Microzine

Vol. I, No. 3

2015-06-06

Contents

0 In Which Various Automated Tools Fail In Interesting Ways 4

1 In Which We'll Take What We Can Get 9

2 In Which Things Get Brilliantly Weird 18

3 Every Byte Is Sacred, Every Byte Is Great, If A Byte Gets Wasted, Woz Gets Quite Irate 26

4 In Which We Can See The Light At The End Of The Tunnel And We Just Hope It's Not An Oncoming Train 30

5 In Which Simplicity Is In The Eye Of The Beholder 34

6 In Which It Doesn't Work 39

Name: Microzine vol. 1, no. 3
Genre: educational
Year: 1983
Publisher: Scholastic, Inc.
Media: double-sided 5.25-inch floppy
OS: DOS 3.3 with custom bootloader
Other versions: none (preserved here for the first time)
Similar cracks: Jumble Jet (no. 331)

Booting side B immediately displays a message telling me to boot side A. So I guess I'll, uh, do that.

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

What does the boot look and sound like?
  1. immediate blank screen
  2. several sequential track reads
  3. DOS prompt
  4. track seek (maybe to T11?)
  5. more disk activity (back and forth
     like file access)
  6. title screen

Does it access the disk after boot?
  Yes, repeatedly.

Does it have an option to read, write,
or format user-supplied data disks?
  Yes.

COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  unable to read any track

EDD 4 bit copy (no sync, no count)
  no read errors, but copy hangs after
    reading one track

Copy][+ nibble editor
  T00 -> standard prologues, modified
    epilogues (FF FF EB)
  T01 -> corrupted address fields,
    claim to be track $00
  T02..T03 -> not full tracks? looks
    like they have some standard-ish
    sectors, but not 16 per track
    (also corrupted address fields)
  T04..T22 -> standard prologues,
    modified epilogues (FF FF EB),
    standard address fields

```
                    --v--

TRACK: 01  START: 2735  LENGTH: 185A
       ^^

2710: FF FF FF FF FF FF FF FF   VIEW
2718: FF FF FF FF FF FF FF FF
2720: FF FF FF FF FF FF FF FF
2728: FF FF FF FF FF FF FF FF
2730: FF FF FF FF FF D5 AA 96  <-2735
                     ^^^^^^^^
                  address prologue

2738: AA AA AA AA AA AA AA AA
      ^^^^^ ^^^^^ ^^^^^ ^^^^^
      V000   T00   S00  chksm

2740: FF FF FF FF FC FF FF FF
      ^^^^^^^^
  address epilogue

2748: FF D5 AA AD F2 FA D7 D7
         ^^^^^^^^
      data prologue

2750: A6 BE FE F7 FB EC 97 B9

--------------------------------------

  A  TO ANALYZE DATA  ESC TO QUIT

  ?  FOR HELP SCREEN  /  CHANGE PARMS

  Q  FOR NEXT TRACK   SPACE TO RE-READ

                    --^--
```

The disk is lying to me. The address field claims to be track $00, but it's really track $01. Bad disk! Stop lying!

```
Disk Fixer
  ["O" -> "Input/Output Control"]
    set Address Epilogue to "FF FF EB"
    set Data Epilogue to "FF FF EB"
  T00 readable
  T01..T03 unreadable (no option to
    ignore the corrupted address field)
  T04..T22 readable
  T11 looks like DOS 3.3 catalog

Copy ][+ sector editor
  ["P" -> "Sector Editor Patcher"]
    set type to "CUSTOM"
    set Address Epilogue to "FF FF"
    set Data Epilogue to "FF FF EB"
  T00, T04..T22 readable

  ["P" -> "Sector Editor Patcher"]
    set CHECK TRACK to "NO"
  T01 readable!
  only parts of T02 and T03 readable:
    T02: S03,04,05,06,07,0A,0B,0C,0D,0E
    T03: S01,02,04,08,09,0C,0F

Why didn't COPYA work?
  modified epilogue bytes (every track)

Why didn't Locksmith FDB work?
  modified epilogue bytes (every track)

Why didn't my EDD copy work?
  I don't know. Maybe a nibble check
  during boot?
```

Next steps:

1. Super Demuffin to convert the tracks that have modified epilogue bytes but are otherwise normal, complete, and uncorrupted
2. Trace the boot
3. See what happens

Chapter 1
In Which We'll Take What We Can Get

When you first run Super Demuffin, it asks for the parameters of the original disk. In this case, the prologue bytes are the same, but the epilogues are "FF FF EB" instead of "DE AA EB".

