DISKEY

For the TRS-80 Color Computer

Written by
David D. McLeod

A Subsidiary of Scott Adams, Inc.
P.O.Box 3435
Longwood, FL 32750
(305) 862-6917

DISKEY

DISK ACCESS AND REPAIR KEY

CONTENTS

Introduction 3
Program Operation 4

DISKEY Functions
(1) DISK MAP 5
(2) EXAMINE/EDIT DISK 6
Graphic Display Mode 6
Hexadecimal Display Mode 7
Edit Mode 8
Editing Control Switches 8
(3) ERASE DISK 9
(4) BACKUP UTILITY 10
(5) COPY TO NEW DISK 10
(6) RECONSTRUCT DIRECTORY 10
(7) PRINT DIRECTORY 12
(8) CALIBRATE DRIVE 13
Disk Drive Diagram 14
Care & Maintenance 15

The Color Computer Disk System ..... Appendix A
General 19
The Directory 21
Directory Layout Diagram 21
Recovering KILLed Files 24

Notes on File Structure ..... Appendix B
Binary BASIC Programs 26
Machine Language Programs 28
ASCII BASIC Programs 28
Data Files 29

Charts & Tables ..... Appendix C
Directory Reconstruction Charts 30
BASIC Tokens 33
CHR$/POKE Comparison 35
Hex To Decimal Conversion 36

Sample Run Through ..... Appendix D

Sample Printouts ..... Appendix E
Track/Sector Printout 44
Directory Printout 45

DIAGNOSTICS

(1) ROM Test 46
(2) RAM Test 46
(3) KEYBOARD Test 46
(4) JOYSTICK Test 47
(5) PRINTER Test 47
(6) CASSETTE Test 47
(7) DISK Test 48
(8) VIDEO Test 49
(9) SOUND Test 50

INTRODUCTION

Anyone who owns a disk system for the Color Computer will, sooner or later, experience that horrible feeling of anguish that always accompanies the discovery of an inexplicably unreadable sector. Murphy's 1,294th law states that this tragedy will likely befall you just before you've had a chance to generate a listing, and certainly before you've made a backup!

How many programmers will skip the backup until they've filled at least half the disk? And how many more will delay getting a listing, in the interest of saving paper, until they are satisfied that the program is (almost) fully debugged. If my own programming habits are any indication, there are probably quite a few of us around!

Those who fall into either category will almost certainly require this program at one time or another. But, even if you don't really need this program to resurrect a crashed disk, I think you'll find that it provides a sound basis for learning how your disk system operates.

DISKEY, written entirely in machine language, was designed primarily to assist you in recovering data from a "slipped disk" and for recovering KILLed files. But it can be used to examine, modify, or copy any disk at all, just as long as there is at least one readable sector somewhere. I think you'll find DISKEY to be a powerful and helpful utility to add to your software library.

DIAGNOSTICS

Often, apparent software errors are really caused by hardware problems. CCDIAG tests all the major functions of your system, including ROMS, RAM, the keyboard, joysticks, printer, cassette and disk drive, video and sound. Appropriate messages are constantly displayed to make the program very easy to use. The program assumes nothing, except that your ROMs are BASIC 1.1, Extended BASIC 1.0, and Disk BASIC 1.0.

Not only does CCDIAG test your system, but it also incorporates interactive graphic and sound tests for experimentation purposes, making it all the more versatile and fun to use.

Minimum system requirements are 16K of RAM and at least one disk drive. Some functions (the BACKUP utility and any COPY involving more than one track at a time) will require at least 32K of RAM. Some features of each of the programs require additional equipment: cassette player, joysticks, and/or 80-column printer.

PROGRAM OPERATION

Before proceeding any further, prepare one or two backups of your Master disk, following the instructions in your **Disk System Owner's Manual**, and put your Master disk away for safe keeping. Obtain a couple of blank, formatted disks to experiment with, and put a copy of your program disk in Drive #0.

When you take a directory, you will notice that only one file is displayed: "MASTER BAS 0 B 1". This is a deliberate lie! In fact, there are several files on your disk. Appendix D "Sample Run Through" will explain how you can find and make use of them.

To get the program in operation, simply type: RUN "MASTER" ENTER, and very shortly the master menu will appear. From this Master Menu, select the program of your choice by pressing the appropriate number key. If the program you have selected is not already in memory, you will be asked to put the program disk in Drive #0. When you press ENTER, that program will automatically load and begin execution.

Both DISKEY and CCDIAG have their own menus for option selection. When you have finished using either of these programs, you must return to the Master Menu before you can return to BASIC. When you do return to BASIC, you will notice that Drive #0 will be restored to Track #0. This makes it easier for the system to locate the directory Track for whatever you wish to do next. It is recommended that you thoroughly familiarize yourself with this manual before using either of the programs.

DISKEY FUNCTIONS

DISK MAP

This unique feature gives you a graphic display of an entire disk by producing a rectangle of 35 tracks (rows) by 18 sectors (columns). Each individual track/sector intersection is color-coded as to its readability —

- a yellow block indicates a **good sector;**
- a blue block indicates a **read** or **CRC** (Cyclic Redundancy Check) or **lost data error;**
- a green block indicates a **write fault;**
- a red block (same as background color) indicates a **drive-not-ready error** or other **input/output error.**

To display a disk map, simply press **1** at the main menu. Respond to the prompt for Drive number, and press **ENTER** when the disk is in the appropriate drive and the door is closed.

The red graphics screen will appear, with the Drive number at the top, track numbers down the left side, and a color-coding legend on the right. The track numbers are organized into 7 groups of 5 tracks each, with only the start and end track numbers of each group (00-04, 05-09, etc.) displayed on the screen. For further clarity, a short green line and a short blue line are placed at the start and end point of each 5-track group.

As the drive begins to read the disk, sector blocks will appear on the screen, starting with Sector #1 on the left side and continuing to Sector #18 on the right side. This process will continue for each track until all tracks have been read.

Occasionally, the drive may encounter a sector which is unreadable. When this happens, the drive will attempt to read the sector a maximum of five times. If it is totally unsuccessful, the display will indicate some type of error, usually a blue (read error) block. The drive will continue to read subsequent sectors as before.

In some (unusual) instances, the drive may detect an error and determine it to be a "drive-not-ready" error, in which case the display will stop altogether, and the cursor will appear in the lower right corner of the screen. When this happens, you can attempt to read past the faulty sector by pressing **C** (continue).

Under most circumstances, the map will be generated uninterrupted and, for a good disk, will produce a 35 by 18 rectangle of yellow blocks. You can abort the display and return to the main menu at any time by pressing

BREAK. Once the display is complete, the cursor will appear in the lower right corner. If you wish to produce another map, you can change disks and press Restart.

Disk Map Key Summary	
BREAK	return to main menu (active at all times)
RESTART	clear screen and produce a new map (active whenever cursor appears)
CONTINUE	ignore faulty sector and continue to read subsequent sectors (active when cursor appears AND map is incomplete)

EXAMINE/EDIT DISK

This is the work-horse of DISKEY. All data from each sector is read into 2 separate buffers: the screen, for display and modification; and a secondary "unedit" buffer. Press 2 at the main menu and respond to the prompts. When the disk is in place and the door is closed, press ENTER.

A) Graphic Display Mode

Shortly, the screen will display an information line (e.g., DRIVE 0/TRACK 17/SECTOR 01), a 256-byte graphic block of 8 lines with 32 characters each (this block is a representation of the sector being read), and a flashing prompt line at the bottom, containing a list of valid keys, as follows:

A	Read the same sector Again; good for changing disks to compare data.
D	Change Drives; you will be prompted for both Drive and Track numbers.
E	Select Edit mode; this will be discussed separately.
H	Select Hexadecimal display mode; this will be discussed separately.
P	Send the current Track/Sector data from the "unedit" buffer to the Printer (see the appendices for a sample printout.)
Q	Quit; return to the main menu.

W	Write the current contents of the display buffer to disk; you will be prompted with "ARE YOU SURE (Y.N)?"; respond accordingly. If you answer Y, the display data will be written to the Track/Sector specified in the information line. (NOTE: if, having written to the disk, you discover you have made an error, and you have NOT YET READ a different sector, you can recover from the error by entering the EDIT mode, performing an UNEDIT, returning to the Graphic Display Mode, and re-writing to the SAME Track/Sector. See EDIT MODE for more details.)
←	Read previous sector.
→	Read next sector.
↓	Read same sector on next track.
↑	Read same sector on previous track.

B) Hexadecimal Display Mode

For every graphic character that appears on the screen under Graphic Display Mode, two hexadecimal digits must be used to represent that character. In other words, 256 graphic bytes (one complete sector) of data will require 512 hex digits for an equivalent display. Since this leaves no room on the screen for additional information, the hex display is divided into two "half-pages" of data. This display uses the contrast of normal versus inverse video to highlight pairs of hex digits. The information line at the top of the screen remains the same as for Graphic Display Mode, but the flashing prompt line is slightly different.

A	Read same sector Again.
D	Change Drives; you will be prompted for Drive and Track number, as before; NOTE that Graphic Display Mode will be automatically re-selected.
F	Flip pages; upon initial selection of Hex Display Mode, the 1st half-page is selected. Pressing F will the cause the display to shift back and forth between the 1st and 2nd half-page.
G	Select Graphic Display Mode.
P	Send data from the "unedit" buffer to the Printer.
Q	Quit; return to main menu.
←	Read previous sector.
→	Read next sector.
↓	Read same sector on next track.
↑	Read same sector on previous track.

Note that you cannot write to disk or select Edit Mode from Hex Display Mode. You must return to Graphic Display Mode first.

C) Edit Mode

The Edit Mode allows you to modify data within the graphics block on the screen, in preparation for writing to the disk later. The information line remains at the top of the screen. Slightly below, and on the left side of the screen, the hex value of the byte under the cursor will appear. This number will change as you move the cursor around. The cursor itself will initially appear at the upper left corner of the graphic block. The nonflashing prompt line at the bottom of the screen is quite different from the previous prompt lines.

D	Delete the character under the cursor; auto-repeat is in effect.
I	Move the character under the cursor (and all other characters to the right and below) one space to the right and Insert a blank (60 Hex); the last byte in the lower right corner of the graphic block is lost; no auto-repeat.
R	Restore the byte under the cursor to its original value; effective as long as the cursor has not been moved since editing of the current byte began.
U	Unedit entire graphic block; that is, restore entire block to its original contents; cursor will remain in its current position.
Z	Zero entire graphic block; that is, load display page with hex FF's (orange block); cursor will be homed to upper left corner.
+	(Plus key) Increment the byte under the cursor; auto-repeat is in effect; if the SHIFT key is held down simultaneously, incrementing will occur *more quickly.*
−	(Minus key) Decrement the byte under the cursor; auto-repeat is in effect; if the SHIFT key is held down simultaneously, decrementing will occur *more slowly.*
←	Move cursor left one space.
→	Move cursor right one space.
↓	Move cursor down one line.
↑	Move cursor up one line. (The arrow keys all incorporate auto-repeat, which can be slowed down for precision by pressing the SHIFT key simultaneously. As well, total wraparound is in effect.)
X	EXit to Graphic Display Mode.

Editing Control Switches

Two additional keys are available to allow you to enter data in slightly different ways: the BREAK and CLEAR keys. To engage either function,

position the cursor where you want data entry to begin, press the appropriate key, enter data until you are done, and press the SAME control key again to de-select and return to the normal Edit Mode.

