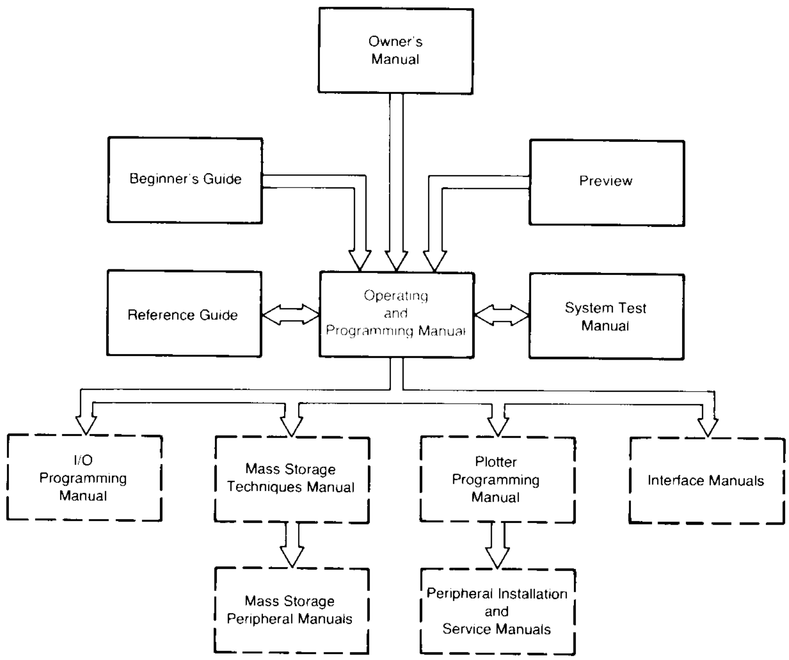
Operating and Programming

HP System 35 Desktop Computer

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525
(For World-wide Sales and Service Offices see back of manual.)

System 35 Manual Reference

The following block diagram shows manuals that are included in the System 35 Documentation scheme and suggested progression. Dotted-line borders indicate those manuals available with specific options; solid borders indicate those manuals that are shipped with every System 35.



Operating and Programming (09835-90000) for all users. All features of the computer and the language are explained.

Beginner's Guide (09835-90001) for the novice programmer. Covers the fundamentals of programming and the BASIC language. The beginner can then progress to the Operating and Programming Manual.

Preview (09835-90002) for the experienced BASIC programmer. A brief "demonstration" of the System 35 to introduce the hardware and extensions of the BASIC language.

Owner's Manual (09835-90005) for System 35 owners. Covers initial set-up, cleaning the computer, ROMs, the tape cartridge, peripherals, and interfaces.

Reference Guide (09835-90010) for all users. A reference to general machine features and all language syntax.

System Test (09835-90040) refer to this manual to test your computer or if there is any doubt that it is operating properly.

Preface

This manual is designed to be used by a wide spectrum of 9835A/B users – from those who are familiar with some type of programming to those who have programmed extensively using the BASIC language.

In general, the various BASIC language components (statements, functions, etc.) are grouped topically. For example, all statements and functions related to output are covered in the same chapter. The topics are arranged logically; you could read the manual straight through if you wanted to. As much as possible, major topics are self-contained; you don't need to read an entire chapter to extract one idea. In some instances, however, statements which haven't been introduced are used to help illustrate the topic being discussed; the PRINT statement is used frequently in this way. It is a good idea to read the first two chapters to get acquainted with the special features of the computer before going on to other topics.

The coverage of each statement, etc., is restricted to its syntax, rules for its usage and some reasons for using it (covered either in text or in an example).

The example programs are not intended to be comprehensive, but to illustrate syntax and example usage. In some cases, dots are used to highlight lines in the example programs. Due to the volume and complexity of the 9835A/B's capabilities, it is not feasible to delve into all of the uses and possibilities of programming and still produce an easy-to-use and easy-to-handle manual.

In limiting the depth of coverage in this manual, a certain amount of previous knowledge on the part of the reader must be assumed. It is assumed that you know how to program. Thus, programming is not taught. Those who don't should turn first to the Beginner's Guide which is supplied with the 9835A/B. The Beginner's Guide covers the fundamentals of programming and the BASIC language. Those of you who know how to program, but not using the BASIC language, may also want to refer to the Beginner's Guide for information about BASIC. The programmers who know BASIC may want to turn to the Preview supplied with the 9835A/B to find the hardware-oriented information which will allow you to write and run programs on the 9835A/B. The Reference Guide can be used by all users as a reference to the language.

Manual Summary

The summary below separates the tabbed information by chapter.

1

Chapter 1: General Information

Introduces you to features of the 9835A/B.

2

Chapter 2: Keyboard Operations

Describes most operations which don't involve programming, including editing, special function keys and operating modes.

3

Chapter 3: Mathematics

Describes basic mathematics including operators and functions and introduces variables.

4

Chapter 4: Programming Information

Describes programming fundamentals, program control operations and miscellaneous statements.

5

Chapter 5: Using Variables

Describes all types of variables, how to dimension them and how to assign values to them.

6

Chapter 6: String Operations

Describes string operations and functions.

7

Chapter 7: Array Operations

Describes all array manipulation statements and functions.

8

Chapter 8: Branching and Subroutines

Covers all statements for branching and subroutines including branching using special function keys.

9

Chapter 9: Subprograms

Describes the concepts and fundamentals of using subprograms.

Table of Contents

To find references to a specific item, please refer to the index in the back of the manual.

Chapter 1
General Information

Introduction

The HP 9835A/B Desktop Computer is a high speed, versatile computational and controlling tool which you can use to perform calculations or program using the BASIC language. Internal Read/Write memory can be expanded up to 256K bytes. A tape cartridge drive allows you to store and retrieve programs and data. Your 9835A/B has either a CRT[1] (9835A) or a single-line LED[2] display (9835B) for viewing information. Optionally, it also can have a 16-character thermal strip printer for hard-copy output.

This chapter introduces some of the physical and operating characteristics of your HP 9835A/B Desktop Computer. The keyboard, CRT or display, and memory are a few of the subjects which are covered.

Installation

Information concerning initial set up, turn on, options and accessories can be found in the 9835A/B Owner's Manual, HP P/N 09835-90005.

1 Cathode Ray Tube
2 Light Emitting Diode

1

Getting Started

There are several things you should check each time you use your 9835A/B –

- If the computer is turned off, set the power switch (found on the right-hand side of the machine) to the "1" position.

  When one of the following displays appears, (allowing up to 30 seconds for memory test and CRT warm-up), the computer is ready to use –

  9835A READY FOR USE.

  or

  9835B READY FOR USE.

  Memory failure at power on is covered in the Owner's Manual

- If the computer is switched on, but the CRT or display is blank, hold down CONT'D, then press STOP. This is known as the reset operation (see Chapter 2). You can also adjust the intensity knob located on the lower right-hand side of the CRT.

  If the display still remains blank, first check the power connection and the fuse, as described in the Owner's Manual. For further assistance, call the nearest HP Sales and Service Office; locations are listed in Appendix E of this manual.

- **Automatic Start**

  The 9835A/B has an automatic start capability. If a tape cartridge is present in the tape drive at power-on, the computer automatically executes the following operation –

  LOAD "AUTOST", 1

  when the power is switched on.

  The autostart routine permits the computer to load and run a supervisory program automatically, which in turn could define special function keys or load other programs without operator instructions. The autostart routine is also performed after a power failure (if a tape cartridge with a file named AUTOST is present in the tape drive), enabling the computer to reload and restart a program automatically. See the LOAD statement (Chapter 11) for more information concerning the LOAD operation.

The Keyboard

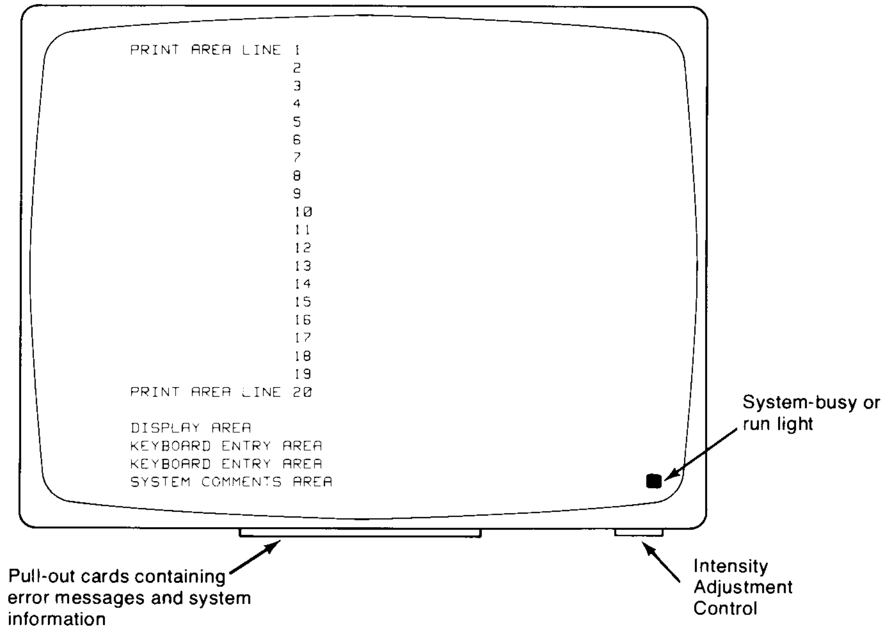
The keyboard is divided into several functional groups. This section gives a general overview of each block of keys. Specific keys are discussed in detail throughout the manual.

1

- **Alphanumeric Keys** – This area is similar to a standard typewriter keyboard. The main difference is that any letter is entered in upper case when that key is pressed and in lower case when pressed with (SHIFT) held down. For example, to enter a capital A, press (A); to enter a lower case a, hold down (SHIFT), then press (A).
- **Numeric Keys** – All the keys needed to enter numbers and do simple arithmetic are located in this block. The numeric keys in the alphanumeric section of the keyboard can also be used for the same purposes. (E) is used for scientific notation, indicating that an exponent follows.
- **Special Function Keys (SFK's)** – These keys can be defined or redefined for use as typing aids for statements, variable names or other series of keystrokes which are used often. Many of them have pre-defined definitions. Any of the special function keys can also be defined to have program interrupt capability (see Chapter 8 for more information).
- **Program Keys** – These keys provide program listing and editing capabilities.
- **Program Control Keys** – These keys – the gold keys found in the alphanumeric area and (STEP) – are used to control a running program. Lines can be stored; the program can be started or stopped. (STEP) is used to execute a program one line at a time; it is located beneath (STOP) in the system keys area.
- **Cursor Keys** – These keys can be used to control the position of the cursor and edit the line which is currently being entered.
- **Display Keys** – These keys can be used to move the cursor and roll the printout area of the CRT on the 9835A to view all lines of output. On the 9835B, these keys can be used to view the entire display and scroll the program while editing.
- **Typing Aid Keys** – These keys provide tab capability, two additional typing modes and the capability to recall previous keyboard entries.
- **System Command Keys** – These keys provide miscellaneous system control features like setting print all mode or rewinding the tape.

1

The CRT (9835A)



The 9835A comes equipped with a 24-functional-line by 80-character CRT (Cathode Ray Tube) display. It is the primary means of viewing data, keyboard inputs, results, program listings, error messages and system comments, and editing programs.

The CRT is divided into four areas while in normal mode as shown in the drawing above. Other modes are accessed while editing a program or special function key. The four areas are –

- **Printout Area** – Lines 1 through 20 are similar to a printing device. When the machine is switched on, this area is the standard system printer to which output from PRINT, PRINT USING, CAT and LIST is directed. It is also, at power-on, the print all printer when in the print all mode; see Chapter 2.

  Notice that the figure above shows 25 lines. Line 21 is a blank line and serves as a separator only, leaving 24 functional lines.

- **Display Line** – Line 22 is used to display output generated by DISP, and any INPUT prompt or question mark.

- **Keyboard Entry Area** – Lines 23 and 24 are accessible only through keyboard inputs. Every line that is typed in is displayed in this area. The first position in line 23 is known as the "home" position of the cursor. As the 148th character is keyed in, a beep indicates that only 12 more characters can be entered.

- **System Comments Line** – Line 25 is reserved for error messages, mode indicators, and the run light: ▒. Results of keyboard operations such as 3+5 (EXEC) or X (EXEC) also appear in this line.

CRT brightness is controlled by the knob underneath the CRT on the right-hand side.

CRT Pull-out Cards

The four cards under the CRT serve as a handy reference for operating the 9835A. They are –

- Error Messages
- Statements
- ROM Error Messages
- About the 9835A

The Display (9835B)

The 9835B has a 32-character, 5 by 7 dot matrix display which is the primary means for viewing keyboard entries, error messages and displays, and for editing programs. Even though only 32 characters can be displayed at one time, up to 160 characters can be keyed in. After the 32nd character, additional characters which are keyed in cause the displayed line to shift to the left. (←) and (→) can be used to view the entire display. After 148 characters are typed, a beep indicates that only 12 more can be entered.

NOTE

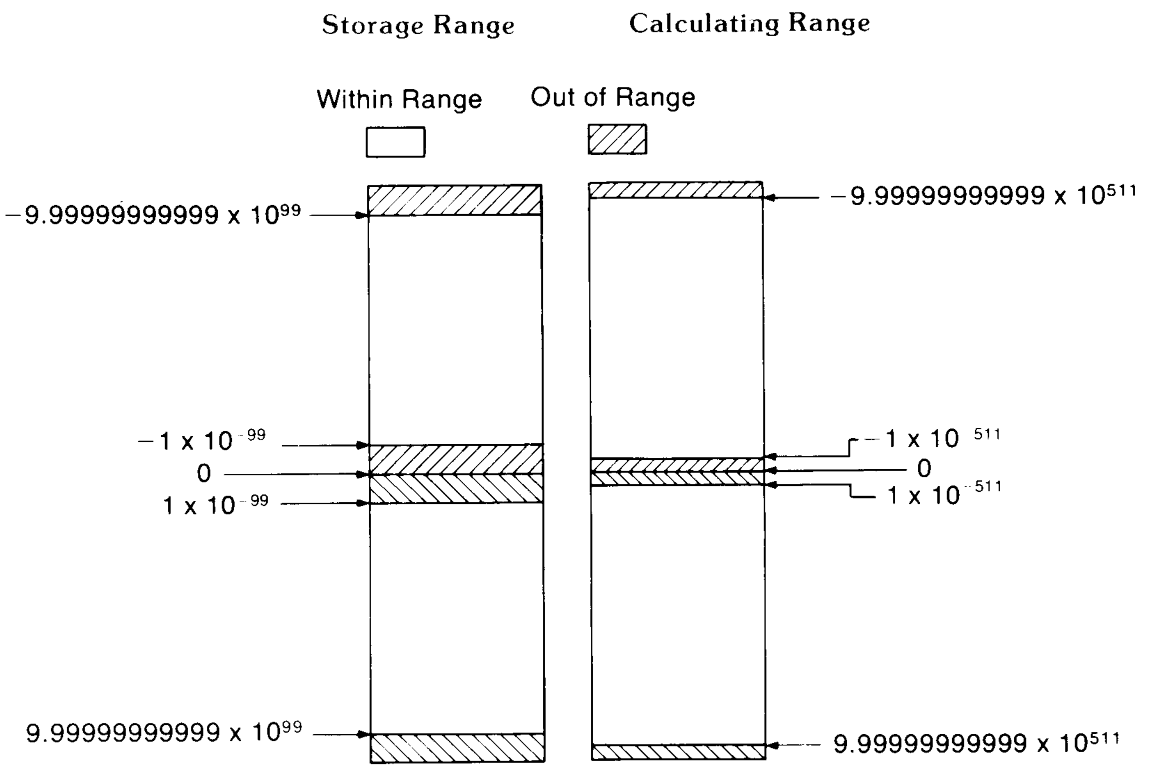
Throughout this manual, references are made to various lines of the CRT. 9835B users should interpret these to mean the display.

The Internal Printer

The optional internal thermal printer is a 16-character printer which can be used for permanent output. It is used with special blue-printout, heat-sensitive (thermal) paper. Paper is advanced using the paper advance wheel to the right of the printer. Ordering and loading paper is covered in the Owner's Manual.

Range

The range of values which can be entered or stored in memory is $-9.99999999999 \times 10^{99}$ through -1×10^{-99}, 0, and 1×10^{-99} through $9.99999999999 \times 10^{99}$. However, the range of intermediate calculations is $-9.99999999999 \times 10^{511}$ through -1×10^{-511}, 0, and 1×10^{-511} through $9.99999999999 \times 10^{511}$.



The extended calculation range is useful for calculations which have intermediate results outside the storage range, but the final result within storage range. For example $(9.2 \times 10^{23} \times 8.6 \times 10^{80}) / (1 \times 10^{24})$. When the first two values are multiplied, their result is (7.912×10^{104}). This intermediate result cannot be stored, but the final result, 7.912×10^{80}, can.

Memory

Read/Write Memory

The 9835A/B uses two types of memory: **Read/Write Memory** and **Read Only Memory** (ROM). Read/Write memory is used to store programs and data. When you store a program or data, you "write" into the memory. When you access a line of your program or a data element, you "read" from memory, thus the term Read/Write. Read/Write memory is temporary; it can be changed or erased. The contents of Read/Write memory are lost when the computer is shut off.

Programs and data in Read/Write Memory can be saved for future use by recording the information on a tape cartridge or other storage medium.

Read/Write Memory is available in various sizes. These options are listed in the Owner's Manual.

Read Only Memory

Read Only Memory differs from Read/Write Memory in that it is permanent. When the computer is turned off, the Read Only Memory is unaffected. ROMs can be inserted into one of the drawers in the front of the machine, making it possible to expand the language and capabilities. A small amount of Read/Write Memory is used by some plug-in ROMs. This area is called "working storage". The working storage used by each ROM is listed in the manual for that ROM.

1

Simplified Read/Write Memory Organization

(high addresses)

Memory area	Description
reserved for internal use [2]	This area is used for system configuration information – CRT vs. display, for example.
used for Option ROM Read/Write [2]	The amount used by each ROM is listed in the manual for that ROM.
(boundary 1)	
binary routines [2]	Binary routines are added to existing ones as they are loaded into memory using LOAD BIN or LOAD.
calling environment execution stack [2]	The execution stacks contain DATA pointers, subroutine return pointers, FOR-NEXT matching, and other indicators for program execution. The current environment execution stack also contains a program pointer to monitor which line is being executed currently. The size of an execution stack varies during program execution.
current environment execution stack [2]	
free memory –used as needed– [2]	Buffers for I/O and mass storage operations use Read/Write Memory. SFK definitions use 82 bytes at power-on.
buffers SFK definitions [2]	
First subprogram symbol table	Each symbol table contains variable names, any variable attributes (integer precision, array, etc.), and a value pointer which points to the value of the variable in the value area.
First subprogram	
Main program symbol table	Each successive subprogram and its symbol table comes "after" (has a higher address than) the previous one.
Main program	
Value Area	Contains the values for all main and subprogram variables.
Common	Contains the values of all variables declared in COM statements.

(low addresses)

1 This boundary is fixed at power-on.
2 This information must be in Block 0; see Appendix F for more information.

For more information about memory organization and how it relates to programming, see Appendix F.

Memory Loss

If your memory size seems smaller than the total amount that is installed in your machine, you may be experiencing a partial memory loss. This condition could be detected when executing SCRATCH A then LIST, which displays the number of bytes of available memory, or when loading a previously used program and getting an unexpected memory overflow (ERROR 2), though the program had fit into memory previously. Any decreases in memory size would be in increments of approximately 8192 bytes. Should this condition occur, try turning the power off, then on, several times, checking the memory size each time to see if it varies. Should the problem persist, call your HP Sales and Service Office.

Error Messages and Warnings

When an error occurs, the machine beeps and displays an error number or a warning message. The error number references a description that helps you pinpoint the cause of the error. For example, typing in 5/0 (EXECUTE) causes ERROR 31 to be displayed. ERROR 31 means division by zero. A warning message can also appear and describes the error. For example, typing in 3*(5/7 causes IMPROPER EXPRESSION to be displayed. On the 9835A, the expression is displayed in the keyboard entry area with the cursor flashing where the parenthesis should be. On the 9835B, pressing (RECALL) returns the expression to the line with the cursor flashing where the parenthesis should be.

If an error occurs within a running program, the machine halts and the line number where the error occurs is displayed. For example, when 150 X=TAN(3*PI/2) is executed, ERROR 24 IN LINE 150 occurs.

A complete list of the error numbers and their meanings is given in Appendix G of this manual, in the 9835A/B Reference Guide supplied with the 9835A/B, and also on pull-out cards 1 and 3 under the CRT of the 9835A.

Chapter 2
Keyboard Operations

Introduction

This chapter introduces the basic concepts of keyboard operations. How to use the keyboard, the system command keys, the special function keys and computer operating modes are all covered.

9835A vs. 9835B

There are differences between the 9835A and 9835B in keyboard operations. These differences are a result of the inherent differences between the CRT and the single-line display. Throughout this manual, references are made to various lines of the CRT ("...is displayed in the system comments line.", for example). 9835B users should interpret these to mean "the display".

The main differences occur while keying in and editing keyboard entries and while viewing the display. Editing on the 9835A and 9835B are discussed separately at the end of this chapter.

Other differences are discussed as they occur in the topics which follow.

The Run Light

While any operation (program, command, etc.) is executing, a run light is displayed. When the operation is complete, the light goes out. On the 9835A, ▒ is displayed on the right-hand end of the system comments line. On the 9835B, a small red light appears in the left-hand end of the display.

Mode Indicators

Various **mode indicators** are used to indicate that a certain mode has been set or cancelled. The modes are typewriter mode, space dependent mode and print all mode. Typewriter mode and print all mode are discussed in this chapter; space dependent mode is discussed in Chapter 4.

On the 9835A, any mode indicator for print all mode remains displayed until it is replaced by some other system message like an error message, while the indicator for typewriter or space dependent mode remains displayed as long as the mode is set. On the 9835B, the indicator is displayed only until another key is pressed.

2

Basic Operations

Executing Operations

Many keyboard operations (numeric computations, commands, and statements without line numbers) are performed (executed) by typing in the operation, then pressing (EXEC). For example –

(2)(+)(2)(EXEC)   4

Pressing (EXEC) performs the operation, displays the result of a computation in the system comments line and stores the operation that was executed in the recall buffer and the result in the RESult buffer for later use. When the operation above is executed, '4' is displayed, '2+2' is stored in the recall buffer and '4' is stored in the RESult buffer.

Simultaneous Computations

Several numeric and string expressions can be entered and solved at the same time by separating each with either a comma or a semicolon. For example, if the diameter of a circle is 12 feet ($d = 12$), both the area ($A = \pi d^2/4$) and the circumference ($C = \pi d$) can be found at the same time by executing –

```
PI*12^2/4, PI*12
```

The result is –

```
113.097335529          37.6991118432
```

More than two expressions can be solved simultaneously. The results are displayed; excessively long results (greater than 80 characters) can be viewed totally by setting the print all mode (press (PRT ALL)).

The only difference between separating the expressions with commas or semicolons is that semicolons cause the results to be packed together while commas leave more space between each result. This is discussed in more detail with the DISP statement.

Print All Mode

The print all mode is set by pressing –

PRT ALL

which is found in the System Keys area. Print All on is displayed to inform you that the print all mode is set.

2

In print all mode, the computer outputs to the print all printer any operations which are executed from the keyboard, including computations, displayed results, stored program lines and error messages. This provides a useful audit trail of previous operations for later references – to duplicate a procedure for example. When a program is running and print all mode is set, all display (DISP) results, trace messages and error messages are output to the print all device.

Print all mode is turned off by pressing PRT ALL again. This causes Print all off to be displayed.

Print All Printer

The standard print all printer is the CRT (9835A) or internal printer (9835B) when the computer is turned on and after SCRATCH A is executed. If you have a 9835B with no internal printer, setting the print all mode causes an I/O error on select code 16. You must specify an external printer as the print all printer.

The print all printer can be changed by executing the PRINT ALL IS statement[1] –

PRINT ALL IS select code [, HP-IB device address]

The definitions of select code and HP-IB device address are found at the beginning of Chapter 4 and in Appendix D. See the I/O ROM or specific peripheral manual for further explanation of HP-IB device addresses.

Here are some examples –

```
PRINT ALL IS 16     ! CRT IS PRINT ALL PRINTER
PRINT ALL IS 0      ! STRIP PRINTER IS PRINT ALL PRINTER
PRINT ALL IS 6      ! PRINTER AT SELECT CODE 6
PRINT ALL IS 7,2    ! HP-IB PRINTER
```

1 The PRINT ALL IS statement can also be programmed.

Recalling Previous Entries

Any keyboard entry followed by STORE, CONT, or EXEC is stored in a recall buffer and can be recalled into the keyboard entry area by pressing –

RECALL

which is found is the Typing Aid keys area.

Entries are stored into a 350-byte (character) recall buffer on a first in, last out basis. Each time RECALL is pressed, a previous keyboard entry is recalled. To move the other direction through the recall buffer (recalling more recent keyboard entries), press RECALL while holding down SHIFT.

When the recall buffer becomes full, each new entry causes one or more of the oldest entries, depending on size, to be lost.

On the 9835B, you can press RECALL after errors resulting from keyboard operations to recall the line containing the error into the display. For many errors, a flashing cursor indicates the location of an error in the line.

Resetting the Computer

If the computer becomes inoperative due to a system or I/O malfunction, it may need to be reset. The computer is reset and returned to a ready state by holding down –

CONT'L

then pressing –

RESET
STOP

Resetting the computer immediately aborts all machine activity. The reset operation is a hardware-oriented operation and returns peripherals and HP-IB interfaces, as well as the computer, to a ready state. If a program is running, any pending or executing I/O operation is terminated and information may be lost.

NOTE

There is a finite possibility that the reset operation will cause the entire memory to be scratched, like executing SCRATCH A. The current program environment may or may not be preserved. Use it only if nothing else, such as pressing the STOP key, brings the machine to a ready state.

See the Reset table in Appendix E for a list of conditions affected by reset.

The Keyboard

Introduction

The typewriter-like keyboard is used to enter operations and program lines into the keyboard entry area. Here are some facts related to keyboard operation –

- **Color** – In general, keys of the same color have similar functions. For example, all of the alphanumeric keys are the same beige color. The control key affects the operation of various keys when it is held down. The control features of these keys are indicated in rust-colored lettering above the key.

- **Spacing** – In general, spaces are not important. It makes no difference, for example, if you key in –

  A+B or A + B

  They are interpreted in the same way. Spacing, however, is important when using text (characters within quotes), when printing and displaying messages and in space dependent mode for program entering (see Chapter 4).

- **Repetition of Keys** – When a key is held down for more than a second, its operation is repeated rapidly. This is an especially useful feature with the editing keys.

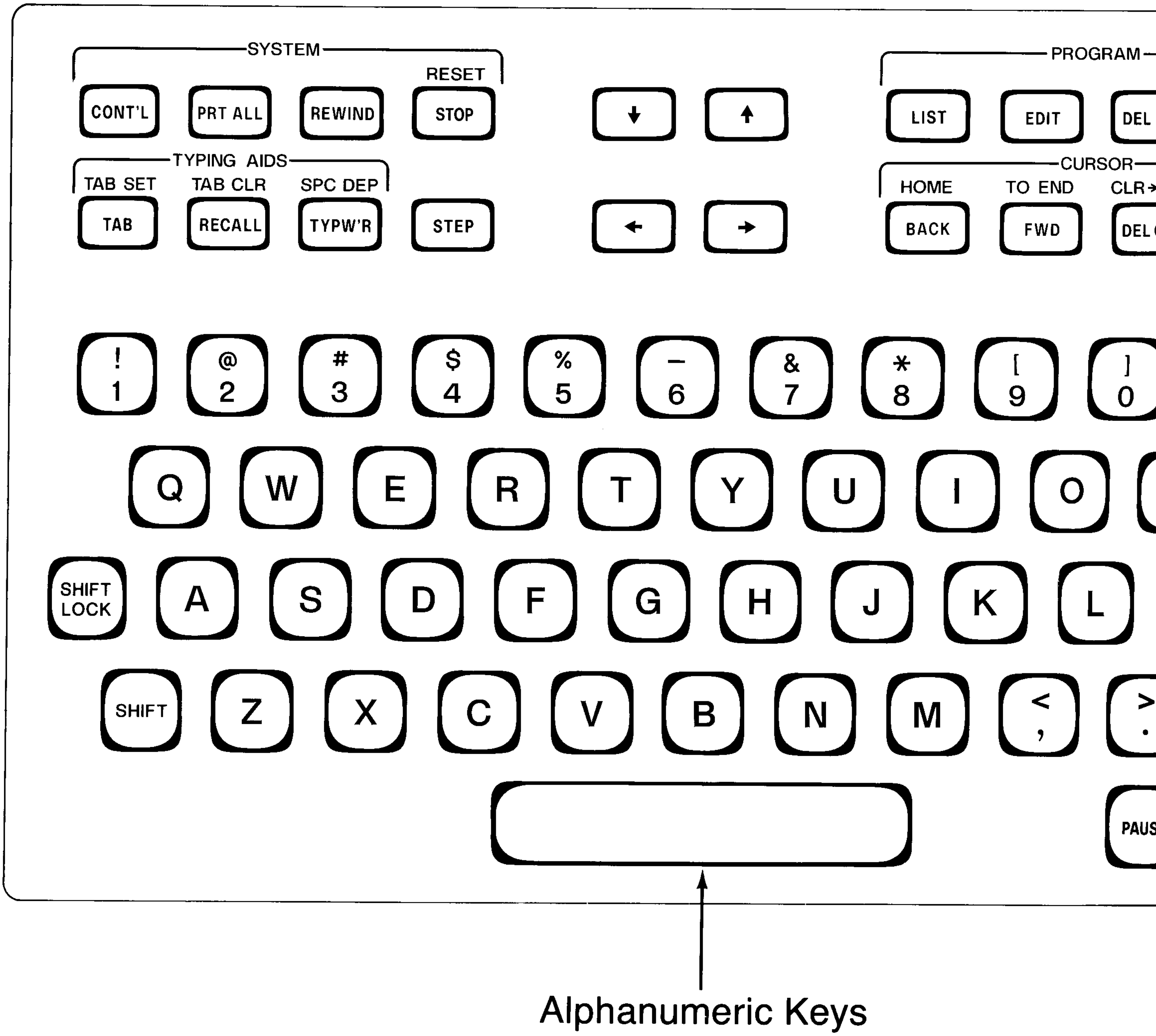
- **Typing Aid Keys** – A typing aid key is one that enters a BASIC keyword or other series of keystrokes. LIST is an example. The special function keys can be defined as typing aids.

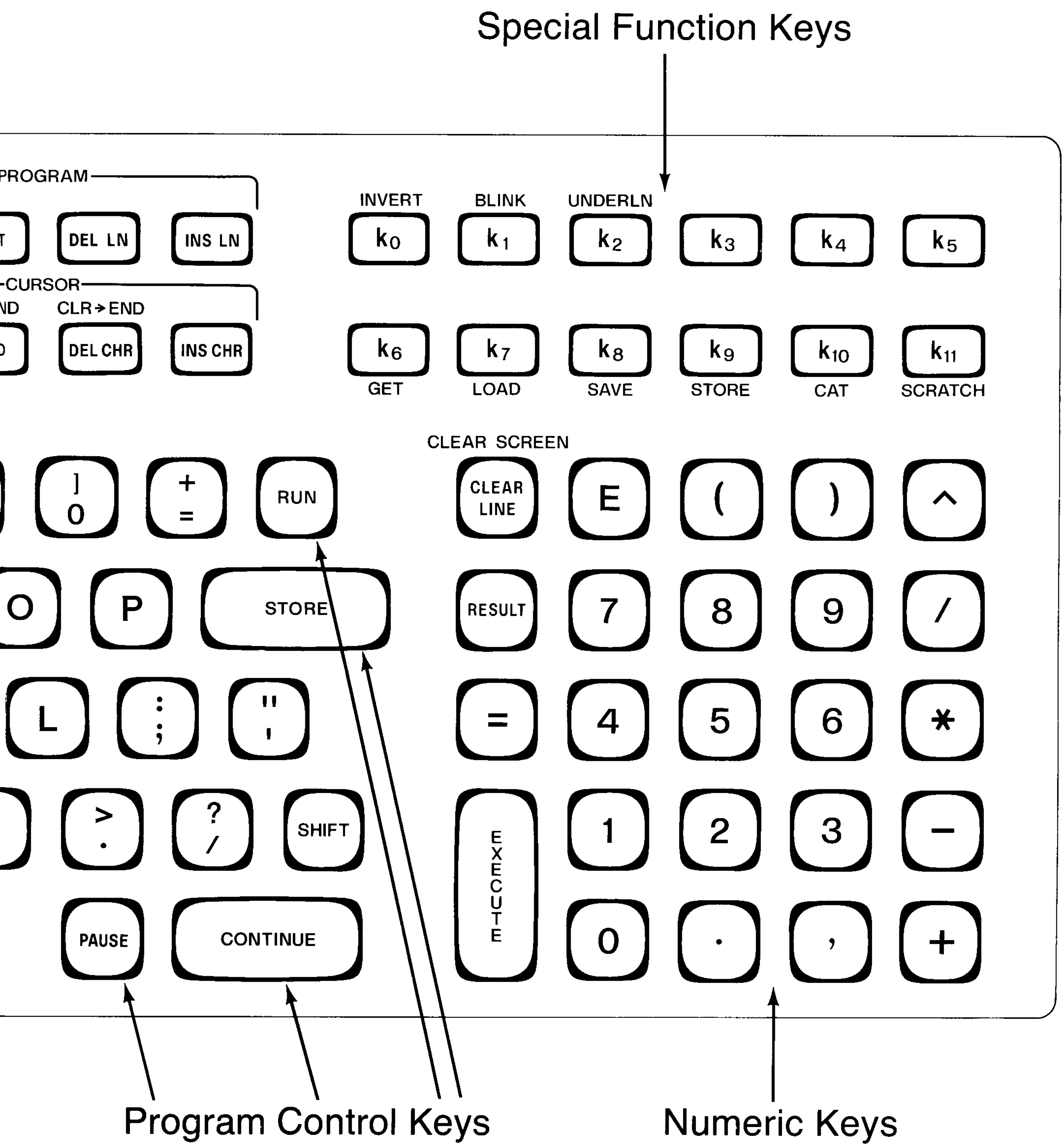
Typewriter Mode

When TYPW'R is pressed, the keyboard is set to the typewriter mode. Normally, when a letter is keyed in, it is in upper case; it appears in lower case when shifted. However, in typewriter mode, lower case is unshifted and upper case is obtained by shifting. Typewriter mode is very useful for entering text.

When TYPW'R is pressed, TYPWTR is displayed on the right-hand side of the system comments line of the 9835A; Typewriter mode is displayed on the 9835B. To exit typewriter mode, press TYPW'R again. Typewriter off is displayed on the 9835B.

The typewriter mode can also be set and unset within a program; for more information, see Chapter 4.


SYSTEM
RESET
CONT'L
PRT ALL
REWIND
STOP
PROGRAM
LIST
EDIT
DEL
TYPING AIDS
TAB SET
TAB CLR
SPC DEP
TAB
RECALL
TYPW'R
STEP
CURSOR
HOME
TO END
BACK
FWD
SHIFT LOCK
SHIFT
Alphanumeric Keys


Special Function Keys
PROGRAM
DEL LN
INS LN
CURSOR
CLR→END
DEL CHR
INS CHR
INVERT
BLINK
UNDERLN
k0
k1
k2
k3
k4
k5
k6
k7
k8
k9
k10
k11
GET
LOAD
SAVE
STORE
CAT
SCRATCH
CLEAR SCREEN
CLEAR LINE
E
(
)
^
]
0
+
=
RUN
O
P
STORE
RESULT
7
8
9
/
L
:
;
"
'
=
4
5
6
*
>
.
?
/
SHIFT
EXECUTE
1
2
3
−
PAUSE
CONTINUE
0
.
,
+
Program Control Keys
Numeric Keys

2

Tab Capabilities

The keyboard has two keys used to control the position of the cursor in the keyboard entry area.

Setting and Using Tabs

Tabs can be set at any of the 160 positions in the keyboard entry area. To set a tab at the current position of the cursor, hold down –

CONT'L

then press –

TAB SET
TAB

When TAB is pressed, the cursor advances to the next tab setting. If it moves across characters that are already keyed in, they don't change. If no characters have been keyed in, the intervening character positions are filled with spaces (blanks). If there are no further tab settings and TAB is pressed, the cursor moves to the 160th character position which is the last position in the keyboard entry area; a beep occurs if the cursor was to the left of position 148 prior to the tab.

Tabs can be very useful for inserting comments at the end of program lines (see Chapter 4). By setting a tab somewhere between columns 40 and 60, you can easily line up all of your comments, making your program listing neat and easier to follow.

Clearing Tabs

Individual tabs can be cleared by using TAB to move the cursor to the position to be cleared, holding down –

CONT'L

then pressing –

TAB CLR
RECALL

Editing on the 9835A

If you make a mistake while keying a line into the keyboard entry area, you can use the line editing keys to change the line. This section covers editing keyboard entries on the 9835A. Editing on the 9835B is discussed later in the chapter.

Clearing the Line

The entire keyboard entry area can be cleared by pressing –

CLEAR LINE

Then a new line can be keyed in.

Moving the Cursor

If a mistake is made in part of a line, the cursor can be repositioned and the mistake corrected.

Examples

For example, suppose you wanted to execute this line –

```
1111+3333+5555
```

But by accident you key in –

```
1111+2222+5555
```

To correct this, position the flashing cursor so it is underneath the first 2. This may be done in any of the following ways –

- Press BACK or ← repeatedly.
- Reposition the cursor to the "home" position (first position in the keyboard entry area). To do this press –

  SHIFT[1] HOME BACK

  Then press → or FWD repeatedly.
- Press → or FWD repeatedly. When the cursor is positioned after the last character in the line, pressing → or FWD causes it to move to the home position.

1 SHIFT indicates that the following key is shifted.

When the flashing cursor is under the first 2, key in 3333.

The resulting display would be –

1111+3333+5555

with the flashing cursor under the second plus sign. Now press (EXEC). The line can be executed regardless of the position of the cursor.

2

As another example, suppose you wanted to change the previous line to –

1111+3+5555

First press RECALL to recall the line into the keyboard entry area. To delete the three 3's, position the flashing cursor so it is under the first 3. Then press DEL CHR three times. Notice that the cursor remains under the last 3.

To change the line to –

1111+3+550055

position the cursor under the third 5. Now press INS CHR. This causes the insert cursor (inverse video) to appear over the 5. Now type in 00. Notice that the line shifted to the right two characters. The insert cursor is still flashing over the 5 indicating that more characters could be inserted.

Let's manipulate this expression –

COLORADO, SKI COUNTRY USA

To change it to –

COLORADO

the clear-to-end function can be used. Position the cursor under the comma, then press (SHIFT) CLR→END DEL CHR.

Now move the cursor to the home position by pressing (SHIFT) HOME BACK. Press INS CHR, then type in LOVELAND,. Now you see –

LOVELAND,COLORADO

To add the zip code, the to-end function can be used. Press [SHIFT] [TO END / FWD]. This positions the cursor to the character position after the last character in the line. Now press the space bar and type in 80537. Now you see –

```
LOVELAND, COLORADO 80537
```

Summary (9835A)

In summary, the character editing keys work as follows –

[CLEAR LINE] — Clears the keyboard entry area and the system comments line of everything except any mode indicators (TYPWTR, SPACE DEPENDENT) and the run light.

[SHIFT] [CLEAR LINE] — Clears the entire CRT of everything except any mode indicators, the run light and any INPUT, LINPUT, or EDIT prompt.

[→] or [FWD] — Moves the cursor one character position to the right. If the cursor is one position to the right of the last character in the line, pressing [→] or [FWD] one more time moves it to the first position in the line.

[SHIFT] [TO END / FWD] — Moves the cursor to the character position immediately following the last character in the line.

[←] or [BACK] — Moves the cursor one character position to the left. If the cursor is at the beginning of the line, pressing [←] or [BACK] one more time moves the cursor to the character position after the last character in the line.

[SHIFT] [HOME / BACK] — Moves the cursor to the home position which is the first position in the keyboard entry area.

[INS CHR] — Causes the insert cursor (change in inverse video mode) to appear over the character at the position of the flashing cursor. Characters are inserted to the left of the cursor, causing the rest of the line to move to the right. The insert character mode is exited by pressing [INS CHR] again, moving the cursor, or by pressing [STORE], [CONT], or [EXEC].

[DEL CHR] — Causes the character at the position of the flashing cursor to be deleted. The cursor remains in the same position and the rest of the line moves one position to the left as each character is deleted.

[SHIFT] [CLR→END / DEL CHR] — Clears the keyboard entry area from the position of the cursor to the end. It also clears the system comments line of everything except any mode indicators and the run light.

Controlling the 9835A CRT Display

Two keys are used to control the printout area of the CRT.

[↑] Moves ("scrolls") the lines in the printout area up one line. If any lines are below the displayed lines, pressing [↑] brings one line up into the bottom line of the printout area.

[↓] Pressing [↓] causes one line, if any, above the top line in the printout area to move into the top line; the lines all scroll down.

2

Editing on the 9835B

If you make a mistake while entering lines into the display, you can use the character editing keys to change the line.

Clearing the Display

The display can be cleared by pressing –

[CLEAR LINE]

then a new line can be keyed in.

Editing the Display

If a mistake is made in part of a line, the character editing and display keys –

[BACK] [FWD] [INS CHR] [DEL CHR] [←] [→]

can be used to edit the display. Two flashing cursors are associated with line editing –

▒ the replace cursor

▒ the insert cursor

Moving the Cursor

If a mistake is made in part of a line, the cursor can be repositioned and the mistake corrected.

Examples

For example, suppose you wanted to execute this line –

```
1111+3333+5555
```

But by accident you key in –

```
1111+2222+5555
```

To correct this, position the replace cursor so that it is over the first 2. This can be done in one of the following ways –

- Press BACK repeatedly.
- Reposition the cursor to the "home" position (first position in the display). To do this, press –

  SHIFT[1] HOME BACK

  then press FWD repeatedly.

When the replace cursor is over the first 2, key in 3333.

The resulting display would be –

```
1111+3333+5555
```

with the replace cursor over the second plus sign. Now press EXEC. The line can be executed regardless of the position of the cursor.

---

1 SHIFT indicates that the following key is shifted.

As another example, suppose you wanted to change the previous line to –

1111+3+5555

First press RECALL to recall the line into the display.

To delete the three 3's, position the cursor so it is over the first 3. Then press DEL CHR three times.

2

To change the line to –

1111+3+550055

position the cursor under the third 5. Now press INS CHR. This causes the insert cursor to appear over the 5. Now type in 00. Notice the line shifted to the right two characters. The insert cursor will still be flashing over the 5 indicating that more characters could be inserted.

Let's manipulate this expression –

COLORADO, SKI COUNTRY USA

To change it to –

COLORADO

the clear-to-end function can be used. Position the cursor under the comma, then press SHIFT CLR→END DEL CHR.

Now move the cursor to the home position by pressing SHIFT HOME BACK. Press INS CHR, then type in LOVELAND, . Now you see –

LOVELAND,COLORADO

To add the zip code, the to-end function can be used. Press SHIFT TO END FWD. This positions the cursor to the character position after the last character in the line. Now press the space bar and type in 80537. Now you see –

LOVELAND,COLORADO 80537

Moving the Display

For viewing or editing keyboard entries which are longer than 32 characters, these two keys –

[←] [→]

can be used to move the line in the display to the left or right, allowing you to view a display of up to 160 characters.

Example

For example, key in the following expression, which repeats each numerical digit eight times.

11111111 + 22222222 + 33333333 + 44444444 + 55555555 + 66666666 + 77777777 + 88888888 + 99999999

Display begins moving to the left.

After the last character has been keyed in, the display looks like this –

```
66666+77777777+88888888+99999999
```

To view any other portion of the expression, hold [→] down for a few seconds; the display is rapidly moved to the right. When the portion of the display that you want to see is visible, release the key. To view the end of the display, you can use [←].

The entire line can be executed at any time, regardless of the current portion being displayed –

[EXEC]

```
499999995
```

Summary (9835B)

CLEAR LINE — Clears the display of everything except the run light.

FWD — Moves the cursor one position to the right. For a line which has just been recalled or typed into the display, pressing FWD causes the cursor to appear over the left-most character visible in the display.

SHIFT TO END FWD — Moves the cursor to the character position immediately following the last character in the line.

BACK — Moves the cursor one position to the left. If the cursor is not visible, pressing BACK causes the cursor to appear over the right-most character visible in the display.

SHIFT HOME BACK — Moves the cursor to the home position (the first position in the line).

INS CHR — Causes the insert cursor to appear over the character at the position of the flashing cursor. Characters are inserted to the left of the cursor, causing the rest of the line to move to the right. The insert character mode is exited by pressing INS CHR again, moving the cursor, or by pressing STORE, CONT, or EXEC.

DEL CHR — Causes the character at the position of the flashing cursor to be deleted. The cursor remains in the same position and the rest of the line moves one position to the left as each character is deleted.

SHIFT CLR→END DEL CHR — Clears the display from the position of the cursor to the end.

← — Moves the line in the display to the left, eight characters at a time.

→ — Moves the line in the display to the right, eight characters at a time.

Operating Modes

The computer can operate in any of five modes –

- Calculator mode – no program is running and the computer is awaiting inputs or calculating keyboard entries.
- Program mode – a program is running.
- Live Keyboard mode – numeric computations and most statements and commands can be executed from the keyboard while a program is running. Program lines can be stored also. The running program is temporarily paused while a keyboard operation is executing.
- Edit Line mode – the program in memory is being edited. See EDIT LINE, Chapter 12.
- Edit Key mode - a Special Function Key is being defined as a typing aid. See EDIT KEY, which is discussed near the end of this chapter.

2

A SYSTEM BUSY message may be displayed if a keyboard operation is executed while a previous one is still executing, though some keyboard operations can overlap each other.

Live Keyboard

Live keyboard allows computations and most statements and commands to be executed from the keyboard while a program is running. It also allows lines of a running program to be changed by typing the new line in and pressing (STORE). You can also check the value of a variable by typing in its name and pressing (EXEC). If execution is currently in a subprogram, you may get an unexpected result if the variable isn't defined in the subprogram.

To see how live keyboard works, key in, store, and run the following program –

```
10   DISP "PROGRAM RUNNING"
20   GOTO 10
30   DISP "PROGRAM DONE"
40   END
```

While the program is running, you can use the numeric keys to balance your checkbook. Now key in and store this line –

```
20   GOTO 30
```

Live keyboard is disabled by executing the SUSPEND INTERACTIVE statement –

```
SUSPEND INTERACTIVE
```

While a program is running, any attempt to execute a keyboard operation or alter the program by storing a line or executing a program control command such as CONT will cause a PROGRAM EXECUTING or SYSTEM BUSY message to appear. When live keyboard is disabled, (STORE) and (EXEC) are disabled as well.

Execute SUSPEND INTERACTIVE and change line 20 of the previous program back to 20 GOTO 10. Now run the program and try to add 2 + 2 or store a program line.

2

Live keyboard is re-enabled by executing the RESUME INTERACTIVE statement –

RESUME INTERACTIVE

The Special Function Keys

The special function keys (SFKs), marked k0 through k11, provide a variety of uses: typing aids for frequently used statements, commands, operations and other series of keystrokes, program interrupting capability and, on the 9835A, accessing CRT special features. Their use as program interrupts is discussed in Chapter 8.

Pre-defined Definitions

These keys have the following definitions at power on or after SCRATCH A is executed.

Key	Function
(CONT'L) (k0 INVERT) *	Inverse video mode
(CONT'L) (k1 BLINK) *	Blinking mode
(CONT'L) (k2 UNDERLN) *	Underline mode
(k6 GET)	(CLEAR LINE) GET
(k7 LOAD)	(CLEAR LINE) LOAD
(k8 SAVE)	(CLEAR LINE) SAVE
(k9 STORE)	(CLEAR LINE) STORE
(k10 CAT)	(CLEAR LINE) CAT
(k11 SCRATCH)	(CLEAR LINE) SCRATCH

* 9835A only.

Special Features (9835A)

The CRT special features – inverse video, blinking and underline – can be used alone or combined. Each mode is entered by holding down CONT'L, then pressing the specific key. For an example of blinking, press –

CONT'L BLINK k1

then type in –

*****!!!!

To add inverse video to blinking, press –

CONT'L INVERT k0

and type in –

####

Each mode is exited by pressing CONT'L and the specific key again or by pressing any of the CLEAR keys. To get back to normal mode in the above example, press –

CONT'L k0 k1

These special features are very useful for highlighting text which is output to the CRT in programs. For example, line 20 of the previous example could be made more "eye-catching" by pressing CONT'L BLINK k1 before and after keying in PROGRAM RUNNING. Make sure that the quote marks aren't blinking.

