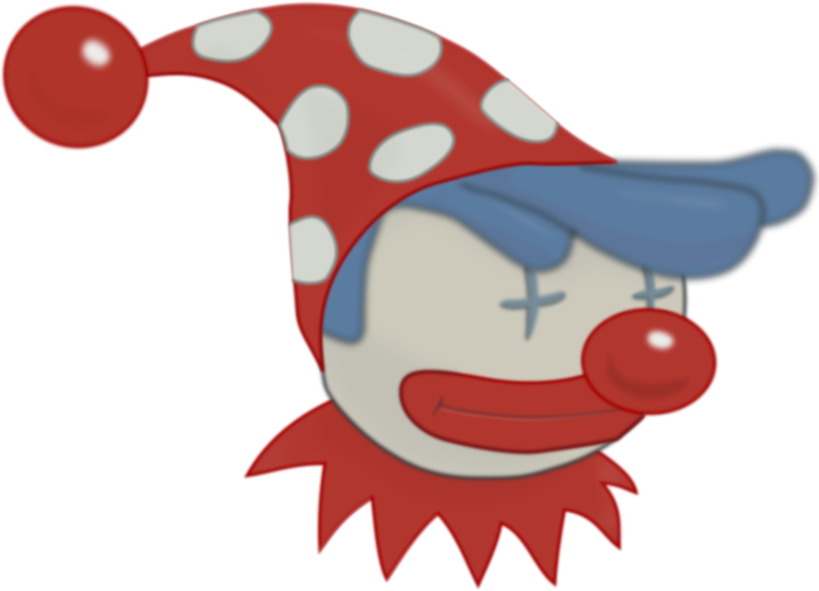
Mr. Do!



2015-09-04

4am

to deprotect and preserve

Contents

```
-----------------Mr. Do!-----------------
A 4am crack                    2015-06-29
--------------------. updated 2015-09-04
                    |____________________

Name: Mr. Do!
Genre: arcade
Year: 1985
Authors: Rick Mirsky
Publisher: Datasoft, Inc.
Media: single-sided 5.25-inch floppy
OS: custom
Other versions: The Cloak

     ______________________________
    {                              }
    { "I don't know anything, but  }
    {  I do know that everything   }
    {  is interesting if you go    }
    {  into it deeply enough."     }
    {                              }
    {             Richard Feynmann }
    {______________________________}
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

```
COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  unable to read any track

EDD 4 bit copy (no sync, no count)
  read errors on tracks $1C-$22
  copy clears screen, turns off drive
    motor, and freezes

Copy ][+ nibble editor
  T00-T1B appear to be 4-4 encoded data
  T1C-T22 appear unformatted

Disk Fixer
  ["O" -> "Input/Output Control"]
    set "CHECKSUM ENABLED" to "NO"
  T00,S00 readable
  nothing else readable

Why didn't COPYA work?
  not a 16-sector disk

Why didn't Locksmith FDB work?
  ditto

Why didn't my EDD copy work?
  I don't know, but it wasn't just a
  structural problem, because the drive
  motor turned off. That doesn't just
  happen. Someone turned it off.

Next steps:

  1. Trace the boot
  2. ???
```

Chapter 1
In Which I Have Just Met You, And I Love You

We're starting from bare metal on this
one. My automated tools, they do
nothing for us. Strap in.

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

]PR#5
...
]CALL -151

*9600<C600.C6FFM

; copy boot sector (T00,S00) to the
; graphics page 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 TRACE0,A$9600,L$109
*9600G
...reboots slot 6...
...reboots slot 5...
```

```
]BSAVE BOOT0,A$2800,L$100
]CALL -151

*800<2800.28FFM

*801L

0801-   74              ???
0802-   4C B0 1C        JMP   $1CB0
```

An illegal opcode, followed by a jump to uninitialized memory? Address $0800 contains $01, so this is the only sector the disk controller ROM reads into memory before passing control. I have no idea how this disk even boots.

.
. [time passes]
.

According to <http://www.ataripreservation.org/websites/freddy.offenga/illopc31.txt>, $74 is an undocumented 6502 opcode that takes a single byte argument and does nothing. Like a double NOP, but with two bytes instead of one.

According to <http://www.6502.org/tutorials/65c02opcodes.html>, $74 is a relatively obscure variant of the STZ (STore Zero) instruction, which was introduced in the 65C02. This form of the STZ instruction takes a one byte operand, a zero page memory location.

The disassembler built into the Apple monitor assumes all unknown opcodes are single-byte, so it misrepresents opcode $74 as a single-byte instruction and incorrectly prints a three-byte JMP instruction on the next line. When the 65c02 made some of those opcodes valid instructions, the monitor disassembler was never updated with information on their mnemonics or arguments, so it has the same problem.

Opcode $74 does nothing of consequence on either CPU, but more importantly, it does nothing in 2 bytes instead of 1.

This is the actual code:

```
0801-   74 4C        DOP   $4C,X ; NOPx2
0803-   B0 1C        BCS   $0821
```

The carry bit is always set coming out of the disk controller routine, so the branch-on-carry at $0803 functions as an unconditional jump.

```
*821L

0821-   48           PHA
0822-   8D BD 08     STA   $08BD
0825-   B0 27        BCS   $084E
```

The accumulator is always $01 coming out of the disk controller routine. So that gets pushed to the stack and saved in $08BD (odd). And more unconditional jumping, since the carry is still set.

```
*84EL

084E-   B0 69       BCS   $08B9

*8B9L

08B9-   B9 05 08    LDA   $0805,Y
08BC-   49 AA       EOR   #$AA
08BE-   99 00 02    STA   $0200,Y
08C1-   88          DEY
08C2-   D0 EE       BNE   $08B2
```

The Y register is always $00 coming out of the disk controller routine, so this loop will decrypt 256 bytes starting at $0805 and store it at $0200. But wait, this code was modified earlier (at $0822) -- the EOR value was changed from $AA to $01 (at $0822). That means this is the actual code:

```
08B9-   B9 05 08    LDA   $0805,Y
08BC-   49 01       EOR   #$01
08BE-   99 00 02    STA   $0200,Y
08C1-   88          DEY
08C2-   D0 EE       BNE   $08B2
```

OK. So now we're going to jump to somewhere in the $0200 range, right?

```
08C4-    A9 FF          LDA   #$FF
08C6-    48             PHA
08C7-    A0 03          LDY   #$03
08C9-    4C 58 FF       JMP   $FF58
```

Well, yes, but in the most roundabout way possible. Remember that we pushed $01 to the stack at $0821. Now we're pushing $FF to the stack as well. So an RTS will "return" to that address + 1, which equals $0200. Then we're jumping to $FF58. What's at $FF58? Just an RTS.

The rest of the boot sector is not actually code. It's not even encrypted code. It's a message from the distant past, a simpler age when computers ran at 1 Mhz and floppy disks were the pinnacle of long-term storage:

```
*FC58G N 400<8CC.8FFM

DON'T WASTE YOUR TIME, ORIGINALS ARE
INEXPENSIVE.JCR
```

Let's waste some time together.

Chapter 2
In Which We Decrypt Code And Run It And Run Code Then Decrypt It And Then Run It Again

The final JMP $FF68 (at $08C9) provides an easy way to interrupt the boot and capture the decrypted code at $0200. I can change that to jump to a routine under my control and copy page 2 to a page that won't get overwritten by rebooting or, you know, typing. (Page 2 is the input buffer for the monitor and DOS.)

```
*9600<C600.C6FFM

; save accumulator and processor flags
; (since the bootloader is sensitive to
; both of them)
96F8-   08          PHP
96F9-   48          PHA

; set up callback after decryption loop
96FA-   A9 08       LDA   #$08
96FC-   8D CA 08    STA   $08CA
96FF-   A9 97       LDA   #$97
9701-   8D CB 08    STA   $08CB

; restore flags and accumulator
9704-   68          PLA
9705-   28          PLP

; start the boot
9706-   4C 01 08    JMP   $0801

; callback is here
; copy decrypted boot0 to graphics page
; so it survives a reboot
9709-   A0 00       LDY   #$00
970B-   B9 00 02    LDA   $0200,Y
970E-   99 00 22    STA   $2200,Y
9711-   C8          INY
9712-   D0 F7       BNE   $970B
```

```
; turn off slot 6 drive motor
9714-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9717-   4C 00 C5    JMP   $C500

*BSAVE TRACE1,A$9600,L$11A
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT0 0200-02FF,A$2200,L$100
]CALL -151
```

I'm going to leave this at $2200 for listing. Relative branches will look correct, but absolute addresses will be off by $2000.

```
*2200L

2200-   4C 22 02    JMP   $0222

*2222L

; munge boot slot into $Cx form
2222-   8A          TXA
2223-   4A          LSR
2224-   4A          LSR
2225-   4A          LSR
2226-   4A          LSR
2227-   09 C0       ORA   #$C0

; and stash it somewhere
2229-   8D 1B 02    STA   $021B
```

```
; set up reset vector
222C-   8C F2 03    STY   $03F2
222F-   A9 02       LDA   #$02
2231-   8D F3 03    STA   $03F3
2234-   A9 A7       LDA   #$A7
2236-   8D F4 03    STA   $03F4
```

Y is $03 at this point (set at $08C7), so the reset vector points to $0203. I bet that's unfriendly.

```
*2203L

; The Badlands (from which there is no
; return) -- wipe all of main memory
; and reboot
2203-   A9 04       LDA   #$04
2205-   8D 0F 02    STA   $020F
2208-   A2 BC       LDX   #$BC
220A-   A0 00       LDY   #$00
220C-   98          TYA
220D-   99 00 04    STA   $0400,Y
2210-   88          DEY
2211-   D0 FA       BNE   $220D
2213-   EE 0F 02    INC   $020F
2216-   CA          DEX
2217-   D0 F4       BNE   $220D

; reboot (high byte was set at $0229)
2219-   4C 00 00    JMP   $0000
```

Quite unfriendly, that. Continuing...

```
*2239L

; clear hi-res page 1
2239-   A2 20       LDX   #$20
223B-   A9 00       LDA   #$00
223D-   A8          TAY
223E-   99 00 20    STA   $2000,Y
2241-   88          DEY
2242-   D0 FA       BNE   $223E
2244-   EE 40 02    INC   $0240
2247-   D0 02       BNE   $224B

; This instruction baffled me for the
; longest time, until I realized that
; this code was originally decrypted
; from the boot sector at $0800. Part
; of that boot sector was a series of
; BCS instructions to jump to the end
; of the page. This instruction was
; originally the "BCS $08B9" at $084E.
; It's harmless here; it just slows
; the loop an imperceptible amount.
; I can't imagine how much thought went
; into making this work.
2249-   B1 68       LDA   ($68),Y
224B-   CA          DEX
224C-   D0 F0       BNE   $223E

; read from ROM / write to RAM bank 2
; and show hi-res screen 1 (blank)
224E-   2C 81 C0    BIT   $C081
2251-   2C 50 C0    BIT   $C050
2254-   2C 81 C0    BIT   $C081
2257-   2C 57 C0    BIT   $C057
225A-   2C 52 C0    BIT   $C052
```

```
; copy all of ROM into RAM bank 2
; (defense against boot tracers like
; Watson and modified "F8" ROMs)
225D-   A2 30       LDX   #$30
225F-   B9 00 D0    LDA   $D000,Y
2262-   99 00 D0    STA   $D000,Y
2265-   88          DEY
2266-   D0 F7       BNE   $225F
2268-   EE 61 02    INC   $0261
226B-   EE 64 02    INC   $0264
226E-   CA          DEX
226F-   D0 EE       BNE   $225F

; set low-level reset vector in RAM
; bank 2
2271-   A2 02       LDX   #$02
2273-   8E FD FF    STX   $FFFD
2276-   E8          INX
2277-   8E FC FF    STX   $FFFC

; back to ROM
227A-   2C 80 C0    BIT   $C080

; OK, now to the matter at hand:
; reading from disk
227D-   A6 2B       LDX   $2B

; look for "D5 CC 96" prologue
227F-   BD 8C C0    LDA   $C08C,X
2282-   10 FB       BPL   $227F
2284-   C9 D5       CMP   #$D5
2286-   D0 F7       BNE   $227F
2288-   BD 8C C0    LDA   $C08C,X
228B-   10 FB       BPL   $2288
228D-   C9 CC       CMP   #$CC
228F-   D0 F3       BNE   $2284
2291-   BD 8C C0    LDA   $C08C,X
2294-   10 FB       BPL   $2291
2296-   C9 96       CMP   #$96
2298-   D0 EA       BNE   $2284
```

```
; then immediately start reading 4-4
; encoded data
229A-    A0 00          LDY    #$00
229C-    BD 8C C0       LDA    $C08C,X
229F-    10 FB          BPL    $229C
22A1-    2A             ROL
22A2-    85 FF          STA    $FF
22A4-    BD 8C C0       LDA    $C08C,X
22A7-    10 FB          BPL    $22A4
22A9-    25 FF          AND    $FF

; into $0700
22AB-    99 00 07       STA    $0700,Y
22AE-    88             DEY
22AF-    D0 EB          BNE    $229C

; and jump there
22B1-    4C 00 07       JMP    $0700
```

