HEWLETT PACKARD

HP-86/87 Operating and BASIC Programming Manual

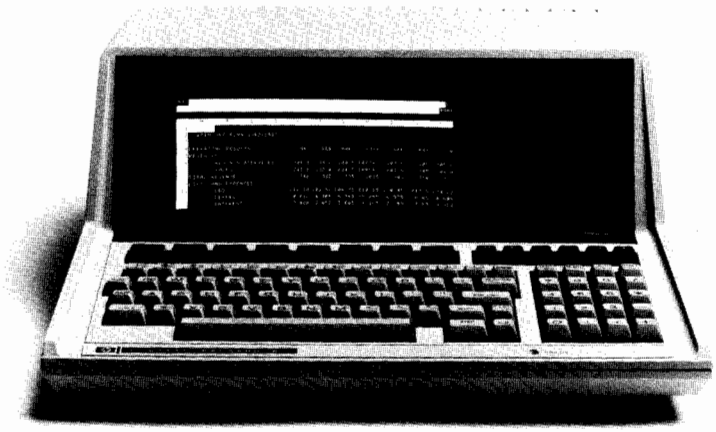
December 1982

Reorder Number
00087-90017

Printed in U.S.A.

Table of Contents

The HP-86 Personal Computer and HP 82913A Monitor



The HP-87 Personal Computer

Preface

Your Series 80 Personal Computer is designed to be the central device for a powerful and flexible computing system. Starting with the computer as your primary building block, you can add a wide variety of peripheral devices, modules, and enhancement ROMs, thereby customizing your system to meet your computing needs:

- By including a compatible disc drive system, you can use the computer to rapidly and conveniently access large amounts of stored information.
- Addition of a full-width printer allows you to produce a permanent record of computer output.
- The computer's powerful graphics statements enable you to draw figures and plot data on the display or, with the addition of the HP-87 Plotter ROM, on high-resolution graphics plotters.
- A variety of optional interface modules, as well as the computer's integrated interfacing capabilities, allow the computer to participate in a broad range of input/output operations.
- By adding one or more plug-in ROMs (Read-Only-Memory units), you can increase the programming power of the computer.
- The amount of user-available memory built into the computer (approximately 60 kilobytes, abbreviated "K bytes," for the HP-86 and 124K bytes for the HP-87XM), is a base figure. Addition of one to four 32K-, 64K-, and 128K-byte memory modules expands user-available memory by up to 512K bytes.

As a member of the Series 80 group of personal computers, your computer has been designed to provide maximum compatibility with other Series 80 products. Most programs written for other Series 80 computers can be run with little or no editing. With its powerful programming features, the computer allows you to compose and run programs in a form of the BASIC (Beginner's All-Purpose Symbolic Instruction Code) language that meets and exceeds the latest ANSI (American National Standard Institute) Standard for Minimal BASIC.

With the computer's powerful problem-solving features at your disposal, you have the ability to:

- Perform arithmetic calculations involving constants, variables, and functions in a straightforward, algebraic manner.
- Enter, edit, list, and run programs. The 80-column CRT display, combined with the ability to indent program lines, enables you to format and document your programs for easy readability.
- Create variables with mnemonic names up to 31 characters long. You can also assign names to program statements.
- Direct output to a peripheral printer. The computer lets you choose the line length to match the width and character set(s) of your printer.
- Utilize a high-speed mass storage system for storing, retrieving, and manipulating programs, data, and graphics.

- Generate sophisticated, high-resolution graphics on the computer display. Numerous scaling, plotting, and labeling statements provide the programming options necessary for generating graphical representations of data.
- Manipulate data using one- and two-dimensional numeric and string arrays.
- Define your own functions within a program. In addition, 14 user-defined keys can operate both as typing aids and as tools for controlling running programs.

Owner's Documentation

The introductory manual is intended to guide you through the initial stages of setting up your system and familiarizing yourself with the computer's features. If this operating and programming manual is the one you've opened first, we strongly suggest you put it down for a while and start reading section 1 of the introductory manual. It is particularly important that you carefully follow the instructions for installing plug-in modules and connecting peripheral devices provided in sections 2 and 3 of that manual. Then, section 4 will take you on a whirlwind "tour" of the features with which you must be familiar in order to perform calculations, use pre-packaged software, and enter and run programs.

Using This Manual

This manual is designed to provide you with the information you need to operate your computer and to learn the computer's BASIC language. The manual is divided into four major parts.

Part I introduces you to the keyboard and display and to the mathematical, logical, and the character-manipulating functions. Once you've read the introductory manual, you'll probably want to spend some time going through the four sections in Part I, taking the time to key in the examples. Even if you are an experienced programmer, you will benefit from the explanations provided in sections 1 through 4.

Part II discusses BASIC language programming. The nine sections in part II have been organized to present topics in the order of increasing difficulty. If you've had some programming experience in any programming language, part II will very likely be sufficient to teach you how to write, enter, and run BASIC language programs.

If you are an experienced programmer, you'll probably use part II (and parts III and IV, as well) more as a reference than as a teaching tool. The pocket guide, with its alphabetically-arranged listings of statements, commands, and functions, will prove to be an efficient reference document.

If you've had little or no experience in designing and writing computer programs, you may find that you need additional teaching support for learning BASIC. If this is the case, refer to the discussion of Customer Training Courses in appendix B for information on the educational resources and technical support available from Hewlett-Packard to owners of Series 80 computers.

Part III of this manual covers the computer's graphics capabilities. If you want to learn to draw illustrations and plot data on the display, or if your system includes an HP-87 Plotter ROM and a peripheral plotter, then you'll want to familiarize yourself with the computer's scaling, plotting, and labeling statements.

Mass storage techniques are discussed in Part IV. The four sections in part IV cover naming and accessing disc-based files, storing and retrieving programs, manipulating data files, and a number of additional mass storage operations.

Customer Comment Card

We at Hewlett-Packard have an ongoing commitment to providing quality documentation for our products. Your feedback about our manuals is essential for providing us with information we need to write future manuals that best meet our customers' needs.

Please complete the customer comment card in the back of this manual. The card is pre-addressed to us, and postage is already paid if mailed in the United States. In advance, thank you.

Power-On

Section 4 of the introductory manual provides complete instructions for switching power on for the first time. The discussion of the power-on procedure in appendix C of this manual describes the operations performed by the computer at power-on.

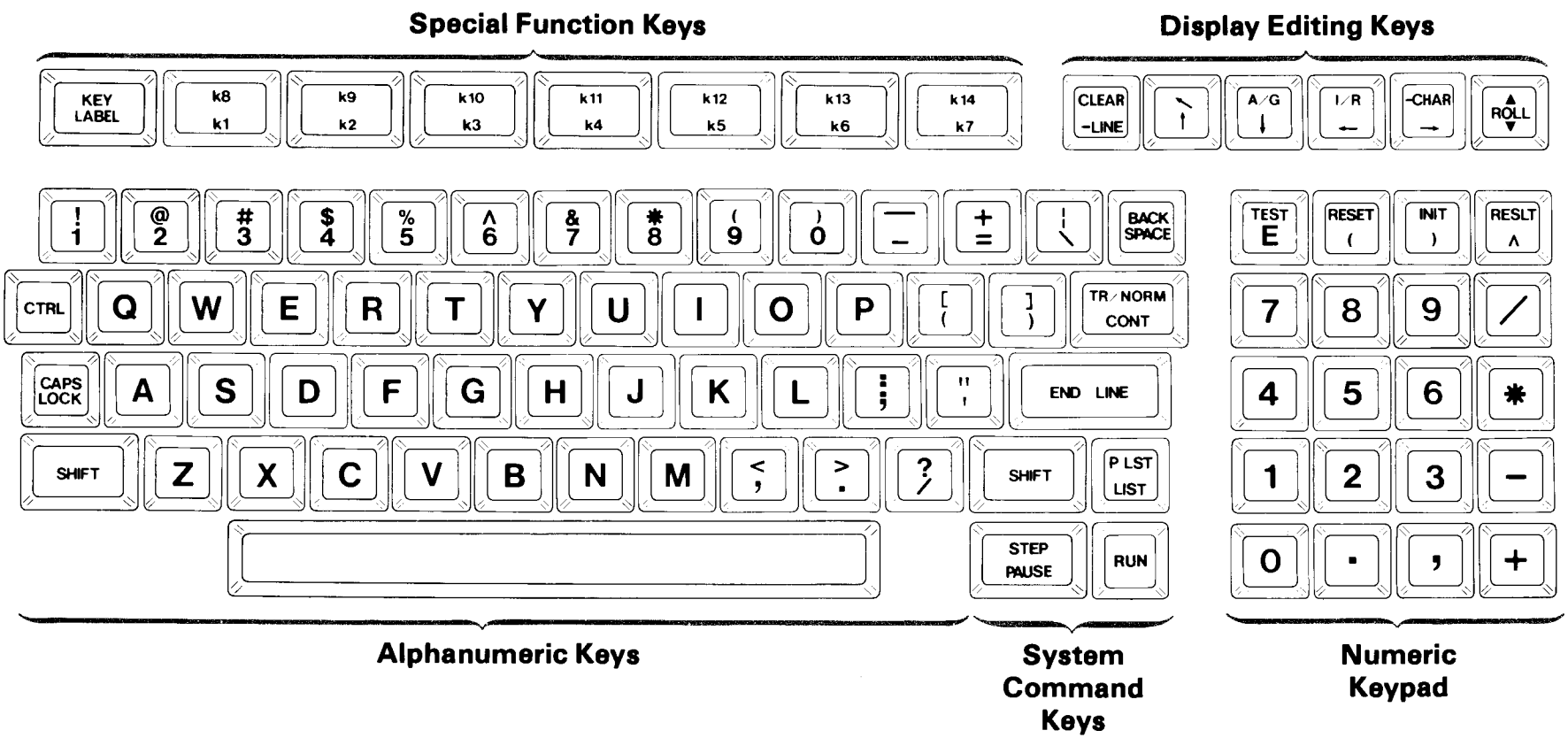
CAUTION

The inspection procedure and initial set-up instructions are presented in sections 2 and 3 of the introductory manual. Please read those sections:

- If you have not yet inspected the computer.
- If there is any doubt regarding the compatibility of the system power requirements to the available power in your area.
- If you must install optional plug-in modules or connect any peripheral devices.

Do not attempt to set up the computer without first becoming thoroughly familiar with the material in sections 2 and 3 of the introductory manual; they contain information essential to avoid damaging your personal computer and peripheral devices when your system is initially set up.

Part I
Evaluating Expressions


Special Function Keys
Display Editing Keys
KEY LABEL
k8 k1
k9 k2
k10 k3
k11 k4
k12 k5
k13 k6
k14 k7
CLEAR -LINE
A/G
I/R
-CHAR
ROLL
BACK SPACE
TEST E
RESET
INIT
RESLT
CTRL
TR/NORM CONT
CAPS LOCK
END LINE
SHIFT
P LST LIST
STEP PAUSE
RUN
Alphanumeric Keys
System Command Keys
Numeric Keypad

Section 1

Keyboard, Display, and Peripheral Printer

The keyboard, CRT display, and optional peripheral printer are your primary means of interacting with the computer. If you keyed in the example calculations and programs in section 4 of the introductory manual, then you're already somewhat familiar with entering information at the keyboard and receiving output on the display and printer. This section will cover the keyboard, display, and control of a peripheral printer in greater detail.

The Keyboard

The computer's keyboard is divided into five areas:

- Alphanumeric (typewriter) keyboard.
- System command keys.
- Numeric keypad.
- Special function keys.
- Display editing keys.

Each of the keys in these five areas perform one or more of the following functions:

- Produces a single character on the display (for example, (A), (5), (*)). The character produced is altered by pressing the key along with (SHIFT) or (CTRL).
- Causes the computer to immediately perform a system command (for example, (RUN), (INIT), or (RESET)).
- Provides cursor or display control (for example, (←), (ROLL)).
- Operates as a typing aid, producing an entire BASIC keyword or other user-defined sequence of characters on the display (the special function keys in *calculator mode;* that is, while you are keying in a statement).
- Alters the order in which parts of a program are executed (the special function keys in *program mode;* that is, while a program is running).

The Alphanumeric Keys

The alphanumeric keys operate much like those on a standard typewriter keyboard. The major exception is that the unshifted letters are uppercase rather than lowercase, since BASIC language keywords are ordinarily entered as uppercase letters. To produce a lowercase letter on the display, use the (SHIFT) key.

When two characters or functions are shown on a key, the upper one is produced by using the (SHIFT) key. For instance, the !, @, and # characters are produced by pressing (SHIFT) and the appropriate number key. Pressing the numeric keypad (E) displays the letter E, while pressing the key along with the (SHIFT) key commands the computer to perform its system (TEST).

Note: This manual does not show the (SHIFT) key when describing keystroke sequences; instead, the shifted character or function is shown (for example, (-CHAR), (INIT)).

All the alphanumeric keys repeat automatically when held down.

The (CAPS LOCK) key affects only the 26 letter keys, but otherwise performs like the equivalent key on a typewriter.

The (CTRL) (control) key is used in conjunction with certain alphanumeric keys to display alternate characters, including *control* characters. The control characters (those with decimal codes 0 through 31 in the table of Character and Key Codes on page 323) are used to control operation of certain peripheral devices.

BASIC and Typewriter Modes

The keyboard is normally in BASIC mode, producing unshifted uppercase and shifted lowercase letters. You can operate the keyboard in typewriter mode, producing unshifted lowercase and shifted uppercase letters, by executing the FLIP statement:

```
FLIP
```

To execute the statement, type FLIP and then press (END LINE). You can use uppercase or lowercase letters to spell "FLIP".

Executing FLIP a second time returns the keyboard to BASIC mode.

System Command Keys

The computer keyboard includes a number of immediate-execute keys. When one of these keys is pressed, the computer immediately performs the indicated operation. Each of these operations are discussed in greater detail elsewhere in this manual. The page numbers in parentheses reference discussions of each key.

Key	Operation
(TR/NORM)	Toggle for initiating or cancelling a TRACE ALL operation. **(Page 159)**
(CONT)	Resumes execution of a paused program. **(Page 92)**
(END LINE)	Causes the computer to enter a statement into program memory or to perform a calculator-mode operation. **(Page 23)**
(LIST)	Lists 15 (PAGESIZE 16) or 23 (PAGESIZE 24) lines of the program currently in memory on the computer display. **(Page 88)**
(P LST)	Produces a listing of the program currently in memory on the system printer. **(Page 88)**
(RUN)	Starts execution of the program currently in memory. **(Page 90)**
(STEP)	Executes a single line of a program. **(Page 160)**
(PAUSE)	Halts execution of a running program. Execution can be resumed from where it left off using (CONT) or (STEP). **(Page 91)**
(TEST)	Causes the computer to perform a self-test. **(Page 318)**
(RESET)	Returns the computer to its power-on state without erasing the program currently in memory. **(Page 28)**
(INIT)	Initializes the program currently in memory. **(Page 89)**

The Numeric Keypad

The numeric keypad includes the digits 0 through 9, the radix (decimal marker) symbol ., and the five arithmetic operator symbols (/, *, −, +, ^). You can use the numeric keypad keys and the corresponding number keys on the typewriter keyboard interchangeably.

The shifted number keys produce the same characters as the corresponding shifted typewriter number keys.

The [E] key is used to enter numbers in exponential notation. You can use the numeric keypad and alphanumeric [E] keys interchangeably. The left and right parentheses are included on the numeric keypad because of their frequent use in arithmetic expressions. The comma is used to separate expressions entered during program execution.

The [RESLT] key displays the result of the last calculator-mode computation at the current cursor position.

The Special Function (User-Defined) Keys

The seven keys located above the typewriter keyboard are called *user-defined* or *special function* keys. You can specify up to 14 key assignments, one each for [k1] through [k7] (unshifted) and [k8] through [k14] (shifted).

The operation performed when you press one of these assigned keys depends on whether or not a program is running.

- When a program is running (program mode), pressing [k1] through [k14] causes the program to immediately branch to a particular statement. The statement to which the program branches must have been previously specified by an ON KEY# statement in the program. ON KEY# branching is discussed in section 12.
- When there is no program running (calculator mode), the user-defined keys can be used as typing aids. When the key is pressed, a BASIC keyword or other sequence of characters appears on the display at the current cursor position.

When you turn the power on, keys [k1] through [k14] are automatically assigned as typing aids for certain BASIC language keywords.

[k1]	AUTO	[k8]	TRACE VAR
[k2]	DISP	[k9]	CRT IS
[k3]	PRINT	[k10]	PRINTER IS
[k4]	MASS STORAGE IS	[k11]	INITIALIZE
[k5]	DELETE	[k12]	SCRATCH
[k6]	LOAD	[k13]	LOADBIN
[k7]	STORE	[k14]	STOREBIN

Executing the ON KEY# statement in calculator mode allows you to reassign the keys as typing aids for other sequences of characters. Reassigning the typing aids is discussed in section 12.

The [KEY LABEL] key is used to display the labels assigned to the user-defined keys. Pressing unshifted [KEY LABEL] displays the labels assigned by the program most recently run (program-mode key labels). Pressing [SHIFT] [KEY LABEL] displays the calculator-mode typing aid assignments.

When you press (SHIFT) (KEY LABEL), the following key labels appear at the bottom of the display inside inverse-video (black characters on white background) boxes. Each label indicates the keyword or keyword sequence for which the key acts as a typing aid. The positions of the labels on the screen correspond to the location of the unshifted (bottom row) and shifted (top row) user-defined keys.

TRACE VAR	CRT IS	PRINTER IS	INITIALIZE	SCRATCH	LOADBIN	STOREBIN
AUTO	DISP	PRINT	MASS STORA	DELETE	LOAD	STORE

When you press shifted or unshifted (KEY LABEL), the cursor automatically moves to home position (upper left corner on the display).

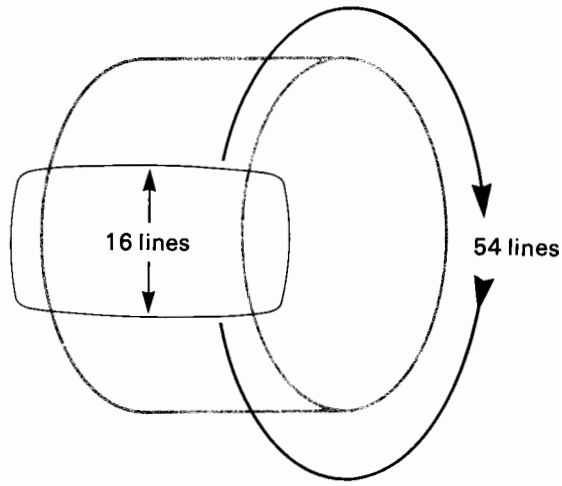
If no program-mode key labels have been assigned, pressing (KEY LABEL) displays empty inverse-video key label boxes.

The Display

The computer display is your primary means for viewing keyboard entries, obtaining program listings, editing programs, and receiving error messages.

When you turn the power on, the CRT (cathode ray tube) screen is capable of displaying 16, 80-character lines at any one time. Actually, you have access to 54 lines (approximately 3½ full screens) of information stored by the computer in CRT memory.

To understand how CRT memory works, picture it as 54 lines of information looped back on itself such that line 1 follows line 54.



The computer CRT screen is actually a viewing "window" into those 54 lines of CRT memory. If you start entering data at the top of the screen and proceed to enter 16 lines, the top line will roll out of the window to make room for line 17. After you've entered 54 lines, line 55 will overwrite line 1.

The (ROLL) key is used to rotate the window through CRT memory, thereby recalling previously displayed information. Unshifted (ROLL) moves information downward, moving the most recently lost lines into the window. Shifted (ROLL) moves information upward.

Changing the PAGESIZE

Normally, the CRT is capable of displaying 16 lines of information at a time. There may be occasions, however, when you'd like to be able to view more information. The computer, therefore, provides the ability to condense the display vertically by moving the lines closer together. In its condensed format, the CRT can display 24 lines at a time.

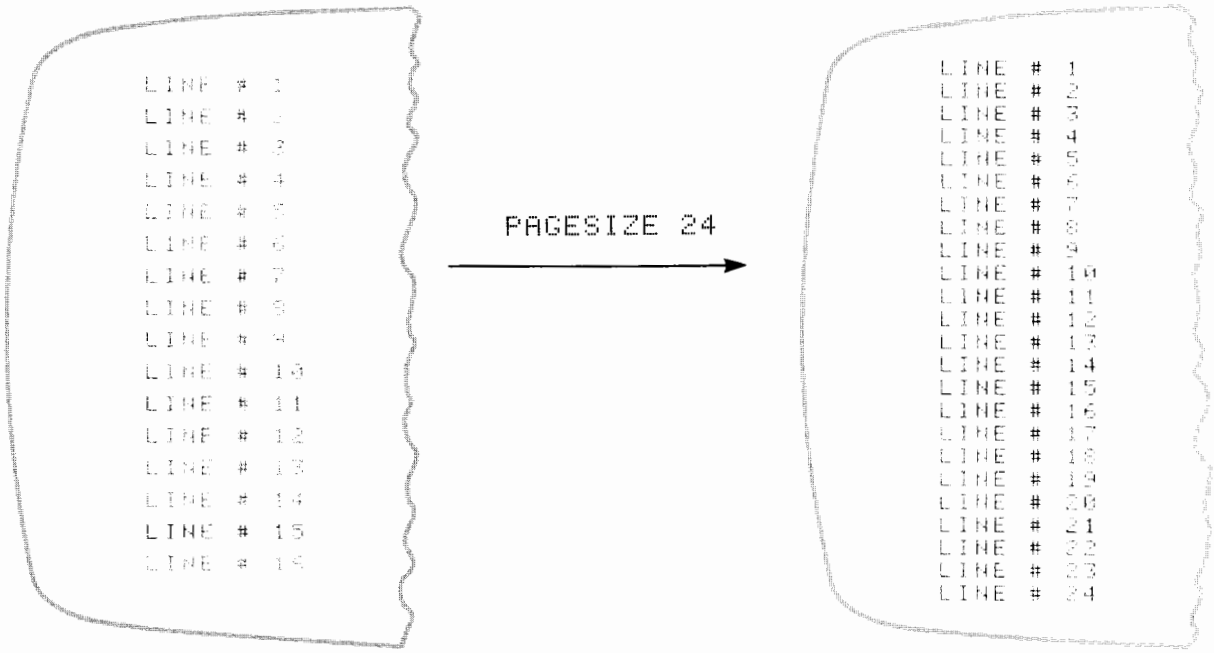
The PAGESIZE statement is used to change the vertical spacing.

PAGESIZE *number of lines*

The number of lines specified must be either 16 or 24. Executing PAGESIZE 24 provides for condensed format; executing PAGESIZE 16 returns the CRT to its regular, 16-line format.

Regardless of the format you specify, CRT memory is capable of storing 54 lines of information. Thus, when you use condensed format you have up to 2¼ screens (54 lines/24 lines per screen) available for viewing.

When the page size is 16, executing PAGESIZE 24 causes the currently displayed 16 lines to condense upwards, rolling eight additional lines into the display window.



When the page size is 24, executing PAGESIZE 16 rolls the bottom eight lines downwards out of view.

Display Editing

The computer's display editing keys are used to position the cursor in the display window, to move the display window through CRT memory, and to change the contents of the display.

Cursor Position ↑ ↓ ← → ↖

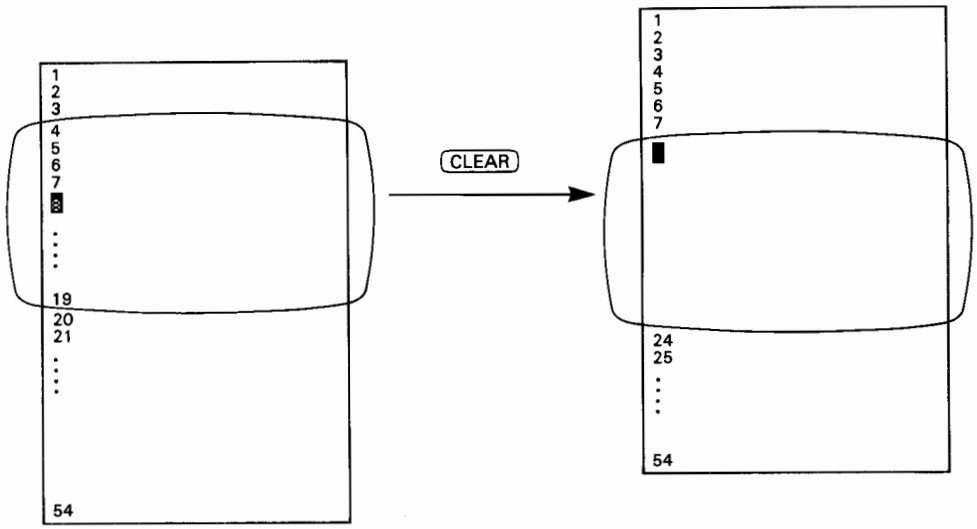
The five cursor position keys move the cursor within the display window. All except ↖ are repeating keys. Moving the cursor off the right side of the screen causes it to reappear on the left side of the next line, whereas moving the cursor off the left side of the screen causes it to reappear on the right side of the previous line. If the cursor is moved off the bottom or top of the screen, it reappears in the same column.

The ↖ key "homes" the cursor by positioning it in the upper left corner of the current screen.

Altering the Display (CLEAR) (-LINE) (I/R) (-CHAR) (BACK SPACE)

(CLEAR) Clear Display

The (CLEAR) key clears 16 lines of the CRT display (24 lines for PAGESIZE 24) starting with the line below the current cursor position. The display rolls upwards so that the window is placed at the cleared lines, and the cursor moves to home position.



To erase the contents of the current window, press (↖), then (CLEAR).

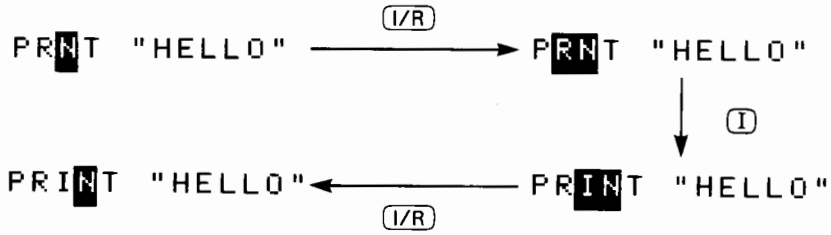
(-LINE) Clear Line

The (-LINE) key deletes the contents of one line from the current cursor position to the end of the line.

(I/R) Insert/Replace

The display is normally in *replace* mode, in that characters typed in over characters already on the display replace the original characters. The (I/R) key allows you to toggle the display between *replace* mode and *insert* mode.

When the display is in replace mode, pressing (I/R) causes the display to enter insert mode. The cursor doubles in width to include the position to the left of the original cursor. Any characters typed in are positioned between the two characters highlighted by the cursor. If the line becomes longer than 80 characters, extra characters automatically wrap around to the next line. You are limited to a final length of 159 characters.



Like the single cursor, the double cursor can be positioned anywhere on the CRT using the cursor position keys.

Pressing (I/R) again switches the display back to replace mode. The (END LINE), (-CHAR), and (BACK SPACE) keys retain their normal functions and also automatically abort insert mode.

> Note: Insert mode cannot be used to insert sequences of characters produced using the typing aids. Pressing (k1) through (k14) while the display is in insert mode causes the typing aid characters to overwrite the current contents of the display.

(-CHAR) Delete Character

The (-CHAR) key deletes the character at the current cursor position. Characters to the right of the deleted character are shifted one column to the left to fill the hole. Successive characters can be deleted by positioning the cursor at the first character to be deleted and then holding down the (-CHAR) key. Deleting characters on the first line of a two-line statement causes characters on the second line to wrap around to the first line.

(BACK SPACE) Backspace

Pressing (BACK SPACE) moves the cursor one place to the left and erases the character at the new cursor position. If you press shifted (BACK SPACE), the cursor rapidly backspaces to the beginning of the line. Holding down (SHIFT) (BACK SPACE) erases previous lines.

Viewing the Graphics Display

The (A/G) key is used to toggle between the display's *alpha* and *graph* mode. *Graph* mode and the graphics display are discussed on pages 26 and 27.

Entering Long Expressions

You can key in expressions as long as 159 characters, including spaces (two, 80-column lines minus one position for the cursor). When you've entered the 80th character, the cursor automatically wraps around to the first column of the next line.

For example, key in:

```
DISP "THIS QUOTED TEXT IS EXACTLY 80 CHARACTERS IN LENGTH AND FITS ON ONE DISPLA
Y LINE" (END LINE)
THIS QUOTED TEXT IS EXACTLY 80 CHARACTERS IN LENGTH AND FITS ON ONE DISPLAY LINE
```

The computer will allow you to type in more than 159 characters. However, when you try to enter the expression using (END LINE), you will receive an error.

An important thing to remember is that the computer's (END LINE) key is not equivalent to the return key on a typewriter. Since pressing (END LINE) causes the computer to process the information you've just entered, it shouldn't be pressed until you've entered the entire expression.

Pressing (END LINE) when the cursor is positioned on an empty line causes the cursor to move to column 1 of the next line (carriage return/line feed).

The Character Set

The computer's character set consists of 256 characters. Of these, 128 are accessed from the keyboard in four ways. The Table of Character and Key Codes on page 323 summarizes the keystrokes necessary to produce these characters.

- The characters displayed when you press the unshifted key are printed on the key. These characters include uppercase letters, digits, punctuation, and arithmetic operators.
- The (SHIFT) key is used to produce lowercase letters (you can also use (CAPS LOCK) or the FLIP statement) and the characters printed on the top of some keys. In addition, the following characters can be produced by pressing (SHIFT) in conjunction with numeric keypad keys. The superscript s indicates use of the (SHIFT) key.

Keystroke	Display Character
(+)^s	▒
(-)^s	}
(*)^s	~
(/)^s	{

- The control characters are those listed in the table of characters on page 323 that have decimal codes ranging 0 through 31. These characters are generated by holding down the (CTRL) key while you press the key specified in the table. The superscript c indicates use of the (CTRL) key.
- Certain control characters are produced by using the (CTRL) key with shifted characters. For instance, the ◄ character is produced by holding down (SHIFT) and (CTRL) while you press (2). Since shifted (2) corresponds to the character @, this keystoke sequence can be shown in two ways.

(2)^{s,c} or (@)^c

Control characters are used in advanced programming for sending instructions to peripheral devices. Techniques for sending control characters to peripheral printers are discussed on page 132.

Note: Pressing (CTRL) in conjunction with other keys changes the character displayed by subtracting 64 from the decimal code. For example, pressing (CTRL) with (SHIFT) (H) (lowercase h, decimal code 104) produces the character (, decimal code 40.

Each character is assigned a decimal number code ranging from 0 through 255. The two functions (NUM and CHR$) that provide conversions between characters and their decimal codes are discussed in section 4.

The characters corresponding to decimal codes 128 through 255 are inverse video characters and are displayed using their decimal codes. Refer to page 130 for instructions on producing inverse video characters.

Printer Control

The PRINTER IS statement is used to establish a peripheral printer as the destination device for all PRINT, PRINT USING, PLIST, and TRACE operations.

PRINTER IS *device selector* [, *line length*]

Brackets around "*line length*" indicate that it is an optional parameter. The line length parameter is a numeric expression that allows you to specify the maximum number of characters printed on a line. The default value is 80. However, the line length can range from 1 through 220. If you specify a line length larger than 220, the computer uses a line length of 220 characters. If you specify a line length of 0, the line length is set to 80.

If the printer is connected using an HP 82939A Serial Interface or an HP 82949A Printer Interface, the device selector is the select code of the interface. If the printer is connected using an HP-IB-type interface, the device selector is a three-digit combination of the select code and the device address.

Examples:

```
PRINTER IS 701,80
PRINTER IS 8,50
PRINTER IS 712
```

Most printers are capable of printing 96 standard characters—those with decimal codes ranging 32 through 127. Refer to the table of Character and Key Codes on page 323 for a list of characters and their decimal codes.

Redefining the Display and Printer

Ordinarily, the output from DISP and DISP USING statements, errors, warnings, and program LISTings are displayed on the CRT screen. PRINT, PRINT USING, PLIST and TRACE operations ordinarily route their output to the system printer.

The computer allows you to "redefine" the CRT and system printer by changing the address of the printer or CRT.

You've already used the PRINTER IS statement to define the address of the system printer. To route all information ordinarily printed to the CRT instead, you must execute another PRINTER IS statement using the device selector of the CRT. The CRT device selector is 1.

Example:

```
PRINTER IS 1
```

The CRT IS statement is used to route information ordinarily displayed to a different output device. To print all information ordinarily displayed, execute:

CRT IS *device selector* [, *line length*]

If you don't specify a line length, it is assumed to be 80 characters.

Examples:

```
CRT IS 701
CRT IS 8
```

The CRT IS statement can also be used to change the line length for CRT output from DISP and DISP USING statements.

CRT IS 1,32	DISP and DISP USING output is formatted with 32-character line length.

Print-All Mode

You may at times want to have a printed record of all information displayed on the CRT. This includes not only output from DISP, DISP USING, and LIST statements, warnings, and errors, but also all input you type into the computer.

The PRINT ALL statement sets the computer to *print-all* mode.

```
PRINT ALL
```

To return to "normal" display mode, execute the NORMAL statement.

```
NORMAL
```

The CRT Graphics Display

So far in this section, discussions of the CRT have dealt exclusively with the alpha display. It's the alpha display that you see when you first turn on the computer. You are viewing the alpha display when you enter programs from the keyboard, receive error and warning messages, list programs, and type calculator-mode expressions and statements. The characters appearing on the screen are stored in the portion of CRT memory devoted to the alpha display.

One of the sample programs in the introductory manual outputs information to the portion of CRT memory devoted to the graphics display. The graphics statements covered in Part III of this manual output data to the graphics display.

When you are viewing the alpha display, the computer is in *alpha* mode. Likewise, the computer is in *graph* mode when you are viewing the graphics display.

The computer switches from *alpha* to *graph* mode whenever:

- You press the [A/G] toggle key while in *alpha* mode.
- A GRAPH statement is executed in a program or from the keyboard.

```
GRAPH
```

The GRAPH statement is discussed in greater detail in section 14.

- A BASIC graphics plotting or labeling statement is executed in a program or from the keyboard. These statements are discussed in sections 16 through 19.

The computer switches from *graph* mode to *alpha* mode whenever:

- You press the (A/G) toggle key while the display is in *graph* mode.
- You press any typewriter key except (SHIFT) or (CTRL).
- An ALPHA statement is executed in a running program.

```
ALPHA
```

- A DISP (*display*) statement is executed in a running program.

Apportioning CRT Memory

When you turn the computer on, CRT memory is apportioned between the alpha display and the graphics display. This apportionment allots 54, 80-character lines to alpha CRT memory and provides a 400 by 240 matrix of dots for the graphics display. In *alpha* mode, you are viewing a window into alpha CRT memory; in *graph* mode, you view the entire contents of allocated graphics CRT memory.

The computer provides the following two statements for changing the allocation of CRT memory:

```
ALPHALL
```

```
GRAPHALL
```

Alpha-All Mode

When you execute an ALPHALL (*alpha all*) statement:

- The entire current contents of CRT memory (both the alpha display and the graphics display) is erased.
- The display enters *alpha-all* mode. In *alpha-all* mode, all CRT memory is devoted to the alpha display. With its increased memory, the alpha display is capable of retaining up to 204 lines of information (12¾ screens for PAGESIZE 16, 8½ screens for PAGESIZE 24).

CRT memory is erased by any operation that reapportions CRT memory.

There are several ways to exit *alpha-all* mode and return the display to power-on CRT apportionment:

- Execute the ALPHA statement. The display is placed in *alpha* mode.
- Execute a GRAPH statement. The CRT enters *graph* mode, and you will be viewing the empty graphics display. The GRAPH statement is discussed further in section 14.
- Press (RESET). The system is reset, and the CRT is placed in *alpha* mode.
- Press (A/G). The CRT enters *graph* mode.

In addition, executing the GRAPHALL statement in *alpha-all* mode switches the computer to *graph-all* mode.

Graph-All Mode

Whenever the computer executes a GRAPHALL statement:

- The entire contents of CRT memory, including any current graphics display, is erased.
- The computer enters *graph-all* mode. In *graph-all* mode, all CRT memory is devoted to the graphics display, which consists of a 544 by 240 matrix of dots.
- The entire keyboard, with the exception of the (RESET), (A/G), (PAUSE), (CONT), (STEP), and (TR/NORM) keys is deactivated. This feature provides protection against accidentally erasing a graph-all display. The keyboard is temporarily reactivated when the program is paused for input to allow input into the graphics display. You cannot execute statements from the keyboard when the display is in *graph-all* mode.

Graph-all mode is discussed in greater detail in section 14.

There are several ways to exit *graph-all* mode and restore normal CRT apportionment:

- Press the (A/G) toggle key.
- Press (RESET).
- Execute an ALPHA, or GRAPH statement within a program.

Executing ALPHALL switches the computer to *alpha-all* mode.

The relationship between *alpha, graph, alpha-all,* and *graph-all* modes is illustrated in figure 1.1 on page 29. The operations and statements used to switch between modes are also indicated.

Resetting the Computer

If the computer becomes inoperative due to a system or input/output malfunction, it may need to be reset. The computer is reset and returned to a ready state by pressing (RESET).

Resetting the computer may be necessary when it appears to be "hung up"—that is, when you can't enter and output information or move the cursor. When you press (RESET), the computer immediately aborts all system activity. The computer, interfaces, and some peripherals are returned to a ready state. If a program is running, any pending or currently active input/output operation is terminated, and information may be lost.

The reset operation is useful when you want to return the system's components to a known configuration before loading or running a program. Resetting the computer erases CRT memory but leaves program memory intact. The graphic scale and pen, data pointers, timers, and *print-all,* CRT apportionment, TRACE, and trigonometric modes are returned to their default power-on state. Peripheral device declarations (PRINTER IS, MASS STORAGE IS, etc.) revert to their power-on assignments.

Refer to the table of Reset Conditions in appendix D, page 324, for a more detailed list of conditions affected by (RESET).

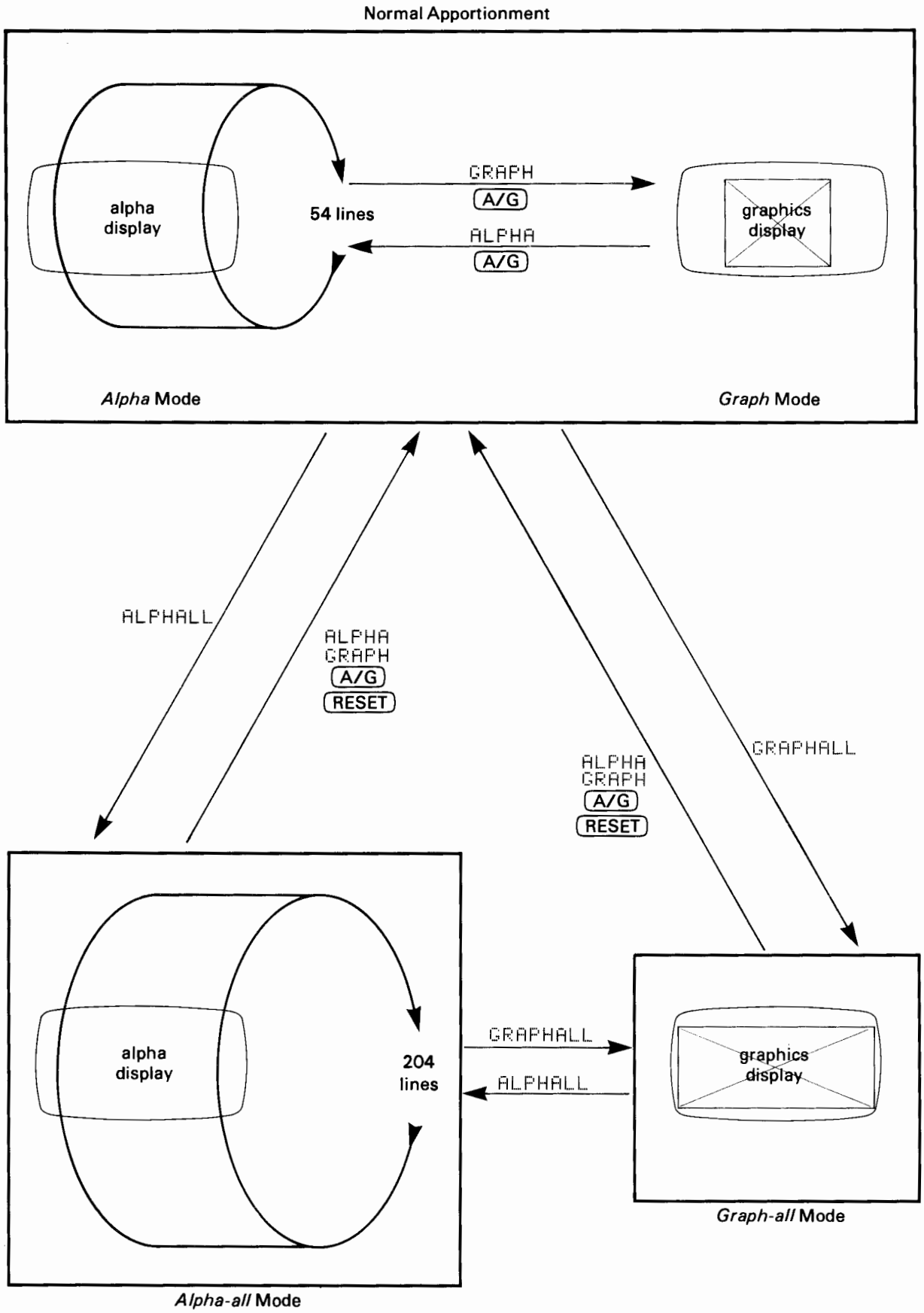


Figure 1.1 CRT Apportionment

Section 2

Arithmetic and Logical Expressions

Introduction

An expression is a combination of numbers, characters, variables, operators, and functions that can be interpreted and evaluated by the computer. This section will cover the following components of expressions:

- Numbers (as they are stored and used by the computer).
- Arithmetic operators.
- Simple numeric and string variables.
- Relational and logical operators.

The math functions will be discussed in section 3.

All the examples shown in this section utilize the computer for manual problem solving rather than for running BASIC programs. However, the expressions discussed here can also be used in programs.

Keyboard Arithmetic

The arithmetic operations that can be performed by the computer are:

- Addition (+).
- Subtraction (−).
- Multiplication (*).
- Division (/).
- Exponentiation (^).
- Integer division (\ or DIV).
- Modulo (MOD).

To compute the answer for an arithmetic expression:

1. Key in the expression, using the numeric keypad or the typewriter keyboard for the numbers and symbols. You do not key in an equal (=) sign.
2. Press (END LINE). The computer will first interpret the meaning of the expression, and then evaluate it for the answer. The answer will appear under the line you just executed. If the computer is unable to understand the expression because you violated a rule of expression construction, you will receive an error message.

Examples:

`5*9` `45`	Enter this expression. The computer returns this answer.
`12-3+7` `16`	Enter this expression. The computer returns this answer.
`5^3` `125`	Enter this expression. The computer returns this answer.

MOD and DIV

In addition to the usual arithmetic operators, +, −, *, / and ^, there are two more arithmetic operators that are often useful. These are DIV (*integer division*) and MOD (*modulo*). They are used exactly like the other five operators.

The DIV operation returns the integer portion of the quotient. The computer performs a normal division, but then truncates all the digits to the right of the decimal point. You can specify integer division by keying in DIV or by using the \ symbol.

The MOD (modulo) operator returns the remainder resulting from a normal division. Given two numbers, A and B, A MOD B is defined by the equation A MOD $B = A - B*$INT(A/B), where INT(A/B) is the greatest integer less than or equal to the quotient of A/B. It turns out that $0 \leqslant (A \text{ MOD } B) < B$ if $B > 0$ and $B < (A \text{ MOD } B) \leqslant 0$ if $B < 0$. By definition, A MOD 0 is A.

Examples:

`16 DIV 5` `3`	Enter this expression. The computer returns this answer.
`16 MOD 5` `1`	Enter this expression. The computer returns this answer.
`5 DIV 16` `0`	Enter this expression. The computer returns this answer.
`-8 MOD 3` `-2`	Enter this expression. The computer returns this answer.
`(-8) MOD 3` `1`	Enter this expression. The computer returns this answer.

The expression `-8 MOD 3` is evaluated the same way as `-(8 MOD 3)` due to the rules of arithmetic hierarchy, discussed next.

The Arithmetic Hierarchy

When an expression contains more than one arithmetic operation, the order in which the operations are performed follows the arithmetic hierarchy. Operations are performed in the following order:

^	Performed first
*, /, MOD, DIV or \	↓
+, −	Performed last

When using the hierarchy to evaluate an expression, the computer performs an operation when the operation directly to the right is of lower or equal rank. Because of the hierarchy rules, the expression `1+3*2` equals 7, since the computer performs the multiplication before the addition operation.

The prescribed order of execution can be altered by using parentheses. When parentheses are used, they take the highest order in the arithmetic hierarchy. For instance, `(1+3)*2` equals 8 because the contents of the parentheses is evaluated before the multiplication is performed. When parentheses are *nested* (that is, when one pair of parentheses is inside another pair), the innermost quantity is evaluated first. So, `2*((2+3)^2)` equals 50, since the innermost parentheses is evaluated to 5, then squared to 25, at which point 25 is multiplied by 2. When parentheses are nested, the number of left parentheses must equal the number of right parentheses.

Example: Suppose you wish to evaluate the expression:

$$2 + \frac{3 \times 6}{(7-4)^2}$$

Key it into the computer in one line as follows:

```
2+3*6/(7-4)^2
```

The computer scans the expression from left to right and performs an operation when the operation directly to the right has lower or equal priority. The preceding expression is executed in the following manner:

Operation	Evaluated Expression
Multiplication	`2+18/(7-4)^2`
Evaluate parentheses	`2+18/3^2`
Exponentiation	`2+18/9`
Division	`2+2`
Addition	`4`

If you are uncertain of the order of execution for an expression, use parentheses to indicate the relative priorities of the operations.

Note: Using parentheses for "implied" multiplication is not allowed. The expression 3(9 − 5) must be entered into the computer as `3*(9-5)`, using the `*` symbol to explicitly indicate multiplication. Note that square brackets, `[ ]`, and parentheses, `( )`, cannot be used interchangeably.

The RESULT Key

The value that is displayed after you press [END LINE] to evaluate a numeric expression is stored in a location called "RESULT." It can be recalled for use in another calculation by pressing [RESLT] (a shifted key).

For instance, if you decide to multiply the last calculation by 3.7:

```
(2+3*6/(7-4)^2)*3.7
```

Use the (RESLT) key to recall the answer of the previous calculation. Key in the expression (RESLT) *3.7 (END LINE). The computer will display:

```
4*3.7
 14.8
```

You may use the (RESLT) key more than once in an expression. For instance, after executing the previous expression, you could compute (RESLT) - (RESLT). The computer displays:

```
14.8-14.8
 0
```

Key in (RESLT) – (RESLT).
The computer returns this answer.

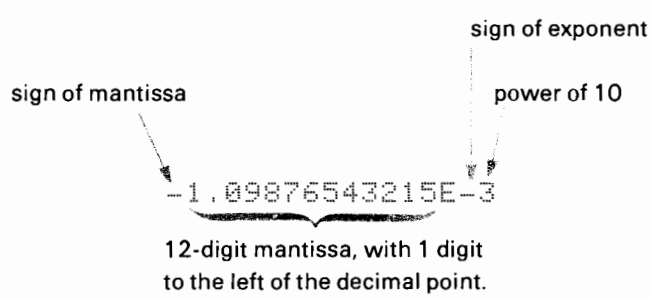
Standard Number Format

All calculations are performed with the full precision of the computer (see Range of Numbers, page 35). For most computations, results appear in an easy-to-read form as specified by ANSI*. Results are displayed or printed in the following standard format unless you specify otherwise using output formatting (discussed in section 10).

- All significant digits of a number (maximum of 12 digits) are printed or displayed. For example, if you type 9876543210.12345 (END LINE), it will reappear on the CRT as 9876543210.12.
- Excess zeros to the right of the decimal point are suppressed. For example, 32.1000000 is output as 32.1.
- Leading zeros are truncated. For example, 003445.6 is output as 3445.6.
- Numbers whose absolute values are greater than or equal to 1, but less than 10^{12} are output showing all significant digits and no exponents.
- Numbers between −1 and 1 are also output showing all significant digits and no exponent if they can be represented precisely in 12 or fewer digits to the right of the decimal point.
- All other numbers are expressed in scientific notation. For instance, .000000000000089 is output as 8.9E-14.

Scientific Notation

When you execute an expression for which the result is too large or too small to be displayed fully in 12 digits, the number will be displayed in scientific notation, as shown below:



*American National Standards Institute

Excess zeros to the right of the decimal point are suppressed.

Examples:

```
60000*90000000
 5.4E12
.00006*.00000009
 5.4E-12
```

Enter this expression.
The computer returns this answer.

Enter this expression.
The computer returns this answer.

You can key in numbers in scientific notation form using the keyboard letter (E) or the symbol (E) provided on the keyboard numeric keypad.

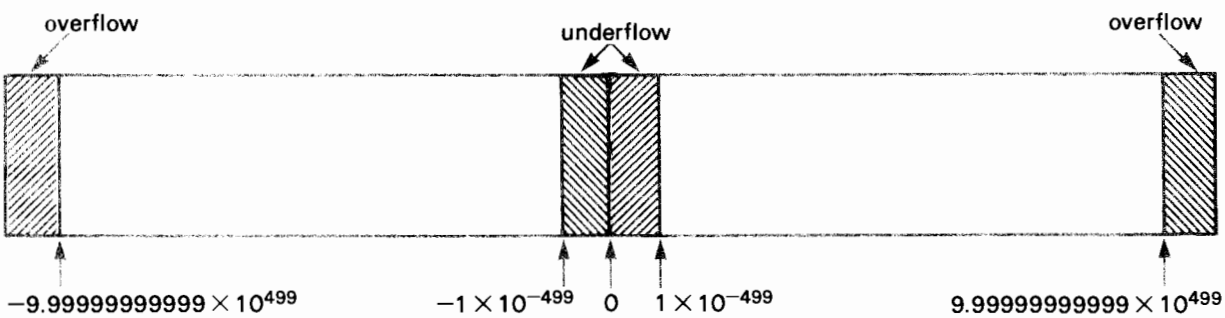
Example: The speed of light, 299790000 (2.99790000×10^8) m/sec, can be squared by entering the following expression.

```
2.9979E8^2
 8.98740441E16
```

Enter this expression.
The computer returns this answer.

Range of Numbers

The range of values that can be entered or stored is shown below:



Variables

Algebraic expressions usually contain symbols or names, called *variables*, that may take on a number of assigned values. In the computer, a variable specifies a location in memory where a piece of information is stored. A name given to the location is used to reference the information stored there. When information is entered into the location, the variable is said to be assigned a value.

The computer allows four types of variables:

- A simple numeric variable is assigned a number value.
- A simple character string variable, usually referred to as a string variable, is assigned a sequence of valid characters—letters, numbers, and/or symbols.
- A numeric array is a collection of numeric variables.
- A string array is a collection of string variables.

With simple variables, each variable name can be assigned only one value or character sequence at a time. An array is really a collection of individual variables, called elements of the array, grouped under a single name and referenced using an array indexing system.

Variables can be assigned values both manually (in calculator mode) and within a program. Calculator-mode variables are temporary—they are cleared from memory whenever you enter or run a program, press (RESET), or execute the SCRATCH command.

Simple string variables will be discussed in section 4. Numeric and string arrays are discussed in section 9. For now, we will concentrate on simple numeric variables.

Precision of Numeric Variables

The computer allows three types of precision for numeric variables:

- REAL numbers are stored with the full precision of the computer. REAL numbers are represented internally with a 12-digit mantissa and a three-digit exponent in the range of −499 through 499; in other words, a 12-digit number in the range $-9.99999999999 \times 10^{499}$ through $-1.00000000000 \times 10^{-499}$, 0, and $1.00000000000 \times 10^{-499}$ through $9.99999999999 \times 10^{499}$.
- SHORT numbers are represented internally with a five-digit mantissa and a two-digit exponent in the range −99 through 99; in other words, a five-digit number in the range -9.9999×10^{99} through -1.0000×10^{-99}, 0, and 1.0000×10^{-99} through 9.9999×10^{99}.
- INTEGER numbers are stored with five digits, with no digits following the decimal point. The range of integers is −99999 through 99999.

All numbers are full precision (REAL) unless you specify otherwise. But you can conserve computer memory if you designate SHORT or INTEGER numbers within a program.

Naming Variables

The computer allows you to use names up to 31 characters long for simple numeic variables. You can use any sequence of letters and numbers, except that the first character must be a letter. You can also use the underline character, _.

Numeric variable names may not include:

- Characters other than letters, numbers, and _, such as *, ^, (, ", etc.
- Blank spaces.

You also are not allowed to use BASIC keywords, such as PRINT, LET, PLOT, etc., as variable names.

Examples:

Acceptable names: VELOCITY, MC2, J, Temperature, BALANCE_DUE

Unacceptable names: 3RDTEST, LENGTH*WIDTH, rate 1, print

The computer distinguishes between uppercase and lowercase letters in variable names (but not in BASIC keywords). Thus, you cannot use uppercase and lowercase letters interchangeably. For instance, NAME, name, NaMe, and nAME are each unique variable names and can be assigned different values.

Assigning Values to Variables

Variables are assigned values using an equal sign to create an assignment statement. Executing the assignment statement stores the value to the right of the equal sign in the location named on the left side of the equation.

Examples: Key in these variable assignments:

`Gravity=9.80665`	Assigns 9.80665 to variable `Gravity`.
`FIVESQUARED=5^2`	Assigns 25 to variable `FIVESQUARED`.

Once variables are assigned, they can be used in math calculations or in other assignment statements. Remember to press (END LINE) after keying in each line.

Examples:

`Gravity*4.5` `44.129925`	Key in this expression. The computer returns this answer.
`FIVESQUARED^2` `625`	Key in this expression. The computer returns this answer.
`FIVECUBED=FIVESQUARED*5`	Variable `FIVECUBED` is assigned the value 125.

Variables can be reassigned to new values. For instance:

`Gravity=32.1740`	`Gravity` is assigned the value 32.1740.
`Gravity=Gravity*.6`	`Gravity` is assigned the value 32.1740 × .6.

To recall the value of any assigned variable, simply type the variable name and press (END LINE).

`Gravity` `19.3044`	Key in the variable name. The computer returns its value.
`FIVECUBED` `125`	Key in the variable name. The computer returns its value.

You can assign the same value to more than one variable in the same line by listing the variables separated by commas. For instance:

`Oranges,Apples,Onions=4`	Key in this assignment.
`Oranges` `4`	Key in the variable name. The computer returns its value.
`Onions` `4`	Key in the variable name. The computer returns its value.

In multiple-assignment statements, the order in which assignments are made is from right to left. Thus, in the example above, `Onions` is the first variable to be assigned the value 4.

Statement Spacing

When you are keying in an expression containing variables, there are a few rules you need to know about spacing, i.e., when to use blank spaces between portions of the statement.

- You must separate variable names from BASIC language keywords with at least one blank space. You may use more than one space if you'd like.

Acceptable:	`INPUT grossincome`
	`FOR counter=1 TO 20`
	`10 MOD 3`
		`INPUT`, `FOR`, `TO` and `MOD` are BASIC keywords.

Unacceptable:	`INPUTgrossincome`
	`FORcounter=1 TO 20`
	`10MOD3`
		`INPUTgrossincome`, `FORcounter` and `MOD3` are interpreted as variable names.

- You must separate BASIC keywords from numeric constants. However, you do not need to insert spaces between keywords and string constants when the strings are enclosed in quotes. (Strings are discussed in section 4.)

Acceptable:	`DISP 23`	Statement directs the computer to display the number 23.
	`DISP"hello"`	Statement directs the computer to display the word `hello`.
Unacceptable:	`DISP23`	The computer interprets `DISP23` as a variable name.
	`DATAhello`	The computer interprets `DATAhello` as a variable name, rather than as a `DATA` statement.

- You are not required to include spaces between variable names and arithmetic operator symbols (e.g., +, *, ^), relational operators (e.g., =, >), or punctuation (e.g., (,)). However, you may place spaces there if you'd like; the computer automatically removes excess spaces.

Acceptable:

```
INPUT grossincome,taxes

Gasconstant=8.3143
```

- You are not allowed to include spaces in the middle of BASIC keywords. For instance, `IN PUT` is not allowed for `INPUT`.

Logical Evaluation

Logical evaluation uses relational and/or logical operators to compare expressions. The expressions being compared may be numbers, variables, or arithmetic and string expressions. Logical evaluation always returns a value of 0 (false) or 1 (true).

Relational Operators

Relational operators are used to determine the value relationship between two expressions. The result of a relational operation is either 1 if the relation is true or 0 if the relation is false.

Operator	Meaning
=	Equal to.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
<> or #	Not equal to (either form is acceptable).

Note that the equal sign is used in both variable assignment statements and in relational operations. The following examples illustrate how the computer differentiates between variable assignments and relational operations.

Examples:

A=3	Interpreted as a variable assignment.
(A=3)	Parentheses specify a relational operation.
3=A	Placing the number on the left specifies a relational operation.

The following examples further illustrate using relational operations. The first three lines assign values to the three variables used.

UN=1	Assigns the value 1 to variable UN.
DEUX=2	Assigns the value 2 to variable DEUX.
TROIS=3	Assigns the value 3 to variable TROIS.
UN<DEUX	Enter this expression.
1	The computer returns 1 for true.
DEUX#TROIS	Enter this expression.
1	True.
UN>TROIS	Enter this expression.
0	False.
UN=(DEUX=TROIS)	Assigns the value 0 to variable UN.
UN	Key in variable name.
0	The computer returns the value of UN.

An important use of relational operators is in making decisions within programs. Program branching is discussed in sections 8 and 11.

Logical Operators

The logical operators AND, OR, EXOR, and NOT are used for creating Boolean expressions. Boolean expressions are evaluated according to the following tables. The parameters *A* and *B* listed in the tables may be relational expressions, but need not be. When *A* and/or *B* are constants or assigned variables, a value of 0 is considered false; all other values are true. Likewise, the computer returns a 0 (false) or 1 (true) result.

AND

A	*B*	*A* AND *B*
T	T	T
T	F	F
F	T	F
F	F	F

OR

A	*B*	*A* OR *B*
T	T	T
T	F	T
F	T	T
F	F	F

EXOR

A	*B*	*A* EXOR *B*
T	T	F
T	F	T
F	T	T
F	F	F

NOT

A	NOT *A*
T	F
F	T

Logical operations are performed according to the following hierarchy. As with arithmetic operations, parentheses are used to reorder priorities.

NOT — Highest priority

=, <, >, <=, >=, #

AND

OR, EXOR — Lowest priority

Examples:

```
UN=1
DEUX=2
TROIS=3
TRES=3
```

Variable assignments.

```
UN<DEUX AND TROIS=TRES
 1
```

Both relational operations are true.
True, the result of true AND true.

```
UN AND TROIS=DEUX
 0
```

The value of UN is 1 (true); TROIS=DEUX is false.
False, the result of true AND false.

```
UN OR DEUX
 1
```

UN and DEUX are each non-zero, therefore true.
True.

```
2=DEUX EXOR NOT DEUX
 1
```

DEUX=2 is true, NOT DEUX is false.
True, the result of true EXOR false.

Math Hierarchy of Arithmetic and Logical Operators

The hierarchy of the mathematical and logical operators discussed so far is as follows:

Parentheses ( )

[^]

[*], [/], MOD, DIV or [\]

[+], [-], NOT

Relational operators =, <, >, <=, >=, <> or #

AND

OR, EXOR

Highest priority

Lowest priority

Notes

Section 3

Mathematics Functions and Statements

A *function* is a prescription for manipulating a given value or set of values that returns a single result. The value(s) acted upon are called *arguments* or *parameters,* and are enclosed in parentheses. If a function requires more than one argument, the arguments are separated by commas. Arguments may be constants, variables, or mathematical expressions containing variables or other functions.

The computer's BASIC language contains a number of predefined functions that can be used in programming or executed from the keyboard. This section covers the computer's predefined math functions. Also covered are the time functions provided to access the computer's internal clock.

In addition to math functions, which manipulate numeric arguments, the computer provides a number of predefined functions for manipulating sequences (strings) of characters. Character strings and string functions are discussed in section 4.

Number Alteration Functions

The following functions allow you to alter or extract portions of numbers. Argument *X* can be a number, numeric variable, or a numeric expression.

Function	Action
ABS(*X*)	Returns the absolute value, or magnitude, of *X*.
IP(*X*)	Returns the integer part of *X*.
INT(*X*) FLOOR(*X*)	Return the greatest integer less than or equal to *X*; differs from IP with negative numbers.
FP(*X*)	Returns the fractional part of *X*.
CEIL(*X*)	Returns the smallest integer greater than or equal to *X*.

INT and FLOOR perform identical operations.

Examples:

```
ABS(-235)
 235
```
|−235|.

```
ABS(5.9)
 5.9
```
|5.9|.

```
IP(123.012)
 123
```
Integer part of 123.012.

```
IP(-45.66)
-45
```
Integer part of -45.66.

```
INT(123.012)
 123
```
Greatest integer ≤ 123.012.

```
INT(-45.66)
-46
```
Greatest integer ≤ -45.66.

```
FLOOR(-45.66)
-46
```
Greatest integer ≤ -45.66.

```
FP(123.012)
 .012
```
Fractional part of 123.012.

```
FP(-45.66)
-.66
```
Fractional part of -45.66.

```
CEIL(123.012)
 124
```
Smallest integer ≥ 123.012.

```
CEIL(-45.66)
 -45
```
Smallest integer ≥ -45.66.

General Math Functions

Several of the general math functions require two arguments and several require no argument.

Function	Action
SQR(*X*)	Returns the positive square root of nonnegative *X*.
SGN(*X*)	Sign function, returning 1 if *X* is positive, 0 if *X* is 0, and -1 if *X* is negative.
MAX(*X*,*Y*)	Compares *X* and *Y*, returning the larger of the two values.
MIN(*X*,*Y*)	Compares *X* and *Y*, returning the smaller of the two values.
RMD(*X*,*Y*)	Divides *X*/*Y* and returns the remainder from the division.
PI	Returns the 12-digit approximation of π; 3.14159265359.
INF	Returns machine infinity (+9.99999999999E499).
EPS	Returns machine epsilon, the smallest positive number (1.E-499).
RND	Returns the next number in a sequence of random numbers with $0 \leq number < 1$.

Examples:

```
SQR(88)
 9.38083151965
```
Square root of 88.

```
SGN(-7)
-1
```
Sign of -7.

```
SGN(23)
 1
```
Sign of +23.

```
MAX(4.5,4.6)
 4.6
```
Larger of the two arguments.

```
MAX(-1.3,-PI)
-1.3
```
Larger of the two arguments.

```
MIN(INF,5E20)
 5.E20
```
Smaller of the two arguments.

```
MIN(EPS,.00887)
 1.E-499
```
Smaller of the two arguments.

```
SPHERE=PI*5^3
```
Evaluates and assigns value to SPHERE.

```
SPHERE
 392.699081699
```
Key in variable name.

Two of the math functions, RMD and RND, require further discussion.

The Remainder Function: RMD

Given two arguments, *X* and *Y*, RMD(*X*,*Y*) returns the remainder of the division *X*/*Y*. The function can be evaluated by the equation RMD(*X*,*Y*) = *X* − *Y* * IP(*X*/*Y*). Although RMD and MOD are defined similarly, they yield different results when *X* and *Y* have different signs. To see the difference, examine the following two versions of the operation (-380)/360.

$$\begin{array}{r} -1 \\ 360\,\overline{)\,-380} \\ -360 \\ \hline -\ 20 \end{array} \qquad\qquad \begin{array}{r} -2 \\ 360\,\overline{)\,-380} \\ -720 \\ \hline +340 \end{array}$$

```
RMD(-380,360)=
-380-360*IP(-380/360)=-20
```

```
(-380) MOD 360=
-380-360*INT(-380/360)= +340
```

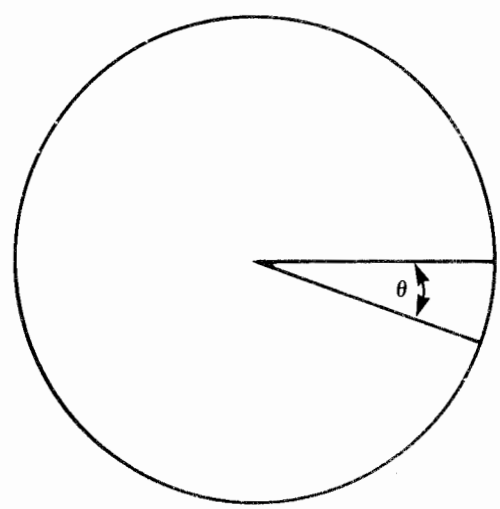
Examples: The following calculations illustrate applications of RMD and MOD.

Resolve the angle -380 to lie between -360° and +360°.

```
ANGLE=RMD(-380,360)
ANGLE
-20
```

Resolve the same angle to lie between 0° and 360°.

```
ANGLE=(-380) MOD 360
ANGLE
 340
```



Generating Random Numbers

The *random number* function is used in programs to generate a sequence of pseudorandom numbers. RND returns a number greater than or equal to 0 and less than 1.

Examples:

```
RND
 .529199358633

RND
 4.35821814444E-2
```

When you turn the power on or press (RESET), the same sequence of random numbers is generated. This is because RND uses the same seed (the number upon which the sequencing is based) each time the computer is reset.

The RANDOMIZE statement defines a new seed for the random number generator.

```
RANDOMIZE [seed]
```

The brackets around *seed* indicate that the parameter is optional. If you include the seed, it can be any number within the computer's range. Whenever you wish to use the same sequence of random numbers, merely execute RANDOMIZE using the same seed. For any non-zero seed, 5×10^{13} values can be generated before the sequence repeats. If you specify a seed equal to 0, RND always returns the value 0.

If you do not include a new seed parameter, the RANDOMIZE statement will generate a non-zero seed using the the computer's internal timer.

Examples:

```
RANDOMIZE 125
RND
 .463385058782
```

The Logarithmic Functions

The computer's logarithmic functions are:

Function	Action
LOG(*X*)	Returns the natural (base e) logarithm of a positive number.
EXP(*X*)	Returns the natural antilogarithm by raising e (2.71828182846) to the *X*th power.
LGT(*X*)	Returns the common (base 10) logarithm of a positive *X*.

The common antilog can easily be calculated using 10^*X*.

Trigonometric Functions and Statements

The computer allows you to specify angles in decimal degrees, radians, or grads. When the computer is turned on or reset, it is in radians mode; that is, it assumes all angles are in radians. You can change the trigonometric mode by executing one of the following statements:

```
DEG
```

DEG sets the computer to *degrees* mode. There are 360 degrees in a circle.

```
GRAD
```

GRAD sets the computer to *grads* mode. There are 400 grads in a circle.

```
RAD
```

RAD returns the computer to *radians* mode. There are 2π radians in a circle.

The computer provides ten trigonometric functions. Angle X is interpreted in radians, degrees, or grads, according to the current trigonometric mode.

Function	Action
SIN(X)	Sine of X
ASN(X)	Arcsine of X; $-1 \leqslant X \leqslant 1$. In first or fourth quadrant.
COS(X)	Cosine of X.
ACS(X)	Arccosine of X; $-1 \leqslant X \leqslant 1$. In first or second quadrant.
TAN(X)	Tangent of X.
ATN(X)	Arctangent of X; in first or fourth quadrant.
CSC(X)	Cosecant of X.
SEC(X)	Secant of X.
COT(X)	Cotangent of X.
ATN2(Y,X)	Arctangent of Y/X in proper quadrant. (X,Y) is the rectangular coordinate position of a point.

In addition to the preceding 10 functions, the following two functions convert angles between radians and degrees, independent of the trigonometic mode.

Function	Action
DTR(X)	Converts angle X in degrees to radians.
RTD(X)	Converts angle X in radians to degrees.

Examples:

```
DEG
SIN(30)
 .5
```

Sets computer to degrees mode.
Key in function.
Computer returns sine of 30 degrees.

```
ATN(1)
 45
```

Key in function.
Computer returns arctangent of 1.

```
RAD
COS(PI)
-1
```

Sets computer to radians mode.
Key in function.
Computer returns cosine of π radians.

```
RTD(PI)
 180
```

Key in this function.
Computer converts π radians to degrees.

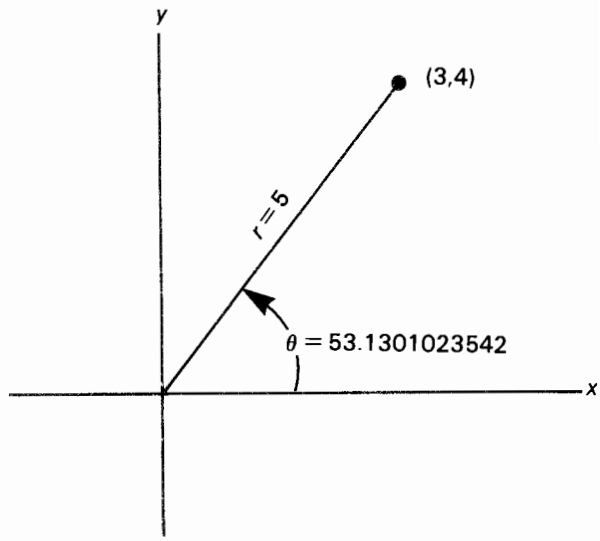
The ATN2(Y,X) function is useful for converting rectangular X,Y coordinates to r,θ polar coordinates using the equations:

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}(y/x) \qquad \text{where } -\pi < \theta \leqslant \pi$$

For instance, point (3,4) translates into polar coordinates as follows:

```
DEG
SQR(3^2+4^2)
 5
ATN2(4,3)
 53.1301023542
```



The Total Math Hierarchy

The order of all mathematical operations is presented below.

Parentheses ( )

Functions

[^]

[*], [/], MOD, [\] or DIV

[+], [-], NOT

Relational operators =, >, <, >=, <=, <> or #.

AND

OR, EXOR

Highest priority

↓

Lowest priority

Time Functions

The computer contains an internal timer that allows you to set and recall the current time and date both from the keyboard and within programs. The timer is designed to meet an accuracy of within 1 second per hour.

> Note: Due to effects of temperature variations, aging, shocks, and vibrations on its quartz crystal, timer accuracy may vary slightly.

When you turn the computer on, the time and date are set to 0 and the computer begins counting in milliseconds. When the timer reaches 86,400 seconds (24 hours), the date is incremented by 1 and the seconds counter is returned to 0.

The SETTIME statement allows you to set the timer as follows:

SETTIME *seconds parameter* , *date*

The seconds parameter can be any numeric expression evaluating to a number ranging 0 through 86400. If you set the seconds parameter equal to the number of seconds elapsed since midnight, the timer will reach 86400 at midnight and automatically increment the date.

You can enter the date in any form you wish, as long as it evaluates to a number ranging 1 through 99999. A commonly used form is *YYDDD*, where *YY* is the year and *DDD* is the Julian date. If you use a non-integer, the number will be rounded to the nearest integer. A date parameter larger than 99999 is entered as 99999.

Two functions allow you to access the computer's timer for information. The TIME function recalls the current value of the seconds counter.

```
TIME
```

The DATE function recalls the current date, expressed in the format you specified in the SETTIME statement.

```
DATE
```

Example: The timer is first set to the current time, February 2, 1981 at 1:45:30 p.m. The seconds parameter is computed as follows:

$$
\begin{array}{r}
13 \text{ hours} = 13 \times 60 = 780 \text{ minutes} \\
+ \underline{\quad 45 \text{ minutes}} \\
825 \text{ minutes} \times 60 = 49500 \text{ seconds} \\
+ \underline{\quad 30} \\
49530 \text{ seconds}
\end{array}
$$

The date is entered as a five-digit integer in the form *YYDDD*, or 81033 (33rd day of year 81).

```
SETTIME 49530,81033
```

A few minutes later, the current time is recalled:

```
TIME
 49894.439
DATE
 81033
```

Section 4

String Variables and Functions

So far, you've been dealing with numeric constants and variables. The second major type of data consists of character strings and string variables. A character string is a continuous group of characters, grouped together and enclosed by quotation marks. Character strings can include any of the computer's characters, including spaces. A string variable is a location in computer memory where a character string can be stored.

Naming String Variables

The maximum length of a simple string variable name is 32 characters. The first character in the name must be a letter; the final character must be a dollar sign, $. The remainder of the name can be any combination of letters, digits, and the underscore character, _. As with numeric variables, you cannot include blank spaces and you cannot use uppercase and lowercase letters interchangeably. BASIC keywords followed by $ are not allowed.

Examples:

Acceptable names: `PHONENUMBER$`, `DateOfBirth$`, `Last_name$`

Unacceptable names: `ThisIsNotaStringVariable`, `4SALE$`, `RENT-FEE$`, `ALPHA$`

Assigning and Dimensioning String Variables

Character strings are assigned to string variable names much like numbers are assigned to numeric variables.

Examples:

`GREETING$="Hello"`	Assigns string to `GREETING$`.
`SALUTATION$="How are you?"`	Assigns string to `SALUTATION$`.
`GREETING$`	Key in string variable name.
`Hello`	The computer returns assigned string.
`SALUTATION$`	Key in string variable name.
`How are you?`	The computer returns assigned string.

The number of characters in a string variable is referred to as the length of the string variable. A string variable can have any length up to 65530 characters, subject to the amount of available memory. However, string variables longer than 18 characters must be dimensioned. Dimensioning a string variable reserves memory for all the characters in the string.

To see how dimensioning works, try to make the following assignment:

```
ADVICE$="Never put off till tomorrow what you can do today."
```

The computer will beep and return `Error 56 : STRING OVF` (overflow), indicating that the string is too long.

The `DIM` statement is used to dimension string variables:

`DIM` *string variable name*[*string length*][, *string variable name*[*string length*]...]

The string length enclosed in brackets must be at least as large as the number of characters in the string.

> Note: A variable assignment to an undimensioned string variable implicitly dimensions the string variable to a length of 18 characters. An attempt to explicitly dimension an implicitly dimensioned string variable generates `Error 35 : DIM EXIST VRBL`.

Examples: Before executing the following statements, execute the `SCRATCH` command to erase all previous variable assignments.

```
DIM ADVICE$[50],WISDOM$[30]
ADVICE$="Never put off till tomorrow what you can do today."
WISDOM$="Never give unwanted advice."
```

The shortest possible string is the null string, which contains no characters or blanks. The following statement assigns a null string to the string variable `NULL$`.

```
NULL$=""
```

String Expressions

The computer allows you to manipulate and modify strings in several ways.

Concatenation

Concatenation means to connect or link in a series or chain. When strings are concatenated, one string is attached to the end of another. The symbol used for string concatenation is the ampersand (&).

Examples:

`Prefix1$="BIRTH"`
`Prefix2$="SUN"`
`Word1$="DAY"`
`Word2$="ON"`
	Assigns strings to string variables.

`Occasion$=Prefix1$ & Word1$`	String variable assignment.
`Day$=Prefix2$ & Word1$`	String variable assignment.
`Occasion$` `BIRTHDAY`	Key in variable name. Computer returns assigned string.
`Day$` `SUNDAY`	Key in variable name. Computer returns assigned string.
`Occasion$ & " " & Word2$ & " " & Day$` `BIRTHDAY ON SUNDAY`	Key in this expression. Computer returns this string.

If the string resulting from a concatenation is longer than 18 characters, a string variable to which it is assigned must be dimensioned large enough to accommodate the result. Note that you can use string constants (for instance, the blank spaces `" "` in the previous example) in concatenation expressions.

Substrings

A *substring* is a portion of a string made up of zero or more contiguous characters. You specify a substring by placing subscripts that refer to character positions in brackets after the string name.

The form for specifying a substring is:

string variable name [*beginning character position* [, *ending character position*]]

If you place only one number in brackets, it is interpreted as the beginning character position; the ending character becomes the last character in the string.

Examples:

`DIM STRING$[32]`	Maximum length of `STRING$` is 32 characters.
`STRING$="An example of substrings follows"`	Assigns string to `STRING$`.
`STRING$[4,24]` `example of substrings`	Key in substring reference. The computer returns the specified substring.
`STRING$[1,1]` `A`	Key in substring reference. The computer returns the first character in the string.
`STRING$[26]` `follows`	Key in substring reference. The computer returns the characters from position 26 to the end of the string.

`N=5`
`STRING$[4,N]`
`ex`
	Key in variable assignment.
	Key in substring reference.
	The computer returns characters 4 and 5.

A substring reference in which the second parameter is 1 less than the first specifies a null string.

Example:

`STRING$="abcde"` `NULL$=STRING$[2,1]`	String variable `NULL$` is assigned the null string.

The first substring parameter cannot be greater than the second by more than 1; for example, `A$[3,1]` is not allowed and will return an error.

Modifying String Variables

After you have assigned a character string to a string variable, all or part of the string can be changed. To change the complete string, simply reassign it using an assignment statement.

Examples:

`LUNCH$="salad"`	Assign variable LUNCH$.
`SUPPER$="hamburger"`	Assign variable SUPPER$.
`LUNCH$`	Key in variable name.
`salad`	Computer returns assignment.
`SUPPER$=LUNCH$`	Reassign variable SUPPER$.
`SUPPER$`	Key in variable name.
`salad`	Computer returns assignment.
`LUNCH$="fruit"`	Reassign variable LUNCH$.
`LUNCH$ & " " & SUPPER$`	Key in concatenated variables.
`fruit salad`	Computer returns assignment.

You can also replace one substring with another. The original string can be shortened or lengthened up to its dimensioned size. If the new substring is shorter than the original, the unassigned characters in the string become blank spaces.

Examples:

`W$="locate"`	Assign variable W$.
`W$[6]="ion"`	Assign substring beginning at character 6.
`W$`	
`location`	New string assigned to W$.
`W$[5]="l"`	Assign substring beginning at character 5.
`W$`	
`local`	New string assigned to W$.
`W$[7]="issue"`	Assign substring beginning at character 7.
`W$`	
`local issue`	New string assigned to W$.

Substrings can also be used to replace the beginning or middle of character strings. If the new substring is shorter than the one being replaced, the extra character positions become blank spaces. If the new substring is longer than the one being replaced, excess characters in the new substring are dropped.

Examples:

`Q$="CONFUSION"`	Assigns string to variable Q$.
`Q$[4,6]="DIT"`	Assigns substring to characters 4 through 6.
`Q$`	
`CONDITION`	

```
Q$[1,3]="TRA"
```

Assigns substring to characters 1 through 3.

```
Q$
TRADITION
```

```
Q$[1,3]="E"
```

Assigns substring to characters 1 through 3.

```
Q$
E  DITION
```

Characters 2 and 3 become blanks.

```
Q$[1,3]="EXTRA"
```

Five-character substring assigned to characters 1 through 3.

```
Q$
EXTDITION
```

Two substring characters are dropped.

String and String-Manipulating Functions

Seven functions allow you to create, analyze, and manipulate strings. The following four numeric functions analyze strings, returning a numeric result:

Function	Action
LEN(*string*)	Returns the number of characters in a string.
POS(*string 1* , *string 2*)	Returns the position of string 2 in string 1.
VAL(*string*)	Returns the numeric value of a string expression composed of digits.
NUM(*string*)	Returns a number corresponding to the decimal code of the first character in the string.

The following three string functions return a string result:

Function	Action
VAL$(*numeric expression*)	Generates a string representation of a number.
CHR$(*numeric expression*)	Returns the character whose decimal code equals the value of the numeric expression.
UPC$(*string*)	Converts all lowercase letters in the string to uppercase letters.

The Length Function

The LEN function returns the number of characters in a string, regardless of its dimensioned maximum length.

LEN(*string expression*)

The string expression can include quoted text, string variable names, substrings, and concatenated strings.

Examples:

`LEN("123456")` `6`	Key in function expression. Computer returns length of quoted string.

`WORD$="random"`
`LEN(WORD$)`
`6`
	Assigns string to variable WORD$.
	Key in function expression.
	Computer returns length of assigned string.

`LEN(WORD$[2,4])` `3`	Function argument is a substring. Computer returns length of substring.
`LEN(WORD$&"123456")` `12`	Function argument is concatenated expression. Computer returns length of expression.

The Position Function

The POS (*position*) function returns the position of one string within another. The two strings must be separated by a comma.

POS(*first string*, *second string*)

If the second string is contained within the first, the POS function returns the character position in the first string of the first character in the second string. If the second string is not contained within the first or if the second string is the null string, the function returns the value 0. If the second string occurs in more than one place within the first string, only the first occurrence is returned.

Examples:

`POS("A summer day","summer")` `3`	Key in function expression. "summer" starts at position #3 in first string.
`STRING$="perpetual"`	Assigns string to variable STRING$.
`SUB$="pet"`	Assigns string to variable SUB$.
`POS(STRING$,SUB$)` `4`	Key in function expression. Computer returns position #4.

Converting Strings to Numbers

Normally, the characters in a string are not recognized as numeric data and can't be used in numeric calculations. The VAL function converts a string or substring containing digits, including an exponent, into a number that can be used in calculations.

VAL(*string expression*)

The first character in the string to be converted must be a digit, a plus or minus sign, a decimal point, or a space. Additional blank spaces can appear anywhere in the string; they will be ignored. Additional signs (+ or −) can follow the first sign character.

The string to be converted must include at least one digit and can have only one decimal point. An E character preceded by one or more digits is interpreted as exponential notation. The E can be followed by one sign and up to three digits.

Examples:

```
PRES$="A. LINCOLN, 16"
```
Assigns string to variable PRES$.

```
VAL(PRES$[13])*10
 160
```
Multiply value of substring starting at position #13 by 10.
Numerical result of 16*10.

```
Z$="INCH/METER  2.54E-2"
```
Assign string to variable Z$.

```
F=VAL(Z$[11])
F
 .0254
```
Position #11, leading space, is ignored. Digit and sign after E are interpreted as exponent.
The number is output in standard format.

```
VAL("- -+ 55")
 55
```
Multiple signs are evaluated algebraically.

The string can contain more than one number. All contiguous numerics are considered part of the number until a non-numeric, an incorrectly positioned sign, or a second decimal point is encountered.

Examples:

```
C$="1ST, 10TH, 430TH"
```
Assigns string to variable C$.

```
VAL(C$)
 1
VAL(C$[6])
 10
VAL(C$[12,12])
 4
```
Letters following the digits halt the conversion.

```
VAL(".111.222.333")
 .111
```
The decimal point following .111 halts the conversion.

```
VAL("+123+456+789")
 123
```
Plus sign after 123 halts the conversion.

Converting Numbers to Strings

The VAL$ function converts a number to the string representation of the number in standard format.

VAL$(*numeric expression*)

Examples:

```
DEGREES$=VAL$(360)
```
Assigns string "360" to variable DEGREES$.

```
DEGREES$ & " degrees"
360 degrees
```
Two strings are concatenated.
Computer returns concatenated string.

```
VAL$(5E1*5)
250
```
The VAL$ argument can be an expression.

```
LEN(VAL$(5E1*5)
 3
```
Length of string "250".

Character Conversions

Each of the computer's characters (numbers, letters, and symbols) has a corresponding decimal code assigned to it. The table of Character and Key Codes on page 323 lists the decimal code assigned to each character. The numbers range 0 through 255.

The computer has three functions based on the decimal codes for the computer's characters:

Function	Action
CHR$(*numeric expression*)	Converts a decimal code to its corresponding character.
NUM(*string expression*)	Converts a character to its corresponding decimal code.
UPC$(*string expression*)	Converts all lowercase letters in string to uppercase letters.

CHR$

The CHR$ (*character*) function converts a numeric value into a string character using the character decimal codes. Numbers outside the range 0 through 255 but in the range -32767 through +32767 are converted MOD 256 to that range. Numbers greater than +32767 or less than -32767 are evaluated as ±32767.

```
CHR$(numeric expression)
```

Examples:

```
CHR$(35)
#
LETTER$[1,1]=CHR$(65)

LETTER$[1,1]
A
```

Key in this function.
The computer returns this character.
Assign character to substring LETTER$[1,1].
Key in substring.
Computer returns its assignment.

The CHR$ function allows you to use quotation marks within a quoted string in PRINT and DISP statements. Since quotation marks are used to define the beginning and end of a literal message, you cannot use quotation marks themselves.

Example:

```
DISP "MAKE A ";CHR$(34);"WISH";CHR$(34)
MAKE A "WISH"
```

Key in this statement.
The computer displays this message.

NUM

The NUM (*numeric*) function converts an individual string character to its corresponding decimal value. If more than one character is included in the string expression, the NUM function finds the decimal equivalent of the first character.

```
NUM(string expression)
```

The NUM function returns the value 0 for the null string.

Examples:

`NUM("A")` `65`	Key in function. Computer returns decimal value.
`NUM("σ")` `9`	To display σ, hold down (CTRL) while you type (I).
`NUM("SKULL")` `83`	Only the first character in the string is converted.

UPC$

The UPC$ (*uppercase*) function enables you to convert a string containing lowercase letters to a string composed of all uppercase letters. The function is useful for comparing strings without regard to upper- and lowercase.

UPC$(*string expression*)

Examples:

`ANSWER$="yes"`	Assigns string to variable ANSWER$.
`TEST$=UPC$(ANSWER$)`	Assigns converted string to variable TEST$.
`TEST$` `YES`	Key in variable name. Computer returns assigned string.
`"NO"=UPC$("no")` `1`	Relational = operation. Value 1 indicates the relation (equality) is true.

