HEWLETT-PACKARD

Mass Storage ROM Manual

HP-83/85

$$Z=\left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & V_{11} & V_{12} \\ 0 & 0 & V_{21} & V_{22} \end{array}\right]=\left[\begin{array}{c|c} I & 0 \\ \hline 0 & V \end{array}\right]$$

$$X=(A^TA)^{-1}A^T$$

$$A=\begin{bmatrix} 0 & & & & 0.2527 \\ 0 & & 0.8660 & 0.8348 & 0.2474 \\ & & 0.0000 & 0.7610 & 1.0506 \\ & & 0.0 & 0.0569 & -0.2520 \\ & & 0.0 & -0.4688 & -0.5963 \end{bmatrix}$$

$$K=\begin{bmatrix} -0.0709 & 0.1861 & -0.8383 & -0.4806 & 0.1632 \\ 0.0726 & -0.4109 & -0.5354 & -0.7253 & -0.1144 \\ -0.2002 & -0.6231 & 0.1027 & -0.1411 & 0.7357 \end{bmatrix}$$

$$R=\begin{bmatrix} 1.4446 & -1.6867 & 1.2530 \\ 0.0 & -1.3389 & -0.1486 \\ 0.0 & 0.0 & 1.1831 \end{bmatrix}$$

HEWLETT PACKARD

HP-83/85
Mass Storage ROM Manual

November 1980

00085-90447

Printed in U.S.A.

Contents

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Section 1

Getting Started

Introduction

The HP-83/85 Mass Storage ROM allows you to interface your HP-83 or HP-85 computer with the HP 9895 and HP 82900-Series Flexible Disc Drives. By adding over 30 BASIC statements and commands, the ROM enables you to access your mass storage system for program and data storage and retrieval. This manual explains the proper use of each of these additional capabilities generated by the ROM.

Coverage in this manual assumes you are familiar with the operation and programming of your HP-83 or HP-85 and that you have your computer's owner's manual available for reference. Sample programs in this manual assume that you have some knowledge of the statements, commands, and functions discussed in the owner's manual.

If your computer includes its own internal tape unit (the HP-85), then you are probably familiar with some of the material discussed here. You may find portions of this manual overlap coverage in the section of your owner's manual that deals with the tape unit and that you can skim through the material you already know. However, you should keep in mind that the presence of the ROM changes the way in which you access your internal tape unit. These changes are discussed in section 3 of this manual.

Installation of the Mass Storage ROM

The ROM must be properly installed in one of the six slots in the HP 82936A ROM Drawer. The ROM Drawer is then plugged into one of the module ports on your HP-83/85. Please refer to the *HP 82936A ROM Drawer Instruction Sheet* or to the portion of your owner's manual dealing with the ROM Drawer for complete instructions. You should never have more than one Mass Storage ROM installed in the ROM Drawer.

Installation of the HP-IB Interface and Disc Drive

Your mass storage device must be connected to your HP-83/85 by the HP 82937A HP-IB interface. Refer to the instructions with your interface and mass storage device for complete installation instructions.

The HP-IB Select Code

Each interface attached to your HP-83/85 must be identified by its own unique interface select code. The interface select code allows you to address an individual interface to which a particular device is attached.

The select code on the HP-IB interface has been factory preset to 7. The serial, BCD, and GPIO interfaces are preset to the other numbers. If you have more than one HP-IB interface connected to your personal computer, you must make sure there is no duplication of select codes among the attached interfaces. Refer to the *HP-IB Peripheral Installation Instructions* or to the *Hewlett-Packard 82937A HP-IB Installation and Theory of Operation Manual*, if necessary, for directions on changing the interface select code.

Samples in this manual assume an interface select code 7 for the HP-IB interface to which your mass storage device is attached.

Device Address Switch

Since each HP-IB interface can accept up to eight mass storage master units, each master unit on the interface must have a unique device address. This device address is then used to access a particular mass storage device. The device address is set using the device address switch located on each master unit. Each master unit has a factory preset device address (refer to the operator's manual for your unit). Since each device on a particular interface must have a different device address, it may be necessary to reset the device address of a unit before configuring it to the computer. The following table lists switch positions for changing a unit's device address. (Refer to the operator's manual for your unit for further instructions.)

Switch			Value
1	2	3	
on	on	on	0
on	on	off	1
on	off	on	2
on	off	off	3
off	on	on	4
off	on	off	5
off	off	on	6
off	off	off	7

The examples in this manual assume you have an HP 82901M Flexible Disc Drive, which is a "master" unit with two drives. The device address for this unit is preset to 0, and the examples in this manual assume the switch has remained set to this number.

If your system contains an add-on unit attached to a master unit, the add-on unit has the same device address as the master unit.

Disc Drive Numbers

The disc drive numbers identify individual drives at a particular device address. These drives include both the master unit and the add-on, if present. A maximum of four drives can be connected at any one address (one dual master unit and one dual add-on unit). Disc drive numbers range from 0 through 3.

The HP 82900-Series Flexible Disc Drives have the following preset drive numbers. The drive numbers appear on the front panel of each unit.

HP 82902M Flexible Disc Drive (single master)	DRIVE 0
HP 82902S Flexible Disc Drive (single add-on)	DRIVE 2
HP 82901M Flexible Disc Drive (dual master)	DRIVE 0, DRIVE 1
HP 82901S Flexible Disc Drive (dual add-on)	DRIVE 2, DRIVE 3

The HP 9895A Flexible Disc Drives have the following preset drive numbers.

HP 9895A Option 010 (single master)	DRIVE 0
HP 9895A Option 011 (single add-on)	DRIVE 2
HP 9895A (dual master)	DRIVE 0, DRIVE 1
HP 9895A Option 012 (dual add-on)	DRIVE 2, DRIVE 3

For information about drive numbers of other Hewlett-Packard mass storage devices, refer to the instructions for those devices.

System Summary

Figure 1 on page 8 summarizes the configurations of a mass storage system. Keep in mind that any one HP-IB interface may have up to eight master units attached, and that more than one interface may be connected to your computer.

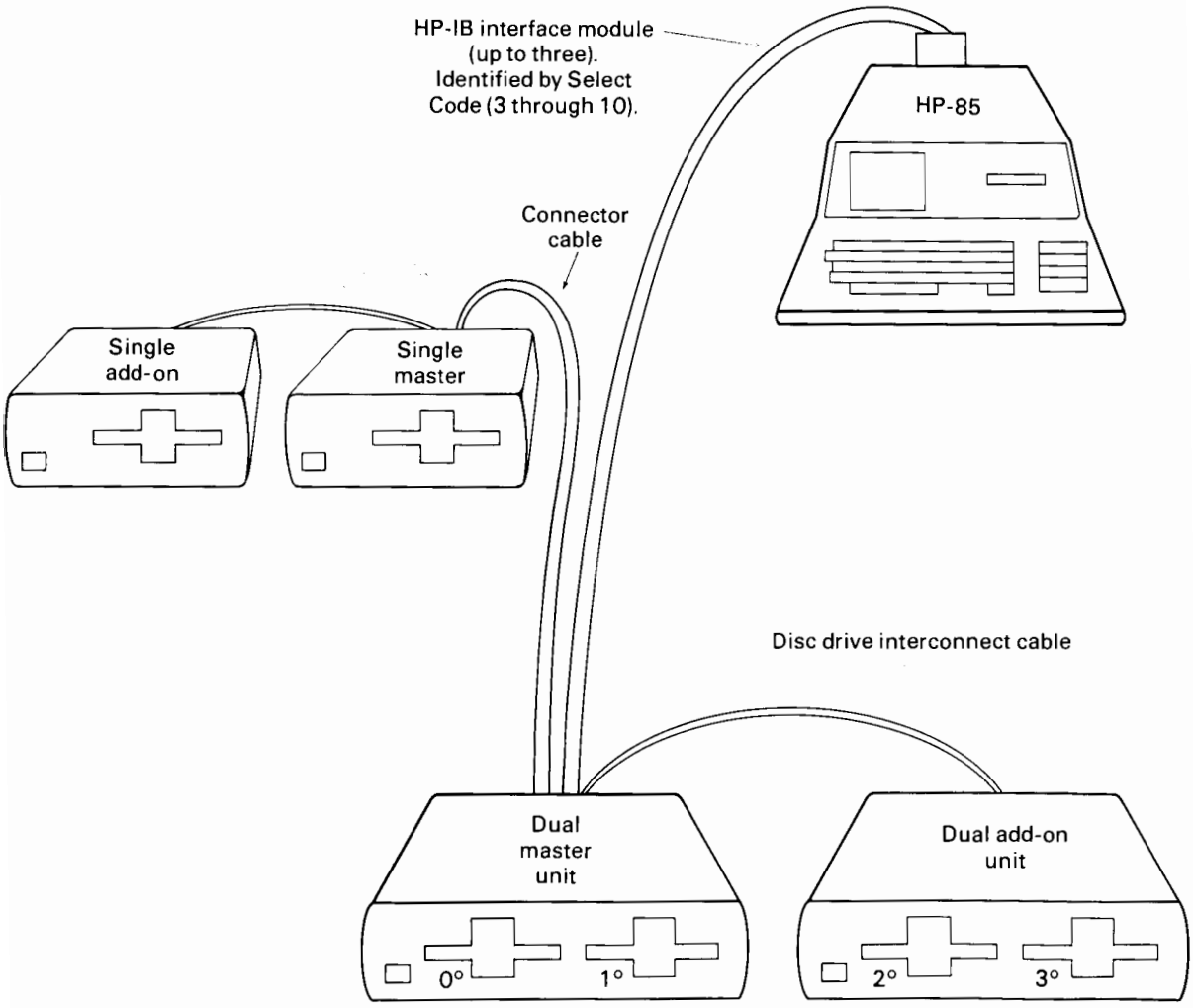


Figure 1. System summary.

Memory Requirements of the Mass Storage ROM

All ROMs utilize a certain amount of computer memory that was previously available working space. The Mass Storage ROM consumes 150 bytes of memory. You may find that a large program written on your HP-83/85 without the ROM in place may be too large to be entered into memory when one or more ROMs are installed. If `Error 19 : MEM OVF` occurs upon attempting to load a large program, you may add a 16K memory module or remove the ROM before entering the program from the keyboard or internal tape drive, if present.

Syntax Guidelines

The following conventions are used in the Mass Storage ROM manual for syntax descriptions of statements and commands.

`DOT MATRIX`	Items shown in dot matrix are typed in exactly as shown, except that lower case letters may be substituted for upper case letters.
italics	Items shown in italics are numeric constants, numeric variables, numeric expressions and string expressions that must be included in the statement.
[ ]	Brackets are used to enclose optional items.
stacked items	When items are placed one above the other, one and only one must be chosen.

Section 2

Accessing Your Mass Storage System

Your mass storage system will greatly expand the capabilities of your HP-83/85. Among the operations available to you are:

- Storing programs for future use.
- Creating and accessing data files tailored to your particular computing needs.
- Storing and retrieving graphics displays.
- Copying files from one mass storage medium to another.
- Running programs whose memory needs exceed available space in your personal computer by storing individual program segments in mass storage and recalling them into computer memory one at a time.

Section 2 covers how to access any particular drive on any mass storage unit in your system. Even if you are familiar with other mass storage systems, you should review this information to familiarize yourself with the syntax of HP-83/85 Mass Storage ROM statements. Instructions on how to access particular files on a disc will be covered in section 3. Sections 4 through 6 discuss accessing stored programs, data files, and graphics displays. A number of other file manipulations and techniques for error processing are covered in sections 7 and 8.

Mass Storage Unit Specifier

In order to store and retrieve information with your mass storage system, you must specify the exact location of the device on which the information is stored. The *mass storage unit specifier*, or msus, is a character string that combines an interface select code, address of the master unit, and drive number to specify the location of a particular file. Accessing the file itself will be covered in section 3.

The msus string has the following form:

" : *device type* [*interface select code device address drive number*] "

All msus character strings begin with a colon.