2

Typing Aids

Keys 6 through 11 are defined at power on and SCRATCH A as typing aids so that frequently used operations can be entered with a single key stroke. These definitions are indicated below the appropriate key.

Defining Special Function Keys

There are 32 special function keys available to be defined as typing aids. These are –

Keys 0-11
Keys 12-23 (0-11 using SHIFT)
Keys 24-31 (4-11 using CONT'L)

The initial definitions of keys 6 through 11 are not permanent, but can be edited, or erased and redefined. These definitions were listed previously in this section.

NOTE

The CRT special feature definitions (9835A) are permanent. They are separate from the typing aid definitions.

To define or edit a key, execute –

EDIT[1] KEY key number

or type in –

EDIT

and press the key to be defined.

The computer is now in the edit key mode with the key number displayed at the top of the CRT and any current definition displayed. On the 9835B, the key number followed by a question mark, or any current definition, is displayed. Any keys on the keyboard, up to 70 keystrokes, can be entered to define a particular key, with these exceptions –

[TYPW'R] [STOP] [CONT'L] [SHIFT] [SHIFT LOCK]

In addition, the SFK itself may not be used in its own definition; this would cause an endless recursion.

Pressing the SFK that is being defined a second time stores the definition and returns the computer to the normal mode. Pressing it immediately after the edit key mode was entered defines that key as null if that key had no previous definition. [STOP] can be pressed at any time to abort the editing of the key; no new definition is stored and any previous definition remains.

1 ([EDIT] can be used to enter EDIT)

2

Example

For example, let's say you are keying in a program that has many PRINT statements. It would be handy to define key 0 as PRINT. Key in –

EDIT

Then press –

k0

Now type in –

PRINTΔ[1]

To store the definition, press –

k0

Now if you wanted to type in: PRINT X, Y, four keystrokes can accomplish this –

k0 X , Y

Example

One SFK definition can be used to define another. For example, say that k1 is defined –

Pay

Key 2 could by defined to be PRINT Pay, H by entering the edit key mode for key 2, then pressing –

k0 k1 , H

then storing the definition by pressing –

k2

Pressing k2 now enters –

PRINT Pay, H

An SFK can also be defined so that it performs an operation immediately. This is accomplished by having the last entry in the definition be one of the special terminator keys –

STORE CONT EXEC RUN PAUSE STEP INS LN DEL LN REWIND

Only one of these keys can be used in a key definition and it must be the last entry in the definition.

1 Δ Indicates a blank space

Example

In the previous example, k2 is defined as PRINT Pay, H. To define k3 as an immediate execute operation to execute PRINT Pay, H, enter the edit key mode for key 3. Then press –

k2 EXEC

Then store the definition by pressing –

k3

Now when you press k3, the values of Pay and H are automatically printed.

Example

As another example, say that you are writing a program which uses the above variables Pay and H and you want the values printed many times throughout the program. By defining k4 to be –

k2 STORE

the entire line PRINT Pay, H can be automatically stored after a line number by pressing key 4 following a line number.

If two or more SFKs that each contain a terminator key as part of their definition are used to define another SFK, execution stops with the first terminator key.

Example

For example, suppose key 12, key 13 and key 14 are defined as follows –

```
KEY 12
  PRINT "K12"
  -Execute
KEY 13
  PRINT "K13"
  -Execute
KEY 14
  -Key 12
  -Key 13
```

Pressing key 14 results in –

```
K12
```

The character editing keys –

BACK FWD ← → DEL CHR INS CHR

can be used to edit an SFK definition, or can be entered as part of an SFK definition. They must be pressed while CONT'L is held down to be entered as part of the key definition.

Example

For example, to change the previous definition of k2, PRINT Pay, H to DISP Pay, H, first enter the edit key mode for k0 which was defined as PRINT.

PRINT_

is displayed. Press ← or BACK (9835A only) 6 times to position the cursor under the P. Now type in –

DISP

To delete the T, press DEL CHR.

To store the new definition, press –

k0

The definition of key 2 is automatically altered because key 0 is part of its definition.

Example

Here's an example of using INS CHR in a key definition. The definition of key 0, DISP, can be changed to include quote marks and an insert cursor so that only the text need be entered into the display statement. Enter the edit key mode for key 0. Key in two quote marks, then hold down CONT'L and press BACK (or ← on the 9835A) and INS CHR. Now press key 0 to store the definition.

Now you can press key 0, type in the text you wish to display, and execute or store the line.

Many of the keys on the keyboard do not have a directly printable character, but instead cause some action to occur when pressed. To represent these keys in the edit key mode, each key has a unique keycode which is displayed on a separate line. The keycodes are –

2

Key	Keycode
TAB	Tab
CONT'L TAB SET / TAB	Tab set
CONT'L TAB CLR / RECALL	Tab clear
RECALL	Recall
CONT'L SPC DEP / TYPW'R	Space dep
STEP	Step
PRT ALL	Print all
REWIND	Rewind
↓	Down arrow
↑	Up arrow
CONT'L →	Fwd (9835A); Right arrow (9835B)
CONT'L ←	Back (9835A); Left arrow (9835B)
LIST	List
EDIT	Edit
DEL LN	Del line
INS LN	Ins line
CONT'L BACK	Back
SHIFT HOME / BACK	Home
CONT'L FWD	Fwd
SHIFT TO END / FWD	To end
CONT'L DEL CHR	Del char
SHIFT CLR→END / DEL CHR	Clear to end
CONT'L INS CHR	Ins char
CLEAR LINE	Clear line
RUN	Run
STORE	Store
PAUSE	Pause

2

Key	Keycode
[CONT]	Continue
[EXEC]	Execute
[RESULT]	Result
[CONT'L] [k0 INVERT]	Inverse Video (9835A); Undefined (9835B)
[CONT'L] [k1 BLINK]	Blink (9835A); Undefined (9835B)
[CONT'L] [k2 UNDERLN]	Underline (9835A); Undefined (9835B)
[CONT'L] [k3]	Undefined
[CONT'L] [k4] thru [k11]	Key 24 thru Key 31
[SHIFT] [k0] thru [k11]	Key 12 thru Key 23
[k0] thru [k11]	Key 0 thru Key 11

When any of these keys is pressed for part of an SFK definition, the previous parts of the definition roll up on 9835A; the keycode for the key just pressed appears on the line above the cursor, with the cursor in the entry area ready for another key.

When editing keycodes on the 9835A, the editing keys [←] [→] [FWD] [BACK] [DEL CHR] [INS CHR] **appear** to have a slightly different function. Using [←] or [BACK] to move the cursor back into previously defined parts causes the display to roll down. [→] causes it to roll up. [INS CHR] allows keystrokes to be inserted above (before) a keycode entry.

2

Example

For example, let's say you wanted to define K10 to set three tabs each three spaces apart but defined it to be –

```
-Tab Clear
-Right arrow
-Right arrow
-Right arrow
-Tab set
-Right arrow
-Right arrow
-Right arrow
-Right arrow
-Tab set
```

To change the Tab clear to Tab set and delete one of the last four Right arrows, do the following –

Enter the edit key mode for K10. The flashing cursor will be in the line under the last Tab set. Now press –

← or BACK

10 times to position Tab Clear in the cursor line. Now hold down –

CONT'L

then press –

TAB SET
TAB

To delete a Right arrow, press –

→ or FWD

four times to position a Right arrow in the cursor line. Now press –

DEL CHR

to delete that entry.

Finally, press K10

When editing key codes on the 9835B, [←] and [→] move the display eight **keystrokes**, while [BACK] and [FWD] move the cursor from one keycode to the next.

Remember, a maximum of 70 keystrokes can be used to define an SFK.

HINT

If you press a defined SFK and get an unexpected UNDEFINED KEY message, check the shift lock key.

Erasing Special Function Keys

To erase a specified key definition, type in –

SCRATCH

(or press [k11 SCRATCH] if it still has its power-on definition) then press the key you wish to erase.

To erase the typing aid definitions of all special function keys, execute –

SCRATCH KEY

Erasing all SFK definitions adds 82 bytes to the power-on value of space available in Read/Write Memory, since the initial SFK definitions use 82 bytes.

Listing SFK definitions

All or selected SFK typing aid definitions can be listed. Executing this command –

LIST[1] KEY

causes all typing aid definitions to be listed on the standard printer. (see PRINTER IS, Chapter 10). To specify a different device on which the listing is to occur, execute –

LIST KEY# select code [, HP-IB device address]

2

A single key can be listed by executing –

LIST KEY [#select code [, HP-IB device address];] SFK number

or

LIST (kn)

Here are some examples of LIST KEY –

```
LIST KEY        ! LISTS ALL KEYS
LIST KEY #6     ! LISTS ALL KEYS TO SELECT CODE 6
LIST KEY #6,2   ! LISTS ALL KEYS - HP-IB PRINTER
LIST KEY 8      ! LISTS KEY NUMBER 8
LIST KEY #6;8   ! LISTS KEY #8 TO SELECT CODE 6
```

1 The (LIST) key is found in the Program Keys and may be used to enter the word LIST.

Chapter 3
Mathematics

Introduction

This chapter covers the concepts related to mathematics on the 9835A/B. This includes keyboard arithmetic, number formats for output, operators, functions, math errors and variables.

While reading this chapter, remember that all of the statements and functions described here can also be programmed.

Keyboard Arithmetic

The arithmetic operations that can be performed on the 9835A/B are addition (+), subtraction (−), multiplication (*), division (/), exponentiation (^ or **; ** is listed ^), integer division (DIV), and modulo (MOD).

To perform an arithmetic operation, such as 8 * 2, first you key in the expression – (8)(*)(2). Then press (EXEC); 16 is displayed. Note that an operation such as 8^{-2} must appear with parentheses as 8^(-2).

If you execute an operation that has a large result, such as 608000 * 90000000 the result is automatically displayed in scientific notation with 11 digits to the right of the decimal point – 5.40000000000E+13.

There's more about number formats later in this chapter.

The RESult Function

The value which is displayed after pressing the execute key is stored in a location called the "RESult" buffer. It is obtained for use in subsequent calculations by pressing [RESULT] or keying in RES (the RESult function). Here are some examples –

[8] [*] [2] [EXEC]	16
[RESULT] [*] [3] [.] [7] [EXEC]	59.2
[R] [E] [S] [−] [1] [5] [EXEC]	44.2

DIV Operator

The DIV (integer division) operator returns the integer portion of the quotient. For example –

-6 DIV 1.8 [EXEC]	-3

The following formula illustrates how DIV is calculated –

A DIV B = SGN(A/B)*INT(ABS(A/B))

MOD Operator

The MOD (modulo) operator returns the remainder resulting from a division. Given two values X and Y, X MOD Y is equal to X - Y * INT (X/Y). For example –

12 MOD 5 [EXEC]	2
12 MOD -5 [EXEC]	-3
-12 MOD 5 [EXEC]	3
-12 MOD -5 [EXEC]	-2

Arithmetic Hierarchy

When an expression has more than one arithmetic operation, the order in which the computer performs the operations depends on the following hierarchy –

^	exponentiation	performed first
MOD, DIV, *, /	Modulo, integer divide, multiplication and division	↓
+, −	addition and subtraction	performed last

An expression is scanned from left to right. Each operator is compared to the operator on its right. If the operator to the right has a higher priority, then that operator is compared to the next operator on its right. This continues until an operator of equal or lower priority is encountered: the highest priority operation, or the first of the two equal operations, is performed. Then any lower priority operations on the left are compared to the next operator to the right. This comparison continues until the entire expression is evaluated.

Parentheses can be used to alter the hierarchy just described. When parentheses are used, they take highest priority. When parentheses are nested, like (5*(4-2)), the innermost quantity (4-2) is evaluated first. For example, here's the order of execution in solving the expression –

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

2+3*6/(7-4)^2	multiplication
2+18/(7-4)^2	evaluate parentheses
2+18/3^2	exponentiation
2+18/9	division
2+2	addition
4	result

Whenever you are in doubt as to the order of execution for any expression, use parentheses to indicate the order.

Using parentheses for "implied" multiplication is not allowed. So 4(5 − 2) must appear as 4*(5-2). The operator, *, must be used to specify explicit multiplication.

Output of Numbers

Three formats are available for displaying and printing numbers: standard, fixed point, and floating point (scientific notation). Standard format is automatically set when the machine is switched on. Reset and SCRATCH A also return the computer to standard format when executed. The format can be changed to fixed or floating point by executing the FIXED or FLOAT statements. Executing the STANDARD statement returns the machine to standard format.

All numbers are output with a trailing blank and a leading blank or minus sign.

3

Standard Format

The standard format is convenient for most computations since results appear in an easy-to-read form. Remember, standard format is set at power on and SCRATCH A. To reset standard format after a FIXED or FLOAT statement was executed, execute the STANDARD statement –

```
STANDARD
```

In standard format, all significant digits of a number are output up to a maximum of twelve. For example, 9876543210.12345 is output as 9876543210.12.

Excess zeros to the right of the decimal point are suppressed; for example, 32.100000 would be output as 32.1. Leading zeros are truncated; for example 00223 is output 223.

All numbers whose absolute values are greater than or equal to 1, but less than 10^{12} are output in fixed format showing all significant digits. Numbers between -1 and 1 are also output in fixed format if they can be represented precisely in twelve or fewer digits to the right of the decimal point. All other numbers are output in scientific notation. The form is the same as FLOAT 11. See the FLOAT statement which is discussed later in this chapter.

Fixed Point Format

With fixed point, you can specify the number of digits you want to appear to the right of the decimal point. For example, specifying two digits to the right of the decimal point would be useful for output of dollar and cent values. The FIXED statement sets fixed point format –

FIXED number of digits

The number of digits parameter is a numeric expression and is rounded to an integer to specify the number of digits to the right of the decimal point. Its range is 0 through 12. For example, set FIXED 2 format –

(F)(I)(X)(E)(D)(2)(EXEC) FIXED 2

Now execute 8.7 –

(8)(.)(7)(EXEC) 8.70

But if you set FIXED 0 format –

(F)(I)(X)(E)(D)(0)(EXEC) FIXED 0

and execute 8.7 – 9

Notice that the number is rounded to the specified format. Also notice that the decimal point is suppressed in FIXED 0.

When fixed point is set and the absolute value of the number to be output is greater than or equal to 1E12 or would require more than 17 digits to represent it, the format temporarily reverts to floating point. For example, in FIXED 12, 100 000 is output at 1E+05.

3

3

Floating Point Format

When working with very large or very small numbers, the floating point format is most convenient. The FLOAT statement sets this format –

FLOAT number of digits

The number of digits parameter is a numeric expression and is rounded to an integer to specify the number of digits to the right of the decimal point. Its range is 0 through 11.

A number output in floating point format has the form –

±d.d...dE±dd

- The leftmost non-zero digit of a number is the first digit output. If the number is negative, a minus sign precedes this digit; if the number is positive or zero, a space precedes this digit.
- A decimal point follows the first digit, except in FLOAT 0.
- Some digits may follow the decimal point; the number of digits is determined by the specified floating point format.
- Then the character E appears followed by a plus sign or minus sign and two digits. This is the exponent, representing a positive or negative power of ten. The exponent represents the power of 10 by which the mantissa should be multiplied in order to express the number in fixed point format.

Examples

Here are some numbers and how they are output in various modes –

Number	Standard	FIXED 4	FLOAT 3
15.00	15	15.0000	1.500E+01
.0547∧9	4.38415537301E-12	.0000	4.384E-12
-.000006	-.000006	-.0000	-6.000E-06
2.75327	2.75327	2.7533	2.753E+00
271	271	271.0000	2.710E+02
2.4E78	2.40000000000E+78	2.4E78	2.400E+78

Rounding

A number is rounded before being displayed or printed if there are more digits to the right of the decimal point than the number format allows. In either case, the rounding is performed as follows: The first excess digit on the right is checked. If its value is 5 or greater, the digit to the left is incremented (rounded up) by one; otherwise it is unchanged. In either case, the number remains unchanged internally. For example –

Execute `FIXED 2`	
Execute `A=1.235`	`1.24`
Execute `FIXED 3`	
Now execute `A`	`1.235`

Variables

Algebraic formulas usually contain names which represent an assigned value. These names are known as **variables** and, with the 9835A/B, specify a location in memory where a value is stored. For instance, the formula for the area of a circle, $A = \pi R^2$, uses two variables, A and R. To use the formula, you assign a value to R (radius) to solve for A.

Types

There are four types of variables available with the 9835A/B –

- Full precision numeric variables
- Short precision numeric variables
- Integer precision numeric variables
- String variables (string of characters)

Significant Digits

Significant digits are those which determine the internal accuracy with which a numeric variable is represented. The number format for output has no effect on this. The amount of memory each type of variable uses is discussed in Chapter 5 . All numbers are full precision unless otherwise specified using a `SHORT`, `INTEGER` or `COM` statement; see Chapter 5. Any excess digits input are truncated when the number is stored in memory. For example, 12345678912365 is represented internally with 123456789123 in the mantissa.

Here are the three types of numeric variables –

- Full (real) precision variables are represented internally with twelve significant digits and an exponent in the range −99 through 99.
- Short precision variables are represented internally with six significant digits and an exponent in the range −63 through 63.
- Integer precision variables have no digits following the decimal point. The range of integer precision numbers is −32768 through 32767.

Short and integer precision variables are useful for conserving memory. All calculations are performed with full–precision accuracy, so short and integer precision numbers are converted before and after an operation.

Forms

There are two forms that any type of variable may have. (See Chapter 5 for more information.)

- Simple (Nonsubscripted)
- Array

Names

All variables must have a name. Names must follow these rules –

- A name has between 1 and 15 characters.
- The first character must be a capital letter.
- The remaining characters must be lowercase letters, digits, or the underscore character obtained by pressing (SHIFT) (6).
- String names must be followed by $ (dollar sign).

Here are some examples of variable names –

```
X
Name$
Address$
Social_security
Acct_number
Owed
Payment
Interest
```

Any name can be used simultaneously for a simple numeric, simple string, numeric array and string array.

Simple Numerics

This section introduces you to using simple numeric variables from the keyboard. String and array variables are fully described in Chapter 5.

Variables are assigned values using an equals sign to create an assignment statement. For example, to assign 150 to Owed and 25 to Payment, enter –

```
Owed=150 (EXEC)
Payment=25 (EXEC)
```

Now that some variables have assigned values, they can be used in place of numbers in math calculations –

```
Owed=Owed-Payment (EXEC)                125
Interest=Owed*.06 (EXEC)                7.5
```

To check the current value of a variable, just type in its name, then press (EXEC). For example –

```
Payment (EXEC)               25
```

If you do this while a program is running (see Live Keyboard, Chapter 2), you may get an unexpected answer if execution is currently in a subprogram.

Relational Operators

Relational operators are used to determine the value relationship between two expressions. This can be especially useful for program branching if a specified condition is true. See the IF statement in Chapter 8.

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; it is listed < >)

The result of a relational operation is either a 1 (if the relation is true) or a 0 (if it is false). Thus, if A is less than B, then the relational expression A <=B is true and results in a value of 1. All comparisons are made on all significant digits, signs and exponents.

The equals sign is also used in the assignment statement, as shown earlier in the chapter. In an assignment statement, the variable is to the left of the equals sign, the value is to the right. If the equals sign is used in such a way that it might be either an assignment or relational operation, the computer assumes that it is an assignment operation. For example, X=Y=Z assigns the value of Z to X and Y. X=(Y=Z) assigns the result of the operation Y=Z to X.

Examples

Here are some examples of relational operations. First let's assign values to the variables A and B. Execute –

```
A = 1
B = 2
```

Now execute these relational operations –

A < B	1 (true)
B < A	0 (false)
4 = A	0 (false)
C = (A = B)	Assigns the value 0 to C

Logical Operators

The logical operators AND, OR, EXOR (exclusive or) and NOT are useful for evaluating Boolean expressions. Any value other than 0 (false) is evaluated as true. The result of a logical operation is either 0 or 1. Logical operators are especially useful in determining whether or not certain sets of conditions are true. See the IF statement in Chapter 8.

AND Operator

numeric expression AND numeric expression

AND compares two expressions. If both expressions are true, the result is true (1). If one or both of the expressions is false, the result is false (0).

OR Operator

numeric expression OR numeric expression

OR compares two expressions. If one or both of the expressions is true, the result is true (1). If neither expression is true, the result is false (0).

EXOR Operator

numeric expression EXOR numeric expression

EXOR (exclusive or) compares two expressions. If only one of the expressions is true, the result is true (1). If both are true, or both are false, the result is false (0).

NOT Operator

NOT numeric expression

NOT returns the opposite of the logical value of an expression. If the expression is true (non-zero), the result is false. If the expression is false (zero), the result is true (1).

The expressions used with logical operators can be either relational or non-relational. If the expression is relational (like A < B), its true or false designation is determined by the particular relational value. If the expression is non-relational (like A), it is true if its arithmetic value is any value other than 0; it is false if its arithmetic value equals 0.

Examples

Here are some examples of logical operations. First assign values to X and Y. Execute –

```
X = 0
Y = 2
```

Now execute these logical operations –

`X AND Y`	0 (false)
`X EXOR Y=2`	1 (true)
`NOT X`	1 (true)
`NOT X OR NOT Y`	1 (true)

Here's a truth table summarizing logical operations –

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

Math Functions and Statements

Math functions available on the 9835A/B are explained in this section. Parentheses must enclose the numeric expression used as the argument of the function if it contains any operators. For example, SINA+B does not equal SIN(A+B). Parentheses enclose the expression when listed. Examples of two functions are combined in some cases.

General Functions

`ABS` numeric expression — Returns the absolute value of the expression.

```
  10   REM  THIS EXAMPLE SHOWS ABS FUNCTION
  20   INPUT "PRINTER SELECT CODE?",A
  30   IF SGN(A)=1 THEN 60
  40   PRINT "SELECT CODE CAN'T BE NEGATIVE"
• 50   PRINT "ABSOLUTE VALUE- ";ABS(A);" -WILL BE USED"
• 60   PRINTER IS ABS(A)
  70   END

SELECT CODE CAN'T BE NEGATIVE
ABSOLUTE VALUE-  6  -WILL BE USED
```

DROUND (numeric expression, number of significant digits)

The digit round function returns the numeric expression rounded to the specified number of significant digits. The number of significant digits parameter is rounded to an integer. If the specified number of digits is greater than 12, no rounding takes place. If it is less than one, 0 is returned. DROUND is useful for checking equality to a specified number of digits.

```
   10   INPUT A,B
 • 20   IF DROUND(A,4)=DROUND(B,4) THEN GOSUB 60
 • 30   PRINT "A=";A,"DROUND (A,4)=";DROUND(A,4)
 • 40   PRINT "B=";B,"DROUND (B,4)=";DROUND(B,4)
   50   STOP
   60   PRINT "A AND B ARE EQUAL TO 4 SIGNIFICANT DIGITS"
   70   RETURN
   80   END
```

```
A AND B ARE EQUAL TO 4 SIGNIFICANT DIGITS
A= 12345                 DROUND (A,4)= 12350
B= 12346                 DROUND (B,4)= 12350
```

FRACT numeric expression

Returns the fractional part of the evaluated expression. It is defined by this formula: expression − INT expression.

INT numeric expression

The integer function returns the greatest integer which is less than or equal to the evaluated expression.

```
   10   REM  THIS PROGRAM SHOWS FRACT AND INT
   20   A=22.987
   30   B=-6.257
   40   PRINT "VALUE";TAB(12);"FRACT";TAB(24);"INT"
 • 50   PRINT A;TAB(12);FRACT(A);TAB(24);INT(A)
 • 60   PRINT B;TAB(12);FRACT(B);TAB(24);INT(B)
   70   END
```

```
VALUE       FRACT       INT
 22.987      .987        22
-6.257       .743       -7
```

MAX (list of numeric expressions)

The maximum function returns the greatest value in the list.

MIN (list of numeric expressions) The minimum function returns the smallest value in the list.

```
  10   REM  THIS PROGRAM SHOWS MAX AND MIN
  20   INPUT "FOUR VALUES",A,B,C,D
  30   PRINT "FOUR VALUES:";A;B;C;D
• 40   PRINT "MAXIMUM:";MAX(A,B,C,D)
• 50   PRINT "MINIMUM:";MIN(A,B,C,D)
  60   END

  FOUR VALUES: 1  5  8  9
  MAXIMUM: 9
  MINIMUM: 1
```

3

PI Returns the value of π. It is represented internally as 3.1415926536.

```
• 10   PRINT "PI=";PI
  20   DEG
  30   INPUT "ANGLE IN DEGREES?",A
  40   PRINT A;" DEGREES =";
• 50   A=A*PI/180     ! CONVERT DEGREES TO RADIANS
  60   PRINT A;" RADIANS"
  70   END

  PI= 3.1415926536
   45  DEGREES = .7853981634  RADIANS
```

PROUND (numeric expression, power-of-ten position) The power-of-ten round function returns the numeric expression rounded to the specified power-of-ten position. Specifying −2 is useful for output of money values.

```
  10   A=127.455
  11   PRINT "ORIGINAL NUMBER:",A
  20   FOR I=-2 TO 3
• 30      PRINT "POWER OF TEN:";I,PROUND(A,I)
  40   NEXT I
  50   END


  ORIGINAL NUMBER:      127.455
  POWER OF TEN:-2       127.46
  POWER OF TEN:-1       127.5
  POWER OF TEN: 0       127
  POWER OF TEN: 1       130
  POWER OF TEN: 2       100
  POWER OF TEN: 3       0
```

RES — Returns the result of the last numeric computation which was executed from the keyboard.

RND — The random number function returns a pseudo random number greater than or equal to 0 and less than 1. The random number is based on a seed set to $\pi/180$ at power on, reset, SCRATCH A, and RUN. Each succeeding use of RND returns a random number which uses the previous one as a seed. The seed can be modified using the RANDOMIZE statement which is described at the end of Chapter 4.


```
  10   REM  EXAMPLE SHOWING RND FUNCTION
• 20   PRINT RND       ! ALWAYS SET TO SAME SEED AT RUN
• 30   PRINT RND*5
  40   RANDOMIZE       ! SCRAMBLES SEED
• 50   PRINT RND
  60   END

   .678219009345
   1.91093429525
   .542190093254
```

SGN numeric expression — The sign function returns a 1 if the expression is positive, 0 if it is 0 and −1 if it is negative.

SQR numeric expression — The square root function returns the square root of a non-negative expression.

```
  10   REM  THIS PROGRAM SHOWS ABS,SGN AND SQR
  20   INPUT "ANY NUMBER",A
• 30   IF SGN(A)=-1 THEN 60          ! Branch when negative
• 40   PRINT "NUMBER: ";A,"SQUARE ROOT: ";SQR(A)
  50   STOP
  60   PRINT "NUMBER IS NEGATIVE, USE ABSOLUTE VALUE"
  70   PRINT "NUMBER: ";A
• 80   PRINT "SQUARE ROOT OF ABSOLUTE VALUE:";SQR(ABS(A))
  90   END

 NUMBER IS NEGATIVE, USE ABSOLUTE VALUE
 NUMBER: -35
 SQUARE ROOT OF ABSOLUTE VALUE: 5.91607978309
```

Logarithmic and Exponential Functions

EXP numeric expression — The exponential function returns the value of the constant Napierian e (= 2.71828182846 to twelve place accuracy) raised to the power of the computed expression.

LGT numeric expression — The common log function returns the logarithm (base 10) of a positive valued expression.

LOG numeric expression — The natural log function returns the logarithm (base e) of a positive valued expression.

3

```
  10   REM  THIS PROGRAMS SHOWS EXP, LGT, AND LOG
  20   A=1
  30   B=7
  40   PRINT "NUMBER";TAB(10),"EXP";TAB(27),"LGT";TAB(44),"LOG"
• 50   PRINT A;TAB(10),EXP(A);TAB(27),LGT(A);TAB(44),LOG(A)
• 60   PRINT B;TAB(10),EXP(B);TAB(27),LGT(B);TAB(44),LOG(B)
  70   END

NUMBER   EXP                LGT                LOG
 1        2.71828182844      0                  0
 7        1096.63315844      .845098040013      1.94591014905
```

Trignometric Functions and Statements

The trigonometric functions use the angular unit mode: degrees, radians, or grads, which is currently set. A trigonometric statement is used to set the angular unit mode.

Radian mode is automatically set at power on, or when SCRATCH A, RUN, or reset is executed.

Degree Mode

To set degree mode, execute –

```
DEG
```

A degree is 1/360th of a circle.

Grad Mode

To set grad mode, execute –

```
GRAD
```

A grad is 1/400th of a circle.

Radian Mode

To reset radian mode, execute –

```
RAD
```

There are 2π radians in a circle.

Functions

`ACS` numeric expression	Returns the principal value of the arccosine of the expression in current angular units. The expression must be in the range −1 through +1.
`ASN` numeric expression	Returns the principal value of the arcsine of the expression in current angular units. The expression must be in the range −1 through +1.
`ATN` numeric expression	Returns the principal value of the arctangent of the expression in current angular units.
`COS` numeric expression	Returns the cosine of the angle represented by the expression in current angular units.
`SIN` numeric expression	Returns the sine of the angle represented by the expression in current angular units.
`TAN` numeric expression	Returns the tangent of the angle represented by the expression in current angular units.


```
  10   PRINT "ANGLE: 60"
  20   FIXED 4
  30   PRINT SPA(22);"SIN";SPA(7),"COS";SPA(7),"TAN";SPA(7),"ASN"
  40   FOR I=1 TO 3
  50      ON I GOSUB Degrees,Radians,Grads
• 60      PRINT SIN(60);TAB(30),COS(60);TAB(40),TAN(60);TAB(50);
• 70      S=SIN(60)
• 80      PRINT ASN(S)
  90   NEXT I
  100  STOP
• 110 Degrees:  DEG        ! SET DEGREE MODE
  120  PRINT "DEGREES:",
  130  RETURN
  140 Radians:  RAD        ! SET RADIAN MODE
• 150  PRINT "RADIANS:",
  160  RETURN
• 170 Grads:  GRAD         ! SET GRAD MODE
  180  PRINT "GRADS:",
  190  RETURN
  200  END
```

```
ANGLE: 60
                          SIN       COS        TAN        ASN
DEGREES:                 .8660     .5000     1.7321     60.0000
RADIANS:                -.3048    -.9524      .3200     -.3097
GRADS:                   .8090     .5878     1.3764     60.0000
```

Total Math Hierarchy

The order of execution for all mathematical operations is shown here.

Highest Priority	( ) parentheses
	Functions
	^ (exponentiation)
	NOT, − (unary minus)
	*, /, MOD, DIV
	+, −
	All relational operators (=, <, >, >=, <=, <> or #)
↓	AND
Lowest Priority	OR, EXOR

3

Remember that the order of execution for operations of the same priority level is from left to right, except when parentheses are used; operations within parentheses are executed first.

Math Errors-Recovery

Many math errors occur due to an improper argument or overflow; if a program is running, execution halts. It is possible to make some of these errors non-fatal so that execution doesn't halt by providing a default value for the number which is out of range. The default values are enabled by executing the DEFAULT ON statement –

```
DEFAULT ON
```

The errors and default values are –

Error (Number)	Default Value
Integer precision overflow (20)	32767 or −32768
Short precision overflow (21)	+ or − 9.99999E63
Full precision overflow (22)	+ or − 9.99999999999E99
Intermediate result overflow (23)	+ or −9.99999999999E511
TAN(N * PI/2), N:odd integer (24)	9.99999999999E511
Zero to negative power (26)	9.99999999999E511
LGT or LOG of zero (29)	−9.99999999999E511
Division by zero (31)	+ or − 9.99999999999E511
X MOD Y , Y=0 (31)	0

Using default values may alter the results of computations; be aware of this when using them.

Default values are disabled by executing the DEFAULT OFF statement –

```
DEFAULT OFF
```

DEFAULT OFF is set at power on, reset, SCRATCH A, SCRATCH P, SCRATCH C and SCRATCH V.

3

Chapter 4
Programming Information

Introduction

An enhanced form of the BASIC language is used in the 9835A/B. This chapter discusses the fundamentals of programming as they relate to the computer and to program control. If you are unfamiliar with programming, the Beginner's Guide contains fundamental information on how to program the 9835A/B using the BASIC language.

Conventions and Terms

The following conventions and terms are used in the statement and command descriptions found in this manual.

dot matrix – All items in dot matrix must appear exactly as shown.

[ ] – Items within square brackets are optional unless the brackets are in dot matrix.

... – Three dots indicate that the previous item can be repeated.

| – A vertical line between two parameters means "or"; only one of the two parameters can be included.

/ – A slash between two parameters means "and/or"; either or both of the parameters can be included.

Statement – Statements are instructions to the computer telling it what to do while a program is running. A statement can be preceded by a line number, stored and executed from a program. Most statements can also be executed from the keyboard without a line number.

Command – A command is also an instruction to the computer which is executed from the keyboard. Commands are executed immediately, do not have line numbers and can't be used in a program. They are used to manipulate programs and for utility purposes, such as listing key definitions.

Constant – A constant is a fixed numeric value within the range of the 9835A/B; for example `29.5` or `2E12`.

Character – A letter, number, symbol or ASCII control code; any arbitrary 8-bit byte defined by the `CHR$` function.

Text – Any combination of characters; for example `"ABC"`. Text can be quoted (literal) or unquoted.

Name – A capital letter followed by 0 through 14 lowercase letters, digits or the underscore character. Names are used for variable names, labels, function names, and subprograms.

Line number – An integer from 1 through 9999. In most cases, when a line number is specified, but is not in memory, the next highest line is accessed.

4

Label – A unique name given to a program line. It follows the line number and is followed by a colon.

Line Identifier – A program line can be identified either by its line number (`GOTO 150`) or its label, if any (`GOTO Routine`).

Program Segment – The main program and each subprogram are known as program segments.

Numeric Expression – A numeric expression is a logical combination of variables, constants, operators, functions, including user-defined functions, grouped within parentheses if needed.

Select Code – An expression (rounded to an integer) in the range zero through sixteen which specifies a setting on an interface card to an I/O device. The following select codes are reserved and can't be set on an interface –

- 0 Optional internal thermal printer
- 15 Tape cartridge
- 16 CRT (9835A only); Optional Internal printer (9835B)

HP-IB Device Address – An expression which specifies the HP-IB address that is set on a device. Its range is 0 through 30.

Programming Fundamentals

A program is a set of instructions to the computer – an organized set of statements. It is ordered by line numbers; each statement in a program **must** be preceded by a unique line number. Remember these points while writing and entering programs –

- Line numbers are arranged in ascending order. However, you can type main program lines in any order because lines are automatically sorted as they are stored. Line numbers 1 through 9999 are allowed. Methods for entering line numbers are covered in the section on Line Numbering later in this chapter.

- Each line number can be followed by a unique **label** – this is optional. A label is a name and must be followed by a colon. For example –

```
50  Show_results: PRINT A,B,C
```

  Show_results is the label.

- The line number and the label are both known as the **line identifiers** for that line. In branching to line 50 above, both –

```
100  GOTO 50
```

  and

```
100  GOTO Show_results
```

  would accomplish the same thing.

- Program lines can be up to 160 characters long including the line number. After each line is typed in, it is entered into memory by pressing (STORE). Pressing (STORE) also causes the line to be checked for syntax errors before it is stored.

- Normal program execution proceeds from the lowest-numbered line to the highest-numbered line. The order of execution can be altered, however; see Chapter 8 on Branching.

- Space dependent mode (covered later in this chapter) and edit line mode (covered in Chapter 12) can be used to make program entering easier. Refer to those topics for more information.

4

Program Pointer

While a program is running, an internal program pointer monitors which line is being executed.

Statements

Programs are made up of statements. Statements are instructions to the computer that can be assigned a line number, stored, and executed from a program. Each statement contains one or more **keywords** which have a special meaning in the BASIC language. They identify operations to be performed or the type of information contained in a statement. Here are some examples of keywords –

```
PRINT
BEEP
FOR
NEXT
DIM
DEF FN
SAVE
IF
```
THEN (secondary keyword)

Most statements can also be executed from the keyboard without a line number. Exceptions are noted.

Statements are either **executable** or **declaratory**. A declaratory statement is part of a program and is used to give the computer information it will need to execute other statements in the program. Following is a list of declaratory statements. Each statement is described later in this manual.

```
COM
DATA
DEF FN
DIM
END
FN END
IMAGE
INTEGER
OPTION BASE
REAL
REM
SHORT
SUB
SUBEND
```

Spacing

In general, spacing between characters is arbitrary; the computer automatically sets proper spacing into each line as it is stored into memory. Only in text, REM statements, comments, and blanks after line numbers and labels does spacing remain exactly as input. These blanks allow lines to be indented.

Space Dependent Mode

The space dependent mode is very useful for keying in a program that has long variable names. It causes spaces, or lack of them, between parts of a statement to become significant when entering program lines. In space dependent mode, variables, subprogram names and labels can be typed in all capital letters or in any combination of upper and lower case, as long as the first letter is upper case. Keywords **must** be separated from other parts of the statement by one or more blanks or by a delimiter like a comma or a #.

Here are some rules to follow when entering programs in space dependent mode –

- Any variable name that is the same as a secondary keyword (e.g. function, logical operator, THEN) **cannot** be entered in all capital letters.

- The label of a line that is the same as any keyword **cannot** be entered in all capital letters. However, when referenced, as in a GOTO statement, it can be entered in all capital letters, except after THEN.

- The first variable in an implied LET statement **cannot** be entered in all capital letters if it is the same as a keyword. This is also the case if the implied LET follows THEN.

Example

For example, in space dependent mode, trying to store –

```
10FORI=1TO10
```

gives an IMPROPER EXPRESSION message with the flashing cursor under the T. The computer is interpreting this as an assignment statement assigning the value 1 to the variable Fori.

4

Space dependent mode is entered by holding down –

CONT'L

then pressing –

SPC DEP
TYPW'R

The words SPACE DEPENDENT appear on the right hand side of the system comments line of the 9835A. On the 9835B, Space dependent mode is displayed.

When a program is listed after it was typed in space dependent mode, all names are converted to their normal spelling: capital letter followed by lower case.

4

To exit the space dependent mode, hold down –

CONT'L

then press -

SPC DEP
TYPW'R

again.

Example

Here is an example of how a program line may be typed in normal and space dependent modes –

Normal Mode –

```
10IFOutcome>PredictionTHENPRINTOutcome,Difference
```

Space Dependent Mode –

```
10 IF OUTCOME > PREDICTION THEN PRINT OUTCOME, DIFFERENCE
```

Both list identically –

```
10 IF Outcome>Prediction THEN PRINT Outcome,Difference
```

Space dependent and typewriter modes are mutually exclusive – if one is entered while the other is in effect, the new one cancels the old.

Remarks

Many times you may want to insert comments in order to make your program logic easier to follow. This can be done by using the REM (remark) statement or the comment delimiter !.

The REM Statement

REM [any combination of characters]

Remarks can be used to explain program lines or set off program segments. For example –

```
100 Remark:   REM          A LABEL CAN PRECEDE REM
110  REM  This section outputs all data
120  REM
430  REM  YOU CAN SAY ANYTHING YOU WANT IN A REMARK STATEMENT
```

4

Comment Delimiter

!, the comment delimiter, can be anywhere in a program line after the line number. If it is immediately following the line number, it is just like a REM statement. **All** characters following a ! are considered part of a comment unless the ! is within quotes. The comment delimiter can also follow a command.

In this way, program lines and commands can contain comments. For example –

```
10   INPUT X,Y            ! REQUEST VALUES
20   IF X>Y THEN 40       ! BRANCH IF X>Y
30   STOP                 ! EXECUTION STOPS
40   PRINT X,Y            ! OUTPUT VALUES
50   REM                  ! TAB KEY GOOD FOR LINING UP COMMENTS
60   END
```

Line Numbering

There are three methods that can be used to enter line numbers. The first is to manually type in the line number before the statement. A second method is to use edit line mode (see Chapter 12) to generate numbers as lines are stored. The third way is to use the AUTO command. A program can also be renumbered.

Auto Numbering

The AUTO command allows lines to be numbered automatically as they are entered and stored. This saves you from having to type the line number each time you key in a statement.

AUTO [beginning line number [, increment value]]

If neither parameter is specified, executing AUTO causes line numbering to begin with 10 and to be incremented by 10 as lines are stored. If only the beginning line number is specified, the increment between line numbers is 10. Both the line number and the increment values must be positive integers. Here are some examples –

```
AUTO              ! BEGIN WITH 10, INCREMENT BY 10
AUTO 100          ! BEGIN WITH 100, INCREMENT BY 10
AUTO 5,5          ! BEGIN WITH 5, INCREMENT BY 5
```

Renumbering

REN [beginning line number [, increment value]]

The renumber command causes the program in memory to be renumbered. This allows you to insert lines or add more lines at the end. If no parameters are specified, the program is renumbered so that line numbering begins with 10 and is incremented by 10. If only the beginning line number is specified, the increment is 10. Here are some examples –

```
REN             ! BEGIN WITH 10, INCREMENT BY 10
REN 200         ! BEGIN WITH 200, INCREMENT BY 10.
REN 5,20        ! BEGIN WITH 5, INCREMENT BY 20
```

When a program is renumbered, all line references (GOTO 50, for example) in the program are automatically adjusted to reflect the new line numbers.

Listing the Program

The LIST command is used to obtain a printed listing of the program or section of the program in memory. The listing is output on the device specified as the standard printer (see PRINTER IS, Chapter 10).

LIST [beginning line identifier [, ending line identifier]]

If no parameters are specified, the entire program is listed. If one line identifier is specified, the program is listed from that line to the end. If two line identifiers are specified, that segment of the program, including beginning and ending lines, is listed.

Here are some examples –

```
LIST                    ! Lists the entire program
LIST 50                 ! Lists program beginning with line 50
LIST 200,250            ! Lists lines 200 through 250
```

Available Memory

When the listing is complete, the amount of unused memory available for use is displayed in the system comments line. So if you execute –

```
SCRATCH A
```

then –

```
LIST
```

the number that is displayed is the total memory available for your use. This memory is expressed in bytes.

Alternate Printing Devices

The LIST command can be directed to a device other than the standard printer by specifying the select code of the alternate device.

LIST# select code [, HP-IB device address] [; beginning line identifier [, ending line identifier]]

Here are some examples –

```
LIST #16         ! LIST TO CRT
LIST #0          ! LIST TO STRIP PRINTER
LIST #6          ! LIST TO SELECT CODE 6
LIST #7,2        ! LIST TO HP-IB PRINTER
LIST #6;50       ! LIST TO SELECT CODE 6--LINE 50 ON
LIST #6;15,55    ! LIST TO SELECT CODE 6--LINES 15-55
```

4

Program Control Operations

Running A Program

Program execution can be started by pressing –

(RUN)

or executing the RUN command –

RUN [line identifier]

The line identifier must be in the main program and specifies that execution is to begin at that line; if no line is specified, execution begins with the first line in memory.

Here are some examples –

```
RUN              ! Begin at lowest-numbered line
RUN 150          ! Begin at line 150
RUN Routine      ! Begin at line labeled Routine
```

RUN causes a short pre-run initialization to occur which clears or resets the following items –

- Variables
- Files table (see Chapter 11)
- DATA pointers (see Chapter 5)
- Subroutine return pointers (see Chapter 8)
- ON KEY and ON END (see Chapters 8 and 11)
- Radian mode (see Chapter 3)
- Random number seed (see Chapter 3)
- ERRL and ERRN (see Chapter 12)
- ENABLE (see Chapter 4)

During the pre-run initialization, doubly defined labels and statements defined in ROMs which aren't present are detected and a warning message is given. However, functions defined in ROMs which aren't present are not detected.

After the pre-run phase, the program is executed.

4

Stepping Through A Program

A program can also be run or continued by pressing –

STEP

When STEP is used, the program is executed one line at a time as STEP is pressed. The next line to be executed is displayed in the system comments line. When using STEP to run a program from the beginning, a pre-run initialization takes place the first time it is pressed. Pressing STEP a second time executes the first program line.

Pausing Execution

Execution can be suspended by pressing –

PAUSE

The current line is completed and the program is halted at the next line to be executed; this line is displayed in the system comments line. Any current I/O operation is completed.

A pause can also be programmed using the PAUSE statement. A useful application is to program a pause so that intermediate results can be checked and execution resumed.

PAUSE

The PAUSE statement can't be executed from the keyboard.

Continuing Execution

Program execution can be resumed where it was halted by pressing –

CONT

or executing the CONT (continue) command –

CONT [line identifier]

The line identifier causes execution to resume at the specified line. If it is a line number that is not in memory, execution resumes with the next highest numbered line. CONT can also be used to start a program that was just run. No pre-run initialization takes place.

Execution of a paused program can also be restarted at the beginning with RUN or RUN.

Terminating Execution

All programs have a logical as well as a physical end. The logical end is that point where all statements have been executed the desired number of times and the program has completed the task for which it was designed. The physical end (highest-numbered line) of a program is the last (highest-numbered) line.

Program execution can also be halted before it is done by pressing –

STOP

When STOP is pressed, all I/O operations are aborted and data may be lost. The program pointer is reset to the first line of the main program.

The STOP Statement

The STOP statement can be used to indicate the logical, rather than the physical end of a program. It's purpose is to tell the computer to terminate execution of the program and reset the program pointer. It may appear at any point in the program. Some programs have several logical ends and so require several STOP statements.

4

```
STOP
```

The STOP statement can't be executed from the keyboard.

The END Statement

The physical end (highest-numbered line) of a main program is indicated by the END statement. END also terminates program execution. It is not mandatory to have an END statement as it is in other BASIC systems; however, it is good programming practice.

```
END
```

Reset

The reset operation (see Chapter 3) can also be used to stop a running program. All I/O operations are aborted and data may be lost.

It is also possible that the program and data can be destroyed just as if SCRATCH A had been executed. Therefore reset should not be used for stopping a program unless pressing STOP fails to halt the program.

Erasing Memory

The SCRATCH command is used to erase all or parts of memory; it can be used to erase programs, variables, keys, or the entire memory. k11 SCRATCH is defined as a typing aid for SCRATCH at power on and SCRATCH A.

SCRATCH [P or V or C or KEY [key number] or kn or A]

Command	Operation
SCRATCH	Erases program including DATA pointers.
SCRATCH A	Erases the entire memory. See the Reset table in Appendix E.
SCRATCH C	Erases the values of all variables, including those in common.
SCRATCH P	Erases the program, variables, binary routines, DATA pointer and the files table.
SCRATCH KEY [key number]	Erases one or all SFK typing aid definitions (but not control features).
SCRATCH V	Erases the values of all variables except those in common.
SCRATCH kn	Erases the typing aid definition of the specified SFK.

4

Interrupting A Program

Normal program execution can be interrupted by conditions specified by ON INT (see the I/O ROM Manual), ON ERROR, ON KEY, and ON END (which are all discussed later in this manual). All ON declarations are enabled at power on and SCRATCH A.

Priority

Priority determines whether a program can be interrupted. At power on, the priority of the system is set to 0. Operations then assume this priority. An operation declared by an ON declarative can be specified as having a higher priority and thus interrupt another operation that has a lower priority. ON KEY, ON ERROR, ON INT, and ON END all have differing effects on priority levels. See those statements for information.

Disabling Declaratives

Any ON KEY and ON INT declaratives are deactivated by executing the DISABLE statement –

DISABLE

One ON KEY interrupt per key and one ON INT interrupt per select code can be logged, but the interrupt routine is not executed until declaratives are re-enabled.

Enabling Declaratives

ON KEY and ON INT declaratives are re-enabled by executing the ENABLE statement –

ENABLE

Miscellaneous Statements

4

The WAIT Statement

The WAIT statement is used to program a delay between the execution of two program statements –

WAIT number of milliseconds

The number of milliseconds is a numeric expression rounded to an integer in the range −32 768 through 32 767. A negative number defaults to a wait of zero. The delay specified by WAIT is not totally accurate. The degree of accuracy is dependent on what is displayed on the CRT. A blank CRT enables the delay to be correct to wristwatch accuracy. When a 30 second delay is specified and the CRT is totally full, the delay ranges between 35 and 40 seconds. Thus, the fuller the CRT, the less the accuracy.

The wait operation can only be interrupted by reset.

Scrambling the Random Number Seed

The random number seed can be re-evaluated by executing the RANDOMIZE statement –

RANDOMIZE [numeric expression]

If the value of the expression is an integer, the value of the seed is set to 0 causing RND to return 0 each time it is used. To obtain a good seed, the expression should have as many digits to the right of the decimal point as possible. A 1, 3, 7 or 9 is the most effective final digit. If no expression is specified, the computer arbitrarily resets the seed to one of 116 possible points.

The SECURE Statement

The SECURE statement is used to prevent selected program lines from being listed; instead, an asterisk appears after the line number. The secured lines execute normally, however.

SECURE [line identifier [, line identifier]]

If no line identifiers are specified, the entire program is secured. If one line identifier is specified, only that line is secured. Two line identifiers secure that block of lines, including the beginning and ending lines.