--v--

SUPER-DEMUFFIN AND FAST COPY
Modified by: The Saltine/Coast to Coast

Address prologue: D5 AA 96

Address epilogue: FF FF EB DISK
                  ^^^^^ ORIGINAL
change from DE EA----+++++

Data prologue: D5 AA AD

Data epilogue: FF FF EB
               ^^^^^
change from DE AA----+++++

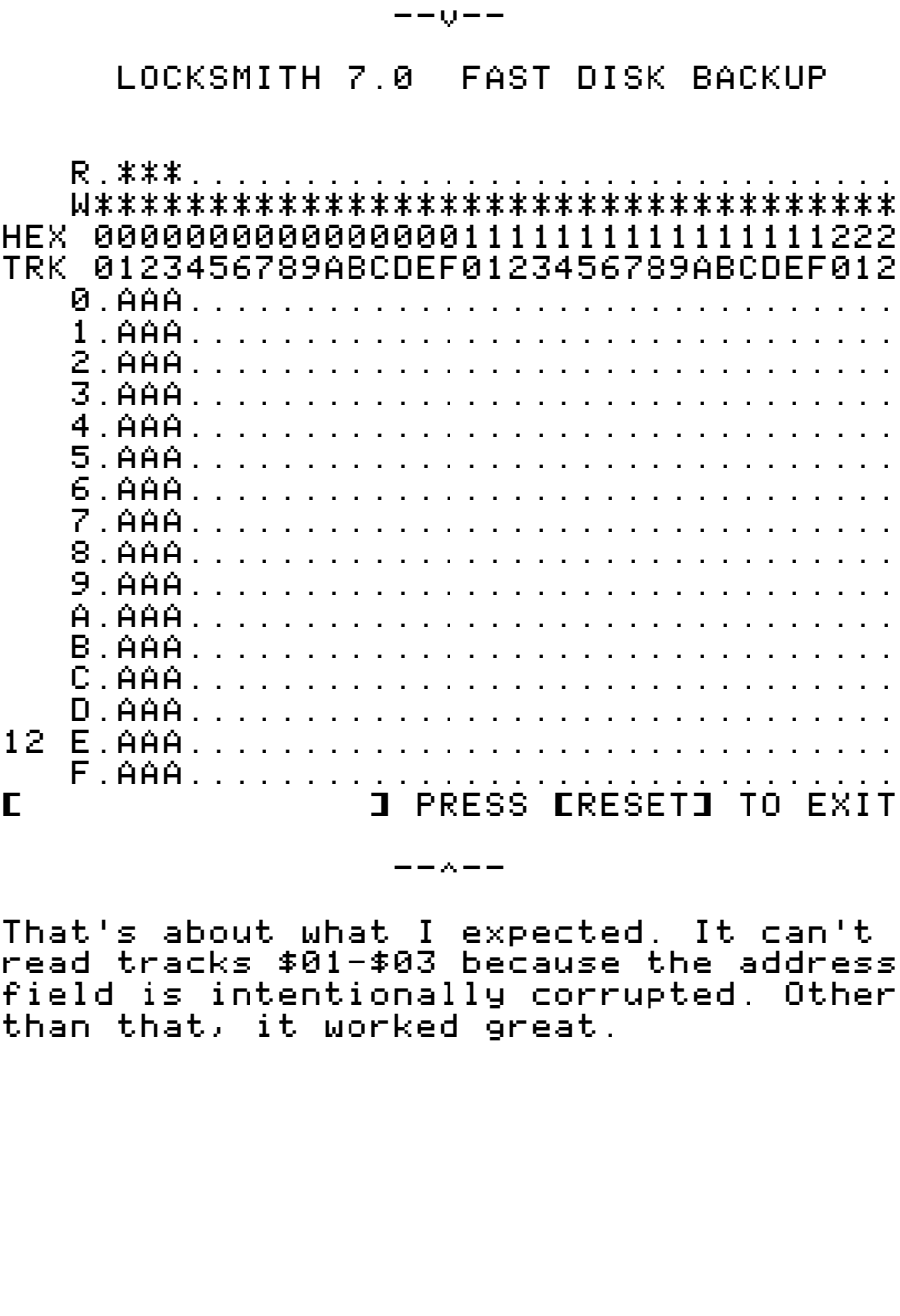
Ignore write errors while demuffining!

```
  D - Edit parameters
      <SPACE> - Advance to next parm
      <RETURN> - Exit edit mode
  R - Restore DOS 3.3 parameters
  O - Edit Original disk's parameters
  C - Edit Copy disk's parameters
  G - Begin demuffin process

                  --^--

Pressing "G" switches to the Locksmith
Fast Disk Copy UI. It assumes that both
disks are in slot 6, and that drive 1
is the original and drive 2 is the
copy.

[S6,D1=original disk]
[S6,D2=blank disk]
```



That's about what I expected. It can't read tracks $01-$03 because the address field is intentionally corrupted. Other than that, it worked great.