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

Chapter 3
In Which We Find Ourselves In A Race Against Time, And Losing

```
*9600<C600.C6FFM

; set up callback #1 after boot0
; decrypts itself into $0200 [same as
; previous trace -- won't show again]
96F8-   08          PHP
96F9-   48          PHA
96FA-   A9 08       LDA   #$08
96FC-   8D CA 08    STA   $08CA
96FF-   A9 97       LDA   #$97
9701-   8D CB 08    STA   $08CB
9704-   68          PLA
9705-   28          PLP

; start the boot
9706-   4C 01 08    JMP   $0801

; callback #1 is here
; set up callback #2 after decrypted
; boot0 loads next sector in $0700
9709-   08          PHP
970A-   48          PHA
970B-   A9 1A       LDA   #$1A
970D-   8D B2 02    STA   $02B2
9710-   A9 97       LDA   #$97
9712-   8D B3 02    STA   $02B3
9715-   68          PLA
9716-   28          PLP

; continue the boot
9717-   4C 00 02    JMP   $0200

; callback #2 is here
; copy $0700 to graphics page so it
; survives a reboot
971A-   A0 00       LDY   #$00
971C-   B9 00 07    LDA   $0700,Y
971F-   99 00 27    STA   $2700,Y
9722-   C8          INY
9723-   D0 F7       BNE   $971C
```

```
; turn off slot 6 drive motor and
; reboot to my work disk
9725-   AD E8 C0    LDA   $C0E8
9728-   4C 00 C5    JMP   $C500

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

]BSAVE BOOT1 0700-07FF,A$2700,L$100
]CALL -151
```

Again, I'll need to leave this in the graphics page because I can't list it at $0700. Absolute addresses will be off by $2000.

```
*2700L

2700-   A0 0B       LDY   #$0B
2702-   59 00 07    EOR   $0700,Y
2705-   99 00 07    STA   $0700,Y
2708-   C8          INY
2709-   D0 F7       BNE   $2702
```

Argh, more encryption. And it uses the value of the accumulator coming out of the previous stage as the seed for the progressive EOR.

```
*9600<C600.C6FFM

.
. [identical to previous trace]
.
; callback #1 is here
; set up callback #2 after $0700 is
; loaded
9709-   08              PHP
970A-   48              PHA
970B-   A9 1A           LDA   #$1A
970D-   8D B2 02        STA   $02B2
9710-   A9 97           LDA   #$97
9712-   8D B3 02        STA   $02B3
9715-   68              PLA
9716-   28              PLP

; continue the boot
9717-   4C 00 02        JMP   $0200

; callback #2 is here
; reproduce the decryption loop at
; $0700
971A-   A0 0B           LDY   #$0B
971C-   59 00 07        EOR   $0700,Y
971F-   99 00 07        STA   $0700,Y
9722-   C8              INY
9723-   D0 F7           BNE   $971C

; now copy the decrypted code to the
; graphics page so it survives a reboot
9725-   A0 00           LDY   #$00
9727-   B9 00 07        LDA   $0700,Y
972A-   99 00 27        STA   $2700,Y
972D-   C8              INY
972E-   D0 F7           BNE   $9727

; turn off drive motor and reboot
9730-   AD E8 C0        LDA   $C0E8
9733-   4C 00 C5        JMP   $C500
```

```
*BSAVE TRACE3,A$9600,L$136
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0700-07FF DECRYPTED,
 A$2700,L$100

]CALL -151

*270BL

; push $02 to the stack (hmm)
270B-   A9 02         LDA   #$02
270D-   48            PHA
270E-   A6 2B         LDX   $2B
2710-   A0 06         LDY   #$06

; and again
2712-   48            PHA
```

An "RTS" right now would "return" to the routine at $0203, which wipes all memory and reboots. So, uh, let's not do that.

```
; death counter?
2713-   A9 FF       LDA   #$FF
2715-   8D C2 07    STA   $07C2

; search for "D5 9B AB B2 9E BE"
2718-   BD 88 C0    LDA   $C088,X
271B-   10 FB       BPL   $2718
271D-   C9 D5       CMP   #$D5
271F-   D0 F7       BNE   $2718
2721-   BD 88 C0    LDA   $C088,X
2724-   10 FB       BPL   $2721
2726-   C9 9B       CMP   #$9B
2728-   D0 F3       BNE   $271D
272A-   BD 88 C0    LDA   $C088,X
272D-   10 FB       BPL   $272A
272F-   C9 AB       CMP   #$AB
2731-   D0 EA       BNE   $271D
2733-   BD 88 C0    LDA   $C088,X
2736-   10 FB       BPL   $2733
2738-   C9 B2       CMP   #$B2
273A-   D0 E1       BNE   $271D
273C-   BD 88 C0    LDA   $C088,X
273F-   10 FB       BPL   $273C
2741-   C9 9E       CMP   #$9E
2743-   D0 D8       BNE   $271D
2745-   BD 88 C0    LDA   $C088,X
2748-   10 FB       BPL   $2745
274A-   C9 BE       CMP   #$BE
274C-   D0 CF       BNE   $271D

; skip 6 nibbles (Y=6 at $0710)
274E-   BD 88 C0    LDA   $C088,X
2751-   10 FB       BPL   $274E
2753-   88          DEY
2754-   D0 F8       BNE   $274E
```

```
; first time through, this will become
; $00, so branch will only be taken on
; subsequent loops
2756-   EE C2 07     INC   $07C2
2759-   D0 06        BNE   $2761

; store last read nibble
275B-   8D C3 07     STA   $07C3

; start over
275E-   4C 1D 07     JMP   $071D

; On second and subsequent loops,
; execution continues here from $0759.
; Check if the last nibble we just read
; is the same as the last nibble we
; read the first time through the loop.
2761-   CD C3 07     CMP   $07C3

; "no" is the correct answer
2764-   D0 0B        BNE   $2771

; if they match, try several more times
2766-   EE C2 07     INC   $07C2
2769-   AD C2 07     LDA   $07C2
276C-   C9 08        CMP   #$08
276E-   D0 AD        BNE   $271D

; but eventually give up and "return"
; to The Badlands at $0203.
2770-   60           RTS
```

OK, that wasn't anything useful at all. Also, it's very weird. Like, completely insane. Here's why:

```
   :1   LDA $C088,X
        BPL :1
```

Over and over, to read the nibbles. But
that is not the normal way to read the
data latch. The usual way is

```
   :1    LDA $C08C,X
         BPL :1
```

In fact, accessing $C088,X will turn
off the drive motor. However, due to
the intricacies of the Disk II drive
controller, it takes some time for the
motor to turn off. (It varies, but you
usually have about a second, which is
actually a lot of time.) In the
meantime, it will continue to read
nibbles from the disk if you ask for
them. Once the drive motor is actually
off, it will just return random values.

The prologue ("D5 9B AB B2 9E BE")
looks important, but it's not. What's
important is what comes after it, and
what's being checksummed over and over:
a few sync bytes ($FF), then a long
sequence of zero bits. Because that is
what is actually on the original disk:
nothing.

When we say a "zero bit," we really
mean "the lack of a magnetic state
change." If the Disk II doesn't see a
state change in a certain period of
time, it calls that a "0". If it does
see a change, it calls that a "1". But
the drive can only tolerate a lack of
state changes for so long -- about as
long as it takes for two bits to go by.

Fun fact(*): this is why you need to use nibbles as an intermediate on-disk format in the first place. No valid nibble contains more than two zero bits consecutively, when written from most-significant to least-significant bit.

So what happens when a drive doesn't see a state change after the equivalent of two consecutive zero bits? The drive thinks the disk is weak, and it starts increasing the amplification to try to compensate, looking for a valid signal. But there is no signal. There is no data. There is just a yawning abyss of nothingness. Eventually, the drive gets desperate and amplifies so much that it starts returning random bits based on ambient noise from the disk motor and the magnetism of the Earth.

Seriously.

Returning random bits doesn't sound very useful for a storage medium, but it's exactly what the developer wanted, and that's exactly what this code is checking for. It's finding and reading and checksumming the same sequence of bits from the disk, over and over, and checking that they differ.

(*) not guaranteed, actual fun may vary

Bit copiers will never duplicate the long sequence of zero bits, because that's not what they read. Whatever randomness they get when they read the original disk will essentially get "frozen" onto the copy. On a copy, this code loops repeatedly, since the nibbles after the prologue are always the same. The BNE branch at $0764 is never taken, so we keep branching back to $071D (from $076E). Eventually, the drive motor turns off and we end up in an infinite loop looking for the prologue on a drive that's not even reading the disk.

Even on the original disk, it turns off the drive motor on purpose before it even starts. Not only do we have to pass the protection check, we have to do it before the disk stops spinning. The entire bootloader is a race against time, and copies always lose.

Chapter 4
You're Very Clever, Young Man, But It's Encryption All The Way Down

Continuing from $0771, after the copy protection passes...

```
; execution continues here (from $0764)
; pop The Badlands off the stack
2771-   68              PLA

; stores $02 in zero page $FD
2772-   85 FD           STA   $FD
2774-   68              PLA

; stores $04 in zero page $FB
2775-   0A              ASL
2776-   85 FB           STA   $FB

; what have we here?
2778-   A0 85           LDY   #$85
277A-   59 00 07        EOR   $0700,Y
277D-   99 00 07        STA   $0700,Y
2780-   C8              INY
2781-   D0 F7           BNE   $277A
2783-   84 FC           STY   $FC
```

Another progressive decryption loop. Why God why. I should have taken up basket weaving.

```
*9600<C600.C6FFM


.
. [identical to previous trace]
.
; reproduce second decryption loop
; ($070B..$07FF)
971A-   A0 0B           LDY   #$0B
971C-   59 00 07        EOR   $0700,Y
971F-   99 00 07        STA   $0700,Y
9722-   C8              INY
9723-   D0 F7           BNE   $971C
```

```
; reproduce the third decryption loop
; ($0785..$07FF)
9725-   A9 04       LDA   #$04
9727-   A0 85       LDY   #$85
9729-   59 00 07    EOR   $0700,Y
972C-   99 00 07    STA   $0700,Y
972F-   C8          INY
9730-   D0 F7       BNE   $9729

; copy the double-decrypted code to
; higher memory so it survives a reboot
9732-   A0 00       LDY   #$00
9734-   B9 00 07    LDA   $0700,Y
9737-   99 00 27    STA   $2700,Y
973A-   C8          INY
973B-   D0 F7       BNE   $9734

; turn off drive motor and reboot
973D-   AD E8 C0    LDA   $C0E8
9740-   4C 00 C5    JMP   $C500

*BSAVE TRACE4,A$9600,L$143
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0700-07FF DECRYPTED 2,
 A$2700,L$100
```

If this continues, I'm going to need a new naming convention.

```
]CALL -151

*2785L

; look for "D5 BD 96" prologue
2785-   BD 88 C0     LDA   $C088,X
2788-   10 FB        BPL   $2785
278A-   C9 D5        CMP   #$D5
278C-   D0 F7        BNE   $2785
278E-   BD 88 C0     LDA   $C088,X
2791-   10 FB        BPL   $278E
2793-   C9 BD        CMP   #$BD
2795-   D0 F3        BNE   $278A
2797-   BD 88 C0     LDA   $C088,X
279A-   10 FB        BPL   $2797
279C-   C9 96        CMP   #$96
279E-   D0 EA        BNE   $278A

; then immediately start reading 4-4
; encoded data
27A0-   BD 88 C0     LDA   $C088,X
27A3-   10 FB        BPL   $27A0
27A5-   2A           ROL
27A6-   85 FF        STA   $FF
27A8-   BD 88 C0     LDA   $C088,X
27AB-   10 FB        BPL   $27A8
27AD-   25 FF        AND   $FF

; and storing it at $0200 ($FC was set
; to $00 at $0783, $FD was set to $02
; at $0772)
27AF-   91 FC        STA   ($FC),Y
27B1-   C8           INY
27B2-   D0 EC        BNE   $27A0

; one nibble delimiter
27B4-   BD 88 C0     LDA   $C088,X
27B7-   10 FB        BPL   $27B4
27B9-   E6 FD        INC   $FD
```

```
; four pages worth ($FB was set to $04
; at $0776)
27BB-    C6 FB          DEC    $FB
27BD-    D0 E1          BNE    $27A0

; and continue there
27BF-    4C 00 02       JMP    $0200
```

And round and round we go.