The device type identifies the type of mass storage device being accessed, either disc or tape. The symbol : D specifies disc and : T specifies the internal tape drive, if present.

All of the optional parameters in the msus (interface select code, device address, drive number) must be included when specifying a flexible disc drive unit. The optional parameters are omitted when specifying the internal tape unit on the HP-85.

The interface select code identifies the HP-IB interface to which the mass storage unit is attached. The interface select code is factory preset to 7, but may be reset to an integer 3 through 10. An interface select code greater than 10 is interpreted as 10.

The device address, an integer from 0 through 7, matches the number set on the device address switch on the mass storage unit.

The drive number, an integer from 0 through 3, specifies the drive on the master or add-on unit you wish to access.

The msus of the dual disc drive units shown in figure 1, page 8, are listed below.

`":D700"`	`":D701"`	`":D702"`	`":D703"`
disc select code 7 device address 0 Drive 0	disc select code 7 device address 0 Drive 1	disc select code 7 device address 0 Drive 2	disc select code 7 device address 0 Drive 3

The following msus specifies the HP-85 internal tape unit.

`":T"`

When specifying an internal tape unit, the interface select code, device address, and drive number are not included.

Volume Labels

Volume labels offer you a convenient way to specify a particular mass storage medium (disc). You cannot assign a volume label to a tape cartridge.

A volume label is a name up to six characters in length that you assign to a disc when the disc is initialized, or by using a `VOLUME IS` statement. The volume label is stored on the disc and remains the disc's name until a new volume label is assigned to the disc.

Like the msus, the volume label is a string and must be enclosed in quotes. The only characters that you should not use in volume labels are: `.` (period), `:` (colon), and `"` (quotes). Volume labels longer than six characters are truncated to six characters. Once a volume label has been assigned to a disc, the character string designating that volume label always must include a period within the quotes preceding the characters.

The form of a volume label is:

```
".AAAAAA"
```

"A" may be any character except a period, colon, or quotation mark. Note that the volume label character string is preceded by a period.

The syntax of the VOLUME IS statement is:

```
VOLUME [ ":msus" | ".old volume label" ] IS "new volume label"
```

Note that the new volume label is not preceded by a period. The period must be included only after the volume label has been assigned. The statement may be executed within a program or from the keyboard.

When a volume label is used to access the medium on which information is stored, the system searches the discs currently in the system until the disc with that volume label is found. If the search fails to find the specified volume label, the Mass Storage ROM returns Error 125 : VOLUME. Because of this search operation, it takes more time to access a file using a volume label than by using the msus. If the same volume label has been assigned to more than one disc in the system, the disc with the lowest msus is accessed.

Examples:

```
20 VOLUME ":D700" IS "DRIVE0"
```

Assigns the volume label ".DRIVE0" to the disc located at msus ":D700".

```
150 VOLUME ".DRIVE0" IS "D-0"
```

Renames the disc formerly labeled ".DRIVE0" to ".D-0".

The examples in the remainder of this manual assume the following volume label assignments. It is also assumed that disc ".DRIVE0" is always located at msus ":D700" and that disc ".DRIVE1" is always located at msus ":D701".

```
VOLUME ":D700" IS "DRIVE0"
VOLUME ":D701" IS "DRIVE1"
```

Specifying Parameters Using Expressions

You can specify any parameter using expressions. When a parameter is a string, you can specify that parameter using string expressions. When a parameter is a number, you can specify that parameter using numeric expressions. In most cases, if the parameter is an integer, a non-integer value supplied by a numeric expression is rounded to the nearest integer. However, attempts to use an expression evaluating to a non-integer as part of a msus will generate Error 126 : MSUS.

Here are some examples of how string expressions can be used to create volume labels and msus values. Refer to your computer owner's manual for additional information about string expressions.

```
50 D$="DRIVE" @ N=1
60 VOLUME ":D701" IS D$ & VAL$(N
)
```

Assigns the volume label ".DRIVE1" to the disc at msus ":D701". Note that the period in the new name is omitted in the VOLUME IS statement.

```
20 B$=":D7" @ J=1
30 VOLUME B$ & VAL$(J-1) & "1" I
S "DRIVE1"
```

These statements accomplish the same task as statements 50 and 60, above.

Initializing a New Storage Disc

Each flexible disc that you use in your system must be initialized at least once. Initializing establishes a volume label, sets up a file directory for the disc, and clears and tests the disc. The INITIALIZE command accomplished these things. You cannot initialize a tape cartridge with this statement.

Optional parameters in the statement can be used to:

- Rename a disc (change the volume label).
- Specify the amount of space allocated to the disc directory.
- Specify how the physical records on the disc are to be numbered.

The initialization process takes about two minutes. Any information stored on the disc is erased by the INITIALIZE command. If you are uncertain whether or not a disc has been previously initialized, insert the disc into DRIVE 0 and type CAT (ENDLINE). The message Error 130 : DISC indicates that the disc has not been initialized. The INITIALIZE command is programmable.

The form of the INITIALIZE command is:

```
INITIALIZE["new volume label"[,"          ":msus"
                               :old volume label"[,directory size
  [,interleave factor]]]]
```

In the INITIALIZE command each listed optional parameter must be preceded by all the optional parameters listed before it. For instance, the directory size must be preceded by both a new volume label and a msus or old volume label.

The new volume label is the new name assigned to the flexible disc being initialized (refer to page 12 for details on volume labels). If the new volume label is omitted, it defaults to blanks.

The msus or old volume label is the existing label or msus of the disc being initialized. If this parameter is omitted, the default disc specified by the MASS STORAGE IS statement is used. You cannot specify msus ":T".

The directory size specifies the number of records to be allocated on the disc for the file directory. Each record holds directory information for eight files. The default value is 14 records (or $14 \times 8 = 112$ files).

The interleave factor specifies how physical records on the disc are to be numbered. Any integer from 1 through 16 may be specified for the HP 82900-Series Flexible Disc Drives, causing sequential records on the disc to be numbered consecutively, by every other record, every third record, etc. The default value for the interleave factor is 5.

The ability to renumber records on a disc by specifying an interleave factor allows you to control the efficiency of your disc drives and to minimize the time required to access mass storage files.

The interleave factor affects how many revolutions of the disc are necessary to transfer information to and from mass storage. Because it takes a finite amount of time to perform accessing operations, and because the disc is spinning rapidly, it is possible that a full revolution might be required to access successive records on the disc. By placing a physical separation between records, the appropriate interleave factor can minimize the number of "wasted" revolutions.

The performance of your mass storage system during a particular application can be improved by adapting the interleave factor to the structure of your data. Since there is no easy way to compute the best interleave factor for a particular data configuration, the simplist way to determine the most efficient interleave factor is by "trial and error."

One method of testing interleave factors involves copying your program and data from a "master" disc to a "test" disc that has been initialized to a different interleave factor. Then, time the execution of the program, using the computer's internal timer. You may initialize the "test" disc repeatedly using a different interleave factor each time, COPY the same data onto the disc (remember, the data was lost when the disc was reinitialized), and re-execute the program to compare execution times.

Below are several examples of the proper form of the INITIALIZE command.

```
10 INITIALIZE "DRIVE1",":D701"

10 INITIALIZE "DRIVEA",".DRIVE1",
,15,2
```

Establishing a Default Mass Storage Medium

A default mass storage medium (disc) is established by the MASS STORAGE IS statement, which has the form:

```
MASS STORAGE IS [ ".volume label" ]
                [ ":msus"         ]
```

Once a default device is set up, the system automatically uses that device when the volume label or msus is not specified. When no default device has been established, the system defaults to the disc with the lowest address number. If no device is present, or if the disc drive is turned off, the system defaults to an internal tape unit (":T"), if present.

Examples:

```
MASS STORAGE IS ".DRIVE0"
```
The default is set to the medium with volume label ".DRIVE0".

```
MASS STORAGE IS ":D701"
```
The default is set to msus ":D701".

Notes

Section 3

Accessing Files

Data and programs are stored on a mass storage medium, such as disc or tape, in files. By assigning each file a name, you can access previously stored information by using the appropriate BASIC statement to call up that file.

If your computer has an internal tape unit, you may already have some experience in storing and retrieving files. However, the Mass Storage ROM requires that a different form of file name be used for storage on both disc and tape. Attempts to store information on your internal tape drive using file name conventions you learned in the section of your computer owner's manual dealing with tapes could result in the information instead being recorded onto a default mass storage medium.

The location of a file in your mass storage system is described by a file specifier. The file specifier consists of two parts: a one- to ten- (six for tape) character file name, and a volume label or msus. The volume label or msus identifies the particular disc drive (or tape) on which the file is located. The file name distinguishes any one file from others stored on the same disc (or tape).

The proper form of a file specifier is:

"*file name* ["*volume label* / :*msus*]"

File specifiers are always enclosed in quotes. Note that the volume label or msus is shown as being optional. This is because the system automatically uses the default device established by the configuration of the system, or specified by a `MASS STORAGE IS` statement, when the optional parameter is omitted. Consequently, the volume label or msus must be included if the file is located elsewhere than the default mass storage medium.

Examples:

`"QUAKE.DRIVE0"`	The file named QUAKE is on the medium having volume label `".DRIVE0"`.
`"QUAKE:D700"`	The file named QUAKE is on the device having msus `":D700"`.
`"QUAKE:T"`	The file named QUAKE is on the HP-85 internal tape unit.

Here are several examples of file specifiers used with the default device established first. Remember that the default device you establish remains in effect until you change it again or reset the computer.

```
10 MASS STORAGE IS ":D701"
```
Establishes a mass storage default medium.

```
20 CREATE "PRESSURES", 5
```
Creates a data file named PRESSURES on the disc having msus ":D701".

```
MASS STORAGE IS ":T"
STORE "BIKE"
```
HP-85 tape unit is the default device. The program BIKE is stored on tape.

The only characters that should not be used in the file name portion of a file specifier are . (period), : (colon) and " (quotes). The period is reserved as the volume label prefix, the colon is the msus prefix, and the quotes are used to delimit strings. Null file names are not allowed; however, blanks are allowed. File names longer than 10 characters (6 for tape) will be truncated to 10 characters (6 for tape).

The File Direc

Each mass storage medium (a disc or tape) automatically maintains a catalog, or file directory, of the files stored on it. The CAT command outputs the contents of the file directory to the computer display.

The proper form of the CAT command is:

```
CAT[ ".volume label" ]
     ":msus"
```

If you have previously initialized a disc as ".DRIVE0", you can now obtain a file directory of that disc by typing in CAT ".DRIVE0" (END LINE).

```
CAT ".DRIVE0"
Volume:DRIVE0
Name          Type    Bytes     Recs
```

Once you have stored programs and created data files on a mass storage medium, the file directory will look similar to the one shown below.

```
CAT ".DRIVE0"
Volume:DRIVE0
Name          Type    Bytes     Recs
TEMPDATA      DATA       48        5
LEASTSQ       PROG      256        2
              NULL      256        1
EARNINGS      DATA      500        2
```

The file directory contains the following information:

Name — This is the name assigned to the file as part of the file specifier.

Type — There are five types of files: DATA, PROG (program), BPGM (binary program), NULL, and **** (extended).

Bytes — The number listed is the number of bytes per file record.

Recs — This is the number of records in the file.

If your computer contains an internal tape unit, typing in CAT ":T" (END LINE) will output the tape directory. The tape directory contains all the information in a disc directory, plus one additional column, FILE, which lists the file number of each file. (Refer to your HP-85 owner's manual for more details on file numbers.)

You may terminate a catalog listing at any time by pressing any key.

File Types

As mentioned in the discussion of file directories, five types of files may be used with a mass storage system: program, data, extended, binary program, and null. Each file type is created and retrieved by different procedures, summarized below. Each file type is discussed at greater length elsewhere in this manual.

