commodore

VIC-1540 USER'S MANUAL

SINGLE DRIVE FLOPPY DISK

by commodore

VIC-1540

SINGLE DRIVE FLOPPY DISK

USER'S MANUAL

P/N 15400018-02

commodore

Commodore Business Machines
3330 Scott Boulevard
Santa Clara, California 95050

TABLE OF CONTENTS

List of Illustrations

List of Tables

Chapter 1
INTRODUCTION

GENERAL INFORMATION

With the purchase of your Commodore VIC-1540 Single Drive Floppy Disk you have greatly enhanced the computing power of your Commodore VIC system. To get the most out of your system you should study your computer's user guide, and if necessary the BASIC manuals listed in Table 1. You will benefit most if you first read through this entire manual, taking note of those features that relate to your particular floppy as well as those which are common to all Commodore Floppys.

The information presented in this manual is extensive and may, in some cases, present information that is currently beyond your particular level of expertise. However, by carefully and thoughtfully studying its contents you will gain the confidence necessary to progressively upgrade your programming skills and expertise.

This manual presents discussions, descriptions, practices and procedures relating to the use and operation of the VIC-1540 Single Floppy Disk Drive.

DESCRIPTION

The VIC-1540 described in this manual is an intelligent single drive diskette storage device. Its individual primary components consists of read/write controls, drive motor electronics, drive mechanism, read/write head, and track positioning mechanism. The disk drive discussed in this manual uses Serial interface same as The VIC-1515 graphic printer. Because the device is an "intelligent" peripheral, its operation requires no space in the computer's memory. This means you have just as much computer memory available to you as when you do not have the disk attached.

Table 1. Suggested Reading List

Pet/CBM Personal Computer Guide.
C.S. Donahue and J.K. Enger, *Osborne/McGraw-Hill*, 630 Brancroft Way, Berkeley, CA 94710

Hands-On Basic with a Pet.
H.D. Peckham, *McGraw-Hill*, 1979

Entering BASIC.
J. Sack and J. Meadows, *Science Research Associates*, 1973

BASIC: A Computer Programming Language.
C. Pegels, *Holden-Day, Inc.*, 1973

BASIC Programming.
J. Kemeny and T. Kurtz, *Peoples Computer Co.*, 1010 Doyle (P.O. Box 3100), Menlo Park, CA 94025, 1967

BASIC FOR HOME COMPUTERS.
Albrecht, Finkle and Brown, *People Computer Co.*, 1010 Doyle (P.O. Box 3100), Menlo Park, CA 94025, 1973

A Guided Tour of Computer Programming in BASIC.
T. Dwyer, *Houghton Mifflin Co.*, 1973

Programing Time Shared Computer in BASIC.
Eugene H. Barnet, *Wiley-Interscience*, L/C 72-175789

Programing Time Shared Computer in BASIC.
Eugene H. Barnett, *Wiley-Interscience*, L/C 72-175789

Programming Language #2.
Digital Equipment Corp., Maynard, MA 01754

101 BASIC Computer Games.
Software Distribution Center, *Digital Equipment Corp.*, Maynard, MA 01754

What do To After You Hit Return.
Peoples Computer Co., 1010 Doyle (P.O.Box 3100), Menlo Park, CA 94025

Basic BASIC.
James S. Coan, *Hayden Book Co.*, Rochelle Park, NJ

WORKBOOKS 1-5.
T.I.S., P.O.Box 921, Los Alamos, NM 87544

Programming the 6502.
R. Zaks, *Sybex*, 1978

24 Tested, Ready-to-Run Game Programs in Basic.
K. Tracton, *Tab Books,* 1978

Some Basic Programs.
M. Borchers and R. Poole, *Osborne & Assoc. Inc.,* 1978

Basic Programming for Business.
I. H. Forkner, *Prentice-Hall,* 1977

The Channel Data Book.
B. Lewis, 5960 Mandarin Ave., Goleta, CA 93017, 1978

PET and the IEEE 488 Bus (GPIP).
Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley, CA 94710

Personal Computing on the VIC 20
Commodore International, Ltd. Norristown,
PA 19403, 1981

VIC-20 Programmer's Reference Guide
Commodore Business Machines Inc.
3330 Scott Blvc. Santa Clara, CA95050, 1981

Front Panel

The front panel of the disk drive consists of an identification panel across the top; slot in which to insert a diskette; and a door to close after inserting the diskette. When the door is closed, the diskette is clamped onto the diskette spindle hub. Also on the front panel are two LED indicator lights. The red LED on the slot side lights when drive is active and flashes whenever a disk error occurs.

The green LED on the lower left side is a power indicator which lights when power is ON.

Back Panel

The Back of the disk drive contains two serial interface connectors. Near the panel's lower edge is the power connector. There is also a "slow blow" fuse.

Interior Configuration

The interior of your floppy contains a disk drive. All the logic for the disk drive is contained within the unit. The mechanical devices are, for the most part, located beneath the disk spindle.

The Diskette

The diskette (also known as a minifloppy, floppy diskette, minidiskette, etc.) is similar to the standard flexible disk. There are several reputable manufacturers of the 5¼-inch diskettes. You should make sure that you buy diskettes for SOFT SECTORED FORMAT. Your Commodore dealer can supply your needs.

Specifications

Table 2 presents the specifications for the VIC-1540.

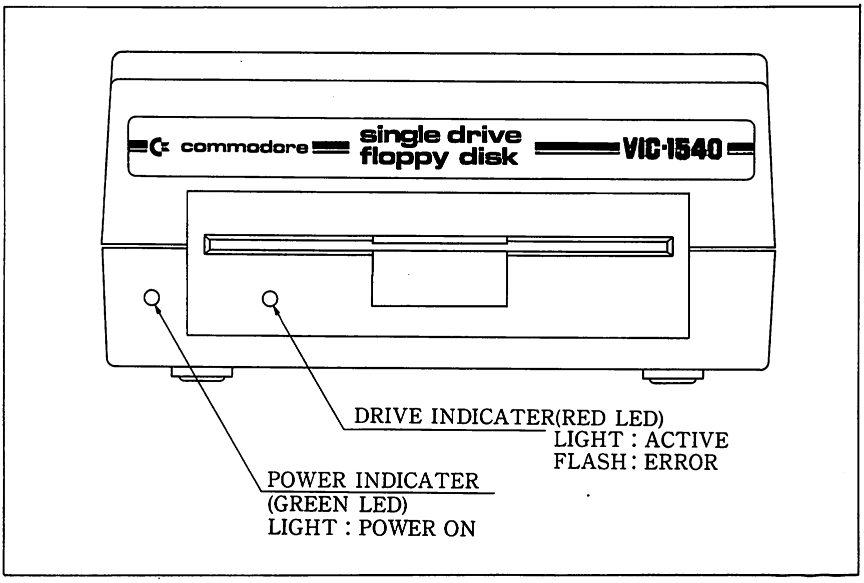


Fig 1. Front Panel

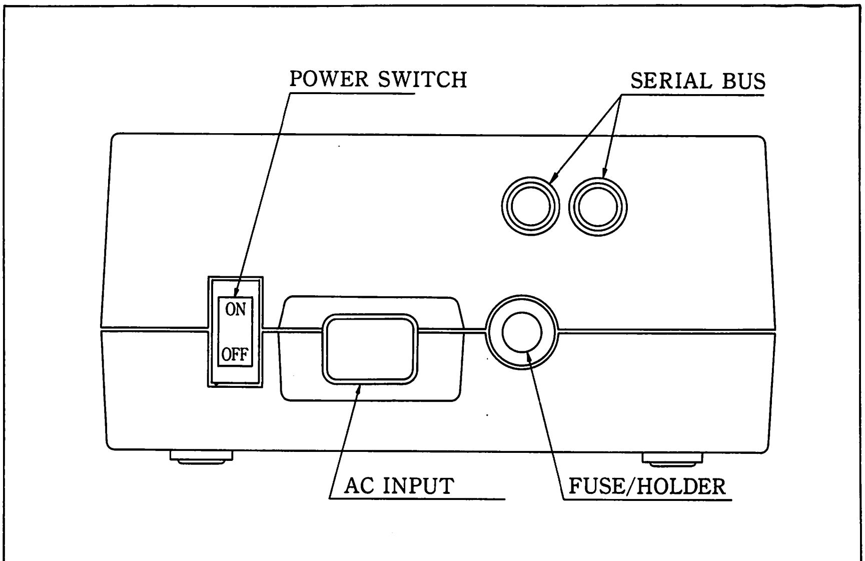


Fig 2. Back Panel

Table 2. Specifications VIC-1540 Single Drive Floppy Disk

STORAGE	
Total capacity	174848 bytes per diskette
Sequential	168656 bytes per diskette
Relative *	167132 bytes per diskette
	65535 records per file
Directory entries	144 per diskette
Sectors per track	17 to 21
Bytes per sector	256
Tracks	35
Blocks	683 (664 blocks free)
IC's:	
6502	microprocessor
6522 (2)	I/O, interval timers
Buffer	
2114 (4)	2K RAM
PHYSICAL:	
Dimensions	
Height	97 mm
Width	200 mm
Depth	374 mm
Electrical:	
Power requirements	
Voltage	100, 120, 220, or 240 VAC
Frequency	50 or 60 Herts
Power	25 Watts
MEDIA:	
Diskettes	Standard mini 5¼", single sided, single density

* Although VIC-1540 is designed with a capability to handle relative files, it can not be used with the current version of VIC-1001 personal computer.

CARE OF THE VIC-1540

The disk drive should be placed on a flat surface free of vibration. It is important that dust particles be kept at a minimum since a particle buildup will interfere with optimum operation. If you should experience a hardware failure contact your Commodore dealer. Any attempt to correct the problem yourself could result in voiding the warranty.

CARE OF THE DISKETTES

Handle diskettes with care. Follow these instructions to maintain the quality of the diskette and to protect the integrity of the data:

1. Return the diskette to its storage envelop whenever it is removed from the drive.
2. Keep the diskettes away from magnetic fields. Exposure to a magnetic field can distort the data.
3. Never leave a diskette on top of your computer or disk drive.
4. Do not write on the plastic jacket with a lead pencil or ball-point pen. Use a felt tip pen or fill out the label before attaching it to the jacket.
5. Do not expose diskettes to heat or sunlight.
6. Do not touch or attempt to clean the diskette surface. Abrasions will cause loss of stored data.
7. Before applying power to the VIC-1540 open the drive door and remove the diskette.

UNPACKING THE DISK DRIVE

Before unpacking the disk drive, inspect the shipping carton for signs of external damage. If the carton is damaged, be especially careful when inspecting its contents. Carefully remove all packing material and the contents of the carton. DO NOT discard any packing material should contain:

1. VIC-1540 Single Floppy Disk Drive
2. User Mannual, Number 1540018-02
3. TEST/DEMO diskette:
4. WARRANTY CARD

If any items are missing, please contact your Commodore dealer immediately.

NOTES

Chapter 2

PREPARING TO USE YOUR DISK DRIVE

Before starting to use your disk drive, make sure it is in good working condition. This includes properly connecting it to your computer, giving it a powr-on and initial checkout test, and finally the performance test using the appropriate TEST/DEMO diskette.

CONNECTING THE DISK DRIVE TO THE COMPUTER

The connector cable is required to interface the floppy to the computer. This cable is be supplied with your disk drive.

NOTE: The disk drive should be the first peripheral attached to the computer if other devices are to be "daisy-chained".

Follow these steps to connect the disk drive to your computer:

STEP 1: Turn power OFF to the computer and the expansion module.

STEP 2: Place the disk drive in a convenient location as close as possible to the computer. DO NOT connect the disk drive to a power outlet at this time.

STEP 3: Connect the serial cable between the serial interface connector on the computer and the connector on the disk drive.

STEP 4: Connect the disk drive power cable to an AC outlet. DO NOT turn on power at this time.

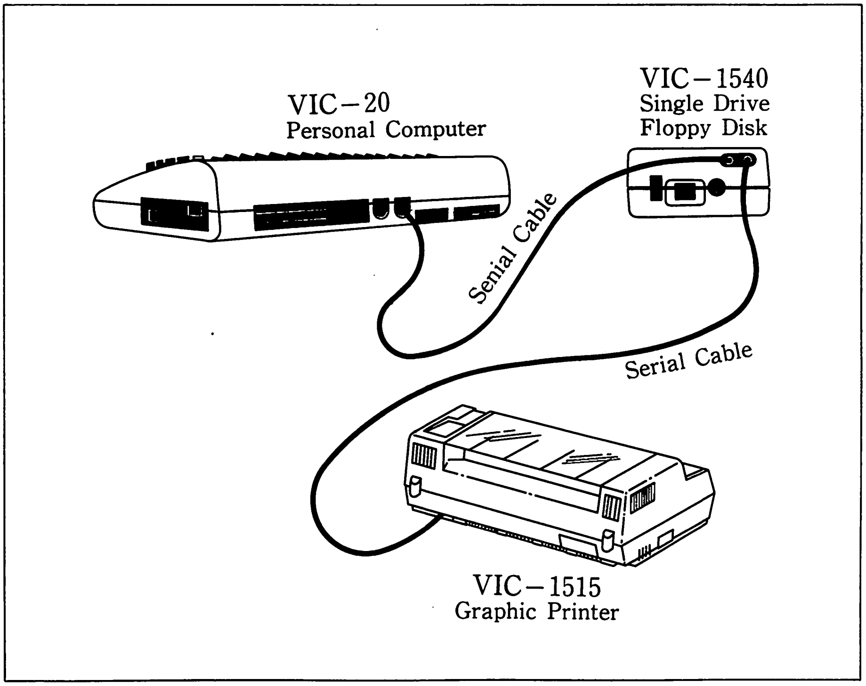


Fig 3. Floppy Disk Hookup

PERFORMING THE POWER-ON TEST

You are now ready to proceed with the power-on part of the checkout:

STEP 1: Open the disk drive door. Ensure that no diskette is present in the drive.