BREAK	(Direct Keyboard Entry) — BASIC's familiar INPUT cursor will appear at the last Edit cursor position; simply type in what you like from the keyboard; when done, press BREAK again to de-select.
CLEAR	(Hexadecimal Entry) — BASIC's cursor will appear at the Hex data display point just above the graphic block; enter data in hexadecimal format; as each pair of digits is typed in, the corresponding graphics character will appear on the graphic block at the last Edit cursor position; when done, press CLEAR again to de-select.

For both of these control switches, automatic wraparound takes place, and the data you enter will over-write any data that was there before. Also, in the BREAK option, if you attempt to enter lowercase (inverse video characters) on the screen, they will appear quite different from what you expect. Have no fear! This is not an error — the characters on the screen are the true representations of the ASCII values you select, not the modified values that BASIC's PRINT statements send to the screen for readability. Take a look at the CHRS$-POKE table in the appendix, or, if you prefer, write your data to disk, reread it, and send the data to the printer.

NOTES: 1. If at any time, one of the aforementioned keys seems to have no effect, try SHIFT 0 first; in all likelihood, you are in lowercase mode.

2. You can use the Examine function to transfer sectors individually from one disk to another. Simply read the appropriate sector from the Source disk, place the Destination disk in the SAME drive, and press W, and answer Y to the warning prompt. If you need to transfer an entire Track, use the Copy function.

ERASE DISK

This feature works in a similar manner to the DSKINI command, by first sending a sequence of zeroed (loaded with hex FF's) blocks to each sector and then rereading the disk for verification. It will NOT format a blank disk, but it will work on any disk that has been properly initialized previously.

Press 3 at the main menu and respond to the prompts. When the disk is in place and the door is closed, hit ENTER. Immediately, the computer will begin to write data to the disk and display an appropriate message. When all 35 tracks have been written, the computer will start over and read data to

verify that everything is correct. Again, an appropriate message will be displayed.

When the process is complete, you will be returned automatically to the main menu.

BACKUP UTILITY

This function is much the same as BASIC's BACKUP command, but the activity is more clearly displayed. The routine transfers data four tracks at a time from any specified drive to any other specified drive. If the source and destination drives are the same, you will be prompted to exchange disks when necessary.

Before beginning, make sure that your source and destination disks are clearly identified. For added protection, it is recommended that you place a write-protect label on your source disk, just in case you confuse disks.

When you are all set, press 4 at the main menu, respond to the prompts for Source and Destination drive numbers, and press ENTER when ready. If you are performing a one-drive backup, you will be prompted to change disks as necessary. The computer will read data from the source disk into its buffer and display an appropriate message as it does so. Then it will write the data to the destination disk and verify that it was written correctly.

When the backup is successfully completed, you will be returned automatically to the main menu.

COPY TO NEW DISK

This routine is functionally identical to the BACKUP utility. The only difference is that you specify a Start and End Track number to COPY. All the displays and prompts will be the same as for the BACKUP utility.

The COPY utility is extremely helpful in recovering data from a flawed disk. If, for example, track 17 (the directory track) appears to be faulty, you can perform two COPY's to the same Destination disk. First, you copy tracks 0 to 16 inclusive, then you copy tracks 18 to 34 inclusive. Simple!

Since the COPY function transfers whole tracks only, you'll need a way to transfer individual sectors. This is explained in the **EXAMINE/EDIT DISK** description.

RECONSTRUCT DIRECTORY

This routine is powerful and easy to use. It does, however, require an understanding of the Disk System's method of file control (explained in

some detail in the appendix). Assuming you have positively identified the start and end point as well as the type and format of each retrievable file on your disk, you can use this routine to reconstruct a directory consisting of up to 68 unique entries.

Before selecting this option, be sure you have gone through the entire disk and located and properly documented each existing file. Use the included charts (See Appendix C) to keep track of all your data. When all your information has been transposed to Chart #3, press 6 at the main menu, and respond to the prompt for Drive number.

In the upper left corner of the screen you will see a constant reminder as to which file (by number) you are currently working on. Immediately below this, you will see one of four questions related to the file, each of which must be answered with an allowable response followed by pressing ENTER.

1. The first question asks for type of file and offers 4 options: 0 BASIC PROGRAM, 1 BASIC DATA FILE, 2 MACHINE LANGUAGE PROGRAM, 3 EDITOR SOURCE FILE.
2. The second question requests storage format, 0 BINARY or 1 ASCII.
3. Question three asks for granule numbers in CHRONOLOGICAL order — that is, the order in which they would normally be loaded into memory — NOT necessarily numerical order. Each granule number MUST be followed by pressing ENTER. When you have entered the last granule number, type 99 to signify that there are no more granules to record. Note that you MUST enter at least one granule number per file.
4. Question four requests the number of sectors in the last granule which are part of the actual file. The response will always be a number from 1 to 9.

After you have responded to all four prompts, the information will be re-displayed on the screen for your verification. If you find that some of your data has been entered in error, you can easily correct it at this point. When your data is correct, respond with Y, and you will be asked if there are any more file entries. If so, just type Y; otherwise, type N. When you have completed all the file entries, or there is no more room for additional entries, the next prompt will be: BUFFER FULL — SEND (Y.N.)? Typing N at this point will return you to the main menu; typing Y will cause the message "PRESS ENTER WHEN READY?" to appear. And when you press ENTER, the data will be sent to the disk, the new directory will be written, and you will be returned to the main menu.

If you now examine track 17, you will notice that filenames in Sectors 3 through 11 consist of a letter (A to H) and a number (1 to 9) in lexicographical order. Each left-justified, blank-filled filename also has a 3-character extension that denotes the type of file: BAS for BASIC program, DAT for BASIC data file, MLP for machine-language program, and ESF for editor source file. When you return to BASIC, you can easily RENAME these files if you wish.

Note also that, for convenience, this subroutine automatically assumes 256 bytes in the last sector of the file. You will see this in bytes 14 & 15 of the related directory entry. Using the information from your charts, you can make necessary corrections to these bytes by going into the EDIT mode and writing the new data to the disk. It should be noted, however, that these corrections are not strictly required for BASIC program files, regardless of storage format, but they are required for all other types of files. See the appendices for more details.

PRINT DIRECTORY

This simple routine, which assumes you have a standard 80-column printer, allows you to obtain a hard copy of your disk directory. Press 7 at the main menu and respond to the prompt for Drive number. You will be asked for the name of the disk, and you can enter up to 57 characters of descriptive information. When you are ready, press ENTER, and the printout will begin.

Following the header line, you will see the filenames and pertinent data in their familiar screen format. To the right of each directory entry, you will also see a data line describing the file type and storage format. The last line of the printout will let you know how much free space (in granules) is available on that disk.

See the appendix for a sample printout.

CALIBRATE DRIVE

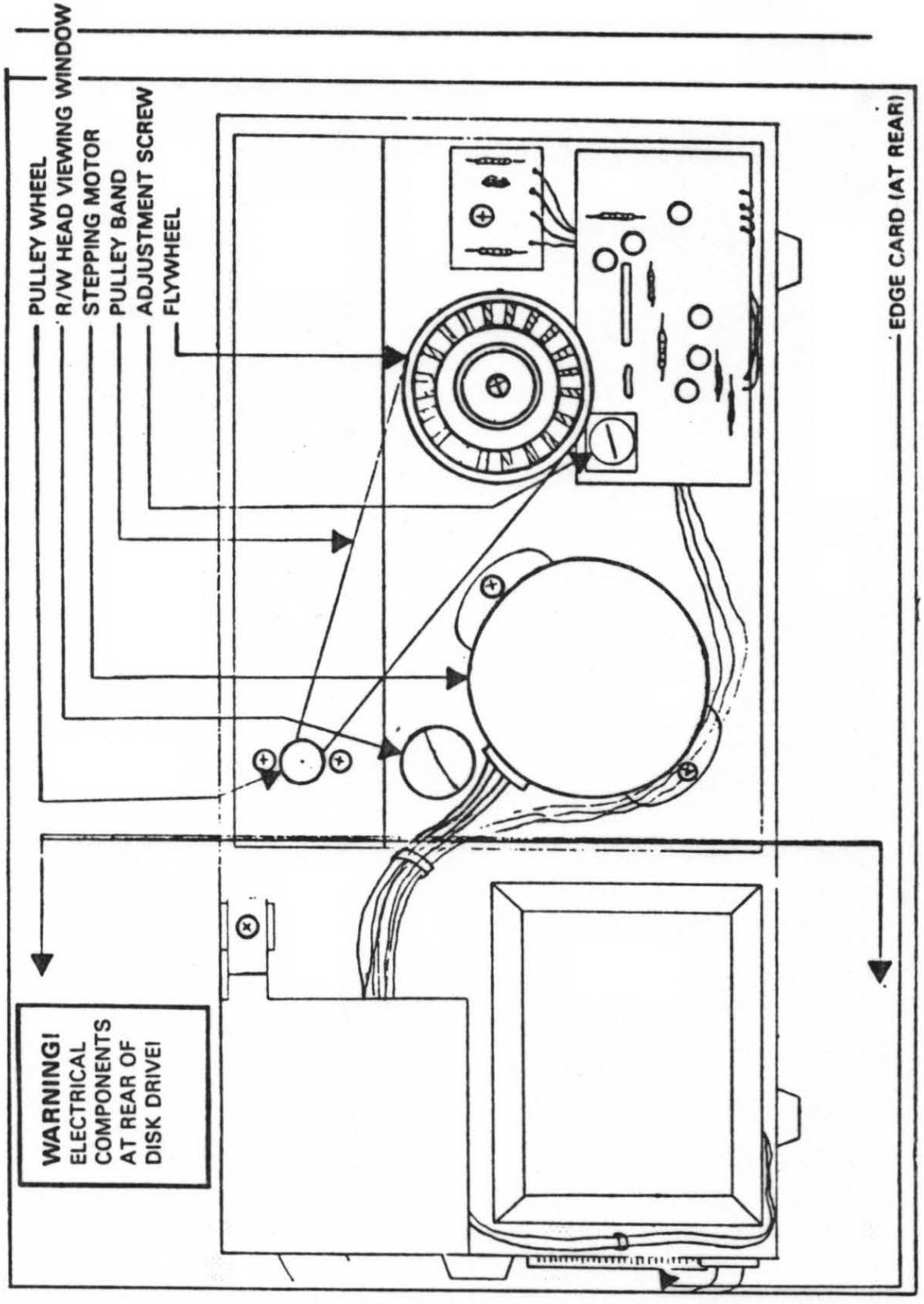
********** NOTICE **********

The following section is provided for informational purposes only, and is not intended for use by the general public. High voltage is present inside your disk drive, and contact with it could result in damage to the drive itself as well as severe injury or death.

Because of these potential hazards, we include this chapter for the convenience of qualified personnel. We recommend that the steps outlined in it be performed only by a reliable service facility.

This function simply turns on the selected disk drive so that you can perform necessary cleaning and adjustments periodically. It will not actually do the calibration for you, so the title of this section is a bit of a misnomer!

To use this function simply press 8 at the main menu and respond to the request for drive number. Immediately, all drives in your system will be turned on, whether they contain disks or not, whether the doors are closed or not. The light on the drive you have selected will illuminate (this light merely indicates that the drive in question is being addressed) and the head will be restored to track #0. All drive motors will remain on until you press BREAK to terminate the function, at which time you will be returned to the main menu. The data that follows is a discussion of disk-related problems and ways in which some of these problems can be cured.