Comparing String Variables

Strings and string variables can be compared using the relational operators (=, <, >, <=, >=, <> or #). Strings are equal if they are the same length and contain exactly the same characters in the same order. Since the comparison process uses the decimal equivalents of the characters, uppercase and lowercase letters are not regarded as equivalent (unless the UPC$ function is used).

String inequalities compare the two strings character by character, from left to right, until a difference is found. If a difference is found, the string containing the character with the lower decimal equivalent is regarded as smaller.

If one string terminates before a difference is found, the shorter string is regarded as smaller.

Examples:

`"ABCD"="ABCd"` `0`	The two strings are not equal.
`"ABCD"<"ABCd"` `1`	First inequality is in 4th position. Decimal code for "D" is smaller than the decimal code for "d".
`"ABCD">"ABCDE"` `0`	First string terminates before inequality is found. Therefore, the relationship is false.

Part II
BASIC Language Programming

Section 5

Introduction to Programming

In sections 2 through 4, you used the calculating capabilities of the computer to perform a number of mathematical and string operations. The real time-saving power of your computer, however, comes from using programs to instruct the computer to perform an ordered set of instructions.

If you have read section 4 of the introductory manual, then you've already acquired some experience in entering, running, storing, and loading computer programs. With the information covered in that section, you can use the computer to run programs developed by Hewlett-Packard or other sources of software support. The list of Series 80 applications software available from Hewlett-Packard is continually updated and expanded as we try to provide you with software to meet your professional and personal computing needs.

To appreciate and utilize the full power of the computer, however, you will want to learn how to write your own programs. Part II of this manual is designed to introduce you to the BASIC language and to the computer's powerful editing and program debugging features. The organization of Part II is designed to lead you through increasingly more sophisticated programming concepts and statements. In section 5, you'll be introduced to a number of definitions and procedures used routinely in computer programming. Section 6 describes a number of fundamental BASIC language statements used to enter, manipulate, and output data. Sections 7 through 13 build upon one another as they introduce additional statements and flexibility of the BASIC language.

Parts III and IV cover two special topics. The statements providing the computer's powerful graphics capabilities are explained in part III. In part IV, you'll learn how to use your mass storage system as an extension of the computer to create, access, and manipulate files of information.

The Structure of BASIC

A BASIC program is an organized set of instructions that directs the computer to perform certain operations. Once a program is written, it is entered into computer memory, where it can be executed once or repeatedly. The instructions must be written and entered in a form recognized by the computer—the computer's own language.

Statements

The instructions in a BASIC program are called *statements*. Like any other language, BASIC has a vocabulary, called keywords, that the computer understands and can act upon. In addition, BASIC statements have rules of syntax. Each statement must be entered in a certain form that follows the rules for constructing that statement.

The *keywords* in a statement identify operations to be performed (*executable* statements) or give the computer information it needs to execute other statements (*declaratory* statements). These actions will become clearer to you as each of the BASIC statements is described. For now, you might want to examine a short list of some BASIC keywords. Brief descriptions of their functions are included to help you see the difference between executable and declaratory statements.

Executable	Declaratory
PRINT — Prints output.	DIM—Defines bounds of arrays.
DISP — Displays output.	REAL—Defines precision of variables.
LET — Variable assignment.	DEF FN — Declares user-defined function.
PLOT — Plots graphics data.	DATA — Establishes list of data items.
GOTO — Orders branching.	ON TIMER# — Establishes timer interrupts.
CLEAR — Clears display.	PRINTER IS } Declares address of
CREATE — Creates data file.	MASS STORAGE IS } peripheral device.

Most statements can be executed both within a program and from the keyboard (in calculator mode). If the statement is typed in without a preceding line number, the computer immediately executes the statement.

Statement Numbers

Every line in a program must be preceded by a unique statement number. Statements can be numbered any integer from 1 through 99999. When statements are entered into the computer, they are stored in ascending order, regardless of the order in which they are entered. Normal program execution proceeds from the lowest numbered statement to the highest numbered statement. A group of program statements covered in sections 8 and 10 allow the program to "branch" away from the normal order of execution.

Commands

A *command* is an instruction that must be executed from the keyboard. Commands are used to manipulate programs or peripheral devices. If you attempt to enter a command preceded by a statement number, the computer will return an error.

The computer's commands are listed below. Some will be explained in this section; others are explained elsewhere in this manual.

AUTO	REN
CONT	RUN
DELETE	SCRATCH
INIT	STORE
LOAD	UNSECURE

(CONT), (INIT), and (RUN) are immediate execute keys. When you press the key, the command is executed immediately. You can also execute these three commands as you would any other command—by typing the command (or using a typing aid) and pressing (END LINE).

Clearing Computer Memory

Before entering a program, you should always clear any previously entered program from computer memory. There are three ways to clear memory:

1. Turn off the computer.
2. Execute the SCRATCH command. To do this type SCRATCH (END LINE). Alternatively, you can use the typing aid provided on (k12).
3. Load another program from mass storage. When you execute the LOAD command, the current program in computer memory is erased before the new program is loaded.

Statement and Command Syntax

A one word command such as SCRATCH is easy to describe. Many BASIC statements and commands have more complicated syntax rules governing their construction. Throughout this manual, the following conventions are used to describe how statements and commands are formulated:

DOT MATRIX	Items in DOT MATRIX are BASIC language keywords and punctuation that must be typed exactly as shown, except that lowercase letters can be substituted for uppercase letters. The computer replaces any lowercase letters in BASIC keywords with uppercase letters.
italics	Items in *italics* are numeric constants, numeric expressions, and string expressions that must be included in the statement.
[ ]	Brackets are used to enclose optional items.
stacked items	When items are placed one above the other, one and only one must be chosen.
...	An elipsis placed after an item or series of items within brackets indicates the contents of the brackets may be repeated.

When example programs are provided, the entire program is shown in dot matrix. This is because the program must be typed in exactly as shown, including variable names and constants. (Program remarks, however, need not be entered.)

A typical syntax description might look like this:

```
                  "format string"
PRINT USING  statement number  [; item[ ; ,  item...]]
                  statement label
```

The words PRINT USING are typed in exactly as shown. You must then include a valid string expression enclosed in quotes, a statement number, or a statement label. The semicolon and first item are optional. If they are included, additional items may be added, separated by commas or semicolons.

Examples:

```
PRINT USING 100; NUMBER, STRING$
```

statement number *item* *item*

```
PRINT USING "/////"
```

format string

Writing and Entering a BASIC Program

Before writing a program, you need to define the problem by determining what you want the program to produce (*output*), what you already know (*input*), and how information will flow through the program to transform the inputs into outputs.

The following short program converts a numeric value for speed in units of meters per second (the input) to the corresponding values in kilometers per hour and knots (the output). Take a few moments to look the program over, since it contains a number of fundamental BASIC statements that will be explained in the next section. Don't enter the program into the computer, however, until you've read the next few pages.

```
10 REM *****This program converts speed in m/sec to km/hour and knots*****
20 DISP "ENTER SPEED IN METERS PER SECOND" ! Prompt for input.
30 BEEP
40 INPUT SPEED !                              Key in information.
50 READ FACTOR1,UNIT1$,FACTOR2,UNIT2$
60 LET SPEED1=SPEED*FACTOR1
70 LET SPEED2=SPEED*FACTOR2
80 PRINT SPEED;"METERS/SEC =";SPEED1;UNIT1$;" AND";SPEED2;UNIT2$
90 DATA 3.6,"KM. PER HOUR",1.944,"KNOTS"
100 END
```

The BASIC keywords used in this program are REM, DISP, BEEP, INPUT, READ, LET, PRINT, DATA, and END.

Before you enter the program, you should become familiar with the computer's automatic numbering, the spacing requirements for statements, and the use of the (END LINE) key.

Automatic Line Numbering

The AUTO command provides for numbering statements automatically as they are entered into computer memory, saving you the time of typing the numbers yourself.

AUTO [*beginning statement number* [, *increment value*]]

You may type in the word AUTO or use the typing aid provided on (k1).

When you execute an AUTO command, the computer displays the specified beginning statement number at the left of the screen. After you've typed in a statement and pressed (END LINE) to enter the statement into computer memory, the computer increments the statement number by the specified value and displays the new statement number on the next line.

Note that the beginning statement number and increment value are optional. If no increment value is specified, statement numbers are incremented by 10. If no beginning statement number is specified, numbering begins at 10.

Example:

AUTO 100,5	Executing this command causes numbering to begin with 100 and increment by 5.

To stop automatic numbering, backspace over the unwanted line number and type NORMAL (END LINE). Auto numbering is also halted by executing any statement or command from the keyboard. For instance, after entering the final statement, you can run the program by pressing (RUN) without executing NORMAL.

Spacing

The computer requires that you use blank spaces in statements to separate BASIC keywords, identifiers, and numeric constants from one another. Identifiers are numeric variable names, string variable names, and statement labels. (Statement labels are covered in section 8.) Keep in mind that the computer is able to recognize BASIC keywords, and that it interprets other valid sequences of letters and digits as variable names.

You do not need to insert spaces between keywords or identifiers and the arithmetic operators (+, −, *, /, \, ^), the relational operators (=, <, >, <=, >=, <>, #), parentheses ( ), commas, and semicolons. However, logical operators (AND, OR, EXOR, NOT) must be separated from keywords and identifiers by blank spaces.

You cannot insert spaces into keywords, identifiers, or logical operators. Non-essential spaces inserted at other places are ignored, with one exception. Spaces between the statement number and the beginning of the statement are preserved, allowing you to indent program lines for better readability.

Refer to the discussion of statement spacing on page 38 for additional information.

Examples: The following examples should make the spacing rules clearer. The arrows indicate places where blank spaces can be inserted without affecting the meaning of the statement.

```
10 FOR COUNT=START TO FINISH

20 FORCOUNT=STARTTOFINISH

30 X=F OR COUNT=STARTTOFINISH
```

Statement 10 is a FOR . . . TO statement; COUNT, START, and FINISH are variable names.

Statement 20 is a variable assignment. The value of variable STARTTOFINISH is assigned to variable FORCOUNT.

Statement 30 computes the logical or of F with COUNT=STARTTOFINISH and assigns the result (0 or 1) to variable X.

Uppercase Versus Lowercase

As discussed previously, the computer distinguishes between uppercase and lowercase letters in variable names. Thus, substituting uppercase letters for lowercase letters (or vice versa) results in a new variable name.

BASIC keywords, on the other hand, may be entered using both uppercase and lowercase letters. Once the computer recognizes a keyword, it converts any lowercase letters in the keyword to uppercase letters. Thus, you may enter:

```
10 PRINT "Hello"     or     10 print "Hello"
```

When you list the program, you will obtain

```
10 PRINT "Hello"
```

Statement Length

The maximum permissible length of statements is 159 characters, including the line number and blank spaces. That's equivalent to two full lines of 80 characters each, minus one space on the second line for an invisible carriage return character, generated when you press (END LINE). However, for certain statements containing complicated mathematical expressions or numerous variable references, the maximum permissible line length may be less than 159 characters.

Example: The following statement is too large to be entered.

```
10 DISP (1*(2*(3*(4*(5*(6*(7*(8*(9*(10*(11*(12*(13*5)))))))))))))
Error 85 : EXPR TOO BIG
```

Keep in mind that the function of (END LINE) during program entry is to enter the entire statement into computer memory. When a line is full, the cursor automatically moves to column 1 of the next line. Since (END LINE) does not function like a typewriter carriage return key, you should not press (END LINE) until you've finished typing the statement.

Entering Program Statements

You can enter program statements into computer memory in either of two ways:

- Use the LOAD command to retrieve a program from mass storage. This was covered briefly in section 4 of the introductory manual and is explained in greater detail in section 21 of this manual.
- Type in the program statements from the keyboard.

Program statements are entered into computer memory by typing in the statement (with its line number) and then pressing (END LINE). When you press (END LINE), the computer translates the statement into its own internal language and checks the statement to insure that it follows the syntax rules. If the syntax is unacceptable, the computer returns an error. If no error is detected, the statement is entered into program memory.

If you type in a statement longer than 80 characters, the cursor will automatically wrap around to the next line. Do not press (END LINE) until the entire statement has been entered.

All calculator-mode variable assignments are scratched whenever a program line is entered into computer memory.

Example: Before entering the units conversion program into computer memory, you should:

1. Erase program memory by executing the SCRATCH command.
2. Clear the CRT display by pressing (CLEAR). This is not a necessary step, but it increases the legibility of the display.
3. Execute the AUTO command to take advantage of automatic numbering. Since we wish the program to start at statement number 10 and increment by 10, the optional AUTO parameters are not necessary.

Now, enter the units conversion program by typing each statement exactly as shown on page 66, pressing (END LINE) after each statement. After you've entered statement 100 and the computer returns line number 110, backspace over the line number and type NORMAL (END LINE) to stop automatic numbering.

Running a Program

Once a program is entered in computer memory, it can be run by pressing the (RUN) key or by typing and entering the RUN command. The computer immediately begins executing the program.

Whenever a program is running, the power light blinks on and off. The blinking stops when program execution is complete, if program execution is halted by an error, or if the program is paused. The light continues to blink when a program is temporarily halted waiting for input.

We will use the units conversion program to convert a speed of 331.4 meters per second to its corresponding value in kilometers per hour and knots. To run the program, press the (RUN) key. The program first causes the computer to beep and display:

```
ENTER SPEED IN METERS PER SECOND
?
```

At this point, the program automatically pauses, waiting for you to enter data. Enter the data, 331.4, by keying in the number and pressing (END LINE).

```
ENTER SPEED IN METERS PER SECOND
?
331.4
```

The system printer will print:

```
331.4 METERS/SEC = 1193.04 KM. PER HOUR AND 644.2416 KNOTS
```

If you have not declared a PRINTER IS device, the output will appear instead on the display.

At this point, program execution is completed. You can run the program again by pressing the (RUN) key.

Pausing a Running Program

You can stop a running program by pressing the (PAUSE) key. When you press (PAUSE), program execution halts and the power light stops blinking. Program execution can be continued from where it was halted by pressing (CONT) (continue).

Pausing a program is discussed in greater detail in section 7.

Section 6

Fundamental BASIC Statements

Each of the statements used in the units conversion program on page 66 has a prescribed set of rules for its construction, called syntax, and a precise effect on the operation of the program. This section will cover a number of BASIC statements, including all those appearing in the units conversion program.

The REM Statement

Comments (remarks) should be included in your programs to make your program logic easier to follow. Remarks can be inserted using the REM statement or by using the comment delimiter, !. Program comments are not executed by the computer.

The syntax of the REM statement is:

REM [*any combination of characters*]

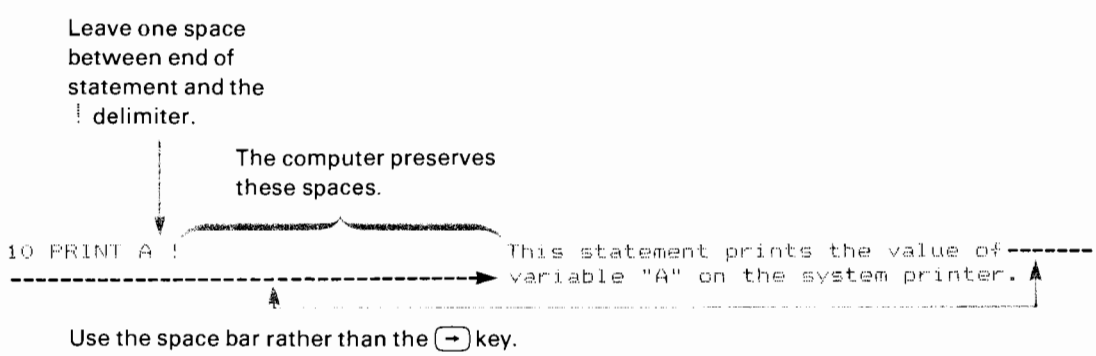
The comment delimiter, !, can be anywhere in a program statement after the line number. All characters following ! are considered part of a comment unless the ! is within quotes. The comment delimiter allows you to include comments along with the statement on the same line.

Example:

```
10 REM *****This program doesn't do anything.*********************************
20 END !    All programs must end with an END or STOP statement.
```

For readability , you may want to arrange your program comments in tabular format with all comments beginning in the same column. If your comment requires two rows, use the space bar to "wrap" the comment to the second row.

Example:



The computer preserves one blank space between the end of the statement and the ! delimiter, and at least one space after the delimiter.

The DISP Statement

The DISP (*display*) statement enables the program to output text and variables to the CRT. The statement has the syntax:

```
DISP [item [; item...]]
```

The items may be:

- Variable names.
- Constants.
- Numeric expressions.
- String expressions.
- Quoted text.
- The TAB function (covered in section 10).

The punctuation between items determines whether the output is spaced with *compact* format (;) or with *wide* format (,).

Example:

```
10 REM **********THIS PROGRAM ILLUSTRATES USING THE DISP STATEMENT**********
20 LET NUMBER=25 !                        Assigns value to NUMBER
30 LET STRING$="TODAY IS THE" !           Assigns value to STRING$
40 DISP STRING$;NUMBER;"TH OF FEBRUARY,";1981;".  BEST WISHES."
50 END

TODAY IS THE 25 TH OF FEBRUARY, 1981 .  BEST WISHES.
```

When items are separated by semicolons, the output is spaced in compact format. Numbers are displayed with a leading blank or minus sign and a trailing blank to separate them from the next item. Strings are output with no leading or trailing blanks.

The difference between using semicolons and commas can be demonstrated by replacing the semicolons in statement 40 with commas. To do that, retype statement 40 as shown below, and enter the statement into computer memory. The new version will replace the old version.

```
40 DISP STRING$,NUMBER,"TH OF FEBRUARY,",1981,".  BEST WISHES."
```

Now run the program again.

```
TODAY IS THE          25                    TH OF FEBRUARY,        1981
.  BEST WISHES.
```

When items are separated by commas, the items are displayed left-justified in 21-column fields beginning at columns 1, 22, 43, and 64. If the item placed in column 64 is too large to fit on the line, the item will instead be displayed in the first field on the next line. Items longer than 21 characters will occupy more than one field.

Two or more commas after an item cause one or more character fields to be skipped. For example, execute the following DISP statement from the keyboard:

```
DISP,100,,200
```

The comma after DISP causes the first field to be skipped. The two commas after 100 cause the third field to be left empty.

field skipped	100	field skipped	200
	column 22		column 64

When no items are included after DISP, a blank line is displayed.

> Note: When a DISP statement executed in calculator mode (or the last DISP statement in a program) places the final DISP item in columns 64 through 80, the next line is filled with blank spaces. The blank line is considered part of the previous line; the cursor automatically skips the blank line and moves to the line beneath it.

When the display list ends with a comma or semicolon, items in the list are not immediately displayed. Instead, the items are placed into a storage location, called the display buffer, where they remain until:

- Another DISP statement without a comma or semicolon at the end of the DISP list is executed.
- An INPUT statement is executed (explained on pages 74 through 76).
- The display buffer is full. The buffer is a location in memory used for temporary storage. The display buffer can hold up to 80 characters. If you specify a shorter line length in a CRT IS 1 statement, the specified line length becomes the buffer capacity.

Example:

```
10 DISP "ENTER MONTH, DAY OF YOUR BIRTHDAY"
20 INPUT MONTH$,DAY
30 DISP !                      Displays blank line.
40 DISP "MY BIRTHDAY IS ";!    Text placed in buffer until next DISP statement is
                               executed.
50 DISP MONTH$;DAY !           Text, MONTH$, and DAY are displayed.
60 END
```

```
ENTER MONTH, DAY OF YOUR BIRTHDAY
?
APRIL,20

MY BIRTHDAY IS APRIL 20
```

The PRINT Statement

The PRINT statement directs output to the system printer declared by the PRINTER IS statement. If you have not established a system printer, PRINT output is displayed on the CRT. The syntax of the PRINT statement is:

```
PRINT [item[ , ; item...]]
```

Like the display list, the print list may include numeric and string variable names, constants, numeric and string expressions, quoted text, and the TAB function.

Commas and semicolons have the same effect on spacing in PRINT statements as they do in DISP statements. A comma after an item causes the next item to be left-justified in the next 21-column field. The number of fields on a line depends on the column width of the system printer. An 80-column printer has three full fields and one 17-column field per line.

Most printers are capable of printing only 96 of the 256 characters available on the computer display. These are the characters having decimal codes in the range 32 through 127. On most printers, the first 32 characters (decimal codes 0 through 31) are control characters. You should not include these characters as print items unless you are familiar with your printer's control codes.

In addition to compact (using semicolons) and wide (using commas) spacing, the computer provides formatted printer and display output. Section 10, Printer and Display Formatting, covers this topic.

The INPUT Statement

The INPUT statement allows you to interact with a running program to assign constants or expressions to program variables from the keyboard. The INPUT statement is programmable only; it cannot be executed from the keyboard.

```
INPUT variable name [ , variable name...]
```

When an INPUT statement is executed, a question mark appears on the display and program execution pauses until you enter a valid expression for each variable name in the input list, separating the expressions by commas. The entire input list cannot exceed 159 characters.

The power light continues to blink while a program awaits input.

Example: The following program computes the area of a piece of pie, assuming the pie has been sliced into equal portions.

```
10 REM *****THIS PROGRAM COMPUTES THE AREA OF A PIECE OF PIE*****
20 DISP "ENTER TYPE PIE, PIE DIAMETER" !     Prompts for input.
30 INPUT PIETYPE$,DIAMETER
40 DISP "NUMBER OF EQUAL SLICES PER PIE" !  Prompts for input.
50 INPUT PIECES
60 AREA=PI *DIAMETER^2/(4*PIECES) !          Computes area.
70 DISP "AREA OF SLICE OF ";PIETYPE$;" PIE =";AREA
80 END
```

Press (RUN). The program prompts (requests) you to enter the type of pie and the pie's diameter. You must respond to this prompt with a string (quoted or unquoted) and a numeric expression, separated by a comma. The second prompt requests numeric input.

```
ENTER TYPE PIE, PIE DIAMETER
?
PIZZA,16
NUMBER OF EQUAL SLICES PER PIE
?
8
AREA OF SLICE OF PIZZA PIE = 25.1327412287
```

When responding to an input prompt, you must enter all required values, separated by commas, before pressing (END LINE). If you respond improperly to an input prompt, the computer displays an error message and another question mark (?). The error causes the computer to ignore your previous response to the prompt; respond to the new prompt by entering all values required by the input list.

Values for string variables can be entered quoted or unquoted; in the above example, PIZZA was entered unquoted but could have been enclosed in quotation marks. Using quotation marks allows you to enter a comma as part of the string assignment, or to include leading and/or trailing blanks. You may enter the null string ("") in response to a string input request.

Note: When you respond to an INPUT prompt, the computer interprets the response as an expression, assigning the value of the expression to the corresponding input variable. An unquoted string is interpreted as a variable name if the string has the form of a valid variable name of the proper type (numeric or string). For instance, if, in response to:

```
INPUT A
```

you respond:

```
NUMBER
```

then the computer interprets NUMBER as the name of a numeric variable and assigns the value of NUMBER to variable A. A response of "NUMBER" would generate an error. Likewise, if you respond to:

```
INPUT STRING$
```

with the entry:

```
NAME$
```

then the computer interprets NAME$ as the name of a string variable, assigning the value of NAME$ to STRING$. To assign the string "NAME$" to STRING$, enclose the response in quotes.

Statements 20 and 40 in the previous example are used in conjunction with the INPUT statement to inform you what data the program is requesting. Without statements 20 and 40, the program displays the question mark only. In some applications, you may wish to display the question mark on the same line as the DISP message. To do this, place a semicolon at the end of the DISP statement preceding the INPUT statement.

For example, if you change statement 20 to:

```
20 DISP "WHAT TYPE OF PIE, WHAT IS THE PIE DIAMETER";
```

the program will display the prompt:

```
WHAT TYPE OF PIE, WHAT IS THE PIE DIAMETER?
```

The function of certain keys (such as (RUN), (LIST), and (CONT)) change when a program is paused for input. Refer to the table of Key Response During Program Execution on page 325 for a list of these keys and their functions.

Entering Input in Typewriter Mode

The FLIP statement was introduced in the discussion of the alphanumeric keyboard in section 1. The statement is used to toggle the keyboard between BASIC mode (unshifted uppercase letters and shifted lowercase letters) and typewriter mode (unshifted lowercase letters and shifted uppercase letters).

When the FLIP statement is included in a program, it has the same effect as when it is executed from the keyboard. Thus, you can design your programs to accept keyboard input in BASIC or typewriter mode.

Example: The following program toggles the keyboard into and out of typewriter mode. The first character of both data entries is a shifted letter.

```
10 FLIP !                    Toggles keyboard to typewriter mode.
20 DISP "ENTER FIRST NAME"
30 INPUT FIRSTNAME$
40 FLIP !                    Toggles keyboard to BASIC mode.
50 DISP "ENTER LAST NAME"
60 INPUT LASTNAME$
70 DISP FIRSTNAME$,LASTNAME$
80 END
```

```
ENTER FIRST NAME
?
Dee
ENTER LAST NAME
?
fOLT
Dee                    fOLT
```

The BEEP Statement

The BEEP statement is used to produce an audible tone of variable frequency and duration.

BEEP [*tone* , *duration*]

The optional tone and duration parameters must be integers in the range 1 through 99999. If no optional parameters are specified, the frequency is approximately 2000 hertz and duration is 100 milliseconds.

Example: The following program loops between statements 10 and 30. The tone parameter equals 10 the first time through the loop, and is incremented by 50 each pass through.

```
10 FOR I=10 TO 500 STEP 50
20    BEEP I,250
30 NEXT I
40 END
```

Notice that the frequency decreases as the tone parameter I increases, and that the duration is a function of the frequency. The BEEP parameters are based on the computer's internal timer. The following formulas can be used to produce a particular frequency and duration:

TONE = 613062.5/(11*FREQ)-134/11

DURATION =SECONDS*613062.5/(11*TONE+134)

FREQ is the desired frequency in hertz.

SECONDS is the desired duration in seconds.

Or, when the frequency is known:

DURATION =SECONDS*FREQ

Variable Assignments: The LET Statement

Any numeric variable can be assigned a value using an assignment statement. String variables can be assigned string expressions using the assignment statement; however, the expression must produce a string less than or equal to the dimensioned size of the string variable (18 characters if not explicitly dimensioned).

The keyword in an assignment statement is LET. However, its use is optional.

[LET] *numeric variable* [, *numeric variable...*] = *numeric expression*

[LET] *string variable* [, *string variable...*] = *string expression*

The LET statement assigns the numeric or string expression on the right of the equal sign to the variable(s) on the left of the equal sign. Any variables contained in the expression on the right should have been previously assigned. The computer displays a NULL DATA warning if you include an unassigned variable in the expression on the right. Program execution continues using a value 0 for an unassigned numeric variable and the null string for an unassigned string variable.

Examples:

```
10 LOOPCOUNTER=1
10 LET LOOPCOUNTER=1
10 LET LOOPCOUNTER,TIMESTHROUGH=1
10 PLANET$="JUPITER"
10 LET PLANET$,RINGEDPLANET$="SATURN"
```

Delaying Program Execution: The WAIT Statement

The WAIT statement is used to introduce a delay between execution of two successive program statements.

WAIT *milliseconds*

The milliseconds parameter can be any expression that evaluates to a number within the computer's range. However, the minimum wait time is 0 and the maximum wait time is 27 minutes (1,666,650 milliseconds). A negative number is interpreted as 0; positive numbers greater than 1666650 are interpreted as 27 minutes.

The WAIT statement can be interrupted by pressing (PAUSE) or any other key. When you press (CONT), program execution begins immediately at the next statement, regardless of whether or not the WAIT time elapsed during the pause period.

Example:

```
230 WAIT 30000
```

Program execution is delayed 30 seconds.

Note: The WAIT statement should not be used for timings requiring accuracy greater than 1/60 second. Instead, use one of the computer's internal timers, discussed on page 149, Timer Interrupts.

The WAIT statement is ignored when executed in calculator mode.

The READ and DATA Statements

The READ and DATA statements together provide another way to assign values to variables within a program. They provide a convenient way to supply the running program with variable assignments without having to pause the program to key in data.

READ *variable name* [, *variable name...*]

DATA *constant* [, *constant...*]

The READ statement specifies the variables whose values are to be assigned within the program. The variables can be of any type —simple numeric, simple string, or subscripted (array elements).

The constants within the DATA statement may be numbers or character strings to be assigned to the READ statement variables. Each data constant must match the type variable to which it is being assigned. The character string constants may be quoted or unquoted; quotation marks allow you to include commas and leading and trailing blanks in the character string.

Example:

```
10 READ BALANCE,INTEREST,NAME$
20 PRINT "STATEMENT FOR FEBRUARY"
30 PRINT NAME$,BALANCE,INTEREST
40 DATA 10000,51.76,"SMITH, HARRY"
50 END


STATEMENT FOR FEBRUARY
SMITH, HARRY          10000                51.76
```

The DATA statement is declaratory, and DATA items are simply ignored if there are no corresponding READ statement variables. Thus, your program can contain more DATA elements than are accessed by READ statements.

DATA statements can be positioned anywhere in the program except after THEN or ELSE in an IF...THEN or IF...THEN..ELSE statement (these statements are discussed in section 8). DATA statements are not allowed in multistatement lines. Remarks cannot be added to DATA statements using the comment delimiter, !.

The order of DATA statements with respect to one another is critical, since they are used in the order of their line numbers. The computer maintains an internal data "pointer" to locate the next data element to be read. The left-most element of the lowest numbered DATA statement is read first. When a data element is read, the pointer moves to the next item in the DATA statement. When all the elements of a data list have been read, the pointer moves to the left-most item in the next-higher numbered DATA statement. If there are no higher-numbered DATA statements, the data pointer remains at the end of the last DATA statement and any attempt to read additional data generates an error.

Because data elements are accessed by a data pointer, one READ statement can use more than one DATA statement. Conversely, a READ statement can access less than an entire data list; the pointer simply remains positioned at the element following the last one accessed.

Example:

```
10 READ NUMBER,NUMBERNAME$
20 DISP NUMBER,NUMBERNAME$
30 GOTO 10 !                          Unconditional branch to statement 10.
40 DATA 1,ONE,2,TWO,3,THREE,4,FOUR,5,FIVE,6
50 DATA SIX,7,SEVEN
60 DATA 8,EIGHT,9,NINE,10
70 END
```

```
 1                    ONE
 2                    TWO
 3                    THREE
 4                    FOUR
 5                    FIVE
 6                    SIX
 7                    SEVEN
 8                    EIGHT
 9                    NINE
Error 34 on line 10 : NO DATA
```

Notice that statement 10 must access a new data statement after NUMBER 6 is read, and again after NUMBERNAME$ SEVEN is read. When the data pointer moves past the item 10 in statement 60, the program is unable to locate a character string for NUMBERNAME$ and an error occurs.

The RESTORE Statement

The RESTORE statement provides for reusing data by moving the data pointer back to a specified DATA statement.

RESTORE[*statement label* / *statement number*]

The statement number or statement label must correspond to a DATA statement. (Statement labels are discussed in section 8).

The RESTORE statement resets the data pointer to the first item of the specified statement. If no statement number or statement label is specified, the pointer moves to the program's lowest-numbered DATA statement.

Example:

```
10 READ ONE,TWO,THREE
20 READ FOUR,FIVE,SIX
30 DISP ONE;TWO;THREE;FOUR;FIVE;SIX;
40 RESTORE 80 !          Moves data pointer to beginning of statement 80.
50 READ SEVEN,EIGHT,NINE
60 DISP SEVEN;EIGHT;NINE
70 DATA 1,2,3
80 DATA 4,5
90 DATA 6
100 END
```

```
1  2  3  4  5  6  4  5  6
                  ↑  ↑  ↑
            SEVEN  EIGHT  NINE
```

The END and STOP Statements

An END or STOP statement must be the last statement executed by a running program.

END

STOP

The two statements can be used interchangeably to inform the computer where the program ends and to terminate program execution.

Frequently, END or STOP is the highest numbered program statement. However, both END and STOP may appear elsewhere in a program. The computer provides a number of branching operations that allow program execution to proceed around an END or STOP statement. For example, programs frequently use END or STOP to separate the main part of a program from user-defined functions and subroutines. (User-defined functions and subroutines are discussed in section 11.)

The NORMAL Statement

You've already been introduced to using the NORMAL statement to exit *print-all* mode and AUTO line numbering. The statement has the syntax:

NORMAL

The NORMAL statement actually has several functions.

When NORMAL is executed from the keyboard (in calculator mode):

- AUTO line numbering, if in effect, is halted.
- *Print-all* mode, if in effect, is cancelled.
- Tracing operations are cancelled for any program subsequently run or continued. Tracing operations are discussed in section 13.

The NORMAL statement can also be executed within a program, where it:

- Immediately cancels *print-all* operations.
- Immediately halts tracing operations.

Error Messages

There are three types of errors that may occur during the development and execution of a program: *syntax* errors, *semantic* errors, and *run-time* errors.

A *syntax* error occurs if you violate a rule of proper statement or command construction while entering a program statement or executing a statement or command from the keyboard. The computer tries to interpret a line first as a BASIC statement and, failing this, then as a keyboard expression. Examples of syntax errors are misspelled keywords, missing operators or punctuation, and illegal constant or variable names.

The computer performs the syntax check when you press (END LINE). If an error is found, the computer returns an appropriate error message and positions the cursor at the first character at which the error was detected. The statement will not be executed or entered into program memory. Syntax errors are corrected by editing the statement or command using the computer's editing keys.

> Note: If you receive an ambiguous error message while attempting to execute a calculator-mode expression, try executing the same expression from the keyboard in a DISP statement. The computer will interpret the line as a statement rather than as an expression, and you may receive a more helpful error message.

Example: The following expression is entered into the computer first as an expression, then as a DISP statement item. The second entry returns a more descriptive error message.

```
5*3*(4+(7*9)
Error 88 : BAD STMT

DISP 5*3*(4+(7*9)
Error 80 : ) EXPECTED
```

Semantic errors occur in the context of your program. During program execution, the computer checks to verify that individual statements make sense with respect to other statements in the program. Examples of semantic errors are referencing a nonexistent statement, duplicate user-defined functions, and illegal array dimensions. If a semantic error is detected, the computer returns an appropriate message and program execution halts. Semantic errors are usually corrected by adding, deleting, or editing statements. Semantic errors can be detected before the program is run by initializing the program (refer to Initializing a Program, page 89).

Run-time errors are detected by the computer only while the program is running; initializing a program does not detect run-time errors. Examples include referencing a nonexistent array element, READ . . . DATA variable mismatch, and attempting to write data to a nonexistent mass storage file. The computer returns an appropriate error message and halts program execution.

Appendix F includes a complete list of error numbers and messages returned by the computer. The introductory manual contains a table of errors returned by the computer's integrated interface. Read the entry for the appropriate error number and message.

Note: If your system includes any optional plug-in ROMs, the list of errors you may receive is expanded. The ERROM function returns a number identifying which ROM generated the error.

```
ERROM
```

If the error was issued by the computer rather than by an optional ROM, ERROM returns 0, 1, or 208. Consult documentation accompanying each ROM for the ROM number and a list of the ROM's errors.

Error messages report the first error that was detected. There may be other unreported error conditions.

Recovering From Math Errors

Math errors, such as using an improper argument in a math function or overflow, would normally halt program execution. The computer, however, provides default values for out-of-range results that occur using certain math operations and functions. These default values override the error condition and prevent the error from halting execution. This default error-processing is automatically implemented when the system is turned on.

The errors and default values are:

Error Number	Condition	Default Value
1	Underflow	0
2	REAL precision overflow SHORT precision overflow INTEGER precision overflow	±9.99999999999E499 ±9.9999E99 ±99999
3	COT or CSC of n*180°; n=integer	±9.99999999999E499
4	TAN or SEC of n*90°; n=odd integer	±9.99999999999E499
5	Zero raised to negative power	±9.99999999999E499
6	Zero raised to zero power	1
7	Uninitialized numeric variable Uninitialized string variable	0 " "
8	Division by zero	±9.99999999999E4999

When the error occurs, the system alerts you to the error by displaying an appropriate warning message.

Example: In the following program, the computer outputs a warning and sets RESULT equal to the default overflow value. The DISP statement then displays the value of RESULT.

```
10 ZERO=0
20 TWENTY=20
30 RESULT=TWENTY/ZERO
40 DISP RESULT
50 END
```

```
Warning 8 on line 30 : /ZERO
 9.99999999999E499
```

The DEFAULT OFF statement cancels use of the default math values for math errors. In the DEFAULT OFF state, the system returns an error instead of a warning, and program execution halts.

The default values can be restored by executing DEFAULT ON.

DEFAULT ON OFF

Example: If you edit the previous program by adding statement 5, the computer returns an error at line 30, and program execution halts.


```
5 DEFAULT OFF
10 ZERO=0
20 TWENTY=20
30 RESULT=TWENTY/ZERO
40 DISP RESULT
50 END
```

```
Error 8 on line 30 : /ZERO
```

To restore the default math values, execute DEFAULT ON from the keyboard.

Multistatement Lines

A multistatement line contains two or more BASIC statements with the same statement number joined by the @ symbol. Multistatement lines can also be executed from the keyboard.

Examples:

```
100 DISP "ENTER BALANCE" @ INPUT BALANCE
PRINTER IS 701 @ PRINT ALL
```

The statements in a multistatement line are executed in the order in which they appear, left to right. The number of statements that can be combined depends on the complexity of the statements; the total length of the line cannot exceed 159 characters.

In addition to shortening program listings, multistatement lines conserve program memory (five bytes for each statement).

There are several programming precautions involving multistatement lines:

- Since statements are executed from left to right, a GOTO statement, if included, should be the last statement in the line.

Example:

```
40 DISP "BRANCHING TO STATEMENT 1000" @ GOTO 1000
```

If the order were reversed, the DISP statement would not be executed.

A GOSUB statement, however, can be placed anywhere in the line. The subroutine's RETURN statement causes the program to branch to the statement following the GOSUB statement.

- Make sure you understand how the computer handles multistatement lines in "decision making" program lines before attempting to include them in programs. For instance, statements after THEN in an IF...THEN operation are executed only if the relational test is true; statements after ELSE in an IF...THEN...ELSE operation are executed only if the relational test is false.

Example:

```
                        Executed if A=1                    Executed if A#1
20 IF A=1 THEN DISP "A=1" @ GOTO 50 ELSE DISP "A#1" @ GOTO 90
```

- Declaratatory statements (such as DIM, REAL, SHORT, and INTEGER) can be made in multistatement lines, but they must be the last statement in the line. DATA statements are not allowed in multistatement lines.

- Anything following REM or the ! delimiter is a remark.

Example: In statement 50 below, the value of A will never be computed.

```
50 REM **compute A** @ A=(X+Y)^3
```

- Multistatement lines are recognized by other HP Series 80 Personal Computers. However, programs involving multistatement lines may not be transportable to other BASIC computers.

- When you are constructing multistatement lines, take care to preserve readability of the program. Two simple statements may be easily read; however, two lengthy statements may be difficult for others to understand and may interfere with program debugging.

- If an ON ERROR ... GOSUB declarative causes branching during execution of a multistatement line, the recovery routine's RETURN statement transfers program execution to the line number following the multistatement line.

Notes

Section 7

Pausing and Editing Programs

The computer has been designed to allow you to easily alter a program in computer memory. This section will cover:

- Adding, deleting, and changing program statements.
- Listing a program on the CRT or printer.
- Initializing and running programs.
- Halting and resuming program execution.
- Determining the amount of available computer memory.

Editing Program Statements

Program statements can be edited the same way you edit anything appearing on the display, using the display editing keys:

[-LINE] [BACKSPACE] [-CHAR] [I/R] [↑] [↓] [←] [→] [↖] [ROLL]

There are two ways to edit a statement in program memory:

1. Recall the program statement to the display by listing the program or by using the [ROLL] key. Next, use the display editing keys to move the cursor to the desired location and make the necessary changes. Finally, press [END LINE] while the cursor is anywhere on the statement line to enter the edited statement into program memory. Before entering the statement, make sure there are no unwanted characters beyond the last character on the line. If there are, remove them by moving the cursor beyond the end of the statement and pressing [-LINE].

2. Retype the entire statement, including the statement number, as you wish it to be. Then press [END LINE] to enter the statement into program memory. The new version of the statement entirely replaces the old version, regardless of the relative lengths of the two versions.

Deleting Statements

You can delete program statements in any of three ways:

1. To delete one statement, type the statement number and press [END LINE].

2. List the program and use the cursor control keys to place the cursor after the statement number of the statement to be deleted. Use the [-LINE] key to erase the statement and then press [END LINE].

3. To delete a section of the program, use the DELETE command.

The DELETE command can be used to delete an individual statement or block of statements. You can type in DELETE or use the typing aid provided on (k5).

DELETE *first statement number* [, *last statement number*]

If only one statement number is specified, then only that program statement is deleted from program memory. If you specify two statement numbers, statements within the specified range are deleted.

Examples:

30	Deletes statement 30.
DELETE 40	Deletes statement 40.
DELETE 60,90	Deletes statements 60 through 90, inclusive.

Adding Statements

To add statements to a program, merely type and enter them into program memory. The computer automatically sorts statements by statement number, so that the new statements are positioned within the program according to their statement numbers.

Renumbering a Program

The REN (*renumber*) command is used to renumber the program in program memory.

REN [*beginning statement number* [, *increment value*]]

Just as with the AUTO command, you can optionally specify the new starting statement number and the increment between successive statements. If no increment value is specified, numbering is incremented by 10. If neither parameter is specified, numbering begins at 10 and is incremented by 10.

The REN command automatically renumbers the entire program, including any branches within programs. References to statement numbers within the program (e.g., GOTO 50) are automatically changed to their new statement numbers. An exception is that the REN command will not change the statement number parameters of any LIST or PLIST statements in the program.

If, during renumbering, the computer reaches line 99999 before the entire program has been renumbered, the computer issues Warning 90 : LINE >99999 and renumbers the program using values of 1 for the beginning statement number and the increment value.

Listing a Modified Program

As discussed previously, the (LIST) and (PLST) keys are immediate execute keys used for listing the program on the display or system printer.

The LIST and PLIST statements can be executed within programs or from the keyboard. The statements include optional parameters that allow you to specify the portion of the program to be listed.

LIST [*beginning statement number* [, *ending statement number*]]

PLIST[*beginning statement number* [, *ending statement number*]]

If neither optional parameter is specified, the statement is executed as if you had used the (LIST) or (PLST) key.

If you specify only the beginning statement number in the LIST statement, listing begins with that statement and continues for one full screen. If both statement numbers are specified, that portion of the program will be displayed; the screen automatically rolls up until all the specified program lines have been displayed.

If you specify only the beginning statement number in a PLIST statement, all program statements from that statement to the end of the program are listed on the system printer. If both parameters are specified, that portion of the program is listed.

To halt a LIST or PLIST operation, press any alphanumeric key.

When the last program line is output during a keyboard LIST or PLIST operation, the computer displays the amount of available memory (in bytes). If the LIST or PLIST statement is executed within a program, the listing does not include the amount of available memory.

Initializing a Program

The program in computer memory can be initialized by pressing the (INIT) key or by executing the INIT command.

INIT

When a program is initialized:

- Any current calculator-mode variable assignments are scratched.
- The computer allocates memory to all program variables.
- Program variables are set to undefined values.
- The program is checked for semantic errors (for example, referencing non-existent statements, duplicate user-defined functions, dimensioning the same variable more than once).
- The program pointer, used by the system to indicate the next statement to be executed, is set to the first statement in the program.

If a program is not initialized prior to being run, memory is allocated to program variables as they are encountered in the program. If your program requires a large amount of memory for variables, initializing the program will inform you if the computer has insufficient memory available to allocate all program variables by returning a memory overflow error.

If you edit a program while it is paused, program variables are no longer allocated. The program should be initialized before execution is resumed. (Refer to Continuing a Paused Program, page 92, for additional information.)

Calculator-mode variable assignments to variables allocated by an initialized program (e.g., X=5 where X is a program variable) are known to the program. To retain the assignment, the program must be executed using the (CONT) (*continue*) key or the CONT command; pressing (RUN) causes the variable assignment to be lost.

Example: Enter and run the following program, pressing (CONT) when the program pauses at statement 20.

```
10 DISP "X=";X
20 PAUSE !
30 DISP "Y=";Y
40 END
```

```
Program pauses until continued.
```

```
Warning 7 on line 10 : NULL DATA
X= 0
Warning 7 on line 30 : NULL DATA
Y= 0
```

Now, press (INIT) to initialize the program and then enter the calculator-mode assignment X=3. Press (CONT). When the program pauses, execute Y=5 and press (CONT). The program displays the calculator-mode variable assignments.

```
X= 3
Y= 5
```

Running a Program

The (RUN) key begins execution of the program in computer memory. When you press (RUN), the program pointer is set to the lowest numbered statement in the program; execution begins immediately at that statement.

The RUN command is used to begin program execution at a specified statement.

RUN [*statement number*]

If no optional statement number is specified, execution begins from the lowest numbered program statement. You cannot use a statement label as a RUN parameter. (Statement labels are discussed in section 8.)

Example:

RUN 100	Program execution begins at statement 100.

When a program is run using an optional statement number, statements having lower statement numbers than the RUN parameter are not executed. If the program was not previously initialized, any variables dimensioned in statements preceding the RUN parameter will not have the specified dimensioned memory allocated to them.

Both the (RUN) key and the RUN command scratch any current calculator-mode and program-mode variable assignments.

Pausing a Program

As discussed in section 5, the (PAUSE) key interrupts execution of a running program. When you press (PAUSE), the computer beeps and completes execution of the current line. Program execution halts before the next line and the power light stops blinking.

> Note: Since the computer completes execution of the current line before halting execution, system operation may not halt immediately when (PAUSE) is pressed. For instance, if the program is paused in the middle of a PRINT or DISP statement, the entire print or display list is output before program execution halts. However, if the program is paused during a LIST or PLIST operation, program execution stops after the line currently being listed has been printed or displayed.

A running program is also paused by pressing any alphanumeric or numeric keypad key, and by most of the display editing keys. The computer first halts the program, and then performs the key's indicated function (e.g., displaying a character, moving the cursor). The (SHIFT), (CTRL), and (CAPS LOCK) keys are inactive while a program is running.

If you press (RUN) while a program is running, the program pauses and the system displays RUN▮. Pressing (CONT) causes execution to resume from where the program was paused. If you'd prefer to rerun the program from the beginning, execute the displayed RUN command by pressing (END LINE) or press (RUN).

The following keys remain active during program execution, and perform their indicated function when pressed without halting the program:

(KEY LABEL)	(ROLL)
(SHIFT) (KEY LABEL)	(TR/NORM)
(K1) through (K14)	(A/G)
(CLEAR)	

The (INIT) and (RESET) keys also remain active during program execution. Pressing either of these keys halts program execution; the computer then performs the initialize or reset operation.

The PAUSE statement is used within programs to halt execution.

```
PAUSE
```

When the statement is executed, the program pauses until continued using the (CONT) key or by typing CONT (END LINE). The system does not beep.

Whenever program execution has been paused, you can perform any normal keyboard activities. You can use the computer, for instance, to perform arithmetic calculations, LIST or PLIST the program, or to execute other statements that can be executed from the keyboard.

When a program is running in *graph-all* mode, pressing (PAUSE), (RUN), any alphanumeric key, numeric keypad key, or display editing key, except (ROLL) and (CLEAR), halts the program and preserves the graphics display. The (KEY LABEL), (ROLL), and (CLEAR) keys are inactive when a program is running in *graph-all* mode. Pressing (A/G) returns the display to its power-on apportionment of CRT memory, erasing the current *graph-all* display, without halting the program.

When a program is paused in *graph-all* mode, all typewriter, numeric keypad, and display editing keys are inactive. Pressing (A/G) or (RESET) reapportions CRT memory to the power-on allocation, erasing the current contents of the graphics display. Pressing (RUN) reruns the program from the beginning.

Example: The following program computes the balance in a savings account after 1, 2, 3, etc. years, assuming 6.5% interest compounded daily. The program pauses after displaying the balance for each year; press (CONT) to compute the next year's balance.

```
10 REM ***Program computes bank balance, 6.5% interest compounded daily***
20 DISP "INITIAL BALANCE";
30 INPUT PV
50 YEAR=1
60 FV=PV*(1+6.5/36500)^365 !        Computes balance after one year elapsed.
70 DOLLARS=IP (FV*100)/100 !        Truncates balance to hundredths.
80 DISP "AFTER";YEAR;"YEAR(S), BALANCE IS";DOLLARS
90 PAUSE
100 YEAR=YEAR+1 !                   Increments year.
110 PV=FV !                         New balance for YEAR becomes old balance for
                                    YEAR+1.
120 GOTO 60 !                       Unconditional branch to statement 60.
130 END
```

```
INITIAL BALANCE?
1000.00
AFTER 1 YEAR(S), BALANCE IS 1067.15
AFTER 2 YEAR(S), BALANCE IS 1138.81
AFTER 3 YEAR(S), BALANCE IS 1215.28
AFTER 4 YEAR(S), BALANCE IS 1296.9
AFTER 5 YEAR(S), BALANCE IS 1383.99
```

Continuing a Paused Program

When a program has been paused, execution can be resumed by pressing the (CONT) (*continue*) key or by executing the CONT command.

CONT [*statement number*]

The optional parameter allows you to continue execution from a specified statement number. You cannot use a statement label as a CONT parameter.

Example:

CONT 100	Continues program execution from statement 100.

A paused program can be executed from the beginning by pressing (RUN), by executing the RUN command, or by executing CONT 1.

A program can be continued after almost every program halt as long as the program has not been deallocated. Editing a program deallocates it. Execution of an edited program can be resumed in two ways:

- Initialize the program. The initialization process allocates the entire program. Then, resume execution using the (CONT) key or the CONT command.

- Use the RUN command. Keep in mind that the RUN command scratches any variable assignments made before the program was paused, and that the RUN command does not allocate the entire program; memory is allocated to variables as they are encountered. Only changes made in program lines executed after execution resumes are known to the program.

The computer returns an error if you attempt to resume execution of a paused, edited program using the (CONT) key or the CONT command.

Computer Memory

The amount of computer memory is measured in *bytes*. A byte is a memory location composed of eight bits (binary digits). A *kilobyte* consists of 1,024 bytes; the abbreviation for "kilo" is "K".

The computer uses two types of memory:

- Random access memory (RAM or read/write memory) is used to store programs and data. Random access memory is temporary; you write to memory when you store a data element or a program line, and you read from memory when you access the information. Executing the SCRATCH command or turning off the computer erases the contents of random access memory; programs and data can be stored on a flexible disc to preserve them for future access.

  The HP-86 and HP-87XM have 80K bytes and 144K bytes of RAM, respectively. CRT memory utilizes 16K bytes; most of remaining RAM is available to the user for programs and data (approximately 4K bytes are used by the operating system). The amount of RAM available to the user can be increased by a maximum of 512K bytes by installing up to four memory modules into the ports on the back of the computer.

- Read only memory (ROM) is permanent memory, and is unaffected by the SCRATCH command or by turning off the computer. The computer has approximately 48K bytes of ROM memory.

  Optional ROM modules can be installed into the computer to expand its language and programming capabilities. A small amount of random access memory is used by some ROM modules.

Memory Requirements of Variables

The table below lists the amount of random access memory used to store the values assigned to the different types of variables.

Variable Type	Precision	Bytes of RAM
Simple numeric	REAL SHORT INTEGER	12 bytes. 8 bytes. 7 bytes.
Simple string	—	11 bytes + 1 byte per character (dimensioned length).
Numeric array	REAL SHORT INTEGER	11 bytes + 8 bytes per element. 11 bytes + 4 bytes per element. 11 bytes + 3 bytes per element.
String array	—	13 bytes per array; Per array element: 2 bytes + 1 byte per character (dimensioned length).

In addition, each variable name requires one byte per character. Statement labels require two bytes per label plus one byte per character in the label.

The storage requirements of variables on a mass storage medium (flexible disc) are discussed in section 22.

Determining Available Memory

As discussed previously, the amount of remaining random access memory is displayed at the end of a program listing or when you press (PLST).

The FRE function allows you to determine the amount of available (*free*) memory both within programs and from the keyboard.

```
FRE
```

If the program has just been typed in or loaded from mass storage, memory has not yet been allocated for program variables. To determine the amount of memory remaining after variable allocation, initialize the program (press (INIT)) before executing FRE.

Certain memory requirements of programs are not detected when the program is initialized. Memory required for string concatenation operations and for mass storage buffers, for instance, is allocated during program execution.

Notes

Section 8

Program Branches and Loops

Introduction

None of the BASIC statements discussed so far have any effect on the order in which program statements are executed; execution has been in sequential order from the lowest numbered statement to the highest numbered statement. There are a number of BASIC statements that provide for nonsequential program execution.

The computer's BASIC language provides the following techniques for implementing nonsequential execution:

- Program *branching*, where execution is transferred to and proceeds from a specified statement.
- *Looping* provides for executing a sequence of statements a specified number of times.
- *Subroutines* allow programs to branch to and execute a portion of the program, returning to the point at which branching occurred.
- The computer allows you to create *user-defined functions*. The program automatically branches to the function definition whenever the program is required to evaluate the function.
- *Interrupt programming* provides a directive to the program to branch whenever a certain specified condition is met.

This section will cover branching and looping. Subroutines and user-defined functions are discussed in section 11. Interrupt programming is covered in section 13.

Statement Labels

Branching is implemented in BASIC by executing a statement that directs the computer to transfer execution to another program statement. There are two ways to specify the statement to which branching occurs: the statement (line) number and the statement label.

You've already used statement numbers in entering and editing programs. The *statement label* is a name assigned to a statement that can be used in place of the line number to reference the statement.

The syntax of a statement label is:

Label name :

The label name can be any sequence of letters, digits, and the underscore character up to 31 characters long; the first character must be a letter. You cannot use BASIC keywords as statement labels; attempting to do so will generate a syntax error or semantic error.

The label name is placed immediately following the line number and before the body of the statement.

Example:

```
90 RESTORE TestScores ! Equivalent to RESTORE 320
 :
 :
320 TestScores: DATA 90,87,78,88,93
```

If two or more program lines have the same statement label, references to the statement label access the lowest numbered statement with that label.

Unconditional Branching: The GOTO Statement

The GOTO statement is used to unconditionally transfer program execution to a specified statement. When the computer executes a GOTO statement, the program pointer moves immediately to the specified statement. Branching is not dependent on any specified condition, hence the term unconditional.

GOTO *statement label* / *statement number*

If the specified statement is not an executable statement (for example, a REM or DATA statement), control is transferred to the next executable statement in the branch. GOTO statements are programmable only; they cannot be executed from the keyboard.

Example: The following program allows you to construct sentences by entering individual words, or groups of words less than 18 characters long, in response to an INPUT statement. To halt the program, press (PAUSE).

```
10 REM *****THIS PROGRAM CONSTRUCTS A SENTENCE*****
20 DIM SENTENCE$[200] !            Dimensions string longer than 18 characters.
30 SENTENCE$="" !                  Initializes SENTENCE$ to null string.
40 NEXTWORD: DISP "NEXT WORD";
50 INPUT WORD$
60 SENTENCE$=SENTENCE$&WORD$&" " ! Builds sentence.
70 DISP SENTENCE$
80 GOTO NEXTWORD
90 END
```

```
NEXT WORD?
HERE'S
HERE'S
NEXT WORD?
A
HERE'S A
NEXT WORD?
PROGRAM
HERE'S A PROGRAM
NEXT WORD?
THAT
HERE'S A PROGRAM THAT
NEXT WORD?
STRINGS
HERE'S A PROGRAM THAT STRINGS
NEXT WORD?
YOU
HERE'S A PROGRAM THAT STRINGS YOU
NEXT WORD?
ALONG!
HERE'S A PROGRAM THAT STRINGS YOU ALONG!
NEXT WORD?
```

The Computed GOTO Statement: ON...GOTO

The ON...GOTO statement transfers execution to one of several specified program statements, depending on the value of a numeric expression.

statement list

ON *numeric expression* GOTO *statement label* / *statement number* [, *statement label* / *statement number*...]

The numeric expression is evaluated and rounded to an integer. A value of 1 causes branching to the first statement in the statement list; a value of 2 causes branching to the second statement, and so on. A value less than 1 or greater than the number of statements in the list produces an error.

Examples:

```
20 ON Percent/10 GOTO 90,170,30
```

The expression Percent/10 is evaluated and rounded to the nearest integer. If the resulting value is 1, branching occurs to statement 90. If the resulting value equals 2, branching occurs to statement 170. A resulting value of 3 causes branching to statement 30. Any other values cause an error.

The following program computes the weekly wages for a firm's employees based on the three pay scales. Overtime is paid at the same rate as regular hours.

Pay Scale	Hourly Wage
1	5.50
2	6.50
3	7.50

```
10 PRINT "NAME","HOURS","PAYSCALE","WAGES" !
                                        Prints headings.
20 DIM NAME$[21] !                      Dimensions NAME$ to maximum of 21
                                        characters.
30 DISP "ENTER NAME, HOURS, PAYSCALE (1,2, OR 3)" !
                                        Prompts for input.
40 INPUT NAME$,HOURS,PAYSCALE
50 ON PAYSCALE GOTO 60,80,100 !         Computed GOTO.
60 WAGES=5.5*HOURS !                    Executed if PAYSCALE=1.
70 GOTO 110
80 WAGES=6.5*HOURS !                    Executed if PAYSCALE=2.
90 GOTO 110
100 WAGES=7.5*HOURS !                   Executed if PAYSCALE=3.
110 PRINT NAME$,HOURS,PAYSCALE,WAGES
120 PAUSE
130 GOTO 30
140 END
```

The program prints a table heading and then prompts for the employee's name, the hours worked, and the pay scale. When you've entered the information, the program computes and prints the employee's salary record and then pauses. Press (CONT) to receive the prompt for the next employee's data. Make sure you use quotes around the employee's name, since it contains a comma.

```
ENTER NAME, HOURS, PAYSCALE (1,2, OR 3)
?
"Frank,A.",40,2
ENTER NAME, HOURS, PAYSCALE (1,2, OR 3)
?
"Patrick,P.",55,3
ENTER NAME, HOURS, PAYSCALE (1,2, OR 3)
?
"Albritton,B.",20,1
```

The printer output should look like this:

NAME	HOURS	PAYSCALE	WAGES
Frank,A.	40	2	260
Patrick,P.	55	3	412.5
Albritton,B.	20	1	110

Conditional Branching: IF...THEN...ELSE Statements

There are times when you'll want the program to decide whether or not to perform a branching operation; i.e., to base branching on whether or not a certain condition is met. Conditional branching is provided by two forms of the IF . . . statement:

The IF . . . THEN statement has the form:

IF *logical expression* / *numeric expression*	THEN	*statement label* / *statement number* / *executable statement*

The IF . . . THEN . . . ELSE statement has the form:

IF *logical expression* / *numeric expression*	THEN	*statement label* / *statement number* / *executable statement*	ELSE	*statement label* / *statement number* / *executable statement*

The IF...THEN Statement

The IF...THEN statement makes a "decision" based upon the outcome of a numeric expression. If the expression is true (in the case of relational operations) or non-zero (in the case of any other numeric or logical expression), the THEN portion of the statement is executed. If the expression is false or zero, the THEN portion of the statement is ignored.

The IF... THEN statement provides for conditional branching when the THEN portion of the statement is:

- A statement label.
- A statement number.
- A GOTO statement (executable statement).

Examples:

`20 IF Dollars>Pennies THEN 200`	Program branches to statement 200 when variable Dollars> Pennies.
`30 IF Dimes>Pennies THEN BANK`	Program branches to statement labeled BANK when Dimes> Pennies.
`40 IF NICKELS<QUARTERS THEN GOTO 200`	Program branches to statement 200 when NICKELS<QUARTERS.

The following program is a revision of the payroll program on page 100. Statements 41 and 42 have been added to provide for overtime pay. Hours worked over the firm's regular 40-hour week are multiplied by the factor 1.5; the total hours for which an employee is paid is computed in statement 42. Since this equation would produce an incorrect result for HOURS less than 40, program execution skips statement 42 when necessary.

```
10 PRINT "NAME","HOURS","PAYSCALE","WAGES" ! Prints headings.
20 DIM NAME$[21] ! Dimensions NAME$ to maximum of 21 characters.
30 DISP "ENTER NAME, HOURS, PAYSCALE (1,2, OR 3)" ! Prompts for input.
40 INPUT NAME$,HOURS,PAYSCALE
41 IF HOURS<= 40 THEN 50 !              Conditional branch to statement 50.
42 HOURS=40+(HOURS-40)*1.5 !            Executed if HOURS > 40.
50 ON PAYSCALE GOTO 60,80,100 !         Computed GOTO.
60 WAGES=5.5*HOURS !                    Executed if PAYSCALE=1.
70 GOTO 110
80 WAGES=6.5*HOURS !                    Executed if PAYSCALE=2.
90 GOTO 110
100 WAGES=7.5*HOURS !                   Executed if PAYSCALE=3.
110 PRINT NAME$,HOURS,PAYSCALE,WAGES
120 PAUSE
130 GOTO 30
140 END
```

NAME	HOURS	PAYSCALE	WAGES
Frank,A.	40	2	260
Patrick,P.	62.5	3	468.75
Albritton,B.	20	1	110

The numeric expression after IF need not be relational. The following program computes the reciprocal of a value, 1/X. Since division by zero yields an error, an IF . . . THEN statement is used to check for X=0.

```
10 REM     *****COMPUTING A RECIPROCAL*****
20 DISP "ENTER X"
30 INPUT X
40 IF X THEN COMPUTE !                  If X#0, program branches to statement 70.
50 DISP "THE RECIPROCAL OF ZERO IS UNDEFINED" !       Executed if X=0.
60 GOTO 20
70 COMPUTE: DISP "1 /";X;"=";1/X !     Executed if X#0.
80 END
```

```
ENTER X
?
3
1 / 3 = .333333333333

ENTER X
?
0
THE RECIPROCAL OF ZERO IS UNDEFINED
ENTER X
?
9
1 / 9 = .111111111111
```

The IF . . . THEN statement can be used to conditionally execute a statement, rather than a branch, by placing an executable statement after THEN.

Example:

```
20 IF A=B THEN PRINT "A equals B"
```

All executable BASIC statements can follow THEN except FOR, NEXT, and IF. (FOR and NEXT will be explained shortly.) The following declaratory statements are not allowed after THEN.

COM	IMAGE
DATA	INTEGER
DEF FN	OPTION BASE
DIM	REAL
FN END	SHORT

Example: This program tests your recall of the multiplication tables for the integers 0 through 9. Press (PAUSE) to stop the drill.

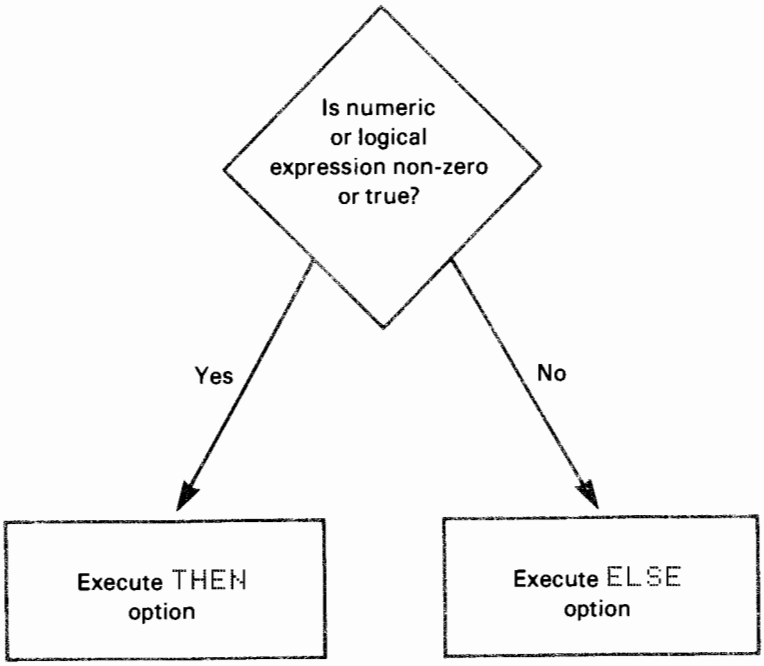
```
10 A=IP (RND *10)
20 B=IP (RND *10)
30 DISP "COMPUTE";A;"x";B
40 ANSWER: INPUT C
50 IF C=A*B THEN DISP "YOU ARE ABSOLUTELY RIGHT" !   Portions of statement
60 IF C#A*B THEN DISP "GUESS AGAIN" @ GOTO ANSWER  ! after THEN executed only
                                                     when expression is true.
70 GOTO 10
80 END
```

Statement 50 provides for displaying a congratulatory message if the answer is correct. Statement 60 uses a multistatement sequence after THEN to display GUESS AGAIN and to cause branching to the INPUT statement.

```
COMPUTE 4 x 2
?
8
YOU ARE ABSOLUTELY RIGHT
COMPUTE 9 x 9
?
64
GUESS AGAIN
?
49
GUESS AGAIN
?
81
YOU ARE ABSOLUTELY RIGHT
COMPUTE 4 x 6
?
24
YOU ARE ABSOLUTELY RIGHT
COMPUTE 9 x 0
?
```

The IF...THEN...ELSE Statement

The ELSE option increases the flexibility of the IF . . . THEN statement. In the previous examples of IF . . . THEN, the portion of the statement after THEN was ignored if the numeric expression evaluated to 0 (or false). The ELSE option allows you to specify alternate instructions.



The statement has the syntax:

IF	*logical expression* *numeric expression*	THEN	*statement label* *statement number* *executable statement*	ELSE	*statement label* *statement number* *executable statement*

Example: A quadratic equation has the form $Ax^2 + Bx + C = 0$. If A#0, the two roots can be found using the formulas:

```
ROOT1 = (-B+SQR(B^2-4*A*C))/(2*A)

ROOT2 = (-B-SQR(B^2-4*A*C))/(2*A)
```

The following program computes the roots of a quadratic equation given the values A, B, and C. The program tests for A=0 (in which case, the equation is not quadratic) and for B^2-4*A*C less than zero (complex roots).

```
10 REM *****QUADRATIC ROOTS*****
20 DISP "ENTER COEFFICIENTS A,B,C"
30 INPUT A,B,C
40 ! ************************************************************************
50 REM ***** Test whether a quadratic *****
60 IF A=0 THEN DISP "NOT A QUADRATIC; RE-ENTER COEFFICIENTS" ELSE 100
70 GOTO 30
80 ! ************************************************************************
90 REM ***** Test for complex roots *****
100 IF B^2-4*A*C>0 THEN 140 ELSE DISP "COMPLEX ROOTS; RE-ENTER COEFFICIENTS"
110 GOTO 30
120 ! ***********************************************************************
130 REM ***** Compute roots *****
140 ROOT1=(-B+SQR (B^2-4*A*C))/(2*A)
150 ROOT2=(-B-SQR (B^2-4*A*C))/(2*A)
160 DISP "THE ROOTS OF";A;"X^2 +";B;"X +";C;"= 0 ARE:"
170 DISP ROOT1,ROOT2
180 END
```

```
ENTER COEFFICIENTS A,B,C
?
0,1,3
NOT A QUADRATIC; RE-ENTER COEFFICIENTS
?
2,3,3
COMPLEX ROOTS; RE-ENTER COEFFICIENTS
?
2,6,2
THE ROOTS OF 2 X^2 + 6 X + 2 = 0 ARE:
-.38196601125          -2.61803398875
```

As with the THEN option, you can specify an executable statement after ELSE. The same stipulations hold for ELSE as for THEN—you can use any executable statement except FOR, NEXT, and IF, and you cannot use declaratory statements.

FOR...NEXT Loops

Repeatedly executing a series of statements is called looping. You've already seen several loops in programs formed by GOTO statements.

The combination of the FOR and NEXT statements provides an efficient way to create program loops. The two statements are used to enclose a sequence of statements to be executed a specified number of times.

FOR *loop counter* = *numeric expression* TO *numeric expression* [STEP *numeric expression*]

NEXT *loop counter*

The FOR statement defines the beginning of the loop and creates a loop counter that determines the number of times the loop is to be executed. The loop counter must be a simple numeric variable. The numeric expressions define the initial and final value of the loop counter and the increment (STEP) between successive values of the loop counter. If the increment value is not specified, the default value is 1.

Example:

```
10 REM *************FOR...NEXT LOOP*********************************
20 FOR COUNTER=2 TO 4 STEP .5 !                Begin FOR...NEXT loop.
30    PRINT COUNTER
40 NEXT COUNTER !                              End FOR...NEXT loop.
50 PRINT "OUT OF LOOP","COUNTER =";COUNTER
60 END
```

```
 2
 2.5
 3
 3.5
 4
OUT OF LOOP          COUNTER = 4.5
```

Note that the counter is incremented beyond the final value specified in the FOR statement when the program exits the loop. The reason for this will be clearer when you've read the following descriptions of the FOR and NEXT statements.

The FOR statement performs two operations:

- It sets the loop counter to the specified initial value.
- It automatically stores the allowable final value for the counter. The final value determines when to stop looping.

The NEXT statement performs three operations:

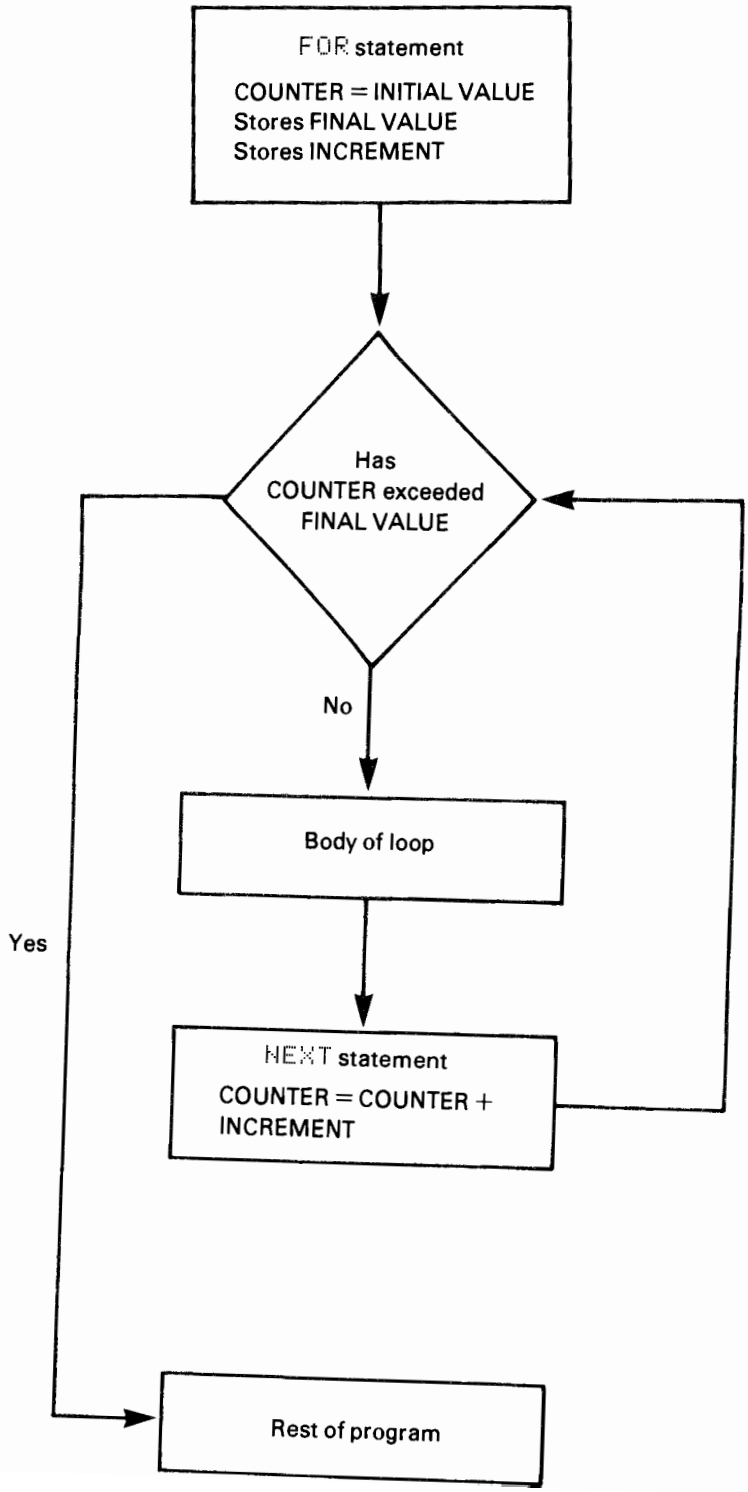
- It defines the end of the loop.
- It increments the loop counter according to the value of the STEP parameter.
- It tests to see if the counter has been incremented beyond the final value. If so, the program exits the loop, executing the statement following the NEXT statement. If the final value has not been exceeded, the program branches to the first executable statement after the FOR statement.

You can use variables and numeric expressions to specify the initial and final values and the increment value for the counter.

Examples:

```
120 FOR P=N/2 TO N*3
200 FOR COUNTER=IP(POINTS) TO CEIL(Y/2) STEP T
```

The following flowchart illustrates the relationship between the FOR and NEXT statements in forming a program loop.



Example: The following program computes the factorial of a positive integer. N! is defined as $N \times (N-1) \times (N-2) \ldots \times 1$. For instance $4! = 4 \times 3 \times 2 \times 1 = 24$.

```
10 ENTERDATA: DISP "ENTER POSITIVE INTEGER"
20 INPUT NUMBER
30 IF NUMBER<0 OR FP (NUMBER)#0 THEN ENTERDATA ! Test for negative or
                                                 non-integer number.
40 FACTORIAL=1 !                                 Initialize FACTORIAL.
50 IF NUMBER=0 THEN 90
60 FOR I=1 TO NUMBER !                           Begin loop.
70    FACTORIAL=FACTORIAL*I
80 NEXT I !                                      End loop.
90 DISP NUMBER;"FACTORIAL =";FACTORIAL
100 END
```

```
ENTER POSITIVE INTEGER
?
-5
ENTER POSITIVE INTEGER
?
5
 5 FACTORIAL = 120
```

The following chart shows how program variables change after 5 is entered.

Times through Loop	Value of I	Value of FACTORIAL
0	—	1
1	1	1×1
2	2	$1 \times 2 = 2$
3	3	$2 \times 3 = 6$
4	4	$6 \times 4 = 24$
5	5	$24 \times 5 = 120$
Out of Loop	6	120

The initial, final, and increment values specified in the FOR statement need not be integers, and may be positive or negative. However, if the initial value is larger than the final value, the increment value must be negative in order for the loop to be executed. Likewise, if the initial value is smaller than the final value, the increment value must be positive in order for the loop to be executed.

Example:

```
10 FOR I=3.2 TO -2.7 STEP -.6
20    DISP I;"  ";!            Semicolon suppresses carriage return/line feed.
30 NEXT I
40 DISP !                      Displays accumulated values of I on one line.
50 END
```

```
 3.2    2.6    2    1.4    .8    .2    -.4    -1    -1.6    -2.2
```

Note that the final displayed value of I is -2.2. The NEXT statement decrements I to -2.8 and compares that value to the final allowable value specified by the FOR statement, -2.7. Since -2.8 is outside the range of permissable values, execution proceeds to statement 40.

There are two rules governing branching into and out of FOR . . . NEXT loops:

- Execution of a FOR . . . NEXT loop should always begin with the FOR statement. Branching into the middle of a loop will produce an error if the NEXT statement is executed before the program executes the corresponding FOR statement.
- It is permissible to branch out of the loop. After an exit is made via a branching statement within the loop, the current value of the counter is retained for possible use later in the program. It is permissible to re-enter the loop, either at a statement within the loop, or at the FOR statement (thereby reinitializing the counter).

Nested Loops

When one loop is contained entirely within another, the inner loop is said to be nested. Programs may contain up to 255 levels of nesting (a loop within a loop within a loop...). A nested loop must be contained entirely within the loop in which it is nested.

Example: The program on the left demonstrates proper loop nesting. Since the "J" loop is contained within the "I" loop, the value assigned to J changes more rapidly than the value assigned to I.

The program on the right is an example of improper nesting. The "I" loop (statements 20 through 50) is executed three times. The "J" loop is cancelled each time NEXT I is executed. When NEXT J (statement 60) is encountered, the program is unable to locate an active matching FOR statement.

Correct Loop Nesting

```
10 PRINT " I  J" !     Prints headings.
20 FOR I=1 TO 3 !      Begins "I" loop.
30    FOR J=4 TO 6 !   Begins "J" loop.
40       PRINT I;J
50    NEXT J !         Ends "J" loop.
60 NEXT I !            Ends "I" loop.
70 END
```

```
I  J
1  4
1  5
1  6
2  4
2  5
2  6
3  4
3  5
3  6
```

Incorrect Loop Nesting

```
10 PRINT " I  J" !     Prints headings.
20 FOR I=1 TO 3 !      Begins "I" loop.
30    FOR J=4 TO 6 !   Begins "J" loop.
40       PRINT I;J
50    NEXT I !         Ends "I" loop.
60 NEXT J !
70 END
```

```
I  J
1  4
2  4
3  4
Error 47 on line 60 : NO MATCHING FOR
```

Notes

Section 9

Numeric and String Arrays

Two types of variables have been discussed so far—simple numeric and simple string variables. This section will cover the two additional types of variables available on the computer—numeric arrays and string arrays.

Array Concepts

An *array variable* (or simply, an *array*) is a collection of data items of the same type collected under one name. Subscripts enclosed in parentheses after the array name reference individual items in the collection.

The computer allows one- and two-dimensional arrays. A *one-dimensional array* can be thought of as a list of items consisting of several rows but only one column; items are referenced by one integer subscript. A *two-dimensional array* (often called a matrix) is like a table of items. The table has multiple rows and columns, and elements in the table are accessed by two integer subscripts separated by commas.

The number of items, or *elements,* in an array is determined by its lower and upper bounds. The *lower bound* of an array is the lowest value subscript; the *upper bound* of an array is the highest value subscript. For example, the following group of ten numbers can be considered as two five-item lists, as five two-item lists, or as one ten-item table.

1	2
3	4
5	6
7	8
9	10

The following assignments treat the data as two five-item lists (i.e., two one-dimensional arrays of five elements each).

```
ODDNUMBERS(0)=1          EVENNUMBERS(0)=2
ODDNUMBERS(1)=3          EVENNUMBERS(1)=4
ODDNUMBERS(2)=5          EVENNUMBERS(2)=6
ODDNUMBERS(3)=7          EVENNUMBERS(3)=8
ODDNUMBERS(4)=9          EVENNUMBERS(4)=10
```

The lower bound of each array is 0; the upper bound is 4. Non-integer subscripts are rounded to the nearest integer. Negative subscripts are not allowed.

Now, we'll combine the two lists into one two-dimensional array. The two subscripts used to reference the items are separated by commas; the first subscript designates the row, the second subscript designates the column.

```
NUMBERS(0,0)=1          NUMBERS(0,1)=2
NUMBERS(1,0)=3          NUMBERS(1,1)=4
NUMBERS(2,0)=5          NUMBERS(2,1)=6
NUMBERS(3,0)=7          NUMBERS(3,1)=8
NUMBERS(4,0)=9          NUMBERS(4,1)=10
```

The lower bound of the NUMBERS array is 0. Note that there are two upper bounds, 4 and 1, for the two subscripts.

The above examples used numeric arrays. The computer also allows string arrays, in which each element is assigned a character or sequence of characters.

Naming Array Variables

The rules for naming simple numeric and string variables also apply to numeric and string arrays. A program can use the same name for a simple and array variable. The simple variable is referenced using the name without subscripts. An array element is referenced using one or two subscripts in parentheses. Certain BASIC statements access an entire array when parentheses are included after the array variable name.

Examples:

VARIABLE	Simple numeric variable.
VARIABLE(2,4)	Element of a numeric array.
VARIABLE()	An entire numeric array.

Specifying Lower Bounds of Arrays

The subscript numbering for an array can begin with 0 or 1. The computer assumes that all array subscripts begin at 0 unless you specify otherwise using the OPTION BASE statement, which has the syntax:

```
OPTION BASE lower bound
```

The lower bound must be 0 or 1. Since the computer assumes OPTION BASE 0, that statement has no effect; however, it is useful for documentation purposes.

An OPTION BASE statement can be included only once in a program. Once an option base has been declared (or assumed), that option base is used throughout the program. The OPTION BASE declaration in a program must appear before any array variables are dimensioned or referenced.

You cannot execute an OPTION BASE statement from the keyboard.

Dimensioning Arrays

Dimensioning an array establishes the array's upper bounds and reserves computer memory for the array elements. In addition, the computer implements a system for referencing the individual elements by their subscripts.

You need not dimension an array if its upper bounds are less than or equal to 10. Any array not explicitly dimensioned is assumed to have upper bound(s) of 10 with the following number of elements:

	One-Dimensional	Two-Dimensional
OPTION BASE 0	11	121 (11 × 11)
OPTION BASE 1	10	100 (10 × 10)

Dimensioning arrays with fewer elements will conserve memory by allocating space for fewer numbers of elements.

Numeric Arrays

A numeric array is a collection of subscripted numeric variables grouped under one variable name. All elements of a numeric array have the same precision.

The maximum upper bound permitted for a numeric array is 65530.

Five declarative statements are available for dimensioning numeric arrays:

- DIM—Dimensions REAL precision numeric arrays.
- REAL—Like DIM, dimensions REAL precision numeric arrays. The REAL statement is used in place of the DIM statement for documentation purposes.
- SHORT—Dimensions SHORT precision numeric arrays.
- INTEGER—Dimensions INTEGER precision numeric arrays.
- COM—Dimensions arrays and reserves them "in common" for chained programs.

The REAL, SHORT, and INTEGER statements were introduced in section 2 as the means for specifying precision of simple numeric variables. Their additional function for dimensioning arrays will require some further explanation.

The DIM statement was first introduced in section 4 as a necessary step for allocating memory to simple string variables longer than 18 characters. It too has a second, array-dimensioning function.

The COM statement, which is used in programs that perform chaining operations, is discussed in section 21.

The statement explicitly dimensioning an array variable must be executed before any elements of the array are referenced. Regardless of the statement used to dimension an array, an array variable can be dimensioned only once in a program. An attempt to execute a second dimensioning statement for an array variable generates Error 35 : DIM EXIST VRBL.

Note: Although an array variable can be dimensioned only once by a running program, more than one dimension statement for an array variable may be included in a program.

Consider the following case:

```
10 INPUT TYPE$
20 IF TYPE$="INTEGER" THEN 50
30 REAL ARRAY(20)
40 GOTO 60
50 INTEGER ARRAY(20)
 :
```

Statements 30 and 50 both dimension ARRAY; however, no run-time error occurs if only one of the declarations is executed. The program can be run repeatedly as long as the same dimensioning statement is executed. However, if the program is initialized or renumbered, or if the second dimensioning statement is executed when the program is run again, the computer returns Error 35 : DIM EXIST VRBL. Program variables must be deallocated before the program can be rerun using the other declaration. To deallocate the program, re-enter any program line or enter an unused line number.

The DIM Statement

The DIM statement has three uses:

- To allocate memory for simple string variables longer than 18 characters (discussed in section 4).
- To dimension full precision numeric arrays.
- To dimension string array variables (to be discussed later in this section).

Numeric arrays, simple string variables, and string array variables can be dimensioned within the same DIM statement.

The statement has the syntax:

DIM *item*[, *item*...]

When the item to be dimensioned is a numeric array, the item has the form:

numeric variable name (*upper bound*[, *upper bound*])

Examples:

```
10 OPTION BASE 0
20 DIM TEMP(5,7), STRING$[56] ! TEMP is a 6 by 8 numeric array; maximum
                                length of STRING$ is 56 characters.
 :
```

```
10 OPTION BASE 1
20 DIM MASS(25), NAME$[20] !     MASS is a 25-element numeric array; maximum
                                 length of NAME$ is 20 characters.
 :
```

The maximum length of a string is not affected by the OPTION BASE declaration.

Precision Declaration Statements

The three precision declaration statements are:

INTEGER *item* [, *item*...]

SHORT *item* [, *item*...]

REAL *item* [, *item*...]

All numeric variables, both simple and array, are assumed to be full precision (type REAL) unless they appear in an INTEGER or SHORT type declaration statement.

When the item is a simple numeric variable, that variable is assigned the specified precision. When the item is an array name, the array is assigned the specified precision and is dimensioned according to the specified upper bounds.

