Borrowed Time

You're seated in your office. Your rundown shoeheels add a few more scuffmarks to the oak desk in front of you.
:

2016-11-19

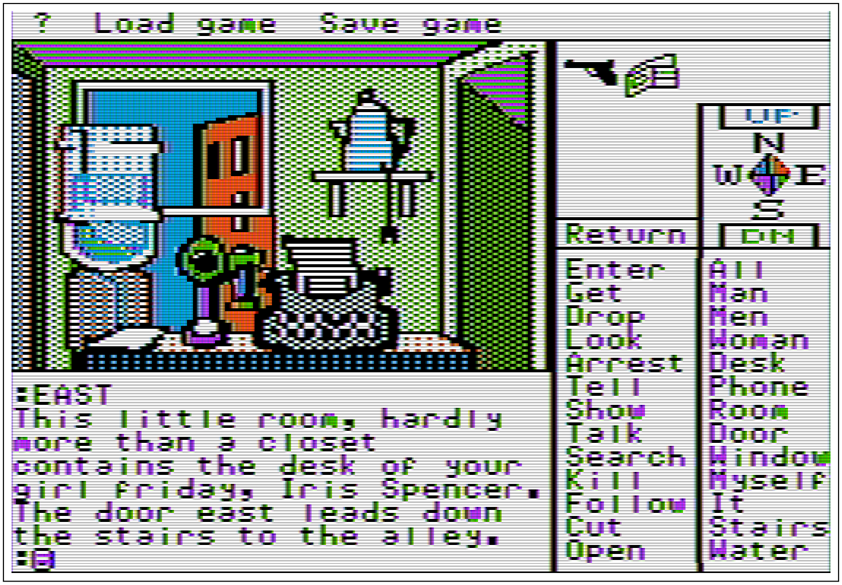
4am
to deprotect and preserve

Contents

```
-------------Borrowed Time--------------
A 4am crack                   2016-11-19
---------------------------------------

Name: Borrowed Time
Genre: adventure
Year: 1985
Authors: Rebecca Heineman, Interplay
  Productions
Publisher: Activision
Platform: Apple ][+ or later (64K)
Media: double-sided 5.25-inch floppy
OS: custom
Previous cracks:
  The Talisman / First Class
  another uncredited crack
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

```
COPYA
  immediate disk read error, but it
  gets a participation trophy just for
  showing up

Locksmith Fast Disk Backup
  unable to read any track

EDD 4 bit copy (no sync, no count)
  works
```



```
:EAST
This little room, hardly
more than a closet
contains the desk of your
girl friday, Iris Spencer.
The door east leads down
the stairs to the alley.
:
```

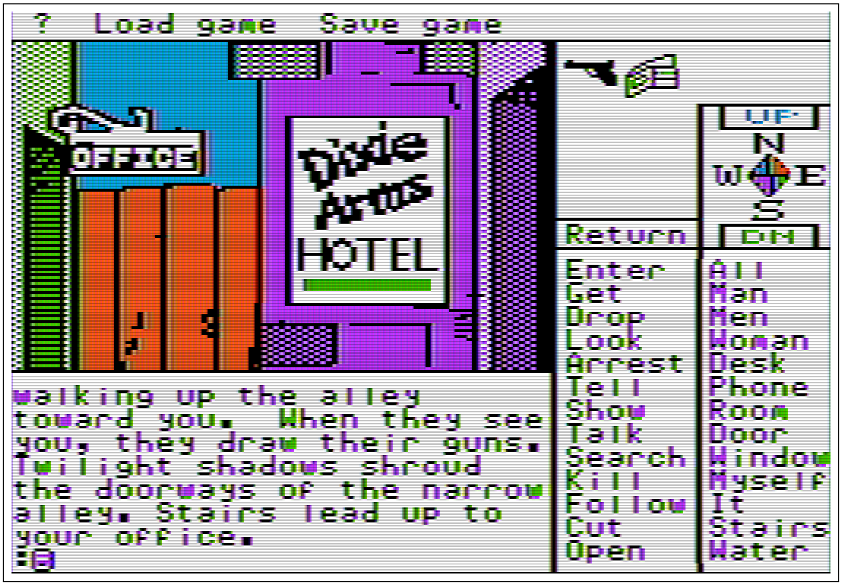
```
Copy ][+ nibble editor
   data fields seem normal, but there's
   no stable prologue/epilogue pattern
   for address field at all -- not even
   within each track(!)