PULLEY WHEEL
R/W HEAD VIEWING WINDOW
STEPPING MOTOR
PULLEY BAND
ADJUSTMENT SCREW
FLYWHEEL
EDGE CARD (AT REAR)
WARNING!
ELECTRICAL
COMPONENTS
AT REAR OF
DISK DRIVE!

DISKEY assumes that you own standard Radio Shack (T.E.C.) disk drives, which, when properly tuned and maintained, are very reliable. Because of the mechanical nature of your drive units, maintenance must be performed manually. Fortunately, the process is very simple, though somewhat lengthy. You will need a few pieces of equipment to do the job properly.

1. A fluorescent lamp
2. One ordinary flat-blade screwdriver approximately ¼ inch wide
3. One Philips screwdriver approximately $\frac{3}{16}$ inch wide
4. A box of Q-Tips™
5. A "Drive Head Cleaning Kit," available at your local Radio Shack Store

> NOTE: This routine requires you to remove the external cover of your disk drive unit, which may void the warranty provided by Radio Shack.

The main reason for keeping a drive unit properly tuned (aside from just taking good care of your equipment) is to avoid the dreaded ?IO ERROR. During disk access, occurrence of this error indicates that, for some reason, the computer is unable to locate a sector that is or will become part of a specified file. Isolated cases cannot be immediately attributed to a faulty drive, but it is better to be safe than sorry. (As we proceed, please refer to the accompanying diagram to get a better understanding of what is happening.)

There are eight ways in which your drive can cause you to suffer repeated ?IO ERROR's.

1. Dirty electrical contacts on the cable, disk-drive controller, computer, or drive unit
2. Incorrect or fluctuating drive rotation speed
3. Dirty read/write head
4. Slippage of the flywheel or pulley motor due to dirt on the pulley band
5. Freezing of the read/write head, usually due to dirt or poor lubrication
6. Electrical malfunction within the drive unit itself
7. Defective ROM in the disk controller cartridge
8. Misaligned read/write head

The first four of these problems are easily fixed, and probably account for a good 90% or more of all disk-related ?IO ERROR's. If you have any reason

to suspect any one of the last four causes, then take your disk drive to the nearest Radio Shack Computer Center for repair or replacement; do NOT attempt to fix these problems yourself.

Since the majority of the problems seem to involve dirt (tobacco smoke is the worst offender, and dust places a close second), we will first address ourselves to the subject of cleaning the drive. Logically, we should proceed from the outside to the inside. This means that we start with the electrical contacts on the edge cards and the connector plugs. And this, of course, means that the entire computer system must be TURNED OFF.

Disconnect the controller from the computer, and the cable from both the controller and the drive unit. Using a cotton swab dipped in high grade denatured isopropyl alcohol (the same stuff that you find in your Drive Head Cleaning Kit), scrub down each of the edge cards (2 on the controller and one on the drive unit itself) on both sides. When the cotton swab gets dirty, replace it. You can consider the contacts to be clean when the swab remains white. Follow the same procedure for the connector plugs on the cable as well as the plug inside the cartridge slot on the right side of the computer. The connector plugs are a little more difficult to clean without disassembly. If you have problems in this area, try using an old toothbrush soaked in the alcohol. Don't worry about excess alcohol — it will evaporate without leaving a trace. When you are satisfied with the cleaning job, reconnect all of the apparatus.

Before powering up the system, remove the external cover of the disk drive by removing the two screws on each side of the drive's base. (Do NOT remove any screws from the bottom of the drive — just the 4 screws around the perimeter.) When all four screws have been removed, the cover will slide upwards to reveal the mechanical and electrical "guts" of the unit. Familiarize yourself with the left side of the drive by referring to the diagram. When you are ready, power up the system in the normal way, and load in the program. (From here on, "turn the drive ON" means to select option 8 from the main menu and respond to the prompts; "turn the drive OFF" means to press BREAK and return to the menu.)

Using the shaft of a cotton swab between the pulley band and the flywheel, carefully remove the pulley band (think of it as a bicycle chain) and turn on the drive. Dip the swab in alcohol and thoroughly clean the drive motor pulley wheel where the pulley band makes normal contact. Turn off the drive and get the excess dirt off the flywheel by manually rotating it and holding a swab next to its contact surface. Replace the pulley band by starting at the pulley wheel, placing the upper strand of the pulley band on the surface of the flywheel, holding it in place with your finger, and slowly rotating the flywheel until the band slips into place.

'Now you can turn the drive unit on again and take advantage of the flywheel rotation to clean the flywheel itself as well as both surfaces of the pulley band. Take a new swab and touch it to the upper and lower surfaces of the pulley band with just enough pressure to ensure that you are, in fact, removing dirt (and nothing else!) — use as many swabs as you need until, as before, the swab remains white on contact. Follow the same procedure for cleaning the contact surface of the flywheel.

While you are doing this, there is no reason why you can't place one of your head-cleaning disks, properly soaked with alcohol, into the drive unit. Close the drive door and make sure that the motor is on. You will hear a noise similar to the sound of an electric sander — don't worry about this, it is perfectly normal. Leave the drive door closed for about 30 seconds — this should be enough to get the head completely clean — and then remove the disk and put it away for future use.

By now, your disk drive should be as clean as it's ever been, and now is the time to perform the adjustment to ensure your drive is operating at optimum speed. Place an ordinary disk in the drive and close the door. Make sure the drive is on and settled into its rotation speed. Place your fluorescent lamp in a position where the light will shine on the strobe markers around the perimeter of the flywheel, but will not be in your way.

Pay close attention to the "apparent motion" of the strobe markers (outer ring for 60 Hz operation; inner ring for 50 Hz operation.) If your drive unit is tuned to the correct speed, the markers will appear to be stationary. If the strobe markers appear to be moving in one particular direction, use your flat-blade screwdriver to slowly turn the yellow adjustment screw in the **opposite** direction until strobe movement appears to stop. This is a delicate operation, and may require several attempts before you get it just right. When all "apparent motion" has stopped, take out the disk and put in another one, just to ensure that the speed remains constant. Try several different disks until you are satisfied that you have found the right adjustment setting.

Here are a few tips that will make your disk drive (and you) breathe a little easier.

1. Keep the drive clean and well adjusted at all times; the entire process described here, which should be performed about once a month, takes only about 20-30 minutes and may save you many hours of heartache, not to mention many of your hard-earned dollars!
2. Clean the read/write head frequently, at least once a month, but more often if you use the drive a lot.

3. Avoid jarring the drive unit at any time as this may be the cause of incorrect rotation speed. When you open the drive door, don't simply press the release mechanism and let the door snap open; rather, use a spare finger to hold the door so it opens slowly.
4. Keep dust away from the drive unit by covering it when it is not in use.
5. If the drive speed seems to fluctuate no matter how much cleaning and adjusting you do, try using a separate power supply; if this doesn't solve your problem, then take the unit in for repair.
6. Treat your disks with as much care as the drive unit itself. Keep each disk in its own envelope when not in use and, if possible, store your disks in the upright position in their own file box (no more than 10 to a box).

If you observe the above procedures on a regular basis and ensure that your drive is in its best condition, you will probably discover that most of your disk-related problems will disappear. But if you find that problems persist in spite of the loving care you devote to your machine, you may have some type of defect that only the experts can fix — don't hesitate to take your machine in for repair if this is the case.

APPENDIX A

THE COLOR COMPUTER DISK SYSTEM

This section is provided for those who may find Chapter 11 of the DISKEY Manual a little confusing. Most of the information here has been obtained through experimentation and trial and error; therefore, it is neither conclusive nor exhaustive.

When you format a disk with the DSKINI command, the computer does some writing to the disk in order to organize. Once this is accomplished, the disk can be considered to contain 35 tracks, numbered 0 to 34.

Each track is further subdivided into 18 sectors (numbered 1 to 18, not 0 to 17) of 256 bytes each. At the same time, each track (except Track 17) is divided into 2 granules. Thus, each granule comprises 9 sectors. For a better grasp of the relationships between tracks, sectors, and granules, see Diagram 1.

Granules are numbered from 0 to 67 and are used by the disk controller to determine where a file is stored. Each file must occupy at least one granule; furthermore, if your file happens to occupy the equivalent of 9 sectors plus 1 byte of computer memory space (a total of 2,305 bytes), it will take up 2 full granules (18 sectors) on the disk. In general, then, the amount of disk space required by your file will be based upon the amount of memory space rounded upwards to include all of the current granule being written to.

For Track numbers below 17, the corresponding granule number can be determined from the formula:

GN = 2 * TR + INT(S/10) [GN = granule number; TR = track number; S = sector number]

For Track numbers above 17, the corresponding granule number can be determined from the formula:

GN = 2 * (TR − 1) + INT(S/10)

Considering that conditional expressions are evaluated to −1 if True and to 0 if False, these two formulae can be combined as follows (as long as TR ‹ › 17):

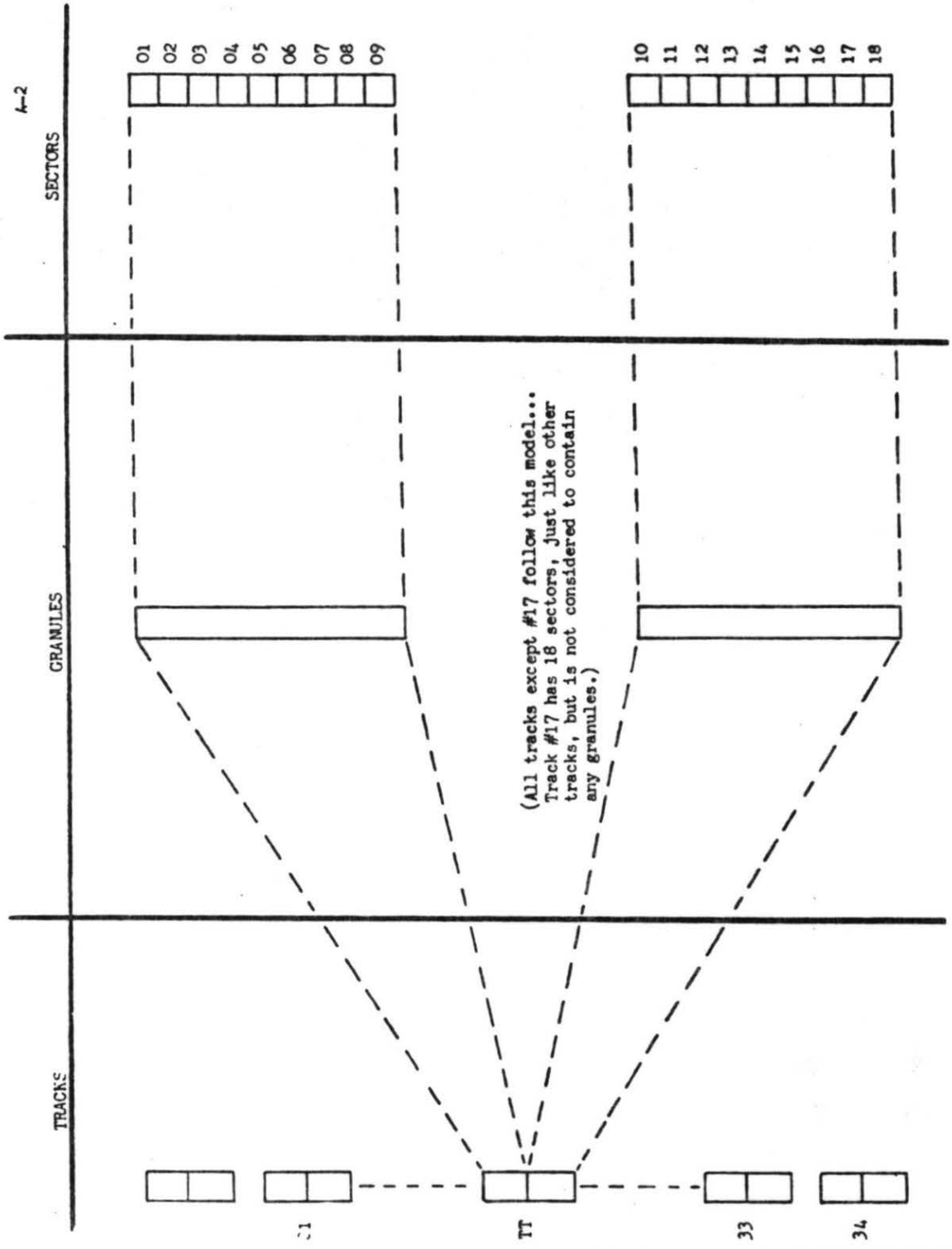
GN = 2 * (TR + (TR › 17)) + INT(S/10)

(Your computer will accept this type of formula, by the way, and return a valid result! Try it!)

