the screen.

**REMARKS:**

Format (1) lights up the dot specified by (expression 1, expression 2).

Format (2) clears the dot specified by (expression 1, expression 2) if lit and lights it if cleared.

The values of expression 1 and expression 2 can be specified within the range of  $-32768$  to  $+32767$ . However, dots on the screen are in the range of  $0-149$  for expression 1 and  $0-31$  for expression 2.

**EXAMPLE:**

```
10: CLS: WAIT 0: DEGREE
20: FOR A = 0 TO 600
30: B = -1 * SIN A
40: Y = INT (B * 16) + 16
50: X = INT (A / 4)
60: PSET (X, Y)
70: NEXT A
80: WAIT: GPRINT
```

Executing this program draws a sine curve on the screen.

---

# RADIAN

---

C,V

**FORMAT:** 1. RADIAN

**Abbreviation:** RAD.

**See Also:** DEGREE, GRAD

---

**PURPOSE:**

RADIAN is used to change the form of angular values to radian form.

**REMARKS:**

The PC-1360 has three forms for representing angular values— decimal degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The RADIAN function changes the form for all angular values to radian form until DEGREE or GRAD is used. Radian form represents angles in terms of the length of the arc with respect to a radius, i.e.,  $360^\circ$  is  $2\pi$  radians since the circumference of a circle is  $2\pi$  times the radius.

**EXAMPLE:**

```
10: RADIAN
20: X = ASN 1
30: PRINT X
```

X now has a value of 1.570796327 or  $\pi/2$ , the Arcsine of 1.



---

# RANDOM

---

C,V

**FORMAT:** 1. RANDOM

**Abbreviation:** RA.

**See Also:** RND

---

**PURPOSE:**

RANDOM is used to reset the seed for random number generation.

**REMARKS:**

When random numbers are generated using the RND function, the PC-1360 begins with a predetermined "seed" or starting number. RANDOM resets this seed to a new randomly determined value.

The starting seed will be the same each time the PC-1360 is turned on, so the sequence of random numbers generated with RND is the same each time, unless the seed is changed. This is very convenient during the development of a program because it means that the behavior of the program should be the same each time it is run even though it includes a RND function. When you want the numbers to be truly random, the RANDOM statement can be used to make the seed itself random.

**EXAMPLE:**

```
10: RANDOM
20: X=RND 10
```

When run from line 20, the value of X is based on the standard seed. When run from line 10, a new seed is used.

# READ

V

**FORMAT:** 1. READ variable list

Where: variable list

is: variable

or: variable, variable list

**Abbreviation:** REA.

**See Also:** DATA, RESTORE

## **PURPOSE:**

READ is used to read values from a DATA statement and assign them to variables.

## **REMARKS:**

When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR...NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the sequential order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

If desired, the values in a DATA statement can be read a second time by using the RESTORE statement.

## **EXAMPLE:**

```
10: DIM B(10)
20: WAIT 32
30: FOR I=1 TO 10
40: READ B(I)
50: PRINT B(I)*2;
60: NEXT I
70: DATA 10, 20, 30, 40, 50, 60
80: DATA 70, 80, 90, 100
90: END
```

[10] Set up an array

[40] Loads the values from the DATA statement into B( )—B(1) is 10, B(2) is 20, B(3) is 30, etc.

---

# REM

---

V

**FORMAT:** 1. REM remark

**Abbreviation:** none

**See Also:**

---

**PURPOSE:**

REM is used to include comments in a program.

**REMARKS:**

Often it is useful to include explanatory comments in a program. These can provide titles, names of authors, dates of last modification, usage notes, reminders about algorithms etc. These comments are included by means of the REM statement.

The REM statement has no effect on the program execution and can be included anywhere in the program. Everything following the REM verb in that line is treated as a comment.

**EXAMPLE:**

```
10: REM THIS LINE HAS NO EFFECT
```

# RENUM

C

**FORMAT:** 1. RENUM [new line number], [old line number], [increment]

**Abbreviation:** REN.

**See Also:** DELETE, LIST

**PURPOSE:**

Renumbers the lines of a program.

**REMARKS:**

The lines numbers are changed from old line number to new line number in the specified increment. If new line number is not specified, the lines are renumbered to start from 10 in increments of 10. RENUM updates referenced line numbers in GOTO, ON GOTO, GOSUB, ON...GOSUB, RESTORE, and (IF)...THEN statements.

If any line number exceeds 65279, or a specified line number does not exist, error 4 is generated. Changing execution order generates error 1. RENUM is only used in BASIC and PRO modes (other modes: error 9.) If a password has been used, the command is ignored and the prompt reappears.

If more than one program has been MERGED in memory, RENUM only renumbers the most recently merged program. If the lowest line number of the renumbered program exceeds the highest line number of the program merged before, the two programs are from then on recognized as a single program.

If a line number includes non-numeric characters error 9 is generated.

Error examples:

GOTO 1 + 2	GOTO ABS (-100)
GOTO "A" + "B"	GOTO LEFT\$ ("ABC",2)
GOTO A	GOTO B\$
GOTO 100.0	GOTO + 1E02

Error 3 is generated if renumbering makes the line length exceed 79 bytes.

If RESTORE is used in a multi-statement line, followed by a colon (e.g., 20 RESTORE: READ A), any later attempt to renumber that program using the RENUM command will generate ERROR 9.

If the display shows “\*”, pressing the **BRK** key will interrupt renumbering. A display of “\*\*” indicates that BREAKing is not possible. Error generation or use of the **BRK** key leaves the program unchanged.

## RENUM ERROR LIST

Error display	Description	Flashing cursor location
ERROR 1	RENUM input specification error	site of error
ERROR 1 IN	“:”, “,” or CR after GOTO or GOSUB; or “:” or CR after THEN	site of error; or end of line
ERROR 3 IN	new line number exceeds 65279; or line length exceeds 79 bytes	head of line; or end of line
ERROR 4	referenced line does not exist; or a parameter exceeds 65279	site of error
ERROR 4 IN	referenced line number does not exist	referencing line
ERROR 9	attempt to RENUM in RUN or TEXT mode	site of error
ERROR 9 IN	non-numeric characters in line number; or non-executable statement after THEN	site of error

**EXAMPLE:**

```

10: INPUT "CONTINUE"; A$
20: IF A$="YES" THEN 60
30: IF A$="NO" THEN 10
40: PRINT "ENTER YES OR NO PLEASE!"
50: GOTO 10
60: END

```

```

RENUM 100, 10, 5
LIST

```

```

100: INPUT "CONTINUE"; A$
105: IF A$="YES" THEN 125
110: IF A$="NO" THEN 100
115: PRINT "ENTER YES OR NO PLEASE!"
120: GOTO 100
125: END

```

# RESTORE

V

**FORMAT:** 1. RESTORE  
2. RESTORE expression

**Abbreviation:** RES.

**See Also:** DATA, READ

---

**PURPOSE:**

RESTORE is used to reread values in a DATA statement or to change the order in which these values are read.

**REMARKS:**

In the regular use of READ the PC-1360 begins reading with the first value in a DATA statement and proceeds sequentially through the remaining values. The first form of the RESTORE statement resets the pointer to the first value of the first DATA statement, so that it can be read again. The second form of the RESTORE statement resets the pointer to the first value of the first DATA statement whose line number is greater than the value of the expression.

If RESTORE is used in a multi-statement line, followed by a colon (e.g., 20 RESTORE: READ A), any later attempt to renumber that program using the RENUM command will generate ERROR 9. Make sure that all RESTORE statements are either on separate lines at the end of any multi-statement line, or in format 2; followed by an expression (e.g., RESTORE X\*Y).

**EXAMPLE:**

```
10: DIM B(10)
20: WAIT 32
30: FOR I=1 TO 10
40: RESTORE
50: READ B(I)
60: PRINT B(I)*I;
70: NEXT I
80: DATA 20
90: END
```

[10] Sets up an array.

[50] Assign the value 20 to each of the elements of B ( ).

---

# RETURN

---

V

**FORMAT:** 1. RETURN

**Abbreviation:** RE.

**See Also:** GOSUB, ON...GOSUB

---

**PURPOSE:**

RETURN is used at the end of a subroutine to return control to the statement following the originating GOSUB.

**REMARKS:**

A subroutine may have more than one RETURN statement, but the first one executed terminates the execution of the subroutine. The next statement executed will be the one following the GOSUB or ON...GOSUB which calls the subroutine. If a RETURN is executed without a GOSUB, and ERROR 5 will occur.

**EXAMPLE:**

```
10: GOSUB 100
20: END
100: PRINT "HELLO"
200: RETURN
```

When run this program prints the word "HELLO" one time.

# RIGHT\$

C,V,F

**FORMAT:** 1. RIGHT\$(X\$,N)  
2. RIGHT\$("string",N)

**Abbreviation:** RI.

**See Also:** LEFT\$, MID\$

**PURPOSE:**

Returns N characters from the right end of any string X\$.

**REMARKS:**

The value of N must be in the range 0 to 80. Fractions will be rounded down (truncated). If N is less than 1, a null string is returned. If N is greater than the number of characters in X\$, the whole string is returned.

**EXAMPLE:**

```
5: WAIT 32
10: XX$ = "SHARP COMPUTER"
20: FOR N = 1 TO 15
30: LET SS$ = RIGHT$(XX$,N)
40: PRINT SS$
50: NEXT N
```

```
RUN
R
ER
TER
UTER
PUTER
MPUTER
OMPUTER
COMPUTER
  COMPUTER
P COMPUTER
RP COMPUTER
ARP COMPUTER
HARP COMPUTER
SHARP COMPUTER
SHARP COMPUTER
>
```



---

# RLINE

**C,V,Gp**

---

**FORMAT:** 1. RLINE (expression 1, expression 2) – (expression 3, expression 4) [, expression 5][, expression 6][, B]

**Abbreviation:** RL.

**See Also:** LLINE, PAINT, COLOR

---

**PURPOSE:**

RLINE is used to draw a line between the two points specified by relative coordinates.

**REMARKS:**

This verb is effective only in the Graphics mode.

RLINE differs from LLINE in that LLINE takes the origin of coordinates specified by SORGN as a reference and specifies the location of each point by coordinates from that origin, whereas RLINE takes the current position of the pen as the origin of coordinates and specifies the location of the next point by coordinates from that origin.

**NOTE:**

When LLINE or RLINE without B is executed after the execution of these commands with B, following command should be executed immediately after the execution of these commands without B;

```
POKE &FB20,0
```

If this command is not executed the computer may draw rectangle by LLINE or RLINE without B;

Descriptions of the specification are the same as LLINE, except that (expression 1, expression 2) cannot be omitted.

**EXAMPLE:**

```
5: OPEN  
10: GRAPH  
20: FOR A=1 TO 3  
30: RLINE (0,-40) - (60,-50)  
40: NEXT A  
50: LTEXT  
60: LPRINT  
70: END
```

# RND

**C,V,F****FORMAT:** 1. RND numeric expression**Abbreviation:** RN.**See Also:** RANDOM**PURPOSE:**

RND is a numeric function which generates random numbers.

**REMARKS:**

If the value of the argument is less than 1 but greater than or equal to zero, the random number is less than 1 and greater than or equal to zero. If the argument is an integer greater than or equal to 1, the result is a random number greater than or equal to 1 and less than or equal to the argument. If the argument is greater than or equal to 1 and not an integer, the result is a random number greater than or equal to 1 and less than or equal to the smallest integer which is larger than the argument: (In this case, the generation of the random number changes depending on the value of the decimal portion of the argument.):

<u>Argument</u>	<u>Result</u>	
	<u>Lower Bound</u>	<u>Upper Bound</u>
.5	0 <	< 1
2	1	2
2.5	1	3

The same sequence of random numbers is normally generated because the same "seed" is used each time the PC-1360 is turned on. To randomize the seed, see the RANDOM verb.

---

# RUN

---

C

**FORMAT:** 1. RUN  
2. RUN { line number }  
          { label }

**Abbreviation:** R.

**See Also:** GOTO, MERGE, ARUN

---

**PURPOSE:**

The RUN command is used to execute a program in memory.

**REMARKS:**

The first form of the RUN command executes a program beginning with the lowest numbered statement in memory.

The second form of the RUN command executed a program beginning with the specified line number.

If more than one program is in memory because programs have been MERGE<sub>d</sub>, the RUN command will execute the last program that was merged unless a program label is used in the program and the command.

RUN differs from GOTO in eight respects:

- 1) The value of the interval for WAIT is reset.
- 2) The display format established by USING statements is cleared.
- 3) Variables and arrays other than the fixed variables are cleared.
- 4) PRINT = PRINT status is set.
- 5) The pointer for READ is reset to the first DATA statement.
- 6) The cursor specification is cleared.
- 7) The graphics cursor is reset to (0,7).
- 8) The serial I/O interface port is closed.

Execution of a program with GOTO is identical to execution with the DEF key. In all three forms of program execution FOR/NEXT and GOSUB nesting is cleared.

**EXAMPLE:**

RUN 100

Executes the program starting from line 100.

---

# SAVE

---

C

**FORMAT:** 1. SAVE

**Abbreviation:** SA.

**See Also:** OPEN, LLIST, LOAD, CSAVE

---

**PURPOSE:**

Sends the program in the PC-1360 out through the serial I/O interface (terminal).

**REMARKS:**

When the circuit of the serial I/O interface is open due to the OPEN command, the program is sent in ASCII code.

The command is ignored if the circuit is closed.

After the entire program is sent, the text end code is sent.

The SAVE command is ignored if a password has been set.

# SET MEM

C

**FORMAT:** 1. SET MEM "C"  
2. SET MEM "B"  
3. SET MEM "D"

**Abbreviation:** SE. M.

**See Also:** MEM, MEM\$

**PURPOSE:**

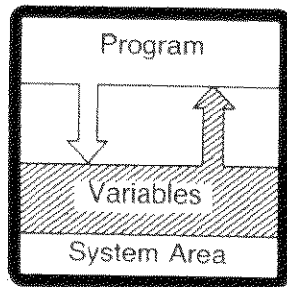
SET MEM sets up which RAM slots are to be used and the memory structure for program area and data variable area.

**REMARKS:**

Refer to the section on the use of the RAM card slots in the earlier chapter.

**1. SET MEM "C"** This format uses just the RAM card in slot 1. The program is stored in the top part and variable area is held at the bottom. Use this mode when only one RAM card is to be used.

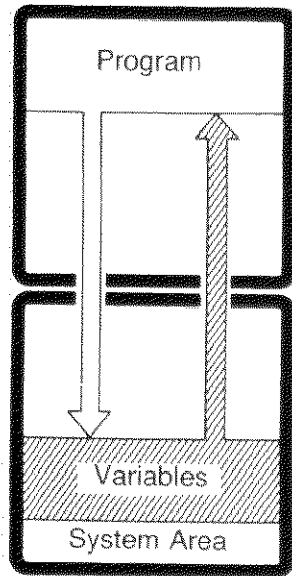
RAM card 1 in Slot 1



MEM returns the byte count for this unused area of the memory.