File Type	Description
PROG (program)	These files contain programs and are created with STORE and retrieved into computer memory using LOAD. Program files are covered in section 4.
DATA	Data files are created using CREATE and PRINT# and retrieved with READ#. Data files are covered in section 5.
**** (extended)	Extended files are used to store graphics displays. GSTORE is the only statement in the Mass Storage ROM that creates extended files. The GSTORE extended files are covered in section 6.
BPGM (binary program)	These files are binary programs and are created using STOREBIN and retrieved using LOADBIN. Binary program files are covered in section 4.
NULL	Null files are empty files created when individual files are purged. They are removed from the directory with PACK. Null files are covered in section 7.

Section 4

Storing and Retrieving Programs

Information in this section covers how to store and retrieve programs using a mass storage system. Use of chaining to expand the capability of the computer in running large programs is also covered.

STORE

The STORE command is used to store the program currently in computer memory on a mass storage medium (tape or disc). STORE attaches a specified name to the program, creates a program file with that name, and then stores the program in the program file using the computer's unique language. The stored program remains in computer memory until scratched, or until another program is loaded.

STORE is not programmable. The command may be typed in, or you may use the typing aid (STORE).

The proper form of the STORE command is:

```
STORE "file specifier"
```

The proper form for file specifiers is covered in section 3.

Examples:

```
STORE "QUAKE.DRIVE0"
```

Names the program in computer memory QUAKE, and stores the program in a program file located on the mass storage medium with volume label ".DRIVE0".

Remember that you can use either a volume label or msus in a file specifier.

```
STORE "QUAKE:D700"
```

Has the same effect as the previous example if ".DRIVE0" is ":D700".

You may omit the volume label or msus portion of the file specifier if the program is to be stored onto the default mass storage medium.

```
MASS STORAGE IS ".DRIVE0"
STORE "QUAKE"
```

Assigns the default mass storage device. (Assume that the volume label was previously assigned.)

If you do not have much experience with mass storage systems, you might want to practice storing (and later in this section, retrieving) a program. The following program converts speeds input in one of four units to any of the other four units. The four units are:

F/S	feet per second
MPH	miles per hour
KM/H	kilometers per hour
M/S	meters per second

If you intend to store this program, you must first make sure you have a disc which has been initialized. If you have not yet initialized a disc, do so now in Drive 0 of your unit, following instructions on page 14.

```
INITIALIZE "DRIVE0",":D700"
```

Note that the msus is optional here, since DRIVE 0 is the default device.

Now, obtain a file directory of the disc by typing CAT (END LINE).

```
CAT ":DRIVE0"
Volume:DRIVE0
Name          Type   Bytes    Recs
```

Type in the program as shown.

```
10 DISP "ENTER SPEED, CURRENT U
UNITS"
20 INPUT S,U$
30 DISP "CONVERSION UNITS";
40 INPUT U1$
50 S1=S
60 IF U$="F/S" THEN 110
70 IF U$="MPH" THEN 130
80 IF U$="KM/H" THEN 150
90 S1=S1*3.281 ! M/S TO F/S
100 IF U1$="F/S" THEN 180
110 S1=S1*.6818 ! F/S TO MPH
120 IF U1$="MPH" THEN 180
130 S1=S1*1.609 ! MPH TO KM/H
140 IF U1$="KM/H" THEN 180
150 S1=S1*.2778 ! KM/H TO M/S
160 IF U$="M/S" THEN 180
170 GOTO 90
180 PRINT USING 190; S,U$,S1,U1$
190 IMAGE 6D.3D,X,AAAA,"=",6D.3D
,X,AAAA
200 END
```

To store the program, type (or use the typing aid):

```
STORE "SPEEDS.DRIVE0"
```

The red pilot light on Drive 0 will be on during the storing process. When the light goes off, the program SPEEDS has been stored on disc ":DRIVE0". To see the updated file directory, execute CAT from the keyboard.

```
CAT ":DRIVE0"
Volume:DRIVE0
Name          Type    Bytes    Recs
SPEEDS        PROG      256       3
```

The directory shows that SPEEDS has been stored in a program file three records in length. Each record contains 256 bytes.

STORE can be used to store a program in computer memory over a program that was stored previously. For instance, after storing SPEEDS, you may edit the program in computer memory, and then re-execute:

```
STORE "SPEEDS.DRIVE0"
```

The new, edited version will be stored, replacing the first version. Because of this "overlay" capability, you must be careful in storing a new program not to accidentally assign to it the name of another program file, thereby overwriting a previously stored program that you still need.

Loading a Program From Mass Storage

Once a program has been stored on a mass storage medium, a copy can be retrieved into computer memory with the LOAD command. Like STORE, the LOAD command is not programmable. The proper form is:

```
LOAD "file specifier"
```

The file specifier must correspond to a program in mass storage. Attempting to LOAD a nonexistent program results in Error 67 : FILE NAME.

When LOAD is executed, any program or data currently in computer memory is scratched before the new program is loaded. Variables that were assigned in calculator (keyboard) mode are also scratched.

If you stored the program SPEEDS, you can now retrieve it. But first, you may want to scratch the contents of computer memory just to prove to yourself that LOAD really works. Execute SCRATCH and then LIST to confirm that the program is no longer in computer memory.

Now, execute:

```
LOAD "SPEEDS.DRIVE0" or LOAD "SPEEDS:D700" or LOAD "SPEEDS"
```

Another way of loading the program is to assign the file specifier to a string variable:

```
F$= "SPEEDS.DRIVE0"
LOAD F$
```

The red pilot light on Drive 0 will light up while the program is being loaded. When the light goes off, LIST the program to confirm that it is in computer memory.

If you used a defined string expression such as F$ to load the program, the string definition was scratched when the program was loaded.

Chaining Programs

The CHAIN statement allows you to load a stored program into computer memory from a running program. When CHAIN is executed in a program:

- The current BASIC program and any data in computer memory are scratched. Specified data may be preserved between two programs by including a COM statement in both programs. Binary programs are not scratched when CHAIN is executed.

- The program specified in the CHAIN statement is immediately loaded into computer memory from mass storage.

- The newly-loaded program is executed automatically.

Note that, unlike the LOAD command, CHAIN is programmable. The proper form for the statement is:

```
CHAIN "file specifier"
```

The COM statement is used to preserve variable definitions between programs. All variables not included in the COM statement are scratched when the chained program is loaded.

The form of the COM statement is:

```
COM item [ , item...]
```

Refer to the discussion of COM in your computer owner's manual for additional information.

COM statements in both the initial and the chained program must agree in the number and type of variable. Particular care must be taken in preserving arrays that the option bases of the two programs agree.

An important function of chaining is that it enables you to execute a program too large for computer memory by separating the program into two or more parts. While the two programs that follow are relatively small, they provide an example of using CHAIN and COM. The first program computes yearly earnings for a company from quarterly earnings over a ten year period from 1970 through 1979. The EARNINGS program then chains to a program that draws a bar graph of the yearly earnings.

First, enter and store the program to draw the bar graph.

```
10 BEEP
20 OPTION BASE 1
30 GCLEAR
40 COM INTEGER Y(10), REAL E(5,
10)
50 SCALE Y(1)-3,Y(10)+1,-30000,
100000
60 XAXIS 0,1,Y(1),Y(10)+1
70 YAXIS Y(1),10000,0,100000
80 LDIR 90
90 FOR I=1 TO 10
100 MOVE Y(I)+.6,-30000
110 LABEL VAL$(Y(I))
120 NEXT I
130 MOVE Y(1)-2, 1000
140 LABEL  "EARNINGS-THOUSANDS"
150 LDIR 0
160 FOR N=10000 TO 90000 STEP 2
0000
170 MOVE Y(1)-1, N-3000
180 LABEL VAL$(N/1000)
190 NEXT N
200 FOR I=1 TO 10
210 MOVE Y(I),0
220 DRAW Y(I),E(5,I) @ DRAW Y(I)
+1,E(5,I) @ DRAW Y(I)+1,0
230 NEXT I
240 END
```

Preserves specified variables.

Establishes scaling factor, draws axes.

Labels X-axis.

Labels Y-axis.

Draws bar graph.

Now, store the program BARGRAPH into mass storage.

```
STORE "BARGRAPH.DRIVE0"
```

Next, execute SCRATCH, and enter the program for computing the yearly earnings:

```
10 OPTION BASE 1
20 COM INTEGER Y(10), REAL E(5,
10)
30 FOR I=1 TO 10
40 Y(I)= 1969 +I
50 DISP "ENTER QUARTERLY EARNIN
GS FOR";Y(I)
60 INPUT E(1,I),E(2,I),E(3,I),E
(4,I)
70 E(5,I)=E(1,I)+E(2,I)+E(3,I)+
E(4,I)
80 NEXT I
90 CHAIN "BARGRAPH.DRIVE0"
100 END
```

Preserves specified variables.

Computes yearly earnings

Loads BARGRAPH.

If you'd like to run the set of programs more than once, be certain to store EARNINGS now since it will be scratched when statement 90 is executed.

Now, execute EARNINGS. You will be asked to enter quarterly earnings for years 1970 to 1979. Enter any values you like, but keep in mind that the Y-axis for the bar graph runs from $0 to $100,000.

When you push (END LINE) after the last data entry, you will hear a beep as statement 10 in BARGRAPH is executed, and the bar graph will be drawn on the CRT. When program execution is completed, you may list the current program in memory if you'd like.

Storing and Retrieving Binary Programs

Some of the programs in the application pacs are binary programs. They function like a ROM, except that they are loaded from mass storage. The statement that accomplishes loading of binary programs is LOADBIN. The statement has the form:

```
LOADBIN "file specifier"
```

LOADBIN loads a binary program without altering existing data or programs in computer memory. Only one binary program can be in memory at a time.

If a binary routine is to be added to a BASIC program, you must first LOAD the main program and then add the binary program using LOADBIN. If you retrieve the binary program first, it will be scratched when the main program is loaded.

In order to edit a program that uses a binary routine, the binary program must be present in computer memory.

Binary programs are stored using the statement STOREBIN, which has the form:

```
STOREBIN "file specifier"
```

Translating Tape-Based Programs to Disc-Based Programs

Any programs written without the Mass Storage ROM in place that access the HP-85 internal tape unit are specific to the internal tape unit (tape-based) and cannot, as written, utilize a disc drive system. When the Mass Storage ROM is installed, those programs will continue to execute as they did before, regardless of the nature of the default mass storage medium.

For instance, suppose you have a program stored on tape that was written without the Mass Storage ROM installed. The program reads a data file, performs a number of calculations, creates a new data file, and then prints results of the calculations onto the new file. When this program is executed with the Mass Storage ROM in place, the program will continue to read, create, and write tape-based data files, even if the default mass storage medium is a disc.

Programs written without the Mass Storage ROM installed must be translated before they can utilize a disc system. After a program loaded from the tape system is translated, the program is compatible with the requirements of the Mass Storage ROM.

A tape-based program is translated by loading the program into computer memory and then executing the TRANSLATE command, which has the form:

```
TRANSLATE
```

A beep signifies that the translation is completed.

If the tape-based program described previously was translated and then executed, the program would read the appropriate data file from the default mass storage device, perform the computations, and store the results in a data file created on the default mass storage device. If the data file being read had been stored initially on tape, it would be necessary to COPY it onto the default mass storage device before running the program.

The translated program can be stored onto a disc simply by executing STORE with an appropriate file specifier.

Section 5

Storing and Retrieving Data

The discussion of file types in section 3 pointed out that mass storage enables you to create and use five different types of files, one of which is the data file. This section covers the operations necessary to store, retrieve, and update data using mass storage. The five operations discussed in this section, all of which are essential in storing and retrieving data, are:

- Creating data files.
- Opening a previously created data file.
- Storing data.
- Retrieving data.
- Closing the data file.

There are two methods for accessing data files: serial access and random access. Serial access stores data sequentially, and is useful when the complete data list is to be stored and retrieved as a unit. Random access allows you to access portions of the data. Both types of files are created, opened, and closed in the same way. However, data is stored and retrieved somewhat differently, so storing and retrieving will be discussed separately for serial and random access.