STEP 2: Turn Power ON to the expansion module (have the optional cartridge inserted before turning power on.)

STEP 3: Turn power ON to the COMPUTER and verify that is working properly.

STEP 4: Apply power to the disk drive. All two indicator lights (LED) on the front panel will light. After about 1 second the red drive indicator will go out and the green power indicator will remain on. If the red drive indicator keeps on flashing, turn the power OFF. Wait one minute and try again. If the above condition continues, contact your Commodore dealer.

Note: If the problem persists, try disconnecting the other devices attached to the sesial interface. This should assure that a problem related to another device does not affect the disk drive.

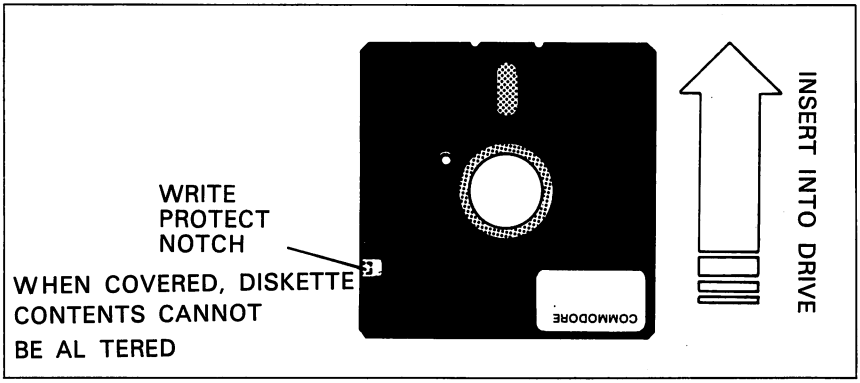


Fig 4. Position for Diskette Insertion

INSERTING THE DISKETTE

CAUTION: NEVER APPLY POWER TO THE DISK DRIVE IF DISKETTE IS PRESENT (LOCKED AND SEATED) IN THE DRIVE.

STEP 1: Ensure that the power to the disk drive is OFF and DO NOT apply power until you complete this step. Open the disk drive door and make sure that no diskette is present in the drive.

STEP 2: If the preceeding conditions have been met, you may apply power to the disk drive.

STEP 3: Insert the diskette into the slot and with the write protect tab oriented to the left.

STEP 4: Once the diskette is in the slot, gently push on it until it is fully seated.

STEP 5: Press DOWN firmly on the spring-loaded door of the drive until you hear a distinct "click". The diskette is now locked and seated in the drive, ready for processing by the computer.

STEP 6: To remove the diskette, insert your index finger under the lip of the spring-loaded door and gently PULL. This will release the door and permit access to the diskette. The diskette is now free to be removed from the drive

DISK DRIVE PERFORMANCE TEST

When you have successfully completed the Power-On test, proceed with the Performance Test. Don't worry if you don't fully understand exactly what is happening in this test. At this point, enter the commands just to get a feel for what you can do with your disk. If UNEXPECTED results are obtained during any step of the test, stop and start over again. The most likely cause of a problem is an improperly entered command. This is to be expected until you become familiar with your disk unit.

All commands are entered via the keyboard and must be followed by a carriage return: press the RETURN key on your keyboard.

NOTE: Commands must be entered exactly as shown. DO NOT insert any spaces unless shown in the example. If the error indicator lights, you may be able to continue the example anyway. Re-enter your last command. If the light goes out, your correction was successful and you may continue.

STEP 1: Insert the DEMO diskette into the drive as previously instructed.

STEP 2: Type: OPEN 1, 8, 15, "10" and press return. This pocedure initializes the diskette and makes it ready for use.

STEP 3: Type LOAD "PERFORMANCE TEST", 8

The screen will display:

```
SEARCHING FOR PERFORMA
NCE TEST
LOADING
READY.
■
```

STEP 4: Type: RUN and press RETURN, the following will display:

```
   PERFORMANCE TEST

  INSERT SCRATCH
    DISKETTE IN DRIVE
    PRESS RETURN
           WHEN READY
    D:SK NEW COMMAND
  WAIT ABOUT 80 SECONDS
```

Do not use diskettes containing any valuable information since the Performance Test Program will re-format it and any data will be lost. The test program will label this diskette "Test Disk". This diskette is ready for further use when the test program is completed and the performance test has been satisfied.

The computer will first format the diskette in the drive. At the end of the operation the screen displays:

```
                   0 OK 0  0
 DRIVE PASS
         MECANICAL TEST
```

The computer conducts the remainder of the Performance Test and displays:

```
OPEN WRITE FILE
                 0 OK 0  0
WRITING DATA
                 0 OK 0  0
CLOSE WRITE DATA
                 0 OK 0  0
OPEN READ FILE
                 0 OK 0  0
READING DATA
                 0 OK 0  0
SCRATCH FILE
                 1  FILE S
CRATCHED 1 0
```

```
WRITE TRACK 35
               0 OK 0  0
WRITE T'?ACK 1
               0 OK 0  0
READ TRACK 35
               0 OK 0  0
READ TRACK 1
               0 OK 0  0
 UNIT HAS PASSED
     PERFORMANCE TEST!

 PULL DISKETTE FROM
  DRIVE BEFORE TURNING
   POWER OFF.
```

STEP 5: Remove the diskettes and return them to their protective jackets. The floppy has passed the Performance Test.

STEP 6: If any problems have been encountered during this phase of the test, return to Step 1 and repeat the entire procedure. If problems persist and you do not reach a satisfactory conclusion to the Performance Test, contact your Commodore dealer.

Chapter 3

LEARNING HOW TO USE YOUR FLOPPY DISK DRIVE

Your Floppy Disk Drive adds and enhances your computing power with added storage and file handling capability and is controlled directly with:

- BASIC commands entered via the keyboard,
- BASIC statements within programs, and
- special disk commands.

In this chapter you will learn how to apply those commands and statements. This chapter is organized in such a way that the functions and format of disk commands are described in a manner which permits the user to perform disk-related tasks.

Before using your floppy disk make sure you know how to:

1. operate your Commandore VIC Computer,
2. do elementary programming in BASIC, and
3. open and close files.

This chapter will first acquaint the user with those fundamental disk commands that perform disk maintenance and file manipulation and will then progressively advance toward an understanding of those BASIC commands used for data handling. Approached in this manner, the user will then have developed the necessary confidence and programming skills to proceed to advanced disk programming techniques. Practice the disk commands, read the examples, and follow the step-by-step illustrations of their usage. The understanding of the more advanced disk programming techniques will depend to a large degree upon how well the fundamentals have been mastered.

To facilitate your understanding and mastery of Commodore BASIC, two computer terms are stressed in this Chapter: Block Availability Map (BAM) and Disk Operating System (DOS). Although these are conventional terms, they will be riefly discussed as they relate to Commodore Floppy Disk Usage.

THE BLOCK AVAILABILITY MAP (BAM)

The BAM is a disk memory representation of available and allocated space on a disk. When the system stores information on a disk, the BAM will be automatically referenced by the DOS to determine what space is available and how many blocks can be allocated. Iff sufficient space is available to store a given file, it will be stored on the disk and the BAM updated to account for the space allocated. However, if the DOS detects that a file will occupy more space than available, an error message will be generated.

Formatting a disk creates the BAM which is then loaded into DOS memory upon initialization. The BAM is stored on diskette in varying locations depending upon the drive used:

BAM Location and Memory Required	
Track 18, Sector 0	128 bytes

As changes occur to the BAM in DOS memory, the BAM on disk will be updated to reflect these changes. Updates to the BAM occur when a program is saved or a CLOSE is performed on a new SEQuential data file.

THE DISK OPERATING SYSTEM (DOS)

The DOS is responsible for managing information exchange between the disk controller and the computer.

The DOS performs many functions which are transparent to the user but which are vital to the operation of the system. For example, the DOS monitors the input/output (I/O) of the disk so that channels are properly assigned and that no lengthy waits for an open channel occur. In addition to monitoring of disk I/O, the DOS also uses the channel structure to search the directory and to delete and copy files.

DISK MAINTENANCE COMMANDS

The following disk commands permit the user to perform file mainpulation and disk maintenance.

	BASIC COMMAND	FUNCTION
Diskette Level	NEW	Formats a disk
	INITIALIZE	Prepare diskette for use
	LOAD "$"	Read disk directory
	VALIDATE	Reconstruct Block Availability Map (BAM)
File Level	COPY	Copies files (optional concatenation)
	RENAME	Renames a file
	SCRATCH	Erases a file

NOTE: Diskette commands can be transmitted to the disk by PRINT# commands. The examples in this chapter assume that a file has been opened with the OPEN 15, 8, 15 command. If the error message ? FILE OPEN ERROR appears upon typing the OPEN command, it means that the logical file was opened but had not been properly closed. When this error appears such logical file must be properly closed by typing CLOSE 15.

NEW

Each time a diskette is placed in one of the drives, both the diskette and the drive must be prepared for use. A previously unused diskette must first be formatted in the soft-sector format recognized by your particular disk drive. This may be accomplished by use of the NEW disk command.

To use the NEW command, to format the diskette and initialize the disk drive, enter the command:

```
        PRINT#15, "commandstring"
```

where 15 is the logical file number of a file which has been opened to the disk command channel (primary address 8, secondary address 15).

The format of NEW is:

"NEWdr:fn, xx"

or

"Ndr:fn,xx"

Where d=the drive number 0 (0 may be omitted)

fn=the file name you wish to assign to the disk. It may be up to 16 characters long.

xx=a unique two-character, alphanumeric identifier supplied by the user.

The NEW command (with ID specified) is used on an unformatted diskette or one which the user wishes to reformat. NEW creates the block headers, writing the sync characters, disk ID, and track and sector numbers at the beginning of each block. The directory header and the BAM are created and the diskette is made ready to accept data. The command may be used on an already formatted diskette (with no ID specified) to clear the disk directory and reinitialize the BAM, deallocating all blocks on the diskette. The time involved in reformatting without an ID is much less than formatting with an ID.

Example 1:
```
OPEN 15, 8, 15
PRINT#15, "NO:TESTDISK, 88"
```

These commands will open the command and error channel to the disk drive and format a disk in the drive, giving it a disk identifer of 88.

The following simplified form may be used for this purpose:

Example 2:
```
OPEN 15, 8, 15, "NO:TESTDISK, 88"
```

Here's an example of reformatting a diskette using the NEW command and no disk ID.

Example 3:
```
OPEN 1, 8, 15, "NO:NEWNAME"
```

The diskette will be assigned the name "NEWNAME" and the directory and BAM will be cleared. This procedure will work only if the diskette has been formatted.

The NEW disk command SHOULD NOT be confused with the NEW command in BASIC. The latter will delete the program currently in memory and clear all variables before entering a new program.

INITIALIZATION

Whenever a diskette is inserted into the drive, for any reason, it MUST be initialized to ensure that the information on the BAM (in the disk memory) is the proper information for the diskette currently in the drive. Failure to properly initialize a diskette each time it is inserted or reinserted into the drive will result in a DISK ID MISMATCH ERROR and/or loss of data.

The VIC-1540 utilizes a DOS 2.6 software routine each time the disk is addresed to determine if initialization is required. If a different ID is detected, the VIC-1540 will automatically initialize the new disk. Operator initialization is not required if unique IDs are assigned each diskette.

The format of INITIALIZE command is:

```
    PRINT#15, "INITIALIZEdr"
```

Where: dr=drive number 0 (0 may be ommitted).

Note: You may abbreviate INITIALIZE to I.

```
Example 1:  OPEN 15, 8, 15
            PRINT#15, "I0"

Example 2:  OPEN 15, 8, 15,"I"
            (You may omit the closing double quotation)
```

NOTE: FILE OPEN ERROR could occur if a previously opened file was addressed with a second OPEN command. When this error appears such logical file must be properly closed by typing CLOSE lfn.

The diskette in the drive is now initialized. Do not confuse formatting and initialization. Remember that formatting is usually a one-time operation and that re-formatting a disk will destroy previously stored data.

Since the VIC-1540 initialization function depends upon a change of ID to detect a change of diskette, inserting a diskette with an ID identical to one previously used may lead to a loss of data. This happens because the computer will reference the BAM left over from the previous diskette. Since the IDs are identical the DOS assumes there have been no change of diskette. A SAVE command may now cause new data to be written over good data already present on the disk because the DOS will use the old map of available storage area, instead of the current one. The results are unpredicable, and the diskette may become totally useless. For this reason, unique disk IDs must be used whenever possible for each diskette.

