DISK DRIVE TESTER

Fine-tune your 5.25-inch drives

Disk drives are strange animals as computer hardware goes. In many systems, they contain the only moving parts, except for the printer. They are very reliable, especially when their hardware constraints are taken into consideration. But when they do fail, the results can be disastrous.

How does the disk drive work? How does software control it? And how can you control the disk drive in your own programs? Disk Drive Tester is a program that serves a useful function on its own, verifying the speed of a disk drive. By looking at how the program works, we can learn a lot about how the disk drive works. And the program will help you keep your drives in top shape.

Drive speed affects how closely together bits are written on the disk. If a drive's speed is abnormal, disks formatted on the eccentric drive may not be readable on normal drives. Also, a fair amount of copy-protected software will refuse to run on drives whose speed diverges too greatly from the norm. If your drive has trouble reading disks written to or formatted by other drives, or if it will not run protected software that runs fine on other drives, then it is a good candidate for a speed adjustment.

USING THE PROGRAM

Using Disk Drive Tester is simple. It is menu based, with one main menu and screen prompts to guide you through the process. Disk Drive Tester works only with 5.25-inch disk drives.

The main menu will prompt you to select an option. It includes the following options:

1. TEST DRIVE SPEED
2. FIND HIGHEST TRACK ACCESSIBLE
3. SELECT A NEW DRIVE TO BE TESTED
4. QUIT

Option 1 will test the drive's speed. You will be prompted to insert a blank disk. Then press Return to proceed or the Escape key to back out. After you press Return, the drive will recalibrate (noisily) and check the disk for data. If the disk you inserted has data on it, the program warns you and asks you to confirm that this disk should be used for the test. If you use a disk with data for this test, the data on track 0 will be destroyed!

After the test starts, the prompt PRESS < ESC > TO HALT TEST appears near the top of the screen. A window is drawn, in which the test results will be displayed. The drive will be continually tested until you press Escape.

Option 2 allows you to find the highest track accessible on your drive. DOS 3.3 and ProDOS both use tracks 0 through 34 of a disk, but Apple disk drives can access track 35 and some drives can access tracks 36-40 as well. Some older copy-protected software used track 35, and there are software patches to DOS that give you access to all 41 tracks if your drive can handle it.

Option 3 lets you reselect the slot and drive of the disk drive to be tested in option 1. The current drive's slot and drive will be offered as default values; press Return to accept these defaults.

Option 4 exits the program and returns you to BASIC. In ProDOS in particular, use this option to exit; it resets the HIMEM: pointer to its proper value.

Testing the Drive's Speed

The speed of your drive can vary. Normal drive speed is five rotations a second, or 300 RPM. It is recommended that the speed be within one or two rotations per minute, although in theory your drive probably can operate with a much greater variation. Copy-protected software, however, is often very picky about drive speed.

Adjusting the speed should be done by an authorized dealer, but it really is not all that difficult. As long as you are faint of heart or exceptionally wealthy, take your drive to your dealer. If you are bold or broke, you may want to try adjusting it yourself. As long

as you are methodical and cautious, all should go well.

To change the speed of a disk drive, disassembly is required, but this process is fairly basic. If you have a Disk II, turn off the power and remove the screws on the bottom of your drive. Slide the cover off toward the cable, and put it and the screws aside.

Now look at the far right corner for the drive speed potentiometer (pot). It is a gold colored disc about 3 mm in diameter with a notch for nonmetallic screwdrivers or thumbnails.

Once you have located the speed pot, made sure that all connections to the computer are still secure and no cables are loose, you can turn on the power. (Yes, that's right. Without the drive cover. Now is your big chance to watch the gizmos inside your drive go.) Run DRIVE.TESTER, and select the proper drive to test with option 3. Choose option 1 and insert a blank disk. Let the test run for a minute, so the drive's speed can stabilize. Now using a non-metallic screwdriver or a fingernail, turn the pot clockwise to increase the speed (make the RPM value higher), or counter-clockwise to decrease the speed. After adjusting, wait for the speed to stabilize and readjust if necessary. Continue until the speed is in the range 299-301 RPM or so. Press Escape to end the test and choose option 3 to test another drive or option 4 to quit. Carefully replace your drive's cover when you are finished.

If your drive has trouble reading disks written to or formatted by other drives, or if it will not run copy-protected software that runs fine on other drives, then it is a good candidate for a speed adjustment.

ENTERING THE PROGRAM

The Disk Drive Tester program has two parts: the Applesoft program and the machine language subroutines. The machine language routines do all the disk access work, since Applesoft is far too slow to access the disk.

Type in the Applesoft program in **Listing 1**, and save it to disk with the command

```
SAVE DRIVE.TESTER
```

Now type in the assembly language program in **Listing 2**; assemble the program and save the object code to disk as DRV.CONTROLLER. If you don't have an assembler, type in the machine language code in **Listing 3** and save it to disk with the command

```
BSAVE DRV.CONTROLLER,A$8000,L$1E1
```

Note: If you using an assembler, leave in the NOPs and seemingly useless instructions — they are time wasters, and timing is very important in disk writing routines.