                 --v--

TRACK: 01  START: 2B7A  LENGTH: 1825

2B58: FF FF FF FF FF FF FF FF   VIEW
2B60: FF FF FF FF FF FF FF FF
2B68: FF FF FF FF FF FF FF FF
2B70: FF FF FF FF FF FF FF FF
2B78: FF FF FF FF D5 D5 97 EE
                     ^^^^^^^^
                 address prologue

2B80: AA AB AA AB AA AA AA AA
      ^^^^^ ^^^^^ ^^^^^ ^^^^^
      V=001 T=$01 S=$00 chksm

2B88: 96 ED EB FF FF FF FF FF
      ^^^^^^^^
   address epilogue

2B90: FF FF FF D5 AA AD 96 96
               ^^^^^^^^
            data prologue (normal)

2B98: 96 96 96 96 96 96 96 96

                 --^--
```

But again, EVERY SECTOR IS DIFFERENT,
even the other sectors on the same
track. Here is track 1, sector 1:

```
                 --v--

   COPY ][ PLUS BIT COPY PROGRAM 8.4
(C) 1982-9 CENTRAL POINT SOFTWARE, INC.
---------------------------------------

TRACK: 01  START: 2E02  LENGTH: 1825

2F68: 96 96 96 96 96 96 96 96   VIEW
2F70: 96 96 D6 DE AA EB FF FF
2F78: FF FF FF FF FF FF FF FF
2F80: FF FF FF FF D5 D5 97 EE
                     ^^^^^^^^
                address prologue (same)

2F88: AA AB AA AB AA AB AA AB
      ^^^^^ ^^^^^ ^^^^^ ^^^^^
      V=001 T=$01 S=$01 chksm

2F90: 97 EE EB FF FF FF FF FF
      ^^^^^^^^
  address epilogue (different)

2F98: FF FF FF D5 AA AD B7 B7
2FA0: 9D F3 EE AF AE B7 B7 9D
2FA8: F3 EE AF AE B7 B7 9D F3

                 --^--
```

And the address prologues on other
tracks are completely different. Whee.

```
Disk Fixer
   Given the right combination of
   prologue and epilogue, I can read
   each sector on the disk. (I didn't
   really try them all, but I was able
   to read all the ones I tried.) So...
   probably no custom nibble translate
   table or other encoding weirdness.
```



```
walking up the alley
toward you.  When they see
you, they draw their guns.
Twilight shadows shroud
the doorways of the narrow
alley. Stairs lead up to
your office.
:
```

Also, this:

```
                 --v--

-------------- DISK EDIT --------------
TRACK $00/SECTOR $0F/VOLUME $FE/BYTE$00
---------------------------------------
$00:   P L E A S E   D O N ' T
$10:   B R E A K   O R   C O P Y
$20:   T H I S   P R O G R A M   I
$30:   S P E N T   A   L O N G
$40:   T I M E   P E R F E C T I N G
$50:   T H E   P U L L - D O W N
$60:   M E N U   S Y S T E M   A N D
$70:   I   W O U L D   B E
$80:   D I S A P P O I N T E D   I N
$90:   Y O U   I F   Y O U   B R O K E
$A0:     T H I S   P R O G R A M
$B0:
$C0:             S I G N E D
$D0:
$E0:         B I L L   H E I N E M A N
$F0:       ( 7 1 4 )   X X X - X X X X
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL

---------------------------------------
COMMAND : _

                 --^--
```

I've masked out the phone number here because it's obviously not valid any longer. But I asked Rebecca Heineman (formerly Bill) about it, and she confirmed that the phone number listed was her direct office line at Interplay Productions. And no, no one ever called her about breaking it.

Why didn't COPYA work?
  modified prologues and epilogues
  (every track)

Why didn't Locksmith FDB work?
  ditto

EDD worked. What does that tell us?
  no half or quarter tracks
  almost certainly no nibble check
  (just structural changes to prologues
  and epilogues)

Next steps:

1. Trace the boot
2. Capture the original RWTS that can
   read this mess
3. Convert the disk to a standard
   format with Advanced Demuffin
4. Declare victory (*)

(*) go to the gym

Chapter 1
In Which We Will Not Be Going To The Gym Anytime Soon

The floppy drive firmware code at $C600 is responsible for aligning the drive head and reading sector 0 of track 0 into main memory at $0800. Because the drive can be connected to any slot, the firmware code can't assume it's loaded at $C600. If the floppy drive card were removed from slot 6 and reinstalled in slot 5, the firmware code would load at $C500 instead.

To accommodate this, the firmware does some fancy stack manipulation to detect where it is in memory (which is a neat trick, since the 6502 program counter is not generally accessible). However, due to space constraints, the detection code only cares about the lower 4 bits of the high byte of its own address.

$C600 (or $C500, or anywhere in $Cx00) is read-only memory. I can't change it, which means I can't stop it from transferring control to the boot sector of the disk once it's in memory. BUT! The disk firmware code works unmodified at any address. Any address that ends with $x600 will boot slot 6, including $B600, $A600, $9600, &c.

Thus, from the monitor:

```
; copy drive firmware to $9600
*9600<C600.C6FFM

; and execute it
*9600G
...boots slot 6...
```

Now then:

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

]PR#5
...
]CALL -151

*9600<C600.C6FFM
*96F8L

96F8-   4C 01 08    JMP   $0801
```