THE DIRECTORY

Confirm that the newly formatted disk has the correct ID and disk name by using one of the following methods to list the directory. The directory display includes the following information:

- Disk name
- Disk ID
- DOS version number
- File name
- File type
- Number of blocks used
- Number of available (free) blocks

To list the directory, at first type LOAD "$0", 8. (LOAD "$", 8)

LOAD$

This procedure will destroy any program currently in computer memory when the directory is LOADed.

STEP 1: Place a formatted disk in the drive.

STEP 2: Type: LOAD"$", 8 then press RETURN.

The screen displays:

```
LOAD"$", 8
SEARCHING FOR $
LOADING
READY'
■
```

STEP 3: Type: LIST

The directory for the drive will be displayed.

Printing The Directory

Quite often, it becomes convenient to affix a diskette directory listing directly on the protective jacket. This permits the user to scan the printed directory listing without having to insert the diskette into the drive to obtain this information. Should you desire to print the directory, place the diskette in the drive and enter the following commands:

LOAD "$", 8	Loads the directory.
OPEN 4, 4:CMD4	Opens device 4 (printer) and changes the primary output device to 4.
LIST	Prints the directory.
PRINT#4:CLOSE4	Returns output to the screen and closes the file.

VALIDATE

The VALIDATE command traces through each block of data contained in all files on the diskette. If this trace is successful, a new BAM is generated in the disk memory and written to the diskette. Any blocks which have been allocated but are not associated with a file name, as in the case of direct access files will be freed for use.

In addition to reconstructing the BAM, VALIDATE deletes files from the directory that were never properly closed. If a READ error is encountered during a VALIDATE, the operation aborts and leaves the diskette in its previous state. If a VALIDATE error does occue, you must re-initialize before proceeding.

The format of VALIDATE is:

```
PRINT#15, "VALIDATEdr"
```

Where: dr=drive number 0 (0 may be ommited)

NOTE: You may abbreviate VALIDATE to V.

Example:
```
OPEN 1, 8, 15
PRINT#1, "V0"
```

or

OPEN 1, 8, 15 "V"

COPY

The COPY command allows you to create multiple copies (under different names) of files on the same diskette. This command can also be used to concatenate data files. Up to four source files can be concatenated into the destination file. The COPY command may be abbreviated with a C.

COPY disk command can be formatted two ways depending upon application:

To copy a single file: PRINT#1fn, "COPYddr:dfn=sfn"

or

PRINT#1fn, "Cddr:dfn=sdr:sfn"

To concatenate and copy: PRINT#1fn, "COPYddr:dfn=sfn, sdr:sfn . . .

or

PRINT#1fn, "Cddr: dfn=sdr;sfn, sdr:sfn .....

Where: ddr=is the destination drive 0 (0 may be ommitted)

dfn=is the destination file name. This name must be a new name.

sdr=is the source drive 0 (0 may be omitted)
sfn=is the source file name.

Example 1: PRINT#1, "CO:ACCT1=0:ACCT"

A file named ACCT, is copied under a new name ACCT1

Example 2: PRINT#1,"CO:JDATA=O: ACCT1, O: ADATA, O:BDATA"

Files are concatenated into a file. Note that file names should be short, as the maximum length of a disk command string is 40 characters.

NOTE: The COPY command is normally used in the Dual Drive Floppy disk to copy files from one diskette to another. But this function can not be used in the VIC-1540 which is a Single Drive Floppy Disk.

RENAME

The RENAME command renames an existing file. A file can not already exist with the file name specified in the command or the FILE EXISTS error message will be generated.

The format of RENAME is:

```
PRINT#1fn, "RENAMEdr:nfn=ofn"
```

Where: dr=the drive number 0 (0 may be ommitted)

nfn=the new name of the file.

ofn=the old name of the file.

lfn=a logical file number. You assign this number arbitrarily and it may be any whole number between 1 and 255.

NOTE: The letter R is a legal abbreviation for RENAME.

NOTE: Close any open files before using the RENAME command since the disk will not execute this command on any active files.

SCRATCH

The SCRATCH command erases unwanted files from the specified diskette and its directory. You can erase on file, several files, or all the files on a diskette.

The format of SCRATCH IS:

```
PRINT# 1fn, "sdr:fn, dr:fn . . . :dr:fn"
```

Where: dr is the drive number 0. (0 may be omitted)
fn is the name of the file to be erased.

To erase one file enter, the entire name of the file:

Example: `PRINT#1, "SO:ACCT"`

To erase several files with unrelated names, enter the entire name of each file to be deleted:

Example: PRINT#1, "SO:ACCT, CUSTOMER, INV"

To erase several files at one time when names have something in common, refer to the rules in clapter 8 concerning pattern matching.

You may erase all files on a diskette using pattern matching as in the following example:

Example: PRINT#1, "S0:*"

NOTE

Chapter 4
BASIC COMMANDS FOR DATA HANDLING

BASIC COMMANDS ASSOCIATED WITH FLOPPY DISK DRIVES

The BASIC commands described in this chapter, allow the user to communicate with and transfer data to and from the disk drive.

These commands are available for ALL versions of Commodore BASIC:

```
OPEN1fn, 8, sa, "dr:fn, ft, mode"     VERIFY "dr:fn", 8
CLOSE1fn                              PRINT#1fn
LOAD "dr:fn" 8                        GET#1fn
SAVE "dr:fn", 8                       INPUT#1fn
```

Where: 1fn=logical file number (any number between 1 and 255)
fn=file name supplied by user
dr=disk drive number 0 : (0 may be omitted)
8=device number (8 for disk, 2 for second cassette, 4 for printer)
sa=secondary address
ft=the file type. It may be SEQ (for sequential), USR (for user), or PRG (for program)
mode=either READ(R) or WRITE(W).

All upper-case characters shown in format are essential for the proper execution of a command and must be typed by use. These commands are entered via the keyboard using unshifted characters only.

NOTE: The device number of the disk drive is set at 8 at the factory prior to shipment. If you wish to change the device number to use multiple number of VIC-1540 etc. Please consult your Commodore dealer (the device number may be any number from 8 to 11)

SAVE (Writing a Program to a Diskette)

If a program is in computer memory, it can be moved to a diskette for storage. This is accomplished with the SAVE commands.

Any data transferred with the SAVE command are automaticaly designated by the DOS as a program (PRG) file. Command transfers PRG files from the computer's memory to the specified diskette. You must specify the drive number, the program name, and the device number. The device number will default to device 1 which is the tape unit if it is not specified.

The format of SAVE is:

```
SAVE"dr:fn", dn
```

Where: dr=is the drive number 0 (0: may be omitted)
fn=is any file name of 16 characters or less you wish to assign to the file to be transferred to the diskette. Blanks are counted as characters.
dn=is the device number and it must be 8.

This following example illustrates creating a one line program, SAVEing it on the diskette in the drive under the name TESTPROG.

Example:
```
10?"THIS IS A TEST"
SAVE"0:TESTROG", 8
```

Another format of SAVE is

```
SAVE"@dr:fn", dn
```

Where @=means to replace the contents of an existing diskette file.

fn=is the name of the diskette file whose contents is to be replaced.

Example: SAVE "@0: TESPROG", 8

In this case a program named TESTPROG will be replaced, but if the specified program does not exist, then normal SAVE procedures are executed.

NOTE: In case @ is used in the SAVE command, avoid its repeated use. If @ must be repeatedly used, execute the VALIDATE command before the SAVE command.

VERIFY

The VERIFY command performs a byte-by-byte comparison of the file just written to the diskette against the file in the computer's memory.

The format of the VERIFY command is:

```
VERIFY "dr:fn", dn
```

Where: dr=the drive number 0 (0: may be omitted)

fn=the same file name as used in the immediately proceding SAVE command.

dn=the device number, in this case 8.

Another format of VERIFY is:

```
VERIFY "*", dn
```

Where: * causes the program just saved to be verified.

You should give the VERIFY command everytime you use the SAVE command to place a file on a diskette. If you do this, you will always know whether or not the file was copied correctly.

LOAD (Reading a Program from a Diskette)

A program stored on diskette may be loaded into memory using the LOAD command.

The LOAD command transfers PRG files from the specified diskette to the computer's memory. You must specify the drive number, the program name, and the device number. The device number will default to unit 1 which is the cassette unit. The format of LOAD is:

```
LOAD"dr:fn", dn
```

Where: dr=is the drive number 0. (0: may be omitted)

fn=is the file name previously specified in the SAVE command and/or stored in the disk directory.

dn=is the device number and it must be 8.

The following example illustrates how a program is loaded from the diskette into the computer memory, then executed. To do this example, first type NEW and depress RETURN key to clear your computer's memory so that you can see that it really works. Don't confuse the NEW command in BASIC with the NEW disk command used to format your disk.

Example 1:
```
LOAD"0:TESTPROG", 8
READY.
RUN
THIS IS A TEST
```

A successful LOAD closes all open files. Therefore you must give a new OPEN command in order to continue communicating with the disk drive command and error channel.

OPEN

This command sets up a correspondence between a logical file number and a file which exists on disk. It also reserves the buffer space within the disk unit for operations on the file being opened.

The format of the complete OPEN command is:

```
OPENlfn, dn, sa, "dr:fn, ft, mode"
```

Where: 1fn=the logical file number

dn=the device number, in this case 8.

sa=the secondary address. It may be any number from 2 to 14 and may be used either for input or output as specified in mode. See note below

dr=the drive number 0. (0: may be omitted)

fn=the name of the file.

ft=the file type. It may be SEQ (for sequential), USR (for user), or PRG (for program).

mode describes how the channel is to be used. It may be either READ (R) or WRITE (W).

NOTE: Secondary address 15 is the command and error channel and has special uses which are discussed in subsequent chapters. Secondary addressed 0 and 1 are reserved by the operating systems (BASIC and DOS) for LOADing and SAVEing programs.

Examples:

```
OPEN 2, 8, 2, "0:FILE1, SEQ, WRITE"

OPEN 3, 8, 9, "0:TESTDATA, PRG, WRITE"

OPEN 8, 8, 8, „0:NUM,USR, READ"
```

The contents of an existing file may be replaced by preceding the drive number with an at sign (@) in the OPEN command.

```
OPEN 3, 8, 5, "@0:JDATA, USR, WRITE"
```

If the spefified file does not exist, then normal OPENing procedures are executed.

You can also assign some of the OPEN parameters to a variable name as illustrated in these examples:

Example 1:
```
FL$="0:FILEA, SEQ, READ"
OPEN 1, 8, 14, FL$
```

Example 2:
```
FL$="0:FILEA"
OPEN 1, 8, 14, FL$+", SEQ, WRITE"
```

The preceding methods are convenient when it is necessary to open several channels to the same file name.

CLOSE

The CLOSE command closes a file opened by the OPEN command. Its format is:

```
CLOSE lfn
```

Where: lfn=the logical file number of a file opened by the OPEN command.

Always close a file after working with it. You are not allowed to have more than ten open files in the computer and five in the disk drive, so it is prudent to make a habit of closing files as soon as possible. This way you will always have the maximum number of files available for use.

CLOSING THE COMMAND CHANNEL

Closing the command channel closes all channels associated with the disk drive. No other part of the logical file environment is affected. That is, the computer does not recognize that the files have been closed.

The following example illustrates a situation in which several channels are closed down by a single CLOSE command.

Example:

`OPEN 1, 8, 15`	The command channel is opened.

`OPEN3, 8, 2, "0:FILE1, SEQ, WRITE"`
`OPEN4, 8, 5, "0:FILE2, SEQ,WRITE"`
	Data Channels are opened for writing.

`PRINT#3, "IMPORTANT DATA"`
`PRINT#4, "MORE DATA"`	

`OPEN3, 4`	A channel is opened to the printer by mistake.

`?FILE OPEN ERROR?`
`READY.`
`■`
	An error message is displayed on the screen.

Since there was an error, all logical files in the computer are closed, but the channels in the disk drive are still open. To close the disk channels, type:

```
OPEN1, 8, 15
CLOSE1
```

Now all data channels in the disk drive are properly closed.

CLOSING THE DATA CHANNEL

The CLOSE command closes a file and the data or command channel associated with it. Whenever you close a file opened with a write channel, the closing of that file writes the final block of data to the disk and updates the disk directory. When you close a file opened with a read channel, that channel is simply closed down.

NOTE: When a drive is initialized with INITIALIZE, NEW, or VALIDATE, all channels associated with that drive are deleted. These commands should not be executed when there are any files open since the files will be disrupted.

PRINT#

The PRINT# command transmits a disk command string to the drive.

The format of PRINT# is:

```
PRINT#lfn, "commandstring"
```

Where: lfn =a file previously opened using secondary address 15

"commandstring"=disk handling or disk file handling commands. These disk commands are discussed in detail in Chapter 3 of this manual.

PRINT# may also be used to transmit data to a previously-opened sequential or user file. In CBM BASIC V2 the logical file number 1 to 127 send carriage return alone and the logical file number 128 to 255 send CRLF. A semicolon (;) must be used as a terminator for each PRINT# statement when using the logical file number 128 to 255 to avoid sending extraneous line feeds to the diskette.
It is important to be aware of this face because the carriage return alone is seen as a terminatory by the DOS. The line feed is then stored in the file as the first character in the next record. To avoid this, use the following format:

Example: `PRINT#128, "JONESABC"; CHR$(13);`

The CHR$(13) is the carriage return necessary for the proper terminating of the record on the disk. When that record is input, the result will be JONESABC which is the desired result.