Files created in mass storage consist of one or more records. The size of the records may vary to accommodate the storage requirements of the data. Before covering how to create data files of different sizes, we will first discuss file structure and storage requirements.

File Rec

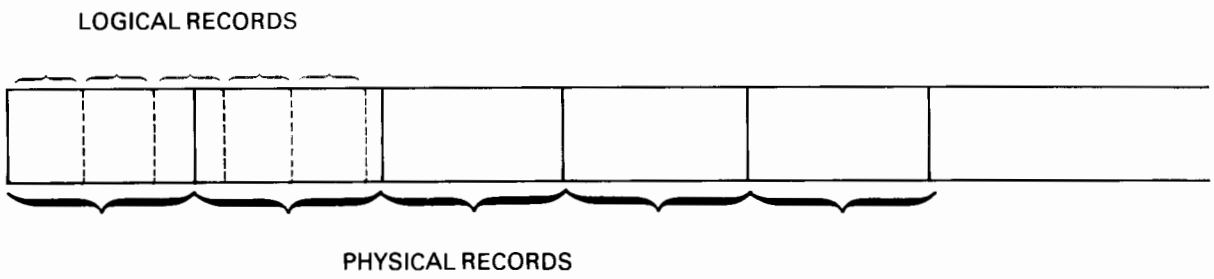
When a data file is created in mass storage, the size of the file is set by specifying the number of records in the file and the length of the records. A record is the smallest addressable location on a mass storage medium such as a disc or tape. Record length is specified in bytes, and all records in a particular file are the same length.

Two types of records are available: physical and logical. The two types of records make it possible to match the structure of data to the file in which it is stored, thus using storage space most efficiently.

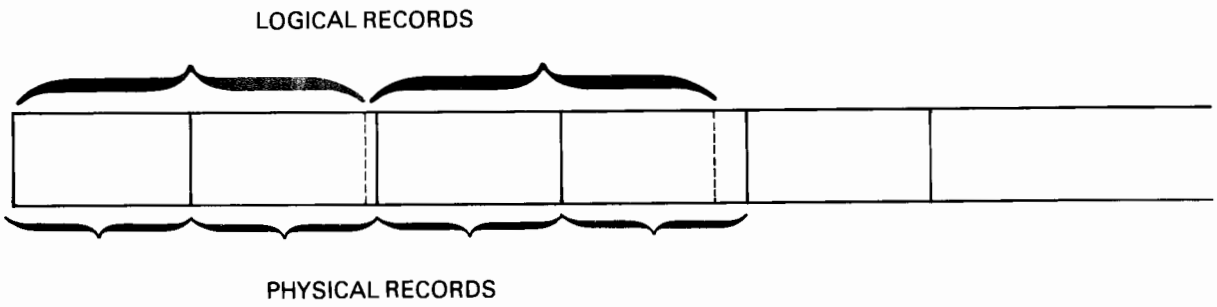
Physical Records — Physical records are always 256 bytes in length and are set up automatically when a program file or data file is created. All files begin at a new physical record. The 256 byte physical record is the smallest addressable storage unit unless a different size addressable unit, called a logical record, is established.

Logical records — Logical records are specified for a file when an addressable unit of length other than 256 bytes is desired. The file will still begin at the start of a physical record; within the file, however, the dividers between physical records are ignored and a logical record may straddle two or more physical records. When a data file is created without specifying logical records, the automatically-created physical records become logical records.

The following diagrams illustrate two files consisting of logical records. The first file contains five records, each 100 bytes long. Note that the file utilizes two physical records and that there are 12 bytes of unusable space, since any new file must begin at a new physical record. The divider between the two physical records is ignored.



The next diagram illustrates a file consisting of two 500-byte logical records. The divisions between physical records within the logical records are ignored; however, 24 bytes of space are rendered unusable, since any new file must start at a new physical record.



Storage Requirements

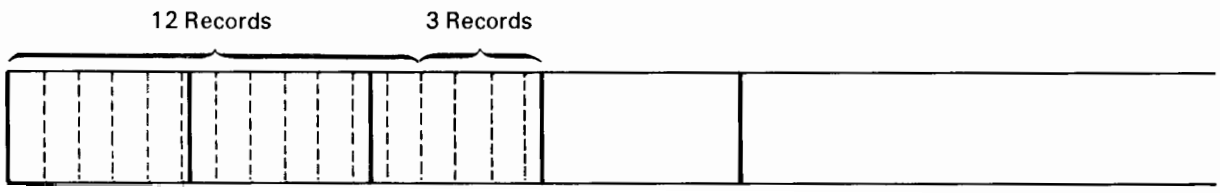
File and record sizes should be specified with the space requirements of the data in mind. The following chart describes the amount of space necessary to store numeric and string data.

Type	Numbers	Strings
Single variable	8 bytes per number	1 byte per character + 3 bytes per string + 3 bytes each time the string crosses into a new logical record.
Array variable	8 bytes × the dimensioned number of elements	Not available.

You can use these space requirements to set up files to match your data. For instance, suppose you would like to create a file that will store the last and first names, social security number, and salary of a dozen employees. You would like each employee's information in a separate record.

Item	Type of data	bytes
last name	12-character string	3 + 12 = 15
first name	10-character string	3 + 10 = 13
social security #	11-character string	3 + 11 = 14
salary	numeric	8
		50

A file can then be created consisting of 12, 50-byte records. When logical records are created, any otherwise wasted space (in this case, 168 bytes) is also allocated into logical records, if possible. The 168 bytes form an additional 3 records added to the file automatically, with 18 unusable bytes.



Creating Data Files

The CREATE statement allocates space on a mass storage medium for the data file. The statement has the form:

```
CREATE "file specifier", number of records [, record length]
```

The number of records specifies how many records the file will contain, and must be an integer from 1 through 32,767. The recond length is the number of bytes in each record, and must be an integer from 4 through 32,767. The default value for the record length is 256 bytes, the size of a physical record. The total number of bytes, obtained by multiplying the number of records by the record length, must not exceed the storage capacity of the mass storage medium.

The following statement creates a data file named EMPLOYEES for storing the identification and salary information for the 12 employees, as discussed above.

```
30 CREATE "EMPLOYEES.DRIVE0",12,
50
```

Creates a data file with 12 logical records of 50 bytes each. (Actually, 15 records will be set up, as discussed in Logical Records, page 32.)

Since the information for each employee is stored in its own record, it can be accessed and updated separately from the data for other employees. If you create this file on ".DRIVE0" and then execute CAT, the file will be listed.

```
CAT ".DRIVE0"
Volume:DRIVE0
Name          Type    Bytes    Recs
SPEEDS        PROG      256       3
EMPLOYEES     DATA       50      15
```

If it was preferable to always store and retrieve the information for all employees at once, a file containing one record could be set up.

```
30 CREATE "EMPLOYEES.DRIVE0", 1,
600
```

Creates a data file of one 600-byte record.

Opening a Data File

Once a data file has been created, it must be opened before it can be accessed to store data. Opening a data file assigns to it a buffer through which data flows from the computer to the mass storage medium, and from the mass storage medium to the computer. (See figure 2.) The ASSIGN # statement accomplishes this. The statement is executable in both program and keyboard modes.

```
ASSIGN #buffer number TO "file specifier"
```

The file specifier must correspond to a previously created data file. The buffer number is a number that rounds to an integer from 1 to 10. Once a buffer has been assigned to a file, that buffer remains assigned to the file until the same buffer number is assigned to a different file, or until the file is closed.

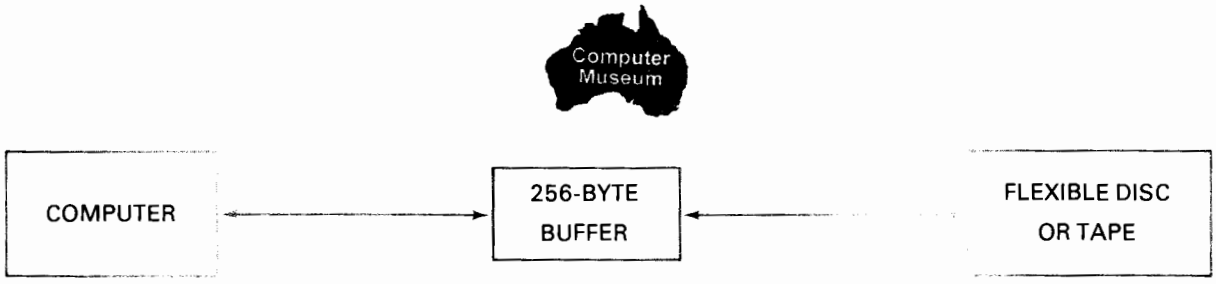


Figure 2. Flow of data through a buffer.

Buffers — A buffer is a 256-byte location in computer memory that is allocated whenever a file is opened. The purpose of the buffer is to reduce wear of the mass storage medium by accumulating data being transferred between the computer and a mass storage medium. The accumulated data is transferred to its final destination whenever one of the following conditions is met.

- The buffer is full.
- The buffer is reassigned to a different file.
- PAUSE, STOP or END is executed.
- Program execution is interrupted (unless program execution is halted by a disc error).
- The file is closed.
- Another logical record in a random file is accessed.
- A PRINT # statement is executed from the keyboard.

The EMPLOYEES file could be opened by executing:

```
40 ASSIGN #1 TO "EMPLOYEES.DRIVE
0"
```

Closing a Data File

When you've completed a data transfer to or from a file, you should close the file. The ASSIGN # statement accomplishes this, and can be executed in program or keyboard mode.

```
ASSIGN #buffer number TO *
```

The buffer number must agree with the buffer number assigned to the file when it was opened. For instance, to close EMPLOYEES previously opened in statement 40, above, execute:

```
200 ASSIGN #1 TO *
```

When a buffer is closed, any data in it is transferred to the final destination (the computer or mass storage medium). If a program error causes a halt while data is in the buffer enroute to mass storage, all the data in the buffer will be printed to the file. The file remains open and thus does not need to be reopened before program execution is continued.

If a disc error causes a halt during program execution, data in a buffer enroute to mass storage is lost unless the file is closed from the keyboard. When the file is closed, the data will be transferred to mass storage.

Serial Access

Serial access is used when a quantity of data is to be stored and retrieved sequentially and updated as one unit. The entire file itself becomes the smallest addressable unit of storage. This is true even if the file being accessed consists of more than one logical record. In serial access data is stored and retrieved without regard to record divisions.

Serial Printing

Data is stored into a file serially using the serial PRINT # statement, which has the form:

```
PRINT #buffer number ; print # list
```

The buffer number must have been previously assigned to a data file. The print # list itemizes the data you wish to store, and may include numbers, numeric variables, string variables, and array names. Items in the print # list are separated by commas. Data items are placed into the file according to the position ofthe file printer.

Pointers — When a file is opened, the file pointer is place at the beginning of the file, and any data items serially printed to or read from the file will access the beginning of the file. The pointer moves through the file sequentially. When an entire print # list has been recorded, the pointer remains at the end of the recorded data, and an end-of-file marker indicates the position of the last recorded data. Execution of a subsequent PRINT # statement records the new print # list at the end of recorded data and moves the end of file marker to the end of the newly recorded data. The pointer will continue to move sequentially through the file until the file is closed or reassigned with an ASSIGN # statement.

The movement of the file pointer and end-of-file marker influence the way in which serial files may be updated. If, after entering a long list of data items serially, the pointer is returned to the beginning of the file using an ASSIGN # statement, a new serial PRINT # statement will record new data items over the old ones. Because an end-of-file marker is placed at the end of the new data items, the entire old data list is lost.