For example –

```
SECURE                  ! Secures all program lines
SECURE Formula          ! Secures the line labeled Formula
SECURE 100,150          ! Secures lines 100 through 150
```

4

There is no provision made for "unsecuring" a program. However, a secured line can be deleted or replaced, and can be listed after that.

A program protected with SECURE can be reproduced onto a mass storage medium using STORE, but not using SAVE.

Typewriter Mode

TYPW'R can be "pressed" from within a program to set the keyboard to typewriter mode, thus making input easier. This is done by executing the TYPEWRITER ON statement –

```
TYPEWRITER ON
```

When this statement is executed, the keyboard behaves just as if TYPW'R had been pressed.

Typewriter mode can be turned off from within a program by executing the TYPEWRITER OFF statement –

```
TYPEWRITER OFF
```

Conserving Memory

Large programs that involve large amounts of data can sometimes require more memory than is available for use. This section presents some ways to conserve memory usage when writing a program and using data.

One way to use less memory in a program is to limit the use of REM statements and comments in the program. This limits program readability and documentation, but does conserve memory usage.

The use of subprograms can also conserve memory usage. Variables used within subprograms either share memory space with calling program variables or use memory only temporarily. So rather than creating new variables for various routines, thus using more memory, a subprogram can be used. In addition, the use of many short program segments results in better memory packing efficiency than a few large segments. See Appendix F for more information.

The use of SHORT and INTEGER precision variables, rather than full precision, is a very good way to conserve memory in a program that has a great deal of data. This technique is most useful when dealing with large arrays. However, this technique has two limitations. All calculations are performed with full-precision accuracy, so INTEGER and SHORT precision variables must be converted before and after the operation. This slows down execution. Another limitation can arise when inverting a matrix that is not full precision; the results will almost never be entirely accurate due to rounding errors during calculation.

A fourth way to conserve memory is to break a program down into several sections and SAVE each section into a different file. This is known as **overlaying**. Each section can be brought into memory using LINK. This operation preserves the values of variables, but erases each section of the program as another one is linked in.

Chapter 5
Using Variables

Introduction

There are four types of variables available with the 9835A/B: full (real) precision numeric, short precision numeric, integer precision numeric, and string. Each type can have two forms: simple (non-subscripted) and array. All variables must have a name. Additional information about variables can be found in Chapter 3.

The following topics and statements are covered in this chapter –

- LET
- Array variables
- String variables – simple and array
- String expressions
- OPTION BASE
- DIM
- INTEGER
- SHORT
- REAL
- COM
- READ, MAT READ, DATA and RESTORE
- INPUT
- MAT INPUT
- LINPUT
- EDIT

Terms

The following terms are used in the syntax descriptions in this chapter –

variable – a name which is assigned a value and specifies a location in memory. Variables can be classified into various categories and subsets of the categories as shown in the diagram below. For example, any reference to a single numeric variable includes simple numerics and elements of numeric arrays. –

- variable
  - single variable
    - numeric variable
      - simple numeric
      - array element
    - string variable
      - simple string
        - substring
      - string array element
        - substring
  - array variable
    - numeric array
      - matrix
      - vector
    - string array
      - matrix
      - vector

name – a capital letter followed by 0 through 14 lowercase letters, digits, or the underscore character.

The LET Statement

Any simple numeric variable can be assigned a value using the LET statement –

[LET] simple variable [= simple variable...] = numeric expression

Implied LET

Omitting LET is an implied LET or implied assignment.

Examples

```
10   X=12                    ! Assign 12 to X
20   LET Y=3*4               ! Assign 12 to Y--use any expression
30   Data1=32.17             ! Assign 32.17 to Data 1
40   A=B=C=0                 ! Assign 0 to A,B & C--multiple assign
50   Counter=Counter+1       ! Adds 1 to the value of Counter
60   LET M=X                 ! Assign the value of X to M
70   STOP
```

If a numeric variable is used in a computation and hasn't been assigned a value, 0 is used as its value.

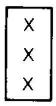
To check the current value of a variable, type in its name, then press (EXEC). This can also be done while a program is running in live keyboard mode. You may get an unexpected result if execution is currently in a subprogram and the variable isn't defined in the subprogram.

The values of non-subscripted (simple) variables are erased by executing SCRATCH V, SCRATCH C, SCRATCH A, or SCRATCH P.

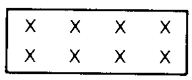
Array Variables

An array variable (array) is a collection of data items of the same type. An array can have one to six dimensions and up to 32 767 elements. It is a convenient tool for handling large groups of data within a program.

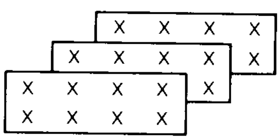
A one-dimensional array (also known as a **vector**[1]) can be thought of as a column of items. The following represents a vector having three items; X represents one item.



The structure of a two-dimensional array (also known as a **matrix**[1]) is rows and columns. Here is a representation of a 2 by 4 (2x4) array.



The structure of a three-dimensional array can be though of as a series of two-dimensional arrays. Here is a representation of a 3 by 2 by 4 array. The 9835A/B interprets it as three 2 by 4 arrays.



The structure of arrays with more dimensions is conceptual and hard to visualize and thus is left to your imagination. These arrays can be useful for structuring data.

1 Vectors and matrices are special types of arrays. Any reference in this manual to an array also includes matrices and vectors.

Explicit Definition

An array can initially be defined in a variable declarative statement (DIM, COM, REAL, SHORT, or INTEGER). There, its maximum size is specified by placing **subscripts** in parentheses after the name. This is known as **dimensioning** the array. The subscripts specify the number of dimensions and upper bound of each dimension.

When an array is dimensioned, its physical or **maximum size** is defined. The **working size** of an array is the **total** amount of elements being used. A new working size can be specified in a REDIM statement or in certain array operation statements (see Chapter 7). The new working size can't have more elements than the maximum size.

Subscripts

Subscripts are integers separated by commas and enclosed in parentheses. The range of each subscript is −32 767 through 32 767, but the size of an array is limited to no more than 32 767 elements by memory size. For example –

```
10  DIM M(2,3)
```


specifies an array with 2 dimensions, upper bounds of 2 and 3 for a total of 12 elements. The lower bound for each dimension is zero. The OPTION BASE statement can be used to change it to one; OPTION BASE is covered later in this chapter.

Here is a representation of array M –

M(2x3)

	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)

Subscripts can also be used to specify the lower as well as upper bound of each dimension. An array the size of array M above could also be specified –

```
10 DIM M(-1:1,-2:1)
```

The upper and lower bounds are separated by a colon.

Implicit Definition

If an array element (discussed next) is used in a program or keyboard computation, but the array has not been defined in a variable declarative statement, the array is then **implicitly dimensioned**. This means that an array is dimensioned having the number of dimensions indicated by the array element. The upper bound of each dimension is 10; the lower bound is 0 or 1, depending on the current `OPTION BASE` setting.

Array Elements

Each **element** in the array can also be referenced by using subscripts and used like a simple variable. An array element is a type of single variable. Thus –

M(1,2)

refers to an element in array M and can be assigned a value and used in calculations and other programming operations.

Example

```
• 10   M(1,2)=10          ! ASSIGNS 10 TO ONE ELEMENT
  20                      ! ALSO IMPLICITLY DIMENSIONS ARRAY
• 30   A=M(1,2)/7
• 40   PRINT M(1,2),A     ! OUTPUTS TWO VALUES
  50   END
```


Array Identifier

All elements of an array (in its working size) can be specified collectively in an input or output operation by using the array identifier: `(*)` after the name. For example –

```
PRINT A(*)
```

prints the entire array A.

String Variables

A string is a series of ASCII characters like –AB12*& which can be stored in a string variable. A string variable can be declared in a DIM or COM statement which specifies the **maximum length** of the string in number of characters up to 32 767. If a string variable is used without being specified in a DIM or COM statement, it is implicitly dimensioned to be eighteen characters maximum. The **current length** of a string refers to the number of characters currently assigned to the string.

Each string variable must have a name which is followed by a dollar sign ($) to specify a string variable as opposed to a numeric variable.

Characters can be assigned to a string variable using the LET statement –

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

---

For example –

```
10    LET Ski$ = "Keystone"
20    X$="*********"
```

Each assigns a value to a string variable.

String Arrays

A string array is a collection of strings; each string is one element. It can be dimensioned in a DIM or COM statement. Every string in the array has the same maximum length. Like a numeric array, a string array can be implicitly dimensioned.

In all string operations, an element of a string array can be used just like a simple string.

Example

```
• 10   Data$(1,1,1)="SKI"     ! IMPLICIT DIMENSION OF STRING ARRAY
  20                          ! ASSIGN "SKI" TO 1 ELEMENT (STRING)
• 30   A$(1,2)=A$(2,3)="**"   ! ANOTHER IMPLICIT DIMENSION
  40                          ! ASSIGN "**" TO 2 ELEMENTS OF ARRAY
• 50   PRINT Data$(1,1,1),! OUTPUTS 1 STRING (1 ELEMENT OF Data$)
  60   END
```

String Expressions

Text within quotes (a literal) is the simplest form of a string expression and can be made up of any ASCII characters excluding quotation marks. The quotation marks are not part of the string. The forms that a string expression can take are –

- Text within quotes
- String variable name
- Substring
- String concatenation operation
- String function
- User-defined string function

As with numeric expressions, a string expression can be enclosed in parentheses, if necessary.

Substrings, concatenation and string functions are covered in Chapter 6. See Chapter 8 for an explanation of user-defined string functions.

Example

Here is a sample program illustrating different ways to output SKI VAIL using string expressions.

```
  10   DIM A$[50]
• 20   PRINT "SKI VAIL"              ! TEXT IN QUOTES
  30   A$="SKI VAIL"
• 40   PRINT A$                      ! STRING VARIABLE
  50   A$="IT'S FUN TO SKI VAIL IN FEBRUARY"
• 60   PRINT A$[13;8]                ! SUBSTRING
• 70   PRINT "SKI"&" VAIL"           ! CONCATENATION
• 80   PRINT REV$("LIAV IKS")        ! STRING FUNCTION
  90   DEF FNA$="SKI VAIL"
• 100  PRINT FNA$                    ! USER-DEFINED STRING FUNCTION
  110  END

   SKI VAIL
   SKI VAIL
   SKI VAIL
   SKI VAIL
   SKI VAIL
   SKI VAIL
```

Declaring and Dimensioning Variables

Five variable declarative statements are used to dimension arrays and strings and declare the precision of numeric variables –

```
COM
DIM
INTEGER
SHORT
REAL
```

These statements also reserve space in memory for the specified variables.

Lower Bounds of Dimensions

Subscripts specify the upper bound of a dimension and can also specify the lower bound. See the Subscripts section earlier in this chapter for a discussion of using a colon to specify the lower bound.

5

The OPTION BASE Statement

When dimensioning arrays, you may want to specify that the default lower bound for dimensions be one rather than zero. This can be done using the OPTION BASE statement –

```
OPTION BASE 1
```

This statement must come **before** any of the variable declarative statements used in a program. Then, any lower bound not specified is 1.

If OPTION BASE 1 is not declared in a program, you may wish to include the statement –

```
OPTION BASE 0
```

for documentation purposes.

The OPTION BASE statement can't be executed from the keyboard.

The DIM Statement

The DIM (dimension) statement is used to dimension and reserve memory for full-precision numeric arrays and initialize each element to zero. It is also used to dimension and reserve storage space for simple strings and string arrays and initialize all strings to the null string.

DIM item [, item...]

The item can be –

numeric array (subscripts)
simple string [number of characters]
string array (subscripts) [[number of characters]]

Example

```
• 10   OPTION BASE 1        ! 1 is lower bound for array dimensions;
                              doesn't affect string lengths
• 20   DIM A(4,4),Array$(4,4)[36]
  30                        ! Line 20 dimensions numeric & string
                              array;  each has 16 elements
• 40   DIM B$[56]           ! Simple string--56 characters maximum
• 50   DIM C$(2,5)          ! Dimensions a 10 element string array--
                              18 characters is default length
  60   STOP
```

5

Remember these things when using DIM –

- The DIM statement must be executed via a program, not from the keyboard. Its location in a program is arbitrary, though it must be after any OPTION BASE statement. At pre-run initialization, the variables are dimensioned and initialized.

- DIM need not be used to assign space for strings with eighteen characters or less or for arrays having upper bounds of ten or less. These can be dimensioned implicitly. This, however, may waste memory by creating arrays which are larger than you need.

- A program can have more than one DIM statement, but the same variable name can be declared only once in a program. Therefore, arrays of differing dimensions can't have the same name. But remember that the same name may be used for a simple numeric, simple string, numeric array and string array.

- The maximum number of dimensions that can be specified is six. The range of subscripts is −32 767 to 32 767. No array can have more than 32 767 elements. No simple string can be longer than 32 767 characters. The size of arrays or strings may be limited by available memory, however.

The INTEGER Statement

The INTEGER statement is used to dimension and reserve memory for integer precision variables – simple and array. Integer-precision variables can be used to conserve memory; all calculations are performed with full-precision accuracy however, so a conversion is made before and after an operation.

INTEGER numeric variable₁ [(subscripts)] [, numeric variable₂ [(subscripts)], ...]

Example

```
30   OPTION BASE 1
40   INTEGER X,Y(2,2)
```

declares X to be an integer and Y to be an integer array of four elements.

The SHORT Statement

The SHORT statement is used to dimension and reserve storage for short-precision variables – simple and array. Short-precision variables can be used to save memory. All calculations are performed with full-precision accuracy, however, so a conversion is made before and after an operation.

SHORT numeric variable₁ [(subscripts)] [, numeric variable₂ [(subscripts)], ...]

Example

```
30 SHORT A(4,4),B(3,3,3),D
```

declares A and B as short precision arrays and D as a simple, short precision variable.

The REAL Statement

The REAL statement is used to dimension and reserve memory for full-precision (real) variables – simple and array.

REAL numeric variable₁ [(subscripts)] [, numeric variable₂ [(subscripts)], ...]

Example

```
10    REAL M(2,2,2,2),N
```

dimensions the array M and declares the simple variable N.

Since the DIM statement can also be used to dimension full-precison variables, the REAL statement can be used for documentation purposes to document which variables are full precision.

The COM Statement

The COM statement is used to dimension and reserve memory for simple and array variables. This includes strings and all three numeric precisions. COM is unique because it reserves memory space in a special "common" area which allows data to be transferred to and from subprograms or to other programs when each program or subprogram has corresponding COM statements.

COM item [, item, ...]

The item can be –

simple numeric
numeric array (subscripts)
simple string [[number of characters]]
string array (subscripts) [[number of characters]]

In addition, any one of the type-words – INTEGER, SHORT, REAL – can precede one or more numeric variables. All variables following a numeric type word have that precision until another type is specified or a string is declared.

Example

```
  10   OPTION BASE 1
• 20   COM A,B(2,4),C$,D,INTEGER E
• 30   COM F$(5)[24],G,SHORT H(5),J
```

The variables A,B(2,4), D and G are all full precision. Full precision is assumed at the beginning of the COM list and for numeric variables which are declared after any string. Since all variables following a numeric type word have that precision until another type is specified or a string is declared, both H(5) and J are short precision.

The items declared in corresponding COM statements in separate programs and subprograms must correspond to preserve values. Each item must be of the same type – integer, short, full-precision and string – as the corresponding item in other COM statements. Arrays must have the same maximum number of dimensions and elements; strings must have the same number of characters dimensioned. Names need not match, however.

COM statements in separate programs need not have the same number of items. A shorter COM statement in a succeeding main program causes the extra data from the first COM statement to be lost. A longer COM list in a succeeding program causes the new elements of the second COM statement to be initialized to 0 or the null string.

The use of COM statements within subprograms is discussed in Chapter 9.

5

Storage of Variables

To determine how many bytes variables require when stored **in memory** (storage on a mass storage medium is different; see Chapter 11), use the following tables.

Simple Variable	Amount of Memory Used
Full precision	10 bytes
Short precision	6 bytes
Integer	4 bytes
String	6 bytes + length (1 byte per character, rounded up to an even integer)

Array Variable	Amount of Memory Used[1]
Full precision	10 bytes + 4 bytes per dimension + 8 bytes per element
Short precision	10 bytes + 4 bytes per dimension + 4 bytes per element
Integer	10 bytes + 4 bytes per dimension + 2 bytes per element
String	12 bytes + 4 bytes per dimension + 2 bytes per element + length of each string (1 byte per character, rounded up to an even integer)

5

Assigning Values to Variables

Values can be assigned to variables during a program, either from within the program or input directly from the keyboard. The following statements can be used for assigning values –

```
LET
READ, MAT READ, DATA, RESTORE
INPUT, MAT INPUT
LINPUT
EDIT
```

The LET statement was discussed earlier in this chapter. The others follow in this chapter.

1 See Appendix F for more information.

READ, MAT READ and DATA Statements

To assign values to variables from within a program, the DATA statement is used with READ and MAT READ.

The DATA statement(s) provides values that are assigned to the variables –

DATA constant or text [, constant or text, ...]

Text in the DATA statement can be quoted or unquoted. A constant can be interpreted as either a numeric or unquoted text. The location of the DATA statements within a program segment is unimportant. If there are multiple DATA statements, make sure they are in the order you want.

Example

```
10 DATA 88,77, April, "100", "Pay", 95
```

READ and MAT READ specify the variables for which values are obtained from DATA statements –

READ variable name [, variable name , ...]
MAT READ array variable [(redim subscripts)] [, array variable [(redim subscripts)], ...]

5

The variables specified in the READ statement can be any single variable or an array identifier: (*) following the array name.

Example

Here's a READ statement which could correspond to the previous DATA statement –

```
100    READ A,A$, Month$, Miles$, Pay$, Total
```

Notice that an unquoted value in the DATA statement (77) can correspond to a string variable in the READ statement (A$). It is interpreted as unquoted text in this case.

The MAT READ statement specifies entire arrays for which values are to be read. The working size of the array can be altered by including the redim subscripts. However, redimensioning of any but the first array takes place only if a DATA item is read for at least one element in that array. The total number of elements can't be greater than the number originally dimensioned. The number of dimensions can't change. The subscripts can be any numeric expression except one containing a multiple-line user-defined function (FN) reference.

Array elements are read in order with the rightmost subscript varying fastest.

Here is an example of MAT READ –

```
  10    OPTION BASE 1
  20    DIM Equipment$(2,2,2)          ! STRING ARRAY
• 30    DATA SKIS,BOOTS,BINDINGS,POLES,SUNGLASSES
• 40    DATA HAT,MITTENS,JACKET,GOGGLES
• 50    MAT READ Equipment$
  60    MAT PRINT Equipment$
  70    END
```

Here is the output –

```
SKIS
BOOTS

BINDINGS
POLES

SUNGLASSES
HAT

MITTENS
JACKET
```

Values are read in the following order –

A(1,1,1), A(1,1,2), A(1,2,1), A(1,2,2), A(2,1,1), A(2,1,2), A(2,2,1), A(2,2,2)

READ, MAT READ and DATA are programmable only; they can't be executed from the keyboard.

DATA Pointer

The computer uses an internal mechanism called a **DATA pointer** to locate the next data element that is to be read. The leftmost element of the lowest-numbered DATA statement in the current environment is read first. After this element is read and another value required by READ, the DATA pointer repositions itself one element to the right, and continues to do so each time another data element is read. After the last element in a DATA statement is read and another value required by READ, the DATA pointer locates the next higher numbered DATA statement and repositions itself at the first element in that statement. If there are no higher-numbered DATA statements, the data pointer remains at the end of the previous DATA statement; any effort to read additional data will result in ERROR 36.

Repositioning the DATA Pointer

The DATA pointer can be repositioned to the beginning of any DATA statement in the current program segment, so that values can be reused, by using the RESTORE statement –

RESTORE [line identifier]

If no line identifier is specified, the pointer is repositioned to the beginning of the lowest-numbered DATA statement in the current program segment. The RESTORE statement can't be executed from the keyboard.

If the specified line is not a DATA statement, the first DATA statement following the specified line is accessed.

Examples

Here are some examples of DATA, READ, MAT READ and RESTORE.

This example shows that several READ statements can apply to the same DATA statement. It also shows that string values can be quoted or unquoted, though quotes are not part of the string; notice that 7.31 is a string value assigned to A$.

```
  10   DIM A(1:3)
• 20   READ A(*)           ! READS THE 4,5 AND 6
• 30   READ A$,B1          ! READS THE 7.31 AND THE 2.69
• 40   READ X$             ! READS "Hours"
• 50   DATA 4,5,6,7.31,2.69,"Hours"
  60   STOP
```

This example illustrates use of RESTORE and MAT READ. The values in line 30 are used as the values of five simple variables, then re-used as the values in array B.

```
  10   OPTION BASE 1
  20   DIM B(5)
• 30   DATA 10,12.7,3,15,8
  40   FOR I=1 TO 5
• 50      READ C
  60      PRINT "C=";C,"SQUARE ROOT OF C=";SQR(C)
  70   NEXT I
  80   PRINT
  90   PRINT               ! LINES 80 AND 90 GIVE BLANK LINES
• 100  RESTORE 30          ! LETS THE DATA IN LINE 30 BE RE-USED
• 110  MAT READ B
  120  MAT PRINT B;
  130  END
```

Here is the output –

```
C= 10                SQUARE ROOT OF C= 3.16227766016
C= 12.7              SQUARE ROOT OF C= 3.56370593624
C= 3                 SQUARE ROOT OF C= 1.73205080756
C= 15                SQUARE ROOT OF C= 3.8729833462
C= 8                 SQUARE ROOT OF C= 2.82842712474


 10  12.7  3  15  8
```

Assigning Values From The Keyboard

The INPUT Statement

The INPUT statement allows values in the form of expressions to be assigned to variables from the keyboard at the request of the program –

INPUT ["prompt" ,] variable name [, ["prompt" ,] variable name , ...]

When the INPUT statement is executed, a ? or the prompt, if present, is displayed. The prompt is any combination of characters; it can be used to help remember for what variable a value is being requested. Each prompt applies only to the variable to its right. If no characters are present between the quotes, nothing is displayed. Any variable not preceded by a prompt uses a question mark by default. A value can then be input for each variable designated in the INPUT statement. Values are entered into the computer by pressing CONT or STEP.

Example

For instance, in the statement –

```
10 INPUT A,B(1,2),C$,"D",D
```

four values are requested.

5

Values can be assigned individually or separated by commas in groups. Values input for strings can be quoted or unquoted. Quotation marks can't be input as part of the string's value. An unquoted value for a string can't contain a comma or exclamation point and all leading and trailing blanks are deleted.

Example

For example, the values 4,5,"Time", and 3 can be assigned to the variables in the example above in many ways; here are two –

4 CONT 5 CONT Time CONT 3 CONT

or

4 CONT 5 , Time, 3 CONT

In both cases, the ? or prompt reappears after CONT is pressed until all four values are input.

Pressing CONT without entering values when an input is requested causes execution to continue with the statement following the INPUT statement, even if values are still requested. STEP or the CONT command can also be used. Variables not assigned values retain their previous value.

Example

Here is an example –

```
  40   X=5
  50   PRINT X
• 60   INPUT "INPUT VALUES FOR A, B AND X",A,B,X
  70   PRINT A,B,X
  80   END
```

By responding to the INPUT statement with –

26, 19 (CONT) (CONT)

X retains the value 5.

The variable list can also include array identifiers. For example –

```
100  INPUT A,B(*)
```

requests values for the simple variable A and the array B.

The INPUT statement is programmable only; it can't be executed from the keyboard.

5

The MAT INPUT Statement

Entire arrays can be given values and optionally redimensioned using the MAT INPUT statement –

MAT INPUT array variable [(redim subscripts)] [, array variable [(redim subscripts)], ...]

When MAT INPUT is executed, a ? appears in the display line. Values in the form of numeric or string expressions can be entered separately, or in groups. As with the INPUT statement, values are stored by pressing (CONT) or (STEP) or using the CONT command. For example –

```
  40   OPTION BASE 1
  50   DIM Array(5,3,5),B(10),C(2,2)
• 60   MAT INPUT Array(5,2,4),B(5),C   ! Array IS REDIMENSIONED
  70   MAT PRINT Array;B;C;
  80   END
```

When this is executed, 45 separate values are requested. If

1, 2, 3, 4, 5, 6 (CONT) (CONT)

is entered, the only values that would change are the elements of Array with subscripts (1,1,1), (1,1,2), (1,1,3), (1,1,4), (1,1,5), (1,2,1). Array B is not redimensioned because no elements were input for it.

Remember, when an array is redimensioned, the number of dimensions can't change and the total number of elements can't exceed the number originally dimensioned.

MAT INPUT can't be executed from the keyboard.

Other ways of assigning values to arrays are discussed in Chapter 7.

The LINPUT Statement

The LINPUT statement is used to assign any combination of characters to a string variable or substring –

LINPUT ["prompt" ,] string variable or substring

5

When LINPUT is executed, a ?, or the prompt if present, is displayed. All characters typed in become the value of the string when (CONT) or (STEP) is pressed. Here is an example –

```
100 LINPUT A$[20]     ! Input value for A$[20]
```

The response could be –

"Compute, then print", he said. (CONT)

Pressing (CONT) or (STEP) without entering a value erases the current value of the string and sets it to the null string.

Notice that the LINPUT statement allows quotation marks to be input for the value of a string variable; this isn't possible with the INPUT statement.

The LINPUT statement can't be executed from the keyboard.

The EDIT Statement

The current value of a string can be viewed and edited by using the EDIT statement –

EDIT ["prompt" ,] string variable or substring

When the EDIT statement is executed, a ?, or the prompt if present, is displayed and the current value of the specified string appears in the keyboard entry area. On the 9835B, the prompt is quickly replaced by the value of the string.

This value can then be edited like any keyboard entry. (CLEAR LINE) can be used to clear the line, allowing a totally new value to be entered, like with LINPUT. However, the original value can't be recalled. Pressing (CONT) stores the characters displayed in the keyboard entry area for the value of the string. For example, EDIT could be used to alter the names in printed output –

```
  10   DIM O$[60]
  20   O$="Ed Smith spent "
  30   INPUT "AMOUNT SPENT",Dollars
  40   PRINT O$;Dollars
• 50   EDIT "NEW NAME",O$
  60   GOTO 30
  70   STOP
```

When line 50 is executed, NEW NAME is displayed. Ed Smith spent appears in the keyboard entry area. Then the character editing keys could be used to change the name.

The limit on the length of the string being edited is 160 characters (the length of the keyboard entry area). So, if a longer string is specified, ERROR 37 would occur. This could be avoided by using substrings; here is an example –

```
  10   DIM A$[200]
  20   A$=RPT$("1234567890",20)
• 30   EDIT "First 160 characters",A$[1,160]
• 40   EDIT "Last 40 characters",A$[161]
  50   PRINT A$
  60   STOP
```

The EDIT statement can't be executed from the keyboard.

5

Chapter 6
String Operations

Introduction

This chapter further explores the use and manipulation of string variables.

The following topics are covered in this chapter –

- Substrings
- String concatenation
- String modification
- Using CRT special features
- String functions
- Relational Operations

Substrings

A substring is a part of a string which is made up of zero or more contiguous characters. A substring is specified by placing **substring specifiers** in **brackets** after the string name. There are three forms a substring can have –

- String variable name [character position]

  The character position is a numeric expression which is rounded to an integer. The substring is made up of that character and all following it.

- String variable name [beginning character position, ending character position]

  This substring includes the beginning and ending characters and all in between. The character positions must be within the dimensioned number of characters. The second subscript must be greater than or equal to the first, minus one.

- String variable name [beginning character position; number of characters]

  This substring begins with the specified character in the string and is the specified length. The number of characters specified can't exceed the dimensioned length, less the beginning character position.

Example

```
  10   OPTION BASE 1
  20   DIM A$(3,4)[25]         ! String array:
                                 12 strings of 25 characters
  30   A$(2,2)=" CARPENTRY"    ! Assign value to a string
                                 note 1st character is a blank
  40   PRINT "HERE ARE SOME SUBSTRINGS:",LIN(1)
• 50   PRINT "  A$(2,2)[8]:";TAB(20);A$(2,2)[8]
• 60   PRINT "A$(2,2)[2,4]:";TAB(20);A$(2,2)[2,4]
• 70   PRINT "A$(2,2)[5;3]:";TAB(20);A$(2,2)[5;3]
  80   END
```

```
HERE ARE SOME SUBSTRINGS:

  A$(2,2)[8]:      TRY
A$(2,2)[2,4]:      CAR
A$(2,2)[5;3]:      PEN
```

String Concatenation

The string concatenation operator joins (concatenates) one string to the back end of another.

string expression & string expression [&string expression...]

Example

```
  10   A$="TRY"             ! Dimension A$ & B$ implicitly
  20   B$="CAR"
• 30   PRINT B$&"PEN"&A$    ! Concatenate many items together
  40   END
```

```
CARPENTRY
```

String Variable Modification

A string or substring can be modified by another string or substring. For example, a part of a string can be changed or characters can be added or deleted. The string containing the modification is called the **modifying string**; the string being modified is the **destination string**. The destination string can be a string or substring. The modifying string can be any string expression.

The characteristics (length and content) of the destination string after modification depend not only on the characteristics of the modifying string, but also on the number of subscripts given for the destination string.

If a string expression is to be stored into a string or substring which is too short to hold it, the result is truncated on the right. ERROR 18 occurs if the destination string is a one-substring-specifier substring which is too short to contain the result. Here is an example of how this can occur –

```
  10   DIM A$[8]
  20   A$="GEOMETRY"
• 30   A$[6]="TRIC"       ! Trying to put 4 characters
                            into characters 6-8
  40   END
```

```
ERROR 18 IN LINE 30
```

Each string of a string array can be modified in the same way as a simple string by the inclusion of subscripts.

No Substring Specifiers

When the destination string has no substring specifiers, the entire destination string is replaced by the modifying string or substring. Its characteristics after modification are the same as those of the modifying string or substring.

Example

```
  10    A$="HELLO"
  20    B$="GOODBYE"
  30    PRINT "A$= ";A$,"B$= ";B$
• 40    B$=A$              ! REPLACE B$
  50    PRINT "B$= ";B$
  60    END
```

```
A$= HELLO             B$= GOODBYE
B$= HELLO
```

If the modifying string is longer than the length of the destination string, ERROR 18 occurs.

One Substring Specifier

6

When the destination string has one substring specifier, the indicated substring is replaced by the modifying string or substring. The destination string can be shortened or lengthened. Attempting to lengthen the destination string beyond its maximum length causes ERROR 18.

Examples

```
  10    A$="Atkins"
  20    PRINT A$
• 30    A$[3]="wood"
  40    PRINT A$
  50    END
```

```
Atkins
Atwood
```

Here's an example of the destination substring being shortened –

```
  10    B$="Atwater"
  20    PRINT B$
• 30    B$[3]="kins"       ! SHORTEN STRING
  40    PRINT B$
  50    END
```

```
Atwater
Atkins
```

Here's an example of the destination string being lengthened –

```
  10    C$="Atwater"
  20    PRINT C$
• 30    C$[3]="kinson"     ! LENGTHEN STRING
  40    PRINT C$
  50    END
```

```
Atwater
Atkinson
```

Characters added to those of a string must be contiguous; that is, they must immediately follow the destination string without any unassigned character spaces. If they are non-contiguous, `ERROR 18` occurs. For example –

```
  10    D$="Andy"
• 20    D$[8]="Atkinson"
  30    PRINT D$
  40    END
```

```
ERROR 18 IN LINE 20
```

`ERROR 18` is caused because character positions 5, 6 and 7 aren't assigned any characters.

Two Substring Specifiers

When the destination string has two substring specifiers, with either a comma or semicolon, the indicated substring is replaced by the modifying string or substring. The left-most character of the modifying string or substring replaces the left-most character of the indicated destination substring. The next adjacent character is replaced, and so forth, until the indicated destination substring is filled. If the modifying string is shorter than the indicated destination substring, the remainder of the destination substring is filled with blanks. If the modifying string is longer than the indicated destination string, the remainder of the modifying string is truncated.

Example

```
  10    A$="Loveland"
  20    PRINT A$
  30    A$[1,4]="Home"
  40    PRINT A$
• 50    A$[1,4]="Up"           ! MODIFYING STRING SHORTER
  60    PRINT A$
• 70    A$[1,4]="Tomorrow"     ! MODIFYING STRING LONGER
  80    PRINT A$
  90    END


  Loveland
  Homeland
  Up  land
  Tomoland
```

6

The length of the destination string after modification either is unchanged, or is greater. When the value of the second substring specifier is greater than the current length of the destination string, the modification results in a lengthened string (within its maximum length).

Example

```
  10    C$="Goodbye"
  20    PRINT C$
• 30    C$[5,9]="times"     ! C$ IS LENGTHENED
  40    PRINT C$
  50    END


  Goodbye
  Goodtimes
```

The Null String

The null string is a string which contains no characters or blanks. The following examples each specify the null string –

```
10  LET N$ = ""
20  M$ = A$[4,3]
```

All strings are initialized to the null string by a `DIM` or initial `COM` statement or when `SCRATCH V` or `SCRATCH C` is executed. The null string can be used to clear a string.

Special Features (9835A only)

The three SFK special features – blinking, inverse video, underline – (see Chapter 3) are especially useful with strings. Strings can be displayed or printed to the CRT with any combination of the special features. Here is an example using underline –

```
  10   DIM A$[25]
● 20   A$="DO IT NOW!!!"     ! CONT'L & K2 were pressed
                               before 'N' was typed and
  30                         ! after 'W' was typed.
  40                         ! This turns underline on and off
  50   PRINT A$
  60   !   Blinking, inverse video and underline can
  70   !   emphasize text.  They can be used separately
  80   !   or combined.
  90   END
```

```
DO IT NOW!!!
```

6

Programming Hint

One factor to consider is that entering any combination of special feature modes adds one character to the length of the string, as does exiting the special features mode. For example, the length of –

"Δ 12345678"

is 11 characters.

---

1 Δ specifies a blank which is considered a character.

String Functions

A string function returns a numeric or string value to an expression. String functions enable you to determine the length of a string and analyze and manipulate its contents. String functions are especially useful in text processing applications.

Length Function

The length (LEN) function returns the number of characters in a string expression –

LEN string expression

The current length of the string expression is returned.

Here are some examples –

```
  10   REM    THIS EXAMPLES USES LEN AND TRIM FUNCTIONS
  20   REM    TO CENTER A LINE OF OUTPUT
  30   PRINTER IS 6,WIDTH(40)
  40   Area$="  KEYSTONE"
• 50   C=(40-LEN(TRIM$(Area$)))/2     ! FINDS CENTERING FACTOR
                                        USING PRINTER WIDTH OF 40
  60   PRINT "*";TAB(40),"*"
• 70   PRINT TAB(C),TRIM$(Area$)      ! CENTERS OUTPUT
  80   PRINT "*";TAB(40),"*"
  90   END
```

```
*                                      *
                KEYSTONE
*                                      *
```

6

Position Function

The position (POS) function determines the position of a substring within a string –

POS (in string expression, of string expression)

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

Here are some examples –

```
  10   REM   EXAMPLES OF POS FUNCTION
  20   PRINT "POSITION OF 'PARK' IN 'WINTER PARK':";
• 30   PRINT POS("WINTER PARK","PARK")
  40   PRINT "POSITION OF 'PARKS' IN 'WINTER PARK':";
• 50   PRINT POS("WINTER PARK","PARKS")
  60   REM
  70   REM NEXT LINE USES POS TO EXTRACT NAME FROM SENTENCE
  80   DIM A$[50]
  90   A$="MR. SMITH SPENT EIGHTY DOLLARS"
• 100  Start=POS(A$,".")       ! NAME IS AFTER PERIOD
• 110  End=POS(A$,"SPENT")     ! NAME IS BEFORE 'SPENT'
  120  PRINT A$[Start+1,End-1]
  130  END
```

6

```
POSITION OF 'PARK' IN 'WINTER PARK': 8
POSITION OF 'PARKS' IN 'WINTER PARK': 0
 SMITH
```

Value Function

With the value (VAL) function, the numeric value of a string or a substring of digits, including an exponent, can be used in calculations. (Normally the characters in a string are not recognized as numeric data and can't be used in numeric calculations.)

VAL (string expression)

The first character to be converted in a string using the VAL function must be a digit, a plus or minus sign, a decimal point or a space. A leading plus sign or space is ignored; a leading minus sign is taken into account. All following characters must be digits, a decimal point or an E. An E character after a numeric and followed by digits or a plus or minus sign and digits is interpreted as exponent of base 10. A decimal point following digits after an E terminates the exponent.

Numeric data entries can be combined logically with input text. All contiguous numerics are considered a part of the number until a non-numeric is reached in the string. This means that a string can contain more than one number. The first character of the string expression after leading spaces, plus signs or minus signs must be a digit or a decimal point. If the leading part of the string is not a valid number, ERROR 32 occurs.

For example –

```
  10   REM    EXAMPLES OF VAL FUNCTION
  20   REM    ANOTHER EXAMPLE IS FOUND WITH THE NUM FUNCTION
  30   A$(2,2)="JONES,J: 291228811"
• 40   PRINT VAL(A$(2,2)[10])
• 50   PRINT VAL("1E2.5")
  60   END
```

```
291228811
100
```

VAL$ Function

The VAL$ function is (nearly) the inverse of the VAL function and returns a string representing the number, in the current output mode –

VAL$ (numeric expression)

Example

```
  10   !    This example shows VAL$ function
  20   Total=0
  30   INPUT "NAME",Name$,"NUMBER OF TRIPS",N
  40   FOR I=1 TO N
  50     INPUT "AMOUNT SPENT ON TRIP?",Spent
  60     Total=Total+Spent
  70   NEXT I
• 80   Name$=Name$&"--$"&VAL$(Total)
  81   !    Line 80 stores Total with name in the string
  90   PRINT Name$
  100  !    The next 2 lines are another example of VAL$
• 110  FLOAT 3
• 120  PRINT LIN(2),"VAL$(120)= ";VAL$(120)
  130  END

  BOB JONES--$78

  VAL$(120)= 1.200E+02
```

Character Function

The character (CHR$) function converts a numeric value in the range −32 768 through 32 767 into a string character. Any number out of the range 0 through 255 is converted MOD 256 to that range. Any 8-bit character code can be stored in a string using the character function which is especially useful for accessing control codes and putting quotes into a string.

CHR$ (numeric expression)

For example –

```
  10   REM USING CHR$ FUNCTION TO PUT QUOTES IN A STRING
  20   DIM A$[50]
• 30   A$="BILL SAID, "&CHR$(34)&"LET'S GO!"&CHR$(34)
  40   PRINT A$
  50   END

  BILL SAID, "LET'S GO!"
```

See Appendix E for a table of correspondence between characters and numbers in the range 0 through 127.

With the 9835A, using this function with numbers in the range 128 through 159 is useful for stylized printed output; see Appendix B.

Numbers in the range 160 through 255 are used to access foreign characters; see Appendix C.

Numeric Function

The numeric (NUM) function converts an individual string character to its corresponding value represented decimally.

NUM (string expression)

The decimal equivalent of the first character of the expression is returned.

For example –

```
  10    !     Using NUM function to find number in a string
  20    !     VAL function is used to output number
  30    DIM A$[50]
  40    INPUT "ENTER NAME AND PHONE NUMBER",A$
  50    FOR I=1 TO LEN(A$)
• 60       IF (NUM(A$[I]))>=48) AND (NUM(A$[I])<=57) THEN 80
  61       ! Branch when a digit is encountered
  70    NEXT I
  80    PRINT "PHONE NUMBER STARTS IN POSITION";I;" OF A$"
• 90    PRINT "PHONE NUMBER IS:";VAL(A$[I])
  100   END

  PHONE NUMBER STARTS IN POSITION 6  OF A$
  PHONE NUMBER IS: 2223333
```

Uppercase Function

The uppercase (UPC$) function returns a string with all lowercase letters converted to uppercase.

UPC$ (string expression)

For example –

```
  10    REM     USING UPC$ FUNCTION WITH INPUT PROMPT
  20    DIM Answer$[25]
  30    INPUT "DO YOU WISH TO CONTINUE?",Answer$
• 40    IF UPC$(Answer$)="NO" THEN STOP
• 50    IF UPC$(Answer$)="YES" THEN 400
  60    BEEP
  70    PRINT "I DON'T UNDERSTAND YOUR ANSWER; ANSWER AGAIN"
  80    GOTO 30
  90    STOP
```

The uppercase function allows strings to be compared without regard to upper and lowercase. It can also be used with printers than can output only uppercase letters.

Lowercase Function

The lowercase (LWC$) function returns a string with all uppercase letters converted to lowercase –

LWC$ (string expression)

For example –

```
10   REM  USING LWC$ FUNCTION FOR ALPHABETIZING
20   INPUT "NUMBER OF NAMES?",N
30   FOR I=1 TO N
40      INPUT "NAME?",Name$(I)
• 50      Name$(I)=LWC$(Name$(I))   ! PUTS NAMES INTO LOWER CASE
60   NEXT I
70   REM  ALPHABETIZING SEQUENCE FOLLOWS ...
```

Repeat Function

The repeat (RPT$) function allows a string of characters to be repeatedly concatenated –

RPT$ (string expression, number of repetitions)

For example –

```
10   REM  USING RPT$ FUNCTION TO UNDERLINE A TITLE
20   INPUT "TITLE OF BOOK?",Title$
30   PRINT Title$
• 40   PRINT RPT$("_",LEN(Title$))
50   END
```

```
GONE WITH THE WIND
__________________
```

The number of repetitions can be any numeric expression in the range 0 through 32 767 when rounded. If 0 is specified, the result is the null string. The length of the result can't exceed 32 767 characters.

6

Reverse Function

The reverse (REV$) function reverses the order of the characters in a string –

REV$ (string expression)

For example –

```
10    !     Using REV$ to output a line of text
            without splitting a word in the middle
20    PRINTER IS 16,WIDTH(40)
30    DIM Line$[80],Temp$[80]
40    Line$[1,28]="STRING FUNCTIONS ARE USEFUL"
50    Line$[29]="FOR TEXT PROCESSING APPLICATIONS"
• 60  Temp$=REV$(Line$[1,40])        ! Reverse 1st 40 characters
70    FOR I=1 TO 40
80       IF Temp$[I,I]=" " THEN 100! Search for blank
90    NEXT I
100   PRINT Line$[1,40-I+1]          ! Output up to blank
110   PRINT Line$[40-I+2]            ! Output from blank on
120   END
```

```
STRING FUNCTIONS ARE USEFUL FOR TEXT
PROCESSING APPLICATIONS
```

6

Trim Function

The trim (TRIM$) function deletes leading and trailing blanks from a string –

TRIM$ (string expression)

For example –

```
10    REM   EXAMPLES OF TRIM$ FUNCTION
• 20  PRINT TRIM$("   STRING   "&"   FUNCTIONS    ")
30    REM   NOTE THAT INTERNAL BLANKS AREN'T TRIMMED
40    REM   TRIM$ IS ALSO USEFUL FOR TRIMMING INPUT RESPONSES
50    END
```

```
STRING      FUNCTIONS
```

Relational Operations

String variables may be compared using the relational operators –

```
=
<
>
<=
>=
<> or #
```

Each character in a string is represented by a standard equivalent decimal code, as shown in the table in Appendix E. When two string characters are compared, the lesser of the two characters is the one whose decimal code is smaller. For example, 2 (decimal code 50) is smaller than R (decimal code 82).

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. For example, "STEVE" is smaller than both "STEVEΔ" and "STEVEN".

Examples

Here is an example which could be used to allow communication between the computer and the user –

```
10    DIM Answer$[20]
20    INPUT "DO YOU WISH TO CONTINUE--YES OR NO?",Answer$
30    IF Answer$="NO" THEN STOP
40    REM   A$ WAS YES
50    REM   CONTINUATION OF PROGRAM
```

In some cases, such as in alphabetic sequencing problems, it is useful to compare strings for conditions other than "equal to" and "not equal to". For example, to arrange several different strings in alphabetical order, the following type of string comparison could be included in a program.

```
10    DIM Name1$[20],Name2$[20],Temp$[20]
20    INPUT "TWO NAMES?",Name1$,Name2$
30    IF Name1$<Name2$ THEN 70
40    Temp$=Name2$
50    Name2$=Name1$
60    Name1$=Temp$
70    PRINT Name1$;" IS LESS THAN ";Name2$
80    END

JOHNSON IS LESS THAN JONES
```

Chapter 7
Array Operations

Introduction

The many operations that can be performed with numeric arrays are discussed in this chapter. Some of the operations can be used with matrices and vectors only; these are noted. In all single-argument array operations such as TRN or CSUM, the operand array can be enclosed in parentheses.

The following topics are covered in this chapter –

- Assigning a constant value
- Copying an array
- Scalar operations
- Arithmetic operations
- Using functions
- Identity matrix
- Matrix multiplication
- Inverse of a matrix
- Transpose of a matrix
- Row and column sums
- Array functions
- Redimensioning an array

Assigning a Constant Value

Three statements allow a constant value to be assigned to every element in an array.

1. MAT...CON

The MAT...CON statement assigns the value 1 to every element –

MAT array variable = CON [(redim subscripts)]

When executed, all elements in the current size of the array are assigned the value 1. The current size can also be redimensioned by including the redim subscripts. The redimensioning is done before the assignment takes place.

Example

In this example the value 1 is assigned to 25 elements of the array A –

```
  10   OPTION BASE 1
  20   DIM A(15,15)
• 30   MAT A=CON(5,5)     ! ASSIGNS 1 TO 25 ELEMENTS OF MATRIX A
  40   MAT PRINT A;
  50   END
```

2. MAT...ZER

The MAT...ZER statement sets all elements in a numeric array to 0. It also allows the array to be redimensioned.

MAT array variable = ZER [(redim subscripts)]

Again, the optional redimensioning takes place before the assignment.

7

Example

```
  90  OPTION BASE 1
• 100 MAT X = ZER (15)
```

15 elements of the array X are assigned the value 0.

Remember, any time an array is redimensioned the following are always true –

- The number of dimensions can't change.
- The total number of elements can't exceed the total originally dimensioned.

3. MAT-Initialize

The MAT – **Initialize** statement assigns the value of the numeric expression to every element in a numeric array.

MAT array variable = (numeric expression)

Example

```
100   MAT Area = (2*PI)
```

This assigns the value of 2*PI to every element in Area.

The numeric expression is evaluated once; it is converted to the numeric type of the array, if necessary.

Example

```
• 10    INTEGER X(4,4)     ! OPTION BASE VALUE IS 0
• 20    MAT X=(PI)         ! VALUE OF PI IS ROUNDED TO AN INTEGER
  30    MAT PRINT X;
  40    END
```

```
   3   3   3   3   3

   3   3   3   3   3

   3   3   3   3   3

   3   3   3   3   3

   3   3   3   3   3
```

Line 20 causes the value 3 to be assigned to every element in X.

Copying An Array

The MAT – **Copy** statement copies the value of each element of a numeric array into the corresponding element of the result array.

MAT result array = operand array

The two arrays must have the same number of dimensions. The number of elements in the result array must be greater than or equal to the number of elements in the current size of the operand array.

Example

```
  10    OPTION BASE 1
  20    DIM C(4,4),D(2,2)
  30    PRINT "MATRIX C:",C(*);
  40    MAT D=(2.5)
  50    PRINT "MATRIX D:",D(*)
• 60    MAT C=D         ! C IS REDIMENSIONED TO SIZE OF D
  70    PRINT "MATRIX C:",C(*)
  80    END
```

```
MATRIX C:
 0   0   0   0

 0   0   0   0

 0   0   0   0

 0   0   0   0

MATRIX D:
 2.5                2.5

 2.5                2.5


MATRIX C:
 2.5                2.5

 2.5                2.5
```

The values of array D are copied into the elements of array C, then the working size of array C is redimensioned to be a 2 by 2 array.

7

Mathematical Operations

There are various mathematical operations that can be performed with arrays. These are covered next.

Scalar Operations

The **scalar operation** statement allows an arithmetic or relational operation to be performed with each element of an array using a constant scalar (any numeric expression). The result of the operation becomes the value of the corresponding element of the result array.

MAT result array = operand array operator (scalar)
MAT result array = (scalar) operator operand array

Example

```
10   OPTION BASE 1
20   DIM B(3,3),C(2,2)
30   FOR I=1 TO 2      ! THESE LOOPS ASSIGN VALUES TO C
40      FOR J=1 TO 2
50         C(I,J)=I+J
60      NEXT J
70   NEXT I
• 80 MAT B=C*(4)       ! EACH ELEMENT IN C IS MULTIPLIED BY 4--
                         B IS ALSO REDIMENSIONED
90   PRINT "MATRIX C:",C(*)
100  PRINT "MATRIX B:",B(*)
110  END
```

```
MATRIX C:
 2                 3

 3                 4


MATRIX B:
 8                 12

 12                16
```

In this example, each element in array C is multiplied by 4 and the result is stored in the corresponding element of array B.