The following format may that be used:

```
PRINT#1fn, A$
```

It will produce the desired value of A$ for the record, and will not interfere with the next record.

Several variables may be written to the disk at the same time.

The format:

```
PRINT#1fn, A$, B$, C$
```

will result in a single variable (A$+B$+C$) being retrieved by the input command.

The format:

```
PRINT#1fn,A$CHR$(13)B$CHR$(13)C$
```

will result in the variables A$, B$, and C$ being separated by carriage returns, and they may then be input as separate variables.

INPUT#

The INPUT# command is used to transfer information from a peripheral device such as the disk drive into computer memory. INPUT# is valied only when used in a program and only when referencing a logical file that has been OPENed for input.

The format for INPUT# is:

```
INPUT#1fn, A$          or          INPUT#1fn, A
```

Where: 1fn=file previously opened using secondary address 15

A$=astring variable which will contain the data transferred.

A=a numeric variable which will contain the data transferred.

INPUT may also be used to transfer several strings of data at one time:

```
INPUT#1fn, A$, B$, C$
```

Where: A$, B$, C$ will contain the data transferred from the disk.

In this format, the data strings must have been separated by carriage returns (CHR$(13)) at the time they were written to the disk in order to be retrived separately. No single string may contain more than 88 characters if it is to be INPUT.

Example 1:

```
.
.
20 INPUT#2, A
.
```

Input the next data item which must be in numeric form and assign the value to aviable A.

Example 2:

```
.
.
10 INPUT#8, A$
.
```

Input the next data item as a string and assign it to variable A$.

Example 3:

```
.
.
60 INPUT#7, B, C$
.
```

Input the next two data items and assign the first to numeric variable B and the second to string variable C$.

For strings longer than 88 characters, the GET# commmand must be used.

GET#

The GET# command is used to transfer individual bytes of information from an serial device such as the disk drive into computer memory. GET# is valid only when used in a program and only when referencing a file that has been OPENed.

The format of GET# is:

```
GET#1fn, A$
```

Where: 1fn=a file previously opened using secondary address 15

A$=a string variable which will contain the data transferred.

GET# may also be used to transfer several bytes of information, which is useful for retrieving strings which has been written to the disk in a format which is unacceptable for the INPUT command (strings longer than 88 characters).

For exampel:

```
10 AA$=""
20 FOR I=I TO 254
30 GET#1fn, A$
40 AA$=AA$+A$
50 NEXT
```

is a program segment which would result in a string of length 254 being transferred from the disk (logical file number 1fn) to the computer memory and stored in the variable AA$.

MOVING A TAPE PROGRAM TO DISK

This example illustrates a session with the computer, a tape cassette and a disk drive. The purpose is to copy a cassette program to a diskette. The program is than read from the diskette to the computer's memory and printed. It is assumed that the BASIC program was previously stored on the cassette.

Example:

```
LOAD"DEMO"

PRESS PLAY ON TAPE
OK

SEARCHING FOR DEMO
FOUND DEMO
LOADING
READY.
```

Load the file from the cassette tape to the computer's memory.

```
SAVE "DEMO" , 8
VERIFY "DEMO", 8
READY.
```

Create a program file containing the program on diskette.

```
NEW
```

Erase everything from memory. (The NEW command in BASIC will clear memory; the NEW disk command will format a disk.)

```
LOAD "DEMO", 8
SEARCHING FOR DEMO
LOADING
READY.
```

Load the program back into the computegr's memory

```
RUN
```

Run the program to verify it has been loaded.

Chapter 5
ADVANCED DISK PROGRAMMING

This chapter provides detailed information about DOS structure and disk utility commands. The utility commands provide the programmer with low-level functions that may be used for special applications such as special disk handling routines and random access techniques.

COMMODORE DISK OPERATING SYSTEM (DOS)

The DOS file interface controller is responsible for managing all information between the disk controller and the serial interface. Most disk I/O is performed on a pipelined basis, resulting in a faster response to a requested operation.

The file system is organized by channels which are opened with the BASIC OPEN statement. When executed with the OPEN statement, the DOS assigns a workspace to each channel and allocates either one or two disk I/O buffer areas. If either the workspace or the buffer is not available, a NO CHANNEL error is generated. The DOS also uses the channel structure to search the directory, and to delete and copy files.

Three of the eight buffers are used by the DOS for the Block Availability Maps (BAM), variable space, command channel I/O, and disk controller's job queue.

The job queue is the vital link between the two controllers. Jobs are initiated on the file side by providing the disk controller with sector header and type of operation information. The disk controller seeks the optimum job and attempts execution. An error condition is then returned in plade of the job command. If the job is unsuccessful, the file side re-enters the job a given number of times, depending upon the operation, before generating an error message.

The secondary address given in the OPEN statement is used by DOS as the channel number. The number the user assigns to a channel is only a reference number that is used to access the work areas, and is not related to the DOS ordering of channels. The LOAD and SAVE statements transmit secondary addresses of 0 and 1, respectively. The DOS automatically interprets these secondary addresses as LOAD and SAVE functions. Unless these functions are desired when opening files, avoid secondary addresses of 0 and 1. The remaining numbers, 2 through 14, may be used as secondary addresses to open up to five channels for data.

Special OPEN and CLOSE Statements for Direct Access

The BASIC statement:

```
    OPEN 2, 8, 4, "#"
    or
    OPEN 2, 8, 4, "#7"
```

opens a channel to one buffer, to be used with the block commands. The first available buffer is allocated to channel 4 in the first example. The second example is an attempt to allocate buffer 7 to the channel. If the buffers are not available, a NO CHANNELS error condition is generated. The explicit buffer allocation can be used to reserve a buffer for position dependent code as in the case of an execute command.

You can find the number of the allocated buffer by executing a GET# statement. The byte transmitted is the buffer number. The only time you can get a buffer number is before any write or read operation to that buffer.

The CLOSE statement clears the opened channel and writes the BAM to the diskette that was last used by that channel. It is recommended that to avoid confusion, you limit yourself to accessing one drive with any direct access channel.

DISK UTILITY COMMAND SET

The disk utility command set consists of the following commands:

Commands	Abbreviations	General Format
BLOCK-READ	B-R	"B-R:"ch,dr,t,s
BLOCK-WRITE	B-W	"B-W:"ch,dr,t,s
BLOCK-EXECUTE	B-E	"B-E:"ch,dr,t,s
BUFFER-POINTER	B-P	"B-P:"ch,p
BLOCK-ALLOCATE	B-A	"B-A:"dr,t,s
BLOCK-FREE	B-F	"B-F:"dr,t,s
memory-write	M-W	"M-W"adl/adh/nc/data
memory-read	M-R	"M-R"adl/adh
memory-execute	M-E	"M-E"adl/adh
USER	U	"Ui:parms"

Where:	ch	= the *channel number* in DOS: identical to the secondary address in the associated OPEN statement.
	dr	= the *drive number:* 0
	t	= the *track number:* 1 through 35.
	s	= the sector number, 0 through 20. For each track number, the sector range is shown in Table 4.
	p	= the *pointer position* for the buffer pointer.
	adl	= the *low byte* of the address*.
	adh	= the *high byte* of the address*.
	nc	= the *number of characters:* 1 through 34*.
	data	= the actual data in hexidecimal. This is transmitted by using the CHR$ function, i.e. CHR$(1) would send the binary equivalent of hexidecimal 01, (decimal 1).
	i	= the *index* to the User Table.
	parms	= the *parameters* associated with the U command (optional).

The values used in conjunction with the memory commands exist in the VIC-1540 as hexidecimal values and must be transmitted as CHR$(n), where n is the decimal equivalent of the desired hexidecimal value.

NOTE: If using variables the format must have only the command in quotes. For example:

```
"B-R:"ch,dr,t,s          correct

"B-R:ch,dr,t,s"          incorrect
```

To avoid confusion, it is good practice to use this format when using variables or constants.

As implied in the preceding format, these commands may be abbreviated to the first character of each of the key words. Abbreviations only are accepted for those commands shown in lower case. The parameters associated with each command are searched for starting at a colon (:), or in the fourth character position if a colon is not present. The example following shows four ways that the same block-read command may be given.

NOTE: Initialize the disk before the buffer read or write.

Examples:
```
"BLOCK-READ: " 2,0,4,0
"B-R"2,0,4,0
"B-R"2;0;4;0
"B-READ:"2;0;4;0
```

Parameters following the key words within quotation marks may be separated by any combination of the following characters:

Character Name	Keyboard Representation
Skip	⟨ cursor right ⟩
Space	Space bar
Comma	

The use of these characters permits sending both ASCII strings and integers.

Parameters not within the confines of quotation marks should be separated by semi-colons (;).

In the following discussions, a PRINT# is assumed in all examples.

BLOCK-READ

This diskette utility command provides direct access to any block on a diskette in either disk drive. Used in conjunction with other block commands, a random access file system may be created through BASIC. This command finds the character pointer in the 0-position of the block. When a character in this position is accessed with GET# or INPUT#, an End-or-Identify (EOI) is sent. This terminates an INPUT# and sets the Status Word (ST) to 64 in the computer.

The format "B-R:"ch;dr;t;s is illustrated in the following example.

Example: "B-R:"5;0;18;0

Reads the block from track 18, sector 0 into channel 5 buffer area.

After using BLOCK READ to transfer the data to the buffer, the data may be transferred to memory by INPUT# or GET# from the logical file opened to that disk channel (i.e., using that secondary address).

The U1 command described under USER is similar to the BLOCK-READ command.

BLOCK-WRITE

When this command is initiated, the current buffer pointer is used as the last character pointer and is placed in the 0 position of the new buffer. The buffer is then written to the indicated block on the diskette and the buffer pointer is left in position 1.

The format "B-W:" ch;dr;t;s is illustrated in the following example.

Example: "B-W:" 7;0;35;10

Writes channel 7 buffer to the block on track 35, sector 10:

The U2 command described under USER is similar to the BLOCK-WRITE command.

BLOCK-EXECUTE

This command allows part of the DOS or user designed routines to reside on disk and be loaded into disk drive memory and executed. B-E is really a B-R with an addition. The File Interface Controller begins execution of the contents after the block is read into a buffer. Execution must be terminated with a return from the subroutine (RTS) instruction. Future system extensions or user-created functions may implement this command.

The format "B-E:"ch;dr;t;s is illustrated in the following example.

Example: "B-E:"6;0;1;10

Reads a block from track 1, sector 10 into channel 6 buffer and executes its contents beginning at position 0 in the buffer:

BUFFER-POINTER

This command changes the pointer associated with the given channel to a new value. This is useful when accessing particular fields of a record in a block or, if the block is divided into records, individual records may be set for transmitting or receiving data.

The format "B-P:"ch,p is illustrated in the following example.

Example: "B-P:"2;1

Sets channel 2 pointer to the beginning of the data area in the direct access buffer:

BLOCK-ALLOCATE

The appropriate BAM is updated in the DOS memory to reflect the indicated block as allocated (used). In future operations, the DOS skips over the allocated block when saving programs or writing sequential files. The updated BAM is written to diskette upon the closure of a write file or the closure of a direct access channel.

If the block requested has been previously allocated, the error channel indicates the next available block (increasing track and sector numbers) with a NO BLOCK error. If there are no blocks available that are greater in number than the one requested, zeroes are displayed as track and sector parameters.

The format "B-A:"dr;t;s is illustrated in the following example.

Example: "B-A:"0;10;0

Requests that block (sector) 0 of track 10 be flagged as allocated on the diskette.

NOTE: The error channel should always be check when using BLOCK ALLOCATE, so that if the block is already allocated, it will not be overwritten. If the block is allocated, the error message will also indicate the next available block.

Example: INPUT#15,EN,EM$,ET,ES

Reads the next track and sector, respectively, into ET and ES, assuming that 1fn = 15 has been previously OPENed to the disk error channel.

BLOCK FREE

The BAM is updated in memory to free the block indicated. The disk is updated on a close as previously mentioned. Neither the B-A nor B-F commands require that a direct access channel be open, since there is no channel relationship associated with the operations. However, if the BAM is to be updated on disk, these operations should be used in conjunction the other block commands.

The format "B-F:" dr;t;s is illustrated in the following example.

Example: "B-F:" 0;9;20
Free up the block (sector) 20 of track 9.

MEMORY

All three MEMORY commands are byte-oriented so that the user may utilize machine language programs. BASIC statements may be used to access information through the MEMORY commands by using the CHR$ function. The system accepts only M-R, M-W, and M-E: neither verbose spelling or the use of the colon (:) is permitted.

Memory-Write

This command provides direct access to the DOS memory. Special routines may be down-loaded to the disk drive through this command and then executed using the MEMORY-EXECUTE command or one of the USER (U) commands. Up to 34 bytes may be deposited with each use of the command. The low byte of the address must precede the high byte of the address.

The format "M-W:"adl/adh/nc/data is illustrated in the following example.