Let's go see what's on those unreadable tracks.

```
[S6,D1=original disk]
[S5,D1=my work disk]

]PR#5
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0

]CALL -151

*800<2800.28FFM

*801L

; set reset vector
0801-   8A           TXA
0802-   4A           LSR
0803-   4A           LSR
0804-   4A           LSR
0805-   4A           LSR
0806-   09 C0        ORA   #$C0
0808-   85 3F        STA   $3F
080A-   8D F3 03     STA   $03F3
080D-   49 A5        EOR   #$A5
080F-   8D F4 03     STA   $03F4
0812-   A9 00        LDA   #$00
0814-   8D F2 03     STA   $03F2

; hmm
0817-   A9 04        LDA   #$04
0819-   48           PHA
```

```
; machine initialization (memory banks,
; TEXT, IN#0, PR#0, &c.)
081A-   8D 81 C0     STA   $C081
081D-   20 2F FB     JSR   $FB2F
0820-   8D 52 C0     STA   $C052
0823-   20 89 FE     JSR   $FE89
0826-   20 93 FE     JSR   $FE93

; clear hi-res screen 1
0829-   A2 20        LDX   #$20
082B-   A0 00        LDY   #$00
082D-   84 06        STY   $06
082F-   A9 20        LDA   #$20
0831-   85 07        STA   $07
0833-   98           TYA
0834-   91 06        STA   ($06),Y
0836-   C8           INY
0837-   D0 FB        BNE   $0834
0839-   E6 07        INC   $07
083B-   CA           DEX
083C-   D0 F6        BNE   $0834

; switch to hi-res screen 1 (blank)
083E-   8D 57 C0     STA   $C057
0841-   8D 50 C0     STA   $C050
0844-   8D 54 C0     STA   $C054
0847-   8D 52 C0     STA   $C052

; set up ($3E) vector to point to the
; sector read routine in the disk
; controller ROM
084A-   A9 5C        LDA   #$5C
084C-   85 3E        STA   $3E

; the disk controller ROM always exits
; via $0801, so set that to an RTS so
; we can JSR and not have to set up a
; loop
084E-   A9 60        LDA   #$60
0850-   8D 01 08     STA   $0801
```

```
; hmm
0853-   A9 72        LDA   #$72
0855-   48           PHA

OK, we've now pushed $04/$72 on the
stack. That's probably important.

; multi-sector read
; Y = start logical sector ($01)
; X = end logical sector ($05)
; A = start address high byte ($9D)
0856-   A0 00        LDY   #$00
0858-   84 FC        STY   $FC
085A-   C8           INY
085B-   A9 9D        LDA   #$9D
085D-   A2 05        LDX   #$05

; multi-sector read routine
085F-   20 77 08     JSR   $0877

; another sector read, 9 more sectors
; ($06..$0E) into $6000..$68FF
0862-   A9 60        LDA   #$60
0864-   A2 0E        LDX   #$0E
0866-   20 77 08     JSR   $0877

; copy a few bytes manually
0869-   A2 07        LDX   #$07
086B-   BD A5 08     LDA   $08A5,X
086E-   9D 00 69     STA   $6900,X
0871-   CA           DEX
0872-   10 F7        BPL   $086B

; another sector read, this time just
; one sector, into $0400 (X is already
; less than Y on entry, so loop will
; exit after one read)
0874-   A9 04        LDA   #$04
0876-   AA           TAX
```

```
; falls through to multi-sector read
; entry point (was also called earlier
; from $085F and $0866)
0877-   85 27        STA   $27
0879-   E8           INX
087A-   86 49        STX   $49
087C-   84 F9        STY   $F9

; map logical into physical sector and
; store it in zero page where the disk
; controller ROM will look for it
087E-   B9 95 08     LDA   $0895,Y
0881-   85 3D        STA   $3D

; read sector via disk controller ROM
0883-   20 90 08     JSR   $0890

; loop until done
0886-   A4 F9        LDY   $F9
0888-   C8           INY
0889-   C4 49        CPY   $49
088B-   90 EF        BCC   $087C
088D-   A5 27        LDA   $27
088F-   60           RTS

0890-   A6 2B        LDX   $2B
0892-   6C 3E 00     JMP   ($003E)

0895-  [00 03 05 07 09 0B 0D 0F]
       [02 04 06 08 0A 0C 0E 01]
```

That's it. Flexible but compact.

It's a weird combination of reads, though. 9 pages at $6000. 5 pages at $9D00. 1 page at $0400 (part of the text page, but it's hidden during boot because we cleared the entire hi-res graphics page and showed that instead).

Of course, we manually pushed $04/$72 on the stack earlier, so once we fall through to the sector read routine and it hits the RTS at $088F, it will "return" to $0472 + 1 = $0473.

Let's interrupt the boot before it gets there.

Chapter 2
In Which Things Get Brilliantly Weird

```
*9600<C600.C6FFM

; set up callback by changing the two
; bytes that are pushed to the stack
96F8-   A9 97       LDA   #$97
96FA-   8D 18 08    STA   $0818
96FD-   A9 04       LDA   #$04
96FF-   8D 54 08    STA   $0854

; start the boot
9702-   4C 01 08    JMP   $0801

; callback is here -- copy $9D00 stuff
; to lower memory so it survives a
; reboot
9705-   A2 05       LDX   #$05
9707-   A0 00       LDY   #$00
9709-   B9 00 9D    LDA   $9D00,Y
970C-   99 00 1D    STA   $1D00,Y
970F-   C8          INY
9710-   D0 F7       BNE   $9709
9712-   EE 0B 97    INC   $970B
9715-   EE 0E 97    INC   $970E
9718-   CA          DEX
9719-   D0 EE       BNE   $9709

; copy code at $0400 to graphics page
; so it survives a reboot, too
971B-   B9 00 04    LDA   $0400,Y
971E-   99 00 24    STA   $2400,Y
9721-   C8          INY
9722-   D0 F7       BNE   $971B

; turn off slot 6 drive motor
9724-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9727-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE,A$9600,L$12A

*BRUN TRACE
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 9D00-A1FF,A$1D00,L$500
]BSAVE BOOT1 6000-6907,A$6000,L$908
]BSAVE BOOT1 0400-04FF,A$2400,L$100
]CALL -151
```