**2. SET MEM "B"** This allows use of both RAM cards at the same time. The program part is held starting in the card in slot 1 while the system area and variable part is held starting in the card in slot 2. Either the program part or the variable part is allowed to extend into the other memory. This arrangement makes the most flexible use of the total memory available but means that the two cards must always be used together. Using just one card, or exchanging one card for another is not possible.

RAM card 1 in Slot 1

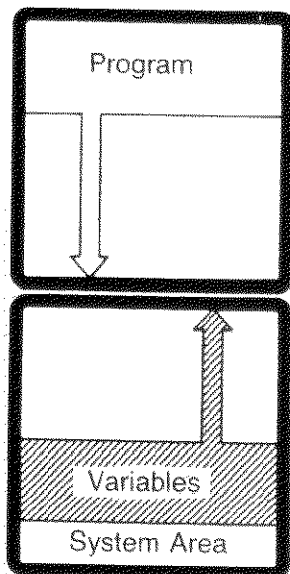


MEM returns the byte count for this unused area of the memory.

RAM card 2 in Slot 2

**3. SET MEM "D"** This also allows use of both RAM cards at the same time. The program part is held only in the card in slot 1 and the variable part is held only in the card in slot 2. No overflow of one area into the memory of the other card is allowed. This restricts the flexibility of using the two cards together. If the memory usage exceeds the capacity of either card errors will be generated. The advantage of this structure is that cards can be swapped individually. For example, one card can be changed with another that was used under the same SET MEM "D" condition. Note that the program holding card must always be in slot 1 and the variable holding card in slot 2.

RAM card 1 in Slot 1



MEM returns the byte count for this unused area of the memory.

RAM card 2 in Slot 2

**Changing RAM card memory usage:**

This is only effective in RUN mode.

From "B" or "D" to "C"

This is putting the program and variable parts held on two cards onto a single card. If the total exceeds the memory area available on the one card, execution stops and an error is generated.

From "D" to "B"

This removes the restriction that program and variable areas must be contained in their respective card memories.

From "B" or "C" to "D"

If the program or data part is too big to be held in just one card an error is generated.

From "C" to "B"

This splits the program and variable parts held on a single card across two cards. First turn off the computer, put the new card in slot 2, then turn on again and execute SET MEM "B".

**Determining total program area:**

In RUN mode, enter CLEAR and press the ENTER key.

For "C" memory structure:

$$(\text{RAM card 1 (KB) capacity} \times 1024) - \text{MEM} - 1634$$

For "B" and "D" memory structure:

$$((\text{RAM card 1} + \text{RAM card 2 (KB) capacities}) \times 1024) - \text{MEM} - 1634$$

When removing a card, first check with power on that the storage structure is either "C" or "D" (using MEM\$) then turn off the power and remove the card. Remember that cards cannot be removed when MEM\$ indicates storage structure "B".

RAM cards holding programs created using PC-1450, and PC-1460 computers can be directly used on the PC-1360. It is also possible to exchange and use cards on different PC-1360 computers.



Programs and data are not destroyed by using SET MEM to switch from one memory structure to another.

The contents of the card in slot 1 can be copied to the card in slot 2 (see the chapter on the RAM cards.)

# SGN

**C,V,F****FORMAT:** 1. SGN (X)**Abbreviation:** SG.**See Also:****PURPOSE:**

Returns the sign of expression X.

**REMARKS:**

X can be any numeric expression. SGN(X) returns either 1, 0 or -1 depending on the sign of expression X as shown below:

Value of X	Value returned by SGN (X)
$X > 0$	1
$X = 0$	0
$X < 0$	-1

**EXAMPLE:**

```

5: WAIT 100
10: FOR N = -3 TO 3
20: PRINT N, SGN (N)
30: NEXT N
40: END

```

RUN

```

-3.    -1.
-2.    -1.
-1.    -1.
 0.     0.
 1.     1.
 2.     1.
 3.     1.

```

---

# SIN

**C,V,F**

---

**FORMAT:** 1. SIN (X)

**Abbreviation:** Sl.

**See Also:** ASN, COS, TAN

---

**PURPOSE:**

Returns the sine of the angle X.

**REMARKS:**

This function returns the sine of the angle X, where X is expressed in degrees, radians or as a gradient value, depending on which mode the computer is set to with the DEGREE, RADIAN or GRAD command.

**EXAMPLE:**

```
10: DEGREE
20: PRINT "SIN OF 30 IS "; SIN (30)
30: PRINT "SIN OF 90 IS "; SIN (90)
40: END
```

```
RUN
SIN OF 30 IS 0.5
SIN OF 90 IS 1.
```

# SORGN

C, V, Gp

**FORMAT:** 1. SORGN

**Abbreviation:** SO.

**See Also:** GLCURSOR

**PURPOSE:**

SORGN is used to change the origin of coordinates for drawing with the pen.

**REMARKS:**

This verb is effective only in the Graphics mode and is used to specify the current position of the pen as the new origin of coordinates.

When drawing a figure, it may not be easy for the printer to do so if the origin of coordinates is located at the left end of the paper. In such a case, move the pen to an arbitrary position on the paper by executing GLCURSOR and then specify that position as the origin of coordinates with SORGN.

This facilitates drawing of a figure with the current position of the pen taken as a reference point.

**EXAMPLE:**

```
10: GRAPH  
20: GLCURSOR (60, 40)  
30: SORGN
```

[30] Specifies the current position (X=60, Y=40) of the pen as the new origin of coordinates.

---

# SQR

**C,V,F**

---

**FORMAT:** 1. SQR (X)

**Abbreviation:** SQ.

**See Also:**

---

**PURPOSE:**

Returns the square root of expression X.

**REMARKS:**

If the expression evaluates to a negative number, SQR(X) generates an error code.  $\sqrt{\quad}$  can be used instead of SQR.

**EXAMPLE:**

SQR (5)

2.236067977

# STOP

V

---

**FORMAT:** 1. STOP

**Abbreviation:** S.

**See Also:** END, CONT

---

**PURPOSE:**

STOP is used to halt execution of a program for diagnostic purposes.

**REMARKS:**

When STOP is encountered in program execution, PC-1360 execution halts and a message is displayed such as 'BREAK IN 200' where 200 is the number of the line containing the STOP. STOP is used during the development of a program to check the flow of the program or examine the state of variables. Execution may be restarted using the CONT command.

**EXAMPLE:**

10: STOP

Causes "BREAK IN 10" to appear in the display.

# STR\$

**C,V,F****FORMAT:** 1. STR\$**Abbreviation:** STR.**See Also:** VAL**PURPOSE:**

Converts numeric data into string data.

**REMARKS:**

The STR\$ function changes an integer number into a string. The string will be composed of the same digits as the original number, but it is treated as a string of SHARP ASCII characters in subsequent processing. The STR\$ function has the opposite effect to the VAL function.

If the numeric data is negative, the string will be preceded by a – sign. If numeric value is too large to be held in the string variable, it is represented in floating point notation.

**EXAMPLE:**

```

:
:
:
110: N = N*3
120: A$ = STR$ (N)
130: B$ = LEFT$ (A$, 1)
140: M = VAL (B$)
:
:

```

[110] The program performs some calculations on the numeric variable N.

[120] The numeric variable N is converted to the string variable A\$. String variables are much easier to manipulate than numerics. In this example, imagine that the first digit of the number is required. Maybe it is a code for some further process. Having converted the number to a string we can use any of the string manipulation commands: LEFT\$, RIGHT\$, MID\$.

[130] This stores just the first digit of the number, or character as it is now treated by the program, in the string variable B\$.

[140] The single digit is reconverted into a numeric variable so that it can be processed by the program as a number.

---

# TAN

---

C,V,F

**FORMAT:** 1. TAN(X)

**Abbreviation:** TA.

**See Also:** ATN, COS, SIN

---

**PURPOSE:**

Returns the tangent of the angle X.

**REMARKS:**

This function returns the tangent of the angle X, where X is expressed in degrees, radians or as a gradient value, depending on which mode the computer is set to with the DEGREE, RADIAN or GRAD command. An error code is generated if X is 90°.

**EXAMPLE:**

```
10: DEGREE
20: PRINT "TAN OF 0 IS"; TAN (0)
30: PRINT "TAN OF 45 IS"; TAN (45)
40: END
```

```
RUN
TAN OF 0 IS 0.
TAN OF 45 IS 1.
```



---

# TEXT

---

C

**FORMAT:** 1. TEXT

**Abbreviation:** TE.

**See Also:** BASIC

---

**PURPOSE:**

Sets the text mode.

(valid only in the program mode)

**REMARKS:**

The text function is used when inputting a program written for a higher level personal computer. The program input by the PC-1360 is sent to the personal computer through the serial I/O interface.

Executing the TEXT command sets the text mode in the text mode, a number corresponding to the line number and then information corresponding to program commands or data are entered. Then the **ENTER** key is pressed to write the input to the program/data area.

However, the written contents, unlike in BASIC mode, are not converted to commands (internal codes). The text is stored as characters and/or numbers in ASCII code. The text is arranged in the order of line number at the beginning of each line. (Line number editing function.)

The text written in the text mode is stored as it is. Therefore, command abbreviations in BASIC (such as I. for INPUT) are display and stored as they are.

If a program is stored in the internal code of the PC-1360 with the text mode set, it is converted to ASCII code.

During program conversion, the “\*\*” mark is displayed on the right end of the 4th line of the display unit.

The prompt symbol is “<” in the text mode. (It is usually “>”.)

One line in the text mode (including line number and **ENTER**) must not exceed 80 characters (80 bytes). If a line exceeds 80 characters due to program conversion, the excess part will be cleared.

80 bytes (end is **ENTER**)

10: PRINT "ABC ..... 1234567890"

Converted to ASCII code.

10: PRINT "ABC ..... 1234567"

80 characters (end is **ENTER**)

In this example, the PRINT command is two bytes in internal code but takes up 6 bytes in ASCII code. Because of this, the last few characters are deleted ("890").

The number of bytes increases when converting a program from internal code to ASCII code, as shown in the example. If, as a result, the capacity of the program area is exceeded, the program converted up to that point is converted back to internal code and an error will result (ERROR 6).

If a password has been set, an error (ERROR 1) occurs when the TEXT command is executed.

---

# TROFF

---

**C,V**

**FORMAT:** 1. TROFF

**Abbreviation:** TROF.

**See Also:** TRON

---

**PURPOSE:**

TROFF is used to cancel the trace mode.

**REMARKS:**

Execution of TROFF restores normal execution of the program.

**EXAMPLE:**

```
10: TRON
20: FOR I = 1 TO 3
30: NEXT I
40: TROFF
```

When run, this program displays the line numbers 10, 20, 30, 30, 30 and 40 as the  is pressed.

# TRON

C,V

**FORMAT:** 1. TRON

**Abbreviation:** TR.

**See Also:** TROFF

**PURPOSE:**

TRON is used to initiate the trace mode.

**REMARKS:**

The trace mode provides assistance in debugging programs. When the trace mode is on, line number of each statement is displayed after each statement is executed. The PC-1360 then halts and waits for the Down Arrow key to be pressed before moving on to the next statement. The Up Arrow key may be pressed to see the statement which has just been executed. The trace mode continues until TROFF is executed or the key operation of the **SHIFT** and **CLS**, is performed. After a result is displayed at the position specified in the **CURSOR** command during the trace mode, the next line number is displayed on the next line of the display. (See **CURSOR**.)

During trace mode, if the display start position has been specified in the program by a **CURSOR** command, calling a variable or executing a manual calculation while the PC-1360 is waiting for the Down Arrow key will clear the **CURSOR** setting when execution is resumed.

**EXAMPLE:**

```
10: TRON
20: FOR I = 1 TO 3
30: NEXT I
40: TROFF
```

When run, this program displays the line number 10, 20, 30, 30, 30 and 40 as the **↓** is pressed.

---

# USING

---

**C,V**

- FORMAT:**
1. USING
  2. USING "editing specification"
  3. USING character variable

**Abbreviation:** U.

**See Also:** LPRINT, MDF, PAUSE, PRINT

Further guide to the use of USING is provided in Appendix C

---

**PURPOSE:**

USING is used to control the format of displayed or printed output.

**REMARKS:**

USING can be used by itself or as a clause within a LPRINT, PAUSE, or PRINT statement. USING establishes a specified format for output which is used for all output which follows until changed by another USING.

The editing specification of USING consists of a quoted string composed of some combination of the following editing characters:

- # Right justified numeric field character
- Decimal point
- ^ Used to indicate that numbers should be displayed in scientific notation
- & Left justified alphanumeric field

For example "####" is an editing specification for a right justified numeric field with room for 3 digits and the sign. In numeric fields, a location must be included for the sign, even if it will always be positive.

Editing specifications may include more than one field. For example "####&&&" could be used to print a numeric and a character field next to each other.

If the editing specification is missing, as in format 1, special formatting is turned off.

**EXAMPLE:**

	<u>Display</u>	
10: A = 125: X\$ = "ABCDEF"		
20: PRINT USING "##.##^";A		1.25E 02
30: PRINT USING "&&&&&&";X\$	ABCDEF	
40: PRINT USING "####&&";A;X\$	125ABC	

# VAL

**C,V,F**

**FORMAT:** 1. VAL (X\$)  
2. VAL ("string")

**Abbreviation:** V.

**See Also:** STR\$

**PURPOSE:**

Converts a string of numeric characters into a decimal value.

**REMARKS:**

The VAL function is the reverse of the STR\$ function. It changes a string composed of numeric characters into a numeric value that can be used as a number in subsequent processing.

If the string is in decimal, it must be composed of the characters 0–9, with optional decimal point and sign. In this form, VAL is the opposite of the STR\$ function.

If illegal characters are included, conversion is performed up to the first occurrence of an illegal character.

**EXAMPLE:**

```
10: INPUT "CYCLE FREQUENCY ";A$
15: IF ASC (A$) <48 OR ASC (A$) > 57 THEN 100
20: F = VAL (A$)
30: PRINT F
40: STOP
:
:
100: PRINT "MUST BE A NUMBER": GOTO 10
```

[10] This inputs a string, which is to be converted to a numeric value.

[20] The string is converted to its numeric equivalent. Inputting a number as a string gives the programmer a chance to check the input to make sure it is of the correct type or in the correct range within the program itself.

# WAIT

C,V

**FORMAT:** 1. WAIT expression  
2. WAIT

**Abbreviation:** W.

**See Also:** PAUSE, PRINT

---

**PURPOSE:**

WAIT is used to control the length of time that displayed information is shown before program execution continues.