Example: "M-W:"CHR$(00)CHR$(7)CHR$(4)CHR$(32)CHR(0)CHR$(17)-CHR$(96)

Writes four bytes to $0700 (decimal 1792):

Memory-Read

The byte pointed to by the address in the command string may be accessed with this command. Variables from the DOS or the contents of the buffers may also be read with this command. The M-R command changes the contents of the error channel since it is used for transmitting information to the computer. The next GET# from the error channel (secondary address 15) transmits the byte. An INPUT# should not be executed from the error channel after a MEMORY-

READ command until a DOS command other than one of the MEMORY commands is executed.

The format "M-R:"adl/adh is illustrated in the following example.

Example: "M-R"CHR$(128);CHR$(0)

Accesses the byte located at $0080 (decimal 128):

Memory-Execute

Subroutines in the DOS memory may be executed with this command. To return to the DOS, terminate the subroutine with RTS ($60).

The format "M-E:" adl/adh is illustrated in the following example.

Example: "M-E"CHR$(128);CHR$(49)

Requests the execution of code beginning at $3180 (decimal 12672).

USER

This command provides a link to 6502 machine code according to a jump table pointed to by the special USER pointer. Refer to Table 5. The second character in this command is used as an index to the table. The ASCII character 0 through 9 or A through 0 may be used. Zero sets the USER pointer to a standard jump table that contains links to special routines.

The special USER commands U1 (or UA) and U2 (or UB) can be used to replace the BLOCK-READ and the BLOCK-WRITE commands.

The format of U1 is:

```
"U1:" ch; dr; t; s
```

U1 forces the character count (buffer pointer) to 255 and reads an entire block into memory. This allows complete access to all bytes in the block.

The format of U2 is:

```
"U2:" ch;dr;t;s
```

U2 writes a buffer to a block on the disk without changing the contents of position 0 as B-W does. This is useful when a block is to be read in (with B-R) and updated (B-P to the field and PRINT#), then written back to diskette with U2.

Table 3. Standard Jump Table

USER DESIGNATION	ALTERNATE USER DESIGNATION	FUNCTION	
U1	UA	BLOCK-READ replacement	
U2	UB	BLOCK-WRITE replacement	
U3	UC	jump to	$0500
U4	UD	jump to	$0503·
U5	UE	jump to	$0506
U6	UF	jump to	$0509
U7	UG	jump to	$050C
U8	UH	jump to	$050F
U9	UI	jump to	$FFFA
U:	UJ	power up vector	

U3 thru U9 commands are user-defined. The locations jumped to are located in the buffer areas of RAM and routines may be written to reside there and downloaded using the M-W command.

STRUCTURE OF DISKETTE

Any block on a diskette may be examined by using the program DISPLAY T&S, provided on the TEST/DEMO diskette.

Table 4. Block Distribution by Track

Track number	Block or Sector Range	Total
1 to 17	0 to 20	21
18 to 24	0 to 18	19
25 to 30	0 to 17	18
31 to 35	0 to 16	17

Tables 5 through 11 will assist the user in interpreting information obtained using the DISPLAY T&S program.

Table 5. VIC-1540 BAM Format

Track 18, Sector 0.		
BYTE	CONTENTS	DEFINITION
0,1	18,01	Track and sector of first directory block.
2	65	ASCII character A indicating 1540 format (Same as CBM 4040).
3	0	Null flag for future DOS use.
4-143		Bit map of available blocks for tracks 1-35.
*1 = available block 0 = block not available (each bit represents one block)		

Table 6. Directory Header

Track 18, Sector 0.		
BYTE	CONTENTS	DEFINITION
144-161		Disk name padded with shifted spaces.
162-163		Disk ID.
164	160	Shifted space.
165,166	50,65	ASCII representation for 2A which is DOS version and format type.
166-167	160	Shifted spaces.
171-255	0	Nulls, not used.
Note: ASCII characters may appear in locations 180 thru 191 on some diskettes.		

Table 7. Directory Format

Track 18, Sector 1.	
BYTE	DEFINITION
0,1	Track and sector of next directory block.
2-31	*File entry 1
34-63	*File entry 2
66-95	*File entry 3
98-127	*File entry 4
130-159	*File entry 5
162-191	*File entry 6
194-223	*File entry 7
226-255	*File entry 8

* Structure of single directory entry

BYTE	CONTENTS	DEFINITION
0	128 + type	File type OR'ed with $80 to indicate properly closed file. TYPES: 0 = DELeted 1 = SEQential 2 = PROGram 3 = USER 4 = RELative
1,2		Track and sector of 1st data block.
3-18		File name padded with shifted spaces.
19,20		Relative file only: track and sector for first side sector block.
21		Relative file only: record size.
22-25		Unused.
26,27		Track and sector of replacement file when OPEN@ is in effect.
28,29		Number of blocks in file: low byte, high byte.

Table 8. Sequential Format

BYTE	DEFINITION
0,1	Track and sector of next sequential data block.
2-256	254 bytes of data with carriage returns as record terminators.

Table 9. Program File Format

BYTE	DEFINITION
0,1	Track and sector of next block in program file.
2-256	254 bytes of program into stored in VIC memory format (with key words tokenized). End of file is marked by three zero bytes.

Figure 5 illustrates an expanded view of a single sector on a diskette formatted for the 1540. In addition to other information, each sector contains a data block consisting of 256 stored characters. Blocks within the same file are linked together by means of a two character block pointer. By pointing to the location of the next data block, block pointers enable the system to retrieve data from noncontinuous blocks. Retrieving the first data block within a file triggers a search for the next data block which, in turn, utilizes block pointers to locate related blocks until the entire file is assembled and made available for display. All PRG, SEQ, and USR files utilize this format.

A data block is addressed by track and sector. A 1540 diskette contains 35 tracks (or rings) numbered 1 to 35. The number of sectors per track will vary (as illustrated in Table 4) due to differences in track circumference and recording frequency.

The 1540 maintains a system file on track 18 which contains the BAM, diskette name, ID, and file directory. The BAM, resident in the first 128 bytes of sector 0, monitors available and occupied storage locations on diskette. The last 128 bytes of sector 0 are used to store the diskette name and ID. The file directory begins on the next sector, sector 1.

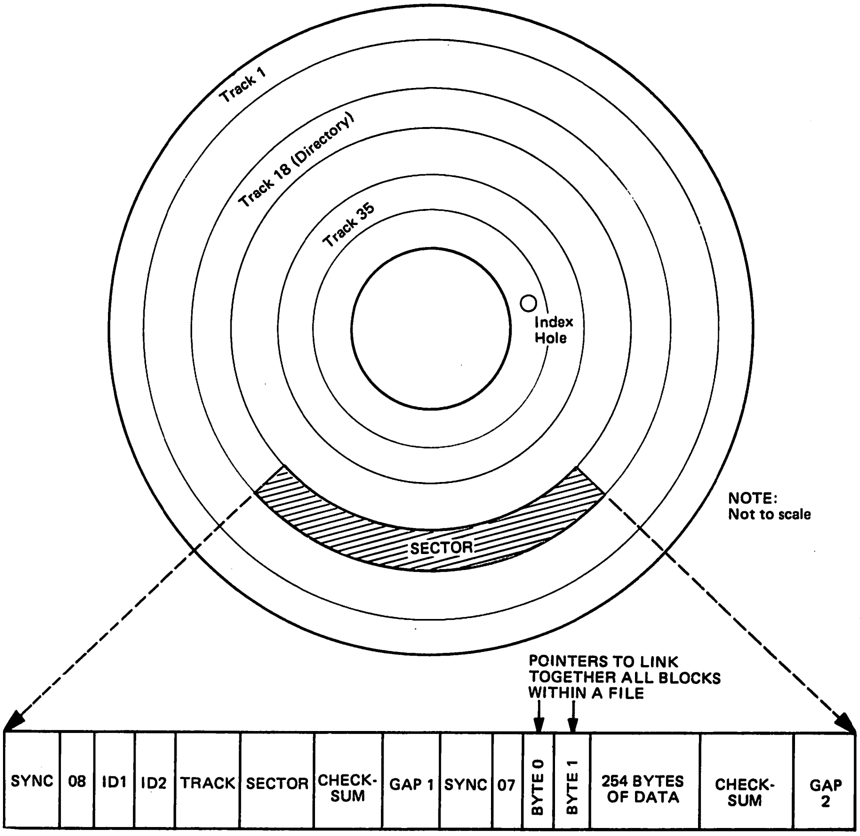


Fig. 5. VIC-1540 Format: Expanded View of a Single Sector

NOTES

Chapter 6
SEQUENTIAL FILE

A file handling of sequential file which is the most basic form of the date file is described in this chapter.

A sequential file is a file which may be read only in the sequence it was written in. For example, a file with records, A, B and C, must be read in the sequence of A, B and C if they were written in that sequence. The record C and not be directly read without reading the record A and B. If it is necessary to read the record C only, then A and B must be read without doing any work.

In the sequential file, no partial re-write of the record can be made. Only change which can be made to the sequential file is an addition of new record to the end of the file.

TO CREATE A SEQUENTIAL FILE AND ACCESSING

The following steps are taken to create a sequential file and accessing.

STEP 1 Open a file for write in.

```
OPEN 2, 8, 2, "0 : DATE, S,W"
```

A sequential file with a name DATE is opened to be written upon. Note the designation W for write in at the end of the command.

.STEP 2 Write in data in the file using PRINT #.

```
PRINT #2 A$; B$; C$
```

In this case the datas are, 3 string variables, A$, B$, and C$.

STEP 3 Close the file for the access later

```
CLOSE 2
```

STEP 4 Open the file with a name DATA to be read out.
The designation R for read out may be omitted.

OPEN #2, 8, 2, "0: DATA, S,R"

STEP 5 Read in the data from the sequential file to the program using INPUT #.

INPUT #2, X$, Y$, Z$

STEP 6 Check the end of the file by the ST variable.

IF (ST) AND 64 THEN CLOSE 2

EXAMPLE OF A SEQUENTIAL FILE PROGRAM

An example of a simple program to create a sequential file is shown below. Records consisting of string field and numeric field are written in and the contents of the records is displayed at the end by typing END.

```
1 REM ***************
2 REM *   EXAMPLE
3 REM * READ & WRITE
4 REM * A SEQUENTIAL
5 REM * DATA FILE
9 REM ***************
10 PRINT"[illegible]INITIALIZE DISK"
20 DIMA$(25)
30 DIMB(25)
40 OPEN15,8,15,"I0"
60 GOSUB 1000
70 CR$=CHR$(13)
80 PRINT
90 PRINT" [illegible]WRITE SEQ TEST FILE[illegible]"
95 PRINT
100 REM **************
101 REM *
102 REM *  WRITE SEQ
103 REM *  TEST FILE
104 REM *
```

```
105 REM *************
110 OPEN2,8,2,"@0:SEQ TEST FILE ,S,W"
115 GOSUB 1000
120 INPUT"A$,B";A$,B
130 IFA$="END"THEN 160
140 PRINT#2,A$","STR$(B)CR$;
145 GOSUB 1000
150 GOTO 120
160 CLOSE 2
200 REM *************
201 REM *
202 REM *  READ SEQ
203 REM *  TEST FILE
204 REM *
205 REM *************
206 PRINT
207 PRINT"  █READ SEQ TEST FILE█"
208 PRINT
210 OPEN2,8,2,"0:SEQ TEST FILE ,S,R"
215 GOSUB 1000
220 INPUT#2,A$(I),B(I)
224 RS=ST
225 GOSUB 1000
230 PRINTA$(I),B(I)
240 IFR S=64 THEN 300
250 IF RS<>0 THEN 400
260 I=I+1
270 GOTO 220
300 CLOSE 2
310 END:
400 PRINT"█BAD█DISK█STATUS█IS██"RS
410 CLOSE 2
420 END
1000 REM ************
1001 REM *
1002 REM *   READ
1003 REM *  THE ERROR
1004 REM *   CHANNEL
1005 REM *
```

```
1006 REM *************
1010 INPUT#15,EN,EM$,ET,ES:
1020 IF EN=0 THEN RETURN:
1030 PRINT"[illegible]ERROR ON DISK"
1040 PRINT"[illegible]"EN,EM$,ET,ES"[illegible]"
1050 CLOSE2
1060 END
```

Chapter 7

RANDOM FILE

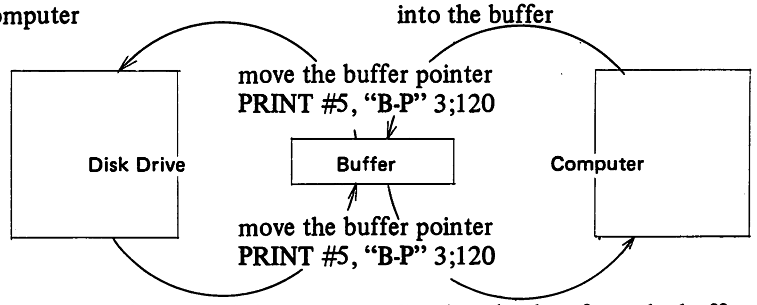
A random access file is created with the BLOCK and USER commands described in Chapter 5-Adranced Disk Programming.

DATA FLOW IN RANDOM FILE

In the random access file the data can not be as simply written in or read from as in the case of a sequential file.

It is necessary to understand the data flow between the computer, buffer and disk drive in order to transfer the data in the random access file. It can be illustrated as Fig. 6.