Chapter 5
In Which We Find An RWTS Of A Most Curious Nature

```
*9600<C600.C6FFM
.
. [identical to previous trace]
.
; reproduce second decryption loop
; ($070B..$07FF) [will not show again]
971A-   A0 0B       LDY   #$0B
971C-   59 00 07    EOR   $0700,Y
971F-   99 00 07    STA   $0700,Y
9722-   C8          INY
9723-   D0 F7       BNE   $971C

; set up callback #3 after nibble check
9725-   08          PHP
9726-   48          PHA
9727-   A9 4C       LDA   #$4C
9729-   8D 78 07    STA   $0778
972C-   A9 3B       LDA   #$3B
972E-   8D 79 07    STA   $0779
9731-   A9 97       LDA   #$97
9733-   8D 7A 07    STA   $077A
9736-   68          PLA
9737-   28          PLP

; continue the boot
9738-   4C 0B 07    JMP   $070B

; callback #3 is here
; reproduce third decryption loop (on
; the second part of page 7, starting
; at $0785)
973B-   A0 85       LDY   #$85
973D-   59 00 07    EOR   $0700,Y
9740-   99 00 07    STA   $0700,Y
9743-   C8          INY
9744-   D0 F7       BNE   $973D
```

```
; set up callback #4 after it loads the
; next phase at $0200..$05FF
9746-   08          PHP
9747-   48          PHA
9748-   A9 4C       LDA   #$4C
974A-   8D BF 07    STA   $07BF
974D-   A9 5C       LDA   #$5C
974F-   8D C0 07    STA   $07C0
9752-   A9 97       LDA   #$97
9754-   8D C1 07    STA   $07C1
9757-   68          PLA
9758-   28          PLP

; continue the boot
9759-   4C 83 07    JMP   $0783

; callback #4 is here
; copy everything from low memory
; ($0200..$05FF) to graphics page so it
; survives a reboot
975C-   A2 04       LDX   #$04
975E-   A0 00       LDY   #$00
9760-   B9 00 02    LDA   $0200,Y
9763-   99 00 22    STA   $2200,Y
9766-   C8          INY
9767-   D0 F7       BNE   $9760
9769-   EE 62 97    INC   $9762
976C-   EE 65 97    INC   $9765
976F-   CA          DEX
9770-   D0 EE       BNE   $9760

; turn off slot 6 drive motor and
; reboot to my work disk
9772-   AD E8 C0    LDA   $C0E8
9775-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE5,A$9600,L$178
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0200-05FF,A$2200,L$400
]CALL -151

*2200L

2200-   4C 1C 02    JMP   $021C

; The Badlands (again)
2203-   A2 BC       LDX   #$BC
2205-   A9 04       LDA   #$04
2207-   8D 0F 02    STA   $020F
220A-   A9 00       LDA   #$00
220C-   A8          TAY
220D-   99 00 04    STA   $0400,Y
2210-   88          DEY
2211-   D0 FA       BNE   $220D
2213-   EE 0F 02    INC   $020F
2216-   CA          DEX
2217-   D0 F4       BNE   $220D
2219-   4C 00 C6    JMP   $C600

; execution continues here (from $0200)
221C-   4C 6E 04    JMP   $046E
```

I'm going to go back to the routine at $046E later. It will make more sense once we see how the code on page 2 is organized. For now, just pretend that JMP doesn't exist and that execution continues on the next line.

```
; if X != 5 on entry then skip the next
; few instructions
221F-   E0 05          CPX   #$05
2221-   D0 0D          BNE   $2230

; this is the slot number (x16) (set
; during the one-time initialization at
; $046E that we haven't looked at yet,
; so you'll just have to trust me)
2223-   A6 5D          LDX   $5D

; switch graphics page 2
2225-   2C 55 C0       BIT   $C055

; turn on the drive motor
2228-   BD 89 C0       LDA   $C089,X

; This is an exact duplicate of the
; nibble check we just ran at $0700. It
; looks for the "D5 9B AB B2 9E BE"
; prologue then checks that the nibbles
; after it differ every time they're
; read. It jumps to The Badlands on
; failure and returns gracefully on
; success.
222B-   20 D3 03       JSR   $03D3

; restore X to its original value (5)
222E-   A2 05          LDX   #$05
```

So if X=5 on entry, we do another nibble check, then we do whatever we were supposed to be doing, which starts at $0230.

```
; execution continues here regardless
2230-   2C 8E D0       BIT   $D08E
2233-   8E 6D 04       STX   $046D
2236-   AE 6D 04       LDX   $046D
```

```
; use X as an index into several arrays
2239-   BD 51 04     LDA   $0451,X
223C-   48           PHA
223D-   BD 58 04     LDA   $0458,X
2240-   85 5E        STA   $5E
2242-   BD 5F 04     LDA   $045F,X
2245-   A8           TAY
2246-   BD 66 04     LDA   $0466,X
2249-   AA           TAX
224A-   68           PLA

A is set from array at $0451
$5E is set from array at $0458
Y is set from array at $045F
X is set from array at $0466

; call routine with these parameters
224B-   20 D8 02     JSR   $02D8

; keep trying until carry is clear
; coming out of $02D8
224E-   B0 E6        BCS   $2236

; turn off drive motor and exit via RTS
2250-   9D 88 C0     STA   $C088,X
2253-   60           RTS

Here are the arrays:

2451- 02 04 24 2C 18 04 36    ; --> A
2458- 00 00 00 00 01 00 00    ; --> $5E
245F- 06 B9 26 35 35 39 05    ; --> Y
2466- 08 07 80 80 80 07 E0    ; --> X

For example, if X=2 on entry at $0200,
then the subroutine at $02D8 is called
with A=$24, $5E=$00, Y=$26, and X=$80.
```

Now let's see how they're used.

```
*22D8L

; save parameters
22D8-   85 5C       STA   $5C
22DA-   86 6E       STX   $6E
22DC-   A2 00       LDX   #$00
22DE-   86 6D       STX   $6D
22E0-   84 57       STY   $57
22E2-   A5 5C       LDA   $5C

; slot number (x16)
22E4-   A6 5D       LDX   $5D

; turn on drive motor
22E6-   DD 8E C0    CMP   $C08E,X
22E9-   DD 89 C0    CMP   $C089,X

; moves drive head to specified phase
; given in accumulator [not shown]
22EC-   20 54 02    JSR   $0254
22EF-   4C 00 03    JMP   $0300
...
```

```
; look for track prologue
; "CC AA B5 96 DE"
2300-   38              SEC
2301-   BD 8C C0        LDA   $C08C,X
2304-   10 FB           BPL   $2301
2306-   C9 CC           CMP   #$CC
2308-   D0 F6           BNE   $2300
230A-   BD 8C C0        LDA   $C08C,X
230D-   10 FB           BPL   $230A
230F-   C9 AA           CMP   #$AA
2311-   D0 F3           BNE   $2306
2313-   BD 8C C0        LDA   $C08C,X
2316-   10 FB           BPL   $2313
2318-   C9 B5           CMP   #$B5
231A-   D0 E4           BNE   $2300
231C-   BD 8C C0        LDA   $C08C,X
231F-   10 FB           BPL   $231C
2321-   C9 96           CMP   #$96
2323-   D0 DB           BNE   $2300
2325-   BD 8C C0        LDA   $C08C,X
2328-   10 FB           BPL   $2325
232A-   C9 DE           CMP   #$DE
232C-   D0 D2           BNE   $2300

; look for sector prologue "AD *",
; where the second value is taken from
; the array at $03C7
232E-   A4 5E           LDY   $5E
```

```
; I think zero page $55 ends up with
; the sector number...
2330-   A9 0D       LDA   #$0D
2332-   85 55       STA   $55

; ...because this looks for the
; sector prologue and decrements $55
; until it finds the right one (so the
; sector prologue is different for
; each sector)
2334-   BD 8C C0    LDA   $C08C,X
2337-   10 FB       BPL   $2334
2339-   C9 AD       CMP   #$AD
233B-   D0 F7       BNE   $2334
233D-   C6 55       DEC   $55
233F-   BD 8C C0    LDA   $C08C,X
2342-   10 FB       BPL   $233F
2344-   D9 C7 03    CMP   $03C7,Y
2347-   D0 EB       BNE   $2334
2349-   F0 15       BEQ   $2360
...

; now read the sector data
2360-   A9 75       LDA   #$75
2362-   85 56       STA   $56
2364-   A0 00       LDY   #$00

; sector data is 4-4 encoded
2366-   BD 8C C0    LDA   $C08C,X
2369-   10 FB       BPL   $2366
236B-   2A          ROL
236C-   85 5F       STA   $5F
236E-   BD 8C C0    LDA   $C08C,X
2371-   10 FB       BPL   $236E
2373-   25 5F       AND   $5F
2375-   85 54       STA   $54

; sector data is encrypted
2377-   45 56       EOR   $56
```

```
; store it in ($6D) (set from X on
; entry, at $02DA)
2379-   91 6D          STA   ($6D),Y
237B-   C8             INY

; do it again (main loop decodes two
; bytes per pass -- maybe timing was
; tight?)
237C-   BD 8C C0       LDA   $C08C,X
237F-   10 FB          BPL   $237C
2381-   2A             ROL
2382-   85 5F          STA   $5F
2384-   BD 8C C0       LDA   $C08C,X
2387-   10 FB          BPL   $2384
2389-   25 5F          AND   $5F
238B-   85 56          STA   $56
238D-   45 54          EOR   $54
238F-   91 6D          STA   ($6D),Y

; loop until done with this sector
2391-   C8             INY
2392-   D0 D2          BNE   $2366

; increase target page
2394-   E6 6E          INC   $6E

; checksum byte
2396-   BD 8C C0       LDA   $C08C,X
2399-   10 FB          BPL   $2396
239B-   2A             ROL
239C-   85 5F          STA   $5F
239E-   BD 8C C0       LDA   $C08C,X
23A1-   10 FB          BPL   $239E
23A3-   25 5F          AND   $5F
23A5-   45 56          EOR   $56

; if checksum fails, try again
23A7-   D0 19          BNE   $23C2
```

```
; decrement sector count (set from Y on
; entry, at $02E0)
23A9-   C6 57        DEC   $57
23AB-   F0 11        BEQ   $23BE

; increment the index into the array
; at $03C7 that determines the second
; nibble of the sector prologue
23AD-   E6 5E        INC   $5E

; decrement the sector number
23AF-   C6 55        DEC   $55

; loop back to read more
23B1-   D0 98        BNE   $234B

; done with all the sectors on this
; track, so reset the sector number,
; increment the phase by 2 (in $5C,
; set from A on entry at $02D8), and
; jump back to advance the drive head
; and keep reading from the next track
23B3-   A9 00        LDA   #$00
23B5-   85 5E        STA   $5E
23B7-   E6 5C        INC   $5C
23B9-   E6 5C        INC   $5C
23BB-   4C E2 02     JMP   $02E2
23BE-   18           CLC
23BF-   60           RTS
23C0-   C6 5E        DEC   $5E
23C2-   C6 6E        DEC   $6E
23C4-   4C 00 03     JMP   $0300
23C7-  [97 9A 9B 9D 9E 9F CB CD]
23CF-  [CE CF D3 D6]
```

```
Here's what I know so far:

- data is stored on whole tracks
- tracks have a 5-nibble prologue
  (same on each track)
- tracks are split up into sectors
  (but not 16 -- probably 12 because
  of the 4-4 encoding)
- each sector is 256 bytes
- sectors have a 2-nibble prologue
  (different for each sector on the
  track, but they follow the same
  pattern from track to track)
- sector data is 4-4 encoded
- sector data is encrypted with a
  rolling XOR
- $02D8 is the entry point to read
  multiple sectors into memory
  on entry, A = phase (track x2)
            X = start address (high)
            Y = sector count
            $5E = starting sector
            (usually 0, but is 1 if X=4
            on the main entry at $0200)
- $0230 is a higher level entry point
  that takes an index in the X register
  and looks up how to read a block
  (using the arrays at $0451, $0458,
  $045F, and $0466)
- there are 7 blocks (X=0-6)
- $0200 is the highest level entry
  point; it functions identically to
  $0230, but if X=5 then it calls $03D3
  first to do a nibble check

And now we're ready to look at the one-
time initialization routine at $046E.
```