Using this formula, you can see that all sectors from 10 to 18 in Track 14 will be included in Granule Number 29. Similarly, Granule Number 44 will contain sectors 1 to 9 in Track 23.

DIAGRAM #1

COMPUTER ORGANIZATION OF DISKETTE



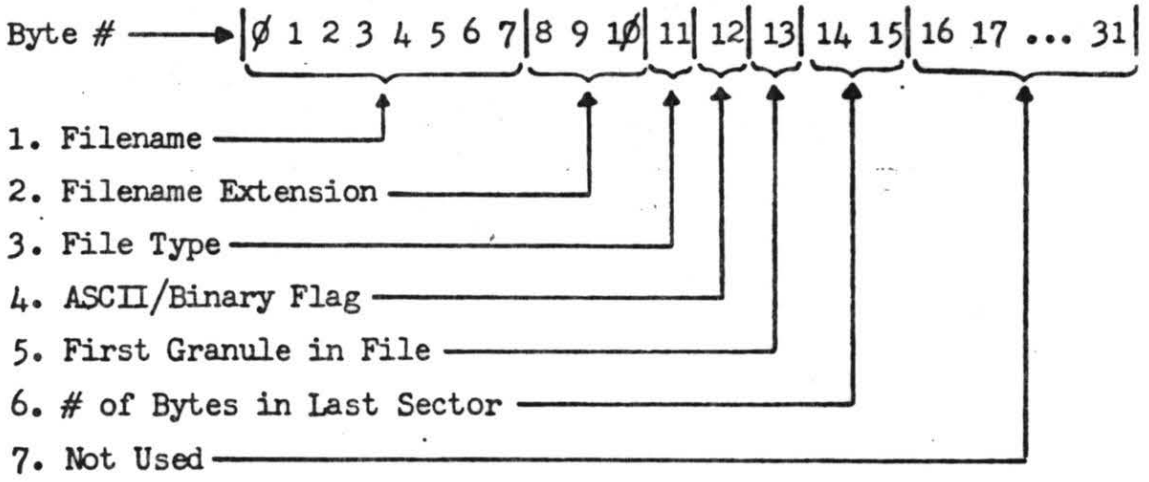
The Directory

Track 17 is reserved exclusively for your Disk Directory, and only Sectors 2 through 11 are used for this purpose. Sector #2 contains the file allocation table and Sectors 3 through 11 contain the actual directory entries.

Let's look first at the way the computer sets up the directory entries. There are a total of 9 sectors dedicated to entries, and, since each entry occupies only 32 bytes of disk space, this means that there's room for 8 entries per sector, for a total of 72 possible directory entries. Realistically, however, since each file requires at least one granule of disk space, there can be no more than 68 unique files — that is, by logical extension, no more than 68 unique directory entries.

DIAGRAM 2

The directory entries have a special format:



The first 4 items on this list are self-explanatory (and fully explained in your owner's manual) — you see them every time you type DIR ENTER. Item 6 merely tells the computer how many bytes of the last sector contain actual file information.

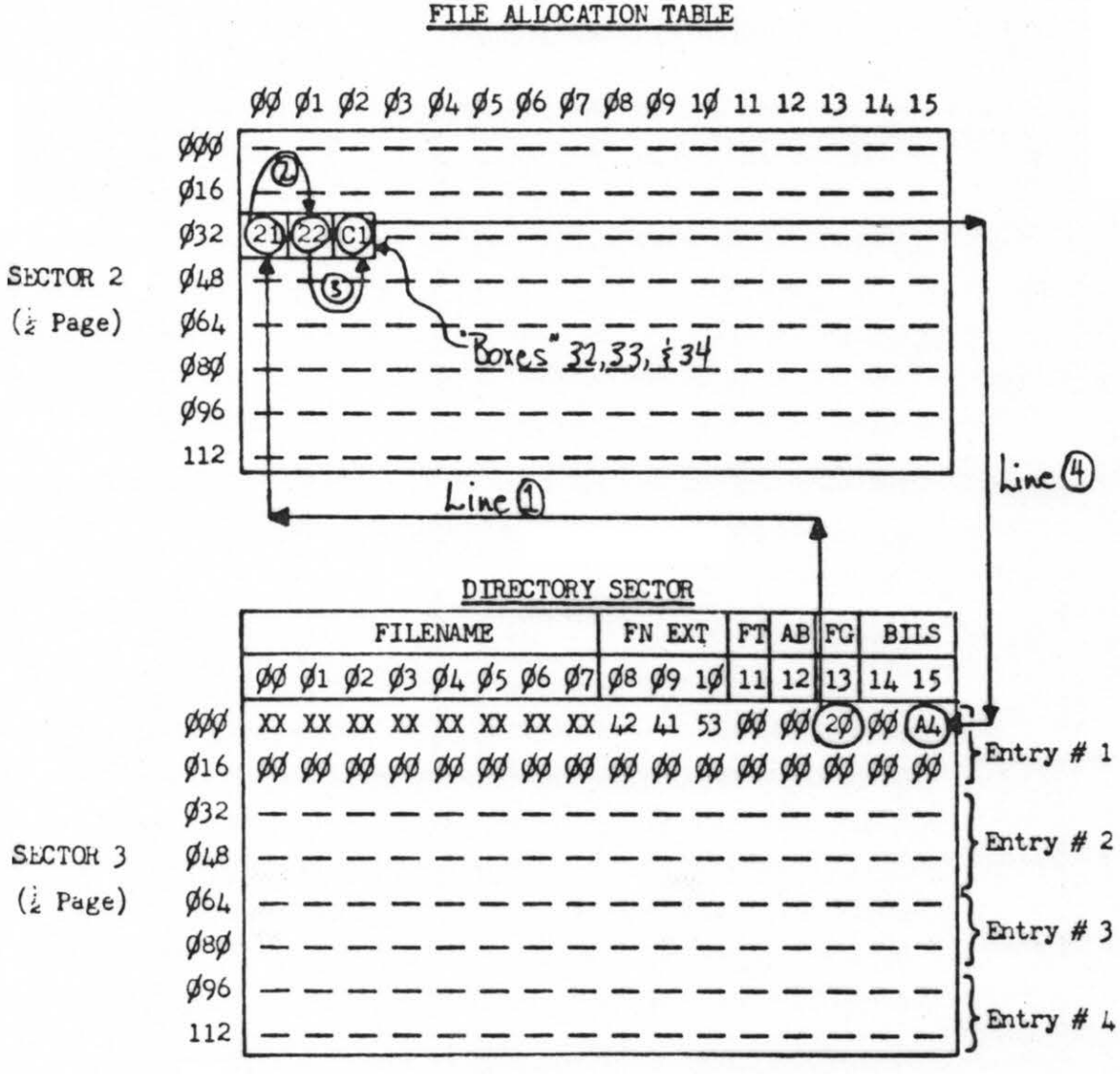
Item 5 on the list proves to be the most interesting piece of information from a programmer's point of view. It also happens to be the most confusing! Byte #13 of the directory entry performs two functions.

1. It tells the computer exactly where to look (by granule number) for the first segment of the file;
2. The computer must also look into the file allocation table (Sector #2) at the position named in byte #13 of the directory entry to determine if the specified granule is the only one, or, if not, where to find the second and subsequent granules in the file.

Now, of course, you are thoroughly confused, right? Well, never fear!! Take a look at the following diagram:

DIAGRAM 3

FILE ALLOCATION TABLE



Suppose there is only one file on our disk and, therefore, only one directory entry. (To avoid confusion, the elements of sector 2 will be called "boxes" and the elements of sector 3 will be called "bytes".)

The filename and extension are not important. Suffice to say that we have here a BASIC program stored in binary. Byte #13 tells us that program storage begins at granule #32 (20 Hex).

When loading the program, the computer will first check box 32 (line 1) to see where it must go next. In this case, box 32 contains the number 21 Hex (33 Decimal), so the computer knows that all of granule 32 must be loaded into memory and that it must then move to granule #33. Before loading granule #33, the computer first checks box 33 (line 2) and discovers 22 Hex (34 Decimal). Aha! Now the computer can load all of granule #33 because it knows there're still more data in granule #34. Then the computer checks box 34 (line 3) and finds C1 Hex — which tells the computer that granule #34 is the last granule in the file and that only the first sector of granule #34 is in use ("C" means last granule; "1" indicates only 1 sector); furthermore, only 164 (A4 Hex) bytes in the last sector of the file (see line 4) are in use (bytes 14 & 15 contain this information).

Trace through the diagram and read the above paragraph a few more times — everything will become clear momentarily.

You may have noticed that granule numbers and box numbers correspond on a 1-to-1 basis. This is because the first 68 locations ("boxes") of the File Allocation Table are in fact indices to succeeding granules. You have probably also noticed that numbers **contained** in the boxes (encircled) seem to run consecutively and are similar in magnitude to actual box numbers — this is no accident. You'll find as you examine more disks that the computer strives for efficient data storage by placing files as close as possible to the directory itself and by using consecutive or nearly consecutive granules so as to minimize movements of the read/write head.

Notice also that the depicted organization takes you through a closed loop — it starts at byte #13, moves into the file allocation table (from 1 to 68 boxes required) and ends up at bytes #14 & 15 of the entry. If this loop is broken anywhere, a load attempt will always result in an error — most likely an ?IO ERROR.

Summary

1. A formatted disk contains 35 tracks.
2. Each track consists of 18 sectors.
3. Each track (except track #17) consists of 2 granules of 9 sectors each.

4. Sector 2 of Track 17 is the File Allocation Table of the directory, the first 68 bytes of which are used as index references to granules having the same number designations. (Bytes 68 to 255 are not used by the computer — except for "garbage collection.")
5. Sectors 3 through 11 of Track 17 contain actual entries in the directory (8 per sector). Each entry occupies a space of 32 bytes, of which only 16 are used by the computer.