Channel 5	**Channel 3**
OPEN 5, 8, 15	OPEN 4, 8, 3, "#7"
PRINT #5, "B-W"; 3; 0; 6; 4	PRINT #4, A$; CHR$(44)I; CHR$(13)
Transfer the data from the buffer to the computer	Write in the data from the computer into the buffer



Transfer the data from the disk to the buffer	Read in the data from the buffer to the computer
PRINT #5, "B-R"; 3; 0; 6; 4	INPUT #4, A$, I

Fig. 6. Data Flow between Computer, Buffer and Disk Drive

The above can be more specifically described as:

STEP 1. Open a file

```
Ⓐ Disk File    OPEN 5, 8, 15
                    |  |  └─Command & Error Channel
                    |  └────Device Number
                    └───────Logical File Number

Ⓑ Buffer File  OPEN 4, 8, 3, "#7"
                    |  |  |    └─Buffer Number
                    |  |  └──────Channel Number
                    |  └─────────Device Number
                    └────────────Logical File Number
```

STEP 2. Move the buffer pointer

```
                        PRINT #5 "B-P:" 3; 120
                               |  |     |   └─Pointer Position
Logical File Number of Ⓐ───────┘  |     └─────Channel Number of Ⓑ
                                  └───────────Setting of Block Pointer
```

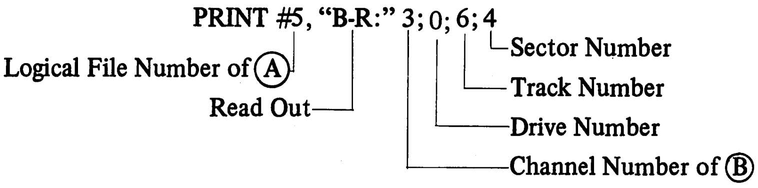
STEP 3. Place the data in the buffer

```
                        PRINT #4, A$; CHR$(44); I; CHR$(13)
                               |  |             |     |
Logical File Number of Ⓑ───────┘  |   Comma     |   Carriage Return
                                  └─────────────┴───Variables
```

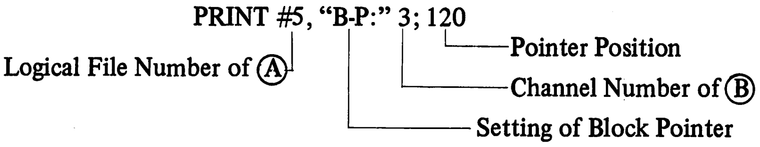
STEP 4. Transfer the data from the buffer to the disk

```
                        PRINT #5, "B-W:" 3; 0; 6; 4
                               |     |   |  |  |  └─Sector number
Logical File Number of Ⓐ───────┘     |   |  |  └────Track number
                          Write In───┘   |  └───────Drive number
                                         └──────────Channel Number of Ⓑ
```

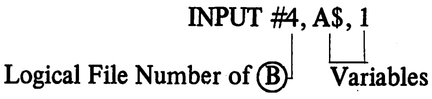
STEP 5. Transfer the data from the disk to the buffer



STEP 6. Move the buffer pointer



STEP 7. Take out the data from the buffer



EXAMPLE OF USING RECORD NUMBER

More practical program which uses variables will be as follows: It assumes OPEN 15, 8, 15 and OPEN CH, 8, CH, "#" and uses U1 and U2 instead of B-W and B-R.

```
5100 REM ***********************
5105 REM * FDD BLOCK READ      *
5108 REM ***********************
5110 GOSUB5330
5120 PRINT#15,"U1:";CH;FD;FT;FS
5130 PRINT#15,"B-P:";CH;FP
5140 GOSUB5270
5150 FORFI=1TOFX
5160 INPUT#CH,FB$(FI):
5180 NEXT
5190 RETURN
```

```
5200 REM ************************
5201 REM * FDD BLOCK WRITE      *
5202 REM ************************
5210 GOSUB5330
5220 PRINT#15,"B-P:";CH;FP
5230 FORFI=1TOF%:PRINT#CH,FB$(FI);CHR$(13);:NEXT
5240 PRINT#15,"U2:";CH;FD;FT;FS
5250 GOSUB5270
5260 RETURN
5270 REM ************************
5275 REM *  ERROR CHECK         *
5278 REM ************************
5280 INPUT#15,EN,EM$,ET,ES
5290 IFEN=0THENRETURN
5300 PRINT"ERROR STATUS:";EN;EM$;ET;ES
5310 INPUT"CONTINUE?";Y$:IFY$="Y"THENRETURN
5320 STOP
5322 REM ************************
5324 REM * SET TRACK & SECTOR  *
5326 REM ************************
5330 IFF<358THENF1=0:F2=22:F3=1:GOTO5370
5340 IFF>357ANDF<471THENF1=357:F2=20:F3=19:GOTO5370
5350 IFF>471ANDF<580THENF1=471:F2=19:F3=25:GOTO5370
5360 IFF>580THENF1=580:F2=18:F3=31
5370 FT=INT(((F-F1)-1)/(F2-1))+F3
5380 FS=F-F1-(FT-F3)*F2+(FT-F3-1)
5390 RETURN
```

Where: CH	= Channel number
FD	= Drive number
FP	= Buffer pointer

The line numbers 5270 to 5320 in the above are the error check routine. Also indicated in above by GOSUB 5330 is a subroutine which is for allocating a sector in the disk.

A track and sector must be designated in the random access which is rather troublesome and, therefore, a concept of record number is used here to reduce the burden of the programmer. For example, serial numbers as shown in Table below are assigned to the sector 0 in the track 1 to the sector 16 of the track 35, in VIC-1540. These serial numbers are called as the record number, which is indicated by F in the subroutine starting from the line number 5330 in the above example. When a F number is designated and this subroutine is called, the corresponding track and sector are allocated automatically. The directory will not be destructed as the track 18 has been avoided.

Table 10. Allocation of Record Number

Track Number	Sector	Number of Sector/Track	Record No.
1 to 17	0 to 20	21	1-357
18 to 24	0 to 8	19	358-471
25 to 30	0 to 17	18	478-579
31 to 35	0 to 16	17	586-664

EXAMPLE OF A RANDOM FILE PROGRAM

A simple program has been made using method to create a random access file described above. This program can be used for the preparation, deletion and inquiry of an address list.

Line number 50 – 76	Job menu.
Line number 100 – 190	Creation of file
Line number 200 – 280	Making deletion to the file
Line number 300 – 390	Search processing
Line number 5100 – 5190	File input routine
Line number 5200 – 5260	File output routine

```
10 REM ************************
12 REM * RANDOM FILE EXAMPLE *
14 REM ************************
16 DIMI$(664):FD=0:FX=5:CH=2:FP=1
18 PRINT"▓▓▓──────────────────────────";
20 PRINT"  ▓INSERT DATA SHEET▓"
22 PRINT"──────────────────────────";
24 PRINT"▓▓▓    START PRESS ▓'S'▓"
26 GETP$:IFP$<>"S"THEN26
28 OPEN15,8,15,"I0":OPEN2,8,2,"#"
30 PRINT"▓▓──────────────────────────";
32 PRINT"INDEX FILE OPERATION"
34 PRINT"──────────────────────────"
```

```
36 INPUT"▨      ▮▮NEW SHEET? N■■■▮";O$:PRINT"▨"
38 IFO$="N"THEN46
40 IFO$<>"Y"THEN36
42 PRINT:PRINT"     WAIT!":FORI=1TO664
44 PRINT"▨▮▮▮▮▮▮▮▮▮▮▨▨▨▨▨▨▨▨▨▨     ■■■■▮"I;:I$(I)="/"
45 PRINTI$(I):NEXT:GOTO50
46 OPEN5,8,5,"0:INDEX,S,R"
47 FORI=1TO664:INPUT#5,I$(I)
48 PRINT"▨▨▨▨▨▨▨▨▨▨▨▮▮▮▮▮▮WAIT!      ■■■■▮"I;I$(I)
49 NEXTI:CLOSE5
50 PRINT"▨▨──────────────────────────";
52 PRINT"      JOB MENU       "
54 PRINT"──────────────────────────";
56 PRINT
58 PRINT"      1=CREATE"
60 PRINT"      2=DELETE"
62 PRINT"      3=SEARCH"
64 PRINT"      0=END"
65 PRINT
66 INPUT"1 , 2 , 3 , 0   1■■▮";O$
68 IFO$="0"THENCLOSE15:CLOSE5:CLOSE2:END
70 IFO$="1"THEN104
72 IFO$="3"THEN300
74 IFO$<>"2"THEN50
76 GOTO200
100 REM ***********************
102 REM * MASTER FILE CREATE  *
103 REM ***********************
104 PRINT"▨▨▨──────────────────────────";
105 PRINT"  MASTER FILE CREATE"
106 PRINT"──────────────────────────"
107 INPUT"▮RECORD NO. = 0■■■";F
109 IF F=0THEN170
110 INPUT"NAME      =.■■▮";FB$(1)
120 INPUT"ADDRESS   =.■■▮";FB$(2)
130 INPUT"ZIP       =.■■▮";FB$(3)
132 INPUT"TEL       =.■■▮";FB$(4)
134 INPUT"COMMENT   =.■■▮";FB$(5)
140 GOSUB5200
150 I$(F)="1"
```

```
160 GOTO104
170 OPEN5,8,5,"@0:INDEX,S,W"
175 FORI=1TO664:PRINT#5,I$(I);CHR$(13);
180 PRINT"█████████████WAIT!      █████"I;I$(I)
185 NEXT:CLOSE5
190 GOTO74
200 REM ***********************
201 REM *  MASTER FILE DELETE *
202 REM ***********************
210 PRINT"███─────────────────────────",
212 PRINT"  MASTER FILE DELETE"
214 PRINT"─────────────────────────":PRINT
220 INPUT"█RECORD NO. = 0████",F
230 IFF=0THEN260
235 IFI$(F)<>"1"THEN220
240 I$(F)="/":PRINT"██RECORD NO."F;"DELETE█"
250 GOTO220
260 OPEN5,8,5,"@0:INDEX,S,W"
265 FORI=1TO664:PRINT#5,I$(I);CHR$(13);
270 PRINT"█████████████WAIT!      █████"I;I$(I)
275 NEXT:CLOSE5
280 GOTO50
300 REM ***********************
301 REM *  FILE SEARCH        *
302 REM ***********************
310 PRINT"███─────────────────────────";
312 PRINT"        SEARCH       "
314 PRINT"─────────────────────────":PRINT
320 INPUT"█RECORD NO. = 0████";F
321 IFF=0THEN50
322 IFI$(F)<>"1"THEN320
325 GOSUB5100
360 PRINT"NAME      = ";FB$(1)
370 PRINT"ADDRESS   = ";FB$(2)
380 PRINT"ZIP       = ";FB$(3)
382 PRINT"TEL       = ";FB$(4)
383 PRINT"COMMENT   = ";FB$(5)
385 PRINT"----------------------"
390 GOTO320
```

```
5100 REM ************************
5105 REM * FDD BLOCK READ       *
5108 REM ************************
5110 GOSUB5330
5120 PRINT#15,"U1:";CH;FD,FT;FS
5130 PRINT#15,"B-P:";CH;FP
5140 GOSUB5270
5150 FORFI=1TOFX
5160 INPUT#CH,FB$(FI):
5180 NEXT
5190 RETURN
5200 REM ************************
5201 REM * FDD BLOCK WRITE      *
5202 REM ************************
5210 GOSUB5330
5220 PRINT#15,"B-P:";CH;FP
5230 FORFI=1TOFX:PRINT#CH,FB$(FI);CHR$(13);:NEXT
5240 PRINT#15,"U2:";CH;FD;FT;FS
5250 GOSUB5270
5260 RETURN
5270 REM ************************
5275 REM *  ERROR CHECK         *
5278 REM ************************
5280 INPUT#15,EN,EM$,ET,ES
5290 IFEN=0THENRETURN
5300 PRINT"ERROR STATUS:",EN;EM$;ET;ES
5310 INPUT"CONTINUE?";Y$:IFY$="Y"THENRETURN
5320 STOP
5322 REM ************************
5324 REM * SET TRACK & SECTOR   *
5326 REM ************************
5330 IFF<358THENF1=0:F2=22:F3=1:GOTO5370
5340 IFF>357ANDF<471THENF1=357:F2=20:F3=19:GOTO5370
5350 IFF>471ANDF<580THENF1=471:F2=19:F3=25:GOTO5370
5360 IFF>580THENF1=580:F2=18:F3=31
5370 FT=INT(((F-F1)-1)/(F2-1))+F3
5380 FS=F-F1-(FT-F3)*F2+(FT-F3-1)
5390 RETURN
```

Chapter 8

ERROR MESSAGES— PATTERN MATCHING FILE NAMES—

REQUESTING ERROR MESSAGES

When the drive indicator of the disk drive flashes, it indicates an error occurred in the disk drive.

The execution of the following program displays the error on the computer screen and resets the device error indicator:

```
10 OPEN 1,8,15
20 INPUT#1,A,B$,C,D
30 PRINT A,B$,C,D
```

where	A	= error message number
	B$	= error message
	C	= track
	D	= sector

SUMMARY OF DOS ERROR MESSAGES

0	OK, no error exists.
1	Files scratched response. Not an error condition.
2–19	Unused error messages: should be ignored.
20	Block header not found on disk.
21	Sync character not found.
22	Data block not present.
23	Checksum error in data.
24	Byte decoding error.
25	Write-verify error.