The form of a numeric array item is identical to the form used with the DIM statement:

numeric variable name (*upper bound* [, *upper bound*])

Example:

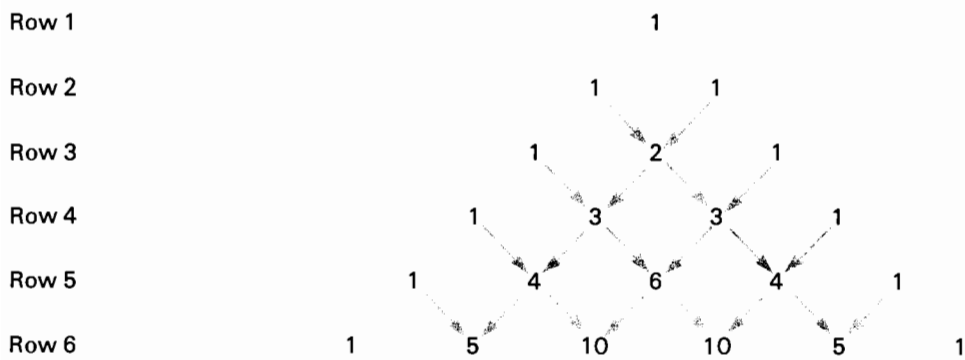
```
10 OPTION BASE 1
20 INTEGER DAY,POINTER(40) !        Simple variable DAY and 40-element array
                                    POINTER are INTEGER precision.
30 SHORT CHARGE(10,15),DELTA,PSI !  Simple variables DELTA, PSI and 150-element
                                    array CHARGE are SHORT precision.
40 REAL LIGHT,ENERGY(20) !          Simple variable LIGHT and 20-element array
                                    ENERGY are REAL precision.
 :
```

Example: The following program uses a one-dimensional array to compute the values of the squares of consecutive integers 1 through 8.

```
10 OPTION BASE 1
20 DIM SQUARES(8)
30 FOR INDEX=1 TO 8
40      SQUARES(INDEX)=INDEX*INDEX
50      PRINT INDEX;"times";INDEX;"=";SQUARES(INDEX)
60 NEXT INDEX
70 END
```

```
 1 times 1 = 1
 2 times 2 = 4
 3 times 3 = 9
 4 times 4 = 16
 5 times 5 = 25
 6 times 6 = 36
 7 times 7 = 49
 8 times 8 = 64
```

This next program uses a two-dimensional array to compute the values for the Nth row of Pascal's triangle. The number of entries in row N equals N; for each row, the first and last entries are 1; each of the other entries is the sum of the two numbers directly above.




```
10 OPTION BASE 1
20 DIM Triangle(20,20)
30 DISP "ENTER ROW YOU WISH COMPUTED ( 1 THROUGH 20 )"
40 INPUT RowNumber
50 Triangle(1,1)=1 ! top of triangle
60 FOR ROW=2 TO RowNumber
70   Triangle(ROW,1),Triangle(ROW,ROW)=1 ! sides of triangle = 1
80   FOR COLUMN=2 TO ROW-1
85      REM ***** compute "inside" columns of current ROW *****
90      Triangle(ROW,COLUMN)=Triangle(ROW-1,COLUMN-1)+Triangle(ROW-1,COLUMN)
100 NEXT COLUMN
110 NEXT ROW
120 FOR COLUMN=1 TO RowNumber
130 DISP Triangle(RowNumber,COLUMN);
140 NEXT COLUMN
150 DISP
160 END
```

```
ENTER ROW YOU WISH COMPUTED ( 1 THROUGH 20 )
?
7
 1  6  15  20  15  6  1
```

String Arrays

A *string array* is a collection of character strings collected under the same string variable name and having the same maximum length. The computer allows both one- and two-dimensional string arrays.

Dimensioning String Arrays

The DIM statement is used to set the upper bounds of the string array and to specify the maximum number of characters in each element.

DIM *item* [, *item...*]

When the item to be dimensioned is a string array, the item has the form:

string variable name (*upper bound* [, *upper bound*]) [*length per element*]

The upper bound(s) and length per element cannot exceed 65530. Refer to the table of Memory Requirements of Variables on page 326 to calculate the memory required to dimension a string array. The lower bound of a string array is determined by the option base of the program. The option base has no effect on the maximum string length.

String arrays, numeric arrays, and simple string variables can be dimensioned in the same DIM statement.

Example:

```
10 OPTION BASE 0
20 REM ***NAMES$ has 11 elements, each with maximum length of 25 characters.
30 REM ***GRADES has 66 REAL precision numeric elements.
40 DIM NAMES$(10)[25], GRADES(10,5)
:
```

If a string array is not explicitly dimensioned, it is implicitly dimensioned with upper bound(s) equal to 10 and maximum string length equal to 18.

The COM statement is used to dimension string arrays which are to be preserved in common between chained programs.

String Expressions and Operations

All the operations and functions provided for manipulating simple string variables can also be used with elements of string arrays.

Operations	Examples

Assignment	`STRING$(1)="eclipse"`
	`STRING$(2)="lunar"`
	`STRING$(3)="75"`

Concatenation	`EVENT$=STRING$(2) & " " & STRING$(1)`
	`DISP EVENT$`
	`lunar eclipse`

Substring	`MOUTH$=STRING$(1)[3,5]`
	`DISP MOUTH$`
	`lip`

Modification	`STRING$(2)[1,3]="sol"`
	`DISP STRING$(2)`
	`solar`

Comparison	`STRING$(2) < STRING$(1)` ` 0`

Functions	Examples
LEN	`LEN(STRING$(1))` ` 7`

POS	`PLACE= POS(STRING$(1),"p")`
	`DISP PLACE`
	` 5`

VAL	`DISP VAL(STRING$(3))` ` 75`

VAL$	`STRING$(4)=VAL$(12345)`
	`DISP STRING$(3)&STRING$(4)`
	`7512345`

CHR$	`STRING$(5)=CHR$(40)`
	`DISP STRING$(5)`
	`(`

NUM	`DECVAL=NUM(STRING$(3))`
	`DISP DECVAL`
	` 55`

UPC$	`SUN$=UPC$(STRING$(2))`
	`SUN$`
	`SOLAR`

Example: The following program sorts a list of words alphabetically. Since string comparisons are based on the decimal codes assigned to each letter, all lowercase letters are converted to uppercase letters before sorting begins.

```
10 OPTION BASE 1
20 DIM WORD$(20)[30] !                Dimensions 20-element string array.
30 FOR I=1 TO 16 !                    This loop reads and prints DATA elements.
40      READ WORD$(I)
50      WORD$(I)=UPC$ (WORD$(I)) !    Converts word to all uppercase letters.
60      PRINT WORD$(I);" ";
70 NEXT I !                           Ends loop.
80 PRINT
90 FOR J=2 TO 16 !                    Begin sort.
100      TEMP$=WORD$(J)
110      FOR I=J-1 TO 1 STEP -1
120           IF TEMP$ >= WORD$(I) THEN GOTO INSERT
130           WORD$(I+1)=WORD$(I) ! Move element down one position.
140      NEXT I
150      INSERT: WORD$(I+1)=TEMP$ !   Insert element at position I+1.
160 NEXT J
170 FOR I=1 TO 16 !                   Print sorted list.
180 PRINT WORD$(I);" ";
190 NEXT I
200 PRINT
210 DATA HOW,CAN,you,BE,IN,TWO,PLACES,AT,once,WHEN,YOU,ARE,not,ANYWHERE,AT,ALL
220 END
```

```
HOW CAN YOU BE IN TWO PLACES AT ONCE WHEN YOU ARE NOT ANYWHERE AT ALL
ALL ANYWHERE ARE AT AT BE CAN HOW IN NOT ONCE PLACES TWO WHEN YOU YOU
```

Initializing Array Variables

Attempting to access a dimensioned array element which has not yet been assigned a value generates a warning message (with DEFAULT ON) or an error message (with DEFAULT OFF). Thus, if your program is designed to access unassigned array elements, you should initialize them. The easiest way to do this is to include an initialization statement in a FOR . . . NEXT loop.

Example: In the following program segment, elements of a numeric array are initialized to 0; elements of a string array are initialized to the null string.

```
10 DIM NUMBERS(10), STRINGS$(10)[30]
20 FOR I=0 TO 10
30   NUMBERS(I)=0
40   STRINGS$(I)=""
50 NEXT I
 :
 :
 :
```

Section 10

Printer and Display Formatting

Introduction

The discussion of the DISP and PRINT statements on pages 72 through 74 showed how to use semicolons, commas, and quoted text in those statements to format program output. The computer provides a great deal of additional control over formats with PRINT USING, DISP USING and IMAGE statements, and the TAB function.

Example: The following programs illustrate how formatted output allows you to round numbers to a specified number of decimal places and to position numbers and text for greater readability.

Formatting with commas and semicolons:

```
10 OPTION BASE 1
20 ASUM,BSUM=0
30 FOR I=1 TO 4
40    READ A(I),B(I)
50    DISP A(I),B(I)
60    ASUM=ASUM+A(I) @ BSUM=BSUM+B(I)
70 NEXT I
80 DISP "TOTAL=";ASUM,"TOTAL=";BSUM
90 DATA 5.0852,7,.3737,8.6,4.332,9,679.4646,.8
100 END
```

```
 5.0852                7
 .3737                 8.6
 4.332                 9
 679.4646              .8
TOTAL= 689.2555       TOTAL= 25.4
```

Formatting with DISP USING and IMAGE statements:

```
10 OPTION BASE 1
20 ASUM,BSUM=0
30 FOR I=1 TO 4
40    READ A(I),B(I)
50    DISP USING 100 ; A(I),B(I)
60    ASUM=ASUM+A(I) @ BSUM=BSUM+B(I)
70 NEXT I
80 DISP USING 110 ; "TOTAL=",ASUM,"TOTAL=",BSUM
90 DATA 5.0852,7,.3737,8.6,4.332,9,679.4646,.8
100 IMAGE 10X,4D.DD,10X,4D.DD
110 IMAGE 3X,7A,4D.DD,3X,7A,4D.DD
120 END
```

```
              5.09              7.00
               .37              8.60
              4.33              9.00
            679.46               .80
   TOTAL=   689.26   TOTAL=    25.40
```

This section covers formatting your printer and display output using these BASIC statements. Also covered is the computer's inverse video (black letters on white screen) capabilities, printer control codes, and programming the interface to customize the end-of-line sequence. The examples in this section assume you are using an 80-column printer.

Printer Line Length

The PRINTER IS statement allows you to specify an optional line length from 1 to 220 characters.

```
PRINTER IS device selector [ , line length]
```

Usually, the line length specified will correspond to the column width of your system printer. If you specify a line length less than the column width, the computer will direct the printer to start a new line when the specified number of characters has been printed. Longer line lengths provide for printing the condensed characters available on some printers.

The default line length is 80 characters.

Formatted Print and Display Statements

A format string is a series of characters used to specify the desired format for output. When a format string is used in a PRINT USING or DISP USING statement, the list of items in the statement is printed or displayed according to the format specified by the format string.

```
                "format string"
DISP USING  statement number  [; item[; item...]]
                statement label
```

```
                "format string"
PRINT USING  statement number  [; item[; item...]]
                statement label
```

The items are the variables, constants, and quoted text to be printed. You can use commas or semicolons to separate items. The choice of commas or semicolons has no effect on the spacing of the output, since the format is totally controlled by the format string.

Notice that you can substitute a statement number or statement label for the format string. The statement number or label must refer to an IMAGE statement within the program. The IMAGE statement contains the format string used to format the items in the DISP USING or PRINT USING statements.

```
IMAGE format string
```

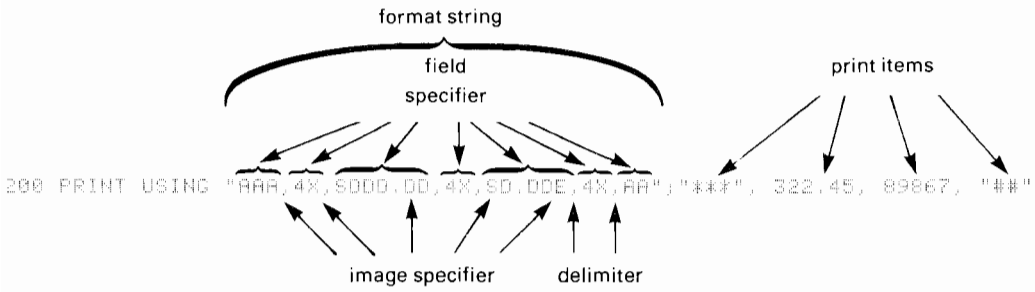
The Format String

To understand how format strings work, you'll need to know the meaning of the following terms:

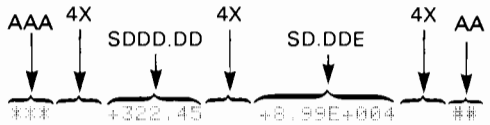
- Field specifier.
- Image specifier.
- Delimiter.

A format string consists of one or more *field specifiers*. Each field specifier defines the format, or *field*, of a particular item in the print or display list. A field specifier consists of one or more characters, called *image specifiers*, that specify either the format of the item within its allotted field or the spacing between items. Field specifiers in the same format string are separated from one another by *delimiters*.

Example: Examine the following `PRINT USING` statement.



Executing the statement produces the following output:



Each item in the print or display list must correspond from left to right to the appropriate field specifier in the format string.

If you place the format string in a `PRINT USING` or `DISP USING` statement, the format string must be enclosed within quotes. When the format string is part of an `IMAGE` statement, it must be unquoted.

When the `PRINT USING` and `DISP USING` statements reference an `IMAGE` statement, they are programmable only, and cannot be executed from the keyboard.

Delimiters

Delimiters are used to separate field specifiers in the same format string from one another. The two delimiters are comma and slash.

,	A comma is used to separate two field specifiers.
/	A slash is used to separate two field specifiers and to cause the output of a carriage return/line feed (CR-LF).

Example:

```
10 PRINT USING 20
20 IMAGE "COST"/"DISCOUNT"
30 END
```

```
COST
DISCOUNT
```

You can specifiy multiple CR-LF sequences by using additional slashes. Replacing statement 20 above with:

```
20 IMAGE "COST"///"DISCOUNT"
```

or

```
20 IMAGE "COST"3/"DISCOUNT"
```

causes the output of two blank lines between COST and DISCOUNT.

The / character can also be used as a field specifier, separated from other field specifiers by commas.

Example: The following statement performs the same function as both forms of statement 20, above.

```
20 IMAGE "COST",3/,"DISCOUNT"
```

Note: A final comma or semicolon placed after the last item in a PRINT USING or DISP USING statement does not suppress the carriage return/line feed. Unless your system includes an HP-87 I/O ROM, the computer is not capable of suppressing the end-of-line sequence (normally CR-LF) at the end of a PRINT USING or DISP USING list.

Image Specifiers

The following table lists the *image specifiers* available for constructing field specifiers. Each image specifier is then discussed in greater detail. A "Yes" entry in the replication column indicates that you can use a numeric factor to specify repeated use of the same image specifier (for instance, DDD.DD or 3D.2D) within a field specifier.

Type of Output	Character	Purpose	Replication
Blank space	X	Specifies a blank space between items.	Yes
Radix symbol	.	Specifies a decimal point in that position.	No
	R	Specifies a comma radix indicator in that position.	No
Digit	D	Digit position to left or right of radix symbol; leading blanks.	Yes
	Z	Digit position to left of radix symbol; leading zeros.	Yes
	*	Digit position to left of radix symbol; leading asterisks (*).	Yes
Digit separator	C	Specifies a comma as a separator in the specified position.	No
	P	Specifies a period as a separator in the specified position.	No
Exponent	E	Numeric field is output in exponential format; exponent consists of three digits plus sign.	No
Sign	S	Specifies sign, "+" or "—".	No
	M	Specifies sign, blank or "—".	No
String	" "	Specifies literal text.	No
	A	Specifies character position; text is left-justified.	Yes
Numeric or string	K	Specifies compact format with no leading or trailing blanks.	No
Miscellaneous	( )	Used to indicate replication of field specifier.	Yes
	/	Specifies a carriage return/line feed; can also be used as a delimiter.	Yes

Numeric Image Specifiers

The numeric image specifiers include symbols for indicating:

- Digit position.
- Radix.
- Digit separators.
- Exponential format.
- Sign.

Digit Symbols

D	Specifies a digit position. If the number of D's to the left of the radix specify a field larger than the numeric item, then the item is right-justified in the field and leading zeros are filled with blanks. If the fractional part of the numeric item requires more places than the number of D's to the right of the radix, then the item is rounded to fit the specified field. D is the only image specifier you can use to specify digits to the right of the radix.
Z	Like D, specifies a digit position, except that leading zeros are filled with 0 characters. You cannot use a Z to the right of a radix symbol.
*	Like D, specifies a digit position, except that leading zeros are filled with asterisks. You cannot use a * to the right of a radix symbol.

Digit specifiers to the left of the radix cannot be mixed, except that Z can always appear in the position immediately to the left of the radix.

Example:

```
10 PRINT USING 40 ; 29.91,5.417,7.2,1.6907
20 PRINT USING 50 ; 29.91,5.417,7.2,1.6907
30 PRINT USING 60 ; 29.91,5.417,7.2,1.6907
40 IMAGE 2D.2D, 2X, 3D.3D, 2X, D.3D, 2X, D.2D
50 IMAGE 2Z.2D, 2X, 3Z.3D, 2X, Z.3D, 2X, Z.2D
60 IMAGE 2*.2D, 2X, 3*.3D, 2X, *.3D, 2X, *.2D
70 END
```

```
29.91    5.417  7.200  1.69
29.91  005.417  7.200  1.69
29.91  **5.417  7.200  1.69
```

Radix Image Specifiers

A radix indicator is the symbol that separates the integer part of a number from the fractional part. In the United States, this is customarily the decimal point (as in 3.14). In Europe, this is frequently the comma (as in 3,14).

A numeric field specifier can contain at most one radix image specifier. Only the D image specifier can be used to specify a digit to the right of the radix indicator.

.	Specifies a decimal point in that position.
R	Specifies a comma radix indicator in that position.

Example:

```
10 PRINT USING 20 ; 473.1,25.392,76.5
20 IMAGE DDD.DD,2X,***.3D,2X,ZZZRDD
30 END
```

```
473.10  *25.392  076,50
```

Sign Image Specifiers

Two sign symbols control the output of the sign characters + and −.

S	Specifies output of a sign: + if the number is positive, − if the number is negative.
M	Specifies output of a sign: − if the number is negative, a blank if it is positive.

Example:

```
10 PRINT USING 20 ; -47.2,-.51,33.5,38.12
20 IMAGE MDD.2D, 2X, SZZ.2D, 2X, SZZ.2D, 2X, MZZ.2D
30 END
```

```
-47.20  -00.51  +33.50   38.12
```

If you do not specify a sign position, a – sign requires one digit position.

Digit Separator Image Specifiers

Digit separators are used to break large numbers into groups of digits (generally three digits per group) for greater readability. In the United States, a comma is customarily used; in Europe, the period is commonly used.

C	Specifies a comma as a separator in the specified position.
P	Specifies a period as a separator in the specified position.

A separator can be used only between two digits and will be output only if digits on both sides of the separator are output. When leading zeros are generated by the Z symbol, they are considered digits and will contain separators. When the * symbol is used to format leading asterisks, the separator is output only if there are digits on both sides of the separator; otherwise, the separator is replaced with an asterisk.

Example:

```
10 PRINT USING 20 ; 22590.49,5390.11,7,410,89
20 IMAGE DDCDDD.2D, 4X, DPDDDRD, 4X, 3ZC3Z, 4X, 3DC3D, 4X, 3*C3*
30 END

22,590.49    5.390,1    000,007        410    *****89
```

The Exponent Symbol: E

The E symbol specifies that the item is to be output in mantissa/exponent notation. The E symbol in the field specifier causes the output of an E exponent symbol, the sign of the exponent (+ or –), and a three-digit exponent.

The mantissa is output according to the specified format, rather than in scientific notation as a number ranging from 1 to 10.

Example:

```
10 PRINT USING 20 ; 157.24,5.762
20 IMAGE D.DDDE, 4X, 3Z.2DE
30 END

1.572E+002    576.20E-002
```

Numeric Field Overflow

If a numeric item requires more digits to the left of the decimal point or radix than the field specifier provides, an overflow condition occurs. With DEFAULT ON, a warning message is displayed and program execution continues. With DEFAULT OFF, the computer returns an error.

A minus sign not explicitly specified requires a digit position. If the position is not available, an overflow situation occurs.

Specifying Strings

Text can be specified in two ways:

" "	Text enclosed within quotation marks is printed or displayed exactly as it is quoted. You can specify quoted text as a PRINT USING/DISP USING list item or you can include the text in a format string in an IMAGE statement. If the quoted text is a PRINT USING or DISP USING item, you must specify a field for the item using the A or K image specifiers.
A	Specifies a single character in a string. A number preceding A or the number of A's placed between delimiters determines the size field reserved for the string. When the specified field is longer than the string item, the item is left-justified and the rest of the field is filled with blanks. If the string item is longer than the number of characters specified, the string is truncated.

Example:

```
10 CRED$="CREDITS" @ DEB$="DEBITS"
20 PRINT USING 30 ; "*****",CRED$,DEB$,"      *****"
30 IMAGE 10A, 7A, "-----",6A,10A
40 END

*****      CREDITS-----DEBITS      *****
```

You cannot use quotation marks to specify literal text within an image format string that is itself contained within quotation marks in a PRINT USING or DISP USING statement.

Example: Attempting to enter the following statement will generate an error. The statement is not understood after the second quotation mark.

```
20 PRINT USING "10A, 7A, "-----",6A,10A";"*****",CRED$,DEB$,"*****"
```

Compact Field Specifier

A single symbol K is used as a field specifier to define an entire field for either a number or string of characters. If the PRINT USING/DISP USING item is a string, the entire string is output in a field the same size as the string length. If the item is numeric, it is output in standard number format, but with no leading or trailing blanks.

Example:

```
10 PRINT USING COMPACT ; "ABC",415,"DEF",.01
20 COMPACT: IMAGE K,2X,K,K,K
30 END
```

```
ABC  415DEF.01
```

Replication

Numeric factors can be used to indicate replication of field specifiers and many image specifiers.

Replication of Image Specifiers

Many of the symbols used as image specifiers can have numbers preceding them indicating replication of the symbol. Some of the examples used in this section have made use of this feature.

Example: Each row shows equivalent field specifiers.

```
AAAAAAAAAA     10A
ZZCZZZ.DDDD    2ZCZZZ.DDDD    2ZC3Z.DDDD    2ZC3Z.4D    ZZC3Z.4D
**.DDE         2*.2DE         **.2DE        2*.DDE
```

In the table on page 125, a "Yes" entry in the replication column indicates the image specifier can be replicated using a numeric factor. The factor must be in the range 1 through 8000.

An entire field specifier or group of field specifiers can be replicated by enclosing the repeated field specifier(s) in parentheses and placing an integer in the range 1 through 8000 before the parentheses.

Example: Each row shows equivalent format strings.

```
DD.D,DD.D,DD.D,DD.D        4(DD.D)                3(DD.D), DD.D
ZZ.D,4A,2X,4A,2X           ZZ.D, 2(4A,2X)
2Z.D,4X,2A,2A,4X,2A,2A     2Z.D, 2(4X,2(2A))
```

Up to 128 levels of nested parentheses can be used for replication.

Reusing the IMAGE Format String

A format string is reused from the beginning if it is exhausted before the entire `DISP USING` or `PRINT USING` list is output.

Example:

```
10 PRINT USING 20 ; 25.71,99.9,2.38
20 IMAGE ZZ.DD,2X
30 END
```

```
25.71  99.90  02.38
```

The TAB Function

The TAB function can be included as an item in a PRINT or DISP statement to specify the column position at which an item will be printed or displayed. The TAB function cannot be used with PRINT USING, DISP USING, or IMAGE statements.

TAB (*character position*)

The character position can be any number within the computer's range. If you specify a character position larger than 80 characters or the line length specified by the PRINTER IS/CRT IS statements, the number will be reduced MOD *line length*. If you specify a negative or 0 character position, the computer returns a warning and interprets the expression as TAB(1).

When you use the TAB function, the items in the PRINT or DISP list must be separated by semicolons, since commas output items in the next 21-column field.

Example:

```
10 FOR I=1 TO 6 !                    Begins loop.
20    ITEM$="ITEM #"&VAL$ (I)
30    PRINT TAB (10*I);ITEM$;!       Semicolon suppresses CR-LF.
40 NEXT I !                          Ends loop.
50 PRINT !                           Empties print buffer.
60 END

          ITEM #1    ITEM #2    ITEM #3    ITEM #4    ITEM #5    ITEM #6
```

Clearing the Alpha Display

The CLEAR statement clears 16 lines (24 lines for PAGESIZE 24) of CRT memory starting with the line below the current cursor position.

CLEAR

The statement performs the same operation as the (CLEAR) key. The 16 (or 24) lines are erased, and the empty lines are rolled upwards into the display window. The cursor moves to home position. (See (CLEAR), page 22.)

Inverse Video Characters

Characters with decimal codes 128 through 255 are the inverse video complements of characters with decimal codes 0 through 127. Inverse video characters can be displayed using the CHR$ function.

Example: The following program converts a string up to 80 characters in length into its inverse video equivalent.

```
10 DIM A$[80],B$[80] !                      Dimensions maximum string lengths.
20 DISP "ENTER A STRING OF CHARACTERS, CALLED A$, UP TO ONE LINE LONG"
30 INPUT A$
40 DISP USING FORMAT ; "THIS IS A$:",A$ !   Displays string just entered.
50   FORMAT: IMAGE /,11A, /, 80A
60 LENGTH=LEN (A$) !                        Computes length of string A$.
70 FOR C=1 TO LENGTH
80   B$[C,C]=CHR$ (NUM (A$[C,C])+128) !     Converts character to inverse
                                            equivalent.
90 NEXT C
100 DISP USING 50 ; "THIS IS B$:",B$ !      Displays inverse video string.
110 END
```

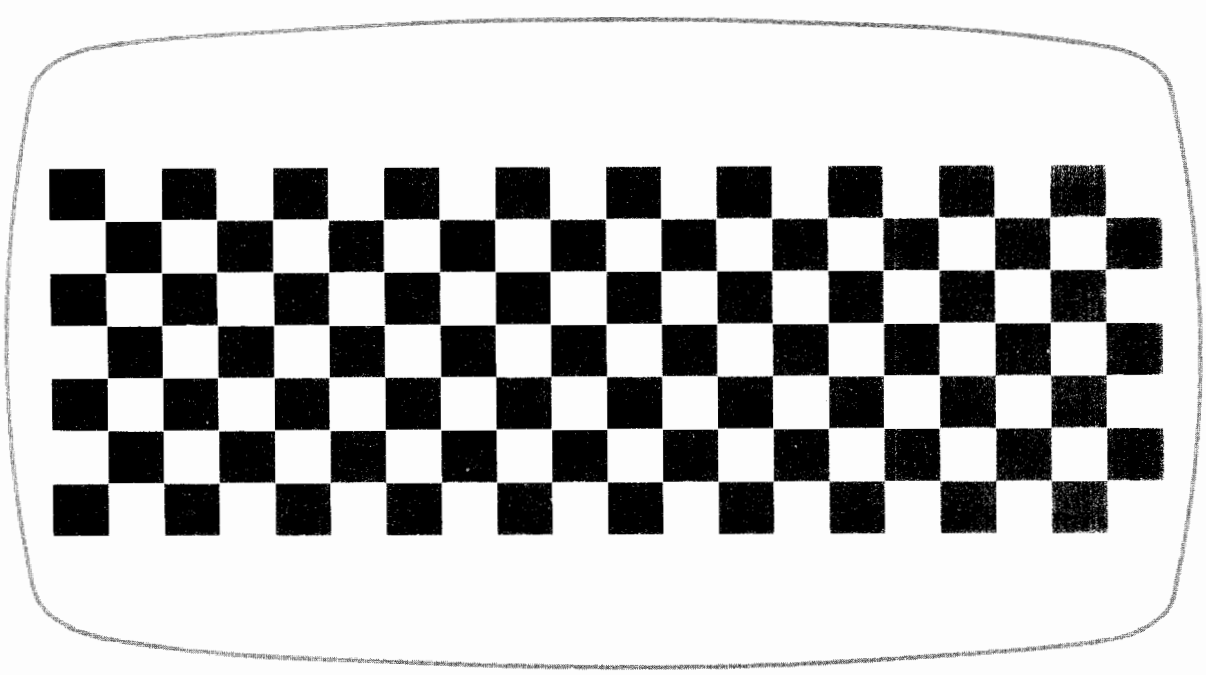
```
ENTER A STRING OF CHARACTERS, CALLED A$, UP TO ONE LINE LONG
?
This string will be converted to inverse video characters.

THIS IS A$:
This string will be converted to inverse video characters.

THIS IS B$:
This string will be converted to inverse video characters.
```

Adjacent inverse video characters can be used to form inverse video boxes. The following program displays a checkerboard pattern.

```
10 OPTION BASE 0
20 DIM S$(79,20)[1]
30 PAGESIZE 24
40 CLEAR
50 FOR J=0 TO 20
60    IF J MOD 6<3 THEN B=32 ELSE B=160
70    IF J MOD 6<3 THEN W=160 ELSE W=32
80    FOR I=0 TO 79
90       IF I MOD 8<4 THEN S$(I,J)=CHR$ (B) ELSE S$(I,J)=CHR$ (W)
100      DISP S$(I,J);
110   NEXT I
120 NEXT J
130 END
```



The program leaves the display set to PAGESIZE 24. To return to power-on vertical format, execute from the keyboard:

```
PAGESIZE 16
```

Printer Control Codes

Characters CHR$(0) through CHR$(31) are called control characters because they are interpreted by most printers as instructions. Peripheral printers, unlike the CRT, do not output these characters. Instead, if the printer recognizes the character or sequence of characters as a valid control code or control code sequence, it carries out the instruction. Control codes vary among printers, so you must refer to your printer owner's manual for a list of the control codes available for your use.

For convenience, ASCII convention assigns a mnemonic to each of the 32 control characters. The table of Character and Key Codes on page 323 lists these mnemonics, as well as the CRT display character and the key code for producing each character.

Control codes can be sent in two ways:

1. By including the CHR$ function in a PRINT or PRINT USING statement.
2. By placing the display character corresponding to the appropriate decimal code within a string expression in the print list.

Examples: The statements below send LF characters (decimal code = 10) to a peripheral printer.

```
PRINT CHR$(10)
PRINT "^"
PRINT USING "A,A,A"; CHR$(10),CHR$(10),"*"
```

Control characters are suppressed in program listings output to a PRINTER IS printer.

Changing the End-of-Line Sequence

The end-of-line (EOL) sequence is the group of instructions sent to a printer when the computer determines that a new print line should be started. An EOL sequence is sent to the printer whenever:

- The last item in a print list is output, unless the EOL sequence is suppressed by placing a semicolon or comma at the end of the print list in a PRINT statement.
- The number of characters sent to the printer since the last EOL sequence equals the line length specified by the PRINTER IS line length parameter (or the default line length of 80 characters).
- A formatted PRINT USING statement or IMAGE statement includes the delimiter or field specifier /.
- The print list in a simple or formatted print statement includes the carriage return character, CHR$(13).

The EOL sequence is stored in control registers 16 through 23 of the interface to which the printer is connected. Register 16 contains the number of instructions in the EOL sequence. Normally, this is 2 (carriage return and line feed), but may be any integer from 0 through 7. Registers 17 through 23 contain the decimal code for each character in the EOL sequence. (Normally, register 17 contains 13, register 18 contains 10, and registers 19 through 23 contain 0.)

The SET I/O statement provides the ability to alter the contents of interface control registers.

SET I/O *select code* , *register number* , *numeric data*

The first parameter specifies the select code of the interface. For changing the EOL sequence, the register number is an integer in the range 16 through 23.

> **CAUTION**
>
> Do not specify register numbers less than 16 unless your system includes an I/O ROM and you are completely familiar with the function of registers numbered 0 through 15. Certain of these registers directly access interface control and data lines. Improper use of these registers can cause a bus malfunction and/or damage to peripheral devices.

Examples: The following examples assume a select code of 7 for the interface to which the printer is attached.

Certain peripheral printers perform an automatic line feed operation whenever they receive a carriage return character, and therefore generate double-spaced output. To provide for single-spaced output, remove the line feed character from the EOL sequence by executing:

SET I/O 7,16,1 Sets the number of instructions in the EOL sequence to 1. Therefore, the EOL sequence is the contents of register 17, carriage return (CHR$(13)).

If your printer is single-spacing output and you'd prefer double-spaced output, execute:

SET I/O 7,16,3 Sets the number of instructions in the EOL sequence to 3. Therefore, registers 17, 18, and 19 contain the EOL sequence.

SET I/O 7,19,10 Places line feed character, CHR$(10), into register 19.

Section 11

User-Defined Functions and Subroutines

Introduction

In addition to the branching and looping statements discussed in section 8, the computer provides two branching techniques useful when particular program segments must be executed several times within a program:

- *Subroutines* allow program execution to "detour" through a specified sequence of statements, returning at the conclusion of the subroutine to the point of departure.
- *User-defined functions* allow you to define your own functions which can be used just as you use mathematical and string functions such as PI, SQR, and CHR$.

Subroutines

Often, the same sequence of statements must be executed several times within a program. One way, already discussed, to program such repetition is using FOR...NEXT loops. When a program's structure doesn't lend itself to looping, subroutines provide another way to repeat portions of programs.

The GOSUB statement directs the program to branch to the first line of a subroutine:

GOSUB *statement number* / *statement label*

The statement number or label must correspond to the first statement in the subroutine.

When a GOSUB statement is executed, the program branches immediately to the specified statement. Execution proceeds from the point to which branching occurred until a RETURN statement is executed.

RETURN

The RETURN statement directs the program to branch to the statement following the particular GOSUB statement that referenced the subroutine.

The GOSUB and RETURN statements are programmable only; they cannot be executed from the keyboard.

The difference in program flow between GOTO and GOSUB statements is illustrated below.

The GOTO Statement

```
10
20
30 GOTO 100
40
:
90
100
:
290
300 END
```

The GOSUB Statement

```
10
20
30 GOSUB 100
40
:
90 STOP
100
:
290 RETURN
300 END
```

A subroutine can begin with any statement except NEXT. A subroutine must end with a RETURN statement.

Within a subroutine, you can perform any operations you can do in the main body of the program, including FOR...NEXT looping and unconditional (GOTO) or conditional (IF...THEN...ELSE) branching.

A subroutine can contain more than one RETURN statement. Branching occurs to the place at which the program entered the subroutine whenever the RETURN statement is executed. Other forms of branching can be used within a subroutine to transfer program execution to other places in the main body of the program.

The computer returns an error if a program attempts to execute a RETURN statement for which no corresponding GOSUB statement was executed. Therefore, a program cannot make an initial entry into a subroutine using GOTO or IF...THEN...ELSE branching. However, program execution can branch out of the subroutine and later branch back in.

All program variables are available for use within a subroutine. New variables can be assigned within the subroutine, or previously assigned variables can be reassigned; variable assignments made within the subroutine are carried back into the main body of the program. Array variables can be dimensioned within subroutines and will be available for use in the main body of the program.

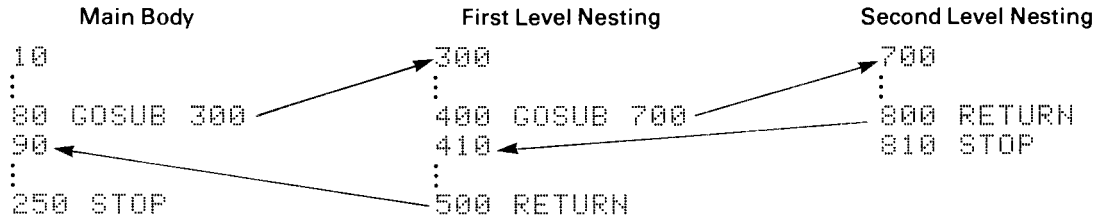
Example: The following program uses a subroutine to simulate two players each tossing one die. The tosses are repeated in case of a tie. After the results of the tosses have been displayed, the players have the opportunity to request another turn.

```
10 GOSUB TOSS
20 DISP USING 125 ; THROW1,THROW2
30 DISP "THROW AGAIN, YES OR NO";
40 INPUT ANSWER$
50 IF ANSWER$="YES" THEN GOSUB TOSS  ELSE GOTO NOMORE
60 GOTO 20
70 NOMORE: STOP
80 TOSS: ! **********Toss Subroutine**********************
90 THROW1=IP (RND *6+1)
100 THROW2=IP (RND *6+1)
110 IF THROW1=THROW2 THEN DISP "FIRST THROW TIED" @ GOTO TOSS
120 RETURN
125 IMAGE "THROW #1 = ",D,5X,"THROW #2 = ",D
130 END
```

```
THROW #1 = 1      THROW #2 = 6
THROW AGAIN, YES OR NO?
YES
THROW #1 = 1      THROW #2 = 4
THROW AGAIN, YES OR NO?
YES
THROW #1 = 2      THROW #2 = 5
THROW AGAIN, YES OR NO?
NO
```

Nesting Subroutines

You can design a program such that one subroutine branches to another subroutine. When subroutines are nested, the RETURN statement of the nested subroutine causes the program to branch back to the subroutine from which control was transferred.



Up to 255 levels of nesting are allowed, subject to limitations of available computer memory. Regardless of the level of nesting, all variables assigned in a subroutine are available for use in the main body of the program.

The Computed GOSUB: ON...GOSUB

The ON...GOSUB (computed GOSUB) statement transfers program execution to one of several specified subroutines, depending on the value of a numeric expression.

statement list

ON *numeric expression* GOSUB *statement label* / *statement number* [, *statement label* / *statement number* ...]

The statement number or statement label must reference the first line of a subroutine.

The ON . . . GOSUB statement operates much like the ON . . . GOTO statement discussed on page 99. The numeric expression is evaluated and rounded to an integer. A value of 1 causes branching to the first subroutine in the list; a value of 2 causes branching to the second subroutine, and so on. A value less than 1 or greater than the number of statements in the list produces an error.

The subroutine's RETURN statement transfers control to the first executable statement after the ON . . . GOSUB statement.

The ON . . . GOSUB statement is programmable only; it cannot be executed from the keyboard.

Example: The following program contains subroutines for finding the minimum and maximum value of a series of numbers. The ON . . . GOSUB statement (statement 90) is used to direct program execution to the desired subroutine.

```
10 OPTION BASE 1
20 DIM RANDOM(25)
30 FOR I=1 TO 25 !                  Loop generates random integers in the range
40    RANDOM(I)=IP (RND *100) !      0 through 99.
50 NEXT I
60 DISP "WANT MINIMUM OR MAXIMUM";
70 INPUT ANSWER$
80 IF ANSWER$="MINIMUM" THEN ANSWER=1 ELSE ANSWER=2
90 ON ANSWER GOSUB SMALL ,BIG   ! If ANSWER=1 transfer to statement SMALL.
                                  If ANSWER=2 transfer to statement BIG.
100 DISP ANSWER$;VALUE
110 STOP !                        Halts program execution.
120 SMALL: VALUE=RANDOM(1) !      Subroutine finds minimum of array RANDOM.
130     FOR I=2 TO 25
140       VALUE=MIN (VALUE,RANDOM(I))
150     NEXT I
160 RETURN !                      End subroutine.
170 BIG: VALUE=RANDOM(1) !        Subroutine finds maximum of array RANDOM.
180     FOR I=2 TO 25
190       VALUE=MAX (VALUE,RANDOM(I))
200     NEXT I
210 RETURN !                      End subroutine.
220 END
```

```
WANT MINIMUM OR MAXIMUM?
MAXIMUM
MAXIMUM 89
```

Run the program again to generate a new series of numbers. Then, determine the minimum.

```
WANT MINIMUM OR MAXIMUM?
MINIMUM
MINIMUM 0
```

User-Defined Functions

The computer's built-in functions allow you to manipulate numeric or string expressions to return a single numeric or string result. With user-defined functions, you can define your own functions within a program and then use those functions in much the same way you use the built-in functions.

A user-defined function may be defined by one program statement (single-line function definition) or by a sequence of statements.

Single-Line Functions

A *single-line function* definition has one of the following forms, depending upon whether the function returns a numeric or a string result:

Numeric Function:

```
DEF FNnumeric function name[(parameter[, parameter...])] = numeric expression
```

String Function:

```
DEF FNstring function name[(parameter[, parameter...])] = string expression
```

The string expression passed back to the program cannot be longer than 18 characters.

Examples:

```
20 DEF FNPOWER(X,Y)=X^Y
80 DEF FNSUBSTRNG$(A$,LENGTH)=A$[1,LENGTH]
```

In statement 20, FNPOWER is a numeric function with two numeric arguments. FNSUBSTRNG$ in statement 80 is a string function with two arguments—one numeric argument and one string argument.

Numeric and string function names must follow the rules for simple numeric and string variable names, respectively. You cannot use a subscripted (array) variable as a function name.

You can specify a maximum of 16 numeric parameters or 7 string parameters in the DEF FN statement. Parameters may include simple numeric and string variables; elements of numeric and string arrays are not allowed. Parameters need not match the function in type—in other words, you can use string parameters in numeric user-defined functions and vice versa. When the DEF FN statement contains both numeric and string parameters, the maximum number of parameters is within the range 7 through 16.

> Note: A temporary buffer is used to store the parameter names in the DEF FN statement. If the statement contains too many parameters or if long parameter names overflow buffer memory, the computer returns Error 85 : EXPR TOO BIG. If this happens, shorten the parameter names and/or reduce the number of parameters. Appendix D contains a discussion of the memory requirements of user-defined functions.

A function definition statement can be placed anywhere in the program. The statement is declaratory, and is ignored if the function is not referenced by the program.

A user-defined function is referenced by including its name and arguments, enclosed within parentheses, in a program expression.

FN *numeric function name* / *string function name* [(*argument*[, *argument*...])]

Arguments can be numeric and string constants, simple numeric and string variables, and elements of numeric and string arrays.

Examples: These statements reference the functions defined in the previous example.

```
200 DATAPOINT=FNPOWER(H(1),5)
250 ABBREVIATION$=FNSUBSTRNG$(WORD$,LETTERS)
```

The number and type of arguments listed when the function is referenced must match the parameter list in the function definition. When the function is referenced, values assigned to the arguments are "passed" to the parameters listed in the function definition according to the order in which they are listed. In the previous examples, the arguments in the function references are passed to the DEF FN parameters as shown below:

Argument	passed to	Parameter
H(1)	→	X
5	→	Y
WORD$	→	A$
LETTERS	→	LENGTH

The DEF FN parameters are *local* variables. Values assigned to X, Y, A$, and LENGTH are not available to the rest of the program.

All defined program variables are available for use within the user-defined function except variables whose names are the same as function parameters.

Example: The following short program defines and accesses the numeric function PLANCK, which has no parameters.

```
10 DEF FNPLANCK = 6.626E-27 ! Defines function PLANCK with no arguments.
20 NU=6.54E14
30 ENERGY=FNPLANCK*NU !       References function PLANCK.
40 DISP ENERGY
50 END

4.333404E-12
```

Example: In the next program, the string function FNSTMT$ uses one numeric argument passed to parameter X.

```
10 DEF FNSTMT$(X) = "Account #"&VAL$ (X)
20 ACCTNUM=10699
30 DISP FNSTMT$(ACCTNUM)
40 END

Account #10699
```

In the example above, the parameter X is a local variable, used only within the function definition. The value of ACCTNUM is temporarily assigned to X within the function definition. The assignment is not available to the rest of the program.

Example: The following programs contains a user-defined numeric function with two arguments passed between the program and the function definition. The function computes the column position at which a centered heading should begin.

```
10 HEADING$="THIS IS THE TITLE"
20 COLUMNS=80
30 DEF FNCENTER(S$,W) = W/2-INT (LEN (S$)/2)
40 DISP TAB (FNCENTER(HEADING$,COLUMNS));HEADING$
50 END
```

```
                    THIS IS THE TITLE
```

In the previous example, the string HEADING$ is shorter than 18 characters, and therefore need not be dimensioned. If a string argument passed to a function definition is longer than 18 characters, space must be allocated for the string within the function definition. The DIM statement cannot be used to allocate memory for parameters within a function definition. Instead, the string is allocated within the DEF FN statement by stating its maximum length in brackets after the parameter name.

Example: If, in the previous example, the string variable HEADING$ was assigned the string "THIS IS THE TITLE OF THE REPORT", the DEF FN statement would be:

```
40 DEF FNCENTER(S$[31],W)=W/2-INT(LEN(S$)/2)
```

Dimensions local variable S$ to 31 characters.

The parameter S$[31] is allocated within the statement. The maximum number of characters that can be allocated for string parameters in a DEF FN statement depends on the complexity of the DEF FN statement. If the argument is too large, the computer returns Error 85 : EXPR TOO BIG.

A function definition cannot be recursive; in other words, you cannot use the function that you are defining in the expression that defines the function. However, you may use other user-defined functions in the defining expression, as long as those function definitions do not reference the function you are currently defining. In other words, functions A and B are both recursive if A is defined in terms of B and B is defined in terms of A.

Multiple-Line Functions

Multiple-line functions work much like single-line functions in that arguments placed in parentheses when the function is referenced are passed to local variables within the function definition. The block of statements defining the function must begin with a DEF FN statement and end with a FN END statement.

DEF FN *numeric function name* / *string function name* [(*parameter*[, *parameter*...])]

FN END

Unlike single-line functions, the numeric or string expression assigned by the function is not included in the DEF FN statement. Instead, a statement within the function definition assigns the function a value.

FN*numeric function name*=numeric expression

FN*string function name*=*string expression*

The maximum number of arguments that can be passed to the function ranges from 7 (string arguments) through 16 (numeric arguments). As with single-line string functions, multiple-line string functions can return a string no longer than 18 characters.

Multiple-line, user-defined functions can be placed anywhere in a program; the definition need not be placed before any reference to the function.

Example: The following program contains a multiple-line, user-defined function that converts a decimal integer to its octal equivalent.

```
10 PRINT "DECIMAL","OCTAL"
20 FOR DECIMAL=1 TO 100 STEP 5
30 PRINT DECIMAL,FNOCTAL(DECIMAL)
40 NEXT DECIMAL
50 STOP
100 DEF FNOCTAL(DUMMY) !            Begins function definition.
110 OCTEQUIV=0
120 REMAINDER=DUMMY
130 FOR I=10 TO 0 STEP -1 !         Loop computes octal digits.
140 PLACE=IP (REMAINDER/8^I)
150 REMAINDER=REMAINDER MOD 8^I
160 OCTEQUIV=OCTEQUIV+PLACE*10^I
170 NEXT I !                        End of loop.
180 FNOCTAL=OCTEQUIV !              Assigns value to the function.
190 FN END!                         End of function definition.
200 END
```

```
DECIMAL              OCTAL
 1                    1
 6                    6
 11                   13
 16                   20
 21                   25
 26                   32
 31                   37
 36                   44
 41                   51
 46                   56
 51                   63
 56                   70
 61                   75
 66                   102
 71                   107
 76                   114
 81                   121
 86                   126
 91                   133
 96                   140
```

String arguments are passed to and allocated within a multiple-line function just like with single-line functions—by including the maximum length in brackets. The maximum combined length of all the string arguments passed into a multiple-line function is approximately 230 characters.

As with single-line functions, multiple-line functions cannot be recursive.

Example: The following program uses a the multiple-line function FNRIGHTJUST$ to right-justify text. Statements 10 through 40 pass quoted strings to the function. Statement 70 displays the right-justified text and passes a null string back to the program. The null string is appended to the end of each line of text.

```
10 DISP FNRIGHTJUST$("Anthropos apteros, perplexed");!          Semicolons sup-
20 DISP FNRIGHTJUST$("To know which turning to take next,");!  CR/LF caused by
30 DISP FNRIGHTJUST$("Looked up and wished he were the bird");!  character in
40 DISP FNRIGHTJUST$("To whom such doubts must seem absurd.");!  column 80.
50 STOP
60 DEF FNRIGHTJUST$(STRING$[80]) !        String function has one string
                                          parameter.
70 DISP TAB (81-LEN (STRING$));STRING$;! Displays right-justified string; semi-
                                          suppresses carriage return/line feed.
80 FNRIGHTJUST$="" !                      Function returns a null string, which
                                          is appended to the justified line.
90 FN END!                                End of function definition.
100 END
```

```
        Anthropos apteros, perplexed
  To know which turning to take next,
Looked up and wished he were the bird
To whom such doubts must seem absurd.
```

Section 12

Interrupt Programming

Introduction

Sections 8 and 11 discussed a number of different branching statements, including unconditional GOTO, conditional IF...THEN...ELSE, FOR...NEXT looping, subroutines, and user-defined functions. All these types of branching have at least one thing in common: if branching occurs, it happens when the branching statement is executed.

Interrupt programming is another form of branching. When a program has declared (activated) a program interrupt, the computer constantly monitors whether or not the specified interrupt condition has occurred. If an interrupt condition is detected, program execution branches to a specified statement or subroutine.

The computer provides three forms of interrupt programming.

Interrupt	Declaration Statement	Interrupt Condition
User-defined key	ON KEY#	Pressing the specified user-defined key.
Timer	ON TIMER#	Specified time interval has elapsed.
Error	ON ERROR	Program generates a run-time error.

Each of these statements is described in greater detail in this section.

End-of-Line Branching

The ON KEY# and ON TIMER# statements provide a form of interrupt programming called *end-of-line branching*. When a statement declares end-of-line branching for a particular condition (for instance, pressing a special function key), the computer constantly monitors whether or not the condition has been met. If the condition is met, the program completes execution of the current statement and then immediately branches to the statement specified in the end-of-line branching declaration. Program execution is said to be "interrupted" by the end-of-line branch.

Another way of looking at it is to imagine that when a program statement activates end-of-line branching, an imaginary IF...<condition>...THEN...<branch> statement is placed at the end of every successive line. If the condition becomes true while a particular program line is being executed, the branch is taken at the end of that line.

Branching Using Special Function Keys

Up to now, your main use of the special function keys has been as typing aids for BASIC keywords. As explained in section 1, these typing aids are in effect when there is no program running.

During program execution, keys (k1) through (k14) can be used to interrupt the running program and cause branching to a specified statement. This interrupt capability is established by the ON KEY# statements:

```
ON KEY# key number [, "key label"] GOTO statement label
                                        statement number
```

```
ON KEY# key number [, "key label"] GOSUB statement label
                                         statement number
```

The key number is a numeric expression that evaluates, when rounded, to an integer in the range 1 through 14. The key number specifies the key for which end-of-line branching is activated. The optional key label is the string expression displayed in an inverse video box at the bottom of the screen when (KEY LABEL) is pressed. Since the box has space for a maximum of 10 characters, key label strings longer than 10 characters are truncated.

When ON KEY# branching is declared for a special function key, end-of-line branching to the specified statement occurs when that key is pressed during program execution. If the ON KEY#...GOTO option has been used, execution proceeds sequentially from the statement to which branching occurred. If the ON KEY#...GOSUB option has been used, the subroutine's RETURN statement causes branching to the executable statement following the statement that was being executed when the key was pressed.

ON KEY#...GOTO Option

```
10 ON KEY#1 GOTO 100
:                          press (k1)
50 DISP A;B;C;D
:
100
:
300 END
```

ON KEY#..GOSUB Option

```
10 ON KEY#1 GOSUB 100
⋮                          press (k1)
50 DISP A;B;C;D
60
⋮
100
⋮
150 RETURN
⋮
300 END
```

If a GOSUB (for example, ON KEY#...GOSUB) interrupt occurs during execution of a branching statement, the subroutine's RETURN statement transfers program control to the statement referenced in the branching statement.

```
10 ON KEY#1 GOSUB 100
⋮                          press (k1)
50 GOTO 50
⋮
100
⋮
130 RETURN
⋮
300 END
```

If a special function key has not been assigned by an ON KEY# statement, pressing it while a program is running does nothing.

You cannot use the special function keys as typing aids during program execution. When the program is paused, the special function keys revert to their typing aid functions until execution is continued. ON KEY# declaratives are temporarily deactivated while a program is paused for input; pressing a key displays its keycode (inverse video character).

Key Labels

The (KEY LABEL) key is used to recall all current key labels assigned to the special function keys. All 14 special function keys can have labels defined and displayed; the positions of the seven inverse video boxes correspond to the positions of the 14 keys on the keyboard. Pressing (KEY LABEL) displays program-mode key labels assigned by ON KEY# declarative statements; pressing (SHIFT) (KEY LABEL) displays the calculator-mode typing aid key assignments.

The KEY LABEL statement is used within programs to display program-mode key labels.

```
KEY LABEL
```

Both the (KEY LABEL) key and the KEY LABEL statement move the cursor to the home (upper left) position on the display.

Example: The following program illustrates the ease with which special function keys can be defined and used within a running program.

```
10 ON KEY# 1,"MID C" GOSUB 200
20 ON KEY# 2,"D" GOSUB 300
30 ON KEY# 3,"E" GOSUB 400
40 ON KEY# 4,"F" GOSUB 500
50 ON KEY# 5,"G" GOSUB 600
60 ON KEY# 6,"A" GOSUB 700
70 ON KEY# 7,"B" GOSUB 800
80 ON KEY# 8,"C#" GOSUB 900
90 ON KEY# 9,"D#" GOSUB 1000
100 ON KEY# 11,"F#" GOSUB 1100
110 ON KEY# 12,"G#" GOSUB 1200
120 ON KEY# 13,"A#" GOSUB 1300
130 ON KEY# 14,"HIGH C" GOSUB 1400
140 CLEAR @ KEY LABEL
150 DISP "THE OCTAVE BETWEEN MIDDLE AND HIGH C"
160 DISP "PLAY MELODIES BY PRESSING THE SPECIAL FUNCTION KEYS"
170 GOTO 170
200 BEEP 201,100 @ RETURN
300 BEEP 178,100 @ RETURN
400 BEEP 157,100 @ RETURN
500 BEEP 147,100 @ RETURN
600 BEEP 130,100 @ RETURN
700 BEEP 114,100 @ RETURN
800 BEEP 101,100 @ RETURN
900 BEEP 189,100 @ RETURN
1000 BEEP 167,100 @ RETURN
1100 BEEP 138,100 @ RETURN
1200 BEEP 122,100 @ RETURN
1300 BEEP 107,100 @ RETURN
1400 BEEP 94,100 @ RETURN
1500 END
```

As soon as you press (RUN), the display is cleared and the key labels are displayed as shown below:

Now, play a few tunes with the special function keys. Notice that each press of a special function key causes one execution of the GOSUB branch as defined by the corresponding ON KEY# statement, and that one key interrupts another.

Statement 170, GOTO 170, is used in the program to create an *idle loop*. Since ON KEY# statements are active only when a program is running, it is often necessary to create a portion of the program that simply waits for a keystroke.

If a program containing ON KEY# statements chains another program into program memory, the key definitions will no longer be active. (Refer to section 21 for an explanation of chaining.)

Cancelling Key Assignments

A program-mode ON KEY# declarative and corresponding key label remain in effect until one of the following conditions occurs:

- A new ON KEY# statement is executed for that key.
- The program is initialized.

- An OFF KEY# statement is executed for that key.
- Program memory is erased by a SCRATCH or LOAD command, or by execution of a CHAIN statement.
- The computer is reset.

The syntax of OFF KEY# is:

```
OFF KEY# [key number]
```

If the optional key number is omitted, all current ON KEY# declaratives are cancelled.

Customizing the Typing Aids

At power-on, the 14 special function keys are automatically assigned to function in calculator mode as typing aids for various BASIC keywords.

Executing a customizing ON KEY# statement allows you to change the typing aid assignment of any of the 14 keys. The customizing ON KEY# statement has the syntax:

```
ON KEY# key number , "key label" , "typing aid"
```

Note the differences in syntax between the branching and customizing ON KEY# statements. The customizing ON KEY# statement is programmable, enabling you to write a set-up program for conveniently reassigning all or some of the typing aids.

Customized typing aids remain in effect until the computer is turned off or until a new customizing ON KEY# statement is executed for that key.

Example: The following statement converts (k1) into a typing tab key.

```
ON KEY#1,"TAB 5","      "
```

Both the key label and the typing aid are quoted strings. The maximum length for the key label is 10 characters; excess characters are dropped. The typing aid string can be up to 28 characters long and can include spaces.

Timer Interrupts

As mentioned in section 3, the computer contains an internal timer that is set using the SETTIME statement and read using the TIME and DATE functions. In addition to that timer, the computer provides three individual timers that can be set to interrupt a program and perform end-of-line branching after a specified time interval.

Interrupt intervals for the timers are declared with ON TIMER# statements:

```
ON TIMER# timer number , milliseconds GOTO statement label
                                          statement number
```

```
ON TIMER# timer number , milliseconds GOSUB statement label
                                          statement number
```

The timer number must be either 1, 2, or 3. The number of milliseconds must be a value between .5 and 99,999,999. If you specify a negative number of milliseconds, the minus sign will be ignored. Numbers outside the given range (including zero) set the interrupt interval to 99,999,999. Fractional millisecond values are rounded.

When the computer executes an ON TIMER# statement, the specified timer is set to zero and immediately activated. Branching occurs when the interrupt interval specified in milliseconds has elapsed. After the branch is taken, the timer is reset to zero and immediately reactivated.

Example: In the following program, timer #1 interrupts the program every 2 seconds to branch to statement 30. Statement 20 is an idle loop from which branching occurs.

```
10 ON TIMER# 1,2000 GOTO 30 ! Activates timer #1.
20 GOTO 20 !                  Idle loop.
30 BEEP @ GOTO 20 !           Interrupt branch.
40 END
```

The timers remain activated and continue to interrupt the system when execution is paused (by (PAUSE) or a PAUSE statement) or delayed (by a WAIT statement). However, the ON TIMER# branch is not taken until the program is continued or until the WAIT time has elapsed.

The timers are deactivated when you edit or scratch the program, press (RESET), or when a CHAIN statement (discussed in section 21) or an OFF TIMER# statement is executed.

OFF TIMER# *timer number*

The OFF TIMER# statement deactivates the corresponding timer; no further interrupts will occur from the specified timer until it is reactivated.

Note: Frequent timer-generated interrupts slow program execution; to obtain maximum performance, deactivate the timers when not in use.

Example: The following program uses all three timers.

```
10 ON TIMER# 1,2000 GOSUB 45
20 ON TIMER# 2,4000 GOSUB 75
30 ON TIMER# 3,6000 GOTO 105
40 GOTO 40
45 ! ***********************************************************************
50 OFF TIMER# 1 !                    Deactivate timer #1.
60 DISP "INTERRUPTED BY TIMER #1"
70 RETURN !                          Branch to statement 40.
75 ! ***********************************************************************
80 OFF TIMER# 2 !                    Deactivate timer #2.
90 DISP "INTERRUPTED BY TIMER #2"
100 RETURN !                         Branch to statement 40.
105 ! **********************************************************************
110 OFF TIMER# 3 !
120 DISP "INTERRUPTED BY TIMER #3"
130 GOTO 10
140 END
```

```
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #2
INTERRUPTED BY TIMER #3
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #2
INTERRUPTED BY TIMER #3
 :
```

and so on, until you halt program execution.

The OFF TIMER# statements (statements 50, 80, and 110) are used to prevent the timers from interfering with one another. To see the effect of omitting these statements, delete statements 50, 80, and 110 and rerun the program.

```
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #2
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #3
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #2
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #1
INTERRUPTED BY TIMER #3
 :
```

Error Recovery

Run-time errors are those that occur when a program is running (for example, division by zero). The computer automatically provides DEFAULT ON values for certain error-causing situations (errors numbered 1 through 8). These DEFAULT ON values allow the computer to display a warning rather than halting program execution with an error message.

The computer provides another way for programs to recover from run-time errors without halting program execution. The ON ERROR statement allows you to direct program execution to a specified *recovery routine* when a run-time error occurs. If an ON ERROR declarative is in effect, branching occurs immediately when a run-time error is detected. Warnings 1 through 8 (with DEFAULT ON) are regarded as errors by an ON ERROR declarative.

```
ON ERROR GOTO statement label
             statement number
```

```
ON ERROR GOSUB statement label
               statement number
```

An ON ERROR statement remains in effect during program execution until another ON ERROR statement replaces it or until an OFF ERROR statement is executed.

```
OFF ERROR
```

If the recovery routine contains an error, it is possible to place the program in an endless loop. Therefore, an OFF ERROR statement should be placed at the beginning of the recovery routine. A program will also loop endlessly if an ON ERROR statement references a nonexistent line.

If branching to a recovery routine occurs during execution of a multistatement line, the remaining statements in the line are not executed; the subroutine's RETURN statement causes branching to the line number following the multistatement line.

Example: The following program computes the log of the absolute value of the integers from -5 to +5. Since LOG(0) is undefined, the program is designed to "trap" the error.

```
10 ON ERROR GOSUB 60 !            Declares ON ERROR branching.
20 FOR NUMBER=-5 TO 5
30     DISP LOG (ABS (NUMBER))
40 NEXT NUMBER
50 STOP
55 ! ***************************************************************************
60 OFF ERROR !                    Deactivates ON ERROR branching to prevent
                                  endless loop.
70 DISP "LOG(0) IS UNDEFINED"
80 ON ERROR GOSUB 60 !            Reactivates ON ERROR branching.
90 RETURN
95 ! ***************************************************************************
100 END
```

```
 1.60943791243
 1.38629436112
 1.09861228867
 .69314718056
 0
LOG(0) IS UNDEFINED
 0
 .69314718056
 1.09861228867
 1.38629436112
 1.60943791243
```

The computer provides four functions, ERRN, ERRL, ERROM, and ERRSC, that allow a program to pinpoint the source of a run-time error. These functions are discussed in section 13, Program Debugging.

Priority of Program Interrupts

The priority of interrupts determines the order in which multiple interrupts are handled when they occur simultaneously (i.e., during execution of the same line). The three interrupts are assigned the following priorities:

ON ERROR	Highest
ON TIMER#	↓
ON KEY#	Lowest

When two or more ON ERROR or ON KEY# interrupts occur during execution of the same line, the most recent has highest priority. Two or more pending ON TIMER# interrupts are handled according to their timer numbers:

ON TIMER# 3	Highest
ON TIMER# 2	↓
ON TIMER# 1	Lowest

Notes

Section 13

Program Debugging

Introduction

The three basic types of errors—syntax errors, semantic errors, and run-time errors—were discussed in section 6. Syntax errors are relatively easy to find, since they are isolated to the statement you are entering. Correcting semantic errors involves checking the program for statements that don't make sense with respect to other statements in the program.

In most cases, the most difficult errors to find are run-time errors, since these are frequently due to flaws in program logic. One easy way to locate logic errors in small programs is to work a test case for which you know the correct answer. In lengthy, complex programs, however, a wrong test case answer probably will not pinpoint the source of the error.

The computer provides several ways to determine the source of run-time errors:

- The ON ERROR statement, discussed in section 12, allows you to "trap" errors in an error recovery routine.
- Two functions, ERRN and ERRL, allow a running program to determine the line number and error number of a run-time error.
- Two additional functions, ERRSC and ERROM, help locate the source of errors involving peripheral devices and optional plug-in ROMs.
- The computer's tracing features allow you to follow the flow of program execution from statement to statement in a running program. You can also trace the values assigned to program variables.
- The (STEP) key allows you to execute a program one line at a time.

The ERRL and ERRN Functions

When a run-time error has caused branching to a program statement specified in an ON ERROR statement, the "recovery routine" can determine the type and source of the error using the ERRL and ERRN functions. The two functions can also be executed from the keyboard when a program is halted by an error.

ERRL is the error line function. It returns the line number at which the most recent program execution error occurred.

```
ERRL
```

ERRN is the error number function. It returns the error number of the most recent program execution error. Appendix F contains a complete list of the error numbers and messages.

```
ERRN
```

Example: The following program uses an ON ERROR routine and the ERRN function to control program flow. The program asks you to enter a number and then computes the natural log (LOG) of the number. The error recovery routine provides for handling input that generates an error (0 or negative number). If you enter 0, the program will inform you of your error and request another entry. Entering a negative value causes the program to inform you that it is computing the log of the absolute value of the number.

```
10 ON ERROR GOTO RECOVER
20 DISP "ENTER A NUMBER"
30 INPUT NUMBER
40 LOGNUMBER=LOG (NUMBER)
50 DISP "NATURAL LOG OF";NUMBER;" =";LOGNUMBER
60 STOP
70 RECOVER: REM *********Error recovery routine****************************
80 OFF ERROR
90 IF ERRN =12 THEN DISP "LOG(0) IS UNDEFINED" @ GOTO 10
100 IF ERRN =13 THEN GOTO 120
110 DISP "ERROR NOT RECOVERED" @ STOP
120 NUMBER=-NUMBER
130 DISP "COMPUTING LOG OF ABSOLUTE VALUE"
140 GOTO 40
150 END
```

The following *print-all* output illustrates how the program traps and reports errors. Instead of pressing (RUN), the RUN command was typed in to show when the program was rerun.

```
RUN
ENTER A NUMBER
?
3
NATURAL LOG OF 3  = 1.09861228867
RUN
ENTER A NUMBER
?
0
LOG(0) IS UNDEFINED
ENTER A NUMBER
?
-2
COMPUTING LOG OF ABSOLUTE VALUE
NATURAL LOG OF 2  = .69314718056
RUN
ENTER A NUMBER
?
"HELLO"
ERROR NOT RECOVERED
```

Interface and ROM Errors

Certain errors are returned by interfaces. When an input/output operation causes an error from an interface, you can determine the select code of the interface from which the error originated using the ERRSC function.

ERRSC

If the error originated with the computer's integrated interface, refer to the table of interface errors in the introductory manual for an explanation of the error condition. Errors originating from an optional interface module are discussed in the documentation accompanying that module.

If an error was not generated by an interface, ERRSC returns the value 0.

The ERROM function is used to determine which portion of computer memory, including any optional plug-in ROMs, generated an error.

```
ERROM
```

The ERROM numbers for the computer's read-only memory are 0, 1, and 208. Error numbers lower than 100 return the value 0.

In addition to increasing the programming power of the computer, each of the available plug-in ROMs provides its own error messages and ERROM number. Refer to the documentation accompanying each ROM for its ERROM number and a table of error numbers and messages generated by the ROM.

Like ERRN and ERRL, both ERROM and ERRSC can be executed from the keyboard or used within a program in conjunction with an error recovery routine.

Tracing Program Execution

A convenient method for debugging logic errors in a program is to trace the order in which statements are executed and to keep a running account of the values assigned to program variables. The computer provides three tracing operations—TRACE, TRACE VAR, and TRACE ALL.

The TRACE statement traces flow of program control through branches. TRACE VAR produces an account of the values of specified program variables. TRACE ALL traces program flow for every program statement regardless of whether or not branching occurred, and outputs value changes for every program variable.

The output from tracing operations is usually quite lengthy and therefore is automatically directed to the PRINTER IS system printer. If the system does not include a printer, the tracing output is displayed.

> Note: If your system does not include a PRINTER IS printer, you cannot obtain tracing output while the computer is in *graph-all* mode.

The statements that initiate trace operations can be executed both from the keyboard and within programs. The (TR/NORM) toggle key is an immediate execute key that allows you to initiate and cancel TRACE ALL operation while a program is running. Once a trace operation is implemented, it remains in effect until tracing is cancelled.

Tracing Branches

The TRACE statement is used to trace the order of program execution during program branching.

```
TRACE
```

Nothing is printed as long as program execution is proceeding sequentially from statement to statement in numerical order. However, whenever a branch occurs in a program, both the source and destination statement numbers are printed in the form:

Trace line *statement number* to *statement number*

Example: The following program uses the computer's random number generator to simulate a coin toss. The coin is tossed five times; the program then outputs the number of heads and tails.

```
10 OPTION BASE 1
20 DIM PENNYTOSS(5)
30 HEADS,TAILS=0
40 FOR THROW=1 TO 5
50    GOSUB TOSS
60 NEXT THROW
70 PRINT "NUMBER OF HEADS =";HEADS;"NUMBER OF TAILS =";TAILS
80 STOP
85 ! **********Begin subroutine TOSS*************************************
90 TOSS: PENNYTOSS(THROW)=IP (RND *2) ! Toss produces 0 or 1.
100 IF PENNYTOSS(THROW)=0 THEN HEADS=HEADS+1 ELSE TAILS=TAILS+1
110 RETURN
115 ! **********End subroutine TOSS*****************************************
120 END
```

After entering the PENNYTOSS program, execute the TRACE statement from the keyboard. Now, press (RUN) to obtain the following output.

```
 Trace line 50 to 90
 Trace line 110 to 60
 Trace line 60 to 50
 Trace line 50 to 90
 Trace line 110 to 60
 Trace line 60 to 50
 Trace line 50 to 90
 Trace line 110 to 60
 Trace line 60 to 50
 Trace line 50 to 90
 Trace line 110 to 60
 Trace line 60 to 50
 Trace line 50 to 90
 Trace line 110 to 60
NUMBER OF HEADS = 2 NUMBER OF TAILS = 3
```

Tracing the Value of Variables

The TRACE VAR statement is used to trace changes in the values of program variables.

The statement has the syntax:

TRACE VAR *program variable* [, *program variable...*]

You cannot trace calculator-mode variables and local variables used within user-defined functions. Program variables must be allocated before they can be itemized in a TRACE VAR statement. Therefore, if you intend to execute the statement from the keyboard before running the program, you must first initialize the program (press (INIT) or execute INIT).

The list of program variables may include simple numeric and string variables and numeric and string arrays. You cannot specify individual elements of an array, such as PENNYTOSS(2). Rather, you must specify the entire array using the form:

Array name ( )	One- or two-dimensional array.
Array name (,)	Two-dimensional array. The comma is for documentation purposes only.

Whenever the value of a specified simple numeric variable or an element of the specified numeric array changes, its new value is output in the form:

Trace line *line number* *numeric variable name*[(*subscript(s)*)] = *variable value*

When a statement operates on an entire numeric array (for example, READ#, explained in section 22), the new value of the first element of the array is output.

When a TRACE VAR operation traces string variables, the new value of the string variable (simple string variable or string array element) is not output. The TRACE VAR output for simple and array string variables is:

Trace line *line number* *string variable name*[(*subscript(s)*)]

Example: To trace the values of the array PENNYTOSS() and the number of heads tossed, HEADS, first cancel the previous TRACE operation by executing:

```
NORMAL
```

If you have not yet run the program, initialize it (press (INIT)) and then execute:

```
TRACE VAR HEADS,PENNYTOSS()
```

Now, run the program to obtain the TRACE VAR output.

```
  Trace line 30 HEADS=0
  Trace line 90 PENNYTOSS(1)=1
  Trace line 90 PENNYTOSS(2)=1
  Trace line 90 PENNYTOSS(3)=0
  Trace line 100 HEADS=1
  Trace line 90 PENNYTOSS(4)=0
  Trace line 100 HEADS=2
  Trace line 90 PENNYTOSS(5)=0
  Trace line 100 HEADS=3
NUMBER OF HEADS = 3 NUMBER OF TAILS = 2
```

TRACE ALL Operation

When TRACE ALL is implemented, the order of execution is traced for every executable statement in the program regardless of whether or not branching occurs. In addition, value changes for all program variables are output in the same format used for TRACE VAR output. Local variables used within user-defined functions are not traced. A program need not be initialized before TRACE ALL operation is declared.

A TRACE ALL operation can be started in two ways:

- Execute the TRACE ALL statement within the program or from the keyboard.

```
TRACE ALL
```

- Press (TR/NORM). If no tracing operation is currently in effect, or if the computer is tracing branches (TRACE) or variables (TRACE VAR), pressing (TR/NORM) immediately establishes TRACE ALL operation. Pressing (TR/NORM) while TRACE ALL operation is established cancels all tracing operations.

Example: To obtain TRACE ALL output from the PENNYTOSS program, press (TR/NORM) or execute:

```
TRACE ALL
```

and then run the program. To terminate the TRACE ALL operation before program execution is completed, press the (TR/NORM) key.

```
  Trace line 1 to 2
  Trace line 2 to 3
  Trace line 3 to 10
  Trace line 10 to 20
  Trace line 20 to 30
  Trace line 30 TAILS=0
  Trace line 30 HEADS=0
  Trace line 30 to 40
  Trace line 40 THROW=1
  Trace line 40 to 50
  Trace line 50 to 90
  Trace line 90 PENNYTOSS(1)=1
  Trace line 90 to 100
  Trace line 100 TAILS=1
  Trace line 100 to 110
  Trace line 110 to 60
  Trace line 60 THROW=2
  .
  .
  .
  Trace line 100 to 110
  Trace line 110 to 60
  Trace line 60 THROW=6
  Trace line 60 to 70
NUMBER OF HEADS = 4 NUMBER OF TAILS = 1
  Trace line 70 to 80
```

Cancelling Tracing Operations

Any tracing operations currently in effect are cancelled by:

- Executing a NORMAL statement.
- Executing SCRATCH.
- Resetting the computer.

In addition, pressing (TR/NORM) during a TRACE ALL operation cancels all tracing.

The STEP Key

The program in computer memory can be executed one line at a time using the (STEP) key. When you press (STEP), the statement at which the program pointer is currently placed is executed and program execution halts. When you press (STEP) again, the next statement is executed.

A program must be initialized before you can step through it. To begin program stepping with the first statement, initialize the program using the INIT command or the (INIT) key. You can also begin stepping through a paused program; pressing (STEP) executes the line at which program execution paused. If the paused program has been edited, however, the program must be initialized before stepping is resumed.

The power light blinks continuously during program stepping until an END, STOP, or PAUSE statement is executed. If you step beyond the highest numbered statement, the program pointer moves to the beginning of the program.

When you step through a program, any input (e.g., INPUT, reading data files) and output (e.g., DISP, PRINT, writing to data files) operations are performed. Data is entered in response to the INPUT statement prompt (?) using the (END LINE) key; the (STEP) key is then used to execute the next statement.

Program stepping is particularly effective when used in conjunction with tracing operations (TRACE, TRACE VAR, and TRACE ALL).

Debugging Operations on Halted Programs

A number of operations can be performed on programs halted by an error, (PAUSE), or during program stepping:

- Values assigned to program variables can be checked by keying in the variable name followed by (END LINE). Array elements can be checked by keying in the element's array variable name and subscript(s). This feature is useful during program stepping for monitoring values assigned to simple and array string variables.
- Program variables can be assigned new values by executing an assignment statement (e.g., VARIABLE(2,3)=5).
- You can add, delete, and edit program statements. When a halted program is edited, it must be initialized before execution can be resumed.
- If an error has occurred, you can execute the functions ERRL, ERRN, ERROM, and ERRSC to ascertain the source of the error.
- You can execute any statement that is executable from the keyboard, including statements that change the "mode" of the system (e.g., DEG, RAD, DEFAULT ON).
- You can initiate any tracing operation by executing the appropriate tracing statement, or you can cancel all tracing operations by executing NORMAL.

Note: When a program is paused in *graph-all* mode, the only active keys are (A/G), (TR/NORM), (STEP), (CONT), and (RESET). The (TR/NORM) key can be used to initiate or cancel TRACE ALL operation. However, none of the other operations discussed above can be performed without first exiting *graph-all* mode.

If program execution is resumed with either (CONT) or (STEP), the operations performed from the keyboard while the program was halted remain in effect. For instance, reassigned variables retain their new values in the program and mode changes (e.g., DEG, DEFAULT ON) remain intact.

If a halted program is initialized and run (using (INIT) and (CONT) or (INIT) and (RUN)), all program variables are initialized to undefined values and the keyboard assignments made before the program was halted are lost. However, changes in the computer's mode (e.g., DEG, PRINT ALL) and tracing operations remain in effect.

Part III
Graphics Programming

Section 14

Introduction to Graphics

The graphics capabilities of your computer enhance your BASIC programming power. Computer graphics enable you to:

- Generate an unlimited number of lines, curves, diagrams, and graphs.
- Scale and size the graphics output to your desired proportions.
- Label all your graphics output with alphanumeric characters.
- Interact with the graphics output from the keyboard.

This section explains how to address the graphics device of your choice and introduces you to some of your computer's graphics capabilities.

Addressing the Plotting Device

Your computer is capable of routing graphics to the CRT display,or if equipped with the HP-87 Plotter ROM, to an external pen plotter. The `PLOTTER IS` statement specifies the destination for all graphics output generated by the computer.

`PLOTTER IS` *device selector*

The device selector has the following form:

device selector = *interface select code* [*device address*]

The interface select code can be any number, variable, or expression between 1 and 10. A select code of 1 or 2 specifies the CRT as the graphics device. A select code between 3 and 10 routes the graphics output to the interface with that select code. With HP-IB devices, the two-digit decimal device address must be included in the device selector. If you are using an external pen plotter, consult the documentation for your plotter to obtain the device address.

Note: In order to address and operate an external pen plotter, your computer must be equipped with the HP-87 Plotter ROM (part number 00087-15002). Without the Plotter ROM, attempts to address a pen plotter return `Error 21 : ROM MISSING`.

Examples:

`PLOTTER IS 1`	Specifies the CRT as the graphics device.
`PLOTTER IS 705`	Specifies the device with an address of 05 connected via an HP-IB interface with a select code of 7.

PLOTTER IS 3	Specifies the device connected via a non-HP-IB interface with a select code of 3.

At power-on, reset, or whenever CRT memory is reapportioned, the address defaults to PLOTTER IS 1 (specifies the CRT).

In addition to addressing the plotting device, executing the PLOTTER IS statement reads the graphics limits from the device (graphics limits are discussed in section 15). PLOTTER IS 1 assigns the default graphics limits to the CRT.

Graphics Default Conditions

The graphics default conditions are active whenever:

- The computer is turned on or reset by pressing (RESET).
- The PLOTTER IS statement or the LIMIT statement is executed. (The LIMIT statement is discussed in section 15.)
- CRT memory is reapportioned.

The default conditions are as follows:

1. Plotting boundaries (set by CLIP and LOCATE) are set to the graphics limits.
2. The plotting area is scaled in graphics units (GUs), the default scale.
3. The computer is set to user units mode with user units (UUs) equal to graphics units (GUs).
4. Pen color is set to PEN 1 (plots white dots on a black background).
5. Lines are drawn in solid line type 1 (LINE TYPE 1).
6. Labels are drawn using the standard character size (CSIZE 5 for the CRT, CSIZE 3 for pen plotters).
7. Labels are positioned according to label origin 1 (LORG 1).
8. Labeling direction is left-to-right (LDIR 0).
9. The pen is moved to the origin (lower-left corner).

Each of the terms and statements referred to above are discussed in detail in sections 15, 16, and 17.

The CRT Graphics Display

The computer provides four different CRT display modes: *alpha, graph, alpha-all,* and *graph-all.* Both *graph* mode and *graph-all* mode display graphics. They differ in the amount of CRT memory available to the graphics display. At power-on, CRT memory is automatically apportioned between the alpha display (viewed in *alpha* mode) and the graphics display (viewed in *graph* mode). The graphics portion of this memory is a 400 by 240 matrix of dots. In *graph-all* mode, the graphics display is allotted all of the CRT memory, and the size of the graphics display size is increased to a 544 by 240 matrix of dots. In *alpha-all* mode, no CRT memory is available for the graphics display.

There are four statements that switch the CRT to the four corresponding display modes: GRAPH, ALPHA, GRAPHALL, and ALPHALL. The GRAPHICS statement also sets the CRT to *graph* mode. It performs the same function as the GRAPH statement and is provided for compatibility with other BASIC language computers.

By changing modes you are either changing the display setting (alpha or graphics), changing CRT memory apportionment (all to alpha, all to graphics, or shared), or both.

The following table summarizes how the four mode-change statements and the (A/G) key affect the CRT display setting and memory apportionment.

Note: Any time CRT memory is reapportioned, all CRT memory is erased.

Display Currently In:	Execute				Press (A/G)
	ALPHA	GRAPH or GRAPHICS	ALPHALL	GRAPHALL	
Alpha mode	No effect.	Sets display to *graph* mode.	Reapportions CRT memory. Sets display to *alpha-all* mode.	Reapportions CRT memory. Sets display to *graph-all* mode.	Sets display to *graph* mode.
Graph mode	Sets display to *alpha* mode.	No effect.	Reapportions CRT memory. Sets display to *alpha-all* mode.	Reapportions CRT memory. Sets display to *graph-all* mode.	Sets display to *alpha* mode.
Alpha-all mode	Reapportions CRT memory. Sets display to *alpha* mode.	Reapportions CRT memory. Sets display to *graph* mode.	No effect.	Reapportions CRT memory. Sets display to *graph-all* mode.	Reapportions CRT memory. Sets display to *graph* mode.
Graph-all mode	Reapportions CRT memory. Sets display to *alpha* mode.	Reapportions CRT memory. Sets display to *graph* mode.	Reapportions CRT memory. Sets display to *alpha-all* mode.	No effect.	Reapportions CRT memory. Sets display to *alpha* mode.

For a more complete discussion of CRT memory apportionment, refer to section 1.

Graph Mode

You can enter *graph* mode from any of the other three display modes (*alpha, graph-all, alpha-all*) by executing the GRAPH statement or the GRAPHICS statement.

```
GRAPH
```

```
GRAPHICS
```

When the computer is in *graph* mode, you are viewing the current graphics display. If the computer is in *alpha-all* or *graph-all* mode when GRAPH is executed, CRT memory is immediately reapportioned to its power-on values and the *alpha-all* or *graph-all* mode display memory is erased.

During normal apportionment, the (A/G) key shifts the display back and forth between *alpha* and *graph* mode. Refer to the table on page 167 for the function of the (A/G) key in *alpha-all* and *graph-all* modes.

In addition, the following graphics statements automatically switch the CRT from *alpha* mode to *graph* mode when the specified graphics operation is performed within the graphics limits.

AXES	GRID	LGRID
BPLOT	IDRAW	PLOT
BREAD	IPLOT	RPLOT
DRAW	LABEL	XAXIS
FRAME	LABEL USING	YAXIS
GCLEAR	LAXES	

Example: Enter the FRAME statement from the keyboard while in *alpha* mode. The computer shifts to *graph* mode and frames the plotting area.

Switching to Alpha Mode

The ALPHA statement switches the display to *alpha* mode. If the display is currently in *alpha-all* or *graph-all* mode, executing ALPHA reapportions CRT memory so it is shared between the alpha and graphics displays. The current CRT display memory is erased.

```
ALPHA
```

Executing LIST, CLEAR, or DISP in *graph* mode, or pressing the (A/G) key or any of the alphanumeric or numeric keys also shifts the display to *alpha* mode. Tracing operations shift the display to *alpha* mode in the absence of a system printer.

Example: If you did the previous example, the display is set to *graph* mode and you are viewing the framed plotting area. Press the (A/G) toggle key to return to *alpha* mode.

Graph-All Mode

Graph-all mode makes a larger plotting area available for oversized plots and other wide screen graphics operations. The GRAPHALL statement shifts the display to *graph-all* mode. Executing GRAPHALL while in *alpha, alpha-all,* or *graph* mode erases all CRT memory.

```
GRAPHALL
```

When the display is in *graph-all* mode, executing ALPHA or pressing the (A/G) key returns CRT memory to power-on apportionment. The *graph-all* mode display memory is erased, and the display switches to *alpha* mode.

Note: The DISP, CLEAR, and LIST statements are ignored in *graph-all* mode as a safeguard against accidentally erasing the *graph-all* mode display. Tracing operations output information in *graph-all* mode only if there is a system printer.

The following precautions are advised when using *graph-all* mode:

- If your program output includes alphanumeric data, route the data to the system printer or to a mass storage data file. Execution of GRAPHALL erases any record of data routed to the CRT.
- If you plot in *graph-all* mode and want a record of your graphics display, include a GSTORE statement within the program. You can't pause the program and type in a GSTORE statement from the keyboard, since entering *alpha* mode to execute GSTORE would erase the *graph-all* mode display. Refer to section 21 for a discussion of the GSTORE (store graphics) statement.
- Any mode change which involves reapportioning CRT memory resets the graphics parameters to the graphics default conditions listed on page 166. For example, if you scale your plot in *graph* mode, then switch to *graph-all* mode, the scaling defaults to graphics units. Execute GRAPHALL before specifying any graphics parameters if you wish to use *graph-all* mode.
- Any errors and warnings generated while in *graph-all* mode are signified by a beep. Since there is no alpha display memory in *graph-all* mode, error and warning messages must be routed to the system printer by executing PRINT ALL or CRT IS with the printer address.
- Remove all GRAPHALL statements from portions of the program that access an external plotter.

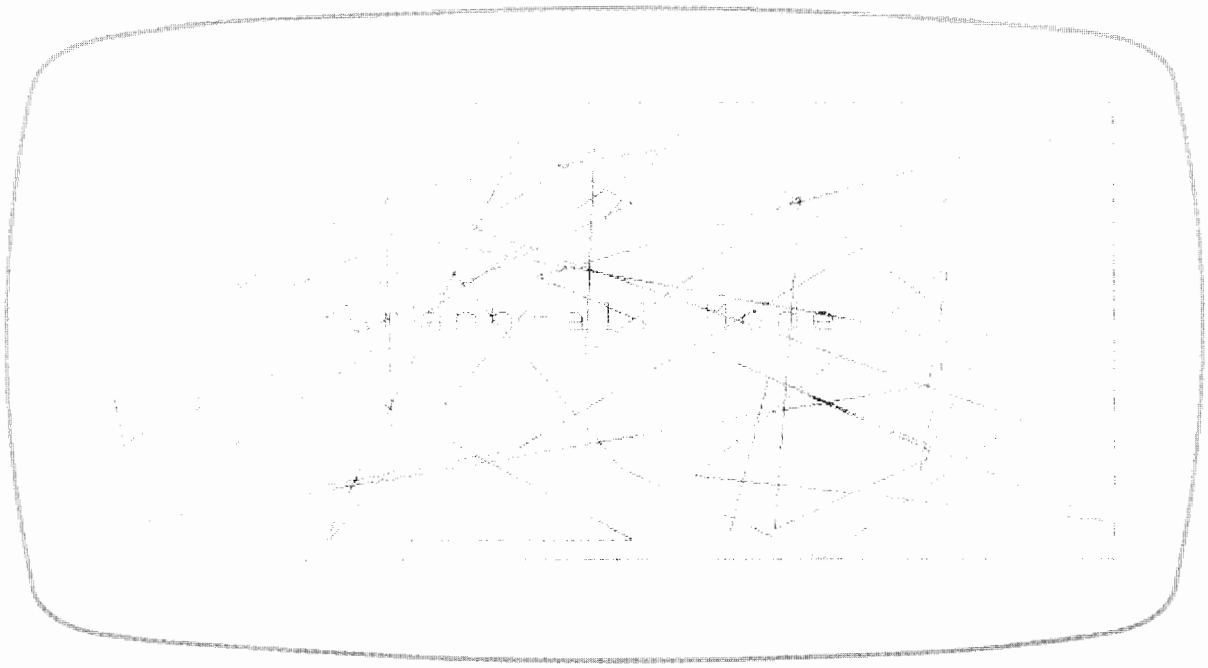
To get an idea of the graphics display size in *graph* and *graph-all* modes, enter and run the following program. The CRT displays the available plotting area for each mode.