**REMARKS:**

In normal execution the PC-1360 halts execution after a PRINT, GPRINT, PSET, PRESET LINE command until the **ENTER** key is pressed. WAIT causes the PC-1360 to display for a specified interval and then proceed automatically (similar to PAUSE). The expression which follows WAIT determines the length of the interval. The interval may be set to any value from 0 to 65535. Each increment is about one fifty-ninth of a second. WAIT 0 is too fast to be read reasonably; WAIT 65535 is about 19 minutes. WAIT with no following expression resets the PC-1360 to the original condition of waiting until the **ENTER** key is pressed.

**EXAMPLE:**

10: WAIT 59

Causes PRINT to wait about 1 second.



# 10 Troubleshooting

This chapter provides you with some hints on what to do when your SHARP PC-1360 does not do what you expect it to do. It is divided into two parts—the first part deals with general machine operation and the second with BASIC programming. For each problem there are a series of suggestions provided. You should try each of these, one at a time, until you have fixed the problem.

## Machine Operation

If:	Then You Should:
You turn on the machine but there is nothing on the display.	<ol style="list-style-type: none"> <li>1. Check to see that the slide power switch is set to ON position.</li> </ol>
There is a display, but no response to keystrokes.	<ol style="list-style-type: none"> <li>1. Push the <b>BRK</b> key to see if AUTO POWER OFF has been activated.</li> <li>2. Replace the batteries.</li> <li>3. Adjust the contrast control.</li> </ol>
You have typed in a calculation or answer and get no response.	<ol style="list-style-type: none"> <li>1. Press <b>CLS</b> key to clear.</li> <li>2. Press <b>CA</b> ( <b>SHIFT</b> <b>CLS</b> ) to clear.</li> <li>3. Turn OFF and ON again.</li> <li>4. Hold down any key and push ALL RESET.</li> <li>5. Push ALL RESET without any key.</li> </ol>
You are running a BASIC program and it displays something, and stops.	<ol style="list-style-type: none"> <li>1. Push <b>ENTER</b>.</li> </ol>
You enter a calculation and it is displayed in BASIC statement format (colon after the first number)	<ol style="list-style-type: none"> <li>1. Push <b>ENTER</b>.</li> <li>1. Change from the PROgram into the RUN mode for calculations.</li> </ol>



You get no response from any keys.

1. Hold down any key and push ALL RESET.
2. If you get no response from any key even when the above operation is performed, push the ALL RESET without pushing any key. This will clear the program, data and all reserved contents.



## BASIC Debugging

When entering a new BASIC program, it is usual for it not to work the first time. Even if you are simply keying in a program that you know is correct, such as those provided in this manual, it is usual to make at least one typing error. If it is a new program of any length, it will probably contain at least one logic error as well. Following are some general hints on how to find and correct your errors.

You run your program and get an error message:

1. Go back to the PROgram mode and use the  or the  keys to recall the line with the error. The cursor will be positioned at the place in the line where the PC-1360 got confused.
2. If you can't find an obvious error in the way in which the line is written, the problem may lie with the values which are being used. For example, CHR\$(A) will produce a space if A has a value of 1. Check the values of the variables in either the RUN or the PROgram mode by typing in the name of the variable followed by **ENTER**.

You run the program and don't get an error message, but it doesn't do what you expect.

3. Check through the program line by line using LIST and the  and  keys to see if you have entered the program correctly. It is surprising how many errors can be fixed by just taking another look at the program.
4. Think about each line as you go through the program as if you were the computer. Take sample values and try to apply the operation in each line to see if you get the result that you expected.

5. Insert one or more extra PRINT statements in your program to display key values and key locations. Use these to isolate the parts of the program that are working correctly and the location of the error. This approach is also useful for determining which parts of a program have been executed. You can also use STOP to temporarily halt execution at critical points so that several variables can be examined.
6. Use TRON and TROFF, either as commands or directly within the program to trace the flow of the program through individual lines. Stop to examine the contents of critical variables at crucial points. This is a very slow way to find a problem, but sometimes it is also the only way.

To continue the program, press the **↓** (Down Arrow) key once. This causes the next line to be executed and its line number to be displayed. Again, you may review the line with the Up Arrow key. You may also check the contents of any variable by typing its name and pressing **ENTER**:

**A** **ENTER** (when A = 4 is input before **A** **ENTER** operation)

4.

It is necessary to press the **↓** (Down Arrow) key once for each line to be executed until the program ends. If you do not wish to continue normal line-by-line execution, press the **ENTER** key to suspend execution of the program. If you change your mind again, suspended programs may be continued with the CONT command.

## TROUBLESHOOTING

A sample session, using our hypotenuse program, follows:

Input	Display
	>
<b>T R O N</b>	TRON_
<b>ENTER</b>	>
<b>R U N</b>	RUN_
<b>ENTER</b>	?
<b>3</b>	3_
<b>ENTER</b>	?
<b>4</b>	4_
<b>ENTER</b>	10:
<b>↑</b>	10: INPUT A, B
<b>↓</b>	20:
<b>↑</b>	20: A=A*A : B=B*B
<b>A</b>	A_
<b>ENTER</b>	9.
<b>B</b>	B_
<b>ENTER</b>	16.
<b>↓</b>	30:
<b>H</b>	H_
<b>ENTER</b>	5.
<b>↓</b>	HYPOTENUSE=5.
<b>↑</b>	40: PRINT "HYPOTENUSE=";
<b>↓</b>	40:
<b>↓</b>	>

No matter how careful you are, eventually you will create a program which does not do quite what you expect it to. In order to isolate the problem, Sharp's designers have provided a special method of executing programs known as the "Trace" mode. In the Trace mode, the PC-1360 will display the line number of each program line and will halt after the execution of that line. This allows you to follow (or trace) the sequence of instructions as they are actually performed. When the program pauses after the execution of a line, you may inspect or alter the values of variables.

The form of the instruction for initiating the Trace mode is simply: TRON. The TRON instruction may be issued as a command (in RUN mode) or it may be embedded, as a verb, within a program. Used as a command, TRON informs the PC-1360 that tracing is required during the execution of all subsequent programs. The programs to be traced are then started in a normal manner, with a GOTO or RUN command.

If TRON is used as a statement, it will initiate the Trace mode only when the line containing it is executed. If, for some reason, that line is never reached, the Trace mode will remain inactive.

Once initiated, the Trace mode of operation remains in effect until canceled by a TROFF instruction. The TROFF instruction may also be issued as either a command or a verb. The Trace mode can also be canceled by the key sequence:

**SHIFT**      **CA**  
**CLS**


As an example in using the Trace mode, enter the following program to compute the length of the hypotenuse of a triangle given the length of the sides:

Program Listing:

```
10: INPUT A, B
20: A = A * A : B = B * B
30: H =  $\sqrt{A + B}$ 
40: PRINT "HYPOTENUSE = "; H
```

In RUN mode, issue the TRON command, followed by the RUN command. Notice the INPUT command operates in the usual manner by displaying a question mark for each input value required. As soon as you have entered two values, the line number of the INPUT statement appears:

```
10:
```

By pressing the  (Up Arrow) key and holding it, you may review the entire line:

```
10: INPUT A,B
```

In the trace mode, after the calculated result is displayed at the location specified with the CURSOR command, the next line number is displayed on the following line. (See CURSOR.)

In the trace mode, if variables are called or if a calculation is performed manually when the display starting position has been specified with the CURSOR command, the display starting position will be cleared.

# 11 Maintenance of the PC-1360

To insure trouble-free operation of your SHARP PC-1360 we recommend the following:

- Always handle the computer carefully as the liquid crystal display is made of glass.
- Keep the computer in an area free from extreme temperature changes, moisture, or dust. During warm weather, vehicles left in direct sunlight are subject to high temperature build up. Prolonged exposure to high temperature may cause damage to your computer.
- Use only a soft, dry cloth to clean the computer. Do not use solvents, water, or wet cloths.
- To avoid battery leakage, remove the batteries when the computer will not be in use for an extended period of time.
- If the computer is subjected to strong static electricity or external noise it may "hang up" (all keys become inoperative). If this occurs, press the ALL RESET button while holding down any key. (See Troubleshooting).
- Keep this manual for further reference.

**Appendix A****Error Messages**

There are nine different codes built into the PC-1360. The following table will explain these codes.

**Error****Number      Meaning**

1            Syntax error

This means that the PC-1360 can't understand what you have entered. Check for things such as semicolons on the ends of PRINT statement, misspelled words, and incorrect usages.

3\*/2

2            Calculation error

Here you have probably done one of three things:

1. Tried to use too large a number.

Calculation results are greater than 9.999999999E 99.

2. Tried to divide by zero.

5/0

3. An illogical calculation has been attempted.

LN -30 or ASN 1.5

3            Illegal Function (DIMension error/Argument error)

Array variable already exists.

Array specified without first dimensioning it.

Array subscript exceeds size of array specified in DIM statement.

DIM B(256)

Illegal function argument. This means that you have tried to make the computer do something that it just can't handle.

The interval that is greater than 65535.

WAIT 66000



## 4 Too Large A Line Number

Here you have probably done one of two things:

1. Tried to use a non-existent line number by the GOTO, GOSUB, RUN, LIST or THEN etc.
2. Tried to use too large a line number. The maximum line number is 65279.

## 5 Next Without A For...

Subroutine nesting exceeds 10 levels.

FOR loop nesting exceeds 5 levels.

RETURN verb without a GOSUB, NEXT verb without a FOR, or READ verb without a DATA

Buffer space exceeded.

## 6 Memory Overflow

Generally this error happens when you've tried to DIMension an array that is too big for memory. This can also happen when a program becomes too large.

The reserve content exceeds 144 bytes.

## 7 PRINT USING error

This means that you have put an illegal format specifier into a USING statement.

## 8 I/O device error

This error can happen when you have the optional printer and/or cassette recorder connected to the PC-1360. This error can also happen when you use serial input/output. It means that there is a problem with communication between the I/O device and the PC-1360.

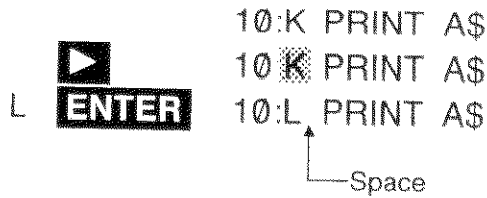
9 Other errors

This code will be displayed whenever the computer has a problem that isn't covered by one of the other eight error codes. One of the most common causes for this error is trying to access data in a variable in one fashion (e.g. A\$) while the data was originally stored in the variable in another fashion (e.g.A).

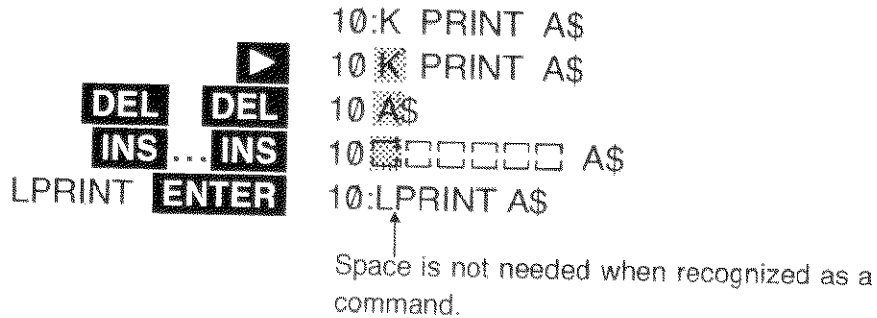
**Regarding Input Errors**

When executing a program, an error may occur due to input errors of the program. In this case note the following.

Example: When KPRINT is entered instead of LPRINT



When corrected in this manner, the computer does not recognize it as a command. In this example, erase KPRINT and re-enter LPRINT.

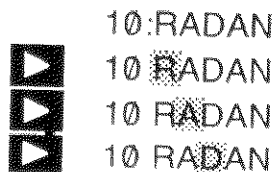


The command can be checked whether entered correctly by using the cursor key.

(Correct input)



(Wrong input)



*Appendix B***Character Code Chart**

The following chart shows the conversion values for use with CHR\$ and ASC. The column shows the first hex character or the first four binary bits, the row shows the second hex character or the second binary bits. The upper left corner of each box contains the decimal number for the character. The lower right shows the character. If no character is shown then it is an illegal character on the PC-1360.

For examples, the character "A" is decimal 65 or hex 41 or binary 01000001. The character '√' is decimal 252 or hex FC or binary 11111100.

The character codes are represented as follows.

Examples;

Code for " \* "

Hexadecimal        &2A

Decimal             42 (32 + 10)

Code for " √ "

Hexadecimal        &FC

Decimal             252 (240 + 12)

Displaying a character using the CHR\$ command:

The character for code 0 (&00) in the table is null. Nothing is displayed. Squares in the table where no characters are listed are spaces.

Printing a character on the CE-126P using the CHR\$ command:

- Do not use code 0 (&00).
- Squares in the table where no characters are listed are spaces.
- Codes 249 (&F9) and 250 (&FA) are spaces.
- The character for code 92 (&5C) is "▲".

Printing a character on the CE-140P using the CHR\$ command:

- For printing characters on the CE-140P, codes 8 (&08), 10 (&0A), 11 (&0B), 13 (&0D), and 27 (&1B) are the control codes for the printer. The rest, 0 (&00) through 31 (&1F), are NULL.
- Any codes other than 0 (&00) through 31 (&1F) not used for characters are printed as spaces.
- Codes 249 (&F9) and 250 (&FA) are spaces.

APPENDICES

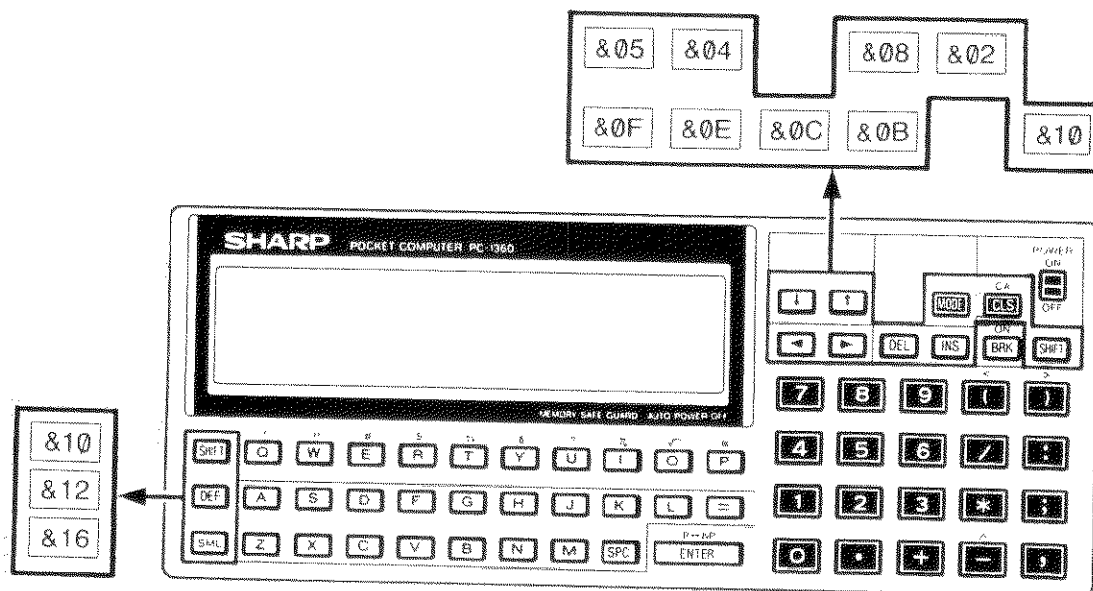
For printing characters on the CE-515P, refer to the Character Code Table in the CE-515P Operation Manual and specify the character code.