The entry point was $0473, so let's start there. I'll have to leave the code at $2400. Relative branches will look correct, but absolute addresses in $04xx will be off by $2000.

```
*2473L

; not sure what $4A is for yet
2473-   46 4A       LSR   $4A

; this decompresses the title screen
; to hi-res screen 2 (not shown)
2475-   20 00 60    JSR   $6000

; this checks for Applesoft in ROM and
; displays an error if it's not
2478-   20 00 A1    JSR   $A100

247B-   A9 A1       LDA   #$A1
247D-   20 0E 04    JSR   $040E

*240EL

240E-   20 33 04    JSR   $0433
```

```
*2433L

; call the following line (then fall
; through and do it again)
2433-   20 36 04    JSR   $0436

; save A and Y
2436-   48          PHA
2437-   98          TYA
2438-   48          PHA

; low-level disk stuff (see below)
2439-   A5 FC       LDA   $FC
243B-   85 FD       STA   $FD
243D-   E6 FC       INC   $FC
243F-   A5 FC       LDA   $FC
2441-   29 03       AND   #$03
2443-   0A          ASL
2444-   05 2B       ORA   $2B
2446-   A8          TAY
2447-   B9 81 C0    LDA   $C081,Y

; wait loop
244A-   A9 30       LDA   #$30
244C-   20 A8 FC    JSR   $FCA8

; more low-level disk stuff
244F-   A5 FD       LDA   $FD
2451-   29 03       AND   #$03
2453-   0A          ASL
2454-   05 2B       ORA   $2B
2456-   A8          TAY
2457-   B9 80 C0    LDA   $C080,Y

; more waiting
245A-   A9 30       LDA   #$30
245C-   20 A8 FC    JSR   $FCA8
```

```
; restore A and Y on the way out
245F-   68          PLA
2460-   A8          TAY
2461-   68          PLA
2462-   60          RTS
```

This is a very clever and compact way to advance the drive head to the next track. Normally DOS 3.3 keeps track of this and has a (much more complicated) routine to move the head back and forth as needed. But this loader only needs to move it forward, so the entire process collapses to this:

1. Set up the Y register to be a slot number (x16) plus the appropriate phase (0-3, depending on which track the drive head is on)

2. LDA $C081,Y to turn on the appropriate stepper motor

3. Wait exactly the right amount of time (as measured in CPU cycles)

4. LDA $C080,Y to turn off the appropriate stepper motor

5. Wait the right amount of time again

...which is exactly what this routine at $0436 is doing. But that only gets us halfway there -- literally, it only moves the drive head by half a track. But! Since $0433 "falls through" to $0436, it ends up doing this twice. Two half tracks equal one whole track, so calling the routine at $0433 will move the drive head to the next whole track.

(By the way, this is why it initialized zero page $FC to $00 at $0858. That's the "current" track where the drive head is at boot; it gets updated when the drive head advances.)

Everything I know about low-level disk stepping, I learned from this excellent Usenet post:
macgui.com/usenet/?group=1&id=31160

Continuing...

```
2411-   A2 0F        LDX   #$0F
2413-   A0 00        LDY   #$00

; store A in zero page $27, used by the
; disk controller ROM routine as the
; target page to store sectors read
; from disk
2415-   85 27        STA   $27

; X is the final sector to read
2417-   E8           INX
2418-   86 49        STX   $49

; Y is the current sector to read
; (starting with whatever was passed in
; and incrementing until it equals the
; value passed in the X register)
241A-   84 F9        STY   $F9
241C-   98           TYA
```

```
; But wait, there's more! Based on the
; high bit of zero page $4A, Y is
; either a logical sector (the map of
; logical->physical sectors is at
; $0263) or a physical sector
241D-   24 4A       BIT   $4A
241F-   30 03       BMI   $2424
2421-   B9 63 04    LDA   $0463,Y

; store physical sector in $3D (again,
; used by the disk controller ROM)
2424-   85 3D       STA   $3D

; read sector by jumping to ($003E),
; which points to $Cx5C (e.g. $C65C if
; booting from slot 6) and exit via
; $0801, which is an RTS by now, so
; this just continues to the next line
2426-   20 00 04    JSR   $0400

; increment sector index
2429-   A4 F9       LDY   $F9
242B-   C8          INY

; are there more sectors to read?
242C-   C4 49       CPY   $49

; yes, branch back and repeat
242E-   90 EA       BCC   $241A

; no, exit with last page (+1) in A
; (disk controller ROM increments this
; after storing sector data, so on exit
; this will be the first page that was
; NOT filled with data in this loop)
2430-   A5 27       LDA   $27
2432-   60         RTS
```

To sum up:

These two lines of code...

```
||  247B-   A9 A1       LDA   #$A1   ||
||  247D-   20 0E 04    JSR   $040E  ||
```

advanced the drive head from track $00 to track $01 and read the entire track into $A100..$B0FF, despite the fact that every sector's address field was corrupted and claimed to be track $00.