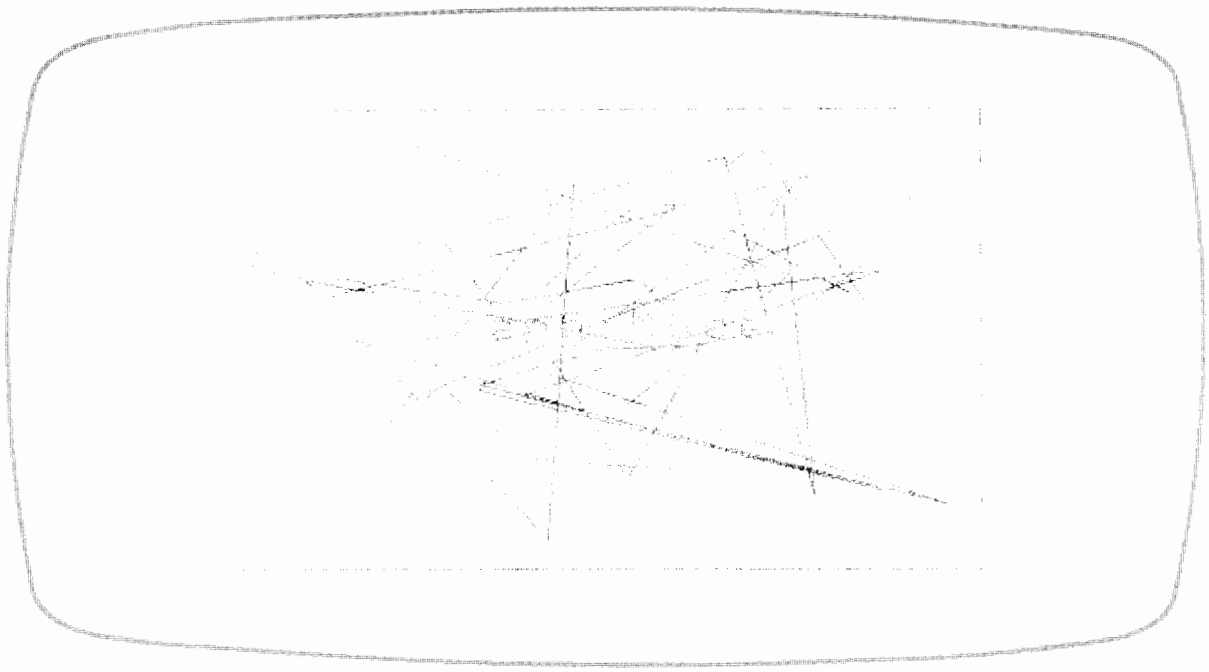
Note: The actual shape of displayed graphics output depends on the horizontal and vertical dot spacing of your display device. Graphics output shown in this manual closely approximates the relative dot spacing of the HP-87 CRT.

First, execute SCRATCH to clear the system memory of previous programs.

Example:

```
10 ! *** Graph-all/Graph Modes ***
20 PLOTTER IS 1 !                          Specifies the CRT as the plotter.
30 G$="Graph-all Mode"
40 GRAPHALL !                              Sets the display to graph-all mode.
50 GOSUB 1000 !                            Goes to Random Drawing Subroutine.
60 WAIT 2000
70 GCLEAR !                                Clears the graphics display.
80 G$="Graph Mode"
90 GRAPH !                                 Sets the display to graph mode.
100 GOSUB 1000 !                           Goes to Random Drawing Subroutine.
110 END
1000 ! *** Random Drawing Subroutine ***
1010 SCALE 0,1,0,1 !                       Scales the plotting area.
1020 FRAME !                               Frames the plotting area.
1030 MOVE .25,.5 !                         Moves the pen to the x,y coordinate.
1040 CSIZE 12 !                            Specifies character size 12.
1050 LABEL G$ !                            Labels the current display mode.
1060 FOR P=1 TO 150 !                      Loop plots 150 random points.
1070 PLOT RND ,RND  !                      Draws a line to the random point.
1080 NEXT P !                              End loop.
1090 RETURN
1100 END
```





Alpha-All Mode

In *alpha-all* mode the entire CRT display memory is devoted to the alpha display. The *alpha-all* mode display is capable of storing 204 lines of information compared to 54 lines for *alpha* mode. Executing the ALPHALL statement erases all CRT memory from *alpha, graph,* or *graph-all* modes, and shifts the display to *alpha-all* mode.

```
ALPHALL
```

If the CRT is set to *alpha-all* mode, execute either GRAPH or GRAPHALL before performing any graphics operations on the CRT display; you can't perform any CRT graphics operations while in *alpha-all* mode. Pen plotters are fully operational in *alpha-all* mode.

Clearing the Graphics Display

The GCLEAR statement clears the graphics display, using the current background color, from the specified *y*-coordinate to the bottom of the screen.

GCLEAR [*y-coordinate*]

The *y*-coordinate is interpreted according to the current scaling units; it could be a number, variable, or expression. More attention is given to scaling units in the next section; for now, at power-on, the *y* scale is 0 to 100 for both *graph* and *graph-all* modes. If no parameter is specified, the GCLEAR statement clears the entire graphics screen.

Example:

GCLEAR 50	Clears lower half of graphics display having a vertical scale of 0 to 100.

It's advisable to use the GCLEAR statement before you begin a new plot in a program, thus assuring that you do not plot over any previous graphics.

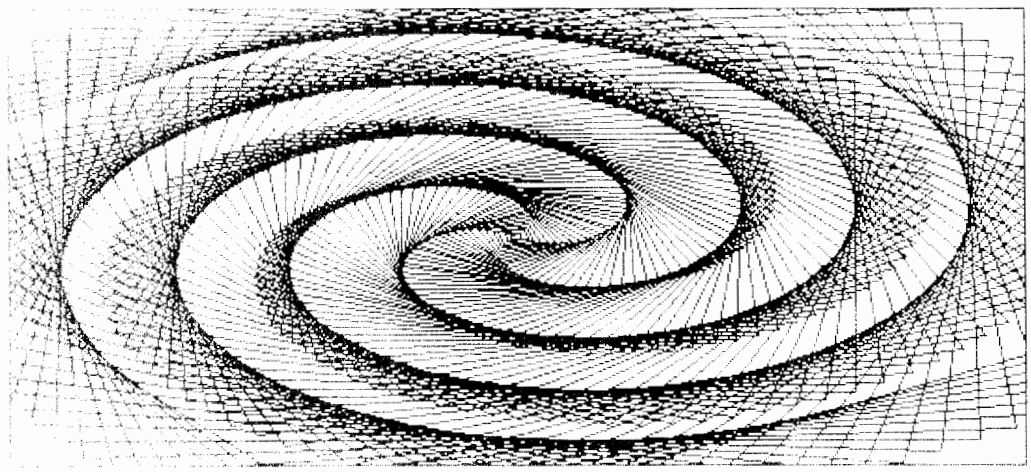
The (RESET) key also clears the graphics display, in addition to performing a number of other functions. (Refer to the table of Reset Conditions in appendix D for a complete list of reset conditions.)

Execute GCLEAR now (without a parameter) to clear the display from our first graphics program. The display switches to *alpha* mode when you type in a graphics statement from the keyboard. It reverts back to the graphics display when the graphics statement is executed.

A Graphics Example

The demonstration disc packaged with your computer has a program entitled GRAMPLE stored on it. Load and run this program for a computer graphics demonstration. Choose whether you want to plot with inverse video and enter YES or NO from the keyboard.

Note: When a program is paused for input with the display in *graph* or *graph-all* mode, data can be entered from the numeric keypad and the alphanumeric keys. The input data is written on the graphics display using the current character size and pen position.



Interchanging Plotter and CRT Graphics

The graphics discussion in this manual emphasizes the use of the CRT graphics display. Unless otherwise noted, external pen plotters function the same way as the CRT. Some exceptions are the BPLOT statement (CRT statement) and the DIGITIZE statement (plotter statement).

To produce the examples written for a CRT on an external pen plotter you need to do the following:

1. Install the HP-87 Plotter ROM. (The procedure for installing ROMs is explained in section 2 of the introductory manual and in the *HP 82936A ROM Drawer Instruction Sheet*).
2. Change the PLOTTER IS statement in the example program to the correct address for your device.
3. Either change the LIMIT statement parameters or delete the LIMIT statement from the example program (the LIMIT statement is discussed in section 15). The PLOTTER IS statement resets the graphics limits to the manually-set or default values for the pen plotter.
4. Delete all GRAPHALL statements from portions of programs that route graphics output to the external plotter. If *graph-all* mode or *alpha-all* mode is used elsewhere in the program, also delete all ALPHA and GRAPH statements from portions of the program routing graphics to the plotter.

The examples in this manual were produced on the HP-87 CRT display and dumped to an HP 7310A Graphics Printer (requires the HP-87 Plotter ROM). The same programs will generate differently proportioned graphics on other plotting devices.

Digitizing with a Plotter

Your computer has the capability to digitize graphics with an external pen plotter when equipped with the HP-87 Plotter ROM. This capability is not available on the CRT. Refer to the documentation for the HP-87 Plotter ROM for a discussion of the DIGITIZE statement.

Attempting to digitize on the CRT will generate Error 109 : DIGITIZE.

Notes

Section 15

Positioning and Scaling Plots

A program written to plot data on a plotting device usually includes some preliminary set-up operations to define the plotting area. You can position and size the plot anywhere within the physical limits of the plotting device and scale this area according to your particular needs. If the default values for the size and scale fit your application, the initial set-up operations are unnecessary.

Graphics operations routed to the CRT assume the horizontal and vertical dot spacings of the HP-87 CRT. If you are using an HP-86 to plot graphics on a monitor, refer to appendix B of *Introduction to the HP-86* for additional information on establishing plotting boundaries and scale units.

Physical and Graphics Limits

The CRT graphics display and all other plotting devices have physical limits which restrict the size of the graphics output. For example, the size of the CRT screen and the platen size of a plotter are the physical limits specific to the plotting device.

Within these physical limits, the user can choose the size and location of the graphics output by setting the graphics limits. The graphics limits are the boundaries, assigned either by default or by the user, for all graphics output. (The only exception is during BPLOT operations; BPLOTting can extend beyond the specified graphics limits, but not the physical limits.)

Your computer is capable of assigning graphics limits to the CRT and other plotting devices from the keyboard or within a program.

Default Graphics Limits

The CRT default graphics limits are in effect whenever:

- The computer is turned on or reset by pressing (RESET).
- The PLOTTER IS 1 statement is executed.
- CRT memory is reapportioned by any of the procedures described on pages 166 through 171 (for example, by executing GRAPHALL while in *alpha, graph,* or *alpha-all* mode).

Unless the CRT is addressed with a LIMIT statement, the default graphics limits remain in effect.

The size of the HP-87 CRT plotting area bounded by the default graphics limits is approximately 125 mm by 75 mm in *graph* mode and 171 mm by 75 mm in *graph-all* mode. The default graphics limits coincide with the physical limits of the CRT graphics display.

The default graphics limits vary for different pen plotters, but are generally close to the physical limits of the device. Refer to the documentation accompanying your pen plotter for additional information regarding graphics limits.

Setting the Graphics Limits

The LIMIT statement enables you to specify the graphics limits anywhere within the physical limits of the plotting device. The LIMIT parameters represent the coordinates, in millimeters, of the lower-left and upper-right corners of the plotting area. The origin (*x min* = 0, *y min* = 0) is normally the lower-left physical limit of the plotting device.

```
LIMIT x min , x max , y min , y max
```

The LIMIT parameters can be numbers, variables, or expressions.

The allowable range for the CRT LIMIT parameters are:

Display Mode	*x min*	*x max*	*y min*	*y max*
Graph mode	0-125.6	0-125.6	0-75.2	0-75.2
Graph-all mode	0-171	0-171	0-75.2	0-75.2

Refer to appendix B of the *Introduction to the HP-86* manual for a discussion of establishing graphics limits for a peripheral monitor.

The upper and lower bounds of the CRT LIMIT parameter range represent the CRT physical limits. If a LIMIT parameter is outside the physical limits of the CRT, the computer returns a warning and assigns the physical limit to the out-of-bounds parameter. For example, in *graph* mode LIMIT -10,200, 100,10 is interpreted by the computer as LIMIT 0,125,75,10.

The LIMIT statement overrides any previously set or default graphics limits. The current graphics limits remain in effect until a new LIMIT statement is executed, or until the default graphics limits are activated by any of the operations listed on page 175.

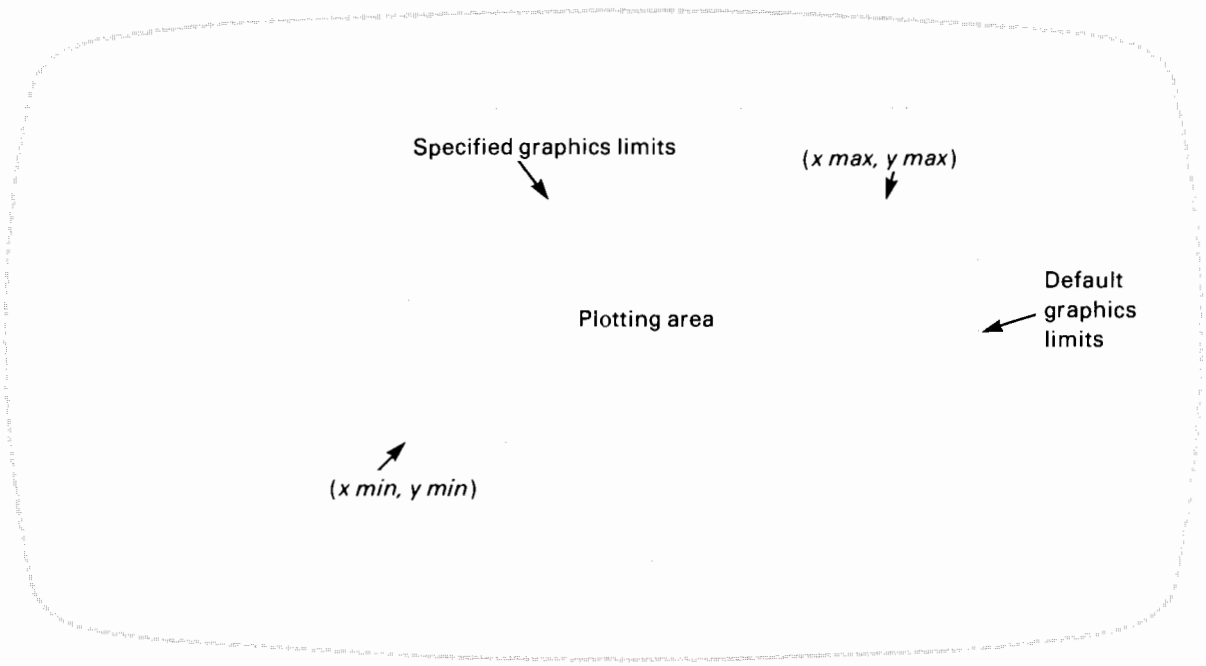
Keep this in mind when you are performing your plots. If, for instance, you do not execute a LIMIT statement in a program and your plot turns out smaller than you expected, the plotting device is probably using the graphics limits set by a previous LIMIT statement.

On most pen plotters the graphics limits can be set manually; the plotter limits are read by the computer upon execution of the PLOTTER IS statement. The LIMIT statement performs the same function for pen plotters as for the CRT, but the maximum allowable parameters are device-dependent. If a LIMIT statement parameter is out-of-bounds for a pen plotter, the computer returns an error and ignores the statement. For more information refer to the documentation for your pen plotter.

Note: In addition to specifying the graphics limits, executing the LIMIT statement also activates the graphics default conditions listed on page 166.

Example: The following program demonstrates the default and user-defined graphics limits:

```
10 ! *** Limit ***
20 PLOTTER IS 1 !                    Specifies the CRT as the plotter and
21 !                                 sets the default graphics limits.
30 GCLEAR !                          Clears the graphics display.
40 LINE TYPE 5 !                     Specifies a dashed line type.
50 FRAME !                           Frames the default plotting area for
51 !                                 reference.
60 LIMIT 30,30+80,20,20+40 !         Specifies an 80mm X 40mm plotting
61 !                                 area that is offset from the CRT's
62 !                                 lower-left physical bounds.
70 LINE TYPE 1 !                     Specifies a solid line type.
80 FRAME !                           Frames the specified plotting area.
90 END
```



Reflecting Plots

The normal sequence of parameters in the LIMIT statement puts the origin of your graph in the lower-left corner of the graphics output. By exchanging parameters you can produce a reflected image of the plot (except labels) without any additional changes in the program. Three kinds of reflected images are possible:

1. LIMIT *x max*, *x min*, *y min*, *y max*
   Reflects the output across the *y*-axis.
2. LIMIT *x min*, *x max*, *y max*, *y min*
   Reflects the output across the *x*-axis.
3. LIMIT *x max*, *x min*, *y max*, *y min*
   Reflects the output across the origin.

The SCALE, SHOW, and LOCATE statements can also be used to reflect plots by exchanging parameters. Refer to section 19 for an example of reflected plots.

These procedures do not reflect labels or BPLOT data. Labels are reflected by the CSIZE statement, which is discussed in section 17.

Ratio Function

On the HP-87, the RATIO function returns a value equal to the ratio of the dimensions of the graphics limits—the horizontal dimension divided by the vertical dimension (x/y). The value of the RATIO function depends on the LIMIT parameters (user-specified graphics limits) or on the plotting device (default or manually-set graphics limits). Refer to appendix B of *Introduction to the HP-86* for an explanation of the RATIO function on the HP-86.

```
RATIO
```

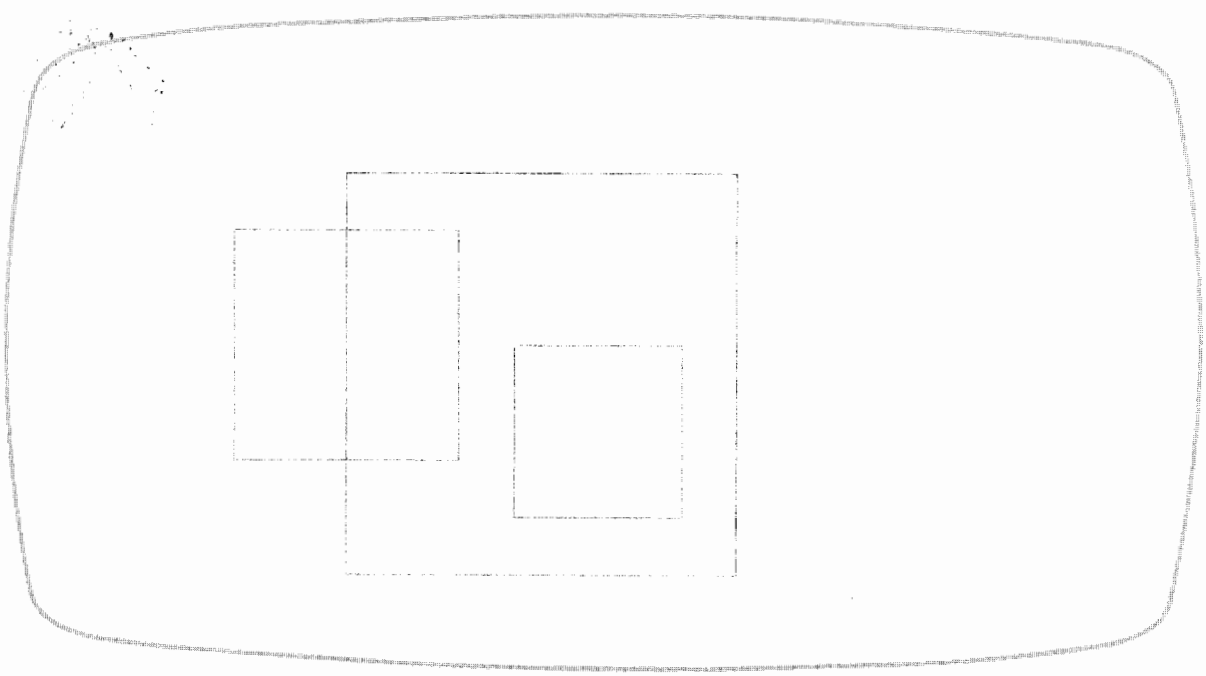
Example: Execute RATIO using the following graphics limits.

```
LIMIT 5,95,10,60
RATIO
 1.8
```

The RATIO function is useful for changing the size or location of the plotting area, without changing the proportions.

Example:

```
10 ! *** RATIO ***
20 PLOTTER IS 1 !                 Specifies the CRT as the plotter.
30 GCLEAR !                       Clears the graphics display.
40 LIMIT 20,90,0,70 !             Specifies the graphics limits.
50 FRAME !                        Frames the plotting area.
60 R=RATIO  !                     Assigns RATIO to the variable R.
70 LIMIT 0,R*40,20,60 !           Specifies the graphics limits while maintaining
71 !                              the same RATIO as the previous LIMIT.
80 FRAME !                        Frames the plotting area.
90 LIMIT 50,80,10,30/R+10 !       Specifies the graphics limits while maintaining
91 !                              the same RATIO as the previous LIMIT.
100 FRAME !                       Frames the plotting area.
110 END
```



Scaling the Plotting Area

Once the plotting area is defined, either by default or by specifying the graphics limits, the scale can be chosen to suit your particular graphics application.

You can use the default scale—graphics units (GUs), or you can specify your own scale—user units (UUs), by executing one of the three scaling statements: `SCALE`, `SHOW`, and `MSCALE`. If you do not specify your own units, the computer sets UUs equal to GUs.

Graphics Units Scale

The graphics units scale is active at power-on or after reset, and remains active until a scaling statement is executed. The graphics units scale is also active whenever CRT memory is reapportioned or the `PLOTTER IS` or `LIMIT` statement is executed.

The computer determines the shortest dimension of the area defined by the graphics limits and scales it 0 to 100. That is, one GU corresponds to one percent of the shortest side of the plotting area. The other dimension is scaled with the same size units (equal unit scaling) starting at 0; the upper-bound of the longest side depends on the ratio of the graphics limits.

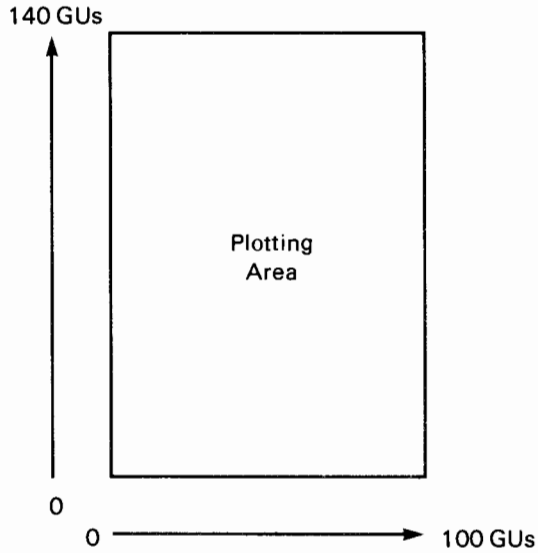
> Note: Equal unit spacing of GUs in CRT graphics assumes the dot spacing of the HP-87 CRT. Monitors used with the HP-86 may produce unequal horizontal and vertical GUs.

The graphics units scale is determined by the graphics limits. *When the graphics limits change, the size of the graphics units scale also changes.*

Example:

```
LIMIT 10,60,0,70
```

The length of one GU is equal to 1/100 (one percent) of the length of the shortest side of the area bounded by the graphics limits. The length of the longest side of the plotting area is something greater than 100, depending on the width/height aspect ratio.

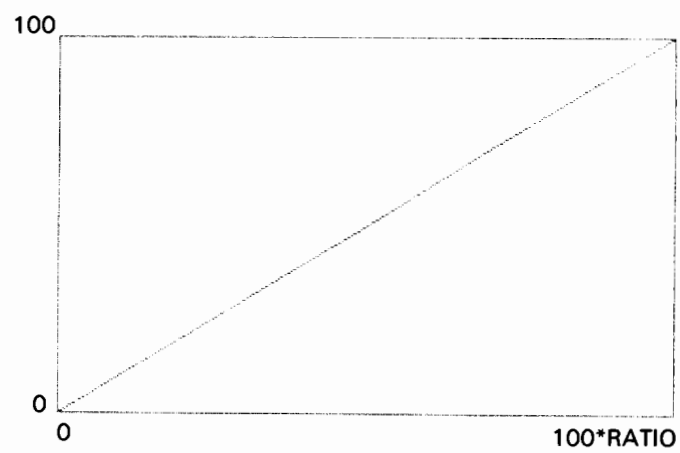


The graphics units scale provides easy access to the plotting area on a percentage basis, regardless of the size of the plotting area.

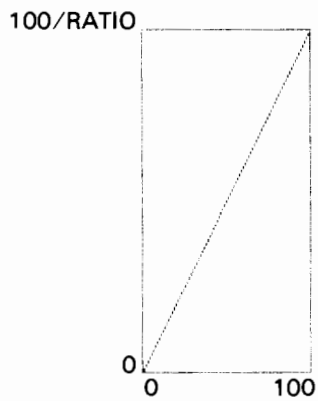
Example: The following program draws a line from point (0,0), in GUs, to the opposite corner of the plotting area. Enter the graphics limits from the keyboard; the RATIO function is used to compute the length in GUs of the longest side of the plotting area.

```
10 ! *** Graphics Units ***
20 PLOTTER IS 1 !                          Specifies the CRT as the plotter.
30 DISP "Enter LIMIT parameters: xmin,xmax,ymin,ymax"
40 INPUT xmin,xmax,ymin,ymax
50 LIMIT xmin,xmax,ymin,ymax !             Specifies the graphics limits.
60 DISP "RATIO = ",RATIO  !                Displays current value of RATIO.
70 WAIT 2000
80 GCLEAR !                                Clears the graphics display.
90 FRAME !                                 Frames the plotting area.
100 MOVE 0,0 !                             Moves the pen to lower-left corner.
110 Xmax=100*MAX (1,RATIO ) !              Determines maximum x value in GUs.
120 Ymax=100*MAX (1,1/RATIO ) !            Determines maximum y value in GUs.
130 DRAW Xmax,Ymax !                       Draws a line to upper-right corner.
140 END
```

```
Enter LIMIT parameters: xmin,xmax,ymin,ymax
?
10,120,0,65
RATIO =                1.69230769231
```



```
Enter LIMIT parameters: xmin,xmax,ymin,ymax
?
20,50,0,60
RATIO =                .5
```



User Units Scale

There are three scaling statements that allow you to specify the user units scale: SCALE, SHOW, and MSCALE. All three scaling statements specify the scale for the current plotting area as defined by the graphics limits or by a LOCATE statement, a statement which specifies plotting boundaries (discussion of plotting boundaries appears later in this section).

The SCALE statement defines minimum and maximum values of x and y for the current plotting area.

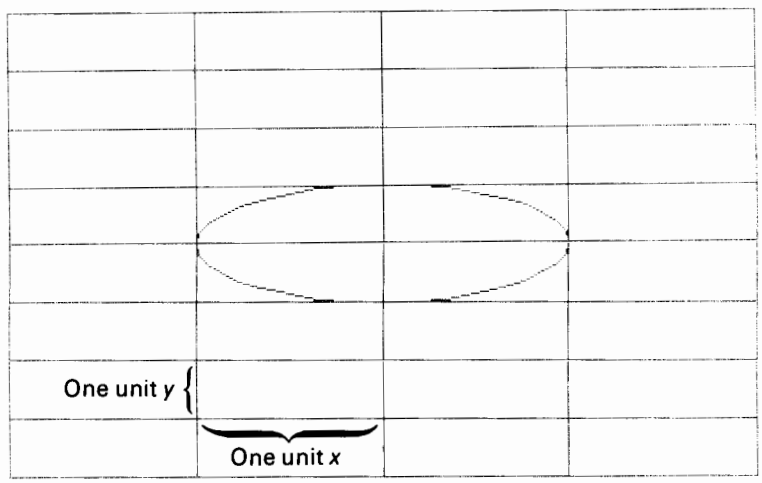
SCALE *x min* , *x max* , *y min* , *y max*

The first two parameters specify the values represented by the left and right boundaries of the plotting area. The last two parameters specify the values represented by the lower and upper boundaries of the plotting area. The parameters can be numbers, variables, or expressions.

The SCALE statement must follow specification of the graphics limits (or the LOCATE-defined plotting boundaries) in order to map the user units scale onto the current plotting area.

Example:

```
10 ! *** Scale ***
20 PLOTTER IS 1 !                    Specifies the CRT as the plotter.
30 GCLEAR !                          Clears the graphics display.
40 DEG !                             Sets degrees mode.
50 SCALE -2,2,-4,4 !                 Specifies user units scale.
60 GRID 1,1,0,0 !                    Draws a grid in current user units.
70 FRAME !                           Frames the plotting area.
80 MOVE 1,0 !                        Moves to the start of the ellipse.
90 FOR DG=0 TO 360 STEP 10 !         Loop draws an ellipse in 10 degree
91 !                                 increments.
100 DRAW COS (DG),SIN (DG) !         Draws a line to point specified in
101 !                                the current units.
110 NEXT DG !                        End loop.
120 END
```



The SCALE statement specifies user units independently in the x and y directions. Like the LIMIT statement, the SCALE statement parameters can be exchanged to produce reflected plots.

The SHOW statement specifies user units for a plotter or the HP-87 CRT such that one unit of x equals one unit of y (equal unit scaling). Thus, the plotting area is scaled with unit squares. The SHOW statement parameters specify the minimum number of units to be mapped onto the current plotting area; units are added by the computer, if necessary, to fill the plotting area.

SHOW *x min* , *x max* , *y min* , *y max*

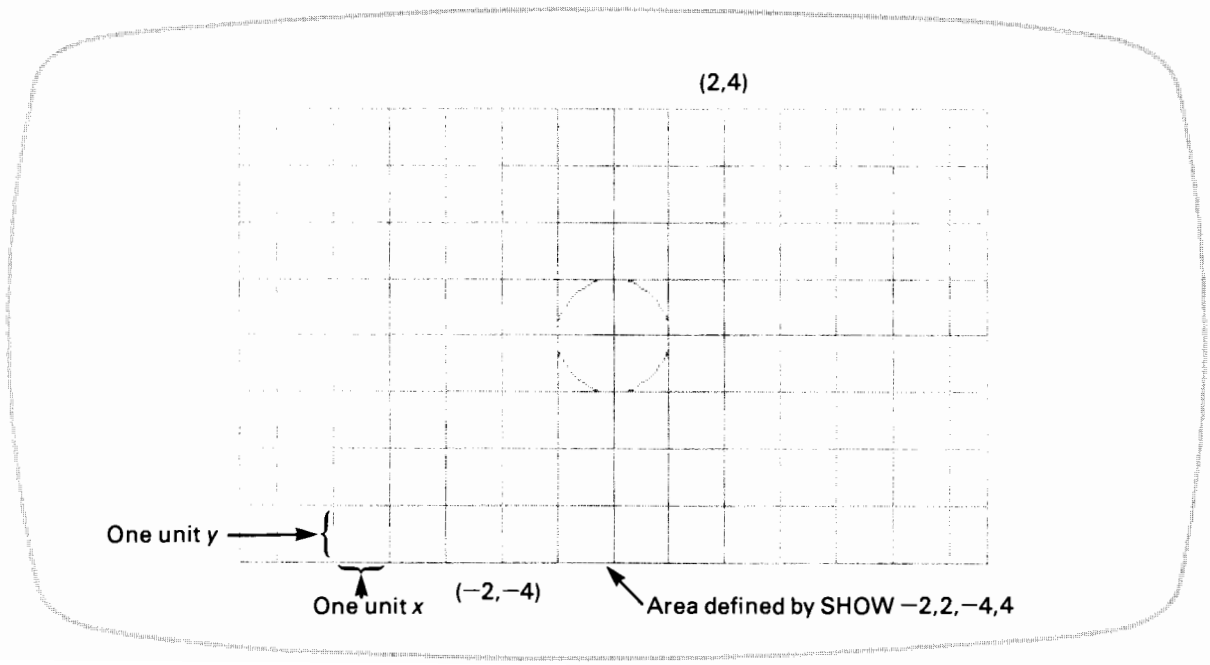
The x min and x max parameters specify the minimum bounds in the x direction. The y min and y max parameters specify the minimum bounds in the y direction. The parameters can be numbers, variables, or expressions.

The SHOW statement must follow specification of the graphics limits (or the LOCATE-defined plotting boundaries) in order to map the user units scale onto the current plotting area. The SHOW statement parameters can be exchanged to produce reflected plots.

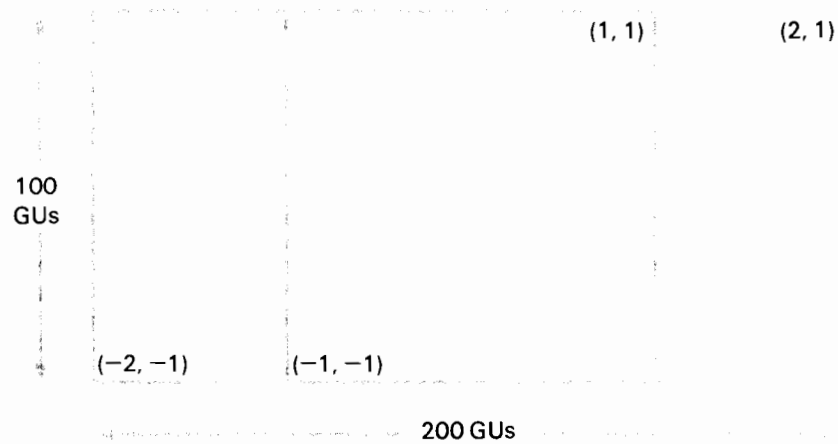
Example: Replace the SCALE statement in the previous example with the following SHOW statement. Because of equal unit scaling, the figure drawn is now a circle instead of an ellipse.

```
50 SCALE -2,2,-4,4 ------>50 SHOW -2,2,-4,4
```

```
10 ! *** Show ***
20 PLOTTER IS 1 !                          Specifies the CRT as the plotter.
30 GCLEAR !                                Clears the graphics display.
40 DEG !                                   Sets degrees mode.
50 SHOW -2,2,-4,4 ! ****************       Specifies UUs as equal units scale.
60 GRID 1,1,0,0 !                          Draws a grid in current user units.
70 FRAME !                                 Frames the plotting area.
80 MOVE 1,0 !                              Moves to the start of the circle.
90 FOR DG=0 TO 360 STEP 10 !               Loop draws a circle in 10 degree
91 !                                       increments.
100 DRAW COS (DG),SIN (DG) !               Draws a line to point specified in
101 !                                      the current units.
110 NEXT DG !                              End loop.
120 END
```



The SHOW statement sets up UUs such that the specified area is as large as possible and is centered within the graphics limits or within the plotting boundaries, if specified. For example, if the LIMIT rectangle is twice as wide as it is high (e.g., LIMIT 0,100,0,50), then SHOW -1,1,-1,1 is equivalent to SCALE -2,2,-1,1.



The MSCALE statement sets millimeters as user units and specifies the location of the origin. MSCALE is useful when correspondence with physically measurable objects is desirable, as in drafting and mapping applications. The accuracy of the metric scale depends on the graphics device in use; the MSCALE statement cannot be used to establish millimeter user units for a peripheral monitor.

Parameter specification in the MSCALE statement is different than in the other scaling statements.

MSCALE *x-offset* , *y-offset*

MSCALE specifies user units equal to millimeters, and offsets the origin (point (0,0) in millimeter space) from the lower-left graphics limits corner by the specified distance, in millimeters. The MSCALE parameters can be numbers, variables, or expressions.

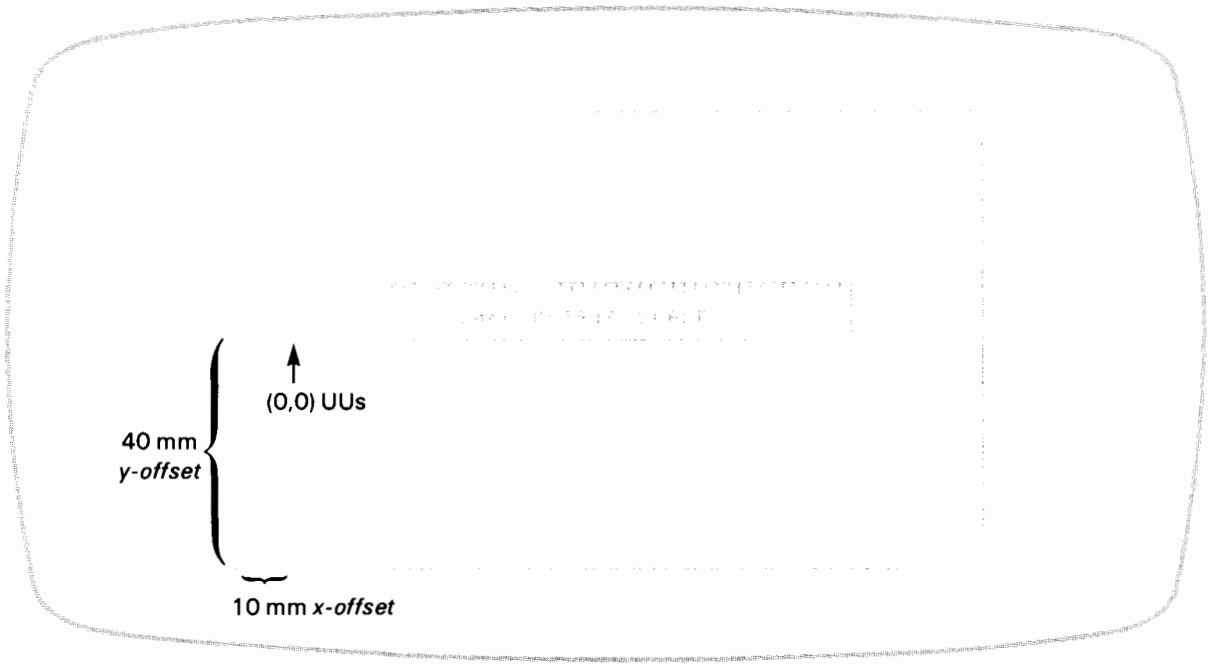
Example:

MSCALE 10,15	Specifies metric user units, with the origin offset 10 mm to the right and 15 mm up from the lower-left corner of the plotting area.

Like SCALE and SHOW, the MSCALE statement must follow specification of the graphics limits or the LOCATE plotting boundaries in order to map the user units scale onto the current plotting area.

Example: The following program uses the MSCALE statement to draw a metric ruler on the HP-87 CRT.

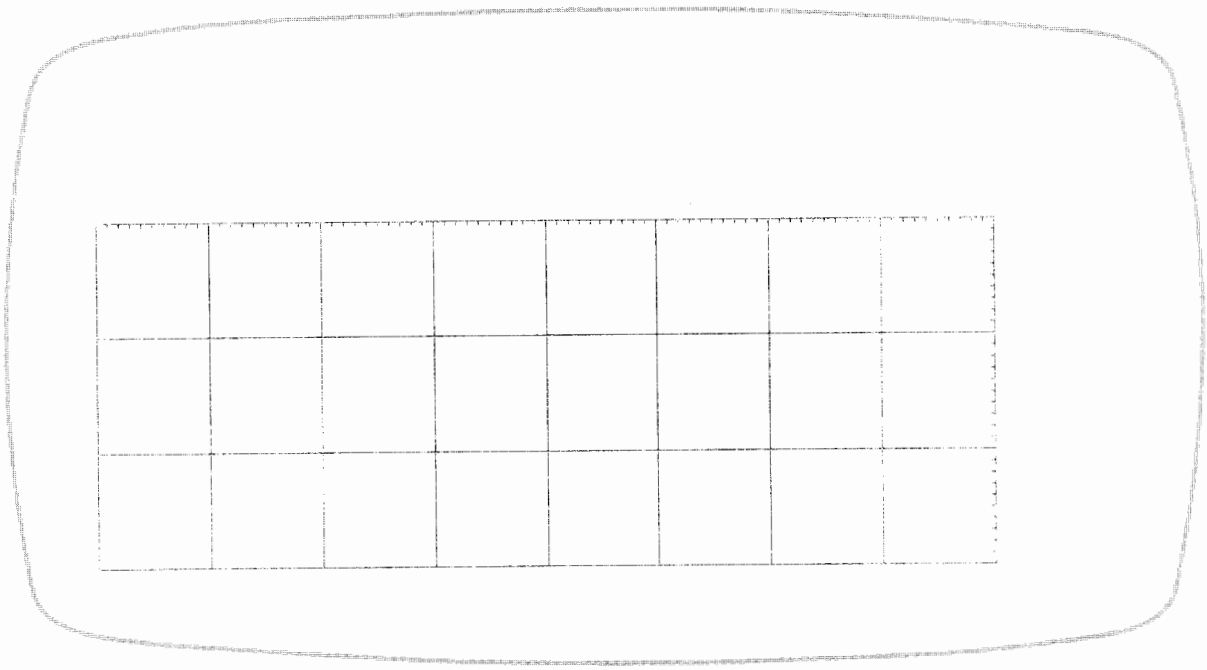
```
10 ! *** Metric Ruler ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME
50 MSCALE 10,40 !                    Specifies metric user units with
51 !                                 10mm x-offset and 40mm y-offset.
60 CLIP 0,100,0,10 !                 Clips plotting area in millimeters.
70 FRAME !                           Frames the plotting area (ruler).
80 AXES 2,10,0,10,5,10,5 !           Draws the ruler's metric scale.
90 MOVE 30,3
100 LABEL USING "K" ; "10cm METRIC SCALE"
110 END
```



The specified MSCALE origin need not be located inside the graphics limits; you can plot, for example, in negative millimeter units by specifying the origin of your MSCALE beyond the maximum x and y dimensions of the graphics limits.

Example: The following program draws a metric grid; the MSCALE origin is offset to the upper-right corner of the plotting area.

```
10 ! *** Negative Mscale ***
20 PLOTTER IS 1
30 GRAPHALL !                          Sets the display to graph-all mode.
40 LIMIT 0,160,0,60 !                  Specifies the graphics limits.
50 MSCALE 160,60 !                     Specifies metric user units and
51 !                                   places the origin at the upper-right
52 !                                   corner of the plotting area.
60 FRAME !                             Frames the plotting area.
70 GRID 2,2,0,0,10,10 !                Draws a metric grid with 10mm spacing
71 !                                   and 2mm tic marks on the x and y axes.
80 END
```



Changing Units: SETGU and SETUU

The two types of units used by the computer in plotting operations are graphics units (GUs) and user units (UUs). The *current units mode* refers to the type of units in use during plotting. At power-on, the computer is set to user units mode and the current user units scale is GUs, by default. However, you can switch modes at any time and access the current UUs and GUs scales by executing the mode change statements: SETGU and SETUU.

The SETGU statement sets the computer to graphics units mode, establishing GUs as the current scale. Executing SETGU does not disturb the current user units scale, and allows you to plot outside the plotting boundaries set by the LOCATE and CLIP statements (discussion of plotting boundaries appears later in this section). The SETGU statement is the only means by which the computer is set to graphics units mode. Unless SETGU is executed, the computer plots according to the current user units scale as defined by SCALE, SHOW, MSCALE, or by default (GUs).

```
SETGU
```

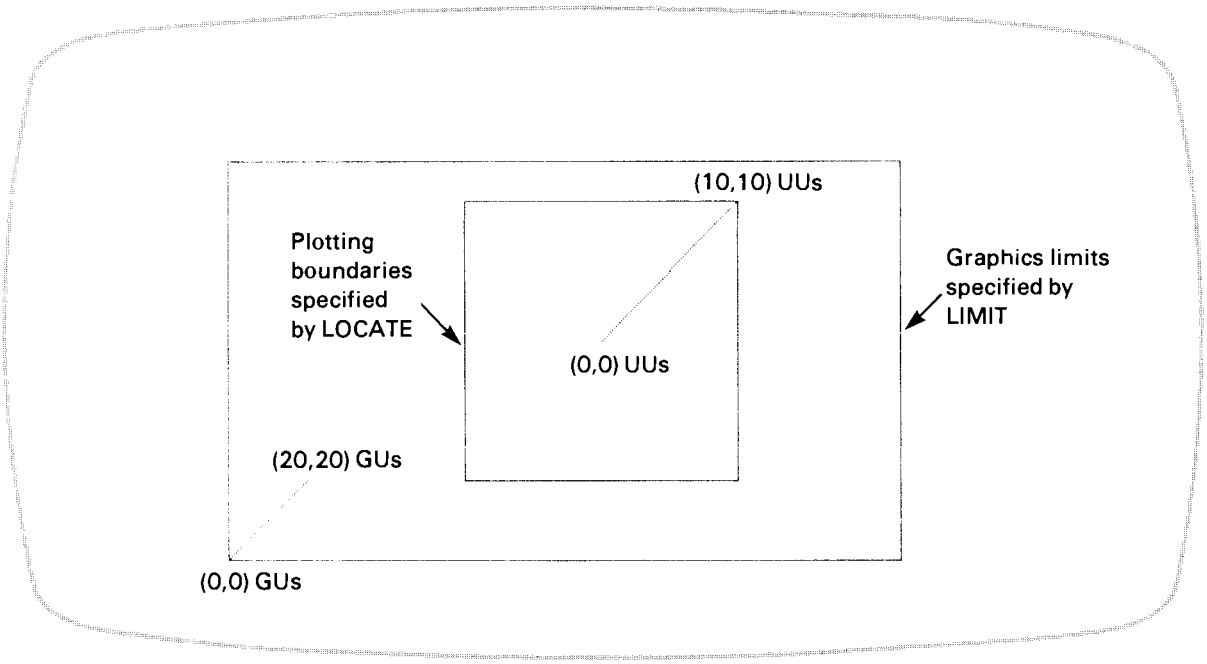
SETUU sets user units (UUs) as the current units mode. User units mode is also set by the SCALE, SHOW, MSCALE, LIMIT, and PLOTTER IS statements, and by default.

```
SETUU
```

If the computer is set to graphics units mode, you need to return the computer to user units mode in order for the plotting boundaries set by LOCATE or CLIP to be active. SETGU establishes the area bound by the graphics limits as the current plotting area.

Example: The following program makes use of both scales: UUs and GUs. The GUs scale is determined by the graphics limits, and is recalled by the SETGU statement.

```
10 ! *** Changing Units ***
20 PLOTTER IS 1
30 GCLEAR
31 ! ******* The computer is set to user units mode *******
40 LIMIT 0,120,0,70 @ FRAME !          Specifies the graphics limits
41 !                                   and frames the plotting area.
50 LOCATE 60,130,20,90 !               Locates plotting boundaries in GUs.
60 FRAME !                             Frames the plotting boundaries.
70 SCALE -10,10,-10,10 !               Scales the area enclosed by the
71 !                                   LOCATE-defined plotting boundaries.
80 MOVE 0,0 !                          Moves the pen to point (0,0) in UUs.
90 DRAW 20,20 !                        Draws only to point (10,10) in UUs.
100 SETGU !                            Sets graphics units mode.
110 MOVE 0,0 !                         Moves the pen to point (0,0) in UUs.
120 DRAW 20,20 !                       Draws to point (20,20) in GUs.
130 END
```



Once a scaling statement is executed, the current user-defined scale is active until one of the following occur:

1. A new scaling statement is executed (SCALE, SHOW, or MSCALE).
2. The computer is reset or turned off, in which case user units defaults to GUs.
3. A LIMIT or PLOTTER IS statement is executed (UUs equals GUs).
4. CRT memory is reapportioned (for example, by switching from *graph* mode to *graph-all* mode), in which case UUs equals GUs.
5. The computer is switched to graphics units mode by executing SETGU.

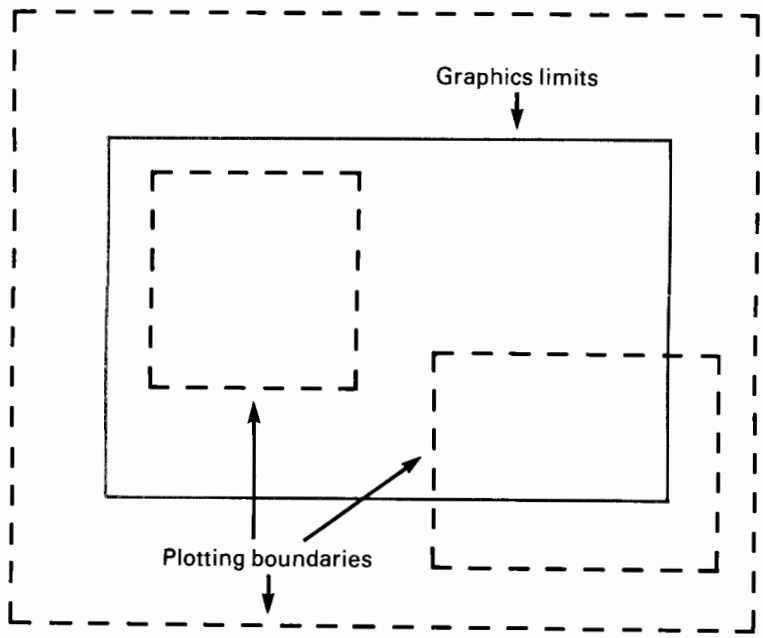
Plotting Boundaries

Plotting is restricted to the area bound by the graphics limits as specified by the LIMIT statement or by default. The LOCATE and CLIP statements specify *plotting boundaries*. Like the graphics limits plotting boundaries mark the limits of the plotting area. Plotting boundaries differ from graphics limits in that they represent conditional limits. Plotting outside the plotting boundaries is possible while the computer is set to graphics units mode or while labeling.

Plotting boundaries can be used for reserving space within the graphics limits for labeling. Plotting boundaries can also be used to create windows for showing portions of a plot.

User units can be mapped onto the LOCATE-defined plotting area but not the CLIP-defined plotting area.

The diagram below shows different ways in which plotting boundaries can be set with respect to the graphics limits. Although the plotting boundaries can extend beyond the graphics limits, or for that matter, the physical limits of the plotting device, you can't plot or label outside the graphics limits.



LOCATE Boundaries

The LOCATE statement enables you to relocate the plotting area within the graphics limits by specifying the plotting boundaries. This allows you to leave space for labels outside of the plotting area, but within the graphics limits. The parameters in the LOCATE statement are expressed in GUs. LOCATE defines the plotting boundaries as a percentage of the graphics limits.

```
LOCATE x min , x max , y min , y max
```

The first two parameters specify the left and right boundaries and the last two parameters specify the lower and upper boundaries. Like the LIMIT statement, the parameters can be exchanged to reflect the plot (refer to section 19 for an example of reflected plots). The parameters can be numbers, variables, or expressions.

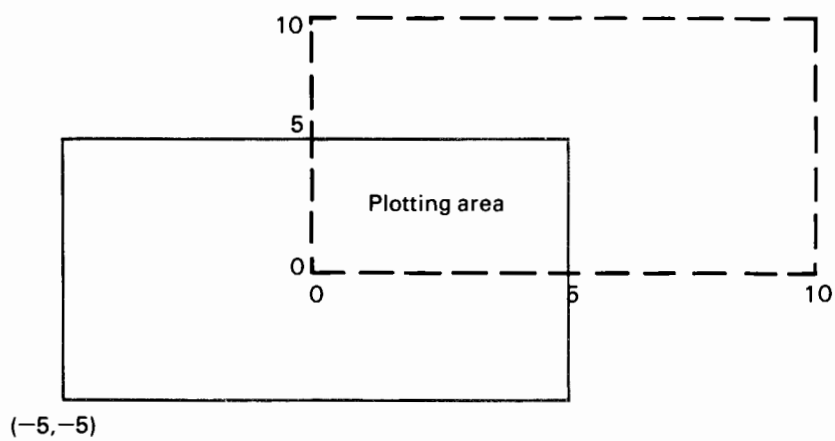
When the LOCATE statement is executed prior to a scaling statement (SCALE, SHOW, or MSCALE), the user units scale is mapped onto the area defined by LOCATE rather than the graphic limits.

The plotting boundaries specified by the LOCATE statement replace any previously defined plotting boundaries. In turn, the LOCATE-defined plotting boundaries are redefined by the CLIP statement. The LIMIT, UNCLIP, and PLOTTER IS statements default the plotting boundaries to the graphics limits. The plotting boundaries are also set to the graphics limits whenever CRT memory is reapportioned or the computer is turned on or reset. When the computer is set to graphics units mode by executing SETGU, the graphics limits define the current plotting area. Executing SETUU restores the LOCATE or CLIP-defined plotting boundaries.

Examples: The drawings below show the available plotting area and the current scale in user units mode and graphics units mode for the given LIMIT, LOCATE, and SCALE statements. Labeling is allowed anywhere within the graphics limits, regardless of the current units mode. The graphics limits are drawn in solid lines; the plotting boundaries are drawn in dotted lines. The plotting area is the shaded portion.

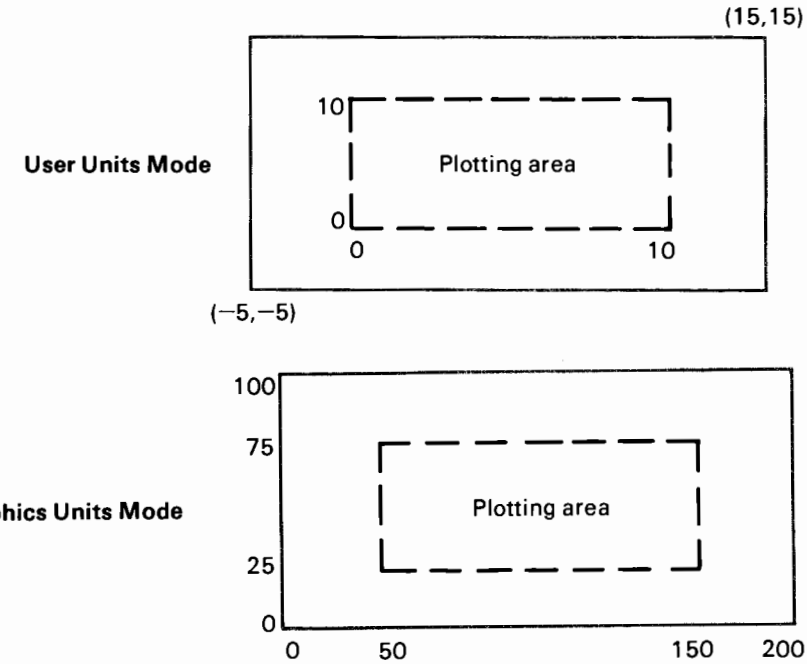
```
LIMIT 0,120,0,60
LOCATE 100,300,50,150
SCALE 0,10,0,10
```

User Units Mode

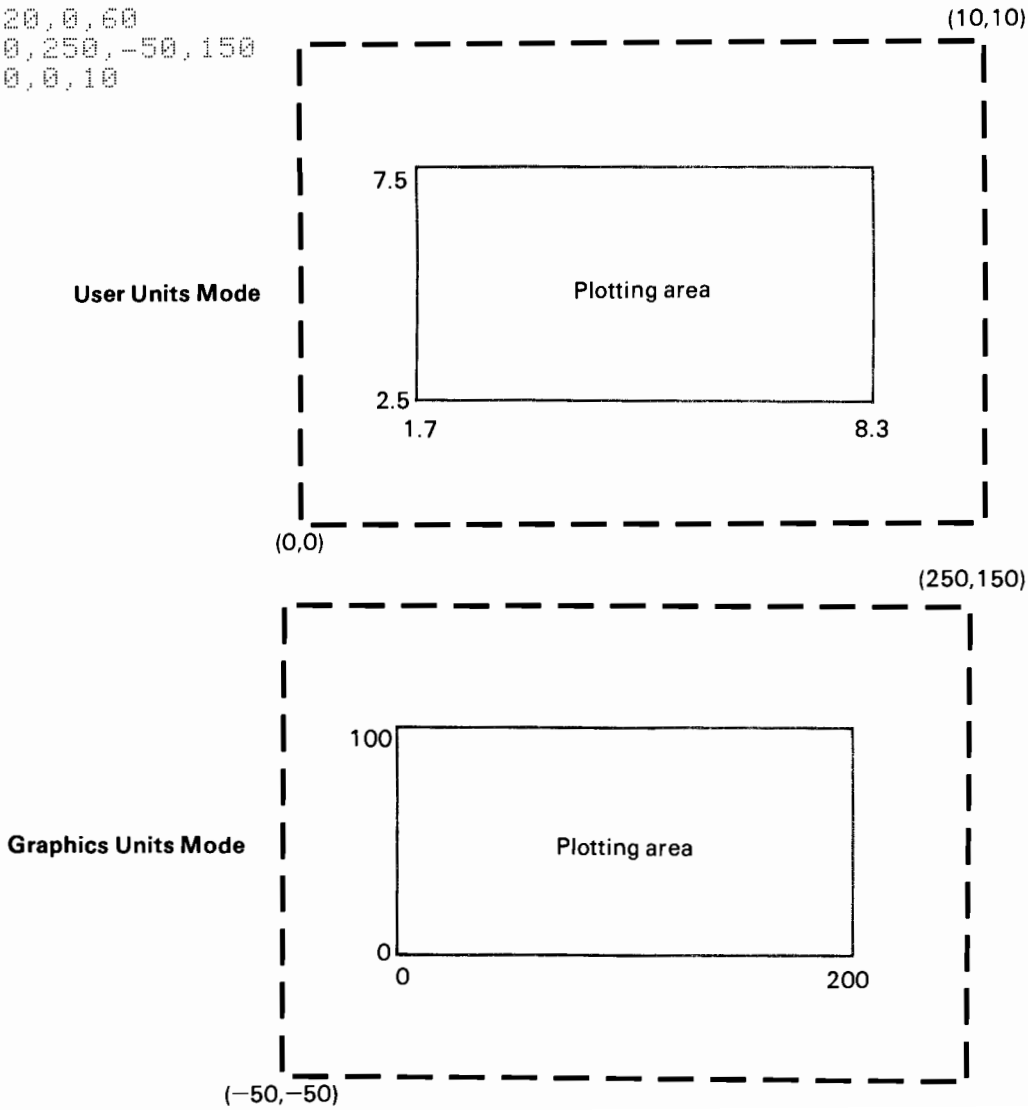


Graphics Units Mode

```
LIMIT 0,120,0,60
LOCATE 50,150,25,75
SCALE 0,10,0,10
```

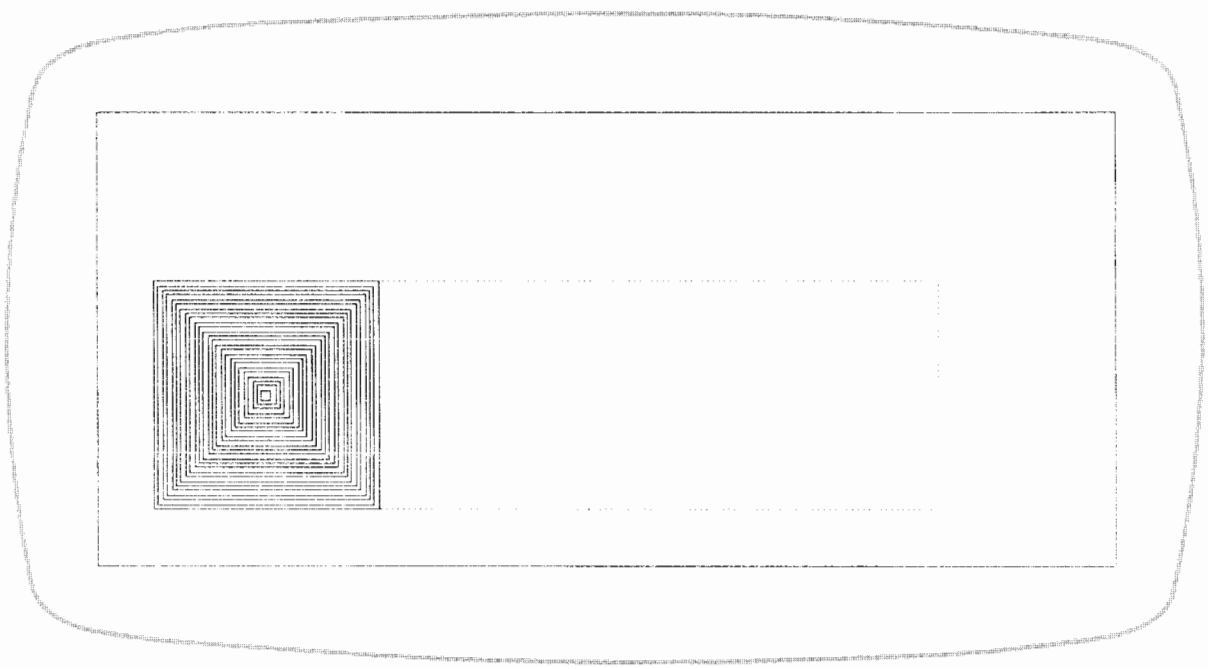



```
LIMIT 0,120,0,60
LOCATE -50,250,-50,150
SCALE 0,10,0,10
```



Example: The following program sequentially frames the default graphics limits, the graphics limits specified by a LIMIT statement, and the LOCATE-specified plotting boundaries.

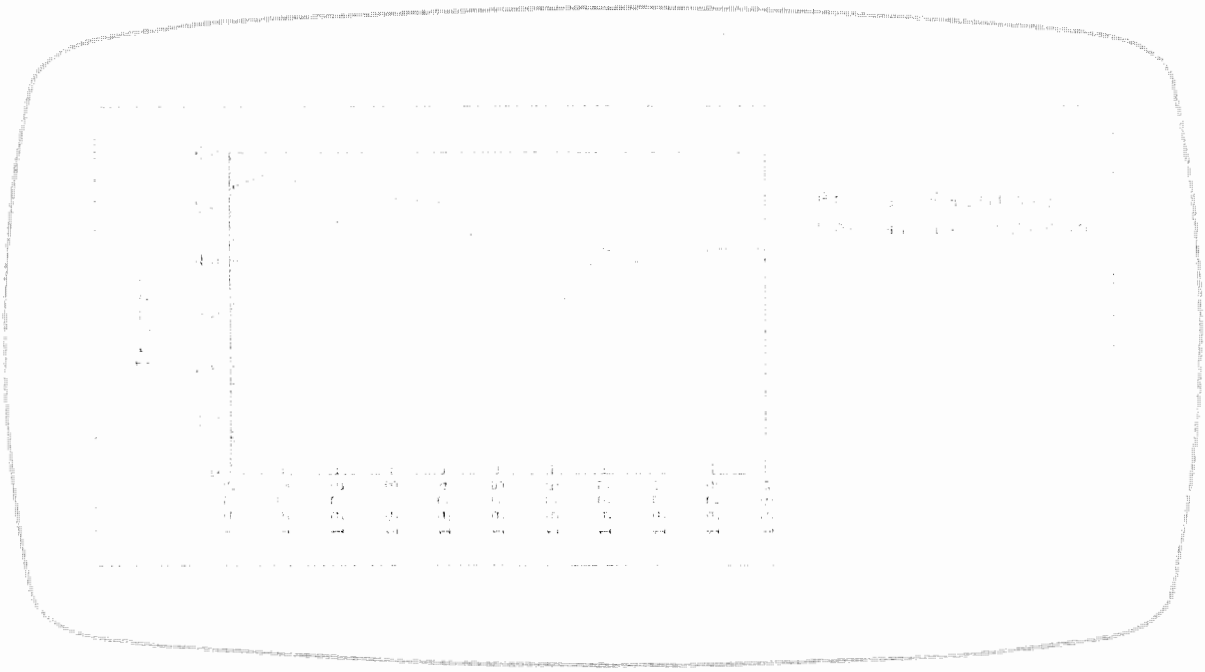
```
10 ! *** Locate ***
20 PLOTTER IS 1
30 GRAPHALL
40 FRAME !                                  Frames the default graphics limits.
50 LIMIT 10,150,10,50
60 LINE TYPE 3,2 !                          Specifies a dotted line type.
70 FRAME !                                  Frames the specified graphics limits.
80 FOR I=0 TO 50 STEP 2
90 LOCATE 50-I,50+I,50-I,50+I !             Specifies plotting boundaries in
91 !                                        increments of 2 GUs.
100 LINE TYPE 1 !                           Specifies a solid line type.
110 FRAME
120 NEXT I
130 END
```



The LOCATE statement is particularly useful for allowing space on the plot for labels.

Example:

```
10 ! *** Oregon Rainfall ***
20 PLOTTER IS 1
30 GRAPHALL !                    Sets the display to graph-all mode.
40 FRAME
50 LOCATE 30,150,20,90 !         Specifies plotting boundaries; space is
51 !                             reserved within the graphics limits for labels.
60 FRAME
70 SCALE 1970,1980,0,60 !        Scales the LOCATE-defined plotting area.
80 LAXES 1,1,1970,0,1,10,4 !     Draws and labels the x and y axes.
90 MOVE 1970,0
100 FOR Yr=1970 TO 1980
110 READ Rn
120 PLOT Yr,Rn !                 Plots rainfall data.
130 NEXT Yr
140 DATA 53.1,57.2,47.1,51.4,50.8,39.3,29.7,41.9,37.6,42.1,42.1
150 MOVE 1981,50
160 CSIZE 6,.6
161 ! ******* Labeling *******
170 LABEL USING 180 ; "Annual Rainfall","Corvallis, Oregon"
180 IMAGE 20A,/
190 MOVE 1968.5,20
200 DEG @ LDIR 90
210 LABEL "Inches"
220 END
```

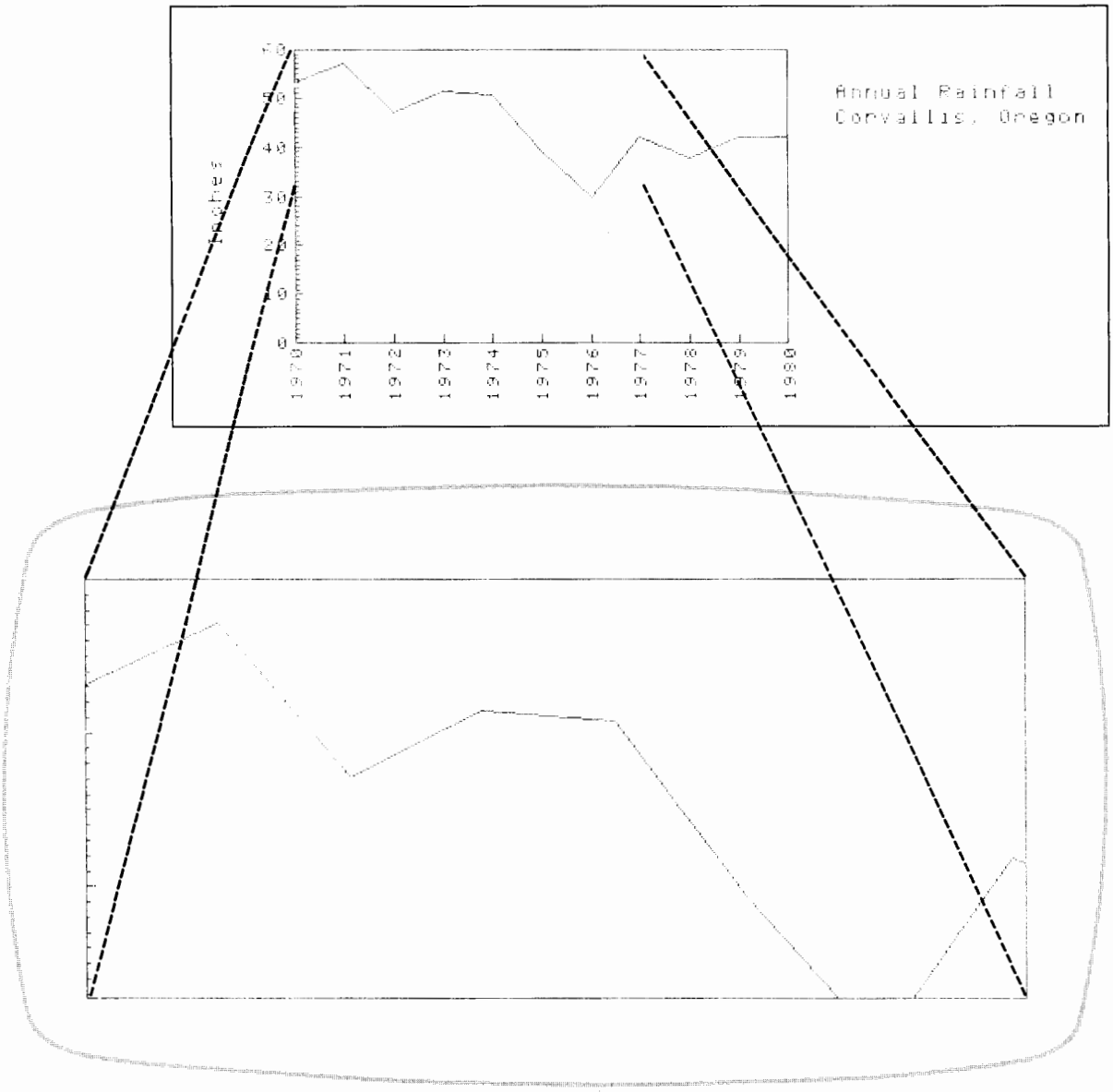


The LOCATE statement can specify an area larger than the graphics limits, in which case the graphics limits define a plotting area window inside the plotting boundaries.

Example: Change the LOCATE statement in the above example to the following, and run the program once again.

```
50 LOCATE 0,320,-120,100
```

The graph is proportionately enlarged so that only the upper-left corner is plotted.



CLIP Boundaries

The CLIP statement specifies the plotting boundaries according to the current units. Previously set plotting boundaries are replaced by the CLIP-defined boundaries. Plotting boundaries set by LOCATE or CLIP affect lines plotted in user units mode, but have no effect on lines plotted in graphics units mode or labels.

CLIP *x min* , *x max* , *y min* , *y max*

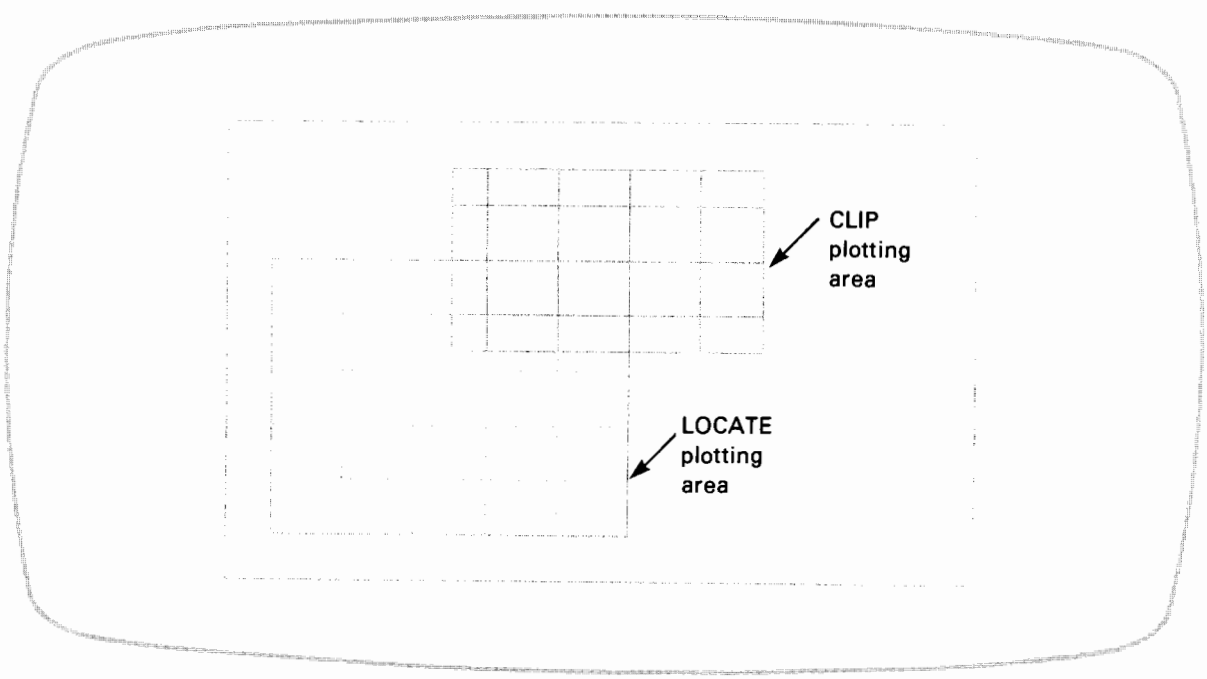
The first two parameters specify the left and right plotting boundaries and the second two parameters specify the lower and upper plotting boundaries. The parameters can be numbers, variables, or expressions.

The CLIP parameters are interpreted according to the current units, in contrast to the LOCATE statement (LOCATE parameters are in GUs). The plotting area defined by the CLIP statement cannot be scaled by any of the three scaling statements: SCALE, SHOW, and MSCALE. If a scaling statement is executed after the CLIP statement, the user units scale is mapped onto the current plotting area as defined by the graphics limits or if specified, the LOCATE plotting boundaries.

The graphics units scale is mapped onto the plotting area defined by the graphics limits.

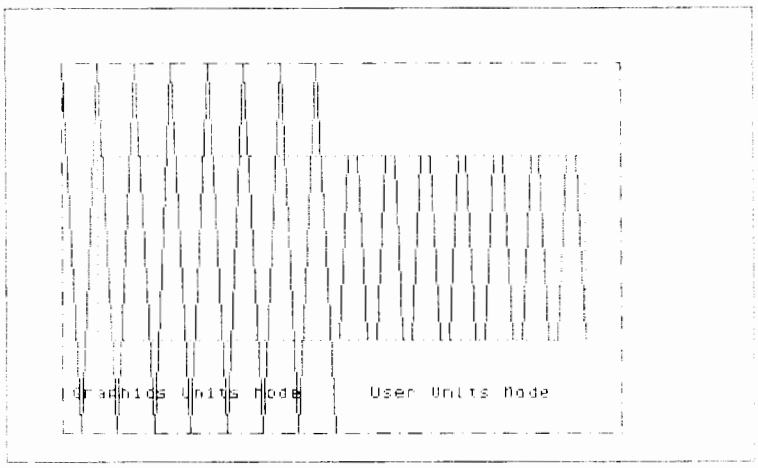
Example:

```
10 ! *** Clip ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME !                          Frames the default plotting area.
50 LOCATE 10,90,10,70 !             Locates the plotting boundaries.
60 FRAME !                          Frames the LOCATE plotting area.
70 CLIP 50,120,50,90 !              Specifies new CLIP plotting boundaries.
80 FRAME !                          Frames the CLIP plotting area.
90 SCALE 0,5,0,5 !                  Scales the LOCATE plotting area.
100 GRID 1,1 !                      Draws a grid within the CLIP-defined
101 !                               plotting area according to the scale
102 !                               mapped onto the LOCATE-defined area.
110 LOCATE 10,90,10,70 !            Returns plotting boundaries to original
111 !                               LOCATE-defined position.
120 LINE TYPE 3 !                   Specifies a dotted line type.
130 GRID 1,1 !                      Draws a grid on the LOCATE plotting area.
140 END
```



Example: The following program uses the CLIP statement to specify plotting boundaries and demonstrates plotting in graphics units mode and user units mode.

```
10 ! *** Clip-Plot ***
20 GCLEAR
30 PLOTTER IS 1 @ FRAME
50 LIMIT 10,110,5,70 !                          Specifies the graphics limits.
60 LINE TYPE 6 @ FRAME
70 CLIP 10,RATIO *100-10,25,75 !                Specifies plotting boundaries in
71 !                                            GUs- the current user units scale.
80 LINE TYPE 3 @ FRAME
90 LINE TYPE 1
100 MOVE 0,100
110 FOR X=5 TO RATIO *100 STEP 5
120 IF X<50*RATIO  THEN SETGU ELSE SETUU !      Sets graphics units mode for left
121 !                                           half of plot, user units mode for
122 !                                           right half of plot. Plotting scale
123 !                                           is GUs for both modes.
130 IF (-1)^(X/5)=1 THEN Y=100 ELSE Y=0
140 PLOT X,Y,-1
150 NEXT X
151 ! ******* Labeling is not restricted by the plotting boundaries *******
160 MOVE 3,10 @ LABEL "Graphics Units Mode"
170 MOVE 85,10 @ LABEL "User Units Mode"
180 END
```



Unclipping Plotting Boundaries

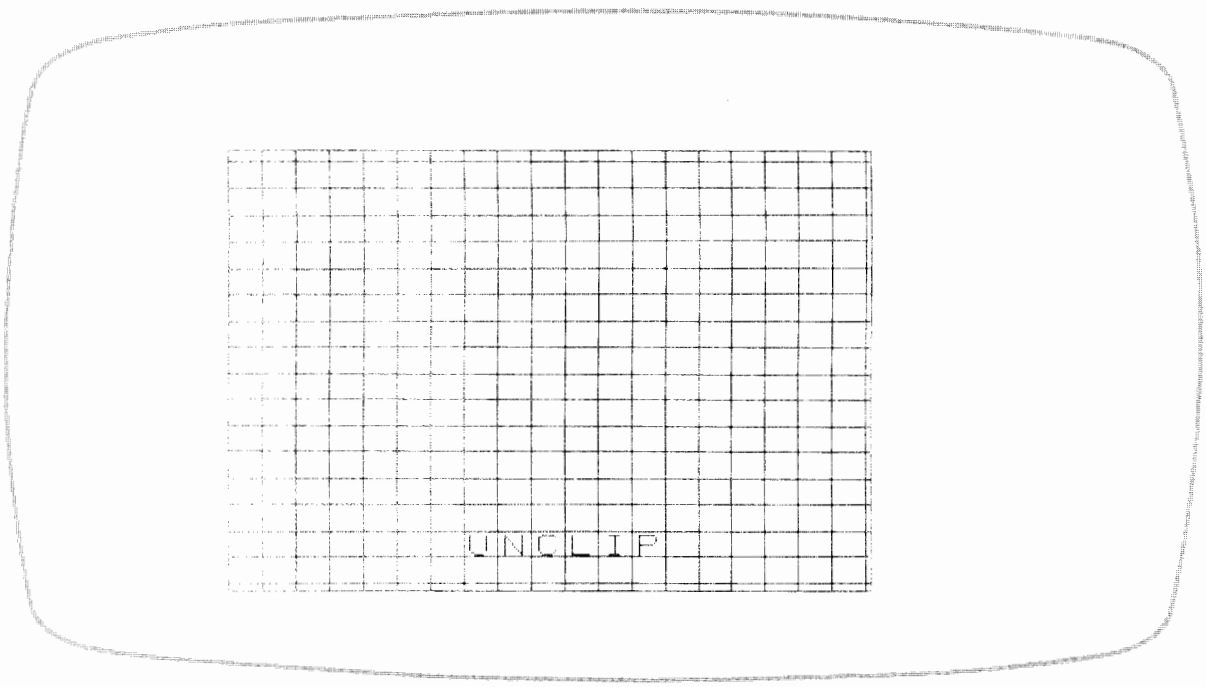
The UNCLIP statement sets the plotting boundaries equal to the graphics limits, establishing the area bound by the graphics limits as the current plotting area.

```
UNCLIP
```

UNCLIP doesn't disturb the current units (the computer remains in the current scaling units mode). The SETGU statement also establishes the area within the graphics limits as the current plotting area, but without resetting the plotting boundaries. SETGU sets the computer to graphics units mode.

Example: The UNCLIP statement is used in the following program to reset the LOCATE-specified plotting boundaries to the graphics limits. The user units scale is preserved.

```
10 ! *** Unclip ***
20 PLOTTER IS 1
30 GCLEAR
40 LIMIT 0,115,0,75 @ FRAME !  Specifies and frames the graphics limits.
50 LOCATE 40,120,20,80 @ FRAME !  Locates and frames the plotting boundaries.
60 SCALE 0,10,0,10 !           Scales the LOCATE plotting area.
70 GRID 1,1 !                  Draws a grid on the LOCATE area.
80 CSIZE 9,.9 !                Specifies character size.
90 MOVE 2.2,-1.9 !             Moves the pen outside the plotting boundaries.
100 LABEL "UNCLIP" !           Labels the character string "UNCLIP".
110 UNCLIP !                   Sets the plotting boundaries equal to the
111 !                          graphics limits.
120 GRID 1,1 !                 Draws a grid on the plotting area bound by the
121 !                          graphics limits; UUs are unchanged.
130 END
```



The following table summarizes the statements and conditions which affect the position and scale of the plotting area.

Summary of Positioning and Scaling Statements

Condition or Statement	Parameter Units	Effect on Scaling Mode	Effect on Scaling Units	Effect on Graphics Limits	Effect on Plotting Boundaries
Power-on (RESET) CRT Memory Reapportioned		Sets to user units mode.	UUs equals GUs.	Sets to default graphics limits.	Sets to default graphics limits.
PLOTTER IS		Sets to user units mode.	UUs equals GUs.	Reads limits from pen plotter. PLOTTER IS 1 resets maximum CRT limits.	Sets to graphics limits.
LIMIT	mm (for plotters and HP-87 CRT)	Sets to user units mode.	UUs equals GUs.	Sets to graphics limits specified by LIMIT.	Sets to graphics limits specified by LIMIT.
LOCATE	GUs	No effect.	No effect.	No effect.	Sets to boundaries specified by LOCATE.
CLIP	current units	No effect.	No effect.	No effect.	Sets to boundaries specified by CLIP.
UNCLIP		No effect.	No effect.	No effect.	Resets to current graphics limits.
SCALE	UUs	Sets to user units mode.	UUs specified by SCALE.	No effect.	No effect.
SHOW	UUs (for plotters and HP-87 CRT)	Sets to user units mode.	UUs specified by SHOW in equal units.	No effect.	No effect.
MSCALE	mm (for plotters and HP-87 CRT)	Sets to user units mode.	UUs specified by MSCALE in millimeters.	No effect.	No effect.
SETGU		Sets to graphics units mode. Plotting area is defined by graphics limits.	GUs.	No effect.	No effect.
SETUU		Sets to user units mode. Plotting area is defined by plotting boundaries.	Current UUs as specified by the above statements and conditions.	No effect.	No effect.

Section 16

Plotting Data

The plotting capabilities of your computer can be grouped into four categories: plotting data (lines, curves, or individual points), drawing axes, labeling, and byte plotting (a type of CRT plotting whereby individual dots are addressed).

By now, you should be familiar with the procedures for setting up and scaling the plotting area. This section discusses the procedures for selecting pen color and line type, controlling the pen's movement, and plotting data.

Pen Color

The term "pen" refers to the ink pen on a pen plotter or to the analogous electronic pen in a CRT device. The two types of pens are controlled with the same statements, although the physical operations are quite different.

On CRT devices, the PEN statement specifies white dots or black dots for plotting on a black or white background. The PEN statement enables you to draw lines, points, or labels in one color (black or white) and erase them by plotting over in the opposite color.

PEN *pen number*

The pen number parameter can be a number, variable, or expression.

For CRT plotting, the pen number parameter is interpreted as follows:

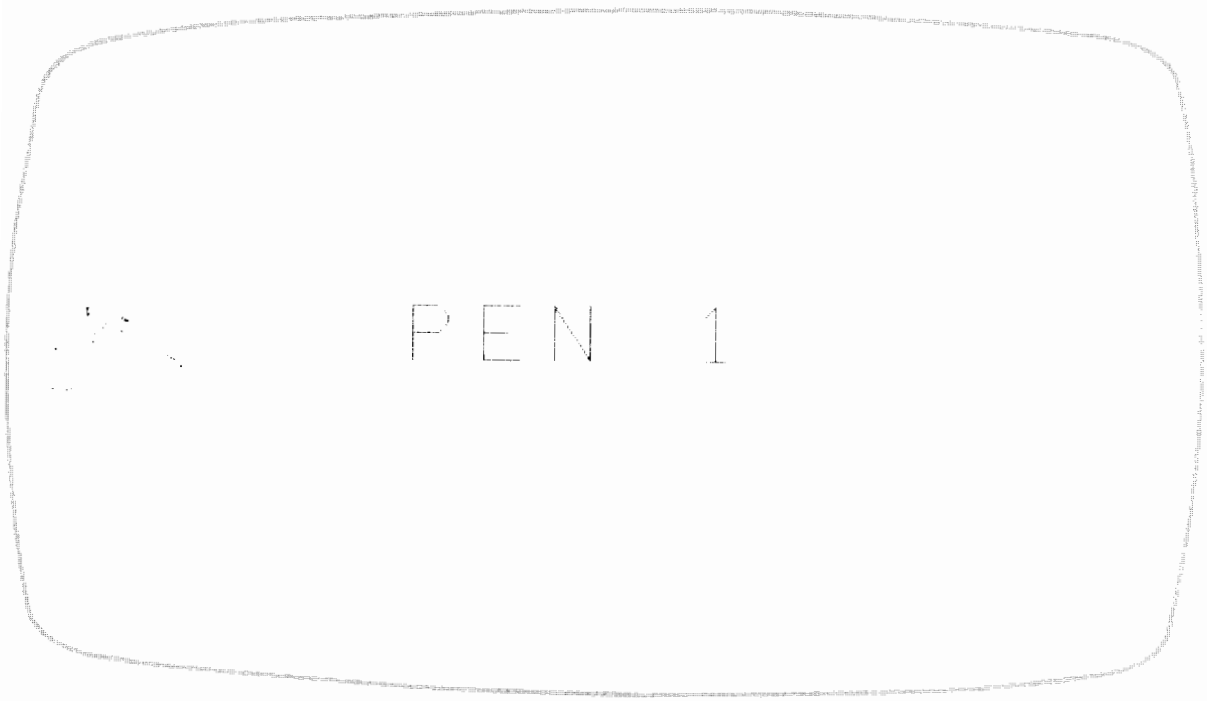
Statement	CRT Pen Operation
PEN 1	Plots white dots.
PEN 0	Pen is deactivated, does not plot.
PEN -1	Plots black dots.
PEN -2	Performs an exclusive or: plots white dots over black dots and black dots over white dots.