7

The following operators are allowed –

- +
- –
- *
- /
- =
- <> or #
- >
- <
- >=
- <=

The two arrays must have the same number of dimensions. The result array can't be smaller than the operand array. The array is redimensioned after the operation so that it has the same working size as the operand array.

Arithmetic Operations

The **arithmetic operation** statement allows an arithmetic or relational operation to be performed with corresponding elements of two numeric arrays; the result becomes the value of the corresponding element in the result array.

MAT result array = operand array   operator   operand array

Examples

In this example, corresponding elements of arrays A and B are added.

```
  10   OPTION BASE 1
  20   DIM A(2,2),B(2,2),Sum(3,3)
  30   FOR I=1 TO 2      ! ASSIGNING VALUES TO A AND B
  40      FOR J=1 TO 2
  50         A(I,J)=J+I
  60         B(I,J)=J*I
  70      NEXT J
  80   NEXT I
• 100  MAT Sum=A+B       ! EACH ELEMENT OF Sum IS THE SUM OF
                           CORRESPONDING ELEMENTS OF A AND B
  110  PRINT "ARRAY A:",A(*),"ARRAY B:",B(*)
  120  PRINT "SUM OF A AND B:",Sum(*)
  130  END
```

```
ARRAY A:
 2                    3

 3                    4

ARRAY B:
 1                    2

 2                    4


SUM OF A AND B:
 3                    5

 5                    8
```

In this example, corresponding elements of arrays Hours and Rate are multiplied together and the result stored in array Pay.

```
  10   OPTION BASE 1
  20   DIM Pay(5),Hours(5),Rate(5)
  30 Rates:   DATA  5.25,5.00,4.25,7.15,4.80
  40 Hours:   DATA  40,42,28,39,40
  50   MAT READ Hours,Rate
• 60   MAT Pay=Hours.Rate
  70   REM  EACH ELEMENT OF Pay IS THE PRODUCT OF
            CORRESPONDING ELEMENTS OF Hours AND Rate
  80   PRINT "PAY TOTALS:",Pay(*);
  90   END


PAY TOTALS:
 210  210  119  278.85  192
```

The following operators are allowed –

+	<> or #
-	>
. (multiply)	<
/	>=
=	<=

Notice that multiplication is indicated by a period. An asterisk indicates matrix multiplication which is covered later in this chapter.

The result and operand arrays must have the same number of dimensions. The operand arrays must have the same number of elements in each dimension; the result array can't be smaller.

Functions

The **function** statement allows each element in the operand array to be evaluated by the specified function. The result becomes the corresponding element of the result array.

MAT result array = function   operand array

The function must be a single-argument system function like SIN, ABS or SQR.

Example

In this example, the square root of each element in array A is assigned to the corresponding element in array B.

```
  10   OPTION BASE 1
  20   DIM A(3,3),B(3,3)
  30   DATA 25,13,55,66,48,15,36,64,80
  40   MAT READ A
• 50   MAT B=SQR(A)    ! Each element of array B is square root of
                         corresponding element of array A
  60   PRINT "MATRIX A:",A(*)
  70   FIXED 5
  80   PRINT "MATRIX B - THE SQUARE ROOTS:",B(*)
  90   END
```

```
MATRIX A:
 25                   13                   55

 66                   48                   15

 36                   64                   80


MATRIX B - THE SQUARE ROOTS:
 5.00000              3.60555              7.41620

 8.12404              6.92820              3.87298

 6.00000              8.00000              8.94427
```

Matrices and Vectors

Many array operations can only be performed using matrices or vectors. These are covered next.

Identity Matrix

The MAT... IDN statement establishes the specified matrix as an identity matrix: all elements in the matrix equal zero except those in the main diagonal (upper left to lower right), which all equal one.

MAT matrix name = IDN [(redim subscripts)]

An identity matrix must be square (two dimensions; each dimension has the same number of elements); when the subscripts are included, this enables the matrix to be redimensioned before the identity matrix is established.

Matrix Multiplication

The **matrix multiplication** statement multiplies two matrices together. This is different from the multiplication of corresponding elements which was discussed previously.

MAT result matrix = operand matrix$_1$ * operand matrix$_2$

The number of columns of the first operand matrix **must** equal the number of rows of the second operand matrix. The result matrix has the same number of rows as the first operand matrix and the same number of columns as the second operand matrix. The result matrix can't be the same matrix as either of the operands. Here is an example –

$$B_{(3x4)} * C_{(4x2)} = A_{(3x2)}$$

Either or both of the operand matrices can also be a vector. The result matrix must also be a vector in this case. Here is an example –

$$X_{(5,6)} * Y_{(6)} = Z_{(5)}$$

If you have not been introduced to matrix multiplication, you might assume that corresponding elements are multiplied together; however, this is not the case. Assume we are multiplying matrix B by matrix C and storing the result into matrix A (MAT A=B*C). To determine the value of any element of matrix A, call it $A_{(x,y)}$, corresponding elements of the xth row of B and the yth column of C are multiplied together. The sum of the resultant products is the value for $A_{(x,y)}$.

Mathematically speaking –

MAT A = B * C

$$A(I,K) = \sum_{J=1}^{N} B(I,J) * C(J,K)$$

where N = the number of columns in B and rows in C

Example

Here's an example of using matrix multiplication to find total sales for four bus routes using old and new prices –

Matrix A – Ticket Sales by Route

Route	Single Trip	Round Trip	Commuter
1	143	200	18
2	49	97	24
3	314	77	22
4	82	65	16

7

Matrix B – Ticket Prices

	Old Price	New Price
Single Trip	.25	.30
Round Trip	.45	.50
Commuter	18.00	17.00

Matrix A, a 4 by 3 matrix, is multiplied by Matrix B, a 3 by 2 matrix, resulting in Matrix C, a 4 by 2 matrix.

Matrix C – Total Sales by Route

Route	Old	New
1	449.75	448.90
2	487.90	471.20
3	509.15	506.70
4	337.75	329.10

Here is the program used to perform the multiplication –

```
   10   OPTION BASE 1
   20   DIM A(4,3),B(3,2),C(4,2)
   30   DATA 143,200,18,49,97,24,314,77,22,82,65,16
   40   DATA 0.25,0.3,0.45,0.5,18,17
   50   MAT READ A,B
 • 60   MAT C=A*B
   70   FIXED 2      ! FOR MONETARY OUTPUT
   71   MAT PRINT A,B
   80   MAT PRINT C
   90   END
```

Here are some things to remember when using the matrix multiply statement –

- The result matrix can't be the same as either of the two operand matrices.
- The number of columns of the first operand matrix must equal the number of rows of the second operand matrix.
- Either or both of the operand matrices can be a vector. In this case, the result matrix must also be a vector.

7

Inverse of a Matrix

The inverse of a square matrix can be found by using the MAT...INV statement –

MAT result matrix = INV operand matrix

If the determinant of the operand matrix (see the DET function) is zero, the matrix doesn't have an inverse. No warning is given to indicate this condition and a meaningless inverse is calculated. The best way to check the inverse is to multiply the original matrix by the inverse using matrix multiplication. The result should be close to an identity matrix.

The inverse of a matrix is useful for solving systems of equations.

Example

$$3X + 4Y = 47$$
$$2X + 2Y = 28$$

These two equations can be represented as matrices –

```
MAT C = A * B
```

$$\overset{A}{\begin{bmatrix} 3 & 4 \\ 2 & 2 \end{bmatrix}} \overset{B}{\begin{bmatrix} X \\ Y \end{bmatrix}} = \overset{C}{\begin{bmatrix} 47 \\ 28 \end{bmatrix}}$$

The solution (the values of X and Y) is determined by multiplying both sides of the equation by the inverse of A. The following program was used to solve the system of equations –

```
  10   REM  FINDING SOLUTION TO SYSTEM OF 2 EQUATIONS
  20   OPTION BASE 1
  30   DIM A(2,2),B(2,1),C(2,1),D(2,2)
  40   DATA 3,4,2,2,47,28
  50   MAT READ A
  60   MAT READ C
• 70   MAT D=INV(A)
  80   PRINT "MATRIX D, THE INVERSE OF MATRIX A:",D(*);
  90   MAT B=D*C
  100  PRINT "MATRIX B, THE VALUES OF X AND Y:",B(*)
  110  END
```

```
MATRIX D, THE INVERSE OF MATRIX A:
-1   2

 1  -1.5


MATRIX B, THE VALUES OF X AND Y:
 9

 5
```

Transpose of a Matrix

The transpose of a matrix can be found by using the MAT...TRN statement –

MAT result matrix = TRN operand matrix

The transpose of a matrix has the same elements as the original, but columns become rows, and rows become columns.

Example

$$\text{Matrix X} = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\text{TRN(X)} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \\ 6 & 3 \\ 8 & 4 \end{bmatrix}$$

The result matrix is redimensioned, if necessary.

Column Sums

The sums of all the columns of a matrix can be found by using the MAT...CSUM statement –

MAT result vector = CSUM operand matrix

Each element in the result vector is the sum of the corresponding column of the operand matrix.

Example

$$\text{Matrix A} = \begin{bmatrix} 2 & 5 & 7 \\ 9 & 8 & 1 \end{bmatrix}$$

```
MAT X= CSUM A
```

$$\text{Vector X} = \begin{bmatrix} 11 & 13 & 8 \end{bmatrix}$$

The result is redimensioned, if necessary.

7

Row Sums

The sums of all the rows of a matrix can be found by using the MAT...RSUM statement –

MAT result vector = RSUM operand matrix

Each element in the result vector is the sum of the corresponding row of the operand matrix.

Example

$$\text{Matrix B} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}$$

```
MAT C = RSUM(B)
```

$$\text{Vector C} = \begin{bmatrix} 12 \\ 9 \end{bmatrix}$$

The result vector is redimensioned, if necessary.

Array Functions

There are five array functions which each return a number that provides information about an array. These are covered next. Examples showing the array functions follow the descriptions of all the functions.

SUM Function

7

The SUM function returns the sum of all the elements in an array.

SUM operand array

ROW Function

The ROW function returns the number of rows in the array according to its current size. The number of rows corresponds to the subscript which is second from the right.

ROW operand array

COL Function

The COL (column) function returns the number of columns in the array according to its current size. The number of columns corresponds to the rightmost subscript.

COL operand array

A vector always has one column.

DOT Function

The DOT function returns the inner (dot) product of two vectors.

DOT (vector name, vector name)

The two vectors must have the same working size. The inner product is the sum of the products of corresponding elements.

Example

A = 2   B = 1
4   2
6   4

DOT (A, B) = (2*1) + (4*2) + (6*4) = 34

DET Function

7

The DET (determinant) function returns the determinant of the specified matrix or of the last matrix which was inverted using the MAT...INV statement. No error given if there is no inverse.

DET
DET operand matrix

If a matrix is not specified, the determinant of the last inverted matrix is returned. This method uses less memory because the determinant is a by-product of the inversion operation.

Examples

Here are some examples of array functions –

```
10    REM    THIS PROGRAM SHOWS DET FUNCTION
20    OPTION BASE 1
30    DIM A(2,2),B(2,2)
40    INPUT "4 VALUES FOR MATRIX A?",A(*)
50    MAT B=INV(A)            ! INVERT MATRIX A
• 60    IF DET=0 THEN 110      ! USE DETERMINANT OF MATRIX A
70    PRINT "ORIGINAL MATRIX:",A(*)
80    PRINT "INVERSE OF MATRIX:",B(*)
• 90    PRINT "DETERMINANT OF INVERSE: ";DET(B)
100   STOP
110   PRINT "DETERMINANT IS 0; THERE IS NO INVERSE"
120   END
```

```
ORIGINAL MATRIX:
 4                  3

 2                  1


INVERSE OF MATRIX:
-.5                 1.5

 1                 -2


DETERMINANT OF INVERSE: -.5
```

```
10    REM    This program shows ROW, COL and SUM functions
20    OPTION BASE 1
30    DIM Totals(5,5)
40    REM  The following data items are daily totals
50 Number_done:   DATA 4,2,5,6,9,7,8,10,5,16,44,15,18
60 More_data:     DATA 1,5,9,12,4,6,18,7,10,5,8,7,4
70    MAT READ Totals    ! This array is number of widgits
                           made each day by each employee
80    INPUT "NUMBER OF EMPLOYEES TO BE TESTED?",E
90    INPUT "NUMBER OF DAYS TO TEST?",D
100   REDIM Totals(E,D)
• 110   PRINT ROW(Totals);" EMPLOYEES"
• 120   PRINT COL(Totals);" DAYS TESTED"
• 130   PRINT "TOTAL WIDGETS MADE IN TEST PERIOD:";SUM(Totals)
140   END
```

```
 3  EMPLOYEES
 2  DAYS TESTED
TOTAL WIDGETS MADE IN TEST PERIOD: 33
```

7

Redimensioning an Array

When an array is redimensioned, it is given a new working size. If the working size is smaller than the physical size, the remaining elements are ignored, but are still part of the array. Thus, when new values are assigned to elements of a redimensioned array, the values of the unused elements are not changed.

A redimensioned array must retain the same number of dimensions as orginally specified. Also, the total number of elements can't exceed the number originally specified.

Redimensioning of an array can be explicitly specified in many of the array statements. MAT INPUT and MAT... IDN are two examples.

Redimensioning can also be implicitly specified in many of the array operation statements. For example, adding the elements of two 3x3 arrays and storing the sums in a 5x5 array causes the result array to be redimensioned.

The following program illustrates how redimensioning of an array is accomplished –

```
   10   OPTION BASE 1
   20   DIM A(3,5)            ! A 3 by 5 array
   30   MAT A=(7)             ! All elements equal 7
   40   PRINT "ARRAY A--3 BY 5",A(*);
 • 50   MAT A=CON(2,2)        ! Redimension to 2 by 2
   60                         ! Elements in new working size =1
   70   PRINT "ARRAY A--NOW A 2 BY 2";A(*);
 • 80   REDIM A(3,5)          ! Back to original size
   90   PRINT "ARRAY A--BACK TO A 3 BY 5";
   91   PRINT "---NOTE WHICH ELEMENTS CHANGED"
   100  MAT PRINT A;
   110  END
```

```
ARRAY A--3 BY 5
 7  7  7  7  7

 7  7  7  7  7

 7  7  7  7  7

ARRAY A--NOW A 2 BY 2
 1  1

 1  1

ARRAY A--BACK TO A 3 BY 5---NOTE WHICH ELEMENTS CHANGED
 1  1  1  1  7
 ------------
 7  7  7  7  7

 7  7  7  7  7
```

7

The REDIM Statement

A new working size for an array can be established by using the REDIM statement.

REDIM array variable (redim subscripts) [, array variable (redim subscripts), ...]

The number of dimensions can't change. The total number of elements can't exceed the number originally dimensioned.

Here are the characteristics of redim subscripts –

- A subscript can be any numeric expression
- One subscript is used to specify the upper bound of a dimension.
- Two subscripts separated by a colon are used to specify the upper and lower bounds of a dimension.
- A comma is used to separate the subscript(s) for each dimension.

Here are some example REDIM statements –

```
  10   REM  THESE ARE EXAMPLE REDIM STATEMENTS
  20   DIM A(5,5),B(2,5,6),C(10)
• 30   REDIM B(1,4,-3:1)   ! UPPER & LOWER BOUNDS CAN BE SPECIFIED
  40   X=2
  50   Y=3
• 60   REDIM A(X,Y),C(7)   ! SUBSCRIPTS CAN BE NUMERIC EXPRESSIONS
  70   END
```

7

Chapter 8
Branching and Subroutines

Introduction

Normal program execution is in sequential order from lowest-numbered line to highest numbered line. Branching and subroutines are two methods of altering the normal flow of program execution.

The following topics and statements are covered in this chapter –

- Unconditional Branching – GOTO and ON...GOTO
- Conditional Branching – IF...THEN
- Looping – FOR, NEXT
- Subroutines – GOSUB and ON...GOSUB
- Defining Functions – DEF FN
- Branching using Special Function Keys – ON KEY

The following parameters are used in this chapter and can be numeric expressions –

initial value
final value
increment value
priority
key number

Unconditional Branching

The GOTO and ON...GOTO statements provide unconditional branching by transferring control to a specified line.

The GOTO Statement

The GOTO statement specifies a higher or lower-numbered line in the current program segment where execution is to be transferred –

GOTO line identifier

Examples

Here's an example using GOTO to branch to both higher-numbered and lower-numbered lines –

```
  10    X=5
• 20    GOTO Print                    ! BRANCHING TO A LABEL
  30    INPUT "NEW VALUE OF X?",X
  40    PRINT "X NOW EQUALS";X
  50    STOP                          ! STOP PREVENTS INFINITE LOOP
  60 Print:     PRINT "X EQUALS";X
• 70    GOTO 30                       ! BRANCHING TO A LINE NUMBER
  80    END
```

The ON...GOTO Statement

The ON...GOTO (computed GOTO) statement allows control to be transferred to one of one or more statements in the current program segment based on the value of a numeric expression –

ON numeric expression GOTO line identifier list

The numeric expression is evaluated and rounded to an integer. A value of one causes control to be transferred to the first statement specified in the list; a value of two causes control to be transferred to the second statement specified in the list, and so on.

8

Example

```
  10   INPUT "FULL, MEDIUM OR OUT-OF-STOCK?(1,2 OR 3)",R
• 20   ON R GOTO 30,50,Problem
  30   PRINT "FULL STOCK; DO NOTHING"
  40   STOP
  50   PRINT "MEDIUM STOCK; IT NEEDS TO BE WATCHED"
  60   STOP
  70 Problem:    PRINT "NO STOCK LEFT--RE-ORDER!!!"
  80   END
```

If the value of the numeric expression is less than one or greater than the number of line identifiers in the list, ERROR 19 (improper value) occurs.

Example

In the following example, when line 20 is executed for the third time, the value of I exceeds the number of line identifiers in the list.

```
  10   I=1
• 20   ON I GOTO 30,30,60
  30   PRINT "I=";I
  40   I=I*2                     ! Value of I is 1, 2, then 4
  50   GOTO 20
  60   PRINT "I GREATER THAN 3"  ! This line is never reached
  70   END


I= 1
I= 2
ERROR 19 IN LINE 20
```

Summary

Here are some facts to remember concerning the GOTO statements –

- All lines specified by GOTO statements must be in the current program segment.
- If the line specified as the destination of a branch is not an executable statement (see Chapter 4 for an explanation of executable statements), program control is transferred to the first executable statement following the specified line. However, execution pauses at the specified line if STEP is being used.
- GOTO statements are programmable only; they can't be executed from the keyboard.

8

Conditional Branching

The IF...THEN statement is used to provide branching which is dependent on a specified condition –

IF numeric expression THEN line identifier

If the numeric expression has a value other than zero, it is considered true and branching to the specified line occurs. If it has a value of zero (false) execution continues with the line following the IF...THEN statement.

Examples

```
  10    INPUT A
• 20    IF A THEN 50            ! BRANCHING ONLY IF A IS NOT 0
  30    PRINT "A=0"
  40    GOTO 10
  50    PRINT "A IS NOT 0 -- A=";A
  60    END
```

The IF...THEN statement is used most often with relational and logical operators.

```
  5     Pay=4.75
  10    INPUT "HOURS WORKED?",H
• 20    IF H>40 THEN Overtime
  30    DISP "NO OVERTIME"
  40    GOTO 10
  50 Overtime:     O=H-40
  60    PRINT O*Pay*1.5
  70    STOP
```

```
  10    INPUT "PRINTER SELECT CODE?",Psc
• 20    IF (Psc<0) OR (Psc>16) OR (Psc=15) THEN 70
  30    REM  LINE 20 CHECKS FOR INVALID SELECT CODE
  40    PRINTER IS Psc
  50    PRINT "PRINTER IS SELECT CODE";Psc
  60    STOP
  70    PRINT Psc;"IS INVALID SELECT CODE; ENTER ANOTHER ONE"
  80    BEEP
  90    GOTO 10
```

Another form of the IF...THEN statement provides conditional execution of a statement without necessarily branching –

IF numeric expression THEN statement

When the value of the numeric expression is not equal to zero (true) the statement is executed. When the value of the numeric expression is zero (false), execution continues with the following line.

Example

```
  10   INPUT "ENTER VALUES FOR X AND Y",X,Y
• 20   IF X=Y THEN PRINT "X=Y=";X
  30   PRINT "X AND Y ARE NOT EQUAL"
  40   END
```

All executable BASIC statements are allowed after THEN with the following exceptions –

FOR statement
NEXT statement
IF statement

The following statements are not allowed after THEN because they are declaratory statements, not executable statements –

COM statement
DATA statement
DEF FN statement
DIM statement
END statement
FN END statement
IMAGE statement
INTEGER statement
OPTION BASE statement
REAL statement
REM statement
SHORT statement
SUB statement
SUBEND statement

8

Looping

Repeatedly executing a series of statements is known as **looping**. The FOR and NEXT statements are used to enclose a series of statements in a **FOR-NEXT loop**, allowing them to be repeated a specified number of times.

```
FOR loop counter = initial value TO final value [STEP increment value]
.
.
.
NEXT loop counter
```

The FOR statement defines the beginning of the loop and specifies the number of times the loop is to be executed. The loop counter must be a simple numeric variable.

The initial, final, and increment values can be any numeric expression. If the increment value is not specified, the default value is one.

Examples

Here's an example of a FOR-NEXT loop –

FOR-NEXT loop range (lines 10–50)

```
10   FOR I=1 TO 5           ! First statement of loop
20      PRINT "I EQUALS";
30      PRINT I
40      REM   Indenting the body of the loop makes
              the program easier to read
50   NEXT I                 ! Last statement of loop
60   PRINT "FINISHED WITH THE LOOP;  I NOW EQUALS";I
70   END
```

```
I EQUALS 1
I EQUALS 2
I EQUALS 3
I EQUALS 4
I EQUALS 5
FINISHED WITH THE LOOP;  I NOW EQUALS 6
```

In this example, I is established as the loop counter and is set to 1 when the FOR statement is executed. The FOR-NEXT loop is executed 5 times – when I = 1, 2, 3, 4 and 5. Each time the NEXT statement is executed, the value of I is incremented by 1, the default increment value. When the value of I exceeds the final value (when I = 6) the loop is finished and execution continues with the statement following the NEXT statement.

The following examples show that differing FOR statements can perform the same task. In each example, the FOR-NEXT loop is executed ten times. Notice the value of the loop counter while the loop is executing and after it is complete.

-
```
10   FOR A=2 TO 12
20   NEXT A
30   PRINT "A=";A
40   END


A= 13
```

-
```
10   X=10
20   Y=100
30   FOR A=X TO Y STEP 10  ! Initial & final values
                             can be expressions
40   NEXT A
50   PRINT "A=";A
60   END


A= 110
```

-
```
10   FOR A=10 TO 1 STEP -1   ! It is easy to
                               decrement the counter
20   NEXT A
30   PRINT "A=";A
40   END

A= 0
```

-
```
10   FOR A=1 TO 19 STEP 2    ! VALUE OF COUNTER IS 1,3,5,ETC,
20   NEXT A
30   PRINT "A=";A
40   END


A= 21
```

8

Programming Hint

An often overlooked aspect of FOR-NEXT looping is that the actual value of the counter when the loop is complete does not equal the final value.

The advantages of using FOR-NEXT looping instead of an IF...THEN statement are shown in the following examples where the numbers 1 through 1000 are printed in succession. The program that uses the FOR-NEXT loop is easier to key in and uses less memory.

IF...THEN statement

```
10    I=1
20    IF I>1000 THEN 60          ! SHOULD LOOP BE EXECUTED AGAIN?
30    PRINT I
40    I=I+1                      ! INCREMENTS THE COUNTER
50    IF I<1000 THEN 30          ! BACK TO BEGINNING OF LOOP
60    END
```

FOR-NEXT loop

```
10    FOR I=1 TO 1000
20       PRINT I
30    NEXT I      ! Increment counter, test to see if loop should
                    be executed again, branch to start of loop
40    END
```

The initial, final and increment values are calculated upon entry into the loop; the calculated values are used throughout execution of the loop. The following example illustrates that the initial, final and increment values can be changed without affecting the number of times the loop is repeated.

```
  10    A=3
  20    INPUT B
• 30    FOR X=A TO A*B STEP B-2  ! STEP value can be an expression
  40       A=A+X
  50       B=B-1                ! Lines 40 & 50 don't affect initial
                                  and final values of the counter
  60       PRINT X,A,B
  70    NEXT X
  80    END
```

8

If 4 is input for the value of B, the loop is repeated 5 times and the output is –

```
3             6             3
5             11            2
7             18            1
9             27            0
11            38           -1
```

Nesting

FOR-NEXT loops can be nested. When one loop is contained entirely within another, the inner loop is said to be nested. The following example illustrates assigning values to an array using a nested FOR-NEXT loop.

```
  10   OPTION BASE 1
  20   DIM Array(4,3)
• 30   FOR X=1 TO 4     ! This loop controls 1st (left) subscript
• 40     FOR Y=1 TO 3   ! This loop control 2nd (right) subscript
  50       Array(X,Y)=X+Y
  60     NEXT Y
  70   NEXT X
  80   MAT PRINT Array
  90   END

   2                   3                   4

   3                   4                   5

   4                   5                   6

   5                   6                   7
```

A FOR-NEXT loop can not overlap another.

Correct Nesting

```
10   FOR I=1 TO 3            ! BEGINNING OF OUTER LOOP
20      FOR J=4 TO 6         ! BEGINNING OF INNER LOOP
30         PRINT I,J
40      NEXT J               ! END OF INNER LOOP
50   NEXT I                  ! END OF OUTER LOOP
60   END
```

Incorrect Nesting

```
10   FOR I=1 TO 3            ! BEGINNING OF OUTER LOOP
20      FOR J=4 TO 6         ! BEGINNING OF INNER LOOP
30         PRINT I,J
40   NEXT I                  ! END OF OUTER LOOP
50      NEXT J               ! END OF INNER LOOP
60   END
```

In the incorrect nesting example, the I loop is activated and then the J loop is activated. The J loop is cancelled when `NEXT I` is executed because it's an inner loop. When the I loop is completed and `NEXT J` is accessed, `ERROR 6 IN LINE 50` is displayed. This is because the J loop was cancelled and was not reactivated after the last I loop.

FOR-NEXT Loop Considerations

Execution of FOR-NEXT loops should always start with the FOR statement. Branching into the middle of a loop produces ERROR 6 when NEXT is executed because no corresponding FOR statement was.

Execution of loops normally ends with the NEXT statement. It is permissible to transfer control out of the loop using a statement within the loop. After an exit is made through this method, the current value of the counter is retained and is available for later use in the program. After leaving a FOR-NEXT loop, it is permissible to re-enter the loop either at a statement within the loop, or at the FOR statement, thereby reinitializing the counter.

Subroutines

Many times, the same sequence of statements is executed in many places within a program. A subroutine allows the group of statements to be keyed in only once and to be accessed from different places in a program. A subroutine return pointer is kept by the system in the execution stack to indicate where execution is to return to when the subroutine is complete. The GOSUB and ON...GOSUB statements are used to access subroutines.

The GOSUB Statement

The GOSUB statement transfers control to the subroutine which begins at the specified line in the current program segment —

GOSUB line identifier

A subroutine ends logically with the RETURN statement —

RETURN

8

which transfers control back to the statement immediately following the GOSUB statement.

Example

Here is an example of accessing a subroutine from different places in a program –

```
      10   INPUT "HOURS WORKED?",Hours
      20   PRINT "TOTAL HOURS WORKED IS";Hours
      30   Rate=5.25
      40   Overtime=.25
    • 50   GOSUB 80        ! Branch to subrouting at line 80
      60   PRINT "TOTAL PAY AT 5.25/HOUR IS";Pay
      70   GOTO 120        ! Branch around subroutine
     ┌80      Pay=Hours*Rate   ! First line of subroutine
Subroutine│90      Ohours=Hours-40
     │100     Pay=Pay+Ohours*Overtime
     └110     RETURN       ! Last line of subroutine; control
                             passes to line after GOSUB
      120  Rate=6.00
      130  Overtime=.29
    • 140  GOSUB 80        ! Branch to subroutine at line 60
      150  PRINT "TOTAL PAY AY 6.00/HOUR IS";Pay
      160  END
```

```
TOTAL HOURS WORKED IS 43
TOTAL PAY AT 5.25/HOUR IS 226.5
TOTAL PAY AY 6.00/HOUR IS 258.87
```

The ON...GOSUB Statement

The ON...GOSUB (computed GOSUB) statement allows any of one or more subroutines in the current program segment to be accessed based on the value of a numeric expression –

ON numeric expression GOSUB line identifier list

The numeric expression is evaluated and rounded to an integer. A value of one causes the subroutine at the first identifier in the list to be accessed; a value of two causes the subroutine at the second identifier in the list to be accessed and so on.

Example

```
  10   FOR X=1 TO 3
• 20   ON X GOSUB 50,70,90    ! EACH PASS THROUGH LOOP ACCESSES
                                DIFFERENT SUBROUTINE
  30   NEXT X
  40   STOP                 ! STOP PREVENTS ACCESSING OF SUBROUTINE
• 50   PRINT "FIRST SUBROUTINE"
  60   RETURN
• 70   PRINT "SECOND SUBROUTINE"
  80   RETURN
• 90   PRINT "THIRD SUBROUTINE"
  100  RETURN
  110  END


  FIRST SUBROUTINE
  SECOND SUBROUTINE
  THIRD SUBROUTINE
```

If the value of the numeric expression is less than one or greater than the number of line identifiers in the list, `ERROR 19` occurs.

A second subroutine can be entered before the `RETURN` of the first is executed.

Example

```
  10   INPUT "INPUT VALUES FOR X AND Y",X,Y
  20   GOSUB 40
  30   STOP
• 40      PRINT X,Y
  50      IF X<Y THEN GOSUB 70    ! Branch to 2nd subroutine
  60      RETURN
• 70         PRINT "X<Y"          ! First line of 2nd subroutine
  80         RETURN
  90   END
```

The subroutine at line 70 is accessed before the one at line 40 is completed.

Subroutines can be accessed in this manner as much as available memory allows. Doing it too many times can cause the execution stack to become too large, thus causing a memory overflow. See Appendix F for more information.

When a `RETURN` is executed, control returns to the subroutine which was entered most recently.

Summary

Here are some facts to remember concerning subroutines and the GOSUB statements –

- A subroutine should always end with a RETURN statement.
- The GOSUB statements are programmable only; they can't be executed from the keyboard.
- All subroutines specified must be in the current program segment.

Defining a Function

If a numeric or string operation has to be evaluated several times, it is convenient to define it as a function. This is done using the DEF FN statement which specifies a user-defined function, returns a single value as the value of the function and can be used like a system function. The simplest form is the single-line function which can be used to define a numeric or string function (there is also a multiple-line function; see Chapter 9).

To define a numeric function –

DEF FN function name [(formal parameter list)] [1]= numeric expression

To define a string function –

DEF FN function name $ [(formal parameter list)] = string expression

The function name must be a valid name. The expression can include both parameters and variables.

Once the function is defined, it is used by referencing it and supplying values by using –

FN function name [(pass parameter list) [1]]

for a numeric function, or –

FN function name $ [(pass parameter list)]

for a string function.

8

[1] Formal and pass parameter lists are discussed in Chapter 9.

When the function reference, FN, is encountered, control is transferred to the corresponding DEF FN. The values of the pass parameters are substituted for the formal parameters and the expression is evaluated. Its value is returned as the value for the referencing syntax. See Chapter 9 for a more detailed explanation of parameters.

Example

Here's an example use of a single-line function –

Say that a program contains these lines –

```
30    Expenses(I)=Room+Car+Food+Gas+Ticket
80    Expenses(I)=Room+Car+Food+Gas+Phone
200   Expenses(I)=Room+Car+Food+Gas+Ski_rental
```

By defining –

variables

```
20    DEF FNExp(X)=Room+Car+Food+Gas+X        ! X IS FORMAL PARAMETER
```

function name   formal parameter

Lines 30, 80, and 200 can be simplified –

```
30    Expenses(I)=FNExp(Ticket)        ! Ticket IS PASS PARAMETER
80    Expenses(I)=FNExp(Phone)
200   Expenses(I)=FNExp(Ski_rental)
```

referencing syntax

NOTE

Single-line functions are local to the program segment in which they are defined. The DEF FN statement can't contain a reference to itself.

Multiple-line function subprograms can also be used to define a function; see Chapter 9.

Summary

Here are some facts to remember when using single-line functions –

- The name of the function must be a valid name.
- The expression used to define the function can contain both variables and formal parameters.
- A single-line function can't contain a reference to itself; that is recursion.
- Single-line functions are local to the program segment in which they are defined. See Chapter 9 for more information.

Branching Using Special Function Keys

The 32 special function keys can be used to interrupt a running program and cause branching. This branching capability is useful for a program which is very user-oriented. Each key can be defined to cause a specific branch, so that the user can steer the program the way he wants it. For example, a 'menu' of various routines can be displayed and accessed using special function keys. Here is where a blank key overlay can be used.

This interrupt capability is declared with an ON KEY# statement which specifies the branching operation and which SFK it relates to.

> ON KEY# key number [, priority] GOTO or GOSUB line identifier
> ON KEY# key number [, priority] CALL subprogram name[1]

The key number is an integer in the range 0 through 31. When a key is pressed and an ON KEY# has been declared for it, the specified branching occurs if the specified priority exceeds current system priority. System priority remains unchanged if GOTO is specified and is set to the indicated priority if GOSUB or CALL is specified.

8

1 Parameters can't be passed. CALL is explained in Chapter 9.

Priority

The priority determines the order in which multiple interrupts are handled. The range of priority is 1 through 15. The higher the priority, the sooner the interrupt is serviced. If no priority is specified, it is assumed to be 1. Pressing a key with a higher priority interrupts a routine enabled by a key with a lower priority. The lower priority routine is finished after the high priority one is complete if the higher priority routine was entered using GOSUB or CALL. Priority is also discussed near the end of Chapter 4.

Example

Here's an example that illustrates ON KEY# and priority –

```
10   REM ************** INTERRUPT WITH KEYS *********************
20   ON KEY #1,4 GOSUB 360          !PRIORITY ONE
30   PRINT PAGE,SPA(20);"ON KEY#1,4 GOSUB 360"
40   ON KEY #2,3 GOSUB 460          !PRIORITY TWO
50   PRINT SPA(20);"ON KEY#2,3 GOSUB 460"
60   ON KEY #3,2 GOSUB 560          !PRIORITY THREE
70   PRINT SPA(20);"ON KEY#3,2 GOSUB 560"
80   ON KEY #4,1 GOSUB 660          !PRIORITY FOUR
90   PRINT SPA(20);"ON KEY#4,1 GOSUB 660"
100  PRINTER IS 6
110  PRINT LIN(5)
120  DISP "WAITING"
121  GOTO 120
130  STOP
140  REM ***** INTERRUPT ROUTINES ******************************
360  FOR I=1 TO 10
370  PRINT "KEY 1";I;TAB(15);"PRIORITY 4";TAB(30);"PRIORITY 4";
375  PRINT TAB(45);"PRIORITY 4";TAB(60);"PRIORITY 4"
380  NEXT I
390  RETURN
460  FOR J=1 TO 10
470  PRINT "KEY 2";J;TAB(15);"PRIORITY 3";TAB(30);"PRIORITY 3";
475  PRINT TAB(45);"PRIORITY 3"
480  NEXT J
490  RETURN
560  FOR K=1 TO 10
570  PRINT "KEY 3";K;TAB(15);"PRIORITY 2";TAB(30);"PRIORITY 2"
580  NEXT K
590  RETURN
660  FOR L=1 TO 10
670  PRINT "KEY 4";L;TAB(15);"PRIORITY 1"
680  NEXT L
690  RETURN
710  END
```

RUN

```
                    ON KEY#1,4 GOSUB 360
                    ON KEY#2,3 GOSUB 460
                    ON KEY#3,2 GOSUB 560
                    ON KEY#4,1 GOSUB 660
```

Pressing keys 4, 3, 2, 1 produces –

```
KEY 4 1     PRIORITY 1
KEY 4 2     PRIORITY 1
KEY 3 1     PRIORITY 2   PRIORITY 2
KEY 2 1     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 1 1     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 2     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 3     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 4     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 5     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 6     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 7     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 8     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 9     PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 1 10    PRIORITY 4   PRIORITY 4   PRIORITY 4   PRIORITY 4
KEY 2 2     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 3     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 4     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 5     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 6     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 7     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 8     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 9     PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 2 10    PRIORITY 3   PRIORITY 3   PRIORITY 3
KEY 3 2     PRIORITY 2   PRIORITY 2
KEY 3 3     PRIORITY 2   PRIORITY 2
KEY 3 4     PRIORITY 2   PRIORITY 2
KEY 3 5     PRIORITY 2   PRIORITY 2
KEY 3 6     PRIORITY 2   PRIORITY 2
KEY 3 7     PRIORITY 2   PRIORITY 2
KEY 3 8     PRIORITY 2   PRIORITY 2
KEY 3 9     PRIORITY 2   PRIORITY 2
KEY 3 10    PRIORITY 2   PRIORITY 2
KEY 4 3     PRIORITY 1
KEY 4 4     PRIORITY 1
KEY 4 5     PRIORITY 1
KEY 4 6     PRIORITY 1
KEY 4 7     PRIORITY 1
KEY 4 8     PRIORITY 1
KEY 4 9     PRIORITY 1
KEY 4 10    PRIORITY 1
KEY 4 1     PRIORITY 1
KEY 4 2     PRIORITY 1
KEY 4 3     PRIORITY 1
KEY 4 4     PRIORITY 1
KEY 4 5     PRIORITY 1
KEY 4 6     PRIORITY 1
KEY 4 7     PRIORITY 1
KEY 4 8     PRIORITY 1
KEY 4 9     PRIORITY 1
```

If multiple ON KEY# declaratives have the same priority, the declarative with the highest key number is given preference when two keys are pressed simultaneously.

ON KEY# statements which specify GOTO or GOSUB are active only in the program segment in which they were declared. ON KEY# declaratives are deactivated while a program is waiting for a response to an INPUT, LINPUT or EDIT statement and after PAUSE is executed.

The ON KEY# declarative holds for a key until another declarative for the same key, SCRATCH A, SCRATCH, SCRATCH P, SCRATCH V, SCRATCH C, (RUN) or OFF KEY# is executed –

OFF KEY# key number

If a certain SFK routine has not been completed and that key is pressed again, the key won't be acknowledged until the original interrupt is completed.

If a special function key has both ON KEY# and typing aid definitions, the ON KEY# has precedence while the program is running. Remember, waits caused by PAUSE, INPUT, LINPUT, and EDIT temporarily deactivate the ON KEY#, so any typing aid definition is active at that time.

Summary

Here are some facts to remember when using ON KEY –

- The range of priority is 1 through 15.
- System priority is not changed when GOTO is specified.
- ON KEY declaratives are temporarily deactivated by INPUT, LINPUT, EDIT and PAUSE.
- An ON KEY declarative is permanently deactivated by another ON KEY for that particular key, SCRATCH, SCRATCH A, SCRATCH C, SCRATCH V, SCRATCH P, (RUN) or OFF KEY.

8

Chapter 9
Subprograms

Introduction

Many programs include various routines that require a long series of statements (such as routines for sorting or computing compound interest). These routines must sometimes be repeated many times in one program. To avoid rewriting a routine each time it is needed, a **subprogram** can be used. A subprogram is a set of statements that performs a certain task under the control of the calling program segment.

A subprogram enables you to repeat an operation many times, substituting different values each time the subprogram is called. Subprograms can be called at almost any point in a program, and are convenient and easy to use. Subprograms can give greater structure and independence to a program. A main program may be a sort of ''skeleton'' program which calls many subprograms, which, in turn, can call other subprograms.

Subprograms can also be used to conserve memory through the use of local variables and dynamic memory allocation. These concepts are covered later in the chapter. See Appendix F also.

The following topics and statements are covered in this chapter –

- Parameters
- Multiple-line Function Subprograms (DEF FN)
- Subroutine Subprograms (SUB)
- Using COM in subprograms
- Declaring variables in subprograms
- Dynamic memory allocation – local variables
- Using data files
- Busy lines

Types of Subprograms

There are two types of subprograms.

- The **multiple-line user-defined function subprogram** is designed to return a single numeric or string value to the calling program and is used like system functions such as SIN or CHR$. It is defined using the DEF FN statement. (The DEF FN statement is also used to define a single-line function; see Chapter 8.)
- A **subroutine subprogram** is designed to perform a specific task under the control of the calling program segment. It is defined using the SUB statement.

Terms

There are a few terms which are important to know when dealing with subprograms.

Main program – The central part of a program from which subprograms can be called is known as the main program. When you press (RUN), you access the main program. The main program can't be called by a subprogram.

Program segment – The main program and each subprogram are known as program segments. Every program segment is independent of every other program segment. Subprograms come after the main program; that is, they are higher numbered. Subprograms are called by the main program or another subprogram. See Appendix F for the relationship between memory allocation and subprograms.

Calling program – When a subprogram is being executed, the program segment (main program or subprogram) which called the subprogram is known as the calling program. Control returns to the calling program when the subprogram is completed.

Current environment – The program segment which is being executed is known as the current environment. See the section on subprogram considerations at the end of this chapter for a discussion of variables and various conditions as they relate to subprograms and the current environment.

The following terms are used in the syntax descriptions in this chapter –

Name – a capital letter followed by 0 through 14 lowercase letters, numbers or the underscore character.

Parameters

Values are passed between a subprogram and the calling program using parameters. There are two kinds of parameters. **Formal parameters** are used in defining the subprogram. **Pass parameters** are used to pass values from the calling program to the subprogram. Each pass parameter corresponds to a formal parameter.

Formal Parameters

The formal parameter list is used in a SUB or DEF FN statement to define the subprogram variables, and to relate them to calling program variables. It can include non-subscripted numeric and string variable names, array identifiers and file numbers (see Mass Storage, Chapter 10) in the form: # file number. Parameters must be separated by commas and the parameter list must be enclosed in parentheses.

Numeric type – REAL, SHORT, INTEGER – can be declared in a formal parameter list by placing the type word before a parameter or group of parameters. Here is an example of a formal parameter list –

```
270  SUB X(A,B$,INTEGER C(*),D,SHORT E,F,#3,G)
```

In this example, the array C and simple variable D are declared as integer precision, E and F are short precision and A and G are full precision. Type words are cumulative like in a COM statement. For example, if INTEGER is specified, all variables following it are declared as be integers until a string, a file number or another type word is specified.

Pass Parameters

The pass parameter list is used in calling the subprogram (using CALL or FN) and includes numeric and string variable names, array identifiers, numeric expressions and file numbers in the form: # file number. Parameters must be separated by commas. The pass parameter list must also be enclosed in parentheses.

All array variables in the pass parameter list must be defined within the calling program. That is, arrays must have been dimensioned, either implicitly or explicitly.

What Happens

When a subprogram is called, (with CALL or FN) each formal parameter is associated with and assigned the value of the pass parameter which is in the corresponding position in the pass parameter list. The parameter lists must have the same number of parameters; the parameters must match in type—numeric or string, simple or array. The following example shows a formal parameter list, (SUB, line 300) and two corresponding pass parameter lists (CALL, lines 70 and 150).

```
10    INTEGER C(2,2),D(2,2)
70    CALL X(A,B$,C(*),(D(1,2)),3,E+F,#6,(G))
75    !     CALL contains the pass parameters
150   CALL X(5,(C$[1,12]),D(*),4,(X(4,3)),(A),#2,E*3)
290   END
300   SUB X(X,Y$,INTEGER Z(*),SHORT K,L,M,#3,N)
305   !    SUB contains the formal parameters
310   SUBEND
```

Notice the correspondence between pass and formal parameters. Notice also that the arrays C and D were declared (line 10) before being passed.

Parameters are passed either by **reference** or by **value**. When a parameter is passed by reference, the corresponding formal parameter shares the same memory area with the pass parameter. Thus, changing the value of the corresponding variable in the subprogram changes the corresponding value of the variable in the calling program.

When a parameter is passed by value, the variable defined by the corresponding formal parameter is assigned the value of the pass parameter and given temporary storage space in memory. Numeric and string expressions are necessarily passed by value. Enclosing a pass parameter in parentheses causes it to be considered an expression and thus passed by value, rather than by reference. Passing by value prevents the value of a calling program variable from being changed within a subprogram.

Examples

In the following example all parameters in line 80 are passed by value; those in line 130 are passed by reference.


```
80    CALL Active(Y+3,(X(2,4)),(X(1,4)),PI,(Y),(L$[20]))
85    !    Parameters in line 80 passed by value
130   CALL Active(Y,X(2,4),X(1,4),A,Z,L$)
135   !    Parameters in line 130 passed by reference
180   SUB Active(A,B,C,D,E,F$)
```

Here is an example of similar program segments. Notice the value of X in each case.

Pass by value

```
   10   A=1
•  20   X=FNZ((A))+A        ! PASS BY VALUE
   30   PRINT "X=";X
   40   STOP
   50   DEF FNZ(Y)
   60   Y=10
   70   RETURN Y
   80   END

   X= 11
```

Pass by reference

```
   10   A=1
•  20   X=FNZ(A)+A         ! PASS BY REFERENCE
   30   PRINT "X=";X
   40   STOP
   50   DEF FNZ(Y)
   60   Y=10               ! CHANGES VALUE OF A IN CALLING PROGRAM
   70   RETURN Y
   80   END


   X= 20
```

Any parameters passed by value are converted, if necessary, to the numeric type – REAL, SHORT, INTEGER – of the corresponding parameter in the formal parameter list. For example, say that `PI` is passed by value to an INTEGER formal parameter. Its value would be rounded to 3 when the subprogram is called.

Those passed by reference must match exactly, otherwise `ERROR 8` occurs. No conversion is made. In the following example, `C(*)`, `D(1,2)` and `G` (`CALL`, line 30) are passed by reference. Their corresponding formal parameters are of different types – `Z(*)` is INTEGER precision, `K` and `N` are SHORT precision. Thus, `ERROR 8` occurs.

```
10    DIM C(3,3),D(3,3)
30    CALL X(A,B$,C(*),D(1,2),G)  ! C(*),D(1,2) & G are passed
                                    by reference
290   END
300   SUB X(X,Y$,INTEGER Z(*),SHORT K,N)
305                               ! Only parameters passed by
                                    value can be converted
310   SUBEND

ERROR 8 IN LINE 30
```

Summary

Here are some facts to remember concerning parameters.

- Formal parameters are used in defining the subprogram (in the DEF FN or SUB statement) and can be simple variables, array identifiers or file numbers.
- Pass parameters are used in the calling program (FN or CALL statement) to pass values to the subprogram and can be single variables, array identifiers, expressions or file numbers.
- The parameter list must be enclosed in parentheses and all parameters must be separated by commas.
- Numeric type – INTEGER, SHORT and REAL can be declared in the formal parameter list.
- Parameters can be passed by reference or by value. Enclosing a pass parameter in parentheses causes it to be passed by value. Parameters passed by reference must match in numeric type.

Multiple-Line Function Subprograms

The multiple-line function subprogram is used to define a numeric or string function which returns a value (numeric or string) to the calling program. There are four syntax which are used with multiple-line function subprograms –

- DEF FN subprogram name [(formal parameter list)]
  DEF FN subprogram name $ [(formal parameter list)]

  The DEF FN statement is the first line of a user-defined multiple-line function subprogram. The second syntax is used for defining a string function. The subprogram name must be a valid name.

- FN END

  The FN END statement is the last statement in a multiple-line function subprogram.

- RETURN numeric expression
  RETURN string expression

  The RETURN statement specifies the value (numeric or string) which is to be returned to the calling program for the value of the function. RETURN also transfers control back to the calling program.

9

- FN subprogram name [(pass parameter list)]

  FN subprogram name $ [(pass parameter list)]

  FN is used to reference the subprogram, like saying SIN(X). When it is encountered, values are passed and control is transferred to the subprogram. FN can't appear in an input or output statement or in redim subscripts.