The following sample program uses serial access to store check register data for the PDQ Music Company. The company opens a new file each day, and records the company to which a check has been written as string C$ and the amount of the check as numeric variable A.

```
10 CREATE "NOV5CHECKS.DRIVE0",4          Creates file of 4, 256-byte records.
20 ASSIGN #1 TO "NOV5CHECKS"             Opens file.
30 DISP "COMPANY NAME";
40 INPUT C$
50 IF C$="NO MORE TODAY" THEN 1
00
60 DISP "AMOUNT OF CHECK";
70 INPUT A
80 PRINT #1;C$,A                         Prints company name and amount of
                                         check to the file serially.
90 GOTO 30
100 PRINT #1;C$
110 ASSIGN #1 TO *                       Closes file.
120 END
```

When the program is run, it prompts for company name and amount of the check until "NO MORE TODAY" is input in response to the company name prompt. If file capacity is exceeded before program execution ends, Error 72 on line 80 : RECORD announces an attempt to print at the end of the file.

> **Note:** When a string printed to a file serially crosses from one record to another, an additional three bytes are needed for the string "header," which identifies the portion of the string contained in the new record.

Reading Files Serially

Data that has been stored onto a mass storage medium must be retrieved, or read, back into computer memory before it can be used. Reading data from a file transfers a copy of the data through a buffer into the computer.

When data is retrieved serially, the entire file contents is accessed sequentially, ignoring any record divisions. Data stored both serially and randomly can be retrieved serially. Serial reading is accomplished by the serial READ # statement, which has the form:

```
READ #buffer number ; read# list
```

The buffer number must match the number previously assigned to the file with an ASSIGN # statement. The read list need not exactly match the print # list used to store the data. However, data items being read must agree in type (string versus numeric) with the contents of the file. Numeric data need not agree in precision (REAL, INTEGER, SHORT). The number will be converted to the precision of the read variable as long as the read precision is less than the print precision. If the read precision exceeds the precision of the stored number, the number is read to the same precision with which it was stored. For example, a printed SHORT number will be converted to the INTEGER precision specified in a read list; however, if you attempt to read the same SHORT precision number with REAL precision, the number will actually be read with SHORT precision.

In reading serial files, the pointer moves through the file sequentially, much like with serial printing. The pointer is moved to the beginning of the file whenever the file is opened, or if an ASSIGN # statement for that file is re-executed. Since a serial PRINT # statement leaves the file pointer at the end of the last recorded data, you must move the pointer to the beginning of the file before reading stored data.

If you used the program on page 37 to create a data file for a check register, you can use the following program to read the file, print its contents, and sum the day's check payments.

```
10 ASSIGN #1 TO "NOV5CHECKS.DRIV
EO"                                   Opens data file.
20 S=0
30 READ #1;N$                         Reads company name.
40 IF N$="NO MORE TODAY" THEN 9
0
50 READ #1;M                          Reads amount of check.
60 PRINT USING 100;N$,M
70 S=S+M
80 GOTO 30
90 PRINT USING 110;S
100 IMAGE 18A,2X,5D.DD
110 IMAGE /,3X,"TOTAL =",10X,5D.
DD
120 ASSIGN #1 TO *                    Closes the data file.
130 END
```

In the above program, the file pointer moves through the data file as each READ # statement is executed repeatedly. If statement 40 were omitted, the READ # statement in line 50 would eventually encounter an end-of-file marker, generating an error.

Random Access

When you wish to print to, read from, or update a portion of a data file, random access enables you to do so. The random PRINT # and READ # statements are designed to access individual records of a data file. Remember that a record is the smallest addressable unit of mass storage and can be as small as four bytes.

Random Printing

The random PRINT # statement has the form:

```
PRINT #buffer number , record number [ ;print # list]
```

The buffer number must match the buffer assigned to the file by an ASSIGN # statement. The record number must be less than or equal to the total number of logical records in the file. The print # list contains all the items to be printed to the record, separated by commas.

The random PRINT # statement operates somewhat differently from the serial PRINT # statement:

- Because random printing accesses a specified record, the record number must be part of the statement.
- When a random PRINT # statement is executed, the file pointer is moved automatically to the beginning of the specified record. Thus, all items printed to a particular record must appear in one random PRINT # statement.
- In random printing, the contents of the file buffer is transferred to its destination each time another record is accessed.
- Record divisions are not ignored in random access operations. Attempts to print to a file when the file pointer is at the end of the specified record results in an error.
- The file pointer may be moved to the beginning of a random record by executing a random PRINT # statement without a print # list. For example:

```
60 ASSIGN #1 TO "DATA.DRIVE0"
70 PRINT #1,3
```

Moves the pointer to the beginning of record 3 of file DATA.

In random access, the print # list must not exceed the storage capacity of the logical record. Error 69 : RANDOM OVF, or Error 72 : RECORD indicates that the print # list has exceeded the capacity of the record.

The following program creates a file for storing and retrieving a check register using random access. Each of the 20 records contains the name of the company to which the check is written and the amount of the check. The string "XXXXXX" and the numeric variable 0 are stored into otherwise empty records. The program prints (or displays) the contents of each record as the checks are entered.

```
10 CREATE "DEC5CHECKS.DRIVE0",2
0,30                                    Creates a 20-record file.
20 ASSIGN #1 TO "DEC5CHECKS.DRI
VE0"
30 FOR I=1 TO 20
40 DISP "COMPANY NAME";
50 INPUT C$
60 IF C$="NO MORE TODAY" THEN 1
20
70 DISP "AMOUNT OF CHECK";
80 INPUT A
90 PRINT USING 150;I,C$,A              Prints (displays) check data.
100 PRINT #1,I; C$,A                   Prints C$ and A to record #I.
110 NEXT I
120 FOR R=I TO 20
130 PRINT #1,R; "XXXXX",0              Prints to all unused records.
140 NEXT R
150 IMAGE 2*,X,18A,2X,4D.DD
150 ASSIGN #1 TO *                     Closes file.
170 END
```

The program uses a FOR-NEXT loop to increment the record number. Note that, unlike the serial access version of this program, this program does not actually store "NO MORE TODAY" to mark the end of the data. Random access would allow you to attempt to retrieve data beyond that entry, since you may move the file pointer to the beginning of any existing random record.

Reading Files Randomly

Random access reading is accomplished with the random read statement, which has the form:

```
READ #buffer number , record number [; read # list]
```

The differences between the random read statement and serial read statement are analogous to the differences between the two types of PRINT # statements:

- The statement must include the record number you wish to access.
- The file pointer automatically moves to the beginning of the specified logical record.
- Logical record divisions are not ignored. An attempt to read past the end of a logical record generates Error 72 : RECORD.
- The file pointer can be moved to the beginning of the record by executing the statement without a read # list.

As with serial reading, the read list must agree in data type (numeric versus string) with the stored data; however, number precision need not agree. (Refer to page 37, Reading Files Serially, for further information.)

The following program allows you to correct any of the entries to the check register DEC5CHECKS and to add additional checks. The program asks whether there are any changes and then prompts for the record number in which the correction is to be made. To make additions, merely specify a previously unused record and replace its current contents, "XXXXX" and 0, with the new data.

After accessing the file randomly for updating, the program then uses serial access to access the entire register, sum the checks, and print the contents.

```
10 ASSIGN #1 TO "DEC5CHECKS"
20 DISP "ANY CHANGES";
30 INPUT J$
40 IF J$="NO" THEN 130
50 DISP "ENTER RECORD # OF ENTR
Y TO BE CHANGED"
60 INPUT I
70 READ #1,I;C$,A
80 DISP C$,A
90 DISP "ENTER CORRECT INFORMAT
ION"
100 INPUT D$,B
110 PRINT #1,I;D$,B
120 GOTO 20
130 ASSIGN #1 TO "DEC5CHECKS"
140 S=0
150 FOR I=1 TO 20
160 READ #1;C$,A
170 PRINT USING 220; I,C$,A
180 S=S+A
190 NEXT I
200 PRINT "TOTAL = ";S
210 ASSIGN #1 TO *
220 IMAGE 2*,X,10A,2X,4D.DD
230 END
```

Opens data file.

Enters record to be updated.
Reads contents of record.

Enters corrections.
Prints corrections to data file.

Moves pointer to beginning of file.

Reads file serially.

Sums check amounts.

Closes data file.

Note that statement 130 is necessary to move the pointer to the beginning of the file. Otherwise, the serial read would start at the last position of the pointer, the end of record I in statement 110.

Storing and Retrieving Arrays

Entire arrays can be stored and retrieved using an array addressing format with the serial or random PRINT # and READ # statements. The proper array addressing formats for one-dimensional and two-dimensional arrays are as follows:

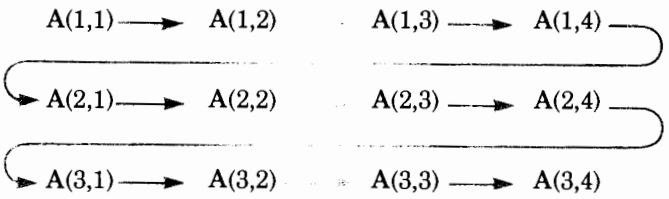
one-dimensional array	*array name* [ ]
two-dimensional array	*array name* [,]

Examples:

```
READ #1; B( )
PRINT #2,4;F(,)
```

Reads one-dimensional array B serially.
Stores two-dimensional array F into record 4 of specified file.

In the case of two-dimensional arrays, the array elements are retrieved item by item without regard to dimensionality, with the second subscript varying more rapidly, i.e., by rows.



Array elements of this 3 × 4 array are retrieved by rows.

Since array elements are stored on mass storage linearly, they may be retrieved with or without an array format. In the case of a two-dimensional array, any combination of A(I,J) dimensions may be used that accesses the desired number of elements. For instance, a 3 × 4 array stored in a file named ARRAY might be retrieved by the following statements. (ARRAY has been assigned buffer #1.)

```
DIM B(3,4)      DIM B(4,3)      DIM B(6,2)      DIM B(12)
READ #1; B(,)   READ #1; B(,)   READ #1; B(,)   READ #1; B()
```

If the array specified in the READ # statement has fewer elements than the stored array, only those elements allowed by the READ # array will be retrieved.

The following program stores temperature data gathered by an instrumented aircraft. The plane takes six temperature readings along each leg of its flight. The four legs cover the same route at altitudes 5000, 10,000, 15,000, and 20,000 feet. Data for each leg is entered into a separate record, so that it can be inspected and updated, if necessary.

```
10 MASS STORAGE IS ":DRIVE0"
20 CREATE "TEMPS",4,48
30 OPTION BASE 1
40 DIM T(6)
50 ASSIGN #1 TO "TEMPS"
60 FOR I=1 TO 4
70 DISP "ENTER DATA FOR LEG";I
80 INPUT T(1),T(2),T(3),T(4),T(
),T(6)
90 PRINT #1,I; T()
100 NEXT I
110 ASSIGN #1 TO *
120 END
```

Sets default mass storage device.
Creates data file.

Opens file.

Prints T(1) through T(6) to record I.

Closes file.

Analysis of the data involves averaging temperatures over height at each of the four locations and computing the vertical temperature differences between points. Data handling is facilitated in this situation by reading in the four 1×6 arrays as one 4×6 array.

```
10 ASSIGN #1 TO "TEMPS.DRIVE0"
20 OPTION BASE 1
30 DIM T(4,6),G(3,6)
40 READ #1;T(,)
50 FOR J=1 TO 6
60 A=(T(1,J)+T(2,J)+T(3,J)+T(4,
J))/4
70 PRINT "AVE. TEMP. FOR LOC.";J
;"=";A
80 FOR I=1 TO 3
90 G(I,J)=T(I+1,J)-T(I,J)
100 NEXT I
110 NEXT J
120 PRINT
130 PRINT "*****TEMPERATURE GRAD
IENTS*****"
140 FOR I=3 TO 1 STEP -1
150 PRINT "BETWEEN";I*5000;"AND"
;(I+1)*5000;"FEET"
160 PRINT "LOC1 LOC2 LOC3 LOC4 L
OC5 LOC6"
170 PRINT USING 200 ; G(I,1),G(I
,2),G(I,3),G(I,4),G(I,5),G(I,6)
180 NEXT I
190 ASSIGN #1 TO *
200 IMAGE /,6(SDDD,X)
210 END
```

Opens data file.