Non-integer pen numbers are rounded to the nearest integer; for example PEN .7 is equivalent to PEN 1. The CRT display interprets positive rounded pen numbers (except PEN 0) as equal to PEN 1 and negative rounded pen numbers (except PEN -2) as equal to PEN -1.

The GCLEAR statement clears the graphics display to the current background color (PEN 1 has a black background color, PEN -1 and PEN -2 have a white background color). If you change pen color, execute the GCLEAR statement before plotting so that the new background color is displayed.

Example: The following program demonstrates PEN 1 and PEN -1 plotting colors. The display is cleared and labeled according to the current pen number. At the end of the program, the "PEN -1" label is erased by switching to PEN 1 and writing over the label.

```
10 ! *** Pen Color ***
20 PLOTTER IS 1
30 FOR I=2 TO 5
40 PEN (-1)^I !                    Specifies pen number according to loop counter.
50 GCLEAR !                        Clears background to the current pen color.
60 MOVE 40,50
70 CSIZE 24
80 LABEL "PEN",(-1)^I !            Labels PEN # in current color.
90 WAIT 3000
100 NEXT I
110 PEN 1 !                        Specifies PEN 1.
120 MOVE 40,50
130 LABEL "PEN-1" !                Erases label using PEN 1.
140 END
```



Example: Plotting with PEN -2 performs an exclusive or on individual CRT dots (turns on black dots, turns off white dots). This allows you to erase lines without changing pen number.

```
10 ! *** PEN -2 ***
20 PLOTTER IS 1
30 GCLEAR
40 PEN -2 !                        Specifies pen number -2.
50 FRAME !                         Frames plotting area using exclusive or (PEN -2).
60 MOVE 30,50 !                    Moves the pen to the specified coordinate.
70 CSIZE 26 !                      Specifies character size 26.
80 LABEL "ERASE" !                 Labels character string.
90 GOTO 50 !                       Repeats framing operation.
100 END
```

Note: Characters in a label are drawn using overlapping line segments. For this reason, characters drawn with PEN -2 are missing dots at corners and other points and lines of intersection.

The default pen status at power-on or after pressing (RESET) is PEN 1 (white dots, black background). When CRT memory is reapportioned or when the PLOTTER IS statement is executed, the pen color defaults to PEN 1.

On a multiple-pen plotter, the PEN statement enables you to select different colored pens for plotting. PEN 0 returns the pen to the stall. Non-integer pen numbers are rounded to the nearest integer. Negative pen numbers are interpreted as PEN 0. With plotters that require manual pen changes, the PEN statement is ignored. Consult the documentation for your pen plotter to obtain information regarding pen changes.

Specifying the Line Type

The LINE TYPE statement selects one of several solid or dashed line types for drawing lines, curves, and axes. Labels are unaffected by the LINE TYPE statement; all labels are drawn with solid lines.

LINE TYPE *type number* [, *length*]

The line type number and length parameter can be numbers, variables, or expressions. There are 8 line types available for CRT graphics, as denoted by the 8 line type numbers, integers 1 through 8. Other line type numbers are interpreted for the CRT as follows:

- Non-integer line type numbers are rounded to the nearest integer.
- LINE TYPE 0 is equivalent to LINE TYPE 1.
- Line type numbers larger than 8 are converted MOD 9 to numbers between 0 and 8.
- Negative line type numbers are converted MOD -9 to numbers between 0 and 8.
- Line type numbers with absolute value greater than or equal to 65535 specify LINE TYPE 6.

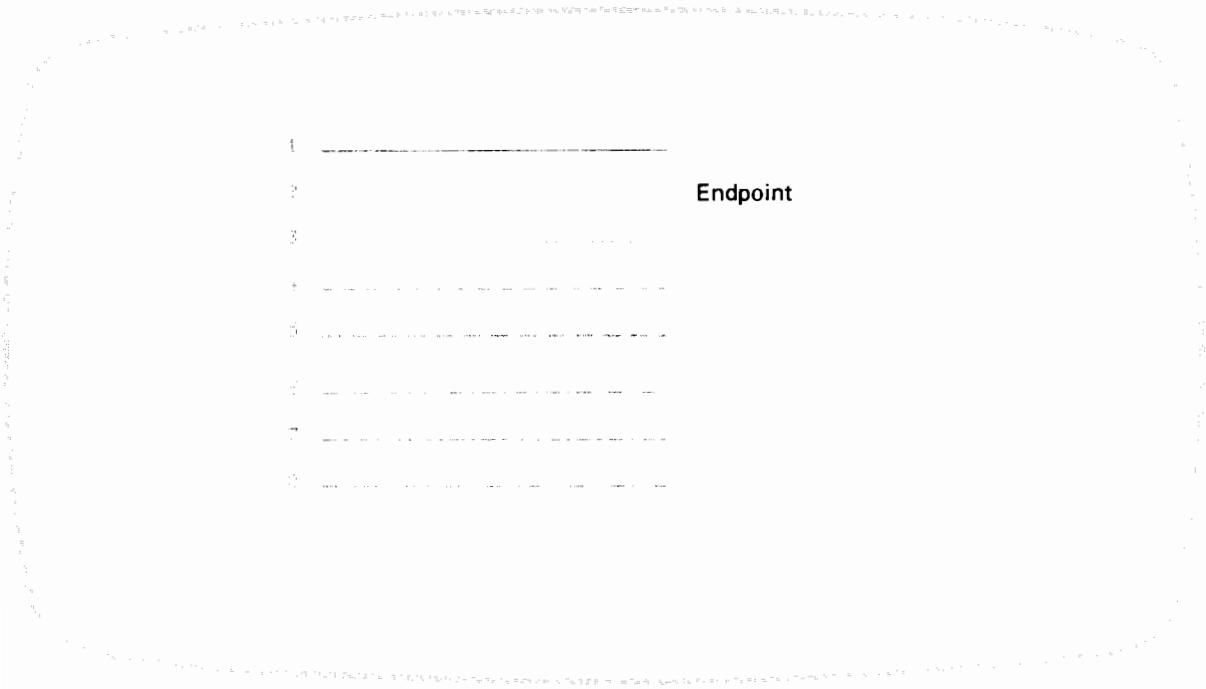
For pen plotters the rounded line type number is preserved (not converted by MOD 9). Line type numbers outside of the range 1 through 8 are interpreted according to the capabilities of the plotter. Line type numbers 1 through 8 produce the same line types as for CRT graphics. Refer to the documentation for your pen plotter for more information on the available line types.

The optional length parameter specifies the length of the repeat pattern for the line type selected. The length parameter is specified in GUs. The repeat pattern for each line type is the shaded portion of the line shown in the following example. If a length parameter is not specified, the length defaults to four GUs for pen plotters. The CRT default length varies for different line types, and ranges from approximately 2 GUs to 10 GUs.

The default line type is solid (LINE TYPE 1).

Example: The following program illustrates the line type patterns available for plotting lines, curves, and axes.

```
10 ! *** Line Type ***
20 PLOTTER IS 1
25 GCLEAR
30 LOCATE 10,120,10,100
40 SCALE 0,10,9,0
50 FOR I=1 TO 8 !                    Loop draws the 8 line types.
60 MOVE 0,I !                        Moves the pen to the specified point.
70 LABEL I !                         Labels loop counter.
80 LINE TYPE I !                     Specifies line type equal to loop counter.
90 MOVE 1,I !                        Moves the pen to the specified point.
100 DRAW 8,I !                       Draws a line using current line type.
110 NEXT I !                         End loop.
120 END
```



Lifting the Pen

The PENUP statement lifts the pen from the plotting surface. This stops line drawing on the CRT and other plotting devices until pen control is changed by another statement.

```
PENUP
```

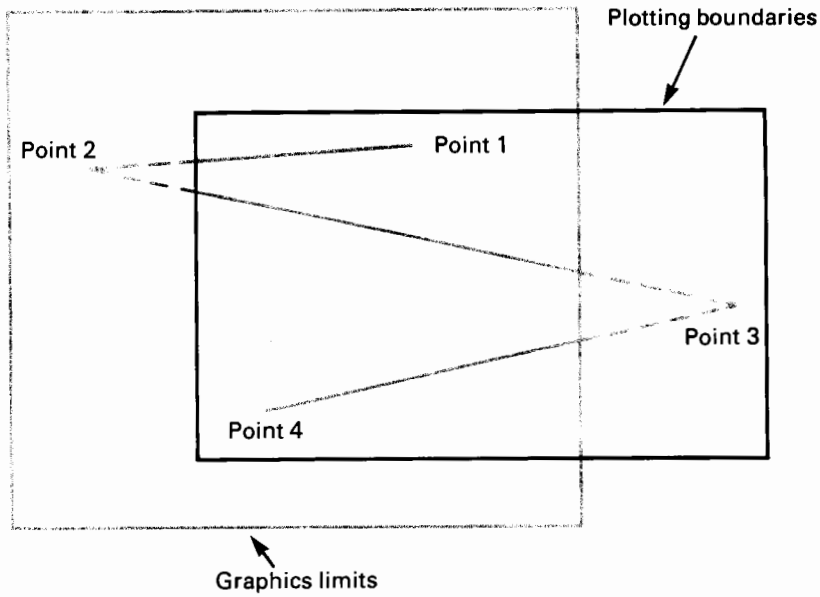
Another means of lifting the pen is provided by three plotting statements: PLOT, IPLOT (*incremental plot*), and RPLOT (*relative plot*). Each of these statements allow you to lift or lower the pen while plotting data. They are discussed later in this section.

Plotting Data

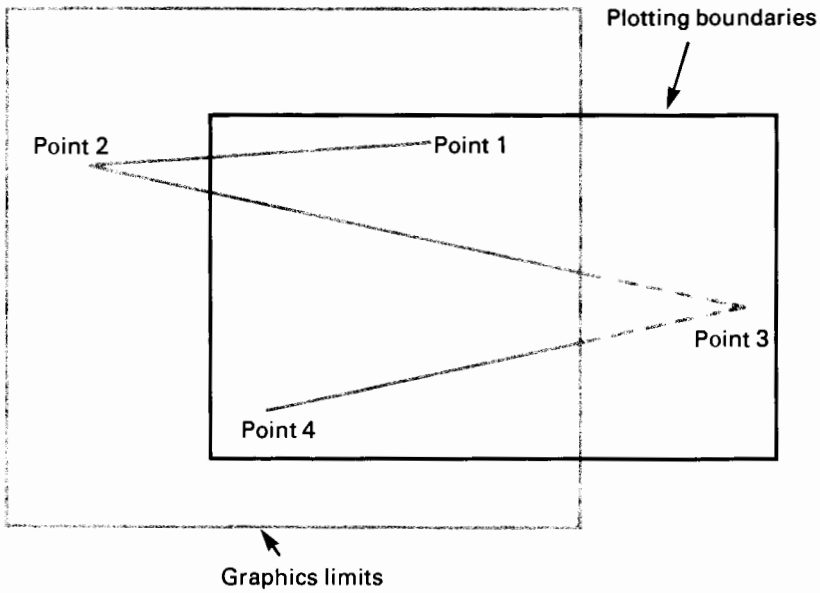
The statements described below are used for positioning the pen and plotting data (lines, points, and curves). Data plotted in user units mode cannot extend beyond the plotting boundaries. Data plotted in graphics units mode can extend beyond the plotting boundaries, but not beyond the graphics limits. All of the plotting statements can be used to position labels outside the plotting boundaries (within the graphics limits) in either mode. Data is plotted according to the current pen number and linetype.

The figures below show the clipping action of plotting boundaries and graphics limits when drawing a line from point 1, to point 2, to point 3, to point 4. The dashed line segment is the clipped portion of the line—this segment is not drawn, but the computer interprets the final pen position as if the pen were moved to the specified point.

Plotting in User Units Mode



Plotting in Graphics Units Mode



The restrictions imposed by the plotting boundaries and graphics limits do not affect the logical position of the pen. For example, if the computer is instructed to draw a line which extends outside of the graphics limits (point 2 to point 3 in the above example), it draws a line to the edge of the plotting area and interprets the final pen position as the point specified in the plotting statement. The pen position is set by the plotting statement according to the current scale, even when the specified point lies outside of the current plotting area.

Pen Control

Three plotting statements provide optional pen control: PLOT, IPLOT, and RPLOT. The optional pen control parameter enables you to control the pen's up or down position and whether the up or down pen change is made before or after the pen moves. The pen control parameter for each of these statements is interpreted as follows:

Pen Control Parameter	Pen Control
Odd	Lowers the pen (pen down).
Even	Lifts the pen (pen up).
Positive	Pen is lifted or lowered after motion.
Negative	Pen is lifted or lowered before motion.

The parity (odd or even) of the parameter determines whether the pen is dropped or lifted. The sign determines when this operation is executed, before or after plotting.

Examples:

Pen Control Parameter	Pen Action
+1 or no parameter	Move or draw, then lower pen.
2 or 0	Move or draw, then lift pen.
−2	Lift pen, then move.
−1	Lower pen, then draw.

Other plotting statements automatically control the pen's up or down status (for example, the DRAW statement drops the pen). Refer to the Pen Status table in appendix D for more information.

The PLOT Statement

The PLOT statement plots data according to the current units mode (user units mode or graphics units mode) and provides for optional pen control.

The *x* and *y* coordinates specify the point to which the pen moves or draws depending on the up or down status of the pen.

```
PLOT x-coordinate , y-coordinate [ , pen control]
```

The *x* and *y* coordinates are interpreted according to the current units mode. The *x* and *y* coordinates and the pen control parameter can be numbers, variables, or expressions.

Example:

```
PLOT 10,-55,-1
```

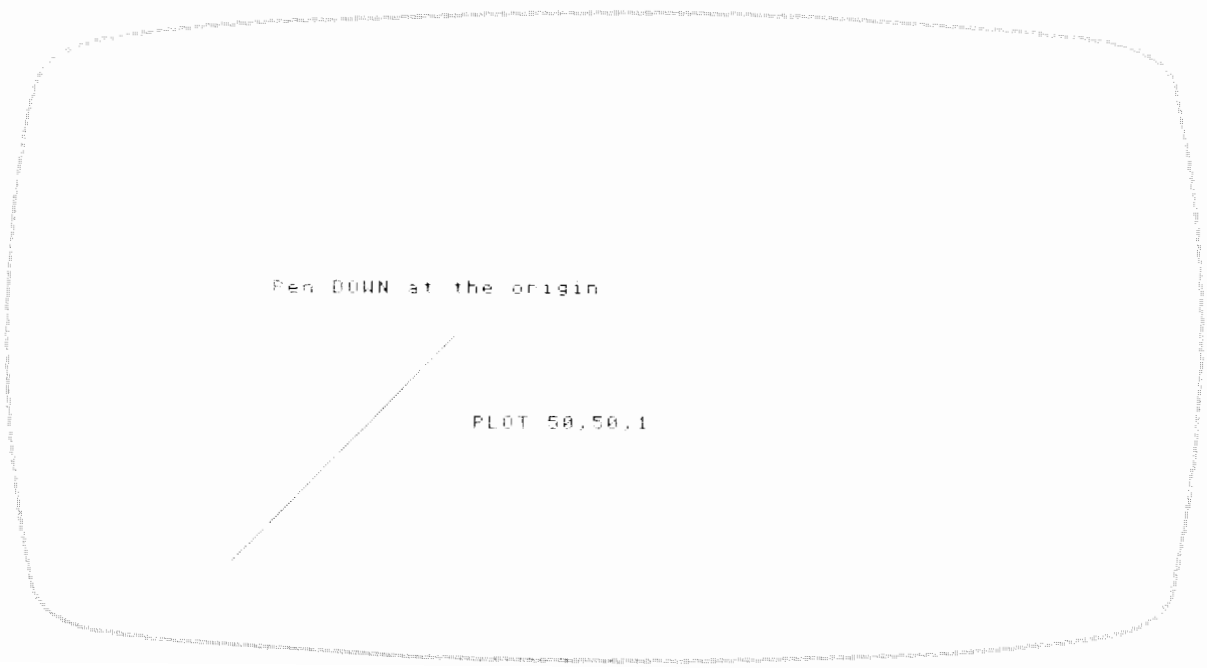
Lowers the pen and draws a line from the current pen postion to the point (10,−55) according to the current scale.

The optional pen control parameter specifies the up or down pen movement. If not specified, pen control defaults to +1. If the pen control parameter is positive, the position of the pen (down or up) preceding execution of PLOT determines whether a line is drawn to the specified point (x,y) or if the pen is moved without drawing a line.

Example: The following program demonstrates the use of the pen control parameter in the PLOT statement. Choose whether the pen is up or down at the origin, then try different pen control parameters to see how they affect the PLOT statement. The program pauses after each trial; press (CONT) to continue the program and specify a new pen control parameter.

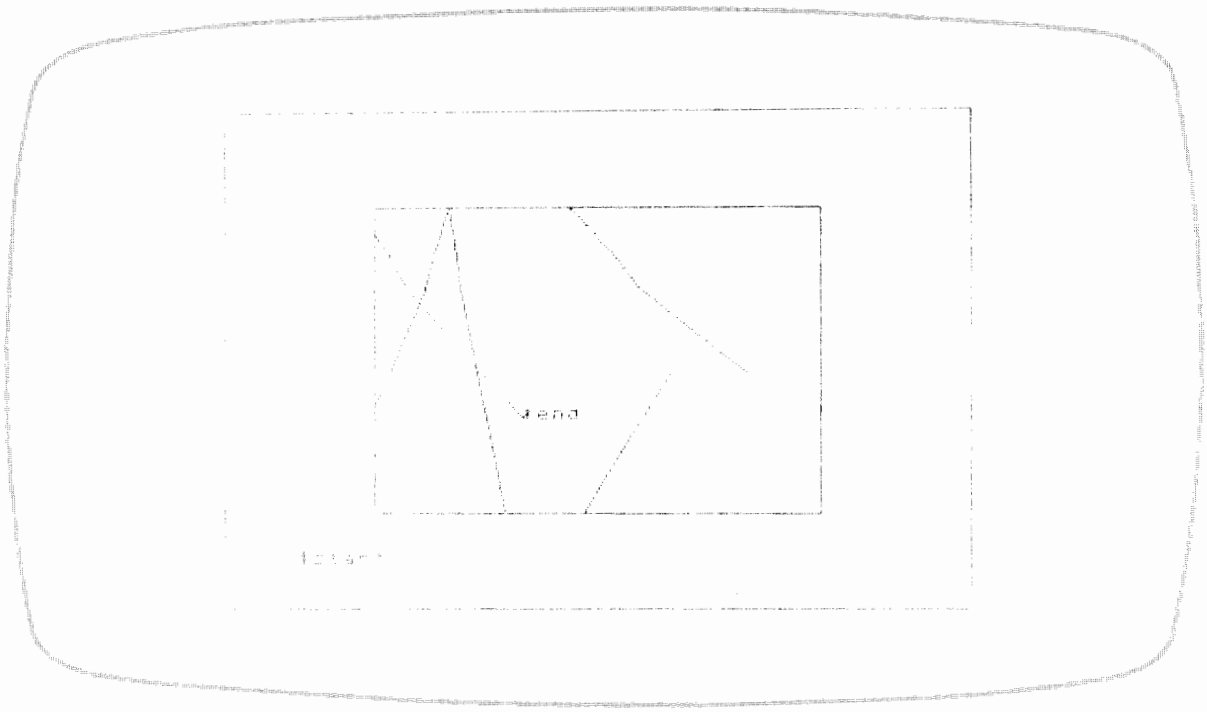
```
10 ! *** Pen Control ***
20 PLOTTER IS 1
30 GCLEAR @ CLEAR
40 DISP "Pen UP or DOWN at origin?" !        Prompts for pen control at origin.
50 INPUT A$
60 IF A$="UP" THEN OP=2 ELSE OP=1
70 DISP "Enter the pen control parameter." !   Prompts for pen control used in
71 !                                              the example PLOT 50,50,P.
80 INPUT P
90 MOVE 10,60 !                              Positons pen for labeling.
100 LABEL USING "K" ; "Pen ",A$," at the origin"
110 MOVE 0,0 !                               Moves the pen to the origin.
120 PLOT 0,0,OP !                            Plots 0,0 using pen control OP.
130 PLOT 50,50,P !                           Plots 50,50 using pen control P.
140 MOVE 55,30 !                             Positions pen for labeling.
150 LABEL USING "K" ; "PLOT 50,50,",P
160 PAUSE !                                  Pauses program.
170 GCLEAR
180 GOTO 30 !                                Repeats pen control routine.
190 END
```

```
Pen UP or DOWN at origin?
?
DOWN
Enter the pen control parameter.
?
1
```



Example: The PLOT statement is used to plot lines, move the pen, and position labels in the following program.

```
10 ! *** Plot ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME
50 SCALE 0,10,0,10 !                    Specifies user units scale.
60 CLIP 2,8,2,8 !                       Defines plotting boundaries in user units.
70 FRAME !                              Frames the plotting boundaries.
80 FOR Pt=1 TO 7
90 READ X,Y,P !                         Reads X,Y coordinates and pen control
100 !                                   parameter from the DATA statement.
110 PLOT X,Y,P !                        Plots X,Y using pen control P.
120 IF Pt=1 THEN LABEL "*start" !       Labels first point.
130 NEXT Pt
140 DATA 1,1,-2,3,8,-1,4,0,1,6,5,1,7,5,-2,-5,20,-1,4,4,1
150 LABEL "*end" !                      Labels last point.
160 END
```



Incremental Plotting

The IPLOT statement provides incremental plotting capability with pen control. IPLOT interprets the *x* and *y* parameters as increments, in contrast to the PLOT statement which uses absolute coordinates. The size of the *x* and *y* increments is determined by the current scale.

IPLOT *x-increment* , *y-increment* [, *pen control*]

The *x* and *y* increments and the pen control parameter can be numbers, variables, or expressions.

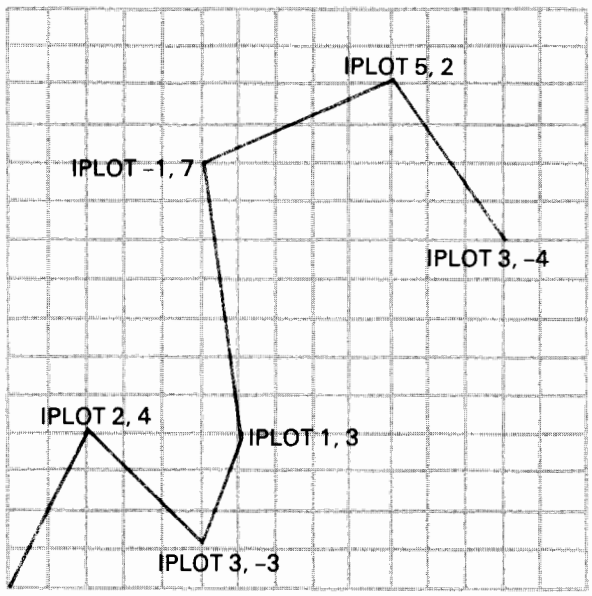
The optional pen control parameter directs the up and down movement of the pen and is interpreted the same as in the PLOT statement.

The reference origin is the pen's location before execution of IPLOT. Since the IPLOT statement changes the pen location, successive IPLOT statements refer to different local origins.

Examples:

`IPLOT 2,3,-1`	Lowers the pen and draws to a point 2 units right and 3 units up from the current pen position.
`IPLOT -2,4`	Draws or moves (depending on the current pen status) to a point 2 units left and 4 units up from the current pen position, then lowers the pen.

Example: The diagram below shows the line drawn by successive `IPLOT` statements. The grid represents the current scale. The initial pen position is at the lower-left corner; the pen is down.



Initial origin

Relative Plotting

The `RPLOT` statement provides relative plotting capability with pen control. `RPLOT` interprets the *x* and *y* parameters as increments relative to a local origin. The reference origin for `RPLOT` is the current pen position as determined by any statement which alters the pen position except `RPLOT`. `RPLOT` differs from `IPLOT` in that successive `RPLOT` statements do not change the reference origin.

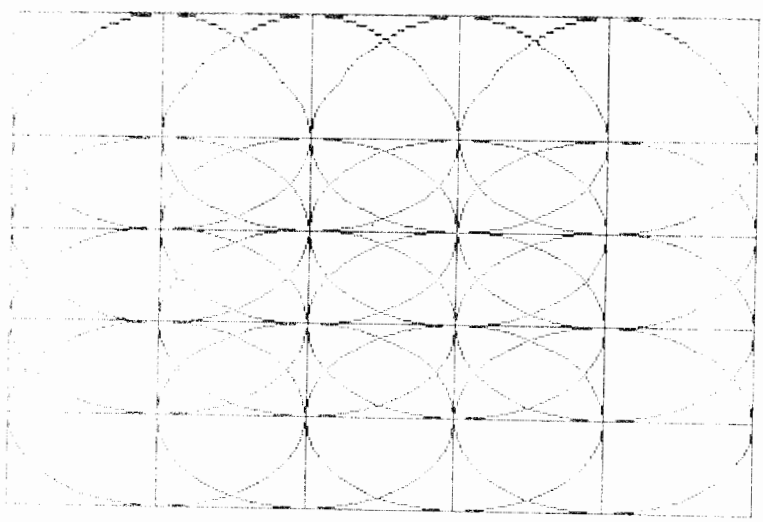
`RPLOT` *x-relative* , *y-relative* [, *pen control*]

The `RPLOT` parameters can be numbers, variables, or expressions. The pen remains at the *x-relative* , *y-relative* point until another plotting statement changes the pen position.

The optional pen control parameter directs the up and down movement of the pen and is interpreted the same as for `PLOT`. If the pen control parameter is not specified, the `RPLOT` statement directs the pen to move to the new location with its current up or down status, and drop the pen.

Example: The RPLOT statement can be used to draw duplicate plots.

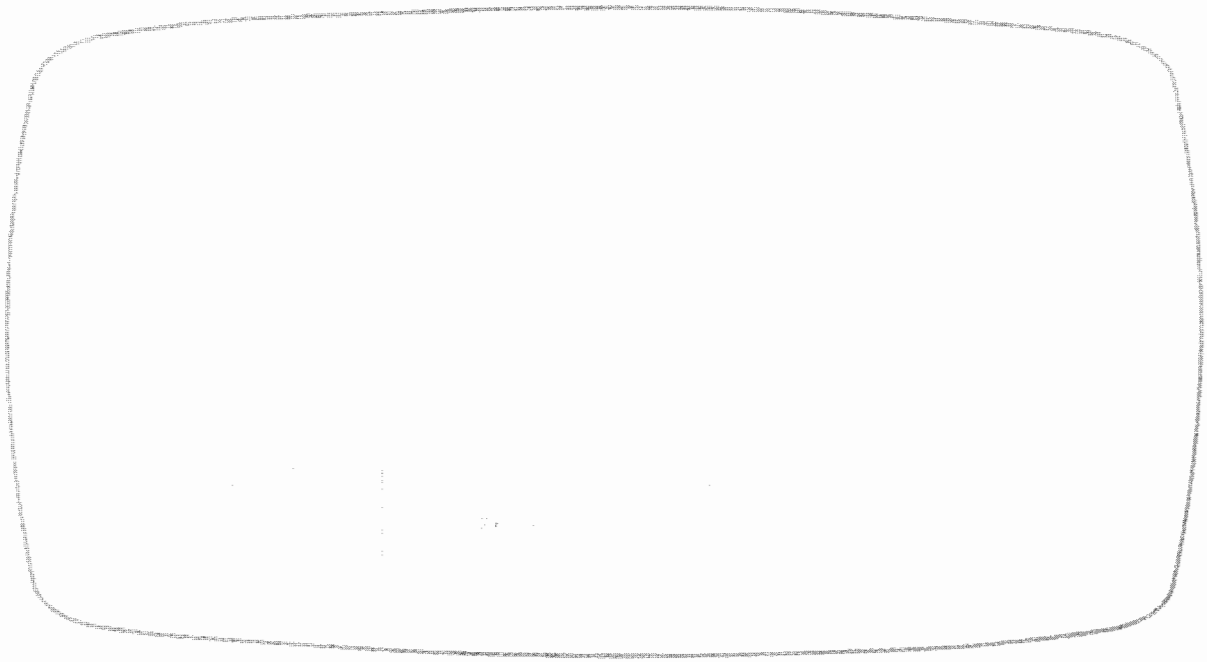
```
10 ! *** Rplot ***
20 PLOTTER IS 1
30 GCLEAR
40 SCALE 0,5,0,5 !                    Specifies user units scale.
50 GRID 1,1,0,0 !                     Draws a grid, x and y spacing = one unit.
60 FOR X=1 TO 4 !                     Loop assigns x coordinate of RPLOT origin.
70 FOR Y=1 TO 4 !                     Loop assigns y coordinate of RPLOT origin.
80 MOVE X,Y !                         Moves the pen to the specified point.
90 GOSUB 1000 !                       Goes to the Rplot subroutine.
100 NEXT Y !                          End loop.
110 NEXT X !                          End loop.
120 END
1000 ! Rplot
1010 DEG
1020 FOR ang=0 TO 360 STEP 10
1030 RPLOT COS (ang),SIN (ang) !      Plots an ellipse using the pen position from
1031 !                                the MOVE statement as the reference origin.
1040 NEXT ang
1050 RETURN
1060 END
```



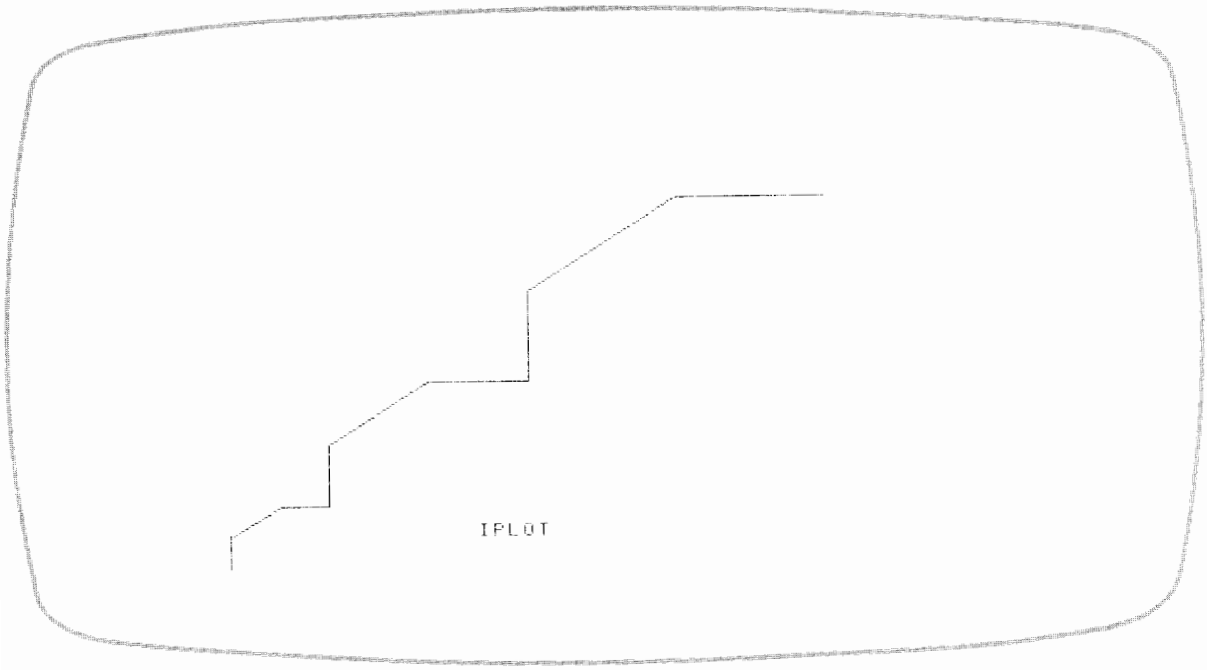
Example: The following program demonstrates the difference between incremental and relative plotting.

```
10 ! *** IRplot ***
20 PLOTTER IS 1 @ GCLEAR
30 SCALE 0,15,0,15 !                    Specifies user units scale.
40 DISP "Choose RPLOT or IPLOT" !       Prompts for choice of RPLOT or IPLOT.
50 INPUT P$ !                           Inputs character string.
60 FOR J=1 TO 12 !                      Loop RPLOTs or IPLOTs 12 points (X,Y).
70 READ X,Y !                           Reads X,Y point from DATA statement.
71 ! ******* Chooses RPLOT or IPLOT according to input string *******
72 ! ******* Pen control = -1 (lowers pen and draws point) *******
80  IF P$="RPLOT" THEN RPLOT X,Y,-1 ELSE IPLOT X,Y,-1
90  NEXT J !                            End loop.
100 DATA 0,1,1,1,1,0,0,0,0,2,2,2,2,0,0,0,0,3,3,3,3,0,0,0
110 MOVE 5,1 @ LABEL P$ !               Moves pen and labels RPLOT or IPLOT.
120 END
```

```
Choose RPLOT or IPLOT
?
RPLOT
```



```
Choose RPLOT or IPLOT
?
IPLOT
```



Moving and Drawing

The moving and drawing statements have built-in pen control. They provide an easy way to manipulate the pen without regard to whether the pen is currently up or down.

The MOVE statement lifts the pen and moves it to the specified *x,y* coordinate according to the current units. The pen remains in the up position until pen control is changed by another statement. The MOVE statement provides an easy way of moving the pen without drawing a line, regardless of whether the pen is currently up or down.

MOVE *x-coordinate* , *y-coordinate*

The *x* and *y* parameters are interpreted according to the current units; they can be numbers, variables, or expressions.

MOVE X,Y is equivalent to PLOT X,Y,-2. (Refer to the PLOT statement.)

Example:

MOVE 5,10	Lifts the pen and moves it to the point (5,10).

All the statements used to plot data can also be used to move the pen to position labels.

The DRAW statement lowers the pen and draws a line to the specified *x,y* coordinate according to the current units. The pen remains in the down position until pen control is changed by another statement. The DRAW statement provides an easy way of drawing a line from the current pen location to the location specified by the DRAW statement parameters, regardless of whether the pen is currently up or down.

```
DRAW x-coordinate , y-coordinate
```

The x and y parameters are interpreted according to the current units; they can be numbers, variables, or expressions.

`DRAW X,Y` is equivalent to `PLOT X,Y,-1`. (Refer to the `PLOT` statement.)

Examples:

`DRAW 5,10`	Draws a line from the current pen position to the point (5,10).
`DRAW 25,100`	Draws a line from the current pen position to the point (25,100).

Incremental Moving and Drawing

The `IMOVE` and `IDRAW` statements are the incremental moving and incremental drawing counterparts to the `IPLOT` statement.

The `IMOVE` (incremental move) statement provides incremental moving capability. `IMOVE X,Y` is equivalent to `IPLOT X,Y,-2`.

```
IMOVE x-increment , y-increment
```

The `IDRAW` (incremental draw) statement provides incremental drawing capability. `IDRAW X,Y` is equivalent to `IPLOT X,Y,-1`.

```
IDRAW x-increment , y-increment
```

The x and y parameters in the `IMOVE` and `IDRAW` statements are interpreted according to the current units relative to a local origin; they can be numbers, variables, or expressions. The local origin is the current pen position. Like the `IPLOT` statement, successive `IMOVE` or `IDRAW` statements refer to different local origins.

Examples:

`IMOVE 1,3`	Moves the pen from the current pen position (point x,y), to a point 1 unit to the right and 3 units up (to point $x+1,y+3$).
`IDRAW 1,3`	Draws a line from the current pen position to a point 1 unit to the right and 3 units up.

Plot Direction

The PDIR (plot direction) statement sets the angle of axis rotation for relative plotting (RPLOT) and incremental plotting (IPLOT, IMOVE, and IDRAW) in the current angular units mode (DEG, RAD, or GRAD). If the plot direction is specified with run/rise parameters, the parameters are interpreted in the current scale units.

```
PDIR angle
     run , rise
```

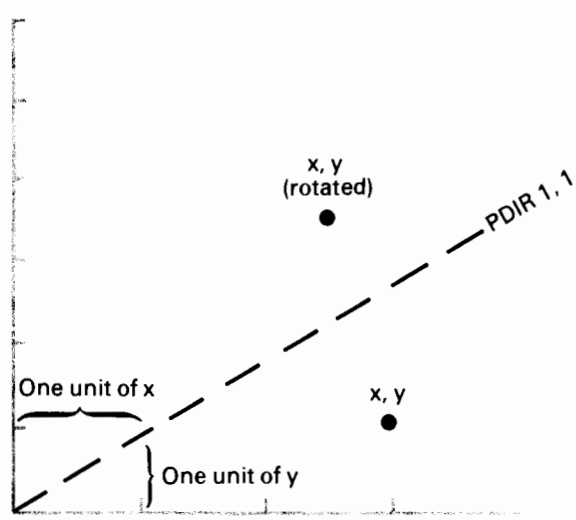
The angle of axis rotation can be specified in either of two ways:

- As the counterclockwise angle, in the current angular units (DEG, RAD, or GRAD) between the new *x*-axis and the horizontal axis.
- As the run and rise units (in current units) of a vector drawn in the desired direction.

The angle parameter and the run and rise parameters can be numbers, variables, or expressions.

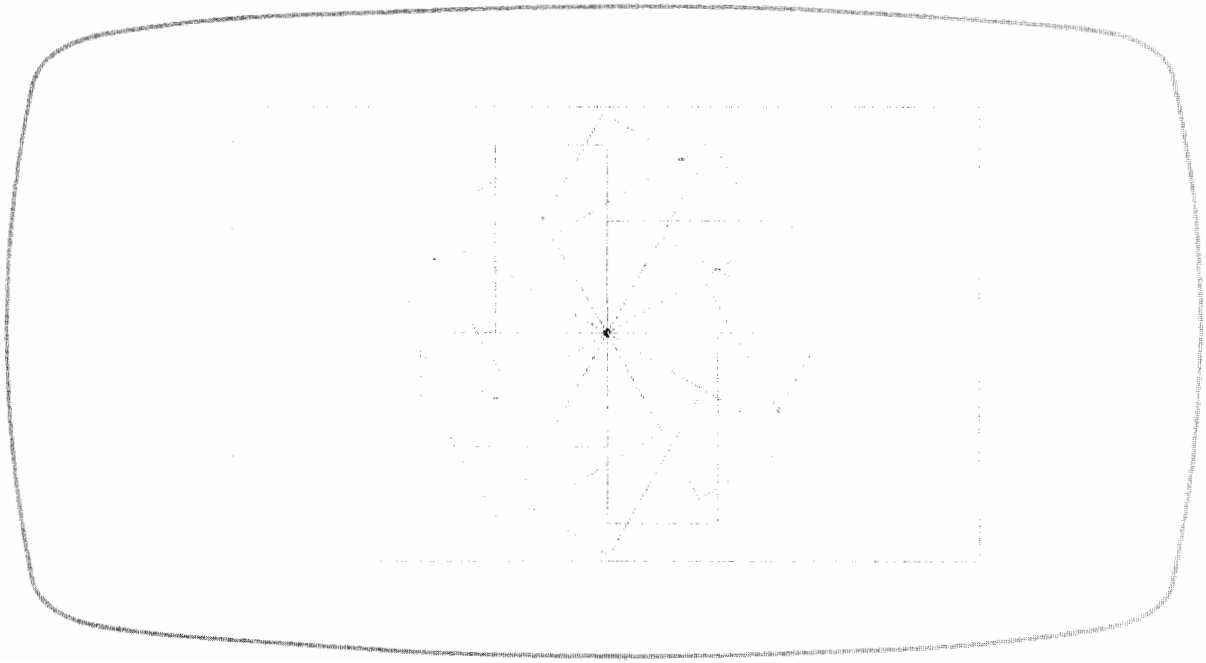
Note: For displayed graphics (PLOTTER IS 1), the PDIR angle parameter is based on the relative *x* and *y* dot spacing of the HP-87 CRT.

The current scaling units are preserved relative to the rotated axes. The PDIR statement only affects data plotted by the RPLOT, IPLOT, IMOVE, and IDRAW statements. You cannot, for example, draw rotated axes. Labels can be rotated by using the LDIR statement (refer to section 17 for a discussion of labeling direction.)



Example: The following program draws a series of rectangles according to the rotated plotting direction specified by PDIR.

```
10 ! *** Plot direction ***
20 PLOTTER IS 1
30 GCLEAR
40 SCALE 0,12,0,12 !                    Specifies user units.
50 FRAME
60 PEN UP @ PLOT 6,6 !                  Lifts pen, plots the point (6,6).
61 ! ******* Plot direction loop - increments plotting direction *******
70 FOR ang=0 TO 360 STEP 30 !           Loop counter = plotting direction angle.
80 DEG @ PDIR ang !                     Specifies plotting direction in degrees.
81 ! ******* Loop plots a rectangle using  current plotting direction *******
90 FOR Pt=1 TO 4 !                      Loop plots rectangle.
95 READ X,Y !                           Reads DATA for IPLOT rectangle.
100 IPLOT X,Y
110 NEXT Pt !                           End loop.
120 RESTORE
130 NEXT ang !                          End loop.
140 DATA 0,3,3,0,0,-3,-3,0
150 END
```



Section 17

Plotting Axes and Labels

Axes and labels are useful enhancements to plotted data. Axes are drawn according to the current units, providing a visual reference scale for your plots. Labels can be used in a variety of applications—labeling axes, plots, or individual drawings. The labeling statements allow you to graphically produce text with flexibility in the character set, size, and shape.

In the first part of this section we discuss how to draw axes. The axes can be normal x and y axis lines, a grid, or a frame around the plotting area. Later in the section we discuss how to generate various types of labels.

Plotting Axes

Axes are drawn using the current pen color and line type. The different types of axes (x and y axes, frames, and grids) require different statements; each statement has its own set of parameters. The optional tick-marks are reference marks drawn on the axes at specified intervals.

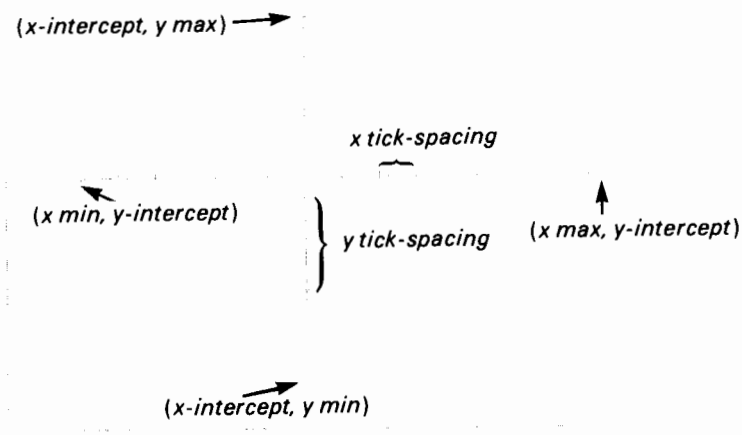
X and Y Axes

The XAXIS and YAXIS statements draw an x-axis and a y-axis, respectively, with optional tick-marks.

```
XAXIS y-intercept [, tick-spacing [, x min, x max]]
```

```
YAXIS x-intercept [, tick-spacing [, y min, y max]]
```

The XAXIS statement generates a horizontal axis at the specified y-intercept value. The YAXIS statement generates a vertical axis at the specified x-intercept value. The x and y-intercept parameters enable you to position the axes anywhere in or out of the plotting area. If the intersection point lies outside of the plotting area, only that portion of the axis within the plotting area is drawn. An intercept point must be specified with the XAXIS and YAXIS statements; the remaining parameters are optional. The parameters can be numbers, variables, or expressions.

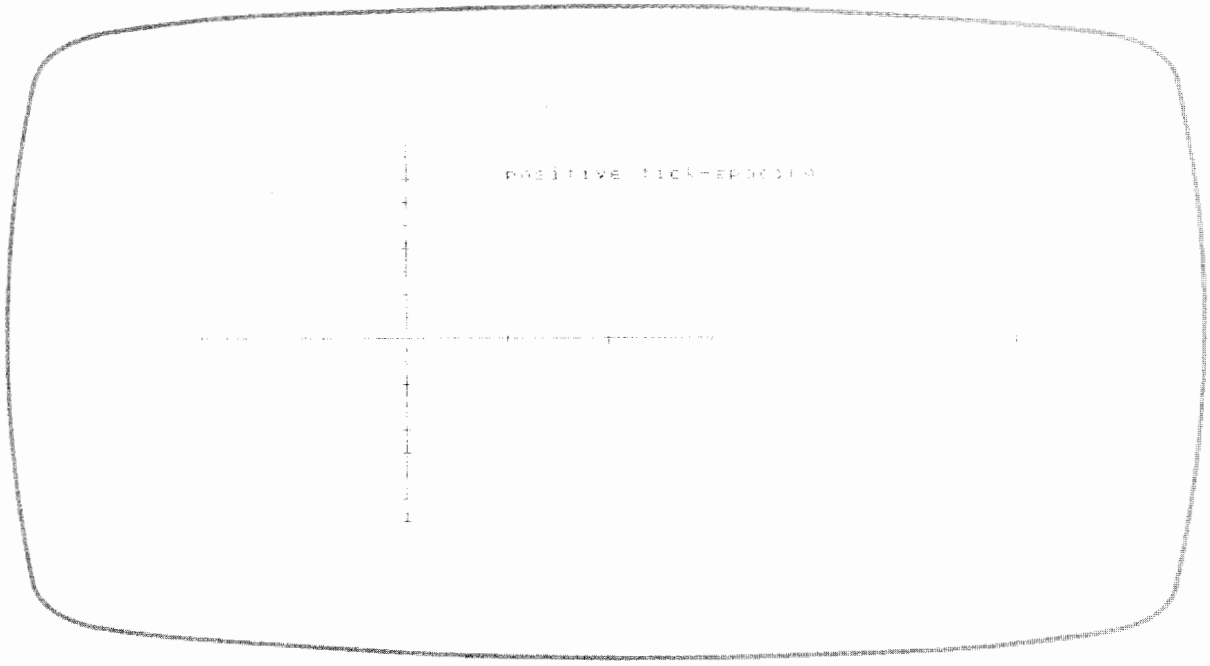


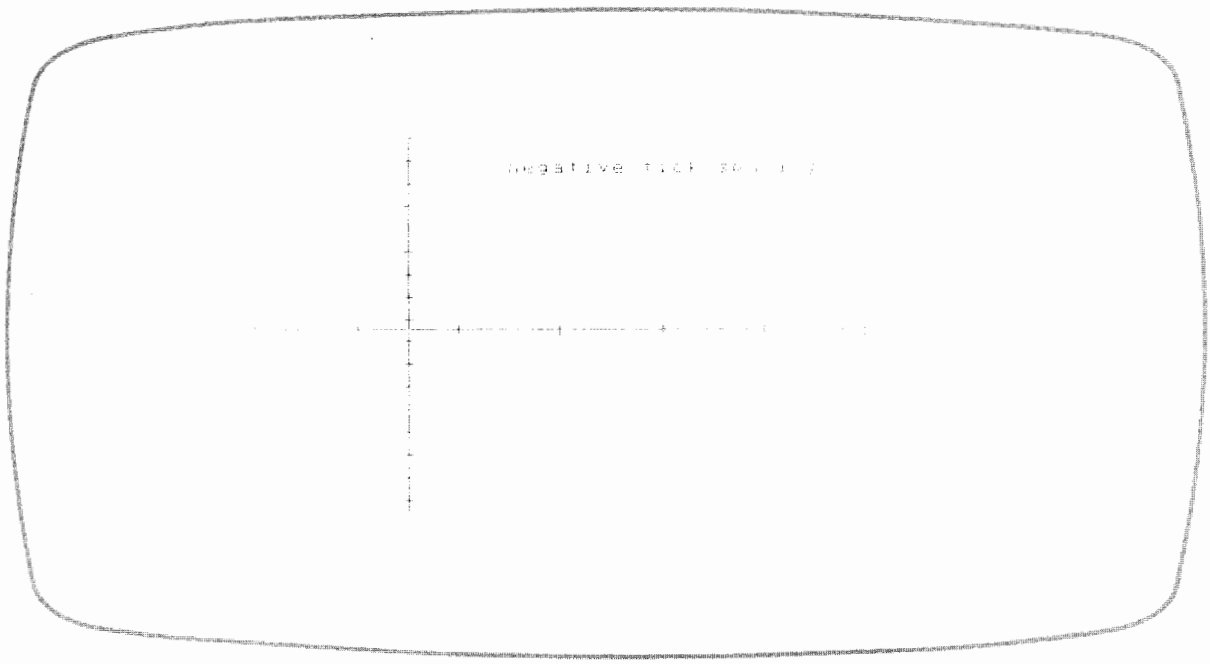
The x and y tick-spacing parameters are interpreted according to the current scale. Ticks are placed on the axis at the specified interval. The sign of the tick-spacing parameter determines how ticks are justified on each axis. Positive tick-spacing parameters specify that ticks are left-justified on the x-axis and bottom-justified on the y-axis. Negative tick-spacing parameters specify that ticks are right-justified on the x-axis and top-justified on the y-axis. Zero or no tick-spacing parameter signifies no ticks at all. Ticks are 2 GUs long.

The optional x min, x max, y min, and y max parameters enable you to specify shorter axes within the current plotting area. If these parameters are not specified, the axes are drawn across the entire plotting area (bounded by the plotting boundaries if in user units mode, and by the graphics limits if in graphics units mode).

Example:

```
10 ! *** XYaxis ***
20 PLOTTER IS 1
30 GRAPHALL
40 SCALE 0,10,-10,10 !          Scales the plotting area: 0 to 10 in the
41 !                            x direction, -10 to 10 in the y direction.
50 MOVE 4,7 @ LABEL "positive tick-spacing"
60 XAXIS 0,1,1,9.5 !            Draws an x-axis: y-intercept = 0, tick-
61 !                            spacing = 1, x min = 1, x max = 9.5.
70 YAXIS 3,1,-8,8.5 !           Draws a y-axis: x-intercept = 3, tick-
71 !                            spacing = 1, y min = -8, y max = 8.5.
80 WAIT 3000
90 GCLEAR
100 MOVE 4,7 @ LABEL "negative tick-spacing"
110 XAXIS 0,-1,1,9.5 !          Draws same x-axis with negative tick-spacing.
120 YAXIS 3,-1,-8,8.5 !         Draws same y-axis with negative tick-spacing.
130 END
```





The AXES statement draws a pair of axes with optional major and minor tick-marks. The major tick-marks are twice the length of the minor tick-marks.

AXES [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x major count* , *y major count* [, *major tick-size*]]]]

The axes intersect at (*x-intersection, y-intersection*); the coordinates are interpreted in the current units. The axes are drawn across the entire plotting area, in contrast to XAXIS and YAXIS which draw the axes from *x* min to *x* max and from *y* min to *y* max.

The x and y tick-spacing parameters specify, in current units, the distance between tick-marks on each axis. The absolute value is taken of negative values; zero signifies no ticks at all. Ticks are always counted from the origin to the ends of the axes.

The optional x and y major count parameters are positive integers which specify the number of tick-intervals between major ticks. For example, an x major count of 2 means that every other tick on the x-axis is a major tick. The tick-marks are drawn so that a major tick falls at the axis intersection point. The default values for the count parameters are 1,1 (all ticks are major ticks). If a tick-spacing parameter is zero, then no major or minor ticks are drawn on that axis.

The optional major tick-size parameter specifies the length of a major tick, end-to-end, in GUs. Minor ticks are always half the size of major ticks. The default length of a major tick is two GUs.

The AXES statement parameters can be numbers, variables, or expressions.

When the AXES statement is executed without parameters, the computer automatically draws a pair of axes that intersect at the lower-left corner of the plotting area; ten to nineteen major ticks are drawn on each axis depending on the current scale (no minor ticks are drawn).

If AXES if executed with the optional tick-spacing parameters and no other parameters, the axes are drawn with the intersection point at (0,0).

Example:

```
10 ! *** AXES ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME !                          Frames the graphics limits.
50 LOCATE 20,140,10,90 !            Locates the plotting boundaries.
60 SCALE 1975,1985,0,100 !          Scales the located plotting area.
70 AXES 1,2,1975,0,1,5,5 !          Draws axes: one minor tick per unit x, one
71 !                                minor tick per two units y, intersection at
72 !                                (1975,0), each minor tick is a major tick on
73 !                                the x-axis, one major tick per 5 minor ticks
74 !                                on the y-axis, major tick size is 5 GUs.
80 END
```

The differences between the AXES statement and the XAXIS, YAXIS statements are summarized below:

AXES **Statement**	XAXIS, YAXIS **Statements**
Draws both *x* and *y* axes.	Draws individual *x*-axis, *y*-axis.
Draws major and minor ticks, specifies tick-size and spacing.	Draws all ticks 2 GUs long, specifies tick-spacing.
Draws both axes across the entire plotting area.	Draws axes to the specified endpoints.
Ticks are justified from the origin. (Negative tick-spacing parameter has no effect on tick count.)	Negative tick-spacing parameter right-justifies ticks on the *x*-axis, top-justifies ticks on the *y*-axis. Positive tick-spacing parameter specifies left and bottom-justified ticks.
Can be executed without parameters for default axes.	Requires specification of *x* and *y*-intercepts (no default).

Framing Plots

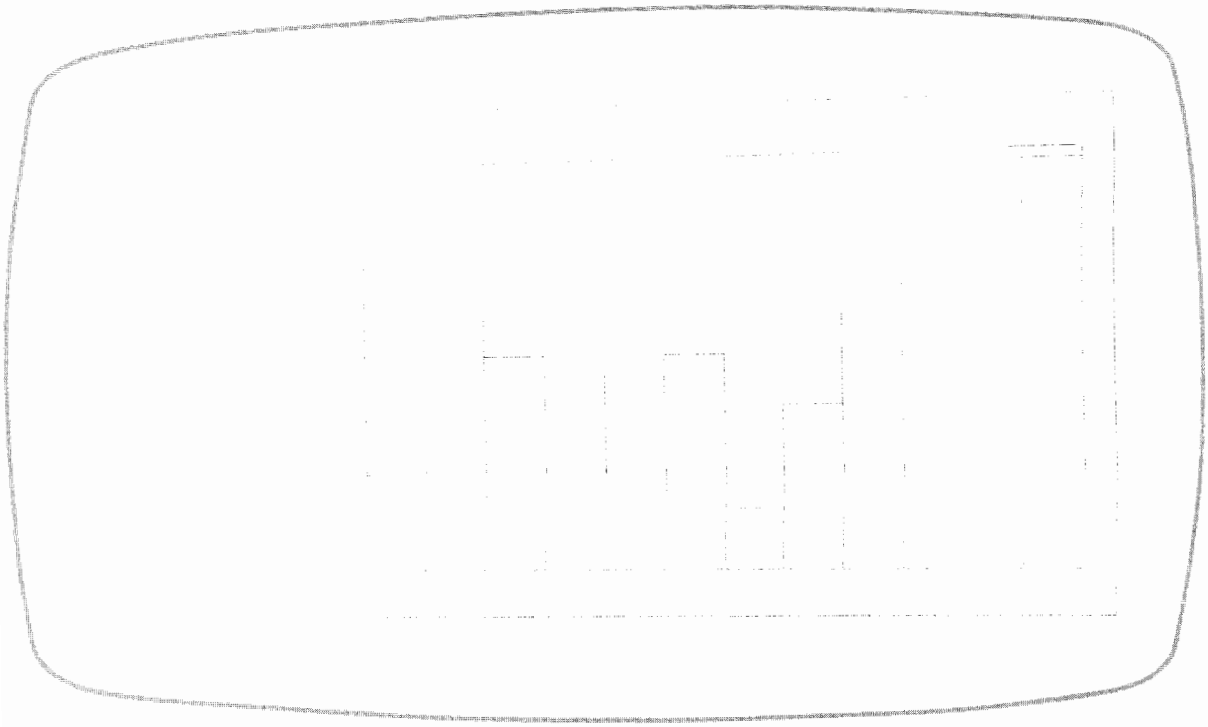
The FRAME statement draws a box around the current plotting area specified by the LIMIT, LOCATE, or CLIP statements.

```
FRAME
```

The box is drawn around the current plotting area using the current pen color and line type. The pen is positioned at the lower-left corner of the frame after the frame is complete.

Example: This program uses the FRAME and CLIP statements to draw a bar chart.

```
10 ! *** Bar Chart ***
20 PLOTTER IS 1
30 GRAPHALL @ FRAME !          Specifies graph-all mode, and frames
31 !                           the default graphics limits.
40 LOCATE 60,220,10,90 @ FRAME ! Locates the plotting boundaries and
41 !                           frames the plotting area.
50 SCALE 0,24,0,100 !          Specifies user units.
60 AXES 0,5,0,0,0,1,5 !        Draws x and y axes to current scale.
70 FOR Bar=1 TO 12 !           Loop draws 12 bars.
80 READ PT !                   Reads bar height from DATA statement.
90 CLIP Bar*2-2,Bar*2,0,PT !   Clips plotting area 2 units by PT units.
100 FRAME !                    Frames the CLIP-specified plotting area.
110 NEXT Bar !                 End loop.
120 DATA 90,65,50,47,32,50,15,40,60,72,65,98
130 END
```



Drawing Grids

The GRID statement can be used as an alternative to the axes statements when grid lines are desired. The GRID statement draws *x* and *y* axes (with optional tick-marks) and grid lines at specified intervals along the axes. All parameters are interpreted according to the current scale; they can be numbers, variables, or expressions.

> GRID [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x grid-spacing* , *y grid-spacing* [, *minor tick-size*]]]]

The *x* and *y* grid-spacing parameters are analogous to the major tick count parameters in the AXES statement. The grid-spacing parameter specifies the number of tick-intervals between the grid lines. If no ticks are specified (tick-spacing equals zero) then no grid lines are drawn. The default grid-spacing values are 1,1 (all ticks are grid lines).

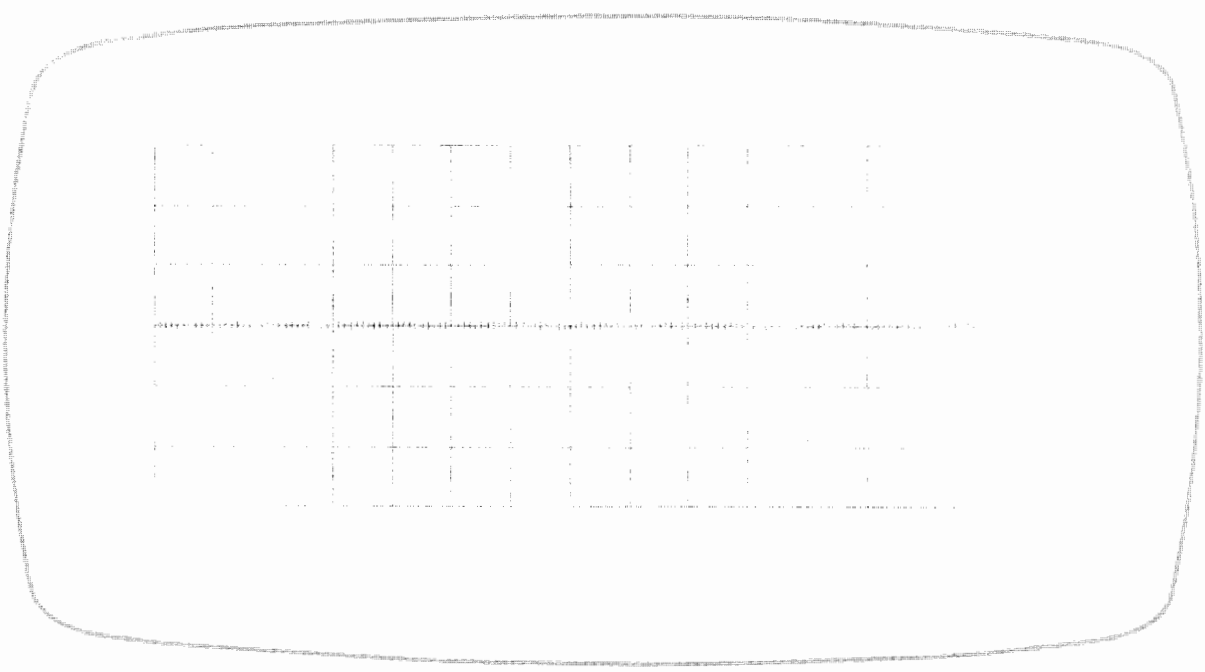
The remaining GRID statement parameters are the same as for the AXES statement, except the last parameter, which specifies the length of the minor ticks. The x-intersection and y-intersection parameters specify the location of x and y axes. The grid lines are drawn across the entire plotting area and are spaced symmetrically about the intersection of the x and y axes. The minor tick-size parameter specifies the length of the tick-marks in GUs; the default tick-size is two GUs. The default value for the x,y intersection is the lower-left corner of the plotting area.

When the GRID statement is executed without parameters, the computer automatically draws ten to nineteen grid lines (depending on the current scale) in the x and y directions.

Like all the axis drawing statements, the grid lines and tick-marks are drawn using the current line type and pen color. When plotting grid lines in a line type other than 1 (solid), portions of the grid are overlapped, creating a pattern which is different from the specified line type.

Example: The following program draws a 140mm by 60mm grid with tick-marks spaced every millimeter.

```
10 ! *** Metric Grid ***
20 PLOTTER IS 1
30 GRAPHALL !                      Specifies graph-all mode.
40 LIMIT 10,150,10,70 !            Specifies the graphics limits.
50 MSCALE 0,0 !                    Specifies metric scale units, no offset.
60 GRID.1,1,30,30,10,10 !          Draws a grid: one tick per unit x, one tick
61 !                               per unit y, intersection at (30,30), one
62 !                               grid line per 10 ticks in x, one grid line
63 !                               per 10 ticks in y.
70 END
```



Plotting Labels

The statements discussed below enable you to plot labels on the graphics display using any of the characters available from the alpha display character set (except inverse video characters). Refer to the Table of Character and Key Codes in appendix D for a complete list of characters available for labeling. The labels can be positioned anywhere within the graphics limits and drawn in any direction, in a variety of shapes, sizes, and slants.

Creating Labels

Label formats are determined by the LABEL and LABEL USING statements. These statements control labeling operations on the graphics display just as DISP and DISP USING control display operations on the alpha display.

```
LABEL [label list]
```

```
                    "format string"
LABEL USING statement number [; label list]
                    statement label
```

The label list may be variable names, constants, numeric expressions, or the TAB function (for the LABEL statement only). The format string for the label list can be specified within the LABEL USING statement, or by an IMAGE statement referenced by the statement number or label. For detailed information on the formatting of labels (format strings and IMAGE statements) refer to section 10, Printer and Display Formatting.

The position and direction of a label are determined by the current pen position and the LORG (label origin) and LDIR (label direction) statements. The size, aspect ratio, and slant of the characters in the label are determined by the CSIZE (character size) statement. These statements are discussed in detail later in this section.

Labels can be plotted anywhere within the graphics limits, inside or outside the plotting boundaries specified by a CLIP or LOCATE statement.

Unlike the DISP statement, the LABEL statement does not place the label list items into a storage buffer when the list ends with a comma or a semicolon.

Example: The LABEL and LABEL USING statements have been added to the Bar Chart program on page 220 to label the chart.

```
10 ! *** Labeled Bar Chart ***
20 PLOTTER IS 1
30 GRAPHALL @ FRAME !                    Specifies graph-all mode, and frames
31 !                                     the default graphics limits.
40 LOCATE 60,220,10,90 @ FRAME !         Locates the plotting boundaries and
41 !                                     frames the plotting area.
50 SCALE 0,24,0,100 !                    Specifies user units.
60 AXES 0,5,0,0,0,1,5 !                  Draws x and y axes to current scale.
70 FOR Bar=1 TO 12 !                     Loop draws 12 bars.
80 READ PT !                             Reads bar height from DATA statement.
90 CLIP Bar*2-2,Bar*2,0,PT !             Clips plotting area 2 units by PT units.
100 FRAME !                              Frames the CLIP-specified plotting area.
110 NEXT Bar !                           End loop.
120 DATA 90,65,50,47,32,50,15,40,60,72,65,98
121 ! ******* Labeling Operations *******
130 FOR M=1 TO 12 !                      Loop labels bars with months.
140 READ Mth$ !                          Reads character string for Mth$.
150 MOVE 2*M-1,5 !                       Moves pen into position for labeling.
160 DEG @ LDIR 90 !                      Sets label direction 90 degrees.
170 LABEL Mth$ !                         Labels month.
180 NEXT M !                             End loop.
190 DATA January,February,March,April,May,June,July,August,September,October,Nov
ember,December
200 FOR Y=0 TO 100 STEP 10 !             Loop labels the y-axis.
210 LDIR 0 @ LORG 8 !                    Sets label direction and origin.
220 MOVE 0,Y !                           Moves pen in position with y tick-mark.
230 LABEL Y !                            Labels y-axis scale.
240 NEXT Y !                             End loop.
250 MOVE -8,70 @ LORG 1
251 ! ******* The LABEL USING statement formats labels. *******
260 LABEL USING 270 ; "Monthly","Umbrella","Sales","Alpine,","Oregon"
270 IMAGE K/K/K//K/K
280 END
```

Label Position and Direction

The LORG (label origin) statement specifies the label position relative to the current pen location. There are 9 different label positions available.

LORG *label position*

The label position parameter is a number, variable, or expression which selects one of the 9 label origin options (1 through 9).

Non-integer label position parameters are rounded to the nearest integer. If the specified position is not in the range 1 through 9, or if the LORG statement is not executed, the default position is LORG 1.

The 9 label position parameters indicate the initial pen position on the following label:

- LORG 1, 2, and 3 left-justify labels.
- LORG 4, 5, and 6 center labels.
- LORG 7, 8, and 9 right-justify labels.

The LORG positions are based on the size of an upper-case letter. The smaller lower-case letters and symbols are shifted below center when using LORG 2, 5, and 8. Slanted characters on the CRT are shifted slightly to the right or left (in the direction of the slant) from the current label origin.

Example: The following program illustrates the 9 label positions. Each cross marks the initial position of the pen for the corresponding label at each LORG position.

```
10 ! *** Label Origin ***
20 PLOTTER IS 1
30 SHOW 0,10,0,10 !                    Specifies user units.
40 LINE TYPE 3,1 !                     Specifies a dotted line type.
50 CSIZE 20 !                          Specifies character size 20.
60 FOR I=1 TO 9 !                      Loop demonstrates 9 label origins.
70 GCLEAR
80 XAXIS 5,0,3,7 !                     Draws horizontal segment of cross.
90 YAXIS 5,0,3,7 !                     Draws vertical segment of cross.
100 LORG I !                           Specifies label origin.
110 MOVE 5,5 !                         Moves the pen to axes intercept.
120 LABEL USING "K" ; "LORG ",I !      Labels the current LORG position.
130 WAIT 2000
140 NEXT I !                           End loop.
150 END
```

Label positions 7, 8, and 9 provide three forms of right-justification for labeling *y*-axis tick-marks with ease. Label position 6 provides an easy means of centering *x*-axis labels beneath tick-marks, regardless of label length.

The LORG statement is often used in conjunction with the LDIR statement.

The LDIR (label direction) statement specifies the direction in which labels are drawn. LDIR is the labeling counterpart to PDIR (plot direction.)

LDIR *angle* / *run* , *rise*

The label direction can be specified in either of the following ways:

- The label direction can be specified as the counterclockwise angle (in the current units—DEG, RAD, or GRAD) between the label and the normal (horizontal) *x*-axis.
- The label direction can be specified as the run and rise units (in current units) of a vector drawn in the desired direction.

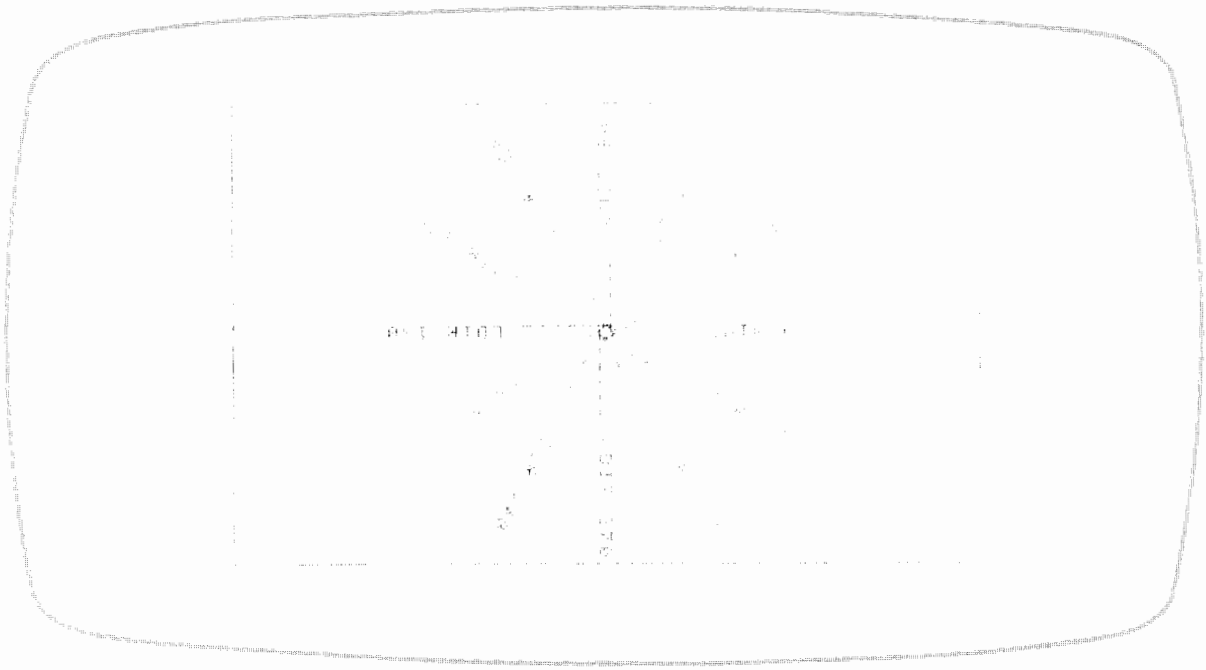
The angle parameter and the run and rise parameters can be numbers, variables, or expressions.

Note: For displayed graphics (PLOTTER IS 1), the LDIR angle parameter is based on the relative *x* and *y* dot spacing of the HP-87 CRT.

If the LDIR statement is not executed, the default direction is left to right (LDIR 0 degrees).

Example: This program draws a label at 30 degree intervals around a center point.

```
10 ! *** Label Direction ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME
50 DEG !                                    Sets degrees mode.
60 SHOW -10,10,-10,10 !                     Specifies user units scale.
70 LORG 2 !                                 Centers labels on left end.
80 FOR D=0 TO 330 STEP 30 !                 Loop increments label direction.
90 LDIR D !                                 Specifies label direction.
100 MOVE 0,0 !                              Moves the pen to the center.
110 LABEL USING "K" ; "______ LDIR ",D !    Labels current label direction.
120 NEXT D !                                End loop.
130 END
```



Character Size and Slant

Unlike the characters produced on the computer's alpha display, characters drawn as labels can be created in a variety of shapes, sizes, and slants.

The CSIZE (character size) statement specifies the height, aspect ratio, and slant of the characters used in labels. The CSIZE statement parameters can be numbers, variables, or expressions.

CSIZE *space height* [, *aspect ratio* [, *slant*]]

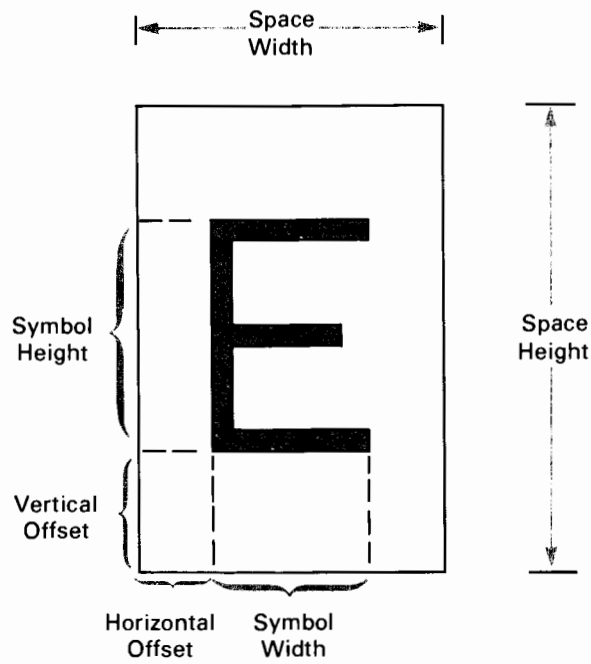
A character is composed of both a symbol and the space that surrounds the symbol, separating it from other characters. The height parameter specifies the height of the space surrounding the symbol, in GUs. The actual height of the symbol is a fraction of the specified height. For full-sized characters (all uppercase letters and numbers for example), the symbol height/space height ratio is ½ for the CRT and pen plotters. Likewise, the width of the symbol is a fraction of the space width: ½ for the CRT and ⅔ for pen plotters (for full-sized characters). The space above and beside the symbol becomes the spacing between lines and characters.

The default height parameter is 5 GUs for the CRT and 3 GUs for pen plotters at power-on, reset, or after the PLOTTER IS statement or the LIMIT statement is executed. Thus the default symbol height is actually 5 × (½) GUs for the CRT and 3 × (½) GUs for pen plotters.

The aspect ratio is the ratio of the symbol width to the symbol height for both CRT and pen plotter characters. The default aspect ratio for the CRT is ⅔ (.666666666667) and .6 for pen plotters. The aspect ratio is an optional parameter; if omitted from the CSIZE statement, the default value is assumed.

The placement of the symbol within the character space differs for CRT and pen plotter characters. The CRT symbol is offset from the lower-left corner of the character space; the pen plotter symbol is positioned in the lower-left corner of the character space.

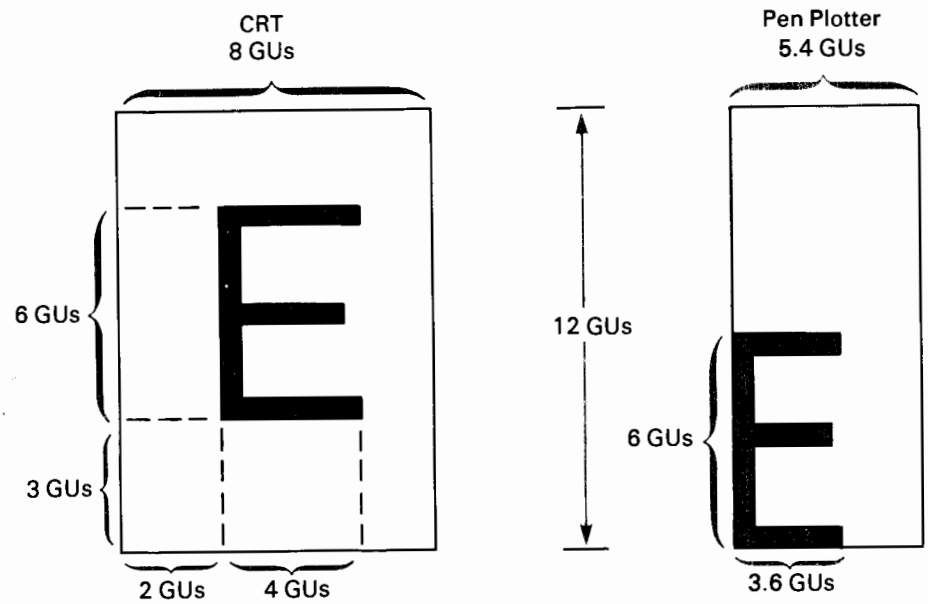
The following table describes how the computer dimensions the characters in a label according to the specified CSIZE height and aspect ratio parameters.The dimensions are different for the CRT and pen plotters.



CSIZE **Character Dimensions**

Character Dimension	CRT Graphics Display	Pen Plotter
Space Height	CSIZE space height parameter default value = 5 GUs	CSIZE space height parameter default value = 3 GUs
Symbol Height	½ × Space Height	½ × Space Height
Space Width	aspect ratio parameter × height parameter	¾ × aspect ratio parameter × height parameter
Symbol Width	½ × Space Width	⅔ × Space Width
Horizontal Offset	¼ × Space Width	none
Vertical Offset	¼ × Space Height	none

Example: `CSIZE 12` is interpreted by the CRT and pen plotters in the following way:

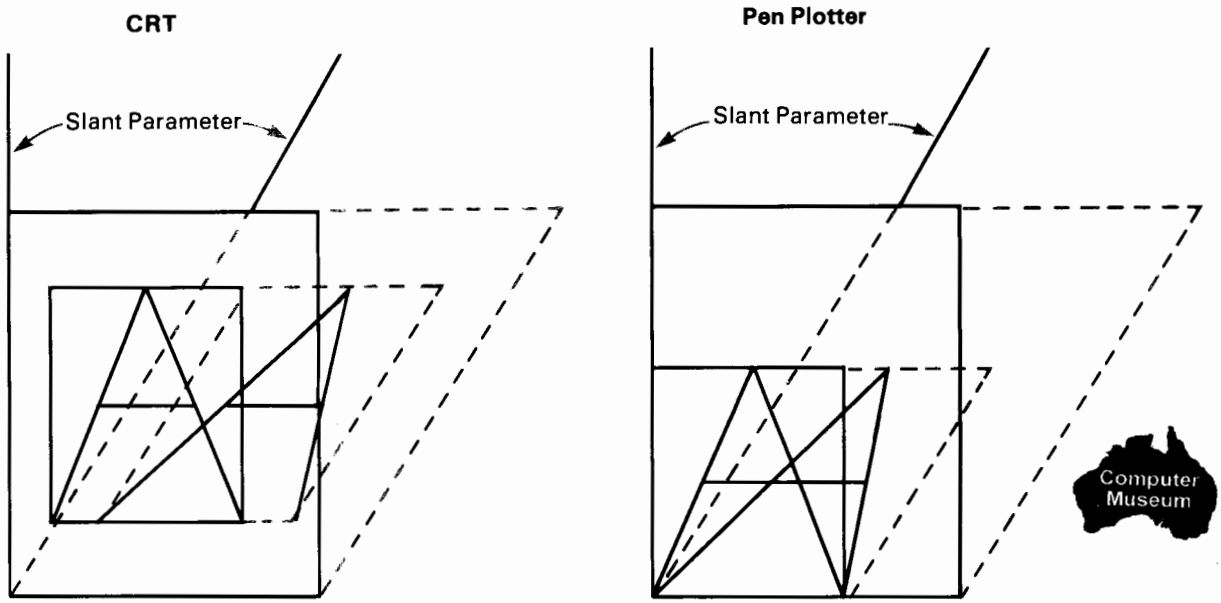


Character Dimension	CRT Graphics Display	Pen Plotter
Space Height	12 GUs	12 GUs
Symbol Height	½ × 12 GUs = 6 GUs	½ × 12 GUs = 6 GUs
Space Width	⅔ × 12 GUs = 8 GUs	¾ × .6 × 12 GUs = 5.4 GUs
Symbol Width	½ × 8 GUs = 4 GUs	⅔ × 5.4 GUs = 3.6 GUs
Horizontal Offset	¼ × 8 GUs = 2 GUs	none
Vertical Offset	¼ × 12 GUs = 3 GUs	none

The optional slant parameter specifies the clockwise character slant from the vertical in the current angular units (`DEG`, `RAD`, or `GRAD`). The default slant is 0 degrees. The characters can be slanted anywhere in the range from −90 degrees to 90 degrees (excluding −90 and 90 degrees). If the slant parameter specifies an angle outside of this range, the character is slanted 180 degrees from the out-of-range slant parameter. For example, `CSIZE 12,.5,-120` is interpreted as `CSIZE 12,.5,60` assuming the current angular units are degrees. It is impractical to slant characters more than 70 degrees in either direction because of poor readability. Slanted CRT characters are shifted in the direction of the slant; this occurs because of the offset position of the symbol within the space.

Note: On monitors used with the HP-86, the `CSIZE` character slant depends on the relative *x* and *y* dot spacings of the monitor.

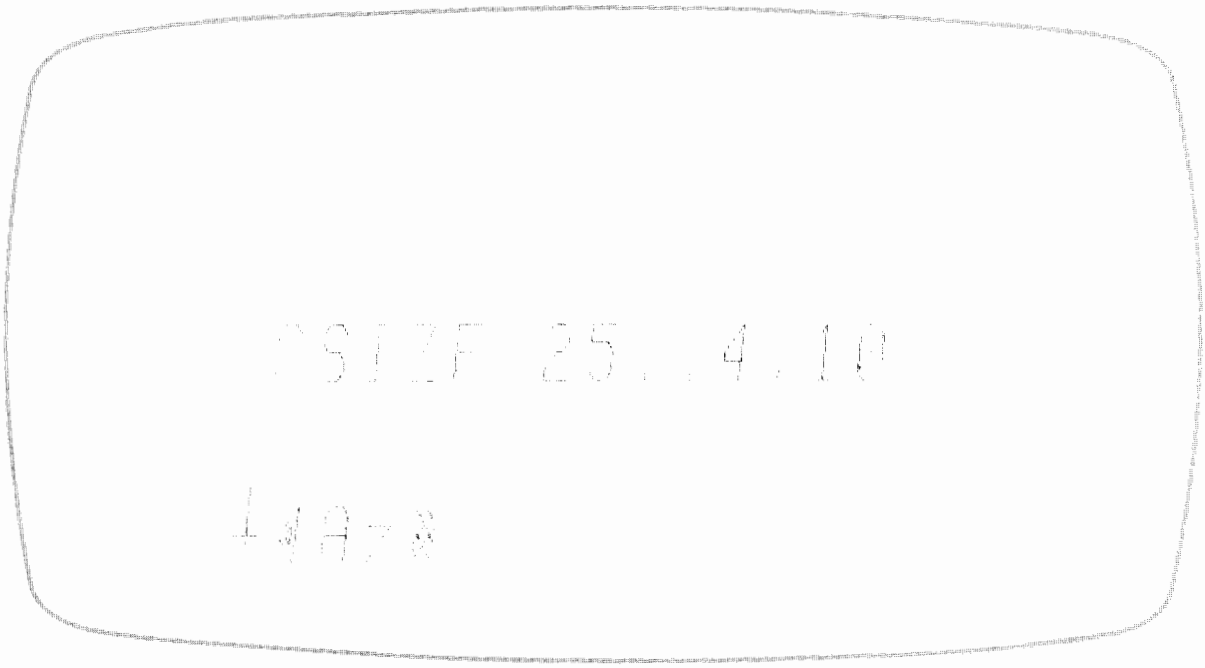
The following diagram shows how the character blocks for CRT and pen plotter labels are slanted using the `CSIZE` statement. Note that the slanted CRT character is shifted across the *x*-axis as the character block is slanted.



Example: The following program allows you to choose the character size parameters from the keyboard and plot characters in the current CSIZE. Pick a character from the Table of Character and Key Codes on page 323 and input the corresponding decimal character code. Inverse video characters cannot be produced as labels; when instructed to label an inverse video character, the computer labels the character in normal video.

```
10 ! *** Character Size ***
20 PLOTTER IS 1
30 CLEAR @ DEG
40 DISP "Enter Character Height, Aspect Ratio, Slant"
50 INPUT H,AS,S !                                  Inputs the CSIZE parameters.
60 GCLEAR
70 MOVE 10,40
80 CSIZE H,AS,S !                                  Specifies character size.
90 LABEL USING "K" ; "CSIZE ",H,",",AS,",",S !     Labels current CSIZE.
100 WAIT 3000
101 ! ******* Choose a character by specifying the character code *******
102 ! ******* and label it using the current CSIZE *******
110 A$="" !                                        Assigns null string to A$.
120 DISP "Enter a character decimal code" !        Prompts for decimal code.
130 ALPHA
140 INPUT code !            Inputs decimal character code.
150 A$=A$&CHR$ (code) !     Concatenates A$ with the CHR$ string specifed by
151 !                       the input character code.
160 MOVE 0,0
170 LABEL A$ !              Labels the string A$.
180 WAIT 2000
190 GOTO 130 !              Repeats concatenation and labeling operation.
200 END
```

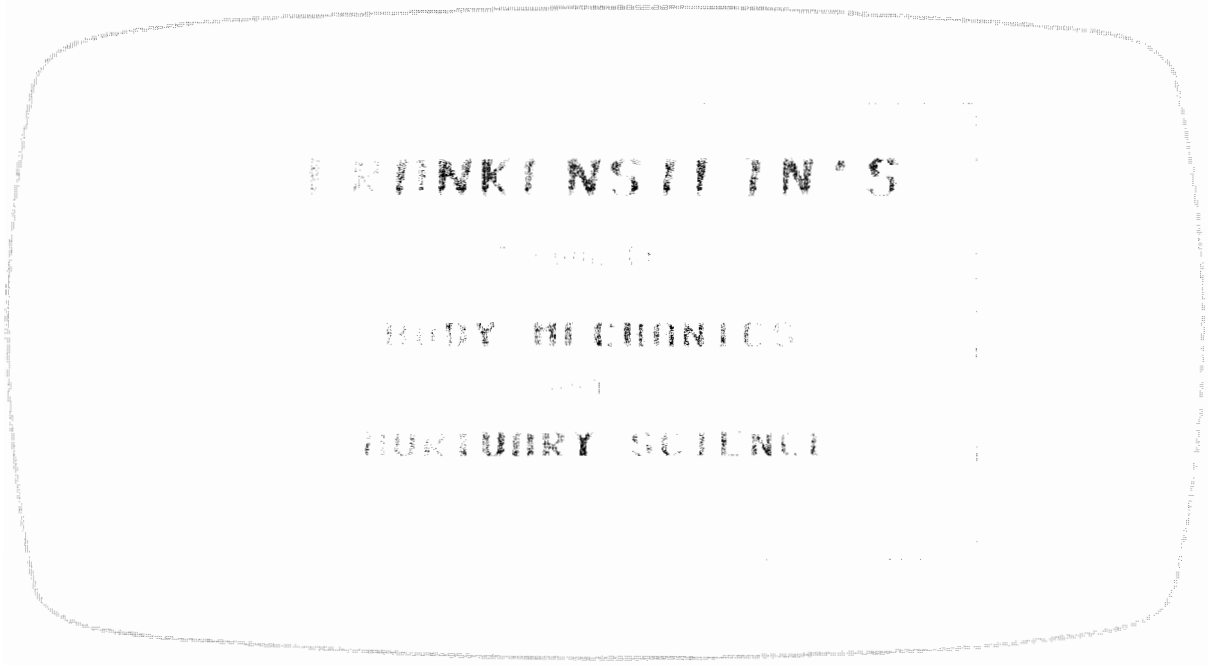
```
Enter Character Height, Aspect Ratio, Slant
?
25,.4,10
Enter a character decimal code.
?
31
?
0
?
65
?
122
?
127
?
```



Note: The height parameter and aspect ratio in the CSIZE statement can be any number, positive or negative. A negative height parameter rotates the label 180 degrees about the current label origin. A negative aspect ratio reflects the label across a vertical line (perpendicular to the current label direction) drawn through the label origin. Refer to section 19 for an example of reflected labels and plots.