Chapter 6
In Which All Apples Are Equal But Some Are More Equal Than Others

```
*246EL

; save slot number (x16) in zero page
246E-   A5 2B       LDA   $2B
2470-   85 5D       STA   $5D

; and set up the reboot instruction in
; The Badlands
2472-   4A          LSR
2473-   4A          LSR
2474-   4A          LSR
2475-   4A          LSR
2476-   09 C0       ORA   #$C0
2478-   8D 1B 02    STA   $021B

; NOP out the jump that called this
; routine (at $021C) -- so this entire
; routine will only ever be called once
; (specifically, the first time that
; $0200 is called with any value of X)
247B-   A9 EA       LDA   #$EA
247D-   8D 1C 02    STA   $021C
2480-   8D 1D 02    STA   $021D
2483-   8D 1E 02    STA   $021E

; check if language card is available
2486-   2C 83 C0    BIT   $C083
2489-   2C 83 C0    BIT   $C083
248C-   A9 00       LDA   #$00
248E-   85 CC       STA   $CC
2490-   A9 82       LDA   #$82
2492-   8D 00 E0    STA   $E000
2495-   CD 00 E0    CMP   $E000
2498-   D0 07       BNE   $24A1

; language card is available -- put $82
; in zero page $CC
249A-   85 CC       STA   $CC
```

```
; and call this (more on this later)
249C-   A2 06       LDX   #$06
249E-   20 33 02    JSR   $0233

; execution continues here regardless
; of whether language card is available

; push $0B/$FF on the stack (perhaps so
; we can "return" to $0C00? there's
; nothing there yet, though)
24A1-   A9 0B       LDA   #$0B
24A3-   48          PHA
24A4-   A9 FF       LDA   #$FF
24A6-   48          PHA

; check for //e or later
24A7-   AD B3 FB    LDA   $FBB3
24AA-   C9 06       CMP   #$06
24AC-   F0 05       BEQ   $24B3

; no //e, continue at $0233 with X=0
24AE-   A2 00       LDX   #$00
24B0-   4C 33 02    JMP   $0233

; found //e or later, so re-enable the
; language card and copy some code
; there
24B3-   2C 83 C0    BIT   $C083
24B6-   2C 83 C0    BIT   $C083
24B9-   A2 00       LDX   #$00
24BB-   BD CD 04    LDA   $04CD,X
24BE-   9D 00 D0    STA   $D000,X
24C1-   BD CD 05    LDA   $05CD,X
24C4-   9D 00 D1    STA   $D100,X
24C7-   CA          DEX
24C8-   D0 F1       BNE   $24BB
```

```
; and continue with the code we just
; copied
24CA-   4C 00 D0     JMP   $D000
```

I want to see this code in place, so here's a little program to help.

```
; write to RAM bank 2
0300-   AD 81 C0     LDA   $C081
0303-   AD 81 C0     LDA   $C081

; copy everything (so monitor routines
; will still work)
0306-   A2 30        LDX   #$30
0308-   A0 00        LDY   #$00
030A-   B9 00 D0     LDA   $D000,Y
030D-   99 00 D0     STA   $D000,Y
0310-   C8           INY
0311-   D0 F7        BNE   $030A
0313-   EE 0C 03     INC   $030C
0316-   EE 0F 03     INC   $030F
0319-   CA           DEX
031A-   D0 EE        BNE   $030A

; reproduce the copy loop at $04B9
031C-   B9 CD 24     LDA   $24CD,Y
031F-   99 00 D0     STA   $D000,Y
0322-   B9 CD 25     LDA   $25CD,Y
0325-   99 00 D1     STA   $D100,Y
0328-   C8           INY
0329-   D0 F1        BNE   $031C
032B-   60           RTS

*300G
```

Now I can interactively switch over to full read/write access of the language card, and the monitor won't crash. Hooray!

```
*C083 C083
*D000L

; check for auxiliary memory (128K)
D000-   8D 05 C0    STA   $C005
D003-   A9 09       LDA   #$09
D005-   8D 00 80    STA   $8000
D008-   A9 23       LDA   #$23
D00A-   8D 01 80    STA   $8001
D00D-   0E 00 84    ASL   $8400
D010-   0E 01 84    ASL   $8401
D013-   8D 04 C0    STA   $C004
D016-   8D 03 C0    STA   $C003
D019-   AD 00 80    LDA   $8000
D01C-   C9 09       CMP   #$09
D01E-   D0 0A       BNE   $D02A
D020-   AD 01 80    LDA   $8001
D023-   C9 23       CMP   #$23
D025-   8D 02 C0    STA   $C002
D028-   F0 06       BEQ   $D030
D02A-   8D 02 C0    STA   $C002

; no usable auxiliary memory, jump back
; to the routine in main memory which
; called $0200 with X=0
D02D-   4C AE 04    JMP   $04AE

; execution continues here (from $D028)
; we found 128K, and now we're going to
; use it
D030-   A0 02       LDY   #$02
D032-   8C 6D 04    STY   $046D

; get X from an array at $D0C9
D035-   BE C9 D0    LDX   $D0C9,Y

Here is the array of $D0C9:

D0C9-  [02 03 04]
```

```
; set up a call to the blockread
; routine at $02D8
D038-   BD 58 04     LDA   $0458,X
D03B-   85 5E        STA   $5E
D03D-   BC 5F 04     LDY   $045F,X
D040-   BD 51 04     LDA   $0451,X

; but always load into $7000
D043-   A2 70        LDX   #$70
D045-   8E 61 D0     STX   $D061

; read from disk
D048-   20 D8 02     JSR   $02D8
```

So we get a block index from the array at $D0C9, then we do a modified disk read to store that block at $7000.

```
D04B-   AC 6D 04     LDY   $046D

; get block index again
D04E-   BE C9 D0     LDX   $D0C9,Y

; set up a memory copy based on another
; array at $D0C3
D051-   BD C3 D0     LDA   $D0C3,X
D054-   8D 64 D0     STA   $D064

; this is the sector count from the
; blockread we just called
D057-   BC 5F 04     LDY   $045F,X
D05A-   A2 00        LDX   #$00

; write to auxiliary memory
D05C-   8D 05 C0     STA   $C005
```

```
; memory copy loop
D05F-    BD 00 70     LDA   $7000,X
D062-    9D 00 70     STA   $7000,X
D065-    CA           DEX
D066-    D0 F7        BNE   $D05F
D068-    EE 61 D0     INC   $D061
D06B-    EE 64 D0     INC   $D064
D06E-    88           DEY
D06F-    D0 EE        BNE   $D05F
D071-    AC 6D 04     LDY   $046D

; write to main memory again
D074-    8D 04 C0     STA   $C004

; do it for all the sectors we read
D077-    88           DEY
D078-    10 B8        BPL   $D032
D07A-    A6 5D        LDX   $5D

; move drive head back to track 0
D07C-    A9 00        LDA   #$00
D07E-    20 54 02     JSR   $0254

; change the BIT instruction at $0230
; to a JMP, so it jumps to $D08E
D081-    A9 4C        LDA   #$4C
D083-    8D 30 02     STA   $0230

; turn off drive motor
D086-    9D 88 C0     STA   $C088,X

; call the RWTS with X=0
D089-    A2 00        LDX   #$00
D08B-    4C 33 02     JMP   $0233
```

Here are the arrays:

```
D0C3-  [FF FF 02 28 5D FF] ;target addr
                           ;in auxmem

D0CC-  [00 00 26 35 35 00] ;page count

D0D2-  [FF FF 80 80 80 FF] ;target addr
                           ;in main mem
```

This is a preloader. It has an array of block indexes (at $D0C9 -- so blocks 2, 3, and 4) that it loads from disk and stores in auxiliary memory. Then it modifies the RWTS entry point at $0230 to jump to $D08E instead. I'm guessing that routine will check if the block is cached and copy it back from auxiliary memory if it can.

```
*D08EL

; read/write access to language card
D08E-   AD 83 C0    LDA   $C083
D091-   AD 83 C0    LDA   $C083

; check if block (given in X) is one of
; the ones we cached
D094-   BC CC D0    LDY   $D0CC,X
D097-   D0 03       BNE   $D09C

; not cached, continue with disk read
; as usual
D099-   4C 33 02    JMP   $0233
```

```
; yes it's cached
; set up the copy from its address in
; auxiliary memory to its address in
; main memory
D09C-    BD D2 D0     LDA   $D0D2,X
D09F-    8D B2 D0     STA   $D0B2
D0A2-    BD C3 D0     LDA   $D0C3,X
D0A5-    8D AF D0     STA   $D0AF
D0A8-    A2 00        LDX   #$00

; read from auxiliary memory
D0AA-    8D 03 C0     STA   $C003

; copy loop
D0AD-    BD 00 FF     LDA   $FF00,X
D0B0-    9D 00 FF     STA   $FF00,X
D0B3-    CA           DEX
D0B4-    D0 F7        BNE   $D0AD
D0B6-    EE AF D0     INC   $D0AF
D0B9-    EE B2 D0     INC   $D0B2
D0BC-    88           DEY
D0BD-    D0 EE        BNE   $D0AD

; read from main memory again
D0BF-    8D 02 C0     STA   $C002
D0C2-    60           RTS
```

Fun fact: one nice side effect of putting this code in the language card is that it's unaffected by the $C002/ $C003/$C004/ $C005 switches to read and write from main or auxiliary memory. You can even have self-modifying copy loops without having to switch back and forth between main and auxiliary memory inside the loop.

And that really was a fun fact.

Chapter 7
In Which We Experiment With Half-Measures

I've traced the entire boot process,
and it's insane. Here's the flow of
execution:

```
1. $0800 ---decrypts---> $0200
2. $0200 ---reads disk---> $0700
3. $0700 ---decrypts--> $0700
4. <nibble check>
5. $0700 ---decrypts---> $0700 (again)
6. $0700 ---reads disk---> $0200..$05FF
7. $046E ---copies---> $D000..$D1FF
8. $D000 ---reads disk---> aux mem
9. $0C00 starts the game
```

Of the first six steps, very little is
actually useful. Clearing the graphics
screen, copying ROM to RAM, and reading
the RWTS into $0200. That's it. I could
probably fit all of that into a single
sector and still have room for a hidden
message about how inexpensive various
things are.