The code in this program should be checked over very carefully; since it accesses the drive directly, anything can happen. Remember, blank disks should be used in the tests. If the test disks have data, they will be erased! When using the program, you should remove the program disk immediately after the main menu comes up. That way, you will lose no valuable data.

HOW THE PROGRAM WORKS

First of all, to ensure compatibility with both ProDOS and DOS 3.3, the machine language routines do not interface with either version of DOS through RWTS in DOS 3.3 or the ProDOS equivalent, the machine language interface (the MLI). Instead, they take direct control of the drive, and read and write raw data.

The operation of this program segment is not too complex. It writes more than 8000 $FFs onto a track and then a single $AA byte. Then it rereads the track, looking for a non-$FF byte. When that comes by, the program counts the $FFs going by until the non-$FF byte floats by again. This is the number of bytes written onto the track give or take one or two.

Now we take the number of bytes written onto the track and multiply it by 32, since it takes 32 cycles to write a byte. This value is divided by the speed of the microprocessor, 1.024 * 10 ^ 6 cycles / second, giving the number of seconds it takes to rotate completely. Now 60 seconds/minute is divided by this value, giving the number of rotations per minute.

THE INNER WORKINGS OF THE DISK DRIVE

This section is only for those who want to understand the disk drive more deeply. It outlines how data is read, written, and decoded. It also requires a good knowledge of machine language.

Disk drives have numerous limitations. Whereas they can write any data byte to disk, they cannot reliably read more than two consecutive zero bits because of drive speed fluctuation. (DOS 3.2 disk drives, which had a slightly different controller card, could reliably read only one zero bit before requiring a one bit.) This means that $A9, whose binary representation is 10101001, would be read correctly almost invariably, but $A8, 10101000, may not, since it ends with three consecutive zero bits.

In addition, bytes read must have their high bit set. Disk drives do not see bytes; instead, they see series of bits, and must figure out where one byte ends and another starts by using the high bit rule. Disk drives start out with their data buffer as zero, and collect bits from the disk, shifting the contents of the buffer to the left. For instance, the string of bytes $D5 AA 96 would be seen as

```
1101010110101010100101 10
```

Suppose the reading process started at the first bit. The data buffer would initially contain 00000000; after picking up the first bit it would hold 00000001, then 00000011, then 00000110, then 00001101, etc. When the high bit was set, the software driver would know that the data was valid, and use it. This bit collection process is done independently of software. If the drive is on, and in read mode, the data buffer is constantly being loaded with new bits.

To ensure that disk drives stay in synchronization with their data, before each new data sector, "self-sync" bytes are written. These are often described as ten-bit bytes, but they are just normal eight-bit bytes with two zero bits written after them. They are usually $FF. Their binary representation would look like this:

```
1111111100
```

With a series of five or more of these, the hardware is guaranteed to be in sync. Let's say that the series $FF FF FF FF FF D5 AA 96 was written to disk, with each $FF being a self-sync byte. The bit pattern would look like this:

```
1111111100 1111111100 1111111100 1111111100
1111111100 11010101 10101010 10010110
```

(The spaces are just for easier reading; they do not really exist on disk.) If the data were read starting with the first bit, the data buffer (called the "data latch") would hold $FF after shifting all eight bits. The two zero bits would then be shifted in, but the high bit of the data latch would not be set until the next eight bits of the next $FF byte were loaded in. So once the hardware is in sync, it stays in sync.

If the reading started on the second bit of the first byte, the first byte read would look like 11111110. The second byte would be in sync, however, since the last zero of the first byte would get shifted into the data latch, but the next byte would have to be completely shifted into it before the high bit was set. After this, the hardware would again be in sync.

If reading started at the last bit in the first byte, the first byte read would be 10011111. The second byte would be 11100111. The third would be 11111001. The fourth would be 11111110. The fifth would be read in sync, as 11111111, since the first zero would be ignored.

If you understand this, you understand one of the most complex, and perhaps one of the most clever procedures that goes on in your Apple. For further clarification, see either *Beneath Apple DOS* or *Beneath Apple ProDOS*, both by Don Worth and Pieter Lechner and published by Quality Software.

Using the Disk Drive's I/O Locations

Every card plugged into a peripheral slot has sixteen I/O locations, which it may use for whatever purpose it wants. The disk drive controller card reserves these locations for telling the card what the software wants to do to the disk drives. If the card is in slot 6, the locations range from $C0E0 to $C0EF. Basically, the range is $C080+*s* to $C08F+*s*, where *s* is the slot number times 16. These are soft switches, much like those that control graphics.

Each even I/O location is paired off with the following odd I/O location to control one switch on the controller card. The odd location turns a switch on; the even location turns that switch off. The switches are known as Q*x*, where the *x* represents the switch number, from 0 to 7. Table 1 lists these switches, their locations, and their functions.