Example: Special characters can be generated by varying parameters in the CSIZE statement.

```
10 ! *** Special Labeling ***
20 PLOTTER IS 1
30 GCLEAR
40 FRAME
50 DEG
60 FOR Angle=1 TO 12 !                Loop varies the CSIZE character slant.
70 CSIZE 16,.6,Angle !                CSIZE slant is Angle, the loop counter.
80 MOVE 18,80
90 LABEL "FRANKENSTEIN'S"
100 NEXT Angle !                      End loop.
110 MOVE 60,65
120 CSIZE 7,.6
130 LABEL "SCHOOL OF"
140 MOVE 72,36
150 LABEL "AND"
160 CSIZE 10
170 FOR I=0 TO 3 !                    Loop varies the label position.
180 MOVE I/3+35,47 !                  Increments label position horizontally.
190 LABEL "BODY MECHANICS"
200 MOVE I/3+30,23 !                  Increments label position horizontally.
210 LABEL "MORTUARY SCIENCE"
220 NEXT I !                          End loop.
230 END
```



Labeling Axes

The LAXES and LGRID statements draw axes and grid lines in the same manner as the AXES and GRID statements, but label the *x* and *y* axes at specified increments with the current scale units; only numeric labels are drawn using LAXES and LGRID. The labels drawn by LAXES and LGRID are automatically placed outside the plotting boundaries.

The numeric labels are drawn using the current CSIZE specifications in the format determined by the FXD statement. The LORG and LDIR statements do not affect LAXES and LGRID labels.

The axes labeling statements always place the labels outside the plotting boundaries specified either by LOCATE or CLIP. Therefore, when you use the LAXES and LGRID statements, you must allow space for the axes labels between the plotting boundaries and the graphics limits. If you do not allow enough space for the labels, you may get parts of labels or no labels at all. This can be used intentionally, if for instance, you wish to draw only one labeled axis with the LAXES statement.

The FXD (fixed) statement specifies the number of digits to the right of the decimal point in labels plotted by the LAXES and LGRID statements.

FXD *number of digits* [, *number of digits*]

The LAXES and LGRID labeling format allows a maximum of 8 characters plus sign in the label. The FXD parameters can be numbers, variables, or expressions with values in the range 0 through 7. Non-integer parameters are rounded to the nearest integer. If the number of digits parameter is greater than 7 or less than 0, the labeling format is FXD 0. If only one parameter is given in the FXD statement, both axes are fixed in the specified format. If two parameters are given, the first parameter specifies the fixed format for the *x*-axis labels and the second parameter specifies the fixed format for the *y*-axis labels. Labels that do not fit the fixed format are forced to exponential notation in the format "MDE" (as expressed in an IMAGE statement). The default labeling format is FXD 0.

The LAXES (label axes) statement draws a pair of axes and labels them with the current scale units at each major tick-mark.

> LAXES [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x major count* , *y major count* [, *major tick-size*]]]]

The parameters are interpreted the same as for the AXES statement. They can be numbers, labels, or expressions. Labels are placed outside the plotting boundaries below the *x*-axis and to the left of the *y*-axis, and centered on each major tick. With positive tick-spacing, labels for the corresponding axis are perpendicular to that axis. With negative tick-spacing, labels are parallel to the axis. The labels are formatted with reference to the current FXD and CSIZE statements.

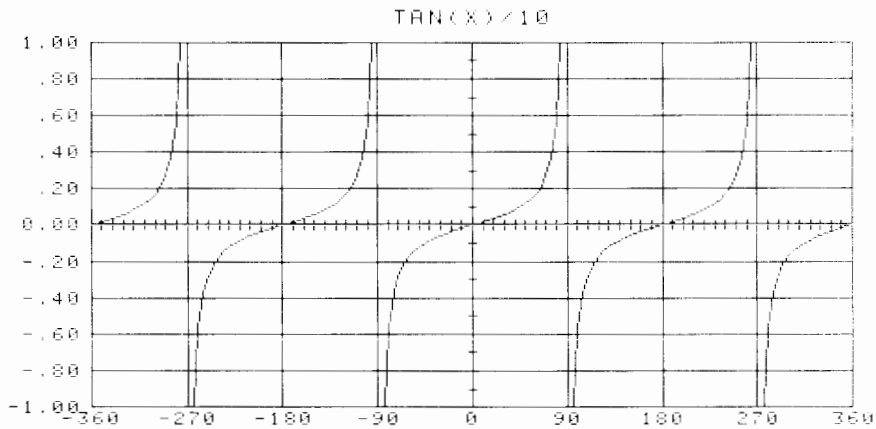
The LGRID (label grid) statement draws a grid, in the same manner as the GRID statement, and labels it at each grid line with the current scale units.

> LGRID [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x grid spacing* , *y grid spacing* [, *minor tick-size*]]]]

The parameters are interpreted the same as for the GRID statement. Labels are placed outside the plotting boundaries, within the graphics limits, on each grid line. The labels are formatted with reference to the current FXD and CSIZE statements.

Example: The LGRID statement is used in the following example to draw and label a grid for plotting (tan x)/10. The x grid lines are drawn and labeled every 90 degrees, from −360 degrees to +360 degrees. The y grid lines are drawn and labeled every 0.2 units, from −1.00 to +1.00.

```
10 ! *** TAN(X)/10 ***
20 PLOTTER IS 1
30 GRAPHALL !                  Sets the display to graph-all mode.
40 LOCATE 20,190,10,90 !       Specifies the plotting boundaries.
50 SCALE -360,360,-1,1
60 DEG
70 FXD 0,2 !                   Fixes 0 digits on x-axis labels, 2 on y-axis labels.
80 LGRID -10,.1,0,0,9,2 !      Draws a labeled grid with: one tick per 10 units in
81 !                           x, x-axis labels parallel to the axis, one tick per
82 !                           0.1 units in y, y-axis labels perpendicular to the
83 !                           axis, intersection at (0,0), one grid line and label
84 !                           per nine ticks in x, one grid line and label per two
85 !                           ticks in y.
90 FRAME
100 MOVE -360,0
101 ! ******* Loop plots TAN(X)/10 *******
110 FOR X=-360 TO 360 STEP 5
120     IF X MOD 360=90 OR X MOD 360=270 THEN PEN UP @ GOTO ENDLOOP
130     PLOT X,TAN (X)/10
140 ENDLOOP: NEXT X
150 MOVE 0,1.1
160 LORG 4
170 CSIZE 6
180 LABEL "TAN(X)/10"
190 END
```


TAN(X)/10
1.00
.80
.60
.40
.20
0.00
-.20
-.40
-.60
-.80
-1.00
-360
-270
-180
-90
0
90
180
270
360

Section 18

Special Graphics Operations

This section covers the more advanced graphics operations which have special applications. Most of the section is devoted to the BPLOT statement.

CRT Graphics—BPLOT and BREAD

BPLOTting or byte plotting is a type of plotting operation in which the computer addresses individual CRT dots, turning them on or off according to the parameters in the BPLOT statement. The plotting area is set up using the same procedures as for plotting data, axes, and labels. You can BPLOT anywhere within the default graphics limits (default limits depend on the current mode—*graph* or *graph-all*). However, the pen cannot be positioned outside the current graphics limits (as specified by LIMIT or by default) prior to BPLOTting. The pen *cannot* be positioned outside of the plotting boundaries for BPLOTting unless the computer is set to graphics units mode (unlike pen postioning for data and labels). The PEN statement is ignored during BPLOT operations. BPLOT plots white or black dots according to the BPLOT statement parameters.

The BREAD statement allows you to read and store the contents of the CRT graphics display. The display is converted dot by dot as binary code to the corresponding character string. BREAD performs the opposite function as BPLOT; the two statements are often used cooperatively for creating and storing CRT dot graphics.

Byte Plotting—BPLOT

BPLOT addresses CRT dots from the current pen position, plotting across the row to the right. Successive BPLOT statements plot rows of CRT dots stacked from top to bottom unless the pen is repositioned by another plotting or positioning statement. If the byte-plotted information extends past the right edge of the CRT's physical limits, BPLOTting wraps around to the left edge of the physical limits and drops down one row of dots. Byte plots can't be reflected.

The BPLOT statement reads the character string expression and interprets each character's binary code (an eight-digit binary number) as the on/off status of eight CRT dots. A "1" in the binary code indicates that a dot is plotted; a "0" indicates that no dot is plotted.

BPLOT *string expression , bytes per row*

With the BPLOT statement, characters and bytes are synonymous. One character specifies the on/off status of eight dots on the graphics display. The string expression contains multiple bytes of information that translates into patterns of dots. The bytes per row parameter specifies the number of characters (bytes) per row; it can be a number, variable, or expression. If the bytes per row parameter is positive, the BPLOT statement performs an exclusive or with the existing dots on the CRT screen. If it is negative, the dots are plotted on top of the existing dots on the graphics display. When the specified number of bytes per row are plotted, BPLOT repositions the pen to the left edge, one row below the previous byte-plotted row. BPLOT continues to plot dots until the entire character string is converted to CRT dots.

For example, BPLOT A$,15 plots 15 characters (15 groups of eight dots) *per row* of dots on the CRT until all of the characters in A$ have been plotted. If A$ contained 64 characters, BPLOT A$,15 would produce a byte plot of four rows of 120 (15 × 8) dots plus one row of 32 (4 × 8) dots.

BPLOT can begin at any dot postion. The starting dot position for BPLOT is determined in two ways:

- If the most recent pen movement was directed by a BPLOT statement, then the next BPLOT string begins at the left edge of and one row below the last byte-plotted string.
- If the most recent pen movement was directed by any statement other than BPLOT, then the BPLOT begins at the current pen position (the closest dot).

The BPLOT statement doesn't affect the pen position for other plotting operations. However, all the other plotting statements which move the pen affect the location of the byte plotted information.

Examples:

BPLOT T$,1	Byte plots T$ using 1 character per row of dots.
BPLOT "a+2B=jsV",-4	Byte plots the string using 4 characters per row of dots. Dots are plotted on top of the existing CRT dots, without an exclusive or.

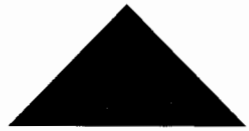
Building the BPLOT String

The procedure for building a BPLOT string is summarized below.

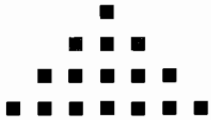
1. Draw the figure you wish to plot.
2. Redraw the figure in matrix form, using dot patterns instead of lines. Graph paper is useful for this task; let each square equal one dot, or one bit of information.
3. Divide the dot figure into columns of dots and spaces, eight squares wide. View each eight blocks as a byte of information where each block specifies a bit. If a dot is specified, the value of the block is one; if no dot is specified, the block's value is zero. Each group of eight dots or spaces specifies a binary number that determines a particular character.
4. Convert each binary number to its decimal equivalent.
5. Build the character string by assigning the character of the specified decimal value (using the CHR$ function) to the appropriate character position in the string. One approach is to write a program that accepts and appends the character to the string through INPUT statements or READ and DATA statements.
6. Use this string with the BPLOT statement to plot the figure.

Example: Build the BPLOT string for a triangle.

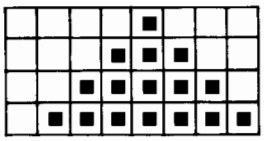
Step 1. Draw the figure.



Step 2. Represent the figure with dots or blocks.

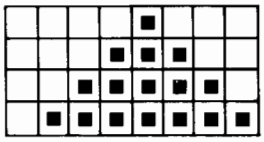


Step 3. Since the base of the triangle is seven dots wide, place it in a four by eight dot matrix.



Each row of the dot matrix specifies a byte (eight bits) of information.

Step 4. Convert each row of the matrix to a decimal value.



Binary Representation	Decimal Value
0 0 0 0 1 0 0 0	8
0 0 0 1 1 1 0 0	28
0 0 1 1 1 1 1 0	62
0 1 1 1 1 1 1 1	127

Step 5. Build the character string using the CHR$ function:

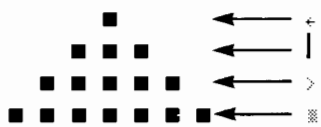
```
10 DIM T$[4] !          Dimensions the string variable.
20 FOR I=1 TO 4 !       Loop reads the decimal values into the appropriate
21  !                   character position in the string.
30 READ V
40 T$[I,I]=CHR$ (V)
50 NEXT I !             End loop.
51 ! ******* Data statement contains the decimal codes for the BPLOT string.
60 DATA 8,28,62,127
70 END
```

From the keyboard, execute T$ to display the BPLOT string:

```
T$
←|>▒
```

Step 6. Use the string with the BPLOT statement to plot the figure. The drawings below represent the outcome of the listed BPLOT statements. BPLOT T$,1 produces a triangle because it plots one character per line. BPLOT T$,4 plots all four characters on the same line.

```
BPLOT T$,1
```



Plots one character per line of T$=" ←|>▒ ", thus producing a triangle.

```
BPLOT T$,2
```

Plots two characters per line of T$=" ←|>▒ ".

BPLOT T$,3

Plots three characters per line of T$=" ←|>▒ ".

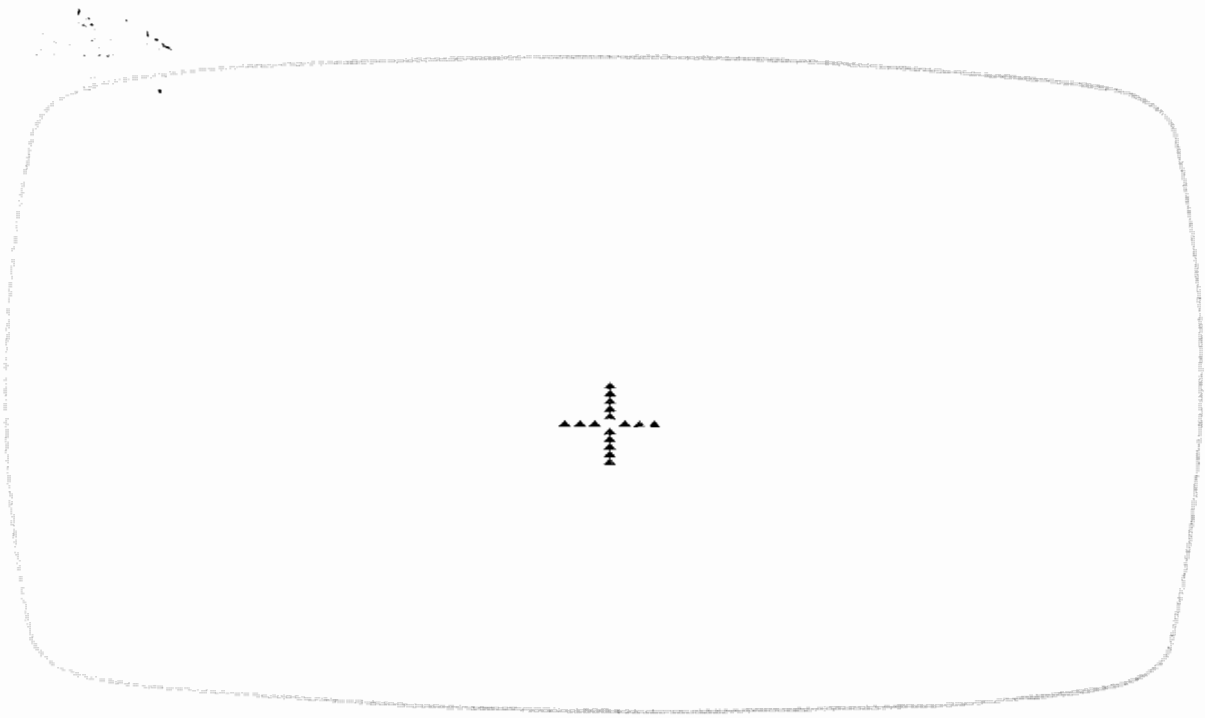
BPLOT T$,4

Plots four characters per line of T$=" ←|>▒ ".

Using the String With BPLOT

Example: The following program uses the string representing the triangle in a series of BPLOT statements, drawing a figure composed of triangles.

```
10 ! *** BPLOT Triangles ***
20 PLOTTER IS 1 @ GCLEAR
30 SCALE 1,400,1,240 !                    Scales the plotting area in dots.
31 ! ******* Loop builds the BPLOT triangle string *******
40 FOR I=1 TO 4
50 READ V
60 T$[I,I]=CHR$ (V)
70 NEXT I
80 MOVE 200,120
81 ! ******* Loop plots a column of 11 BPLOT triangles ******
90 FOR I=1 TO 11
100 BPLOT T$,1
110 NEXT I
111 ! ******* Loop plots a row of 7 BPLOT triangles *******
120 FOR X=176 TO 224 STEP 8
130 MOVE X,100
140 BPLOT T$,1
150 NEXT X
160 DATA 8,28,62,127
170 END
```



The example above illustrates most of the facts you need to know about BPLOT. We enumerate them here:

1. Scale the CRT display from 1 to 400 in the horizontal direction (in *graph* mode) and from 1 to 240 in the vertical direction. (Scale from 1 to 544 in the horizontal direction and from 1 to 240 in the vertical direction while in *graph-all* mode). This allows you to keep track of the pen location with respect to individual CRT dots. Sometimes the one-to-one correspondence between CRT dots and the specified scale is not exact. This is most frequent at the boundary of the plotting area. Repositioning the plot will sometimes solve this problem.

2. If the BPLOT statement is executed several times without changing the original pen location, succesive BPLOT strings are plotted beneath each other.

3. When the pen is moved by any means other than the BPLOT statement (130 MOVE X,100 in the above example) the BPLOT string begins at the current pen location, plotting to the right until the bytes per row parameter is satisfied. Then BPLOT continues down one row of dots beginning at the same column until the BPLOT string is completely plotted.

4. BPLOT performs an exclusive or with existing dots on the display when the bytes per row parameter is positive. In the above example the middle triangle was erased by plotting it twice. Run the same program but with a negative bytes per row parameter in statement 140 (140 BPLOT T$,-1); the middle triangle is not erased (no exclusive or is performed).

The table below illustrates all possible conditions and outcomes of the exclusive or operation between a dot on the display and the same dot specified by a BPLOT string. The third and fourth columns give the resulting dot condition; 0 means the dot is off and 1 means the dot is on.

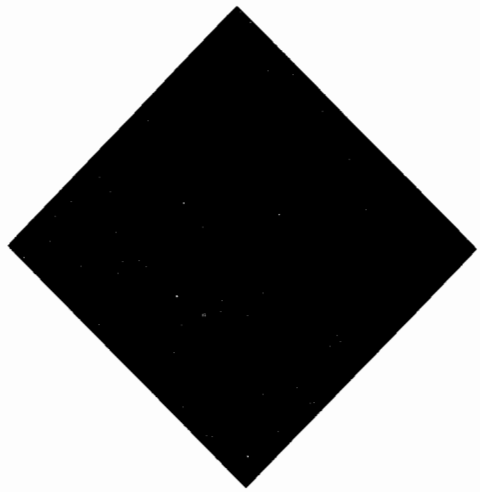
Dot Before BPLOT	Same Dot Specified By BPLOT String	Positive Row Parameter (Exclusive Or): Resulting Dot Condition	Negative Row Parameter (No Exclusive Or): Resulting Dot Condition
1 (on)	1	0 (off)	1 (on)
1 (on)	0	1 (on)	0 (off)
0 (off)	1	1 (on)	1 (on)
0 (off)	0	0 (off)	0 (off)

BPLOT Animation

Figures created with BPLOT strings can be made to move across the display by shifting the pen location between BPLOT statements.

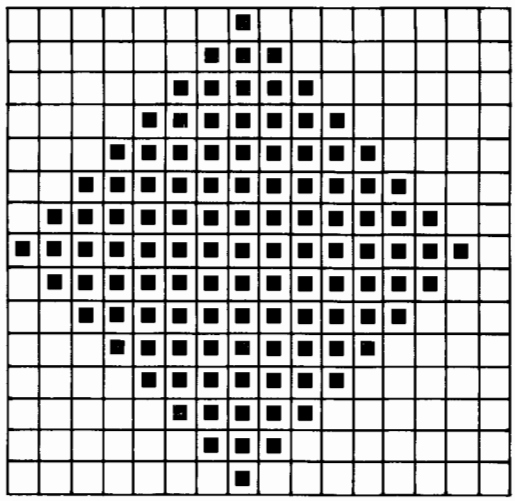
Example: Create a two-byte wide diamond and move it across the display along the x-axis. Move the figure one byte at a time so there is some overlap between successive diamonds.

1. First draw a diamond.



2. Now redraw the figure on graph paper in matrix form.

Matrix Form



Binary Representation

0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0

3. Divide the dot figure into columns of dots and spaces eight squares wide. The diamond is two bytes or 16 squares wide. Convert the figure into its binary representation.

4. Convert each eight-digit binary number to its decimal equivalent.

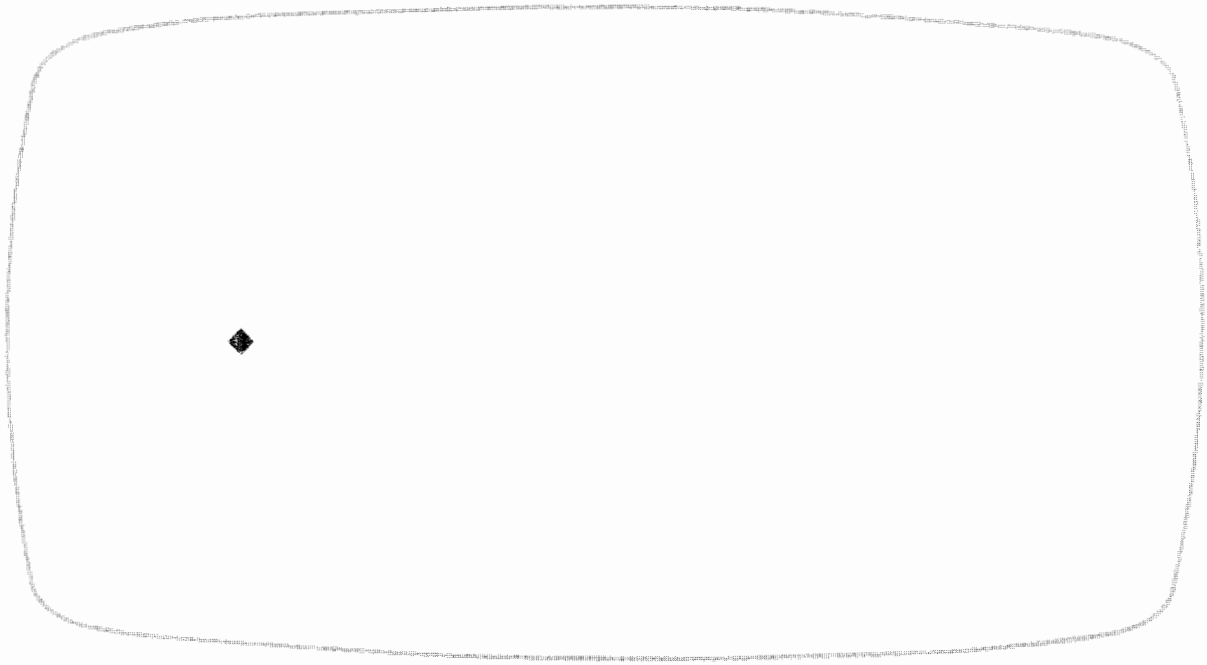
Binary Representation		Decimal Value	
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	0
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	3	128
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	7	192
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	15	224
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	31	240
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	63	248
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	127	252
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	255	254
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	127	252
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	63	248
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	31	240
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	15	224
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	7	192
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	3	128
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	0

5. Build the character string using the CHR$ function.

```
10 ! *** BPLOT Diamond ***
20 PLOTTER IS 1 @ GCLEAR
30 SCALE 1,400,1,240
40 DIM D$[30] !                        Dimensions string to 15 rows times 2 bytes
41 !                                   per row equals 30 bytes or characters.
42 ! ******* Loop builds BPLOT string for diamond *******
50 FOR I=1 TO 30
60 READ V
70 D$[I,I]=CHR$ (V)
80 NEXT I
90 DATA 1,0,3,128,7,192,15,224,31,240,63,248,127,252,255,254,127,252,63,248,31,2
40,15,224,7,192,3,128,1,0
100 END
```

6. Use this string with the BPLOT statement to plot the diamond.

```
10 ! *** BPLOT Diamond ***
20 PLOTTER IS 1 @ GCLEAR
30 SCALE 1,400,1,240
40 DIM D$[30] !                        Dimensions string to 15 rows times 2 bytes
41 !                                   per row equals 30 bytes or characters.
42 ! ******* Loop builds BPLOT string for diamond *******
50 FOR I=1 TO 30
60 READ V
70 D$[I,I]=CHR$ (V)
80 NEXT I
90 DATA 1,0,3,128,7,192,15,224,31,240,63,248,127,252,255,254,127,252,63,248,31,2
40,15,224,7,192,3,128,1,0
100 MOVE 1,120
110 BPLOT D$,2 !                       Byte-plots the character string, 2 bytes
111 !                                  or characters per row.
120 END
```



In order to move the diamond across the display to the right, one byte at a time, we need to erase the left half of the diamond while plotting the shifted diamond. This is easily accomplished by adding a matrix of black dots 8 dots wide by 15 dots high to the left of the diamond figure. If the exclusive or function is disabled (by using a negative bytes per row parameter), this matrix of black dots (zeros) erases any white dots (ones) which are beneath it while BPLOTting.

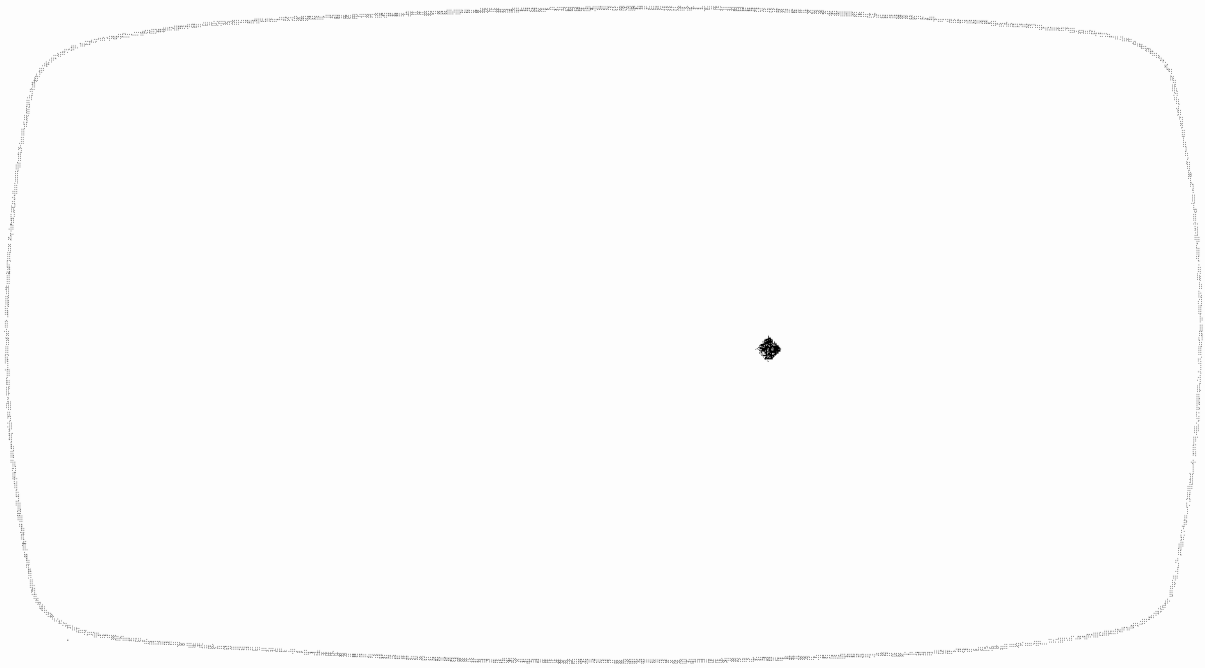
Original Diamond Figure		Decimal Value	
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	0
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	3	128
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	7	192
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	15	224
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	31	240
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	63	248
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	127	252
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	255	254
0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	127	252
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	63	248
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	31	240
0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	15	224
0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	7	192
0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	3	128
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	0

Shifted Diamond Figure			Decimal Value		
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	0	1	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	0	3	128
0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	0	7	192
0 0 0 0 0 0 0 0	0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	0	15	224
0 0 0 0 0 0 0 0	0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	0	31	240
0 0 0 0 0 0 0 0	0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	0	63	248
0 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	0	127	252
0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0	255	254
0 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1	1 1 1 1 1 1 0 0	0	127	252
0 0 0 0 0 0 0 0	0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0	0	63	248
0 0 0 0 0 0 0 0	0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0	0	31	240
0 0 0 0 0 0 0 0	0 0 0 0 1 1 1 1	1 1 1 0 0 0 0 0	0	15	224
0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 1	1 1 0 0 0 0 0 0	0	7	192
0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1	1 0 0 0 0 0 0 0	0	3	128
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	0	1	0

The character string for the moving diamond is three bytes wide. The first byte in each row erases the right half of the diamond just plotted. The second and third bytes plot the diamond again, shifted over one byte.

```
10 ! *** Moving Diamond ***
20 PLOTTER IS 1 @ GCLEAR
30 SCALE 1,400,1,240
40 DIM D2$[45] !                        Dimensions string to 15 rows times 3 bytes
41 !                                    per row equals 45 bytes or characters.
42 ! ******* Loop builds BPLOT string for moving diamond *******
50 FOR J=1 TO 45
60 READ W
70 D2$[J,J]=CHR$ (W)
80 NEXT J
90 DATA 0,1,0,0,3,128,0,7,192,0,15,224,0,31,240,0,63,248,0,127,252,0,255,254,0,1
27,252,0,63,248,0,31,240,0,15,224,0,7,192,0,3,128,0,1,0
100 MOVE 1,120
110 FOR P=1 TO 50
120 BPLOT D2$,-3 !                      Byte plots the character string, 3 bytes
121 !                                   or characters per row, without an
122 !                                   exclusive or on existing dots.
130 IMOVE 8,0
140 WAIT 150
150 NEXT P
160 END
```

The diamond moves across the screen from left to right.



Byte Reading—BREAD

The BREAD (*byte read*) statement performs the opposite of BPLOT: it reads groups of eight dots from the graphics display and stores them as characters in a string variable. The byte reading begins at the current pen position and moves down one row of dots after reading the specified number of bytes per row. The BREAD statement continues to read bytes across and down—building the character string until the string variable has reached its allocated length. Recall that strings longer than 18 characters must be allocated memory through a DIM statement.

BREAD *string variable* , *bytes per row*

The bytes per row parameter can be a number, variable, or expression; negative bytes per row are interpreted as their absolute value. Like the BPLOT statement, BREAD wraps around from the right edge of the physical limits to the left edge. BREAD does not affect the pen location for any plotting operation other than BREAD and BPLOT.

Example:

```
BREAD St$,32
```

Byte reads the CRT, 32 characters or 8 × 32 equals 256 dots at a time.

Example: By dimensioning a string variable to the proper size it is possible to BREAD the entire CRT graphics display. The *graph* mode display contains 240 rows of 400 dots, or equivalently, 240 rows of 50 bytes or a total of 240 × 50 = 12000 bytes of information. In the following program, a figure is drawn using the IDRAW statement in a FOR . . . NEXT loop. The graphics display is then BREAD into a string variable. Plot the figure again using the BPLOT statement and the byte read string. Note how much quicker it is to BPLOT the string than to use the conventional data plotting approach.

```
10 ! *** Bread ***
20 PLOTTER IS 1
30 GCLEAR @ DEG
40 SCALE -36000,36000,-36000,36000
50 MOVE 0,0
60 FOR ang=0 TO 36000 STEP 91 !          Loop for plotting hyperbolic spiral.
70 IDRAW ang*COS (ang),ang*SIN (ang)
80 NEXT ang !                            Repeats loop for next angle.
90 DIM D$[12000] !                       Dimensions string for entire graphics
91 !                                     display.
100 MOVE -36000,36000
110 BREAD D$,50 !                        Byte reads entire graphics display.
120 END
```

The string variable D$ contains all the information from the graphics display. From the keyboard, execute:

```
GCLEAR
MOVE -36000,36000
BPLOT D$,50
```

The figure is cleared from the display and then byte-plotted according to BREAD D$,50.

The BLINK and NOBLINK Statements

The BLINK and NOBLINK statements provide compatibility with the HP-83/85, allowing BPLOT programs written for the HP-83/85 to be run on your computer.

```
BLINK
```

```
NOBLINK
```

The BLINK and NOBLINK statements have no effect on CRT graphics operations with your computer.

The CURSOR and WHERE Statements

The CURSOR and WHERE statements perform the same operation for CRT graphics. Both statements assign the pen position coordinates to the *x* and *y* variable parameters and the pen status (0 if the pen is up, 1 if the pen is down) to the pen status variable.

CURSOR *x variable* , *y variable* [, *pen status variable*]

WHERE *x variable* , *y variable* [, *pen status variable*]

The pen position coordinates are interpreted according to the current units and can define any position inside *or* outside the plotting area. The pen status variable is an optional parameter. All three parameters must be numeric variables.

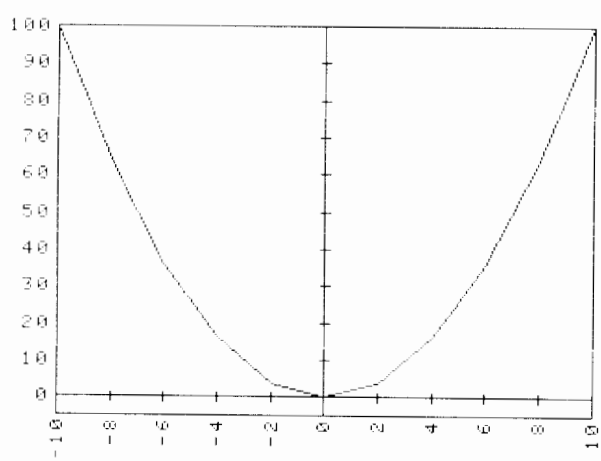
Example:

CURSOR X,Y,P	Assigns the current pen position and pen status to the variables X, Y, and P (*x*-coordinate is assigned to X, *y*-coordinate is assigned to Y, pen status is assigned to P).

The CURSOR and WHERE statements perform different functions when interpreted by the Plotter ROM for use with external plotters. Refer to the Plotter ROM owner's manual for further discussion of the CURSOR and WHERE statements.

Example: In the following example, the WHERE statement is used to find the pen position after a LABEL statement. This allows you to position a series of labels in columnar form as the curve is plotted.

```
10 ! *** Where ***
20 PLOTTER IS 1
30 GCLEAR
40 LOCATE 15,135,10,95
50 FRAME
60 FXD 0
70 SCALE -10,10,-5,100
80 LAXES 2,10,0,0
90 MOVE 11,100 !                    Moves pen to column head position.
100 LABEL " X   Y" !                Labels column heading.
110 WHERE A,B !                     Assigns pen position to A,B.
120 MOVE -10,100 !                  Moves to start of curve.
130 FOR X=-10 TO 10 STEP 2 !        Loop plots parabola and lists X,Y values.
140 Y=X^2
150 DRAW X,Y
160 MOVE A,B !                      Moves pen to position found by WHERE.
161 ! ******* Labels last drawn parabola coordinates *******
170 LABEL USING "3D,X,3D" ; X,Y
180 WHERE A,B !                     Reassigns pen position to A,B.
190 MOVE X,Y !                      Moves back to parabola plot.
200 NEXT X !                        End loop.
210 END
```



X	Y
-10	100
-8	64
-6	36
-4	16
-2	4
0	0
2	4
4	16
6	36
8	64
10	100

Notes

Section 19

Graphics Programming Applications

The example programs in this section demonstrate some of your computer's graphics capabilities. They are intended to familiarize you with the applications; you may want to adapt some of the programs according to your own needs.

Reflected Plots

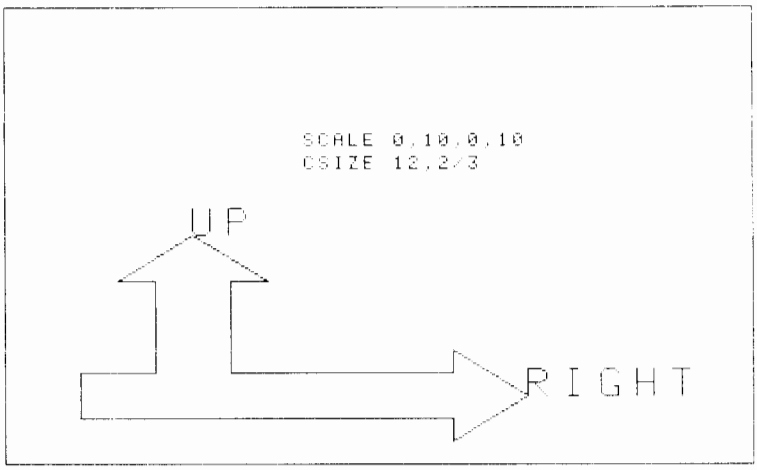
By exchanging parameters in the LIMIT, LOCATE, SCALE, or SHOW statements it's possible to produce a reflected image of plotted data and axes. Labels are reflected by using negative parameters in the CSIZE statement. Reflected plots are useful for various applications where special visual effects are required. For example, if you wish to draw on the back side of clear plastic to produce a normal image when read from the front, you would need to plot a mirror image of the desired plot. The three possible types of reflections are summarized below. Each reflection requires a different exchange of parameters in the LIMIT, LOCATE, SCALE, or SHOW statement and a different use of negative parameters in the CSIZE statement. The SCALE statement is used below as an example in reflecting data and axes. The same effect is achieved by exchanging parameters in the LIMIT, LOCATE, or SHOW statement.

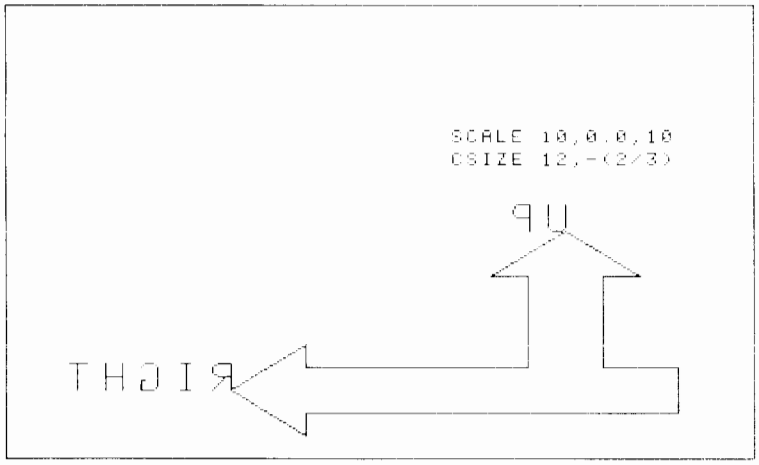
1. Unreflected plot:
   SCALE *x min* , *x max* , *y min* , *y max*
   CSIZE *height* , *aspect ratio*
2. Reflection across the *y*-axis (mirror image):
   a. Reflect axes and data.
      SCALE *x max* , *x min* , *y min* , *y max*
   b. Reflect labels.
      CSIZE *height* , −*aspect ratio*
3. Reflection across the *x*-axis:
   a. Reflect axes and data.
      SCALE *x min* , *x max* , *y max* , *y min*
   b. Reflect labels.
      CSIZE −*height* , −*aspect ratio*
4. Reflection across the origin:
   a. Reflect axes and data.
      SCALE *x max* , *x min* , *y max* , *y min*
   b. Reflect labels.
      CSIZE −*height* , *aspect ratio*

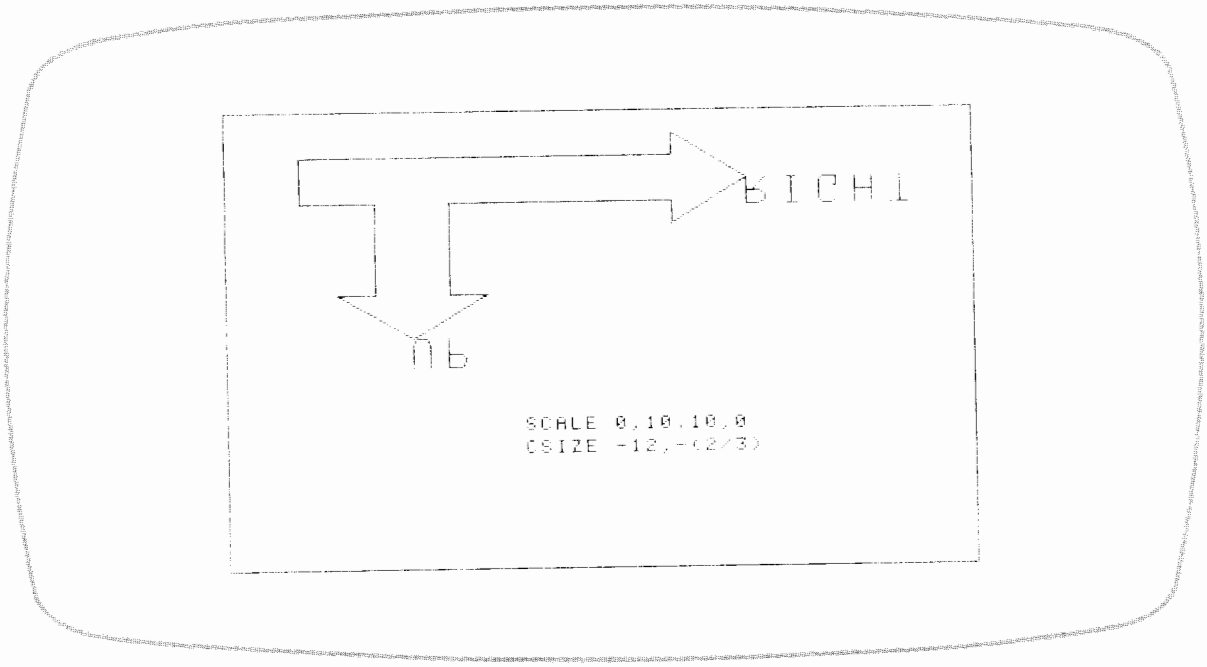
When more than one of the plot-reflecting statements (LIMIT, LOCATE, SCALE, and SHOW) are used in the same program, the net reflected result is the combination of reflections generated by the individual statements. For example, if you exchange the *x* min and *x* max parameters in both the LIMIT statement and the SCALE statement, and leave the LOCATE parameters unchanged, the net result is no noticeable change in the plot. In this case, the LIMIT parameters reflect the plot and the SCALE parameters reflect the already reflected plot, producing the original plot.

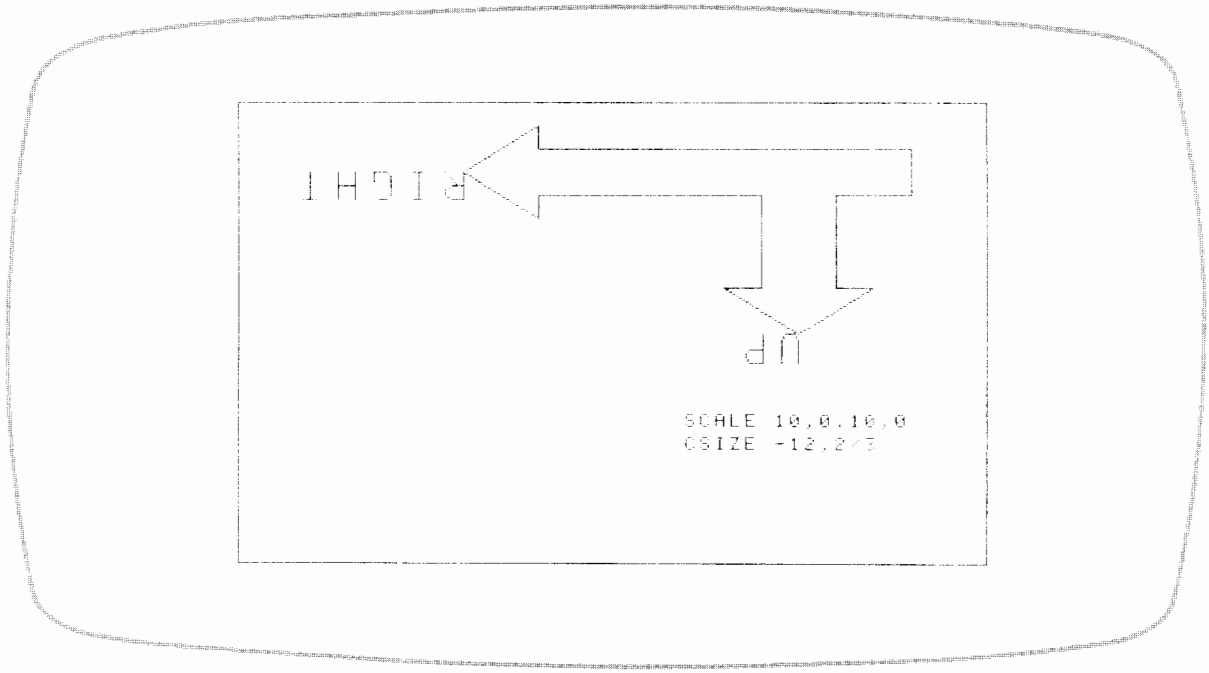
The following program demonstrates the reflection of plots and labels by first drawing the unreflected plot and labels, and then drawing the same figure and labels using a different SCALE and CSIZE statement.

```
10 ! *** Reflected Plots ***
20 PLOTTER IS 1
30 GCLEAR
31 !
32 ! ******* Normal plot and labels *******
40 SCALE 0,10,0,10 !  Specifies user units: parameters in normal sequence.
50 CSIZE 12,2/3 !     Assigns CSIZE positive height and positive aspect ratio.
60 S$="SCALE 0,10,0,10" @ C$="CSIZE 12,2/3"
70      GOSUB Plotting_Subroutine
71 !
72 ! ******* Reflection across y-axis *******
80 SCALE 10,0,0,10 !  Specifies user units: x min and x max parameters are
81 !                  exchanged.
90 CSIZE 12,-(2/3) !  Assigns CSIZE positive height and negative aspect ratio.
100 S$="SCALE 10,0,0,10" @ C$="CSIZE 12,-(2/3)"
110      GOSUB Plotting_Subroutine
111 !
112 ! ******* Reflection across x-axis *******
120 SCALE 0,10,10,0 ! Specifies user units: y min and y max parameters are
121 !                 exchanged.
130 CSIZE -12,-(2/3) ! Assigns CSIZE negative height and negative aspect ratio.
140 S$="SCALE 0,10,10,0" @ C$="CSIZE -12,-(2/3)"
150      GOSUB Plotting_Subroutine
151 !
152 ! ******* Reflection across origin *******
160 SCALE 10,0,10,0 ! Specifies user units: x min and x max, y min and y max
161 !                 parameters are exchanged.
170 CSIZE -12,2/3 !   Assigns CSIZE negative height and positive aspect ratio.
180 S$="SCALE 10,0,10,0" @ C$="CSIZE -12,2/3"
190      GOSUB Plotting_Subroutine
200 END
201 !
202 !
1000 Plotting_Subroutine:
1010 GCLEAR @ FRAME
1020 MOVE 1,1
1030 FOR P=1 TO 14
1040 READ X,Y
1050 DRAW X,Y
1060 NEXT P
1070 DATA 6,1,6,.5,7,1.5,6,2.5,6,2,3,2,3,4,3.5,4,2.5,5,1.5,4,2,4,2,2,1,2,1,1
1080 RESTORE
1090 MOVE 7,1.5
1100 LABEL "RIGHT"
1110 MOVE 2.5,5
1120 LABEL "UP"
1130 CSIZE 5
1140 MOVE 4,7
1150 LABEL USING "K/K" ; S$,C$
1160 WAIT 4000
1170 RETURN
1180 END
```


SCALE 0,10,0,10
CSIZE 12,2/3
UP
RIGHT


SCALE 10,0,0,10
CSIZE 12,-(2/3)
UP
RIGHT


SCALE 0,10,10,0
CSIZE -12,-(2/3)


SCALE 10,0,10,0
CSIZE -12,2/3

Keyboard Plotting

When used in conjunction with plotting statements, the ON KEY# statement enables you to generate graphics from the keyboard. In the following program the user-defined keys are assigned subroutines which control moving and drawing operations and pen color. The graphics display is scaled according to the matrix of CRT dots; an *x,y* coordinate is assigned to each individual dot in the *graph* mode plotting area (SCALE 1,400,1,240). This allows you to plot any individual CRT dot. In the example program on page 254, the user-defined keys are assigned the following capabilities.

ON KEY# 1: Draws a line to the left, N dots long (N is specified by ON KEY# 6).

ON KEY# 2: Draws a line to the right, N dots long.

ON KEY# 3: Draws a line down, N dots long.

ON KEY# 4: Draws a line up, N dots long.

ON KEY# 5: Selects PEN 1 (plots white dots).

ON KEY# 6: Selects the increment, N, for plotting and moving the pen (in number of dots). Each time the key is pressed the increment is increased by 1, up to 20 dots. The increment value begins at N = 1. The current increment value is shown at the top of the alpha display.

ON KEY# 8: Moves the pen N dots to the left.

ON KEY# 9: Moves the pen N dots to the right.

ON KEY# 10: Moves the pen N dots down.

ON KEY# 11: Moves the pen N dots up.

ON KEY# 12: Selects PEN -1 (plots black dots). This key can be used to change pen color and erase individual dots and lines.

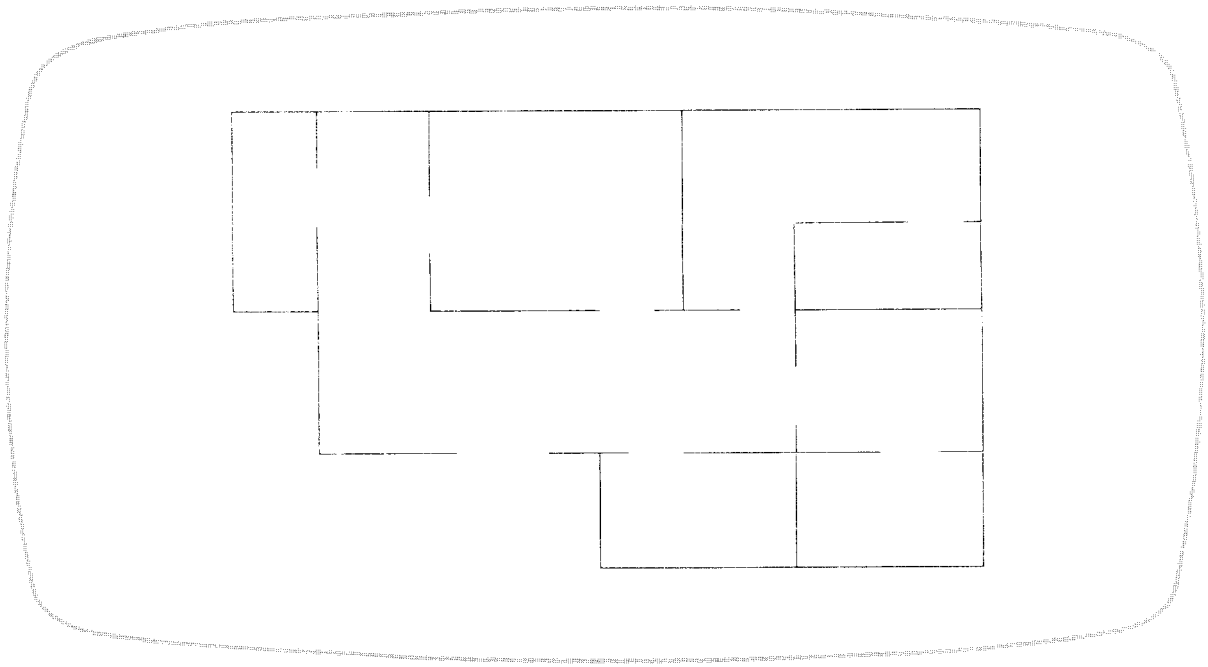
ON KEY# 13: Clears the graphics display (GCLEAR).

Use the (A/G) key to shift back and forth between the graphics and alpha displays to view your plot and recall the key labels.

You can store your plot for future reference by using the GSTORE statement (refer to section 21).

The example graphics design on page 255 was created using 20 dot increments. Keyboard plotting was used to design the floor plan for a house.

```
10 ! *** Keyboard Plotting ***
20 PLOTTER IS 1
30 GCLEAR
40 I=1 !                                  Sets dot increment equal to 1.
50 SCALE 1,400,1,240 !                    Scales the plotting area dot by dot.
60 ! ******* Key assignments *******
70 ON KEY# 1,"DRAW LEFT" GOSUB 250
80 ON KEY# 2,"DRAW RIGHT" GOSUB 270
90 ON KEY# 3,"DRAW DOWN" GOSUB 290
100 ON KEY# 4,"DRAW UP" GOSUB 310
110 ON KEY# 5,"PEN 1" GOSUB 330
120 ON KEY# 6,"DOT INCRMT" GOSUB 470
130 ON KEY# 8,"MOVE LEFT" GOSUB 350
140 ON KEY# 9,"MOVE RIGHT" GOSUB 370
150 ON KEY# 10,"MOVE DOWN" GOSUB 390
160 ON KEY# 11,"MOVE UP" GOSUB 410
170 ON KEY# 12,"PEN-1" GOSUB 430
180 ON KEY# 13,"GCLEAR" GOSUB 450
190 CLEAR
200 DISP "INCREMENT=";I,"PRESS THE [A/G] KEY TO VIEW THE PLOT"
210 KEY LABEL
220 MOVE 200,120 !                        Moves the pen to the center of the
221 !                                     plotting area.
230 GOTO 230
240 END
250 IDRAW -I,0 !                          Draws I dots to the left.
260 RETURN
270 IDRAW I,0 !                           Draws I dots to the right.
280 RETURN
290 IDRAW 0,-I !                          Draws I dots down.
300 RETURN
310 IDRAW 0,I !                           Draws I dots up.
320 RETURN
330 PEN 1 !                               Specifies PEN 1, plots white dots.
340 RETURN
350 IMOVE -I,0 !                          Moves the pen I dots to the left.
360 RETURN
370 IMOVE I,0 !                           Moves the pen I dots to the right.
380 RETURN
390 IMOVE 0,-I !                          Moves the pen I dots down.
400 RETURN
410 IMOVE 0,I !                           Moves the pen I dots up.
420 RETURN
430 PEN -1 !                              Specifies PEN -1, plots black dots.
440 RETURN
450 GCLEAR !                              Clears the graphics display.
460 RETURN
470 I=I+1 !                               Adds 1 to the dot increment.
480 IF I>20 THEN I=1 !                    Allows a maximum increment of 20 dots.
490 CLEAR @ KEY LABEL
500 DISP "INCREMENT=";I,"PRESS THE [A/G] KEY TO VIEW THE PLOT"
510 RETURN
520 END
```



Locating Windows

The following program enables you to change the proportions or the size of a plot by mapping your plot onto a LOCATE-defined plotting area. The area chosen to be resized is called a window. The contents of the window is expanded or shrunk to fit onto the default *graph* mode plotting area. The plot in this example is generated by the plotting subroutine beginning with statement number 1000. This subroutine could be replaced with any other subroutine which plots curves on the *graph* mode display. Labels are not proportioned or sized by the LOCATE procedure in this program; label size and aspect ratio are determined by the CSIZE statement.

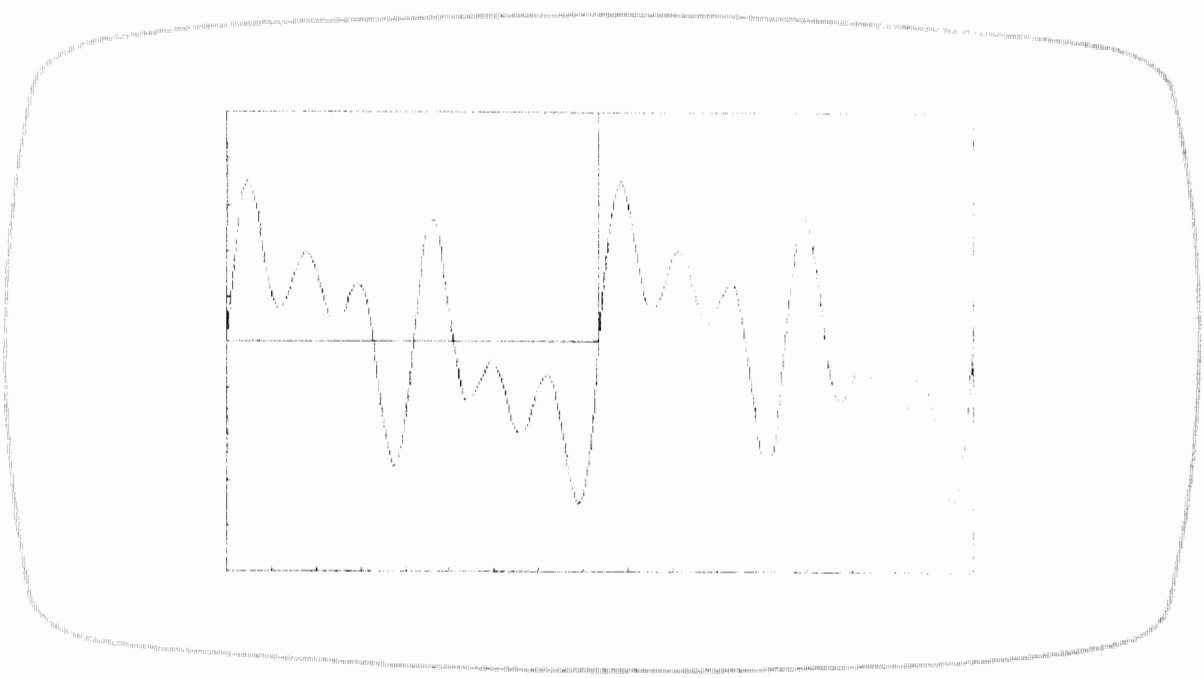
The windowing procedure in the example program below operates in the following manner:

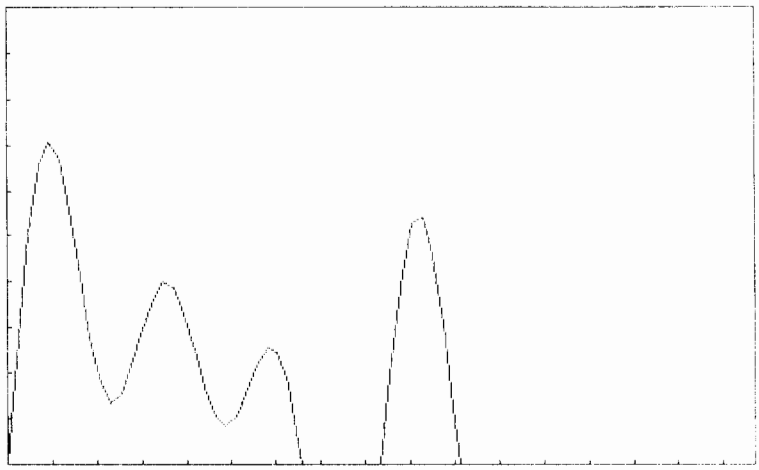
1. The original plot is drawn onto the default *graph* mode plotting area. The axes are drawn in GUs to facilitate locating the window; the tick-marks are spaced at 10 GU intervals.
2. The computer prompts you to input the boundaries for the LOCATE window in GUs: x min,x max,y min,y max. The window is framed for your reference. If the window boundaries fall within the graphics limits (x range: 0 to RATIO*100, y range: 0 to 100) the window is expanded to the size of the graphics limits. If the window boundaries are outside the graphics limits, the window is reduced in size and mapped onto the graphics limits.
3. The computer plots the window onto the default *graph* mode plotting area. The proportions of the plot are determined by the aspect ratio of the window. If the aspect ratio of the window is the same as the graphics limits ((x max − x min)/(y max − y min) = RATIO) the proportions of the resized plot are the same as the original plot.

Two example outputs of the LOCATE Windows program follow the program listing.

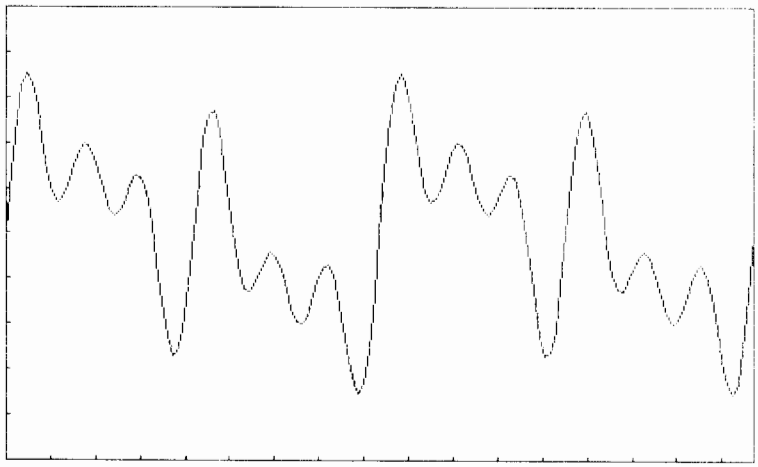
```
10 ! *** LOCATE Windows ***
20 PLOTTER IS 1
30 DEG @ GCLEAR
40 GOSUB 1000 !                    Goes to plotting subroutine.
50 WAIT 3000
60 R=RATIO *100
70 CLEAR
80 DISP "Enter the LOCATE boundaries xmin,xmax,ymin,ymax in GUs"
90 INPUT X1,X2,Y1,Y2 !             Inputs the LOCATE window boundaries in GUs.
100 LOCATE X1,X2,Y1,Y2 !           Locates the selected window.
110 FRAME !                        Frames the window.
120 WAIT 3000
121 ! ******* Calculates the parameters necessary to plot the selected
122 !   window onto the area bound by the default graphics limits *******
130 DX1=X1*R/(X2-X1)
140 DX2=R*(R-X2)/(X2-X1)
150 DY1=Y1*100/(Y2-Y1)
160 DY2=100*(100-Y2)/(Y2-Y1)
170 LOCATE -DX1,R+DX2,-DY1,100+DY2 !  Locates the plotting boundaries so
171 !                                 that the selected window is plotted
172 !                                 onto the area bound by the default
173 !                                 graphics limits.
180 GCLEAR
190 GOSUB 1000 !                   Goes to plotting subroutine.
200 END
1000 ! *** Plotting Subroutine ***
1010 FRAME
1020 SETGU @ AXES 10,10 !          Sets computer to graphics units mode and
1021 !                             plots axes with x and y tick-spacing = 10 GUs.
1030 SCALE -720,720,-4,4
1040 MOVE -720,0
1050 FOR A=-720 TO 720 STEP 10
1060 PLOT A,SIN (A/2)+SIN (A)+SIN (2*A)+SIN (3*A)
1070 NEXT A
1080 RETURN
1090 END
```

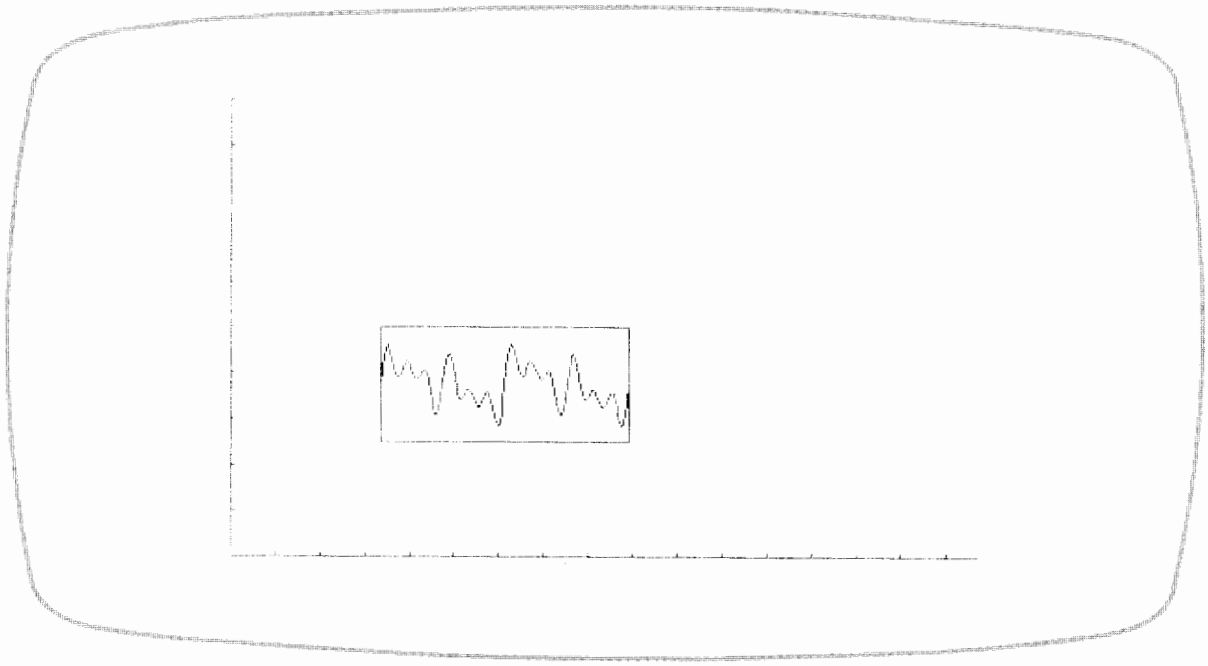
```
Enter the LOCATE boundaries xmin,xmax,ymin,ymax in GUs
?
0,50*RATIO,50,100
```





```
Enter the LOCATE boundaries xmin,xmax,ymin,ymax in GUs
?
-100,400,-100,300
```

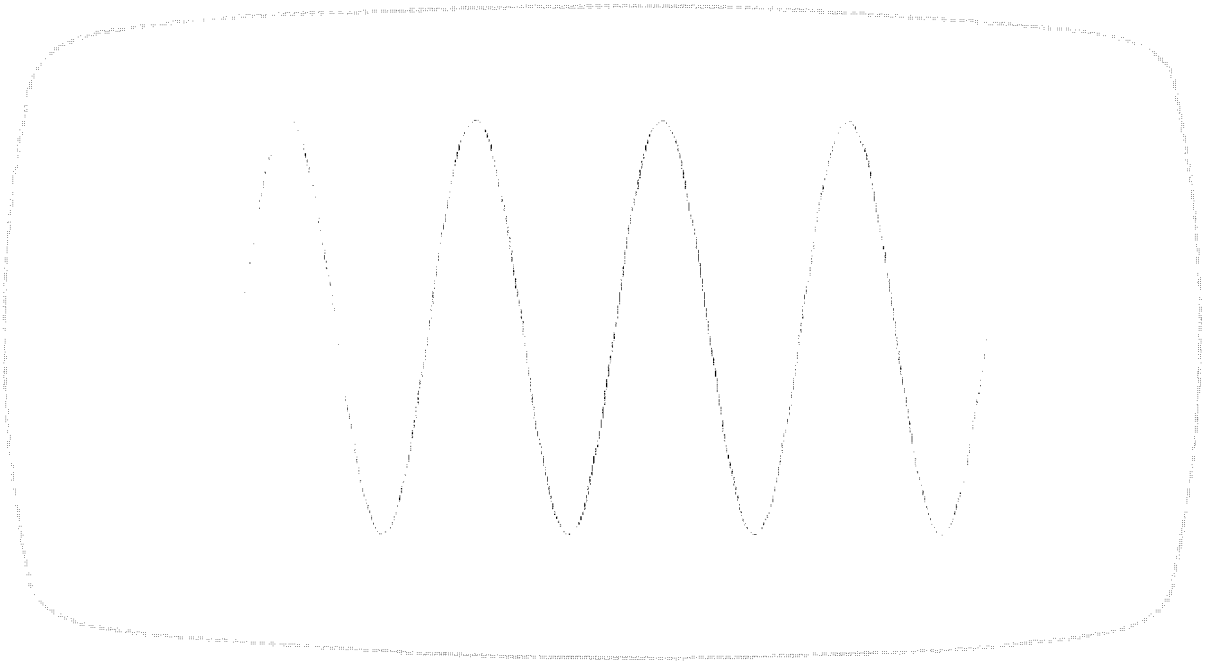




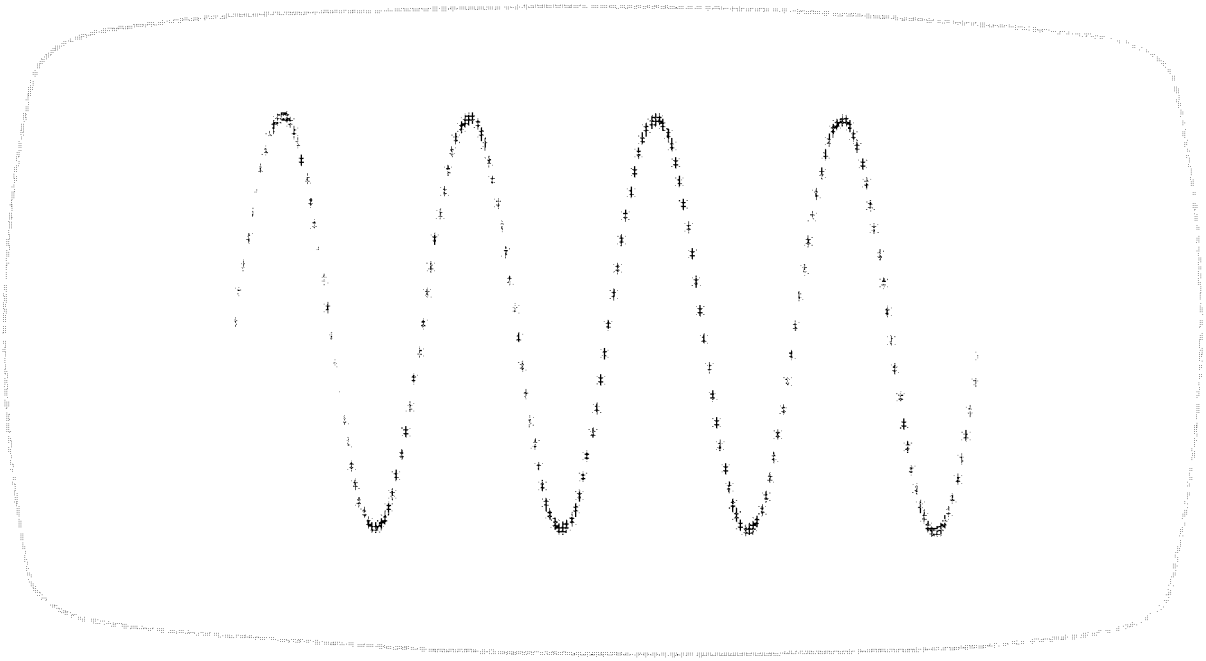
Plotting With Characters

Up to now the distinction between labeling and plotting data has been clear. Plotted data has been represented by points, lines, and curves. Labels have been used as an enhancement to plotted data or as text. However, it is also possible to plot data using individual characters or strings by replacing the PLOT statement with a MOVE and a LABEL statement. The following program allows you to choose a character or string and plots the function $y = \sin(4x)$ using the selected string. The function is first drawn as a curve using the PLOT statement, then drawn again with the selected character or string using the MOVE and LABEL statements. Plotting with characters allows you to enhance or individualize your data; it can be very useful when a series of curves are drawn on the same plotting area.

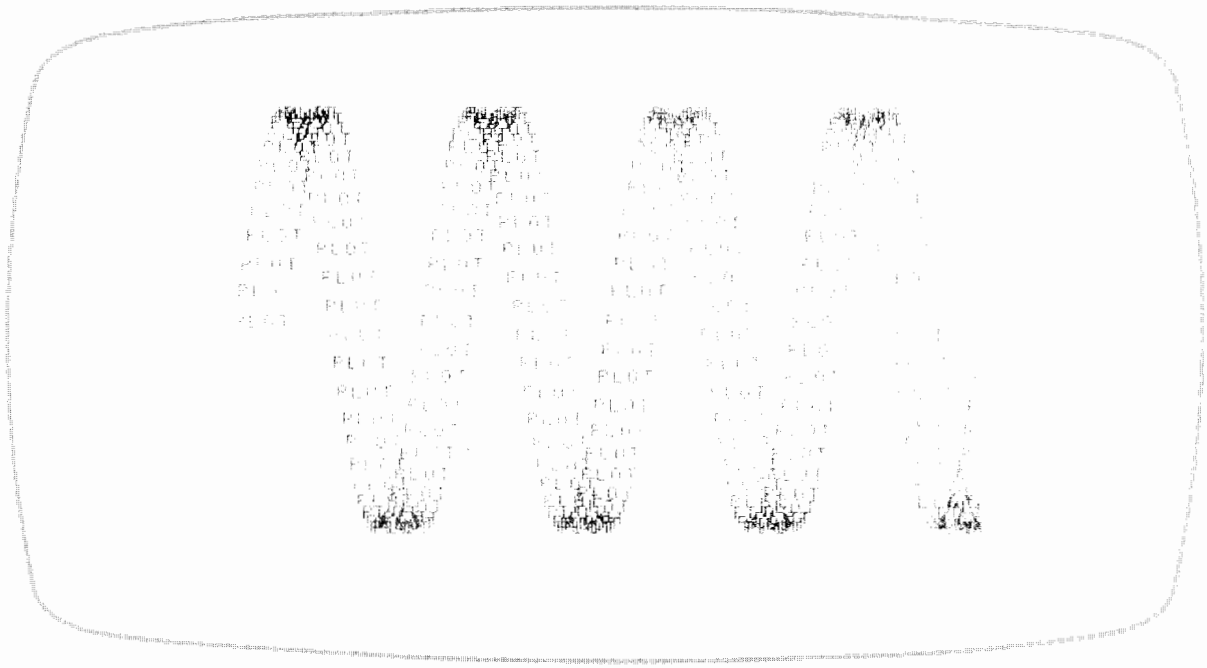
```
10 ! *** Character Plot ***
20 PLOTTER IS 1
30 GCLEAR @ DEG
40 SCALE 0,360,-1.1,1.1
50 FOR A=0 TO 360 STEP 2 !          Loop begins for plotting sin(4x).
60 PLOT A,SIN (4*A) !               Plots specified coordinate.
70 NEXT A !                         End loop.
80 WAIT 2000
90 GCLEAR @ CLEAR
100 DISP "Enter character string"
110 INPUT C$
120 FOR A=0 TO 360 STEP 2 !         Loop begins for character plotting sin(4x).
130 MOVE A,SIN (4*A) !              Moves pen to specified coordinate.
140 LABEL C$ !                      Labels string.
150 NEXT A !                        End loop.
160 WAIT 2000
170 GOTO 90
180 END
```



```
Enter character string
?
*
```



```
Enter character string
?
PLOT
```



Labeling Along a Curve

Another useful technique for enhancing plots with labels is made possible by the LDIR statement. The LDIR statement allows you to label in any direction by specifying the rise and run or the angle of the slope at which labels are drawn. The direction of the label can be varied character by character according to a predefined function by using variable parameters in the LDIR statement. The following program plots a series of curves and labels each individual curve with characters that follow the direction of the curve. The curves represent the function $y = \exp(-Tx)$ where T = .1, .3, .5, and 1.5. The label direction is varied by using the function for the slope of $y = \exp(-Tx)$ in the LDIR rise, run parameters. The rise to run ratio or slope of these curves can be calculated at any point along the curve from the first derivative of the exponential function, dy/dx.

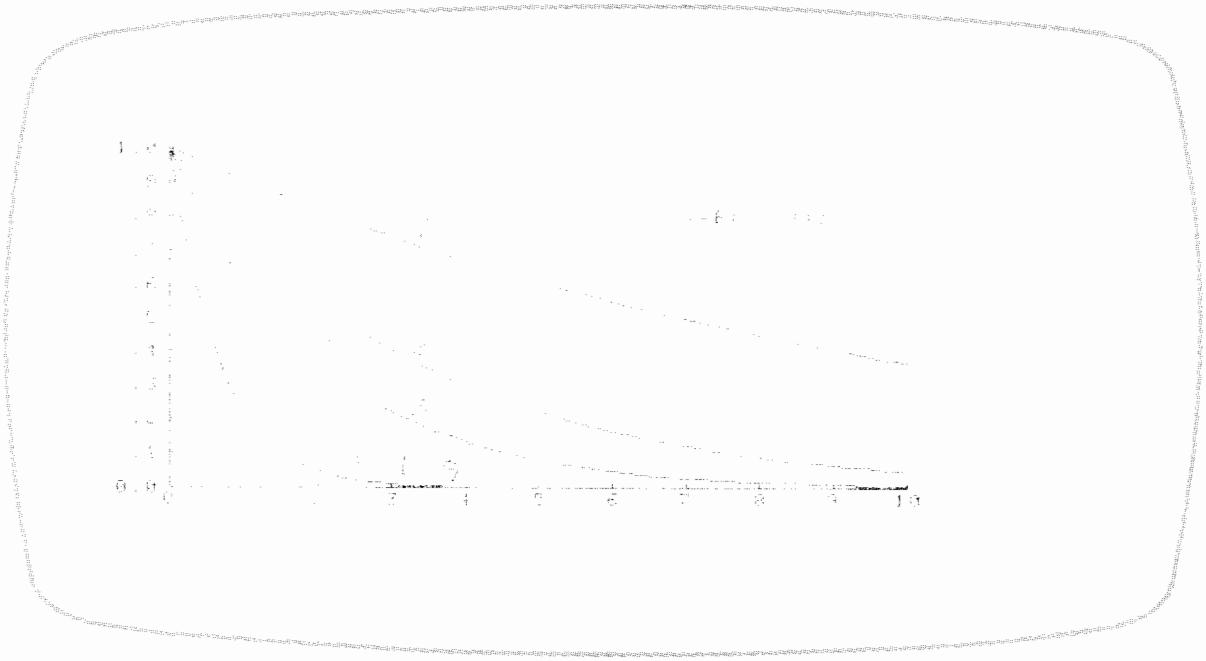
rise/run = slope = $dy/dx = -T\exp(-Tx)$

In order for the labels to match the slope of the curve at any given point the label direction is specified as follows:

LDIR *run*, *rise* = LDIR *dx*, *dy* = LDIR 1,-(T*EXP (-(T*X)))

Labeling along a curve makes your labels more distinctive while taking up less space on crowded plots.

```
10 ! *** Labeling on a Curve ***
20 PLOTTER IS 1
30 GRAPHALL
40 LOCATE 15,180,15,90
50 SCALE 0,10,0,1
60 FXD 0,1
70 LAXES -1,.1
80 MOVE 7,.8
90 LABEL "Y=EXP(-T*X)"
91 ! ******* Loop plots four curves of Y=EXP(-T*X), T=.1,.3,.5,1.5,
92 ! and labels each curve with the corresponding T value. *******
100 FOR C=1 TO 4
110 READ T
120 MOVE 0,1
130 CSIZE 8
140 FOR X=0 TO 10 STEP .2 !        Nested loop for plotting Y=EXP(-T*X).
150 PLOT X,EXP (-(T*X))
160 NEXT X !                       End loop.
170 L$="T =" !                     Assigns label to string L$.
180 I=0
190 FOR X=2 TO 2.75 STEP .25 !     Nested loop for labeling T value along
191 !                              the curve Y=EXP(-T*X).
200 I=I+1 !                        Counter for label characters.
210 MOVE X,EXP (-(T*X))+.06 !      Moves the pen .06 units above the curve.
220 LDIR 1,-(T*EXP (-(T*X))) !     Specifies label direction parallel to curve.
221 ! ******* Labels character by character in current label direction. *******
230 IF I=4 THEN LABEL T ELSE LABEL L$[I,I]
240 NEXT X !                       End loop.
250 NEXT C !                       End loop.
260 DATA .1,.3,.5,1.5
270 END
```



Part IV
Mass Storage Operations

Section 20

Accessing Your Mass Storage System

Introduction

Mass storage is a means of storing and retrieving information. If your computer applications require that information be retained when the machine is turned off or that quantities of information exceeding computer memory be manipulated, then mass storage is an essential capability of your system.

The computer's BASIC language includes a number of statements for communicating with a mass storage device (disc drive unit) which in turn accesses a mass storage medium (flexible disc). Among the operations available to you are:

- Storing programs for future use.
- Creating and accessing data files tailored to your particular computing needs.
- Storing and retrieving graphics displays.
- Copying files from one mass storage medium to another.
- Running programs whose memory requirements exceed available computer memory by storing individual program segments in mass storage and recalling them into computer memory one at a time.

Information is stored onto and retrieved from mass storage media as files. This section provides a general discussion of how to access any particular file in your mass storage system. Program and data files are discussed separately in sections 21 and 22, respectively.

Addressing Parameters

Accessing mass storage files requires that you understand the addressing system used by the computer to locate the desired file. The address of a particular file is defined by a series of parameters that progressively "zero in" on the file's location.

The addressing parameters are:

- The select code of the interface on which the disc drive unit is located.
- The device address of the disc drive unit.
- The number of the drive into which the flexible disc is inserted.
- The name assigned to the file.

The Select Code

The *select code* of an interface distinguishes that interface from any others connected to the computer. Each interface must have a unique select code. If your disc drive unit is connected using the computer's integrated interface, refer to the appropriate section of your computer owner's documentation for

information regarding the factory preset select code and, where applicable, for directions on changing the select code. If your disc drive(s) is (are) connected using an optional interface module, refer to documentation accompanying that module.

Examples in this manual assume an interface select code 7 for the interface to which your disc drive device is attached.

Device Address

The *device address* of a disc drive unit differentiates that unit from other devices on the same interface. When more than one "master" unit are present on the same interface, each must have a unique device address.

Refer to documentation accompanying your disc drive for information regarding the preset device address and for procedures for changing the device address, where applicable.

The examples in this manual assume that the device address of the disc drive being accessed is 0.

If your system contains an add-on unit attached to a master unit, the add-on unit has the same device address as the master unit.

Disc Drive Numbers

Disc drive numbers identify individual drives at a particular device address. These drives include both the master unit and the add-on, if applicable.

The drive numbers of most disc drive units appear on the front panel of the device. If necessary, refer to documentation accompanying the device for information regarding the number of drives that can be located at a particular address and for the drive numbers assigned to each drive.

Drive numbers of the HP 82900-Series Flexible Disc Drives are factory preset and cannot be changed by the user.

Throughout this manual, it is assumed that the drive numbers present at device address 0 are DRIVE 0 and DRIVE 1.

The Default Mass Storage Location

When the computer is switched on or reset, it automatically searches the HP-IB-type interface having the lowest select code for the disc drive unit having the lowest device address. The lowest numbered drive at that location is designated the default mass storage address. If no mass storage device is found on the interface, the computer determines if any other HP-IB-type interfaces are present and searches those in order of their select codes.

In order for the HP-87 to locate the default drive, the disc drive unit must be turned on before the computer is switched on or reset.