Let's do that. Starting with my failed EDD bit copy, I can replace T00,S00 with just the useful parts of the original boot:

```
; clear hi-res screen and show it
; (originally at $0239 during boot)
0801-    A2 20          LDX   #$20
0803-    A9 00          LDA   #$00
0805-    A8             TAY
0806-    99 00 20       STA   $2000,Y
0809-    88             DEY
080A-    D0 FA          BNE   $0806
080C-    EE 08 08       INC   $0808
080F-    CA             DEX
0810-    D0 F4          BNE   $0806
0812-    2C 81 C0       BIT   $C081
0815-    2C 50 C0       BIT   $C050
0818-    2C 81 C0       BIT   $C081
081B-    2C 57 C0       BIT   $C057
081E-    2C 52 C0       BIT   $C052

; copy ROM to language card
; (originally at $025D during boot)
0821-    A2 30          LDX   #$30
0823-    B9 00 D0       LDA   $D000,Y
0826-    99 00 D0       STA   $D000,Y
0829-    88             DEY
082A-    D0 F7          BNE   $0823
082C-    EE 25 08       INC   $0825
082F-    EE 28 08       INC   $0828
0832-    CA             DEX
0833-    D0 EE          BNE   $0823
0835-    2C 80 C0       BIT   $C080
```

```
; read RWTS into $0200..$05FF
; (originally at $0785 during boot)
0838-   84 FC       STY   $FC
083A-   A9 02       LDA   #$02
083C-   85 FD       STA   $FD
083E-   0A          ASL
083F-   85 FB       STA   $FB
0841-   A6 2B       LDX   $2B
0843-   BD 8C C0    LDA   $C08C,X
0846-   10 FB       BPL   $0843
0848-   C9 D5       CMP   #$D5
084A-   D0 F7       BNE   $0843
084C-   BD 8C C0    LDA   $C08C,X
084F-   10 FB       BPL   $084C
0851-   C9 BD       CMP   #$BD
0853-   D0 F3       BNE   $0848
0855-   BD 8C C0    LDA   $C08C,X
0858-   10 FB       BPL   $0855
085A-   C9 96       CMP   #$96
085C-   D0 EA       BNE   $0848
085E-   BD 8C C0    LDA   $C08C,X
0861-   10 FB       BPL   $085E
0863-   2A          ROL
0864-   85 FF       STA   $FF
0866-   BD 8C C0    LDA   $C08C,X
0869-   10 FB       BPL   $0866
086B-   25 FF       AND   $FF
086D-   91 FC       STA   ($FC),Y
086F-   C8          INY
0870-   D0 EC       BNE   $085E
0872-   BD 8C C0    LDA   $C08C,X
0875-   10 FB       BPL   $0872
0877-   E6 FD       INC   $FD
0879-   C6 FB       DEC   $FB
087B-   D0 E1       BNE   $085E
```

```
; disable in-RWTS nibble check by
; putting an "RTS" at $03D3
087D-    A9 60          LDA    #$60
087F-    8D D3 03       STA    $03D3

; start the game
0882-    4C 00 02       JMP    $0200
```

Turning to my trusty Disk Fixer sector editor, I typed all that into T00,S00 and wrote it to disk.

```
]PR#6
...works...
```

Call that "Mr. Do (4am crack).nib".

I'm not done yet, but this proves that I understand enough about the copy protection to bypass it and enough about the bootloader to replace it. The resulting disk is easily copyable with any bit copier. I also imaged it as a .nib file, which works in modern emulators.

Chapter 8
In Which We Examine Our Options

Not content with a bit-copyable crack, it is now time to consider my next steps. A straight track-by-track conversion is out of the question; the original uses a 4-4 encoding scheme with custom prologues. I'll need to reconstruct the disk from scratch. I'm reasonably confident that I can use the disk's own RWTS to read each of the blocks (passing X=0-6 into $0200 and capturing the result in memory).

Here's the problem: this is not a single-load game. Even with 128K and the built-in preloader, the original still accesses the disk to reload the main menu after you finish a game. (This is block 5.) The preloader caches blocks 2, 3, and 4 in auxiliary memory, but that only leaves $2E00 for block 5, which is $3900. (Block 0 is only loaded once to calibrate the joystick and show the credits. Block 6 is at $E000.) I might be able to split up the block and cache part of it in the language card, assuming that's not used for storage during the game.

If I can't fit everything in memory, then I'll need an RWTS. I could dictate that my crack requires 64K and stash a DOS-shaped RWTS in the language card. But several parts of the language card are already in use ($D000 for the preloader and cache routines, and six pages at $E000 that are loaded in block 6). Those may or may not be easy to relocate. Also, I'd need to be careful about which text page addresses are in use, because the DOS 3.3 RWTS uses the "screen holes" in the text page for temporary storage.

Another option: leave the RWTS in low memory but rewrite it to support 16-sector tracks. Looking at the blockread arrays, block 1 starts at $0700 and goes all the way up to $BFFF. Further investigation reveals that the game uses pages 5, 6, and most of the stack for data. (This "investigation" consisted of me naively assuming those pages were available and trying to put code there, then cursing profusely when they got overwritten.) So, I would need an RWTS that is relocatable to $0200 and fits in three pages of memory... including temporary storage used during denibblizing.

Memory is extremely tight, but I like the last option. It would allow my crack to run on all the same machines as the original game (48K, 64K, 128K), while taking full advantage of each.

Chapter 9
In Which We Attempt To Use The Disk As A Weapon Against Itself

I need to write a boot tracer that will boot the original disk right up until it jumps to the game, then break into the monitor. But not just that; I need to modify it so it reads every possible block (0-6).

Here is the heart of it, inside the one-time initialization routine that starts at $048E:

```
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``                                    .,
``   04A1-   A9 0B       LDA   #$0B   .,
``   04A3-   48          PHA          .,
``   04A4-   A9 FF       LDA   #$FF   .,
``   04A6-   48          PHA          .,
``   04A7-   AD B3 FB    LDA   $FBB3  .,
``   04AA-   C9 06       CMP   #$06   .,
``   04AC-   F0 05       BEQ   $04B3  .,
``   04AE-   A2 00       LDX   #$00   .,
``   04B0-   4C 33 02    JMP   $0233  .,
``                                    .,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
``'-.,_,.-'``'-.,_,.='``'-.,_,.-'``'-.,
```

I can control the values pushed on the stack at $04A3 and $04A6, so I can break to the monitor instead of jumping to $0C00 to start the game. And I can control which block is read by changing the value of X (at $04AF) going into the RWTS entry point at $0233. (I'll also need to disable the branch at $04AC so it always falls through to immediately read the block I want.)

```
$04A2 --> $FF (pushed to stack)
$0435 --> $58 (pushed to stack)
$04AC --> $24 (never branch over $04AE)
$04AF --> $00..$06 (block to read)
```

Let's do it.

```
*9600<C600.C6FFM

; set up callback #1
96F8-   08          PHP
96F9-   48          PHA
96FA-   A9 08       LDA   #$08
96FC-   8D CA 08    STA   $08CA
96FF-   A9 97       LDA   #$97
9701-   8D CB 08    STA   $08CB
9704-   68          PLA
9705-   28          PLP

; start the boot
9706-   4C 01 08    JMP   $0801

; callback #1 is here
; set up callback #2
9709-   08          PHP
970A-   48          PHA
970B-   A9 1A       LDA   #$1A
970D-   8D B2 02    STA   $02B2
9710-   A9 97       LDA   #$97
9712-   8D B3 02    STA   $02B3
9715-   68          PLA
9716-   28          PLP

; continue the boot
9717-   4C 00 02    JMP   $0200
```

```
; callback #2 is here
; decrypt page 7
971A-   A0 0B       LDY   #$0B
971C-   59 00 07    EOR   $0700,Y
971F-   99 00 07    STA   $0700,Y
9722-   C8          INY
9723-   D0 F7       BNE   $971C

; set up callback #3
9725-   08          PHP
9726-   48          PHA
9727-   A9 4C       LDA   #$4C
9729-   8D 78 07    STA   $0778
972C-   A9 3B       LDA   #$3B
972E-   8D 79 07    STA   $0779
9731-   A9 97       LDA   #$97
9733-   8D 7A 07    STA   $077A
9736-   68          PLA
9737-   28          PLP

; continue the boot
9738-   4C 0B 07    JMP   $070B

; callback #3 is here
; decrypt page 7 (again)
973B-   A0 85       LDY   #$85
973D-   59 00 07    EOR   $0700,Y
9740-   99 00 07    STA   $0700,Y
9743-   C8          INY
9744-   D0 F7       BNE   $973D
```

```
; set up callback #4
9746-   08          PHP
9747-   48          PHA
9748-   A9 4C       LDA   #$4C
974A-   8D BF 07    STA   $07BF
974D-   A9 5C       LDA   #$5C
974F-   8D C0 07    STA   $07C0
9752-   A9 97       LDA   #$97
9754-   8D C1 07    STA   $07C1
9757-   68          PLA
9758-   28          PLP

; continue the boot
9759-   4C 83 07    JMP   $0783

; callback #4 is here
; disable branch at $04AC
975C-   A9 24       LDA   #$24
975E-   8D AC 04    STA   $04AC

; set return address to break to the
; monitor after reading the block
9761-   A9 FF       LDA   #$FF
9763-   8D A2 04    STA   $04A2
9766-   A9 58       LDA   #$58
9768-   8D A5 04    STA   $04A5

; continue the boot
976B-   4C 00 02    JMP   $0200

*BSAVE TRACE6,A$9600,L$16E
*9600G
...reboots slot 6...
<beep>
```

```
*C00L

0C00-   AD 55 C0    LDA   $C055
0C03-   AD 52 C0    LDA   $C052
0C06-   AD 51 C0    LDA   $C051
0C09-   A2 28       LDX   #$28
0C0B-   20 A8 FC    JSR   $FCA8
0C0E-   CA          DEX
0C0F-   10 FA       BPL   $0C0B
0C11-   A9 00       LDA   #$00
0C13-   85 BF       STA   $BF
0C15-   85 3E       STA   $3E
0C17-   A9 08       LDA   #$08
0C19-   85 41       STA   $41
0C1B-   A9 00       LDA   #$00
0C1D-   85 40       STA   $40

*C00G
```

...shows credits screen, then starts joystick calibration...

Hello, Mr. Do. It's a pleasure to finally meet you.

Chapter 10
In Which We Try To Come Up With A Pun About "Building Blocks" But Finally Lego

I think I finally know enough about the
boot process to capture all 7 blocks
and save them to my work disk.

First, let's reboot and save block 0.

```
]PR#5
...
]BRUN TRACE6
...reboots slot 6...
<beep>

*2800<800.DFF      ; block 0 is $06 pages
*C500G
...
]BSAVE OBJ0.0800-0DFF,A$2800,L$600
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
975C-   A9 24       LDA   #$24
975E-   8D AC 04    STA   $04AC
9761-   A9 FF       LDA   #$FF
9763-   8D A2 04    STA   $04A2
9766-   A9 58       LDA   #$58
9768-   8D A5 04    STA   $04A5

; force it to load block 1 instead of 0
976B-   A9 01       LDA   #$01
976D-   8D AF 04    STA   $04AF

; start the boot
9770-   4C 00 02    JMP   $0200
```

```
*BSAVE TRACE7,A$9600,L$1773
*9600G
...reboots slot 6...
<beep>
```

Good news: I have block 1 in memory.
Bad news: it's $B900 long, starting at
$0700 (text page) and filling all the
rest of main memory.