That's where the disk controller ROM
code ends and the on-disk code begins.
But $9600 is part of read/write memory.
I can change it at will. So I can
interrupt the boot process after the
drive firmware loads the boot sector
from the disk but before it transfers
control to the disk's bootloader.

```
; instead of jumping to on-disk code,
; copy boot sector to higher memory so
; it survives a reboot
96F8-   A0 00       LDY   #$00
96FA-   B9 00 08    LDA   $0800,Y
96FD-   99 00 28    STA   $2800,Y
9700-   C8          INY
9701-   D0 F7       BNE   $96FA

; turn off slot 6 drive motor
9703-   AD E8 C0    LDA   $C0E8

; reboot to my work disk in slot 5
9706-   4C 00 C5    JMP   $C500

*BSAVE TRACE,A$9600,L$109
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT0,A$2800,L$100
```

Now let's see how this disk boots.

```
]CALL -151

; move boot0 back into place
*800<2800.28FFM
*801L

; 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
0801-   A9 60       LDA   #$60
0803-   8D 01 08    STA   $0801

; switch to ROM
0806-   2C 82 C0    BIT   $C082
```

```
; set up a slot-independent vector to
; call $Cx5C to read more sectors
0809-   8A             TXA
080A-   4A             LSR
080B-   4A             LSR
080C-   4A             LSR
080D-   4A             LSR
080E-   09 C0          ORA   #$C0
0810-   8D 25 08       STA   $0825

; loop to read more sectors on track 0
0813-   A9 0A          LDA   #$0A
0815-   85 00          STA   $00
0817-   A4 00          LDY   $00

; physical sector number goes in zp$3D
0819-   B9 58 08       LDA   $0858,Y
081C-   85 3D          STA   $3D

; target address goes in zp$27
081E-   B9 68 08       LDA   $0868,Y
0821-   85 27          STA   $27
0823-   20 5C C6       JSR   $C65C
0826-   C6 00          DEC   $00
0828-   D0 ED          BNE   $0817
```

Looking at the array at $0868, it seems we are reading sectors into $0900+:

```
*868.

0868- 08 09 0A 0B 0C 0D 0E 0F
0870- 10 11 12
```

$0801 is now an "RTS" instruction, so the JSR $C65C at $0823 will simply read a sector from disk and return, then continue with the next instruction. This is a common, elegant technique.

Continuing from $082A...

```
; machine initialization stuff (PR#0,
; IN#0, TEXT, HOME)
082A-   20 89 FE    JSR   $FE89
082D-   20 93 FE    JSR   $FE93
0830-   20 2F FB    JSR   $FB2F
0833-   20 58 FC    JSR   $FC58

; check for 64K
0836-   2C 83 C0    BIT   $C083
0839-   2C 83 C0    BIT   $C083
083C-   A9 FF       LDA   #$FF
083E-   8D 00 E0    STA   $E000
0841-   EE 00 E0    INC   $E000
0844-   D0 55       BNE   $089B

; display "ADVENT2 VERSION 1.00"
0846-   A2 00       LDX   #$00
0848-   BD 73 08    LDA   $0873,X
084B-   F0 06       BEQ   $0853
084D-   9D D8 07    STA   $07D8,X
0850-   E8          INX
0851-   D0 F5       BNE   $0848
0853-   A6 2B       LDX   $2B

; continue elsewhere (in the code we
; just read from disk)
0855-   4C 00 09    JMP   $0900
```

And that's where I get to interrupt the boot.

