

Dammiller

BDD

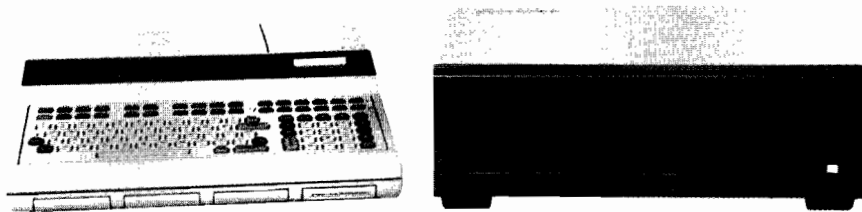
Hewlett-Packard 9825A Calculator Disk Programming



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Disk Programming



HP 9825A Calculator and HP 9885M Flexible Disk Drive

Hewlett-Packard Fort Collins Division
P.O. Box 1550, Fort Collins, Colorado 80522
(For World-wide Sales and Service Offices see back of manual.)
Copyright by Hewlett-Packard Company 1977

Printing History

Each new edition incorporates all material updated since the previous edition. Each new or revised page is indicated by a revision (rev) date. Manual Change sheets are issued between editions and contain information to be corrected or inserted in the manual by the user.

The date on the back cover changes only when each new edition is published. Minor corrections or additions may be made as the manual is reprinted between editions.

First EditionDec 1976

Second Edition.....Sept 1977

Table of Contents

Chapter 1 - General Information

| | |
|------------------------------|---|
| Introduction | 1 |
| 9825A Calculator | 1 |
| 9885M Disk Drive | 2 |
| 9885S Disk Drive | 2 |
| The Disk | 2 |
| Option 025 Interface Kit | 3 |
| Requirements | 3 |
| Suggested Disk Manufacturers | 3 |
| Getting Started | 4 |

Chapter 2 - Disk Structure

| | |
|-------------------------------|----|
| Introduction | 7 |
| Systems Area | 8 |
| System Table | 9 |
| File Directory | 9 |
| Availability Table | 10 |
| Bootstraps Area | 10 |
| Backup Track | 11 |
| Storage Area | 11 |
| File Structure | 11 |
| Program Files | 11 |
| Data Files | 12 |
| Serial File Access | 12 |
| Random File Access | 13 |
| Serial vs. Random File Access | 13 |
| Write Protecting the Disk | 14 |

Chapter 3 - Program File Operations

| | |
|------------------------------|----|
| Introduction | 15 |
| Conventions | 15 |
| Requirements | 16 |
| Program File Statements | 17 |
| Drive Statement | 17 |
| File Names | 17 |
| Save Statement | 18 |
| Catalog Statement | 20 |
| Get Statement | 22 |
| Chain Statement | 25 |
| Kill Statement | 27 |
| Rename Statement | 28 |
| Resave Statement | 28 |
| Save Keys Statement | 29 |
| Get Keys Statement | 29 |
| Program Storage Requirements | 30 |

Chapter 4 - Data File Operations

| | |
|----------------------------------|----|
| Introduction | 31 |
| Overview of Data File Operations | 31 |
| Conventions | 32 |
| Data File Statements | 33 |
| Open Statement | 33 |
| Files Statement | 34 |
| Data File Pointers | 35 |
| Assign Statement | 36 |
| Serial Print Statement | 38 |
| Serial Read Statement | 41 |
| Random Print Statement | 43 |
| Random Read Statement | 46 |
| Positioning the Pointer | 47 |
| On End Statement | 50 |
| Type Function | 51 |
| Data Storage Requirements | 53 |
| Summary of EOR and EOF Marks | 54 |

Chapter 5 - Other Operations

| | |
|----------------------------|----|
| Additional Statements | 57 |
| Save Memory Statement | 57 |
| Get Memory Statement | 57 |
| Repack Statement | 58 |
| Verify Statements | 58 |
| Copy Statements | 59 |
| Disk Copy | 59 |
| File Copy | 60 |
| Partial File Copy | 60 |
| Dump Statements | 62 |
| Disk Dump | 62 |
| File Dump | 62 |
| Load Statements | 63 |
| Disk Load | 63 |
| File Load | 63 |
| Get Binary Statement | 64 |
| Error Recovery Routines | 64 |
| Record Header Error (d5) | 64 |
| Track Not Found Error (d6) | 65 |
| Data Checkword Error (d7) | 66 |
| Binary Programs | 67 |
| Initialization Routine | 67 |
| Bootstrap Routine | 67 |
| Verify Boots Routine | 67 |
| Killall Routine | 67 |
| Error Messages | 68 |

Appendix A - Disk Specs and Care

| | |
|--------------------|----|
| Specifications | 69 |
| Disk Capacity | 69 |
| Disk Speed | 69 |
| Transfer Times | 70 |
| Disk Care | 72 |
| Guidelines | 72 |
| System Reliability | 73 |
| Maintenance A | 73 |

Appendix B - Installation and Set Up

| | |
|--|----|
| Getting Started | 75 |
| 1 Unpacking Your System | 75 |
| Equipment Supplied | 76 |
| Additional Equipment | 76 |
| Option 002 Rack Mount Kit | 77 |
| 2 Checking Fuses, Voltage and Power Cords | 78 |
| Fuses | 78 |
| Power Requirements | 79 |
| Option 001 for 50Hz Operation | 79 |
| Power Cords | 80 |
| 3 Connecting Calculators, Drives and Interface Cable Cards | 81 |
| 4 Setting Drive Switches and Select Codes | 82 |
| 5 Installing the ROM Card | 83 |
| 6 Installing the Disk | 84 |
| 7 Turn On | 85 |
| 8 Pattern Test (Verification Routine) | 86 |
| 9 Testing the System | 87 |
| Disk System Cartridge Programs | 87 |
| Checkread Test | 88 |
| HPL Disk Test | 89 |
| Self Test | 89 |
| 10 Initializing Blank Disks | 90 |

Appendix C - Terms, Statements and Errors

| | |
|---------------------------|-----|
| Disk Terms | 93 |
| Disk Statement Summary | 96 |
| ASCII Table | 106 |
| Sales and Service Offices | 107 |
| Index | 109 |
| Error Messages | 112 |

Chapter 1

General Information

Introduction

The HP 9885 Disk Drive is a mass storage device that uses a flexible disk as the storage medium. Flexible disks can be accessed much faster than tape cartridge and have more than twice the storage capacity. The amount of available storage space is more than .4 million data bytes per disk. In addition, short data access time makes the system extremely powerful and file access by name makes the system easy to use.

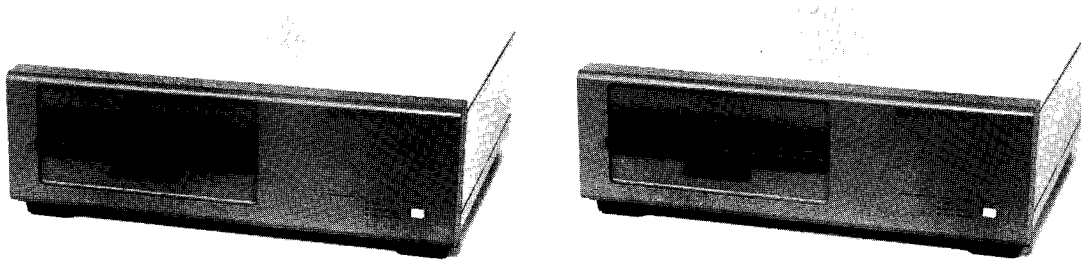
The 9825A/9885 Flexible Disk System can accommodate up to a total of 32 HP 9885M (Master) Disk Drives and HP 9885S (Slave) Disk Drives in combination. Each component of the system is discussed briefly in the following pages.

9825A Calculator

The system requires an HP 9825A Calculator with any one of the available memory sizes. The HP 98032A Option 085 Interface Cable connects the 9825A Calculator to the 9885M Drive.

9885M Disk Drive

The 9885M is the controller drive in single and multiple drive systems. The 9885M can hold and operate one flexible disk at a time. At least one 9885M is required for the system to operate, although up to eight* 9885M's can be connected to a calculator.



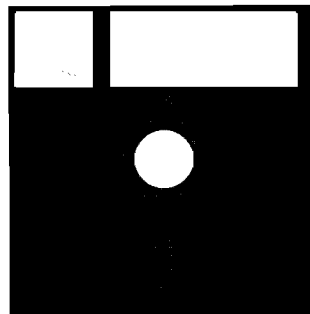
HP 9885M and 9885S Flexible Disk Drives

9885S Disk Drive

The 9885S is the slave drive used in multiple drive systems. Up to three 9885S Drives can be connected to each 9885M in the system. Each drive can hold and operate one flexible disk at a time. The 09885-61607 cable connects the drives in a multiple drive system.

The Disk

The flexible disk is the storage medium for the system. Each disk can hold more than .4 million bytes. Only the lower surface of the disk is used for storage.



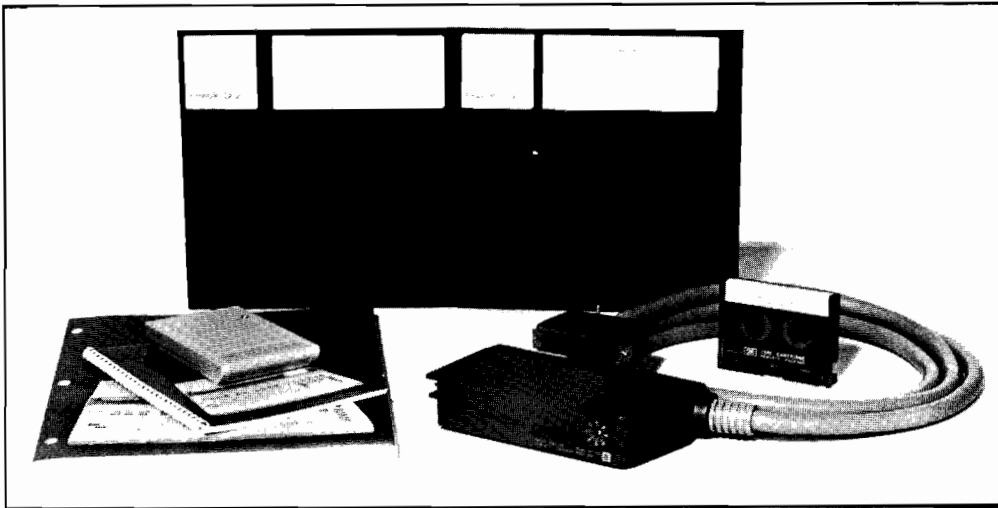
Flexible Disk

Each disk must be initialized before using it for the first time. For this procedure, refer to Initializing New Disks, page 90. (One of the disks supplied with your system is already initialized and ready for use.)

*An HP 9878A I/O Expander is required if more than three interface cables (including those for 9885M Drives) are connected to the calculator.

Option 025 Interface Kit

The Option 025 Interface Kit consists of an HP 9885 Disk ROM Card, an HP 98032A Option 085 Interface Cable, a Disk System Cartridge, two disks and manuals. When the Disk ROM is installed in your calculator, it requires 1140 bytes of RWM (read/write memory).



Option 025 Interface Kit

The Disk System Cartridge contains programs to test the system, initialize new disks and load the disk system's "bootstraps" (system statements needed to operate the disk drive, not found in the Disk ROM) and Error Recovery Routines.

Requirements

Before using this manual, you should be familiar with the calculator operations and the HPL programming language described in the HP 9825A Operating and Programming Manual.

Suggested Disk Manufacturers

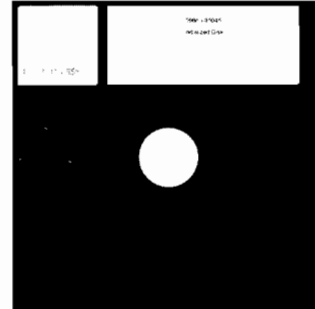
A list of approved disk manufacturers is available through your HP Sales and Service Office. Use only those disks with your 9885 Flexible Disk System, or loss of data, damage to the read/write head and high maintenance costs are likely to result.

IMPORTANT

Do not use disks other than those approved by HP, otherwise permanent damage to your drive will result.

Getting Started

An initialized disk is supplied with your system. With this disk installed, you can begin using your system immediately. Follow the steps below (referring to the detailed instructions in the Appendix, when necessary) to get your system set up and ready to use.



- 1** Once your 9825A Calculator and 9885M Drive (or Drives) are unpacked, inspect them for damage...further instructions are found on page 75.
- 2** Check for the appropriate fuse, line voltage and power cords... more electrical information is found on page 78.

WARNING

ALWAYS DISCONNECT THE DRIVE FROM ANY AC POWER SOURCE BEFORE CHANGING FUSES OR SETTING VOLTAGE SELECTOR SWITCHES.

- 3** Connect the calculator to the drive, or drives, using the 98032A Interface Card cable, or cables. Then connect the calculator and the drive, or drives, to an ac power source... for further information about connecting the calculator and drives, see page 81.
- 4** Set the drive number switch on the back panel of each drive for the appropriate drive number-0 thru 3. (The drive number selected is the one opposite the dot on the switch.)Set all select codes-8 thru 15 on all Interface Card cables for the appropriate select code...drive number 0, select code 8 are most often used...see page 82 for additional information.
- 5** Install the HP 98217 Disk ROM in the calculator... instructions to do this are on page 83.

6 Install the HP Disk with the Initialized Disk label if you want to use your system immediately. (The other disk provided is not initialized and should be used only if your next step is 10-Initializing a Blank Disk.)...more instructions for installing a disk are on page 84.

7 Turn on the calculator and drive, or drives, in your system.. see page 85 if you want further turn on instructions.

8 To be sure your system is installed correctly and functioning properly, perform the Checkread Test on page 86.

Then if you want to start using your system immediately, skip to Chapter 3 - Program File Statements. (Chapter 2 covers disk structure, program files, data files and the difference between random and serial data file access. The next two steps involve further testing of your system and initializing blank disks.)

9 Testing...the entire system can be tested using the Pattern Test and the HPL Disk Test by following the steps on page 88.

To test the electrical performance of the drive with a blank disk or with no disk at all, use the Self Test on page 89.

10 Initializing blank disks...to initialize a blank disk and load bootstraps, follow the procedure on page 90.

IMPORTANT

should not be pressed during any disk operation. Pressing during a write operation (i.e. anything that changes information on the disk) can leave the disk in an inconsistent state. For example, if is pressed during `kill`, the file entry may be removed from the directory but the available space may not be returned to the availability table.

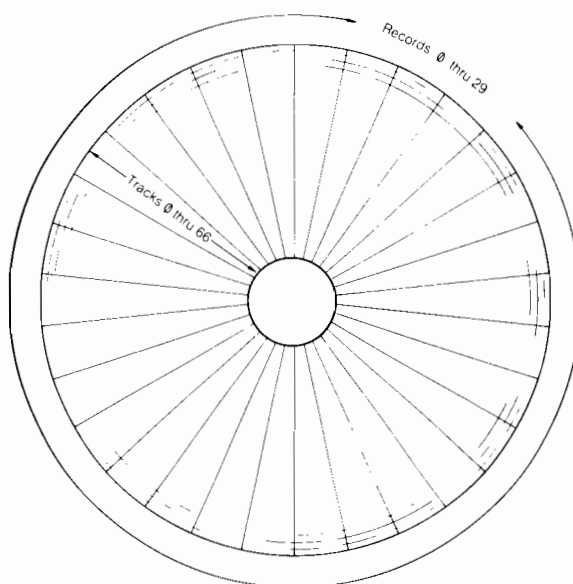
Chapter 2

Disk Structure

Introduction

The disk used in the Flexible Disk System is a circle of plastic 20cm (7 7/8") in diameter, enclosed in a sealed black plastic jacket. Bonded onto the surface of the disk is a ferromagnetic iron oxide with characteristics similar to magnetic tape. Data is stored in the form of binary digits (bits) represented by magnetized spots on the disk. Information is stored and retrieved by means of a read/write head that comes in contact with the lower surface of the disk.

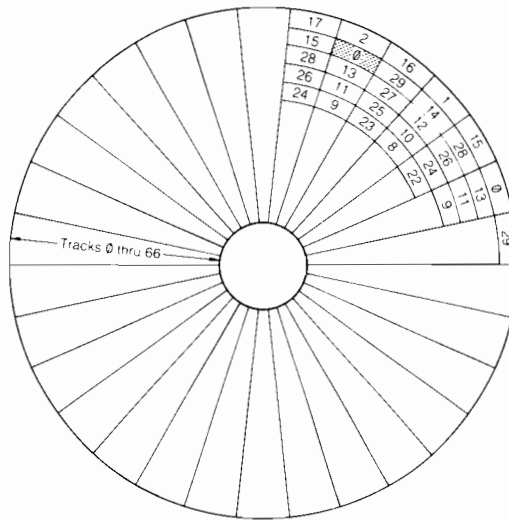
Data is stored in concentric tracks on the disk. Each disk has 67 circular tracks, numbered 0 thru 66. The disk is also subdivided into 30 pie-shaped sectors. Each sector contains 67 records (1 record = 256 bytes).



Disk Structure

Records are not numbered sequentially; instead, they are numbered alternately. This shortens the time it takes to access successive records, since a complete revolution of the disk between execution of read (or write) statements is avoided.

Shown below is a diagram of disk tracks and records with their alternating numbering system. The shaded area shows the location of a specific record - Track 1, Record 0.



Flexible Disk Records

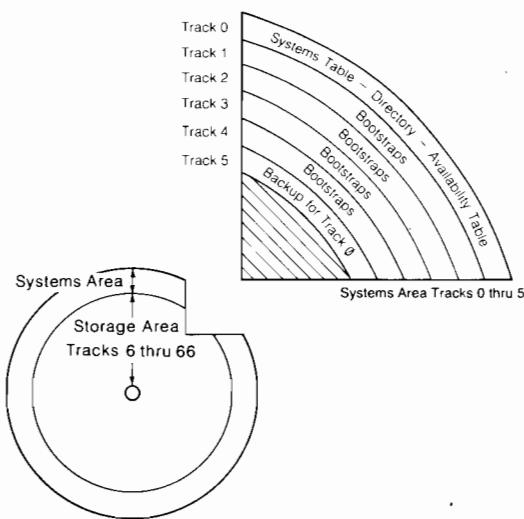
In addition to an alternating numbering system, the location of the beginning record (Record 0) of each track is staggered to avoid a revolution when the drive accesses (steps to) a new track. For example, after Record 29, Track 0 is accessed, Record 0, Track 1 can be accessed without an extra revolution.

Systems Area

Some of the area on the disk is reserved for use by the system (Tracks 0 thru 5). The rest of the disk area (Tracks 6 thru 66) is available for your use. The system area has four tracks containing bootstraps (1 thru 4) and two tracks (Tracks 0 and 5*) containing –

- A Systems Table.
- A Directory of files, their locations, types and sizes, (once you've defined them).
- An Availability Table that monitors remaining usable disk space.

*The same information found in Track 0 is duplicated in Track 5 as backup.



Systems Area of Disk

Systems Table

The systems table in Record 0 indicates the calculator used to initialize the disk, the number of defective tracks and the beginning location of the user area.

When a disk is initialized, the number of defective tracks is recorded in the systems table. If more than six tracks are defective, the disk is rejected and error B1 is displayed.* The physical location of the defective tracks is not known to you. This means your disk has effectively, a contiguous set of logical tracks with no intervening defective tracks. For example, if there are two defective tracks on a disk, your usable tracks will be numbered 0 to 64.