So we'll do this in stages.

```
*2800<800.1FFFM
*C500G
...
]BSAVE OBJ1.0800-1FFF,A$2800,L$1800
]BRUN TRACE7
...reboots slot 6...
<beep>

*C500G
...
]BSAVE OBJ1.2000-5FFF,A$2000,L$4000
]BRUN TRACE7
...reboots slot 6...
<beep>

*2000<6000.9FFFM
*C500G
...
]BSAVE OBJ1.6000-9FFF,A$2000,L$4000
]BRUN TRACE7
...reboots slot 6...
<beep>

*2000<A000.BFFFM
*C500G
...
]BSAVE OBJ1.A000-BFFF,A$2000,L$2000
```

To get the final page of block 1 (that loads at $0700), I'll need a separate trace program.

```
]CALL -151

*9600<C600.C6FFM

.
. [identical to previous trace]
.
975C-   A9 24       LDA   #$24
975E-   8D AC 04    STA   $04AC
9761-   A9 FF       LDA   #$FF
9763-   8D A2 04    STA   $04A2
9766-   A9 58       LDA   #$58
9768-   8D A5 04    STA   $04A5

; force it to load block 1 instead of 0
976B-   A9 01       LDA   #$01
976D-   8D AF 04    STA   $04AF

; ...but load it at $2700
9770-   A9 27       LDA   #$27
9772-   8D 67 04    STA   $0467

; ...and only 1 sector (instead of $B9)
9775-   A9 01       LDA   #$01
9777-   8D 60 04    STA   $0460

; start the boot
977A-   4C 00 02    JMP   $0200

*BSAVE TRACE8,A$9600,L$17D
*9600G
...reboots slot 6...
<beep>
```

```
*C500G
...
]BSAVE OBJ1.0700-07FF,A$2700,L$100
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
975C-   A9 24       LDA   #$24
975E-   8D AC 04    STA   $04AC
9761-   A9 FF       LDA   #$FF
9763-   8D A2 04    STA   $04A2
9766-   A9 58       LDA   #$58
9768-   8D A5 04    STA   $04A5
976B-   A9 02       LDA   #$02 ;block 2
976D-   8D AF 04    STA   $04AF
9770-   4C 00 02    JMP   $0200

*BSAVE TRACE9,A$9600,L$173
*9600G
...reboots slot 6...
<beep>

*2000<8000.A5FFM
*C500G
...
]BSAVE OBJ2.8000-A5FF,A$2000,L$2600
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
976B-   A9 03       LDA   #$03 ;block 3
976D-   8D AF 04    STA   $04AF
9770-   4C 00 02    JMP   $0200
```

```
*BSAVE TRACE10,A$9600,L$173
*9600G
...reboots slot 6...
<beep>

*2000<8000.B4FFM
*C500G
...
]BSAVE OBJ3.8000-B4FF,A$2000,L$3500
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
976B-    A9 03          LDA    #$04 ;block 4
976D-    8D AF 04       STA    $04AF
9770-    4C 00 02       JMP    $0200

*BSAVE TRACE11,A$9600,L$173
*9600G
...reboots slot 6...
<beep>

*2000<8000.B4FFM
*C500G
...
]BSAVE OBJ4.8000-B4FF,A$2000,L$3500
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
976B-    A9 05          LDA    #$05 ;block 5
976D-    8D AF 04       STA    $04AF
```

```
; load at $2700 (instead of $0700)
9770-   A9 27        LDA   #$27
9772-   8D 6B 04     STA   $046B
9775-   4C 00 02     JMP   $0200

*BSAVE TRACE12,A$9600,L$178
*9600G
...reboots slot 6...
<beep>

*C500G
...
]BSAVE OBJ5.0700-3FFF,A$2700,L$3900
]CALL -151

*9600<C600.C6FFM


.
. [identical to previous trace]
.
976B-   A9 06        LDA   #$06 ;block 6
976D-   8D AF 04     STA   $04AF

; load at $2000 (instead of $E000)
9770-   A9 20        LDA   #$20
9772-   8D 6C 04     STA   $046C
9775-   4C 00 02     JMP   $0200

*BSAVE TRACE13,A$9600,L$178
*9600G
...reboots slot 6...
<beep>

*C500G
...
]BSAVE OBJ6.E000-E4FF,A$2000,L$500
```

```
]CATALOG

C1983 DSR^C#254
012 FREE

 A 002 HELLO
*B 003 TRACE0
 B 003 BOOT0
*B 003 TRACE1
 B 003 BOOT0 0200-02FF
*B 003 TRACE2
 B 003 BOOT1 0700-07FF
*B 003 TRACE3
 B 003 BOOT1 0700-07FF DECRYPTED
*B 003 TRACE4
 B 003 BOOT1 0700-07FF DECRYPTED 2
*B 003 TRACE5
 B 006 BOOT1 0200-05FF
 B 003 TRACE6
 B 008 OBJ0.0800-0DFF
*B 003 TRACE7
 B 026 OBJ1.0800-1FFF
 B 066 OBJ1.2000-5FFF
 B 066 OBJ1.6000-9FFF
 B 034 OBJ1.A000-BFFF
*B 003 TRACE8
 B 003 OBJ1.0700-07FF
*B 003 TRACE9
 B 040 OBJ2.8000-A5FF
*B 003 TRACE10
 B 055 OBJ3.8000-B4FF
*B 003 TRACE11
 B 055 OBJ4.8000-B4FF
*B 003 TRACE12
 B 059 OBJ5.0700-3FFF
*B 003 TRACE13
 B 007 OBJ6.E000-E4FF
```

Whew.

Chapter 11
In which Two Wrongs Make A Write

Here's the plan:

1. write out each block to a standard 16-sector disk
2. find an RWTS that fits @ $0200-$05FF
3. create a bootloader that loads the RWTS at $0200, loads the preloader at $D000, and reproduces the other useful bits from the original disk
4. mimic the calling convention of the original RWTS at $0200 (X = block index from $00-$06) so I don't need to change any game code

Looking at the length of each block and dividing by 16, I can space everything out on separate tracks and still have plenty of room. This means each block can start on its own track, which saves a few bytes by being able to hard-code the starting sector for each block. (Space for the memory-resident RWTS is *extremely* tight.)

The disk map will look like this:

X	tr	sector	address range
0	01	0F..0A	$0800..$0DFF
1	02	0F..00	$0700..$16FF
1	03	0F..00	$1700..$26FF
1	04	0F..00	$2700..$36FF
1	05	0F..00	$3700..$46FF
1	06	0F..00	$4700..$56FF
1	07	0F..00	$5700..$66FF
1	08	0F..00	$6700..$76FF
1	09	0F..00	$7700..$86FF
1	0A	0F..00	$8700..$96FF
1	0B	0F..00	$9700..$A6FF
1	0C	0F..00	$A700..$B6FF
1	0D	0F..07	$B700..$BFFF
2	0E	0F..00	$8000..$8FFF
2	0F	0F..00	$9000..$9FFF
2	10	0F..0A	$A000..$A5FF
	11	dummy disk catalog	
3	12	0F..00	$8000..$8FFF
3	13	0F..00	$9000..$9FFF
3	14	0F..00	$A000..$AFFF
3	15	0F..0B	$B000..$B4FF
4	16	0F..00	$8000..$8FFF
4	17	0F..00	$9000..$9FFF
4	18	0F..00	$A000..$AFFF
4	19	0F..0B	$B000..$B4FF
5		reuses block 1	

[...]

6	1A	0F..0B	$E000..$E4FF
	1B		
	1C		
	1D		
	1E	unused	
	1F		
	20		
	21		
	22		

Here's a little utility program to help
write out each block.

```
; write a sector
0300-   A9 03         LDA   #$03
0302-   A0 28         LDY   #$28
0304-   20 D9 03      JSR   $03D9

; decrement sector, wrap around to $0F
0307-   AC 2D 03      LDY   $032D
030A-   88            DEY
030B-   10 05         BPL   $0312
030D-   A0 0F         LDY   #$0F

; increment track
030F-   EE 2C 03      INC   $032C
0312-   8C 2D 03      STY   $032D

; increment memory page
0315-   EE 31 03      INC   $0331

; decrement sector count
0318-   CE 21 03      DEC   $0321
```

```
; loop until done
031B-   D0 E3       BNE   $0300
031D-   60          RTS
```

$0321 is the sector count, and the RWTS parameter table starts at $0328.

```
*BSAVE WRITER,A$300,L$40
```

Write block 0 ($06 sectors) to track 1:

```
*321:06      ; sector count
*32C:01 0F   ; start track/sector
*331:08      ; start address (high)
*320.33F

0320- 00 06 00 00 00 00 00 00
         ^^
    sector count

0328- 01 60 01 00 01 0F 3B 03
                  ^^ ^^
         start track/sector

0330- 00 08 00 00 02 00 FE 60
      ^^^^^       ^^
  start address   write command

0338- 01 00 00 00 01 EF D8 00

*BLOAD OBJ0.0800-0DFF,A$800
*300G
```

Write block 1 (first $02 sectors):

```
*321:02
*32C:02 0F
*331:27
*320.33F

0320- 00 02 00 00 00 00 00 00
0328- 01 60 01 00 02 0F 3B 03
0330- 00 27 00 00 02 00 FE 60
0338- 01 00 00 00 01 EF D8 00

*BLOAD OBJ1.0700-07FF,A$2700
*BLOAD OBJ1.0800-1FFF,A$2800
*300G
```

Write block 1 (last $B7 sectors):

```
*321:B7
*32C:02 0D
*331:08
*320.33F

0320- 00 B7 00 00 00 00 00 00
0328- 01 60 01 00 02 0D 3B 03
0330- 00 08 00 00 02 00 FE 60
0338- 01 00 00 00 01 EF D8 00

*BLOAD OBJ1.0800-1FFF,A$700
*BLOAD OBJ1.2000-5FFF,A$1F00
*BLOAD OBJ1.6000-9FFF,A$5F00
*BLOAD OBJ1.A000-BFFF,A$9F00
*300G
```

Write block 2 ($26 sectors):

```
*321:26
*32C:0E 0F
*331:80
*BLOAD OBJ2.8000-A5FF,A$8000
*300G
```

Write block 3 ($35 sectors):

```
*321:35
*32C:12 0F
*331:80
*BLOAD OBJ3.8000-B4FF,A$8000
*300G
```

Write block 4 ($35 sectors):

```
*321:35
*32C:16 0F
*331:80
*BLOAD OBJ4.8000-B4FF,A$8000
*300G
```

Block 5 is actually just the first part of block 1, so I don't need to write it to disk. (This is true on the original disk, too -- look closely at the phase array at $0451.)

Write block 6 ($05 sectors):

```
*321:05
*32C:1A 0F
*331:20

*BLOAD OBJ6.E000-E4FF,A$2000
*300G
```

And that's all she wrote (to disk).(*)

(*) sorry(**)

(**) not sorry

Chapter 12
In Which We Pull Ourselves Up By Our Bootsector

Tracks $01-$1A now contain the original game code, but in a standard 16-sector format. Now for the fun part: designing track $00.