First 4 Bits

Second 4 Bits

Hex	0	1	2	3	4	5	6	7	8	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1110	1111
0	0	16	32	48	64	80	96	112	128	224	240
0000	NUL		SPACE	0	@	P	q	p			
1	1	17	33	49	65	81	97	113	129	225	241
0001			!	1	A	Q	a	q			
2	2	18	34	50	66	82	98	114	130	226	242
0010			"	2	B	R	b	r			
3	3	19	35	51	67	83	99	115	131	227	243
0011			#	3	C	S	c	s			
4	4	20	36	52	68	84	100	116	132	228	244
0100			\$	4	D	T	d	t			
5	5	21	37	53	69	85	101	117	133	229	245
0101			%	5	E	U	e	u			♣
6	6	22	38	54	70	86	102	118	134	230	246
0110			&	6	F	V	f	v			♥
7	7	23	39	55	71	87	103	119	135	231	247
0111			'	7	G	W	g	w			♦
8	8	24	40	56	72	88	104	120	136	232	248
1000			(	8	H	X	h	x			♠
9	9	25	41	57	73	89	105	121	137	233	249
1001			)	9	I	Y	i	y			■
A	10	26	42	58	74	90	106	122	138	234	250
1010			*	:	J	Z	j	z			□
B	11	27	43	59	75	91	107	123	139	235	251
1011			+	;	K	[	k	{			π
C	12	28	44	60	76	92	108	124	140	236	252
1100			,	<	L	\	l				√
D	13	29	45	61	77	93	109	125	141	237	253
1101			-	=	M	]	m	}			
E	14	30	46	62	78	94	110	126	142	238	254
1110			.	>	N	^	n	~			
F	15	31	47	63	79	95	111	127	143	239	255
1111			/	?	O	_	o				

The figure below shows the key layout and the values returned when using INKEY\$. Most of the codes are given in the key code chart but this figure gives the non-printable key codes too.



INKEY\$ Character Code Table

Low \ High	0	1	2	3	4	5	6
0		SHIFT	SPC	0		P	
1				1	A	Q	
2	CLS	DEF		2	B	R	
3				3	C	S	
4	↑			4	D	T	
5	↓			5	E	U	
6		SML		6	F	V	
7				7	G	W	
8	MODE		(	8	H	X	
9			)	9	I	Y	
A			*	:	J	Z	
B	INS		+	;	K		
C	DEL		,	'	L		
D	ENTER		-	=	M		
E	▶		.		N		
F	◀		/		O		

**Appendix C****Formatting Output**

It is sometimes important or useful to control the format as well as the content of output. The PC-1360 controls display formats with the USING verb. This allows you to specify:

- The number of digits
- The location of the decimal point
- Scientific notation format
- The number of string characters

These different formats are specified with an “output mask.” This mask may be a string constant or a string variable:

```
10: USING “#####”
```

```
20: M$=“&&&&&&”
```

```
30: USING M$
```

When USING is used with no mask, all special formatting is cancelled.

```
40: USING
```

USING may also be used within a PRINT statement:

```
50: PRINT USING M$; N
```

Wherever USING is used, it will control the format of all output until a new USING is encountered.

**Numeric Masks**

A numeric USING mask may only be used to display numeric values, i.e., numeric constants or numeric variables. If a string constant or variable is displayed while a numeric USING mask is in effect, the mask will be ignored. A value which is to be displayed must always fit within the space provided by the mask. The mask must reserve space for the sign character, even when the number will always be positive. Thus a mask which shows four display positions may only be used to display numbers with three digits.

The MDF function can be used with USING to round decimal numbers before printing out. MDF enables values to be held in the computer to the same number of decimal places as expressed in the USING statement. Values are usually rounded up to the next whole number if the truncated part is 5 or greater. MDF allows this threshold rounding up value to be changed.

## Specifying Number of Digits

The desired number of digits is specified using the '#' character. Each '#' in the mask reserves space for one digit. The display or print always contains as many characters as are designated in the mask. The number appears to the far right of this field; the remaining positions to the left are filled with spaces. Positive numbers therefore always have at least one space at the left of the field. Since the PC-1360 maintains a maximum of 10 significant digits, no more than 11 '#' characters should be used in a numeric mask. When the total number of columns of the integer part specified exceed 11, this integer part is regarded as 11 digits in the PC-1360.

**Note:** In all examples in this appendix the beginning and end of the displayed field will be marked with a '|' character to show the size of the field.

<u>Statement</u>	<u>Display</u>
10: USING "####"	(Set the PC-1360 to the RUN mode, type RUN, and press <b>ENTER</b> .)
20: PRINT 25	25.
30: PRINT -350	-350.
40: PRINT 1000	<b>ERROR 7 IN 40</b>

Notice that the last statement produced an error because 5 positions (4 digits and a sign space) were required, but only 4 were provided in the mask.

## Specifying a Decimal Point

A decimal point character, '.', may be included in a numeric mask to indicate the desired location of the decimal point. If the mask provides more significant

decimal digits than are required for the value to be displayed, the remaining positions to the right will be filled with zeros. If there are more significant decimal digits in the value than in the mask, the extra digits will be truncated (not rounded):

<u>Statement</u>	<u>Display</u>
10: USING "####.##"	
20: PRINT 25	25.00
30: PRINT -350.5	-350.50
40: PRINT 2.547	2.54

## Specifying Scientific Notation

A '^' character may be included in the mask to indicate that the number is to be displayed in scientific notation. The '#' and '.' characters are used in the mask to specify the format of the "characteristic" portion of the number, i.e., the part which is displayed to the left of the E. Two '#' characters should always be used to the left of the decimal point to provide for the sign character and one integer digit. The decimal point may be included, but is not required. Up to 9 '#' characters may appear to the right of the decimal point. Following the characteristic portion, the exponentiation character, E, will be displayed followed by one position for the sign and two positions for the exponent. Thus, the smallest scientific notation field would be provided by a mask of "##^" which would print numbers of the form '2E 99'. The largest scientific notation field would be "##.#####^" which would print numbers such as '-1.234567890E -12':

<u>Statement</u>	<u>Display</u>
10: USING "##.##^"	
20: PRINT 2	2.00E 00
30: PRINT -365.278	-3.65E 02



## Specifying Alphanumeric Masks

String constants and variables are displayed using the ‘&’ character. Each ‘&’ indicates one character in the field to be displayed. The string will be positioned at the left end of this field. If the string is shorter than the field, remaining spaces to the right will be filled with spaces. If the string is longer than the field, the string will be truncated to the length of the field:

<u>Statement</u>	<u>Display</u>
10: USING “&&&&&”	
20: PRINT “ABC”	<b>ABC</b>
30: PRINT “ABCDEFGHI”	<b>ABCDEF</b>

## Mixed Masks

In most applications a USING mask will contain either all numeric or all string formatting characters. Both may be included in one USING mask, however, for certain purposes. In such cases, each switch from numeric to string formatting characters or vice versa marks the boundary for a different value. Thus, a mask of “#####&&&&” is a specification for displaying two separate values—a numeric value which is allocated 5 positions and a string value which is allocated 4 positions:

<u>Statement</u>	<u>Display</u>
10: USING “#####.###&&”;25;“CR”	<b>25.00CR</b>
20: PRINT –5.789;“DB”	<b>–5.78DB</b>

**Remember:** Once specified, a USING format is used for all output which follows until cancelled or changed by another USING verb.

**Appendix D****Expression Evaluation and Operator Priority**

When the SHARP PC-1360 is given a complex expression, it evaluates the parts of the expression in a sequence which is determined by the priority of the individual parts of the expression. If you enter the expression:

$$100/5 + 45$$

as either a calculation or as a part of a program, the PC-1360 does not know whether you mean:

$$\frac{100}{5 + 45} = 2 \quad \text{or} \quad \frac{100}{5} + 45 = 65$$

Since the PC-1360 must have some way to decide between these options, it uses its rules of operator priority. Because division has a higher "priority" than addition (see below), it will choose to do the division first and then the addition, i.e., it will choose the second option and return a value of 65 for the expression.

**Operator Priority**

Operators on BASIC of the SHARP PC-1360 are evaluated with the following priorities from highest to lowest:

Level	Operations
1	Parentheses
2	Variables and Pseudovariables
3	Functions
4	Exponentiation (^)
5	Unary minus, negative sign (-)
6	Multiplication and division (*, /)
7	Addition and subtraction (+, -)
8	Relational operators (<, <=, =, <>, >=, >)
9	Logical operators (AND, OR, NOT)

When there are two or more operators at the same priority level the expression will be evaluated from left to right. (The exponentiation will be evaluated from right to left). Note that with  $A+B-C$ , for example, the answer is the same whether the addition or the subtraction is done first.

When an expression contains multiple nested parentheses, the innermost set is evaluated first and evaluation then proceeds outward.

For level 3 and 4, the last entry has a higher priority.

For example:  $-2^4 \rightarrow -(2^4)$   
 $3^{-2} \rightarrow 3^{-2}$

## Sample Evaluation

Starting with the expression:

$$((3+5-2)*6+2)/10^{\text{LOG } 100}$$

The PC-1360 would first evaluate the innermost set of parentheses. Since '+' and '-' are at the same level it would move from left to right and would do the addition first:

$$((8-2)*6+2)/10^{\text{LOG } 100}$$

Then it would do subtraction:

$$((6)*6+2)/10^{\text{LOG } 100}$$

or:

$$(6*6+2)/10^{\text{LOG } 100}$$

In the next set of parentheses it would do the multiplication first:

$$(36+2)/10^{\text{LOG } 100}$$

And then the addition:

$$(38)/10^{\text{LOG } 100}$$

or:

$$38/10^{\text{LOG } 100}$$

Now that the parentheses are cleared, the LOG function has the highest priority so it is done next:

$$38/10^2$$

The exponentiation is done next:

$$38/100$$

And last of all the division is performed:

$$0.38$$

This is the value of the expression.

## Appendix E

## Key Functions in BASIC

**ON**  
**BRK**

(ON)

Use to turn the PC-1360 power on when the auto power off function is in effect.

(BREAK)

- Pressing this key during program execution functions as a BREAK key and causes the program to interrupt execution.
- When pushed during manual execution, execution of such I/O commands as BEEP and CLOAD is interrupted.

**SHIFT**

- The yellow key marked "SHIFT" must be used to designate a second function. (appearing above each key.)

Ex. **SHIFT** <sup>?</sup>**U** → ? is entered.

- When storing something in or recalling something from the ReSerVe mode, press before the labelled key.

**CLS**

- Use to clear the contents of the entry and the display.
- Use to reset the error.

**SHIFT CA**

- Not only clears the display contents, but resets the computer to its initial state.  
—Initial state—
  - Resets the WAIT timer.
  - Resets the display format. (USING format)
  - Resets the TRON state (TROFF).
  - Resets PRINT = LPRINT.
  - Resets error.

**MODE**

- Use to change the operational mode selection from RUN to PROgram or from PROgram to RUN.

**SHIFT MODE**

- Use to set the ReSerVe mode.

**0** ~ **9**

- Numeric keys.

**.**

- Decimal point.
- Use to enter an abbreviation of a command/verb/function.
- Use to designate the decimal portion in USING format designation.

**E**

- Use to designate an exponent in scientific notation. (This key is a letter E key: upper case)

**/**

- Division key.

**\***

- Multiplication key.
- Use to designate an array variable in INPUT#, PRINT#, etc.

**+**

- Addition key.

**-**

- Subtraction key.

**SHIFT** **^**

- Use for power calculation instructions.
- Use to specify the exponent display system for numerical data in USING statement instructions.

**SHIFT** **<****SHIFT** **>**

- Use when entering logical operations in IF sentences.

**DEF**

- When any one of eighteen keys (A,S,D,F,G,H,J,K,L,=,Z,X,C,V,B,N,M,SPaCe) is pushed after pressing the **DEF** key, it starts to execute the program from the program line that has the same label as the key code pressed.

**A** ~ **Z**

- Alphabet keys. You are probably familiar with these keys from the standard typewriter keyboard. If pressed as is, an upper case character is displayed. If an alphabet key is pressed after pressing **SML**, a lower case character is displayed.

**SPC**

- Use to provide space when entering programs or characters.

=

- In assignment statements, use to assign the contents (number or character) on the right for the variable specified on the left.
- Use when entering logical operators in IF sentences.

SHIFT

!

"

#

\$

%

&amp;

SHIFT

@

- Use to designate these symbols.
  - "": ● Use to designate and cancel characters.
  - #: ● Use to specify labels.
  - #: ● Use with USING statement, to provide the instruction to define the display format of numerical data.
  - \$: ● Use when assigning character variables.
  - &: ● Use with USING statement, to provide the instruction to define the display format of character string.
  - @: ● Use to designate hexadecimal numbers.
  - @: ● Use for reserve contents when the reserve key is used as a program key.
    - Example: GOTO 100 @
  - !%: ● Use as a character string within " ".

SHIFT

?

- Use to enter CLOAD?

:

- Use to divide two or more statements in one line.

,

- Use to provide pause between two equations, and between variables or comments.

;

- Use to provide multi-display (two or more values/contents displayed at a time).
- Use to provide pause between the instruction and the variable.

(


)

- Use to enter parentheses.

▶

- Shifts the cursor to the right (press once to advance one position, hold down for automatic advance).
- Executes playback instructions.
- Call the cursor if not displayed while the contents are displayed.
- Clears an error condition in manual operation.



- Shifts the cursor to the left (press once to advance one position, hold down for automatic advance).
- Otherwise the same as the  key.

**INS**

- Inserts one space (□ appears) of 1-step capacity between the address (N) indicated by the cursor and the preceding address (N-1).

**DEL**

- Deletes the contents of the address indicated by the cursor.

**SHIFT** 

- Use to designate pi ( $\pi$ ).

**SHIFT** 

- Use to designate square root.