File Directory

The file directory in Records 1 thru 22 contains entries for 352 possible files, one entry for every possible file written on the disk. Each entry contains information such as file name, location, size and type of each file.

If the directory in track 0 cannot be read, the spare directory from track 5 is automatically read. When this occurs, the message `SPARE DIR.` is printed.

*Contact your HP Sales and Service Office for a replacement disk.

Availability Table

The availability table in Records 23 thru 28 monitors the amount and location of remaining disk space. The availability table is automatically updated* after any file is added to, or removed from, the disk .

Record 29 of the systems area is unused.

Bootstraps Area

The bootstraps area in tracks 1 thru 4 contains statements and routines used by the system that are not contained in the Disk ROM. The bootstraps are automatically loaded from the Disk System Cartridge immediately following initialization.

Each time a disk statement is executed, it becomes part of the calculator's read/write memory so that if it is executed again, the same statement is not reloaded into the calculator's read/write memory. This reduces the time required to execute that same disk statement again (if no other statements are executed in between*) since the calculator no longer needs to access the disk bootstraps. (Each new statement overlays its bootstraps on the previous one.) This increases the speed of repetitive print and read operations.

In addition, during execution of a series of read statements, the calculator checks which record is in the calculator memory before it reads the next record. If the record in the memory is the same as the next record number to be read, the calculator does not reread the data in the record .

Any space on the disk that becomes available (after execution of a `kill` statement) is automatically combined with other available disk space if the areas are contiguous. This creates a larger available spaces on the disk instead of numerous shorter spaces.

*And if the drive door is not opened.

Backup Track

Track 5 contains the same system information (systems table, file directory and availability table) as Track 0. The information on Track 5 is automatically used if Track 0 should become defective.

Storage Area

Tracks 6 thru 66 are used for recording your files and programs. With 30 records per track and 256 bytes per record, there are 468,480 bytes of available storage space per disk. Whenever new information is added to, or deleted from, the storage area of the disk or whenever the disk is reorganized (repacked) the information in the systems area is also automatically updated.

File Structure

The Flexible Disk System is organized around user defined memory areas called files. Each disk can have up to 352 files, depending on the size of each file. Files can be used to hold data (data files), programs (program files), calculator memory (memory files) and special function keys (key files). Binary programs can be stored in binary files from tape cartridge only.

It is up to you to create these files, name them and – for data files – specify their size. The HPL Disk programming statements described in the next chapters enable you to store information on, and retrieve information from, your disk.

Each file contains one or more records, 256 (8 bit) bytes in size. A record is the smallest addressable unit of data on the disk which can be accessed directly by the system. A file cannot be greater than 1830 records (the maximum available storage space on the disk).

The size of a program, memory or key file is automatically determined - it is the number of records required to store the program. When you create a data file however, you must specify its size, in records. The differences between program files and data files follow.

Program Files

Programs are stored on a disk using as many complete records as necessary, each record containing 256 bytes. Therefore, if a program is 257 bytes in length, two records are required to store it in a program file.

Data Files

There are two ways to store and access data – serially and randomly. It is up to you to determine which method of data access best suits your needs for a particular problem. This decision is based on the nature of the problem you must solve the amount of usable disk storage space and the time available to solve your problem.

For example, suppose you are working with thousands of customer account numbers and their balances due and your job is to output a daily list of all customers and their balances. In this situation, it is best to pack all data items (customer numbers and balances due) together tightly in a data file to save space on the disk and to save time when accessing the data. This is the serial access method of data storage.

To update individual customer balances, you'll need another file containing customer numbers, names, addresses, items purchased and balances due. The data in this file is arranged so that each individual item (customer name or number) can be accessed. This method of storing data usually takes more space on the disk. The advantage to this method is that any item can be easily updated since individual items can be accessed much faster. This is the random access method of data storage.

Serial File Access

Data treated as a unit (instead of as individual items) can be handled using serial print (`SPRT`) and serial read (`SREAD`) statements. When serial print statements are used to store data on the disk, data lists are stored compactly without identifiable marks between lists. These data lists make up a file and can use as many records (256 bytes = 1 record) as necessary. Data lists can contain numerics and strings.

All or part of the information stored originally can be retrieved in one serial read statement. The list of data elements read does not have to be identical to the list originally printed in the file, but these data lists must be identical in type and order. (The names you assign to these elements can vary.)

The beginning of a serial file is the only point where access is normally begun. Storage space is utilized with maximum efficiency when serial print statements are used, since data is packed solidly and no unused space is left between items.

Random File Access

When data items are handled individually (instead of as a unit) random print (`rprint`) and random read (`rread`) statements are used. The data is stored starting at the beginning of a specified record so that every data item is directly accessible. Storing data randomly may not utilize storage space efficiently, since only a part of a record required for storage is used and the rest of the record is not.

Each of the data items stored originally can be retrieved using a random read statement. The list of data items does not have to be identical to the list originally stored in the record, but these data lists must be identical in type and order. (The names you assign to these elements can still vary.)

When working with data using random print and read statements, you must specify which record within a file you want to access. The advantage to this method is that every record is directly accessible, in any order.

Serial vs. Random File Access

As mentioned before, you must decide which method of data accessing is best for your particular needs. This decision is usually not made easily, because of the advantages and disadvantages of both methods. More efficient storage space utilization must be sacrificed for a shorter access time, and vice versa. Once your decision has been made, it is difficult to change later, so choose your method carefully.

The advantages and disadvantages of accessing data from a file serially and randomly are summarized in the following table.

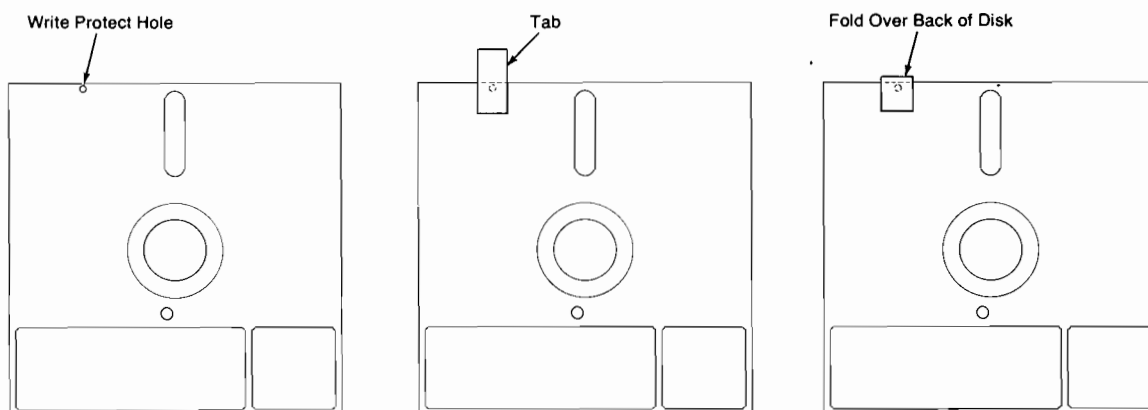
Serial vs. Random File Access

| Feature | Serial File Access | Random File Access |
|--------------------|---|---------------------------------------|
| Storage Efficiency | Good-Data is packed solidly | Varies-Wasted space for short records |
| Access Time | Varies-Longer for higher numbered records | Good-Direct access to any record |
| Record Length | Can be more than 256 bytes | Less than or equal to 256 bytes |

Write Protecting the Disk

The data (write) protect feature protects the data and programs on a disk from being written over. The disk is write-protected by a hole in the sealed protective jacket in the location shown below. When the write protect hole is open, nothing can be written on the disk.

When the write protect hole is covered, writing is allowed on the disk. HP disks are shipped with the write protect hole covered as shown below.



Covering the Write Protect Hole

Chapter **3**

Program File Operations

Introduction

Disk files can be used to hold programs, data, special function keys, binary programs and the entire calculator memory. This chapter discusses program and key files and the statements used to catalog, store, retrieve and erase these files from the disk. All disk statements covered in this chapter can be executed from within a program, from the keyboard or in the live keyboard mode, except for the `set`, `chain` and `getk` (get key) statements which can't be executed while in the live keyboard mode.

A description of each program file statement is shown in this chapter and again in the Appendix.

If the disk and the drive are not properly connected, the message `DISK IS DOWN` is flashed in the display.

Conventions

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

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

`brackets []` – All items enclosed within brackets are optional, unless the brackets are in dot matrix.

The following definitions are used in the statement descriptions.

These parameters can be numeric expressions –

- drive number(except in the `files` statement, where it must be a numeric constant)
- select code
- line number
- number of records
- position number
- file number
- record number
- tape file number

These parameters can be text (e.g. `"ABC"`) or string expressions (e.g. `A$` or `cap(A$)`) unless otherwise stated –

- buffer name
- file name (except in the `files` statement, where it must be text)
- label

Data items used in print and read statements can be –

- simple variables (e.g. `A`)
- array elements (e.g. `A[1]`)
- entire arrays (e.g. `A[*]`)
- entire string variables or arrays (e.g. `A$`)
- substrings (e.g. `A$[X,Y]`)
- r registers (e.g. `r5`)

In addition, data items in print statements can also be –

- numeric constants (e.g. `5.12`)
- numeric expressions (e.g. `5A`)
- text (e.g. `"ABC"`)
- string expressions (e.g. `cap(A$)`)

To use string variables, the String ROM must be installed. To send a catalog listing to an output device, the General and Extended I/O ROMs must be installed in the calculator.

Program File Statements

The Drive Statement

The `drive` statement specifies the drive being accessed by drive number (0 thru 3) and, optionally, by select code (8 thru 15) on the interface cable —

```
drive drive number [ , select code]
```

The default value for the drive number is 0 and for the select code, 8. The default values are automatically in effect when the calculator is turned on, **RESET** is pressed or the calculator memory is erased. When a `drive` statement is executed, the drive number and select code indicated are valid until another `drive` statement is encountered, `erase a` is executed, **RESET** is pressed or the calculator is turned off.

Executing a `drive` statement does not affect file pointers (covered in Chapter 4) unless the optional select code parameter is used.

File Names

Every program or data file on the disk must have a unique name which can contain up to six characters and can be a string variable. These characters cannot be used in a file name —

- quotation marks (" ")
- commas (,)
- semicolons (;)
- colons (:)
- blanks (i.e. spaces)
- special characters with octal codes less than 41 (See the ASCII Character Code Table in the Appendix.)

The Save Statement

The `save` statement stores an entire program or part of it in a specified file.

```
save "file name" [ ; 1st line number [ ; 2nd line number]]
```

With a program in the calculator's memory, execution of the `save` statement stores the program on the disk. The specified program file takes up the number of whole records needed for the program.

The optional line number parameters enable you to store part of your program rather than all of it. With one line number specified, the `save` statement stores only the lines after (and including) the specified line. With both parameters, the lines between (including the specified lines) are stored on the disk. Your whole program is still present in the calculator, whether all or only a portion of it is saved on the disk.

To illustrate use of the `save` statement, key in the following program.

```
0: ent J
1: for I=J to J+
  10
2: prt I
3: next I
4: end
*27317
```

You can store the previous program in a file named `Count`, for example, by executing this statement from the keyboard –

```
save "Count"
```

A squaring program is added to the original program just stored on the disk.

```
0: dsp "Counting
   Program";wait
  1000
1: dsp "For saving, key in
   100";wait 1000
2: dsp "then
   the number to
   be squared.";
   wait 1000.
```

```

3: ent "Key in
   an integer",J
4: if J>99:ato 9
5: for I=J to J+
   10
6: prt I
7: next I
8: end
9: dsp "Squaring
   Program":wait
   1000
10: ent "Key in
   another integer
   ",K
11: for L=K to
   K+10
12: prt L,L+2
13: next L
14: end
*31160

```

Lines 0 thru 8 print ten consecutive numbers from the first number entered. Lines 9 thru 14 print ten consecutive numbers and their squares, from the second number entered.

Once this program has been keyed into the calculator, you can save it in a file named Master, for example, by executing this statement from the keyboard –

```
save "Master"
```

You can also store the second half of the program in a file named Square, for example, by executing this statement from the keyboard –

```
save "Square",9
```

Finally, to store the first half of this program in a file named First, for example, execute the following statement from the keyboard –

```
save "First",0,8
```

You now have four programs stored using different file names. Once a program is stored on the disk, it can be loaded back into the calculator memory using the `get` or `chain` statements discussed later.

The Catalog Statement

The catalog (`cat`) statement prints a list of every file on the disk and specific information about each file, including its –

Name

Type

P - Program File

K - Key File

D - Data File

M - Memory File

B - Binary Program File

O - Other

Size

Number of Bytes (B) for a Program, Key, Memory or Binary File

Number of Records (R) for a Data File

At the top of the listing the drive being cataloged and the bootstraps revision letter are listed along with the number of remaining available records.

`cat [select code] or ["buffer name "]*`

If no select code is used following `cat`, or `cat 0` is executed, the internal printer outputs the catalog listing. For example, execute `cat` –

```
CAT DRIVE    A /A
AVL RCRDS 1825

NAME TYPE SIZE
= = = = =
Count   P   48   B
Master  P  284   B
Square  P  100   B
First   P  186   B
```

When 16 follows the catalog statement, complete information about every file on the disk you are using is output to the calculator printer. Drive number, bootstraps revision letter and remaining available records are printed, as with other catalog statements.

*See the General and Extended I/O Manuals.

Here's the catalog listing that's output when the `cat 16` statement is executed –

```
CAT DRIVE    0 /A
AVL RCRDS 1825

NAME TYPE SIZE
#REC TRCK RCRD
= = = = =

Count  P   48B
  1    T6    R0
Master P  284B
  2    T6    R1
Square P  100B
  1    T6    R3
First  P  186B
  1    T6    R4
```

This listing includes the information from the previous catalog statement (Name, Type and Size) plus –

#Rec Number of Records in the file
Trck Specific file location by Track (T)
Rcrd Specific file location by Record (R)

When a select code between 2 and 15 follows the catalog statement, the complete catalog listing is output to the HP - IB or specified device or to the I/O buffer.* For example, to output a catalog listing to the HP 9871A Printer, execute `cat 6` (the factory set select code for the HP 9871A Printer) –

```
CAT DRIVE    0 /A
AVL RCRDS 1825
NAME TYPE SIZE      #REC TRCK RCRD
= = = = = = = = = =
Count  P   48B      1    T6    R0
Master P  284B      2    T6    R1
Square P  100B      1    T6    R3
First  P  186B      1    T6    R4
```

When an I/O buffer name or text (e.g. "ABC ") follows the catalog statement, complete information about all user files is output to the specified I/O buffer. See the Extended I/O Programming Manual for more information. For example –

```
cat "buffer"
```

*The General and Extended I/O ROMs are required to output to the I/O buffer.

The Get Statement

The `get` statement loads a program from the disk to the calculator. All variables and arrays are lost.*

```
get "file name" [ ; 1st line number [ ; 2nd line number ]]
```

Whenever the `get` statement is executed, all program lines in the specified file are loaded into memory. All program lines previously in the calculator memory are erased except those lines preceding the first line number, if one is specified. If the second line number is included, program execution automatically begins at that line number.

The following statement, executed from the keyboard, loads the program from the file named `Master` into the calculator. (This program was stored previously; see page 18.)

```
get "Master"
0: dsp "Counting
   Program";wait
   1000
1: dsp "For squa
   ring, key in
   100";wait 1000
2: dsp "then
   the number to
   be squared.";
   wait 1000
3: ent "Key in
   an integer";J
4: if J>99;eto 9
5: for I=J to J+
   10
6: prt I
7: next I
8: end
9: dsp "Squaring
   Program";wait
   1000
10: ent "Key in
   another integer
   ";K
11: for L=K to
   K+10
12: prt L;L+2
13: next L
14: end
*31160
```

*The `chain` statement, described next, retains the values of variables. *

The program can be altered once it is in the calculator, but the information on the disk remains unchanged (unless you change it using a `resave` statement).

From within a running program, if the first line number is specified in a `set` statement, the program is loaded beginning with that line number and is renumbered* from that line number. Program lines with numbers lower than the first line number, are retained in memory; all other lines previously in memory are erased. If the second line number is not included, the default value becomes the first line number and program execution automatically begins at the first line number specified from within a running program only.

The following statement, in addition to loading the program into the calculator, renumbers the lines, starting with line 15.

```
set "Master",15
```

A listing of the calculator memory at this point is shown below.

```
0: dsp "Counting
   Program";wait
   1000
1: dsp "For squa
   ring: key in
   100";wait 1000
2: dsp "then
   the number to
   be squared.";
   wait 1000
3: ent "Key in
   an integer";J
4: if J>99;eto 9
5: for I=J to J+
   10
6: prt I
7: next I
8: end
9: dsp "Squaring
   Program";wait
   1000
10: ent "Key in
   another integer
   ";K
11: for L=K to
   K+10
12: prt L,L+2
13: next L
14: end
```

*Any `eto` statement addresses (line numbers) in the program are not automatically renumbered.

```

15: dsp "Countin
    e Program";wait
    1000
16: dsp "For
    squaring, key
    in 100";wait
    1000
17: dsp "then
    the number to
    be squared.";
    wait 1000
18: ent "Key in
    an integer";J
19: if J>99;sto
    9
20: for I=J to
    J+10
21: prt I
22: next I
23: end
24: dsp "Squarin
    e Program";wait
    1000
25: ent "Key in
    another inteser
    ";K
26: for L=K to
    K+10
27: prt L,L+2
28: next L
29: end
*28209

```

As you can see, the program originally loaded has not been erased; the second program has been loaded after it (beginning at line 15). Had the second program been renumbered from line 5, only lines 0 thru 4 of the original program would have remained.

When a second line number is included, the `set` statement causes program execution to begin immediately at the specified line number. For example -

```
set "Master",15,9
```

By executing the statement above, the program is renumbered from line 15 again, and automatically executed beginning at line 9.

The Chain Statement

The `chain` statement is identical to the `get` statement discussed previously, except that current variables are not lost. When executed from the keyboard, the loaded program will not be automatically run unless the second line number is specified, as with the `get` statement.

```
chain "file name "[, 1st line number [, 2nd number]]
```

The `chain` statement is most often used in a program to link a program previously stored on the disk to one currently in the memory.

The following three programs are used to illustrate the `chain` statement and how the values of variables are retained –

Key in the program shown and then execute –

```
save "Begin"

0: prt "Begin
   executed", ""
1: chain "Middle
   ", 0, 0
2: end
*26760
```

Key in this program and execute –

```
save "Middle"

0: 0+J
1: sto 3
2: 2+J
3: if J=1:sto 7
4: if J=2:sto 9
5: prt "Middle
   chained to
   Begin and
   executed from
   0", ""
6: chain "End",
   0, 0
7: prt "Middle
   chained from
   End and   exec
   uted from 3", ""
8: chain "End",
   10, 2
9: prt "Middle
   chained from
   End and   exec
   uted from 2", ""
10: end
*10727
```


Then key in the last program and execute -

```

save "End"

0: if J=2;eto 14      *
1: prt "End chain
   ned fromMiddle
   and exec-uted
   from 0",""
2: 1+J
3: chain "Middle
   ",0,3
4: prt "End chain
   ned fromMiddle
   and exec-uted
   from 3",""
5: end
*25051

```

These programs are now stored on the disk and can be run by executing -

```
set "Begin",0,0
```

In these programs, the values of variable J are retained when chaining from program to program, causing selected portions of the last two program segments to be run.