Chapter 2
Boot Trace and Chill

```
*9600<C600.C6FFM

; change JMP instruction at $0855 to a
; callback that I control (below)
; instead of continuing to $0900
96F8-   A9 05       LDA   #$05
96FA-   8D 56 08    STA   $0856
96FD-   A9 97       LDA   #$97
96FF-   8D 57 08    STA   $0857

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

; (callback is here)
; copy everything to higher memory so
; it survives a reboot
9705-   A2 10       LDX   #$10
9707-   A0 00       LDY   #$00
9709-   B9 00 08    LDA   $0800,Y
970C-   99 00 28    STA   $2800,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

; turn off slot 6 drive motor
971B-   AD E8 C0    LDA   $C0E8

; switch to ROM
971E-   AD 82 C0    LDA   $C082

; reboot to my work disk in slot 5
9721-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE2,A$9600,L$124
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1,A$2800,L$1000
]CALL -151

; copy the code back to where it was
; originally loaded
*800<2800.37FFM
*900L

0900-   4C 2A 0B     JMP   $0B2A

*B2AL

; more machine initialization switches
0B2A-   8D 0E C0     STA   $C00E
0B2D-   8D 0C C0     STA   $C00C
0B30-   8D 00 C0     STA   $C000
0B33-   8D 02 C0     STA   $C002
0B36-   8D 04 C0     STA   $C004
0B39-   8D 08 C0     STA   $C008

; zero out zero page
0B3C-   A0 00        LDY   #$00
0B3E-   98           TYA
0B3F-   99 00 00     STA   $0000,Y
0B42-   C8           INY
0B43-   D0 FA        BNE   $0B3F

; save boot slot (was previously in
; zp$2B during early boot, but we just
; wiped that along with the rest of
; zero page)
0B45-   86 5A        STX   $5A
```

```
; text mode
0B47-    2C 51 C0     BIT   $C051
0B4A-    2C 57 C0     BIT   $C057
0B4D-    2C 54 C0     BIT   $C054
0B50-    2C 52 C0     BIT   $C052

; reset vector to reboot from whence we
; came
0B53-    A9 00        LDA   #$00
0B55-    8D F2 03     STA   $03F2
0B58-    85 71        STA   $71
0B5A-    8A           TXA
0B5B-    4A           LSR
0B5C-    4A           LSR
0B5D-    4A           LSR
0B5E-    4A           LSR
0B5F-    09 C0        ORA   #$C0
0B61-    8D F3 03     STA   $03F3
0B64-    49 A5        EOR   #$A5
0B66-    8D F4 03     STA   $03F4

; Munge the boot slot from $60 to $EC.
; Are we setting up a fastloader RWTS?
0B69-    8A           TXA
0B6A-    09 8C        ORA   #$8C
0B6C-    8D 33 09     STA   $0933
0B6F-    8D 4A 09     STA   $094A
0B72-    8D 60 09     STA   $0960
0B75-    8D 74 09     STA   $0974
0B78-    8D 89 09     STA   $0989

; sets up some RWTS tables (not shown)
0B7B-    20 81 11     JSR   $1181
```

```
; copy ROM to LC RAM bank
0B7E-   A2 FF         LDX   #$FF
0B80-   9A            TXS
0B81-   2C 81 C0      BIT   $C081
0B84-   2C 81 C0      BIT   $C081
0B87-   E8            INX
0B88-   BD 00 F8      LDA   $F800,X
0B8B-   9D 00 F8      STA   $F800,X
0B8E-   CA            DEX
0B8F-   D0 F7         BNE   $0B88
0B91-   EE 8A 0B      INC   $0B8A
0B94-   EE 8D 0B      INC   $0B8D
0B97-   D0 EF         BNE   $0B88
0B99-   A9 60         LDA   #$60
0B9B-   85 5D         STA   $5D

; this looks like a high-level RWTS
; entry point
0B9D-   A9 12         LDA   #$12
0B9F-   A2 00         LDX   #$00
0BA1-   A0 BA         LDY   #$BA
0BA3-   20 29 11      JSR   $1129

; ...especially since right after the
; JSR, we're calling code that wasn't
; in memory just a minute ago
0BA6-   20 00 60      JSR   $6000
```

So let's have a look at $1129.

```
*1129L

1129-   85 5C         STA   $5C
112B-   84 5B         STY   $5B
112D-   86 5E         STX   $5E
```

```
; check which sectors on this track we
; want to read and where they should go
in memory (not shown)
; returns carry set if this is the last
; track we need to read
112F-   20 3E 11    JSR   $113E
1132-   08          PHP

; reads a track into memory (see below)
1133-   20 30 0A    JSR   $0A30

; loop forever on read error
1136-   B0 FB       BCS   $1133

; increment track and loop back if we
; have more to read
1138-   E6 5C       INC   $5C
113A-   28          PLP
113B-   90 F2       BCC   $112F
113D-   60          RTS
```

And $0A30 is the main entry point to
read a track.

```
*A30L

0A30-   A9 02       LDA   #$02
0A32-   85 51       STA   $51
0A34-   0A          ASL
0A35-   85 70       STA   $70

; turn on the drive motor, reset the
; data latch, and select drive 1
0A37-   A6 5A       LDX   $5A
0A39-   BD 8E C0    LDA   $C08E,X
0A3C-   BD 8C C0    LDA   $C08C,X
0A3F-   BD 89 C0    LDA   $C089,X
0A42-   BD 8A C0    LDA   $C08A,X
```

```
; move to the desired track (not shown)
0A45-   A5 5C        LDA   $5C
0A47-   20 C5 0A     JSR   $0AC5

0A4A-   A5 5C        LDA   $5C
0A4C-   20 CD 11     JSR   $11CD

*11CDL

; track $00 is treated specially
11CD-   09 00        ORA   #$00
11CF-   F0 10        BEQ   $11E1

; otherwise, take low 4 bits of track #
11D1-   29 0F        AND   #$0F

; and use that as an index into two
; different lookup tables
11D3-   A8           TAY
11D4-   B9 AD 11     LDA   $11AD,Y
11D7-   8D C8 09     STA   $09C8
11DA-   B9 BD 11     LDA   $11BD,Y
11DD-   8D D3 09     STA   $09D3
11E0-   60           RTS

; special case for track 0 (called from
; $11CF)
11E1-   A9 AA        LDA   #$AA
11E3-   8D C8 09     STA   $09C8
11E6-   A9 96        LDA   #$96
11E8-   8D D3 09     STA   $09D3
11EB-   60           RTS
```