Table 1: Drive Controller I/O Locations (Slot 6)

Lines	Description	
$C0E0	Q0 off	
$C0E1	Q0 on	Stepper motor Phase 0
$C0E2	Q1 off	
$C0E3	Q1 on	Stepper Motor Phase 1
$C0E4	Q2 off	
$C0E5	Q2 on	Stepper Motor Phase 2
$C0E6	Q3 off	
$C0E7	Q3 on	Stepper Motor Phase 3
$C0E8	Q4 off	Drive off
$C0E9	Q4 on	Drive on
$C0EA	Q5 off	Drive one select
$C0EB	Q5 on	Drive two select
$C0EC	Q6 off	(various)
$C0ED	Q6 on	(various)
$C0EE	Q7 off	(various)
$C0EF	Q7 on	(various)

To turn on a switch, access the memory location turning on that switch. Like many other I/O locations, any access will do: STA, LDA, BIT, INC, and so on. For instance, to turn the drive on, STA $C0E9 would do the trick (assuming slot 6).

Q0 through Q3 turn on or off stepper motors, or phases, also numbered 0 to 3. These motors move the disk arm. The arm-moving routines must either note the arm's current position or recalibrate the disk arm back to track zero, as the hardware has no idea where the disk arm is. Each phase moves the arm one-half track, so the drive must stop on an even phase to be on a full track (only full tracks are used in DOS and ProDOS).

To move the arm, each of the four phase motors must be turned on, then off again. If this phase switching is done in descending order, the arm moves inwards, toward track 35. If this switching is done in ascending order, the arm moves toward track 0 (or, if the arm is on track 0, you hear the grinding noise of recalibration, normally heard when initializing disks. After a recalibration, the arm is always on track 0). After each motor is turned on, there must be some delay before it is turned off; the actual delay depends on your drive and the track number, but 20 milliseconds should be enough for any drive. (You may want to experiment with different values.) The routine in the drive tester operates in half tracks (so a $22 must be loaded to go to track $11), with the delay routine borrowed (and slightly modified) from ProDOS.

Normal drive speed is 300 RPM. Your drive's speed should be within one or two revolutions per minute of that, although in theory your drive can probably operate with a greater variation.

If all the phases are not off prior to calling, the arm move routine may not function properly. For instance, if phase 0 is on, and phase 1 is turned on, the drive arm will not move (mine won't, anyway). Therefore, it is good practice to make sure your arm move routine leaves no phase switches on. Warning: the $C600 boot ROM leaves phase 0 on. If you are writing a boot routine, make sure that the arm move routine turns it off (by accessing $C080+*slot**16). If you do not, the arm may not move to the correct track!

Q4 is simple enough. When Q4 is on, the selected drive goes on. When Q4 is turned off, the selected drive turns off, within about a second's delay. This delay speeds up other disk accesses within that time by making it unnecessary for the drive to warm up. *Note:* It is not necessary for the drive to be on to select read or write mode or to select the drive. In fact, it is recommended that you select the correct drive and mode before turning on the drive. However, to read bytes from the disk, move the arm, or sense a write protect tab, the drive must be spinning. Also, the drive takes a small amount of time to warm up — by watching the data latch, and waiting for it to change, this delay can be reduced.

When Q5 is on, drive two becomes the selected drive; it goes on if Q4 is on, and all reading and writing becomes directed to that drive. If Q5 is off, drive one is the selected drive.

Q6 and Q7 are difficult to explain in just a few words. These two switches control whether the drive is reading, writing, or sensing a write protect. When both Q6 and Q7 are off, the drive selected is in read mode, and constantly shifts bits from the disk into the data latch, accessed here through $C0EC. A sample read routine would be

```
LDA  $C0EE  ;to ensure read mode
            ;other code if desired
READ LDA $C0EC
     BPL READ
```

The BPL ensures that the data is valid by waiting for a high bit to be shifted into the data latch. Each byte should be gotten before 32 cycles have elapsed; otherwise, bytes may be missed.

When Q6 is on and Q7 is off, the data latch holds either $FF if the drive is write protected, or a value less than $80 if not. A sample write protect test is

```
CHECKTAB LDA $C0EE
         LDA $C0ED
         BMI WRTPRTCT
```

where WRTPRTCT is the write protect handler. Oddly, when the phase 1 stepper motor is on, the disk is as good as write protected. Therefore, it should be stressed that turning off the phase switches is not only good practice, but necessary when writing data.

When Q7 is on, the drive is in write mode with no regard to the state of Q6. However, Q6 is used as a flag to indicate that more

data should be loaded into the data latch. So in the write routine, the data is stored in $C0ED in 32-cycle loops, and your program must access $C0EC to indicate that no more data should be put into the latch. An example of a write a byte routine is

```
LDA #DATA
STA $C0ED
CMP $C0EC
```

Again, the writing must be done in 32-cycle loops (from one STA $C0ED to the next); otherwise, excess zeros will be written out. If self-sync bytes are to be written, use 36-cycle loops or 40-cycle loops after the self-sync bytes have been stored.

The drive should always be tested for write-protection before your program tries to write. This not only prevents a waste of time and effort (by trying to write to a disk that cannot accept it), but is required to synchronize the program on the controller card (known as the logic state sequencer) with the software. After checking for a write protect, the first byte can be written at any time.