Beautiful.

Chapter 3
Every Byte Is Sacred,
Every Byte Is Great,
If A Byte Gets Wasted,
Woz Gets Quite Irate

```
2480-   20 9D 04    JSR   $049D

*249DL

; advance the drive head to track $02
249D-   20 33 04    JSR   $0433

; zero page fiddling
24A0-   A9 00       LDA   #$00
24A2-   85 41       STA   $41
24A4-   38          SEC
24A5-   66 4A       ROR   $4A

; call the multi-sector read routine
; again, but this time only read 5
; sectors, into $B100..$B5FF
24A7-   A9 B1       LDA   #$B1
24A9-   A0 01       LDY   #$01
24AB-   A2 05       LDX   #$05
24AD-   20 15 04    JSR   $0415

; move the drive head one phase only,
; to the next HALF track
24B0-   20 36 04    JSR   $0436

[now on track 2.5]

; read more sectors ($06..$0A) from
; track 2.5
24B3-   A2 0A       LDX   #$0A
24B5-   20 15 04    JSR   $0415

; advance another half track
24B8-   20 36 04    JSR   $0436

[now on track 3]
```

```
; read more sectors ($0B..$0F) from
; track 2
24BB-   A2 0F        LDX   #$0F
24BD-   20 15 04     JSR   $0415

; fiddle with $4A again
24C0-   46 4A        LSR   $4A
24C2-   60           RTS
```

So here's the deal with $4A: we initialized it at $0473 by a blind LSR, which clears the high bit. This tells the multi-sector read routine at $0415 to use logical sectors. Then we set the high bit at $04A4 with SEC + ROR, indicating we want $0415 to read physical sectors. Then we read a few sectors from track 2, a few from track 2.5, and a few from track 3. Then we reset $4A with another LSR, and we're back to using logical sectors.

This explains why my EDD bit copy failed. This disk is storing data on half tracks. Worse, it's storing data on *adjacent* half tracks -- a few from track 2, a few from track 2.5, and a few from track 3. Due to limitations of the Disk II drive mechanism, that would be virtually impossible for a generic bit copier to reproduce on a blank floppy disk.

Every part of this code is brilliant. AND it fits in a single sector in low memory. AND it's flexible enough to read from virtually uncopyable disks.

```
Continuing...

; now put slot number (x16) into...
; an RWTS parameter table?!?
2483-   A6 2B       LDX   $2B
2485-   8E E9 B7    STX   $B7E9

; set up DOS globals (tracking where
; the drive head is)
2488-   20 8E BE    JSR   $BE8E
248B-   A5 FC       LDA   $FC
248D-   99 78 04    STA   $0478,Y
2490-   4A          LSR
2491-   8D 78 04    STA   $0478

; push $B7/$3A on the stack
2494-   A9 B7       LDA   #$B7
2496-   48          PHA
2497-   A9 3A       LDA   #$3A
2499-   48          PHA

; and exit through HOME (which will
; wipe this loader from memory)
249A-   4C 58 FC    JMP   $FC58

Execution continues at $B73B (because
we just pushed $B7/$3A on the stack).
```

Chapter 4
In Which We Can See The Light At The End Of The Tunnel And We Just Hope It's Not An Oncoming Train

I can interrupt the boot by changing the values pushed on the stack at $0494 and $0497.

```
*9600<C600.C6FFM

; set up callback #1 after boot0 loads
; boot1 into $0400
96F8-   A9 97       LDA   #$97
96FA-   8D 18 08    STA   $0818
96FD-   A9 04       LDA   #$04
96FF-   8D 54 08    STA   $0854

; start the boot
9702-   4C 01 08    JMP   $0801

; callback #1 is here
; change final stack push to continue
; execution at my callback instead
9705-   A9 97       LDA   #$97
9707-   8D 95 04    STA   $0495
970A-   A9 11       LDA   #$11
970C-   8D 98 04    STA   $0498

; continue the boot
970F-   4C 73 04    JMP   $0473

; callback #2 is here
; copy everything that was loaded from
; the unreadable tracks to the graphics
; screen so it wil survive a reboot
9712-   A2 20       LDX   #$20
9714-   A0 00       LDY   #$00
9716-   B9 00 A0    LDA   $A000,Y
9719-   99 00 20    STA   $2000,Y
971C-   C8          INY
971D-   D0 F7       BNE   $9716
971F-   EE 18 97    INC   $9718
9722-   EE 1B 97    INC   $971B
9725-   CA          DEX
9726-   D0 EE       BNE   $9716
```

```
; turn off slot 6 drive motor
9728-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
972B-   4C 00 C5    JMP   $C500

*BSAVE TRACE2,A$9600,L$12E
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT2 A000-BFFF,A$2000,L$2000
]CALL -151

*3800L
.
. appears to be a DOS-shaped RWTS
.
3898-   A6 27       LDX   $27
389A-   20 BB B8    JSR   $B8BB
389D-   A9 FF       LDA   #$FF
389F-   20 B8 B8    JSR   $B8B8
38A2-   A9 FF       LDA   #$FF
38A4-   20 B8 B8    JSR   $B8B8
38A7-   A9 EB       LDA   #$EB
...
```