Based on the special case code ($11E1), I'm guessing that $09C8 and $09D3 are the addresses of the second and third address prologue nibbles in the RWTS. Let's take a look at those lookup tables:

```
*11AD.11BC

11AD- .. .. .. .. .. 96 97 9A
11B0- 9B 9D 9E 9F A6 ED EE EF
11B8- F2 F3 F4 F5 F6

*11BD.11CC

11BD- .. .. .. .. .. ED EE EF
11C0- F2 F3 F4 FF F7 96 A6 AA
11C8- D5 DF EA AE FE
```

I verified these manually against the values I saw in the Copy II Plus nibble editor, and they match up. Track $01 uses an address prologue "D5 97 EE"; track $02 uses "D5 9A EF"; and so on. Track $00 needs to use the standard "D5 AA 96" because it's read with the drive firmware code at $Cx5C. But track $10 uses the custom "D5 96 ED", exactly as this table would predict.

Continuing from $0A4F...

```
0A4F-   A0 40       LDY   #$40
0A51-   84 50       STY   $50
0A53-   A6 5A       LDX   $5A
0A55-   20 AD 09    JSR   $09AD
```

```
*9ADL

; standard address prologue matching
; code, similar to DOS 3.3
09AD-   A0 FC       LDY   #$FC
09AF-   84 52       STY   $52
09B1-   C8          INY
09B2-   D0 04       BNE   $09B8
09B4-   E6 52       INC   $52
09B6-   F0 F3       BEQ   $09AB
09B8-   BD 8C C0    LDA   $C08C,X
09BB-   10 FB       BPL   $09B8
09BD-   C9 D5       CMP   #$D5
09BF-   D0 F0       BNE   $09B1
09C1-   EA          NOP

; but remember, these second and third
; prologue nibbles were just modified
; by the subroutine at $11CD
09C2-   BD 8C C0    LDA   $C08C,X
09C5-   10 FB       BPL   $09C2
09C7-   C9 AA       CMP   #$AA     <-- !
09C9-   D0 F2       BNE   $09BD
09CB-   A0 03       LDY   #$03
09CD-   BD 8C C0    LDA   $C08C,X
09D0-   10 FB       BPL   $09CD
09D2-   C9 96       CMP   #$96     <-- !
09D4-   D0 E7       BNE   $09BD
```

```
; parse address field, store it in
; zp$56+
09D6-   A9 00         LDA   #$00
09D8-   85 53         STA   $53
09DA-   BD 8C C0      LDA   $C08C,X
09DD-   10 FB         BPL   $09DA
09DF-   2A            ROL
09E0-   85 52         STA   $52
09E2-   BD 8C C0      LDA   $C08C,X
09E5-   10 FB         BPL   $09E2
09E7-   25 52         AND   $52
09E9-   99 56 00      STA   $0056,Y
09EC-   45 53         EOR   $53
09EE-   88            DEY
09EF-   10 E7         BPL   $09D8
09F1-   A8            TAY
09F2-   D0 B7         BNE   $09AB

; Y = sector number (just parsed above)
09F4-   A4 57         LDY   $57

; match epilogue nibbles directly
; against... the same two lookup tables
; we used to rotate the prologue!
09F6-   BD 8C C0      LDA   $C08C,X
09F9-   10 FB         BPL   $09F6
09FB-   D9 AD 11      CMP   $11AD,Y
09FE-   D0 AB         BNE   $09AB
0A00-   EA            NOP
0A01-   BD 8C C0      LDA   $C08C,X
0A04-   10 FB         BPL   $0A01
0A06-   D9 BD 11      CMP   $11BD,Y
0A09-   D0 A0         BNE   $09AB
0A0B-   18            CLC
0A0C-   60            RTS
```

To sum up: this RWTS demands 16 different address prologues -- one for every track for 16 tracks, then repeat. Simultaneously, it demands 16 different address epilogues -- one for every sector, then repeat on the next track. Which means... [drum roll please]...

THIS DISK HAS 256 DIFFERENT ADDRESS PROLOGUE/EPILOGUE COMBINATIONS.

That's gross. I love it.

Chapter 3
In Which We Attempt To Use The Original Disk As A Weapon Against Itself, And It Goes Incredibly Poorly

The rest of the RWTS is unsurprising
from a copy protection standpoint. I
mean, it's brilliant and fast, but the
data fields are 100% standard, so let's
move on to the part where we try to use
it against itself.

The best tool I have to convert disks
to a standard format is Advanced
Demuffin, but it's kind of a poor fit.
This disk's RWTS is centered around
tracks, but Advanced Demuffin, like DOS
3.3, is centered around sectors. To top
it off, the RWTS sits exactly where
Advanced Demuffin sits in memory, just
because f--- you.