Examples

Here's an example of a numeric function –

```
  10   DIM C(2,2)              ! Must dimension arrays to pass them
  20   A=RND
  30   B=7
  40   MAT C=(8*RND)
• 50   Y=FNTotal(A,B,C(*))
  60   PRINT "TOTAL IS";Y      ! FNTotal can't be in PRINT statement
  70   END
• 80   DEF FNTotal(X,Y,Z(*))! DEF FN is first line in a multiple-
                              line function subprogram
• 90   RETURN SUM(Z)+X+Y       ! Transfers control back
• 100  FNEND                   ! FNEND is last line in a multiple-
                                line function subprogram


  TOTAL IS 35.1956728609
```

Here's an example of a string function –

```
  10   DIM C$[100],D$[100],A$[100]
  20   DATA J. SMITH, ENGINEER,7,B. JONES,BANJO PLAYER,9
  30   FOR I=1 TO 2
  40      READ C$,D$,E
• 50      A$=FNClassify$((C$),D$,E)   ! C$ PASSED BY VALUE
  60      PRINT A$
  70   NEXT I
  80   END
• 90   DEF FNClassify$(X$,Y$,Z)
• 100  RETURN X$&"'S JOB IS "&Y$&" WHICH IS ON LEVEL "&VAL$(Z)
• 110  FNEND


  J. SMITH'S JOB IS ENGINEER WHICH IS ON LEVEL 7
  B. JONES'S JOB IS BANJO PLAYER WHICH IS ON LEVEL 9
```

There can be more than one RETURN statement in a subprogram, but only one is executed each time the subprogram is executed. Here's an example based on the previous numeric function subprogram –

```
  10    DIM C(2,2)
  20    RANDOMIZE
  30    A=RND
  40    B=7
  50    MAT C=(8*RND)
  60    Y=FNTotal(A,B,C(*))
  70    PRINT "TOTAL (<=55) IS";Y
  80    END
  90    DEF FNTotal(X,Y,Z(*))
  100   A=SUM(Z)+X+Y
• 110   IF A>55 THEN RETURN 55      ! 55 IS MAXIMUM
  120   !  THERE CAN BE MORE THAN ONE RETURN, BUT ONLY ONE IS
           EXECUTED EACH TIME THE SUBPROGRAM IS ACCESSED
• 130   RETURN A
  140   FNEND


  TOTAL (<=55) IS 50.6967164637
```

References to multiple-line function subprograms are not allowed in input or output statements or in redim subscripts. For example, if line 60 of the previous example were changed to –

```
  60    PRINT A,B,FNTotal(A,B,C(*))
```

ERROR 39 would occur.

Remember, values of variables in the calling program can be changed from within a subprogram if the parameter is passed by reference. If, in the previous example of a string function subprogram, C$ had been passed by reference, its value would have been changed because the value of X$ was changed in the subprogram.

If a single-line and multiple-line function are defined with the same name and the name is referenced, the single-line function is the one that is accessed if it is defined within the calling program segment.

Subroutine Subprograms

Subroutine subprograms allow you to repeat a series of operations many times using different values or to break a large problem down into a series of smaller ones. A subroutine subprogram performs a specific task.

There are four statements which are used with subroutine subprograms –

- SUB subprogram name [(formal parameter list)]

  The SUB statement is the first statement of a subroutine subprogram. The subprogram name must be a valid name.

- SUBEND

  The SUBEND statement is the last line of a subroutine subprogram and transfers control back to the calling program.

- SUBEXIT

  The SUBEXIT statement can be used within the body of a subprogram to transfer control back to the calling program before SUBEND is executed.

- CALL subprogram name [(pass parameter list)]

  The CALL statement is used to transfer control and pass values to the subprogram.

Examples

Here is a simple example of a subroutine used to write a heading for data output. Notice that no parameters are passed.

```
• 10   CALL Heading
  20   END
• 30   SUB Heading    ! SUB is 1st line of subroutine subprogram
  40   PRINT TAB(11),"NAME";TAB(27),"AMOUNT"
  50   PRINT RPT$("_",40)
• 60   SUBEND         ! SUBEND is last line of subroutine sub-
                        program; passes control to calling program
```

```
          NAME            AMOUNT
________________________________________
```

Here is another example which manipulates the parameters and could be used to output a readable table when values are supplied –

```
  10   OPTION BASE 1
  20   DIM Prodnums(25),Percent(25)
  30   INPUT "TOTAL NUMBER OF PRODUCTS?",N
  40 Prodnums:     DATA 25,15,31,30,45,97,35
  50 Percents:     DATA 27,13,10,12,18,2,28
  60   MAT READ Prodnums(N)  ! REDIMS Prodnums TO # OF PRODUCTS
  70   RESTORE Percents      ! READ DATA FROM Percents
  80   FOR I=1 TO N
  90      READ Percent(I)
  100  NEXT I
• 110  CALL Table(Prodnums(*),Percent(*),N)
  120  END
• 130  SUB Table(Pr(*),Per(*),N)
  140  OPTION BASE 1          ! OPTION BASE VALUES MUST MATCH
  150  PRINT "PRODUCT","% OF SALES"
  160  FOR I=1 TO N           ! ONE LINE PER PRODUCT
  170     PRINT Pr(I),Per(I)
  180  NEXT I
• 190  SUBEND
```

```
PRODUCT             % OF SALES
 25                  27
 15                  13
 31                  10
 30                  12
 45                  18
```

The SUBEXIT statement is used to transfer control back to the calling program before SUBEND is executed. Here is an example –

```
  10   INPUT "VALUES FOR X AND Y",X,Y
• 20   CALL Pay(X,Y)
  30   PRINT "BACK FROM 1st SUBPROGRAM",LIN(1)
  40   A=DROUND(RND,1)
• 50   CALL Accounts(A)
  60   PRINT "BACK FROM 2nd SUBPROGRAM"
  70   END
• 80   SUB Pay(X,Y)
  90   IF X>Y THEN 120
  100  PRINT "X IS LESS THAN OR EQUAL TO Y"
• 110  SUBEXIT        ! CONTROL RETURNS TO CALLING PROGRAM
  120  PRINT "X IS GREATER THAN Y BY";X-Y
• 130  SUBEND
• 140  SUB Accounts(A)
  150  ON A+1 GOTO 160,180
  160  PRINT "RANDOM NUMBER IS LESS THAN .5"
• 170  SUBEXIT
  180  PRINT "RANDOM NUMBER IS GREATER THAN OR EQUAL TO .5"
• 190  SUBEND
```

```
X IS GREATER THAN Y BY 3
BACK FROM 1st SUBPROGRAM

RANDOM NUMBER IS GREATER THAN OR EQUAL TO .5
BACK FROM 2nd SUBPROGRAM
```

Subprogram Considerations

What Happens

When entering a subprogram the following occur –

- The DATA pointer is reset to the first DATA statement in subprogram.
- Any file assignments that are not passed are cleared.
- RAD, STANDARD, and OPTION BASE 0 are the modes defaulted to.
- Any ON KEY#, ON END#, ON INT or ON ERROR associated with a GOTO or GOSUB is no longer active; however ON KEY interrupts are logged for processing upon return to the calling program.

Upon return to the calling program, all of the above are restored to their previous state.

Using the COM Statement

Values can also be passed to a subprogram with a COM statement. The list of items in the subprogram COM may be a subset of the main program COM statement; that is, it must match up to some point in the main program COM. Here are some examples –

```
  10    OPTION BASE 1
• 20    COM A(4,4),B,INTEGER C,D(3,3),E$[28],F$(2,4)[56]
  30    MAT A=(5)
  40    B=C=7
  50    MAT D=(3)
  60    CALL Routine
  70    Y=FNCount
  80    PRINT Y
  90    END
  100   SUB Routine
  110   OPTION BASE 1         ! OPTION BASE MUST MATCH IF ARRAYS
                                ARE PASSED IN COM LIST
• 120   COM X(4,4),Y,INTEGER Z,Q(3,3)
  130   PRINT X(*);Y,Z,LIN(2),Q(*)
  140   SUBEND
  150   DEF FNCount
• 160   COM I(1:4,1:4)        ! THE LOWER BOUND CAN BE SPECIFIED
                                TO MAKE OPTION BASE MATCH
  170   RETURN I(2,2)*RND
  180   FNEND
```

```
 5   5   5   5

 5   5   5   5

 5   5   5   5

 5   5   5   5

 7                    7

 3                    3                    3

 3                    3                    3

 3                    3                    3


 3.39109504673
```

Arrays can be specified in a subprogram COM statement using an array identifier. This method is very useful for editing. If you change the dimensions of an array in a main program COM statement, you won't have to edit each subprogram COM to make the dimensionality match. Using an array identifier also avoids an error if an array declared with COM was redimensioned in the calling program segment. A variable can't be an item in a subprogram COM statement if it is also a formal parameter.

Example

```
200  SUB Sub(X,Y,Z$)
210  COM A(*),B,C
```

Variable Allocation Statements

Subprograms may also have any variable allocation statements: DIM, REAL, SHORT, and INTEGER. However, the variables declared may not be in the subprogram COM statement or the formal parameter list.

Here is an example –

```
250  SUB X(X,Y(*),Z$,A)
260  COM B(*),C$,D,SHORT E
270  DIM F(5,2),G$(2,2)[50]
280  SHORT H,I(3,7,2)
290  REM   COMPUTATIONS ....
```

Within subprogram variable allocation statements, array subscripts and maximum string lengths can be specified with a numeric expression which can contain constants and formal parameters because storage for them is temporarily allocated **before** execution of the subprogram begins.

Local Variables

All variables in a subprogram that are not part of the formal parameter list or the COM statement are known as "local" variables and cannot be accessed from any other program segment. Storage of local variables is temporary, and is returned to main user Read/Write Memory upon return to the calling program. This is known as dynamic memory allocation.

All variable names in a subprogram are independent of variables with the same name in other program segments. Thus, if you check the value of a variable using live keyboard while a program is running, you may get an unexpected result if the variable isn't defined in the program segment which is executing currently.

Files

File numbers of files opened in the calling program can be passed to a subprogram in the parameter list.

For example –

```
  10   CREATE "Data",1
• 20   ASSIGN #1 TO "Data"
• 30   CALL Routine(#1)       ! FILE ASSIGNMENT IS PASSED
  40   READ #1,1;A            ! PRINT# IS IN SUBPROGRAM
  50   PRINT A
  60   END
• 70   SUB Routine(#3)        ! #3 IS NOW ASSIGNED TO "Data"
  80   PRINT #3;RND
  90   SUBEND
```

```
.678219009345
```

Any operations, such as `PRINT#`, which involve file #3 in the subprogram will affect file #1, Data, in the calling program.

File numbers can also be implicitly assigned within the calling program from within a subprogram. For example –

```
  10   CREATE "Pay",1
• 20   CALL X(#4)
  30   READ #4,1;A        ! NO ASSIGN IN CALLING PROGRAM
  40   PRINT A
  50   END
• 60   SUB X(#2)
• 70   ASSIGN #2 TO "Pay"
  80   PRINT #2,1;RND
  90   SUBEND
```

```
.678219009345
```

9

When control returns to the calling program, #4 is still assigned to the file Pay.

A file can also be implicity buffered in this manner –

```
  100  CALL Data(#4)
  310  END
  320  SUB Data(#2)
  330  ASSIGN #2 TO "Pay"
• 340  BUFFER #2       ! Implicitly buffers #4 in calling program
  350  SUBEND
```

When control returns to the calling program, #4 is still assigned to Pay and it is still buffered.

If a file is actually opened in a subprogram and wasn't passed as a parameter, it is automatically closed upon return to the calling program.

See Chapter 11 for explanation of BUFFER and ASSIGN.

Editing

There are two ways to add a new subprogram to a main program and any subprograms. It must either replace an existing subprogram or it must come **after** all other subprograms.

In order to delete the first line of a subprogram (the SUB or DEF FN statement), the entire subprogram must be deleted.

The SUB statement can be edited as long as it remains a SUB statement or is changed to a DEF FN.

Busy Lines

When a subprogram is accessed from a calling program, a condition is created known as a busy line or a busy subprogram. Here is an example of a busy line –

```
10    A=FNX(B)          ! THIS LINE IS 'BUSY' UNTIL CONTROL RETURNS
                          FROM THE SUBROUTINE
90    END
100   DEF FNX(D)
190   RETURN D^2
200   FNEND
```

Line 10 is busy after the subprogram at line 100 is accessed and remains busy until RETURN is executed.

Here is an example of a busy subprogram –

```
50   CALL X(A,B,C)
90   END
100  SUB X(X,Y,Z)
110  CALL Y
190  SUBEXIT
200  SUB Y
300  SUBEXIT
```

The subprogram X at line 100 becomes busy when line 100 is executed. It becomes unbusy when the SUBEXIT at line 190 is executed.

Busy lines and subprograms can have an effect when editing a running program or executing LINK. Attempting to delete or alter a busy line causes an error message. Program execution has to be stopped in order to delete or alter the line.

Chapter 10
Output

Introduction

Output is a method of recording information in the computer onto an output device. Program results and listings are two examples of information which can be output. Output may take many different forms, including printout, visual display and punched paper tape.

The following topics and statements are covered in this chapter –

- Audible output – BEEP
- Displayed output – DISP
- The standard printer
- PRINT
- Output functions
- Formatted Output – PRINT USING and IMAGE
- Overlapped Processing

Terms

The following parameters used in this chapter can be numeric expressions –

select code
HP-IB device address
number of characters per line
character position
number of spaces
number of linefeeds

10

Audible Output

The BEEP statement is used to create a brief audible tone which can be used in a number of ways.

BEEP

BEEP can signal that a particular computation or program segment is complete. It can also be used to audibly indicate that the computer is ready for input, so that the operator does not have to remain at the keyboard.

Here's an example use for BEEP –

```
   10    FOR I=1 TO 7
•  20       BEEP          ! SIGNALS USER WHEN AN INPUT IS REQUIRED
   30       INPUT "DATA VALUE?",N(I)
   40    NEXT I
   50    PRINT N(*)
   60    END
```

In this case, a beep signals the operator when the program is ready for input.

Displayed Output

The DISP (display) statement allows text and variables to be output in the display line.

DISP [display list]

The display list can contain the following –

- variable names
- array identifiers
- numeric expressions
- string expressions
- TAB function*
- SPA function*

Multiple-line user-defined functions aren't allowed in the display list, alone or in an expression. The items in the display list must be separated by commas or semicolons. The list may end with a comma or semicolon.

* The output functions are discussed later in this chapter.

10

Examples

```
  10   X=3.5
• 20   DISP "X EQUALS";X,"X SQUARED EQUALS";X^2
  30   REM  NOTE THE USE OF COMMAS AND SEMICOLONS
  40   END
```

```
X EQUALS 3.5         X SQUARED EQUALS 12.25
```

```
• 10   DISP 10,20,30,40          ! 20 CHARACTER FIELDS
• 20   DISP 10;20;30;40          ! TIGHT SPACING
  30   END
```

```
 10                  20                  30                  40
 10  20  30  40
```

Notice the difference in spacing between the numbers. This is caused by use of a comma or a semicolon. When an item is followed by a comma, it is left justified in a field that is 20 characters wide. Two or more commas after an item cause one or more character fields to be skipped. For example –

```
• 10   DISP 125000,,250000       ! 2 COMMAS SEPARATE ITEMS
  20   END
```

```
 125000                                  250000
```

When an item is followed by a semicolon, no additional blanks are output after the item. Remember that every number has a leading blank or minus sign and a trailing blank for spacing (see Number Formats, Chapter 3). For example –

```
• 10   DISP 100;-20;77.3
  20   END
```

```
 100 -20  77.3
```

10

Normally, one display replaces a previous one. Successive displays in a program can be prolonged with the WAIT statement (which is discussed at the end of Chapter 4).

When the display list ends with a comma or semicolon, any future DISP statement output is appended to the current display line. For example –

```
  10    Date$="June 1"
• 20    DISP "TODAY IS ";        ! MORE DISPLAYS ARE APPENDED
  30    WAIT 1000
  40    DISP Date$
  50    END
```

The following are displayed in succession –

```
Today is
Today is June 1
```

If the information being displayed is longer than 80 characters, a carriage return/linefeed (CR-LF) is automatically output after every 80th character causing a new line to overwrite the previous. Only the last line of the displayed information is visible. You can see all of the displayed information by setting the print all mode. This causes every display to be printed on the print all printer (see Chapter 3).

Printed Output

Five statements are used to control printed output: PRINT, MAT PRINT, PRINT USING, IMAGE and PRINTER IS.

Defining the Standard Printer

The PRINTER IS statement defines the standard print device for the system. For the 9835A, the CRT, select code 16, is standard at power on, and SCRATCH A. For the 9835B the thermal strip printer, select code 16, is standard. If you do not have a printer in your 9835B, any operation directed to select code 0 or 16 causes an I/O error.

PRINTER IS select code [, HP-IB device address] [, WIDTH number of characters per line]

All output from PRINT, PRINT USING, LIST and CAT, and syntax error messages from GET or LINK are directed to the standard printer.

The specified device must be an acceptable printing device, like a printer or tape punch; it may be any device which can accept strings of ASCII characters.

The WIDTH parameter is a numeric expression and specifies the number of characters per line of the standard printer. Its range is 16 through 260; 80 is the power on and default value unless the internal printer is specified. 16 is the default width in this case.

Here are some examples –

```
10   REM  THESE ARE EXAMPLES OF PRINTER IS STATEMENT
20   PRINTER IS 16              ! PRINTER IS CRT
30   PRINTER IS 0               ! PRINTER IS STRIP PRINTER
40   PRINTER IS 6               ! PRINTER AT SELECT CODE 6
50   PRINTER IS 7,2             ! HP-IB PRINTER
60   PRINTER IS 6,WIDTH(160)    ! 160 CHARACTERS PER LINE
70   END
```

The PRINT Statement

The PRINT statement causes text and variables to be output on the standard printer.

PRINT [print list]

The print list can contain the following items –

- variable names
- array identifiers
- numeric expressions
- string expressions
- TAB function
- SPA function
- LIN function
- PAGE function

Multiple-line user-defined functions aren't allowed, alone or in an expression. All items must be separated by commas or semicolons.

10

Here are some examples of the PRINT statement –

```
  10   FOR I=1 TO 5
• 20      PRINT "I EQUALS";I
  30   NEXT I
  40   END
```

```
I EQUALS 1
I EQUALS 2
I EQUALS 3
I EQUALS 4
I EQUALS 5
```

```
• 10   PRINT "***!!!";"///^^^";"%%%@@@"    ! TIGHT SPACING
• 20   PRINT "***!!!","///^^^","%%%@@@"    ! 20 CHARACTER FIELDS
  30   END
```

```
***!!!///^^^%%%@@@
***!!!              ///^^^              %%%@@@
```

Notice in the previous example that commas and semicolons perform in the PRINT statement just like in the DISP statement. A comma after an item causes it to be left justified within a 20-character field. A semicolon after an item suppresses additional blanks. A comma or semicolon after the last item in the list allows a future print list to be appended by suppressing the CR-LF. A CR-LF is automatically output when the WIDTH is exceeded.

The current numeric output form (see Chapter 3) determines how a number is output with both DISP and PRINT. For example –

```
10   FIXED 2
20   PRINT 20;81.1596;32.9
30   END
```

```
 20.00  81.16  32.90
```

The variable width of the standard printer can be especially useful when outputting non-printable characters such as escape codes. Here is an example to try using the CRT of the 9835A as the output device –

```
  10   REM       TRY THIS EXAMPLE
• 20   PRINTER IS 16,WIDTH(160)     ! NORMAL WIDTH IS 80
  30   FOR I=1 TO 40
• 40      PRINT CHR$(129)&"*"&CHR$(128)&"*";
  50      !  LINE 40 PRINTS 4 CHARACTERS AT A TIME--160 TOTAL
  60   NEXT I
  70   END
```

In this example, CHR$(129) and CHR$(128) are non-printable characters used to turn inverse video mode on and off. Please refer to Appendix B for more explanation of this use of CHR$.

Output Functions

Four output functions are available to increase formatting capabilities. TAB and SPA can be used with both DISP and PRINT; LIN and PAGE can be used only with PRINT. They must be separated from the next item in the display or print list with either a comma or a semicolon. However, both the comma and semicolon function identically after an output function; they merely serve to separate it from the next item.

The TAB Function

The TAB function causes the next item in the list to be output beginning in the specified column.

TAB character position

The character position can be specified by any numeric expression, except one containing a multiple-line function, and it is rounded to an integer. If it is less than 1, it defaults to 1. For example –

```
100 DISP 147; TAB (10), "THIS STARTS IN THE 10th COLUMN"
```

If the specified column has already been filled, a CR-LF is output, and then the TAB is completed. For example, if line 100 above is changed to –

```
100 DISP 147, TAB (10), "THIS STARTS IN THE 10th COLUMN"
```

a CR-LF would be output after 147 (notice that the comma causes 147 to be output in a 20 column field) and only the text would remain in the display.

10

When the character position specified is greater than the number of columns in the standard printer, it is reduced by this formula –

(character position−1) MOD N + 1

N is the number of columns specified as standard printer width. For example, with printer width 80 –

```
10 PRINT TAB (10), 1; TAB (90),2;TAB (170),3

         1
         2
         3
```

The SPA Function

The SPA (space) function is used with DISP and PRINT to output the specified number of blank spaces up to the end of the current line.

SPA number of spaces

Here's an example –

```
DISP 1; SPA (10);2;SPA(10),3

 1          2          3
```

The number of spaces can be specified by any non-negative numeric expression, except one containing a multiple line function, and it is rounded to an integer. If it specifies more blanks than remain in the line, the next item begins the next line. For example –

```
PRINT "**"; TAB 70,"**"; SPA 20, "**"
```

is printed –

```
**                                                                   **
**
```

10

The LIN Function

The LIN function is used with PRINT and causes the specified number of linefeeds to be output.

LIN number of linefeeds

The number of linefeeds can be specified by any numeric expression, except one containing a multiple-line function, and it is rounded to an integer. Its range is −32 768 through 32 767.

Here's an example –

```
  10   A$="AUGUST 28"
• 20   PRINT "TODAY IS ";A$,LIN(3),"DATA COMPLETE"
  30   REM   3 LINEFEEDS ARE OUTPUT BETWEEN STRINGS
  40   END
```

```
TODAY IS AUGUST 28


DATA COMPLETE
```

When the number of linefeeds is positive, a carriage return precedes the linefeeds. When zero linefeeds are specified, only a carriage return is output. When the number of linefeeds is negative, no carriage return is output; the number of linefeeds output equals the absolute value of the expression. Some printers, such as the internal printer, don't suppress the carriage return. For example –

```
  10   PRINTER IS 16
• 20   PRINT "Today";LIN(-2);"Is";LIN(-2),"Saturday"
  30   END
```

```
Today

     Is

       Saturday
```

10

The PAGE Function

The PAGE function can be used with PRINT and causes a form feed character to be output, so further printing can begin on a new page or at the top of the next form on devices that can understand ASCII form feed (CHR$(12)). The internal printer does not "understand" form feed; a linefeed is output instead.

PAGE

Here's an example –

```
100 PRINT"DATA";LIN(2),A(*),PAGE,"RESULT"; LIN(2);B(*)
```

When the standard printer is the CRT, PAGE clears the entire print area.

Printing Arrays

The MAT PRINT statement is also used to print arrays on the standard printer.

MAT PRINT array variable [, or ; [array variable, or ; ...]]

The comma or semicolon following an item specifies open or close spacing between elements. For example –

```
  10   OPTION BASE 1
  20   DIM A(3,3)
  30   MAT A=(5)
  40   PRINT "20 CHARACTER FIELDS"
● 50   MAT PRINT A
  60   PRINT
  70   PRINT "CLOSE SPACING"
● 80   MAT PRINT A;         ! NOTICE THE SEMICOLON
  90   END


  20 CHARACTER FIELDS
   5                   5                   5

   5                   5                   5

   5                   5                   5


  CLOSE SPACING
   5  5  5

   5  5  5

   5  5  5
```

When an array is printed, every printed row is followed by a blank line. The last row is followed by two blank lines.

When an array has more than two dimensions, the last subscript varies fastest and defines the length of a row. For example –

```
  10   OPTION BASE 1
• 20   DIM A(2,3,4)
  30   FOR I=1 TO 2           ! Affects 1st (left) subscript
  40      FOR J=1 TO 3        ! Affects 2nd subscript
  50         FOR K=1 TO 4     ! Affects 3rd (right) subscript
  60            A(I,J,K)=X    ! Assigns values to array elements
  70            X=X+1
  80         NEXT K
  90      NEXT J
  100  NEXT I
  110  PRINT "THREE-DIMENSIONAL ARRAY - (2,3,4)"
• 120  MAT PRINT A;
  130  END
```

```
THREE-DIMENSIONAL ARRAY - (2,3,4)
 0  1  2  3

 4  5  6  7

 8  9  10  11

 12  13  14  15

 16  17  18  19

 20  21  22  23
```

In this example, array A(2,3,4) is interpreted as two matrices, each 3 by 4, for output or input purposes.

Arrays can also be printed by the `PRINT` statement using an array identifier, `(*)`. In the previous example, line 120 could be changed to –

```
120 PRINT A(*);
```

10

Formatted Output

Two statements, PRINT USING and IMAGE, provide the capability of generating printed output with complete control of the format.

PRINT USING string expression[; print using list]

PRINT USING line identifier[; print using list]

IMAGE format string

The print using list can contain the following items –

variable names
array identifiers
numeric expressions
string expressions

Remember, no multiple-line user-defined functions can be specified in the print using list. The items in the list are separated by commas or semicolons. However, the commas and semicolons have no effect on the printout, as in PRINT or DISP; they are used only to separate items. The output is totally controlled by the format string.

The string expression in the first syntax must be a valid format string at the time of execution. It can be any string expression. The line identifier in the second syntax must refer to an IMAGE statement; the IMAGE statement contains the format string corresponding to the particular PRINT USING statement.

Format String

The format string is a list of **field specifiers** separated by delimiters. It is used to specify numeric and string fields, blanks, and carriage control. Each numeric or string field specifier must correspond to an appropriate item in the print using list. Each field specifier is made up of various symbols and determines how a single item in the print using list is to be output.

Delimiters

Three delimiters are used to separate field specifiers –

, A comma is used only to separate two specifiers.

/ A slash can be used to separate two specifiers. It also causes output of a CR-LF.

10

@ The commercial at sign can be used to separate two specifiers. It also causes output of a formfeed character, starting a new page of output on devices that have this capability.

The / and @ symbols can also be used as field specifiers by themselves; that is, they may be separated from other specifiers by a comma. Only the / can be directly replicated (see the Replication section which is later in this chapter).

Blank Spaces

A blank space is specified with –

X NX specifies N blanks. Any X specifier can be imbedded within any other field specifier without delimiters.

String Specification

Text can be specified in two ways –

" " A literal specifier is text enclosed in quotes. This specifier may be imbedded without delimiters within any other field specifier.

For example –

```
10   IMAGE "**",4X"Results"4X,"**"    ! BLANKS AND LITERALS OUTPUT
20   PRINT USING 10
30   END

**    Results    **
```

A A is used to specify a single string character. NA specifies N characters. The length of the string specifier is determined by the number of A's that are specified between delimiters; this corresponds to one item in the print using list.

The above example could also have been written –

```
10   A$="Results"
20   IMAGE "**"4X7A4X"**"    ! LITERAL, BLANK AND STRING SPECIFIED
30   PRINT USING 20;A$
40   END

**    Results    **
```

10

If the string item in the print using list is longer than the number of characters specified, the string is truncated. For example –

```
10    PRINT USING "5A";"RESULTS"
20    END
```

```
RESUL
```

If the item is shorter, the item is left justified and the rest of the field is filled with blanks.

Numeric Specification

Numeric specifiers can be made up of various types of symbols: digit symbols, sign symbols, radix symbols, separator symbols and an exponent symbol.

Digit Symbols

D   Specifies a digit position. ND specifies N digit positions. Leading zeros are replaced with a blank space as a fill character.

For example –

```
10    PRINT USING "DDDDD,2X,DD";250,25
20    REM     LINE 10 SPECIFIES A 5-DIGIT FIELD,
30    REM     2 BLANKS AND A 2-DIGIT FIELD
40    END
```

```
  250  25
```

Z   Specifies a digit position. NZ specifies N digit positions. Leading zeros are replaced with 0 as a fill character.

For example –

```
10   IMAGE XAXXXX,AAA/           ! Two literal fields & blanks
                                   followed by CR/LF
20   IMAGE ZZZ,3X,ZZZ            ! Two 3-character fields
30   PRINT USING 10;"I","I*4"    ! Print heading
40   FOR I=1 TO 3
50      PRINT USING 20;I,I*4     ! Print 2 numbers
60   NEXT I
70   END

 I     I*4

001   004
002   008
003   012
```

---

* Specifies a digit position. N* specifies N digit positions. Leading zeros are replaced with * as a fill character.

For example –

```
10   IMAGE *****,2X,"Dollars and",XDDX,"cents"//
20   !    5 digits, 2 blanks, literal, digits & blanks,
30   !    literal and 2 CR/LF's
40   PRINT USING 10;250,52   ! Dollars & cents are separate
50   PRINT USING "32A";"(*'s good for check protection)"
60   END

**250  Dollars and 52 cents

(*'s good for check protection)
```

Only the symbol D is allowed to the right of any radix indicator symbol (discussed later). Any digit symbol can be used to specify the integer portion of any number but, with one exception, they can not be mixed. That is, for example, if D is used they must all be D. The exception is that the digit symbol specifying the one's place can be a Z regardless of the other symbols. For example –

```
10   IMAGE DDD.DD/***.DD      ! D's and *'s to left of radix
20   IMAGE DDZ.DD/**Z.DD      ! Like above but Z in 1's place
30   PRINT USING 10;.25,.75
40   PRINT USING 20;.25,.75
50   END

   .25
***.75
  0.25
**0.75
```

10

Radix Symbols

A radix indicator is used to separate the integer part of a number from the fractional part. In the United States for example, this is customarily the decimal point, as in 34.7. In Europe, this is frequently the comma as in 34,7. Only one symbol for a radix indicator at most can appear in a numeric specifier.

. Specifies a decimal point radix indicator in that position.

R Specifies a comma radix indicator in that position.

Here are some examples –

```
10    A$="United States"
20    E$="Europe"
30    IMAGE DDD.DD,2X,13A          ! Decimal point as radix
40    IMAGE DDDRDD,2X,6A           ! Comma as radix
50    PRINT USING 30;225.05,A$
60    PRINT USING 40;225.05,E$ ! Radix must be a period in
                                 the print using list
70    END

225.05  United States
225,05  Europe
```

If the number to be output contains more digits to the right of the radix indicator than are specified, the number if rounded. Here is an example –

```
10    IMAGE DD.DD               ! 2 places after decimal
20    PRINT USING 10;25.256     ! 3 places after decimal
30    END

25.26
```

Sign Symbols

Two sign symbols are used to control the output of the sign characters + and –. Only one sign symbol at most can appear in a numeric specifier.

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.

If the sign symbol appears before all digit symbols in a numeric specifier, it floats (see the section of Floating Symbols which is later in this chapter) to the left of the leftmost significant digit output.

When no sign symbol is specified and the number to be output is negative, the minus sign occupies a digit position.

Here's an example –

```
10    IMAGE SDDD,3X,MDDD        ! One with S, one with M
20    PRINT USING 10;250,250 ! Both numbers positive
30    PRINT USING 10;-5,-10   ! Both numbers negative
40    IMAGE /SDDDD.DD,2X,"Monthly profit"
50                            ! Sign floats
60    PRINT USING 40;25.15,-4000.25
70    END

+250     250
  -5     -10

  +25.15  Monthly profit
-4000.25  Monthly profit
```

Digit Separator Symbols

Digit separators are used to break large numbers into groups of digits (generally three digits per group) for greater readability. In the United States, the comma is customarily used; in Europe, the period is commonly used. The X symbol can also be used to cause digits to be separated with a blank space.

C   Specifies a comma as a separator in the specified position.

P   Specifies a period as a separator in the specified position.

The digit separator is output in an item only if a digit in that item has already been output; the separator must appear between two digits. When leading zeroes are generated by the Z symbol, they are considered digits and will contain separators if specified.

Here is an example showing digit separators –

```
10    IMAGE DDDCDDD,2XDDDCDDD/
20    PRINT USING 10;2525,250   ! No comma output in 2nd number
30    IMAGE ZZZCZZZ
40    PRINT USING 30;25          ! Comma output between 0's
50    IMAGE /DDDPDDDPDDD,2X,"Houses in Hamburg"
60    IMAGE  DDDCDDDCDDD,2X,"Houses in Loveland"
70    PRINT USING 50;21345
80    PRINT USING 60;19874
90    PRINT USING "/Z.5DX5DX2D";SIN(1) ! Blanks separate digits
100   END

  2,525       250

000,025

     21.345  Houses in Hamburg
     19,874  Houses in Loveland

0.84147 09848 05
```

10

Exponent Symbol

E Specifies that the number related to the numeric field that it is contained in is to be output in scientific notation. E causes the output of an E, sign of the exponent and two digit exponent. At least one digit symbol must precede the E symbol in a numeric specifier.

Here are some examples –

```
10    PRINT USING "D.DDDE";125.25
20    PRINT USING "DDD.DDE";2.505
30    END

1.253E+02
250.50E-02
```

Floating Symbols

Floating symbols – S, M, X, or text in quotes – that precede all digit symbols in a numeric specifier "float" past blanks to the leftmost digit of the number, or to the radix indicator. This is useful for output of monetary values so that the dollar sign will be output next to the first digit.

Here are some examples –

```
10    IMAGE "("DDD.DD")"          ! PARENTHESES FLOAT
20    PRINT USING 10;2.37,10.87
30    IMAGE "$"DCDDDCDDD.DD     ! DOLLAR SIGN FLOATS
40    PRINT USING 30;1736.73,42.35
50    END

  (2.37) (10.87)
    $1,736.73         $42.35
```

Sign symbols and text that are imbedded between digit symbols do not float.

Here are some examples of floating and non-floating symbols –

floating	non-floating
"$"DDD.DD	"$",DDD.DD
MDDD.DD	D"$"DD.DD
	DMDD.DD

X, S, M, or text imbedded in a numeric field stops the floating field.

Replication

Many of the symbols used to make up field specifiers can be replicated (repeated) to specify multiple symbols by placing an integer in the range 1 through 32 767 in front of the symbol. The following IMAGEs all specify the same format string –

```
10   PRINT USING "DDD.DD";123.45
20   PRINT USING "D2D.2D";123.45
30   PRINT USING "3D.DD";123.45
40   PRINT USING "3D.2D";123.45
50   END

123.45
123.45
123.45
123.45
```

Here's an example of replication –

```
  10   OPTION BASE 1
  20   DIM A(3,3)
  30   MAT A=(2)
• 40   PRINT USING "3(D,X)/";A(*)
  50   END

2 2 2
2 2 2
2 2 2
```

Placing an integer before a symbol works exactly like having multiple adjacent symbols.

The following symbols **can't** be replicated –

" " literal fields	E	
S	;	
M	@	
.	K	(see next section for explanation)
R	+	
C	-	
P	#	

10

In addition to symbol replication, an entire specifier or group of specifiers can be replicated by enclosing it in parentheses and placing an integer in the range 1 through 32 767 before the parentheses. For example –

```
10    REM    HERE ARE EXAMPLES OF REPLICATION
20    IMAGE DD.D,6(DDD.DD)
30    IMAGE 4(2X,4*Z.D,(2X,D))
```

So, specifying `3(DD)` is the same as specifying `DD,DD,DD`.

In this manner, both `K` and `@` can be repeated –

```
10   IMAGE 4(K),2(@)
```

Up to four levels of nested parentheses can be used for replication.

Compacted Specifier

A single symbol, `K`, is used to define an entire field for either numeric or string output. If the corresponding print using item is a string, the entire string is output. If it is a numeric, it is output in STANDARD form. K outputs no leading or trailing blanks. For example –

```
10    PRINT USING "K/";"AGES:"
20    IMAGE KXX,2D          ! Name can be any length
30    FOR I=1 TO 3
40       READ A$,A
50       PRINT USING 20;A$,A
60    NEXT I
70    DATA Mary,10,Hildegard,20,Amy,15
80    END

AGES:

Mary  10
Hildegard  20
Amy  15
```

Carriage Control

The CR-LF normally output when the print using list is exhausted can be altered by using a carriage control symbol as the first item in a format string; a comma must separate it from the next item.

`#`	Suppresses both the carriage return and linefeed.
`+`	Suppresses the linefeed.
`-`	Suppresses the carriage return.

For example –

```
10   IMAGE #,4(A,2X)
20   IMAGE K
30   PRINT USING 10;"A","B","C","D"
40   PRINT USING 20;"****"
50   END

A   B   C   D   ****
```

Notice that `PRINT USING "+"` is equivalent to `PRINT LIN(0);` and `PRINT USING "-"` is equivalent to `PRINT LIN(-1);`.

Reusing the Format String

A format string is reused from the beginning if it is exhausted before the print using list. This is also a way to replicate fields. For example –

```
PRINT USING "DDD.DD"; 25.71, 99.9
Δ25.71Δ99.90
```

Field Overflow

If a numeric item requires more digits than the field specifier provides, an overflow condition occurs. When this happens, all preceding, correct items are output, followed by a CR-LF. The item which overflowed is output in STANDARD format followed by the field specifier which caused the overflow. Another CR-LF is output, then the rest of the print using list is output. For example –

```
10   PRINT USING "3(DD.DD)/";25.5,250.25,20.25
20   PRINT USING "DD.D,DDD,DD";25.5,-250.5
30   END

  25.50
• 250.25 DD.DD
  20.25

  25.5
• -250.5 DDD
```

An important thing to remember is that a minus sign not explicitly specified with `S` or `M` requires a digit position.

No error message occurs when a field overflow occurs, but the computer beeps.

Summary

Here is a summary table of image symbols and their uses –

Image Symbol	Replication Allowed?	Purpose	Comments
X	Yes	blank	Can go anywhere
" "	No	Text	Can go anywhere
D	Yes	Digit	Fill = blanks
Z	Yes	Digit	Fill = zeroes
*	Yes	Digit	Fill = asterisks
S	No	Sign	"+" or "–"
M	No	Sign	"Δ" or "–"
E	No	Exponent	Format = ESDD
.	No	Radix	Output "."
C	No	Comma	Conditional digit separator
R	No	Radix	Output ","
P	No	Decimal point	Conditional digit separator
A	Yes	Characters	Strings
( )	Yes	Replicate	For specifiers, not symbols
#	No	Carriage control	Suppress CR-LF
+	No	Carriage control	Suppress LF
-	No	Carriage control	Suppress CR
K	No	Compact	Strings or numerics
;	No	Delimiter	
/	Yes	Delimiter	Output CR-LF
@	No	Delimiter	Output FF

Considerations

One factor that must be taken into account when creating formatted output with PRINT USING is the printer width. When dealing with numeric output, format strings should be designed so that a line of characters doesn't exceed the number of characters per line of the printer. PRINT USING does not provide carriage return-linefeeds to keep lines within the width of the printer.

Advanced Printing Techniques

Advanced printing techniques on the CRT (9835A) are covered in Appendix B.

Overlapped Processing

The 9835A/B has a capability which can enable a program to run faster and more effiiciently. This capability is known as overlapped processing or overlapped I/O. In overlap mode, I/O initiated by a program statement proceeds in parallel with the execution of subsequent program lines, while in serial mode the I/O is completed before the next line is executed. Overlap mode is set by the OVERLAP statement –

```
OVERLAP
```

By clever programming techniques and matching computation with I/O, the speed with which a program runs can increase by a factor of up to (number of I/O devices used +1). A program that has a large difference between the amounts of I/O and computation won't run more efficiently in OVERLAP mode.

If you are using ON ERROR (see Chapter 12) to trap errors, I/O errors (numbers 54-103) aren't trapped if overlap mode is in effect.

The computer is returned to the serial processing mode which is the default mode at power on, SCRATCH and SCRATCH A by the SERIAL statement –

```
SERIAL
```

Using serial mode is recommended during program debugging to avoid confusing results.

Chapter 11
Mass Storage Operations

Introduction

Data and programs can be stored on various mass storage media for later use. Many mass storage devices can be used with the 9835A/B. All devices are operated with the same statements and commands.

The basic mass storage operations are covered in this chapter. Specifics of the tape cartridge are discussed in the 9835A/B Owner's Manual. The Mass Storage Techniques Manual covers techniques for using mass storage devices in greater detail.

The following topics and statements are covered in this chapter –

- Standard Mass Storage Device
- Files
- Records
- EOF's and EOR's
- The Directory
- INITIALIZE
- Cataloging files
- Storing and retrieving programs
- Storing and retrieving data
- CHECK READ
- Protecting a file
- Purging a file
- Copying a file
- Renaming a file
- Storing keys, binaries and memory
- Rewinding the tape

11

Terms

The following terms are used in mass storage operations –

file number – the number assigned to a mass storage data file by an `ASSIGN` statement. Its range is one through ten.

file name – a one to six character string expression with the exception of a colon, quote mark, ASCII NULL, or CHR$ (255). Blanks are ignored. Here are some examples of file names –

"Data1"
"TEMP"
"ACCTS"
"Names"
A$ when A$="Diagns"

select code – an expression (rounded to an integer) in the range zero through sixteen. The following select codes are reserved –

- 0 Internal Thermal Printer
- 15 Tape drive
- 16 CRT (9835A); Internal printer (9835B)

mass storage unit specifier – any string expression of the form –
: device type [select code [, controller address | 9885 unit code [, unit code]]]

The letters specifying the various mass storage device types are –

Letter	Device
T	Tape cartridge
F	9885 Flexible Disk
Y	7905A Removable Platter
Z	7905A Fixed Platter
C	7906A Removable Platter
D	7906A Fixed Platter
P	7920A Disc Pack

The select code can be an integer in the range 1 through 15 with 15 reserved for the tape drive. 15 is default for T devices, 8 for F and 12 for all others.

The controller address can be an integer from zero through seven. Zero is the default address.

The 9885 unit code can be an integer from zero through three. Zero is the default code.

The unit code can be an integer from zero through seven. Zero is the default code. It is ignored for the 9885 and tape cartridge.

Mass storage unit specifier is abbreviated msus.

Here are some examples of mass storage unit specifiers –

msus	Explanation
":T15"	Tape cartridge drive
":F8"	9885 flexible disk at select code 8
":Y4,0,3"	7905A removable platter, select code 4, controller address 0, unit code 3

Remember that the mass storage unit specifier can be any string expression. The following program segment illustrates this.

```
10   Type$="F"
20   Select_code=7
30   MASS STORAGE IS ":"&Type$&VAL$(Select_code)
40   STOP
```

file specifier – a string expression of the form – file name [mass storage unit specifier]. Here are some examples –

"Data:F8"
"Backup:Y4,2"
"TEMP"&":T15"
"ACCTS"&Msus$ when Msus$=":F10"

protect code – any valid string expression except one with a length of zero. Only the first six characters are recognized as the protect code, however.

The following parameters used in this chapter can be numeric expressions –

- select code
- HP-IB device address
- controller address
- 9885 unit code
- unit code
- interleave factor
- heading suppression
- number of defined records
- record length
- file number
- defined-record number

11

The Standard Mass Storage Device

At power on and SCRATCH A, the tape cartridge drive, T15, is the standard mass storage device for the system. This is the device to which all mass storage operations are directed if no device is specified. The default device concept is very powerful in creating device independent programs.

The standard default device is changed by executing the MASS STORAGE IS statement –

MASS STORAGE IS mass storage unit specifier

For example –

```
10   REM   EXAMPLES OF MASS STORAGE IS
20   MASS STORAGE IS ":T15"     ! Tape cartridge drive
30   MASS STORAGE IS ":F8"      ! Flexible disk - select code 8
40   Msus$=":Y4,0,3"
50   MASS STORAGE IS Msus$      ! 7905A removable platter -
                                  select code 4, controller
60                              ! address 0, unit code 3
70   STOP
```

Structure

All mass storage operations deal with files and records, the basic components of a storage medium.

Files

Files are the basic unit into which programs and data are stored. Storage of all files is "file-by-name" oriented; that is, all files must be assigned unique names. The form these names must take is covered in the "Terms" section at the beginning of this chapter.

There are 6 types of files –

- Program files
- Data files
- KEY files
- STOREALL files
- Binary program files
- Binary data files (Mass Storage ROM)

Records

Every file is composed of a varying number of records. A record is the smallest addressable unit on a mass storage medium.

There are three types of records –

1. Physical records are 256-byte, fixed units which are established when a medium is initialized. Every file starts at the beginning of a physical record; this is an important fact for optimum device use. Otherwise, you need not be concerned with physical records.
2. Defined records are established using the CREATE statement and can be specified as having any number of bytes in the range 4 through 32767 (rounded up to an even number). A defined record is the smallest unit of storage which is directly addressable.
3. A logical record, a user-level rather than machine concept, is a collection of data items which are conceptually grouped together.

When a file is established with a STORE or SAVE statement (discussed later), the computer uses as many records of 256 bytes as it needs to store the program. Logical and defined records are not used with STORE.

Using the CREATE statement for data files, you can specify how many defined records you wish the file to contain. You don't need to be concerned with the correspondence between physical and defined records, except to remember that the first defined record of a file starts at the beginning of a physical record.

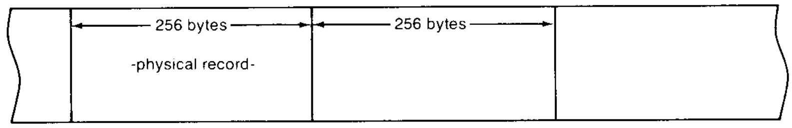
EOF's and EOR's

Files and records are bounded on the storage medium by end-of-file (EOF) and end-of-record (EOR) marks which signify their ends. This section illustrates and describes the organization of files and records on a storage medium.

11

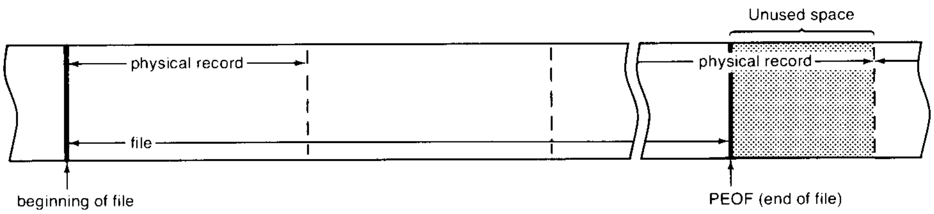
Physical Records

A storage medium is divided into 256-byte fixed physical records when it is initialized.



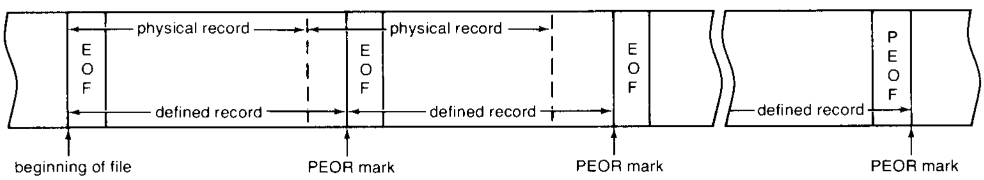
End-of-File and End-of-Record Marks

When a file is created, its end is designated by a physical-end-of-file (PEOF) mark. Any space between the PEOF and the beginning of the next physical record is unused space.

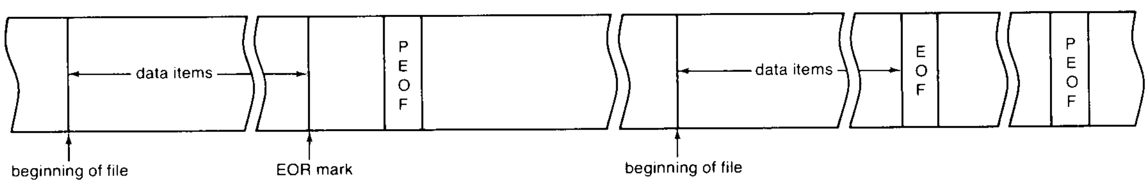


When a file is created using the CREATE statement (discussed later in this chapter), an end-of-file (EOF) mark is placed at the beginning of each defined record. Each EOF mark takes two bytes of storage space.

At the same time, a physical-end-of-record (PEOR) mark is placed at the end of each defined record. Numeric data items can't cross a PEOR mark.



As data is written to a file, the EOF marks are over-written. An EOF mark can be printed at the end of the data by printing END (see the PRINT# statement) after the data. If an EOF mark is not placed after the data, an end-of-record (EOR) mark automatically is.



11

The Directory

The directory is the storage medium's record of all of its file information; it includes each file's name, type, length, location and loading information. The directory information is automatically revised when a file is created or purged. A spare directory is maintained on the medium in the event that the first becomes unreadable. You are warned with a message every time the spare directory is accessed if the main directory becomes unreadable. It is accessed automatically by the system when necessary. Here is the message –

```
SPARE DIRECTORY ACCESS
```

There is no provision made for recovering information stored on a medium if both directories are destroyed. If the main directory becomes inaccessible it is wise to transfer all valuable data on the medium to another one before the spare directory is destroyed. Rewriting the main directory from the spare directory by adding, deleting or changing the name of a file may help the problem, but not necessarily solve it.

Tape Cartridge Directory

When a tape cartridge is being used to store and retrieve information, its directory is written into memory the first time it is accessed. This is done to save wear on the tape and improve performance by reading the directory from memory rather than from the tape. The directory on the tape is accessed only when it needs to be rewritten. The directory is erased from memory under any of the following conditions –

- Reset
- `SCRATCH A`
- Removing the tape from the drive

11

Basic Operations

Initializing a Mass Storage Medium

The INITIALIZE statement enables an unused mass storage medium to be used with the 9835A/B by establishing physical records and main and spare directories.

A used medium can also be re-initialized; in the process, it is cleared of all information it previously contained.