At this point, we have a full copy of DOS 3.3 in memory, albeit put there in the most roundabout way. Spot checking the RWTS, it's perfectly normal except it expects "FF FF EB" epilogue bytes. Which, by the way, is just the sort of RWTS that could read tracks $04-$22.

I'll need to patch it to read the standard epilogue instead of FF FF EB.

```
*389E:DE
*38A3:AA
*3935:DE
*393F:AA
*3991:DE
*399B:AA
*3CAE:DE
*3CB3:AA
*BSAVE RWTS FIXED,A$3800,L$800
```

Chapter 5
In Which Simplicity Is In The Eye Of The Beholder

```
Let's write $A000..$BFFF back to tracks
$01 and $02, but all regular and normal
and stuff. No half tracks, no spirals,
no tricks, no traps. Just, you know,
sectors on tracks on a disk.

[S6,D1=demuffin'd copy with T04-T22]
[S5,D1=my work disk]

]PR#5
...
]CALL -151

*300L

; page count (decremented)
0300-   A9 20       LDA   #$20
0302-   85 FF       STA   $FF

; logical sector (incremented)
0304-   A9 00       LDA   #$00
0306-   85 FE       STA   $FE

; call RWTS to write sector
0308-   A9 03       LDA   #$03
030A-   A0 88       LDY   #$88
030C-   20 D9 03    JSR   $03D9

; increment logical sector, wrap around
; from $0F to $00 and increment track
030F-   E6 FE       INC   $FE
0311-   A4 FE       LDY   $FE
0313-   C0 10       CPY   #$10
0315-   D0 07       BNE   $031E
0317-   A0 00       LDY   #$00
0319-   84 FE       STY   $FE
031B-   EE 8C 03    INC   $038C
```

```
; Convert to the interleave order that
; this disk expects
031E-   B9 40 03    LDA   $0340,Y
0321-   8D 8D 03    STA   $038D

; increment page to write
0324-   EE 91 03    INC   $0391
0327-   C6 FF       DEC   $FF

; loop until done with all pages
0329-   D0 DD       BNE   $0308
032B-   60          RTS

; sector interleave table
*340.34F

0340- 00 06 05 04 03 02 01 0F
0348- 0E 0D 0C 0B 0A 09 08 07

; RWTS parameter table, pre-initialized
; with slot 6, drive 1, track $01,
; sector $00, address $2000, and RWTS
; write command ($02)
*388.397

0388- 01 60 01 00 01 00 FB F7
0390- 00 20 00 00 02 00 00 60

*BSAVE MAKE,A$300,L$98
*BLOAD BOOT2 A000-BFFF,A$2000
*BLOAD RWTS FIXED,A$3800
*300G
```

Now I need to modify the bootloader at $0473 to read those pages from tracks $01 and $02. Specifically, I need to do three things:

1. Instead of reading track $01 into $A100..$B0FF, read tracks $01-$02 into $A000..$BFFF. This is just one JSR, because $040B is designed to read two tracks in a row. $040B literally calls $040E to advance the drive head and read a full track, then falls through to $040E to do it all again. HOW F---ING ELEGANT IS THAT.

2. Modify the routine that advances the drive head so it updates zero page $41 with the current track. The sector read routine at $C65C compares the track listed in the address field to zero page $41 and loops forever until it matches. $C600 initializes $41 to 0, and the original disk never updates $41, but everything works because the address fields are corrupted and all claim to be track 0. HOW F---ING ELEGANT IS THAT, AGAIN.

3. Skip all the spiral/half track stuff. I now have all the data on consecutive whole tracks.

So part of it will be simpler, because we'll no longer be spiraling between tracks. But part of it will actually be more complicated because the address fields are no longer corrupted. Weird.

```
T00,S07,$7C change "A1 20 0E 04 20"
                to "A0 20 0B 04 2C"

----------- DISASSEMBLY MODE ----------
007B:A9 A0           LDA   #$A0
007D:20 0B 04        JSR   $040B
0080:2C 9D 04        BIT   $049D

T00,S07,$9D change "20 33 04 A9 00"
                to "E6 41 4C 36 04"

----------- DISASSEMBLY MODE ----------
009D:E6 41           INC   $41
009F:4C 36 04        JMP   $0436

T00,S07,$34 change "36" to "9D"

----------- DISASSEMBLY MODE ----------
0033:20 9D 04        JSR   $049D
```

Chapter 6
In Which It Doesn't Work

]PR#6
...loads, displays main menu...

[select "Mystery at Pinecrest Manor"]
...works...