But let's see what we can do to build
ourselves a compatible RWTS.

```
*C500G
...hold down <Esc> to avoid Diversi-DOS
   relocating to the language card...
```

OK, I have a DOS-shaped RWTS at $B800
that can read standard disks.

```
]BLOAD ADVANCED DEMUFFIN 1.5
```

```
]CALL -151

; standard "IOB" interface --
; A = phase (track x 2)
; Y = sector
; X = address high
1400-   4A          LSR
1401-   8D 22 0F    STA   $0F22
1404-   8C 23 0F    STY   $0F23
1407-   8E 27 0F    STX   $0F27
140A-   A9 01       LDA   #$01
140C-   8D 20 0F    STA   $0F20
140F-   8D 2A 0F    STA   $0F2A
1412-   AD 22 0F    LDA   $0F22

; before we call $BD00 to read a sector
; we need to munge the RWTS like the
; routine at $11CD on the original disk
; (see listings below)
1415-   20 CD 14    JSR   $14CD
1418-   AC 23 0F    LDY   $0F23
141B-   20 EC 14    JSR   $14EC

; now call the RWTS to read a sector
; off the original disk
141E-   A9 0F       LDA   #$0F
1420-   A0 1E       LDY   #$1E
1422-   4C 00 BD    JMP   $BD00
```

And here are the two functions that set up our RWTS to read a specific sector, choosing the right combination of address prologue and epilogue out of the 256 (!) possibilities.

```
*14CDL

; verbatim logic from $11CD on original
; to set the second and third nibbles
; of the address prologue (at $B95F and
; $B96A in a standard DOS-shaped RWTS)
14CD-   09 00       ORA   #$00
14CF-   F0 16       BEQ   $14E7
14D1-   29 0F       AND   #$0F
14D3-   A8          TAY
14D4-   B9 AD 14    LDA   $14AD,Y
14D7-   8D 5F B9    STA   $B95F
14DA-   B9 BD 14    LDA   $14BD,Y
14DD-   8D 6A B9    STA   $B96A
14E0-   60          RTS

; special case for track 0
14E1-   A9 AA       LDA   #$AA
14E3-   8D 5F B9    STA   $B95F
14E6-   A9 96       LDA   #$96
14E8-   8D 6A B9    STA   $B96A
14EB-   60          RTS

; given the sector number (in Y), set
; the expected address epilogues (at
; $B991 and $B99B) from the same lookup
; tables
14EC-   B9 AD 14    LDA   $14AD,Y
14EF-   8D 91 B9    STA   $B991
14F2-   B9 BD 14    LDA   $14BD,Y
14F5-   8D 9B B9    STA   $B99B
14F8-   60          RTS
```

And these two lookup tables are copied verbatim from $11AD and $11BD on the original disk:

```
*14AD.14BC

14AD- .. .. .. .. .. 96 97 9A
14B0- 9B 9D 9E 9F A6 ED EE EF
14B8- F2 F3 F4 F5 F6

*14BD.14CC

14BD- .. .. .. .. .. ED EE EF
14C0- F2 F3 F4 FF F7 96 A6 AA
14C8- D5 DF EA AE FE

*BSAVE IOB,A$1400,L$100

; let's do this thing!
*800G
```

There's nothing left to configure, because both the RWTS and our custom IOB are already in memory, so...

[press "C" to convert disk]

```
ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======PRESS ANY KEY TO CONTINUE=======
TRK:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
+.5:
    0123456789ABCDEF0123456789ABCDEF012
SC0:...................................
SC1:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC2:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC3:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC4:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC5:...................................
SC6:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC7:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC8:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SC9:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SCA:...................................
SCB:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SCC:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SCD:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SCE:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SCF:...................................
=======================================
16SC $00,$00-$22,$0F BY1.0 S6,D1->S6,D2
```

--^--

Read errors on 75% of the disk. And not just any 75% -- 12 out of 16 sectors on each track. And not just any 12... I've made a grave error of logic somewhere.

Let's back up.

Chapter 4
When I Was Young, It Seemed That Life Was So Wonderful, A Miracle Oh It Was Beautiful, Magical

It took me a long time to figure out
why this didn't work. The logic is
impeccable. Every track gets two
prologue values from our lookup tables;
every sector gets two epilogue values
from the same tables. The data field
prologues and epilogues are entirely
standard, and the nibble translate
table that converts the data field from
nibbles to bytes is entirely standard.

What. Is. The. Difference.

Finally I set up the IOB to break into
the monitor after reading a sector, so
I could examine the RWTS in situ.

```
141E-   A9 0F       LDA   #$0F
1420-   A0 1E       LDY   #$1E
1422-   20 00 BD    JSR   $BD00
1425-   4C 59 FF    JMP   $FF59   <-- !
```

Then I re-ran Advanced Demuffin and
converted a single sector: T01,S01.
This is the earliest sector that uses
both custom prologues and epilogues. I
know the same failure pattern occurs on
track 0, but I also know that track 0
is a special case on the original disk.
Anyway, that was my thinking.