The first byte is written in a different way from the rest; it is stored into $C0EF (which also starts the write cycle); after that, your routine should access $C0EC. The following bytes should be stored into $C0ED, followed by an access to $C0EC. This is because any location between $C0E0 and $C0EF will put data into or take data out of the data latch. But by storing the data into $C0ED, you're killing two birds with one stone by storing away the data and turning on Q6, indicating to the logic state sequencer that another byte is ready to be shifted. Q6 can be turned off right away, because a maximum of four 6502 cycles are require to transfer the data, which is how long a store instruction (STA) takes anyway.

That is basically how the drive is used at the lowest possible level. If you find these processes interesting, refer to *Beneath Apple DOS* or *Beneath Apple ProDOS* for a more in-depth discussion.

LISTING 1: DISK DRIVE.TESTER

```
37 | 10  REM **********************
C0 | 20  REM * DISK DRIVE TESTER  *
B9 | 30  REM *  BY RICHARD KISS   *
AE | 40  REM * COPYRIGHT (C) 1988 *
CB | 50  REM * BY MICROSPARC, INC *
24 | 60  REM * CONCORD, MA. 01742 *
45 | 70  REM **********************
A2 | 80  NOTRACE : TEXT : NORMAL : SPEED= 255:S = 6:
       D = 1:PD =  PEEK (48896) = 76
CE | 90  HOME : PRINT  CHR$ (21): PRINT : ONERR  GOT
       O 970
CD | 100  PRINT  TAB( 12)"DISK DRIVE TESTER"
2C | 110  PRINT  TAB( 13)"BY RICHARD KISS"
D9 | 120  PRINT  TAB( 11)"COPYRIGHT (C), 1988"
1F | 130  PRINT  TAB( 11)"BY MICROSPARC, INC."
FB | 140  COLOR= 10: HLIN 0,39 AT 0: VLIN 0,12 AT 0:
       VLIN 0,12 AT 39: HLIN 0,39 AT 14: POKE 34,
       9
72 | 150  IF  PEEK (32768) <  > 76 OR  PEEK (32769)
       <  > 104 OR  PEEK (32770) <  > 128 THEN
       HOME : PRINT  CHR$ (4)"BLOADDRV.CONTROLLER
       ": HIMEM: 32768 - 1024 * PD
6D | 160  HOME
BA | 170  PRINT "SELECT AN OPTION:" TAB( 25)"(SLOT "
       S" DRIVE "D")"
A3 | 180  PRINT "1) TEST DRIVE SPEED"
5E | 190  PRINT "2) FIND HIGHEST TRACK ACCESSIBLE"
14 | 200  PRINT "3) SELECT A NEW DRIVE TO BE TESTED"

7E | 210  PRINT "4) QUIT"
73 | 220  PRINT : PRINT "YOUR OPTION-->";
1B | 230  GET O$: IF O$ < "1" OR O$ > "4" THEN 230
97 | 240  PRINT O$: IF O$ = "4" THEN 480
AA | 250  HOME
7F | 260  IF O$ = "3" THEN 810
5A | 270  IF O$ = "2" THEN  PRINT  TAB( 7)"FIND HIGH
       EST ACCESSIBLE TRACK": GOTO 290
D3 | 280  PRINT  TAB( 12)"TEST DRIVE SPEED"
1D | 290  PRINT : HTAB 2: INVERSE : PRINT "WARNING";
       : NORMAL : PRINT ":THIS OPTION ERASES THE D
       ISK!"
45 | 300  PRINT : PRINT  TAB( 5)"INSERT A ";: INVERS
       E : PRINT "BLANK";: NORMAL : PRINT " DISK I
       N SLOT "S","
98 | 310  PRINT "DRIVE "D", AND PRESS <RETURN> TO PR
       OCEED."
9B | 320  PRINT : PRINT  TAB( 9)"PRESS <ESC> TO BACK
       OUT."
A0 | 330  POKE 49168,0
9B | 340  IF  PEEK (49152) < 141 THEN 340
67 | 350 X =  PEEK (49152): POKE 49168,0: IF X = 155
       THEN 160
9D | 360  IF X <  > 141 THEN 340
A8 | 370  POKE 0,16 * S: POKE 1,D: CALL 32768
FF | 380  IF  PEEK (2) = 0 THEN 460
84 | 390  VTAB 20: INVERSE : PRINT "WARNING!";:
       NORMAL
FE | 400  PRINT " THIS DISK HAS DATA ON IT! TYPE"
EE | 410  PRINT "<RETURN> TO CONFIRM TEST, <ESC> TO
       ABORT";: CALL  - 198: CALL  - 198
4E | 420  POKE 49168,0
EE | 430  IF  PEEK (49152) < 141 THEN 430
F8 | 440 X =  PEEK (49152): POKE 49168,0: IF X = 155
       THEN 160
FF | 450  IF X <  > 141 THEN 420
1A | 460  IF O$ = "1" THEN 500
C2 | 470  GOTO 680
AD | 480  TEXT : IF PD THEN  CALL 48888
F1 | 490  HOME : VTAB 23: END
64 | 500  HOME
BC | 510  PRINT  TAB( 5)"PRESS <ESC> TO HALT THE TES
       TING"
9B | 520  HLIN 13,26 AT 22
84 | 530  VLIN 22,38 AT 13: VLIN 22,38 AT 26
B8 | 540  HLIN 13,26 AT 38
42 | 550  POKE 32,15: POKE 33,11: POKE 34,12: POKE 3
       5,19: HOME : FOR I = 1 TO 10: PRINT : NEXT

63 | 560  CALL 32771: IF  PEEK (4) = 0 THEN 660
1A | 570 B =  PEEK (3) +  PEEK (4) * 256
98 | 580 P = 1920000 / B
FC | 590 SP =  INT (P * 10 + .5) / 10
41 | 600  PRINT : PRINT SP TAB( 8)"RPM";
91 | 610  IF  PEEK (49152) <  > 155 THEN 560
5B | 620  TEXT : POKE 34,9
B2 | 630  VTAB 24: HTAB 4: PRINT "PRESS <RETURN> FOR
       THE MAIN MENU";
36 | 640  IF  PEEK (49152) <  > 141 THEN 640
FF | 650  POKE 49168,0: GOTO 160
2A | 660  TEXT : POKE 34,9: HTAB 1
FA | 670  VTAB 22: PRINT "  DRIVE ERROR -- CHECK DIS
       K AND DRIVE";: CALL  - 198: CALL  - 198:
       GOTO 630
C5 | 680  VTAB 11
9A | 690  CALL  - 958
94 | 700 T = 30: POKE 8,T: CALL 32774
2F | 710 T = T + 1: POKE 8,T
70 | 720  GOSUB 780: VTAB 14: HTAB 9: PRINT "TESTING
       TRACK "T" ($"A$")"
8F | 730  CALL 32777
16 | 740  IF  PEEK (3) = 0 THEN 670
8D | 750  IF  PEEK (3) = T - 1 THEN 710
85 | 760 T = T - 1: GOSUB 780: PRINT : PRINT T" ($"A
       $") IS THE HIGHEST ACCESSIBLE TRACK"
D8 | 770  GOTO 630
99 | 780 A$ = "":Z% = T / 16: GOSUB 800
```