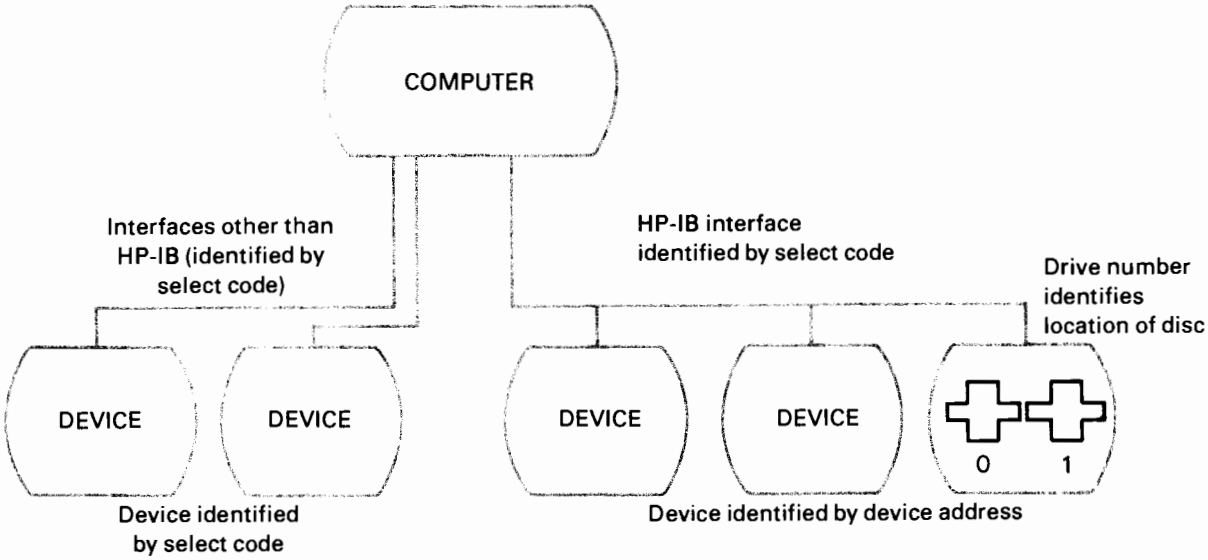
> Note: During the HP-86 mass storage search, the printer/disc interface is regarded as an HP-IB-type interface (select code 7), except that the search ends at the DRIVE 0 receptacle (device address 0, drive number 0). Therefore, the HP-86 will not locate a default disc drive at the DRIVE 1 receptacle or at a select code greater than 7.

The default drive can be changed by executing a `MASS STORAGE IS` statement. The syntax for that statement is discussed later in this section.

Throughout this manual, DRIVE 0 at device address 0 on an interface with select code 7 is assumed to be the default mass storage location.

Mass Storage Unit Specifier

The *mass storage unit specifier (msus)* is a character string that combines an interface select code, the address of the master unit, and the drive number to specify the location of a particular flexible disc on which a file is located.



The msus has the form:

" : *device type* [*interface select code* *device address* *drive number*]"

All msus character strings begin with a colon.

The device type identifies the type of mass storage device being accessed, either disc or tape. D specifies disc; T specifies an internal tape unit for compatibility with other HP Series 80 Personal Computers.

Examples: The following quoted strings are valid mass storage unit specifiers:

":D700"	":D701"	":D702"	":D703"
Disc drive unit	Disc drive unit	Disc drive unit	Disc drive unit
Select code 7	Select code 7	Select code 7	Select code 7
Device address 0	Device address 0	Device address 0	Device address 0
DRIVE 0	DRIVE 1	DRIVE 2	DRIVE 3

Volume Labels

Volume labels provide a convenient way to specify a particular disc.

A *volume label* is a name up to six characters in length that you assign to a disc when the disc is initialized or by executing a `VOLUME IS` statement. The volume label is stored on the disc and remains the disc's name until a new volume label is assigned to the disc. Once a volume label has been assigned to a disc, the disc can be accessed using its msus or its volume label.

An assigned volume label has the form:

```
".AAAAAA"
```

String up to six characters in length

Period preceding the string.

where *A* is any character. To avoid confusion, you should not use a period (.), a colon (:), or quotation marks (") as the first *A* character in the string.

The syntax of the `VOLUME IS` statement is:

```
VOLUME  ":msus"  IS  "new volume label"
        ".old volume label"
```

Note that the new volume label is not preceded by a period. However, once the volume label has been assigned, the string of characters (*AAAAAA*) that comprise the volume label must be preceded by a period.

When a volume label is used to access the medium on which information is stored, the system searches the discs currently in the system until the disc with that volume label is found. If the search fails to find the specified volume label, the computer returns `Error 125 : VOLUME`. Because of this search operation, it may take more time to access a file using the volume label than by using the msus.

Examples:

`VOLUME ":D700" IS "DRIVE0"`	Assigns volume label `".DRIVE0"` to the disc located at msus `":D700"`.
`VOLUME ".DRIVE0" IS "A/R"`	Renames the disc formerly labeled `".DRIVE0"` to `".A/R"`.

The examples in this manual assume the following volume label assignments:

```
VOLUME ":D700" IS "DRIVE0"
VOLUME ":D701" IS "DRIVE1"
```

Initializing a Flexible Disc

Each empty flexible disc must be initialized before it is used for the first time. The `INITIALIZE` statement sets up a file directory and clears and tests the disc. Optional parameters in the statement can be used to:

- Establish a volume label.
- Specify the amount of space allocated to the disc directory.
- Specify how the physical records on the disc are to be numbered.

The initialization process takes about two minutes. Any information stored on the disc is erased by the INITIALIZE statement. If you are uncertain whether or not a disc has been previously initialized, insert the disc into DRIVE 0 and type CAT (END LINE). The message Error 130 : DISC indicates that the disc has not been initialized.

The syntax of the INITIALIZE statement is:

```
INITIALIZE["new volume label" [, "  ":msus"  ,old volume label" [, directory size[, interleave factor]]]]
```

You cannot use a period (.) or colon (:) as the first character in the new volume label. You may not specify a null string as the new volume label.

> Note: Make certain you thoroughly understand the syntax of the INITIALIZE statement before using it. Remember that the first parameter is a new volume label and that the second parameter specifies the disc to be initialized. **If the disc to be initialized is located elsewhere than the default drive, you must assign a volume label to it during the initialization process.**

In the INITIALIZE statement, each optional parameter must be preceded by all the optional parameters listed before it. For instance, the directory size must be preceded by both a new volume label and a msus or old volume label.

The new volume label is the new name assigned to the flexible disc being initialized. If the disc being initialized is located in the default drive, the volume label, if omitted, defaults to blanks.

The msus or old volume label is the existing label or msus of the disc being initialized. If this parameter is omitted, the default disc specified by the MASS STORAGE IS statement is used.

The directory size specifies the number of records to be allocated on the disc for the file directory. Each record holds directory information for eight files. The default value is 14 records (or $14 \times 8 = 112$ files).

The *interleave factor* is an integer specifying how physical records on the disc are numbered. When the factor is 1, 2, 3, ...etc., records are numbered consecutively, by every other record, every third record, ...etc. The default value for the interleave factor is 6. Consult documentation accompanying your disc drive unit for the range of permissible values.

The ability to renumber records on a disc by specifying an interleave factor allows you to control the efficiency of your disc drives and to minimize the time required to access mass storage files.

The interleave factor affects how many revolutions of the disc are necessary to transfer information to and from mass storage. Because it takes a finite amount of time to perform accessing operations, and because the disc is spinning rapidly, it is possible that a full revolution might be required to access successive records on the disc. By placing a physical separation between records, the appropriate interleave factor can minimize the number of wasted revolutions.

The performance of your mass storage system during a particular application can be improved by adapting the interleave factor to the structure of your data. Since there is no easy way to compute the best interleave factor for a particular data configuration, the simplist way to determine the most efficient interleave factor is by trial and error.

One method for testing interleave factors involves copying data files accessed by a program from a "master" disc to a "test" disc that has been initialized to a different interleave factor. Then, time the execution of the program using the computer's internal timer. You may initialize the test disc repeatedly using a different interleave factor each time, COPY the same data onto the disc (remember, the data was lost when the disc was reinitialized), and rerun the program to compare execution times.

Examples:

`INITIALIZE`	Initializes the disc at the default location; no volume label is assigned.
`INITIALIZE "DRIVE1",":D701"`	Initializes the disc at ":D701" and assigns volume label ".DRIVE1".
`INITIALIZE "DRIVEA",".DRIVE1",15,2`	Initializes disc ".DRIVE1" and assigns new volume label ".DRIVEA". The directory consists of 15 records; the interleave factor is 2.

Establishing a New Default Mass Storage Location

At power-on, the computer automatically establishes the drive having the lowest numbered msus as the default mass storage device. The MASS STORAGE IS statement allows you to specify an alternative default address for mass storage operations.

```
MASS STORAGE IS  ":msus"
                 ".volume label"
```

If the volume label is specified, the drive at which that disc is located is designated the default address.

Once a default drive is set up, the system automatically uses that drive when accessing files unless you specify otherwise.

Examples:

`MASS STORAGE IS ".A/R"`	The default is set to the drive containing the disc with volume label ".A/R".
`MASS STORAGE IS ":D701"`	The default is set to msus ":D701".

Accessing Files Using the File Specifier

Data and programs are stored on a mass storage disc in *files*. By assigning each file a name, you can access previously stored information by using the appropriate BASIC statement to access that file.

The location of a file in your mass storage system is described by a file specifier consisting of two parts: a one- to ten-character file name, and a volume label or msus. The volume label or msus identifies the

particular disc drive on which the file is located. The file name distinguishes any one file from others stored on the same disc.

The form for the file specifier is:

```
"file name [ :msus
           , volume label ]"
```

When the volume label or msus is omitted, the computer automatically accesses the default device established by the configuration of the system or specified by a `MASS STORAGE IS` statement. The volume label or msus must be included if the file is located elsewhere than on the default mass storage device.

Examples:

`"QUAKE.DRIVE0"`	The file named `QUAKE` is on the medium having volume label `".DRIVE0"`.
`"QUAKE:D700"`	The file named `QUAKE` is on the device having msus `":D700"`.

The following example establishes a default mass storage device and then accesses a file located there.

`MASS STORAGE IS ":D701"`	Establishes a mass storage default device.
`CREATE "PRESSURES", 5`	Creates a five-record data file named `PRESSURES` on the disc at msus `":D701"`.

The only characters that cannot be used in the file name portion of a file specifier are period (.), colon (:), and quotes ("). The period is reserved as the volume label prefix, the colon is the msus prefix, and the quotes are used to delimit strings. File names longer than 10 characters are truncated to 10 characters.

The File Directory

Each flexible disc maintains a *catalog*, or *directory*, of the files stored on it. The `CAT` statement outputs the contents of the file directory to the computer display.

The syntax of the `CAT` statement is:

```
CAT [" :msus"
    " .volume label"]
```

If you have previously initialized a disc as `".DRIVE0"`, you can now obtain a file directory of that disc by executing `CAT ".DRIVE0"`.

```
[ Volume ]: DRIVE0
Name          Type    Bytes    Recs
```

Once you have stored programs and created data files on a mass storage medium, the file directory will look similar to the one shown below:

```
[ Volume ]: DRIVE0
Name          Type    Bytes    Recs
TEMPDATA      DATA      48        5
LEASTSQ       PROG     256        2
              NULL     256        1
EARNINGS      DATA     500        2
```

The file directory contains the following information:

Name	This is the name assigned to the file as part of the file specifier.
Type	There are five types of files: DATA, program (PROG), binary program (BPGM), NULL, and extended (**** or mnemonic). Graphics (GRAF) files are a type of extended file.
Bytes	The number listed is the number of bytes per file record.
Recs	This is the number of records in the file.

File records and bytes are discussed in section 22.

You can terminate a catalog listing at any time by pressing any key.

File Types

As mentioned in the discussion of file directories, five types of files may be used with a mass storage system. Each file type is created and retrieved by different procedures, summarized in the following table.

File Type	Description
PROG	Contains programs. Information is stored using the STORE command and retrieved using the LOAD command. Program files are covered in section 21.
DATA	Contains numeric and string data. File is created by the CREATE statement. Data is stored by the PRINT# statement and retrieved by the READ# statement. Data files are covered in section 22.
BPGM	Contains binary programs. Information is stored using the STOREBIN statement and retrieved by the LOADBIN statement. Binary program files are covered in section 21.
NULL	Empty file. Null files are created when individual files are purged. Packing the disc removes null files from the disc directory. Null files are discussed in section 23.
**** or mnemonic	Extended files. Refer to the discussion of extended files on page 306. GRAF files are extended files containing graphics displays. Information is stored using the GSTORE statement and retrieved using the GLOAD statement. GRAF files are discussed in section 21.

Specifying Parameters Using Expressions

Both numeric expressions and string expressions can be used as mass storage parameters. In most cases, a non-integer value supplied by a numeric expression is rounded to the nearest integer for parameters requiring an integer value. An exception is the mass storage unit specifier—attempts to use an expression evaluating to a non-integer as part of an msus generate `Error 126 : MSUS`.

Example:

```
⋮
50 D$="DRIVE" @ N=0
60 VOLUME ":D700" IS D$&VAL$ (N) ! Msus ":D700" is assigned volume label
                                   ".DRIVEO".
70 !
80 FILENAME$="A/R"
90 CREATE FILENAME$&".DRIVEO",3 !  File named "A/R" created on disc with
                                   volume label ".DRIVEO".
⋮
```

Section 21

Storing and Retrieving Programs and Graphics

Information in this section covers how to store and retrieve programs and CRT graphics displays using your mass storage system. Use of chaining to expand the capability of the computer in running large programs is also covered.

Storing a Program

The STORE command is used to store the program currently in computer memory into a program file on a disc. Programs are stored encoded in the computer's internal language. The STORE command attaches a specified name to the program, creates a program file with that name, and then stores the program in the program file. The stored program remains in computer memory until scratched or until another program is loaded.

The keyword STORE can be displayed using the typing aid provided on (k7).

```
STORE "file specifier"
```

Examples:

```
STORE "QUAKE.DRIVE0"
```

Names the program in computer memory QUAKE, and stores the program in a program file located on the disc having volume label ".DRIVE0".

Remember that you can use either a volume label or msus in a file specifier.

```
STORE "QUAKE:D700"
```

Has the same effect as the previous example if ".DRIVE0" is ":D700".

You may omit the volume label or msus portion of the file specifier if the program is to be stored onto the disc in the default drive or onto the disc declared the default mass storage medium by a MASS STORAGE IS statement.

Examples:

```
MASS STORAGE IS ".DRIVE0"
STORE "QUAKE"
```

Assigns the drive containing the disc having volume label ".DRIVE0" as the default mass storage device and stores the program QUAKE onto that disc.

```
INITIALIZE "DRIVE0"
```

Next, type in a program which you will store. The following program will be used later in this section to demonstrate program chaining. The program uses data generated by another program, and therefore generates an error if run by itself.

```
10 REM ****************DRAWSTAR****************************************
20 GCLEAR
30 OPTION BASE 1
40 COM REAL X(16),Y(16) ! COM is explained later in this section.
50 PLOTTER IS 1 !         Defines CRT as plotting device.
60 SHOW -2,2,-2,2 !       Scales the graphics CRT.
70 MOVE X(16),Y(16)
80 FOR I=1 TO 16
90    DRAW X(I),Y(I) !    Draws the star.
100 NEXT I
110 END
```

To store the program, execute:

```
STORE "DRAWSTAR.DRIVE0"
```

The drive light on DRIVE 0 will be on during the storing process. When the light goes off, the program DRAWSTAR has been stored on disc ".DRIVE0". To see the updated file directory, execute CAT from the keyboard.

```
[ Volume 3: DRIVE0
Name        Type  Bytes  Recs
DRAWSTAR    PROG   256      2
```

The directory shows that DRAWSTAR has been stored in a program file two, 256-byte records in length.

STORE can be used to store a program in computer memory over a program that was stored previously. For instance, after storing DRAWSTAR, you can edit the program in computer memory, and then re-execute:

```
STORE "DRAWSTAR.DRIVE0"
```

Because of this overwrite capability, you must be careful in storing a new program not to accidentally assign to it the name of another program file, thereby overwriting a previously stored program you still need. File security, discussed in section 23, allows you to protect files from being overwritten.

> Note: The STORE command will not overwrite file types other than PROG. For example, attempting to store a program using the name of a currently existing data file generates Error 68 : FILE TYPE.

Loading a Program

Once a program has been stored on a mass storage medium, a copy can be retrieved into computer memory with the LOAD command.

```
LOAD "file specifier"
```

The file specifier must correspond to the name of a program file. Attempting to LOAD a nonexistent program results in Error 67 : FILE NAME. If the msus or volume label is omitted from the file specifier, the computer accesses the default disc.

When LOAD is executed, any program or data currently in computer memory is scratched before the new program is loaded. Variables that were assigned in calculator mode are also scratched.

If you stored the program DRAWSTAR, you can now retrieve it. But first, you may want to scratch the contents of program memory just to prove to yourself that LOAD really works. Execute SCRATCH and then (LIST) to confirm that the program is no longer in computer memory and then execute:

```
LOAD "DRAWSTAR.DRIVE0"
```

The drive light on DRIVE 0 will be on while the program is being loaded. When the light goes off, list the program to confirm that it is in computer memory.

Autostart Programs

The *autostart* feature enables the computer to load and run a program automatically at power-on.

When the computer is turned on, it searches the directory of the disc located in the default drive for a program file named Autost. If the file is found, the program is automatically loaded into the computer and executed.

Loading HP-83/85 Programs

Programs written for and stored using the HP-83/85 computers require translation before they can be run on the HP-86/87. When an HP-83/85 BASIC program is loaded into the HP-86/87, the computer automatically translates the program into a form executable by the HP-86/87. Refer to appendix B of the introductory manual for information on translating HP-83/85 programs.

Chaining Programs

The CHAIN statement allows you to load a stored program into computer memory from a running program. When CHAIN is executed in a program:

- The current BASIC program and any data in computer memory are scratched. Specified data can be preserved between two programs by including a COM statement in both programs. Binary programs are not scratched when CHAIN is executed.
- The program specified in the CHAIN statement is immediately loaded into computer memory from mass storage.
- The newly loaded program is executed automatically.

Note that, unlike the LOAD command, the CHAIN statement is programmable.

```
CHAIN "file specifier"
```

The COM Statement

The COM statement is used to dimension arrays and to preserve variable definitions between programs. All variables not included in the COM statement are scratched when the chained program is loaded.

```
COM [precision] item [ , item...] [ , [precision] item [ , item...]...]
```

Precisions for numeric items can be: REAL, SHORT, or INTEGER.

Each item may be:

- A simple numeric variable.
- A simple string variable, with the dimensioned number of characters enclosed within brackets, [].
- A numeric array (subscript(s)).
- A string array (subscript(s))[].

Simple string variables must be explicitly dimensioned in the COM statement, regardless of their length.

Example:

```
25 COM A,B(4,3),C$[5],G$(5,7)[50],INTEGER E,F,SHORT H(10),I
```

The simple variable A and the numeric array B(4,3) are full precision. Full precision is assumed at the beginning of the COM list and for numeric variables declared after a type REAL declaration. From left to right in a given COM list, all variables following a precision declaration keyword have that precision until another precision declaration keyword appears in the list. In the above example, both E and F are INTEGER precision; the array H(10) and I are both SHORT precision.

Each program may contain any number of COM statements. Variable names need not match; the values will be preserved and assigned to COM item variables based on their order in the program. The total number of COM items in a chained program cannot exceed the number of COM items in the program containing the CHAIN statement.

Example: The following three short programs preserve four variables. Note that the variables are assigned new names as they are passed between programs.

```
10 REM ***************PROGRAM #1*********************************************
20 COM A,B$[1],C,D
30 A=1 @ B$="x" @ C=3 @ D=4
40 PRINT "PROGRAM #1";A;B$;C;D
50 CHAIN "PROGRAM#2"
60 END
```

```
10 REM ***************PROGRAM #2*********************************************
20 COM T,Y$[1]
30 COM C,D
40 PRINT "PROGRAM #2";T;Y$;C;D
50 CHAIN "PROGRAM#3"
60 END
```

```
10 REM ***************PROGRAM #3*********************************************
20 COM Q,R$[1]
30 COM W
40 PRINT "PROGRAM #3";Q;R$;W
50 END
```

When PROGRAM #1 is run, the following output is printed:

```
PROGRAM #1 1 x 3  4
PROGRAM #2 1 x 3  4
PROGRAM #3 1 x 3
```

Variables held in common between programs must agree in type (numeric or string, simple or array) and precision (REAL, INTEGER, or SHORT). Common arrays must agree in lower (OPTION BASE) and upper bounds. Simple and array string variables must agree in dimensioned maximum number of characters.

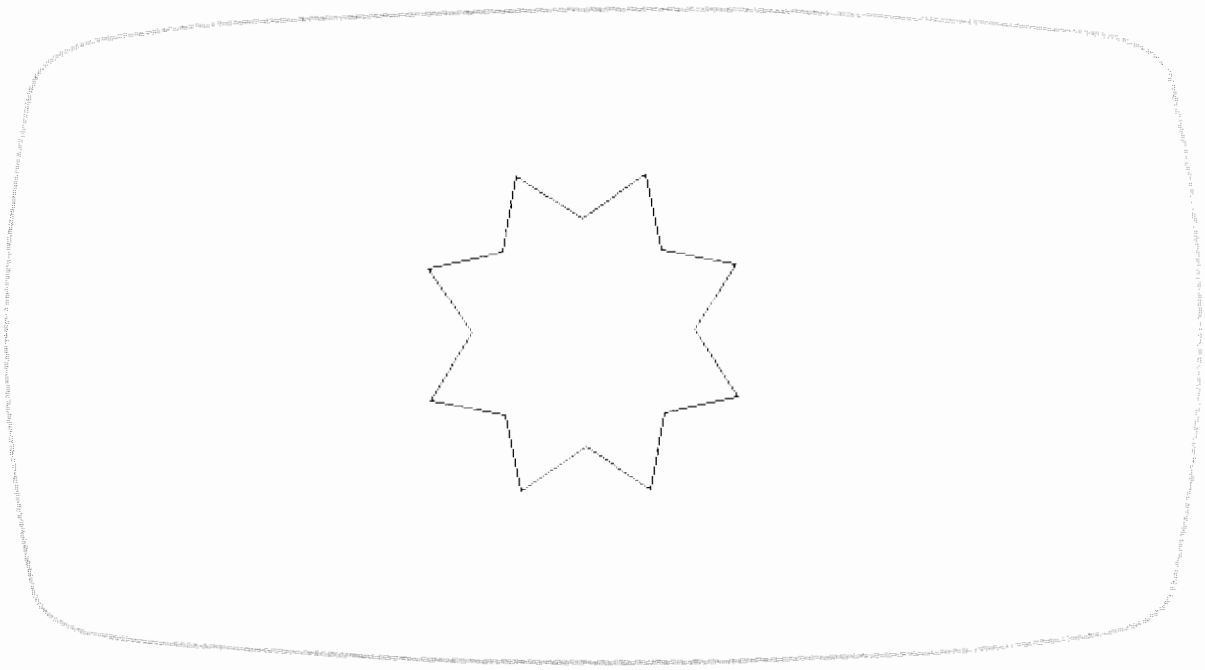
COM statements must be placed following the OPTION BASE statement and before any other reference to the variable. You cannot include a COM statement within a function definition, and you cannot include the same variable in more than one COM statement.

Example: The following program generates data points for drawing an eight-pointed star. Statement 140 chains the program DRAWSTAR on page 276. Note the agreement between STAR and DRAWSTAR of their COM statements, their OPTION BASEs, and the upper bounds of the arrays.

```
10 REM ************************STAR*****************************************
15 REM ***This program generates the points for the DRAWSTAR program ***
20 OPTION BASE 1
30 COM REAL X(16),Y(16) !                    Dimensions arrays X and Y and pre-
                                             serves them for chained DRAWSTAR.
40 INDEX=0
50 RAD
60 FOR ANGLE=0 TO 1.75 STEP .25
70    INDEX=INDEX+1
80    X(INDEX)=COS (ANGLE*PI )
90    Y(INDEX)=SIN (ANGLE*PI )
100   X(INDEX+1)=1.5*COS ((ANGLE+1/8)*PI ) ! Coordinates of the points of the
110   Y(INDEX+1)=1.5*SIN ((ANGLE+1/8)*PI ) ! star.
120   INDEX=INDEX+1 !                       INDEX is incremented by 2 each time
                                            through the loop.
130 NEXT ANGLE
140 CHAIN "DRAWSTAR" !                      Loads and executes DRAWSTAR
150 END
```

If you'd like to run the set of programs more than once, be certain to store STAR now since it will be scratched when statement 140 is executed.

Now, execute STAR. When statement 140 is executed, the program accesses ":DRIVE0" to retrieve a copy of DRAWSTAR. The drive light on DRIVE 0 will come on as DRAWSTAR is loaded into computer memory, and the computer automatically switches to *graph* mode as it draws the star.



Storing and Retrieving Graphics Displays

The computer allows you to store the contents of the computer's graphics display onto a disc and to retrieve the display without re-executing the display-generating program. The operation of loading a stored display into the computer's graphics display leaves variable assignments and the program currently in computer memory intact.

The statements used to store and retrieve graphics displays access graphics (GRAF) files.

Storing a Graphics Display

The contents of the computer's graphics display is stored onto a disc by executing the GSTORE statement. GSTORE can be executed both in a program or from the keyboard.

```
GSTORE "file specifier"
```

You can store displays generated in either *graph* or *graph-all* modes. However, to store a display generated in *graph-all* mode, you must include the GSTORE statement in your display generating program, since toggling to *alpha* mode to execute GSTORE from the keyboard would scratch the graphics display.

Example: To store the previously generated star, execute:

```
GSTORE "STARGRAPH.DRIVE0"
```

As in program storing, the contents of a GRAF file can be altered by executing GSTORE with the same file specifier and a different graphics display.

Retrieving a Graphics Display

The GLOAD statement retrieves a previously GSTOREd graphics display.

```
GLOAD "file specifier"
```

Execution of GLOAD overwrites the current graphics display as the contents of the GRAF file is copied into the graphics display. As GLOAD is executed, the computer automatically switches to *graph* or *graph-all* mode, and you can see the stored display appearing on the CRT.

Whether or not alpha CRT memory is affected by a GLOAD operation depends on the mode of the stored graphics display. If the graphics display being GLOADed was originally generated and GSTOREd in *graph-all* mode, then alpha CRT memory is scratched when the display is GLOADed. If, on the other hand, the display was generated and stored in *graph* mode, then the contents of the alpha display are left intact.

Attempting to GLOAD a graphics display from *alpha-all* mode generates Error 24 : MODE.

Example:

```
GLOAD "STARGRAPH.DRIVE0"
```

Retrieves the contents of the GRAF file STARGRAPH into the graphics display.

Storing and Retrieving Binary Programs

Some of the programs on the demonstration disc and in the applications software are *binary programs*. They function like a ROM, except that they are loaded from mass storage. Binary programs cannot be listed or edited.

The LOADBIN statement retrieves a binary program into computer memory.

```
LOADBIN "file specifier"
```

LOADBIN loads a binary program without altering the existing BASIC program, binary programs, or data in computer memory. A maximum of five binary programs can be present in computer memory at one time; attempting to LOADBIN a sixth program generates an error.

In addition to its name, a binary program has a binary program number. Each binary program LOADBINned into computer memory must have a unique binary program number. Attempting to LOADBIN a binary program whose binary progam number matches a binary program already present generates an error.

Note: Attempting to LOADBIN an absolute binary program whose address is already used by computer read-only memory or another binary program generates an error.

Binary programs are scratched when a LOAD command is executed. If binary programs are to be added to accompany a BASIC program, you must LOAD the BASIC program before retrieving the binary programs.

In order to edit a program that uses binary programs, the binary programs must be present in computer memory.

A binary program is stored using the STOREBIN statement:

```
STOREBIN "file specifier"
```

Binary programs are identified in computer memory by the name used to LOADBIN the binary program. The file name used with STOREBIN must be identical to the name used with LOADBIN.

Notes

Section 22

Storing and Retrieving Data

Introduction

The discussion of file types in section 20 pointed out that mass storage enables you to create and use five different types of files, one of which is the data file. This section covers the five operations necessary to store and retrieve data:

- Creating data files.
- Opening a previously created data file.
- Storing data (printing data to the file).
- Retrieving data (reading data from the file).
- Closing the data file.

There are two methods for accessing data files: serial access and random access. Serial access stores and retrieves data sequentially, and is useful when the complete data list is to be stored and retrieved as a unit. Random access allows you to access portions of the data. Since data is accessed somewhat differently with serial and random access, serial storing and retrieving is discussed separately from random storing and retrieving.

Files created in mass storage consist of one or more records. The size of the records can be varied to accommodate the storage requirements of the data. Before covering how to create data files of different sizes, we will first discuss file structure and storage requirements.

File Records

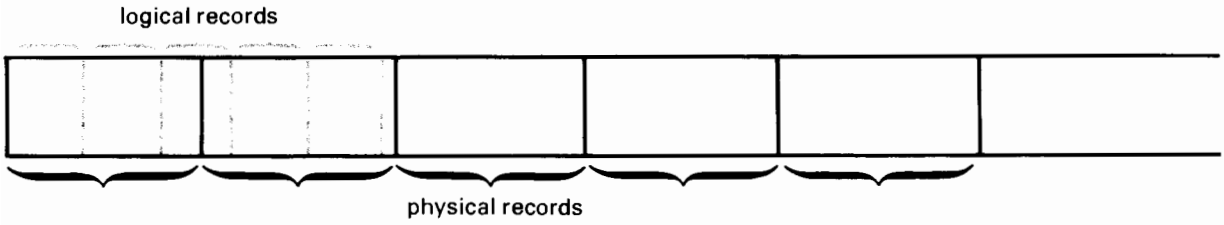
When a data file is created in mass storage, the size of the file is set by specifying the number of records in the file and the length of the records. A record is the smallest addressable location on a mass storage medium such as a disc or tape. Record length is specified in bytes, and all records in a particular file are the same length.

Two types of records are available: physical and logical. The two types of records make it possible to match the structure of data to the file in which it is stored, thus using storage space most efficiently.

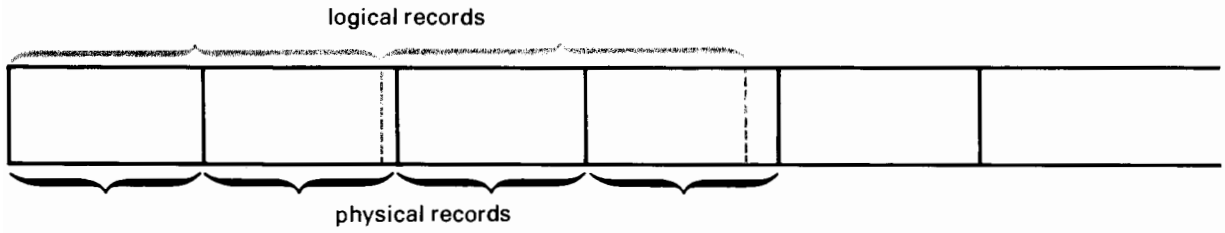
Physical Records—Physical records are always 256 bytes in length and are set up automatically when program, graphics, or data files are created. All files begin at a new physical record. The 256-byte physical record is the smallest addressable storage unit unless a different size addressable unit, called a logical record, is established.

Logical Records—Logical records are specified for a file when an addressable unit of length other than 256 bytes is desired. The file will still begin at the start of a physical record; within the file, however, the divisions between physical records are ignored and a logical record may straddle two or more physical records. When a data file is created without specifying logical records, the automatically-created physical records become logical records.

The following diagrams illustrate two files consisting of logical records. The first file contains five records, each 100 bytes long. Note that the file utilizes two physical records and that there are 12 bytes of unusable space, since any new file must begin at a new physical record. The divider between the two physical records is ignored.



The next diagram illustrates a file consisting of two 500-byte logical records. The divisions between physical records within the logical records are ignored; however, 24 bytes are not usable, since any new file must start at a new physical record.



Storage Requirements

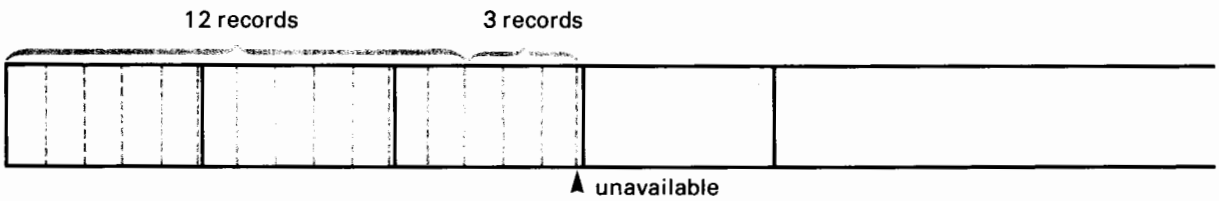
File and record sizes should be specified with the space requirements of the data in mind. The following chart describes the amount of space necessary to store numeric and string data.

Type Variable	Space Requirements
Simple numeric	8 bytes.
Simple string	3 bytes + 1 byte per character + 3 bytes each time the string crosses into a new logical record.
Numeric array	Per array element: 8 bytes.
String array	Per array element: 3 bytes + 1 byte per character + 3 bytes each time a string crosses into a new logical record.

You can use these space requirements to set up files to match your data. For instance, suppose you would like to create a file for storing the last and first names, social security number, and salary of a dozen employees. You would like each employee's information in a separate record.

Item	Type of data	Bytes
last name	12-character string	3 + 12 = 15
first name	10-character string	3 + 10 = 13
social security	11-character string	3 + 11 = 14
salary	numeric	8
		50

A file can then be created consisting of 12, 50-byte records. When logical records are created, any otherwise wasted space (in this case, 168 bytes) is also allocated into logical records if possible. The 168 bytes form an additional three records added to the file automatically, with 18 unusable bytes.



Creating Data Files

The CREATE statement allocates space on a mass storage medium for the data file.

```
CREATE "file specifier", number of records [, record length]
```

The number of records specifies how many logical records the file will contain, and must be an integer from 1 through 32,767. The record length is the number of bytes in each record, and must be an integer from 4 through 32,767. The default value for the record length is 256 bytes, the size of a physical record. The total number of bytes, obtained by multiplying the number of records by the record length, must not exceed the storage capacity of the mass storage medium.

Example: The following statement creates a data file named EMPLOYEES for storing the identification and salary information for the 12 employees, as discussed above.

```
30 CREATE "EMPLOYEES.DRIVE0",12,50
```

Creates a data file with 12 logical records of 50 bytes each. (Actually, 15 records will be set up, as discussed in Storage Requirements, page 286.)

Since the information for each employee is stored in its own record, it can be accessed and updated separately from the data for other employees. If you create this file on ".DRIVE0" and then execute CAT, the file will be listed.

```
[ Volume 3: DRIVE0
Name          Type  Bytes  Recs
:
EMPLOYEES     DATA     50    15
```

If it were preferable to always store and retrieve the information for all employees at once, a file containing one record could be set up.

```
30 CREATE "EMPLOYEES.DRIVE0", 1, 600
```

Creates a file of one 600-byte record.

Opening a Data File

Once a data file has been created, it must be opened before it can be accessed to store data. Opening a data file assigns to it a buffer through which data flows from the computer to the disc and from the disc to the computer (see figure 22.1, page 288). The ASSIGN# statement is used to open a data file:

```
ASSIGN# buffer number TO "file specifier"
```

The file name must be the name of a previously created data file. The buffer number is a number that rounds to an integer from 1 to 10. Once a buffer has been assigned to a file, that buffer remains assigned to the file until the same buffer number is assigned to a different file, or until the file is closed.

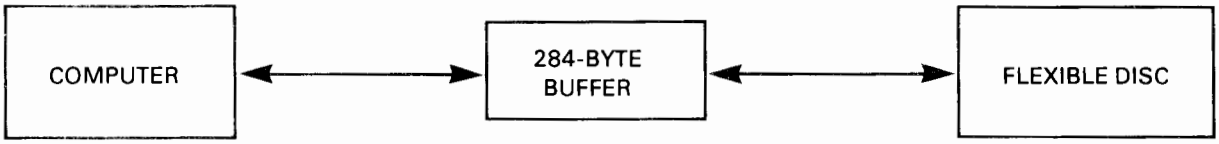


Figure 22.1 Flow of Data Through a Buffer

A mass storage buffer is a 284-byte location in computer memory that is allocated whenever a file is opened. The purpose of the buffer is to decrease access time and to reduce wear of the mass storage medium by accumulating data being transferred between the computer and a mass storage medium.

Data acccumulated in a mass storage buffer is transferred to the disc whenever one of the following conditions is met:

- The buffer is full. A buffer can hold 256 bytes of data.
- The buffer is reassigned to a different file.
- PAUSE, STOP, or END is executed.
- Program execution is interrupted.
- The file is closed.
- Another logical record is accessed using a random access READ# or PRINT# statement.
- A PRINT# statement is executed from the keyboard.

Example:

```
50 ASSIGN# 1 TO "EMPLOYEES.DRIVE0"
```

Opens EMPLOYEES file and assigns to it buffer #1.

Closing a Data File

When you've completed a data transfer to or from a file, you should close the file. The ASSIGN# statement accomplishes this:

```
ASSIGN# buffer number TO *
```

The buffer number must agree with the buffer number assigned to the file when it was opened.

Example: To close EMPLOYEES previously opened in statement 50, above, execute:

```
200 ASSIGN# 1 TO *
```

When a buffer is closed, any data in it is transferred to the disc. If a program error causes a halt while data is in the buffer en route to mass storage, all the data in the buffer will be printed to the file. The file remains open and thus does not need to be reopened before program execution is continued.

If a disc error causes a halt during program execution, data in a buffer en route to mass storage is lost unless the file is closed from the keyboard. When the file is closed, the data will be transferred to mass storage.

Serial Access

Serial access is used when a quantity of data is to be stored and retrieved sequentially, and updated as one unit. The entire file itself becomes the smallest addressable unit of storage. This is true even if the file being accessed consists of more than one logical record; in serial access, data is stored and retrieved without regard to record divisions within the file.

Serial Printing

Data is stored into a file serially using the serial PRINT# statement, which has the form:

PRINT# *buffer number* ; *print# list*

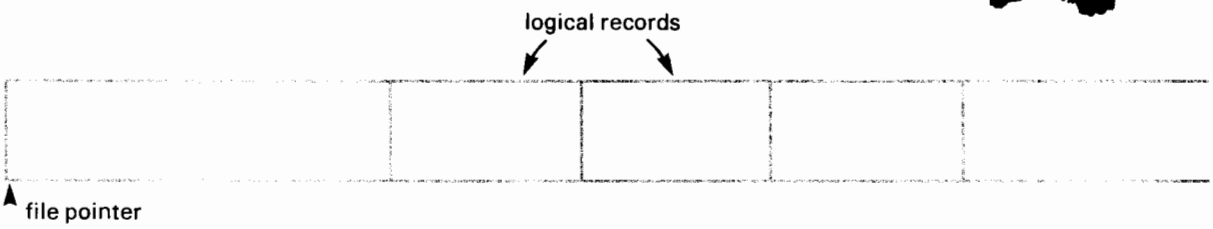
The buffer number must have been previously assigned to a data file. The print# list itemizes the data you wish to store, and may include numbers, simple numeric and string variables, and numeric and string array names. Items in the print# list are separated by commas.

The computer uses a pointer to locate and access data items. When a file is opened, the file pointer is placed at the beginning of the file, and data serially printed to the file are stored starting at the beginning of the file. The pointer moves through the file sequentially as the print# list is stored. When the entire print# list has been recorded, the pointer remains at the end of the recorded data, and an end-of-file marker indicates the position of the last recorded data. Execution of a subsequent PRINT# statement with the same buffer records the new print# list at the end of recorded data and moves the end-of-file marker to the end of the newly recorded data. The pointer will continue to move sequentially through the file until the pointer is moved to the beginning of a specified logical record using a random PRINT#/READ# statement, or until the file is closed or reassigned with an ASSIGN# statement.

Examples: The following illustrations demonstrate movement of the file pointer during serial printing.

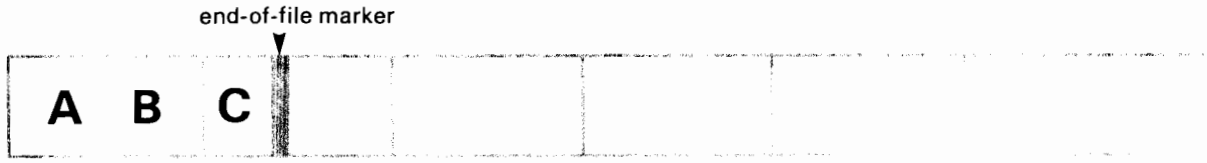
Opening the file:

```
ASSIGN# 1 TO "FILE.DRIVE0"
```



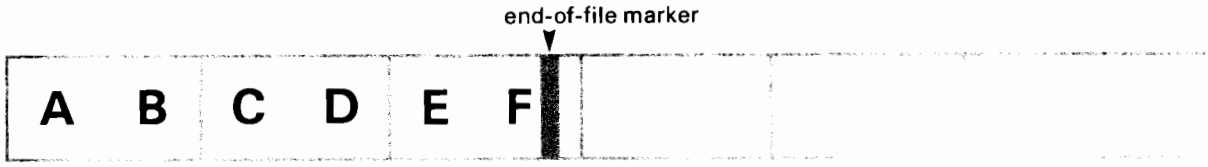
Printing three items to the file:

```
PRINT# 1; A,B,C
```



Printing three additional items to the file:

```
PRINT# 1; D,E,F
```



The movement of the file pointer and end-of-file marker influence the way in which serial files are updated. If, after entering a long list of data items serially, the pointer is returned to the beginning of the file using a random READ#/PRINT# statement or an ASSIGN# statement, a new serial PRINT# statement will record new data items over the old ones. However, an end-of-file marker is placed at the end of the new data items. The result is that the entire old data list is lost.

Example: The following program uses serial access to store check register data for the PDQ Music Company. The company opens a new file each day, and records the company to which a check has been written as string COMPANY$ and the amount of the check as numeric variable AMOUNT.

```
10 CREATE "NOV5CHECKS:D701",4 !       Creates file of 4, 256-byte records.
20 ASSIGN# 1 TO  "NOV5CHECKS:D701" ! Opens the file.
30 DIM COMPANY$[40]
40 NEWENTRY: DISP "COMPANY NAME, AMOUNT OF CHECK";
50 INPUT COMPANY$,AMOUNT
60 IF COMPANY$="NO MORE TODAY" THEN DONE
70 PRINT# 1 ; COMPANY$,AMOUNT !       Prints company name and amount of check to
                                      the file serially.
80 GOTO NEWENTRY
90 DONE: PRINT# 1 ; COMPANY$
100 ASSIGN# 1 TO  * !                  Closes file.
110 END
```

When the program is run, it prompts for company name and amount of the check until NO MORE TODAY is input as the company name. If file capacity is exceeded before program execution ends, the computer returns an error announcing an attempt to print at the end of the file.

```
COMPANY NAME, AMOUNT OF CHECK?
Teddy's Security, 68.85
COMPANY NAME, AMOUNT OF CHECK?
Ant Bee's Pest Control, 98.00
COMPANY NAME, AMOUNT OF CHECK?
Bert Tenkey - CPA, 45.76
COMPANY NAME, AMOUNT OF CHECK?
Dusty's Janitorial Service, 155.25
COMPANY NAME, AMOUNT OF CHECK?
Count von Tou's Inventory Service, 97.55
COMPANY NAME, AMOUNT OF CHECK?
NO MORE TODAY, 0
```

Note: When a string serially printed to a file crosses from one record to another, an additional three bytes are needed for the string *header*, which identifies the portion of the string contained in the new record.

Reading Files Serially

Data that has been stored onto a mass storage medium must be retrieved, or read, back into computer memory before it can be used. Reading data from a file transfers a copy of the data through a buffer into computer memory.

When data is retrieved serially, the entire file contents is accessed sequentially, ignoring any record divisions. Data stored both serially and randomly can be retrieved serially. Serial reading is accomplished by the READ# statement:

READ# *buffer number* ; *read# list*

The buffer number must match the number previously assigned to the file with an ASSIGN# statement. The read list need not exactly match the print# list used to store the data. However, data items being read must agree in type (string versus numeric) with the contents of the file.

Example: Data printed to file by the statement:

```
PRINT# 1; A,B,C$
```

can be retrieved by the statement:

```
READ# 1; Q,R,S$
```

Numeric data need not agree in precision (REAL, INTEGER, SHORT). The number will be converted to the precision of the read# variable. If the read# variable has lower precision than the print# variable, the number is rounded.

During serial reading, the pointer moves through the file sequentially, much like with serial printing. At the conclusion of the read# list, the pointer remains positioned after the last item read. An attempt to read data when the pointer has encountered the end-of-file marker generates an error.

Example: If you used the program on page 290 to create a data file for a check register, you can use the following program to read the file, print its contents, and sum the day's check payments.

```
10 ASSIGN# 1 TO  "NOV5CHECKS:D701" !          Opens data file.
15 DIM COMPANY$[40]
20 SUM=0 !                                    Initializes SUM of day's checks.

30 NEWCOMPANY: READ# 1 ; COMPANY$ !           Retrieves company name.
40   IF COMPANY$="NO MORE TODAY" THEN DONE
50   READ# 1 ; AMOUNT !                       Retrieves amount of check.
60   PRINT USING 100 ; COMPANY$,AMOUNT
70   SUM=SUM+AMOUNT
80 GOTO NEWCOMPANY  !                         Branch to retrieve another
                                              company name.
90 DONE: PRINT USING 110 ; SUM
100 IMAGE 40A,2X,5D.DD
110 IMAGE /,3X, "TOTAL =", 32X,5D.DD
120 ASSIGN# 1 TO  *
130 END
```

```
Teddy's Security                          68.85
Ant Bee's Pest Control                    98.00
Bert Tenkey - CPA                         45.76
Dusty's Janitorial Service               155.25
Count von Tou's Inventory Service         97.55

   TOTAL =                               465.41
```

In the above program, the file pointer moves through the data file as both READ# statements are executed repeatedly. If statement 40 were omitted, the READ# statement in line 50 would eventually encounter the end-of-file marker, generating an error.

Data read from mass storage initially is stored in a temporary memory location before being transferred to memory allocated to the variable. If you receive an unexpected memory overflow error while attempting to read# a long string from a data file, you will need to break the string into substrings and print# the substrings into logical records using random access. The substrings can then be read back into the computer one at a time.

Random Access

Random access enables you to print to, read from, or update a portion of a data file by accessing individual logical records. Since size is specified in the CREATE statement and can be as small as four bytes, random access allows you to update small portions of data without affecting the rest of the file.

Random Printing

The random PRINT# statement has the syntax:

```
PRINT# buffer number , record number [ ; print# list]
```

The buffer number must match the buffer assigned to the file by an ASSIGN# statement. The record number must be less than or equal to the total number of logical records in the file. The print# list contains all the items to be printed to the record, separated by commas.

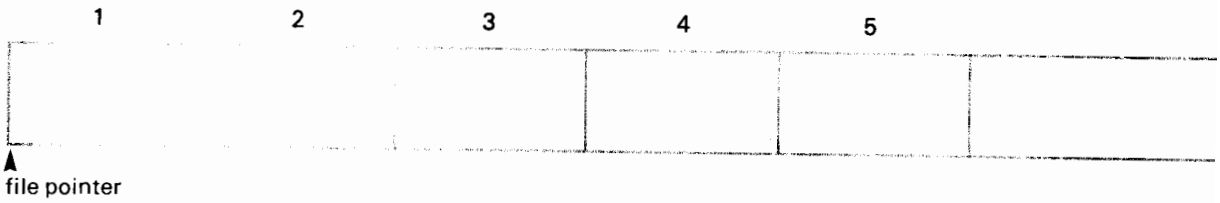
The random PRINT# statement operates somewhat differently from the serial PRINT# statement:

- Because random printing accesses a particular record, the record number must be specified in the statement.
- When a random PRINT# statement is executed, the file pointer moves to the beginning of the specified record. The print# list is printed to the record and an end-of-record marker is placed after the last print# item.
- In random printing, the contents of the file buffer is transferred to its destination each time another record is accessed.
- Record divisions are not ignored in random access operations. The print# list must not exceed the storage capacity of the logical record. Error 69 : RANDOM OVF or ERROR 72 : RECORD indicates that you are attempting to print too much data to the record.
- The file pointer is moved to the beginning of a random record by executing a random PRINT# statement without a print# list.

Example: The following illustrations demonstrate movement of the file pointer during random printing.

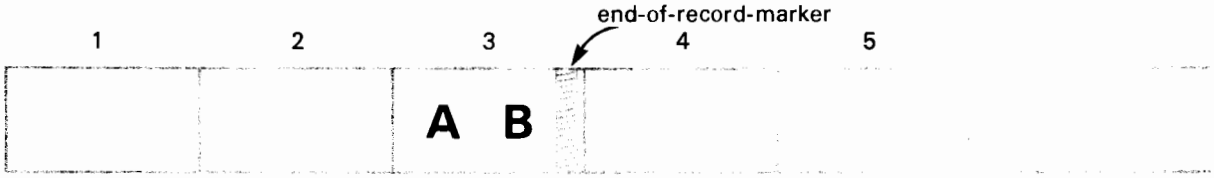
Opening the file:

```
ASSIGN# 1 TO "FILE.DRIVE0"
```



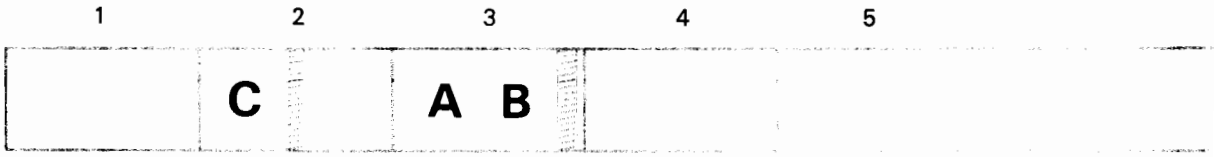
Printing data to record #3:

```
PRINT# 1,3; A,B
```



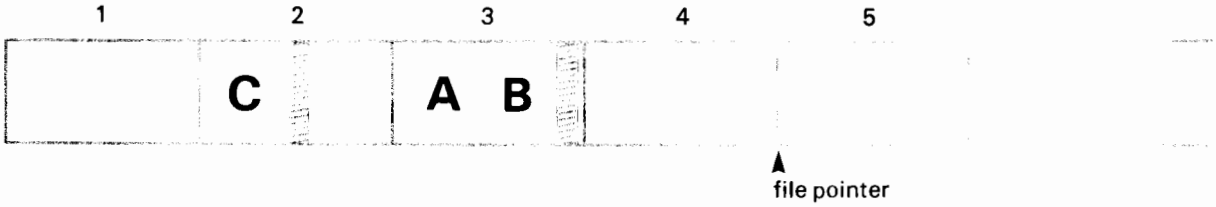
Printing data to record #2:

```
PRINT# 1,2; C
```



Moving file pointer to the beginning of record #5:

```
PRINT# 1,5
```



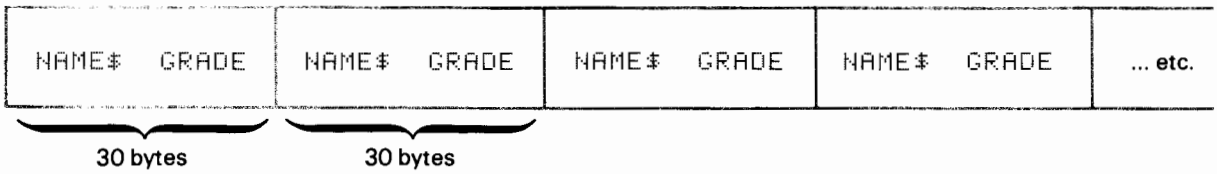
Example: The following program creates and accesses a 10-record data file for storing exam grades. Each of the ten records can contain the name of a student and the student's final exam score. The string XXXXX and numeric value 0 are entered into otherwise empty records.

```
10 CREATE "FINALEXAM:D701",10,30 !        Creates data file.
20 ASSIGN# 1 TO  "FINALEXAM:D701" !       Opens data file.
30 DISP "ENTER NUMBER OF STUDENTS";
40 INPUT N
50 FOR I=1 TO N
60    DISP "ENTER STUDENT NAME AND GRADE";
70    INPUT NAME$,GRADE
80    PRINT# 1,I ; NAME$,GRADE !          Print data to record I.
90 NEXT I
100 FOR J=N+1 TO 10 !                     Loop fills otherwise empty records.
110   PRINT# 1,J ; "XXXXX",0 !            Print to record J.
120 NEXT J
130 ASSIGN# 1 TO  * !                     Close data file.
140 END
```

The program creates the file and then requests data:

```
ENTER NUMBER OF STUDENTS?
6
ENTER STUDENT NAME AND GRADE?
BILL FOLD,78
ENTER STUDENT NAME AND GRADE?
GREG GARIOUS,81
ENTER STUDENT NAME AND GRADE?
ED IFY,94
ENTER STUDENT NAME AND GRADE?
DEB ONAR,99
ENTER STUDENT NAME AND GRADE?
ANNA PURNA,67
ENTER STUDENT NAME AND GRADE?
CLARA KNET,90
```

Data is entered into the file as shown below.



Reading Files Randomly

Random access reading is accomplished with the random read statement:

READ# *buffer number* , *record number* [; *read# list*]

The differences between the random read statement and serial read statement are analogous to the differences between the two types of PRINT# statements:

- The statement must include the record number you wish to access.
- The file pointer automatically moves to the beginning of the specified logical record.
- Logical record divisions are not ignored. An attempt to read past the end of a logical record generates Error 72 : RECORD.
- The file pointer can be moved to the beginning of the record by executing the statement without a read# list.

As with serial reading, the read list must agree in data type (numeric versus string) with the stored data; however, number precision need not agree. (Refer to page 291, Reading Files Serially, for further information.)

Example: The following program allows you to correct any previous entries to the FINALEXAM file and to add additional entries to records containing XXXXX,0. The program requests the record number of the data you wish to alter, displays the current contents of the record, and provides for replacing that data with the corrected information.

```
10 ASSIGN# 1 TO  "FINALEXAM:D701" !      Open data file.
20 DISP "RECORD TO BE CHANGED"
30 INPUT RECORD
40 IF RECORD=0 THEN DONE
50 READ# 1,RECORD ; NAME$,GRADE !        Read# contents of record RECORD.
60 DISP "CONTENTS OF RECORD";RECORD;" ";NAME$;GRADE
70 DISP "ENTER CORRECT INFORMATION"
80 INPUT NAME$,GRADE
90 PRINT# 1,RECORD ; NAME$,GRADE !       Print corrected data to RECORD.
100 GOTO 20
110 DONE: ASSIGN# 1 TO  * !              Close data file.
120 END
```

The following display output demonstrates changing the contents of records 7 and 5.

```
RECORD TO BE CHANGED
?
7
CONTENTS OF RECORD 7  XXXXX 0
ENTER CORRECT INFORMATION
?
DENNIS STURTS,55
RECORD TO BE CHANGED
?
5
CONTENTS OF RECORD 5  ANNA PURNA 67
ENTER CORRECT INFORMATION
?
SUSIE MARCOM,100
RECORD TO BE CHANGED
?
0
```

Storing and Retrieving Arrays

Entire arrays can be stored and retrieved using an array addressing format with the serial or random PRINT# and READ# statements. The proper array addressing formats for one-dimensional and two-dimensional numeric and string arrays are:

One-dimensional array—*array name* ( )

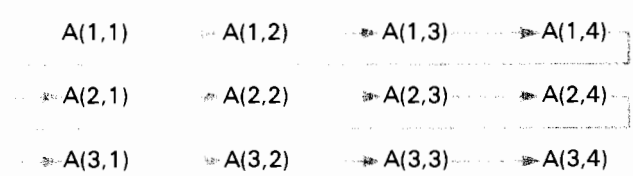
Two-dimensional array—*array name* ( ) or *array name* (,)

The comma is optional, for documentation purposes only.

Examples:

READ# 1; FREQ( )	Reads one-dimensional array FREQ serially.
PRINT# 2,4; AMP$(,)	Stores two-dimensional array AMP$ into record 4 of specified file.

In the case of two-dimensional arrays, the array elements are retrieved item by item without regard to dimensionality, with the second subscript varying more rapidly, that is, by rows.



Array elements of this 3 × 4 array are retrieved by rows.

Since array elements are stored on mass storage linearly, they may be retrieved with or without an array format, and any combination of upper limits can be used that accesses the desired number of elements. For instance, a 3 × 4 array stored in a file assigned buffer #1 might be retrieved by any of the following sets of statements (assuming OPTION BASE 1):

```
DIM B(3,4)        DIM B(4,3)        DIM B(6,2)        DIM B(12)
READ# 1; B(,)     READ# 1; B(,)     READ# 1; B(,)     READ# 1; B()
```

If the array specified in the READ# statement has fewer elements than the stored array, only those elements allowed by the READ# array will be retrieved.

Example: The following program creates a data file named POINTS and stores into it the array A(I,J) in which the integer part of the number equals the I value and the fractional part of the number equals the J value.

	J = 1	J = 2	J = 3	J = 4	J = 5
I = 1	1.1	1.2	1.3	1.4	1.5
I = 2	2.1	2.2	2.3	2.4	2.5
I = 3	3.1	3.2	3.3	3.4	3.5
I = 4	4.1	4.2	4.3	4.4	4.5

```
10 CREATE "POINTS:D701",20,8 !        Creates data file.
20 OPTION BASE 1
30 SHORT A(4,5) !                     Dimensions 4 by 5 array.
40 FOR I=1 TO 4
50    FOR J=1 TO 5
60       A(I,J)=I+J/10 !              Assigns values to array elements.
70       PRINT A(I,J);
80    NEXT J
90    PRINT
100 NEXT I
110 ASSIGN# 1 TO  "POINTS:D701" !      Opens data file.
120 PRINT# 1 ; A(,) !                  Prints array to data file.
130 ASSIGN# 1 TO  * !                  Closes data file.
140 END
```

The following program retrieves all the elements of array A(I,J) and prints the value of the elements for which I = J.

```
10 OPTION BASE 1
20 SHORT A(4,5)
30 ASSIGN# 1 TO  "POINTS:D701"
40 READ# 1 ; A()
50 FOR I=1 TO 4
60   PRINT A(I,I)
70 NEXT I
80 ASSIGN# 1 TO  *
90 END
```

```
1.1
2.2
3.3
4.4
```

An undefined simple numeric variable, simple string variable, or numeric array element is printed to a file as 0 or the null string (" "); a NULL DATA warning is generated during the PRINT# operation. Undefined string array elements are converted to null strings during the PRINT# operation; however, no warning is generated.

Notes

Section 23

Other File Operations

Your mass storage system enables you to perform a variety of file manipulations in addition to the ones already covered. This section covers the following additional file operations:

- Determining the data type of the next item in a data file.
- Copying files from one disc to another.
- Renaming files.
- Purging files.
- Packing a disc for more efficient use of mass storage space.
- Securing files.
- Verifying data.
- Accessing extended files.

Determining Data Types—The TYP Function

The TYP function allows you to determine the data type of the next item in a data file. The function also allows you to determine whether the file pointer is at the end of the record or at the end of the file.

```
TYP (buffer number)
```

The buffer number must correspond to the buffer assigned to the file being accessed. The function returns an integer from 1 to 10 according to the following table.

Type Value	Data Type
1	Number
2	Full String
3	End-of-File
4	End-of-Record
8	Start of String
9	Middle of String
10	End of String

When you are using the TYP function, the pointer can be moved through the file in much the same way as it is moved in serial and random printing and reading. One difference is that record divisions are not ignored when the pointer is moved serially.

Examples: We will use the TYP function to access data items in the file named AGES, which is organized into logical records as shown below:

1	2	3	4	5	6
Bill 30	Phyllis 31	Hank	Don 35 Plu	sorminus10	

Now, the following statements are executed from the keyboard:

```
ASSIGN #1 TO "AGES.DRIVE1"
READ #1,1
TYP(1)
 2
```

Opens AGES file.
Moves pointer to beginning of record 1.

```
READ #1;N$
TYP(1)
 1
```

Moves pointer past first item in record 1.

```
READ #1;A
TYP(1)
 4
```

Moves pointer past 2nd item in record 1.

```
READ #1,3;N$
TYP(1)
 4
```

Moves pointer past first item in record 3.

```
READ #1,4;N$,A
TYP(1)
 8
```

Moves pointer past first two items in record 4.

```
READ #1,5
TYP(1)
 10
```

Moves pointer to beginning of record 5.

```
READ #1,6
TYP(1)
 3
```

Moves pointer to beginning of record 6.

Copying Files

Any file not secured against copying can be copied from one disc to another. The COPY statement copies the specified file and adds the name of the copied file to the destination medium's file directory.

```
COPY "source file specifier" TO "destination file specifier"
```

The destination file can be given the same or a different file name. If the destination file name already exists on the destination medium, the computer returns Error 63 : DUP NAME.

You cannot copy a file secured against copying (type 1 security). If you attempt to do so, no error is generated, but the secured file is not copied to the destination medium.

Example: The following statement copies the file named SPEEDS on disc with volume label ".DRIVE0" into a new file named VELOCITY on the disc having volume label ".DRIVE1".

```
COPY "SPEEDS.DRIVE0" TO "VELOCITY.DRIVE1"
```

Copying an Entire Disc

The COPY statement can be used to copy all the files on a specified disc to another disc. The source disc's files are added to the destination disc without affecting the original contents of the destination medium.

```
COPY  ".source volume label"   TO   ".destination volume label"
      ":source msus"               ":destination msus"
```

If duplicate file names are encountered during copying, Error 63 : DUP NAME is generated, and the copy operation terminates. All files copied up to the termination remain intact.

Files secured against copying (type 1 security) are not copied when the entire contents of one disc are copied to another disc. The secured file is simply ignored, and no error is generated.

If there is not enough space on the destination medium to hold all the files being copied, the copy operation terminates and Error 128 : FULL results when the available space is exhausted. Copying also terminates when the directory space on the destination storage medium is exhausted, generating Error 124 : FILES. Files copied before generation of the error remain intact.

Example: The following statement copies the entire contents of the disc located at msus ":D701" to the disc having volume label ".FLOPPY".

```
COPY ":D701" TO ".FLOPPY"
```

Renaming Files

Any file, regardless of its type, can be given a new name using the RENAME statement:

```
RENAME "old file specifier"  TO  "new file name"
```

The old file specifier must correspond to a currently-existing file specifier. When the statement is executed, the name of the file as listed in the file directory is changed. Thereafter, the file must be accessed using the new name.

Example:

```
RENAME "AGES.DRIVE1" TO "BIRTHDATE"
```

Renames AGES on DRIVE 1 to BIRTHDATE.

Purging Files

The PURGE statement prevents further access to a file and removes the file name from the directory.

```
PURGE  "file specifier" [, purge code]
```

The file specifier can correspond to an existing file of any type. The purge code can be any number; however, a purge code other than zero is ignored.

When a file is purged without a purge code, the file name is removed from the file directory, and NULL is substituted for the type of file in the Type column of the directory. The NULL file is available for future use, and will be used the first time you create another file of any type that fits into the available space.

When a purge code of 0 is included, the specified file and all files after it on the storage medium are purged. The directory does not create NULL files; the directory will contain a listing for only those files up to (and not including) the file specified in the PURGE statement.

Examples: The following directories show the results of purging a file without a purge code. The file ODDNUMBERS is replaced by a NULL file.

```
CAT ":DRIVE1"

[ Volume ]: DRIVE1
Name        Type  Bytes  Recs
SPEEDS      PROG    256     3
DATA1       DATA     80     3
ODDNUMBERS  PROG    256     2
EVEN1       PROG    256     1
TESTS       DATA    256     1
SERIAL      PROG    256     4

PURGE "ODDNUMBERS:DRIVE1"
CAT ":DRIVE1"

[ Volume ]: DRIVE1
Name        Type  Bytes  Recs
SPEEDS      PROG    256     3
DATA1       DATA     80     3
            NULL    256     2
EVEN1       PROG    256     1
TESTS       DATA    256     1
SERIAL      PROG    256     4
```

Now, the last two files in the directory, TESTS and SERIAL, will be purged:.

```
PURGE "TESTS:DRIVE1",0
CAT ":DRIVE1"

[ Volume ]: DRIVE1
Name        Type  Bytes  Recs
SPEEDS      PROG    256     3
DATA1       DATA     80     3
            NULL    256     2
EVEN1       PROG    256     1
```

Packing a Disc

The PACK statement removes NULL files generated when files are purged without a 0 purge code.

```
PACK[" ,volume label" ]
     " :msus"
```

Example:

```
PACK ":DRIVE1"
CAT ":DRIVE1"
```

```
[ Volume 3: DRIVE1
Volume        Type  Bytes  Recs
SPEEDS        PROG    256     3
DATA1         DATA     80     3
EVEN1         PROG    256     1
```

The time required to pack a disc varies with the number of files on the disc. The operation may take several minutes.

File Security

Files can be secured to prevent program files from being listed, duplicated, or overwritten, and to prevent data files from being overwritten or copied. You can also remove a file name from the directory listing without creating a NULL file; the file can still be accessed by anyone who knows its name.

Securing Files

The SECURE statement places various types of security on files.

SECURE "*file specifier*", "*security code*", *security type*

The file specifier must refer to an existing file of the proper type.

The security code is a quoted string or a string expression that becomes associated with the file for security types 0 and 1. Only the first two characters of the security code string are actually used; uppercase and lowercase letters can be used interchangeably. If the string has only one character, the second character is a blank.

The security type is an integer from 0 through 3, and designates the type of security:

Security Type	File Type	Effect
0	PROG	Protects file against LIST, PLIST, and editing operations. The file can be loaded, run, and traced. The file name remains in the file directory.
1	PROG,BPGM	Same as type 0, but also protects the file against duplication. An attempt to store the program in another file generates an error.
2	PROG,BPGM DATA, GRAF	Prevents the file from being overwritten. Attempts to store or print# to the file generate Error 22 : SECURED. However, the file can be duplicated.
3	All types	File name is removed from the directory. The file can still be accessed by anyone knowing its name.

A security type greater than 3 is reduced MOD 4 to the range 0 through 3.

You can secure a file with more than one security type by executing more than one SECURE statement for the same file. However, a file cannot be secured for both types 0 and 1 security simultaneously.

Regardless of the types(s) security specified, a file can always be purged.

Examples:

```
SECURE "EVEN1.DRIVE1","NOLIST",0
```

Establishes file security type 0. "NO" is the security code.

```
SECURE "EVEN1.DRIVE1","STORENOT",2
```

Establishes file security type 2. The security code is ignored. (The file can be UNSECURED with any security code.)

Type 3 security has the following effect on the file directory:

```
SECURE "SPEEDS.DRIVE1","DONTCAT", 3
CAT ".DRIVE1"
```

```
[ Volume 3: DRIVE1
Name          Type    Bytes    Recs
              PROG      256       3
DATA1         DATA       80       3
EVEN1         PROG      256       1
```

The name SPEEDS is removed from the file directory.

Removing File Security

The UNSECURE command cancels previously established file security.

UNSECURE "*file specifier*", "*security code*", *security type*

The security type (0 through 3) must correspond to the security type previously established with a SECURE statement that you wish to cancel. For types 0 and 1 security, the security code must match the security code established by the SECURE statement. Any two characters can be used for the security code for types 2 and 3 security.

Examples:

```
UNSECURE "EVEN1.DRIVE1","NO",0
```

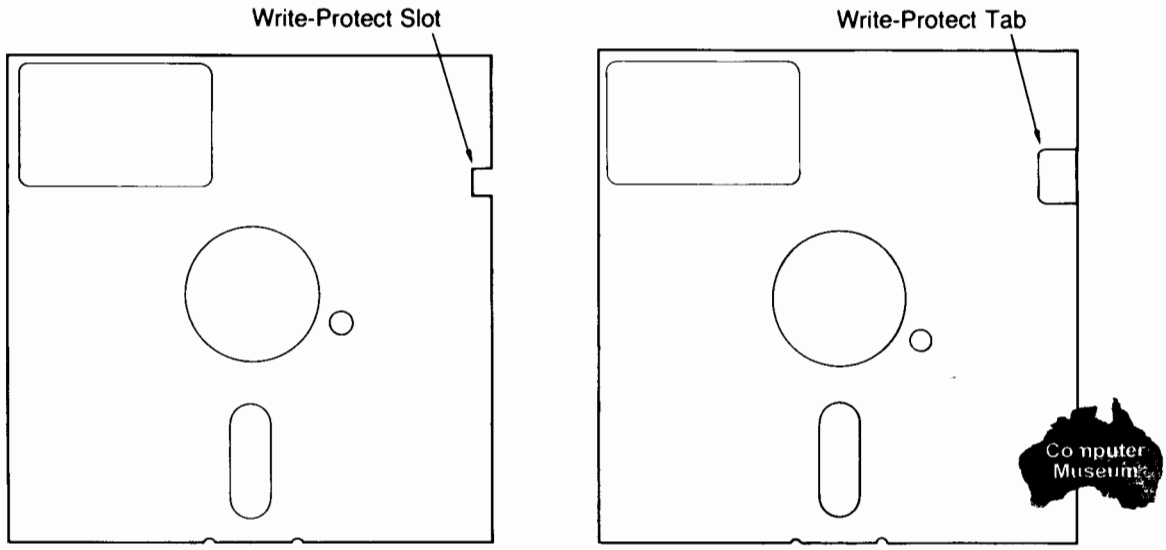
Removes previously established type 0 security. The security code matches the SECURE statement.

```
UNSECURE "SPEEDS.DRIVE1","OK",3
```

Restores the file name SPEEDS to the file directory. The security code need not match the SECURE statement.

Disc Write-Protection

You can prevent any write operation onto a disc by covering the write-protect tab (provided with the discs) as shown in the illustration below.



Not Write Protected **Write Protected**

The procedure for write-protecting other discs may not be the same as the above. Refer to the documentation for your mass storage device for write protection information.

The write-protect procedure prevents you from writing any information onto the disc. However, the disc can be read normally. To write to a protected disc, remove the write-protect tab.

Verification of Data

The CHECK READ# statement is used to verify that data printed to a disc data file has been properly recorded onto the disc. When CHECK READ# is activated, an immediate READ# operation is performed on any data printed through the specified buffer. If the two lists do not match, indicating failure of the storage medium (disc) itself, the computer returns Error 127 : READ VFY (read verify).

```
CHECK READ# buffer number
```

CHECK READ# errors are rare. If you encounter one, you may wish to compare your PRINT# statement to the contents of the data file. Then try re-executing the PRINT# statement, since the failure which generated the error may have been momentary. If you obtain another CHECK READ# error, it is likely that the disc has failed.

CHECK READ# is turned off by the CHECK READ OFF# statement:

```
CHECK READ OFF# buffer number
```

Examples:

`CHECK READ# 1`	Verifies all data printed to buffer #1.
`CHECK READ OFF# 1`	Turns off `CHECK READ#` for buffer #1.

Extended Files

Extended files, or source files, include all file types except `PROG`, `DATA`, `BPGM`, and `NULL` files. The `GSTORE` and `GLOAD` statements access a type of extended file (`GRAF`). In addition, certain optional ROM modules create and access various types of extended files.

When an extended file is created using the programming capabilities provided by an optional ROM module or a binary program, the entry in the `Type` column of the directory is a mnemonic name for that file type. If the ROM is removed, however, the directory listing indicates file type `****` for extended files generated by that ROM. The `****` entry indicates the file cannot be accessed by the computer in its current configuration.

Notes

Appendix A

Accessories

Standard Accessories

The items packaged with your computer are listed in section 2 of the introductory manual.

Optional Accessories

In addition to the standard accessories shipped with your computer, Hewlett-Packard makes available the following optional accessories:

HP Series 80 Plug-in Modules

- The following memory modules can be plugged into the module ports on the rear of the computer to increase the amount of memory available for programming and data storage. You can plug in as many modules as you'd like, subject to the number of available ports.

  32K Memory Module (HP 82907A).
  64K Memory Module (HP 82908A).
  128K Memory Module (HP 82909A).

- The HP 82936A ROM Drawer allows you to add up to six plug-in ROMs to your system.
- Each of the available enhancement ROMs is designed to increase the programming capabilities and versatility of your computer. For additional information about currently available and future enhancement ROMs compatible with your computer, contact your local authorized HP Series 80 dealer or the nearest HP sales and service facility.
- Series 80 interface modules allow the computer to communicate with a wide variety of peripheral devices. Up to four interface modules can be installed, subject to the number of available ports.

  HP 82937A HP-IB Interface.
  HP 82939A Serial Interface.
  HP 82940A GPIO Interface.
  HP 82941A BCD Interface.
  HP 82949A Printer Interface (Parallel Printer Interface).

- The HP 82928A System Monitor greatly facilitates the debugging and modification of binary programs written using the Assembler ROM.
- The HP 82929A Programmable ROM Module provides the ability to add erasable, customer-programmed ROM (EPROM) enhancements to the computer.
- The HP 82900A CP/M® System allows you to use the HP-86/87 to develop and/or run programs using the CP/M operating system.
- The HP 82950A Modem provides easy-to-use data communication by telephone between HP Series 80 Personal Computers and other computers.

CP/M® is a registered trademark of Digital Research, Inc.

HP Series 80 Applications Software

Each piece of applications software offers one or more programs in a particular field or discipline recorded on a mass storage medium. Included with each product is a detailed instruction manual and, where appropriate, a quick reference card.

For information regarding availability of applications software, contact your local authorized HP Series 80 dealer or the nearest HP sales and service facility. The user's newsletter *Basic Exchange* frequently includes discussions of available software.

Owner's Documentation

Additional copies of the owner's documentation can be purchased. Specify each document by its title and reorder number printed on the title page.

Three-Ring Manual Binders and Dividers

Additional HP Series 80 manual binders (part number HP 82935A) can be purchased, enabling you and members of your staff to organize your Series 80 documentation conveniently. The binder, measuring 29 cm (11.5 in.) high, 28 cm (11 in.) long, and 6.5 cm (2.5 in.) wide, includes sheet lifters and a set of dividers.

Ordering Accessories

Contact your local authorized HP Series 80 dealer or your nearest sales and service facility for further information on ordering and purchasing accessory items. If you are unable to locate your local dealer, you can obtain that information by contacting:

In the United States:

Hewlett-Packard
Personal Computer Division
1010 N.E. Circle Blvd.
Corvallis, OR 97330

Toll-free number (7 a.m. to 4:30 p.m., Pacific Time)
Call (800) 547-3400 (except Oregon, Alaska, and Hawaii)

Oregon, Alaska, Hawaii: Tel. (503) 758-1010

TTY users with hearing or speech
impairments, please dial (503) 758-5566

In Europe:

Hewlett-Packard S.A.
7, rue du Bois-du-Lan
P. O. Box
CH-1217 Meyrin 2
Geneva
Switzerland

Other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Tel. (415) 857-1501

Appendix B

Customer Support and Training

Hewlett-Packard makes available a variety of technical and educational services to support its Series 80 Personal Computers.

Basic Exchange

The *Basic Exchange* newsletter, published quarterly, assists owners of HP Series 80 Personal Computers in deriving maximum benefit from their systems. It is the primary vehicle for disseminating software updates and corrections and for informing users of important developments within the HP Series 80 product line. Articles discuss a variety of topics, including programming techniques, Users' Library programs, answers to frequently asked questions, and overviews of the computers' versatility. In addition, the newsletter carries announcements of new HP products, promotional offers, and price changes.

To provide us with a mailing address to which we can send your copies, please fill out and return the card packaged with your owner's documentation.

Series 80 Users' Library

The Series 80 Users' Library maintains a compilation of programs written by Hewlett-Packard, Series 80 Personal Computer users, and certain software suppliers. Library programs can be ordered by non-members as well as by members. The brochure packaged with your owner's documentation provides additional information about the library, including the benefits of becoming a library member.

Obtaining Programming and Applications Assistance

The dealer or HP sales and service facility where you purchased your computer should be able to answer most questions from first-time users.

Hewlett-Packard has developed support service programs to provide assistance to customers in obtaining maximum benefit from their HP Series 80 computer systems. Contact your local authorized HP Series 80 dealer or the nearest HP sales and service facility for further information.

Customer Training Courses

The *HP Series 80 Beginner's Course* is designed to teach customers with little or no programming experience the fundamentals of operating and programming their Series 80 Personal Computers. Other courses, currently available or under development, are designed for intermediate and advanced programmers.

For information on course schedules, cost, and availability, contact your nearest HP sales and service facility.

Service Contracts

Hewlett-Packard recommends that you consider purchasing a service contract for your computer. For additional information on the types of service contracts available, contact your local authorized HP Series 80 dealer or your nearest HP sales and service facility.

Notes

Appendix C

Maintenance and Service

Operational Considerations

General Cleaning

Disconnect the computer from its ac power source before cleaning.

The computer can be cleaned using a soft cloth dampened with clean water or with water containing a mild detergent. Do not use any abrasive cleaners, especially on the display screen.

The display screen can be cleaned using a soft cloth dampened with most non-abrasive household glass cleaners. Do not use any oil-based or wax-based cleaner. Do not spray the cleaner onto the screen, and avoid splashing glass cleaner onto the computer case.

Potential for Radio/Television Interference

For U.S.A. Only:

This HP Series 80 Personal Computer generates and uses radio frequency energy and may cause interference to radio and television reception. Your computer complies with the specifications in Subpart J of Part 15 of the Federal Communications Commission rules for a Class B computing device. These specifications provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If the computer does cause interference to radio or television reception, which can be determined by turning the computer off and on, you can try to eliminate the interference problem by doing one or more of the following:

- Reorient the receiving antenna.
- Change the position of the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet so that the computer and receiver are on different branch circuits.

If necessary, consult an authorized HP dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet, prepared by the Federal Communications Commission, helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C., 20402, Stock No. 004-000-00345-4.

Power-on Procedure

When the power is switched on, the computer performs the following tests before the cursor appears:

- **Self-test.** The length of time required to complete the self-test depends on the amount of random access memory present. If the computer detects a circuitry problem, it displays the message `Error 23 : SELF TEST`. Failure to pass the self-test indicates that the computer is not operating properly and may require service. Contact your local authorized dealer or your nearest HP sales and service facility for additional information.
- **ROM Check.** The computer searches read-only memory to determine whether any incompatible ROM modules have been installed. Use only ROM modules designed for your model Series 80 computer. If you receive `Warning 27 : 85 ROM IGNORED`, you have installed an incompatible ROM. In some cases, the presence of an incompatible ROM causes the computer to operate improperly. To remove an incompatible ROM, follow the instructions for removing ROMs provided in section 2 of the introductory manual.
- **Mass Storage Search.** The computer searches for the disc drive unit and drive with the lowest mass storage unit specifier. This location becomes the default mass storage address. The HP-87 requires that the disc drive unit be switched on in order for it to be detected. If a disc is installed in the default drive, the system searches that disc for an `Autost` (*autostart*) program.

System Self-Test

If you suspect that the computer is malfunctioning due to a problem in its circuitry, use the (TEST) key to perform the system self-test. The self-test is an electronic check of the computer's internal components.

If everything is working properly, the computer displays the following characters at the end of the test. The last two characters will vary, depending on the contents of computer memory.

```
◀⌂⌧ÑαßΣστΦΘΩδ∞∅∈∩≡±≥≤⌠⌡÷≈°∙·√ⁿ²■ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂ÆS
```

The graphics display will be erased. However, programs and variables in computer memory remain intact.

If the system is not operating properly, it displays `Error 23 : SELF TEST`. If you receive Error 23, your system requires servicing; contact your nearest HP dealer or HP sales and service facility for further information.

Warranty Information

The complete warranty statement is included in the information packet shipped with the computer. Please retain this statement for your records.

If you have any questions concerning this warranty, please contact:

In the United States: One of the six Field Repair Centers listed on the Warranty and Service Information Sheet packaged with your owner's documentation.

In other countries: Contact your nearest sales and service facility. If you are unable to contact that facility, please contact:

In Europe:

Hewlett-Packard
7, rue du Bois-du-Lan
P. O. Box
CH-1217 Meyrin 2
Geneva
Switzerland

Tel. (022) 82 70 00

Other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.

Tel. (415) 857-1501

Service

If at any time you suspect the computer is malfunctioning, the following procedures should help you determine if the computer requires servicing.

Check to see whether the power light is on. If the light is off and the power switch is set to ON, set the switch to OFF and then:

1. Determine that the voltage selector switch is set to the correct nominal line voltage for your area (115 Vac or 230 Vac).
2. Unplug the power cord from the power outlet. Then, remove the power cord from the power cord receptacle on the back of the computer. Inspect the power contacts on both the computer and power cord and clean them if necessary.
3. Check to see that the correct fuse is installed for the power supply in your area. Refer to section 2 of the introductory manual for fuse specifications.
4. Make sure the power cord is securely plugged into the computer and into a grounded ac outlet. Turn the power back on. If the power light fails to come on, the computer requires service.

If the power light comes on but no cursor appears, adjust the display brightness. If the screen remains blank or behaves erratically, or if the keyboard fails to respond to keyboard commands:

1. Reset the computer (press [RESET]) and press [TEST].
2. If the computer fails the self-test, turn the power off.
3. Examine the system to insure that modules and cables are installed properly and have not become loosened.
4. Turn the power on. If the cursor fails to appear, the computer requires service. If the cursor appears, perform the self-test. If the computer fails the self-test, it requires service.

Obtaining Repair Service

Not all Hewlett-Packard facilities offer service for your computer. For information on obtaining service in your area, consult the service information included in the Service Information Sheet packaged with your owner's documentation, or contact your authorized HP Series 80 dealer or the nearet Hewlett-Packard sales and service facility. (Addresses are listed in the back of this manual.)

If your computer requires repair, you can help assure efficient servicing by following these guidelines:

- Leave the configuration of the computer exactly as it was at the time of the malfunction; any plug-in modules and flexible discs in use at that time should be kept in place.
- Write a description of the malfunction symptoms for Service personnel.
- Save printouts or any other materials that illustrate the problem.
- Have on hand a sales slip or other proof of purchase to establish the warranty coverage period.

Serial Number

Each HP Series 80 Personal Computer carries an individual serial number plate on the rear panel. We recommend that owners keep a separate record of this number. Should your unit be lost or stolen, the serial number is often necessary for tracing and recovery, and for any insurance claims.

Hewlett-Packard does not maintain records of individual owners of Series 80 computers and unit serial numbers.

General Shipping Instructions

Should you ever need to ship your computer, be sure it is packed in a protective package to avoid in-transit damage. Use the original shipping case if possible. Shipping damage is not covered by the warranty. All customs and duties are the customer's responsibility.

Hewlett-Packard recommends that the customer always insure shipments.

Further Information

Should other problems or questions arise regarding repairs, please contact your local authorized HP Series 80 dealer or your nearest HP sales and service facility.

Not all Hewlett-Packard repair facilities offer service for HP Series 80 Personal Computers. However, you can be sure that servicing can be obtained in the country where you bought your computer.

If you are outside the country where you bought the computer, contact the nearest HP sales and service facility. All customs and duties are the customer's responsibility.

Notes

Appendix D

Reference Tables

Character and Key Codes

A numeric code is attached to each character and/or key. Non-keyboard characters are displayed on the CRT by using the [CTRL] key (indicated by superscript **c**) or the [SHIFT] key (indicated by superscript **s**) with the indicated key.

Each character with a decimal mode in the range 0 through 127 has a complementary inverse video character with a decimal code in the range 128 through 255. Inverse video characters are displayed using the `CHR$` function. For instance, `CHR$(74+128)` is inverse video J. Inverse video characters are also displayed by certain keys when a program is paused at an `INPUT` statement.

Characters with decimal codes 0 through 31 are referred to as control characters; they are interpreted by certain peripheral devices as instructions. For convenience, ASCII* convention assigns a mnemonic to each control character.

		EQUIVALENT FORMS			EQUIVALENT FORMS			EQUIVALENT FORMS			EQUIVALENT FORMS	
Mnemonic	Char.	Binary	Dec	Char.	Binary	Dec	Char.	Binary	Dec	Char.	Binary	Dec
NUL	$@^c$	00000000	0	SPACE	00100000	32	@	01000000	64	`	01100000	96
SOH	A^c	00000001	1	!	00100001	33	A	01000001	65	a	01100001	97
STX	B^c	00000010	2	"	00100010	34	B	01000010	66	b	01100010	98
ETX	C^c	00000011	3	#	00100011	35	C	01000011	67	c	01100011	99
EOT	D^c	00000100	4	$	00100100	36	D	01000100	68	d	01100100	100
ENQ	E^c	00000101	5	%	00100101	37	E	01000101	69	e	01100101	101
ACK	F^c	00000110	6	&	00100110	38	F	01000110	70	f	01100110	102
BEL	G^c	00000111	7	'	00100111	39	G	01000111	71	g	01100111	103
BS	H^c	00001000	8	(	00101000	40	H	01001000	72	h	01101000	104
HT	I^c	00001001	9	)	00101001	41	I	01001001	73	i	01101001	105
LF	J^c	00001010	10	*	00101010	42	J	01001010	74	j	01101010	106
VT	K^c	00001011	11	+	00101011	43	K	01001011	75	k	01101011	107
FF	L^c	00001100	12	,	00101100	44	L	01001100	76	l	01101100	108
CR	M^c	00001101	13	-	00101101	45	M	01001101	77	m	01101101	109
SO	N^c	00001110	14	.	00101110	46	N	01001110	78	n	01101110	110
SI	O^c	00001111	15	/	00101111	47	O	01001111	79	o	01101111	111
DLE	P^c	00010000	16	0	00110000	48	P	01010000	80	p	01110000	112
DC1	Q^c	00010001	17	1	00110001	49	Q	01010001	81	q	01110001	113
DC2	R^c	00010010	18	2	00110010	50	R	01010010	82	r	01110010	114
DC3	S^c	00010011	19	3	00110011	51	S	01010011	83	s	01110011	115
DC4	T^c	00010100	20	4	00110100	52	T	01010100	84	t	01110100	116
NAK	U^c	00010101	21	5	00110101	53	U	01010101	85	u	01110101	117
SYN	V^c	00010110	22	6	00110110	54	V	01010110	86	v	01110110	118
ETB	W^c	00010111	23	7	00110111	55	W	01010111	87	w	01110111	119
CAN	X^c	00011000	24	8	00111000	56	X	01011000	88	x	01111000	120
EM	Y^c	00011001	25	9	00111001	57	Y	01011001	89	y	01111001	121
SUB	Z^c	00011010	26	:	00111010	58	Z	01011010	90	z	01111010	122
ESC	$[^c$	00011011	27	;	00111011	59	[	01011011	91	{ [/]^s	01111011	123
FS	$\backslash^c$	00011100	28	<	00111100	60	\	01011100	92	|	01111100	124
GS	$]^c$	00011101	29	=	00111101	61	]	01011101	93	} [-]^s	01111101	125
RS	$^{\wedge c}$	00011110	30	>	00111110	62	^	01011110	94	~ [*]^s	01111110	126
US	$\_^c$	00011111	31	?	00111111	63	_	01011111	95	▒ [+]^s	01111111	127

*American Standard Code for Information Interchange.