RECOVERING KILLED FILES

The "Hex Display Mode" of the EXAMINE/EDIT DISK subroutine is structured as per the diagram on page A-4. This display format is provided primarily for looking at the Directory track of your disk, but it can be used quite profitably for examination of any sector at all.

When you KILL a file, the computer does two things:
1. it writes a "00" to byte number 0 of the directory entry (the first character of the filename) but leaves the rest of the entry intact.
2. it writes "FF" to each entry in the File Allocation Table (Track 17/Sector 2) associated with the file you are KILLing.

All other data related to the file, including the entire contents of the file, are left intact.

This makes the directory entry and all granules associated with the file available for re-use. DISKEY has made no special provision for retrieving a KILLed file, but if you haven't subsequently SAVEd something on the same disk, you can retrieve your file as follows:

1. From the main menu, select option 2 (Drive # of your choice, Track #17).
2. Move to Sector 3 (or whichever sector contains the entry in question) and look at byte #13 of the directory entry containing the KILLed file; this will tell you which granule to look at first.
3. Trace through the disk until you locate the end of the file, keeping track of granule numbers (in CHRONOLOGICAL sequence) that are devoted to the KILLed file.
4. When all the information is compiled, return (using the arrow keys) to Track 17/Sector 3 and select Edit mode; enter an ASCII value corresponding to an uppercase letter (41-5A Hex) at byte #0 of the KILLed directory entry, eXit and Write the new data to the disk.
5. Move into Sector #2 and enter the values determined in step 3. Make sure you enter the values in the correct locations! (Use the diagram on page A-4 for reference.)

This method will work for any KILLed file, provided all data is intact when you start. Optionally, you can use the RECONSTRUCT DIRECTORY

option, but bear in mind that you will have to provide data for ALL the files on your disk.

Summary

There is nothing really mystical about your Color Computer Disk System, although at times it may seem that way. The information in this appendix, if used in conjunction with your owner's manual, should provide you with adequate background knowledge in system operation. Don't worry too much if it all seem like Greek to you right now. As you examine (and modify) more and more disks, everything will eventually begin to make sense. Pretty soon, you'll not only be able to retrieve "lost" data, but you'll also be able to use DISKEY as a serious debugging tool.

APPENDIX B

NOTES ON FILE STRUCTURE

If you have ever attempted something like LOADM "PROGRAM" when "PROGRAM" happens to be a BASIC program stored in binary, you will have discovered that the computer can differentiate quite nicely between file-types. The consequence of such an attempt would likely be a ? NE ERROR or perhaps a ? IO ERROR, which tells you that the computer cannot be fooled so easily!

And right now, you're sitting there smirking and saying, "Of course that'll happen — the information is right there in black and green (sic!) in the directory!" That's quite true — but here's a challenge for you. Write a short BASIC program, save it on disk, use the EXAMINE / EDIT subroutine of DISKEY to change the contents of byte #11 of the corresponding directory entry from 00 to 02, and then try to LOADM "PROGRAM". Now, when the computer checks the directory, it finds out that there is indeed a machine-language program named "PROGRAM". But what happens? One of two things:

1. You get an error of some type, probably ? FS ERROR or ? IO ERROR.
2. The program loads but will not RUN, and if you are gutsy enough to try EXEC, in all likelihood, the computer will wander off into outer space, never to return (except by turning off the machine)!

All of which proves that there is more to file storage than simply dumping data onto a few sectors and placing a "map" in the directory. As it turns out, each FILE-TYPE has a different storage format.

BINARY BASIC Programs

As mentioned in Appendix A, when LOADing, the computer will first check the directory entry and file allocation table to determine which granules are in use and how many bytes of the last sector are part of the program. Then the computer will look in the first sector of the first granule of the program file and check byte # 0, which must contain FF Hex; if not, LOADing will stop and you'll get an ? FS ERROR (bad file structure). The "FF" in byte # 0 is a flag that confirms that the file is in fact a BASIC program stored in binary.

Bytes # 1 & 2 of this sector tell the computer how much memory space is required when the program is LOADed. If we use the example from Appendix A, you'll remember our program occupied all of 14 sectors and 164 bytes of the 15th, for a total of 14 * 256 + 164, or 3748 bytes of disk space. Bytes # 1 & 2 of the first sector of the sample file would contain the

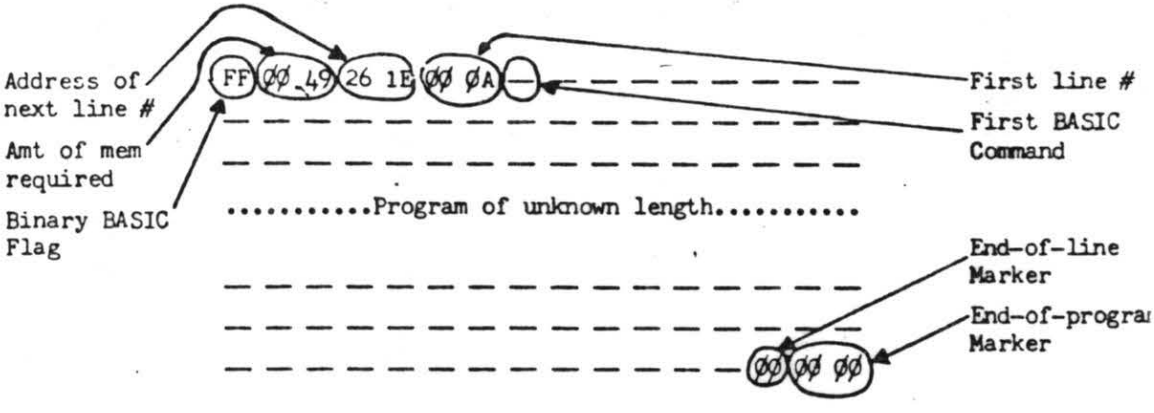
number 3745, three less than 3748. Why? Because the first 3 bytes of the file are not actually LOADed anywhere — they are simply control bytes for the operating system.

Bytes # 3 & 4 contain the address (a pointer, if you will) of the **next** program line that will be executed after the first one. (A bit of PEEKing into program memory will reveal to you that, even though your program LISTs in order by ascending line numbers, the program lines may actually be **stored** in quite a different order. The first two bytes of **every** program line contain a pointer to the next line, which may, under such a system, be several **thousand** bytes away in memory. This is an example of a "linked list.")

Bytes # 3 & 4 are the first two bytes that will actually be loaded into program memory, and some knowledge of your computer's memory layout will help you to predict exactly where they will load. In the case of a 32K Disk system after a normal power-up (automatic PCLEAR 4), the first available program location will be 9729 decimal (2601 Hex), which is where program loading will begin, unless you change the PCLEAR setting or use a different FILES value.

Take a look at the following diagram, which represents a BINARY BASIC program of unknown length, as it would be stored on the disk:

DIAGRAM 4



Except for the first 3 bytes, everything coincides exactly with the way in which programs are stored in memory. (Notice that the computer does not insert spaces after line numbers in this format — all spaces are inserted when the program is converted to ASCII format for LISTing or SAVEing.) Remember that all pointers are SAVEd exactly as they existed in memory. Subsequent LOADs of a given program will cause the computer to apply an offset to every address in the file if you are loading to a different PCLEAR configuration than when the program was saved, but, of course, the disk contents will not be affected.

Machine Language Programs

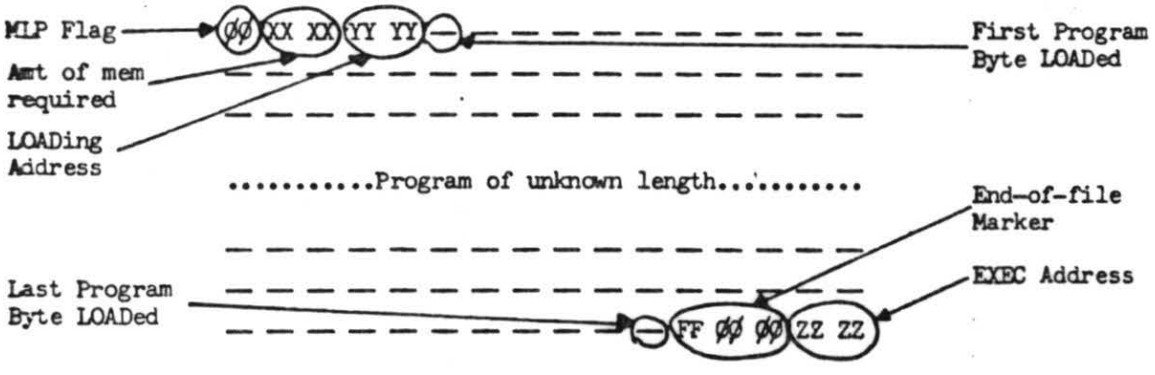
Whereas a BINARY BASIC program file reserves the first 3 bytes for system controls, a machine-language program file reserves the first 5 bytes *and* the last 5 bytes of the file, for a total of 10 system control bytes.

Byte # 0 contains a machine-language file flag, set to 00 instead of FF. Bytes # 1 & 2 contain the amount of memory space required for the file when LOADed, as before. This number will come to 10 less than that specified by information in the directory, because, once again, none of the control bytes are actually LOADed into program memory. Bytes # 3 & 4 tell the computer where to start LOADing the program. No end address (such as that specified in a SAVEM command) is required because the computer just continues LOADing until it reaches the end of the file. The last 5 bytes of the file contain a triple byte end-of-file marker — FF 00 00 — followed by two bytes which contain the address at which program EXECution will begin.

(In actual fact, this triple byte marker may be a link to additional data or program information which will ORiGinate at a new location in memory when loaded. If this is the case, the "FF" will be replaced by "00", the following two bytes will be a count of how much data to load, and the so-called "EXEC" address will now be the new "LOAD" address.)

The following diagram will help to clarify the situation:

DIAGRAM 5



BASIC Program in ASCII

In this type of file, every character in the program, including each digit of each line number and each letter of each BASIC command, is converted to its ASCII equivalent and SAVEd in that format.

There is only one peculiarity about the ASCII format — the very first byte of the file always contains 0D Hex (carriage return). Additionally, every line number is followed by a space (not the case for binary BASIC program

files), and every program line ends with a carriage return. This means that the entire ASCII file is bracketed by carriage returns.

Data Files

The organization of other data files is based primarily on how you have chosen to structure them — i.e., how you use FILES, FIELD, and other Disk System commands. Essentially, though, from what can be "seen" in data files, there are no control bytes apart from those contained in the directory sector and the file allocation table. In other words, byte # 0 of your data file contains your first byte of data, whatever it might be.

Summary

Each type of file has its own unique storage format over and above that specified in Track # 17. If you have a good grasp of this information, you should be able to construct all types of files from scratch, without first writing the file to memory.

NOTE: Don't get locked into the belief that values contained in byte # 0 of your file are **always** control values. They are merely a confirmation of information in the directory. Remember, there's no reason why a simple data file cannot contain 00, 0D, or FF in byte # 0.