Here's the printout -

```

Begin executed

Middle chained
to Begin and
executed from 0

End chained from
Middle and exec-
uted from 0

Middle chained
from End and
executed from 3

Middle chained
from End and
executed from 2

End chained from
Middle and exec-
uted from 3

```

*The `eto` statement addresses are not automatically renumbered. Therefore, the `eto` address in line 0 refers to a line number in the program after its lines have been renumbered. To avoid this situation, use labels instead of line numbers as `eto` statement addresses.

The Kill Statement

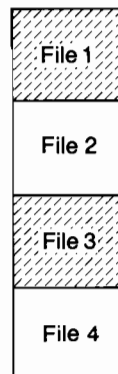
The `kill` statement erases from the disk the file named and releases the space it occupied for further storage.

```
kill "file name "
```

By executing the following statement, the program stored in Count can be erased from the disk –

```
kill "Count"
```

An automatic update of the availability table is performed, following the `kill` statement. This means that the space released is automatically combined with any contiguous space on the disk to create larger available storage areas. For example, in the diagram below, files 2 and 4 have already been killed.



Creating Contiguous Areas

If file 3 is killed, the contiguous areas held by files 2, 3 and 4 are combined and become one larger available area instead of three smaller available areas.

The Rename Statement

The rename (`renm`) statement changes the name of a file from the original name to the new name specified.

```
renm "old file name" , "new file name"
```

For example, the data file named `Master` can be renamed `MASTER` (as shown in the catalog listing below) by executing —

```
renm "Master", "MA  
STER"
```

When the catalog statement is executed —

```
CAT DRIVE    0 /A
AVL RCRDS 1822
NAME TYPE SIZE
= = = = =
MASTER P   284 B
Square P   100 B
First  P   186 B
Begin  P    50 B
Middle P   268 B
End    P   154 B
```

The Resave Statement

The `resave` statement enables you to store all or part of a program, which has been modified, back in the same file.

```
resave "file name" [, 1st line number[, 2nd line number]]
```

First and second line numbers are used to indicate the portion of the program being resaved as in the `save` statement.

This allows you to store a modified program in one step instead of two.

The Save Keys Statement

The save keys (`save k`) statement stores special function key definitions in a specified file.

```
save k "file name "
```

The definitions of all 24 special function keys can be stored in one file at the same time. Since only key definitions are saved with this statement, two files are required when a program that uses special function keys is to be stored on the disk.

The Get Keys Statement

The get keys (`get k`) statement loads special function key definitions from a specified file of the disk to the calculator special function keys.

```
get k "file name "
```

When `get k` has been executed, the original information is returned to the special function keys. All of the special function keys can then perform the same operation they did when their definitions were saved in the key file.

In addition, when `get k` is executed, previously defined variable values are not lost, similar to the `chain` statement.

Program Storage Requirements

When the number of bytes required to store a program is greater than 256, more than one record is required to store it. If the number of bytes is not a multiple of 256, then the system rounds the number of records in the file to the next whole number. For example, a program that takes 255 bytes requires one record, while a program that takes 257 bytes to store it, requires two records.

Chapter 4

Data File Operations

Introduction

Chapter 3 of this manual describes program and key files; this chapter covers the statements which are useful when working with data files. All statements used either with program files only (i.e., `get`, `chain`, `save`, `getk`, `savek` or `resave`) or those used with both program and data files (i.e., `drive`, `kill`, `renn` or `cat`) are discussed in Chapters 3 and 5.

The statement description conventions used in the previous chapter still apply.

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

`brackets []` – All items enclosed within brackets are optional, unless the brackets are in dot matrix.

A description of each data file statement is shown in this chapter and again in the Appendix. The message `DISK IS DOWN` is flashed in the display if the calculator and drive are not properly connected.

Overview of Data File Operations

Before you can do any data file operations, you must first find a free area on the disk and give that area a name. This is done with the `open` statement and that area is then called a data file. The file can be used to contain whatever data you want to put into it, like readings from an instrument, inventory information, etc.

There can be numerous data files on a disk and different programs (also on the disk) can use different data files. This means that a program must tell the system which file (s) are to be used. This is done by including a `files` or `asen` statement in the program.

Once the program indicates the data files to be used, it can store information (numbers and strings) in the files using `rprt` or `srpt` statements and can read back the information using `rread` or `sread` statements.

Conventions

The following definitions are used in the statement descriptions.

These parameters can be numeric expressions –

- drive number (except in the `files` statement, where it must be a numeric constant)
- select code
- line number
- number of records
- position number
- file number
- record number
- tape file number

These parameters can be text (e.g. "ABC") or string expressions (e.g. `A$` or `cap (A$)`) unless otherwise stated –

- buffer name
- file name (except in the `files` statement, where it must be text)
- label

Data items used in print and read statements can be –

- simple variables (e.g. `A`)
- array elements (e.g. `A[1]`)
- entire arrays (e.g. `A[*]`)
- entire string variables or arrays (e.g. `A$`)
- substrings (e.g. `A$[X,Y]`)
- r registers (e.g. `r5`)

In addition, data items in print statements can also be –

- numeric expressions (e.g. `5A`)
- numeric constants (e.g. `5.12`)
- string expressions (e.g. `cap (A$)`)
- text (e.g. "ABC")

All data file statements covered in this chapter can be executed from the keyboard, from a program or from the live keyboard mode.

Data File Statements

The Open Statement

The `open` statement creates a specified size space for a data file with the indicated number of records, assigns it a name, and places an end of file (EOF) mark at the beginning of each record in the file. Any data in the records used by this file is automatically erased when the file is opened.

```
open "name" , number of records
```

An `open` statement indicating file name and size must be executed before data can be printed in that data file. Each data file must be assigned a unique name.*

The size of a data file is specified in number of records; this parameter can be a numeric expression. A file can contain a minimum of one record (256 bytes) to a maximum of 1830 records. Opening files larger than the largest number of records (1830) results in error D8.

Once a file has been opened (i.e., created) and space has been reserved for it, the file can be used until it is erased with a `kill` statement.

The first statement in the example programs that follow (illustrating data storage and retrieval) is generally an `open` statement. The `open` statement is included only to remind you that data files must be opened before data can be printed in them. It is best to execute the `open` statement from the keyboard, rather than from a program, since error D5 results when you run the same program (i.e. try to open the same file) more than once.

*For more information about file names, see page 17.

The Files Statement

The `files` statement indicates which data files are to be used and optionally the number of the drive accessed (for each file name). The files listed are assigned numbers in the order in which they appear. File numbers are convenient labels used to reference specific files in print and read statements (discussed later).

```
files Name or * [# drive number]
[ , name or * [# drive number]][ , ...]
```

The position of the file name in the `files` statement determines the file number. For example, in the following statement, `Name` is assigned file number 1 and `Address` is assigned file number 2.

```
filesName,Address
```

Up to ten file names, each with an optional drive number, can be used in a `files` statement. Quotation marks enclosing file names are allowed in all statements except the `files` statement. String variables cannot be used for file names in the `files` statement. In addition, when the optional drive number is used in the `files` statement, it can be a numeric constant, only.

A single asterisk (*) can reserve space for a file in a `files` statement. Then an `assign` statement is used later to complete the file assignment by specifying the file name.

If a file specified in the `files` statement has not been previously opened, error D4 is displayed when the `files` statement is executed.

When the `files` statement is stored with an error, a flashing cursor is displayed showing the location of the error when the RECALL key is pressed.

The `files` statement enables you to access more than one drive simultaneously. This is useful when your program is on a disk in one drive and data is on disks in other drives. A single `files` statement can access files from different drives when the optional drive number parameter is used, as long as the select code is the same for all drives being accessed.

However, the drive number from the last `drive` statement is still in effect; only by executing a new `drive` statement can the drive number be changed.

The following program illustrates how data from a disk in drive 0 can be read and printed on a disk in drive 1. (Assume the file named give is open and contains data.)

```


0: drive 1
1: open "take",
  10
2: files give:0,
  take:1
3: for I=1 to 10
4: rread 1,I,A,
  B,C
5: rprt 2,I,A,B,
  C,"end"
6: next I
7: end
*14338

```

Data File Pointers

A maximum of ten data file numbers can be maintained using the `files` statement. File numbers speed up print and read operations since the file directory (containing file locations) does not have to be accessed each time a print or read statement is executed; instead file pointers corresponding to file numbers indicate file locations.

After executing a `files` (or `open`) statement, a file pointer is initialized to point to the beginning of the first record in the file. A new `files` statement obsoletes the previous `files` statement; it clears all previous pointers and sets up new ones for the drive, or drives, specified.

Executing a `drive` statement without a select code parameter has no effect on the file pointers. However, when the select code parameter is used in a `drive` statement, all file pointers are cleared. All file pointers are also cleared for a specific drive when the door to that drive is opened. Pressing  key, turning the calculator off or executing `erase a` also clears all file pointers.

When an `open` statement is executed, only the file pointer specified is affected; none of the other file pointers are affected.

The Assign Statement

The assign (`assign`) statement assigns a number (1 thru 10) to a single file name and, optionally allows a different drive number for the file specified. A return variable can also be used for further file information.

```
assign "name" : file number [ : drive number [ : return variable]]
```

The `assign` statement enables you to use a string variable for a file name and a variable or a numeric expression for a drive number.

The file number specifies the position of the file name in the `files` statement to be referenced in later print or read statements. The number must be a positive expression whose integer value is between 1 and 10. If it's not an integer, its rounded value is automatically taken.

An optional return variable can be used in an `assign` statement to determine a file's status. This parameter can be a simple or an array variable. For example –

```
0: files Name,*  
Number  
1: assign "Address"  
  ,2,0,K  
*31309
```

In this example, the asterisk in the `files` statement (in the second file position) is assigned the file name, `Address`. (Assume that the data file, `Address` was opened before the `assign` statement was executed.) Additional `assign` statements can be placed later in the same program to reassign a different file name to any file position. The `assign` statement overrides any `files` statement preceding it. An `assign` statement sets the data file pointer to the first item of the first record in the specified file.

A return variable can be used to determine the status or type of the file. In the previous example, `K` is the return variable. Its value is determined during execution of the `assign` statement and can be used anytime in the program. The value of the return variable indicates these conditions –

| Value of Variable | Meaning |
|-------------------|--------------------------------|
| 0 | file is available and assigned |
| 1 | file doesn't exist |
| 2 | program file |
| 3 | key file |
| 4 | file type not defined |
| 5 | memory file |
| 6 | binary program file |
| 7 | file type not defined |
| 8 | file number out of range |

By checking the value of the return variable, you can avoid errors like D4, file not found .

The following program shows how the `open` statement can be used –

```

0: dsp "This
   program is used
   to open";wait
   1000
1: dsp "new data
   files.";wait
   1000
2: dim A$(6);
   fxd 0
3: ent "Enter
   file name",A$
4: open A$,1,0,X
5: if X=1 goto 8
6: dsp A$,"was
   opened. New
   name"
7: goto 3
8: ent "Number
   of Records",J
9: open A$,J
10: prt A$,"is
   now opened.
   Records=",J
11: goto 3
12: end
*12773

```

In this example, line 5 instructs the calculator to branch to line 8 (to open the file) if the file name you enter does not exist.

A string variable cannot be used directly as a file name in a `files` statement, although it can be used in an `open` statement as shown in the previous program.

As shown in the example above, it is not necessary to use a `files` statement before using `open`. The `open` statement can be used to set the data pointer to the first item of a specified file without affecting file pointers for any other files previously specified.

The Serial Print Statement

The serial print (`sprt`) statement is used to store data on the disk. Data is printed serially in the specified file after the last item previously read or printed.

```
sprt file number, data items [, "end" or "ens"]
```

The file number refers to the number assigned using the `files` or `open` statement.

The data items in the serial print statement can be constants, variables, arrays, strings, substrings or text. The length of the list of data items is limited by the length of the HPL line (80 characters) or by the size of the file.

Either "end" or "ens" can be used as the last parameter in a serial print statement. Using "end" causes an end of file (EOF) mark to be written after the last data item printed. If "end" is not included, an end of record (EOR) mark is printed. This makes it easy to find out how much data is stored in a file using the `type` function or `on end` statement (explained later).

When "ens" is used as the last parameter, the data list is printed without printing either an end of record or end of file mark after it. Because of this, you should use the "ens" parameter with care.

Serial print and read statements can print and read past record boundaries (256 bytes); as many records as necessary are used to store the data listed.

The data file pointer moves through the file as you store or retrieve data items. Print statements overwrite EOF or EOR marks. As data is read (or written), the pointer moves to the next data item.

Here is an example using the serial print statement to record five student's identification numbers and test grades.

```
0: open "I.D.",1
1: open "Grades"
   ,1
2: files I.D.,
   Grades
3: for I=1 to 5
4: ent "Student"
   s I.D.#",X
5: sprt 1,X,"end
   "
```

```

6: ent "What is
   the next test
   score?";R
7: sprt 2;R;"end"
"
8: next I
9: end
*1319

```

This program can be used to print identification numbers in the file named *I.D.*, and the corresponding grades in the file named *Grades*.

| I.D. Number | Grade |
|----------------|-------|
| 1111 | 88 |
| 2222 | 67 |
| 3333 | 98 |
| 4444 | 81 |
| 5555 | 99 |

In the previous program, two separate files are used – one for the students' identification numbers and one for their grades. The information can be combined and stored in one file using the following program –

```

0: open "Scores"
  ,1
1: files Scores
2: for I=1 to 5
3: ent "Student'
   s I.D.#";X;"Gra
   de";R
4: sprt 1;X;R;
   "end"
5: next I
6: end
*18963

```

Line 4 prints the I.D. numbers and test scores of the students alternately in the file named *Scores*. Line 4 also places an EOF mark* after the five sets of data elements are printed.

*If "end" is a part of the data being stored, two "end" statements must be included in the program line to place an EOF mark where you want it. If "ens" is a part of the data being stored, two "ens" statements must be included.

A String Variable ROM enables you to enter student's names, instead of I.D. numbers. The I.D. variable, X, is replaced by a string variable and the following program prints string names as data on the file. The data is shown below –

| Name | Grade |
|----------------|-------|
| Rob Rood | 99 |
| Piper Aune | 90 |
| Carol Hafford | 88 |
| Andrew Jackson | 74 |
| Eric Landry | 80 |

Now key in and run the following program after installing a String Variable ROM in your calculator.

```

0: open "Class",
1
1: dim N$[15]
2: files Class
3: for I=1 to 5
4: ent "Student'
  s None",N$,"Gra
  de",R
5: spt 1,N$,R,
  "end"
6: next I
7: end
*951

```

To illustrate use of the "ens" parameter, the grade of 99 is changed to 100 by executing –

```

rread1,1;rread1,
N$
spt1,100,"ens"

```

The random read statement (described later) without a data items list is used to position the pointer to the beginning of the file where the correction is to be made. Then the data is read serially to locate the record and data item to be changed by reading and comparing data until the next item is the one to be changed. (In this case the first item is the one to be corrected.) The serial print statement then prints over the incorrect data item to change the 99 to 100. The "ens" parameter prevents the placing of an EOR or EOF mark after the corrected item.

The Serial Read Statement

The serial read (`sread`) statement reads numbers and strings serially from the specified file, starting after the last item read or printed.

`sread file number , data variables`

Before you can work with data which has been stored in a file, you must first read the data into the calculator. Remember that you are not erasing the data stored on the disk by reading it. Instead, data is copied using the variables specified.

The program on a previous page is used to print data on the files, I.D. and Grades. To read the data from these files back into the calculator and print the information, use the following program –

```
0: files I.D.,
  Grades
1: sread 1,S
2: sread 2,T
3: prt "I.D.#",S
4: prt "Grade":
  T,""
5: sto 1
6: end
*00855
```

In this program, the `files` statement serves two purposes – it establishes the files with their file numbers in the `sread` statement (lines 2 and 3) and it resets the pointers to the beginning of both files before the `sread` statements are executed. Here's the printout that results when the program is run –

| | |
|-------|---------|
| I.D.# | 1111.00 |
| Grade | 88.00 |
| I.D.# | 2222.00 |
| Grade | 67.00 |
| I.D.# | 3333.00 |
| Grade | 98.00 |
| I.D.# | 4444.00 |
| Grade | 81.00 |
| I.D.# | 5555.00 |
| Grade | .99.00 |

When this program is executed, error F0 is displayed because an attempt is made to read data after an EOF mark is reached.

Data printed in the file named `Class` (see the program on a previous page) can also be read back into the calculator. Use the following program to print this data –

```
0: dim P$(15)
1: files Class
2: for I=1 to 5
3: sread 1,P$,Y
4: prt "Student"
   ,P$,"Score",Y,
   ""
5: next I
6: end
*22479
```

Notice that the `sread` statement must specify the types of data (data elements or string variables) in the order in which they were originally stored in the file. Line 3 reads a string variable and then a score. This program can run only when the order of the data in the file named `Class` is known. Here's the printout –

```
Student
Rob Rood
Score      99.00

Student
Piper Aune
Score      90.00

Student
Carol Hafford
Score      88.00

Student
Andrew Jackson
Score      74.00

Student
Eric Landry
Score      80.00
```

The variables into which you read data items do not necessarily have to be the same variables from which you printed the data items in the file. Although the variable name changes (from N\$ and R when stored, to P\$ and Y, when retrieved), the order in which the two data types are accessed and the types themselves are the same.

When the serial read statement encounters the EOF mark, previously placed by the last print statement, the program ends and error F0 (file overflow) is displayed. The program can be written to end without displaying an error when the `on end` statement (described later) is used.

The Random Print Statement

The random print (`rprt`) statement is used to store individual data items in specified records within a file.

```
rprt file number, record number [, data items, "end" or "ens"]
```

As in serial file access, a pointer keeps track of the data item currently being accessed. Unlike serial file access, however, in random file access a specific record number within a file must be included in each random print and random read statement. The pointer is positioned at the beginning of the specified record before printing or reading occurs. Data is printed or read consecutively from the beginning of the record and cannot be read past the end of record mark.

The record number represents the location of a record in a specific file. This number can be any integer or expression which does not exceed the number of records in the file. The `rprt` statement prints data items in the form of variables, numbers, strings or substrings of characters from the beginning of the specified record. Using the `rprt` statement, each record can hold only 256 bytes (1 record) of data.

Either `"end"` or `"ens"` can be used as the last parameter in an `rprt` statement. Using `"end"` causes an end of file (EOF) mark to be written after the last item in the statement is printed. If `"end"` is not included, an end of record (EOR) mark is printed. This makes it easy to find out how much data is stored in a file using the `type` function explained later.

When `"ens"` is used as the last parameter, a list of data items is printed without printing an EOR or EOF mark. Because of this, you should use the `"ens"` parameter with care. (See the `"ens"` example on page 40.)

The program below prints consecutive numbers in each odd numbered record of a 10 record file named Ten.

```

0: open "Ten",10
1: 1→R→A
2: files Ten
3: if R>10!end
4: rpt 1,R,A,
   "end"
5: A+1→A
6: R+2→R
7: sto 3
*31153

```

In line 4, the record number is specified by the variable R. Line 6 increments this variable by 2 so that only odd numbered records are accessed.

By printing in specific records of file Ten, previous data in those records is erased and replaced by new data. An EOF marker is automatically placed at the end of each data list (i.e., the one data item A) in each odd numbered record.