26	Attempt to write with write protect on.
27	Checksum error in header.
28	Data extends into next block.
29	Disk id mismatch.
30	General syntax error.
31	Invalid command.
32	Long line.
33	Invalid filename.
34	No file given.
39	Command file not found.
50	Record not present.
51	Overflow in record.
52	File too large.
60	File open for write.
61	File not open.
62	File not found.
63	File exists.
64	File type mismatch.
65	No block.
66	Illegal track or sector.
67	Illegal system track or sector.
70	No channels available.
71	Directory error.
72	Disk full or directory full.
73	Power up message, or write attempt with DOS mismatch.
74	Drive not ready.

DESCRIPTION OF DOS ERROR MESSAGES

NOTE: Error message numbers less than 20 should be ignored with the exception of 01 which gives information about the number of files scratched with the SCRATCH command.

20: READ ERROR (block header not found)
The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.

21: READ ERROR (no sync character)
The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/write head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.

22: READ ERROR (data block not present)
The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.

23: READ ERROR (checksum error in data block)
This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.

24: READ ERROR (byte decoding error)
The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.

25: WRITE ERROR (write-verify error)
This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.

26: WRITE PROTECT ON
This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. Typically, this is caused by using a diskette with a write protect tab over the notch.

27: READ ERROR (checksum error in header)
The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.

28: WRITE ERROR (long data block)
The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad diskette format (the data extends into the next block), or by hardware failure.

29: DISK ID MISMATCH
This message is generated when the controller has been requested to access a diskette which has not been initialized. The message can also occur if a diskette has a bad header.

30: SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel.

Typically, this is caused by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.

31: SYNTAX ERROR (invalid command)
The DOS does not recognize the command. The command must start in the first position.

32: SYNTAX ERROR (long line)
The command sent is longer than 58 characters.

33: SYNTAX ERROR (invalid file name)
Pattern matching is invalidly used in the OPEN or SAVE command.

34: SYNTAX ERROR (no file given)
The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon (:) has been left out of the command.

39: SYNTAX ERROR (invalid command)
This error may result if the command sent to command channel (secondary address 15) is unrecognizable by the DOS.

50: RECORD NOT PRESENT
Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.

51: OVERFLOW IN RECORD
PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.

52: FILE TOO LARGE
Record position within a relative file indicates that disk overflow will result.

60: WRITE FILE OPEN
This message is generated when a write file that has not been closed is being opened for reading.

61: FILE NOT OPEN
This message is.generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.

62: FILE NOT FOUND
The requested file does not exist on the indicated drive.

63: FILE EXISTS
The file name of the file being created already exists on the diskette.

64: FILE TYPE MISMATCH
The file type does not match the file type in the directory entry for the requested file.

65: NO BLOCK
This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use.

66: ILLEGAL TRACK AND SECTOR
The DOS has attempted to access a track or sector which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.

67: ILLEGAL SYSTEM T OR S
This special error message indicates an illegal system track or sector.

70: NO CHANNEL (available)
The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.

71: DIR ERROR
The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem, reinitialize the diskette to restore the BAM in memory. Some active files may be terminated by the corrective action. NOTE: BAM = Block Availability Map

72: DISK FULL
Either the blocks on the diskette are used or the directory is at its limit of

144 entries.

73: CBM DOS V2.6 V170
The DOS version of the VIC-1540 is 2.6. DOS 2.6 and 1.0 (CBM2040/3040) are read compatible but not write compatible, that is, the disk formatted on DOS1.0 may be read but can not be written upon with VIC-1540, and the disk formatted on DOS2.6 may be read but can not be written upon with DOS1.0. DOS2.6 and 2.0 (CBM4040) are read and write compatible, that is, the disk formatted on either version may be read and written upon with the other version. DOS2.6 and 2.5 (CBM8050) are not read or write compatible, that is, the disk formatted on either version can not be read or written upon with the other version. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. This message may also appear after power up.

74: DRIVE NOT READY
An attempt has been made to access the VIC-1540 Single Drive Floppy Disk without any diskette present in the drive.

PATTERN MATCHING

Pattern matching of file names is available on all Commodore floppys. Pattern matching uses the question mark (?) and the asterisk (*) to perform operations on several files with similar names.

The asterisk is used at the end of a string of characters to indicate that the rest of the name is insignificant. For example:

```
        FIL*        could refer to files named
        FIL
    or  FILE1
    or  FILEDATA
    or  FILLER
```

or any other file name starting with the letters FIL.

The question mark may be used anywhere within the string of characters to indicate that the character in that particular position should be disregarded. For example:

```
       ?????.SRC     could refer to files named
       TSTER.SRC
or     DIAGN.SRC
or     PROGR.SRC
```

but not SRC.FILES

Both the characters and the position of the characters are significant.

The question mark and asterisk may be combined in many ways:

```
       *J??????
```

does not make sense because the question marks are in an area which is insignificant (because of the asterisk).

```
       P???FIL*      will access files with the names
       PET FILE
or     PRG FILE-32
or     POKEFILES$$
```

or any other files starting with P and having FIL in positions 5-7.

SCRATCH with pattern matching should be used carefully, since multiple files will be scratched. LOAD will load the first file which fits the pattern matching. OPEN with pattern matching may be used to open an existing file, in which case the first existing file encountered which fits the description will be opened. However, OPEN should not be used with pattern matching when creating a new file. Never use RENAME, SAVE, or COPY for pattern matching since an error condition will result, if attempted.

NOTES

APPENDIX

The following is a list of the programs contained on the TEST/DEMO diskette.

```
0 [illegible]
4     "DIR"                 PRG
6     "VIEW BAM"            PRG
14    "DISPLAY T&S"         PRG
4     "CHECK DISK"          PRG
9     "PERFORMANCE TEST"    PRG
5     "SEQUENTIAL FILE"     PRG
13    "RANDOM FILE"         PRG
609 BLOCKS FREE.
```

DIR simplifies the commands for the display of the disk directory or the disk operation.

VIEW BAM will read the BAM on VIC-1540 diskette and display on the VIC-1001 personal computer. It is a matrix type map with the sector numbers on the vertical axis and the track number on the horizontal axis. The empty blocks are displayed brightly and the used blocks are displayed dark. Because of the formatting of VIC-1540, sector 19 on is not persent on tracks 18 to 24.The VIEW BAM program will show these blocks as allocated.

DISPLAY T&S will show the users any block of information from a diskette (including the directory and BAM) to the screen or printer in both ASCII and hexidecimal. (Refer to Chapter 5.)

CHECK DISK will perform a validate on a disk then identify blocks that can not be read by the disk. This is useful for detecting a damaged disk.

PERFORMANCE TEST performs a check out on the performance of VIC-1540.

SEQUENTIAL FILE is an example of how to write and read a sequential file data. (Refer to Chapter 6.)

RANDOM FILE is an example of direct access file handling using BLOCK and USER commands. (Refer to Chapter 7.)

1. DIR

```
4 OPEN2,8,15
5 PRINT"■■":GOTO 10000
10 OPEN1,8,0,"$0"
20 GET#1,A$,B$
30 GET#1,A$,B$
40 GET#1,A$,B$
50 C=0
60 IF A$<>"" THEN C=ASC(A$)
70 IF B$<>"" THEN C=C+ASC(B$)*256
80 PRINT"■■"MID$(STR$(C),2);TAB(3);"■■";
90 GET#1,B$:IF ST<>0 THEN 1000
100 IF B$<>CHR$(34) THEN 90
110 GET#1,B$:IF B$<>CHR$(34)THEN PRINTB$;:GOTO110
120 GET#1,B$:IF B$=CHR$(32) THEN 120
130 PRINT TAB(18);:C$=""
140 C$=C$+B$:GET#1,B$:IF B$<>""  THEN 140
150 PRINT"■■"LEFT$(C$,3)
160 GET T$:IF T$<>"" THEN GOSUB 2000
170 IF ST=0 THEN 30
1000 PRINT" BLOCKS FREE■"
1010 CLOSE1:GOTO 10000
2000 IF T$="Q" THEN CLOSE1:END
2010 GET T$:IF T$="" THEN 2000
2020 RETURN
4000 REM DISK COMMAND
4010 C$="":PRINT">";
4011 GETB$:IFB$="" THEN4011
4012 PRINTB$;:IF B$=CHR$(13) THEN 4020
4013 C$=C$+B$:GOTO 4011
4020 PRINT#2,C$
5000 PRINT"■";
5010 GET#2,A$:PRINTA$;:IF A$<>CHR$(13)GOTO5010
5020 PRINT"■"
10000 PRINT "D-DIRECTORY"
10010 PRINT ">-DISK COMMAND"
10020 PRINT "Q-QUIT PROGRAM"
10030 PRINT "S-DISK STATUS "
10100 GETA$:IFA$=""THEN10100
10200 IF A$="D" THEN 10
10300 IF A$="." OR A$=">" OR A$=">" THEN 4000
10310 IF A$="Q" THEN END
10320 IF A$="S" THEN 5000
10999 GOTO 10100
```

2. VIEW BAM

```
100 REM ****************************
101 REM *  VIEW BAM FOR VIC-1540   *
102 REM ****************************
105 OPEN15,8,15
110 PRINT#15,"I0":NU$="N/A N/A N/A N/A N/A":Z4=1
120 OPEN2,8,2,"#"
130 Y$="■■■■■■■■■■■■■■■■■■■■■■■■■"
140 X$="■■■■■■■■■■■■■■■■■■■■■■■■■"
150 DEF FNS(Z) = 2↑(S-INT(S/8)*8) AND (SB(INT(S/8)))
```

```
160 PRINT#15,"U1:";2;0;18;0
170 PRINT#15,"B-P";2;1
180 PRINT"♥";
190 Y=22:X=1:GOSUB430
200 FORI=0TO20:PRINT:PRINT"▐▐"RIGHT$(STR$(I)+" ",3);:NEXT
210 GET#2,A$
220 GET#2,A$
230 GET#2,A$
240 TS=0
250 FORT=1TO17:GOSUB450
260 Y=22:X=T+4:GOSUB430:GOSUB540:NEXT
270 FORI=1TO2000:NEXT:PRINT"♥"
280 Y=22:X=1:GOSUB430
290 FORI=0TO20:PRINT:PRINT"▐▐"RIGHT$(STR$(I)+" ",3);:NEXT
300 FORT=18TO35
310 GOSUB450
320 Y=22:X=T-13:GOSUB430:GOSUB540:NEXT
330 FORI=1TO1000:NEXT
340 PRINT"♥QQQQQ"
350 PRINT#15,"B-P";2;144
360 N$="":FORI=1TO20:GET#2,A$:N$=N$+A$:NEXT
370 PRINT" "N$" "TS-17;"BLOCKS FREE"
380 FORI=1TO4000:NEXT
390 PRINT"♥"
400 INPUT"QQQ■■ANOTHER DISKETTE N■■■■";A$
410 IFA$="Y"THENRUN
420 IFA$<>"Y"THENEND
430 PRINTLEFT$(Y$,Y)LEFT$(X$,X)"■";
440 RETURN
450 GET#2,SC$:SC=ASC(RIGHT$(CHR$(0)+SC$,1))
460 TS=TS+SC
470 GET#2,A$:IFA$=""THENA$=CHR$(0)
480 SB(0)=ASC(A$)
490 GET#2,A$:IFA$=""THENA$=CHR$(0)
500 SB(1)=ASC(A$)
510 GET#2,A$:IFA$=""THENA$=CHR$(0)
520 SB(2)=ASC(A$)
530 RETURN
540 PRINT"Q■"RIGHT$(STR$(T),1);"■▐▐";
550 REM  PRINTT"    "SC"  "SB(0)" "SB(1)" "SB(2)=CHR$(0)
560 IFT>24ANDS=18THEN:PRINTMID$(NU$,Z4,1);:GOTO660
570 FORS=0TO20
580 IFT<18THEN620
590 IFT>30ANDS=17THEN:PRINTMID$(NU$,Z4,1);:GOTO660
600 IFT>24ANDS=18THEN:PRINTMID$(NU$,Z4,1);:GOTO660
610 IFT>24ANDS=19THENPRINTMID$(NU$,Z4,1);:GOTO660
620 IFT>17ANDS=20THENPRINTMID$(NU$,Z4,1);:Z4=Z4+1:GOTO660
630 PRINT"■";
640 IF FNS(S)=0 THEN PRINT"+";:GOTO660
650 PRINT"■+";:REMRIGHT$(STR$(S),1);Z4,1);:GOTO72
660 PRINT"▐■■";
670 NEXT
680 RETURN
```