Here's what my RWTS looks like after it
fails to read T01,S01:

```
*B94FL

B94F-   BD 8C C0    LDA   $C08C,X
B952-   10 FB       BPL   $B94F
B954-   C9 D5       CMP   #$D5        (1)
B956-   D0 F0       BNE   $B948
B958-   EA          NOP
B959-   BD 8C C0    LDA   $C08C,X
B95C-   10 FB       BPL   $B959
B95E-   C9 97       CMP   #$97        (2)
B960-   D0 F2       BNE   $B954
B962-   A0 03       LDY   #$03
B964-   BD 8C C0    LDA   $C08C,X
B967-   10 FB       BPL   $B964
B969-   C9 EE       CMP   #$EE        (3)
B96B-   D0 E7       BNE   $B954
B96D-   A9 00       LDA   #$00
B96F-   85 27       STA   $27
B971-   BD 8C C0    LDA   $C08C,X
B974-   10 FB       BPL   $B971
B976-   2A          ROL
B977-   85 26       STA   $26
B979-   BD 8C C0    LDA   $C08C,X
B97C-   10 FB       BPL   $B979
B97E-   25 26       AND   $26
B980-   99 2C 00    STA   $002C,Y
B983-   45 27       EOR   $27
B985-   88          DEY
B986-   10 E7       BPL   $B96F
B988-   A8          TAY
B989-   D0 B7       BNE   $B942
B98B-   BD 8C C0    LDA   $C08C,X
B98E-   10 FB       BPL   $B98B
B990-   C9 97       CMP   #$97        (4)
B992-   D0 AE       BNE   $B942
[...]
```

```
B994-   EA          NOP
B995-   BD 8C C0    LDA   $C08C,X
B998-   10 FB       BPL   $B995
B99A-   C9 EE       CMP   #$EE      (5)
B99C-   D0 A4       BNE   $B942
B99E-   18          CLC
B99F-   60          RTS
```

Five values, all as expected:

```
  (1) D5 \
  (2) 97  } address prologue
  (3) EE /

  (4) 97 \
  (5) EE / address epilogue
```

And here's what T01,S01 looks like on
the original disk:

```
                 --v--

TRACK: 01  START: 1800  LENGTH: 3DFF

2138: 96 96 96 96 96 96 96 96   VIEW
2140: 96 D6 DE AA EB FF FF FF
2148: FF FF FF FF FF FF FF FF
2150: FF FF FF FF D5 D5 97 EE
                     ^^^^^^^^
                 address prologue

2158: AA AB AA AB AA AB AA AB  <-215C
      ^^^^^ ^^^^^ ^^^^^ ^^^^^
      V=001 T=$01 S=$01 chksm

2160: 97 EE EB FF FF FF FF FF
      ^^^^^
 address epilogue

2168: FF FF FF D5 AA AD B7 B7
2170: 9D F3 EE AF AE B7 B7 9D  FIND:
2178: F3 EE AF AE B7 B7 9D F3  AA AB

                 --^--
```

The address prologue is "D5 97 EE", and
the address epilogue is "97 EE".

What. Is. The. Difference.

[...time passes...]

[...time passes...]

[...it is pitch black...you are likely
to be eaten by a grue...]

In desperation, I consulted "Beneath Apple DOS," which is actually not a bad choice if you're ever at the point of desperation for any reason. And there, on page 3-23, was the answer that had eluded me.

--v--

COMPARISON OF SECTOR SKEWING

PHYSICAL SECTOR	LOGICAL DOS 3.3	
0	0	<--
1	7	
2	E	
3	6	
4	D	
5	5	<--
6	C	
7	4	
8	B	
9	3	
A	A	<--
B	2	
C	9	
D	1	
E	8	
F	F	<--

--^--

The address field on disk contains the
metadata for the data that follows:
disk volume number, track, sector, and
a checksum. But the "sector" is not a
logical sector number (that DOS 3.3
expects), but a physical sector number.
As shown in the table above, the
logical sectors are "skewed" -- out of
order, originally thought to provide a
good compromise so that sequential
decrementing sector reads would be
faster.

(In fact, this disk's RWTS reads every
sector it finds and converts the data
field quickly enough that it can read
an entire track in one revolution,
regardless of the skewing. And DOS 3.3
managed to add an unnecessary memory
move so it "misses" the next sector and
has to wait an entire disk revolution
for it to come around again. This is
why Apple DOS 3.3 is so slow and
third-party products like Pronto-DOS
and Diversi-DOS are so much faster.
But never mind that.)

The point is this: Advanced Demuffin, calling a DOS-shaped RWTS, deals in logical sectors, but the address field deals in physical sectors. There's a lookup table to convert between them, but in the absence of that conversion, there are only 4 sectors where the physical and logical sector numbers are the same: $00, $05, $0A, and $0F. And those were the sectors that I was able to read -- totally by accident, because the physical and logical sector numbers happened to line up.