File Ten now contains the following information.

| Record Number | Data |
|------------------|-------|
| 1 | 1 EOF |
| 2 | EOF |
| 3 | 2 EOF |
| 4 | EOF |
| 5 | 3 EOF |
| 6 | EOF |
| 7 | 4 EOF |
| 8 | EOF |
| 9 | 5 EOF |
| 10 | EOF |

The following program erases every third record of file Ten, which was opened and accessed previously.

```

0: 1→A
1: files Ten
2: rpt 1,A
3: A+3→A
4: if A>10!sto 6
5: sto 2
6: end
*20859

```

The information which is now left in the file is shown below.

| Record Number | Data |
|------------------|-------|
| 1 | EOR |
| 2 | EOF |
| 3 | 2 EOF |
| 4 | EOR |
| 5 | 3 EOF |
| 6 | EOF |
| 7 | EOR |
| 8 | EOF |
| 9 | 5 EOF |
| 10 | EOR |

The "ens" parameter can be used to write data in a record without placing any EOR or EOF marks after the item printed.

When the data list is omitted in an `rpwt` statement, an EOF mark is stored in the beginning of the record.

```
rpwt file number ; record number ; "end"
```

This EOF mark in the beginning word of the specified record is the same as the EOF mark placed automatically when the file was originally opened. The EOF mark makes the data contained in the record beyond the EOF mark inaccessible to any serial read or random read statement.

The Random Read Statement

The random read (`rread`) statement reads numbers and strings from a specified record in a file, starting from the beginning of that record. A variation of this syntax can be used to reposition the file pointer (see page 47).

```
rread file number, record number [: data variables]
```

As in the case of serial read statements, the variables into which you read data items do not necessarily have to be the same variables used to print the data items in the record, but they must be the same type and in the same order.

The following program reads the data printed in the 5th and 9th records of the file named Ten.

```
0: files Ten
1: rread 1,5,X
2: rread 1,9,Y
3: prt "Data
   item 1=",X
4: prt "Data
   item 2=",Y
5: end
*7591
```

This data was originally printed in odd numbered records of file Ten (see page 44). The data in records 1 and 7 was erased. The program above reads the data from records 5 and 9 and then prints it using the calculator printer. If the calculator were programmed to read data from each record, `error F0` would be displayed, indicating that an EOR mark was detected at the beginning of record 1.

The printout from this program is shown below.

```
Data item 1=
          3.00
Data item 2=
          5.00
```

Positioning the Pointer

As mentioned earlier, pointers are maintained by the disk system to specify where data storage or data retrieval begins.

The pointer can be positioned at the beginning of a specified record in a file by executing a random read statement without a data list —

```
rread file number, record number
```

This positions the pointer at the beginning of the specified record. A serial read statement can then be executed, to access the beginning of the first data item of that record.

The pointer is automatically positioned at the beginning of the first record in a file after execution of a `files` or an `open` statement. It is positioned at the next available storage location in the file after execution of a random or serial print statement.

To see how this works, first use the following program to store consecutive numbers beginning from the eleventh record of a 15 record file named Data 15.

```
0: open "Data15"
   ,15
1: files Data15
2: I→I
3: rread 1,11
4: spnt 1,I,"end
   "
5: I+1→I
6: sto 4
7: end
*22700
```

The `files` statement (line 1) sets the pointer to the beginning of the first record in the file. The pointer is repositioned to the beginning of the eleventh record of Data15 by executing line 3.

After printing in records 11 thru 15 of Data15, error F0 is displayed. This indicates that the end of the file has been reached and no additional data can be printed in the file.

The following program is used to read the data from the beginning of record 14 –

```
0: files Data15
1: rread 1,14
2: sread 1,A,B,
  C,D,E,F,G,H
3: prt A,B,C,D,
  E,F,G,H
4: sto 2
5: end
*24282
```

The `files` statement (line 0) automatically sets the pointer to the beginning of the first record. The pointer is repositioned to the beginning of the fourteenth record in Data15 by executing line 1. The serial read statement begins reading data from that point on.

Since each full precision number uses 8 bytes of memory, 32 numbers can be printed in a (256 byte) record. On the file named Data 15, the following numbers are stored on these corresponding records.

| Record Number | Numbers |
|---------------|---------|
| 1 thru 10 | none |
| 11 | 1-32 |
| 12 | 33-64 |
| 13 | 65-96 |
| 14 | 97-128 |
| 15 | 129-160 |

The previous program reads the data on records 14 and 15 (i.e., numbers 97 thru 160) and lists this information as shown in the printout below.

```
97.00
98.00
99.00
100.00
101.00
102.00
103.00
104.00
105.00
106.00
107.00
108.00
109.00
110.00
111.00
112.00
113.00
114.00
```

115.00
116.00
117.00
118.00
119.00
120.00
121.00
122.00
123.00
124.00
125.00
126.00
127.00
128.00
129.00
130.00
131.00
132.00
133.00
134.00
135.00
136.00
137.00
138.00
139.00
140.00
141.00
142.00
143.00
144.00
145.00
146.00
147.00
148.00
149.00
150.00
151.00
152.00
153.00
154.00
155.00
156.00
157.00
158.00
159.00
160.00

error F0 is displayed at this point, indicating that an EOF mark is reached and that there is no remaining data to be read. This error message can be avoided by using the `on end` statement, discussed later.

The On End Statement

The `on end` statement sets up a branching condition in the program. This avoids error F0, file overflow, making it possible to use a file whose exact contents are unknown.

`on end file number : line number or "label "`

The program branches to the line number specified in the previous `on end` statement if –

- an EOF mark is encountered during execution of an `sread` statement, or an attempt is made to `sprt` past the (physical) end of a file.
- an EOR or EOF mark is encountered during execution of an `rread` statement or an attempt is made to `rpri` past the (physical) end of a record.

In the previous program, error F0 is displayed after the completion of the program, telling you that an EOF mark is encountered and no more data can be read. This error message can be avoided by including an `on end` statement in the program.

When an end of file mark is reached, the program branches to the line number specified in the `on end` statement. The rest of the current line is not executed. This branching condition prevails until another `on end` statement, with a different line number parameter for the same file, is executed. All previous `on end` statements are cancelled when a `files` statement is executed, while an `open` statement cancels the `on end` statement only for the individual file specified in the `open` statement.

The program from the previous page is modified to include an `on end` statement and is shown below.

```
0: files Data15
1: rread 1,14
2: on end 1,6
3: sread 1,A,B,
   C,D,E,F,G,H
4: prt A,B,C,D,
   E,F,G,H
5: goto 3
6: prt "","End
   of File"
7: end
*4170
```

*If the line number or label to which the `on end` statement refers does not exist, error 31 is displayed.

In this program, when all the data is read, the pointer comes to an EOF mark. The `on end` statement (line 2) causes the program to branch to line 6 when the read statement (line 3) encounters the EOF mark. At this point, lines 6 and 7 are executed, informing you that the EOF mark is the next item in the file.

For fastest execution, the `on end` statement need be executed only once before entering the read/print loop. This is because the program would be slowed considerably and unnecessarily if the `on end` statement were executed each time the loop is executed.

An `on end` statement sets up a condition to be executed when an EOR or EOF mark is detected to change the program flow. If you attempt to access a non-existent or invalid record without a previously executed `on end` statement, error F0 is displayed. If a `files` or `open` statement has not yet been executed, error F6 results when the `on end` statement is executed.

When using the `on end` statement, it is better to use labels since the line number parameters are not changed following program editing. Therefore, in the previous example, if line 4 were deleted, the `on end` statement would still access line 6 instead of line 5 as it should.

The Type Function

The `type` function is used to identify the type of the next item in a specified file.

```
type ([-] file number )
```

The `type` function indicates what the next data item is by returning a value. It can be used before a read statement for this purpose. The values returned and their meanings are listed below.

| Value | Meaning |
|-------|---|
| 0 | type undefined |
| 1 | full precision number |
| 2 | string (complete in one record) |
| 2.1 | the first part of a string (which overlaps record boundaries) |
| 2.2 | an intermediate part of string (which overlaps record boundaries) |
| 2.3 | the end part of a string (which overlaps record boundaries) |
| 3 | end of file mark |
| 4 | end of record mark |

If `type` is executed with a positive file number, values 0 thru 3 can be returned; if `type` is executed with a negative parameter, a 4 can be returned in addition to the other values listed. A negative parameter in the `type` function is useful for detecting EOR marks when `rread` statements are used. When a positive parameter is used with serial reads, EOR marks are ignored and the type of the first item in the next record is found.

The following programs illustrate the `type` function -

```
0: open "type1",
  5
1: files type1
2: dim A[1],B#[2
  0]
3: 1111→A[1]
4: "XXXXXXXXXX"→
  B#
5: sort 1,A[1],
  B#
6: end
*29800
```

```
0: files type1;
  dim A#[1000]
1: "A":jmp int(t
  ype(-1))+1
2: prt "unknown
  type";stp
3: sread 1,X;
  prt "number",X;
  sto "A"
4: sread 1,A#;
  prt "Strings",
  A#;sto "A"
5: prt "EOF Mark
  ";stp
6: type(1)→X;
  prt "EOR Mark";
  sto "A"
7: end
*31129
```

If the (absolute value of) the `type` function parameter is negative, the type of the item at the file pointer is returned without moving the file pointer. If the (absolute value of) the `type` function parameter is positive and the item at the pointer is an EOR mark or the (physical) end of the record, the file pointer is moved to the beginning of the following record and the type of that item is returned, (even if it is another EOR mark). The `type` function with a positive parameter (in line 6) is used to step over an EOR mark.

Data Storage Requirements

A full precision number requires eight bytes of memory for storage; this means that 32 full precision numbers can be stored in one 256 byte record. An EOR mark requires two bytes of memory for storage except when the items in the record exactly fill the record (256 bytes). When the items exactly fill the record, an EOR mark isn't needed since the actual end of the record and the end of record mark are not differentiated by the read statements. On the other hand, an EOF mark is always written, to the next record, if necessary, and requires two overhead bytes.

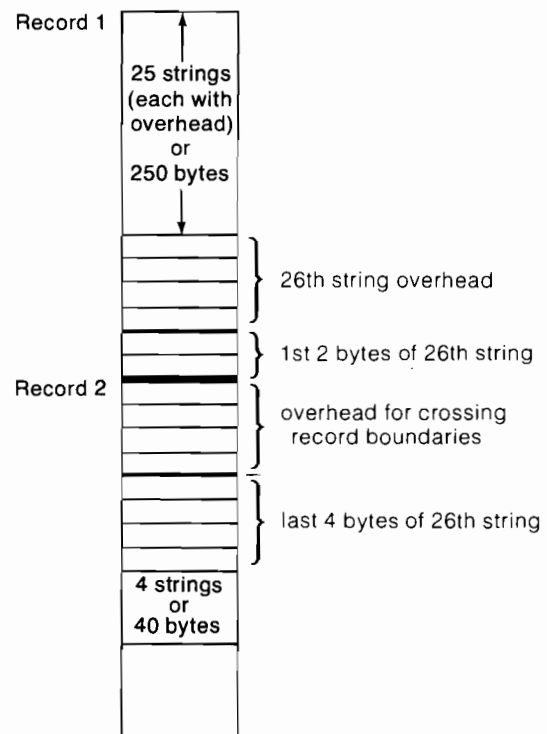
A string requires an extra four bytes of memory, called overhead, when stored, aside from the normal one byte per character storage requirement (plus one extra byte if the number of characters is odd). Therefore, up to 252 bytes of a string (plus 4 bytes of overhead) can be stored in one 256 byte record.

Strings or string arrays can overlap record boundaries using the serial print statement. Four extra overhead bytes are required for each extra record used to store the string or each string of a string array.

For example, to store a string array, A\$[30,6], each string of the array requires –

- 4 bytes of overhead (per string)
- 1 byte if the number of characters is odd
- 4 bytes of overhead whenever the string crosses record boundaries

Therefore 30 strings, 6 characters long, each require 4 bytes of overhead for a total of 300 bytes, plus 4 extra bytes where the string crosses record boundaries. (If a string exactly fills a record, the overhead for crossing record boundaries is not required.)



Summary of EOR and EOF Marks

Serial Printing

When a file is opened, EOF marks are placed at the beginning of each record. This stops any read statements but allows print statements. At the right a file with four records is opened.

| | |
|---|-----|
| 1 | EOF |
| 2 | EOF |
| 3 | EOF |
| 4 | EOF |

Using a serial print statement, 20 numbers are stored in the first record of the file. If the "end" parameter is used, an EOF mark is printed after all data items have been stored in the file, as shown.

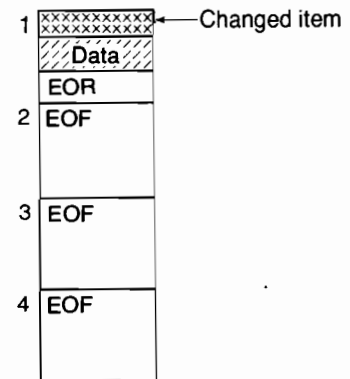
| | |
|---|------|
| 1 | Data |
| | EOF |
| 2 | EOF |
| 3 | EOF |
| 4 | EOF |

If the "end" parameter is omitted from a print statement, an EOR mark is printed after all items (20 numbers) are stored, as shown.

| | |
|---|------|
| 1 | Data |
| | EOR |
| 2 | EOF |
| 3 | EOF |
| 4 | EOF |

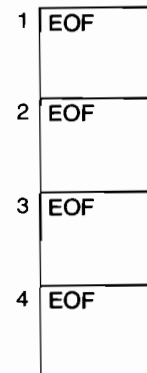
If the "ens" parameter is used in a print statement (to change the first number in the data list from the previous illustration, for example), no mark is printed after the data item (or items) is stored.

The data file pointer is left where it finished when a serial print or read statement is executed.

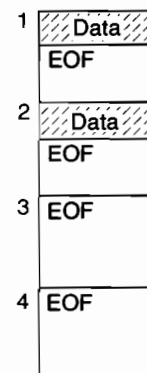


Random Printing

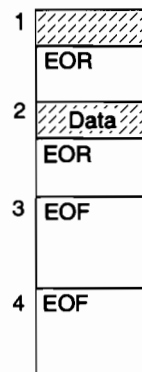
At the right a file with four records is opened and EOF marks are automatically placed at the beginning of each record.



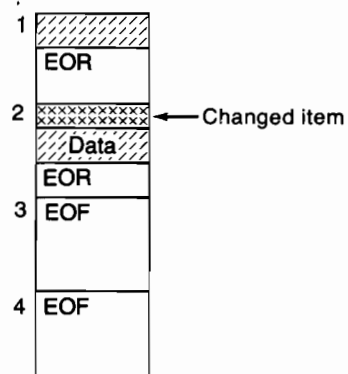
Using a random print statement, numbers are stored in the first two records of the file. If the "end" parameter is used, an EOF mark is printed after the data items in each record, as shown.



If the "end" parameter is omitted from a random print statement, an EOR mark is printed after the data items in each record, as shown.



If the "ens" parameter is used in a random print statement (to add to the first data item in the second record from the previous illustration, for example) no mark is printed after the data item is stored. Since all data in the file was not changed, the EOR mark remains unaffected. If more than 20 numbers were changed in this example, the EOR mark would be written over and not replaced leaving no mark at the end of all of the data.



Chapter **5**

Other Operations

Additional Statements

The Save Memory Statement

The save memory (`save`) statement stores the calculator's entire read/write memory (program, data, keys, pointers, etc.) in a disk file. A file large enough to store the memory is automatically allocated when the save memory statement is executed.

```
save "file name "
```

To store the calculator's memory in a file named `Memory`, for example, execute -

```
save "Memory"
```

The Get Memory Statement

The get memory (`get`) statement loads a previously recorded memory file (read/write memory) from the disk and returns the calculator to the state when save memory was executed.

```
get "file name "
```

The save and get memory statements are extremely useful in areas where frequent power interruptions occur.

To restore the calculator's memory file just saved using the save memory statement, execute -

```
get "Memory"
```


The Repack Statement

The repack (`repack`) statement moves all user files to the beginning of the user area on a disk leaving a single continuous available space for more efficient use of disk space.

`repack`

This statement consolidates all unused space (from files previously killed) into one area, making future use of the disk more efficient. Once a repack statement is executed and the files are consolidated, execution of disk statements becomes faster, since the average distance between user files is decreased.

A minimum number of bytes (1666) in the calculator must be available to execute the repack statement, or error 40 results. For fastest results, execute `erase a` before doing a repack.

Most calculator operations can be executed in seconds except for a long repack (where many empty spaces between files exist). This is because the directory is updated after each file is repacked. The amount of available memory also affects the repack statement: the more available memory there is, the faster repack is executed. (This is because a larger number of records can be moved at one time requiring fewer move operations.) Repack can take from a few seconds to about 15 minutes. If a power failure occurs during execution of a repack statement, one file may be lost.

If `repack` encounters a checkword (d7) or header (d5) error during its execution, it will beep and continue repacking. An unlimited number of checkword errors are allowed. One header error is allowed; if a second one occurs repacking stops and the error is displayed. Otherwise, any error(s) that occur will be displayed when `repack` is done. If more than one checkword error occurs, the last error is the one displayed.

The Verify Statements

The verify statements enable (`von`) and disable (`voff`) the verify mode and affect only the print and `copy` statements. The verify mode reads user data under stricter specifications (a shorter amount of time) and then compares the data on the disk to the data in the calculator's memory. Its purpose is to decrease your error rate and increase your level of confidence about the accuracy of your data.

The verify mode is disabled when the calculator is turned on.

`von` enables the verify mode
`voff` disables the verify mode

Data is stored on the disk in the form of binary digits represented by magnetized spots on the rotating disk. After the record is read, the checkword written at the end of the record is compared to a checkword generated by reading the record, as a check for data validity. In the verify mode, the disk is read under more stringent conditions: the time allowed to read each magnetized spot (to decode it into a 1 or a 0) is shortened. This is known as reading under a tight margin.

If your data cannot be verified, `error d9` or `error D9` is displayed. This means that your data cannot be read and must be reprinted on the disk. `error D9` indicates that data, as it appears on the disk, is not marginal and has no checkword error, but does not compare with the data just written on the disk. This may be caused by a bad interface cable connection, or a problem with the drive or calculator. If your data is marginal, `error d9` is displayed. In this case your data is readable but may not be readable for long and should be copied to a new area using the `COPY` statement.

When the verify mode is disabled, the `COPY` statement (explained next) can be executed in a shorter amount of time. The same is true of the print statements.

The Copy Statements

The `COPY` statements enable you to duplicate an entire disk, a specific data file, or part of a file depending on the parameters used.

The speed of the `COPY` statement depends on the amount of available memory (the more memory, the faster the execution), whether verify is on or off (off is faster) and the type of `COPY` being done (partial file `COPY` with parameters that don't require writing or checking for extra EOFs* is fastest). There are no minimum memory requirements. Execute `erase a` before executing `COPY` (if possible) for fastest results.

Disk Copy

The disk `COPY` statement duplicates the entire contents of a source disk onto a destination disk. The destination disk must have as many, or more, usable tracks as the source disk or the `COPY` cannot be executed. The destination disk becomes an exact duplicate of the source disk.

```
COPY source drive number [ , source select code] , "to" ,
    destination drive number [ , destination select code]
```

The "to" is required to distinguish disk `COPY` from the other `COPY` statements.

*This is because `COPY` with those parameters requires only one bootstrap. Therefore if several such `COPY` statements are executed without intervening statements which also use bootstraps (like `type` and `sread` or `rread` with string arguments) the system doesn't need to fetch bootstraps repeatedly. All other `COPY` statements require more than one bootstrap.