LISTING 1: DISK DRIVE.TESTER *continued*

```
E5  |  790 Z% = T - Z% * 16
E1  |  800 A$ = A$ +  CHR$ (48 + Z% + 7 * (Z% > 9)):
        RETURN
69  |  810  PRINT "WHERE IS THE DRIVE TO BE TESTED?":
        PRINT :DS = S:DD = D
A1  |  820  PRINT  SPC( 12)"SLOT - "DS CHR$ (8);
3B  |  830  GET A$: IF A$ =  CHR$ (27) THEN 160
DF  |  840  IF A$ =  CHR$ (13) THEN A$ =  STR$ (DS)
D3  |  850  IF A$ < "1" OR A$ > "7" THEN 830
30  |  860 A =  VAL (A$): IF  PEEK (49153 + A * 256)
        < > 32 THEN  HTAB 1: PRINT "NO DISK DEVIC
        E CONNECTED TO SLOT "A"." CHR$ (7): PRINT "
        PRESS RETURN TO CONTINUE ":: POKE  - 16368,
        0: GET Q$: FOR I = 1 TO 1000: NEXT : HTAB 1
        : CALL  - 868:CV =  PEEK (37): VTAB CV: HTA
        B 1: CALL  - 868: GOTO 820
23  |  870 DS = A: PRINT DS
86  |  880  PRINT  SPC( 11)"DRIVE - "DD CHR$ (8);
FA  |  890  GET A$: IF A$ =  CHR$ (27) THEN 160
5E  |  900  IF A$ =  CHR$ (13) THEN A$ =  STR$ (DD)
FA  |  910  IF A$ < "1" OR A$ > "2" THEN 890
A0  |  920 DD =  VAL (A$): PRINT DD
7B  |  930  PRINT : PRINT "IS THIS CORRECT? (Y/N) ";
AD  |  940  GET A$: IF A$ < > "Y" AND A$ < > "N" AND
        A$ < > "n" AND A$ < > "y" THEN 940
C1  |  950  PRINT A$: IF A$ = "Y" OR A$ = "y" THEN S =
        DS:D = DD: GOTO 160
F1  |  960  HOME : GOTO 810
69  |  970 EN =  PEEK (222):EL =  PEEK (218) + 256 *
        PEEK (219): POKE 216,0
82  |  980  IF EN = 6 THEN CV =  PEEK (37):CH =  PEEK
        (36): VTAB 22: HTAB 1: PRINT "DRV.CONTROLLE
        R FILE NOT FOUND": PRINT "PRESS RETURN TO C
        ONTINUE ":: POKE 16368,0: GET A$: POKE 35,2
        0: HOME : POKE 35,23: VTAB CV + 1: HTAB CH
        + 1: GOTO 90
8A  |  990  TEXT : HOME : VTAB 23: PRINT "ERROR #";EN;
        " AT LINE ";EL;: END
```