Now let's take another look at the original disk's RWTS:

```
; Y = PHYSICAL sector number (just
; parsed from the address field)
09F4-   A4 57         LDY   $57

09F6-   BD 8C C0      LDA   $C08C,X
09F9-   10 FB         BPL   $09F6

; look up the first epilogue in the
; first lookup table, with the PHYSICAL
; sector number as the index
09FB-   D9 AD 11      CMP   $11AD,Y
09FE-   D0 AB         BNE   $09AB
0A00-   EA            NOP
0A01-   BD 8C C0      LDA   $C08C,X
0A04-   10 FB         BPL   $0A01
```

```
; and look up the second epilogue in
; the second lookup table, using the
; PHYSICAL sector number as the index
0A06-   D9 BD 11    CMP   $11BD,Y
0A09-   D0 A0       BNE   $09AB
0A0B-   18          CLC
0A0C-   60          RTS
```

So there it is. That's the difference.
It's only logical.

Chapter 5
Then We Shall Make Another!

As I mentioned, a standard DOS-shaped RWTS has a lookup table to convert logical to physical sectors, at $BFB8.

```
BFB8- 00 0D 0B 09 07 05 03 01
BFC0- 0E 0C 0A 08 06 04 02 0F
```

So I can add two lines to my IOB code to convert the logical sector numbers I'm using to read from the original disk to the physical sector numbers required to look up the proper address prologues and epilogues in the lookup tables. That sentence was too long, but I don't know how to make it simpler. Sorry.

```
*C500G
...
]BLOAD ADVANCED DEMUFFIN 1.5
]BLOAD IOB
]CALL -151

14EC-   B9 B8 BF    LDA   $BFB8,Y   <--
14EF-   A8          TAY             <--
14F0-   B9 AD 14    LDA   $14AD,Y
14F3-   8D 91 B9    STA   $B991
14F6-   B9 BD 14    LDA   $14BD,Y
14F9-   8D 9B B9    STA   $B99B
14FC-   60          RTS

*BSAVE IOB,A$1400,L$100
*800G

[press "C" to convert disk]
```

```
ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======PRESS ANY KEY TO CONTINUE=======
TRK:...................................
+.5:
    0123456789ABCDEF0123456789ABCDEF012
SC0:...................................
SC1:...................................
SC2:...................................
SC3:...................................
SC4:...................................
SC5:...................................
SC6:...................................
SC7:...................................
SC8:...................................
SC9:...................................
SCA:...................................
SCB:...................................
SCC:...................................
SCD:...................................
SCE:...................................
SCF:...................................
=======================================
16SC $00,$00-$22,$0F BY1.0 S6,D1->S6,D2
```

--^--

\o/ Hooray! It worked!

(I converted side B the same way.)

Now I have a 100% standard disk that boots and immediately hangs because it can't read itself, because it's looking for 256 flavors instead of 1.

Let's fix that.

Turning to my trusty Disk Fixer sector
editor, I found the original nibble
munger (the subroutine at $11CD) on
T00,S09. That entire routine can just
be an "RTS".

```
T00,S09,$CD: 09 -> 60
```

But there's still the matter of the
epilogue checking. That code is spread
across sectors 1 and 2:

```
T00,S01,$FB -> C9 DE EA
T00,S02,$06 -> C9 AA EA
```

```
]PR#6
```
...boots to title screen, then I press
"1" to select a new game, it asks me to
flip the disk, I do so, and it refuses
to accept that I have done so...

That's odd. I used Advanced Demuffin to
convert side B, and it converted all
tracks, all sectors. It has the same
256-flavors-of-evil structure as the
boot disk, but suddenly the RWTS can't
read it?

On a lark, I inserted the original disk
side B, and it worked. Whoa! Curiouser
and curiouser. Do it undo my patches to
the RWTS?

No, a much simpler explanation: there is a second RWTS. Turning to my trusty Disk Fixer sector editor, I searched for the hex sequence "BD 89 C0" and found an entirely separate RWTS on track $03. (It loads into a different range in memory, which I suppose is why it exists. Not just to f--- with me. Not everything is about me.)

This second RWTS uses the same basic technique to modify itself just before matching an address field prologue, then using the lookup tables directly to match the address epilogue.

Here are the necessary patches (still on side A, the boot disk):

```
; disable the prologue modifications
T03,S0D,$3B: 09 -> 60

; restore standard epilogue checking
T03,S0B,$56 -> C9 DE EA
T03,S0B,$61 -> C9 AA EA

]PR#6
...works, and it is glorious...
```

Quod erat liberandum.

Easter Eggs

Type "REGRUB" during the scrolling credits on the title page to see a secret screen.

Type "BURGER" as a command in the game for an alternate ending. (Save your game first!)