Reads in data as 4×6 array.

Computes average location temperature.

Computes gradients (differences).

Closes data file.

The TEMPS file is read serially in statement 40. The effect is to perform the following rearrangement of the data.

Data as printed to file

Record 1	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)
Record 2	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)
Record 3	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)
Record 4	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)

Data as read from file

leg 1	T(1,1)	T(1,2)	T(1,3)	T(1,4)	T(1,5)	T(1,6)
leg 2	T(2,1)	T(2,2)	T(2,3)	T(2,4)	T(2,5)	T(2,6)
⋮						⋮
leg 4	T(4,1)					T(4,6)

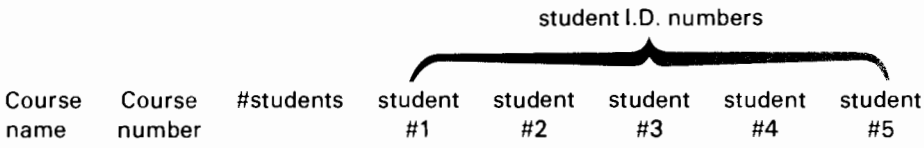
Updating Data Files

When data is stored on a mass storage medium, it must be read into computer memory before it can be updated. Since a record is the smallest addressable unit in mass storage, one practical way to change the contents of a data file is to read in the contents of an entire record, and later reinsert the updated record into the file. The sample program on page 00 contained an example of making corrections on a check register. The following program illustrates adding to, deleting from, and changing the contents of a 40-record file.

A small community college maintains a file, named COURSES, of its course registration data. The college offers 40 courses, numbered sequentially:

Subject	Course Numbers	Record Numbers
English	101 through 110	1 through 10
History	201 through 210	11 through 20
Math	301 through 310	21 through 30
Science	401 through 410	31 through 40

Each record is set up as follows. Course enrollment is limited to five students.



Each term before registration, the file is initialized by entering in course names and numbers, and setting enrollment data to 0. The following program processes student registration forms. The program prompts for the student I.D. number and then conducts the appropriate file operations for adding and dropping courses. The program assumes a previously initialized file named COURSES located on medium "DRIVE0".

```
10 OPTION BASE 1
20 DIM S(5)
30 DISP "ENTER STUDENT NUMBER"
40 INPUT N                                  Enters student I.D. number.
50 IF N=0 THEN 370
60 DISP "ADD,DROP, OR DONE"
70 INPUT 0$                                 Enters operation—add, drop, or done.
80 IF 0$="DONE" THEN 30
90 DISP "ENTER CLASS NUMBER"
100 INPUT D$                                Enters class number (101-110, 201-210,
                                            etc.)
110 I=VAL(D$[2,3])+10*(VAL(D$[1             Computes record # from course
,1])-1)                                     number.
120 ASSIGN #1 TO "COURSES.DRIVE             Opens data file.
0"
130 READ #1,I;C$,D$,E,S()                   Reads in contents of appropriate
                                            record.
```

```
140 IF O$="DROP" THEN 260
150 FOR J=1 TO 5
160 IF S(J)=N THEN 250

170 IF S(J)=0 THEN 200
180 NEXT J
190 IF J=6 THEN 240

200 S(J)=N
210 E=E+1
220 PRINT #1,I;C$,D$,E,S()
230 GOTO 60
240 DISP "CLASS FULL" @ GOTO 60
250 DISP "STUDENT ALREADY ENROLL
ED" @ GOTO 60
260 FOR J=1 TO 5
270 IF S(J)=N THEN 300
280 NEXT J
290 IF J=6 THEN 360

300 E=E-1
310 FOR K=J TO 4
320 S(K)=S(K+1) @ S(K+1)=0

330 NEXT K
340 PRINT #1,I;C$,D$,E,S()
350 GOTO 60
360 DISP "STUDENT WASN'T ENROLLE
D" @ GOTO 60
370 ASSIGN #1 TO *
380 END
```

Tests for student already being enrolled.
Tests for opening in the class.

Statement is executed when class is full.
Enrolls student.
Updates enrollment number.
Prints updated record to file.

If student is dropping the course, this loop searches to see whether she is actually enrolled.
Statement 290 executed if student wasn't enrolled.
Updates enrollment.

This loop removes student's I.D. number from the class list.

Closes data file.

Data files should be designed so that information requiring frequent updating can be readily accessed and altered. In the previous program, statement 110 directly converts the course number into a record number, enabling the program to access a course's record without searching through the entire file. If the organization of the data didn't lend itself to direct computation of record number, it would have been necessary to search for the appropriate course. In such cases, it is more efficient to store a master list of courses in one file and maintain a separate file for student enrollment. The master list could then include a pointer for each course, indicating in which record the student enrollment can be found.

Section 6

Storing and Retrieving Graphics

The Mass Storage ROM allows you to store the contents of the computer's graphics display onto a disc and to retrieve the display without re-executing the display-generating program. The operation of loading the stored display into the computer's graphics display leaves alpha mode and any program currently in computer memory intact. (Refer to the graphics section of your HP-83/85 owner's manual for a discussion of alphanumeric and graphics mode.)

The statements covered in this section create and access extended files, and are the only Mass Storage ROM statements to do so. Unlike data files, extended files are not opened and closed. An attempt to assign a buffer to an extended file generates `Error 68 : FILE TYPE`.

Storing a Graphics Display

The contents of the computer's graphics display can be stored onto a disc by executing the `STORE` statement. `GSTORE` may be executed from the keyboard or within a program. The statement automatically creates an appropriately-sized extended file; you cannot use a `CREATE` statement to create an extended file.

The proper form of the `GSTORE` statement is:

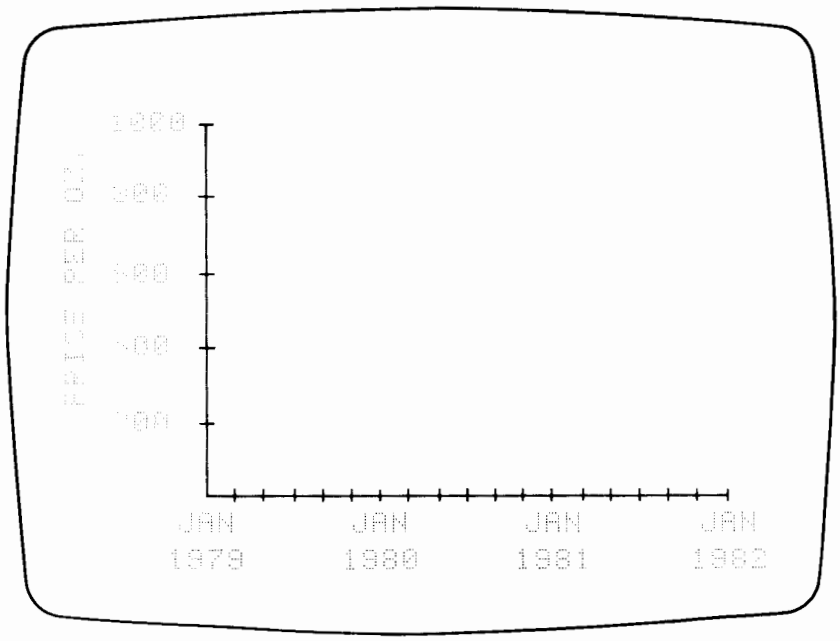
```
GSTORE "file specifier"
```

The statement stores a copy of the computer's graphics display into the file. Computer memory is unaffected, and the stored graphics display remains in the computer's graphics display until execution of `GCLEAR` or `GLOAD`.

Extended files are useful for storing the scaled and labeled X- and Y-axes of a graph for later use in plotting data. The following program generates a graphics display which will be retrieved for use later in this section. The program creates the X- and Y-axes for a graph of gold prices from January, 1979, to January, 1982. After the program is keyed in and executed, the graphics display it generates will be stored into an extended file named GOLD.

```
10 GCLEAR
20 SCALE -10,40,-400,1100
30 XAXIS 0,2,0,36
40 YAXIS 0,200,0,1000
50 LDIR 0
60 FOR I=0 TO 3
70 MOVE 12*I-2,-100 @ LABEL "JAN
"
80 MOVE 12*I-2,-200 @ LABEL VAL$
(1979+I)
90 NEXT I
100 FOR I=200 TO 1000 STEP 200
110 MOVE -6.5,I-25 @ LABEL VAL$(
I)
120 NEXT I
130 LDIR 90 @ MOVE -8,200 @ LABE
L "PRICE PER OZ."
140 END
```

Now, run the program to generate the X- and Y-axes for the gold prices plot:



Before you can execute GSTORE, you must return the computer to alpha mode. This is easily done by pressing any alphanumeric or display control key. Once you are in alpha mode, execute:

```
GSTORE "GOLD.DRIVE0"
```

Stores contents of the computer's graphics display into an extended file named GOLD.

The contents of the graphics display has now been copied into the file named GOLD located on the disc with volume label ":DRIVE0". We could have chosen to store the display as part of the program. Inserting the GSTORE statement after line 130 accomplishes this.

As in program storing, the contents of an extended file can be altered by executing GSTORE with the same file specifier and a different computer graphics display.

Retrieving a Graphics Display

Once a graphics display has been stored with a GSTORE statement, it can be retrieved by executing a GLOAD statement, either from the keyboard or within a program. The proper form of GLOAD is:

```
GLOAD "file specifier"
```

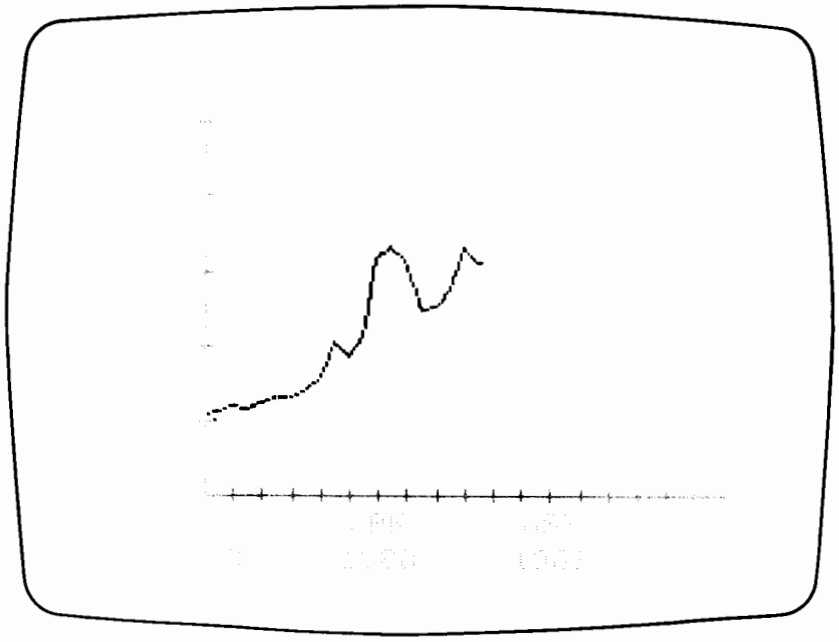
The file specifier must be the name of a previously-GSTOREd extended file.

Execution of GLOAD places a copy of the graphics display contained in the extended file into the computer's graphics display. The contents of the computer's graphics display at the time GLOAD is executed will be scratched as the stored display is retrieved. As GLOAD is executed, the computer automatically switches to graphics mode, and you can see the stored display appear on the CRT.