File Copy

The file `COPY` statement duplicates the contents of a specified file (program, data, memory or key) into another file.

```
COPY "source file name" [ : drive number [ : select code] ]
      "destination file name" [ : drive number [ : select code] ]
```

The default drive number and select code are the current system values if the optional parameters are omitted until `RESET` is pressed, `erase a` is executed or the calculator is turned off. When any of these occur, the default values become drive 0, select code 8.

When the source file is not a data file, the destination file is automatically opened (created) on the destination disk by the file `COPY` statement. The destination file cannot exist before the file `COPY` statement is executed.

When the source file is a data file, the destination file may or may not exist before the file `COPY` statement is executed. If it doesn't exist, it is automatically opened (created) by the file `COPY` statement to be the same size as the source file.

If the destination file is longer than the source file, an EOF mark is printed in the beginning of each of the extra records in the destination file.

If the source file is longer than the destination file, then all extra source file records must begin with EOF marks, otherwise `error F0` is displayed and none of the file is copied. If all extra source file records begin with EOF marks, the file is copied. EOF marks found in the source file records which are not extra do not affect copying.

Partial File Copy

The partial file `COPY` statement duplicates portions of a specified source data file (which have been assigned using a `files` or `asn` statement) into a destination file.

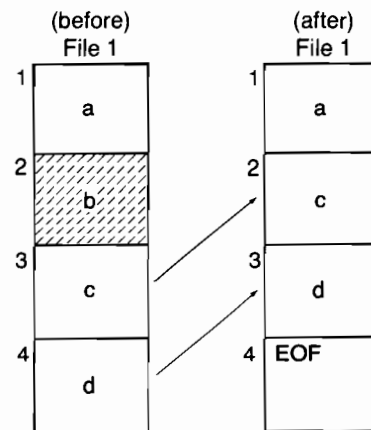
```
COPY source file number : source beginning record number : destination file number
      : destination beginning record number [ : number of records]
```

After executing a partial file `COPY` statement, the file pointers for each file involved point to the beginning of their respective files. Records before the starting points and after the ending points are ignored and unaffected.

For example, to delete a record from a file, say record 2 from a four record file, the third and fourth records are moved up to replace the second and third records and then an EOF mark is printed. This can be done by executing -

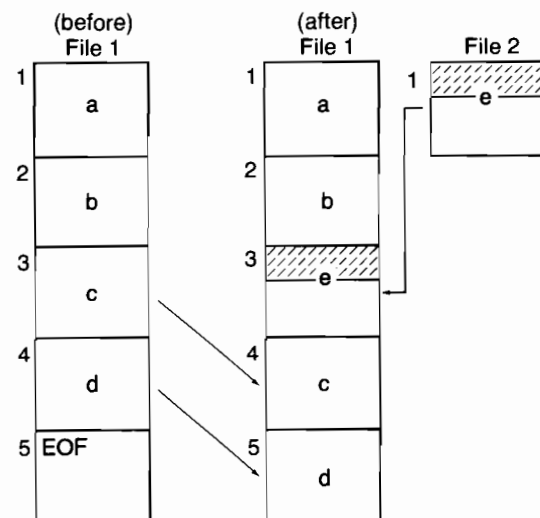
```
files File1
copy 1,3,1,2
```

This copies records 3 and 4 into records 2 and 3 and puts an EOF mark in record 4. An optional 2 can be added to the statement above to indicate that two records are to be copied. In that case, the EOF will not be written in record 4.



To add a record to a file, say a new record 3 to a five record file, you can move records 3 and 4 down to the fourth and fifth positions in the file and add a new record 3. To do this, execute -


```
files File1, File2
copy 1,3,1,4
copy 2,1,1,3,1
```



The first `COPY` statement copies file 1, records 3 and 4 to record 4 and 5 and checks record 5 to make sure it begins with an EOF mark. An optional 2 can be added to this statement to indicate that two records are to be copied. In that case, record 5 is not checked for an EOF mark. The second copy statement copies the first record in file 2 to the third record in file 1. The last parameter, 1, indicates that only one record is copied. The `COPY` statement can also be used to check for an empty data file (where N = an integer between 1 and 10) by executing `copy N,1,N,9999`. The `COPY` statement can be used to erase data in a file without killing the file itself by executing `copy N,9999,N,1`.

The Dump Statements

The `dump` statements store the entire disk or indicated data files from the disk to the specified tape files.

If you execute `load` or `dump` from the keyboard, a beep is used to indicate your response to the messages displayed when you press . However beeps that occur during the actual dumping or loading process are similar to the beeps that occur during a repack operation. (See the Repack Statement.) The only difference is that any number of header errors are allowed. However, `dump` and `load` proceed only through disk errors; if a tape error occurs, `dump` or `load` stops immediately and the error is displayed.

Disk Dump

The `disk dump` statement transfers the entire disk onto two or three tape cartridges starting with track 0, file 0 of the first tape cartridge.

```
dump [-]1[0]
```

The optional argument following the `dump` statement indicates whether the tape is to be marked automatically, or not, and the number of disk records to be dumped per tape file.

- If the argument is positive, the tape is marked automatically.
- If negative, you must have marked the tapes before executing `dump`.
- 1 or 10 determines the number of records to be dumped into a tape file.

If 10 is used, 2560 bytes of memory must be available when the `dump` (and corresponding `load` statement) is executed. (Otherwise, error 40 is displayed.) If 1 is used, there is no memory requirement, but the `dump` (and corresponding `load`) statement takes longer to execute and requires more tape.

- The default value is +10.

The size of the tape file is 256 bytes when the parameter 1 is used and 2560 when 10 is used. To `dump` an entire disk, normally three tape cartridges are required when the parameter is 1 and two cartridges when the parameter is 10.


File Dump

The file `dump` statement transfers the indicated data file from the disk to the specified tape file.

```
dump "data file name" : tape file number [, [-]1[0]]
```

The optional parameter has the same effect here as for the disk `DUMP` statement.

The `DUMP` statement transfers disk information on the current track (starting at the first file number given) until the current tape track is filled (or the null file is reached, for premarked tapes). After the track is filled, if the current track is 0, `DUMP` automatically rewinds and continues with track 1, file 0. If the current track is track 1 `DUMP` displays `NEXT TAPE, PRESS CONTINUE`, waits for tapes to be changed and then continues with track 0, file 0 of the new tape.

If the optional parameter is not given or is positive, and you insert a write protected tape, `SET RECORD, PRESS CONTINUE` is displayed. Pull out the cartridge, slide the Record tab to the record position and reinsert the cartridge and press .

The Load Statements

The disk `LOAD` statement transfers the entire disk from the tape files created by the disk `DUMP` statement.

Disk Load

The disk `LOAD` statement transfers the entire disk from the tape files used in the disk `DUMP` statement.

`load`

The disk is loaded starting from tape track 0, record 0. If the `DUMP` used ten records per tape file, there must be 2560 bytes of available memory to perform the load, or `error 40` results. With one record per tape file, there are no memory requirements.

File Load

The file `LOAD` statement transfers data starting from a specified tape file on the current track into the disk data file named. The tape file must have been created by a file `DUMP` statement.

`load "disk data file name" , tape file number`

The data file is transferred starting at the specified tape file number on the current track. The tape file number must be the same as for the corresponding `DUMP` statement. If the file `DUMP` is used ten records per tape file, there must be 2560 bytes of available memory to perform the `load`, or `error 40` results. With one record per tape file, there are no memory requirements. The size of the disk file must be greater than or equal to the size of the original file from which the data was dumped, otherwise `error D8` is displayed. Any extra records in the destination file are not affected.

The Get Binary Statement

The get binary (`getb`) statement loads a binary program from a disk file into the binary program area of the calculator memory.

```
getb "file name "
```

Error Recovery Routines

A binary program and user language program containing the Error Recovery Routines can be found on your Disk System Cartridge. Error recovery routines enable you to recover from soft (non-hardware) and sometimes hardware failure errors. Using these binary routines, you can sometimes read these records or tracks and recover your data from them.

IMPORTANT

If a recoverable error occurs (`error d5`, `error d6` or `error d7`) it is important that you load the Error Recovery Routine immediately so that the error information is not lost. If the calculator is turned off, `erase a` is executed or `rprt`, `srpt` or `copy` (in the verify mode) is performed, the error recovery information you need will be lost.

If there is no error, the message `error info lost` is printed. If there is an error, the error number, select code, track number and record number are printed. If the error occurs during the transfer of a number of records, the error number and approximate location are printed. For example –

```
error d5 was
somewhere
between
S8 D0 T9 R27
S8 D0 T11 R7
```

`error d5` Error Recovery

When `error d5` occurs, data in a record can't be read because the record header is lost. To recover your data, load and run the Error Recovery routine by executing –

```
ldbl .
```

You can then read and load the contents of the track into tape cartridge files, 1 record per tape file, by first marking (30 X number of tracks involved) files, each 256 bytes in length and then executing –

```
dtrk beginning tape file number
```

Using the binary program, data can be read because the header errors are ignored. This track (or tracks) can be reinitialized by executing –

```
tinit *
```

To return your data from the tape files to the reinitialized track, execute –

```
ltrk beginning tape file number
```

error d6 Error Recovery

When error d6 occurs, the calculator cannot read the data on the track because it can't recognize any of the headers on the track. The data on this track is effectively lost. When error d6 occurs, load and run the binary Error Recovery routine by executing –

```
ldb1
```

(If possible, execute dtrk to dump the track, although this may not always work since the calculator may not be able to locate the track because of the loss of the headers.)

Then reinitialize the track by executing –

```
tinit
```

If the track lost is track 0, the spare directory can be copied to the disk in the main directory area without affecting anything else on the disk. This can be done by executing –

```
dirc
```

The latest drive number and select code are used as the default values for the dirc statement.

*The track initialization statement can be used with an optional track number, but the optional track number parameter should be used carefully to avoid losing data by reinitializing the wrong track.

```
tinit [track number]
```

tinit without the optional number automatically initialized the track (s) in which the error occurred.

error d7 Error Recovery

When error d7 occurs, data in a record can't be read because the checkword for that record is not identical to the checkword generated when the record was read. To read and correct the data from the record, load and run the binary and HPL Error Recovery routine by executing –

```
ldb1
ldf2
run
```

The HPL program first asks for the track and record numbers where the error occurred. The entire contents of the record is then printed out, item by item. For each item, the item number, type number (see the Type Function table, page 51) and the item itself are printed. Then the question `type?` is printed.

- If the type and the contents are correct, press . The calculator skips to the next item in the record, prints it out along with its type number and repeats the question, `type?`.
- If the type is correct, but not the contents (the item itself) enter the same type number and press . The item itself (value, string, partial string or EOR/EOF mark) is then questioned, e.g. `mantissa?` and `exponent?` or `length?`. The incorrect number, string or EOR/EOF mark can be corrected by keying the correction and pressing . As a final check, the item is reprinted and `type?` is displayed so that another correction can be made, if necessary. Then the whole procedure is repeated for the next item in the record.
- If neither the type or the contents are correct, the correct type number can be keyed in and executed. Then the item itself can be corrected. As a final check, the item is reprinted and `type?` is displayed, so that another correction can be made, if necessary. Then the whole procedure is repeated for the next item.

This is repeated for the entire record (32 numbers or the characters of a string, etc.) until the end of the record is reached. Then the calculator prints `0-back 1-ok?`. If 0 is pressed, the entire contents of the record is displayed again, item by item, to verify that all corrections were made. If 1 is pressed the record is written back to the disk.

There are additional error messages associated with these routines which can be found on page 68.

Binary Programs

Other binary programs stored on the Disk System Cartridge enable you to—

- Initialize a blank disk
- Load only the bootstraps on a disk (already initialized) without destroying data in the storage area.
- Verify the boots stored on the cartridge with boots on the disk.
- Kill all user files without destroying initialization and the boots on the disk.

There are additional error messages associated with these routines which can be found on page 68. The routines can be used once `ldbo` is executed.

The Initialization Routine

The initialization statement (`init`) enables you to initialize a blank disk as described in the Appendix, page 90.

The Bootstrap Routine

The bootstrap statement (`boot`) enables you to load a new set of boots on a previously initialized disk without destroying data in the storage area. This routine is described in detail in the Appendix, page 90.

The Verify Boots Routine

The verify boots statement (`vfxb`) allows you to verify the boots on your disk against the boots on the Disk System Cartridge.

The KillAll Routine

The killall statement (`killall`) erases all user files from the disk without affecting the initialization or bootstraps on the disk.

Error Messages

These errors may result during the binary Initialization and Error Recovery Routines

- error B0 Wrong syntax, argument out of range or variable not properly dimensioned.
- error B1 More than six defective tracks on the disk.
- error B2 Verify error. Boots on the disk not identical to boots on the cartridge.
- error B3 dtrk, tinit or ltrk not allowed because error information is lost or error is not d5, d6, d7 or d9.
- error B4 Attempt to access a record for error correction which isn't part of the data file.
- error B5 Improper string length (inconsistent with given length in header).
- error B6 Not enough space in record buffer for data item or item can't be placed in this part of buffer.
- error B7 Missing Disk or String ROM
- error B8 Track still bad after tinit.

Appendix **A**

Disk Specs and Care

Specifications

Disk Capacity

The following table lists the number of bytes of memory required to store full precision data elements and string variables. Strings and numerics can be mixed within a record.

468,480 bytes - maximum usable storage per disk

1830 records - maximum usable storage per disk

352 files - maximum number of files per disk

58,560 full precision numbers per disk

461,160 string characters per disk

256 bytes per record

8 bytes per full precision number

1 byte per string character*

Disk Speed

360 revolutions per minute - disk speed

267 ms - average access time

$$(\text{a.a.t.} = \left(\frac{\# \text{ of tracks} - 1}{3} \right) (\text{step time}) + \left(\frac{\text{revolution time}}{2} \right) + (\text{head settling time})$$

$$267 = \frac{66}{3} \times 8 + \frac{166}{2} + 8$$

23,000 bytes per second - transfer rate (for numerics)

46,000 bytes per second - maximum transfer rate

62,000 bytes per second - instantaneous speed

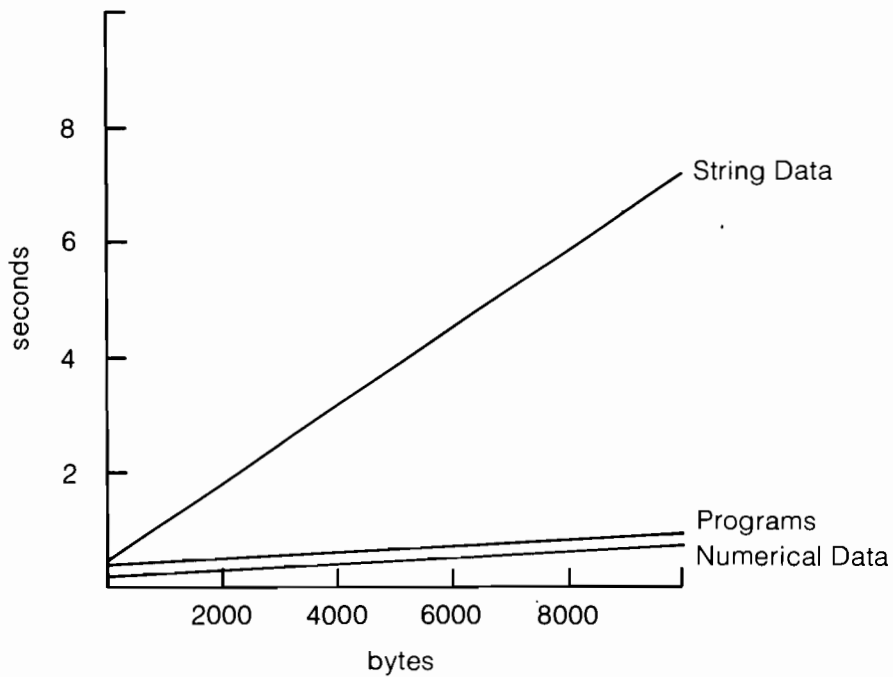
1 byte per 8 microseconds - transfer rate

*Plus overhead of four bytes per string and four bytes for each additional record used to record the string (and one extra byte if the number of characters in the string is odd).

Transfer Times

To calculate the approximate time it takes to transfer programs, numerical data and string data, you must know the transfer rate (23,000 bytes per second), the access time (267ms) and the number of records per track (30).

Transfer Times



| | Bytes | | |
|-----------------|----------|-----------|-----------|
| | 1000 | 5000 | 10000 |
| Program Data | .58 sec. | .75 sec. | .97 sec. |
| Numerical Data* | .31 sec. | .48 sec. | .70 sec. |
| String Data | .96 sec. | 3.75 sec. | 7.22 sec. |

*Records with less than 70 string characters.

To transfer a program to the disk, the access time is doubled (because the directory and bootstraps must also be accessed) and then added to the number of bytes divided by the transfer rate. For example, transferring a 10,000 byte program takes, on the average -

$$2 \times .267 + \frac{10,000}{23,000} \text{ or } .97 \text{ seconds}$$

To transfer numerical data and to print and read strings, the access time is added to the number of bytes divided by the transfer rate. Therefore, transferring 5000 bytes of numerical data takes, on the average -

$$.267 + \frac{5,000}{23,000} \text{ or } .48 \text{ seconds}$$

To transfer string data over 70 characters, the access time is added to the number of bytes multiplied by 16* and is then divided by the transfer rate. Transferring 1000 bytes of string data takes, on the average -

$$.267 + \frac{1,000}{23,000} \times 16 \text{ or } .96 \text{ seconds}$$

*Because an extra revolution is required to access the next record.

Disk Care

Guidelines

The Flexible Disk is basically maintenance free, but should be handled with care. Here are some guidelines to avoid loss of data or damage to your disks. By following these suggestions, you'll greatly improve the reliability of your disks.

CAUTION

Use only HP approved disks since use of others can result in damage to your drive. (Contact your local HP Sales and Service Office for a list of recommended manufacturers.)

- Replace worn disk envelopes and always return disks to their storage envelopes after removing them from the drive to protect them from damage. A looseleaf notebook is provided for disk and envelope storage. Envelopes can be ordered from your HP Sales and Service Office.
- Since fingerprints on the disk can cause loss of data, avoid touching the surface of the disk showing through the protective sealed jacket.
- Avoid writing on the sealed plastic jacket with lead pencil or ball-point pen. Use a soft felt tip pen and write on the label only.
- Although the disk is flexible, do not bend or fold it since this, too, can cause damage to the disk.
- Never subject disks to temperatures below 10°C (50°F) or above 52°C (125°F) or relative humidity in excess of 20% to 80%.
- Contamination from dust, ashes smoke etc. can damage disks.
- Avoid placing disks in strong magnetic fields like those produced by transformers or magnets, since this can cause loss of data.
- Never remove disks from their sealed protective jackets.
- The inside surface of the sealed protective jacket is coated with a special material that cleans the disk as it rotates. Any other method of cleaning may scratch the disk and cause loss of data.

System Reliability

The reliability of your system depends directly on the care you exercise in handling your disks and in avoiding the situations just described. Disks and drives that are not subjected to these "extremes", will perform maintenance free for a longer period of time than those handled without regard to the disk care guidelines.

A year from original date of delivery you should contact your HP Sales and Service Office for a preventative maintenance check up. Preventative maintenance should be performed once a year thereafter by an HP representative.

Maintenance Agreements

Service is an important factor when you buy Hewlett-Packard equipment. If you are to get maximum use from your equipment, it must be in good working order. An HP Maintenance Agreement is the best way to keep your equipment in optimum running condition.