APPENDIX C

CHARTS AND TABLES

CHART #1

Condition of Track-Sector Combinations

Track(s)	Sector(s) (Grans Below)	Data (Y/N)	Readable (Y/N)

CHART #2

Track-Sector Data

Track(s)	Sector(s)	Granule #	File Name	File Type	Format (A/B)	Gran Pos'n	BILS

CHART #3

Assumed Layout of Directory

File Name	File Type	Storage Format	Granules Used (Chronological)	Sectors Last Gran	BILS

Table #1 — BASIC TOKENS (Lexicographical Order)

COMMAND	TOKEN	COMMAND	TOKEN	COMMAND	TOKEN
'	83	GO	81	RESET	9D
*	AD	HEX$	FF9C	RESTORE	8F
+	AE	IF	85	RETURN	90
-	AC	INKEY$	FF92	RIGHT$	FF8F
/	AE	INPUT	89	RND	FF84
<	B4	INSTR	FF9E	RSET	D7
=	B3	INT	FF81	RUN	8E
>	B2	JOYSTK	FF8D	SAVE	D8
ABS	FF82	KILL	D2	SCREEN	BF
AND	B0	LEFT$	FF8E	SET	9C
AS	FFA7	LEN	FF87	SGN	FF80
ASC	FF8A	LET	BA	SIN	FF85
ATN	FF94	LINE	BE	SKIPF	A3
AUDIO	A1	LIST	94	SOUND	A0
BACKUP	DC	LLIST	9E	SQR	FF9E
CHR$	FF8B	LOAD	D3	STEP	A9
CIRCLE	C2	LOC	FFA4	STOP	91
CLEAR	95	LOF	FFA5	STR$	FF83
CLOAD	97	LOG	FF99	STRING$	FFA1
CLOSE	9A	LSET	D4	SUB	A6
CLS	9E	MEM	FF93	TAB	A4
COLOR	C1	MERGE	D5	TAN	FF96
CONT	93	MID$	FF90	THEN	A7
COPY	DE	MKN$	FFA6	TIMER	FF9F
COS	FF95	MOTOR	9F	TO	A5
CSAVE	98	NEXT	8B	TROFF	B8
CVN	FFA2	NEW	96	TRON	B7
DATA	86	NOT	A8	UNLOAD	DB
DEF	B9	OFF	AA	USING	CD
DEL	E5	ON	88	USR	FF83
DIM	8C	OPEN	99	VAL	FF89
DIR	CE	OR	B1	VARPTR	FF9D
DLOAD	CA	PAINT	C3	VERIFY	DA
DRAW	C6	PCLEAR	C0	WRITE	D9
DRIVE	CF	PCLS	BC	↑	AF
DSKI$	DF	PCOPY	C7		
DSKINI	DC	PEEK	FF86		
DSKO$	E0	PLAY	C9		
EDIT	B6	PMODE	C8		
ELSE	3A84 *	POINT	FF91		
END	8A	POKE	92		
EOF	FF8C	POS	FF9A		
EXEC	A2	PPOINT	FFA0		
EXP	FF97	PRESET	BE		
FIELD	D0	PRINT	87		
FILES	D1	PSET	BD		
FIX	FF98	PUT	C5		
FN	CC	READ	8D		
FOR	80	REM	82		
FREE	FFA3	RENAME	D6		
GET	C4	RENUM	CB		

*"3A" (colon) in the "ELSE" token (3A84) is supplied by BASIC

Table #2 — BASIC TOKENS (Numerical Order)

TOKEN	COMMAND	TOKEN	COMMAND	TOKEN	FUNCTION
80	FOR	B3	=	FF80	SGN
81	GO	B4	<	FF81	INT
82	REM	B5	DEL	FF82	ABS
83	'	B6	EDIT	FF83	USR
84	ELSE	B7	TRON	FF84	RND
85	IF	B8	TROFF	FF85	SIN
86	DATA	B9	DEF	FF86	PEEK
87	PRINT	BA	LET	FF87	LEN
88	ON	BB	LINE	FF88	STR$
89	INPUT	BC	PCLS	FF89	VAL
8A	END	BD	PSET	FF8A	ASC
8B	NEXT	BE	PRESET	FF8B	CHR$
8C	DIM	BF	SCREEN	FF8C	EOF
8D	READ	C0	PCLEAR	FF8D	JOYSTK
8E	RUN	C1	COLOR	FF8E	LEFT$
8F	RESTORE	C2	CIRCLE	FF8F	RIGHT$
90	RETURN	C3	PAINT	FF90	MID$
91	STOP	C4	GET	FF91	POINT
92	POKE	C5	PUT	FF92	INKEY$
93	CONT	C6	DRAW	FF93	MEM
94	LIST	C7	PCOPY	FF94	ATN
95	CLEAR	C8	PMODE	FF95	COS
96	NEW	C9	PLAY	FF96	TAN
97	CLOAD	CA	DLOAD	FF97	EXP
98	CSAVE	CB	RENUM	FF98	FIX
99	OPEN	CC	FN	FF99	LOG
9A	CLOSE	CD	USING	FF9A	POS
9B	LLIST	CE	DIR	FF9B	SQR
9C	SET	CF	DRIVE	FF9C	HEX$
9D	RESET	D0	FIELD	FF9D	VARPTR
9E	CLS	D1	FILES	FF9E	INSTR
9F	MOTOR	D2	KILL	FF9F	TIMER
A0	SOUND	D3	LOAD	FFA0	PPOINT
A1	AUDIO	D4	LSET	FFA1	STRING$
A2	EXEC	D5	MERGE	FFA2	CVN
A3	SKIPF	D6	RENAME	FFA3	FREE
A4	TAB	D7	RSET	FFA4	LOC
A5	TO	D8	SAVE	FFA5	LOF
A6	SUB	D9	WRITE	FFA6	MKN$
A7	THEN	DA	VERIFY	FFA7	AS
A8	NOT	DB	UNLOAD		
A9	STEP	DC	DSKINI		
AA	OFF	DD	BACKUP		
AB	+	DE	COPY		
AC	-	DF	DSKI$		
AD	*	E0	DSKO$		
AE	/				
AF	↑				
B0	AND				
B1	OR				
B2	>				

Table #3 — CHR$ / POKE Comparison

##	CHR$	POKE	##	CHR$	POKE	##	CHR$	POKE
00		@ (I)	2B	+	+ (I)	56	V	V
01		A (I)	2C	,	, (I)	57	W	W
02		B (I)	2D	-	- (I)	58	X	X
03	BRK	C (I)	2E	.	. (I)	59	Y	Y
04		D (I)	2F	/	/ (I)	5A	Z	Z
05		E (I)	30	0	0 (I)	5B	[	[
06		F (I)	31	1	1 (I)	5C	\	\
07		G (I)	32	2	2 (I)	5D	]	]
08		H (I)	33	3	3 (I)	5E	↑	↑
09		I (I)	34	4	4 (I)	5F	←	←
0A		J (I)	35	5	5 (I)	60	@ (I)	SPC
0B		K (I)	36	6	6 (I)	61	A (I)	!
0C	CLR	L (I)	37	7	7 (I)	62	B (I)	"
0D	ENT	M (I)	38	8	8 (I)	63	C (I)	#
0E		N (I)	39	9	9 (I)	64	D (I)	$
0F		O (I)	3A	:	: (I)	65	E (I)	%
10		P (I)	3B	;	; (I)	66	F (I)	&
11		Q (I)	3C	<	< (I)	67	G (I)	'
12		R (I)	3D	=	= (I)	68	H (I)	(
13		S (I)	3E	>	> (I)	69	I (I)	)
14		T (I)	3F	?	? (I)	6A	J (I)	*
15		U (I)	40	@	@	6B	K (I)	+
16		V (I)	41	A	A	6C	L (I)	,
17		W (I)	42	B	B	6D	M (I)	-
18		X (I)	43	C	C	6E	N (I)	.
19		Y (I)	44	D	D	6F	O (I)	/
1A		Z (I)	45	E	E	70	P (I)	0
1B		[(I)	46	F	F	71	Q (I)	1
1C		\ (I)	47	G	G	72	R (I)	2
1D		] (I)	48	H	H	73	S (I)	3
1E		↑ (I)	49	I	I	74	T (I)	4
1F		← (I)	4A	J	J	75	U (I)	5
20	SPC	SPC(I)	4B	K	K	76	V (I)	6
21	!	! (I)	4C	L	L	77	W (I)	7
22	"	" (I)	4D	M	M	78	X (I)	8
23	#	# (I)	4E	N	N	79	Y (I)	9
24	$	$ (I)	4F	O	O	7A	Z (I)	:
25	%	% (I)	50	P	P	7B	[(I)	;
26	&	& (I)	51	Q	Q	7C	\ (I)	<
27	'	' (I)	52	R	R	7D	] (I)	=
28	(	((I)	53	S	S	7E	↑ (I)	>
29	)	) (I)	54	T	T	7F	← (I)	?
2A	*	* (I)	55	U	U			

NOTES: 1. The notation "(I)" indicates that the specified character will appear on the screen in Inverse Video (green on a black background).

2. All characters from 80 to FF Hex (128 to 255 Decimal are special graphics characters defined by BASIC. See your programming manual for details.

Table #4 — One-Byte Hexadecimal to Decimal Conversion

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
00	0	34	52	67	103	9A	154	CD	205
01	1	35	53	68	104	9B	155	CE	206
02	2	36	54	69	105	9C	156	CF	207
03	3	37	55	6A	106	9D	157	D0	208
04	4	38	56	6B	107	9E	158	D1	209
05	5	39	57	6C	108	9F	159	D2	210
06	6	3A	58	6D	109	A0	160	D3	211
07	7	3B	59	6E	110	A1	161	D4	212
08	8	3C	60	6F	111	A2	162	D5	213
09	9	3D	61	70	112	A3	163	D6	214
0A	10	3E	62	71	113	A4	164	D7	215
0B	11	3F	63	72	114	A5	165	D8	216
0C	12	40	64	73	115	A6	166	D9	217
0D	13	41	65	74	116	A7	167	DA	218
0E	14	42	66	75	117	A8	168	DB	219
0F	15	43	67	76	118	A9	169	DC	220
10	16	44	68	77	119	AA	170	DD	221
11	17	45	69	78	120	AB	171	DE	222
12	18	46	70	79	121	AC	172	DF	223
13	19	47	71	7A	122	AD	173	E0	224
14	20	48	72	7B	123	AE	174	E1	225
15	21	49	73	7C	124	AF	175	E2	226
16	22	4A	74	7D	125	B0	176	E3	227
17	23	4B	75	7E	126	B1	177	E4	228
18	24	4C	76	7F	127	B2	178	E5	229
19	25	4D	77	80	128	B3	179	E6	230
1A	26	4E	78	81	129	B4	180	E7	231
1B	27	4F	79	82	130	B5	181	E8	232
1C	28	50	80	83	131	B6	182	E9	233
1D	29	51	81	84	132	B7	183	EA	234
1E	30	52	82	85	133	B8	184	EB	235
1F	31	53	83	86	134	B9	185	EC	236
20	32	54	84	87	135	BA	186	ED	237
21	33	55	85	88	136	BB	187	EE	238
22	34	56	86	89	137	BC	188	EF	239
23	35	57	87	8A	138	BD	189	F0	240
24	36	58	88	8B	139	BE	190	F1	241
25	37	59	89	8C	140	BF	191	F2	242
26	38	5A	90	8D	141	C0	192	F3	243
27	39	5B	91	8E	142	C1	193	F4	244
28	40	5C	92	8F	143	C2	194	F5	245
29	41	5D	93	90	144	C3	195	F6	246
2A	42	5E	94	91	145	C4	196	F7	247
2B	43	5F	95	92	146	C5	197	F8	248
2C	44	60	96	93	147	C6	198	F9	249
2D	45	61	97	94	148	C7	199	FA	250
2E	46	62	98	95	149	C8	200	FB	251
2F	47	63	99	96	150	C9	201	FC	252
30	48	64	100	97	151	CA	202	FD	253
31	49	65	101	98	152	CB	203	FE	254
32	50	66	102	99	153	CC	204	FF	255
33	51								

APPENDIX D

SAMPLE RUN THROUGH

If you are like most programmers, you are very careful with your disks — after all, you invest money in the materials, and time and effort in the programs you store on them. Unfortunately, though, sometimes things just go wrong. Like last night, just when you were so excited about applying the finishing touches to your all-singing, all-dancing 48K arcade game — you came home from a hard day, dumped your coat and boots, ran to your computer, loaded your program, and ... and ... got a ?IO ERROR. Huh? Oh, sure, I picked up the wrong disk — but, just to be sure ... D I R ENTER ... grind ... grind ... grind ... grind ... pause ... ?IO ERROR. What gives? You go through 53 other disks and, in mounting panic, fail to find the program you're looking for. No doubt about it now — the program WAS on the first disk. You realize that without a listing or backup, you have only one alternative — rack your brain and try to remember what it was that you spent 192 hours and 43 minutes typing and editing. You contemplate suicide, but that seems just a tad melodramatic. Instead, you just sob....