TOTAL: 511A

END OF LISTING 1

LISTING 2: DRV.CONTROLLER Source Code

```
 1 ********************************
 2 *                              *
 3 * DRV.CONTROLLER Source Code   *
 4 * For use with DRIVE.TESTER    *
 5 * By Richard Kiss              *
 6 * Copyright (C) 1988           *
 7 * By MicroSparc Inc.           *
 8 * Concord, MA 01742            *
 9 *                              *
10 ********************************
11
12          ORG   $8000
13
14 SLOT     =     $0
15 DRIVE    =     $1
16 COUNT    =     $2
17 TRKLENLO =     $3
18 TRKLENHI =     $4
19 CTRACK   =     $5          ;For MOVEARM
20 DTRACK   =     $6
21 TIMELEFT =     $7
22 TESTTRAK =     $8
23
24 WAIT     =     $FCA8
25
26          JMP   TEST
27          JMP   WRITE
28          JMP   WONEPAT
29          JMP   VALID?
30
31 WONEPAT  JSR   WPAT
32          JMP   OFF
33
34 ON       LDX   SLOT        ;Get slot
35          STA   $C08E,X     ;READ mode
36          LDA   DRIVE
37          ORA   SLOT
38          TAX
39          STA   $C089,X     ;Select correct drive
40          LDX   SLOT
41          STA   $C089,X     ;Turn on drive
42          LDY   #7
43          STY   COUNT
44 ON?      LDA   $C08C,X     ;Wait for
45          CMP   $C08C,X     ;data to change
46          BEQ   GONE
47          DEY               ;or for
48          BNE   ON?         ;$707 bytes
49          DEC   COUNT       ;to go by
50          BNE   ON?
51 GONE     RTS
52
53 *
54 * This routine recalibrates
55 * (grinds) the track arm head
56 * to track 0.  Without
57 * recalibration, we could never
58 * really be sure what track we
59 * were on.
60 *
61
62 RECAL    JSR   ON
63          LDA   #95         ;Make MOVEARM think
64          STA   CTRACK      ;we're on track 95
65          LDA   #0          ;Go to track 0
66
67 MOVEARM  STA   DTRACK      ;Go to track A
68 MOVE1    LDA   CTRACK
69          CMP   DTRACK
70          BNE   WWAY?       ;Which Way?
71          LDX   SLOT        ;We are there!
72          RTS
73 WWAY?    BGE   DEC
74          INC   CTRACK      ;INC twice since
75          INC   CTRACK      ;a DEC follows
76 DEC      DEC   CTRACK
77          LDA   CTRACK
78          AND   #3          ;Get correct
79          ASL               ;phase number
80          ORA   SLOT
81          TAX
82          LDA   $C081,X     ;Turn on stepper motor
83          JSR   DELAY       ;Wait a while
84          LDA   $C080,X     ;Turn it off
85          JMP   MOVE1
86
87 *
88 * This routine checks for
89 * a D5 AA AD header on the disk.
90 * This header usually means
91 * 16-sector data, so give a
92 * warning before erasing it.
93 *
94 * A 00 in COUNT means
95 * that no header was found.
96 *
97
98 TEST     JSR   RECAL
99          LDY   #$18
100         STY   COUNT
101 READ1   JSR   READ        ;Get one byte
102 D5?     CMP   #$D5
103         BNE   NOTSEQ      ;Not the sequence
104         JSR   READ
105         CMP   #$AA
106         BNE   D5?
107         JSR   READ
108         CMP   #$AD
109         BNE   D5?
110         BEQ   OFF
111 NOTSEQ  DEY
112         BNE   READ1
113         DEC   COUNT
114         BNE   READ1
115 OFF     STA   $C08E,X     ;Set read mode (just in case)
116         STA   $C088,X     ;Turn off drive
117         RTS
118
119 READ    LDA   $C08C,X     ;Check data latch
120         BPL   READ        ;Wait for hi bit
121         RTS
122
123 WRITE   JSR   ON          ;Drive ON!
124         LDA   #80         ;Wait a while
125         JSR   WAIT
126         LDY   #$30        ;Write
127         STY   COUNT       ;a whole bunch
128         LDA   #0          ;of $FFs
129         STA   TRKLENHI
130         LDA   $C08D,X     ;Check write protect switch
131         LDA   $C08E,X
132         BMI   OFF         ;Yes, write protected
133         LDA   #$FF
134         STA   $C08F,X     ;Turn on write mode
135         CMP   $C08C,X     ;Write first byte
136         PHA               ;Wait 31 more cycles
137         PLA
138         JSR   GONE
139         JSR   GONE
140 WRITLOOP STA  $C08D,X     ;5 cycles
141         CMP   $C08C,X     ;+4=9 cycles
142         DEY               ;+2=11
143         BNE   OFFSET1     ;=13 OR 14
144         DEC   COUNT       ;+5=18
145         JMP   OFFSET2     ;+3=21
146 OFFSET1 BIT   COUNT       ;+3=17
147         NOP               ;+2=19