Consider these important advantages –

- **Fixed Cost** – The cost is the same regardless of the number of calls, so it is a figure that you can budget.
- **Priority Service** – Your Maintenance Agreement assures that you receive priority treatment, within an agreed upon response time.
- **On-Site Service** – There is no need to package your equipment and return it to HP. Fast and efficient modular replacement at your location saves you both time and money.
- **A Complete Package** – A single charge covers labor, parts and transportation.
- **Regular Maintenance** – Periodic visits are included, per factory recommendations, to keep your equipment in optimum operating condition.
- **Individualized Agreements** – Each Maintenance Agreement is tailored to support your equipment configuration and your requirements.

After considering these advantages, we are sure you will see the cost effectiveness of a Maintenance Agreement. For more information contact your local HP Sales and Service Office.

Appendix **B**

Installation and Service

Getting Started...

1 Unpacking Your System

You should have already carefully removed your 9825A Calculator and 9885M Drive (or Drives) and 9885S Drive (or Drives) if ordered, from their shipping packages. After unpacking the drive (or drives), remove the foam shipping piece from the drive door.

The individual parts of your HP 9885 Disk Drive system were thoroughly inspected before they were shipped to you. All equipment should be in good operating order. Carefully check the drive (or drives) the ROM and other items for any physical damage sustained in transit. Notify HP and file a claim with the carrier if there is any such damage.

Please check to ensure that you have received all of the items which you ordered and that any options specified on your order have been installed in your calculator. Refer to the table on the next page and check that all accessories are present.

If you have any difficulties with your system, if it is not operating properly, or if any items are missing, please contact your nearest HP Sales and Service Office; addresses are supplied at the back of this manual.

Equipment Supplied

Check to be sure the following equipment is supplied with your 9885M Flexible Disk System.

Equipment Supplied

| Description | Part Number | 9885M 025 | 9885S |
|--------------------------------|---------------|--------------|---------|
| Disk Programming Manual | 09885-90000 | 1 | 0 |
| HP 9825A Quick Reference Guide | 09825-90011 | 2 | 0 |
| Disk Care Note | 09885-90020 | 1 | 1 |
| Disk ROM | 98217A | 1 | 0 |
| Initialized Disk | 09885-90045 | 1 | 0 |
| Blank Disk | * | 1 | 2 |
| Disk System Cartridge | 09885-90035 | 1 | 0 |
| HP 98032A Interface Cable | 98032 Opt 085 | 1 | 0 |
| Power Cord | (see page 80) | 1 | 1 |
| Spare Fuses (3 amp) | 2110-0381 | 1 | 1 |
| (2 amp for 220 V Drives) | 2110-0303 | 1 | 1 |
| Fuse Cap, European | 2110-0544 | 1 | 1 |
| Drive Number Labels (0 thru 3) | 7120-5839 | 1 Set | 1 Set |
| Select Code Labels (8 thru 15) | 7120-5840 | 1 Set | 1 Set |
| Disk Labels | 7120-6049 | 1 Set | 1 Set |
| Write Protect Tabs | 7120-5388 | 1 Sheet | 1 Sheet |
| Notebook | 9282-0580 | 1 | 0 |
| 9885S Cable | 09885-61607 | 0 | 1 |
| 9885S Operating Note | 09885-90006 | 0 | 1 |
| 9885M Installation Note | 09885-90007 | 1 | 0 |

* Blank disks may be ordered in packages of 5 using part number 09885-80004 and packages of 25 using part number 09885-80005.

Additional Equipment

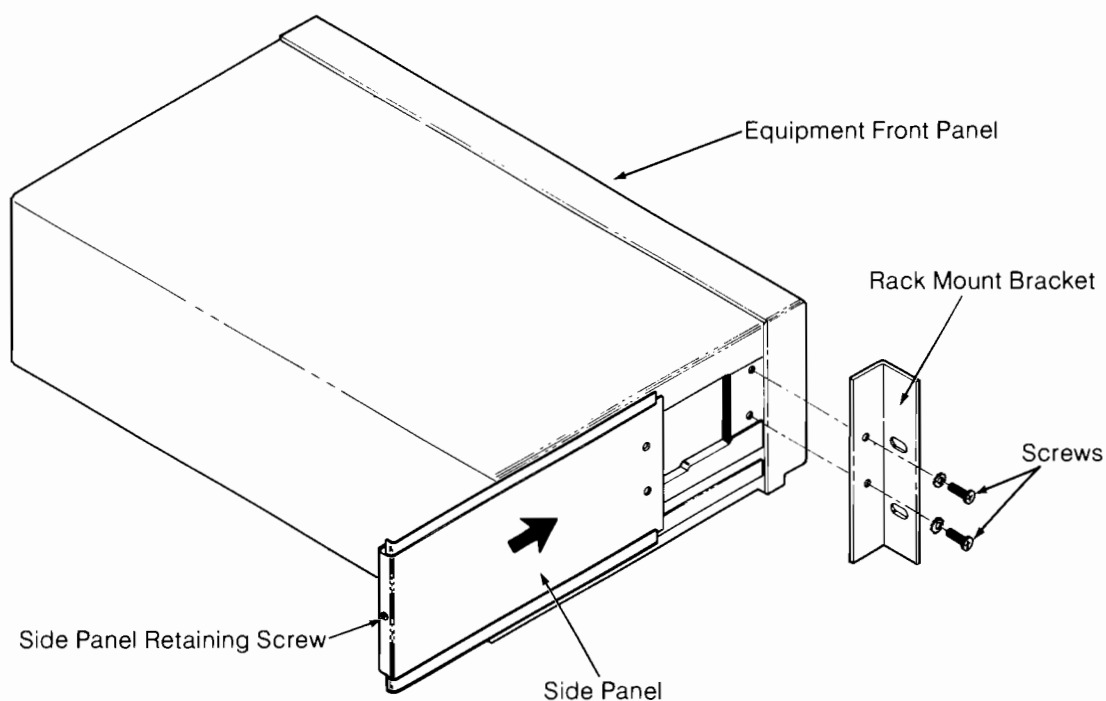
String, Matrix and Advanced Programming statements can be used with Disk statements for greater flexibility. String and Advanced Programming operations require the 98210A String/Advanced Programming ROM Card.

Option 002 Rack Mount Kit

This option allows you to mount your drive or an HP 9878A I/O Expander in a standard 19-inch rack mount cabinet. This option is installed at the factory although a rack mount field installation kit is available.

The rack mount brackets are not able to support the entire weight of the equipment. A shelf or other support should be provided by the equipment rack or cabinet to support the weight.

- Replace the standard side panels with those supplied in the rack mount kit (refer to the figure below).
- Install the rack mount brackets with the screws provided in the kit.



Rack Mount Kit Installation

2 Checking Fuses, Voltage and Power Cords

Fuses

Always be sure that the correct fuse is installed. Failure to follow this precaution may result in damage to the drive.

A different fuse is required for each of the two voltage ranges of 100-120 Vac and 220-240 Vac. Be sure that the fuse on the rear panel is the proper type and rating, as shown below.

| Voltage Setting | Fuse Rating | HP Part Number |
|-----------------|-------------|----------------|
| 100, 120 | 3 Amp (SB) | 2110-0381 |
| 220, 240 | 2 Amp (SB) | 2110-0303 |

Fuses

WARNING

ALWAYS DISCONNECT THE DRIVE FROM ANY AC POWER SOURCE BEFORE CHANGING THE FUSE.

To change a fuse –

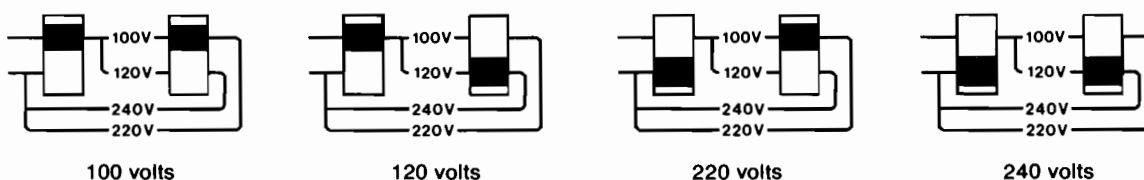
- Insert a screwdriver or a coin in the slot of the fuse cap on the rear panel (see page 81).
- Press in slightly on the cap and turn it counterclockwise.
- Pull the fuse cap from the rear panel.
- Remove the original fuse from the fuse cap and install the new fuse (either end) in the cap.
- Install the fuse cap and fuse on the rear panel. Press in slightly on the cap and turn it clockwise.

Power Requirements

The 9885M or S can operate on line voltages of either 100, 120, 220, or 240 Vac (+5%, -10%). The line frequency must be within 3.5% of 50 or 60 Hz. The voltage selector switches on the rear panel must be set to the nominal ac line voltage in your area. The illustration below shows the correct settings for each nominal line voltage.

WARNING

ALWAYS DISCONNECT THE DRIVE FROM ANY AC POWER SOURCE BEFORE SETTING THE VOLTAGE SELECTOR SWITCHES.



Switch Settings for the Nominal Powerline Voltages

To alter the setting of the selector switches -

- Insert the tip of a small screwdriver (or any small tool) into the slot on the switch.
- Slide the switch so that the position of the slot corresponds to the appropriate voltage, as shown.

Option 001 for 50Hz Operation

This option is installed at the factory. It enables the drive to operate properly on a 50Hz line frequency.

Power Cords

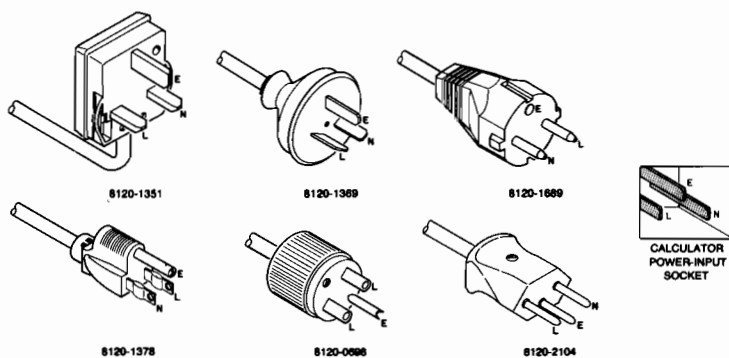
Power cords with different plugs are available for the equipment; the part number of each cord is shown below. Each plug has a ground connector. The cord packaged with the equipment depends upon where the equipment is to be delivered. If your equipment has the wrong power cord for your area, please contact your local HP Sales and Service Office.

Power cords supplied by HP have polarities matched to the power-input socket of the equipment, as shown -

- L = Line or Active Conductor (also called "live" or "hot")
- N = Neutral or Identified Conductor
- E = Earth or Safety Ground

WARNING

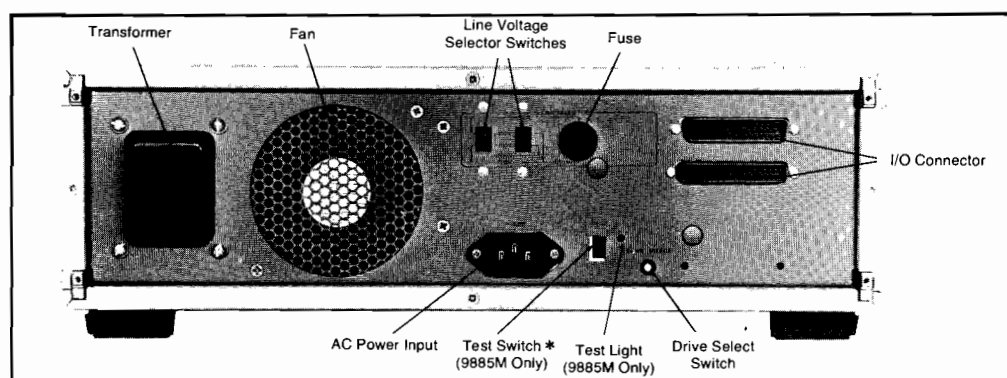
IF IT IS NECESSARY TO REPLACE THE POWER CORD, THE REPLACEMENT CORD MUST HAVE THE SAME POLARITY AS THE ORIGINAL. OTHERWISE A SAFETY HAZARD FROM ELECTRICAL SHOCK TO PERSONNEL, WHICH COULD RESULT IN INJURY OR DEATH, MIGHT EXIST. IN ADDITION, THE EQUIPMENT COULD BE SEVERELY DAMAGED IF EVEN A RELATIVELY MINOR INTERNAL FAILURE OCCURRED.



Power Cord Options

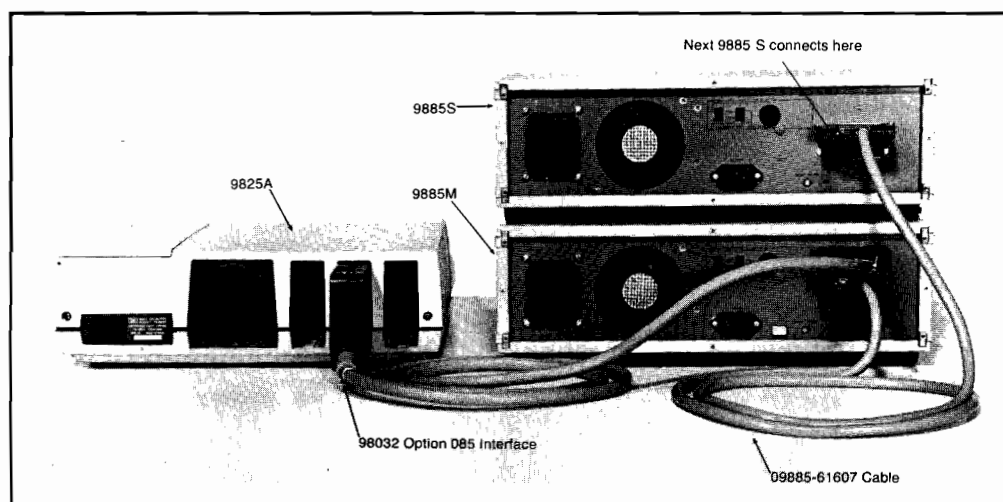
3 Connecting Calculators, Drives and Interface Cable Cards

- For a single drive system, connect the 9885M to the calculator by inserting the interface card end of the interface cable into the back of the calculator. Connect the other end of the interface cable to the top I/O connector on the back of the 9885M.



9885M Rear Panel

- For multiple drive systems, connect the 9885M to the calculator as just described. Then up to three 9885S drives can be connected in series to the 9885M drive using the 09885-61607 cable between drives. (Note: The 9885S Drive cannot be connected directly to the 9825A Calculator.)



Connecting the 9885S Drives

- Repeat this procedure for systems with more than one 9885M Drive.
- Connect one end of the ac power cord to the power input connector on the rear panel of the calculator and the other end to an appropriate ac power source.
- Connect one end of the ac power cord(s) to the input connector on the rear panel of the drives(s) and the other end(s) to an appropriate ac power source.

*9885S does not have a Self Test Switch.

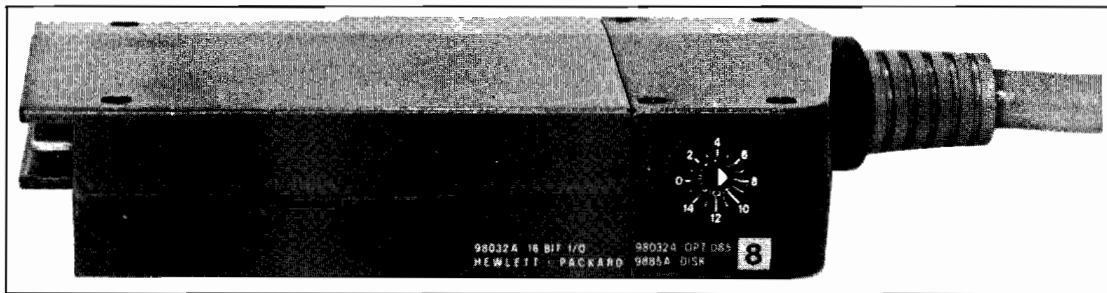
4 Setting Drive Switches and Select Codes (Usually drive 0, select code 8.)

- Once all drives are connected, set the drive select switch on the rear panel of each drive to the desired number (0 thru 3). The drive number selected is the one opposite the dot on the switch. Each of the drives connected to the calculator through an interface cable must have a different drive number. A maximum of four drives can be connected using an 98032A Interface Card Cable.



Drive Select Switch

- Set each HP 98032A Option 085 Interface Cable in your system to a different select code (8 thru 15). Up to eight* 9885M drives may be connected to one 9825A Calculator using an interface cable, each having a different select code.

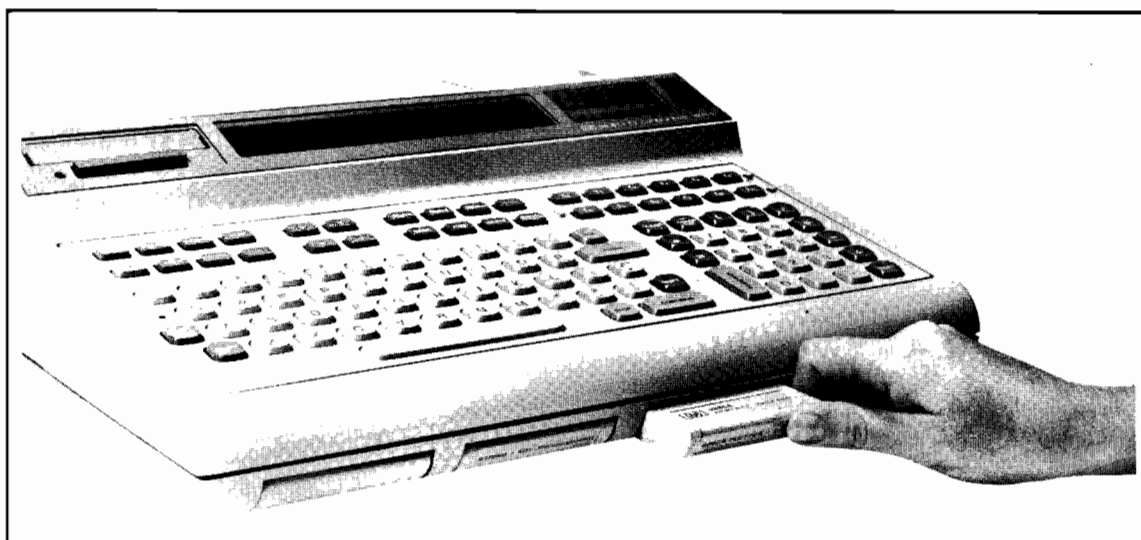


Select Code Switch

* An HP 9878A I/O Expander is required if more than three interface cables (including those for 9885M Drives) are connected to the calculator.

5 Installing the ROM Card

Be sure the calculator is off before installing the Disk ROM (Read Only Memory) Card. With the label right side up, slide the ROM through the ROM slot door. Press it in until the front of the ROM card is even with the front of the calculator, as shown.