Of course, sector $00 will be loaded at $0800 by the disk controller ROM. From there, I'll need to load four sectors. I can't load them directly into their final locations, because I need to re-use the disk controller ROM and it uses $0200..$037F as temporary storage.

sc	initial	final
0E	$0900	$0200
0D	$0A00	$0300
0C	$0B00	$0400
0B	$0C00	$D000

```
; read four sectors into $0900..$0CFF
0801-   CE 19 08     DEC   $0819
0804-   30 0F        BMI   $0815
0806-   E6 3D        INC   $3D
0808-   8A           TXA
0809-   4A           LSR
080A-   4A           LSR
080B-   4A           LSR
080C-   4A           LSR
080D-   09 C0        ORA   #$C0
080F-   8D 14 08     STA   $0814
0812-   4C 5C 00     JMP   $005C

; execution continues here (from $0804)
; after all sectors are read
; store boot slot (x16) in RWTS
0815-   8E 1B 09     STX   $091B
```

```
; $0819 will be $FF by the time this is
; executed, so this just resets the
; stack
0818-    A2 04        LDX   #$04
081A-    9A           TXS

; clear hi-res screen
081B-    E8           INX
081C-    A0 20        LDY   #$20
081E-    8A           TXA
081F-    9D 00 40     STA   $4000,X
0822-    E8           INX
0823-    D0 FA        BNE   $081F
0825-    EE 21 08     INC   $0821
0828-    88           DEY
0829-    D0 F4        BNE   $081F

; ...and show it
082B-    2C 50 C0     BIT   $C050
082E-    2C 55 C0     BIT   $C055
0831-    2C 57 C0     BIT   $C057
0834-    2C 52 C0     BIT   $C052

; copy ROM to language card (original
; does this and relies on it, because
; the languard card remains active
; throughout the game)
0837-    2C 81 C0     BIT   $C081
083A-    2C 81 C0     BIT   $C081
083D-    A0 30        LDY   #$30
083F-    BD 00 D0     LDA   $D000,X
0842-    9D 00 D0     STA   $D000,X
0845-    E8           INX
0846-    D0 F7        BNE   $083F
0848-    EE 41 08     INC   $0841
084B-    EE 44 08     INC   $0844
084E-    88           DEY
084F-    D0 EE        BNE   $083F
```

```
; move RWTS into place
; S0E --> $0200
; S0D --> $0300
; S0C --> $0400
0851-   BD 00 09    LDA   $0900,X
0854-   9D 00 02    STA   $0200,X
0857-   BD 00 0A    LDA   $0A00,X
085A-   9D 00 03    STA   $0300,X
085D-   BD 00 0B    LDA   $0B00,X
0860-   9D 00 04    STA   $0400,X
0863-   E8          INX
0864-   D0 EB       BNE   $0851

; zero page $57 is used by the RWTS to
; hold the last read track
0866-   86 57       STX   $57

; zero page $5B is used by the RWTS as
; the low byte of the target address to
; store data read from disk
0868-   86 5B       STX   $5B

; this code is taken from the original
; game to determine whether it should
; load the autoplaying demo into $E000
; (the game will work on 48K machines
; but autoplaying demo requires 64K)
086A-   86 CC       STX   $CC
086C-   2C 83 C0    BIT   $C083
086F-   2C 83 C0    BIT   $C083
0872-   A9 82       LDA   #$82
0874-   8D 00 E0    STA   $E000
0877-   CD 00 E0    CMP   $E000
087A-   D0 07       BNE   $0883
```

```
; hey, we have at least 64K, so set the
; zero page indicator at $CC and load
; block 6 at $E000
087C-    85 CC          STA   $CC
087E-    A2 06          LDX   #$06
0880-    20 00 02       JSR   $0200

; set "return" address via stack since
; this page will be overwritten when we
; read block 0
0883-    A9 0B          LDA   #$0B
0885-    48             PHA
0886-    A9 FF          LDA   #$FF
0888-    48             PHA

; check machine ID byte to determine
; whether we should call the preloader
; at $D000 or just start the game
0889-    AD B3 FB       LDA   $FBB3
088C-    C9 06          CMP   #$06
088E-    F0 05          BEQ   $0895

; Apple II+ --> just start the game
0890-    A2 00          LDX   #$00
0892-    4C 00 02       JMP   $0200

; //e or later --> set up the preloader
0895-    2C 83 C0       BIT   $C083
0898-    2C 83 C0       BIT   $C083
089B-    A2 00          LDX   #$00
089D-    BD 00 0C       LDA   $0C00,X
08A0-    9D 00 D0       STA   $D000,X
08A3-    E8             INX
08A4-    D0 F7          BNE   $089D

; ...and continue from there
08A6-    4C 00 D0       JMP   $D000
```

Chapter 13
In Which Smaller Is Better, But Mini Is Best

For the RWTS, I chose Mini-RWTS by The Stack. It's only $297 bytes long, plus an additional $56 bytes for temporary storage during denibblizing. It decodes nibbles in place into the target memory page, so it doesn't require a 256-byte buffer like DOS 3.3. It has an easy API for moving the drive head to a track and reading multiple sectors into consecutive memory, which are the two things I need to do. It also comes with a maker program to relocate it to any page.

By default, Mini-RWTS uses zero page $F1-$FF, but I had to change that to $51-$5F because those are the only zero page addresses left unused during the game. I also had to strip out some functionality (like the ability to read from drive 2) to make space for the blockread API at $200, but it all fits. Barely.

```
; if 128K, the initialization routine
; at $D000 will change this to a JMP
; so preloaded resources are copied
; from auxiliary memory
0200-   2C 8E D0     BIT   $D08E

; switch to page 2 (it turns out it's
; always safe to do this, not only on
; block 5)
0203-   2C 55 C0     BIT   $C055

; My trials and tribulations getting
; this to fit in $300 bytes left me
; with one byte to spare. Here it is:
0206-   EA           NOP
```

I thought about using the illegal two-
byte NOP $74, in honor of the original
disk's boot sector, but I literally
couldn't spare the extra byte.

Now to set up The Stack's Mini-RWTS.

```
; $5C is high byte of starting address
0207-   BD EB 03     LDA   $03EB,X
020A-   85 5C        STA   $5C

; $5D is sector count
020C-   BD F2 03     LDA   $03F2,X
020F-   85 5D        STA   $5D

; $5E is starting track
0211-   BD F9 03     LDA   $03F9,X
0214-   85 5E        STA   $5E

; $5F is starting sector (always $0F)
0216-   A9 0F        LDA   #$0F
0218-   85 5F        STA   $5F

; X is the slot number (x16)
; this was actually set at $0815 to the
; boot slot x16, so my crack will boot
; from any slot
021A-   A2 60        LDX   #$60

; Mini-RWTS starts here
021C-   A5 5D        LDA   $5D
021E-   D0 01        BNE   $0221
0220-   60           RTS
```

```
; drive initialization (like DOS 3.3)
0221-   BD 8E C0    LDA   $C08E,X
0224-   BD 8E C0    LDA   $C08E,X
0227-   A0 08       LDY   #$08
0229-   BD 8C C0    LDA   $C08C,X
022C-   48          PHA
022D-   68          PLA
022E-   48          PHA
022F-   68          PLA
0230-   DD 8C C0    CMP   $C08C,X
0233-   D0 03       BNE   $0238
0235-   88          DEY
0236-   D0 F1       BNE   $0229
0238-   08          PHP
0239-   BD 89 C0    LDA   $C089,X
023C-   A9 D8       LDA   #$D8
023E-   85 5A       STA   $5A
0240-   A9 EF       LDA   #$EF
0242-   85 59       STA   $59
0244-   28          PLP
0245-   08          PHP
0246-   D0 08       BNE   $0250
0248-   A0 08       LDY   #$08
024A-   20 00 04    JSR   $0400
024D-   88          DEY
024E-   D0 FA       BNE   $024A
```

```
; move drive head to start track
0250-   A5 5E       LDA   $5E
0252-   20 8A 03    JSR   $038A
0255-   28          PLP
0256-   D0 11       BNE   $0269
0258-   A4 5A       LDY   $5A
025A-   10 0D       BPL   $0269
025C-   A0 12       LDY   #$12
025E-   88          DEY
025F-   D0 FD       BNE   $025E
0261-   E6 59       INC   $59
0263-   D0 F7       BNE   $025C
0265-   E6 5A       INC   $5A
0267-   D0 F3       BNE   $025C
0269-   A0 30       LDY   #$30
026B-   84 52       STY   $52
026D-   A0 FC       LDY   #$FC
026F-   84 55       STY   $55
0271-   C8          INY
0272-   D0 04       BNE   $0278
0274-   E6 55       INC   $55
0276-   F0 4F       BEQ   $02C7

; read address prologue
0278-   BD 8C C0    LDA   $C08C,X
027B-   10 FB       BPL   $0278
027D-   C9 D5       CMP   #$D5
027F-   D0 F0       BNE   $0271
0281-   EA          NOP
0282-   BD 8C C0    LDA   $C08C,X
0285-   10 FB       BPL   $0282
0287-   C9 AA       CMP   #$AA
0289-   D0 F2       BNE   $027D
028B-   A0 03       LDY   #$03
028D-   BD 8C C0    LDA   $C08C,X
0290-   10 FB       BPL   $028D
0292-   C9 96       CMP   #$96
0294-   D0 E7       BNE   $027D
```

```
; read and store address field
0296-   A9 00        LDA   #$00
0298-   85 58        STA   $58
029A-   BD 8C C0     LDA   $C08C,X
029D-   10 FB        BPL   $029A
029F-   2A           ROL
02A0-   85 53        STA   $53
02A2-   BD 8C C0     LDA   $C08C,X
02A5-   10 FB        BPL   $02A2
02A7-   25 53        AND   $53
02A9-   99 53 00     STA   $0053,Y
02AC-   45 58        EOR   $58
02AE-   88           DEY
02AF-   10 E7        BPL   $0298
02B1-   A8           TAY
02B2-   D0 13        BNE   $02C7

; read address epilogue
02B4-   BD 8C C0     LDA   $C08C,X
02B7-   10 FB        BPL   $02B4
02B9-   C9 DE        CMP   #$DE
02BB-   D0 0A        BNE   $02C7
02BD-   EA           NOP
02BE-   BD 8C C0     LDA   $C08C,X
02C1-   10 FB        BPL   $02BE
02C3-   C9 AA        CMP   #$AA
02C5-   F0 17        BEQ   $02DE
02C7-   C6 52        DEC   $52
02C9-   10 A2        BPL   $026D
```

```
; can't read sector -- move to track 0
; then back, and try again
02CB-   A5 57         LDA   $57
02CD-   48            PHA
02CE-   A9 60         LDA   #$60
02D0-   85 57         STA   $57
02D2-   A9 00         LDA   #$00
02D4-   20 8A 03      JSR   $038A
02D7-   68            PLA
02D8-   20 8A 03      JSR   $038A
02DB-   18            CLC
02DC-   90 8B         BCC   $0269

; execution continues here (from $02C5)
02DE-   A4 55         LDY   $55
02E0-   C4 57         CPY   $57
02E2-   F0 07         BEQ   $02EB
02E4-   A5 57         LDA   $57
02E6-   84 57         STY   $57
02E8-   18            CLC
02E9-   90 ED         BCC   $02D8
02EB-   A4 5F         LDY   $5F

; is this the sector we wanted?
02ED-   B9 F0 04      LDA   $04F0,Y
02F0-   C5 54         CMP   $54

; no, try again
02F2-   D0 D3         BNE   $02C7
02F4-   A0 20         LDY   #$00
02F6-   88            DEY
02F7-   F0 CE         BEQ   $02C7
```

```
; we're on the right sector, now find
; the data prologue
02F9-    BD 8C C0     LDA   $C08C,X
02FC-    10 FB        BPL   $02F9
02FE-    49 D5        EOR   #$D5
0300-    D0 F4        BNE   $02F6
0302-    EA           NOP
0303-    BD 8C C0     LDA   $C08C,X
0306-    10 FB        BPL   $0303
0308-    C9 AA        CMP   #$AA
030A-    D0 F2        BNE   $02FE
030C-    A0 56        LDY   #$56
030E-    BD 8C C0     LDA   $C08C,X
0311-    10 FB        BPL   $030E
0313-    C9 AD        CMP   #$AD
0315-    D0 E7        BNE   $02FE

; decode nibbles and store directly in
; target page (uses a small buffer at
; $0497 for temporary storage)
0317-    A9 00        LDA   #$00
0319-    88           DEY
031A-    84 56        STY   $56
031C-    BC 8C C0     LDY   $C08C,X
031F-    10 FB        BPL   $031C
0321-    59 97 03     EOR   $0397,Y
0324-    A4 56        LDY   $56
0326-    99 97 04     STA   $0497,Y
0329-    D0 EE        BNE   $0319
032B-    84 56        STY   $56
032D-    BC 8C C0     LDY   $C08C,X
0330-    10 FB        BPL   $032D
0332-    59 97 03     EOR   $0397,Y
0335-    A4 56        LDY   $56
0337-    91 5B        STA   ($5B),Y
0339-    C8           INY
033A-    D0 EF        BNE   $032B
033C-    BC 8C C0     LDY   $C08C,X
033F-    10 FB        BPL   $033C
0341-    D9 97 03     CMP   $0397,Y
0344-    D0 81        BNE   $02C7
```

```
; read data epilogue
0346-    BD 8C C0     LDA   $C08C,X
0349-    10 FB        BPL   $0346
034B-    C9 DE        CMP   #$DE
034D-    D0 F5        BNE   $0344
034F-    EA           NOP
0350-    BD 8C C0     LDA   $C08C,X
0353-    10 FB        BPL   $0350
0355-    C9 AA        CMP   #$AA
0357-    D0 EB        BNE   $0344

; finish denibblizing
0359-    86 58        STX   $58
035B-    A0 00        LDY   #$00
035D-    A2 56        LDX   #$56
035F-    CA           DEX
0360-    30 FB        BMI   $035D
0362-    B1 5B        LDA   ($5B),Y
0364-    5E 97 04     LSR   $0497,X
0367-    2A           ROL
0368-    5E 97 04     LSR   $0497,X
036B-    2A           ROL
036C-    91 5B        STA   ($5B),Y
036E-    C8           INY
036F-    D0 EE        BNE   $035F

; turn off drive motor (we'll turn it
; back on if there are more sectors)
0371-    A6 58        LDX   $58
0373-    BD 88 C0     LDA   $C088,X

; increment target page
0376-    E6 5C        INC   $5C

; decrement sector, wrap around to $0F
0378-    A4 5F        LDY   $5F
037A-    88           DEY
037B-    10 04        BPL   $0381
037D-    A0 0F        LDY   #$0F
```

```
; increment track
037F-   E6 5E       INC   $5E
0381-   84 5F       STY   $5F

; decrement sector count
0383-   C6 5D       DEC   $5D

; are we done yet?
0385-   F0 0B       BEQ   $0392

; no, loop back to read another sector
0387-   4C 38 02    JMP   $0238

; entry point to move drive head to
; desired track (called from $0252)
; accumulator holds track (not phase)
038A-   0A          ASL
038B-   06 57       ASL   $57
038D-   20 93 03    JSR   $0393
0390-   46 57       LSR   $57
0392-   60          RTS
```

```
; this is essentially identical to the
; SEEKABS routine at $B9A0 in DOS 3.3
0393-   86 58       STX   $58
0395-   85 55       STA   $55
0397-   C5 57       CMP   $57
0399-   F0 4F       BEQ   $03EA
039B-   A9 00       LDA   #$00
039D-   85 53       STA   $53
039F-   A5 57       LDA   $57
03A1-   85 54       STA   $54
03A3-   38          SEC
03A4-   E5 55       SBC   $55
03A6-   F0 31       BEQ   $03D9
03A8-   B0 06       BCS   $03B0
03AA-   49 FF       EOR   #$FF
03AC-   E6 57       INC   $57
03AE-   90 04       BCC   $03B4
03B0-   69 FE       ADC   #$FE
03B2-   C6 57       DEC   $57
03B4-   C5 53       CMP   $53
03B6-   90 02       BCC   $03BA
03B8-   A5 53       LDA   $53
03BA-   C9 0C       CMP   #$0C
03BC-   B0 01       BCS   $03BF
03BE-   A8          TAY
03BF-   38          SEC
03C0-   20 DD 03    JSR   $03DD
03C3-   B9 15 04    LDA   $0415,Y
03C6-   20 00 04    JSR   $0400
03C9-   A5 54       LDA   $54
03CB-   18          CLC
03CC-   20 DF 03    JSR   $03DF
03CF-   B9 21 04    LDA   $0421,Y
03D2-   20 00 04    JSR   $0400
03D5-   E6 53       INC   $53
03D7-   D0 C6       BNE   $039F
03D9-   20 00 04    JSR   $0400
03DC-   18          CLC
                             [...]
```

```
03DD-   A5 57          LDA   $57
03DF-   29 03          AND   #$03
03E1-   2A             ROL
03E2-   05 58          ORA   $58
03E4-   AA             TAX
03E5-   BD 80 C0       LDA   $C080,X
03E8-   A6 58          LDX   $58
03EA-   60             RTS
```