[select "Computer Stuff"]
[select "Initialize a data disk"]
...works...
[select "Return to Table of Contents"]
...grinds and displays error...

--v--

THERE IS SOMETHING WRONG WITH YOUR DISK

PRESS RETURN TO RESTART: _

--^--

I know I patched the second stage RWTS correctly, because there's plenty of disk activity after the initial boot. Entire programs, even. But after it initializes a data disk, the RWTS is being corrupted -- or reverted.

]PR#5
...

```
]CATALOG,S6,D1

C1983 DSR^C#254
009 FREE

 A 002 MICROZINE SIDE 1
 A 004 HELLO
 A 015 TABLE OF CONTENTS
 A 027 TWISTAPLOT
 A 049 TP.2
 A 010 CREDITS
 A 020 UTILITIES
 A 004 LOAD PIC
 A 003 TURN OFF
 B 002 ST.HAND
 B 020 HRCG
 B 010 PICDRAW
 B 002 RUNPACK
 B 033 GRAVE.PIC
 B 004 INIT.OBJ
 B 012 MZINE2.PAK
 B 015 UTILITIES.PAK
 T 002 MZ.PARAMETER FILE
 T 063 TWISTAPLOT FILE
 T 002 SIDE
 B 016 BOAT.SPC
 B 010 END.SPC
 B 006 FRANCES.SPC
 B 015 GANG.SPC
 B 007 KITCHEN.SPC
 B 016 LIBRARY.SPC
 B 005 LOIS.SPC
 B 004 JOEY.SPC
 B 004 MARIE.SPC
 B 004 MONTANA.SPC
 B 011 PALETNOTE.SPC
 B 006 RALPH.SPC
 B 011 SMITHNOTE.SPC
 B 008 STATUE.SPC
 B 025 TWISTAPLOT.SPC
[...]
```

```
 B 011 UNCLE.SPC
 B 007 WARDROBE.SPC
 B 005 F1.SPC
 B 002 F2.SPC
 B 004 N1.SPC
 B 004 N2.SPC

"UTILITIES" looks promising.

]LOAD UTILITIES
]LIST
.
.
.
 2000  REM
 2001 D = 2: IF ND = 1 THEN D = 1

 2005  PRINT HO$: VTAB 2: HTAB 10
     : PRINT "INITIALIZE A DATA D
     ISK"
 2006  VTAB 8: PRINT "For your da
     ta disk, you can use either
     a": PRINT "new, blank disk,
     or an old disk that you": PRINT
     "no longer need. If there is
      a write-": PRINT : PRINT "p
     rotect tab on the data disk,
      take it": PRINT : PRINT "of
     f now."
 2007  PRINT D$"BLOAD INIT.OBJ,D1
     ,A$9300"
 2009  GOSUB 900
 2010  PRINT HO$: VTAB 11: IF ND =
     1 THEN  HTAB 1: PRINT "Put d
     isk to be initialized in the
      drive.": GOTO 2020
 2015  HTAB 2: PRINT "Put disk to
      be initialized in Drive 2."
[...]
```

```
2020 MICRO = 0: VTAB 13: GOSUB 9
     10
2030  PRINT HO$: VTAB 11: PRINT
     "This will " CHR$ (7);BO$"ER
     ASE"UN$;" the disk now in dr
     ive ";BO$D;UN$".": PRINT "Do
      you want to go ahead? (";BO
     $"Y"UN$"/"BO$"N"UN$"): ";
2040  GOSUB 300:ESC = 0: IF CX$ <
      > "Y" THEN ESC = 1: GOSUB 4
     000
2050  PRINT HO$
2099  RETURN
3000  REM
3005 VN$ = "TABLE OF CONTENTS":D
     R% = D:QX% = 0
3006  GOSUB 61000: ONERR  GOTO 6
     2000
3008  IF  NOT DER% THEN  PRINT H
     O$: VTAB 12: HTAB 3: PRINT  CHR$
     (7);"You have the Microzine
     in the drive!": GOSUB 900: RETURN

3010  PRINT HO$: VTAB 12: HTAB 4
     : PRINT "Please wait. Initia
     lizing disk."
3015 ER = 0:ID% = 1
3017  CALL 46592
3020  CALL 37632,SL,D,ER
3024  CALL 46595
3030  PRINT HO$: IF  NOT ER THEN
      VTAB 12: HTAB 3: PRINT "The
      data disk has been initiali
     zed.": GOTO 3090
3040  VTAB 11: HTAB 2: PRINT "So
     mething is wrong. The data d
     isk has": PRINT : HTAB 5: PRINT
     "not been initialized. Try a
     gain."
[...]
```

```
 3090  GOSUB 900
 3095  IF D = 1 AND  NOT MI THEN
       GOSUB 4000
 3099  RETURN
```

Bingo! In particular, these three lines are calling binary routines:

```
 3017  CALL 46592
 3020  CALL 37632,SL,D,ER
 3024  CALL 46595
```