ROM Installation

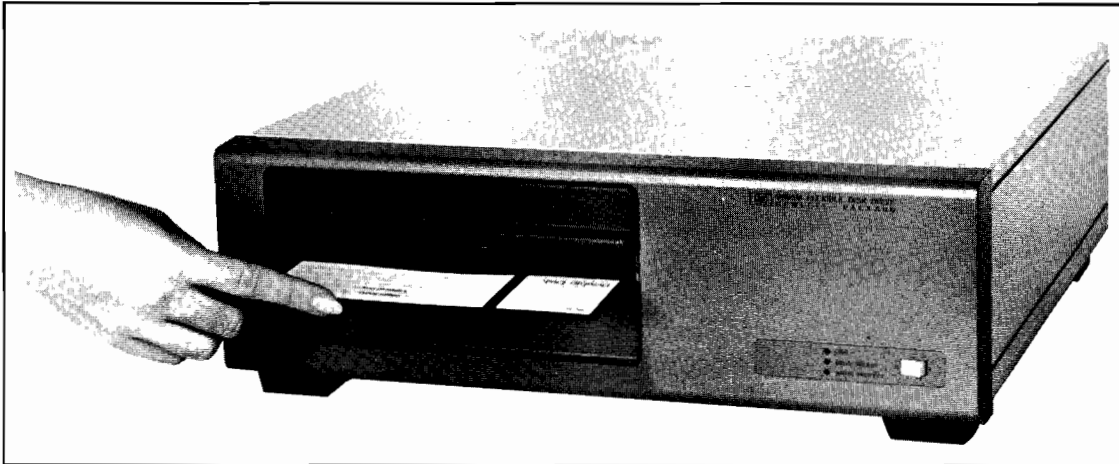
6 Installing the Disk

Follow the steps below to install a disk in your drive -

WARNING

USE ONLY FLEXIBLE DISKS APPROVED BY HP. ANY OTHER DISK MAY CAUSE PERMANENT DAMAGE TO THE READ/WRITE HEAD IN THE DRIVE. FOR A LIST OF APPROVED DISKS, CONTACT AN HP SALES AND SERVICE OFFICE.

- Once all drives are properly connected to the calculator, open the door of the drive by pushing in on the small bar on the front of the drive, below the door handle.
- Then remove the disk from its protective envelope,* and carefully slide the disk in (label side up and nearest you) until you hear a click.



Installing a Disk

- Close the door by pressing down firmly on the handle until the door locks closed. (The disk can be installed with power on and the spindle rotating without damage to the disk.) If a non-initialized disk is inserted in the drive, do not close the door until `init` is executed. (See page 91.)

NOTE

Do not execute any disk or mainframe statements with a blank (non-initialized) disk installed in the drive and the drive door closed, since errors or temporary loss of calculator control can result.

- The disk can be removed by pressing the bar below the handle on the front of the drive. The door springs open and the disk is released. When the disk is removed, it should always be replaced in its protective envelope.

7 Turn On

Once the calculator is properly connected to the drive (or drives) and the Disk ROM and the disk are installed, your system is ready to turn on -

- Turn on the 9825A Calculator using the power switch on the right.
- Turn on all 9885M and 9885S power switches located at the right on the front panel of each drive. All drives in a system must be turned on before the system can be operated.

8 Pattern Test

The Pattern Test is performed on initialized disks without useful files to test the disk surface and the hardware to be sure they are functioning properly. Since this test writes a number pattern in each record of the area specified and then reads the pattern to verify it, it should not be performed on disks containing useful data. The Pattern Test is programmable once the binary program is loaded into the calculator. If drive 0, select code 8 is being tested, the Disk ROM is not required.


NOTE

In the following instructions, do not confuse the number 1 (1) character with the letter l (l) character.

Insert the Disk System Cartridge and execute `trk0!ldb3`. The Pattern Test can be run by executing –

```
ptrn tst [number of test cycles[ ] starting track number[ ] ending track number ] ] ]
```

These parameters* can be numeric expressions. If parameters specifying the number of tests and the test area are omitted, the entire disk is tested once. If the calculator and the drive are not properly connected, the message `DISK IS DOWN` flashes in the calculator display.

To run the test continuously, use 0 for the number of test cycles; the number of the cycle is displayed as each test is completed. To stop the test, press .

If the Pattern Test is performed on a disk containing useful data, an error message is printed and the test halts. For example –

```
PTRN TST ILLEGAL
DATA ON DISK
```

The Pattern Test should not be performed on tracks 0 thru 5 of an initialized disk. If the test is run on any of the first six tracks, the information in these tracks is replaced by zeros and an error message is printed. For example –

```
BOOTS DESTROYED
TRK 0 DESTROYED
TRK 5 DESTROYED
```

CAUTION

THIS TEST ERASES ALL DATA AND INITIALIZATION ON THE SPECIFIED AREAS TO BE TESTED.

*If the disk being tested has any defective tracks (see page 9), be sure to use parameters to limit the test to the existing tracks, otherwise the message `errord6` is repeatedly printed since non-existent (defective tracks) are repeatedly searched for, for testing.

The disk must be reinitialized. (See page 90.) It is advisable to test only areas outside of the systems area to avoid reinitialization.

If, during a Pattern Test, the pattern written is not the same as the pattern read, a compare fail error occurs and an error message is printed indicating track, record and failing pattern. For example –

```
COMPARE FAIL
155555
```

These test patterns are used –

```
143306 (base 8)
066154 (base 8)
155555 (base 8)
133333 (base 8)
000000 (base 8)
```

If no errors are found, PTRN TST PASSED is printed.

9 Testing the System

Disk System Cartridge Programs

The Disk System Cartridge contains the following programs –

Track 0

```
File 0 Initialization Routine (binary)
File 1 Error Recovery Routine (binary)
File 2 Error Recovery Routine (HPL language)
File 3 Exerciser – Checkread and Pattern Tests (binary)
File 4 Exerciser – HPL Disk Test (HPL language)
Files 5-9 Unused
Files 10-69 Bootstraps
```

These programs are repeated on track 0, files 100 thru 169 and track 1, files 0 thru 69 and files 100 thru 169. This gives you a total of four copies of each of these programs.

Checkread Test

All components should be connected, installed and switched on before testing. If a `drive` statement (with drive number other than 0 and select code other than 8) has not been executed, the default values (0,8) are assumed. To test other drives, execute the `drive` statement with the drive number and select code of the drive to be tested. If drive 0, select code 8 is being tested, the Disk ROM is not required.

To load these tests into the calculator –


- Insert the Disk System Cartridge in the calculator.
- Execute `trk0` (Exercisers are duplicated on track 0, file 103 and track 1 files 3 and 103, as backup.)
- Execute `ldb3`

The Checkread Test is normally used to test disks containing useful data. This test sequentially reads each record in the area specified, the number of times indicated and then verifies the data. The Checkread Test does not destroy the data on your disk.

To run the Checkread Test, execute –

```
ckrd [number of test cycles[ , starting track number [ , ending track number ] ] ]
```

These parameters* can all be numeric expressions. If parameters specifying the number of tests and test area are omitted, the entire disk is tested once. If the calculator and the drive are not properly connected, the message `DISK IS DOWN` flashes in the calculator display.

To run the test continuously, use 0 for the number of test cycles. The number of the test cycle is displayed as each test is completed. To stop the test, press .

Error messages indicating error, track and record numbers are printed for any read error during this test. For example –

```
ERROR d7 T3 R21
```

If no errors are found, `CKRD PASSED` is printed.

The Checkread Test is programmable once the binary program is loaded into the calculator.

*If the disk being tested has any defective tracks (see page 9), be sure to use parameters to limit the test to the existing tracks, otherwise the message `error d6` is repeatedly printed since non-existent (defective) tracks are repeatedly searched for, for testing.

HPL Disk Test

The HPL Disk Test should be used on an initialized disk which has no needed information.

```
trk 0: ldf 4
run
```

Then type in the drive number and press . Type in the select code when requested and press . EXERCISER PASSED is printed when the test is passed and complete. An error message is displayed if the test fails.

Self Test

You may wish to check the electrical performance of the drive. The drive can be checked with or without a disk installed. The disk door must be closed before the self test can be performed, even if a disk is not installed.

NOTE

Performing the self test with a disk installed will erase data and initialization on the disk. Use a blank (non-initialized) disk for the self test. (If the disk is to be used later, it must be completely reinitialized.)

To perform the self test –

- Disconnect the interface cable between the 9885M and the calculator.
- Close the doors on all the drives in the system.
- Insert the blade of a screwdriver into the slot of the test switch on the rear panel of the 9885M and press the switch down; then release it.

Without the disk installed, the self test –

- Checks the microprocessor and program memory.
- Checks the drive control and drive status circuits.
- Checks the I/O functions.

With the disk installed, the self test –

- Checks the microprocessor and program memory.
- Checks the drive control and drive status circuits.
- Checks the I/O functions.
- Checks the read/write electronics.
- Checks the head positioning circuits.

Although the self test does not check all of the drive functions, it gives a high confidence level that the drive is functioning properly. The test takes less than one minute to complete. When the test is complete, the self test light (by the self test switch) will go out.

If the light stays on longer than 1 minute, the test has failed. To repeat the test, be sure all drive doors are closed properly and then press the self test switch again. If the test fails again, contact your HP Sales and Service Office for assistance.

10 Initializing Blank Disks

The initialization procedure writes addresses on the disk so that specific locations may be referenced by the system. During initialization, test patterns are also written on the disk and then read for verification. This takes about four minutes per disk. Immediately following initialization, the bootstraps needed to operate the drive are loaded onto the disk. This takes about one minute. Once initialization starts, all previous information on the disk is lost.

Each blank disk must be initialized and the bootstraps loaded before it can be used with your system. Once this procedure is complete, the disk remains initialized* and does not have to be reinitialized each time the system is turned on.

NOTE

One initialized disk is provided with your system and is ready for use. If the disk you are using is already initialized, skip this procedure and continue on page 85 with step 7.

*Unless accidentally erased, as explained on pages 86 thru 89.

Once all components of the system are connected, installed and switched on and the door to the drive (containing the disk* to be initialized) is open, follow the procedure below to initialize and load the bootstraps onto your disk -

- Insert the Disk System Cartridge in the calculator.
- Key in and execute `trk 0`. (Initialization and bootstraps procedures are duplicated on track 0, file 100 and trk 1 files 0 and 100 as backup.)
- Then key in and execute `ldb 0`. (error 54 will be displayed if all variables are not erased. If this occurs, execute `erase v` and then reload the program.)
- When the cartridge file has been loaded into the calculator memory and control returns, key in and execute `init`. (The drive door can now be closed.)
- From this point on, your calculator display will instruct you. The first message is -

DRIVE NUMBER?

- Key in the drive number of the drive you are using to initialize your disk (0, 1, 2 or 3) and press . (The calculator will continue to display this message if the drive number is not any of the drives 0 thru 3.)
- The next message displayed is -

DRIVE NUMBER = N. HIT STOP IF NOT.

where 'N' is the drive number you specified. If correct, press . If not correct, press and execute `init` again.

At this point, the track-by-track initialization and verification and bootstrap loading routine is executed. As mentioned previously, this takes approximately 5 minutes. If six or less tracks are defective, the next message printed and displayed indicates the number of defective tracks -

N DEFECTIVE TRACKS

where 'N' is the number of defective tracks. If more than six tracks are defective, execution stops and error B2 is displayed. If more than six of the tracks are defective, contact your local HP Sales and Service Office for a replacement disk.

*The disk should have a write protect tab on it. (See page 14 for more information.)

If six or less of the tracks are defective, initialization of that disk is complete and the disk is ready to use.

- If you have another disk to be initialized, place it in the drive and repeat this procedure starting with execution of `init`.
- If you want to initialize a disk in another drive (using a select code other than 8), repeat the procedure above (starting with execution of `init`) after executing a `drive` statement (see page 17) indicating appropriate drive number and select code.

Appendix **C**

Terms, Statements and Errors

Disk Terms

availability table - Table in systems area that monitors the amount and location of remaining disk space.

backup track - Track 5 of an initialized disk contains the same information as track 0: the systems table, the file directory and the availability table.

bootstraps - Binary programs loaded from the Disk System Cartridge onto the disk during initialization. These programs are part of the system software and consist of disk statements and routines.

checksum - A unique 16 bit word written on the disk at the end of each record which is generated by the controller during a write operation. Also called the CRC - Cyclic Redundancy Code.

checksum error - When a record is read, a checksum is generated and is compared to the checksum at the end of the record for data validity. If not identical after nine rereads, `error d7` (Checksum error) is displayed.

controller - A printed circuit assembly in the 9885M drive (not contained in the 9885S) that monitors and controls all drive functions.

defective track - A track on the disk where the reading and writing of data is not possible, usually because of a scratch, dirt, or lack of magnetic oxide on the surface of the disk.

The number of defective tracks is identified during initialization and is recorded in the systems table.

disk - The disk is the storage medium for the 9885M or 9885S drive. Data is written on a thin magnetic oxide film coated on mylar plastic. The disk is enclosed in a sealed plastic jacket to protect it.

drive - The 9885M and 9885S are also referred to as drives.

drive number - The drive number (0 thru 3) is selected by the drive select switch on the rear panel of the drive.

double density - The type of recording techniques used by the 9885, giving increased storage capacity and higher transfer rates over tape cartridge.

end of file mark - Mark written in the first word of each record when a file is opened and at the end of the data in a file when the "end" parameter is used. Data cannot be read past this mark, although it can be written.

end of record mark - Mark placed after the last data item when the "end" or "ens" parameter is omitted.

error recovery routines - Binary or HPL language programs allowing the user to read a file, ignoring header and checkword errors.

file - A file is one or more user records written on the disk.

file directory - A directory in the systems area containing entries for every file on the disk indicating file name, size, type and location.

flexible disk - The disk is also referred to as a flexible disk.

head - The read/write head contains the read, write and erase elements (coils) encased in ceramic. The head is in contact with the lower disk surface of the disk when data is transferred.

header - A unique bit pattern representing the address of the record, written at the beginning of each record during initialization.

hard error - Usually the result of a hardware failure. A hard error is usually non-recoverable. The software error recovery routines, however, can be used to try to recover from the error.

initialize - When a disk is initialized, addresses are written on it, it is tested by writing and reading patterns from the disk and the systems area (bootstraps and tables) is set up.

load pad - Pad opposite the head (touching the upper surface of the disk).

random file access - Method of storing and retrieving data items individually.

record - A pattern of bits representing data written on the disk following the header.

seek - Movement of the head from one track to another. (Also called stepping.)

soft error - Soft errors are recoverable and are usually caused by dirt in the air or on the disk, random electrical noise, small defects on the disk or a defective load pad.

select code - In a 9825/9885 system, each 98032A Option 085 Interface Cable must be set to a different select code (8 thru 15).

serial file access - Method of storing and retrieving data items serially instead of individually.

storage area - Tracks 6 thru 66 available for your data storage.

systems area - The systems area consists of disk tracks 0 thru 5 containing the systems table, file directory, availability table, bootstraps and backup track.

systems table - Table in the systems area indicating the calculator used to initialize the disk, the number of defective tracks and the beginning of the storage area.

track - Any one of 67 concentric circles on the surface of the disk .012 inches wide and numbered 0 thru 66.

transition - A flux reversal caused by writing on the disk which produces an electrical signal during a read that is decoded into bits (0 or 1).

tight margin - A restriction in the time allowed for a read during which a flux transition can be interpreted as a bit (a 1 or 0).

user's area - Tracks 0 thru 66 available for user data storage.

verify error - error generated after reading ten times under a tight margin or during the Pattern Test indicating that the record was read correctly but flux transitions were marginal for reading and they might not be read correctly on the next attempt.

write protect tab - Opaque tab which allows writing on the disk. When the write protect hole is open, writing on the disk is prevented.

Disk Statement Summary

All disk statements are programmable. In addition, all disk statements are executable from the keyboard and the live keyboard mode, except for `set`, `chain` and `setk` which are not allowed in live keyboard mode.

Conventions

The following conventions are used in the statement descriptions –

brackets [] – All items enclosed within brackets are optional.

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

The following definitions are used in the statement descriptions.

These parameters can be numeric expressions –

drive number (except in the `files` statement, where it must be a numeric constant)

select code

line number

number of records

position number

file number

record number

tape file number

These parameters can be text (e.g. "ABC") or string expressions (e.g. `A$` or `cap (A$)`) unless otherwise stated.

buffer name

file name (except in the `files` statement, where it must be text)

label

Data items used in print and read statements can be –

simple variables (e.g. `A`)

array elements (e.g. `A[1]`)

entire arrays (e.g. `A[*]`)

entire string variables or arrays (e.g. `A$`)

substrings (e.g. `A$[X,Y]`)

r registers (e.g. `r5`)

In addition, data items in print statements can also be –

numeric expressions (e.g. `5A`)

numeric constants (e.g. `5.12`)

string expressions (e.g. `cap (A$)`)

text (e.g. "ABC")

Drive Statement (Page 17)

`drive drive number [; select code]`

Specifies the drive (0 thru 3) to be used and, optionally, the select code (8 thru 15) indicating the 9885M drive being addressed. The drive number default value is 0; select code default value is 8.

Catalog Statement (Page 20)

`cat [select code or "buffer name"]`

Outputs information about all user files on the disk, including -
Number of remaining available records and bootstraps revision letter.

File name

File type

P - Program file

K - Key file

D - Data file

M - Memory file

B - Binary Program file

O - Other

File size (in bytes for program, binary, memory or key files and in records for data files.)

Select code 0 outputs to the calculator printer, 2 thru 15 to other output devices. Select code 16 lists previously indicated information plus file size in records and file location (track and record numbers) on the calculator printer. When a buffer name (string) follows `cat`, complete information about all user files is output to an I/O buffer.

Save Statement (Page 18)

`save "file name" [; 1st line number [; 2nd line number]]`

Stores an entire program, or the lines between and including the specified line numbers, into the file named.

Get Statement (Page 22)

```
get "file name" [ , 1st line number [ , 2nd line number ] ]
```

Loads the program specified from the disk into the calculator memory. (Variable values are not retained.)

1st line number – If specified, the loaded program lines are renumbered with the beginning line number corresponding to the specified first line number. (Program lines in memory with line numbers lower than the first line number are retained.)

2nd line number - Execution starts at the second line number. (If executed from within a program, execution of the program begins automatically.) If the second line number is omitted, program execution begins at the first line number, the default value.

Chain Statement (Page 25)

```
chain "file name" [ , 1st line number [ , 2nd line number ] ]
```

Loads the program specified from the disk into the calculator memory and retains the values of all variables. (Same line number rules as for the `get` statement apply.)

Resave Statement (Page 28)

```
resave "file name" [ , 1st line number [ , 2nd line number ] ]
```

Stores a new program, or the lines indicated by the line numbers, on the disk using a previous file name. (Same line number rules apply as for the `save` statement.)

Savekeys Statement (Page 29)

```
savek "file name"
```

Stores all present special function key definitions in the named file on the disk.

Getkeys Statement (Page 29)

```
getk "file name "
```

Loads all special function key definitions from the specified file of the disk to the calculator special function keys.

Open Statement (Page 33)

```
open "file name " ; number of records
```

Creates a data file on the disk, with the indicated number of (256 byte) records and assigns it the name specified. End of file (EOF) marks are written in the beginning of each record.

Kill Statement (Page 27)

```
kill "file name "
```

Erases the program, data memory or binary or key file named, from the disk and makes the file space available. The availability table is automatically updated (repacked) following execution of a `kill` statement.

Files Statement (Page 34)

```
files name [ # drive number ] [ , ... ]
```

Assigns file numbers (1 thru 10) to the files named and optionally the drive number for each file named. Asterisks may be substituted for file names if an `open` statement follows.

Assign Statement (Page 36)

```
assign "file name" ; position [ ; drive number [ ; return variable ] ]
```

Assigns a file number to a single file name and optionally, the drive number for the file specified. An optional return variable can be used for further file information.

| Value of Variable | Meaning |
|-------------------|--------------------------------|
| 0 | file is available and assigned |
| 1 | file doesn't exist |
| 2 | program file |
| 3 | key file |
| 4 | file type not defined |
| 5 | memory file |
| 6 | binary program file |
| 7 | file type not defined |
| 8 | file number out of range |

Serial Print Statement (Page 38)

```
serial file number ; data items [ ; ... ] [ ; "end" or "ens" ]
```

Prints specified data items in the file number indicated after the last item printed or read. An end of record (EOR) mark is printed after all items if neither the "end" or "ens" parameter is used. An end of file (EOF) mark is printed if the "end" is used. Neither an EOR or EOF mark is printed if the "ens" parameter is used.