**ENTER**

- Enters a program line into the computer.
- Use when writing programs.
- Requests manual calculation or direct execution of a command statement by the computer.
- Use to restart a program temporarily stopped by an INPUT or PRINT command.

P  $\longleftrightarrow$  NP**SHIFT** **ENTER**



- Use to set print and non-print mode when an optional printer is connected.

**SML**

- Specifies and releases the lower case mode. (Turns on and off the SML symbol display.)
- The SML symbol is displayed when **SML** is pressed. If **A** **B** and **C** are pressed, a b c is displayed. If **SML** is pressed again, the SML symbol disappears and upper case characters are entered.



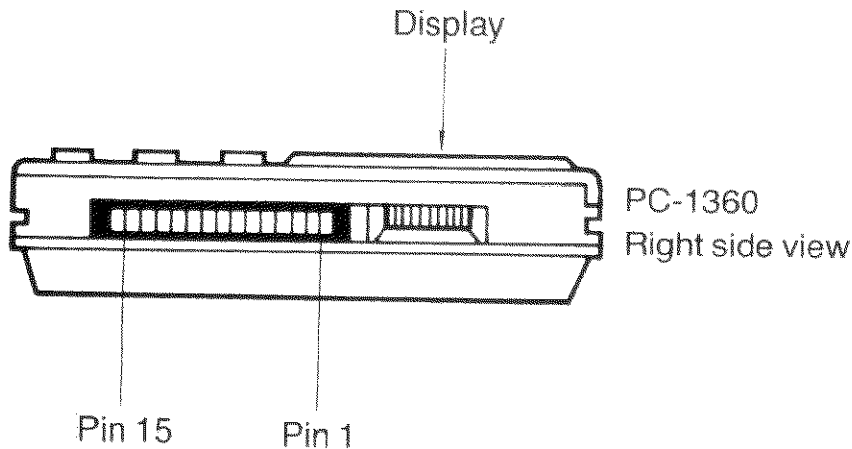
The  and  keys have the following functions, depending on designated modes, as well as the state of the computer.

Mode	State			
RUN	Program being executed	/		
	Program is temporarily interrupted	To execute the next line	To display program line being executed or already executed, hold this key down.	
	PRINT, LINE, PRESET, PSET, GPRINT statement being executed			
	PRINT statement just now executed			
	Under break	non function		
	INPUT statement being executed	non function		
	Error condition during executing program	/		To display error-producing line, hold this key down.
TRON condition	To execute debugging operation	To display program line being executed or already executed, hold this key down.		
PRO	(When the mode is changed from RUN to PRO and program line is not being displayed)			
	Program is temporarily interrupted.	To display the line interrupted	Same as left.	
	Error condition	To display the line with error	Same as left	
	Other condition	To display the first line	To display the last line	
	(When the program line is being displayed)	To display the next program line	To display the preceeding program line	
RESERVE	/		/	

- On the display the **ENTER** key is the same as a space.
- If no key is entered in the key input request mode for approximately 11 minutes, the power is automatically turned off (auto-power off function).

**Appendix F****Signals Used in the Serial I/O Terminal**

The PC1360 is equipped with a 15-pin connector for the serial I/O terminal. The pins used and their signals are described below.

**Pin Connections Used**

Pin	Name	Symbol	I/O	Function
1	Frame Ground	FG	—	Protective chassis ground
2	Send Data	SD	Out	DC output signal
3	Receive Data	RD	In	DC input signal
4	Request to Send	RS	Out	HIGH: Sends carrier
5	Clear to Send	CS	In	HIGH: Transmission enabled
7	Signal Ground	SG	—	Reference 0 voltage for all signals
8	Data Carrier Detect	CD	In	HIGH: Carrier signal received
10		VC		Power supply
11	Receive Ready	RR	Out	HIGH: Receive enabled
12	Peripheral Acknowledge	PAK	In	See Note 3.
13		VC	—	Power supply
14	Data Terminal Ready	ER	Out	HIGH: Local terminal ready
15	Peripheral Request	PRQ	Out	See Note 3.

- Note:
1. HIGH: VC voltage level; LOW: SG voltage level.
  2. The computer uses CMOS components. Application of voltages exceeding the allowable range, i.e., voltage level between SG and VC, may damage the computer.
  3. This signal confirms if the connected terminal is the CE-140P printer or not. This signal is not used for communication with other terminals.

**Appendix G****Specifications**

<b>Model:</b>	PC-1360 Pocket Computer	
<b>Processor:</b>	8 bit CMOS CPU	
<b>Programming Language:</b>	BASIC	
<b>System ROM:</b>	136 K Bytes	
<b>Memory Capacity:</b>	RAM:	
	System internal	960 Bytes
	System area	1282 Bytes
held in	User	
RAM card	Fixed Memory Area	208 Bytes
(8KB)	(A ~ Z, A\$ ~ Z\$)	
	Program/Data Area	6558 Bytes
	Reserve Area	144 Bytes
		can be extended to 64KB using the two slots
<b>Stack:</b>	Sub-routine: 10 stacks	Function: 16 stacks
	FOR-NEXT: 5 stacks	Data: 8 stacks
<b>Operators:</b>	Addition, subtraction, multiplication, division, trigonometric and inverse trigonometric functions, logarithmic and exponential functions, angle conversion, square and square root, sign, absolute, integer, relational operators, logical operators etc...	
<b>Numeric Precision:</b>	10 digits (mantissa) + 2 digits (exponent).	
<b>Editing Features:</b>	Cursor left and right, line up and down, character insert, character delete.	
<b>Memory Protection:</b>	CMOS Battery backup.	
<b>Serial Input/Output Features:</b>	Standards:	Start-stop transmission (asynchronous) system. Only half duplex.
	Baud Rates:	300, 600, 1200 Baud
	Data Bits:	7 or 8 Bits
	Parity Bits:	Even, odd, or no-parity
	Stop Bit:	1 or 2 Bits
	Connectors Used:	15-pin connector (for external equipment)

Output Signal Level: C-MOS level (4–6 Volts)  
 Interfacing Signals: Inputs: RS, CS, CD  
 Outputs: SD, RS, RR, ER  
 Others: SG, FG, VC

**Printer interface capability:**

11 pin (For CE-126P, CE-124)

**Graphics interface capability:**

CE-140P commands (CIRCLE, PAINT etc.).

**2 Ram Card Slots:**

each card 2KB, 4KB, 8KB, 16KB, or 32 KB.

**Display:**

4-line 24-digit liquid crystal display with 5×7 dot characters or 150×32 dot graphics.

**Keys:**

62 keys. Alphabetic, numeric, special symbols, and functions. Numeric pad. User defined keys.

**Power Supply:**

6.0 V DC: Lithium cells.

type: CR-2032×2

**Power Consumption:**

6.0 V DC @ 0.03W

Approximately 120 hours of continuous operation under normal conditions (based on 10 minutes of operation or program execution and 50 minutes of display per hour at a temperature of 20°C). The time may vary slightly depending on usage and the type of battery used.

**Operating****Temperature:**

0°C – 40°C (32°F – 104°F)

**Dimensions:**

182(W) × 72(D) × 16(H) mm.

7<sup>5</sup>/<sub>32</sub>" (W) × 2<sup>27</sup>/<sub>32</sub>" (D) × 5<sup>5</sup>/<sub>8</sub>" (H)

**Weight:**

Approximately 220g (0.49 lbs.) (with cells and a RAM card)

**Accessories:**

Hard cover, one 8KB RAM card (CE-212M), two lithium cells (built in), one keyboard template and operation manual.

**Options:**

Plug-in RAM cards 2 KB (CE-210M), 4 KB (CE-211M)  
 8 KB (CE-212M), 16 KB (CE-2H16M), 32 KB (CE-2H32M)

Cassette Tape Recorder (CE-152)

Printer/Cassette Interface (CE-126P)

Printers (CE-140P, CE-515P, CE-516P) etc.

**Copy Capability:**

RAM cards can be copied from slot 1 to slot 2.

**Appendix H****Using Programs Developed for other SHARP Models on the PC-1360**

This chapter includes information for program compatibility for the following models:

PC-1260, PC-1261

PC-1350

PC-1401, PC-1402

PC-1450, PC-1460

**PC-1260, PC-1261**

1. These models do not feature the 4 line display of the PC-1360. Use the CLS command to clear the display after each one or two line output.
2. Any character code 96 (&60) specifying a space must be changed to code 32 (&20) as code 96 specifies a single quote on the PC-1360. These codes usually appear in CHR\$ commands.



**PC-1350**

Programs made on the PC-1350 is basically usable on the PC-1360. When the PC-1360 reads into the PC-1350 programs on tape, conversion is automatically executed. But, depending on programs, the case that generates memory over, and the case that cannot reserve variables sometimes occur because program size sometimes become too large. Also, depending on programs, it generates an error when one line consists of over 80 bytes. In order to use the PC-1360 programs on the PC-1350, it is possible to SAVE on tape by the command below:

CSAVE@ ["file name"][, "PASSWORD"] **ENTER**

**Note:** It is not possible to SAVE the reserve contents by the command above because it is different between the two computers.

## PC-1401, PC-1402

1. The  key is not a defined key in this computer. For programs which define the  key, define another key. For example:

100 “,”: (change to) 100 “=”:

2. The PC-1401 has more function commands than the PC-1360. Errors will occur if the program on the PC-1360 is found to contain these commands that cannot be recognized. As the program is read into the PC-1360, the commands will be displayed with “~” symbol in place of the command.

## PC-1450, PC-1460

These models do not feature the 4 line display of the PC-1360. Use the CLS command to clear the display after each one or two line output.





# Program Examples

Having read the description of each of the various functions in the preceding chapters, you have by now gained a knowledge of a number of program commands. However, in order for you to have a command of developing application programs in BASIC language, it is absolutely necessary that you write and execute your own practical application programs as well as those explained in this manual.

Just as you can improve your driving skill by actually operating the steering wheel or your tennis game by swinging the racket, proficiency in programming can only be attained by practicing as many programs as possible, regardless of the degree of your skill at each practice.

It is also very important for you to refer to programs developed by others. In this chapter, some programming examples using various commands in "BASIC" language are introduced to your reference.

For better understanding of the programming examples in this chapter, the conventions used in such examples are explained as follows:

## 1. PROGRAM LIST

All the program lists contained in the programming examples are provided using the hard-copy outputs from the CE-126P or CE-140P printer in actual size or in 55% reduced size.

## 2. PROGRAM CAPACITY

At the end of each program list, the capacity of the program itself is indicated in number of bytes.

## 3. PRINTOUT

For a program requiring a printout, the output of the program executed using the CE-126P or CE-140P printer is given in actual size or in 55% reduced size. A printout reduced to other than 55% is indicated with the different reduction rate used.

## 4. MEMORY CONTENTS

In the table of memory contents in each program example, variables with predetermined use are indicated by their specific use and those without predetermined use (e.g., those to be stored in the work area to retain intermediate results of a calculation, etc.) are indicated by the checkmark "√".

## 5. Then using CE-515P (or CE-516P)

In the table of contents of this chapter, "P" preceding a program title indicates that the CE-140P, CE-515P, or CE-516P may be used as a peripheral unit in executing the program. However, when using the CE-515P or CE-516P, observe the following points:

## 1. DIP switch setting

- With CE-515P ..... Set all the DIP switch pins to the OFF position.
- With CE-516P ..... Set the No. 1 to 5 switch pins to the OFF position and the No. 6 switch pin to the ON position.

## 2. Paper to be used

Use a paper roll in either case.

**NOTE:** When the CE-515P or CE-516P is used, its hard-copy outputs differ partially from those of the CE-140P in the following:

## (1) Color

CE-140P: purple → CE-515P (or CE-516P):

## (2) Character

CE-140P: a → CE-515P: α

(CE-516P prints the same character as CE-140P.)

SHARP CORPORATION and/or its subsidiaries assume no responsibility or obligation for any financial losses or damages that may be incurred from using any of the examples of programs described in this manual. When using these programs, be aware that these programs may not fully satisfy your purpose or some programs may not be as precise as you wish them to be. Therefore, please carefully check the data in each of the program examples you use and confirm that they meet your requirements. If not, please modify them as required to meet your purpose before using them.

## CONTENTS

(Title)	(Page)
1. COMPARATIVE BELT GRAPH [P] .....	314
2. MODIFIED MOVING AVERAGE .....	322
3. TRANSFER OF PROGRAM FILE .....	325
4. ROUND GRAPHIC [P] .....	329
5. SKI JUMP .....	333

# Program Title: COMPARATIVE BELT GRAPH

Color Dot Printer  
CE-140P Required

## ■ OVERVIEW

This graph is used to compare the ratio by per cent of items in total yearly import.

## ■ INSTRUCTIONS

1. Program starts at **DEF A**.
2. Input the number of belts between the range of 1-25.
3. According to the display, input as follows; name of belts (within seven characters), the number of items (between the range of 2-12), and the name of items (within seven characters).
4. Following 3., input the data of each items. It is possible to input up to eight digits number.
5. Input the graph title (within eight characters). The program will end after printing out the data chart and the comparative belt graph.

## ■ REMARKS

It is possible to change the pattern and the color of graph by changing the DATA in line 560.

CL	0	1	2	3	4	5	6
COLOR	BLACK	PURPLE	RED	MAGENTA	GREEN	CYAN	YELLOW

PT	1	2	3
PATTERN	HORIZONTAL	VERTICAL	CROSSED STRIPES

If you set 42, vertical green lines will be printed out.

## ■ EXAMPLE

Comparative belt graph will be made by the data below.