INITIALIZE mass storage unit specifier [, interleave factor]

The interleave factor is a numeric expression which defines the number of revolutions per track to be made for a complete data transfer. It is ignored for all devices except the 9885. See the Mass Storage Techniques Manual for its use.

Main and spare directories and all physical records are established and tested when a mass storage medium is initialized.

The INITIALIZE operation can take place at the same time as execution of a program if the program doesn't utilize the mass storage device that is involved in the initialization process. If the program attempts to use the drive on which an initialization is in progress, program execution is suspended until the operation is complete.

Here are some examples –

```
10   INITIALIZE ":T15"     ! INITIALIZE TAPE
20   A$=":F8"
30   INITIALIZE A$         ! INITIALIZE FLEXIBLE DISK
40   INITIALIZE ":F7",5    ! 5 IS INTERLEAVE FACTOR
50   END
```

Cataloging Files

The CAT (catalog) statement outputs a listing of directory information for a storage medium: file names, types, and physical specifications.

CAT [selective catalog specifier/msus [, heading suppression]]

CAT# select code [, HP-IB device address] [; selective catalog specifier/msus [, heading suppression]]

The selective catalog specifier is a string expression one through six characters in length. Only those files whose names begin with that combination of characters are cataloged.

If the value of the numeric expression is one, the heading is suppressed.

The second syntax directs the catalog output to the specified device.

Here are some examples –

```
10   CAT ":T15"            ! Catalog tape cartridge
20   CAT "Ab:F8"           ! Catalog all files starting with
                             'Ab' on disk at select code 8
30   CAT #6;"Dat:F8",1     ! Catalog all files starting with
                             'Dat' on disk  at select code 8
40                         ! Output goes to select code 6;
                             Heading is suppressed
50   MASS STORAGE IS ":F7,2"
60   CAT "S"               ! Catalog all files starting with
                             'S' on 9885 disk at select code 7
70   END
```

11

The information for each file is printed on one line. Here is a sample catalog output.

```
     1        2    3        4          5           6
     NAME     PRO  TYPE   REC/FILE  BYTES/REC    ADDRESS
   7 T15         8 2

     Uri           ALL       40        256           5
     SETUP         PROG       2        256          45
     SETUP2        PROG       2        256          47
     SETUP3        PROG       2        256          49
     SEARCH        DATA       2        256          51
     STRING        DATA       2        256          53
     AIDS          KEYS       1        256          55
```

1. NAME — The name given to the file when the information is stored on the medium.

2. PRO — An asterisk in this column designates a protected file.

3. TYPE — The various file types are specified by the following:
   PROG for a program file
   DATA for a data file
   KEYS for a KEY file
   ALL for a STOREALL file
   BPRG for a binary program file
   BDAT for a binary data file (Mass Storage ROM)
   If a medium is being cataloged that was not initialized on the 9835A/B, but on another HP computer such as the System 45, the 9835A/B attempts to determine what types the files are and put a question mark after the type in the catalog output. The type may or may not be correct; the 9835A/B may not be able to interpret the file.

4. REC/FILE — The number of defined records in the file.

5. BYTES/REC — The number of bytes per defined record.

6. ADDRESS — The address of the physical record number with which the file begins. With the tape cartridge, it is the number of the first physical record. See the Mass Storage Techniques Manual for information about other devices.

7. msus — The mass storage device on which the catalog was performed.

8. Available tracks — The number of tracks available for use. This is most important with the 9885; see the Mass Storage Techniques Manual.

Storing and Retrieving Programs

Programs can be stored onto a mass storage medium in two different ways, into two types of files.

The first type of file for storing programs is known as a **data file**. When a program is stored into a data file, it is stored as a series of strings, with one string per program line. This method is not the fastest method of storing and retrieving programs, but it has other advantages. A program stored into a data file can be accessed as string data by other programs. Programs are stored into data files with the SAVE statement and retrieved with the GET statement.

The second type of file is known as a **program file**. When a program is stored into a program file, it is stored in a compiled, internal code interpretation. Storing the program also stores all binary routines currently in memory along with the program. This is the fastest method for storing and retrieving programs. Programs are stored into program files with the STORE statement and retrieved with the LOAD statement.

Data Files

The SAVE Statement

The SAVE statement stores the program and any subprograms in computer memory into a data file on the storage medium.

SAVE file specifier [, beginning line identifier [, ending line identifier]]

Execution of the SAVE statement creates a data file by "listing" the program and saving the list on the medium as string data, one program line per string. In this way, the file can be read, modified, or rewritten as string data by other programs.

When only the file specifier is given, the entire program is saved. If the beginning line identifier is specified, the program from that number to the end is saved. If both line identifiers are specified, the program section, from the first line identifier to the second, inclusive, is saved. If the first line identifier is a label which is in a subprogram and execution is not currently in that subprogram, ERROR 3 occurs.

11

Examples

```
10    SAVE "LIFE"             ! SAVES PROGRAM INTO 'LIFE' ON
                                STANDARD MASS STORAGE DEVICE
20    SAVE "PAYROL:F8",40     ! SAVES PROGRAM STARTING WITH LINE 40
                                INTO 'PAYROL' ON FLEXIBLE DISK
30    A$="Checks"
40    SAVE A$,100,300         ! SAVES LINES 100-300 OF PROGRAM
                                INTO 'Checks'
50    STOP
```

The GET Statement

The partner of the SAVE statement, the GET statement retrieves and puts into memory a program saved previously with the SAVE statement, or any string data file consisting of valid BASIC statements preceded by line numbers, stored one line per string.

GET file specifier [, line identifier [, execution line identifier]]

Execution of the GET statement causes the computer to read the specified data file and expect to find a succession of strings that are valid program lines.

If no line identifiers are specified, the entire stored program is loaded into computer memory, destroying any programs or data (except data stored with COM) in memory.

If one line identifier is specified, the program is renumbered as it is loaded so that it begins with the number of the specified line of the program currently in memory. Any lower-numbered lines from a previous program are retained. The numbering remains the same on the storage medium.

If the GET was executed in a program, program execution is restarted with –

- The program line immediately following the GET statement in the original program or with
- The first line of the loaded program if there were no lines after the GET statement or if these lines were destroyed by the GET statement.

If two line identifiers are specified, program execution is restarted with the second line identifier.

When a program retrieved with GET has an invalid line in it, the invalid line and an error message is listed on the standard system printer. An example of how this can occur is when a program is SAVEd with the Mass Storage ROM installed in the machine and later retrieved with GET when the ROM is not installed. Any lines which have mass storage unit specifiers other than :T15 are listed with an error message.

11

Examples

```
10    GET "SMALL"       ! PROGRAM IN 'SMALL' IS RETRIEVED
20    GET "LIFE",75     ! PROGRAM IN 'LIFE' IS RETRIEVED AND
                          RENUMBERED TO BEGIN WITH 75
30    A$="PAYROL:F8"
40    GET A$,100,10     ! PROGRAM IN 'PAYROL' IS RETRIEVED AND
                          RENUMBERED TO BEGIN WITH 100
50                      ! EXECUTION BEGINS WITH LINE 10 IN MEMORY
60    STOP
```

The LINK Statement

The LINK statement is identical to the GET statement discussed previously, except that the current values of all variables are retained.

LINK file specifier [, line identifier [, execution line identifier]]

If no line identifiers are specified, the program is loaded, destroying the current program in memory.

The first line identifier specifies that the loaded program is to be renumbered and is to begin with the line number of the specified line.

If two line identifiers are specified, execution begins with the second line specified.

In effect, GET performs a RUN operation on the loaded program, whereas LINK performs a CONT operation, involving no pre-run initialization of variables.

Examples

```
10    LINK "LIFE"         ! PROGRAM IN 'LIFE' IS RETRIEVED
20    LINK "Checks",50    ! PROGRAM IN 'Checks' IS RETRIEVED AND
                            RENUMBERED TO BEGIN WITH 50
30    A$="PAYROL:F8"
40    LINK A$,100,10      ! PROGRAM IN 'PAYROL' IS RETRIEVED AND
                            RENUMBERED TO BEGIN WITH 100
50                        ! EXECUTION BEGINS WITH LINE 10 IN MEMORY
60    STOP
```

The RE-SAVE Statement

A program stored in a data file can be loaded into memory and edited. It can then be re-saved into the same file using the RE-SAVE statement –

RE-SAVE file specifier [, protect code] [, beginning line identifier [, ending line identifier]]

RE-SAVE is equivalent to PURGE followed by SAVE.

The protect code is used only if the file has been protected. When no line identifiers are specified, the entire program is saved. When one line identifier is specified, the program is saved from that line to the end. When two line identifiers are specified, that block of lines is saved.

Examples

```
10    RE-SAVE "LIFE"          ! Purge program in 'LIFE' & save
                                program in memory into it
20    RE-SAVE "Checks",50     ! Purge program in 'Checks' & save
                                current program; start at line 50
30    A$="PAYROL"
40    RE-SAVE A$,100,250      ! Purge program in 'PAYROL' & save
                                lines 100-250 of current program
50    RE-SAVE "SMALL","SM"    ! Resave 'SMALL' using protect
                                 code 'SM'
60    STOP
```

Program Files

The STORE Statement

The STORE statement creates a program file and stores the program and any binary routines in memory into it.

STORE file specifier

Examples

```
10    STORE "LIFE"        ! STORES PROGRAM INTO 'LIFE' ON
                            STANDARD MASS STORAGE DEVICE
20    STORE "PAYROL:F8"   ! STORES PROGRAM INTO 'PAYROL' ON
                            FEXIBLE DISK AT SELECT CODE 8
30    A$="Checks"
40    STORE A$            ! STORES PROGRAM INTO 'Checks'
50    STOP
```

If you attempt to STORE a program which has been SECUREd, the information written to the tape is meaningless.

The LOAD Statement

Programs saved with STORE are retrieved with the LOAD statement.

LOAD file specifier [, execution line identifier]

11

Execution of the LOAD statement destroys any program and data in memory and loads the program and any binary routines. However, any data stored in common is preserved if the loaded program has a COM statement. If the LOAD statement comes from the keyboard and no line identifier is specified, control returns to the keyboard after loading. If it comes from execution of a program line in memory, execution begins at the first line of the loaded program.

When the line identifier is specified, execution of the loaded program begins at that line.

Examples

```
10   LOAD "LIFE"          ! LOADS PROGRAM IN 'LIFE' INTO MEMORY
20   LOAD "Checks",50     ! LOADS PROGRAM IN 'CHECKS' INTO MEMORY;
                            EXECUTION BEGINS WITH LINE 50
30   A$="PAYROL:F8"
40   LOAD A$              ! LOADS PROGRAM IN 'PAYROL' ON FLEXIBLE
                            DISK AT SELECT CODE 8 INTO MEMORY
50   STOP
```

The RE-STORE Statement

A program file can be loaded into memory and edited, then re-stored into the same file using the RE-STORE statement –

RE-STORE file specifier [, protect code]

RE-STORE is equivalent to PURGE followed by STORE.

The protect code is used only if the file has been protected.

Examples

```
10   RE-STORE "LIFE"      ! PURGES PROGRAM IN 'LIFE' AND STORES
                            PROGRAM IN MEMORY INTO IT
20   A$="PAYROL:F8"
30   RE-STORE A$,"XX"     ! STORES PROGRAM INTO 'PAYROL' USING THE
                            PROTECT CODE 'XX'
40   STOP
```

Data

11

Data in the form of numbers and strings can be stored into a data file. This is the same type of file as the one created by the SAVE statement, but it is created differently. A group of conceptually related items is known as a logical record. It is advisable, for the sake of ease of handling, to save logical records into separate defined records, rather than putting all data in one combined record.

There are five basic data file operations: creating a file, opening a file, recording data, retrieving data, and closing a file.

Creating a Data File

The CREATE statement is used to create a data file.

CREATE file specifier, number of defined records [, record length]

The record length specifies the length of a defined record in bytes and is rounded up to an even integer. If it is not specified, a defined record length of 256 bytes is assumed.

The number of records specified can be 1 through 32 767. The length that can be specified is 4 through 32 767 bytes. However, the size of a file created is limited by the amount of available space on the medium. A medium overflow error (ERROR 64) occurs if more records are specified than the medium can hold.

CREATE also puts an EOF mark in the first word of every defined record.

Here are some examples –

```
10   REM   EXAMPLES OF CREATE
20   CREATE "DATA:F8",10   ! CREATES A 10-RECORD FILE NAMED 'DATA'
30   CREATE "Names",10,50  ! CREATES TEN 50-BYTE RECORDS
40   STOP
```

When creating data files, you must be sure that the length and number of your defined records suit the storage requirements of the logical records you plan to store. To determine storage requirements, see the section on Data Storage which is later in this chapter. Attempts to store data into an insufficient amount of storage space results in an error.

Opening a File

Data files must be opened before they can be accessed. This is done with the ASSIGN statement. The two syntax shown below are equivalent.

ASSIGN file specifier TO# file number [, return variable [, protect code]]

ASSIGN# file number TO file specifier [, return variable [, protect code]]

The ASSIGN statement sets up or references an existing internal **files table** and allows you to utilize data files (with PRINT# and READ# statements). The files table has room for ten entries. All entries are cleared when a program is run, and when SCRATCH, SCRATCH V, SCRATCH C, SCRATCH A or reset is executed. The file number is a numeric expression; its range is 1 through 10. The ASSIGN statement also assigns a file pointer used for data access to the file number, and positions the pointer at the beginning of the file.

The optional return variable can be a simple numeric variable or array element and is set after execution to indicate various results. Its value is used to check for errors. If no return variable is specified, an error occurs if the file isn't found, is protected or is of the wrong type.

Return Variable	Meaning
0	File available
1	No such file found
2	File is protected, or wrong file type

The protect code is a string expression, and is necessary only if the file was protected earlier. For all disks it must be the same protect code as the one to protect the file. If the file isn't protected, including the protect code causes an error. The null string as protect code corresponds to an unprotected file.

Here are some examples –

```
10   ASSIGN #1 TO "Data"
20   ASSIGN "SCORES" TO #4,Return
30   ASSIGN "SCORES" TO #5
40   STOP
```

Line 20 illustrates a return variable. Lines 20 and 30 show that more than one number can be assigned to a file.

Storing and Retrieving Data

There are two methods of storing and retrieving data in a file: serial access and random access.

11

Serial File Access

Serial file access is used to store or retrieve data items one after the other, without regard to defined records. Logical records can be longer or shorter than defined record length. For each data file opened, a file pointer keeps track of the data item currently being accessed. As you store or retrieve data, the pointer moves serially forward through the file.

Serial Printing

The serial PRINT# statement records values onto the specified file from the specified variables or strings in computer memory.

PRINT# file number; data list [, END]
PRINT# file number; END

The data list is a collection of items separated by commas. The items can be variables, array identifiers, numbers, or strings of characters. The last or only item can be END, which causes an EOF mark to be printed. Otherwise, an EOR mark is placed after the data list is printed.

Printing begins at the position of the pointer after the data item most recently stored or retrieved, or at the beginning of the file if nothing has been stored or retrieved, or if the pointer has been repositioned to the beginning of the file (see Repositioning the Pointer).

When storing a long string, it might be too long to be contained in one defined record. In that case, the string is automatically broken up and stored into as many defined records as it needs. This adds four bytes to the amount needed to store the string each time the string crosses over into another defined record. The parts of the string are identified as first, intermediate, or last.

The length of data in the list must equal or be less than the storage space that remains in the file after the pointer; otherwise, an EOF error occurs, signaling that you have filled your file. Data can also be stored using the PRINT# statement in a file created with the SAVE statement if the file has been assigned a number. SAVE, in effect, performs a serial print onto a file.

Here are some examples –

```
  10   CREATE "TEMP",4      ! A 4-RECORD FILE
  20   ASSIGN #3 TO "TEMP"
  30   A=B=C=D=RND
  35   DIM E(2,2)
  40   MAT E=(5)
• 50   PRINT #3;A,B,C
• 60   PRINT #3;D,E(*)
  70   END
```

These two statements record values for A, B, C, D, and E (*) onto file #3. This data constitutes a written record. The EOR which was placed after the data when line 50 was executed is overwritten when line 60 is executed. Another EOR is printed after the data in line 60. Remember, an EOR signifies that there is no more data between the file pointer and the end of the defined record.

The serial PRINT# statement can also be used to generate program lines into a file. Such a file can be retrieved with GET. Here's an example –

```
 10 CREATE "PRGRM",3,50
 20 ASSIGN #1 TO "PRGRM"
•30 PRINT #1;"10   X=5","20   Y=7","30   PRINT X,Y,X*Y","40   END"
•40 GET "PRGRM"
 50 END
```

When this program is run, the output is –

```
 5                   7                   35
```

Executing LIST produces –

```
 10   X=5
 20   Y=7
 30   PRINT X,Y,X*Y
 40   END
```

Serial Reading

The serial READ# statement retrieves values for variables and strings of characters from the specified file.

READ# file number ; variable list

Before you can re-use data which has been stored in a data file with a PRINT# statement, you must read the data back into computer memory. The data is not erased from the file; it is merely copied into the variables specified in the same order in which it was stored with the PRINT# statement. Therefore, variables do not have to have the same names specified in the PRINT# statement. Reading begins after the last item printed or read on the specified file. To begin reading from the beginning of the file, you must reposition the pointer (see Repositioning the Pointer) or do another ASSIGN. Data can be updated and restored into the file or into a new file.

Data in the form of strings can be read from a file created with the SAVE statement.

In order to retrieve all of the information stored, your READ# statement(s) data list must match in number and type (string vs. numeric) the PRINT# statement(s) data list previously stored. If the READ# statement list specifies more data items than were originally stored, an EOR (or EOF if END was printed) error occurs, meaning there is no more data.

Data that is read must correspond to the type – numeric or string – that was printed. However, a numeric data item need not be of the same precision. Precision is automatically converted. You can also print an array and read back simple variables or other arrays, and vice versa. Here is an example using a data file called "XX".

```
  10   F=2.17598824
  20   C=3
  30   PRINT "FULL-PRECISION:",LIN(1),"F=";F,"C=";C
  40   CREATE "XX",4          ! Create a 4-record file
  50   ASSIGN #1 TO "XX"
• 60   PRINT #1,1;F,C         ! Record values of F & C into file
  70   ASSIGN #2 TO "XX"
  80   SHORT B,D
• 90   READ #2;B,D            ! Read values for short-precision
                                variables
  100                         ! Values are read from full-
                                precision values
  110  PRINT "SHORT-PRECISION:",LIN(1),B,D
  120  END

FULL-PRECISION:
F= 2.17598824          C= 3
SHORT-PRECISION:
 2.17599                 3
```

Notice that value of F is rounded when used as the value for B.

However, an overflow or underflow can occur. This is illustrated by the following example.

```
  10   ASSIGN #1 TO "XX"         ! FILE FROM PREVIOUS EXAMPLE
• 20   PRINT #1,4;2.5E99         ! PRINTS FULL-PRECISION NUMBER
  30   SHORT A
• 40   READ #1,4;A               ! READS SHORT-PRECISION NUMBER
  50   PRINT A
  60   END
```

This causes `ERROR 21 IN LINE 40` to be displayed and a beep to occur. To avoid the error, `DEFAULT ON` can be executed. The default value is used.

Here's an example of corresponding `PRINT#` and `READ#` operations -

```
  10   FIXED 2
  20   OPTION BASE 1
  30   DIM A(50)
  40   MAT A=(PI)
  50   ASSIGN #1 TO "XX"
• 60   PRINT #1;X,Y,A(*)
• 70   PRINT X,Y,A(*);
  80   DIM X(25),Y(25)
  90   ASSIGN #6 TO "XX"
• 100  READ #6;X(*),Y(*),A,B
  110  PRINT X(*),Y(*)
  120  END
```

Notice that 52 items are printed and 52 are read; they don't need to match as far as simple or array variable goes. Arrays are stored as a series of single data items with no regard to dimensionality.

In serial read mode an EOR mark is ignored, causing the file pointer to skip to the next record in an attempt to read data.

Random File Access

Random file access is used to store or retrieve data items from a specific defined record.

Random file access requires you to specify which defined record you wish to access. The pointer is positioned at the beginning of that defined record.

11

Random Printing

The random PRINT# statement is like the serial PRINT# statement except that it records data onto the file starting at the beginning of the specified record. However, EOF marks at the end of records aren't ignored. The data can't be larger than the record.

PRINT# file number, defined-record number[; data list [, END]]
PRINT# file number, defined-record number [; END]

The data list is identical to that used in the serial PRINT# statement. The random PRINT# statement records data into the specified record of the file. Printing starts at the beginning of the specified defined record. Any previous data in the record is overwritten. Any data not overwritten because the new logical record is shorter is inaccessible via that pointer. Specifying END causes an EOF mark to be printed after the data or at the beginning of the record (second syntax).

The written record set down by the list(s) of data must fit in the defined record, otherwise an EOR error occurs. If you attempt to specify a defined record number greater than the number specified in the CREATE statement, an EOF error occurs.

When no data list or END is specified, an EOR is printed in the first word of the record, which makes the data in that record inaccessible.

Here is an example –

```
  10   ASSIGN #1 TO "XX"
• 20   PRINT #1,1;A,B
• 30   PRINT #1,2;B,C
• 40   PRINT #1,3                    ! PRINTS EOR IN FIRST WORD
  50   END
```

Records 1 and 2 each have two values in them. Record 3 has an EOR in the first word.

Random Reading

The random READ# statement is like the serial READ# statement except that reading of data into the computer begins at the beginning of the specified defined-record and won't read past an EOR or EOF mark.

READ# file number, defined record number [; variable list]

Again, as in the serial READ# statement, the variables into which you read values do not necessarily have to have the same names or precision type specified in the PRINT# statement.

If the number of items making up the data list is greater than the data in the defined record, an EOR occurs.

11

Here's an example –

```
230  READ #1,1;X,Y
240  READ #1,2;A,B(1,2)
```

These two operations retrieve the data stored in the previous example.

Repositioning the Pointer

If the variable list is omitted, the pointer is repositioned to the beginning of the specified record. To reposition the pointer to the beginning of a file (for use with serial file access) execute –

READ# file number, 1

Random vs. Serial Method

The decision to choose random or serial methods depends upon the structure of the data which is to be recorded and retrieved. Serial file usage makes the most efficient use of the storage medium by packing all data tightly in the file. However, the data must be retrieved from the beginning of the file and therefore an item in the middle of a file cannot be accessed until all data coming before it is accessed. Random file usage is less efficient in its use of the storage medium but it provides access to data at various points (logical records) within the file without previously accessing the data which comes before.

Storing and Retrieving Arrays

Entire arrays can be stored and retrieved, using either serial or random access, by use of the MAT PRINT# and MAT READ# statements.

MAT PRINT# file number [, defined record number]; array variable
[, array variable...] [, END]

MAT READ# file number [, defined record number]; array variable
[(redim subscripts)] [, array variable [(redim subscripts)] , ...]

Arrays are stored and retrieved element by element without regard to dimensionality with the last subscript varying fastest.

Here's an example –

```
  10   ASSIGN #1 TO "XX"
  20   OPTION BASE 1
  30   DIM A(2),B(4),X(2),Y(2),Z(2)
  40   MAT A=(121)
  50   MAT B=(212)
• 60   MAT PRINT #1,3;A,B       ! PRINTS SIX VALUES
  70   MAT PRINT A;B;
• 80   MAT READ #1,3;X,Y,Z      ! READS SIX VALUES
  90   MAT PRINT X;Y;Z;
  100  END
```

```
 121  121

 212  212  212  212

 121  121

 212  212

 212  212
```

Arrays can also be printed and read with the `PRINT#` and `READ#` statements. Lines 60 and 80 above could also read –

```
60   PRINT #1,3;A(*),B(*)
80   READ #1,3;X(*),Y(*),Z(*)
```

11

Determining Data Type

The type function is used to determine what type of data the pointer will access next.

TYP ([-] file number)

The possible values for the function and their meanings are –

Value	Meaning
0	Option ROM missing or data pointer lost.
1	Full precision number.
2	Total string
3	End-of-file mark
4	End-of-record mark
5	Integer precision number
6	Short precision number
7	Unused
8	First part of a string
9	Middle part of a string
10	Last part of a string

If the file number is negative, the data pointer doesn't move. If it is positive, the pointer moves until it is positioned at something other than an EOR mark. In effect, a negative file number causes a random read. A positive file number causes a serial read, ignoring EOR marks.

Trapping EOR and EOF Conditions

Normally, encountering an EOF or EOR during a random access READ# or PRINT# operation or encountering an EOF during serial access causes a fatal error. The ON END# statement is a declarative which causes a branching operation to occur when an EOF or EOR is encountered.

ON END# file number GOTO line identifier
ON END# file number GOSUB line identifier
ON END# file number CALL subprogram name

Specifying ON END disables OVERLAP mode for that file. The routine branched to should service the EOF or EOR condition. Here's an example –


```
  10   DIM A(5,5),B(5,5),Q(100)
  20   CREATE "DATA1",4
  30   CREATE "DATA2",4
  40   ASSIGN #3 TO "DATA1"
  50   PRINT #3;A(*)
  60   ASSIGN #4 TO "DATA2"
  70   PRINT #4;B(*)
  80   ASSIGN #1 TO "DATA1"
• 90   ON END #1 GOSUB Reposition
  100  FOR I=1 TO 22
  110  READ #1;Q(I)
  120  PRINT Q(I);
  130  NEXT I
  140  STOP
• 150 Reposition:  ASSIGN #1 TO "DATA2"
  160  RETURN
```

If an EOF is encountered while reading values, another file, "DATA 2" is opened and used.

ON END is disabled during an INPUT, LINPUT or EDIT response request. ON END can interrupt ON ERROR and ON KEY routines.

An ON END# declarative is deactivated with the OFF END# statement.

OFF END# file number

EOR Errors

To recover from EOR errors, you can either shorten the data in precision or amount, or purge and recreate the file with the defined records longer or more numerous.

The following example illustrates a condition in which an EOR condition is generated.

```
10   CREATE "SHUN",2,16
20   ASSIGN #1 TO "SHUN"
30   DIM A$[20]
40   A$="ABCDEFGHIJKLMNOPQR"
50   PRINT #1,1;A$
60   END
```

Execution causes an EOR condition (ERROR 60); A$ is longer than the record. The EOR condition can be avoided by increasing the number of bytes in "SHUN" or changing line 50 to read `PRINT# 1;A$`.

The following example shows how an EOF can be generated.

```
10   CREATE "IVNESS",5,10
20   ASSIGN #2 TO "IVNESS"
30   DIM B$(6)[2]
40   FOR I=1 TO 6
50      B$(I)=CHR$(I+32)
60      PRINT B$(I)
70      PRINT #2;B$(I)
80   NEXT I
90   END
```

An EOF is generated when I = 6, B$(6) is "after" the end of file IVNESS.

Data Storage

When storing data, it is possible to optimize the use of your storage medium by minimizing the amount of unused space. The best way to do this is to create your files so they are suited to the amount of data you wish to store and to storage medium capacities.

The following tables indicate how many bytes are needed to store various variables.

Single Variable	
Full precision	8 bytes
Short precision	4 bytes
Integer precision	4 bytes
String	1 byte per character + 4 bytes + 4 bytes each time string crosses into a new defined record.

Array Variable	
Full precision	8 bytes × dimensioned number of elements
Short precision	4 bytes × dimensioned number of elements
Integer precision	4 bytes × dimensioned number of elements
String	4 bytes per element + total needed for all strings as defined above.

By summing up how many bytes of storage your data requires, you can tailor your file and defined record lengths to suit your needs and minimize waste. However, keep in mind that a file always begins on a new physical record. If a file requires a total of 520 bytes (2 physical records plus 8 bytes), 248 bytes are unused, and therefore, are wasted space.

11

Buffering a File

The BUFFER statement is used to attach a buffer from user Read/Write Memory to a file number to reduce device wear and increase efficiency by reducing device transfers.

BUFFER# file number

The BUFFER statement allocates buffers from the main user Read/Write Memory by attaching a 256-byte, semi-permanent buffer to the specified file number. PRINT# statements cause transfers to the buffer (rather than to the actual medium); when the buffer is full, its contents are dumped to the medium. READ# statements fetch data from the buffer until it is exhausted; the buffer is then refilled from the medium.

Buffering files is most advantageous if all files being accessed on a specific device are buffered.

A buffer that is assigned to a file number is also dumped under these conditions –

- ASSIGNing that number to a different file
- A PAUSE, (PAUSE), STOP or END

All buffers are dumped when any ASSIGN is done.

A buffer is returned to main Read/Write Memory under these conditions –

- RUN
- SCRATCH A
- END
- Reset
- STOP
- Closing the file (see next section)
- Returning from the subprogram in which the file being buffered was opened.

The BUFFER statement can't be executed from the keyboard.

Closing a File

The ASSIGN statement is also used to close a file; any subsequent attempts to access that file number result in an error. It is recommended that a file be closed before its number is assigned to another file. The two syntax shown below are equivalent.

ASSIGN * TO # file number
ASSIGN# file number TO *

Verifying Information

The CHECK READ statement is used to verify information written to a storage medium.

CHECK READ [# file number]

When no file number is specified, all storage operations are verified. The file number causes only PRINT# operations to that file to be verified. This is a bit for bit comparision.

CHECK READ has the additional function of forcing transfer to the medium of the current data record after every PRINT# operation. However, the BUFFER statement has precedence over CHECK READ. The data record is verified only when the buffer allocated by the BUFFER statement is dumped to the actual medium.

The CHECK READ operation reduces the speed of operations and increases wear on the tape cartridge. Use only when necessary.

The CHECK READ operation can be cancelled by executing the CHECK READ OFF statement.

CHECK READ OFF [# file number]

11

Protecting a File

The PROTECT statement is used to guard a file against accidental erasure, especially with disks.

PROTECT file specifier , protect code

The file specifier must specify an established file on a device.

The protect code is any valid string expression except the null string. Only the first six characters are recognized as the protect code.

Examples

```
10    REM  THESE ARE EXAMPLES OF PROTECT
20    PROTECT "DATA",Date$
30    PROTECT "NAMES:F8","XXX"
40    END
```

NOTE

A file on the tape cartridge can be purged using any protect code; it need not be the one it was protected with.

Purging a File

The PURGE statement eradicates any file (program, data, etc.) by removing its name from the name table in the directory, thereby preventing any access to the file.

PURGE file specifier [, protect code]

The protect code is necessary only if the file was previously protected. The records of the file are then returned to "available space", being combined with adjacent available records, if any.

Examples

```
10    REM   THESE ARE EXAMPLES OF PURGE
20    PURGE "TEMP"
30    PURGE "EXTRA:F8"
40    PURGE "BACKUP","KEY"      ! FILE WAS PROTECTED
50    END
```

Copying a File

The COPY statement is used to copy the information in a file into another file.

COPY source file specifier TO destination file specifier [, protect code]

The protect code is necessary only if the source file is protected.

Execution of the COPY statement causes all records of a file to be copied. The first file specified can be of any type. A check of the name of the destination file is made; an error is given if the name is present. If not, a file of the same characteristics as the source file is created. The same storage medium can be both source and destination.

Examples

```
10   REM  THESE ARE EXAMPLES OF COPY
20   COPY "FILE" TO "BACKUP:F8"
30   COPY "DATA1" TO "DATA2"           ! CAN BE SAME MEDIUM
40   COPY "PROGA" TO "PROGB","***"     ! PROGA WAS PROTECTED
50   END
```

The COPY statement is very useful for duplicating a storage medium. Each file can be copied individually, thus duplicating the entire medium.

Renaming a File

The RENAME statement is used to give a file a different name.

RENAME old file specifier TO new file name [, protect code]

Examples

```
10   REM  THESE ARE RENAME EXAMPLES
20   RENAME "JUNE1" TO "JUNE 2"
30   RENAME "TEMP" TO "FINAL","TRIAL"      ! TEMP WAS PROTECTED
40   END
```

11

Storing SFK Definitions

The typing aid definitions of all special function keys can be stored onto a mass storage medium using the STORE KEY statement.

STORE KEY file specifier

This creates a "KEY" file.

The stored definitions can be loaded back into the keys by executing the LOAD KEY statement –

LOAD KEY file specifier

Examples

```
10    REM  THESE ARE EXAMPLES OF STORE AND LOADKEY
20    STORE KEY "TEMPKY"
30    STORE KEY "AIDS:F8"
40    LOAD KEY "TEMPKY"
50    LOAD KEY "AIDS:F8"
60    END
```

Binary Programs

All binary routines currently in memory can be recorded separately from programs with the STORE BIN statement.

STORE BIN file specifier

Stored binary routines are retrieved and added to current binary routines using the LOAD BIN statement.

LOAD BIN file specifier

Examples

```
10    REM  THESE ARE EXAMPLES OF STORE AND LOADBIN
20    STORE BIN "ROUTIN"
30    STORE BIN "COMPIL:T15"
40    LOAD BIN "ROUTIN"
50    LOAD BIN "COMPIL:T15"
60    END
```

Storing Memory

The entire user Read/Write Memory state: programs, variables, keys, binaries – can be stored into a special memory file. The files table is not stored into the STORE ALL file, however.

STORE ALL file specifier

The file created by STORE ALL is very large; the minimum is 38 records. STORE ALL can't be executed during execution of a subprogram.

Memory can be returned to the state it was in previously by using the LOAD ALL command.

LOAD ALL file specifier

All files being used when the corresponding STORE ALL was executed must be reassigned.

Examples

```
REM  THESE ARE EXAMPLES OF STORE AND LOADALL
STORE ALL "MEMORY"
STORE ALL "2/3/78:F8"
LOAD ALL /"MEMORY"
LOAD ALL "2/3/78:F8"
```

The Tape Cartridge

This section covers general information for using the tape cartridge for mass storage operations.

NOTE

Occassionally when using the tape cartridge, unexpected high-speed movements may occur. Ignore these; they in no way affect usage, but merely assure proper tape tension.

11

Recording on the Tape

To record on the tape cartridge, the record tab must be in the rightmost position, in the direction of the arrow (as shown).

Write Protection

If the record tab is moved to the left, no information can be written to the tape. Information can only be read from the tape.

General Tape Cartridge Information

Rewind time	19 seconds
Initialization time	3 minutes
Tape length	42.67m (140 feet)
Number of tracks	2 independent tracks
Tape capacity	847 user-accessible physical records (216 832 bytes)
	42 files (directory entries)
Access rate (search speed)	11 770 bytes/second
Transfer rate	1 438 bytes/second
Typical tape life	50-100 hours
Typical error rate[1]	< 1 in 10^7 bytes
Mass storage unit specifier	:T15

1 This is dependent on the cleanliness of the tape head, tape care, and the cleanliness of the environment.

Rewinding the Tape

The REWIND statement rewinds the tape to its beginning.

REWIND [mass storage unit specifier]

If no parameter is specified, the default device is used. If it is not a tape cartridge, the statement is ignored. There is also a key REWIND in the system keys area which can be used to rewind the tape.

Operations which do not involve the tape cartridge can take place while the tape rewinds.

Mass Storage Errors

When using the tape cartridge, wear caused by contact between the tape and the read/write head can occur. If at any time, the tape makes rattling sounds while moving, or error 84, 87, 88 or 89 or a SPARE DIRECTORY ACCESS warning begin to occur frequently, it is advisable that steps be taken to prevent the loss of information stored on the tape.

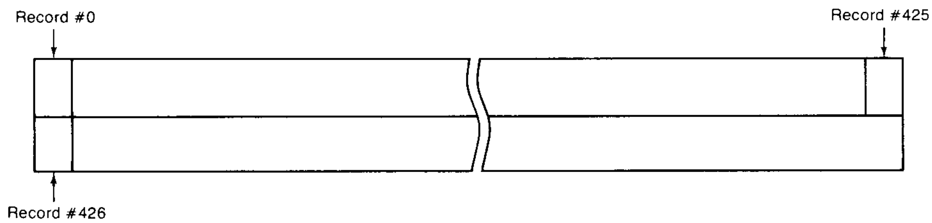
The first step is to clean the tape head and capstan as discussed in the Owner's Manual. If this does not alleviate the problem, the next step is to transfer the information to a new medium, retiring the worn tape. Continued use could cause loss of information or damage to the tape drive itself.

ERROR 81 can occur when either the tape drive or the cartridge itself fails. To determine the source of the problem, a different cartridge can be inserted. If ERROR 81 stops occurring, assume the tape itself is bad and replace it. If ERROR 81 continues to occur, the drive itself is bad. In this case, call your HP Sales and Service Office for assistance.

11

Optimizing Tape Use

The tape cartridge used with a 9835A/B has two tracks with 426 records on each track. Records are numbered consecutively; record 0 and record 426 are both at the same end of the tape, on different tracks. Thus, records 425 and 426 are at opposite ends of the tape. This can cause a situation in which one file spans two tracks, making access time-consuming and wearing to the tape.



To avoid this situation, you can create a dummy file in record number 425, making it impossible for one file to span two tracks. The following set of operations can be used on a tape with no files on it to create this dummy file.

```
10   CREATE "A",420
20   CREATE "DUMMY",1
30   PURGE "A"
40   END
```

The file DUMMY will be in record number 425; the first five records on the tape are used by the directory which is why file A is created with only 420 records.

Chapter 12
Editing and Debugging

Program Editing

A program in memory can easily be edited by entering the edit line mode. In the edit line mode, lines can be edited, deleted, or added. For more information about editing subprograms, see Chapter 9. This can be done while a program is running which causes the program to pause. The edit line mode is entered by executing the EDIT LINE command –

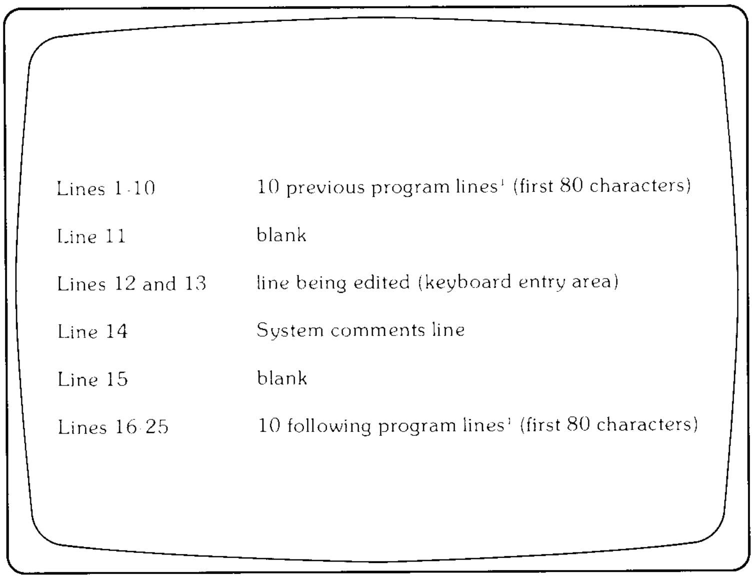
EDIT[1] [LINE] [line identifier [, increment value]]

Examples

```
EDIT             ! ACCESS LOWEST LINE IN MEMORY
EDIT 150         ! ACCESS LINE 150
EDIT Routine,5   ! ACCESS Routine, INCREMENT NEW LINES BY 5
```

1 There is a key in the program keys area which is defined as "EDIT". This key can be used to enter the word EDIT.

When the EDIT LINE command is executed, the line specified by the EDIT LINE command, or the first line in memory if one is not specified, is displayed in line 12 of the CRT (on the 9835A). Line 13 is also reserved for that program line. Here is a diagram of how the CRT looks in the edit line mode –



On the 9835B, the line being edited appears on the display.

The cursor can then be moved in the line and the line edited. When the line is the way you want it, press (STORE). The next highest numbered line is then displayed (in the keyboard entry area of the 9835A).

If you make an error and try to store the line, the machine beeps and displays a message. On the 9835A, line 14 of the CRT is used for errors. On the 9835B, the error message is displayed, replacing the line. Pressing (RECALL) brings the program line back into the display with the flashing cursor over the improper character.

To position a different line in the keyboard entry area, [↑] and [↓] can be used to move the program lines up or down. [↑] brings the next highest line into the editing line, while [↓] brings the previous line into the editing line.

1 If there are less than 10 lines before and after the line being edited, you may be experiencing a partial memory loss. Memory loss is covered in the Owner's Manual.

Increment Value

After the end of the program is reached, the next line number is generated. It is greater than the previous line number by the increment value or 10 if the increment isn't specified. The increment value must be an integer.

Automatic Indent

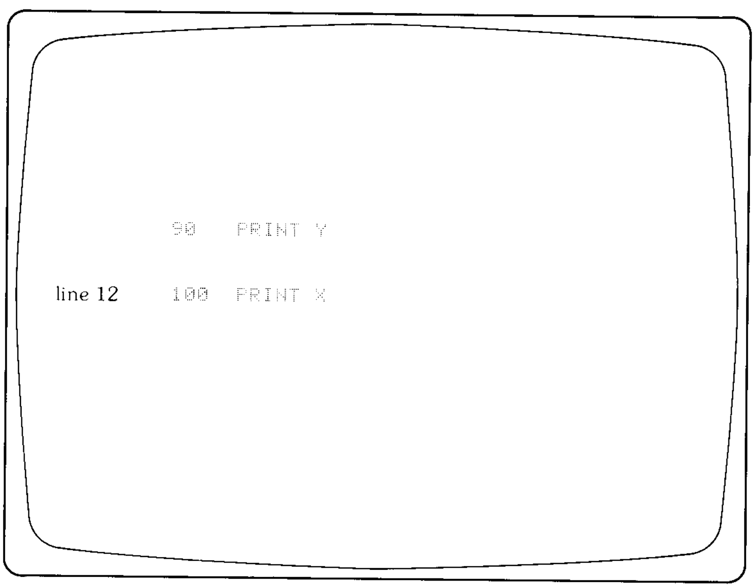
Using the edit line mode can allow you to automatically indent program lines. This is possible if you are adding lines at the end of the program or inserting lines (discussed next). If you indent a line and store it, when the next line is generated, the cursor is indented as many spaces as it was in the previous line.

Inserting Lines

Lines can easily be inserted between existing program lines while in the edit line mode. The insert line mode is entered by pressing –

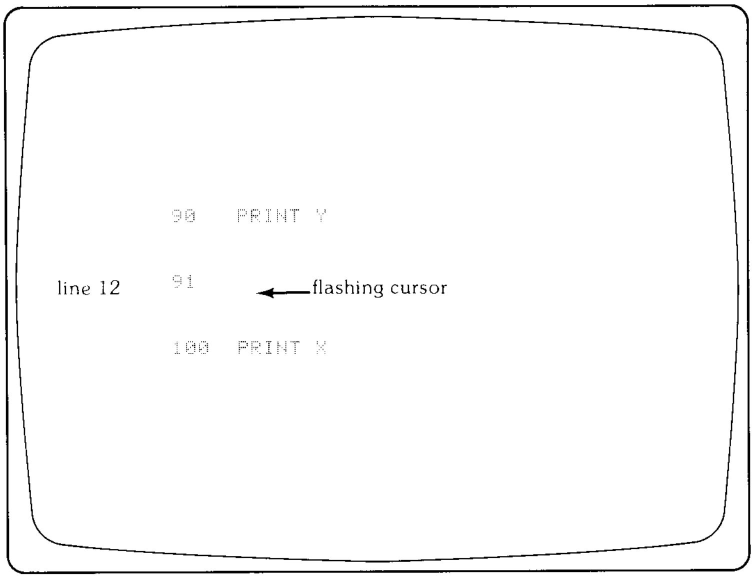
INS LN

Lines can then be inserted before the line which was in the keyboard entry area. A line number which is one greater than the previous line is generated and appears in the keyboard entry area (of the 9835A). For example, say the CRT looks like[1] –



1 Note to 9835B users: On the 9835B, the display corresponds to line 12 of the 9835A while in edit line mode.

Pressing [INS LN] would cause it to look like –



When line 91 is stored, line number 92 is generated. This continues until the insert mode is exited by one of the following –

- Pressing [INS LN] again
- Pressing [DEL LN]
- Rolling the program with [↑] or [↓]
- Changing the line number
- There is no more room between lines to insert another line. When this happens, the machine beeps and a warning appears –

```
No room to insert line
```

A line can also be inserted into a program when not in the edit line mode by typing in the line number and line itself and storing it.

Deleting Lines

In the edit line mode, the line currently in the keyboard entry area can be deleted from memory by pressing –

[DEL LN]

The next line is then displayed in the keyboard entry area, and the rest of the lines scroll up.

The DEL (delete) command is used to delete a line or section of a program when not in the edit line mode.

DEL first line identifier [, last line identifier]

If only one line identifier is specified, then only that line is deleted. Specifying two line identifiers causes that block of lines to be deleted. For example, to delete lines 40, and 100 through 150 from a program, execute –

```
DEL 40
```

and

```
DEL 100, 150
```

To delete a SUB or multiple-line DEF FN statement, the entire subprogram must be deleted.

Exiting the Edit Line Mode

The edit line mode is exited by pressing PAUSE.

STEP CONT STOP RUN EXEC or SHIFT CLEAR LINE can also be used.

Debugging a Program

Tracing a program is a convenient method of debugging the logic errors in the program. There are two types of tracing which allow the logic flow and variable assignments of a running program to be monitored. Output from TRACE operations goes to the system comments line. When tracing, it is advisable to set the print all mode (see Chapter 2) and specify a printer other than the CRT as the print all printer so that TRACE outputs are more permanent.

Tracing statements can be programmed or executed from the keyboard. They do not increase program Read/Write Memory requirements when executed from the keyboard or from the program. In general, trace modes aren't cumulative. If two TRACE statements of the same type are executed, the second cancels the first.

Tracing operations cause the computer to temporarily revert to SERIAL mode even if OVERLAP is in effect.

12

Tracing Program Logic Flow

The TRACE statement is used to trace program logic flow in all or part of a program. When any branching occurs in a program, both the line number of the line where the branch is from, and the line number of the line where the branch is to are output.

TRACE [beginning line identifier [, ending line identifier]]

When a branch occurs, the output is –

TRACE--FROM line # TO line #

If no line identifiers are specified, all branches in the program are monitored. When one line identifier is specified, tracing doesn't begin until that line is executed. When both line identifiers are specified, tracing begins when the first line is executed and continues (regardless of where the program is executing), then stops when the second line is executed.

Delayed Tracing

Delayed tracing using the TRACE WAIT statement in conjunction with any other TRACE statement causes a specified delay to occur after each statement which causes a trace output. It is useful for monitoring and examining trace output as it occurs.

TRACE WAIT number of milliseconds

The delay is specified by a numeric expression in the range −32 768 through 32 767 which indicates the number of milliseconds after each trace printout. A negative number defaults to zero.

Tracing with PAUSE

To check whether or not a line in a program is reached, or to monitor the number of times a specified line is executed, use the TRACE PAUSE statement.

TRACE PAUSE [line identifier [, numeric expression]]

If no parameters are specified, execution pauses when this statement is executed; the next line to be executed is displayed. This allows you to pause a running program and know where it is paused, which is not possible with the PAUSE statement.

When only the line identifier is specified, the running program stops when execution reaches the specified line, but **before** the line is executed. When the numeric expression is specified, it is rounded to an integer, call it N. The program stops when the specified line is reached for the Nth time; the line isn't executed. Execution can be resumed with that line by pressing (CONT). Every subsequent time the line is encountered, the program pauses before the line is executed.

This type of tracing can be disabled by letting the line identifier be one that is not a line identifier in memory. The most efficient way is to let it be a lower number than the lowest numbered line in memory.

12

Tracing the Values of Variables

To trace changes in values of variables without using an output statement, use the TRACE VARIABLES statement.

TRACE VARIABLES variable list

The variable list can contain simple numeric and string variables, and array identifiers; there can be one to five items separated by commas. The value of any variable which changes is printed. The output is –

TRACE--LINE line number, variable name [(subscripts)]= value

The line number is the line in which the change occurred. If the change comes from a live keyboard operation, the line number is replaced by KEYBOARD. The new value of the variable is indicated. In the case of an array, the values of the subscript(s) at the time are printed following the name.

When an entire array changes value, the printout is –

TRACE--LINE line number, array name (*) CHANGED VALUE

Tracing variables also detects changes in subprograms of variables passed by reference. For example, suppose –

TRACE VARIABLES A, B

is executed and A is passed by reference to a subprogram. If the corresponding variable in the subprogram is changed, a trace message for A occurs.

12

To trace all variables with the ability to specify lines, use the TRACE ALL VARIABLES statement.

TRACE ALL VARIABLES [beginning line identifier [, ending line identifier]]

When no line identifiers are specified, all variables are traced throughout the program. When one line identifier is specified, tracing begins after that line is executed. The ending line identifier causes tracing to stop after that line is executed.

TRACE ALL VARIABLES cancels and is cancelled by TRACE VARIABLES.

This method of tracing can be turned off by letting the first line identifier be a line identifier which is not in memory such as an undefined label or line number which is lower than the lowest line number in memory.

Comprehensive Tracing

To trace all program logic and variables, like executing both TRACE and TRACE ALL VARIABLES, use the TRACE ALL statement.