Sound familiar? Well, just hold on a sec! DISKEY might be able to help. Sure, maybe parts of your program are lost, but you've got nothing to lose by trying. And maybe you'll be able to recover enough data to sufficiently refresh your memory so that you can recreate the program. It just may be that the program is completely intact, but your directory is fried. (You should be so lucky!)

For the sake of simplicity, let's assume that one or two files have mysteriously disappeared from your directory, as is the case with the BACKUP copy you have made of DISKEY. We'll use this backup for the rest of this appendix, and you should now have the disk in Drive 0 and the main menu on the screen. Be sure you have a copy of each chart in Appendic C — we'll be using these to log pertinent data.

Step One

Select option I at the menu. We'll take a look at the map of the disk to isolate any faulty sectors, if there are any. Since this is a brand spanking new program, and you have only made one backup, there should be no problem with the Disk Map — that is, your screen should display a 35 x 18 rectangle of yellow blocks. If, however, you have any other indications at all, respond by pressing C, if necessary. (In this case, the presence of blue, green, or red blocks would probably indicate that your disk was flawed to begin with or that your drive is incorrectly tuned. See the main text for details.)

If the Directory had been fried as I suggested at the start of this exercise, the map would likely show one or more blue blocks on Track 17, starting at Sector 2. We would normally log these sectors, and any others, as being unreadable. Since this is not the case in this example, just note in column 4 of Chart #1 that ALL of the disk is readable.

Step Two

No matter what kind of problem you may have with your disk, you should always examine the Directory track whenever possible. If it is intact, even if only in part, you will save a lot of time in the reconstruction process if you use whatever information is available.

To simplify things for yourself, I recommend that you take a printout (from the EXAMINE / EDIT DISK routine) of each sector on Track #17 that contains readable information, starting with Sector #2. Keep the printout(s) handy for use later.

Step Three

The next thing to do is to determine where all data is located on the disk. We will do this by a logical search sequence. Since BASIC organizes all files to begin in the first sector of a granule, it can be assumed that we need only search these sectors (Sectors #1 and #10 in each Track) to positively determine whether or not that granule contains data.

From the menu, select option 2, Drive 0, Track #0 and press ENTER when ready. The display will show a block of orange, indicating an empty granule. (Remember, this is only true if we're talking about "normal" BASIC files — there's no reason why some smart-alec programmer can't store pertinent data in, say, Track 32 / Sector 17 — but, in this example, you can trust me!) Use the down arrow to page through even-numbered granules (Sector #1 on each Track) until something other than an orange block appears on the screen. For this disk, orange blocks will continue to show up until Track #16.

Continue paging until you reach Track #34, noting which tracks contain data and which do not. When you're finished the first pass, you will have recorded Tracks 16, 18, 32, 33, and 34 as bearing data. (Remember, Track #17 will come up empty in this sequence because BASIC does not use Sector #1. In any event, Track #17 does not constitute part of a granule.)

Now use the right arrow to move over to Sector #10, and use the up arrow to page backwards through the odd-numbered granules (Sector #10 on each track) until you return to Track #0. Remember to note again which granules contain data. When you are done, you will have logged Tracks 34, 33, 32, and 16.

The information we have gathered so far indicates that data is contained in Granules #32 to #34 inclusive, as well as #62 to #67 inclusive. If you examine your printouts from the Directory Track, you will see that the File Allocation Table appears to be rather full. This is because the File Allocation Table for this disk has been severly modified to prevent accidental writing on the disk. In actual fact, the directory entry for "MASTER/BAS" requires only one granule, #32. It would be logical to assume that the data for the "missing" file(s) is contained in Granules #33 and #34 (you can safely ignore Granules #62 to #67 for this exercise.) You can and should verify this information by performing a detailed analysis of Tracks #16 and #18, to find out exactly which sectors belong to each file. Chart #1 should look as follows:

Condition of Track-Sector Combinations

Track(s)	Sector(s) (Grans Below)	Data (Y/N)	Readable (Y/N)
0 - 15	All (0 - 31)	No	Yes
16	S1 (32)	Yes	Yes
16	S10 - S14 (33)	Yes	Yes
17 (Dir)	S1 - S18	Yes (S2 & S3)	Yes
18	S1 - S6 (34)	Yes	Yes
19 - 34	All (35 - 67)	No	Yes

In our example, you can get away with using data in the directory to help you determine which files are stored where. Whenever this is possible, do it! In many cases, however, you will find that the directory itself has been obliterated, and, as a result, you must page through the disk manually in order to determine the necessary information required to reconstruct the directory. In order to make this exercise more interesting, the "missing" files have not only been KILLed, but all references to them in the directory have been eliminated as well. This will require you to go through the disk and find the data.

Step Four

If the disk you're working on happens to contain a number of unreadable sectors, the next step would be to transfer all GOOD data to a new, formatted disk. You would do this using the COPY function as well as the Write subroutine in the EXAMINE/EDIT DISK function. You have all

the necessary information for transfer in chart #1. In our example, this is not required, but you are welcome to do it if you want the practice.

Anytime data transfer is necessary, do it in a logical sequence of steps. COPY all good tracks first, keeping in mind that the Start and End Track values you enter will be considered to be inclusive. When this is done, you can transfer individual sectors from any track which is not totally readable by first READing the sector from the faulty disk, placing your good disk in the SAME drive, and Writing the data back to the same place on the new disk. Repeat this process as often as necessary until all good sectors have been transferred. If, for example, Track 17 / Sector 2 was unreadable, you would use the COPY function to transfer track 0 to 16 inclusive and repeat the procedure for tracks 18 to 34 inclusive. Then you would use the EXAMINE / EDIT DISK function to transfer all readable sectors from Track #17.

NOTE: It is (obviously) not necessary to transfer Tracks or Sectors which are known to be completely empty. This would be an unnecessary waste of time.

Step Five

Now that our data is transferred to a new disk, we can go through the process of reconstructing the directory. This is a fairly easy task, all things considered. If you've faithfully gone through Appendix B, you should have a pretty good idea of how to identify files by looking at the sectors where they are stored. Put the good disk into drive 0 and we're ready to go.

The data for one of our three files is displayed quite clearly in the printouts we obtained earlier. If you wish, you can enter the data into Chart #2, but you can save yourself some time if you enter it directly into Chart #3. File #1 is a BASIC program stored in Binary, starting and ending in Granule #32 (only one Sector is used), and the last sector of the file contains only 109 (60 Hex) bytes of data.

When you enter this data into Chart #3, apply the filenames "A1", (and "A2" and "A3" as new data becomes available), as this is the name the computer will assign. Now you can examine Tracks #16 and #18, containing our "stray" files.

Look at Track 16 / Sector 10 carefully. Flip back and forth between Graphic and Hex Display if you wish. Note that the first byte in the sector is FF Hex. We know from Appendix B that this could indicate the start of a BASIC Program stored in Binary format. Is this the case? If it were your own program, you'd have little difficulty answering that question, but since it was written by someone else, you have to do some deductive reasoning.

Press E to enter the Edit mode and move the cursor around. Pay particular attention to the characters that precede the "@" symbols (which could indicate the presence of a "PRINT" statement). Note the Hex value of the character just to the left — 87 Hex. Doesn't mean much, does it? But look at the BASIC Token Chart in Appendix C and see if you can find 87 Hex . . . there it is! A PRINT statement for sure. Still not satisfied? Exit from Edit mode and move through sectors until you find one that has a number in it that looks suspiciously like a line number. Now think — what BASIC statement(s) can precede a line number? How about GOTO, GOSUB, THEN, or ELSE? Check them out the same way you checked out the PRINT statement.

One more piece of information is important to us — how many bytes of the last sector are part of the file? Well, first we have to find the end of the file, which just happens to be Sector #14. This we know because, having diligently checked out the disk, we noticed that there was data in Sector #14 but nothing in Sector #15. So Sector #14 is the end of this file. We know from Appendix B that the end of a BASIC-Binary program file can be detected by the presence of a triple zero marker ("000"). But what happens if, for some reason, this marker exists more than once? This is not likely, but it is possible, especially if a new, shorter program has overwritten an older, longer one — always look for the marker that's closest to the beginning of the file. If you look at Sector #14, you will see this "000" marker (it will appear as 3 inverse video "@" symbols) on the 4th line from the top of the display. To be sure it's the right one, put your finger on the 3rd "@" and, using the left and right arrows, flip back and forth between Sector #13 (the second last sector) and Sector #14 (the last sector). You will notice as you do this that only the data to the right of and below your finger will change, indicating that the byte beneath your finger is, in fact, the last byte in the file. Here's what happened: when the computer writes a file to disk, it sends 256 bytes at a time to its write buffer, and writes the data to the appropriate sector. As each sector is written, new data over-writes old data in the write buffer. After the second last sector has been written, the remaining part of the file may not fill the entire write buffer, so, whatever was in the buffer before will not be completely replaced. But the entire buffer is sent to the disk anyway, even though the last few bytes of the last sector may be "garbage". For this reason, the computer must keep track of how many bytes of the last sector are actual data.

Using this method you can easily count the number of bytes in the last sector of the file. Our first "missing" file then, is a BASIC program stored in Binary, starting and ending in Granule #33, occupying 5 sectors of the granule, and there are 99 bytes in the last sector. Make the appropriate

entries in Chart #2 and proceed in the same fashion for the other "missing" file in Track 18 / Sector 1. This file, you will quickly see, is also a BASIC program, but this time it's stored in ASCII. I'll leave it to you to determine the correct parameters to enter in Chart #2.

Chart #2 should now look like the following:

Track-Sector Data

Track(s)	Sector(s)	Granule #	File Name	File Type	Format (A/B)	Gran Pos'n	BILS
16	S1 (1)	32	A1	BAS	Bin	Last	109
16	S10-S14(5)	33	A2	BAS	Bin	Last	99
18	S1 -S6 (6)	??	??	???	???	????	???

The number in parentheses in column 2 represents the number of sectors in the last granule. Column 7 is provided so that when you transfer information to Chart #3, you can easily list granule numbers in chonological order. Notice that entries are made in this chart in order by increasing track numbers, but that filenames are determined according to the way in which BASIC normally organizes files. Remember, BASIC will always strive to enter new files as close as possible to the directory track. If you run into problems with a disk that has many files on it, keep this tendency in mind — it will make the reconstruction process much easier.