Number of belts: 6

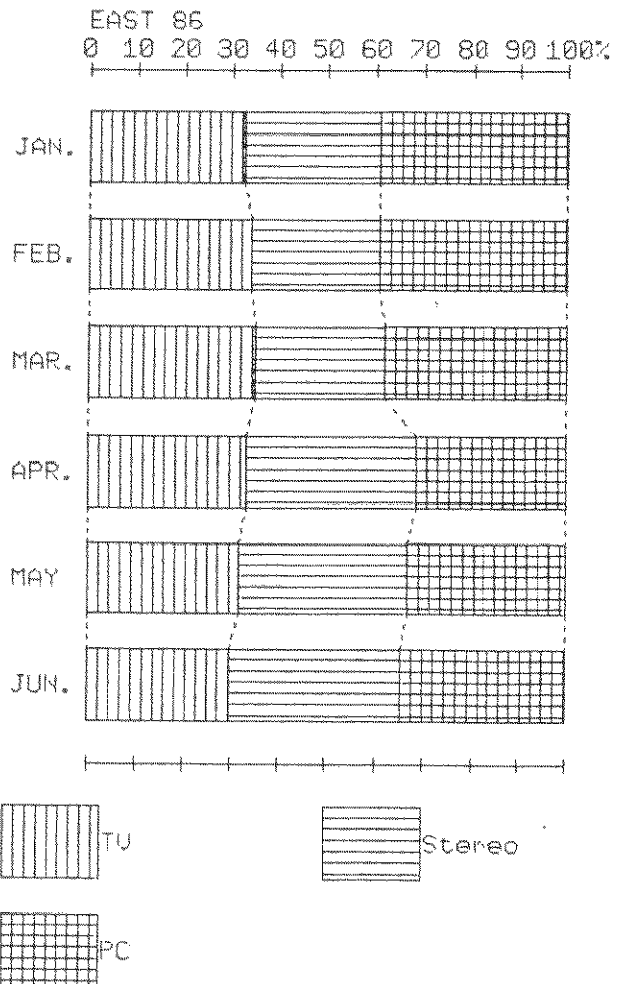
Number of items: 4

Graph Title: EAST 86

Month	Items	TV	Stereo	PC
JAN.		30	26	36
FEB.		28	22	32
MAR.		26	20	28
APR.		34	36	32
MAY		29	32	30
JUN.		25	30	29

■ PRINTED OUTPUT

	TU	Stereo	PC
JAN.	30	26	36
FEB.	28	22	32
MAR.	26	20	28
APR.	34	36	32
MAY	29	32	30
JUN.	25	30	29



# ■ KEY OPERATION

1) **DEF A**

NO. OF BELT = \_

2) **6 ENTER**

NAME OF BELT (1) =  
?

3) **JAN. ENTER**

NAME OF BELT (2) =  
?

⋮  
Input as above

6) **JUN. ENTER**

NO. OF ITEM = \_

7) **3 ENTER**

NAME OF ITEM (1) =  
?

8) **TV ENTER**

NAME OF ITEM (2) =  
?

⋮  
Input as above

10) **PC ENTER**

JAN.  
TV

DATA = \_

11) **30 ENTER**

JAN.  
STEREO

DATA = \_

⋮  
Input as above

26) 29 **ENTER**

GRAPH TITLE =  
?

27) EAST 86 **ENTER**

>

# PROGRAM LIST

```

10:"A": WAIT 0: CLEAR : CLS : USING
20:INPUT "NO. OF BELT=";YK
30:IF YK<1 OR YK>25 BEEP 2: GOTO 10
40:DIM Y$(YK),T$(0)*18,YT(YK)
50:FOR I=1 TO YK
60:CLS : PRINT "NAME OF BELT("; STR$ I
;")=": INPUT Y$(I)
70:IF LEN Y$(I)>> BEEP 2: GOTO 60
80:NEXT I
90:CLS : INPUT "NO. OF ITEM=";XK
100:IF XK<2 OR XK>12 BEEP 2: GOTO 90
110:DIM X$(XK),DD(XK,YK),XT(XK),CX(XK-1
),HP(XK)
120:FOR I=1 TO XK
130:CLS : PRINT "NAME OF ITEM("; STR$ I
;")=": INPUT X$(I)
140:IF LEN X$(I)>> BEEP 2: GOTO 130
150:NEXT I
160:FOR Y=1 TO YK: FOR X=1 TO XK
170:CLS : PRINT Y$(Y): PRINT X$(X)
180:D=0: INPUT "DATA=";D
190:D=INT D: IF LEN STR$ D>8 BEEP 2:
GOTO 170
200:DD(X,Y)=0: NEXT X: NEXT Y
210:CLS : T$(0)="": PRINT "GRAPH TITLE="
: INPUT T$(0)
220:IF LEN T$(0)>8 BEEP 2: GOTO 210
230:CLS : GOSUB 1170
240:S1=INT (XK/5)+1:X=0:A=96
250:FOR O=1 TO S1
260:S2=4: IF O=S1 LET S2=XK-5*O+5
270:GLCURSOR (0,-20): SORGN
280:X2=AK(S2+1)
290:LLINE -(X2,0),L1
300:FOR I=0 TO S2+1
310:LLINE (X2-I*A,0)-(X2-I*A,-24),L1:
NEXT I
320:FOR I=0 TO S2
330:GLCURSOR (A*I+8,-18): LPRINT "P";X$
(X+I): NEXT I
340:FOR Y=1 TO YK: GLCURSOR (0,-24):
SORGN
350:RLINE -(X2,0),L1
360:FOR I=0 TO S2+1
370:LLINE (X2-I*A,0)-(X2-I*A,-24),L1:
NEXT I
380:FOR I=0 TO S2
390:IF X+I=0 GLCURSOR (A*I+8,-18):
LPRINT "P";Y$(Y): GOTO 410
400:GLCURSOR (A*I-5,-18): LPRINT CHR$ 2
7;"c11": LPRINT "P"; USING "#####
##";DD(X+I,Y)
410:LPRINT CHR$ 27;"c12": NEXT I: NEXT
Y
420:LLINE (X2,-24)-(0,-24),L1: SORGN
430:X=X+4: NEXT O
440:LTEXT
450:ZX=350:ZY=500
460:MS=0: FOR K=1 TO YK:L=LEN Y$(K):
IF L>MS LET MS=L
470:NEXT K: IF MS=1 LET MS=2
480:PY=-60:X=MS*12+6
490:IF (ZX+MS*12+5L)>480 LET ZX=480-MS*K
12-5L
500:IF (ZX+X+36)>480 LET ZX=480-X
510:PX=INT ((480-ZX)/2): IF PX<X LET P
X=X
520:XH=INT (ZX/10):PA=XH/10
530:KH=INT (ZY/(YK*2+(YK+1))):OB=KH*2
540:IF OB>75 LET OB=75:KH=38
550:RESTORE 560: FOR I=1 TO XK: READ H
P(I): NEXT I
560:DATA 11,22,33,02,12,23,31,01,13,21,
32,03
570:FOR K=1 TO XK:XT(K)=0: FOR I=1 TO Y
K:XT(K)=XT(K)+DD(K,I): NEXT I: NEXT
K
580:FOR K=1 TO YK:YT(K)=0: FOR I=1 TO X
K:YT(K)=YT(K)+DD(I,K): NEXT I: NEXT
K
590:GOSUB 1170: USING : GLCURSOR (PX,-
30): CSIZE 2: LPRINT "P";T$(0)
600:GLCURSOR (PX,PY): SORGN
610:YU=5:YD=-5:XM=-XH:L=12: IF ZX<300
LET L=6
620:CSIZE 2
630:FOR K=0 TO 10:XM=XM+XH:AX=.4*KL
640:IF K)=1 LET AX=.9*KL
650:IF K=10 LET AX=1.3*KL
660:GLCURSOR (XM-AX,YU*2)
670:LPRINT "P"; STR$ (K*10): IF K=10
LPRINT "P%"
680:LLINE (XM,YU)-(XM,YD),L1: NEXT K

```



```

690:LLINE (XM,YD+YU)-(0,0),L1: CSIZE 2
700:SY=OB-4
710:FOR KA=1 TO YK: SX=0: SY=SY-OB-KH
720:MX=-(MS:K12+6): MY=SY-KH-6
730:GLCURSOR (MX,MY): LPRINT "P";Y$(KA)
740:TT=0: PN=0
750:FOR K=1 TO XK: PN=PN+1: TT=TT+DD(K,KA
)
760:IF YT(KA)=0 LET EX=0: GOTO 780
770:PC=TT/YT(KA)*100: EX= INT (PC*PA)
780:IF EX=SX AND K<>1 THEN 820
790:LLINE (SX,SY)-(EX,SY),L1
800:LLINE -(EX,SY-OB),L1
810:LLINE -(SX,SY-OB),L1
820:IF K=1 RLINE -(0,OB),L1
830:IF KA=1 THEN 910
840:L1=2
850:IF K=XK LLINE (EX,SY)-(EX,SY+KH),L1
860:IF K=XK THEN 900
870:IF K=1 LLINE (SX,SY)-(SX,SY+KH),L1
880:LLINE (CX(K),SY+KH)-(EX,SY),L1
890:LLINE -(EX,SY),L1
900:L1=0
910:IF EX<>SX GOSUB 1190
920: SX=EX: IF K<XK LET CX(K)=EX
930:NEXT K: NEXT KA
940: SX=0: SY=SY-OB-KH-5
950:GLCURSOR (SX,SY): SORGN
960: YU=5: YD=-5: XM=-XH: SW=1
970:FOR K=0 TO 10: XM=XM+XH
980:IF SW<>1 LLINE (XM,YD)-(XM,YU),L1:
GOTO 1000
990:LLINE (XM,YU)-(XM,YD),L1
1000:SW=-SW: NEXT K
1010:LLINE (XM,YD+YU)-(0,0),L1
1020:GLCURSOR (0,-KH-5): SORGN
1030:SY=0: S1=1: FOR PN=1 TO XK
1040:IF INT S1<0 LET SX=XH*5: EX=SX+XH*K
2: GOTO 1060
1050: SX=-50: EX=XH*2-60
1060:LLINE (SX,SY)-(EX,SY),L1
1070:LLINE -(EX,SY-OB),L1
1080:LLINE -(SX,SY-OB),L1
1090:LLINE -(SX,SY),L1
1100:GOSUB 1190: X=EX+3: Y=SY-OB/2-6
1110:GLCURSOR (X,Y): LPRINT "P";X$(PN)
1120:IF S1<0 LET SY=SY-OB-KH
1130:S1=-S1: NEXT PN
1140:SORGN
1150:GLCURSOR (0,-100)
1160:LPRINT CHR$ 27+"0": END
1170:LPRINT : LPRINT : CSIZE 2: LPRINT
: LTEXT : LPRINT : GRAPH
1180:COLOR 0: L1=0: SORGN : RETURN
1190:CL= INT (HP(PN)/10): PT=HP(PN)-CL*
10: EY=SY-OB
1200:COLOR CL,7
1210:ON PT GOSUB 1240,1290,1340
1220:COLOR 0
1230:RETURN
1240:HH=8: SW=1
1250:FOR X=SX+HH TO EX STEP HH: IF X<E
X THEN 1280
1260:IF SW<>1 LLINE (X,EY)-(X,SY),L1:
GOTO 1280
1270:LLINE (X,SY)-(X,EY),L1
1280:SW=-SW: NEXT X: RETURN
1290:HH=8: SW=1
1300:FOR Y=SY-HH TO EY STEP -HH: IF Y<
EY THEN 1330
1310:IF SW<>1 LLINE (EX,Y)-(SX,Y),L1:
GOTO 1330
1320:LLINE (SX,Y)-(EX,Y),L1
1330:SW=-SW: NEXT Y: RETURN
1340:GOSUB 1240: GOSUB 1290: RETURN

```

3207 bytes

## ■ MEMORY CONTENTS

Variables	Contents	Variables	Contents
A	✓	PX	origin of X
D	input data	PY	origin of Y
I	loop counter	S1	✓
K	loop counter	S2	✓
L	✓	SW	flag
M	✓	SX	graph print
O	loop counter	SY	graph print
W	✓	TT	✓
X	✓	X2	✓
Y	✓	XH	✓
AX	gauge print	XK	number of item
CL	color set	XM	gauge print
EX	graph print	YD	gauge print
EY	graph print	YK	number of belt
HH	width of hatching	YU	gauge print
KA	loop counter	ZX	length of belt
KH	width between belts	ZY	✓
LI	✓	CX (XK-1)	joint data
MS	✓	HP (XK)	hatching data
MX	print items	XT (XK)	total by each item
MY	print items	YT (XK)	total by each belt
OB	width of belt	DD (XK, YK)	data
PA	✓	T\$(O)	title
PC	print belt	X\$(XK)	name of item
PN	loop counter	Y\$(YK)	name of belt
PT	for hatching		

## ■ LINE MAP

Line Number	Contents
10	initialization
20~ 40	number of belt input
50~ 80	name of belt input
90~ 110	number of item input
120~ 150	name of item input
160~ 200	data input
210~ 220	graph title input
230~ 430	data chart print
440~ 580	calculation
590	graph title print
600~ 690	upper gauge print
700~ 730	graph print
940~1000	lower gauge print
1010~1130	notes print
1140~1150	end
1160	printer initialize sub
1170~1230	hatching pattern processing
1240~1280	horizontal hatching sub
1290~1330	vertical hatching sub
1340	horizontal/vertical hatching sub

# Program Title: MODIFIED MOVING AVERAGE

Thermal printer  
cassette interface  
CE-126P required

## ■ OVERVIEW

Even changes difficult to read can be grasped at a glance!

Modified moving average is used for when you search for the average point at all fixed parts in the data and when you want to know overall data trends.

Using this technique highlights long term trends in such volatile indicators such as sales return figures, stock prices and exchange rates.

## ■ INSTRUCTIONS

1. Program starts by keying in **DEF A**. Input the number of items, and input further elements. By inputting each element the input data will be output to the printer, and also the input data and the average values will be output to the printer from  $n + 1$ st.
2. To terminate the program, press just the **ENTER** key in response to a wait for data input.

## ■ CONTENTS

Processing varies with the number of averaging items ( $n$ ) being an odd number or even number.

1.  $n$  is an odd number.

$$\bar{X}_t = \sum_{i=1}^n X_i/n$$

$$\bar{X}_1 = \sum_{i=2}^{n+1} X_i/n$$

$$\vdots$$

2.  $n$  is an even number:

$$\bar{X}_1 = \left( \frac{X_1}{2} + \frac{X_{n+1}}{2} + \sum_{i=2}^n X_i \right) / n$$

$$\bar{X}_2 = \left( \frac{X_2}{2} + \frac{X_{n+2}}{2} + \sum_{i=3}^{n+1} X_i \right) / n$$

⋮

## ■ EXAMPLE

Input the following data and search for the modified moving average. Set the number of items at 4.

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$	$X_{11}$	$X_{12}$	$X_{13}$
82	15	8	53	0	41	22	3	18	75	64	39	46

## ■ PRINTED OUTPUTS

```
* z(1)=82.
* z(2)=15.
* z(3)=8.
* z(4)=53.
* z(5)=0.
  Average=29.25
* z(6)=41.
  Average=22.25
* z(7)=22.
  Average=27.25
* z(8)=3.
  Average=22.75
* z(9)=18.
  Average=18.75
* z(10)=75.
  Average=25.25
* z(11)=64.
  Average=34.75
* z(12)=39.
  Average=44.5
* z(13)=46.
  Average=52.5
```

## ■ KEY OPERATION SEQUENCE

1) **DEF A** [Program start]

Number of items = \_

2) 4 **ENTER** [Number of items input]

Number of items = 4  
X(1) = ?