148         NOP               ;+2=21
149 OFFSET2 NOP               ;+2=23
150         NOP               ;+2=25
151         NOP               ;+2=27
152         BEQ   WRITEAA     ;=29 OR 30
153         BNE   WRITLOOP    ;+3=32
154
155 WRITEAA  LDA   #$AA        ;+2=32
156
157         STA   $C08D,X     ;=5
158         CMP   $C08C,X     ;+4=9
159         JSR   GONE        ;+12=21
160         PHA               ;+3=24
161         PLA               ;+4=28
162         NOP               ;+2=30
163         CLC               ;+2=32
164         LDA   $C08E,X     ;Turn off
165         LDA   $C08C,X     ;write mode
166         LDY   #0
167 FINDNON JSR   READ        ;Look for something
168         CMP   #$FF        ;besides an $FF
```

```
169       BEQ   FINDNON
170       INY
171 FINDANFF JSR   READ        ;Wait for the $FFs
172       CMP   #$FF          ;to start again
173       BNE   FINDANFF
174 FINDANON JSR   READ        ;Wait for another non-$FF
175       INY                 ;Hopefully, it will
176       BNE   NOINC         ;be the same one
177       INC   TRKLENH+1
178 NOINC CMP   #$FF
179       BEQ   FINDANON
180       STY   TRKLENLO      ;Got track length!
181       JMP   OFF           ;Turn off drive
182
183 *
184 * This routine is used
185 * to test for a track's
186 * validity.  It writes a pattern
187 * on the current track, and
188 * then tests the previous track
189 * for that pattern.  If it has
190 * it too, then this track
191 * cannot be accessed.
192 *
193
194 VALID?  JSR   ON            ;Turn on drive
195       JSR   MPAT          ;Write the pattern
196       JSR   RECAL         ;Recalibrate drive
197       LDA   TESTTRAK      ;for position verification
198       ASL
199       TAY
200       DEY
201       DEY
202       TYA
203       JSR   MOVEARM       ;Go to the previous track
204       LDY   #0            ;and look for
205       STY   TRKLENLO      ;the pattern
206 EH?   JSR   READ
207       CMP   #$DA          ;Prelude
208       BEQ   GET'RAK
209       DEY
210       BNE   EH?
211       JMP   OFF           ;Nothing here!
212 GET'RAK JSR   READ
213
214       ROL                 ;Decode encoded track value
215       STA   TRKLENHI
216       JSR   READ
217       AND   TRKLENHI
218       STA   TRKLENHI
219       JSR   READ
220       CMP   #$DD          ;Epilogue byte
221       BNE   EH?
222       LDA   TRKLENHI
223       STA   TRKLENLO
224 TOOFF   JMP   OFF           ;Turn off drive
225
226 *
227 * This routine writes the pattern
228 * on the specified track.
229 * It uses a prelude byte of $DA,
230 * the track encoded in 4 & 4
231 * (shifted, ORed with $AA,
232 * then ORed without shifting)
233 * followed by an epilogue byte
234 * of $DD.
235 *
236
237 MPAT    JSR   ON            ;Turn on the drive
238       LDA   TESTTRAK      ;Move arm to
239       ASL                 ;the test track
240       JSR   MOVEARM
241       LDA   #80
242       JSR   WAIT          ;Wait a while
243       LDA   TESTTRAK      ;Do the shifting now
244       LSR
245       ORA   #$AA
246       STA   TRKLENHI
247       ORA   #$AA          ;2
248       STA   TRKLENHI      ;2
249       LDA   #$A           ;2
250       STA   TIMELEFT      ;Write the series
251       LDA   $C08D,X       ;$A80 times
252       LDA   $C08E,X
253       BMI   TOOFF         ;Write-protected
254       STA   $C08F,X       ;Turn on write mode
255       NOP                 ;+2=7 (To make STA $C08D,X)
256       LDY   COUNT         ;+3=10 (a multiple of 4)
257 PUTLOOP LDA   #$DA          ;2
258       STA   $C08D,X       ;+5=7
259       CMP   $C08C,X       ;+4=11
260       LDA   TRKLENLO      ;+3=14
261       JSR   GONE          ;+12=26
262       JSR   GONE          ;+12=38=2
263       STA   $C08D,X       ;+5=7
264       CMP   $C08C,X       ;+4=11
265       LDA   TRKLENHI      ;+3=14
266       JSR   GONE          ;+12=26
267       JSR   GONE          ;+12=38=2
268       STA   $C08D,X       ;+5=7
269       CMP   $C08C,X       ;+4=11
270       LDA   #$DD          ;+2=13
271       JSR   GONE          ;+12=25
272       PHA                 ;+3=28
273       PLA                 ;+4=32
274       NOP                 ;+2=34
275       NOP                 ;+2=36
276       NOP                 ;2
277       STA   $C08D,X       ;+5=7
278       CMP   $C08C,X       ;+4=11
```