TRACE ALL

Either 'part' of the TRACE ALL mode can be altered without cancelling the other part. For example, if TRACE VARIABLES A, B is executed after TRACE ALL, tracing of all variables is cancelled, and only A and B are traced, but the TRACE part of TRACE ALL is not affected.

Although the volume of printout is high, TRACE ALL is useful if a logic problem in a program hasn't been isolated with selective tracing.

Canceling Trace Operations

All tracing statements are cancelled by executing SCRATCH, SCRATCH A, SCRATCH C, SCRATCH V, SCRATCH P, or the NORMAL statement –

NORMAL

Error Testing and Recovery

Run-time errors are those which occur only when a program is running. Dividing by zero is an example. A run-time error normally halts execution. Through use of the ON ERROR statement, run-time errors can be recovered from so that execution can continue with the specified line after execution of the line in which the error occurred. The ON ERROR statement specifies a branching which takes place after an error occurs.

ON ERROR GOTO line identifier
ON ERROR GOSUB line identifier
ON ERROR CALL subprogram name[1]

12

The ON ERROR statement declares what should happen if an error occurs. An ON ERROR statement need be executed only once in each program segment to establish the ON ERROR condition. Execution of another ON ERROR statement cancels the previous one.

When a run-time error occurs and the ON ERROR condition has been established, execution is transferred to the specified line. Then the ERRN, ERRL, and ERRM$ functions (discussed next) could be tested, error recovery procedures or "DISP ERRM$" could be executed. The error is "ignored" if the statement referenced by a GOSUB is a RETURN statement; execution continues with the line after the one in which the error occurred.

NOTE

When a program is running in OVERLAP mode, ON ERROR won't trap most I/O errors (54-103). It is advisable to use SERIAL mode when trapping errors with ON ERROR.

If the recovery routine contains an error, it is possible to program into a endless loop. It can be stopped by pressing (PAUSE) or (STOP).

If the ON ERROR statement specifies a GOSUB or CALL, computer priority is set at the highest level until RETURN is executed. This means that the routine can't be interrupted by any other interrupts.

A routine accessed with GOTO can be interrupted because system priority isn't changed.

1 Can't pass parameters

Error Functions

One string and two numeric functions can be used with ON ERROR.

ERRL — The error line function returns the line number in which the most recent program execution error occurred.

ERRN — The error number function returns the number of the most recent program execution error.

12

ERRM$ — The error message string returns the most recent program execution error message, a combination of ERRL and ERRN.

ON ERROR is disabled with the OFF ERROR statement –

```
OFF ERROR
```

Appendix A
HP Compatible BASIC

The BASIC language as implemented on the 9835A/B Computer is an enhanced form of HP Compatible BASIC. HP Compatible BASIC consists of statements, functions, operators and commands that will be implemented in new HP BASIC machines. The HP Compatible BASIC is implemented in the 9835A/B as Level I. Level I refers to the highest performance computational products; thus, any program consisting entirely of Level I BASIC language can be transported to any Level I BASIC machine.

Below is a list of HP Level I BASIC. Contact you HP Sales and Service Office to obtain information concerning the transporting of programs between machines.

Operators

+	=	AND
−	>	OR
*	<	NOT
/	≥	EXOR
∧	≤	
DIV	< >	&
MOD		

Functions

ABS	SIN	COL
EXP	COS	DET
INT	TAN	DOT
LGT	ASN	ROW
LOG	ACS	SUM
MAX	ATN	
MIN		
RND	LWC$	TYP
SGN	REV$	
DROUND	UPC$	LIN
PROUND	CHR$	SPA
FRACT	LEN	TAB
PI	NUM	PAGE
SQR	POS	
	RPT$	
ERRL	TRIM$	
ERRN	VAL	
ERRM$	VAL$	

Statements

ASSIGN
BEEP
CALL
COM*
COPY
CREATE
DATA
DEF FN
DEG
DIM
EDIT
END
FIXED
FLOAT
FN END
FOR
NEXT
GOSUB
GOTO
GRAD
IF
IMAGE
INPUT
INTEGER
LET
LINPUT
MAT array = array
MAT array = array + − * / . = < > # < ≤ > ≥ array
MAT...CON

* The type words INTEGER, SHORT and REAL are the only ones which can be specified in a COM statement or formal parameter list. Dimensionality of arrays is limited to 6 dimensions.

MAT array = (num. exp.)
MAT array = (num. exp.) + − * / = < > # < ⩽ > ⩾ array
MAT array = array + − * / = < > # < ⩽ > ⩾ (num. exp.)
MAT...INV
MAT...TRN
MAT...IDN
MAT...ZER
MAT...CSUM
MAT... RSUM
MAT INPUT
MAT PRINT
MAT PRINT#
MAT READ
MAT READ#
OFF END
OFF ERROR
ON END
ON ERROR
ON...GOSUB
ON...GOTO
OPTION BASE
PAUSE
PRINT
PRINT#
PRINT USING
PURGE
RAD
RANDOMIZE
READ
READ#
REAL
REDIM
REM
RESTORE
RETURN
SHORT
STANDARD
STOP
SUB
SUBEXIT
SUBEND
WAIT

Appendix B
Advanced CRT Techniques

This appendix introduces you to the more advanced printing capabilities of the CRT of the 9835A.

Some of the special capabilities are accessed by using various ASCII* control characters. See the table in Appendix E for a complete list of ASCII characters. Another capability allows the CRT special features: inverse video, blinking, and underline modes to be accessed in a program rather than using the CONTROL key. A third capability uses escape codes to selectively address any location on the CRT. The escape code sequences are compatible with those used by HP 2640-series terminals.

The examples in this appendix are meant to be tried because it is impossible to show many of the CRT capabilities on the printed page.

A summary of escape code sequences can be found at the end of this appendix.

CRT Memory

Every line that is printed to the CRT is stored in the CRT memory. This memory can hold 50 80-character lines. Fewer longer lines or more shorter lines can be stored. When the memory becomes full, each new line printed to the CRT causes the oldest line in memory to be lost. All lines in CRT memory can be viewed with ⬆ or ⬇. The CRT memory is cleared with SHIFT CLEAR LINE, a formfeed character (`PRINT PAGE`, `PRINT CHR$(12)`) or with ^{E}C.

* American Standard Code for Information Interchange.

CRT Special Features

The special features: blinking, underline and inverse video can be accessed in a program by using the CHR$ function or escape code sequence within an output statement. Any time a mode is accessed or cleared, one character is added to the length of what is output, though it is an unprinted character. Any combination of the features can be accessed by outputting –

CHR$ (n)

Where n is an integer in the range 128 through 159 and specifies which combination of features is to be accessed. Remember, larger numbers are reduced MOD 256.

The following table shows which numbers provide access to which features.

CLR	IV	BL	IV BL	UL	IV UL	BL UL	IV BL UL
128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151
152	153	154	155	156	157	158	159

The following escape code sequence can be used to access the special features –

EC&dX

X can be –

X	Result
@	CLR
A	BL
B	IV
C	IV, BL
D	UL
E	UL, BL
F	UL, IV
G	UL, IV, BL

CLR – Clear all special features
IV – Inverse video
BL – Blinking
UL – Underline

All special features accessed with CHR$ remain in effect until specifically cleared. Those accessed with the escape code sequence remain in effect until the end of the line or until another one is specified. This can be done with the CLR feature above or by pressing SHIFT CLEAR LINE.

Here are some examples to try –

```
10    PRINTER IS 16
20    PRINT "THE FOLLOWING LINES SHOW";
30    PRINT " EXAMPLES OF THE VARIOUS CRT FEATURES"
40    PRINT CHR$(135);"All three features"
50    PRINT CHR$(135);"Notice underline is green"
60    PRINT CHR$(133);"Inverse video, underline"
70    PRINT CHR$(132);"Underline"
80    PRINT CHR$(130);"This line blinks"
90    PRINT CHR$(128);"All features cleared"
100   END
```

This example illustrates the differing effects of commas and semicolons –

```
10    PRINT "These lines illustrate the effects";
20    PRINT " of ; and , on field lengths."
30    PRINT "First feature is inverse video, ";
40    PRINT "second feature is underline.";LIN(1)
50    A$="**"
60    PRINT "SEMICOLON AFTER ** :"
70    PRINT "Comma after turning feature on:"
80    PRINT CHR$(129),A$;CHR$(132),A$;CHR$(128)
90    PRINT "Semicolon after turning feature on:"
100   PRINT CHR$(129);A$;CHR$(132);A$;CHR$(128)
110   PRINT LIN(2)
120   PRINT "COMMA AFTER ** :"
130   PRINT "Comma after turning feature on:"
140   PRINT CHR$(129),A$,CHR$(132),A$,CHR$(128)
150   PRINT "Semicolon after turning feature on:"
160   PRINT CHR$(129);A$,CHR$(132);A$,CHR$(128)
170   END
```

Using Control Codes

ASCII characters are letters, numbers, characters and codes which each correspond to a unique 7-bit byte pattern. Each character also has decimal, binary and octal representations. The first 32 are control codes which pass control information between devices.

The control codes can be accessed for output using (CONT'L) or the CHR$ function. A two-letter symbol specifies the control code. To determine what keys can be used with (CONT'L) to obtain a control code, the ASCII table in Appendix E can be used. By following the line all the way across from the desired code, the two or three keys which produce the desired character when pressed with (CONT'L) can be determined. For example, LF (linefeed) can be obtained by pressing (CONT'L) with either (SHIFT)(*/8), (J), (SHIFT)(J), or (*). The DEL character is the only one that can't be obtained using (CONT'L).

Basic control operations on the 9835A utilize five control codes which affect output to the CRT or internal thermal printer. Here are the codes and their various results –

Control Code	CRT (DISP)	CRT (PRINT)
BELL	Beep	Beep
BS(backspace)	Back up and replace	Back up and replace
LF (line feed)	Nothing	Generate line feed only
FF (form feed)	Clear display line	Clear printout area and CRT memory
CR (carriage-return)	Clear display line	Return to beginning of line

With the exception of the control codes described above, HT (horizontal tab, `CHR$(9)`) and ESC (escape code, `CHR$(27)`) which are discussed later in this appendix, all other control codes are ignored by the 9835A/B.

Here are some examples –

Command	Output
`PRINT "ABC␈D"`	`ABD`
`PRINT "ABC␌"`	Clears CRT
`PRINT "ABC␍DE"`	`DEC`
`PRINT "ABC␊DE"`	`ABC` `   DE`

Considerations

There are a few things to consider when using control codes in programming.

Control codes used within a BASIC statement are executed even when a program is listed. This can produce some undesirable results. For example, try listing these program lines –

```
10 PRINTER IS 16
20 PRINT " HEADING␊␊␊"  !␊ is control J
30 PRINT "<=>␍///"      !␍ is control M
```

Thus, a program listing will be more readable if control codes are generated with the `CHR$` function.

Another thing to consider is the fact that escape codes and control codes remain in effect until they are deactivated.

CRT Selective Addressing

Introduction

The top twenty lines of the CRT are known as the printout area. All lines printed to the CRT are stored in CRT memory which was discussed at the beginning of this appendix. Through the use of escape code sequences, any line in CRT memory can be selectively addressed and modified.

The operations available are as follows –

- Cursor Positioning
  - Absolute addressing
  - Relative addressing
  - Backspace
  - Space
  - Up
  - Down
  - Set tab
  - Clear tab
  - Tab
  - Home position – first row
  - Home position – row after last row
- Display Positioning
  - Roll up
  - Roll down
  - Next page
  - Previous page
  - Memory lock
- Editing
  - Delete line
  - Insert line
  - Clear to end of line
  - Clear to end of screen
  - Insert character
  - Delete character

B

Selective cursor addressing and the other operations which are covered in the rest of this appendix are very useful for form filling and text processing applications. It is recommended that you use PRINT USING to output the escape code sequences to avoid unexpected carriage return/linefeeds which can occur from length added to the output.

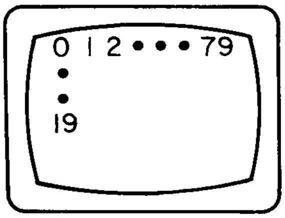
There are two example programs at the end of this appendix which combine many of the operations to manipulate output.

The Cursor

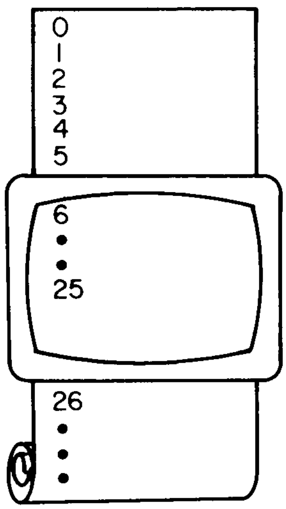
Any location on the screen can be addressed and a non-visible cursor specified as being there. (This cursor is **not** the same as the flashing cursor which is present in normal keyboard usage.) This cursor refers to a logical print position in CRT memory where the next character will be printed. In this appendix, the word "cursor" always refers to the logical print position.

Addressing Schemes

The printout area is addressed using rows 0 through 19 and columns 0 through 79. The following drawing illustrates this –



CRT memory is addressed using columns 0 through 79. The number of rows depends on the size of the memory installed in your computer and on line length. The maximum number of lines was covered at the beginning of this appendix. The following drawing illustrates addressing of CRT memory –



In this drawing, line 6 of the CRT memory is positioned on line 0 of the printout area.

Setting the Cursor Position

The cursor can be set to any character position in the 20 lines of the printout area using absolute or relative addressing, or a combination absolute and relative addressing.

Absolute Addressing

The cursor can be set to an absolute row and column position with any of the following escape code sequences –

```
␛&a nn r nn C
␛&a nn c nn R
␛&a nn y nn C
␛&a nn c nn Y
```

Here are some guidelines for using these escape code sequences –

- nn specifies a one or two-digit number which is used to specify the row and column number. The digits preceding the R(r) specify the row number of CRT memory. The digits preceding the Y(y) specify the row number of the printout area. The digits preceding the C(c) specify the column number.
- The first column of the printout area is addressed using 0. The maximum column address is 79; if anything greater is specified, 79 is used.
- The first row of either CRT memory or printed output is addressed using 0.
- If the specified row of CRT memory is not on the CRT screen, the display will roll up or down as necessary.

B

The cursor can be moved within a row by omitting the R and preceding digits. Here is the escape code sequence –

```
␛&a nn C
```

Similarly, the cursor can be moved within a column by omitting C and preceding digits. Here are the escape code sequences –

```
␛&a nn R
␛&a nn Y
```

Here are some examples of absolute addressing –

`PRINT USING"#,K";CHR$(27)&"&a25r60C"`	Moves the cursor to row 25, column 60.
`PRINT USING"#,K";CHR$(27)&"&a60c17R"`	Moves the cursor to column 60, row 17.
`PRINT USING"#,K";CHR$(27)&"&a15R"`	Moves the cursor to row 15, current column.
`PRINT USING"#,K";CHR$(27)&"&a30C"`	Moves the cursor to column 30, current row.
`PRINT USING"#,K";CHR$(27)&"&a7Y"`	Moves the cursor to row 7 of the printout area, current column

Relative Addressing

The cursor can also be repositioned using relative addressing. From its current position, the cursor can be moved up (negative number) or down (positive number), left (negative number) or right (positive number). Here are the escape code sequences to use –

```
␛&a S nn r S nn C
␛&a S nn c S nn R
␛&a S nn y S nn C
␛&a S nn c S nn Y
␛&a S nn C
␛&a S nn R
␛&a S nn Y
```

Here are some guidelines for using these escape code sequences –

- nn specifies a one or two digit number which is used to specify the number of rows and/or columns the cursor is to move. The digits preceding the R(r) or Y(y) specify the number of rows; the digits preceding the C(c) specify the number of columns.
- S specifies a sign : + or –. A plus sign (+) specifies right or down. A minus sign (–) specifies left or up.
- If the number of columns specified is greater than the number of columns remaining after the cursor in the current line, the cursor is positioned in the first column (negative movement specified) or in the last column (positive movement specified). If the number of rows specified in the negative direction is greater than the current row, the cursor is positioned in the first row.

Here are some examples of relative addressing –

`PRINT USING"#,K";CHR$(27)&"&a+8r-10C"`	Moves the cursor down 8 rows, left 10 columns from its current position.
`PRINT USING"#,K";CHR$(27)&"&a+7C-11R"`	Moves the cursor right 7 columns, up 11 rows from its current position
`PRINT USING"#,K";CHR$(27)&"&a -8R"`	Moves the cursor up 8 rows from its current position
`PRINT USING"#,K";CHR$(27)&"&a+10C"`	Moves the cursor right 10 columns from its current position.

Combining Absolute and Relative Addressing

The cursor can be positioned to a new position using a combination of absolute and relative addressing. Here are some examples –

`PRINT USING"#,K";CHR$(27)&"a&+8r60C"`	Moves the cursor to column 60 and down 8 rows from its current row.
`PRINT USING"#,K";CHR$(27)&"&a -15C 10R"`	Moves the cursor to row 10 and left 15 columns from its current position.

Moving the Cursor

The following escape code sequences can also be used to move the cursor –

␛A	Move cursor up one row
␛B	Move cursor down one row
␛C	Move cursor right one column
␛D	Move cursor left one column
␛H	Moves the cursor to first row of CRT memory, first column
␛F	Move cursor to row after last row of CRT memory, first column

These escape code sequences can be used very easily by defining special function keys to set the cursor position, then move it up, down, left, and right.

␛A – ␛D cause the cursor to "wrap around" when the edge of the screen is reached. When the cursor is being moved to the right it wraps around on to the next line. When the cursor is being moved to the left, it wraps around onto the previous line. ␛F can be used to return the cursor for normal printing after using cursor-moving escape code sequence. ␛H and ␛F cause the lines to scroll, if necessary.

Here is an example using cursor moving to fill in blanks in a form letter.

```
  10   PRINTER IS 16
  20   DIM Name$[25],Magazine$[50]
  30   PRINT PAGE
• 40   A$=CHR$(27)&"D"     ! MOVES CURSOR TO LEFT
• 50   PRINT USING "#,K";"Dear __________"&RPT$(A$,10)
  60   INPUT "Name?",Name$
  70   PRINT Name$&",         "
• 80   PRINT USING "#,K";"Your subscription to __________"&RPT$(A$,10)
  90   INPUT "Magazine?",Magazine$
  100  PRINT Magazine$&" is ";LIN(1);"about to expire."
  110  PRINT "Please renew it as soon as possible"
  120  PRINT "so you don't miss any important news."
  130  END
```

B

Using Tabs

The following escape code sequences are used to set and clear tabs –

␛1	Sets a tab at the column of the cursor
␛2	Clears a tab at the column of the cursor
␛3	Clears all tabs

The cursor can be moved to the next tab setting using the control code ␉ (horizontal tab) which can also be accessed using `CHR$(9)`. If no tabs are set a TAB moves the cursor to the beginning of the next line.

Clearing, Inserting and Deleting Lines

The following escape code sequences can be used in editing lines –

␛J	Clears the screen from the cursor position (remainder of the line and all lines following)
␛K	Clears the remainder of the line from the cursor position
␛L	Inserts a blank line before the cursor line. Cursor remains on same line of CRT and all following lines move down
␛M	Deletes the cursor line and closes up the gap. Cursor remains on same line of CRT, and all following lines roll up

These escape code sequences are very useful for text processing applications.

Inserting and Deleting Characters

The following escape code sequences can be used for inserting and deleting characters –

␛P	Deletes the character at the cursor position
␛Q	Turns on the insert character mode. Characters can be inserted to the left of the cursor
␛R	Turns off the insert character mode

Example

```
  10   PRINT PAGE
  20   PRINT "THIS IS THE OLD TEXT"
  30   WAIT 2000
● 40   PRINT CHR$(27)&"&a1r12C";RPT$(CHR$(27)&"P",3);
● 50   PRINT CHR$(27)&"QNEW AND IMPROVED";CHR$(27)&"R"
  60   END
```

Rolling the Display

The display in the printout area of the CRT can be rolled using the following escape code sequences –

␛S	Rolls the printout up one line (like [↑])
␛T	Rolls the printout down one line (like [↓])
␛U	Rolls the printout area up 20 lines (next page)
␛V	Rolls the printout area down 20 lines (previous page)

When using the escape code sequence with S and T to roll the printout, the cursor stays in the same line of the CRT. Using [↑] and [↓] moves the cursor also. When using the escape code sequence with U and V, the cursor is positioned to the upper left hand corner of the CRT. The printout can only be rolled as far as the lines in memory; it can't be rolled past the existing lines to unused lines. You can't roll all existing lines off the screen.

These escape code sequences are useful for accessing a line that is not currently displayed, then moving the cursor in that line.

B

Selective Scrolling (Memory Lock)

Through use of an escape code sequence, it is possible to "freeze" a selected number of the upper lines of the CRT in place. [↑] and [↓] and escape code sequences used for rolling the display then have no effect on these lines. This can be done with the following escape code sequence –

␛l (lowercase L)	Freezes all lines which are above the cursor line.

The remaining bottom lines can scroll up or down without moving the frozen lines. However, absolute row addressing is disabled when memory lock is on. Output of a formfeed character won't clear the frozen lines. ␛H positions the cursor to the first unfrozen line.

When memory lock is on, the cursor can't be positioned using R to address a row of memory. Y must be used to address a line of the printout area. When the printout is rolled using the U and V escape code sequences, the cursor is positioned to the first unlocked line.

The frozen lines can be unfrozen with [SHIFT] [CLEAR LINE], ␛E, or by using the following escape code sequence –

␛m	Unfreezes the lines which were frozen previously.

Example

```
10    PRINT "X","SIN X","COS X","TAN X"
20    PRINT "_","_____","_____","_____"
• 30  PRINT CHR$(27)&"1"
40    FOR X=0 TO 2*PI STEP .1
50       PRINT X,SIN(X),COS(X),TAN(X)
60    NEXT X
70    END
```

Disabling Control Codes

All control codes can be disabled (their action won't be performed) and viewed using the following escape code sequence –

␛Y

The only control code which is then recognized is CR (carriage return). When one is encountered, ␍ is printed and a carriage return-linefeed is executed. To see how this works, output ␛Y, then list the program in the Considerations section.

The control codes are re-activated using SHIFT CLEAR LINE or the following escape code sequence –

␛Z

All control features are cleared, the display reset, and CRT memory cleared using SHIFT CLEAR LINE or the following escape code sequence –

␛E

Summary of Escape Codes

Escape Code Sequence	Action
ESC A	Moves cursor up one row
ESC B	Moves cursor down one row
ESC C	Moves cursor right one column
ESC D	Moves cursor left one column
ESC E	Resets the CRT – clears control features

Escape Code Sequence	Action
ESC F	Moves cursor to row after last row of CRT memory, first column
ESC H	Moves cursor to first row of CRT memory, first column
ESC J	Clears screen from cursor (rest of line and all lines following)
ESC K	Clears line from cursor position
ESC L	Inserts a blank line before cursor line
ESC M	Deletes cursor line and closes up gap
ESC P	Deletes character at cursor position
ESC Q	Turns on insert character mode; inserts to left of cursor
ESC R	Turns off insert character mode
ESC S	Rolls printout up one line (like [↑])
ESC T	Rolls printout down one line (like [↓])
ESC U	Rolls printout up 20 lines (next page)
ESC V	Rolls printout down 20 lines (previous page)
ESC Y	Disables control codes and allows them to be viewed
ESC Z	Reactivates control codes
ESC l (lowercase L)	Freezes all lines above cursor line
ESC m	Unfreezes the lines which were frozen previously
ESC 1	Sets a tab at column of the cursor
ESC 2	Clears a tab at column of the cursor
ESC 3	Clears all tabs
ESC &a	Addresses the cursor
ESC &d	Accesses CRT special features

B

Examples

The first example listed is used to move blocks of text. The second example must be run to see how it manipulates a table. This example has been included on the System 35 test tape, under the name 'CRTADR'. To use this program, execute –

```
GET "CRTADR:T15"
```

Example 1

```
10   !   This program uses CRT addressing to move blocks of text
20   !   This section outputs the text     ***********************
30   PRINTER IS 16                         ! sets CRT as printer
40   PRINT PAGE;                           ! clears the CRT
50   PRINT "3.  This is the third paragraph."
60   PRINT "    It should be the last one"
70   PRINT "    in the group."
80   PRINT "2.  This is the second para-"
90   PRINT "    graph.  It should be the"
100  PRINT "    second one."
110  PRINT "1.  This is the first paragraph"
120  PRINT "    in the group.  It should"
130  PRINT "    be first."
140  PRINT
150  DISP "PRESS CONTINUE TO START ORDERING"
160  PAUSE
170  !  This section puts paragraphs in right order  ************
180  Ec$=CHR$(27)                          ! escape code
190  E$=CHR$(27)&"&a"                      ! escape code & '&a'
200 I1:   IMAGE #,K                        ! IMAGE for PRINT USING
210  PRINT USING I1;E$&"3Y"                ! move cursor to 1st line
                                             of paragraph 2
220  PRINT USING I1;Ec$&"l"                ! turn on memory lock
230  PRINT USING I1;RPT$(Ec$&"S",6)        ! roll remaining lines
                                             off screen
240  PRINT USING I1;Ec$&"m"                ! turn off memory lock
250  PRINT USING I1;Ec$&"H"                ! move cursor to 1st row,
                                             1st column
260  DISP "PRESS CONTINUE TO GO ON"
270  PAUSE
280  DISP                                  ! clear display
290  PRINT USING I1;E$&"3R"                ! move cursor to paragraph 1
300  PRINT USING I1;Ec$&"l"                ! turn on memory lock
310  PRINT USING I1;RPT$(Ec$&"S",3)        ! roll til cursor is in
                                             paragraph 3
320  PRINT USING I1;Ec$&"m"                ! turn off memory lock
330  PRINT USING I1;Ec$&"H"                ! move cursor to 1st row,
                                             1st column
340  END
```

Example 2

```
10   !   This program uses various CRT addressing operations to
20   !   create and manipulate a table.  Each line of the table
30   !   is numbered to reflect the line number of CRT memory.
40   STANDARD                              ! for number output
50   PRINTER IS 16                         ! CRT is printer
60   OPTION BASE 1
70   DIM Expenses(17,4),Accounts$(17) ! arrays for types and $'s
80   DATA  25,40,22.38,75,100,205.75,82,172.30,20,7.50,75.36,49
90   DATA  2000,827,40,1537,7,0,50.75,41,5000,4700,5130,4900,50,20
100  DATA  75,125,400,700,0,95,276,347,172.50,99.30,52.13,41.26
110  DATA  13,25.52,20,40,0,55,10,0,15,12,62,17.32,36.21,72.47
120  DATA  30.75,69.85,22,50,237,845,99,49,5,15,75,0,40,99,1275,83
130  DATA  Travel,Motels,Off. supplies,Machines,Entertainment,Pay
140  DATA  Overtime,Desks,Printing,Postage,Accts Payable,Donations
150  DATA  Miscellaneous,Rental cars,Advertising,Fees,Tooling
160  MAT READ Expenses,Accounts$           ! get values into arrays
170  Ec$=CHR$(27)                          ! escape code
180  E$=CHR$(27)&"&a"                      ! escape code & '&a'
190  Ts$=Ec$&"1"                           ! set tab at cursor position
200  T$=CHR$(9)                            ! horizontal tab
210  Del$=Ec$&"M"                          ! delete cursor line
220  Insl$=Ec$&"L"                         ! insert blank line above
                                             cursor line
230  Rol$=Ec$&"V"                          ! roll printout down
240 I1:    IMAGE #,K                       ! image for PRINT USING
250 Heading:  ! THIS SECTION OUTPUTS THE HEADING  ****************
260  PRINT " 0  ";RPT$("*",76),LIN(1)," 1",LIN(1)," 2"
270  PRINT " 3      Expense";TAB(25);"January";TAB(40);"February";
280  PRINT TAB(55);"March";TAB(70);"April"
290  PRINT " 4",LIN(1)," 5  ";RPT$("*",76)
300  L$=CHR$(124)                          ! vertical bar for spacing
310  PRINT " 6";TAB(20);L$;TAB(35);L$;TAB(50);L$;TAB(65);
320  PRINT L$;TAB(80);L$
330  PRINT E$&"7R"&Ec$&"l"                 ! put memory lock on heading
340 Table:  ! THIS SECTION OUTPUTS THE TABLE  ********************
350  PRINT USING I1;E$&"24C"&Ts$           ! set tab at cursor position
360  PRINT USING I1;E$&"39C"&Ts$
370  PRINT USING I1;E$&"54C"&Ts$
380  PRINT USING I1;E$&"69C"&Ts$
390  PRINT USING I1;E$&"7y0C"              ! reposition cursor to row 7
400  Line=7
410  FOR I=1 TO 17                         ! these loops print table
420     PRINT USING "#,DD,K";Line,"    "&Accounts$(I)
430     FOR J=1 TO 4                       ! output figures
440        PRINT T$;Expenses(I,J);         ! T$ tabs to next setting
450     NEXT J
460     PRINT                              ! go to next line
470     Line=Line+1                        ! output line number of CRT
480  NEXT I
```

```
490 Change:  ! THIS SECTION LETS NUMBERS BE CHANGED  ************
500  INPUT "Are there any figures you wish to change(Y OR N)?",A$
510  IF A$="N" THEN Reposition        ! branch to next section
520  INPUT "Enter row number of expense you want to change",Row
530  R$=VAL$(Row)
540  INPUT "Which month do you want to change(J,F,M or A)?",Month$
550  IF Month$="J" THEN Rep=1         ! Rep is for tabbing
560  IF Month$="F" THEN Rep=2
570  IF Month$="M" THEN Rep=3
580  IF Month$="A" THEN Rep=4
590  INPUT "Enter the new figure(<=9 digits)",Expenses(Row-6,Rep)
600  X=Expenses(Row-6,Rep)
610  PRINT Rol$                       ! roll printout down
620  IF Row>19 THEN Roll
630  PRINT USING I1;E$&R$&"y0C"       ! move cursor to right line
640  PRINT RPT$(T$,Rep);X;SPA(5)      ! tab to proper column,
                                        print number
650  INPUT "Are there more figures you want to change(Y or N)?",A$
660  IF A$="N" THEN Reposition        ! go to next section
670  GOTO 520                         ! start of this section
680  STOP
690 Roll:   !  THIS SECTION COMPENSATES FOR ROWS >19  ***********
700  PRINT USING I1;RPT$(Ec$&"S",Row-19)  ! position row in row 19
710  PRINT USING I1;E$&"19Y"          ! position cursor in row 19
720  PRINT RPT$(T$,Rep);X;SPA(5)      ! tab to proper column,
                                        print number
730  GOTO 650
740  STOP
750 Reposition:  ! THIS SECTION LETS A LINE BE MOVED   **********
760  INPUT "Which line do you want to move?",L1
770  INPUT "Which line do you want it to go above?",L2
780  L1$=VAL$(L1)
790  L2$=VAL$(L2)
800  PRINT USING I1;Rol$              ! roll printout down
810  IF (L1>19) OR (L2>19) THEN Roll2
820  IF L1<L2 THEN L2$=VAL$(L2-1)     ! make up for deletion of L1
830  PRINT USING I1;E$&L1$&"Y"        ! move cursor to line moving
840  PRINT USING I1;Del$              ! delete line being moved
850  PRINT USING I1;E$&L2$&"Y"        ! move cursor to L2
860  PRINT USING I1;Insl$             ! insert blank line above L2
870  PRINT USING "#,DD,K";L1,"    "&Accounts$(L1-6) !line# & acc't
880  PRINT RPT$(" ",13-LEN(Accounts$(L1-6)));   ! for shorter names
890  FOR I=1 TO 4                     ! reprint the deleted line
900     PRINT T$;Expenses(L1-6,I);
910  NEXT I
920  DISP "PRESS CONTINUE TO RENUMBER THE LINES"
930  PAUSE
940  PRINT USING I1;Rol$              ! roll down
950  PRINT USING I1;E$&"7y0C"         ! position cursor to line 7
960  FOR I=7 TO 23                    ! renumber all lines from 7
970     PRINT USING "DD";I
980  NEXT I
990  DISP                             ! clears display
1000 STOP                             ! PROGRAM STOPS HERE ******
```

```
1010 Roll2:    ! EITHER OR BOTH LINES > 19   ********************
1020 IF (L1>19) AND (L2<=19) THEN R1
1030 IF (L1<=19) AND (L2>19) THEN R2
1040                                   ! both L1 and L2 > 19
1050 PRINT USING I1;RPT$(Ec$&"S",4)    ! roll last line to line 19
1060 L1$=VAL$(L1-4)                    ! make up for rolling
1070 L2$=VAL$(L2-4)
1080 IF L1<L2 THEN L2$=VAL$(L2-5)      ! make up for deleted line
1090 GOTO 830
1100 R1:    ! L1>19 AND L2<=19     ******************************
1110 PRINT RPT$(Ec$&"S",L1-19)         ! roll line being moved to 19
1120 PRINT USING I1;E$&"19y0C"         ! move cursor to that line
1130 PRINT USING I1;Del$               ! delete line being moved
1140 PRINT USING I1;Rol$               ! roll printout down
1150 GOTO 850
1160 R2:    ! L1<=19 AND L2>19    *******************************
1170 PRINT USING I1;E$&L1$&"Y"         ! put cursor in line moving
1180 PRINT USING I1;Del$               ! delete line being moved
1190 PRINT USING I1;RPT$(Ec$&"S",3)    ! roll last line to line 19
1200 L2$=VAL$(L2-4)                    ! compensates for rolling
1210 GOTO 850
1220 END
```

B

B

Appendix C
Foreign Characters

You can easily access various foreign characters using the CHR$ function. The characters and their corresponding decimal value for the CHR$ function are listed below.

Character	CHR$ Value
¨ (umlaut)	171
° (degree sign)	179
ç	181
Ñ	182
ñ	183
¡	184
¿	185
â	192
ê	193
ô	194
û	195
á	196
é	197
ó	198
ú	199
à	200

Character	CHR$ Value
è	201
ù	203
ä	204
ë	205
ö	206
ü	207
Å	208
î	209
å	212
í	213
Ä	216
Ö	218
Ü	219
ï	221
β	222

Appendix D
Glossary

BASIC Syntax Guidelines

[ ] – all items enclosed in brackets are optional unless the brackets are in dot matrix.

dot matrix – all items in dot matrix must appear as shown.

... – three dots indicate that the previous item can be duplicated.

| – a vertical line between two parameters means "or"; only one of the two parameters can be included.

/ – a slash between two parameters means that either or both parameters can be included.

Terms

Calculator mode – No program is running and the computer is awaiting inputs or calculating keyboard entries.

Calling program – When a subprogram is being executed, the program segment (main program or subprogram) which called the subprogram is known as the calling program. Control returns to the calling program when the subprogram is completed.

Character – A letter, number, symbol or ASCII control code; any arbitrary 8-bit byte defined by the CHR$ function.

Command – An instruction to the computer which is executed from the keyboard. Commands are executed immediately, do not have line numbers and can't be used in a program. They are used to manipulate programs and for utility purposes, such as listing key definitions.

Constant – A fixed numeric value within the range of the 9835A/B; for example 29.5 or 2E12.

Controller address – An integer from zero through seven which specifies the address of a hard disc controller. Zero is the default address.

Current environment – The program segment which is being executed.

Defined record – The smallest unit of storage on a mass storage medium which is directly addressable. A defined record is established using the CREATE statement and can be specified as having any number of bytes in the range 4 through 32 767 (rounded up to an even number).

Display line – Line 22 of the CRT is used to display output generated by DISP, and any INPUT prompt or question mark.

Edit key mode – A Special Function Key is being defined as a typing aid. See EDIT KEY, which is discussed near the end of Chapter 2.

Edit line mode – The program in memory is being edited. See EDIT LINE, Chapter 12.

Files – The basic unit into which programs and data are stored. Storage of all files is "file-by-name" oriented; that is, each file must be assigned a unique **name**.

File name – A one to six character string expression with the exception of a colon, quote mark, ASCII NULL, or CHR$ (255). Blanks are ignored.

File number – The number assigned to a mass storage data file by an ASSIGN statement. Its range is one through ten.

File specifier – A string expression of the form: file name [mass storage unit specifier]

Formal parameter – Used to define subprogram variables and can be a non-subscripted variable, array identifier or file specified by # file number. A type word can come before parameters to specify numeric type. Parameters must be separated by commas; the parameter list must be enclosed in parentheses.

HP-IB device address – An expression which specifies the HP-IB address that is set on a device. Its range is 0 through 30.

Interleave factor – Defines the number of revolutions per track to be made for a complete data transfer on a 9885 Disk. It is specified in an INITIALIZE statement.

Keyboard entry area – Lines 23 and 24 of the CRT are accessible only through keyboard inputs. Every line that is typed in is displayed in this area. The first position in line 23 is known as the "home" position of the cursor. As the 148th character is keyed in, a beep indicates that only 12 more characters can be entered.

Label – A unique **name** given to a program line. It follows the line number and is followed by a colon.

Line identifier – A program line can be identified either by its line number (GOTO 150) or its label, it any (GOTO Routine).

Line number – An integer from 1 through 9999. In most cases, when a line number is specified, but is not in memory, the next highest line is accessed.

Live keyboard mode – Numeric computations and most statements and commands can be executed from the keyboard while a program is running. Program lines can be stored also. The running program is temporarily paused while a keyboard operation is executing.

Local variable – A variable in a subprogram that isn't declared in the formal parameter list or COM statement; it can't be accessed from any other program segment. Storage of local variables is temporary and returned to user Read/Write Memory upon return to the calling program.

Logical record – A user-level rather than machine concept; a collection of data items which are conceptually grouped together for mass storage operation.

Main program – The central part of a program from which subprograms can be called is known as the main program. When you press (RUN), you access the main program. The main program can't be called by a subprogram.

D

Mass storage unit specifier – Any string expression of the form –

: device type [select code [, controller address | 9885 unit code [, unit code]]]

The letters specifying the various mass storage device types are –

Letter	Device
T	Tape cartridge
F	9885 Flexible Disk
Y	7905A Removable Platter
Z	7905A Fixed Platter
C	7906A Removable Platter
D	7906A Fixed Platter
P	7920A Disc Pack

Mass storage unit specifier is appreviated msus.

Msus – The abbreviation for mass storage unit specifier.

Name – A capital letter followed by 0 through 14 lower case letters, digits or the underscore character. Names are used for variable names, labels, function names, and subprogram names.

Numeric expression – A logical combination of variables, constants, operators, functions, including user-defined functions, grouped with parentheses if needed.

D

Pass parameter – Used in calling a subprogram to pass a value and can be a variable, array identifier, expression or file specified by #file number; any variable can be enclosed in parentheses causing it to be passed by value.

Physical record – A 256-byte, fixed unit which is established when a mass storage medium is initialized. Every **file** starts at the beginning of a physical record; this is an important fact for optimum device use. Otherwise, you need not be concerned with physical records.

Printout area – Lines 1 through 20 of the CRT are similar to a printing device. When the machine is switched on, this area is the standard system printer to which output from PRINT, PRINT USING, CRT and LIST is directed. It is also, at power-on, the print all printer when in the print mode; see Chapter 2.

Priority – A number in the range 1 through 15 which determines whether or not an interrupt is serviced. The priority of the interrupt must be higher than current system priority to be serviced.

Program mode – A program is running.

Program segment – The main program and each subprogram are known as program segments. Every program segment is independent of every other program segment. Subprograms come after the main program; that is, they are higher numbered. Subprograms are called by the main program or another subprogram. See Appendix F for the relationship between memory allocation and subprograms.

Protect code – Any valid string expression except one with a length of zero. Only the first six characters are recognized as the protect code, however.

Redim subscripts – Numeric expressions separated by commas and enclosed in parentheses.

Read Only Memory (ROM) – Permanent memory which can't be changed or erased. Option ROMs are used to expand the language and capabilities of the computer.

Read/Write Memory (RWM) – Used to store programs, data and related information. The information in Read/Write Memory can be changed and is lost when the computer is shut off.

Scalar – A numeric expression used as a constant in mathematical operations.

D

Select code – An expression (rounded to an integer) in the range zero through sixteen. The following select codes are reserved by the system and can't be set on an interface –

- 0 Optional Internal Thermal Printer
- 15 Tape Drive
- 16 CRT (9835A); Internal printer (9835B)

Special Function Keys (SFK's) – These keys can be defined or redefined for use as typing aids for statements, variable names or other series of keystrokes which are used often. Many of them have pre-defined definitions. Any of the special function keys can also be defined to have program interrupt capability (see Chapter 8 for more information).

Standard mass storage device – The device to which all mass storage operations are directed if no device is specified. It is the tape cartridge at power on and can be changed using the MASS STORAGE IS statement.

Standard printer – The printer to which all PRINT, PRINT USING, CRT and LIST output is directed if no device is specified. At power on, it is the CRT (9835A) or strip printer (9835B); it can be changed using the PRINTER IS statement.

Statement – An instruction to the computer telling it what to do while a program is running. A statement can be preceded by a line number, stored and executed from a program. Most statements can also be executed from the keyboard without a line number.

Subscript – An integer used to specify the range of an array dimension. One subscript is used to specify the upper bound of a dimension; two subscripts separated by a colon are used to specify the upper and lower bounds of a dimension. A comma is used to separate the subscripts for each dimension.

System comments line – Line 25 of the CRT is reserved for error messages, mode indicators, and the run light: ▒. Results of keyboard operations such as 3+5 EXEC or X EXEC also appear in this line.

Text – Any combination of characters; for example "ABC". Text can be quoted (literal) or unquoted.

Unit code – The address set on a hard disc drive; it can be an integer from zero through seven. Zero is the default code. It is ignored for the 9885 and tape cartridge.

The 9885 unit code is the address set on a 9885 disk drive; it can be an integer from zero through three. Zero is the default code.

Variable – A name which is assigned a value and specifies a location in memory. Variables can be classified into various categories and subsets of the categories as shown in the diagram below. For example, any reference to a single numeric variable includes a simple numerics and elements of numeric arrays.

- variable
  - single variable
    - numeric variable
      - simple numeric
      - array element
    - string variable
      - simple string
        - substring
      - string array element
        - substring
  - array variable
    - numeric array
      - matrix
      - vector
    - string array
      - matrix
      - vector

Appendix E
Reference Tables

Reset Conditions

The following table shows the status of various conditions when the indicated operations are performed.