## PROGRAM LIST

```

10:"A": CLEAR : CLS :
    WAIT 0: LPRINT "":
    USING
20:N=1:E=0:C=1
30:INPUT "Number of items = ";A
40:DIM X(A-1)
50:IF A<> INT (A/2)*2
    THEN 120
60:FOR I=0 TO A-1:
    GOSUB 200: NEXT I
70:FOR I=0 TO A-1:
    GOSUB 220: CURSOR 5+
    LEN STR$ N,C: INPUT
    D: GOTO 90
80:GOTO 250
90:F=X(I): GOSUB 210
100:LPRINT "      Average="
    "(E+.5*(D+F))/A"
110:E=E-F: NEXT I: GOTO
    70
120:FOR I=0 TO A-2:
    GOSUB 200: NEXT I
130:B=A-1: GOSUB 220:
    CURSOR 5+ LEN STR$ N
    ,C: INPUT D
140:GOSUB 230:X(B)=D
150:GOSUB 240
160:FOR I=0 TO B: GOSUB
    220: CURSOR 5+ LEN
    STR$ N,C: INPUT D:
    GOTO 180
170:GOTO 250
180:E=E-X(I): GOSUB 210
190:GOSUB 240: NEXT I:
    GOTO 160
200:GOSUB 220: CURSOR 5+
    LEN STR$ N,C: INPUT
    D
210:GOSUB 230:X(I)=D:
    RETURN
220:PRINT : PRINT "x(" +
    STR$ N + ")=": RETURN
230:LPRINT "* x(" + STR$
    N + ")=": D:E=E+D:N=N+1
    :C=C+(C<>3): RETURN
240:LPRINT "      Average
    =" : E/A: RETURN
250:LPRINT : LPRINT :
    CLS : END
    
```

575 bytes.

3) 82 **ENTER** [X(1) input]

```

Number of items = 4
X (1) = 82
X (2) = ?
    
```

input as above

4) 46 **ENTER** [X(13) input]

```

X (11) = 64
X (12) = 39
X (13) = 46
X (14) = ?
    
```

5) **ENTER** [Program completed]

>

## MEMORY CONTENTS

A	number of items
B	✓
C	✓
D	data
E	data total
F	✓
I	loop counter
N	✓
X(A-1)	data

# Program Title: TRANSFER OF PROGRAM FILE

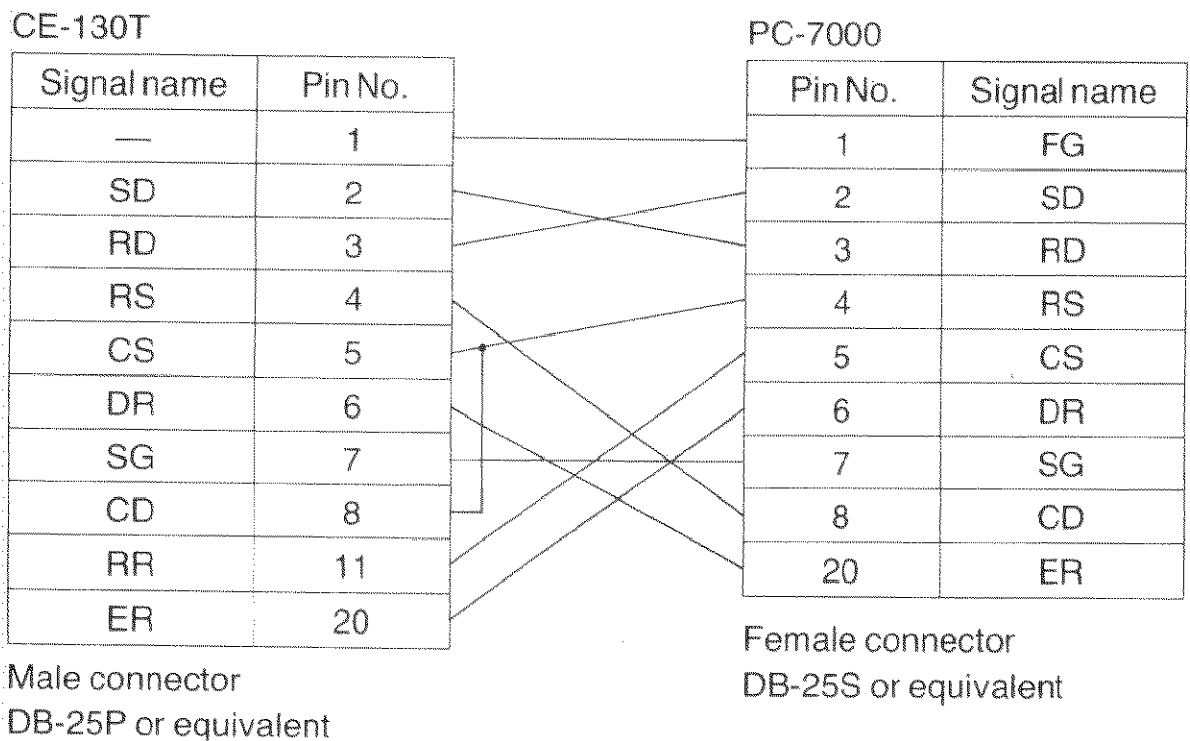
Required Peripheral  
Equipment  
CE-130T

In the past, only cassette (micro cassette) tape was available as an external memory for pocket computers. But now a floppy disk in a personal computer can be used as an external memory for the PC-1360 pocket computer through the medium of the CE-130T level converter.

Here, transfer of program files to and from the PC-1360 pocket computer is explained using the PC-7000 as an example of the personal computers.

## ■ PREPARATION

- In addition to the PC-7000 personal computer, you must prepare a cable for connecting the CE-130T to the RS-232C interface of the PC-7000. The following table shows the cable requirements and pin connections necessary for interfacing.



..... denotes that the connection between these two pins is not mandatory.

- **UPLOADING PROGRAM FILE (PC-1360 → PC-7000)** First, the method of transferring a program file from the PC-1360 to the PC-7000 is introduced here. The PC-7000 will in turn store the data received from the PC-1360 in a 5-1/4-inch floppy disk. The contents of the program file being received can be confirmed on the screen of the PC-7000.

## ■ OPERATING PROCEDURE

PC-1360

Program input

Input a program to be transferred to the PC-7000.

PC-7000


Program input

Input Program 1-1. (See below.)

Preparation for Program File Transfer

Set PRO or RUN mode.  
CLOSE **ENTER**  
OPEN "1200, N, 8, 1, A, C, &1A"  
**ENTER**  
→ SAVE command is now executable.

Program File Receiving

RUN   
→ The computer executes Program 1-1.  
→ The computer waits for receipt of program file.

Program File Transfer

SAVE **ENTER**.  
→ The computer transfers program file.

The computer receives program file.  
→ The contents of the received file are displayed one line at a time on the screen.  
Input file name of received program.  
→ Storage of program file in floppy disk is now completed.

## ■ PROGRAM LIST 1-1

```

10 CLEAR: DIM A$(1000)
20 CLOSE: OPEN "COM1:1200, N, 8, 1" AS #1
30 Z$=INPUT$(1,1): IF Z$=CHR$(&H1A) THEN 100
40 LINE INPUT #1, A$
50 A$=Z$+A$
60 PRINT A$
70 A$(I)=A$
80 I=I+1
90 GOTO 30
100 INPUT "FILE NAME=" ; FI$
110 OPEN "O", #2, FI$
120 FOR J=0 TO I-1
130 PRINT #2, A$(J)
140 NEXT J
150 CLOSE
160 END

```

No programming for downloading is required on the part of the PC-1360.



## ■ MEMORY CONTENTS

I	Counter
J	Loop counter
A\$	For input of one program line
Z\$	✓
FI\$	File name
A\$(1000)	✓

## ■ LINE MAP

Line Number	Contents
10~ 20	initialization
30~ 90	receive program through RS-232C
100	file name input
110~160	received program is written into FD

### ● DOWNLOADING PROGRAM FILE (PC-7000 → PC-1360)

The method of transferring the program file in the floppy disk of the PC-7000 to the PC-1360 is covered here. The contents of the program file being transferred can be confirmed on the screen of the PC-7000.

## ■ OPERATING PROCEDURE

PC-7000

Program input

Input Program 2-1. (See below.)

Program File Transfer

RUN 

- The computer executes Program 2-1. Input file name of program to be transferred.
- The contents of program file being transferred are displayed one line at a time.
- The computer transfers program file.

PC-1360

Preparation for Receiving Program File

Set PRO or RUN mode.

CLOSE **ENTER**

OPEN "1200, N, 8, 1, A, C, &1A"

**ENTER**

→ LOAD command is now executable.

Program File Receiving

LOAD **ENTER**

- The computer waits for receipt of program file.
- The computer receives program file.
- End of transfer

## ■ PROGRAM LIST 2-1

```

10 CLOSE:OPEN "COM1:1200,N,8,1" AS #1
20 INPUT "FILE NAME=";FI$
30 OPEN "I",#2,FI$
40 IF EOF(2)=-1 THEN PRINT #1,CHR$(&H1A);:CLOSE:END
50 LINE INPUT #2,A$
60 PRINT A$
70 PRINT #1,A$+CHR$(&HD);
80 GOTO 40

```

No programming for downloading is required on the part of the PC-1360.

## ■ MEMORY CONTENTS

A\$	For output of one program line
FI\$	File name

## ■ LINE MAP

Line Number	Contents
10	initialization
20	filename input
30~80	output to RS-232C

# Program Title: ROUND GRAPHIC

Color Dot Printer  
CE-140P Required

## OVERVIEW

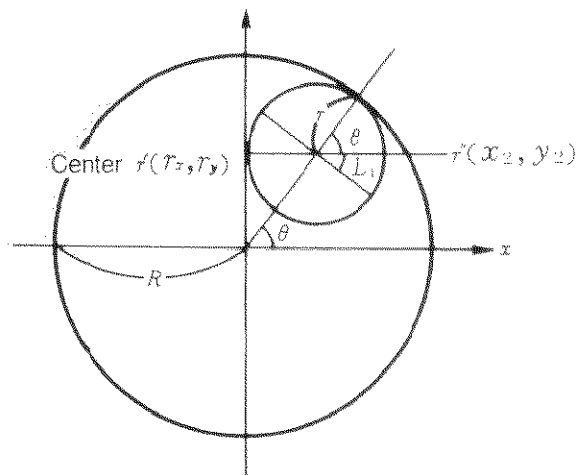
You can draw various interesting curves in color by this short program.

## INSTRUCTIONS

1. Program starts at **DEF A**.
2. Specify the color for graphic.

0	1	2	3	4	5	6
BLACK	PURPLE	RED	MAGENTA	GREEN	CYAN	YELLOW

3. Input L, which is the distance between the smaller circle radian and the moving point from the center of the smaller circle.
4. After completing the input described above, a graphic will be printed out.



Value  $r''$  to variable  $\theta$   
shall be gotten as follows.

$$x_2 = x_1 \cos \theta_1 - y_1 \sin \theta_1 + r_x$$

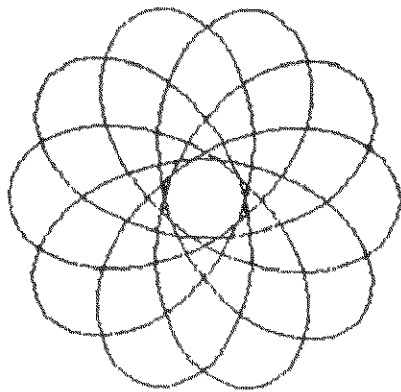
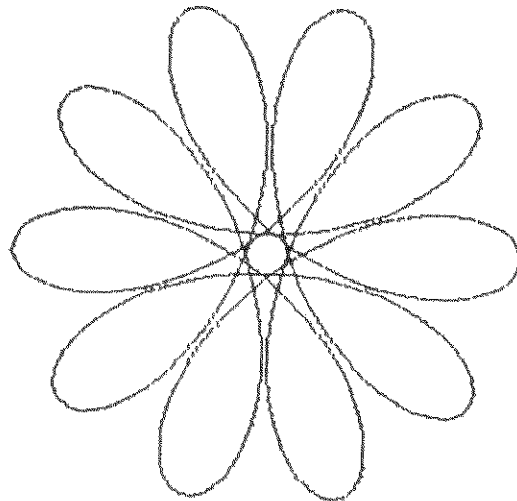
$$y_2 = x_1 \sin \theta_1 + y_1 \cos \theta_1 + r_y$$

$$\text{Note } \begin{cases} \theta_1 = \theta - 90^\circ \\ x_1 = L \cos(90^\circ - \frac{R}{r} \theta) \\ y_1 = L \sin(90^\circ - \frac{R}{r} \theta) \\ r_x = (R - r) \cos \theta \\ r_y = (R - r) \sin \theta \end{cases}$$

**■ EXAMPLE**

Making graphics using the data in the chart below.

	Color	r	L
1	5	140	40
2	3	140	70

**■ PRINTOUT...**(Printed out in color)**EXAMPLE 1****EXAMPLE 2**

## PROGRAM LIST

```

10:"A": WAIT 0: CLEAR : LPRINT :
  CONSOLE 39
20:LPRINT : LTEXT : LPRINT : GRAPH :
  GLCURSOR (240,-240): SORGN
30:CLS : USING : PRINT "  ** ROUND GRA
  PHIC **"
40:INPUT "COLOR(0-6)=";M: COLOR M,7
50:INPUT "r=";C,"L=";L
60:R=200:H=10: CLS
70:CURSOR 5,1: PRINT "** PRINTING **"
80:P=10000
90:D=0: 60SUB "CAL":U=0:V=0:E=0:F=0:
  GLCURSOR (U,U)
100:FOR I=H TO P STEP H
110:D=I: 60SUB "CAL":W=0:X=0
120:LLINE -(W,X)
130:U=W:V=X: IF F=U IF E=U THEN 150
140:NEXT I
150:CLS : LTEXT : LPRINT : LF 10: END
160:"CAL":G=R/C*D:S=(R-C)* COS D:T=(R-C
  )* SIN D:K=L* COS (90-G):N=L* SIN (
  90-G)
170:A= SIN (D-90):B= COS (D-90)
180:O=K*B-A*N+S:Q=K*A+B*N+T
190:RETURN

```

453 bytes

## MEMORY CONTENTS

C	r (smaller circle radian)
D	$\theta$
E	first X coordination
F	first Y coordination
G	$\theta'$
H	$\sqrt{\quad}$
J	$\theta_1$
K	$X_1$
L	L
M	color set
N	$Y_1$
O	$X_2$
P	limit value of O
Q	$Y_2$
R	R (larger circle radian)
T	rx
T	ry

## LINE MAP

Line Number	Contents
10~ 20	initialization
30~ 50	data input
60~150	graphic output
160~190	calculation sub

## ■ KEY OPERATION

### EXAMPLE 1

1) **DEF A**

```

** ROUND GRAPHIC **
COLOR (0-6) = _

```

2) 5 **ENTER**

```

** ROUND GRAPHIC **
COLOR (0-6) = 5
r = _

```

3) 140 **ENTER**

```

** ROUND GRAPHIC **
COLOR (0-6) = 5
r = 140
L = _

```

4) 40 **ENTER**

```

** PRINTING **

```

```
>
```

### EXAMPLE 2

1) **DEF A**

```

** ROUND GRAPHIC **
COLOR (0-6) = _

```

2) 3 **ENTER**

```

** ROUND GRAPHIC **
COLOR (0-6) = 3
r = _

```

3) 140 **ENTER**

```

** ROUND GRAPHIC **
COLOR (0-6) = 3
r = 140
L = _

```

4) 70 **ENTER**

```

** PRINTING **

```

```
>
```

# Program Title: SKI JUMP

Let's attempt the longest jump!