Reset Conditions

The following table shows the status of specific parameters when the indicated commands are executed.

"R" designates a parameter or condition restored to power-on values.

"—" indicates the command has no effect on the parameter's status.

The **Power-On** column indicates the default value of the parameter at power-on.

Parameter or Condition	Power-On	RESET	SCRATCH	RUN	CHAIN	INIT	CONT
Program variables (except COMmon variables)	none	—	R	R	R	R	—
Calculator variables	none	R	R	R	R	R	R
COMmon variables	none	—	R	R	—	R	—
Result	0	R	R	—	—	—	—
Trigonometric Mode	RAD	R	—	—	—	—	—
Typing Mode	BASIC	—	—	—	—	—	—
PAGESIZE	16	—	—	—	—	—	—
Default Errors	DEFAULT ON	R	—	—	—	—	—
PRINT ALL Mode	off	R	—	—	—	—	—
Output Device	CRT IS 1 PRINTER IS 2 PLOTTER IS 1 MASS STORAGE IS *lowest address*	R	—	—	—	—	—
Display Apportionment	ALPHA/GRAPH	R	—	—	—	—	—
Typing Aids	See page 17	—	—	—	—	—	—
User-defined keys	none	R	R	R	R	R	—
DATA Pointers	none	R	R	R	R	R	—
System Timer	0	—	—	—	—	—	—
Random Number Seed	default value	R	—	—	—	—	—
ON TIMER# ON KEY# ON ERROR	off	R	R	R	R	R	—
TRACE TRACE ALL	off	R	R	—	—	—	—
TRACE VAR	off	R	R	—	—	—	—
Binary Programs	none	—	R	—	—	—	—
LIMIT	default value	R	—	—	—	—	—
Scaling Units	UUs = GUs	R	—	—	—	—	—
LOCATE	default limit	R	—	—	—	—	—
CLIP	default limit	R	—	—	—	—	—
LINETYPE	1 (solid)	R	—	—	—	—	—
CSIZE	CRT: 5 Plotter: 3	R	—	—	—	—	—
LORG	1 (lower-left corner)	R	—	—	—	—	—
LDIR	0° (horizontal)	R	—	—	—	—	—
PDIR	0°	R	—	—	—	—	—
Pen Color	PEN 1	R	—	—	—	—	—
Pen Status (up or down)	up	R	—	—	—	—	—
Pen Location	(0,0)	R	—	—	—	—	—
ASSIGN# buffers	none	R	R	R	—	R	—

Key Response During Program Execution

The following table describes the response of the system when the specified key is pressed while a program is running and when a program is paused awaiting input.

During program execution:

"P" Pauses the program and causes the system to perform the key's indicated function. In *graph-all* mode, keys affecting the alpha display only pause the program.

"L" The specified key is live and performs its indicated function without halting execution.

"N" No effect; the key is deactivated.

When a program is paused at an INPUT statement:

"A" Key is active on input and performs its indicated function.

"C" Key produces character corresponding to the decimal code assigned to the key. In *alpha* and *alpha-all* modes, an inverse video character is displayed. In *graph* and *graph-all* modes, the key's decimal code is reduced MOD 128 to produce normal video characters.

Key	Decimal Code	Response In: *Alpha, Alpha-all* Modes	Response In: *Graph* Mode	Response In: *Graph-all* Mode	Key	Decimal Code	Response In: *Alpha, Alpha-all* Modes	Response In: *Graph* Mode	Response In: *Graph-all* Mode
k1	128	L/C	L/C	L/C	—	151	—	—	—
k2	129	L/C	L/C	L/C	—	152	—	—	—
k3	130	L/C	L/C	L/C	BACK SPACE	153	P/A	P/A	P/A
k4	131	L/C	L/C	L/C	END LINE	154	P/A	P/A	P/A
k8	132	L/C	L/C	L/C	BACK SPACE [s]	155	P/A	P/C	P/C
k9	133	L/C	L/C	L/C	k7	156	L/C	L/C	L/C
k10	134	L/C	L/C	L/C	-LINE	157	P/A	P/C	P/C
k11	135	L/C	L/C	L/C	I/R	158	P/A	P/C	P/C
-CHAR	136	P/A	P/C	P/C	←	159	P/A	P/C	P/C
CLEAR	137	L/A	L/C	N/C	E	160	P/A	P/C	P/C
—	138	—	—	—	k5	161	L/C	L/C	L/C
RESET	139	P/A	P/A	P/A	k6	162	L/C	L/C	L/C
INIT	140	P/C	P/C	P/C	↑	163	P/A	P/C	P/C
RUN	141	P*/C	P*/C	P/C	↓	164	P/A	P/C	P/C
PAUSE	142	P/A	P/A	P/A	k12	165	L/C	L/C	L/C
CONT	143	N†/C	N†/C	N†/C	RESLT	166	P/A	P/C	P/C
STEP	144	P/C	P/C	P/C	—	167	—	—	—
ROLL [s]	145	L/A	L/C	N/C	A/G	168	L/A	L/C	L/C
TEST	146	P/C	P/C	P/C	ROLL	169	L/A	L/C	N/C
k14	147	L/C	L/C	L/C	→	170	P/A	P/C	P/C
LIST	148	P/C	P/C	P/C	—	171	—	—	—
PLST	149	P/C	P/C	P/C	k13	172	L/C	L/C	L/C
KEY LABEL	150	L/A	L/C	N/C	TR/NORM	173	L/A	L/C	L/C

[s] Shifted key.

* System displays RUN.

† Causes computer to beep.

Memory Requirements of Variables

Variable Type	Precision	Bytes of Computer Random Access Memory	Bytes in Mass Storage
Simple numeric	REAL SHORT INTEGER	12 bytes 8 bytes 7 bytes	8 bytes 8 bytes 8 bytes
Simple string		11 bytes + 1 byte per character	3 bytes + 1 byte per character + 3 bytes each time the string crosses into a new logical record
Numeric array	REAL SHORT INTEGER	11 bytes + 8 bytes per element 11 bytes + 4 bytes per element 11 bytes + 3 bytes per element	8 bytes per element 8 bytes per element 8 bytes per element
String array		13 bytes per array; Per array element: 2 bytes + 1 byte per character	Per array element: 3 bytes + 1 byte per character + 3 bytes each time the string crosses into a new logical record

In addition, each variable requires one byte per character in the variable name.

Memory Requirements of User-Defined Functions

A program line can utilize a maximum of 255 bytes of computer memory. A user-defined function has the following memory requirements:

Per function:
10 bytes + 1 byte per character in the function name

For each numeric parameter:
13 bytes + 1 byte per character in the parameter name

For each string parameter:
12 bytes + 1 byte per character in the parameter name (excluding $) + 1 byte per character in the string

Example:

```
100 DEF FNhensteed(DAYSLATE,EXCUSE$[40])
```

Per function:	10 + 8 =	18 bytes
DAYSLATE	13 + 8 =	21 bytes
EXCUSE$	12 + 6 + 40 =	58 bytes
		97 bytes

Pen Status

The following table shows the pen status (up or down) after the indicated statements have been executed.

Statement	Pen Status	Statement	Pen Status
AXES	up	LGRID	up
DRAW	down	LIMIT	up
FRAME	up	MOVE	up
GRID	up	PENUP	up
IDRAW	down	PLOT	optional
IMOVE	up	PLOTTER IS	up
IPLOT	optional	RPLOT	optional
LABEL	up	XAXIS	up
LABEL USING	up	YAXIS	up
LAXES	up		

Appendix E

Glossary

The HP-87 BASIC language consists of operators, functions, statements, and commands. Operators and functions are used with numbers, numeric variables, character strings, and string variables to create numeric and string expressions. BASIC statements and commands are composed of BASIC language keywords and numeric and string expressions.

Operators

Arithmetic

+	Add
-	Subtract
*	Multiply
/	Divide
^	Exponentiation
MOD	Modulo: *A* MOD *B* evaluates to the remainder of the division *A*/*B*. *A* MOD *B*=*A*-*B**INT(*A*/*B*)
\ or DIV	Integer division: *A**B* evaluates to the integer portion of the division *A*/*B*. *A* DIV *B*=IP(*A*/*B*)

Logical Evaluation

Logical expressions return the values 0 (false) or 1 (true). Non-zero values are considered true; zero values are considered false.

Relational

=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or #	Not equal to

Strings are compared using the decimal character codes of the characters. Two strings are compared character by character from left to right until a difference is found. If one string ends before a difference is found, the shorter string is considered the lesser.

Logical

AND
OR
EXOR
NOT

Truth Table

A	B	A AND B	A OR B	A EXOR B	NOT A	NOT B
T	T	1	1	0	0	0
T	F	0	1	1	0	1
F	T	0	1	1	1	0
F	F	0	0	0	1	1

1 = true 0 = false

String Operators

&	Concatenation
[*first character*] [*first character* : *last character*]	Substring reference

Specifying only the first character returns the portion of the string from that character position to the end of the string. Specifying both first and last character returns the portion of the string bounded by those character positions.

Math Hierarchy

( )	Performed first
Functions	↓
^	
NOT	
*, /, MOD, \ or DIV	
+, −	
Relational operators (=, >, <, >=, <=, <> or #)	
AND	
OR, EXOR	Performed last

The computer scans the expression from left to right and performs an operation when the operation to the right has lower or equal priority. Operations within parentheses are performed first. Nested parentheses are evaluated outwardly from the most deeply nested set.

Data Precision

Precision	Accuracy	Range	Memory Requirement
REAL	12 digits	±9.99999999999E±499	12 bytes
SHORT	5 digits	±9.9999E±99	8 bytes
INTEGER	5 digits	−99999 through 99999	7 bytes

Special Characters

@	Delimits individual statements in multistatement line.
!	Remark follows.
?	Input prompt; the program awaits input.
" "	String delimiters; mark beginning and end of literal text.
■	Double cursor; the display is in insert mode.

Variables

Variable Names

Maximum length is 31 characters. Names may include any combination of letters, numbers, and the underscore character, except that the first character must be a letter. String variable names must end with a dollar sign, $.

Simple Numeric Variables

REAL precision is assumed unless SHORT or INTEGER type is declared.

Simple String Variables

The default maximum length is 18 characters unless otherwise specified in a DIM or COM statement. The maximum dimensioned length is limited to 65,530 characters, subject to the amount of available memory.

Numeric Arrays

One- and two-dimensional arrays are permitted. The upper bound is established by a dimensioning statement—DIM, REAL, SHORT, INTEGER, or COM. If the array is not dimensioned, the upper bound defaults to 10. The lower bound is determined by the OPTION BASE of the program; the default OPTION BASE is 0.

String Arrays

One- and two-dimensional arrays are permitted. The upper bound is established by a dimensioning statement—DIM or COM. The maximum number of characters per element is enclosed in brackets; for example, DIM NAME$(5)[50]. If the array is not dimensioned, the upper bound defaults to 10 and the maximum number of characters per element defaults to 18. The lower bound is determined by the OPTION BASE of the program; the default OPTION BASE is 0.

Image Specifiers

The following image specifiers are used to create field specifiers for numeric and string items in formatted printer and display output.

Image Specifiers

Type of Output	Character	Purpose	Replication
Blank space	X	Specifies a blank space between items.	Yes
Radix symbol	.	Specifies a decimal point in that position.	No
	R	Specifies a comma radix indicator in that position.	No
Digit	D	Digit position to left or right of radix symbol; leading blanks.	Yes
	Z	Digit position to left of radix symbol; leading zeros.	Yes
	*	Digit position to left of radix symbol; leading asterisks (*).	Yes
Digit separator	C	Specifies a comma as a separator in the specified position.	No
	P	Specifies a period as a separator in the specified position.	No
Exponent	E	Numeric field is output in exponential format; exponent consists of three digits plus sign.	No
Sign	S	Specifies sign, "+" or "−".	No
	M	Specifies sign, blank or "−".	No
String	" "	Specifies literal text.	No
	A	Specifies character position; text is left-justified.	Yes
Numeric or string	K	Specifies compact format with no leading or trailing blanks.	No
Miscellaneous	( )	Used to indicate replication of field specifier.	Yes
	/	Specifies a carriage return/line feed; can also be used as a delimiter.	Yes

Syntax Guidelines

The following conventions are used to describe the syntax of BASIC functions, statements and commands:

DOT MATRIX	Items in DOT MATRIX are BASIC language keywords and punctuation that must be entered exactly as shown, except that lowercase letters can be substituted for uppercase letters. The computer replaces any lowercase letters in BASIC keywords with uppercase letters.
italics	Items in italics are numeric and string expressions that must be included in the statement.
[ ]	Brackets are used to enclose optional items.
stacked items	When items are placed one above the other, one and only one must be chosen.
...	An ellipsis placed after an item or series of items within brackets indicates the contents of the brackets may be repeated.

Predefined Functions

ABS(*X*)	Absolute value of *X*.	**Page 43**
ACS(*X*)	Arcosine of *X*, in 1st or 2nd quadrant.	**Page 47**
ASN(*X*)	Arcsine of *X*, in 1st or 4th quadrant.	**Page 47**
ATN(*X*)	Arctangent of *X*, in 1st or 4th quadrant.	**Page 47**
ATN2(*Y*, *X*)	Arctangent of *Y*/*X*, in proper quadrant.	**Page 47**

CEIL(*X*)	Smallest integer ≥ *X*.	**Page 43**
CHR$(*X*)	Character whose decimal character code is *X*, 0 ≤ *X* ≤ 255.	**Page 58**
COS(*X*)	Cosine of *X*.	**Page 47**
COT(*X*)	Cotangent of *X*.	**Page 47**
CSC(*X*)	Cosecant of *X*.	**Page 47**
DATE	Julian date in format *yyddd* (assumes system timer has been set properly).	**Page 49**
DTR(*X*)	Degree to radian conversion.	**Page 47**
EPS	Smallest positive machine number (1E-499).	**Page 44**
ERRL	Line number of latest error.	**Page 155**
ERRN	Number of latest error.	**Page 155**
ERROM	ROM number of ROM that generated the most recent error.	**Page 157**
ERRSC	Select code of interface that generated the most recent error.	**Page 156**
EXP(*X*)	e^x	**Page 46**
FLOOR(*X*)	Same as INT(*X*) (relates to CEIL).	**Page 43**
FP(*X*)	Fractional part of *X*.	**Page 43**
FRE	Available bytes of computer memory.	**Page 94**
INF	Largest machine number (9.99999999999E499).	**Page 44**
INT(*X*)	Largest integer ≤ *X*.	**Page 43**
IP(*X*)	Integer part of *X*.	**Page 43**
LEN(*S$*)	Length of string *S$*.	**Page 55**
LGT(*X*)	Log to the base 10 of *X*, *X* > 0.	**Page 46**
LOG(*X*)	Natural logarithm, *X* > 0.	**Page 46**
MAX(*X*, *Y*)	If *X* > *Y* then *X*, else *Y*.	**Page 44**
MIN(*X*, *Y*)	If *X* < *Y* then *X*, else *Y*.	**Page 44**
NUM(*S$*)	Decimal character code of first character of *S$*.	**Page 58**
PI	3.14159265359	**Page 44**
POS(*S1$*, *S2$*)	Searches string *S1$* for the first occurrence of string *S2$*. Returns starting index if found, otherwise returns 0.	**Page 56**
RATIO	Returns the ratio of the physical dimensions of the graphics limits, horizontal dimension/vertical dimension.	**Page 178**
RMD(*X*, *Y*)	Remainder of *X*/*Y*: *X* − *Y* * IP(*X*/*Y*).	**Page 44**
RND	Next number, *X*, in a sequence of pseudo-random numbers, 0 ≤ *X* < 1.	**Page 44**
RTD(*X*)	Radian to degree conversion.	**Page 47**
SEC(*X*)	Secant of *X*.	**Page 47**
SGN(*X*)	The sign of *X*, −1 if *X* < 0, 0 if *X* = 0, and +1 if *X* > 0.	**Page 44**
SIN(*X*)	Sine of *X*.	**Page 47**
SQR(*X*)	Positive square root of *X*.	**Page 44**
TAB(*X*)	Skips to specified column.	**Page 130**
TAN(*X*)	Tangent of *X*.	**Page 47**
TIME	Time in seconds since midnight (assumes system timer has been set properly).	**Page 49**
TYP(*X*)	Determines data type of next item in file assigned buffer *X*.	**Page 299**
UPC$(*S$*)	Returns string with all lowercase alphabetic characters converted to uppercase.	**Page 59**
VAL(*S$*)	Returns the numeric equivalent of the string *S$*.	**Page 56**
VAL$(*X*)	String equivalent of *X*.	**Page 57**

BASIC Statements and Commands

Statements preceded by a line number are entered into computer memory as program lines. All statements are programmable and most can be executed from the keyboard. Exceptions are: COM, DATA, DEF FN, FN END, GOSUB, GOTO, IMAGE, INPUT, NEXT, ON...GOSUB, ON...GOTO, ON ERROR, ON KEY#...GOSUB, ON KEY#...GOTO, ON TIMER#, OPTION BASE, READ, RESTORE, and RETURN. FOR...NEXT loops can be executed from the keyboard if the entire loop is contained in one multistatement line. Commands must be executed from the keyboard; they cannot be included in programs.

Commands

AUTO [*beginning statement number* [, *increment value*]]	**Page 66**
CONT [*statement number*]	**Page 92**
DELETE *first statement number* [, *last statement number*]	**Page 88**
INIT	**Page 89**
LOAD "*file specifier*"	**Page 277**
REN [*first statement number* [, *increment value*]]	**Page 88**
RUN [*statement number*]	**Page 90**
SCRATCH	**Page 64**
STORE "*file specifier*"	**Page 275**
UNSECURE "*file specifier*", "*security code*", *security type*	**Page 304**

Statements

ALPHA	**Page 27**
ALPHALL	**Page 170**
ASSIGN# *buffer number* TO * / "*file specifier*"	**Page 288**
AXES [*x tick-spacing*, *y-tick spacing* [, *x-intersection*, *y-intersection* [, *x major count*, *y major count* [, *major tick size*]]]]	**Page 217**
BEEP [*tone*, *duration*]	**Page 76**
BLINK	**Page 245**
BPLOT *string expression*, *bytes per row*	**Page 235**
BREAD *string variable*, *bytes per row*	**Page 244**
CAT [" / ":*msus*" / ",*volume label*"]	**Page 271**
CHAIN "*file specifier*"	**Page 277**

CHECK READ [OFF] # *buffer number* **Page 305**

CLEAR **Page 130**

CLIP *x min* , *x max* , *y min* , *y max* **Page 193**

COM [*precision*] *item* [, *item*...] [, [*precision*] *item* [, *item*...] ...] **Page 278**

COPY "*source file specifier*" TO "*destination file specifier*" **Page 300**

COPY "*:source msus*" / "*,source volume label*" TO "*:destination msus*" / "*:destination volume label*" **Page 301**

CREATE "*file specifier*" , *number of records* [, *record length*] **Page 287**

CRT IS *device selector* [, *line length*] **Page 25**

CSIZE *height* [, *aspect ratio* [, *slant*]] **Page 226**

CURSOR *x-variable* , *y-variable* [, *pen status variable*] **Page 245**

DATA *constant* [, *constant*...] **Page 78**

DEF FN *numeric function name* [(*parameter* [, *parameter*...])] [= *numeric expression*] **Page 139, 142**

DEF FN *string function name* [(*parameter* [, *parameter*...])] [= *string expression*] **Page 139, 142**

DEFAULT ON / OFF **Page 83**

DEG **Page 46**

DIM *string variable name* [*string length*] [, *string variable name* [*string length*] ...] **Page 52**

DIM *item* [, *item*...] **Page 114, 117**

String variable item:
string variable name [*length of string*]

Numeric array item:
numeric variable name (*upper bound* [, *upper bound*])

String array item:
string variable name (*upper bound* [, *upper bound*]) [*length per element*]

DISP [*item* [; / , *item*...]] **Page 72**

DISP USING "*format string*" / *statement number* / *statement label* [; *item* [; / , *item*...]] **Page 122**

DRAW *x-coordinate* , *y-coordinate* **Page 211**

END **Page 80**

FLIP **Page 18**

FN*numeric function name*=*numeric expression* **Page 142**

FN*string function name*=*string expression* **Page 142**

FN END **Page 142**

FOR *loop counter*=*numeric expression* TO *numeric expression* [STEP *numeric expression*] **Page 104**

FRAME **Page 219**

FXD *number of digits* [, *number of digits*] **Page 231**

GCLEAR [*y-coordinate*] **Page 171**

GLOAD "*file specifier*" **Page 281**

GOSUB *statement label* / *statement number* **Page 135**

GOTO *statement label* / *statement number* **Page 98**

GRAD **Page 46**

GRAPH **Page 167**

GRAPHALL **Page 168**

GRAPHICS **Page 167**

GRID [*x tick-spacing*, *y tick-spacing* [, *x-intersection*, *y-intersection*
[, *x grid-spacing*, *y grid-spacing* [, *minor tick size*]]]] **Page 220**

GSTORE "*file specifier*" **Page 280**

IDRAW *x-increment*, *y-increment* **Page 211**

IF *logical expression* / *numeric expression* THEN *statement label* / *statement number* / *executable statement* [ELSE *statement label* / *statement number* / *executable statement*] **Page 100**

IMAGE *format string* **Page 122**

IMOVE *x-increment*, *y-increment* **Page 211**

INITIALIZE ["*new volume label*" [, " : *msus*" / , *old volume label*" [, *directory size*
[, *interleave factor*]]]] **Page 269**

INPUT *variable name* [, *variable name*...] **Page 74**

INTEGER *item* [, *item*...] **Page 115**

IPLOT *x-increment* , *y-increment* [, *pen control*] **Page 206**

KEY LABEL **Page 147**

LABEL [*label list*] **Page 222**

LABEL USING { "*format string*" / *statement number* / *statement label* } [; *label list*] **Page 222**

LAXES [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x major count* , *y major count* [, *major tick size*]]]] **Page 232**

LDIR { *angle* / *run* , *rise* } **Page 225**

[LET] *numeric variable* [, *numeric variable*...] = *numeric expression* **Page 77**

[LET] *string variable* [, *string variable*...] = *string expression* **Page 77**

LGRID [*x tick-spacing* , *y tick-spacing* [, *x-intersection* , *y-intersection* [, *x grid-spacing* , *y grid-spacing* [, *minor tick size*]]]] **Page 232**

LIMIT *x min* , *x max* , *y min* , *y max* **Page 176**

LINE TYPE *type number* [, *length*] **Page 201**

LIST [*beginning statement number* [, *ending statement number*]] **Page 89**

LOADBIN "*file specifier*" **Page 281**

LOCATE *x min* , *x max* , *y min* , *y max* **Page 188**

LORG *label position* **Page 224**

MASS STORAGE IS { ":*msus*" / ":*volume label*" } **Page 270**

MOVE *x-coordinate* , *y-coordinate* **Page 210**

MSCALE *x-offset* , *y-offset* **Page 184**

NEXT *loop counter* **Page 105**

NOBLINK **Page 245**

NORMAL **Page 80**

OFF ERROR **Page 151**

OFF KEY# [*key number*] **Page 149**

OFF TIMER# *timer number* **Page 150**

TRACE VAR *program variable* [, *program variable...*] **Page 158**

UNCLIP **Page 195**

VOLUME " : *msus* " " , *old volume label* " IS " *new volume label* " **Page 268**

WAIT *milliseconds* **Page 78**

WHERE *x variable* , *y variable* [, *pen status variable*] **Page 245**

XAXIS *y-intercept* [, *tick-spacing* [, *x min* , *x max*]] **Page 215**

YAXIS *x-intercept* [, *tick-spacing* [, *y min* , *y max*]] **Page 215**

Notes

Appendix F

Error Messages

Number	Message	ERROM Number	Error Condition (Default for errors 1 through 8 in parentheses.)
			Defaultable Errors
1	UNDERFLOW	0	Underflow; expression underflows machine. (0)
2	OVERFLOW	0	Overflow: • Expression overflows machine. (±9.99999999999E499) • Attempt to store integer value >99999 or <−99999 in INTEGER variable. (±99999) • Attempt to store value >9.9999E99 or <−9.9999E99 in SHORT variable. (±9.9999E99)
3	COT/CSC=INF	0	COT or CSC of n*180°; n = integer. (±9.99999999999E499)
4	TAN/SEC=INF	0	TAN or SEC of n*90°; n = integer. (9.99999999999E499)
5	0^NEG	0	Zero raised to negative power. (9.99999999999E499)
6	0^0	0	Zero raised to zero power. (1)
7	NULL DATA	0	Null data: • Uninitialized string variable or missing string function assignment. (" ") • Uninitialized numeric variable or missing numeric function assignment. (0)
8	/ZERO	0	Division by zero. (±9.99999999999E499)
			The remaining errors have no default values.
			Math Errors
9	NEG^NON-INT	0	Negative value raised to non-integer power.
10	SQR(-)	0	Square root of negative number.
11	ARG OUT OF RANGE	0	Argument (parameter) out of range: • ATN2 (0,0) • ASN or ACS with n > 1 or n <−1. • On expression GOTO/GOSUB; expression out of range.
12	LOG(0)	0	Logarithm of zero.
13	LOG(-)	0	Logarithm of negative number.
14			Not used.

Number	Message	ERROM Number	Error Condition
			System Errors
15	SYSTEM	0	System error: • Correct by reloading the program, (RESET), or power-off then power-on. • Calculator-mode FOR ... NEXT loop has generated a numeric result; ignore the error.
16	CONTINUE BEFORE RUN	0	Continue before run; program not allocated.
17	FOR NESTING	0	FOR nesting too deep; more than 255 levels of nesting.
18	GOSUB NESTING	0	GOSUB nesting too deep; more than 255 levels of nesting.
19	MEM OVF	0	Memory overflow: • Attempting to run a program that requires more than available memory. • Attempting to edit too large a program; delete a nonexisting line to deallocate the program, then edit. • Attempting to load a program larger than available memory. • Attempting to open a file with insufficient memory available for the buffer. • Attempting any operation that requires more memory than available. • Attempting to load or run a large program after a ROM has been installed. • Attempting a concatenation operation that produces a string larger than available memory. • Retrieving a long string with READ#.
20			Not used.
21	ROM MISSING	0	ROM missing: • Attempting to list or run a program that requires a plug-in ROM. An attempt to list or edit program with missing ROM will usually scratch memory. • Attempting to address a peripheral plotter with no HP-87 Plotter ROM installed.
22	SECURED	0	Attempting to edit, list, store, or overwrite a secured program.
23	SELF TEST	0	Self-test error; system needs repair.
24	MODE	0	Attempting to execute GLOAD in *alpha-all* mode.
25	BAD BIN LOAD	0	LOADBIN operation has failed: • Attempting to load a binary program with five binary programs present in memory. • Attempting to load absolute binary at address already occupied by the system or by another binary program. • Attempting to load a binary program whose binary program number matches a binary program already present.
26	STACK OVFL (warning)	0	Stack overflow: • Attempting to enter a line containing too many levels of nesting. • Attempting to call a user-defined function that is too complex.
27	85 ROM IGNORED (warning)	0	An HP-83/85 ROM is installed in the ROM drawer. The ROM is ignored by the system if no ROM with the same ROM number is present.
28 and 29			Not used.

Number	Message	ERROM Number	Error Condition
			Program Errors
30	OPTION BASE	0	Option base error: • Duplicate OPTION BASE declaration. • OPTION BASE after array declaration. • OPTION BASE parameter not 0 or 1.
31	CHAIN	0	CHAIN error; CHAIN to an invalid program; e.g., chaining to a binary program.
32	COM MISMATCH	0	Common variable mismatch.
33	DATA TYPE	0	DATA type mismatch: • READ variable and DATA type do not agree. • READ# found a string but required a number.
34	NO DATA	0	No data to read: • DATA list expired. • RESTORE executed with no data statement.
35	DIM EXIST VRBL	0	Dimensioned existing variable; attempt to dimension a variable that has been previously declared or used.
36	DIM ILLEGAL	0	Illegal dimension: • Illegal dimension in default array declaration. • Array dimensions don't agree.
37	DUP FN	0	Duplicate user-defined function.
38	NO FN END	0	Function definition within function definition; needs FN END.
39	FN MISSING	0	Reference to a nonexistent user-defined function: • Finding FN END with no matching DEF FN. • Exiting a function that was not entered with a function call after branching to the middle of a multiple-line function.
40	FN PARAM	0	Illegal function parameter; function parameter mismatch.
41	FN=	0	Function assignment does not occur between DEF FN and FN END.
42	RECURSIVE FN CALL	0	Recursive user-defined function.
43	NUMERIC INPUT	0	Numeric input required.
44	TOO FEW INPUTS	0	Fewer items were given than requested by an INPUT statement.
45	TOO MANY INPUTS	0	More items were given than requested by an INPUT statement.
46	NEXT MISSING	0	FOR with no matching NEXT.
47	NO MATCHING FOR	0	NEXT with no matching FOR.
48	END	0	END statement necessary.
49	NULL DATA	0	Uninitialized data.
50	BIN PROG MISG	0	Binary program missing; attempt to list or run program that requires a binary program. An attempt to edit or list will usually scratch memory.
51	RETURN W/O GOSUB	0	RETURN encountered before GOSUB reference.

Number	Message	ERROM Number	Error Condition
52	IMAGE	0	Illegal IMAGE format string: • Unrecognized character in IMAGE. • IMAGE string enclosed in quotes.
53	PRINT USING	0	Illegal PRINT USING: • Data overflows IMAGE declaration. • Numeric data with string IMAGE. • String data with numeric IMAGE. • PRINT USING image format string is incorrect.
54	TAB	0	Illegal TAB argument. With DEFAULT ON, an illegal TAB argument gives a warning message and defaults to TAB(1).
55	SUBSCRIPT	0	Array subscript out of range.
56	STRING OVF	0	String variable overflow; string too large for variable.
57	MISSING LINE	0	Reference to a non-existent statement.
58 and 59			Not used.
			Mass Storage Errors
60	WRITE PROTECT	0	Write protect: • Mass storage medium is write-protected. • File is secured.
61 and 62			Not used.
63	DUP NAME	0	Duplicate file name for RENAME, CREATE, or COPY.
64 and 65			Not used.
66	FILE CLOSED	0	File closed: Attempting to READ#/PRINT# to a closed file. Attempting to close a closed file.
67	FILE NAME	0	File name: • Name does not exist attempting to LOAD, ASSIGN#, LOADBIN, PURGE, RENAME, SECURE, COPY, UNSECURE, or GLOAD. • Name not in quotes. • Attempt to purge an open file.
68	FILE TYPE	0	File type mismatch: • Attempting to treat program file as data file, or vice versa. • Attempting to treat binary program as BASIC program, or vice versa. • Attempting to treat binary program file as data, or vice versa.
69	RANDOM OVF	0	Random overflow: • Attempting to READ#/PRINT# beyond existing number of bytes in logical record with random file access. • Attempting to randomly PRINT# a string to a logical record with fewer than 4 bytes available in the record.
70	READ	0	System cannot read mass storage medium.
71	EOF	0	End-of-file; file pointer has encountered EOF marker.
72	RECORD	0	Record: • Attempting to READ#/PRINT# to nonexistent record. • Attempting to READ#/PRINT# at end of file. • Lost in record; close file to release buffer.
73 through 79			Not used.

Number	Message	ERROM Number	Error Condition
			Syntax Errors
80	) EXPECTED	0	Right parentheses,), expected.
81	BAD EXPRESSION	0	Bad BASIC statement or bad expression. If it is an expression, try again with DISP <*expression*> to get a more descriptive error.
82	STRING EXPR	0	String expression error; e.g., right quote missing or null string given for a file name.
83	"," MISSING	0	Comma missing or more parameters expected (separated by commas).
84	EXCESS CHARS	0	Excess characters; extra characters at end of good line.
85	EXPR TOO BIG	0	Expression too long for system to interpret; certain expressions less than 159 characters in length cannot be entered; e.g.: • Complicated math expressions with numerous variable references. • DEF FN statements with numerous multi-character variables. • Expressions with many parentheses.
86	ILLEGAL AFTER THEN	0	Illegal statement after THEN.
87	BAD DIM STATEMENT	0	Bad dimension statement.
88	BAD STATEMENT	0	Bad BASIC statement or expression: • COM in calculator mode. • User-defined function in calculator mode. • INPUT in calculator mode. • Attempt to enter HP-85 keywords COPY, ERASETAPE, CTAPE, REWIND, or TRANSLATE as a program statement or variable name.
89	INVALID PARAM	0	Invalid statement or command parameter.
90	LINE>99999	0	Line number larger than 99999.
91	MISSING PARAM	0	Missing parameter in statement or command.
92	SYNTAX	0	Syntax error. Cursor returns to character where error was detected.
			ROM or I/O Errors
Certain error numbers greater than 100 have two entries. To determine which entry corresponds to the current error condition, match the error number with the appropriate error message or ERROM number.			
101	LIMIT OUT OF BOUND (warning)	1	Parameter in LIMIT statement is out of range.
102 through 108			Not used.
109	DIGITIZE	1	Attempt to digitize CRT.
110	ADDR	1	Invalid device address.

Number	Message	ERROM Number	Error Condition
110	I/O CARD	208	Interface card: • A plug-in interface module has the same select code as the computer integrated interface. • An interface assembly has failed the self-test or has failed after interrupting the computer; the interface requires servicing. Execute the ERRSC function to determine which interface generated the error.
111	SELECT CODE	1	Invalid HP-IB select code.
111	IOP	208	An illegal operation, statement, or command has been sent to an interface. Execute ERRSC to determine which interface generated the error.
112	P/P ROM	1	A portion of computer memory has failed the self-test; the computer requires service.
112	M.S. ROM	208	A portion of computer memory has failed the self-test; the computer requires service.
113 through 122			These error numbers are reserved for errors originating in interfaces. Use ERRSC to determine the source of the error and refer to documentation accompanying the interface.
120	NO M.S. DEVICE	208	No mass storage device is currently active.
124	FILES	208	The file directory on the storage medium is full.
125	VOLUME	208	The specified volume label wasn't found.
126	MSUS	208	The specified mass storage unit specifier wasn't found.
127	READ VFY	208	A read verify error occurred.
128	FULL	208	The statement or command cannot be executed because the mass storage medium is full.
129	MEDIUM	208	The mass storage medium is damaged.
130	DISC	208	The storage medium is not initialized, the drive latch is open, the drive number specified is not present, or the disc drive unit is not operating properly.
131	TIMEOUT	208	The interface select code or device address specified is not present, or system hardware has failed.

Notes

Index

D

I

K

L

M

Q-R

S

Product Line Sales/Support Key

Key	Product Line
A	Analytical
CM	Components
C	Computer Systems Sales only
CH	Computer Systems Hardware Sales & Services
CS	Computer Systems Software Sales & Services
E	Electronic Instruments & Measurement Systems
M	Medical Products
MP	Medical Products Primary SRO
MS	Medical Products Secondary SRO
P	Personal Computing Products
*	Sales only for specific product line
**	Support only for specific product line

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HP distributors are printed in italics.

ANGOLA
Telectra
Empresa Técnica de Equipamentos Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 41-I DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
E,M,P

ARGENTINA
Hewlett-Packard Argentina S.A.
Avenida Santa Fe 2035
Martinez 1640 **BUENOS AIRES**
Tel: 798-5735, 792-1293
Telex: 122443 AR CIGY
Cable: HEWPACKARG
A,E,CH,CS,P

Biotron S.A.C.I.y.M
Av Paseo Colon 221, 9 Piso
1399 ***BUENOS AIRES***
Tel: 30-4846, 30-1851, 30-8384
Telex: 17595 BION/AR
M

Fate S.A. I.C.I./Electronica
Venezuela 1326
1095 ***BUENOS AIRES***
Tel: 379026, 379027
Telex: 18137, 22754 ALVAR AR
P

AUSTRALIA

Adelaide, South Australia Office
Hewlett-Packard Australia Pty.Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,CH,CM,,E,MS,P

Brisbane, Queensland Office
Hewlett-Packard Australia Pty.Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
SPRING HILL, Queensland 4000
Tel: 229-1544
Telex: 42133
Cable: HEWPARD Brisbane
A,CH,CM,,E,MS,P

Canberra, Australia Capital Territory Office
Hewlett-Packard Australia Pty.Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2609
Tel: 80 4244 Telex: 62650
Cable: HEWPARD Canberra
A*CH,CM,E,MS,P

Melbourne, Victoria Office
Hewlett-Packard Australia Pty.Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne
A,CH,CM,CS,E,MS,P

Perth, Western Australia Office
Hewlett-Packard Australia Pty.Ltd.
141 Stirling Highway
NEDLANDS, W.A. 6009
(effective 28 Sept. 1981:
261 Stirling Highway
CLAREMONT, W.A. 6010)
Tel: 386-5455
Telex: 93859
Cable: HEWPARD Perth
A,CH,CM,,E,MS,P

Sydney, New South Wales Office
Hewlett-Packard Australia Pty.Ltd.
17-23 Talavera Road
P.O. Box 308
NORTH RYDE, N.S.W. 2113
Tel: 887-1611
Telex: 21561
Cable: HEWPARD Sydney
A,CH,CM,CS,E,MS,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Grottenhofstrasse 94
Verkaufsburo Graz
A-8052 **GRAZ**
Tel: 21-5-66
Telex: 32375
CH,CM,E*

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 23-65-11
Telex: 135823/135066
A,CH,CM,CS,E,MS,P

BAHRAIN
Green Salon
P.O. Box 557
BAHRAIN
Tel: 25503-250950
Telex: 84419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 256123
Telex: 8550 WAEL GJ
M

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 ***BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CH,CM,CS,E,MP,P

BRAZIL
Hewlett-Packard do Brasil I.e.C. Ltda.
Alameda Rio Negro, 750
ALPHAVILLE 06400 Barueri SP
Tel: 421-1311
Telex: 011 33872
Cable: HEWPACK Sao Paulo
A,CH,CM,CS,E,MS

Hewlett-Packard do Brasil I.e.C. Ltda.
Avenida Epitacio Pessoa, 4664
22471 **RIO DE JANEIRO**-RJ
Tel: 286-0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CH,CM,E,MS,P*

CANADA

Alberta
Hewlett-Packard (Canada) Ltd.
210, 7220 Fisher Street S.E.
CALGARY, Alberta T2H 2H8
Tel: (403) 253-2713
A,CH,CM,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
11620A-168th Street
EDMONTON, Alberta T5M 3T9
Tel: (403) 452-3670
A,CH,CM,CS,E,MS,P*

British Columbia
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND, British Columbia V6X 2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CH,CM,CS,E*,MS,P*

Manitoba
Hewlett-Packard (Canada) Ltd.
380-550 Century Street
WINNIPEG, Manitoba R3H 0Y1
Tel: (204) 786-6701
A,CH,CM,E,MS,P*

New Brunswick
Hewlett-Packard (Canada) Ltd.
190 Wilbur Street
MONCTON, New Brunswick E2B 2VQ
Tel: (506) 386-1677
Telex: 01931470
CH**

Nova Scotia
Hewlett-Packard (Canada) Ltd.
P.O. Box 931
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
Telex: 01931470
CH,CM,CS,E*,MS,P*

Ontario
Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
Telex: 610-352-1201
A,CH,CM,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246
A,CH,CM,CS,E,MP,P

Hewlett-Packard (Canada) Ltd.
2670 Queensview Dr.
OTTAWA, Ontario K2B 8K1
Tel: (613) 820-6483
Telex: 610-563-1636
A,CH,CM,CS,E*,MS,P*

Quebec
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
KIRKLAND, Quebec H9J 2M5
Tel: (514) 697-4232
Telex: 05821521
A,CH,CM,CS,E,MP,P*

Hewlett-Packard (Canada) Ltd.
Les Galeries du Vallon
2323 Boulevard du Versant Nord
STE. FOY, Quebec G1N 4C2
Tel: (418) 687-4570
CH

CHILE
Jorge Calcagni y Cia. Ltda.
Arturo Burhle 065
Casilla 16475
SANTIAGO 9
Tel: 222-0222
Telex: Public Booth 0001
A,CM,E,M

Olympia (Chile) Ltd.
Rodrico de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 225-5044
Telex: 40565 OLYMP CL
C,P

CHINA, People's Republic of
CEIEC Inc.
44 Beiwei Rd.
***BEIJING**, China*
Telex: 22475 CEIEC CN
A,CH,CM,CS,E,P

COLOMBIA
Instrumentación
H. A. Langebaek & Kier S.A.
Apartado Aéreo 6287
***BOGOTÁ** 1, D.E.*
Carrera 7 No. 48-75
***BOGOTA**, 2 D.E.*
Tel: 287-8877
Telex: 44400
Cable: AARIS Bogota
A,CM,E,M,P

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSE
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR
CM,E,M

CYPRUS
Telerexa Ltd.
P.O. Box 4809
14C Stassinos Avenue
NICOSIA
Tel: 62698
Telex: 2894 Levidocy
E,M,P

CZECHOSLOVAKIA
Hewlett-Packard
Obchodni Zastupitelstvi v CSSR
Post. schranka 27
CS-118 01 ***PRAHA 011***
Tel: 66-296
Telex: 121353 IHC

DENMARK
Hewlett-Packard A/S
Datavej 52
DK-3460 **BIRKEROD**
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CH,CM,CS,E,MS,P

Hewlett-Packard A/S
Navervej 1
DK-8600 **SILKEBORG**
Tel: (06) 82-71-66
Telex: 37409 hpas dk
CH,CM,E

ECUADOR
CYEDE Cia. Ltda.
Avenida Eloy Alfaro 1749
Casilla 6423 CCI
QUITO
Tel: 450-975, 243-052
Telex: 2548 CYEDE ED
A,CM,E,P

Hospitalar S.A.
Robles 625
Casilla 3590
QUITO
Tel: 545-250, 545-122
Telex: 2485 HOSPTL ED
Cable: HOSPITALAR-Quito
M

EGYPT
International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO
Tel: 23-829
Telex: 93830
CH,CS,E,M

Informatic For Systems
22 Talaat Harb Street
CAIRO
Tel: 759006
Telex: 93938 FRANK UN
CH,CS,P

Egyptian International Office for Foreign Trade
P.O.Box 2558
CAIRO
Tel: 984935
Telex: 93337 EGPOR
P

EL SALVADOR
IPESA de El Salvador S.A.
Boulevard de los Heroes 1148
SAN SALVADOR
Tel: 252787, 259621
Telex: Public Booth 20107
A,CH,CM,CS,E,P

FINLAND
Hewlett-Packard Oy
Revontulentie 7
SF-02100 **ESPOO** 10
Tel: (90) 455-0211
Telex: 121563 hewpa sf
A,CH,CM,CS,E,MS,P

SALES & SUPPORT OFFICES
Arranged alphabetically by country

FRANCE
Hewlett-Packard France
Le Ligoures
Bureau de Vente de Aix-en-Provence
Place Romée de Villeneuve
F-13090 **AIX-EN-PROVENCE**
Tel: (42) 59-41-02
Telex: 410770F
A,CH,CM,E,MS,P*

Hewlett-Packard France
Boite Postale No. 503
F-25026 **BESANCON**
28 Rue de la Republique
F-25000 **BESANCON**
Tel: (81) 83-16-22
CH,M

Hewlett-Packard France
Bureau de Vente de Lyon
Chemin des Mouilles
Boite Postale No. 162
F-69130 **ECULLY** Cédex
Tel: (78) 33-81-25
Telex: 310617F
A,CH,CM,CS,E,MP

Hewlett-Packard France
Immeuble France Evry
Tour Lorraine
Boulevard de France
F-91035 **EVRY** Cédex
Tel: (60) 77-96-60
Telex: 692315F
CM,E

Hewlett-Packard France
5th Avenue Raymond Chanas
F-38320 **EYBENS**
Tel: (76) 25-81-41
Telex: 980124 HP GRENOB EYBE
CH,CM

Hewlett-Packard France
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
F-93153 **LE BLANC MESNIL**
Tel: (01) 865-44-52
Telex: 211032F
CH,CM,CS,E,MS

Hewlett-Packard France
Le Montesquieu
Avenue du President JF Kennedy
F-33700 **MERIGNAC**
Tel: (56) 34-00-84
Telex: 550105F
CH,CM,E,MS

Hewlett-Packard France
32 Rue Lothaire
F-57000 **METZ**
Tel: (87) 65-53-50
CH,CM

Hewlett-Packard France
3 Rue Julien Videment
F-44200 **NANTES**
Tel: (40) 48-09-44
CH**

Hewlett-Packard France
Zone Industrielle de Courtaboeuf
Avenue des Tropiques F-91947 Les Ulis Cédex **ORSAY**
Tel: (1) 907-78-25
Telex: 600048F
A,CH,CM,CS,E,MP,P

Hewlett-Packard France
Paris Porte-Maillot 13, 15 25
Boulevard De L'Amiral Bruix
F-75782 **PARIS** Cédex 16
Tel: (01) 502-12-20
Telex: 613663F
CH,CM,MS,P

Hewlett-Packard France
2 Allee de la Bourgonette
F-35100 **RENNES**
Tel: (99) 51-42-44
Telex: 740912F
CH,CM,E,MS,P*

Hewlett-Packard France
98 Avenue de Bretagne
F-76100 **ROUEN**
Tel: (35) 63-57-66 CH**,CS

Hewlett-Packard France
4 Rue Thomas Mann
Boite Postale 56
F-67200 **STRASBOURG**
Tel: (88) 28-56-46
Telex: 890141F
CH,CM,E,MS,P*

Hewlett-Packard France
20 Chemin de la Cépière
F-31081 **TOULOUSE** Cédex
Tel: (61) 40-11-12
Telex: 531639F
A,CH,CM,CS,E,P*

Hewlett-Packard France
Bureau de Vente de Lille
Immeuble Péricentre
Rue Van Gogh
F-59650 **VILLENEUVE D'ASQ**
Tel: (20) 91-41-25
Telex: 160124F
CH,CM,E,MS,P*

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Technisches Büro Berlin
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
A,CH,CM,E,M,P,X

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 **BÖBLINGEN**
Tel: (07031) 667-1
Telex: 07265739 bbn or 07265743
A,CH,CM,CS,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Dusseldorf
Emanuel-Leutze-Strasse 1
D-4000 **DUSSELDORF**
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CH,CM,CS,E,MS,P

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 **FRANKFURT** 56
Tel: (0611) 50-04-1
Telex: 04 13249 hpffm d
A,CH,CM,CS,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Hamburg
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: (040) 63804-1
Telex: 21 63 032 hphh d
A,CH,CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D-3000 **HANNOVER** 91
Tel: (0511) 46-60-01
Telex: 092 3259
A,CH,CM,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Mannheim
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: (621) 70050
Telex: 0462105
A,C,E

Hewlett-Packard GmbH
Technisches Büro Neu Ulm
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel: 0731-70241
Telex: 712816 HP ULM-D
A,C,E*

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D-8500 **NÜRNBERG**
Tel: (0911) 56-30-83
Telex: 0623 860
CH,CM,E,MS,P

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **TAUFKIRCHEN**
Tel: (089) 6117-1
Telex: 0524985
A,CH,CM,E,MS,P

GREAT BRITAIN
Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Chesire WA14 1NU
Tel: (061) 928-6422
Telex: 668068
A,CH,CS,E,M

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton
BRISTOL BS8 2BN
Tel: 36806
Telex: 444302
CH,CM,M,P

Hewlett-Packard Ltd.
14 Wesley Street
CASTLEFORD
Yorkshire WF10 1AE
Tel: (0977) 550016
Telex: 5557355
CH

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY
Herts., AL2 1HA
Tel: (0727) 24400
Telex: 1-8952716
CH,CS,E

Hewlett-Packard Ltd
Tradax House, St. Mary's Walk
MAIDENHEAD
Berkshire, SL6 1ST
Tel: (0628) 39151
CH,CS,E,P

Hewlett-Packard Ltd.
308/314 Kings Road
READING, Berkshire
Tel: 61022
Telex: 84-80-68
CM,P

Hewlett-Packard Ltd.
Quadrangle
106-118 Station Road
REDHILL, Surrey
Tel: (0737) 68655
Telex: 947234 CH,CS,E

Hewlett-Packard Ltd.
Westminster House
190 Stratford Road
SHIRLEY, Solihull
West Midlands B90 3BJ
Tel: (021) 7458800
Telex: 339105
CH

Hewlett-Packard Ltd.
King Street Lane
WINNERSH, Wokingham
Berkshire RG11 5AR
Tel: (0734) 784774
Telex: 847178
A,CS,E,M

GREECE
Kostas Karaynnis S.A.
8 Omirou Street
***ATHENS** 133*
Tel: 32-30-303, 32-37-371
Telex: 21 59 62 RKAR GR
A,CH,CM,CS,E,M,P

PLAISIO S.A.
G. Gerardos
24 Stournara Street
ATHENS
Tel: 36-11-160
Telex: 21 9492
P

GUATEMALA
IPESA
Avenida Reforma 3-48
GUATEMALA 9
Tel: 316627, 314786
Telex: 4192 TELETRO GU
A,CH,CM,CS,E,M,P

HONG KONG
Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 66678 HEWPA HX
Cable: HP ASIA LTD Hong Kong
E,CH,CS,P

Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 28th Floor
Connaught Road, C.
HONG KONG
Tel: 5-455644
Telex: 74766 SCHMX HX
A,M

ICELAND
Elding Trading Company Inc.
Hafnarnvoli-Tryggvagotu
P.O. Box 895
IS-REYKJAVIK
Tel: 1-58-20, 1-63-03
M

INDIA
Blue Star Ltd.
11 Magarath Road
***BANGALORE** 560 025*
Tel: 55668
Telex: 0845-430
Cable: BLUESTAR
A,CH,CM,CS,E

Blue Star Ltd.
Band Box House
Prabhadevi
***BOMBAY** 400 025*
Tel: 422-3101
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
***BOMBAY** 400 025*
Tel: 422-6155
Telex: 011-4093
Cable: FROSTBLUE
A,CH,CM,CS,E,M

Blue Star Ltd.
Kalyan, 19 Vishwas Colony
*Alkapuri, **BORODA**, 390 005*
Tel: 65235
Cable: BLUE STAR
A

Blue Star Ltd.
7 Hare Street
***CALCUTTA** 700 001*
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
133 Kodambakkam High Road
***MADRAS** 600 034*
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
***NEW DELHI** 110 024*
Tel: 682547
Telex: 031-2463
Cable: BLUESTAR
A,CH,CM,CS,E,M

Blue Star Ltd.
15/16:C Wellesley Rd.
***PUNE** 411 011*
Tel: 22775
Cable: BLUE STAR
A

Blue Star Ltd.
1-1-117/1 Sarojini Devi Road
***SECUNDERABAD** 500 033*
Tel: 70126
Telex: 0155-459
Cable: BLUEFROST
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
***TRIVANDRUM** 695 013*
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

INDONESIA
BERCA Indonesia P.T.
P.O.Box 496/JKT.
Jl. Abdul Muis 62
JAKARTA
Tel: 373009
Telex: 46748 BERSAL IA
Cable: BERSAL JAKARTA
P

BERCA Indonesia P.T.
Wisma Antara Bldg., 17th floor
JAKARTA
A,CS,E,M

BERCA Indonesia P.T.
P.O. Box 174/SBY.
Jl. Kulei No. 11
SURABAYA
Tel: 68172
Telex: 31146 BERSAL SB
Cable: BERSAL-SURABAYA
A,E,M,P*

IRAQ
Hewlett-Packard Trading S.A.
Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 2455 HEPAIRAQ IK
CH,CS

IRELAND
Hewlett-Packard Ireland Ltd.
Kestrel House
Clanwilliam Court
Lower Mount Street
DUBLIN 2, Eire
Tel: 680424, 680426
Telex: 30439
A,CH,CM,CS,E,M,P

Cardiac Services Ltd.
Kilmore Road
Artane
***DUBLIN** 5, Eire*
Tel: (01) 351820
Telex: 30439
M

ISRAEL
Electronics Engineering Division
Motorola Israel Ltd.
16 Kremenetski Street
P.O. Box 25016
TEL-AVIV 67899
Tel: 3-338973
Telex: 33569 Motil IL
Cable: BASTEL Tel-Aviv
A,CH,CM,CS,E,M,P

ITALY
Hewlett-Packard Italiana S.p.A.
Traversa 99C
Giulio Petrone, 19
I-70124 **BARI**
Tel: (080) 41-07-44
M

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/111
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CH,CM,E,MS

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291 C,P

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVLIGLIO**
Tel: (2) 903691
Telex: 334632
A,CH,CM,CS,E,MP,P

Hewlett-Packard Italiana S.p.A.
Via Nuova san Rocco A
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
A,CH,CM,E

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07 E,C

Hewlett-Packard Italiana S.p.A.
Via Turazza 14
I-35100 **PADOVA**
Tel: (49) 664888
Telex: 430315
A,CH,CM,E,MS

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA**
Tel: (06) 54831
Telex: 610514
A,CH,CM,CS,E,MS,P*

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **TORINO**
Tel: (011) 682245, 659308
Telex: 221079
CH,CM,E

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1-1348-3, Asahi-cho
ATSUGI, Kanagawa 243
Tel: (0462) 24-0451
CM,C*,E

Yokogawa-Hewlett-Packard Ltd.
Sannomiya-Daiichi Seimei-Bldg. 5F
69 Kyo-machi Chuo-ku
KOBE 650
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi Yasoji Bldg 4F
3-4 Chome Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
CH,CM,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
1-4-73, San-no-maru
MITO, Ibaragi 310
Tel: (0292) 25-7470
CH,CM,E

Yokogawa-Hewlett-Packard Ltd.
Sumitomo Seimei Nagoya Bldg.
11-2 Shimo-sasajima-cho
Nakamura-ku
NAGOYA, Aichi 450
Tel: (052) 571-5171
CH,CM,CS,E,MS

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
5-4-20 Nishinakajima, 5-chome
Yodogawa-ku
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
A,CH,CM,CS,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
3-29-21 Takaido-Higashi 3-chome
Suginami-ku
TOKYO 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
A,CH,CM,CS,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
3-30-4 Tsuruya-cho
Kanagawa-ku,
YOKOHAMA Kanagawa, 221
Tel: (045) 312-1252
CH,CM,E

JORDAN
Mouasher Cousins Company
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
CH,E,M,P

KENYA
ADCOM Ltd., Inc.
City House, Wabera Street
P.O.Box 30635
NAIROBI
Tel: 331955
Telex: 22639
E,M

KOREA
Samsung Electronics
Industrial Products Div.
76-561 Yeoksam-Dong
Kangnam-Ku
C.P.O. Box 2775
SEOUL
Tel: 555-7555, 555-5447
Telex: K27364 SAMSAN
A,CH,CM,CS,E,M,P

KUWAIT
Al-Khalidya Trading & Contracting
P.O. Box 830 Safat
KUWAIT
Tel: 42-4910, 41-1726
Telex: 2481 Areeg kt
CH,E,M

Photo & Cine Equipment
P.O. Box 270 Safat
KUWAIT
Tel: 42-2846, 42-3801
Telex: 2247 Malin
P

LEBANON
G.M. Dolmadjian
Achrafieh
P.O. Box 165.167
BEIRUT
Tel: 290293
MP

LUXEMBOURG
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 ***BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CH,CM,CS,E,MP,P

MALAYSIA
Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
KUALA LUMPUR
Tel: 483544
Telex: MA31011
A,CH,E,M,P*

Protel Engineering
Lot 319, Salok Road
P.O.Box 1917
*Kuching, **SARAWAK***
Tel: 53544
Telex: MA 70904 PROMAL
Cable: PROTELENG
A,E,M

MALTA
Philip Toledo Ltd.
Nalabile Rd.
MRIEHEL
Tel: 447 47, 455 66
Telex: MW.649
P

MEXICO
Hewlett-Packard Mexicana, S.A. de C.V.
Avenida Periferico Sur No. 6501
Tepepan, Xochimilco
MEXICO CITY 23, D.F.
Tel: (905) 676-4600
Telex: 017-74-507
A,CH,CS,E,MS,P

Hewlett-Packard Mexicana, S.A. de C.V.
Rio Volga 600
Colonia del Valle
MONTERREY, N.L.
Tel: 78-42-93, 78-42-40, 78-42-41
Telex: 038-410
CH

MOROCCO
Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

NETHERLANDS
Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL 1181KK **AMSTELVEEN**
P.O. Box 667
NL1080 AR **AMSTELVEEN**
Tel: (20) 47-20-21
Telex: 13 216
A,CH,CM,CS,E,MP,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL 2906VK **CAPPELLE**, A/D Ijessel
P.O. Box 41
NL2900 AA **CAPELLE**, Ijssel
Tel: (10) 51-64-44
Telex: 21261 HEPAC NL
A,CH,CM,CS

Koning en Hartman Electrotechnick B.V.
Koperwerf 30
2544 En den Haag
The **NETHERLANDS**
Tel: 070-210101
Telex: 31528
CM

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
169 Manukau Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 687-159
Cable: HEWPACK Auckland
CH,CM,E,P*

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
Kilbirnie, **WELLINGTON 3**
P.O. Box 9443
Courtenay Place, WELLINGTON
Tel: 877-199
Cable: HEWPACK Wellington
CH,CM,E,P

Northrop Instruments & Systems Ltd.
369 Khyber Pass Road
P.O. Box 8602
AUCKLAND
Tel: 794-091
Telex: 60605
A,M

Northrop Instruments & Systems Ltd.
110 Mandeville St.
P.O. Box 8388
CHRISTCHURCH
Tel: 486-928
Telex: 4203
A,M

Northrop Instruments & Systems Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 3380
A,M

NORTHERN IRELAND
Cardiac Services Company
95A Finaghy Road South
***BELFAST** BT 10 OBY*
Tel: (0232) 625-566
Telex: 747626
M

NORWAY
Hewlett-Packard Norge A/S
Folke Bernadottesvei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN** (BERGEN)
Tel: (05) 16-55-40
Telex: 16621 hpnas n
CH,CM,E

Hewlett-Packard Norge A/S
Oesterndalen 18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: (02) 17-11-80
Telex: 16621 hpnas n
A*,CH,CM,E,MS,P

OMAN
Khimji Ramdas
P.O. Box 19
MUSCAT
Tel: 72-22-17, 72-22-25
Telex: 3289 BROKER MB MUSCAT
P

PAKISTAN
Mushko & Company Ltd.
10, Bazar Road
Sector G-6/4
ISLAMABAD
Tel: 26875
Cable: FEMUS Rawalpindi
A,E,M

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
***KARACHI** 0302*
Tel: 511027, 512927
Telex: 2894 MUSKO PK
Cable: COOPERATOR Karachi
*A,E,M,P**

PANAMA
Electrónico Balboa, S.A.
Calle Samuel Lewis
Aparlado 4929
Panama 5
Edificio "Alfa" No. 2
CIUDAD DE PANAMA
Tel: 64-2700
Telex: 0383 ELECTRON PG
A,CM,E,M,P

SALES & SUPPORT OFFICES
Arranged alphabetically by country

PANAMA (Con't.)
Foto Internacional, S.A.
Free Zone Colon
Apartado 2068
COLON 3
Tel: 45-2333
Telex: 379 8626, 386 8722
P

PERU
Cía Electro Médica S.A.
Los Flamencos 145, San Isidro
Casilla 1030
LIMA 1
Tel: 41-4325, 41-3703
Telex: Pub. Booth 25306
A,CM,E,M,P

PHILIPPINES
The Online Advanced Systems Corporation
Rico House, Amorsolo Cor. Herrera Street
Legaspi Village, Makati
P.O. Box 1510
Metro ***MANILA***
Tel: 85-35-81, 85-34-91, 85-32-21
Telex: 3274 ONLINE
A,CH,CS,E,M

Electronic Specialists and Proponents Inc.
690-B Epifanio de los Santos Avenue
Cubao, ***QUEZON CITY***
P.O. Box 2649 Manila
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 40018, 42000 ITT GLOBE MACKAY BOOTH
P

POLAND
Buro Informasji Technicznej
Hewlett-Packard
Ul Slawki 2, 6P
PL00-950 ***WARSZAWA***
Tel: 39-59-62, 39-67-43
Telex: 812453 hepa pl

PORTUGAL
SOQUIMICA
Av. da Libedade 220-2
P-1298 ***LISBOA Codex***
Tel: 56 21 81, 56 21 82
Telex: 13316

Telectra-Empresa Técnica de Equipmentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-LISBON 1
Tel: (19) 68-60-72
Telex: 12598
CH,CS,E,P

Mundinter
Intercambio Mundial de Comércio S.a.r.l
P.O. Box 2761
Avenida Antonio Augusto de Aguiar 138
P-LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

PUERTO RICO
Hewlett-Packard Puerto Rico
P.O. Box 4407
CAROLINA, Puerto Rico 00630
Calle 272 Edificio 203
Urb. Country Club
RIO PIEDRAS, Puerto Rico 00924
Tel: (809) 762-7255
Telex: 345 0514
A,CH,CS

QATAR
Nasser Trading & Contracting
P.O. Box 1563
DOHA
Tel: 22170
Telex: 4439 NASSER
M

Computearabia
P.O. Box 2570
DOHA
Tel: 329515
Telex: 4806 CHPARB
P

ROMANIA
Hewlett-Packard Reprezentanta
Boulevard Nicolae Balcescu 16
BUCURESTI
Tel: 130725
Telex: 10440

SAUDI ARABIA
Modern Electronic Establishment
P.O. Box 193
AL-KHOBAR
Tel: 44-678, 44-813
Telex: 670136
Cable: ELECTA AL-KHOBAR
CH,CS,E,M,P

Modern Electronic Establishment
P.O. Box 1228, Baghdadiah Street
JEDDAH
Tel: 27-798
Telex: 401035
Cable: ELECTA JEDDAH
CH,CS,E,M,P

Modern Electronic Establishment
P.O. Box 2728
RIYADH
Tel: 62-596, 66-232
Telex: 202049
CH,CS,E,M,P

SCOTLAND
Hewlett-Packard Ltd.
Royal Bank Buildings
Swan Street
BRECHIN, Angus, Scotland
Tel: 3101, 3102
CH,CM

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9GT
GB-Scotland
Tel: (031) 3311000
Telex: 72682
A,CH,CM,CS,E,M

SINGAPORE
Hewlett-Packard Singapore (Pty.) Ltd.
P.O. Box 58 Alexandra Post Office
SINGAPORE, 9115
6th Floor, Inchcape House
450-452 Alexandra Road
SINGAPORE 0511
Tel: 631788
Telex: HPSGSO RS 34209
Cable: HEWPACK, Singapore
A,CH,CS,E,MS,P

SOUTH AFRICA
Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 120
Howard Place
Pine Park Center, Forest Drive, Pinelands
CAPE PROVINCE 7450
Tel: 53-7955, 53-7956, 53-7957
Telex: 57-0006
A,CH,CM,E,MS,P

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 37099
Overport
DURBAN 4067
Tel: 28-4178, 28-4179, 28-4110
CH,CM

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 33345
Glenstantia 0010 **TRANSVAAL**
1st Floor East
Constantia Park Ridge Shopping Centre
Constantia Park
PRETORIA
Tel: 01298-1126
Telex: 32163
CH,E

Hewlett-Packard South Africa (Pty.) Ltd.
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111, 802-5125
Telex: 89-84782
Cable: HEWPACK Johannesburg
A,CH,CM,CS,E,MS,P

SPAIN
Hewlett-Packard Española S.A.
c/Entenza, 321
E-BARCELONA 29
Tel: (3) 322-24-51, 321-73-54
Telex: 52603 hpbee
A,CH,CM,CS,E,MS,P

Hewlett-Packard Española S.A.
c/San Vicente S/N
Edificio Albia II,7 B
E-BILBAO 1
Tel: (944) 423-8306, 423-8206
A,CH,CM,E,MS

Hewlett-Packard Española S.A.
Calle Jerez 3
E-MADRID 16
Tel: 458-2600
Telex: 23515 hpe
A,CM,E

Hewlett-Packard Española S.A.
c/o Costa Brava 13
Colonia Mirasierra
E-MADRID 34
Tel: 734-8061, 734-1162
CH,CS,M

Hewlett-Packard Española S.A.
Av Ramón y Cajal 1-9
Edificio Sevilla 1,
E-SEVILLA 5
Tel: 64-44-54, 64-44-58
Telex: 72933
A,CM,CS,MS,P

Hewlett-Packard Española S.A.
C/Ramon Gordillo, 1 (Entlo.3)
E-VALENCIA 10
Tel: 361-1354, 361-1358
CH,CM,P

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
P.O. Box 20502
S-16120 **BROMMA**
Tel: (08) 730-0550
Telex: (854) 10721 MESSAGES
Cable: MEASUREMENTS STOCKHOLM
A,CH,CM,CS,E,MS,P

Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 **LUND**
Tel: (46) 13-69-79
Telex: (854) 10721 (via BROMMA office)
CH,CM

Hewlett-Packard Sverige AB
Vastra Vintergatan 9
S-70344 **OREBRO**
Tel: (19) 10-48-80
Telex: (854) 10721 (via BROMMA office)
CH,CM

Hewlett-Packard Sverige AB
Frötallisgatan 30
S-42132 **VÄSTRA-FRÖLUNDA**
Tel: (031) 49-09-50
Telex: (854) 10721 (via BROMMA office)
CH,CM,E,P

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASLE**
Tel: (61) 33-59-20
A,CM

Hewlett-Packard (Schweiz) AG
Bahnhoherweg 44
3018 **BERN**
Tel: (031) 56-24-22
CH,CM

Hewlett-Packard (Schweiz) AG
47 Avenue Blanc
CH-1202 **GENEVA**
Tel: (022) 32-30-05, 32-48-00
CH,CM,CS

Hewlett-Packard (Schweiz) AG
29 Chemin Château Bloc
CH-1219 **LE LIGNON**-Geneva
Tel: (022) 96-03-22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva
A,CM,E,MS,P

Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
Allmend 2
CH-8967 **WIDEN**
Tel: (57) 50-111
Telex: 59933 hpag ch
Cable: HPAG CH
A,CH,CM,CS,E,MS,P

SYRIA
General Electronic Inc.
Nuri Basha-Ahnat Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS
E

Middle East Electronics
Place Azmé
Boite Postale 2308
DAMASCUS
Tel: 334592
Telex: 11304 SATACO SY
M,P

TAIWAN
Hewlett-Packard Far East Ltd.
Kaohsiung Office
2/F 68-2, Chung Cheng 3rd Road
KAOHSIUNG
Tel: 241-2318, 261-3253
E,MS,P

Hewlett-Packard Far East Ltd.
Taichung Office
#33, Cheng Tr Rd.
10th Floor, Room 5
TAICHUNG
Tel: (042) 289274

Hewlett-Packard Far East Ltd.
Taiwan Office
5th Floor
205 Tun Hwa North Road
TAIPEI
Tel:(02) 751-0404
Cable:HEWPACK Taipei
A,CH,CS,E,MS,P

Ing Lih Trading Co.
3rd Floor 18, Po-Al Road
TAIPEI (100)
Tel: (02) 311-1914
Cable: INGLIH TAIPEI
A

THAILAND
Unimesa
30 Patpong Ave., Suriwong
BANGKOK 5
Tel: (234-091) (234-092)
Telex: TH 81160, TH 81038
Cable: UNIMESA Bangkok
A,C,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671
Telex: 87669-BEQUIPT TH
Cable: BUSIQUIPT Bangkok
P

TRINIDAD & TOBAGO
Caribbean Telecoms Ltd.
50/A Jerningham Avenue
P.O. Box 732
PORT-OF-SPAIN
Tel: 624-4213, 624-4214
Telex: 3235, 3272 HUGCO WG
A,CM,E,M,P

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
E,P

Corema
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY
Teknim Company Ltd.
Iran Caddesi No. 7
Kavaklidere, ***ANKARA***
Tel: 275800
Telex: 42155
E

E.M.A.
Medina Eldem Sokak No.41/6
Yuksel Caddesi
ANKARA
Tel: 175 622
M,P

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 354121, 354123
Telex: 68136 Emitac SL
CH,CS,E,M,P

UNITED KINGDOM
see: GREAT BRITAIN
NORTHERN IRELAND
SCOTLAND

UNITED STATES

Alabama
Hewlett-Packard Co.
700 Century Park South
Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
CH,CM,MP

Hewlett-Packard Co.
P.O. Box 4207
8290 Whitesburg Drive, S.E.
HUNTSVILLE, AL 35802
Tel: (205) 881-4591
CH,CM,CS,E,M*

Alaska
Hewlett-Packard Co.
1577 "C" Street, Suite 252
ANCHORAGE, AK 99510
Tel: (206) 454-3971
CH*,CM

Arizona
Hewlett-Packard Co.
2336 East Magnolia Street
PHOENIX, AZ 85034
Tel: (602) 273-8000
A,CH,CM,CS,E,MS

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85702
Tel: (602) 889-4631
CH,CM,E,MS**

Arkansas
Hewlett-Packard Co.
P.O. Box 5646
Brady Station
LITTLE ROCK, AR 72215
Tel: (501) 376-1844, (501) 664-8773
CM,MS

California
Hewlett-Packard Co.
99 South Hill Dr.
BRISBANE, CA 94005
Tel: (415) 330-2500
CH,CM,CS

Hewlett-Packard Co.
7621 Canoga Avenue
CANOGA PARK, CA 91304
Tel: (213) 702-8300
A,CH,CM,CS,E,P

Hewlett-Packard Co.
1579 W. Shaw Avenue
FRESNO, CA 93771
Tel: (209) 224-0582
CM,MS

Hewlett-Packard Co.
1430 East Orangethorpe
FULLERTON, CA 92631
Tel: (714) 870-1000
CH,CM,CS,E,MP

Hewlett-Packard Co.
5400 W. Rosecrans Boulevard
LAWNDALE, CA 90260
P.O. Box 92105
LOS ANGELES, CA 90009
Tel: (213) 970-7500
CH,CM,CS,MP

Hewlett-Packard Co.
3939 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282
Regional Headquarters

Hewlett-Packard Co.
3200 Hillview Avenue
PALO ALTO, CA 94304
Tel: (415) 857-8000
CH,CM,CS,E

Hewlett-Packard Co.
646 W. North Market Boulevard
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,CH,CM,CS,E,MS

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333 **SAN DIEGO**, CA 92123
Tel: (714) 279-3200
CH,CM,CS,E,MP

Hewlett-Packard Co.
Suite A
5553 Hollister
SANTA BARBARA, CA 93111
Tel: (805) 964-3390

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
A,CH,CM,CS,E,MP

Colorado
Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 771-3455
A,CH,CM,CS,E,MS

Connecticut
Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CH,CM,CS,E,MS

Florida
Hewlett-Packard Co.
P.O. Box 24210
2727 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
CH,CM,CS,E,MP

Hewlett-Packard Co.
4080 Woodcock Drive, #132
Brownett Building
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
CM,C*,E*,MS**

Hewlett-Packard Co.
P.O. Box 13910
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,CH,CM,CS,E,MS

Hewlett-Packard Co.
6425 N. Pensacola Blvd.
Suite 4, Building 1
PENSACOLA, FL 32575
Tel: (904) 476-8422
A,CM,MS

Hewlett-Packard Co.
110 South Hoover, Suite 120
Vanguard Bldg.
TAMPA, FL 33609
Tel: (813) 872-0900
A*,CH,CM,CS,E*,M*

Georgia
Hewlett-Packard Co.
P.O. Box 105005
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,CH,CM,CS,E,MP

Hewlett-Packard Co.
Executive Park Suite 306
P.O. Box 816
AUGUSTA, GA 30907
Tel: (404) 736-0592
CM,MS

Hewlett-Packard Co.
P.O. Box 2103
1172 N. Davis Drive
WARNER ROBINS, GA 31098
Tel: (912) 923-8831
CM,E

Hawaii
Hewlett-Packard Co.
Kawaiahao Plaza, Suite 190
567 South King Street
HONOLULU, HI 96813
Tel: (808) 526-1555
A,CH,CM,E,MS

Idaho
Hewlett-Packard Co.
11311 Chinden Boulevard
BOISE, ID 83707
Tel: (208) 376-6000
CH,CM,M*

Illinois
Hewlett-Packard Co.
211 Prospect Road
BLOOMINGTON, IL 61701
Tel: (309) 663-0383
CH,CM,MS**

Hewlett-Packard Co.
1100 31st Street
DOWNERS GROVE, IL 60515
Tel: (312) 960-5760
CH,CM,CS

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
A,CH,CM,CS,E,MP

Indiana
Hewlett-Packard Co.
P.O. Box 50807
7301 No. Shadeland Avenue
INDIANAPOLIS, IN 46250
Tel: (317) 842-1000
A,CH,CM,CS,E,MS

Iowa
Hewlett-Packard Co.
5815 S.W. 5th Street
DES MOINES, IA 50315
Tel: (515) 243-5876
CH,CM,MS**

Hewlett-Packard Co.
2415 Heinz Road
IOWA CITY, IA 52240
Tel: (319) 351-1020
CH,CM,E*,MS

Kansas
Hewlett-Packard Co.
1644 S. Rock
WICHITA, KA 67207
Tel: (316) 265-5200
CH,CM

Kentucky
Hewlett-Packard Co.
10170 Linn Station Road
Suite 525
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CH,CM,CS,MS

Louisiana
Hewlett-Packard Co.
P.O. Box 1449
3229 Williams Boulevard
KENNER, LA 70062
Tel: (504) 443-6201
A,CH,CM,CS,E,MS

Maryland
Hewlett-Packard Co.
7121 Standard Drive
HANOVER, MD 21076
Tel: (301) 796-7700
A,CH,CM,CS,E,MS

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
Telex: 710-828-9685
A,CH,CM,CS,E,MP

Massachusetts
Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
A,CH,CM,CS,E,MP

Michigan
Hewlett-Packard Co.
23855 Research Drive
FARMINGTON HILLS, MI 48024
Tel: (313) 476-6400
A,CH,CM,CS,E,MP

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CH,CM,CS,MS

Minnesota
Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CH,CM,CS,E,MP

Mississippi
Hewlett-Packard Co.
P.O. Box 5028
322 N. Mart Plaza
JACKSON, MS 39216
Tel: (601) 982-9363
CM,MS

Missouri
Hewlett-Packard Co.
11131 Colorado Avenue
KANSAS CITY, MO 64137
Tel: (816) 763-8000
Telex: 910-771-2087
A,CH,CM,CS,E,MS

Hewlett-Packard Co.
1024 Executive Parkway
ST. LOUIS, MO 63141
Tel: (314) 878-0200
A,CH,CM,CS,E,MP

Nebraska
Hewlett-Packard
7101 Mercy Road
Suite 101, IBX Building
OMAHA, NE 68106
Tel: (402) 392-0948
CM,MS

Nevada
Hewlett-Packard Co.
Suite D-130
5030 Paradise Blvd.
LAS VEGAS, NV 89119
Tel: (702) 736-6610
CM,MS**

New Jersey
Hewlett-Packard Co.
Crystal Brook Professional Building
Route 35
EATONTOWN, NJ 07724
Tel: (201) 542-1384
A*,CM,C*,E*,P*

Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CH,CM,CS,E,MP

Hewlett-Packard Co.
60 New England Av. West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CH,CM,CS,E

New Mexico
Hewlett-Packard Co.
P.O. Box 11634
11300 Lomas Blvd.,N.E.
ALBUQUERQUE, NM 87123
Tel: (505) 292-1330
Telex: 910-989-1185
CH,CM,CS,E,MS

New York
Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
Telex: 710-444-4691
A,CH,CM,E,MS

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
Telex: 710-523-1893
CH

Hewlett-Packard Co.
200 Cross Keys Office
FAIRPORT, NY 14450
Tel: (716) 223-9950
Telex: 510-253-0092
CH,CM,CS,E,MS

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
NEW YORK, NY 10119
Tel: (212) 971-0800
CH,CM,CS,E*,M*

Hewlett-Packard Co.
5858 East Molloy Road
SYRACUSE NY 13211
Tel: (315) 455-2486
A,CH,CM,E,MS

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
Telex: 510-221-2183
A,CH,CM,CS,E,MS

North Carolina
Hewlett-Packard Co.
P.O. Box 15579
2905 Guess Road (27705)
DURHAM, NC 27704
Tel: (919) 471-8466
C,M

Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27409
Tel: (919) 852-1800
A,CH,CM,CS,E,MS

Ohio
Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CH,CM,CS,MS

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
Telex: 810-423-9430
A,CH,CM,CS,E,MS

Hewlett-Packard Co.
962 Crupper Ave.
COLUMBUS, OH 43229
Tel: (614) 436-1041
CH,CM,CS,E*

SALES & SUPPORT OFFICES
Arranged alphabetically by country

Ohio (Con't)
Hewlett-Packard Co.
330 Progress Rd.
DAYTON, OH 45449
Tel: (513) 859-8202
A,CH,CM,E*,MS

Oklahoma
Hewlett-Packard Co.
P.O. Box 366
1503 W. Gore Blvd., Suite #2
LAWTON, OK 73502
Tel: (405) 248-4248
C
Hewlett-Packard Co.
P.O. Box 32008
304 N. Meridan Avenue, Suite A
OKLAHOMA CITY, OK 73107
Tel: (405) 946-9499
A*,CH,CM,E*,MS
Hewlett-Packard Co.
Suite 121
9920 E. 42nd Street
TULSA, OK 74145
Tel: (918) 665-3300
A**,CH,CM,CS,M*

Oregon
Hewlett-Packard Co.
1500 Valley River Drive, Suite 330
EUGENE, OR 97401
Tel: (503) 683-8075
C
Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CH,CM,CS,E*,MS

Pennsylvania
Hewlett-Packard Co.
1021 8th Avenue
King of Prussia Industrial Park
KING OF PRUSSIA, PA 19406
Tel: (215) 265-7000
Telex: 510-660-2670
A,CH,CM,CS,E,MP
Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CH,CM,CS,E,MP

South Carolina
Hewlett-Packard Co.
P.O. Box 6442
6941-0 N. Trenholm Road
COLUMBIA, SC 29260
Tel: (803) 782-6493
CH,CM,E,MS
Hewlett-Packard Co.
814 Wade Hampton Blvd.
Suite 10
GREENVILLE, SC 29609
Tel: (803) 232-0917
C

Tennessee
Hewlett-Packard Co.
P.O. Box 22490
224 Peters Road
Suite 102
KNOXVILLE, TN 37922
Tel: (615) 691-2371
A*,CH,CM,MS
Hewlett-Packard Co.
3070 Directors Row
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,CH,CM,MS
Hewlett-Packard Co.
Suite 103
478 Craighead Street
NASHVILLE, TN 37204
Tel: (615) 383-9136
CM,MS**

Texas
Hewlett-Packard Co.
Suite 310W
7800 Shoalcreek Blvd.
AUSTIN, TX 78757
Tel: (512) 459-3143
CM,E
Hewlett-Packard Co.
Suite C-110
4171 North Mesa
EL PASO, TX 79902
Tel: (915) 533-3555
CH,CM,E*,MS**
Hewlett-Packard Co.
5020 Mark IV Parkway
FORT WORTH, TX 76106
Tel: (817) 625-6361
CM,C*
Hewlett-Packard Co.
P.O. Box 42816
10535 Harwin Street
HOUSTON, TX 77036
Tel: (713) 776-6400
A,CH,CM,CS,E,MP
Hewlett-Packard Co.
3309 67th Street
Suite 24
LUBBOCK, TX 79413
Tel: (806) 799-4472
M
Hewlett-Packard Co.
P.O. Box 1270
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,CH,CM,CS,E,MP
Hewlett-Packard Co.
205 Billy Mitchell Road
SAN ANTONIO, TX 78226
Tel: (512) 434-8241
CH,CM,CS,E,MS

Utah
Hewlett-Packard Co.
3530 W. 2100 South Street
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,CH,CM,CS,E,MS

Virginia
Hewlett-Packard Co.
P.O. Box 9669
2914 Hungary Spring Road
RICHMOND, VA 23228
Tel: (804) 285-3431
A,CH,CM,CS,E,MS
Hewlett-Packard Co.
P.O. Box 4786
3110 Peters Creek Road, N.W.
ROANOKE, VA 24015
Tel: (703) 563-2205
CH,CM,E**
Hewlett-Packard Co.
P.O. Box 12778
5700 Thurston Avenue
Suite 111
VIRGINIA BEACH, VA 23455
Tel: (804) 460-2471
CH,CM,MS

Washington
Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4000
A,CH,CM,CS,E,MP
Hewlett-Packard Co.
Suite A
708 North Argonne Road
SPOKANE, WA 99206
Tel: (509) 922-7000
CH,CM,CS

West Virginia
Hewlett-Packard Co.
4604 MacCorkle Ave., S.E.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,CM,MS

Wisconsin
Hewlett-Packard Co.
150 S. Sunny Slope Road
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CH,CM,CS,E*,MP

URUGUAY
Pablo Ferrando S.A.C. e.I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 80-2586
Telex: Public Booth 901
A,CM,E,M
Guillermo Kraft del Uruguay S.A.
Av. Lib. Brig. Gral. Lavalleja 2083
MONTEVIDEO
Tel: 234588, 234808, 208830
Telex: 6245 ACTOUR UY
P

U.S.S.R.
Hewlett-Packard Co.
Representative Office
Pokrovsky Blvd. 4/17 KV12
MOSCOW 101000 Tel: 294-2024
Telex: 7825 HEWPACK SU

VENEZUELA
Hewlett-Packard de Venezuela C.A.
Apartado 50933
3A Transversal Los Ruices Norte
Edificio Segre
CARACAS 1071
Tel: 239-4133
Telex: 25146 HEWPACK
A,CH,CS,E,MS,P

YUGOSLAVIA
Iskra-Commerce-Representation of Hewlett-Packard
Sava Centar Delegacija 30
Milentija Popovica 9
11170 ***BEOGRAD***
Tel: 638-762
Telex: 12042, 12322 YU SAV CEN
Iskra-Commerce-Representation of Hewlett-Packard
Koprska 46
61000 ***LJUBLJANA***
Tel: 321674, 315879
Telex:

ZAIRE
Computer & Industrial Engineering
25 Avenue de la Justice
B.P. 10-976
Kinshasha I/Zaire
GOMBE
Tel: 32063
Telex: 21-457 SGEKIN ZR
CH,CS

ZIMBABWE
Field Technical Sales
45 Kelvin Road, North
P.B. 3458
SALISBURY
Tel: 705 231
Telex: 4-122 RH
C,E,M,P

FOR COUNTRIES AND AREAS NOT LISTED:

CANADA
Ontario
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246

EASTERN USA
Maryland
Hewlett-Packard Co.
4 Choke Cherry Road
Rockville, MD 20850
Tel: (301) 258-2000

MIDWESTERN USA
Illinois
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA
Georgia
Hewlett-Packard Co.
P.O. Box 105005
450 Interstate N. Parkway
ATLANTA, GA 30339
Tel: (404) 955-1500

WESTERN USA
California
Hewlett-Packard Co.
3939 Lankersim Blvd.
LOS ANGELES, CA 91604
Tel: (213) 877-1282

EUROPEAN AREAS NOT LISTED, CONTACT SWITZERLAND
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 **MEYRIN 2**, Switzerland
Tel: (022) 83-81-11
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

EAST EUROPEAN AREAS NOT LISTED, CONTACT AUSTRIA
Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066

MEDITERRANEAN AND MIDDLE EAST AREAS NOT LISTED, CONTACT GREECE
Hewlett-Packard S.A.
Mediterranean & Middle East Operations
32 Kifisias Ave.
Atrina Center
PARADISOS, Amaroussion
Tel: 808-1741-4
Telex: 21-6588 HPAT GR
Cable: HEWPACKSA Athens

INTERNATIONAL AREAS NOT LISTED, CONTACT OTHER AREAS
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK

FOR COUNTRIES AND AREAS NOT LISTED, CONTACT:

AFRICA

NORTHERN AND CENTRAL AFRICA

SWITZERLAND
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 **MEYRIN 2**, Switzerland
Tel: (022) 98-96-51
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

ASIA

HONG KONG
Hewlett-Packard Asia Ltd.
6th Floor, Sun Hung Kai Center
30 Harbor Rd.
G.P.O. Box 795
HONG KONG
Tel: 5-832 3211
Telex: 66678 HEWPA HX
Cable: HP ASIA LTD Hong Kong

EUROPE

EASTERN EUROPE

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O.Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066

NORTHERN EUROPE

THE NETHERLANDS
Hewlett-Packard S.A.
Uilenstede 475
NL-1183 AG **AMSTELVEEN**, The Netherlands
P.O.Box 999
NL-1180 AZ **AMSTELVEEN**, The Netherlands
Tel: 20 437771

SOUTH EASTERN EUROPE

SWITZERLAND
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 **MEYRIN 2**, Switzerland
Tel: (022) 98-96-51
Telex: 27835 hpse
Cable: HEWPACKSA Geneve
(Offices in the World Trade Center)

MEDITERRANEAN AND MIDDLE EAST

GREECE
Hewlett-Packard S.A.
Mediterranean and Middle East Operations
Atrina Centre
32 Kifissias Ave.
Amaroussion, **ATHENS**, Greece
Tel: 808-0359 808-0429
Telex: 21-6588
Cable: HEWPACKSA Athens

OTHER INTERNATIONAL AREAS
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK
August 1981 5952-6900

HEWLETT PACKARD

Personal Computer Division
1010 N.E. Circle Blvd., Corvallis, OR 97330 U.S.A.

Reorder Number
00087-90017

Printed in U.S.A. 12

00087