The following program graphs approximate gold prices at the beginning of each month from January, 1979, to September, 1980, using the axes stored in the GOLD extended file. Since data is not available for the entire X-axis time span, the program tests for a price of $0.00 to signal the end of the data list.

```
10 OPTION BASE 0
20 DIM P(2,11)
30 SCALE -10,40,-400,1100
40 GLOAD ":DRIVE0"                    Loads axes into graphics mode.
50 PENUP                              Lifts pen before plotting first point.
60 FOR I= 0 TO 2
70 FOR J=0 TO 11
80 READ P(I,J)
90 IF P(I,J)=0 THEN 150               Test for 0.00 price
100 PLOT I*12+J,P(I,J)                Plots price.
110 NEXT J
120 NEXT I
130 DATA 220,230,245,235,250,270
,270,290,320,420,380,430
140 DATA 635,665,635,505,510,560
,665,625,640,0
150 END
```

When the program is executed, the CRT switches to graphics mode at line 30. The axes is retrieved, and then the prices are plotted onto the graph. When the data has been plotted, the display will look like this:



You may store the graph onto mass storage by executing a GSTORE statement with either the "GOLD.DRIVE0" file specifier, to replace the empty axes, or with another file name. Once the graph has been stored, it can be updated as new data up to January, 1982, becomes available by GLOADing the graph and plotting additional points.

Notes

Section 7

Other File Manipulations

The Mass Storage ROM enables you to perform a variety of file manipulations in addition to the ones already covered. Section 7 covers the following additional file operations:

- Determining the type data of the next item in a data file.
- Copying files from one mass storage medium to another.
- Renaming files.
- Purging files.
- Packing files for more efficient use of mass storage space.
- Securing files.

Determining Data Types—The TYP Function

The TYP function allows you to determine the type of data of the next item in a data file. The function also allows you to determine whether the file pointer is at the end of the record or at the end of the file. The function has the form:

```
TYP [buffer number]
```

The buffer number must correspond to the buffer assigned to the file being accessed. The function returns an integer from 1 to 10 according to the following table.

Type Value	Data Type
1	Number
2	Full String
3	End-of-File
4	End-of-Record
8	Start of String
9	Middle of String
10	End of String

In using the TYP function, the pointer can be moved through the file in much the same way as it is moved in serial and random printing and reading. One difference is that record divisions are not ignored when the pointer is moved with serial statements.

We will use the TYP function to access data items file named AGES, which is organized into logical records as shown below:

record 1	record 2	record 3	record 4	record 5	record 6
Bill 30	Phyllis 31	Hank	Don 35 Plu	sorminus10	

Now, the following statements are executed from the keyboard:

```
ASSIGN 1 TO "AGES.DRIVE0"
READ #1,1
TYP(1)
 2
READ #1;N$

TYP(1)
 1
READ #1;A

TYPE(1)
 4
READ #1,3;N$

TYP(1)
 4
READ #1,4;N$,A

TYPE(1)
 8
READ #1,5
TYP(1)
10
READ #1,6
TYP(1)
 3
```

Opens AGES file.
Moves pointer to beginning of record 1.

Moves pointer past first item in record 1.

Moves pointer past 2nd item in record 1.

Moves pointer past first item in record 3.

Moves pointer past first two items in record 4.

Moves pointer to beginning of record 5.

Moves pointer to beginning of record 6.

Copying Files

Any file in mass storage not secured against copying can be copied to another mass storage medium. The COPY statement accomplishes this and adds the name of the copied file to the destination medium's file directory.

```
COPY "source file specifier" TO "destination file specifier"
```

The source file specifier corresponds to a file present on a mass storage medium. The destination file specifier may include the same or a different name. You cannot copy a file secured against copying (type 1 security). If you attempt to do so, no error is generated, but the secured file will not be copied to the destination medium.

Examples:

```
COPY "SPEEDS.DRIVE0" TO "SPEEDS.
DRIVE1"
```

Copies the file named SPEEDS on the disc with volume label ".DRIVE0" onto the disc having volume label ".DRIVE1".

```
COPY "QUAKE.DRIVE0" TO "EARTH:T"
```

Copies the file named QUAKE on disc ".DRIVE0" onto an HP-85 tape cartridge, naming the new file on the tape EARTH.

COPY can also be used to copy all the files on a mass storage medium to another medium. The COPYing process does not affect the original contents of the destination medium. The source medium's contents are simply added on.

```
COPY ".source volume label" TO ".destination volume label"
     ":source msus"             ":destination msus"
```

Both source and destination volume labels must have been previously assigned with a VOLUME IS statement.

One use of COPY is transferring the contents of several partially filled discs or tapes onto one medium. If duplicate names are encountered during copying, Error 63 : DUP NAME is generated, and the COPY operation terminates. All files copied up to the termination remain intact.

Files secured against copying (type 1 security) are not copied when the entire contents of one disc are copied to another disc. The secured file is simply ignored, and no error is generated.

If there is not enough space on the destination medium to hold all the files being copied, the copy operation terminates and Error 128 : FULL results when the available space is exhausted. Copying also terminates when the directory space on the destination storage medium is exhausted, generating Error 124 : FILES. Files copied before generation of the error remain intact.

Renaming Files

Any file, regardless of its type, can be given a new name using the RENAME statement:

```
RENAME "old file specifier" TO "new file name"
```

The old file specifier must correspond to a previously specified file. When the statement is executed, the name of the file as listed in the file directory is changed. The file itself is untouched. However, it must now be addressed using the new name.

Examples:

```
RENAME "AGES.DRIVE0" TO "BIRTHD
ATE"
RENAME "SORT:T" TO "SHELL"
```

Renames AGES on Drive 0 to BIRTHDATE.

Renames tape file SORT to SHELL.

Purging Files

The PURGE statement prevents further access to a file by removing the file name from the directory. The space that was occupied by a purged file becomes a NULL file and is available for future use.

```
PURGE "file specifier" [, purge code]
```

The file specifier must correspond to an existing file of any type—program, data, extended, or binary program. The purge code may be any number; however, any purge code other than zero is ignored.

When a file is purged without a purge code, the file name is removed from the file directory, and NULL is substituted for the type of file in the Type column of the directory. The NULL file is available for future use, and will be used when you store or create another file that fits into the available space.

When a purge code of 0 is included, the specified file and all files after it on the storage medium are purged. The directory does not create NULL files; the directory will contain a listing for only those files up to (and not including) the file specified in the PURGE statement.

The following catalog shows the results of purging a file without a purge code.

```
PURGE "ODDNUMBERS.DRIVE1"
CAT ".DRIVE1"
Volume:DRIVE1
Name        Type      Bytes     Recs
SPEEDS      PROG        256        3
DATA1       DATA         80        3
            NULL        256        2
EVEN1       PROG        256        1
TESTS       DATA        256        1
SERIAL      PROG        256        4
```

Now, two files from the end of the directory will be purged.

```
PURGE "TESTS.DRIVE1",0
CAT ":DRIVE1"
Volume:DRIVE1
Name          Type      Bytes      Recs
SPEEDS        PROG        256         3
DATA1         DATA         80         3
              NULL        256         2
EVEN1         PROG        256         1
```

Packing Files

The PACK statement fills in NULL file gaps created when files are purged without a 0 purge code. PACK cannot be used with files on an internal tape unit.

```
PACK[ ":volume label" ]
     ":msus"
```

Example:

```
PACK ":DRIVE1"
CAT ":DRIVE1"
Volume:DRIVE1
Volume        Type     Bytes     Recs
SPEEDS        PROG       256        3
DATA1         DATA        80        3
EVEN1         PROG       256        1
```

File Security

File security is used to prevent program files from being listed, duplicated and overwritten, and to prevent data files from being copied or changed. You may also remove a file name from the directory listing without creating a NULL file; the file can still be accessed by anyone who knows its name.

Securing Files

The SECURE command places various levels of security on files. SECURE is programmable, and can also be executed from the keyboard. None of the levels of file security prevent a file from being purged.

```
SECURE "file specifier", security code, security type
```

The file specifier must refer to a file already existing in mass storage. The security code may be either a quoted string or a string expression that becomes associated with the file for security levels 0 and 1. Only the first two characters of the security code string are actually used. If the string has only one character, the second character is a blank.

The security type is an integer from 0 through 3, and represents various levels of security according to the following table:

Security type	File is secured against:
0 (program files only)	LIST, PLIST, and editing.
1 (program files only)	LIST, PLIST, editing and duplication (STORE).
2 (program, data, or extended)	STORE (overwriting), PRINT#, STOREBIN, GSTORE.
3 (program, data, or extended)	CAT (a blank appears where the name should be).

You may secure a file with more than one security type by executing more than one SECURE statement for the same file. However, you may not secure a file for both types 0 and 1 security at the same time.

The security types have the following effects:

Type 0 — Type 0 is used for program files only. The program is protected against LISTing, PLISTing, and editing. The file name still appears in the directory and the program can be loaded. However, attempts to list the program generate an error.

Type 1 — Type 1 incorporates all the features of type 0 but adds protection against duplication of the program. Any attempt to store the program in another file will generate an error.

Type 2 — Type 2 prevents a program or data file from being overwritten. Any attempt to store or print to the file generates Error 60: WRITE PROTECT. However, the file contents can be duplicated into another file.

Type 3 — Type 3 security removes the file name from the directory. Anyone knowing the name, however, can still access the file.

Regardless of the type(s) of file security specified, a file can always be purged.

Examples:

```
SECURE "EVEN1.DRIVE0","NOLIST",
0
SECURE "EVEN1.DRIVE1","CANTSTORE
",2
```

Establishes file security type 0. "NO" is the security code.

Establishes file security type 2. The security code is ignored. (The file can be unsecured with any security code.)

Type 3 security has the following effect on the file directory:

```
SECURE "SPEEDS.DRIVE1","DONTCAT"
,3
CAT "SPEEDS.DRIVE1"
Name          Type    Bytes     Recs
              PROG      256        3
DATA1         DATA       80        3
EVEN1         PROG      256        1
```

Secures SPEEDS with type 3 security.

The name SPEEDS is removed from the file directory.

Removing File Security

The UNSECURE statement cancels a previously specified file security. The statement is not programmable.

```
UNSECURE "file specifier" , security code , security type
```

In removing a particular type of security, the security type (0 through 3) must correspond to the security type you wish to eliminate, previously specified with a SECURE statement. The security code must match the code established by the SECURE statements for types 0 and 1 security. Any two characters may be used for types 2 and 3.

Examples:

```
UNSECURE "EVEN1.DRIVE1", "NO",0
```

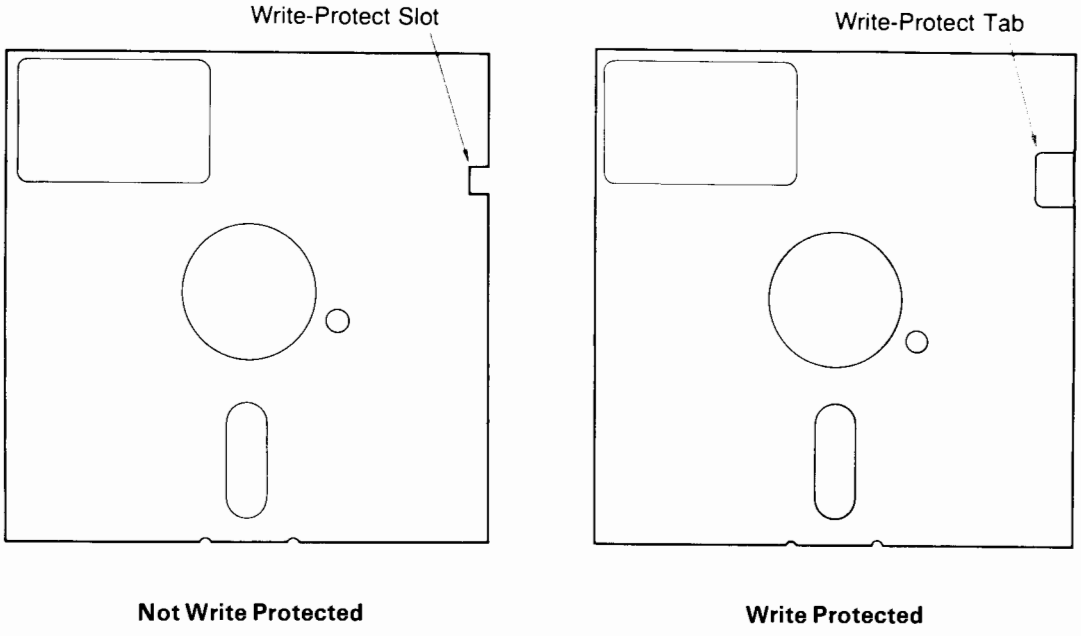
Removes previously established type 0 security. The security code matches the SECURE statement.

```
UNSECURE "SPEEDS.DRIVE1","OK",3
```

Restores the file name SPEEDS to the file directory. The security code need not match the SECURE statement.