Here are the game-specific arrays for the block reads (referenced from $0207, $020C, and $0211):

```
03EB- 08 07 80 80 80 07 E0    ; address
03F2- 06 B9 26 35 35 39 05    ; length
03F9- 01 02 0E 12 16 02 1A    ; track #
```

```
; wait loop (at $BA00 in DOS 3.3)
; (needs to be page-aligned because
; otherwise the branch takes too long
; and throws off the cycle count)
0400-   86 56          STX   $56
0402-   A2 11          LDX   #$11
0404-   CA             DEX
0405-   D0 FD          BNE   $0404
0407-   E6 59          INC   $59
0409-   D0 02          BNE   $040D
040B-   E6 5A          INC   $5A
040D-   38             SEC
040E-   E9 01          SBC   #$01
0410-   D0 F0          BNE   $0402
0412-   A6 56          LDX   $56
0414-   60             RTS
```

```
; arm move delay table
; (at $BA11 in DOS 3.3)
0415-                    01 30 28
0418- 24 20 1E 1D 1C 1C 1C 1C
0420- 1C 70 2C 26 22 1F 1E 1D
0428- 1C 1C 1C 1C 1C

; nibble read translate table
; (at $BA96 in DOS 3.3)
042D-                    00 01 98
0430- 99 02 03 9C 04 05 06 A0
0438- A1 A2 A3 A4 A5 07 08 A8
0440- A9 AA 09 0A 0B 0C 0D B0
0448- B1 0E 0F 10 11 12 13 B8
0450- 14 15 16 17 18 19 1A C0
0458- C1 C2 C3 C4 C5 C6 C7 C8
0460- C9 CA 1B CC 1C 1D 1E D0
0468- D1 D2 1F D4 D5 20 21 D8
0470- 22 23 24 25 26 27 28 E0
0478- E1 E2 E3 E4 29 2A 2B E8
0480- 2C 2D 2E 2F 30 31 32 F0
0488- F1 33 34 35 36 37 38 F8
0490- 39 3A 3B 3C 3D 3E 3F

0497- [temporary denibblizing storage]

; physical to logical sector map
; (referenced from $02ED)
04F0- 00 0D 0B 09 07 05 03 01
04F8- 0E 0C 0A 08 06 04 02 0F
```

Chapter 14
In Which We Note The Small Irony
Of Saving The Preloader For Last

Finally, the preloader and cache-read routines at $D000. (This is T00,S0B.)

```
; load block 0 (X=0 here)
D000-   20 00 02    JSR   $0200

; show credits on text page 2 (loaded
; as part of block 0)
D003-   2C 55 C0    BIT   $C055
D006-   2C 51 C0    BIT   $C051

; test for 128K
D009-   8D 05 C0    STA   $C005
D00C-   A9 09       LDA   #$09
D00E-   8D 00 80    STA   $8000
D011-   A9 23       LDA   #$23
D013-   8D 01 80    STA   $8001
D016-   0E 00 84    ASL   $8400
D019-   0E 01 84    ASL   $8401
D01C-   8D 04 C0    STA   $C004
D01F-   8D 03 C0    STA   $C003
D022-   AD 00 80    LDA   $8000
D025-   C9 09       CMP   #$09
D027-   D0 0A       BNE   $D033
D029-   AD 01 80    LDA   $8001
D02C-   C9 23       CMP   #$23
D02E-   8D 02 C0    STA   $C002
D031-   F0 04       BEQ   $D037

; no 128K, nothing left to do
D033-   8D 02 C0    STA   $C002
D036-   60          RTS
```

```
; preload blocks 2, 3, and 4 into aux
; memory (adapted from original game
; preloader)
D037-   A0 02       LDY   #$02
D039-   8C FF D0    STY   $D0FF
D03C-   BE CC D0    LDX   $D0CC,Y
D03F-   A9 70       LDA   #$70
D041-   85 5C       STA   $5C
D043-   8D 6E D0    STA   $D06E
D046-   BD F2 03    LDA   $03F2,X
D049-   85 5D       STA   $5D
D04B-   BD F9 03    LDA   $03F9,X
D04E-   85 5E       STA   $5E
D050-   20 16 02    JSR   $0216
D053-   AC FF D0    LDY   $D0FF
D056-   BE CC D0    LDX   $D0CC,Y
D059-   A9 00       LDA   #$00
D05B-   9D F9 03    STA   $03F9,X
D05E-   BD C6 D0    LDA   $D0C6,X
D061-   8D 71 D0    STA   $D071
D064-   BC F2 03    LDY   $03F2,X
D067-   A2 00       LDX   #$00
D069-   8D 05 C0    STA   $C005
D06C-   BD 00 FF    LDA   $FF00,X
D06F-   9D 00 FF    STA   $FF00,X
D072-   CA          DEX
D073-   D0 F7       BNE   $D06C
D075-   EE 6E D0    INC   $D06E
D078-   EE 71 D0    INC   $D071
D07B-   88          DEY
D07C-   D0 EE       BNE   $D06C
D07E-   AC FF D0    LDY   $D0FF
D081-   8D 04 C0    STA   $C004
D084-   88          DEY
D085-   10 B2       BPL   $D039
```

```
; set RWTS entry point to jump to the
; following routine (at $D08E) to check
; the cache in aux memory
D087-    A9 4C       LDA   #$4C
D089-    8D 00 02    STA   $0200
D08C-    60          RTS

; entry point to load preloaded blocks
; from cache instead of disk
D08E-    AD 83 C0    LDA   $C083
D091-    AD 83 C0    LDA   $C083

; is this block cached in aux memory?
D094-    BD F9 03    LDA   $03F9,X
D097-    F0 03       BEQ   $D09C

; no, jump back to RWTS to read block
; from disk
D099-    4C 03 02    JMP   $0203

; yes, copy it from aux memory
D09C-    BD C6 D0    LDA   $D0C6,X
D09F-    8D B2 D0    STA   $D0B2
D0A2-    BD EB 03    LDA   $03EB,X
D0A5-    8D B5 D0    STA   $D0B5
D0A8-    BC F2 03    LDY   $03F2,X
D0AB-    A2 00       LDX   #$00
D0AD-    8D 03 C0    STA   $C003
D0B0-    BD 00 FF    LDA   $FF00,X
D0B3-    9D 00 FF    STA   $FF00,X
D0B6-    CA          DEX
D0B7-    D0 F7       BNE   $D0B0
D0B9-    EE B2 D0    INC   $D0B2
D0BC-    EE B5 D0    INC   $D0B5
D0BF-    88          DEY
D0C0-    D0 EE       BNE   $D0B0
D0C2-    8D 02 C0    STA   $C002
D0C5-    60          RTS
```

Arrays for which blocks to cache and
where to put them in aux memory:

```
D0C8- 02 28 5D    ; address (high byte)
D0CC- 04 03 02    ; block index
```

Quod erat liberandum.

Epilogue: Cheats and Hacks

```
I have not enabled any of these hacks,
but I have verified that they work.

Infinite lives:
T03,S09,$A7 change "C6" to "A5"

Make <Ctrl-S> jump to next level
  instead of toggling sound:
T0B,S00,$39 change "A5 3E 49 01 85 3E"
                to "A2 FF 9A 4C DC 09"
T18,S09,$39 change "A5 3E 49 01 85 3E"
                to "A2 FF 9A 4C DC 09"

Make <Ctrl-S> give you an extra life
  instead of toggling sound:
T0B,S00,$39 change "A5 3E 49 01 85 3E"
                to "A2 FF 9A 4C AA 09"
T18,S09,$39 change "A5 3E 49 01 85 3E"
                to "A2 FF 9A 4C AA 09"

Make <Esc> pause the game instead of
  <space>:
T0B,S00,$30 change "A0" to "9B"
T18,S09,$30 change "A0" to "9B"

Turn off 128K preloader (game will boot
  faster but access disk more later):
T00,S00,$8E change "F0" to "24"
```

Changelog

2015-09-04

- better explanation of why a long
  sequence of zero bits on the disk
  appears to return random noise

2015-09-03

- typos (thanks qkumba)

2015-07-05

- re-rip block 3 (was corrupted on work
  disk due to typo during original rip)
- fix auxmem block array in preloader
  (caused corruption on "extra life"
  screen because not all blocks are the
  same size, so some cached blocks were
  being overwritten by others)
- added note on extra life cheat since
  I had to find it anyway to test the
  fix for the cache corruption

Thanks to usotsuki for finding these
bugs and testing the fixes.

2015-06-29

- initial release