Serial Read Statement (Page 41)

```
serial file number ; data variables [ ; ... ]
```

Reads data from the specified file starting after the last item printed or read.

Random Print Statement (Page 43)

`rpri file number ; record number ; data items [; ...] [; "end" or "ens"]`

Prints specified data items in the file number indicated starting at the beginning of the record number specified. An EOR mark is printed at the end of all data if neither the "end" or "ens" parameter is used. An EOF is printed if the "end" parameter is used. Neither EOR or EOF is printed if the "ens" parameter is used.

`rpri file number ; record number ; "end"`

Erases the specified record by placing an EOF mark in the beginning of it.

Random Read Statement (Page 46)

`rrea file number ; record number ; data variables [; ...]`

Reads data from a specified file starting at a specified record.

`rrea file number ; record number`

Repositions the file pointer to the beginning of the specified record in the file indicated.

Type Function (Page 51)

```
type [[-] file number ]
```

Identifies the type of the next item in a specified file. A negative file number is used to detect an end of record (EOR) mark.

| Type Code | Meaning |
|-----------|---|
| 0 | type undefined |
| 1 | full precision number |
| 2 | string (complete in one record) |
| 2.1 | the first part of a string (which overlaps record boundaries) |
| 2.2 | an intermediate part of a string (which overlaps record boundaries) |
| 2.3 | the end part of a string (which overlaps record boundaries) |
| 3 | end of file (EOF) mark |
| 4 | end of record (EOR) mark |

Rename Statement (Page 28)

```
renn "Old " , "New "
```

Changes the name of a file from the original name to the new name specified.

On End Statement (Page 50)

```
on end file number , line number or "label "
```

Sets up a branching condition which changes the program flow to a specified new location (by line number or label) when an end of file (EOF) mark is encountered using `sread` or an end of file (EOF) or end of record (EOR) mark using `rread`.

Save Memory Statement (Page 57)

```
save "file name "
```

Stores the calculator's entire read/write memory in the specified file.

Get Memory Statement (Page 57)

```
getn "file name "
```

Loads the calculator's entire read/write memory from the specified file and returns the calculator to its state before `save` was executed.

Copy Statements (Page 59)

Disk Copy

```
copy source drive number [ : select code]
"to" : destination drive number [ : select code]
```

Duplicates the entire contents of a specified source disk to a specified destination disk which has as many or more usable tracks as the source disk, using optional select codes, if necessary.

File Copy

```
copy "source file name " [ : drive number [ : select code] ] :
"destination file name " [ : drive number [ : select code] ]
```

Duplicates the contents of a specified source file into the specified destination file using optional drive numbers and select codes, if necessary.

Partial File Copy

```
copy source file number : beginning record number : destination file number :
beginning record number [ : number of records]
```

Duplicates a specified source file beginning at the indicated record number into the specified destination file beginning at the indicated record number, for the number of records specified.

Dump Statements (Page 62)

Disk Dump

```
dump[[-]1[0]]
```

Stores the entire disk onto up to three cartridges.*

File Dump

```
dump "file name" , tape file number [, [-]1[0]]
```

Stores the named data file from the disk into the specified tape file.*

Load Statements (Page 63)

Disk Load

```
load
```

Loads entire disk from the tape files on the tape cartridges used to dump the disk, starting with track 0, file 0 of the first tape.

File Load

```
load "file name" , tape file number
```

Loads data from a specified tape file to the disk file named.

*The optional extra parameters, if negative, suppress automatic tape marking; 1 or 10 indicates the number of disk records to be stored per tape file.

Verify Statements (Page 58)

`von or voff`

Enables (`von`) or disables (`voff`) a verify mode which does a read after a write under stricter than normal specifications and compares the data read to the data in the calculator memory.

Repack Statement (Page 58)

`repk`

Rearranges user files on the disk for more efficient use of available (contiguous) space.

Get Binary Statement (Page 64)

`getb "name "`

Loads a binary program from the disk into the binary program area of the memory.

Disk System Cartridge Statements (Page 67)

These statements are available when the binary programs on the Disk System Cartridge are loaded.

| Name | Explanation |
|--------------------------------|---|
| <code>init</code> | Writes addresses on the disk, tests the disk by writing and reading patterns on the disk and loads the systems area on the disk (including the bootstraps). |
| <code>boot</code> | Loads only the bootstraps onto the disk. |
| <code>vfyb</code> | Compares the bootstraps on the disk to the bootstraps on the Disk System Cartridge. |
| <code>killall</code> | Erases all user files from the disk without affecting the systems area or the bootstraps. |
| <code>dtrk[tape file #]</code> | Dumps a badtrack from the disk into the tape cartridge file specified. |
| <code>tinit [track #]</code> | Reinitializes one or more bad tracks. |
| <code>ltrk[tape file #]</code> | Returns corrected data from calculator memory to a reinitialized track on the disk. |
| <code>dirc</code> | Copies the Backup Track (track 5) into the systems area (track 0). |

ASCII Character Codes

| ASCII Char. | EQUIVALENT FORMS | | | ASCII Char. | EQUIVALENT FORMS | | | ASCII Char. | EQUIVALENT FORMS | | | ASCII Char. | EQUIVALENT FORMS | | |
|-----------------|------------------|-------|-----|-------------|------------------|-------|-----|-------------|------------------|-------|-----|-------------|------------------|-------|-----|
| | Binary | Octal | Dec | | Binary | Octal | Dec | | Binary | Octal | Dec | | Binary | Octal | Dec |
| NULL | 00000000 | 000 | 0 | space | 00100000 | 040 | 32 | @ | 01000000 | 100 | 64 | ^ | 01100000 | 140 | 96 |
| SOH | 00000001 | 001 | 1 | ! | 00100001 | 041 | 33 | A | 01000001 | 101 | 65 | a | 01100001 | 141 | 97 |
| STX | 00000010 | 002 | 2 | " | 00100010 | 042 | 34 | B | 01000010 | 102 | 66 | b | 01100010 | 142 | 98 |
| ETX | 00000011 | 003 | 3 | # | 00100011 | 043 | 35 | C | 01000011 | 103 | 67 | c | 01100011 | 143 | 99 |
| EOT | 00000100 | 004 | 4 | \$ | 00100100 | 044 | 36 | D | 01000100 | 104 | 68 | d | 01100100 | 144 | 100 |
| ENQ | 00000101 | 005 | 5 | % | 00100101 | 045 | 37 | E | 01000101 | 105 | 69 | e | 01100101 | 145 | 101 |
| ACK | 00000110 | 006 | 6 | & | 00100110 | 046 | 38 | F | 01000110 | 106 | 70 | f | 01100110 | 146 | 102 |
| BELL | 00000111 | 007 | 7 | ' | 00100111 | 047 | 39 | G | 01000111 | 107 | 71 | g | 01100111 | 147 | 103 |
| BS | 00001000 | 010 | 8 | (| 00101000 | 050 | 40 | H | 01001000 | 110 | 72 | h | 01101000 | 150 | 104 |
| HT | 00001001 | 011 | 9 |) | 00101001 | 051 | 41 | I | 01001001 | 111 | 73 | i | 01101001 | 151 | 105 |
| LF | 00001010 | 012 | 10 | * | 00101010 | 052 | 42 | J | 01001010 | 112 | 74 | j | 01101010 | 152 | 106 |
| VTAB | 00001011 | 013 | 11 | + | 00101011 | 053 | 43 | K | 01001011 | 113 | 75 | k | 01101011 | 153 | 107 |
| FF | 00001100 | 014 | 12 | , | 00101100 | 054 | 44 | L | 01001100 | 114 | 76 | l | 01101100 | 154 | 108 |
| CR | 00001101 | 015 | 13 | - | 00101101 | 055 | 45 | M | 01001101 | 115 | 77 | m | 01101101 | 155 | 109 |
| SO | 00001110 | 016 | 14 | . | 00101110 | 056 | 46 | N | 01001110 | 116 | 78 | n | 01101110 | 156 | 110 |
| SI | 00001111 | 017 | 15 | / | 00101111 | 057 | 47 | O | 01001111 | 117 | 79 | o | 01101111 | 157 | 111 |
| DLE | 00010000 | 020 | 16 | 0 | 00110000 | 060 | 48 | P | 01010000 | 120 | 80 | p | 01110000 | 160 | 112 |
| DC ₁ | 00010001 | 021 | 17 | 1 | 00110001 | 061 | 49 | Q | 01010001 | 121 | 81 | q | 01110001 | 161 | 113 |
| DC ₂ | 00010010 | 022 | 18 | 2 | 00110010 | 062 | 50 | R | 01010010 | 122 | 82 | r | 01110010 | 162 | 114 |
| DC ₃ | 00010011 | 023 | 19 | 3 | 00110011 | 063 | 51 | S | 01010011 | 123 | 83 | s | 01110011 | 163 | 115 |
| DC ₄ | 00010100 | 024 | 20 | 4 | 00110100 | 064 | 52 | T | 01010100 | 124 | 84 | t | 01110100 | 164 | 116 |
| NAK | 00010101 | 025 | 21 | 5 | 00110101 | 065 | 53 | U | 01010101 | 125 | 85 | u | 01110101 | 165 | 117 |
| SYNC | 00010110 | 026 | 22 | 6 | 00110110 | 066 | 54 | V | 01010110 | 126 | 86 | v | 01110110 | 166 | 118 |
| ETB | 00010111 | 027 | 23 | 7 | 00110111 | 067 | 55 | W | 01010111 | 127 | 87 | w | 01110111 | 167 | 119 |
| CAN | 00011000 | 030 | 24 | 8 | 00111000 | 070 | 56 | X | 01011000 | 130 | 88 | x | 01111000 | 170 | 120 |
| EM | 00011001 | 031 | 25 | 9 | 00111001 | 071 | 57 | Y | 01011001 | 131 | 89 | y | 01111001 | 171 | 121 |
| SUB | 00011010 | 032 | 26 | : | 00111010 | 072 | 58 | Z | 01011010 | 132 | 90 | z | 01111010 | 172 | 122 |
| ESC | 00011011 | 033 | 27 | ; | 00111011 | 073 | 59 | [| 01011011 | 133 | 91 | { | 01111011 | 173 | 123 |
| FS | 00011100 | 034 | 28 | < | 00111100 | 074 | 60 | \ | 01011100 | 134 | 92 | | 01111100 | 174 | 124 |
| GS | 00011101 | 035 | 29 | = | 00111101 | 075 | 61 |] | 01011101 | 135 | 93 | } | 01111101 | 175 | 125 |
| RS | 00011110 | 036 | 30 | > | 00111110 | 076 | 62 | ^ | 01011110 | 136 | 94 | ~ | 01111110 | 176 | 126 |
| US | 00011111 | 037 | 31 | ? | 00111111 | 077 | 63 | _ | 01011111 | 137 | 95 | DEL | 01111111 | 177 | 127 |

Subject Index

a

additional equipment 76
 additional statements 57
 approved disks 3, 72, 84
 ASCII Code Table 106
 assign (assign) statement 36
 availability table 5, 8, 10, 27

b

backup track 9, 11
 binary program statements . 64, 65, 67, 105
 boot statement 67, 105
 bootstrap routine (boot) 67, 105
 bootstraps area 9, 10, 67, 87

c

calculator (HP 9825A) 1
 cat 0 20, 97
 cat 16 20, 97
 cat "buffer" 21, 97
 catalog (cat) statements 20, 97
 chain statement 25, 98
 Checkread Test 88
 connecting calculators,
 drives, cables 4, 81
 contiguous file areas 27
 conventions, manual 15, 32
 COPY, disk 59, 103
 COPY, file 60, 103
 COPY, partial file 60, 103
 COPY statements 59, 103

d

data files 12, 13, 17, 31, 53, 96
 data file numbers 34
 data file operations 33
 data file pointers 35, 47
 data file statements 33
 data storage requirements 53
 default values (0,8) 4, 17, 60, 82, 88
 dirc statement 65, 105
 directory 59
 disk 1, 2, 3, 7, 72, 84
 disk capacity 69
 disk care 72
 disk COPY statement 59, 103
 disk drives (9885M/9885S) 2
 disk dump statement 62, 104
 disk installation 5, 84
 Disk is Down 15, 31
 DISK IS DOWN 86, 88
 disk load statement 63, 104
 disk manufacturers 3, 72, 84
 disk ROM (HP 98217A) 4, 83
 disk specifications 69
 disk speed 69
 disk structure 7
 Disk System Cartridge 87
 disk system tests 87, 88, 89
 disk terms (glossary) 93
 disk test 89
 drive number switch 4
 drive statement 17, 97
 dump statements 62, 104
 dump track(dtrk) statement 65, 105

e

electrical (self) test 5, 89
 "end" 38, 43, 54
 "ens" 38, 43, 54
 EOF marks 38, 43, 54
 EOF/EOR Summary 54
 EOR marks 38, 43, 54
 equipment supplied 78
 error info lost 64
 error messages 68
 error recovery routines (d5, d6, d7) 64
 exercisers 86, 88

f

file `COPY` statement 60, 103
 file directory 5, 9
 file `DUMP` statement 62, 104
 file `load` statement 63, 104
 file names (limitations) 17
 file numbers 34, 35, 36
 file pointers 35
 file structure 11
`files` statement 34
 flexible disk 1, 2
 fuses 4, 78

g

`get binary` (`getb`) statement 64, 105
`get keys` (`getk`) statement 29, 99
`get memory` (`getm`) statement ... 57, 103
`get` statement 22, 98
 getting started with your system 4, 75
 glossary of disk terms 93

h

HP approved disks 3, 72, 84
 HPL Disk Test 5, 89

i

initialization (`init`) routine ... 67, 90, 105
 initializing a disk 67, 90, 105
 interface cable
 (HP 98032A Opt. 85) 4, 81

k

`kill` statement 27, 99
`killall` statement 67, 105

l

line voltage 4, 78
`load` statements 63, 104
`load track` (`ltrk`) statement 65, 105
`loading bootstrap` (`boot`)
 routine 67, 105
`loading spare directory`
 (`dirc`) statement 65, 105

m

manual conventions 15, 32
 manual requirements 3


O

`on end` statement 50, 102
`open` statement 33, 99
 Opt. 001 for 50 Hz Operation 79
 Opt. 002 Rack Mount Kit 77
 Opt. 025 Interface Kit 3
 other operations 57

p

partial file `COPY` 60, 103
 Pattern Test 86
 positioning the file pointer 47
 power cords 4, 80
 power requirements 4, 79
 preventative maintenance 72, 73
 program file operations 15
 program files 11, 17
 program storage requirements 30
 protecting the disk from writes 14

r

rack mount kit installation 77
 random file access 13
 random print (`rprt`) statement 43
 random read (`rread`) statement 101
 records 8
 recoverable errors 64
 rename (`renn`) statement 28
 repack (`repk`) statement 102
 replacement disks 9, 91
 repositioning the file pointer 47
 requirements, manual 3
 requirements, ROM 76
`reset` statement 28, 98
 5
 ROM installation 4, 83

S

Sales and Service Offices 107
`save` statement 18, 97
 save keys (`savek`) statement 29, 98
 save memory (`savem`) statement 57, 103
 select code settings 4, 82
 Self Test (electrical) 5, 89
 serial file access 12, 13
 serial print (`sprt`) statement 38, 100
 serial read (`sread`) statement ... 41, 100
 service 72, 107
 setting drive number switches 4, 82
 setting select codes 4, 82
 SPARE DIR. 9, 11
 spare directory 9, 11
 spare directory (`dirc`) 65, 105
 statement summary 96
 storage area 11
 string overhead 53, 69
 suggested disk manufacturers ... 3, 72, 84
 summary of EOR/EOF marks 54
 system installation 4, 75
 system reliability 73
 system set up 4, 75
 systems area 8
 systems table 9

t

`tinit` statement 65, 105
 transfer times 70
 TRK NOT FOUND 86, 88
 turn on instructions 4, 75
 type function 51, 102

u

unpacking your system 4, 75

V

verify boots (`vfyb`) statement 67, 105
 verify (`voff/von`) statements ... 58, 105
`voff` statement 58
 voltage requirements 78, 79
`von` statement 59

W

warranty statement inside front cover
 write protect tabs 14, 91
 write protecting the disk 14

9885 Error Messages

Hardware Errors

- error d0 Firmware/driver out of synchronization. More than six defective tracks in a row.
(Press)
- error d1 All drives in system not powered.
- error d2 Door opened while disk is being accessed.
- error d3 Disk not in drive or no such drive number.
- error d4 Write not allowed to protected disk.
- error d5 Record header error. (Use Error Recovery Routine)
- error d6 Track not found. (Use Error Recovery Routine)
- error d7 Data checkword error. (Use Error Recovery Routine)
- error d8 Hardware failure. (Press)
- error d9 Verify error due to drive problem. Marginal data. (Reprint data)

Software Errors

- error D0 Improper argument.
- error D1 Argument out of range.
- error D2 Improper file size (negative, 0 or >32767).
- error D3 Invalid file name.
- error D4 File not found.
- error D5 Duplicate file name.
- error D6 Wrong file type.
- error D7 Directory overflow.
- error D8 Insufficient storage space on disk.
- error D9 Verify error due to cable, calculator or drive problem. Bad data (Reprint data.)
- error F0 File overflow when read or print executed.
- error F1 Bootstraps not found. (Reload bootstraps)
- error F2 String read but wrong data type encountered.
- error F3 Attempt to read data item but type doesn't match.
- error F4 Availability table overflow. (Repack)
- error F5 Attempt on end branch from other than running program.
- error F6 Unassigned data file pointer.
- error F7 Disk is down so line cannot be reconstructed.
- error F8 Disk is down and pressed.
- error F9 System error. (Save files individually and reinitialize)

These mainframe errors take on additional meaning when the Disk ROM is installed.

error 03 Mnemonic not found because disk may be down.

error 29 Line can't be executed because ROM (usually String) is missing.

error 31 Line not found.

error 50 Get or chain should be last statement in a line.

error 51 ROM now installed which wasn't when `saven` was executed.

error 52 ROM now missing which wasn't when `saven` was executed.

error 63 Disk load operation would overlay `esb` return address so load not executed.

error 64 `get`, `chain` or `getk` not allowed from live keyboard mode or during an
ent statement.

These errors may result during the binary Initialization and Error Recovery Routines.

error B0 Wrong syntax, argument out of range or variable not properly dimensioned.

error B1 More than six defective tracks on the disk.

error B2 Verify error. Boots on the disk not identical to boots on the cartridge.

error B3 `dtrk`, `tinit` or `ltrk` not allowed because error information lost or error
not d5, d6, d7 or d9.

error B4 Attempt to access record for error correction which isn't part of data file.

error B5 Improper string length (inconsistent with length given in header).

error B6 Not enough space in calculator buffer for data item or item can't be placed in
this part of buffer.

error B7 Missing disk or String ROM.

error B8 Track still bad after `tinit`.

Warning Messages

SPARE DIR. is printed when the spare directory in the backup track automatically
replaces the main directory.

DISK IS DOWN is displayed when running a program that uses a drive number of a
drive that is not connected to the system, not powered or whose door
is open.