3. DISPLAY T & S

```
100 REM******************************
110 REM* DISPLAY ANY TRACK $ SECTOR *
120 REM* ON THE VIC TO THE SCREEN   *
130 REM* OR THE VIC PRINTER         *
140 REM******************************
150 PRINT"▒▒▒─────────────────────";
160 PRINT"DISPLAY BLOCK CONTENTS";
165 PRINT"─────────────────────":
170 REM******************************
180 REM* SET PROGRAM CONSTANT       *
190 REM******************************
200 SP$=" ":NL$=CHR$(0):HX$="0123456789ABCDEF"
210 FS$="":FORI=64 TO 95:FS$=FS$+"▒"+CHR$(I)+"■":NEXT I
220 SS$="  ":FOR I=192 TO 223:SS$=SS$+"▒"+CHR$(I)+"■":NEXT I
240 DIM A$(15),NB(2)
251 D$="0"
253 PRINT"          ▒S■CREEN▒■■■■■■■■OR ▒■P■RINTER"
254 GETJJ$:IF JJ$="" THEN254
255 IF JJ$="S"THENPRINT"        ▒▒▒SCREEN■▒"
256 IF JJ$="P"THENPRINT"        ▒▒PRINTER■▒"
260 OPEN15,8,15,"I"+D$:GOSUB 650
265 OPEN4,4,7
270 OPEN 2,8,2,"#":GOSUB 650
280 REM******************************
290 REM* LOAD TRACK AND SECTOR      *
300 REM* INTO DISK BUFFER           *
310 REM******************************
320 INPUT"▒▒▒TRACK, SECTOR";T,S
330 IF T=0 OR T>35 THEN PRINT#15,"I"D$:CLOSE2:CLOSE4:CLOSE15:PRINT"END":EN
340 IF JJ$="S" THEN PRINT"▒▒▒TRACK"T" SECTOR"S"▒"
341 IF JJ$="P" THEN PRINT#4:PRINT#4,"TRACK"T" SECTOR"S:PRINT#4
350 PRINT#15,"U1:2,"D$;T;S:GOSUB650
360 REM******************************
370 REM* READ BYTE 0 OF DISK BUFFER *
390 REM******************************
400 PRINT#15,"B-P:2,1"
410 PRINT#15,"M-R"CHR$(0)CHR$(6)
420 GET#15,A$(0):IFA$(0)=""THENA$(0)=NL$
428 IF JJ$="S"THEN430
430 IF JJ$="P"THEN460
431 REM******************************
432 REM* READ & CRT DISPLAY         *
433 REM* REST OF THE DISK BUFFER    *
434 REM******************************
436 K=1:NB(1)=ASC(A$(0))
438 FOR J=0 TO 63:IF J=32 THEN GOSUB 710:IF Z$="N"THEN J=80:GOTO 458
440 FOR I=K TO 3
442 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
444 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
446 NEXT I:K=0
448 A$="":B$=":":N=J*4:GOSUB 790:A$=A$+":"
450 FOR I=0 TO 3:N=ASC(A$(I)):GOSUB 790
452 C$=A$(I):GOSUB 850:B$=B$+C$
454 NEXT I:IF JJ$="S" THEN PRINTA$B$
458 NEXT J:GOTO571
```

```
460 REM********************************
462 REM* READ & PRINTER DISPLAY       *
464 REM********************************
466 K=1:NB(1)=ASC(A$(0))
468 FOR J=0 TO 15
470 FOR I=K TO 15
472 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
474 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
476 NEXT I:K=0
478 A$="":B$=":":N=J*16:GOSUB 790:A$=A$+":"
480 FOR I=0 TO 15:N=ASC(A$(I)):GOSUB 790:IF Z$="N"THEN J=40:GOTO 571
482 C$=A$(I):GOSUB 850:B$=B$+C$
484 NEXT I
486 IF JJ$="P" THEN PRINT#4,A$B$
488 NEXT J:GOTO571
571 REM********************************
572 REM* NEXT TRACK AND SECTOR        *
573 REM********************************
575 PRINT"NEXT TRACK AND SECTOR"NB(1)NB(2) "▚"
580 PRINT"DO YOU WANT NEXT TRACK AND SECTOR"
590 GET Z$:IF Z$="" THEN590
600 IF Z$="Y" THEN T=NB(1):S=NB(2):GOTO330
610 IF Z$="N" THEN 320
620 GOTO 590
630 REM********************************
640 REM* SUBROUTINES                  *
650 REM********************************
660 REM* ERROR ROUTINE                *
670 REM********************************
680 INPUT#15,EN,EM$,ET,ES:IF EN=0 THEN RETURN
690 PRINT"▩DISK ERROR▀"EN,EM$,ET,ES
700 END
710 REM********************************
720 REM* SCREEN CONTINUE MSSG         *
730 REM********************************
740 PRINT"▚▶▶▶▶▐CONTINUE(Y/N)▩"
750 GETZ$:IF Z$="" THEN 750
760 IF Z$="N" THEN RETURN
770 IF Z$<>"Y" THEN 750
780 PRINT"▜TRACK" T " SECTOR"S "▜":RETURN
790 REM********************************
800 REM* DISK BYTE TO HEX PRINT       *
810 REM********************************
820 A1=INT(N/16):A$=A$+MID$(HX$,A1+1,1)
830 A2=INT(N-16*A1):A$=A$+MID$(HX$,A2+1,1)
840 A$=A$+SP$:RETURN
850 REM********************************
860 REM* DISK BYTE TO ASC DISPLAY     *
870 REM* CHARACTER                    *
880 REM********************************
890 IF ASC(C$)<32 THEN C$=" ":RETURN
910 IF ASC(C$)<128 OR ASC(C$)>159 THEN RETURN
920 C$=MID$(SS$,3*(ASC(C$)-127),3):RETURN
```

4. CHECK DISK

```
1 REM CHECK DISK -- VER 1.4
2 DN=8:REM FLOPPY DEVICE NUMBER
5 DIMT(100):DIMS(100):REM BAD TRACK, SECTOR ARRAY
9 PRINT"▮▮▮▮────────────────────────";
10 PRINT"  CHECK DISK PROGRAM"
12 PRINT"──────────────────────"
20 D$="0"
30 OPEN15,DN,15
35 PRINT#15,"V"D$
45 N%=RND(TI)*255
50 A$="":FORI=1TO255:A$=A$+CHR$(255AND(I+N%)):NEXT
60 GOSUB900
70 OPEN2,DN,2,"#"
80 PRINT:PRINT#2,A$;
85 T=1:S=0
90 PRINT#15,"B-A:"D$;T;S
100 INPUT#15,EN,EM$,ET,ES
110 IFEN=0THEN130
115 IFET=0THEN200:REM END
120 PRINT#15,"B-A:"D$;ET;ES:T=ET:S=ES
130 PRINT#15,"U2:2,"D$;T;S
134 NB=NB+1:PRINT" CHECKED  BLOCKS"NB
135 PRINT" TRACK     ▮▮▮▮▮"T;" SECTOR     ▮▮▮▮▮"S"▮▮"
140 INPUT#15,EN,EM$,ES,ET
150 IF EN=0THEN85
160 T(J)=T:S(J)=S:J=J+1
165 PRINT"▮▮▮BAD BLOCK:▮▮▮",T;S"▮"
170 GOTO85
200 PRINT#15,"I"D$
210 GOSUB900
212 CLOSE2
215 IFJ=0THENPRINT"▮▮▮▮▮▮NO BAD BLOCKS!":END
217 OPEN2,DN,2,"#"
218 PRINT"▮▮▮BAD BLOCKS","TRACK","SECTOR▮"
220 FORI=0TOJ-1
230 PRINT#15,"B-A:";D$,T(I);S(I)
240 PRINT,,T(I),S(I)
250 NEXT
260 PRINT"▮"J"BAD BLOCKS HAVE BEEN ALLOCATED"
270 CLOSE2:END
900 INPUT#15,EN,EM$,ET,ES
910 IF EN=0 THEN RETURN
920 PRINT"▮▮▮ERROR #"EN,EM$;ET;ES"▮"
930 PRINT#15,"I"D$
```

5. PERFORMANCE TEST

```
1000 REM  PERFORMANCE TEST  1.1
1010 :
1020 REM  VIC-1540 SINGLE FLOPPY DISK DRIVE
1030 :
1040 :
1050 OPEN 1,8,15:OPEN15,8,15
1060 LT=35
1070 LT$=STR$(LT)
```

```
1080 NT=30
1090 PRINT"⌐▯▯────────────────────────"
1100 PRINT"⌐   PERFORMANCE TEST"
1110 PRINT"────────────────────────────"
1120 PRINT
1130 PRINT"  ▪INSERT SCRATCH"
1140 PRINT
1150 PRINT"    DISKETTE IN DRIVE"
1160 PRINT
1170 PRINT"▯   PRESS ▪RETURN▪"
1180 PRINT
1190 PRINT"            WHEN READY▯"
1200 FOR I=0 TO 50:GET A$:NEXT
1210 GET A$:IF A$<>CHR$(13) THEN 1210
1220 :
1230 :
1240 TI$="000000"
1250 TT=18
1260 PRINT#1,"N0:TEST DISK,00"
1270 C1$="▪i   DISK NEW COMMAND   "
1280 C2$="▪▯ WAIT ABOUT 80 SECONDS"
1290 CC$=C1$+C2$:GOSUB 1840
1300 IF TI<NTTHEN1370
1310 PRINT"▯SYSTEM IS"
1320 PRINT"▯         NOT RESPONDING"
1330 PRINT" CORRECTLY TO COMMANDS"
1340 GOSUB 1880
1350 :
1360 :
1370 PRINT"▯DRIVE PASS"
1380 PRINT"        MECHANICAL TEST▯"
1390 TT=21
1400 OPEN 2,8,2,"0:TEST FILE,S,W"
1410 CC$="OPEN WRITE FILE"        :GOSUB 1840
1420 CH=2:CC$="WRITE DATA"        :GOSUB 1930
1430 CC$="CLOSE "+CC$             :GOSUB 1840
1440 OPEN 2,8,2,"0:TEST FILE,S,R"
1450 CC$="OPEN READ FILE"         :GOSUB 1840
1460 CH=2                         :GOSUB 1990
1470 PRINT#1,"S0:TEST FILE"
1480 CC$="SCRATCH FILE▯":TT=1     :GOSUB 1840
1490 :
1500 :
1510 TT=21
1520 OPEN 4,8,4,"#"
1530 NN%=(1+RND(TI)*254+NN%)AND255:PRINT#1,"B-P";4;NN%
1540 NN$="":FOR I=1 TO 255:NN$=NN$+CHR$(I):NEXT
1550 PRINT# 4,NN$;
1560 PRINT# 1,"U2:";4;0;LT;0
1570 CC$="WRITE TRACK"+LT$:GOSUB 1840
1580 PRINT#1,"U2:";4;0;1;0
1590 CC$="WRITE TRACK 1"          :GOSUB 1840
1600 PRINT#1,"U1:";4;0;LT;0
1610 CC$="READ TRACK"+LT$         :GOSUB 1840
1620 PRINT#1,"U1:";4;0;1;0
1630 CC$="READ TRACK 1"           :GOSUB 1840
1640 CLOSE 4
1650 :
1660 :
```

```
1670 PRINT"ЖЖ UNIT HAS PASSED"
1680 PRINT"      PERFORMANCE TEST!"
1690 PRINT"Ж PULL DISKETTE FROM"
1700 PRINT"Ж  DRIVE BEFORE TURNING"
1710 PRINT"   POWER OFF."
1720 END
1730 :
1740 :
1750 PRINT"    ЖЖCONTINUE (Y/N)?█";
1760 FOR I=0 TO 50:GET A$:NEXT
1770 GET A$:IF A$="" THEN 1770
1780 PRINT A$"Ж"
1790 IF A$="N" THEN END
1800 IF A$="Y" THEN RETURN
1810 GOTO 1760
1820 :
1830 :
1840 PRINT CC$
1850 INPUT# 1,EN,EM$,ET,ES
1860 PRINTTAB(12)"█"EN;EM$;ET;ES;"█"
1870 IF EN<2 THEN RETURN
1880 PRINT"Ж UNIT IS FAILING"
1890 PRINT"Ж   PERFORMANCE TEST"
1900 TM$=TI$:GOSUB 1750:TI$=TM$:RETURN
1910 :
1920 :
1930 PRINT"WRITING DATA"
1940 FOR I=1000 TO 2000:PRINT#CH,I:NEXT
1950 GOSUB1850
1960 CLOSE CH:RETURN
1970 :
1980 :
1990 PRINT"READING DATA"
2000 GETA$
2010 FOR I=1000 TO 2000
2020 INPUT# CH,J
2030 IF J<>I THEN PRINT"█READ ERROR:█":GOSUB 1850
2040 NEXT
2050 GOSUB 1850
2060 CLOSE CH:RETURN
```

——ERRATA——

	Wrong	Correct
P 12 ℓ16	TYPE : OPEN1,8,15,"10"	TYPE : OPEN1,8,15,"I0"
P56, P60	5340 IFF>357ANDF<471 5360 IFF>580THENF1=580	5340 IFF>357ANDF<472 5360 IFF>579THENF1=579
P57 table10	25to30 0to17 18 478−579 31to35 0to16 17 586−664	25to30 0to17 18 472−579 31to35 0to16 17 580−664
P69	DISPLAY T&S RANDOM FILE	These programs require an expander RAM.
P72	410 PRINT#15,"M−R" CHR$(0) CHR$(6)	410 PRINT#15,"M−R" CHR$(0) CHR$(5)

——CAUTION——

Attempting to LOAD Disk Progranrs that exceed your VIC's memory area will overflow into the screen area. To avoid this problem you should add sufficient memory to your VIC with an expander RAM cartridge.

1982−06−02

VIC-1540