	SCRATCHA or Power On (Value)	Reset	SCRATCH	RUN	CONT
Variables	R (none)	–	R	R	–
RESult	R (0)	–	–	–	–
Subroutine return pointers	R (none)	R	R	R	–
Angular units	R (RAD)	R	R	R	–
Numeric output mode	R (STANDARD)	R	R	–	–
Random number seed	R (π/180)	R	R	R	–
Standard printer	R (select code 16)	–	–	–	–
Printall printer	R (select code 16)	–	–	–	–
Standard mass storage device	R (:T15)	–	–	–	–
SFK definitions	R (Initial)	–	–	–	–
Processing mode	R (SERIAL)	–	R	–	–
Live keyboard mode	R (INTERACTIVE)	R	–	–	–
Binary routines	R (none)	–	–	–	–
Files table	R (none)	R	R	R	–
DATA pointers	R (none)	R	R	R	–
ERRL, ERRN	R (0,0)	R	R	R	–

– means unchanged

R means restored to power on values

ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec
NULL	00000000	000	0	space	00100000	040	32	@	01000000	100	64	`	01100000	140	96
SOH	00000001	001	1	!	00100001	041	33	A	01000001	101	65	a	01100001	141	97
STX	00000010	002	2	"	00100010	042	34	B	01000010	102	66	b	01100010	142	98
ETX	00000011	003	3	#	00100011	043	35	C	01000011	103	67	c	01100011	143	99
EOT	00000100	004	4	$	00100100	044	36	D	01000100	104	68	d	01100100	144	100
ENQ	00000101	005	5	%	00100101	045	37	E	01000101	105	69	e	01100101	145	101
ACK	00000110	006	6	&	00100110	046	38	F	01000110	106	70	f	01100110	146	102
BELL	00000111	007	7	'	00100111	047	39	G	01000111	107	71	g	01100111	147	103
BS	00001000	010	8	(	00101000	050	40	H	01001000	110	72	h	01101000	150	104
HT	00001001	011	9	)	00101001	051	41	I	01001001	111	73	i	01101001	151	105
LF	00001010	012	10	*	00101010	052	42	J	01001010	112	74	j	01101010	152	106
VTAB	00001011	013	11	+	00101011	053	43	K	01001011	113	75	k	01101011	153	107
FF	00001100	014	12	,	00101100	054	44	L	01001100	114	76	l	01101100	154	108
CR	00001101	015	13	–	00101101	055	45	M	01001101	115	77	m	01101101	155	109
SO	00001110	016	14	.	00101110	056	46	N	01001110	116	78	n	01101110	156	110
SI	00001111	017	15	/	00101111	057	47	O	01001111	117	79	o	01101111	157	111
DLE	00010000	020	16	Ø	00110000	060	48	P	01010000	120	80	p	01110000	160	112
DC_1	00010001	021	17	1	00110001	061	49	Q	01010001	121	81	q	01110001	161	113
DC_2	00010010	022	18	2	00110010	062	50	R	01010010	122	82	r	01110010	162	114
DC_3	00010011	023	19	3	00110011	063	51	S	01010011	123	83	s	01110011	163	115
DC_4	00010100	024	20	4	00110100	064	52	T	01010100	124	84	t	01110100	164	116
NAK	00010101	025	21	5	00110101	065	53	U	01010101	125	85	u	01110101	165	117
SYNC	00010110	026	22	6	00110110	066	54	V	01010110	126	86	v	01110110	166	118
ETB	00010111	027	23	7	00110111	067	55	W	01010111	127	87	w	01110111	167	119
CAN	00011000	030	24	8	00111000	070	56	X	01011000	130	88	x	01111000	170	120
EM	00011001	031	25	9	00111001	071	57	Y	01011001	131	89	y	01111001	171	121
SUB	00011010	032	26	:	00111010	072	58	Z	01011010	132	90	z	01111010	172	122
ESC	00011011	033	27	;	00111011	073	59	[	01011011	133	91	{	01111011	173	123
FS	00011100	034	28	<	00111100	074	60	\	01011100	134	92	|	01111100	174	124
GS	00011101	035	29	=	00111101	075	61	]	01011101	135	93	}	01111101	175	125
RS	00011110	036	30	>	00111110	076	62	^	01011110	136	94	~	01111110	176	126
US	00011111	037	31	?	00111111	077	63	_	01011111	137	95	DEL	01111111	177	127

E

Metric Conversion Table

Linear Measure

		1 millimetre	=	0.03937 inch
10 millimetres	=	1 centimetre	=	0.3937 inch
10 centimetres	=	1 decimetre	=	3.937 inches
10 decimetres	=	1 metre	=	39.37 inches or 3.2808 feet or 0.1988 rod
10 metres	=	1 decametre	=	393.7 inches
10 decametres	=	1 hectometre	=	328.08 feet
10 hectometres	=	1 kilometre	=	0.621 mile or 3 280.8 feet
10 kilometres	=	1 myriametre	=	6.21 miles

Square Measure

		1 square millimetre	=	0.00155 square inch
100 square millimetres	=	1 square centimetre	=	0.15499 square inch
100 square centimetres	=	1 square decimetre	=	15.499 square inches
100 square decimetres	=	1 square metre	=	1 549.9 square inches or 1.196 square yards
100 square metres	=	1 square decametre	=	119.6 square yards
100 square decametres	=	1 square hectometre	=	2.471 acres
100 square hectometres	=	1 square kilometre	=	0.386 square mile or 247.1 acres

Weights

10 milligrams	=	1 centigram	=	0.1543 grain or 0.000353 ounce (avdp.)
10 centigrams	=	1 decigram	=	1.5432 grains
10 decigrams	=	1 gram	=	15.432 grains or 0.035274 ounce (avdp.)
10 grams	=	1 decagram	=	0.3527 ounce
10 decagrams	=	1 hectogram	=	3.5274 ounces
10 hectograms	=	1 kilogram	=	2.2046 pounds
10 kilograms	=	1 myriagram	=	22.046 pounds
10 myriagrams	=	1 quintal	=	220.46 pounds
10 quintals	=	1 metric ton	=	2 204.6 pounds

Land Measure

1 square metre	=	1 centiare	=	1 549.9 square inches
100 centiares	=	1 are	=	119.6 square yards
100 ares	=	1 hectare	=	2.471 acres
100 hectares	=	1 square kilometre	=	0.386 square mile or 247.1 acres

Volume Measure

1 000 cubic millimetres	=	1 cubic centimetre	=	0.06102 cubic inch
1 000 cubic centimetres	=	1 cubic decimetre	=	61.023 cubic inches or 0.0353 cubic foot
1 000 cubic decimetres	=	1 cubic metre	=	35.314 cubic feet or 1.308 cubic yards

Capacity Measure

10 millilitres	=	1 centilitre	=	0.338 fluid ounce
10 centilitres	=	1 decilitre	=	3.38 fluid ounces or 0.1057 liquid quart
10 decilitres	=	1 litre	=	1.0567 liquid quarts or 0.9081 dry quart
10 litres	=	1 decalitre	=	2.64 gallons or 0.284 bushel
10 decalitres	=	1 hectolitre	=	26.418 gallons or 2.838 bushels
10 hectolitres	=	1 kilolitre	=	264.18 gallons or 35.315 cubic feet

HEWLETT PACKARD

SALES & SERVICE OFFICES

AFRICA, ASIA, AUSTRALIA

ANGOLA
Telectra
Empresa Técnica de
Equipamentos
Eléctricos, S A R L
R Barbosa Rodrigues, 42-I DT
Caixa Postal, 6487
Luanda
Tel 35515/6
Cable TELECTRA Luanda

AUSTRALIA
Hewlett-Packard Australia
Pty Ltd
31-41 Joseph Street
Blackburn, Victoria 3130
P O Box 36
Doncaster East Victoria 3109
Tel 89-6351
Telex 31-024
Cable HEWPARD Melbourne

Hewlett-Packard Australia
Pty Ltd
31 Bridge Street
Pymble
New South Wales, 2073
Tel 449-6566
Telex 21561
Cable HEWPARD Sydney

Hewlett-Packard Australia
Pty Ltd
153 Greenhill Road
Parkside, S A 5063
Tel 272-5911
Telex 82536 ADEL
Cable HEWPARD ADELAID

Hewlett-Packard Australia
Pty Ltd
141 Stirling Highway
Nedlands, W A 6009
Tel 86-5455
Telex 93859 PERTH
Cable HEWPARD PERTH

Hewlett-Packard Australia
Pty Ltd
121 Wollongong Street
Fyshwick, A C T 2609
Tel 95-2733
Telex 62650 Canberra
Cable HEWPARD CANBERRA

Hewlett Packard Australia
Pty Ltd
5th Floor
Teachers Union Building
495-499 Boundary Street
Spring Hill 4000 Queensland
Tel 229-1544
Cable HEWPARD Brisbane

GUAM
Medical/Pocket Calculators Only
Guam Medical Supply, Inc
Jay Ease Building, Room 210
P O Box 8947
Tamuning 96911
Tel 646-4513
Cable EARMED Guam

HONG KONG
Schmidt & Co (Hong Kong) Ltd
P O Box 297
Connaught Centre
39th Floor
Connaught Road, Central
Hong Kong
Tel H-255291-5
Telex 74766 SCHMC HX
Cable SCHMIDTCO Hong Kong

INDIA
Blue Star Ltd
Kasturi Buildings
Jamshedji Tata Rd
Bombay 400 020
Tel 29 50 21
Telex 001-2156
Cable BLUEFROST

Blue Star Ltd
Sahas
414 2 Vir Savarkar Marg
Prabhadevi
Bombay 400 025
Tel 45 78 87
Telex 011-4093
Cable FROSTBLUE

Blue Star Ltd
Band Box House
Prabhadevi
Bombay 400 025
Tel 45 73 01
Telex 011-3751
Cable BLUESTAR

Blue Star Ltd
7 Hare Street
P O Box 506
Calcutta 700 001
Tel 23-0131
Telex 021-7655
Cable BLUESTAR

Blue Star Ltd
7th & 8th Floor
Bhandari House
91 Nehru Place
New Delhi 110024
Tel 634770 & 635166
Telex 031-2463
Cable BLUESTAR

Blue Star Ltd
Blue Star House
11 11A Magarath Road
Bangalore 560 025
Tel 55668
Telex 043-430
Cable BLUESTAR

Blue Star Ltd
Meeakshi Mandiran
xxx 1678 Mahatma Gandhi Rd
Cochin 682 016
Tel 32069,32161,32282
Telex 0885-514
Cable BLUESTAR

Blue Star Ltd
1-1-117 1
Sarojini Devi Road
Secunderabad 500 003
Tel 70126, 70127
Cable BLUEFROST
Telex 015-459

Blue Star Ltd
2 34 Kodambakkam High Road
Madras 600034
Tel 82056
Telex 041-379
Cable BLUESTAR

INDONESIA
BERCA Indonesia P T
P O Box 496 Jkt
JLN.Abdul Muis 62
Jakarta
Tel 40369, 49886,49255,356038
JKT 42895
Cable BERCACON

BERCA Indonesia P T
63 JL Raya Gubeng
Surabaya
Tel 44309

ISRAEL
Electronics & Engineering Div
of Motorola Israel Ltd
17 Kremenetski Street
P O Box 25016
Tel-Aviv
Tel 38973
Telex 33569
Cable BASTEL Tel-Aviv

JAPAN
Yokogawa-Hewlett-Packard Ltd
Ohashi Building
59-1 Yoyogi 1-Chome
Shibuya-ku, **Tokyo** 151
Tel 03-370-2281/92
Telex 232-2024YHP MARKET
TOK 23-724
Cable YHPMARKET

Yokogawa-Hewlett-Packard Ltd
Chuo Bldg, 4th Floor
4-20, Nishinakajima 5-chome
Yodogawa-ku, Osaka-shi
Osaka 532
Tel 06-304-6021

Yokogawa-Hewlett-Packard Ltd
Nakamo Building
24 Kami Sasajima-cho
Nakamura-ku, **Nagoya** 450
Tel (052) 571-5171

Yokogawa-Hewlett-Packard Ltd
Tanigawa Building
2-24-1 Tsuruya-cho
Kanagawa-ku
Yokohama 221
Tel 045-312-1252
Telex 382-3204 YHP YOK

Yokogawa-Hewlett-Packard Ltd
Mito Mitsu Building
105, Chome-1 San-no-maru
Mito, Ibaragi 310
Tel 0292-25-7470

Yokogawa-Hewlett-Packard Ltd
Inoue Building
1348-3, Asahi-cho, 1-chome
Atsugi, Kanagawa 243
Tel 0462-24-0452

Yokogawa-Hewlett-Packard Ltd
Kumagaya Asahi
Hackijuni Building
4th Floor
3-4, Tsukuba
Kumagaya, Saitama 360
Tel 0485-24-6563

KENYA
Technical Engineering
Services(E A)Ltd
P O Box 18311
Nairobi
Tel 557726/556762
Cable PROTON

Medical Only
International Aeradio(E A)Ltd
P O Box 19012
Nairobi Airport
Nairobi
Tel 336055/56
Telex 22201/22301
Cable INTAERIO Nairobi

KOREA
Samsung Electronics Co , Ltd
20th Fl, Dongbang Bldg 250, 2-KA
C P O Box 2775
Taepyung-Ro, Chung-Ku
Seoul
Tel (23) 6811
Telex 22575
Cable ELEKSTAR Seoul

MALAYSIA
Teknik Mutu Sdn Bhd
2 Lorong 13/6A
Section 13
Petaling Jaya, **Selangor**
Tel 54994/54916
Telex MA 37605

Protel Engineering
P O Box 1917
Lot 259, Satok Road
Kuching, **Sarawak**
Tel 2400
Cable PROTEL ENG

MOZAMBIQUE
A N Goncalves, Lta
162, 1 Apt 14 Av D Luis
Caixa Postal 107
Lourenco Marques
Tel 27091, 27114
Telex 6-203 NEGON Mo
Cable NEGON

NEW ZEALAND
Hewlett-Packard (N Z) Ltd
P O Box 9443
Courtenay Place
Wellington
Tel 877-199
Cable HEWPACK Wellington

Hewlett-Packard (N Z) Ltd
Pakuranga Professional Centre
267 Pakuranga Highway
Box 51092
Pakuranga
Tel 569-651
Cable HEWPACK,Auckland

Analytical Medical Only
Medical Supplies N Z Ltd
Scientific Division
79 Carlton Gore Rd , Newmarket
P O Box 1234
Auckland
Tel 75-289
Cable DENTAL Auckland

Analytical Medical Only
Medical Supplies N Z Ltd
P O Box 1994
147-161 Tory St
Wellington
Tel 850-799
Telex 3858
Cable DENTAL, Wellington

Analytical Medical Only
Medical Supplies N Z Ltd
P O Box 309
239 Stanmore Road
Christchurch
Tel 892-019
Cable DENTAL, Christchurch

Analytical Medical Only
Medical Supplies N Z Ltd
303 Great King Street
P O Box 233
Dunedin
Tel 88-817
Cable DENTAL Dunedin

NIGERIA
The Electronics
Instrumentations Ltd
N6B 770 Oyo Road
Oluseun House
P M B 5402
Ibadan
Tel 61577
Telex 31231 TEIL Nigeria
Cable THETEIL Ibadan

The Electronics Instrumenta-
tions Ltd
144 Agege Motor Road, Mushin
P O Box 6645
Lagos
Cable THETEIL Lagos

PAKISTAN
Mushko & Company, Ltd
Oosman Chambers
Abdullah Haroon Road
Karachi-3
Tel 511027, 512927
Telex 2894
Cable COOPERATOR Karachi

Mushko & Company, Ltd
388, Satellite Town
Rawalpindi
Tel 41924
Cable FEMUS Rawalpindi

PHILIPPINES
The Online Advanced
Systems Corporation
Rico House
Amorsolo cor Herrera Str
Legaspi Village, Makati
Metro **Manila**
Tel 85-35-81, 85-34-91
Telex 3274 ONLINE

RHODESIA
Field Technical Sales
45 Kelvin Road North
P O Box 3458
Salisbury
Tel 705231 (5 lines)
Telex RH 4122

SINGAPORE
Hewlett-Packard Singapore
(Pte) Ltd
1150 Depot Road
Alexandra P O Box 58
Singapore 4
Tel 270-2355
Telex HPSG RS 21486
Cable HEWPACK, Singapore

SOUTH AFRICA
Hewlett-Packard South Africa
(Pty) Ltd
Private Bag Wendywood
Sandton, Transvaal 2144
Hewlett-Packard Centre
Daphne Street, Wendywood
Sandton, Transvaal 2144
Tel 802-10408
Telex 8-4782
Cable HEWPACK JOHANNESBURG

Service Department
Hewlett-Packard South Africa
(Pty) Ltd
P O Box 39325
Gramley, Sandton 2018
451 Wynberg Extension 3,
Sandton 2001
Tel 636-8188/9
Telex 8-2391

Hewlett-Packard South Africa
(Pty), Ltd
P O Box 120
Howard Place, Cape Province, 7450
Pine Park Centre, Forest Drive,
Pinelands, Cape Province, 7405
Tel 53-7955 thru 9
Telex 57-0006

Service Department
Hewlett-Packard South Africa
(Pty), Ltd
P O Box 37099
Overport, Durban 4067
Braby House
641 Ridge Road
Durban, 4001
Tel 88-7478
Telex 6-7954

TAIWAN
Hewlett-Packard Far East Ltd ,
Taiwan Branch
39 Chung Hsiao West Road
Sec 1, 7th Floor
Taipei
Tel 3819160-4
Cable HEWPACK TAIPEI

Hewlett-Packard Far East Ltd
Taiwan Branch
68-2, Chung Cheng 3rd Road
Kaohsiung
Tel (07) 242318-Kaohsiung

Analytical Only
San Kwang Instruments Co , Ltd ,
No 20, Yung Sui Road
Taipei
Tel 3715171-4 (5 lines)
Telex 22894 SANKWANG
Cable SANKWANG TAIPEI

TANZANIA
Medical Only
International Aeradio (E A), Ltd
P O Box 861
Dar es Salaam
Tel 21251 Ext 265
Telex 41030

THAILAND
UNIMESA Co , Ltd
Elcom Research Building
2538 Sukumvit Ave
Bangkok
Tel 3932387, 3930338
Cable UNIMESA Bangkok

UGANDA
Medical Only
International Aeradio(E A), Ltd ,
P O Box 2577
Kampala
Tel 54388
Cable INTAERIO Kampala

ZAMBIA
R J Tilbury (Zambia) Ltd
P O Box 2792
Lusaka
Tel 73793
Cable ARJAYTEE, Lusaka

OTHER AREAS NOT LISTED, CONTACT:
Hewlett-Packard Intercontinental
3200 Hillview Ave
Palo Alto, California 94304
Tel (415) 493-1501
TWX 910-373-1267
Cable HEWPACK Palo Alto

CANADA

ALBERTA
Hewlett-Packard (Canada) Ltd
11620A - 168th Street
EdmontonT5M 3T9
Tel (403) 452-3670
TWX 610-831-2431

Hewlett-Packard (Canada) Ltd
210, 7220 Fisher St S E
Calgary T2H 2H8
Tel (403) 253-2713
Twx 610-821-6141

BRITISH COLUMBIA
Hewlett-Packard (Canada) Ltd
837 E Cordova Street
Vancouver V6A 3R2
Tel (604) 254-0531
TWX 610-922-5059

MANITOBA
Hewlett-Packard (Canada) Ltd
513 Century St
St James
Winnipeg R3H OL8
Tel (204) 786-7581
TWX 610-671-3531

NOVA SCOTIA
Hewlett-Packard (Canada) Ltd
800 Windmill Road
Dartmouth B3B 1L1
Tel (902) 469-7820
TWX 610-271-4482 HFX

ONTARIO
Hewlett-Packard (Canada) Ltd
1020 Morrison Dr
Ottawa K2H 8K7
Tel (613) 820-6483
TWX 610-563-1636

Hewlett-Packard (Canada) Ltd
6877 Goreway Drive
Mississauga L4V 1M8
Tel (416) 678-9430
TWX 610-492-4246

QUEBEC
Hewlett-Packard (Canada) Ltd
275 Hymus Blvd
Pointe Claire H9R 1G7
Tel (514) 697-4232
TWX 610-422-3022
TLX 05-821521 HPCL

FOR CANADIAN AREAS NOT LISTED:
Contact Hewlett-Packard (Canada)
Ltd in Mississauga

CENTRAL AND SOUTH AMERICA

ARGENTINA
Hewlett-Packard Argentina
S A
Av Leandro N Alem 822 - 12
1001**Buenos Aires**
Tel 31-6063,4,5,6 and 7
Telex 122443 AR CIGY
Cable HEWPACK ARG

BOLIVIA
Casa Kavlin S A
Calle Potosi 1130
P O Box 500
La Paz
Tel 41530,53221
Telex CWC BX 5298,ITT 3560082
Cable KAVLIN

BRAZIL
Hewlett-Packard do Brasil
I e C Ltda
Avenida Rio Negro 980
Alphaville
06400**Barueri** SP
Tel 429-3222

Hewlett-Packard do Brasil
I e C Ltda
Rua Padre Chagas, 32
90000-**Pôrto Alegre**-RS
Tel (0512) 22-2998, 22-5621
Cable HEWPACK Potto Alegre

Hewlett-Packard do Brasil
I E C Ltda
Rua Siqueira Campos, 53
Copacabana
20000-**Rio de Janeiro**
Tel 257-80-94-DDD (021)
Telex 391-212-1905 HEWP-BR
Cable HEWPACK
Rio de Janeiro

CHILE
Calcagni y Metcalfe Ltda
Alameda 580-Of 807
Casilla 2118
Santiago, 1
Tel 398613
Telex 3520001 CALMET
Cable CALMET Santiago

COLOMBIA
Instrumentación
Henrik A Langebaek & Kier S A
Carrera 7 No 48-75
Apartado Aéreo 6287
Bogotá, I D E
Tel 69-88-77
Cable AARIS Bogotá
Telex 044-400

COSTA RICA
Científica Costarricense S A
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
San José
Tel 24-38-20, 24-08-19
Telex 2367 GALGUR CR
Cable GALGUR

ECUADOR
Calculators Only
Computadoras y Equipos
Electrónicos
P O Box 6423 CCI
Eloy Alfaro #1824 3 Piso
Quito
Tel 453482
Telex 02-2113 Sagita Ed
Cable Sagita-Quito

EL SALVADOR
Instrumentacion y Procesamiento
Electronico de el Salvador
Bulevar de los Heroes 11-48
San Salvador
Tel 252787

GUATEMALA
IPESA
Avenida La Reforma 3-48
Zona 9
Guatemala City
Tel 63627, 64786
Telex 4192 Teletro Gu

MEXICO
Hewlett-Packard Mexicana,
S A de C V
Av Periférico Sur No 6501
Tepepan, Xochimilco
Mexico 23, D F
Tel 905-676-4600

Hewlett-Packard Mexicana,
S A de C V
Ave Constitución No 2184
Monterrey, N L
Tel 48-71-32, 48-71-84
Telex 038-410

NICARAGUA
Roberto Terán G
Apartado Postal 689
Edificio Terán
Managua
Tel 25114, 23412,23454
Cable ROTERAN Managua

PANAMA
Electrónico Balboa, S A
P O Box 4929
Calle Samuel Lewis
Cuidad de Panama
Tel 64-2700
Telex 3483103 Curunda,
Canal Zone
Cable ELECTRON Panama

PERU
Compañía Electro Médica S A
Los Flamencos 145
San Isidro Casilla 1030
Lima 1
Tel 41-4325
Cable ELMED Lima

PUERTO RICO
Hewlett-Packard Inter-Americas
Puerto Rico Branch Office
Calle 272
No 203 Urb Country Club
Carolina 00924
Tel (809) 762-7255
Telex 345 0514

URUGUAY
Pablo Ferrando S A
Comercial e Industrial
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel 40-3102
Cable RADIUM Montevideo

VENEZUELA
Hewlett-Packard de Venezuela
C A
P O Box 50933
Caracas 105
Los Ruices Norte
3a Transversal
Edificio Segre
Caracas 107
Tel 35-00-11 (20 lines)
Telex 25146 HEWPACK
Cable HEWPACK Caracas

FOR AREAS NOT LISTED, CONTACT:
Hewlett-Packard
Inter-Americas
3200 Hillview Ave
Palo Alto, California 94304
Tel (415) 493-1501
TWX 910-373-1260
Cable HEWPACK Palo Alto
Telex 034-8300, 034-8493

EUROPE, NORTH AFRICA AND MIDDLE EAST

AUSTRIA
Hewlett-Packard Ges m b H
Handelskai 52
P O box 7
A-1205 **Vienna**
Tel (0222) 351621 to 27
cable HEWPAK Vienna
Telex 75923 hewpak a

BELGIUM
Hewlett-Packard Benelux
S A /N V
Avenue de Col-Vert 1
(Groenkraaglaan)
B-1170 **Brussels**
Tel (02) 672 22 40
Cable PALOBEN Brussels
Telex 23 494 paloben bru

CYPRUS
Kypronics
19 Gregorios & Xenopoulos Rd
P O Box 1152
CY-**Nicosia**
Tel 45628/29
Cable KYPRONICS PANDEHIS
Telex 3018

CZECHOSLOVAKIA
Vyvojova a Provozni Zakladna
Vyzkumnych Ustavu v Bechovicich
CSSR-25097 **Bechovice u Prahy**
Tel 89 93 41
Telex 121333

Institute of Medical Bionics
Vyskumny Ustav Lekarskej Bioniky
Jedlova 6
CS-88346 **Bratislava-Kramare**
Tel 44-551/45-541

DDR
Entwicklungslabor der TU Dresden
Forschungsinstitut Meinsberg
DDR-7305
Waldheim/Meinsberg
Tel 37 667
Telex 112145

Export Contact AG Zuerich
Guenther Forgber
Schlegelstrasse 15
1040 **Berlin**
Tel 42-74-12
Telex 111889

DENMARK
Hewlett-packard A/S
Datavej 52
DK-3460 **Birkerod**
Tel (02) 81 66 40
Cable HEWPACK AS
Telex 37409 hpas dk

Hewlett-Packard A/S
Navervej 1
DK-8600 **Silkeborg**
Tel (06) 82 71 66
Telex 37409 hpas dk
Cable HEWPACK AS

FINLAND
Hewlett-Packard OY
Nahkahousuntie 5
P O Box 6
SF-00211 **Helsinki** 21
Tel (90) 6923031
Cable HEWPACKOY Helsinki
Telex 12-1563 HEWPA SF

FRANCE
Hewlett-Packard France
Quartier de Courtaboeuf
Boite Postale No 6
F-91401 **Orsay** Cédex
Tel (1) 907 78 25
Cable HEWPACK Orsay
Telex 600048

Hewlett-Packard France
Agency Régionale
"Le Saquin"
Chemin des Mouilles
B.P 162
F-69130 **Ecully**
Tel (78) 33 81 25
Cable HEWPACK Eculy
Telex 31 06 17

Hewlett-Packard France
Agence Régionale
Péricentre de la Cépière
Chemin de la Cépière, 20
F-31300 **Toulouse-Le Mirail**
Tel (61) 40 11 12
Cable HEWPACK 51957
Telex 510957

Hewlett-Packard France
Agence Régionale
Aéroport principal de
Marseille-Marignane
F-13721 **Marignane**
Tel (91) 89 12 36
Cable HEWPACK MARGN
Telex 410770

Hewlett-Packard France
Agence Régionale
63, Avenue de Rochester
B.P 1124
F-35014 **Rennes** Cédex
Tel (99) 36 33 21
Cable HEWPACK 74912
Telex 740912

Hewlett-Packard France
Agence Régionale
74 Allée de la Robertsau
F-67000 **Strasbourg**
Tel (88) 35 23 20/21
Telex 890141
Cable HEWPACK STRBG

Hewlett-Packard France
Agence Régionale
Centre Vauban
201, rue Colbert
Entrée A2
F-59000 **Lille**
Tel (20) 51 44 14
Telex 820744

Hewlett-Packard France
Centre d Affaires Paris-Nord
Bâtiment Ampère
Rue de La Commune de Paris
B.P 300
F-93153 **Le Blanc Mesnil Cédex**
Tel (01) 931 88 50

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Bernerstrasse 117
Postfach 560 140
D-6000 **Frankfurt** 56
Tel (0611) 50 04-1
Cable HEWPACKSA Frankfurt
Telex 04 13249 hpffmd

Hewlett-Packard GmbH
Technisches Buero Böblingen
Herrenbergerstrasse 110
D-7030 **Böblingen**, Württemberg
Tel (07031) 667-1
Cable HEPAK Böblingen
Telex 07265739 bbn

Hewlett-Packard GmbH
Technisches Buero Dusseldorf
Emanuel-Leutze-Str 1 (Seestern)
D-4000 **Dusseldorf** 11
Tel (0211) 59711
Telex 085/86 533 hpdd d

Hewlett-Packard GmbH
Technisches Buero Hamburg
Wendenstrasse 23
D-2000 **Hamburg** 1
Tel (040) 24 13 93
Cable HEWPACKSA Hamburg
Telex 21 63 032 hphh d

Hewlett-Packard GmbH
Technisches Buero Hannover
Am Grossmarkt 6
D-3000 **Hannover 91**
Tel (0511) 46 60 01
Telex 092 3259

Hewlett-Packard GmbH
Werk Groetzingen
Ohmstrasse 6
D-7500 **Karlsruhe 41**
Tel (0721) 69 40 06
Telex 07-825707

Hewlett-Packard GmbH
Technisches Buero Nuremberg
Neumeyer Str 90
D-8500 **Nuremberg**
Tel (0911) 56 30 83 85
Telex 0623 860

Hewlett-Packard GmbH
Technisches Buero Munchen
Unterhachinger Strasse 28
ISAR Center
D-8012 **Ottobrunn**
Tel (089) 601 30 61 7
Cable HEWPACKSA Munchen
Telex 0524985

Hewlett-Packard GmbH
Technisches Buero Berlin
Keith Strasse 2-4
D-1000 **Berlin** 30
Tel (030) 24 90 86
Telex 18 3405 hpbln d

GREECE
Kostas Karayannis
08, Omirou Street
GR-**Athens** 133
Tel 3237731
Cable RAKAR Athens
Telex 21 59 62 rkar gr

Analytical Only
"INTECO" G. Papathanassiou & Co
Marni 17
GR - **Athens** 103
Tel 522 1915
Cable INTEKNIKA Athens
Telex 21 5329 INTE GR

Medical Only
Technomed Hellas Ltd
52 Skoufa Street
GR - **Athens** 135
Tel 362 6972, 363 3830
Cable ETALAK athens
Telex 21-4693 ETAL GR

HUNGARY
MTA
Müszerügyi és Méréstechnikai
Szolgalata
Lenin Krt. 67
1391 Budapest VI
Tel 42 03 38
Telex 22 51 14

ICELAND
Medical Only
Elding Trading Company Inc
Hafnarhvoli - Tryggvatotu
IS-**Reykjavik**
Tel 1 58 20
Cable ELDING Reykjavik

IRAN
Hewlett-Packard Iran Ltd
No 13, Fourteenth St
Miremad Avenue
P O Box 41/2419
IR-**Tehran**
Tel 851082-7
Telex 21 25 74 khrm ir

IRAQ
Hewlett-Packard Trading Co
4/1/8 Mansoor City
Baghdad
Tel 5517827
Telex 2455 hepairaq ik
Cable HEWPACDAD
Baghdad Iraq

IRELAND
Hewlett-Packard Ltd
King Street Lane
GB-**Winnersh**,Wokingham
Berks, RG11 5AR
Tel (0734) 78 47 74
Telex 847178/848179

ITALY
Hewlett-Packard Italiana S p A
Via Amerigo Vespucci 2
Casella postale 3645
I-20100 **Milano**
Tel (2) 6251 (10 lines)
Cable HEWPACKIT Milano
Telex 32046

Hewlett-Packard Italiana S p A
Via Pietro Maroncelli 40
(ang Via Visentin)
I-35100 **Padova**
Tel (49) 66 48 88
Telex 41612 Hewpacki

Medical only
Hewlett-Packard Italiana S p A
Via d Aghiardi, 7
I-56100 **Pisa**
Tel (050) 2 32 04
Telex 32046 via Milano

Hewlett-Packard Italiana S p A
Via G Armellini 10
I-00143 **Roma**
Tel (06) 54 69 61
Telex 61514
Cable HEWPACKIT Roma

Hewlett-Packard Italiana S p A
Corso Giovanni Lanza
I-1031 **Torino**
Tel (011) 682245-659308

Medical/Calculators Only
Hewlett-Packard Italiana S p A
Via Principe Nicola 43 G/C
I-95126 **Catania**
Tel (095) 37 05 04

Hewlett-Packard Italiana S p A
Via Amerigo Vespucci 9
I-80142 **Napoli**
Tel (081) 33 77 11

Hewlett-Packard Italiana S p A
Via E Masi 9/B
I-40137 **Bologna**
Tel (051) 30 78 87

KUWAIT
Al-Khaldiya Trading &
Contracting Co
P O Box 830-Saft
Kuwait
Tel 424910-411726
Telex 2481 areeg kt
Cable VISCOUNT

LUXEMBURG
Hewlett-Packard Benelux
S A /N V
Avenue du Col-Vert, 1
(Groenkraaglaan)
B-1170 **Brussels**
Tel (02) 672 22 40
Cable PALOBEN Brussels
Telex 23 494

MOROCCO
Gerep
190, Blvd Brahim Roudani
Casablanca
Tel 25-16-76/25-90-99
Cable Gerep-Casa
Telex 23739

NETHERLANDS
Hewlett-Packard Benelux N V
Van Heuven Goedhartlaan 121
P O Box 667
NL-1134 **Amstelveen**
Tel (020) 47 20 21
Cable PALOBEN Amsterdam
Telex 13 216 hepa nl

NORWAY
Hewlett-Packard Norge A/S
Nesveien 13
Box 149
N-1344 **Haslum**
Tel (02) 53 83 60
Telex 16621 hpnas n

POLAND
Biuro Informacji Technicznej
Hewlett-Packard
U1 Stawki 2, 6P
00-950 **Warszawa**
Tel 395962 395187
Telex 81 24 53 hepa pl

UNIPAN
Zaklad Doswiadczalny
Budowy Aparatury Naukowej
U1 Krajowej Rady Narodowej 51 55
00-800 **Warszawa**
Tel 36190
Telex 81 46 48

Zaklady Naprawcze Sprzetu
Medycznego
Plac Komuny Paryskiej 6
90-007 **Lódz**
Tel 334-41, 337-83

PORTUGAL
Telectra-Empresa Técnica de
Equipamentos Eléctricos S a r l
Rua Rodrigo da Fonseca 103
P O Box 2531
P-**Lisbon** 1
Tel (19) 68 60 72
Cable TELECTRA Lisbon
Telex 12598

Medical only
Mundinter
Intercambio Mundial de Comércio
S a r l
Av A A de Aguiar 138
P O Box 2761
P - **Lisbon**
Tel (19) 53 21 31 7
Cable INTERCAMBIO Lisbon

RUMANIA
Hewlett-Packard Reprezentanta
Bd N Balcescu 16
Bucharest
Tel 158023/138885
Telex 10440

I I R U C
Intreprinderea Pentru
Intretinerea
Si Repararea Utilajelor de Calcul
B-dul prof Dimitrie Pompei 6
Bucharest-Sectorul 2
Tel 12 64 30
Telex 11716

SAUDI ARABIA
Modern Electronic Establishment
King Abdul Aziz str (Head office)
P O Box 1228
Jeddah
Tel 31173-332201
Cable ELECTRA

P O Box 2728 (Service center)
Riyadh
Tel 62596-66232
Cable RAOUFCO

SPAIN
Hewlett-Packard Española, S A
Jerez, Calle 3
E-**Madrid** 16
Tel (1) 458 26 00 (10 lines)
Telex 23515 hpe

Hewlett-Packard Española, S A
Milanesado 21-23
E-**Barcelona** 17
Tel (3) 203 6200 (5 lines)
Telex 52603 hpbe e

Hewlett-Packard Española, S A
Av Ramón y Cajal 1
Edificio Sevilla, planta 9,
E-**Seville** 5
Tel 64 44 54/58

Hewlett-Packard Española S A
Edificio Albia II 7 B
E-**Bilbao**-1
Tel 23 83 06/23 82 06

Calculators Only
Hewlett-Packard Española S.A
Gran Via Fernando El Catolico, 67
E-**Valencia**-8
Tel 326 67 28/326 85 55

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 1-3
Fack
S-161 20 **Bromma** 20
Tel (08) 730 05 50
Cable MEASUREMENTS
Stockholm
Telex 10721

Hewlett-Packard Sverige AB
Ostra Vintergatan 22
S-702 40 **Orebro**
Tel (019) 14 07 20

Hewlett-Packard Sverige AB
Frötallsgatan 30
S-421 32 **Vastra Frölunda**
Tel (031) 49 09 50
Telex 10721 Via Bromma Office

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
P O Box 307
CH-8952 **Schlieren-Zurich**
Tel (01) 730 52 40 730 18 21
Cable HPAG CH
Telex 53933 hpag ch

Hewlett-Packard (schweiz) AG
Château Bloc 19
CH-1219 **Le Lignon-Geneva**
Tel (022) 96 03 22
Cable HEWPACKAG Geneva
Telex 27 333 hpag ch

SYRIA
Medical/Calculator only
Sawah & Co
Place Azmé
B P 2308
SYR-**Damascus**
Tel 16367 19697 14268
Cable SAWAH, Damascus

TURKEY
Telekom Engineering Bureau
P O Box 437
Beyoglu
TR-**Istanbul**
Tel 49 40 40
Cable TELEMATION Istanbul
Telex 23609

Medical only
E M A
Muhendislik Kollektif Sirketi
Adakale Sokak 41 6
TR-**Ankara**
Tel 175622

Analytical only
Yilmaz Ozyurek
Milli Mudafaa Cad No 16/6
Kizilay
TR-**Ankara**
Tel 25 03 09
Telex 42576 ozek tr

UNITED KINGDOM
Hewlett-Packard Ltd
King Street Lane
GB-**Winnersh**, Wokingham
Berks RG11 5AR
Tel (0734) 78 47 74
Cable Hewpie London
Telex 847178/9

Hewlett-Packard Ltd
Trafalgar House
Navigation Road
Altrincham
Cheshire WA14 1NU
Tel (061) 928 6422
Telex 668068

Hewlett-Packard Ltd
Lygon Court
Hereward Rise
Dudley Road
Halesowen,
West Midlands B62 8SD
Tel (021) 550 9911
Telex 339105

Hewlett-Packard Ltd
Wedge House
799, London Road
GB-**Thornton Heath**
Surrey CR4 6XL
Tel (01) 684 0103 8
Telex 946825

Hewlett-Packard Ltd
c o Makro
South Service wholesale Centre
Wear Industrial Estate
Washington
GB-**New Town**, County Durham
Tel Washington 464001 ext 57 58

Hewlett-Packard Ltd
10, Wesley St
GB-**Castleford**
West Yorkshire WF10 1AE
Tel (09775) 50402
Telex 557355

Hewlett-Packard Ltd
1, Wallace Way
GB-**Hitchin**
Herts
Tel (0462) 52824 56704
Telex 825981

Hewlet-Packard Ltd
2C Avonbeg Industrial Estate
Long Mile Road
Dublin 12
Tel Dublin 509458
Telex 30439

USSR
Hewlett-Packard
Representative Office USSR
Pokrovsky Boulevard 4 17-KW 12
Moscow 101000
Tel 294-2024
Telex 7825 hewpak su

YUGOSLAVIA
Iskra-standard/Hewlett-Packard
Miklosiceva 38/VII
61000 **Ljubljana**
Telb 31 58 79 32 16 74
Telex 31583

SOCIALIST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard Ges m b H
P O Box 7
A-1205 **Vienna**, Austria
Tel (0222) 35 16 21 to 27
Cable HEWPAK Vienna
Telex 75923 hewpak a

MEDITERRANEAN AND MIDDLE EAST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard S A
Mediterranean and Middle
East Operations
35, Kolokotroni Street
Platia Kefallariou
GR-Kifissia-**Athens**, Greece
Tel 8080337/359/429
Telex 21-6588
Cable HEWPACKSA Athens

FOR OTHER AREAS NOT LISTED CONTACT
Hewlett-Packard S.A
7, rue du Bois-du-Lan
P O Box
CH-1217 Meyrin 2 - **Geneva**
Switzerland
Tel (022) 82 70 00
Cable HEWPACKSA Geneva
Telex 2 24 86

UNITED STATES

ALABAMA
8290 Whitesburg Dr , S E
P.O Box 4207
Huntsville 35802
Tel (205) 881-4591

Medical Only
228 W Valley Ave
Room 220
Birmingham 35209
Tel (205) 942-2081/2

ARIZONA
2336 E Magnolia St
Phoenix 85034
Tel (602) 244-1361

2424 East Aragon Rd
Tucson 85706
Tel (602) 294-3148

***ARKANSAS**
Medical Service Only
P O Box 5646
Brady Station
Little Rock 72215
Tel (501) 376-1844

CALIFORNIA
1430 East Orangethorpe Ave
Fullerton 92631
Tel (714) 870-1000

3939 Lankershim Boulevard
North Hollywood 91604
Tel (213) 877-1282
TWX 910-499-2671

5400 West Rosecrans Blvd
P O Box 92105
World Way Postal Center
Los Angeles 90009
Tel (213) 970-7500

***Los Angeles**
Tel (213) 776-7500

3003 Scott Boulevard
Santa Clara 95050
Tel (408) 249-7000
TWX 910-338-0518

***Ridgecrest**
Tel (714) 446-6165

646 W North Market Blvd
Sacramento 95834
Tel (916) 929-7222

9606 Aero Drive
P O Box 23333
San Diego 92123
Tel (714) 279-3200

COLORADO
5600 South Ulster Parkway
Englewood 80110
Tel (303) 771-3455

CONNECTICUT
12 Lunar Drive
New Haven 06525
Tel (203) 389-6551
TWX 710-465-2029

FLORIDA
P O Box 24210
2806 W Oakland Park Blvd
Ft. Lauderdale 33311
Tel (305) 731-2020

***Jacksonville**
Medical Service only
Tel (904) 398-0663

P O Box 13910
6177 Lake Ellenor Dr
Orlando 32809
Tel (305) 859-2900

P O Box 12826
Pensacola 32575
Tel (904) 476-8422

GEORGIA
P O Box 105005
Atlanta 30348
Tel (404) 955-1500
TWX 810-766-4890

Medical Service Only
***Augusta** 30903
Tel (404) 736-0592

P O Box 2103
Warner Robins 31098
Tel (912) 922-0449

HAWAII
2875 So King Street
Honolulu 96814
Tel (808) 955-4455
Telex 723-705

ILLINOIS
5201 Tollview Dr
Rolling meadows 60008
Tel (312) 255 9800
TWX 910-687-2260

INDIANA
7301 North Shadeland Ave
Indianapolis 46250
Tel (317) 842-1000
TWX 810-260-1797

IOWA
2415 Heinz Road
Iowa City 52240
Tel (319) 338-9466

KENTUCKY
Medical Only
Atkinson Square
3901 Atkinson Dr
Suite 407 Atkinson Square
Louisville 40218
Tel (502) 456-1573

LOUISIANA
P O Box 840
3229-39 Williams Boulevard
Kenner 70063
Tel (504) 443-6201

MARYLAND
6707 Whitestone Road
Baltimore 21207
Tel (301) 944-5400
TWX 710-862-9157

2 Choke Cherry Road
Rockville 20850
Tel (301) 948-6370
TWX 710-828-9684

MASSACHUSETTS
32 Hartwell Ave
Lexington 02173
Tel (617) 861-8960
TWX 710-326-6904

MICHIGAN
23855 Research Drive
Farmington Hills 48024
Tel (313) 476-6400

724 West Centre Ave
Kalamazoo 49002
Tel (606) 323-8362

MINNESOTA
2400 N Prior Ave
St. Paul 55113
Tel (612) 636-0700

MISSISSIPPI
***Jackson**
Medical Service only
Tel (601) 982-9363

MISSOURI
11131 Colorado Ave
Kansas City 64137
Tel (816) 763-8000
TWX 910-771-2087

1024 Executive Parkway
St. Louis 63141
Tel (314) 878-0200

NEBRASKA
Medical Only
7171 Mercy Road
Suite 110
Omaha 68106
Tel (402) 392-0948

NEW JERSEY
W 120 Century Rd
Paramus 07652
Tel (201) 265-5000
TWX 710-990-4951

Crystal Brook Professional
Building
Eatontown 07724
Tel (201) 542-1384

NEW MEXICO
P O Box 11634
Station E
11300 Lomas Blvd N E
Albuquerque 87123
Tel (505) 292-1330
TWX 910-989-1185

156 Wyatt Drive
Las Cruces 88001
Tel (505) 526-2484
TWX 910-9983-0550

NEW YORK
6 Automation Lane
Computer Park
Albany 12205
Tel (518) 458-1550

201 South Avenue
Poughkeepsie 12601
Tel (914) 454-7330
TWX 510-253-5981

650 Perinton Hill Office Park
Fairport 14450
Tel (716) 223-9950

5858 East Molloy Road
Syracuse 13211
Tel (315) 454-2486
TWX 710-541-0482

1 Crossways Park West
Woodbury 11797
Tel (516) 921-0300
TWX 710-990-4951

NORTH CAROLINA
P O Box 5188
1923 North Main Street
High Point 27262
Tel (919) 885-8101

OHIO
16500 Sprague Road
Cleveland 44130
Tel (216) 243-7300
TWX 810-423-9430

330 Progress Rd
Dayton 45449
Tel (513) 859-8202

1041 Kingsmill Parkway
Columbus 43229
Tel (614) 436-1041

OKLAHOMA
P O Box 32008
Oklahoma City 73132
Tel (405) 721-0200

OREGON
17890 SW Lower Boones
Ferry Road
Tualatin 97062
Tel (503) 620-3350

PENNSYLVANIA
111 Zeta Drive
Pittsburgh 15238
Tel (412) 782-0400

1021 8th Avenue
King of Prussia Industrial Park
King of Prussia 19406
Tel (215) 265-7000
TWX 510-660-2670

SOUTH CAROLINA
6941-0 N Trenholm Road
Columbia 29260
Tel (803) 782-6493

TENNESSEE
***Knoxville**
Medical Service only
Tel (615) 523-5022

3027 Vanguard Dr
Director s Plaza
Memphis 38131
Tel (901) 346-8370

Nashville
Medical Service only
Tel (615) 244-5448

TEXAS
P O Box 1270
201 E Arapaho Rd
Richardson 75080
Tel (214) 231-6101

10535 Harwin Dr
Houston 77036
Tel (713) 776-6400

205 Billy Mitchell Road
San Antonio 78226
Tel (512) 434-8241

UTAH
2160 South 3270 West Street
Salt Lake City 84119
Tel (801) 972-4711

VIRGINIA
P O Box 12778
No 7 Koger Exec Center
Suite 212
Norfolk 23502
Tel (804) 461-4025/6

P O Box 9669
2914 Hungary Springs Road
Richmond 23228
Tel (804) 285-3431

WASHINGTON
Bellefield Office Pk
1203-114th Ave S E
Bellevue 98004
Tel (206) 454-3971
TWX 910-443-2446

***WEST VIRGINIA**
Medical/Analytical Only
Charleston
Tel (304) 345-1640

WISCONSIN
9004 West Lincoln Ave
West Allis 53227
Tel (414) 541-0550

FOR U.S. AREAS NOT LISTED:
Contact the regional office
nearest you Atlanta Georgia
North Hollywood California
Rockville Maryland Rolling Meadows
Illinois Their complete
addresses are listed above

***Service Only**

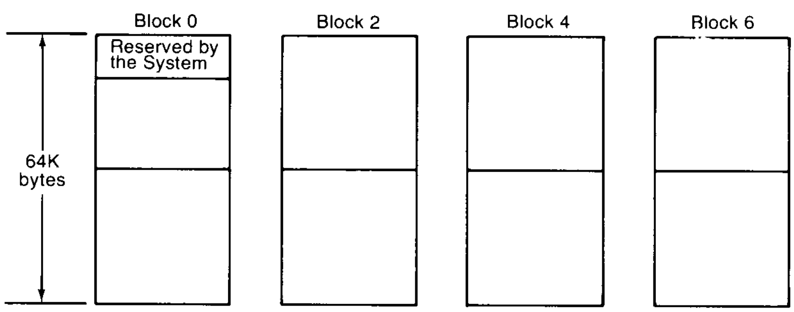
1/78

E

Appendix F
Memory Organization

The appendix delves further into organization of User Read/Write Memory. It is not intended to be a complete explanation of memory organization, but to explain memory configuration as it relates to programming operations.

Read/Write Memory is divided into blocks. Each block is 64K bytes. The following diagram illustrates the blocks of memory. Odd-numbered blocks are used by the system and are not part of user Read/Write Memory.



The division of memory into blocks imposes limitations on programs and variables. The limitations are –

- No main program or subprogram can be larger than one block of memory. A 1000-line program typically fills one half of a 64K block.

- No main program or subprogram can cross a block boundary. That is, the main or subprogram must be contained entirely in one block of memory.

  This limitation may cause you to get an unexpected memory overflow error, `ERROR 2`, though executing `LIST` indicates there is ample memory available. The reason for this is that the available memory is not all in the same block.

- To avoid this situation, it is advisable to organize your program into a series of a short main program and short subprograms, rather than using long program segments. This works well because a block can contain more than one subprogram. Additionally, a program can consist of a main program and subprograms in several different blocks.

- No simple variable or array element can cross a block boundary. Arrays of long strings and long simple strings can also cause an unexpected memory overflow or waste large amounts of memory. For example, suppose you are allocating memory to some variables in a `DIM` or `COM` statement. Suppose that there is a 25K byte character string following a numeric variable, but only 10K bytes left in the block after the numeric variable is allocated memory space. The string will have to be stored in the next block, thus wasting 10k bytes of memory. Thus, the order of large strings in `DIM` or `COM` statements can affect the amount of memory needed to run a program.

- Each time an array crosses a block boundary, six bytes of memory are added to the total amount needed to store the array.

- The execution stack and any binary routines must be contained in block 0. You could get a memory overflow when the other blocks are virtually empty if the execution stack gets too large. This can be caused by recursive subprogram calls and intermediate results involving long strings. Some program restructuring may be necessary.

Appendix G
Error Messages

Mainframe Errors

1 Missing ROM or configuration error

2 Memory overflow; subprogram larger than block of memory. (See Appendix F)

3 Line not found or not in current program segment

4 Improper return

5 Abnormal program termination; no END or STOP statement

6 Improper FOR/NEXT matching

7 Undefined function or subroutine

8 Improper parameter matching

9 Improper number of parameters

10 String value required

11 Numeric value required

12 Attempt to redeclare variable

13 Array dimensions not specified

14 Multiple OPTION BASE statements or OPTION BASE statement preceded by variable declarative statements

15 Invalid bounds on array dimension or string length in memory allocation statement

16	Dimensions are improper or inconsistent; more than 32 767 elements in an array
17	Subscript out of range
18	Substring out of range or string too long
19	Improper value
20	Integer precision overflow
21	Short precision overflow
22	Real precision overflow
23	Intermediate result overflow
24	TAN (N*π/2), when N is odd
25	Magnitude of argument of ASN or ACS is greater than 1
26	Zero to negative power
27	Negative base to non-integer power
28	LOG or LGT of negative number
29	LOG or LGT of zero
30	SQR of negative number
31	Division by zero; X MOD Y with Y = 0
32	String does not represent valid number or string response when numeric data required
33	Improper argument for NUM, CHR$, or RPT$ function
34	Referenced line is not IMAGE statement
35	Improper format string
36	Out of DATA
37	EDIT string longer than 160 characters
38	I/O function not allowed
39	Function subprogram not allowed
40	Improper replace, delete or REN command

G

41	First line number greater than second
42	Attempt to replace or delete a busy line or subprogram
43	Matrix not square
44	Illegal operand in matrix transpose or matrix multiply
45	Nested keyboard entry statements
46	No binary in memory for STORE BIN or no program in memory for SAVE
47	Subprogram COM declaration is not consistent with main program
48	Recursion in single-line function
49	Line specified in ON declaration not found
50	File number less than 1 or greater than 10
51	File not currently assigned
52	Improper mass storage unit specifier
53	Improper file name
54	Duplicate file name
55	Directory overflow
56	File name is undefined
57	Mass Storage ROM is missing
58	Improper file type
59	Physical or logical end-of-file found
60	Physical or logical end-of-record found in random mode
61	Defined record size is too small for data item
62	File is protected or wrong protect code specified
63	The number of physical records is greater than 32767
64	Medium overflow (out of user storage space)
65	Incorrect data type
66	Excessive rejected tracks during a mass storage initialization
67	Mass storage parameter less than or equal to 0

G

68	Invalid line number in GET or LINK operation
69 – 79	See Mass Storage ROM errors
80	Cartridge out or door open
81	Mass storage device failure
82	Mass storage device not present
83	Write protected
84	Record not found
85	Mass storage medium is not initialized
86	Not a compatible tape cartridge
87	Record address error; information can't be read
88	Read data error
89	Check read error
90	Mass storage system error
91–99	See Mass Storage ROM errors
100	Item in print using list is string but image specifier is numeric
101	Item in print using list is numeric but image specifier is string
102	Numeric field specifier wider than printer width
103	Item in print using list has no corresponding image specifier
104 – 109	Unused
110– 113	See Plotter ROM errors

System Error octal number; octal number

This error indicates an error in the machine's firmware system; it is a fatal error. If reset does not bring control back, the machine must be turned off, then on again. If the problem persists, contact your Sales and Service Office.

I/O Device Errors

Two error messages can occur when attempting to direct an operation to an I/O device that is not ready for use. A printer which is out of paper or no device at a specifed select code are examples. The first message that appears is –

I/O ERROR ON SELECT CODE select code

If the condition is not corrected, the machine beeps intermittently and the following message replaces the first –

I/O TIMEOUT ON SELECT CODE select code

The I/O device can be made usable by correcting the error (loading paper for example), then executing the READY# command –

READY# select code

This command readies the I/O device and the operation which was attempted is attempted again. The select code must be specified by an integer.

In some cases, such as an interface which is not connected, READY# for that select code may not solve the I/O error. In this case, STOP should be pressed to regain control of the computer. Be sure to turn the power off before inserting an interface. After the problem is remedied, the operation or program can be tried again.

G

Mass Storage ROM Errors

69	Format switch off
70	Not a disc interface
71	Disc interface power off
72	Incorrect controller address, or controller power off
73	Incorrect device type in mass storage unit specifer
74	Drive missing or power off
75	Disc system error
76	Incorrect unit code in mass storage unit specifier
77–79	Unused
91–99	Unused

Plotter ROM Errors

110	Plotter type specification not recognized
111	Plotter has not been specified
112	Unused
113	LIMIT specifications out of range.
114–119	Unused

G

Subject Index

a

b

c

d

e

f

g

h

i

k

l

m

n

o

p

t

U

V

W

Your Comments, Please...

Your comments assist us in improving the usefulness of our publications; they are an important part of the inputs used in preparing updates to the publications.

In order to write this manual, we made certain assumptions about your computer background. By completing and returning the comments card on the following page, you can assist us in adjusting our assumptions and improving our manuals.

Feel free to mark more than one reply to a question and to make any additional comments.

If the comments card is missing, please address your comments to:

HEWLETT-PACKARD COMPANY
Desktop Computer Division
3404 East Harmony Road
Fort Collins, Colorado 80525 U.S.A.
Attn. Controller Documentation