Converting to hex, that's calling

```
  $B600
  $9300
  $B603
```

Line 2007 loaded INIT.OBJ at $9300, so let's take a look at $B600. I have that on my work disk -- it was part of the $A000..$BFFF chunk that was originally on the unreadable spiral/half tracks.

```
]BLOAD BOOT2 A000-BFFF,A$2000,S5,D1
]CALL -151

*B600<3600.36FFM
*B600L

; branch to $B61D
B600-   18          CLC
B601-   90 1A       BCC   $B61D

; branch to $B623
B603-   18          CLC
B604-   90 1D       BCC   $B623
```

```
;[not part of this routine, but
; interesting nonetheless]
;B606-   A9 FF        LDA   #$FF
;B608-   85 D6        STA   $D6
;B60A-   AD 00 C0     LDA   $C000
;B60D-   C9 83        CMP   #$83
;B60F-   D0 06        BNE   $B617
;B611-   2C 10 C0     BIT   $C010
;B614-   6C 94 BA     JMP   ($BA94)
;B617-   A5 39        LDA   $39
;B619-   CD 03 9D     CMP   $9D03
;B61C-   60           RTS

; from $B600
; set A and Y to standard epilogues
B61D-   A9 DE        LDA   #$DE
B61F-   A0 AA        LDY   #$AA
B621-   D0 03        BNE   $B626

; from $B603
; set A and Y to non-standard epilogues
B623-   A9 FF        LDA   #$FF
B625-   A8           TAY

; every path ends up here
; change the epilogue bytes in memory
; that the RWTS looks for after both
; address and data fields
B626-   8D 91 B9     STA   $B991
B629-   8D 35 B9     STA   $B935
B62C-   8D AE BC     STA   $BCAE
B62F-   8D 9E B8     STA   $B89E
B632-   8C 9B B9     STY   $B99B
B635-   8C 3F B9     STY   $B93F
B638-   8C B3 BC     STY   $BCB3
B63B-   8C A3 B8     STY   $B8A3
B63E-   60           RTS
```

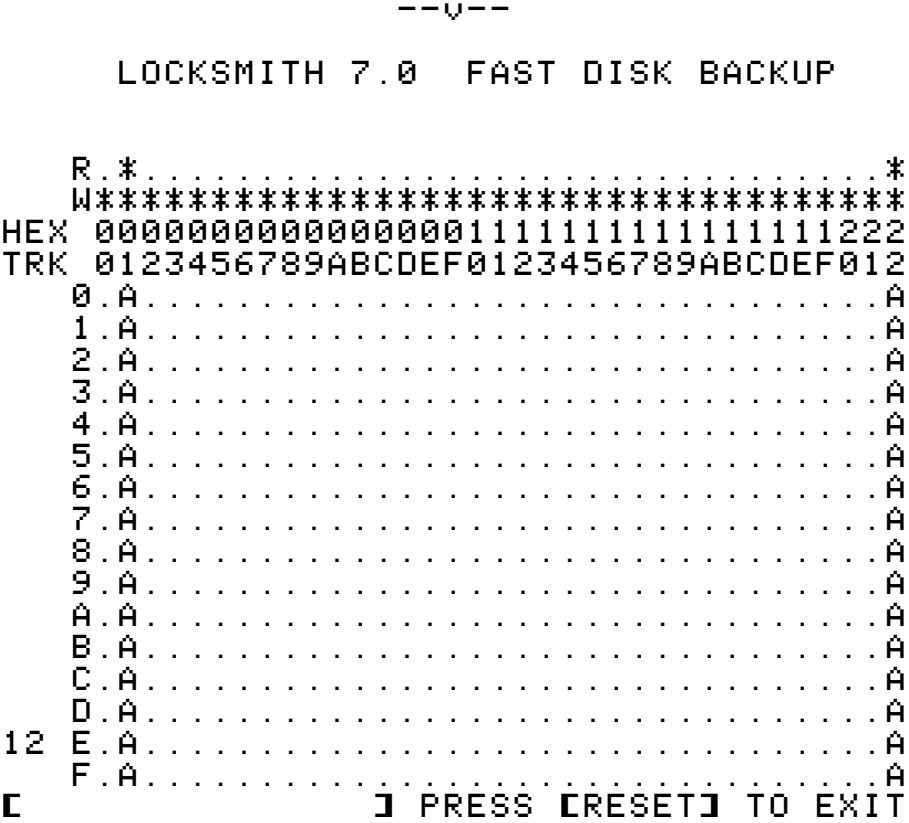
A well-placed "RTS" at $B626 should neutralize all this fiddling. A quick sector search for "91 B9" shows that $B600 ended up on T02,S01. (Remember, the sector interleaving is non-standard and I never changed it.)

T02,S01,$26 change "8D" to "60"

]PR#6
...everything works...

Side B isn't bootable, but every track uses the "FF FF EB" epilogues. Well, almost every track... as you can see in this screenshot from Super Demuffin:

--v--



--^--

Tracks $01 and $22 are unformatted and marked as used in the DOS 3.3 VTOC. All programs on the second side work, even after saving to a data disk and going back to the program disk. There doesn't appear to be any further protection.

Quod erat liberandum.