Today there is fine weather, ideal for ski-jumping. You are enthusiastic about the set-up of the longest jump distance. Jump with perfect timing after level skiing, keeping your balance despite wind from the right and left sides.

Now, let us see how many meters you can make!

## ■ HOW TO PLAY

1. **R U N ENTER** displays the title "The Longest Distance". Press the space key and then the game starts.
2. After the jumping stand is displayed on the screen, the man starts skiing. During the course of his level skiing, press the key **5** to let him jump. If the jumper fails to jump due to a failure of timing, "Om" will be displayed on the screen and then the game returns to the initial display of the title.
3. Although the skier has jumped, keep the jumper's balance despite wind from the left and rights sides by using the following keys.

Wind from left side (→): Press key **4**

Wind from right sidce (←): Press key **6**

(The jumper's balance is indicated by the display of a vertical bar "I".)

- When the jumper lands properly on the ground, the distance covered is displayed, but when he falls halfway due to losing his balance, "Fallen" is displayed on the screen.

In such a case, no display of the flying distance will appear on the screen.

When the above procedure is over, the initial display of the game will reappear. Now, you should wait to input the game if you wish to try again.

## ■ REFERENCE

<The method of calculating the distance covered >

## 1. Timing of the jump

J (the descending speed of the jumper depending on the timing) is calculated through the X coordinate (125-110) when pressing key **5**.

$$J = (X - 110) \times 0.02$$

Move the jumper along each dot up to X-coordinates 105-80. As in every movement, the jumper decreases in accordance with the formula of "Y=Y+J" (Y is for the altitude).

In short, the timing of pressing key **5** affects the altitude at the point of the X-coordinates 80.

## 2. Balance

The altitude decreases at a regular speed. The X-coordinates are calculated by the following formula.

$$X = X - (5 - \text{ABS}(B)) \times 0.2$$

B shows the balance and covers the range of the figures from -5 to +5. (When B is above or below the figure, the jumper falls down.)

Calculation and display are repeated with the above sequence and then S (flying distance) is calculated from the X-coordinates when the landing judgement formula "Y > 31 - INT (X × 0.05)" is established.

$$S = (100 - X) \times 1.5$$

Therefore, keeping the proper balance is the knack of increasing the distance covered in flight.



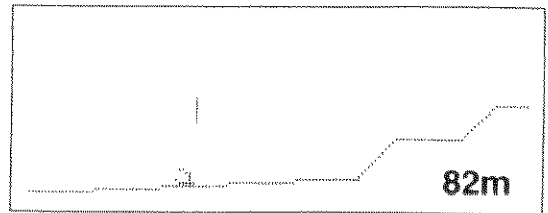
## ■ KEY OPERATION SEQUENCE

- 1) **R U N ENTER** [Program starts at]

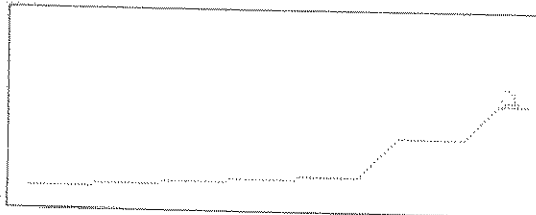
```

*** Ski Jump ***
The Longest Distance:
                                0m
Press the space key
    
```

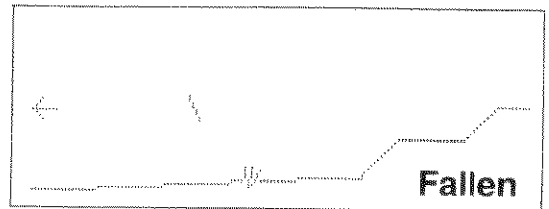
«display example of the well landing»



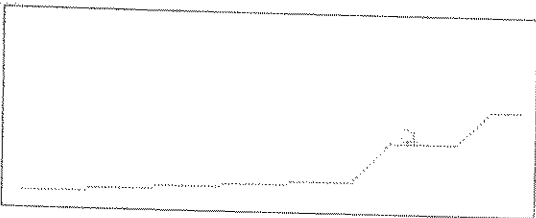
- 2)  (space key) [The game starts]



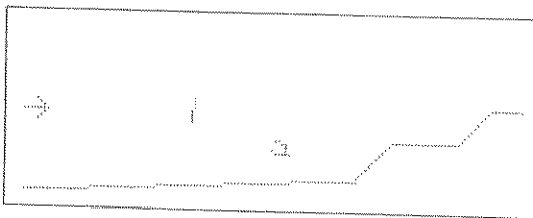
«display example in the case of falling halfway»



- 3) **5**



- 4) **4**



Continue the game by operating 4 or 6 key.

# PROGRAM LIST

```

10:WAIT 0:H=0
20:F1$="40888890F08000"
   F2$="40888890F08000
   00":F3$="1EE05E20182
   4"
30:DIM S$(5)*16:S$(1)="
   030C30C000000000":S$(
   2)="00000FF00000000
   0"
40:S$(3)="000000FF00000
   0":S$(4)="000000F00F
   000000":S$(5)="00000
   000C0300C03"
50:CLS : WAIT 0:
   CURSOR 1,0: PRINT "*"
   ** Ski Jump ***":
   BEEP 1
60:CURSOR 1,1: PRINT "T
   he Longest Distance
   :":
70:CURSOR 17,2: PRINT
   USING "####":H;"m":
80:CURSOR 1,3: PRINT "P
   ress the space key":
90:I$= INKEY$: IF I$<>
   " " THEN 90
100:CLS : LINE (149,5)-(
   140,5): LINE -(130,1
   5): LINE -(110,15):
   LINE -(99,26)
110:FOR I=1 TO 5: LINE (
   119-I*20,26+I)-(100-
   I*20,26+I): NEXT I
120:GDCURSOR (141,4):
   GPRINT F1$: GDCURSOR
   (141,4): BEEP 3:
   GPRINT "0000000000000
   0"
130:FOR I=0 TO 10:
   GDCURSOR (134-I,4+I):
   GPRINT F1$: NEXT I
140:X=125
150:I$= INKEY$: IF I$="
   5" THEN 210
160:X=X-2: GDCURSOR (X,14
   ): GPRINT F2$: IF X>
   110 THEN 150
170:GDCURSOR (X,14):
   GPRINT "0000000000000
   0"
180:FOR I=0 TO 11:
   GDCURSOR (103-I,15+I)
   : GPRINT F1$: NEXT I
190:CURSOR 19,3: WAIT 16
   0: PRINT "0 m":
200:GOTO 50
210:J=(X-110)*.02: FOR I
   =X TO 105 STEP -2:
   GDCURSOR (I,14):
   GPRINT F2$: NEXT I
220:Y=14: FOR I=105 TO 8
   0 STEP -1:Y=Y+J:
   GDCURSOR (I,Y):
   GPRINT F1$: NEXT I:B
   =0,X=90
230:R= RND 3: ON R GOTO
   240,250,260
240:GDCURSOR (0,10):
   GPRINT "101010109254
   3910":B=B+1: GOTO 27
   0
250:GDCURSOR (0,10):
   GPRINT "0000000000000
   0000": GOTO 270
260:GDCURSOR (0,10):
   GPRINT "103854921010
   1010":B=B-1
270:I$= INKEY$
290:IF I$="4" LET B=B-1
290:IF I$="6" LET B=B+1
300:IF B<-5 OR B>5 THEN
   390
310:GDCURSOR (50,10):
   GPRINT S$(B*.5+3)
320:X=X-(5- ABS (B))* .2:
   Y=Y+.26: IF Y>31-
   INT (X*.05) THEN 360
330:GDCURSOR (X,Y):
   GPRINT F2$
340:IF Y>27 LET C=
   POINT (X,Y+1): IF C
   GOSUB 420
350:GOTO 230
360:S=(100-X)*1.5:
   CURSOR 18,3: PRINT
   USING "####":S:
   WAIT 320: PRINT "m"
370:IF S>H LET H=S
380:GOTO 50
390:FOR I=Y TO 31:
   GDCURSOR (X,I):
   GPRINT F3$: NEXT I
400:CURSOR 18,3: WAIT 16
   0: PRINT "Fallen":
410:GOTO 50
420:FOR I=2 TO 5: LINE (
   119-I*20,26+I)-(100-
   I*20,26+I): NEXT I:
   RETURN

```

1394 bytes

## ■ MEMORY CONTENTS

B	balance
C	√
H	The longest flying distance
I, I\$	loop counter, √
J	descending speed
R	direction of wind
S	flying distance
X	level distance
Y	altitude
F1\$	for display (figure for man)
F2\$	for display (figure for man)
F3\$	for display (figure for man)
S\$(5)	for display (balance)

## ■ LINE MAP

Line Number	Contents
10~ 40	initialization
50~140	initialized display
150~200	jump timing processing
210~220	jump process
230~260	direction of wind process
270~350	balance process
360~380	landing process
390~410	fall process
420	course display



# INDEX

## —A—

ALL RESET 11–12, 279–280  
 Arithmetic expressions 54  
 Arrays 47–54, 147–148, 151  
 Automatic program execution 108–110

## —B—

BASIC 43–63  
     commands 62–63, 103–278  
     concepts 43–60  
     functions 37–39, 59, 99–100, 103–278  
     statements 61–63, 99–278  
     verbs 62, 99, 102–278  
 Batteries 12–15, 87, 279

## —C—

Calculator usage 19–41  
     chained calculations 32–33  
     errors 22–27, 40–41  
     serial calculations 27–28  
     priority 29–30, 39  
 Care of the computer 2, 285  
 Cassette tape  
     recorder 83–91  
     commands 115–117, 123–126, 136–137  
     saving/loading programs 87–91  
 Cells 12–15, 87, 279  
 Character  
     code table 289–291  
     manipulation 100, 181–182, 215, 251  
     strings 43, 54  
 Clearing  
     arrays 122, 151, 217, 254  
     variables 122, 217, 254

## INDEX

- Commands 62–63, 99, 102–278
- Compatibility with other computers 102, 308–309
- Constants 43
- Contrast control 12, 279
- Copying RAM cards 101–102
- Cursor control 7, 9, 22–26, 155–156

– D –

- Debugging 273–274, 280–284
- Declaring variables 30–33, 44–54
- Dimensioning arrays 48–54, 151–152
- Display 6–7, 9–10
- Display contrast control 12, 279

– E –

- Editing 64–68
  - program line deleting 149–150
  - program line renumbering 250–251
  - recalling entries 22–27

### Errors

- messages 286–288
- on input 22–27, 64–68, 288

- Executing a program 63, 64, 108–110

### Expressions

- arithmetic 54
- logical 56–58
- relational 55–56
- string 54

– F –

- Fixed variables 46
- Floating point representation 33–34, 294–295
- Formatting output 273–274, 292–295
- Functions 37–39, 59–60, 99–100, 103–278

## -- G --

Graphics mode 100, 173–174, 207

Graphics screen 72–77, 100

## -- H --

Hexadecimal numbering 44

## -- I --

Initializing 11, 15–17, 97–98

Input from keyboard

errors 22–27

hints 20–21

maximum length 36

Input from program/tape 89–90, 119–121, 127–130, 216

I/O ports 3

printer 83–85, 91–94

RS-232C serial 83–85, 91–95, 304–305

## -- J --

## -- K --

Keys

description 4–8, 299–303

input hints 20–21

values 172, 289–290

Keyboard

layout 3–8, 290

template 82

## -- L --

Labels 79

Loading programs 89–90, 119–121, 127–130, 216

Logical expressions 56–58

## - M -

- Machine language commands 105, 118, 130, 141, 232, 235
- Maintenance 285
- Memory 306-307
  - RAM sizes 97
  - RAM allocation 97-98, 260-263
  - Merging programs 216-218

## - N -

- Numeric functions 59-103

## - O -

## Output

- formatting 275-276, 292-295
- graphic 104, 159-162, 167-171
- printing 91-94
- serial I/O port 91-95, 304-305

## - P -

- Passwords 229
- Peripherals 83-95
- Plotter/printer graphics commands 83, 104, 123-125, 133-134, 193-195, 227-228, 255
- Ports 3
  - printer 83-85, 91-94
  - RS-232C serial 83-85, 91-95, 304-305
- Power on/off 17-20
- Printer
  - control 85-87, 91-94
  - types 83-84
- Printing 85-87, 91-94
- Priority 29-30, 39, 294-296
- Program listing 69-86, 192, 196-197
- Program memory 306-307



## Programming

- concepts 43–60
  - examples 61–78, 311–337
  - line numbers 61–62, 250–251
  - on other computers 102, 308–309
- PROGRAM mode 4, 6, 9, 63
- Pseudovariabes 99, 296

– Q –

– R –

- RAM cards 97–102
- copying 101–102
  - mounting/removing 13–15, 97–99
  - sizes 97
  - usage and initialization 16–17, 97–100, 260–263
- Recalling entries 22–26
- Relational expressions 55–56
- RESERVE mode 80–82
- RS-232C I/O port 83–85, 91–95
- commands 131, 224–226
  - specification 304–305
- RUN mode 4, 6, 9, 59, 63
- Running a program 63–72

– S –

- Saving programs 87–89, 140–141
- Scientific functions 37–39, 59
- Scientific notation 33–34, 293–294
- Screen 6–7, 9–10
- contrast control 12, 279
  - coordinates 76–77
  - graphics commands 72–77, 104
- Serial I/O
- commands 131, 224–226
  - usage 83–85, 91–95
  - signals–304–305
- Shortcuts 79–82

## Specifications

PC-1360 306-307

cassette recorder 85

Statements 61-63, 99-278

Storing programs in RAM 71

## String

comparisons 56

constants 43

expressions 55

functions 103

manipulation 181-182, 215, 251

- T -

Template 82

Text mode 173-174, 211, 273-274

Trace mode 282-284

Troubleshooting 279-284

- U -

- V -

## Variables

clearing 126, 221, 254

names 44, 47, 51-52, 67

types 44-54

Verbs 62, 99, 102-278

Verifying saved programs 88-89

- W -

- X -

- Y -

- Z -



# SHARP CORPORATION

OSAKA, JAPAN

©1986 SHARP CORPORATION

9L 0.5-I(TINSE1074ECZZ)00