LISTING 2: DRV.CONTROLLER

Source Code *continued*

```
281       DEY                 ;+2=13
282       BNE   WAIT1         ;+15 OR 16
283       DEC   TIMELEFT      ;+5=20
284       BPL   WAIT2         ;+2? OR 23
285       JSR   GONE          ;+12=34     a
286       NOP                 ;+2=36
287       STA   $C08E,X       ;Turn off
288       STA   $C08C,X       ;write mode
289       RTS                 ;and go
290 WAIT1   JSR   GONE          ;+12=28
291       BIT   COUNT         ;+3=31
292       NOP                 ;+2=33
293       JMP   PUTLOOP       ;+3=36
294 WAIT2   INC   COUNT         ;+5=28
295       DEC   COUNT         ;+5=33
296       JMP   PUTLOOP       ;+3=36
297
298 *
299 * The scientifically calculated
300 * between-stepper-motor delay
301 * routine follows (I found it
302 * in ProDOS)
303 *
304
305 DELAY   LDA   #94
306 DELAY1  LDY   #23
307 DELAY2  DEY
308       BNE   DELAY2        ;Wait a bit
309       SEC   #1
310       SBC   DELAY1
311       RTS                 ;Fine, enough

END OF LISTING 2
```

LISTING 3: DRV.CONTROLLER

```
Start: 8000                   Length:1E1

3C 8000:4C 68 80 4C 9A 80 4C 0C
ED 8008:80 4C 12 81 20 52 81 4C
9D 8010:8D 80 A6 00 9D 8E C0 A5
02 8018:01 05 00 AA 9D 89 C0 A6
D5 8020:00 9D 89 C0 A0 07 84 02
C3 8028:8D BC C0 DD BC C0 F0 07
EB 8030:88 D0 F5 C6 02 D0 F1 60
F4 8038:20 12 80 A9 5F 85 05 A9
36 8040:00 85 06 A5 05 C5 06 60
54 8048:03 A6 00 60 BD 04 E6 05
4E 8050:E6 05 C6 05 A5 05 29 03
53 8058:0A 05 00 AA BD 81 C0 20
BB 8060:D5 81 BD 80 C0 4C 43 80
CB 8068:20 38 80 A0 18 84 02 20
7E 8070:94 80 C9 05 D0 10 20 94
B9 8078:80 C9 AA D0 F5 20 94 80
2C 8080:C9 AD D0 EE F0 07 85 08
CF 8088:E6 C6 02 D0 E2 9D 8E C0
49 8090:90 88 C0 60 BD 8C C0 10
AA 8098:FB 60 20 12 80 A0 50 20
9A 80A0:A8 FC A0 30 84 02 A9 00
69 80A8:85 04 BD 8D C0 BD 8E C0
5B 80B0:38 DB A9 FF 9D 8F C0 DD
4E 80B8:BC C0 48 68 88 D0 F2 80 20
96 80C0:37 80 9D 8D C0 DD BC C0
3A 80C8:88 D0 05 C6 02 4C D4 80
94 80D0:24 02 EA EA EA EA EA F0
9A 80D8:02 D0 E7 A9 AA 9D 8D C0
99 80E0:DD 8C C0 20 37 80 48 68
AF 80E8:EA 18 BD 8E C0 BD 8C C0
48 80F0:A8 00 20 94 80 C9 FF F0
0C 80F8:F9 C8 20 94 80 C9 FF D0
2C 8100:F9 20 94 80 C8 D0 02 E6
0B 8108:04 C9 FF F0 F4 84 03 4C
D3 8110:80 80 20 12 80 20 52 81
18 8118:20 38 80 A5 04 A0 AA 88
75 8120:88 98 20 41 80 A0 00 84
E3 8128:03 20 94 80 C9 DA F0 06
1C 8130:88 D0 F6 4C BD 80 20 94
90 8138:80 38 24 85 04 20 94 80
D9 8140:25 04 85 04 20 94 80 C9
1E 8148:DD D0 DE A5 04 85 03 4C
27 8150:80 80 20 12 80 A5 08 0A
33 8158:20 41 80 A9 50 20 A8 FC
30 8160:A5 08 4A 09 AA 85 03 A5
73 8168:08 09 AA 85 04 A9 0A 85
CD 8170:07 BD BD C0 BD 8E C0 30
61 8178:D6 9D 8F C0 EA 24 02 A9
5F 8180:DA 9D BD C0 DD 8C C0 A5
68 8188:03 20 37 80 20 37 80 9D
5C 8190:BD C0 DD 8C C0 A5 04 20
3C 8198:37 80 20 37 80 9D BD C0
67 81A0:DD 8C C0 A9 DD 20 37 80
09 81A8:4B 68 EA EA EA 9D BD C0
D6 81B0:DD 8C C0 88 D0 0F C6 07
57 81B8:10 14 20 37 80 EA 9D 8E
46 81C0:C0 9D 8C C0 60 20 37 80
71 81C8:24 02 EA 4C 7F B1 E6 02
66 81D0:C6 02 4C 7F B1 A9 5E A0
7D 81D8:17 88 D0 FD E9 01 D0 F7
95 81E0:60

TOTAL: E6F3

END OF LISTING 3
```