Step Six

Now that all our data is compiled in Chart #2, it is easy to transfer the information to Chart #3. Chart #2 is organized to facilitate looking through the disk in sequence by track number, while Chart #3 is organized to facilitate data entry in the RECONSTRUCT DIRECTORY function. When the data is transposed, Chart #3 should look as follows:

Assumed Layout of Directory

File Name	File Type	Storage Format	Granules Used (Chronological)	Sectors Last Gran	BILS
A1	BAS Prog	Binary	32	1	006D
??	???	???	??	?	????
A3	BAS Prog	ASCII	34	6	005A

Step Seven

Now it is a simple case of entering known data into the RECONSTRUCT DIRECTORY routine of the DISKEY program. This I will leave to you as an exercise. After you have rebuilt the directory, remember to use the information in column #6 of Chart #3 to correct the Bytes In Last sector as reflected in bytes 14 and 15 in each directory entry. Use the EXAMINE/EDIT DISK function and CLEAR to enter the data in Hex notation.

Step Eight

After all the data have been correctly entered, the only thing that remains is to verify that it was correctly written to the disk. Use the EXAMINE function to look at Track #17, Sector #2 and #3, to satisfy yourself that everything is correct. If it is, then Exit the program as described in the main text and RENAME the files as follows:

A1/BAS ———> MASTER/BAS
A2/BAS ———> PROGSEL1/BAS
A3/BAS ———> PROGSEL2/BAS

"PROGSEL1" and "PROGSEL2" represent the same program, stored in different formats. Their function is quite simple — they allow you to select and execute any program on your disk just by pressing a button on the keyboard. Try one of them — you may want to keep a copy on each of your disks — and you'll find that they are very easy to use.

Summary

Any disk that starts giving you trouble can be taken through these eight steps. There are no guarantees that your data will be saved, but at the very least you now have a route to follow. Good luck, and have fun!

APPENDIX E

SAMPLE PRINTOUTS

A. Track / Sector Printout

In this type of printout, the ASCII representations of all hex bytes greater than 7F have been normalized so that bit #7 is cleared to zero. This ensures that all graphics characters are converted to "printable" data. After this conversion, any character that remains unprintable is represented by a period.

```
DRIVE 0/TRACK 17/SECTOR 02

C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1     AAAAAAAAAAAAAAAA
A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2     !"!"!"!"!"!"!"!"
A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2 A1 A2     !"!"!"!"!"!"!"!"
A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8     $($($($($($($($(
A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8 A4 A8     $($($($($($($($(
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4     DDDDDDDDDDDDDDDD
```

```
DRIVE 0/TRACK 17/SECTOR 03

4D 41 53 54 45 52 20 20 42 41 53 00 00 20 00 6D     MASTER  BAS.. .m
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF     ................
```

B. Directory Printout.

In this type of printout, the number of free granules is determined not by an examination of the File Allocation Table (which is how BASIC does it), but rather by counting the number of granules occupied by each directory entry and subtracting the total from 68. This procedure will produce correct results for any standard disk, but if the disk has actually been filled by non-standard means (as was done with the DISKEY master disk), this will produce erroneous results.

DISKETTE DIRECTORY FOR: 〈〈〈DISKEY〉〉〉 & 〈〈〈CCDIAG〉〉〉

MASTER BAS 0 B 01 BASIC PROGRAM / BINARY FORMAT

FREE SPACE REMAINING: 67 GRANULES

CCDIAG FUNCTIONS

ROM TEST

This simple test calculates a two-byte checksum for the ROM selected. (The checksum is a running total of the contents of each memory location in the ROM.) As the checksum is being determined, it will be displayed (in hexadecimal) and continually updated on the screen.

The values displayed as "CHECKSUM SHOULD BE: $xxxx" are based on known values for BASIC 1.1, EXTENDED BASIC 1.0, and DISK BASIC 1.0.

Once the selected test is complete, press ENTER to return to the sub-menu. To perform another test, simply press another key, or press BREAK to return to the main menu.

RAM TEST

This test checks all RAM addresses from 0 to the end of 16K or 32K memory, including the area occupied by the program. It does this by writing all possible values from 0 to 255 into each location, checking to ensure that the data is correctly stored there and then restoring the original contents. As this process takes place, the screen will display the current address being checked.

Once the check is complete, a message will be displayed indicating the status of your RAM. If any bad addresses are detected, the message will indicate the first bad address and list the individual bits which are at fault. This will enable you to replace a specific bad chip, instead of the entire RAM.

The BREAK key is inactive during this test. If you must halt the test for any reason, you can do so by pressing the RESET button. However, AVOID doing this while the displayed address is between $1000 and $2600, as you may cause part of the program to be changed. This could have disastrous results later on.

KEYBOARD TEST

Selection of this test will cause a keyboard matrix to be displayed on the screen. Each time a key or a combination of keys is pressed, a blue square will cover the appropriate spot on the matrix. Provision is made for testing the joystick buttons as well. The matrix will remain on the screen until a 10-second delay between keypresses occurs, after which you will be returned to the main menu.

JOYSTICK TEST

This test uses a green 128 x 64 graphic screen, split into 2 vertical halves — the left side for the left joystick, and the right side for the right joystick. As the joystick is moved around, a red dot appears on the screen in the appropriate position. Holding down the firing button during this movement will cause dots to be erased.

The CLEAR key will erase the entire screen, and the BREAK key will return you to the main menu.

PRINTER TEST

This test has two distinct functions. First is the status check which allows testing of the computer's ability to detect a READY / NOT READY condition for various printer states. Try this test with power ON / OFF, printer ONLINE / OFFLINE, RESTART / RESET, and printer OUT OF PAPER. When done, press BREAK to return to the sub-menu.

The second test assumes that the printer is an 80-column device, and sends 10 lines of standard characters at the selected baud rate. (Note that pressing ENTER will cause 600 baud to be selected.) When complete, you will be automatically returned to the sub-menu. Press BREAK to return to the main menu.

CASSETTE TEST

This test first writes data to the tape and then reads it back to ensure it was written properly, and can be completed simply by following the directions on the screen. Possible errors include:

I. CHECKSUM ERROR — data as read are not the same as were written;

II. MEMORY ERROR — data are correct on tape but will not store properly in memory, indicating possible bad RAM;

III. LOST DATA ERROR — data were written correctly and read correctly into good RAM, but are no longer the same as originally written, indicating either intermittently bad RAM, or the original checksum was incorrectly calculated.

You can exit this function by pressing BREAK any time the flashing cursor appears.

DISK TEST

This is a four step test that checks the ability of the specified drive to read, write and smoothly operate the stepping motor. Press BREAK at the sub-menu to quit.

A. Full Test

This part of the test checks all of the functions at once by calling each of the other subroutines. First, you are asked to place an UNFORMATTED disk in the specified drive. Strictly speaking the disk need not be unformatted, but be aware that this routine re-initializes (and erases) the disk, in exactly the same way as the DSKINI command works. This tests the controller's ability to write and then read all track information, including system data. It also confirms that the drive speed is within allowable limits. Once this part of the test is complete, the read/write/verify check is performed, followed by the stepping motor check.

B. Stepping Motor Test

This subroutine and those that follow will prompt you to put a FORMATTED disk in the specified drive. (The prompt will NOT be present if you are doing the full test.) This part checks rapid motion of the stepping motor, and will do 10 passes consisting of stepping from track 0 to track 34 and back again.

An error condition during this test could be caused by any of the following:

I. Dirty contacts between the computer and the drive;
II. A bad stepping motor;
III. The read/write head snagging the disk itself, due to close proximity to the disk surface, resulting in uneven wear on the head.

C. Read Test

This subroutine requires a formatted disk and will read only the data (not system information) from each sector of each track. An error condition during this test could indicate any of the following:

I. Dirty contacts;
II. Dirty or misaligned read-write head;
III. Incorrect or fluctuating drive speed;
IV. Slipping pulley band, caused by dirt;
V. A stepping motor problem;
VI. A faulty disk.

D. Write Test

The write test, which also requires a formatted disk, will first read the data from each sector of each track, write the same data back to the same sectors, and then read it again to verify that it was written correctly. Any error condition can be caused by the problems listed in the "Read Test" section.

NOTE: For further information on the care of disks and drives, refer to page 13.

VIDEO TEST

This test consists of two parts, which are selected from the submenu. Press BREAK to return to the main menu.

A. Color Adjustment

Use this test to adjust the brightness, contrast, color intensity, and tint of your color monitor. The screen displays 8 vertical color bars which can be moved left or right by means of the arrow keys for fine-tuning purposes. Press SPACE BAR to invert the test pattern. Press BREAK to return to the sub-menu.

B. Graphics Check

This is an interactive test that allows you not only to test the various graphic modes, but also to experiment with them. At the start, the screen will display a standard text screen with inverse video "@" symbols filling the entire screen. A flashing cursor will appear in the upper left corner. Several keys have been implemented to allow changes from one mode to another. All keys incorporate automatic repeat.

↑ / ↓	— Increase or decrease the 3-bit graphic mode value.
→ / ←	— Increase or decrease the 8-bit color set select value.
> / <	— Increase or decrease the ASCII/Graphic value of the vertical line of characters beneath the cursor.
P	— Send the current graphic mode and color set information to the printer (at the last selected baud rate).
CLEAR	— Fill the entire screen with the character under the cursor.
SPACE BAR	— Move the cursor one position to the right and wraparound to the start if at the end of the top line.
BREAK	— Return to the sub-menu.

NOTE: 1. If you have only 16K, some of the graphic modes may produce garbage at the bottom of the screen.
2. Regardless of which screen is displayed, you can identify different graphic modes by the size and type of cursor being displayed. (In some modes, you may see two distinct cursors).
3. There are 8 different graphic modes and 32 different color/screen types, for a total of 256 different graphic combinations (some of which require as much as 9K RAM!)

SOUND TEST

This is another interactive test that allows you to experiment while confirming that your digital/analog converter works properly. The basis of the routine is quite simple.

```
SOUND   LDA    #$3F
        STA    $FF23      ENABLE SOUND
SND01   LDA    BASE       GET STARTING VALUE
SND02   ORA    #2         KEEP PRINTER QUIET
        STA    $FF20      MAKE A NOISE
        LDB    DURATN     GET NOTE TIMER
SND03   DECB              COUNTDOWN
        BNE    SND03      LOOP TILL DONE
        ADDA   MOD        ADD IN MODULATION
        BRA    SND02      AND KEEP LOOPING
```

It is actually quite a bit more complex than this, but it gives you the idea of how sound is created. The variables BASE, DURATN, and MOD are constantly displayed on the screen, both in hexadecimal and in ASCII. You can use the following keys (with repeat) to change their values (initially set to 80 Hex).

↑ / ↓	— Increment/decrement the BASE VALUE variable.
SHIFT ↑ / ↓	— Rotate all variables upward/downward one position.
> / <	— Increment/decrement the DURATION variable.
→ / ←	— Increment/decrement the MODULATION variable.
SHIFT → / ←	— Increment/decrement ALL 3 variables.
SPACE BAR	— Complement (invert) all values.
BREAK	— Return to main menu.

NOTE: No sound (except a low-volume clicking) will occur while any key is depressed.