Disc Write Protection

You can prevent any write operation onto a flexible disc by covering the write-protect slot with a write-protect tab (provided with the discs) as shown in the illustration below.



The procedure for write-protecting other discs may not be the same as the above. Refer to the documentation for your system for write protection information.

The write-protect procedure prevents you from writing any information onto the disc. The disc can, however, be read normally. To write on a protected disc, you must reverse the write-protect procedure.

Tape cartridges used with the HP-85 internal tape unit may be write-protected by sliding the RECORD slide tab to the left before inserting the cartridge into the HP-85. Write-protection is removed by sliding the tab to the right.

Notes

Section 8

Data Verification and Error Processing

Verifying Data

The CHECK READ statement can be used to verify that data printed to a data file located on a disc has been properly recorded onto the disc. When CHECK READ is activated, an immediate read # is performed on any data printed to a specified file. If the two lists do not match, indicating failure of the storage medium (disc) itself, the ROM will return Error 127 : READ VFY (read verify). CHECK READ cannot be used for data files located on an internal tape unit.

```
CHECK READ # buffer number
```

The buffer number specified must match the buffer assigned to the data file.

CHECK READ errors are rare. If you should ever encounter one, you may wish to compare your PRINT # statement again, since the failure which generated the error may have been momentary. If you obtain another CHECK READ error, it is likely that the disc has failed.

CHECK READ is turned off by the CHECK READ OFF statement:

```
CHECK READ OFF #buffer number
```

Examples:

```
CHECK READ #1
CHECK READ OFF #1
```

Verifies all data printed to buffer #1.
Turns off CHECK READ at buffer #1.

Error Processing

Several statements are available for determining whether an error in a running program has been generated by a ROM or by an interface.

ROM-Issued Errors

When you receive any error message, you may use the ERROM function to determine whether the error was issued by one of the ROMs.

```
ERROM
```

ERROM returns the number of the ROM that issued the error. If the error was issued by the computer rather than by a ROM, the function returns 0. The Mass Storage ROM number is 208.

ERROM can be used with an ON ERROR statement to direct program flow. The program below displays a message when a Mass Storage ROM error occurs.

```
10 ON ERROR GOTO 200
20 CREATE "FILE,DRIVE7",1
30 DISP "THE FILE IS CREATED"
.
.
.
200 OFF ERROR
210 IF ERROM =208 THEN 300
.
.
300 DISP "MASS STORAGE ROM ERROR"
.
.
310 END
```

Refer to your computer owner's manual for more information regarding error processing.

Interface Module Errors

When an illegal operation elicits an error from an interface, you can determine the select code of the interface at which the error originated with the ERRSC function. ERRSC returns the select code of the interface.

```
ERRSC
```

ERRSC can be used to direct program flow after an error has occurred. (See example under ERROM, above.) Refer to your computer owner's manual for more information regarding error processing.

Notes

Section 9

Tape Commands

The following commands are applicable only to HP-85 tape cartridge operation.

```
CTAPE
```

This function conditions the magnetic tape cartridge by running it forward to the end of the tape and then back to the beginning of the tape. Programs and data on the tape are not affected by the CTAPE operation.

```
ERASETAPE
```

This function is similar to the INITIALIZE command. ERASETAPE sets up a directory on the tape cartridge. All previous information on the tape is destroyed.

```
REWIND
```

This simply rewinds the magnetic tape cartridge to the beginning of the tape.

Syntax Summary

Syntax Guidelines

DOT MATRIX	Items shown in dot matrix must be typed as shown; however, you can use lower case letters if you wish.
[ ]	Items enclosed in brackets are optional parameters.
parameter	Items in italics are optional parameters.
stacked items	When items are placed one above the other, one must be chosen.

ASSIGN# *buffer number* TO "*file specifier*" **Page 35**

ASSIGN# *buffer number* TO * **Page 36**

CAT["*volume label*" / "*msus*"] **Page 20**

CHAIN "*file specifier*" **Page 26**

CHECK READ[OFF] # *buffer number* **Page 63**

COPY "*source file specifier*" TO "*destination file specifier*" **Page 54**

COPY "*source volume label*" / "*source msus*" TO "*destination volume label*" / "*destination msus*" **Page 55**

CREATE "*file specifier*", *# of records* [, *record length*] **Page 34**

ERROM **Page 63**

ERRSC **Page 64**

GLOAD "*file specifier*" **Page 49**

GSTORE "*file specifier*" **Page 47**

INITIALIZE["*volume label*"[, "*msus*" / "*volume label*"[, *directory size*[, *interleave factor*]]]] **Page 14**

LOAD "*file specifier*" **Page 25**

LOADBIN "*file specifier*" **Page 28**

Notes

Appendix A

Maintenance, Service and Warranty

Maintenance

The Mass Storage ROM does not require maintenance. However, there are several areas of caution that you should be aware of. They are:

WARNING: Do not place fingers, tools, or other foreign objects into the plug-in ports. Such actions may result in minor electrical shock hazard and interference with some pacemaker devices. Damage to plug-in port contacts and the computer's internal circuitry may also result.

CAUTION: Always switch off the computer and any peripherals involved when inserting or removing modules. Use only plug-in modules designed by Hewlett-Packard specifically for the HP-83/85. Failure to do so could damage the module, the computer, or the peripherals.

CAUTION: If a module or ROM drawer jams when inserted into a port, it may be upside down or designed for another port. Attempting to force it may damage the computer or the module. Remove the module carefully and reinsert it.

CAUTION: Handle the plug-in ROMs very carefully while they are out of the ROM drawer. Do not insert any objects in the contact holes on the ROM. Always keep the protective cap in place over the ROM contacts while the ROM is not plugged into the ROM drawer. Failure to observe these cautions may result in damage to the ROM or ROM drawer.

For instructions on how to insert and remove the ROM and ROM drawer, please refer to the ROM Drawer Instruction Sheet or the HP-83/85 owner's manual, appendix B.

Service

If at any time you suspect that the ROM drawer or Mass Storage ROM may be malfunctioning, do the following:

1. Turn the computer and all peripherals OFF. Disconnect all peripherals and remove the ROM drawer from the computer ports. Turn the computer back ON. If the computer does not respond or displays `Error 23 : SELF TEST`, the computer requires service.

2. Turn the computer OFF. Install the ROM drawer, with the Mass Storage ROM installed, into any port. Turn the computer back ON.

   - If `Error 112 : M.S.ROM` is displayed, indicating that the ROM is not operating properly, turn the computer OFF and try the ROM in another ROM drawer slot. This will help you determine if particular slots in the ROM drawer are malfunctioning, or if the ROM itself is malfunctioning.
   - If the cursor does not appear, the system is not operating properly. To help determine what is causing the improper operation, repeat step 2 with the ROM drawer installed in a different port, both with the Mass Storage ROM installed in the ROM drawer and with the Mass Storage ROM removed from the ROM drawer.

3. Refer to How to Obtain Repair Service for information on how to obtain repair service for the malfunctioning device.

Warranty Information

The complete warranty statement is included in the information packet shipped with your ROM. Additional copies may be obtained from any authorized HP-83/85 dealer, or the HP sales and service office where you purchased your system.

If you have questions concerning the warranty, and you are unable to contact the authorized HP-83/85 or the HP sales and service office where you purchased your computer, please contact:

In the U.S.:

Hewlett-Packard
Corvallis Division Customer Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330
Tel. (503) 758-1010
Toll Free Number: (800) 547-3400 (except in Oregon, Hawaii and Alaska).

In Europe:

Hewlett-Packard S.A.
7, rue du Bois-du-lan
P.O. Box
CH-1217 Meyrin 2
Geneva
Switzerland

Other Countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Tel. (415) 857-1501

How to Obtain Repair Service

Not all Hewlett-Packard facilities offer service for the HP-83/85 and its peripherals. For information on service in your area, contact your nearest authorized HP dealer or the nearest Hewlett-Packard sales and service office.

If your system malfunctions and repair is required, you can help assure efficient servicing by having the following items with your unit(s) at the time of service:

1. A description of the configuration of the computer, exactly as it was at the time of malfunction, including any plug-in modules, tape cartridges or other accessories.
2. A brief description of the malfunction symptoms for service personnel.
3. Printouts or any other materials that illustrate the problem area.
4. A copy of the sales slip or other proof of purchase to establish the warranty coverage period.

Computer and peripheral design and circuitry are proprietary to Hewlett-Packard and service manuals are not available to customers.

Serial Number

Each computer and peripheral carries an individual serial number. It is recommended that you keep a separate record of this number. Should your unit be stolen or lost, the serial number is often necessary for tracing and recovery, as well as any insurance claims. Hewlett-Packard does not maintain records of individual owner's names and unit serial numbers.

General Shipping Instructions

Should you ever need to ship any portion of your computer system, be sure it is packed in a protective package (use the original case), to avoid in-transit damage. Hewlett-Packard suggests that the customer always insure shipments.

If you happen to be outside of the country where you bought your computer or peripheral, contact the nearest authorized HP-83/85 dealer or the local Hewlett-Packard office. All customs and duties are your responsibility.

Index

W

Error Messages

The HP-83/85 Mass Storage ROM makes available a number of additional error messages. Errors 60 through 75 are available on the HP-85 with or without the Mass Storage ROM. All of these errors are new, however, for the HP-83.

Error Number	Error Condition
Error 60 : WRITE PROTECT	The mass storage medium is write-protected.
Error 61: >42 FILES	HP-83: Not used. HP-85: Attempting to store more than 42 files on a tape.
Error 62 : CARTRIDGE OUT	HP-83: Not used. HP-85: Cartridge is out when attempting a tape operation.
Error 63 : DUP NAME	Duplicate file name.
Error 64 : EMPTY FILE	Attempting to access an empty program file.
Error 65 : END OF TAPE	HP-83: Not used. HP-85: Tape run-off or tape is full.
Error 66 : FILE CLOSED	Attempting to READ#/PRINT# to a closed file. (A warning is issued for attempting to close a closed file.)
Error 67 : FILE NAME	Name does not exist, or name not in quotes.
Error 68 : FILE TYPE	File type mismatch: • Attempting to treat program file as data file, or vice versa. • Attempting to treat binary program as BASIC program or vice versa. • Attempting to treat data as binary program, or vice versa.
Error 69 : RANDOM OVF	Attempting to access beyond existing number of bytes in logical record, using random file access.
Error 70 : READ	System cannot read mass storage medium.
Error 71 : EOF	End-of-file.
Error 72 : RECORD	Record: • Attempting to access a record that doesn't exist. • Attempting to READ#/PRINT# at the end of file. • Lost in record—close file to release the buffer.
Error 73 : SEARCH	HP-83: Not used. HP-85: Bad tape cartridge, or tape not initialized.
Error 74 : STALL	HP-83: Attempting to use non-existent tape drive. HP-85: Tape is stalled.
Error 75 : NOT HP-85 FILE	HP-83: Not used. HP-85: Not an HP-85 file; cannot read.
Error 110 : I/O CARD	The I/O card failed self test and requires service.
Error 111 : IOP	An invalid I/O operation has been attempted.
Error 112 : M.S. ROM	The Mass Storage ROM failed self-test.

1000 N.E. Circle Blvd., Corvallis, OR 97330

00085-90447

Printed in U.S.A.