Championship
Lode Runner

2015-03-26

4am
to deprotect
and preserve

Contents

0 In Which Various Automated Tools Fail In Interesting Ways 4

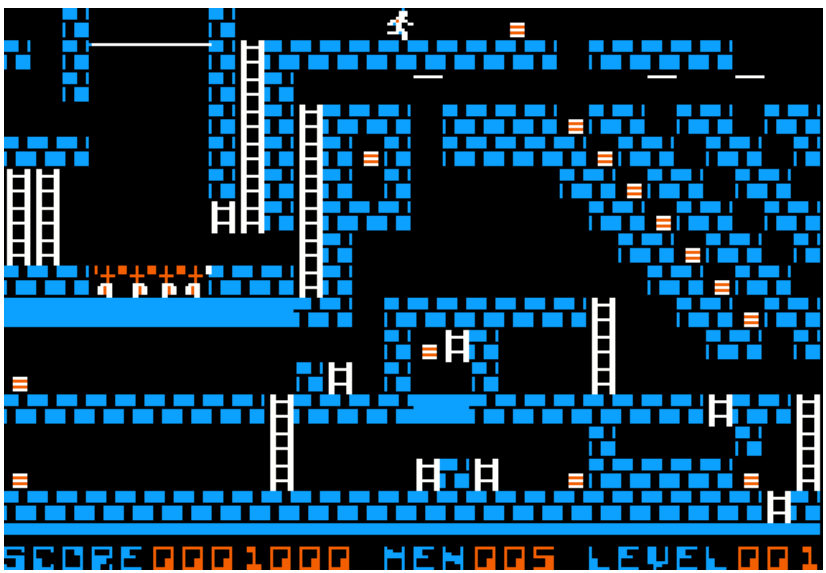
1 In Which It Is Not At All Clear What's Going On 8

2 In Which We Kinda Sorta Comprehend What's Going On, But Not Really 17

3 In Which We Fake It 'Til We Make It 28

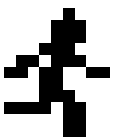
4 If You Wish To Play A Game, You Must First Create The Universe 33

A PostScript: Cheat codes 40



--------Championship Lode Runner--------
A 4am crack 2015-03-26

Name: Championship Lode Runner
Genre: arcade
Year: 1984
Authors: Doug Smith
Publisher: Broderbund Software
Media: single-sided 5.25-inch floppy
OS: custom
Other versions: The Burglar/MPG



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

Copy][+ automatic bit copy
  --> "LODE RUNNER (CHAMPIONSHIP)"
  parm entry says to copy T00, then
    T03-T0C with sector copy without
    address epilogue checking, then
    T0D.25-T1B.25 (a.k.a. "quarter
    tracks")
  It also says that the quarter tracks
    are extremely difficult to copy,
    which matches my experience. (I
    never succeeded in doing so.)

EDD 4 bit copy (no sync, no count)
  read errors on T01-02, T1C-T22
  copy just hangs on boot

EDD 4 bit copy (redo with quarter
  tracks T0D.25-T1B.25)
  no success; copy still hangs on boot

Copy][+ nibble editor
  T03-T0C appear to be mostly normal
    with modified address epilogue (not
    consistent, but starts with "DE")
  I can see nibble data on the quarter
    tracks. Appears to be 4-4 encoded.

--v--

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

TRACK: 0D.25  START: 3706  LENGTH: 015F
       ^^^^^

39B8: AA AA AA AA AA AA AA AA   VIEW
39C0: AA AA AA AA AA AA AA AA
39C8: AA AA AA AA AA AA AA AA
39D0: AA AA AA AA F5 AA AA BF
39D8: AE EA AA FA EA EA EA EA  <-39DE
39E0: EA FA EA FA EA FA FA FA
39E8: FA FE FA FE FA FE FA FE
39F0: FE FE FE FF FE FF FE AA
39F8: AA AA AF FF FF AA AA AE

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

  A  TO ANALYZE DATA  ESC TO QUIT

  ?  FOR HELP SCREEN  /  CHANGE PARMS

  Q  FOR NEXT TRACK   SPACE TO RE-READ
```

--^--

Disk Fixer
  ["0" -> "Input/Output Control"]
    set CHECKSUM ENABLED=NO
  T03-T0C readable, appear to be level
    data (first 50 sectors have titles
    like "HELLO...WELCOME", "MUSIC
    MAESTRO", "LADDERS GALORE", &c.)
  no ability to read quarter tracks, so
    the rest of the disk is a mystery

Why didn't COPYA work?
  so many reasons

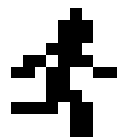
Why didn't Locksmith FDB work?
  LOL

Why didn't my EDD copy work?
  My first attempt didn't copy the
  quarter tracks. I've heard that it's
  theoretically possible to copy them,
  but it's very difficult even when you
  know exactly which quarter tracks to
  copy. All my attempts to do so were
  unsuccessful, for unknown reasons.

This is going to be one of those
"capture the game in memory and rebuild
it from the ground up" cracks.

Next steps:

  1. Trace bootloader
  2. Capture game code in memory
  3. Write game to a standard disk and
     build my own bootloader to load it



Chapter 1
In Which It Is Not At All Clear What's Going On

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

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

]CALL -151
*800<2800.28FFM
*801L

; clear hi-res graphics screens (both)
0801-   A0 00        LDY   #$00
0803-   A9 20        LDA   #$20
0805-   A2 40        LDX   #$40
0807-   84 00        STY   $00
0809-   85 01        STA   $01
080B-   98           TYA
080C-   91 00        STA   ($00),Y
080E-   C8           INY
080F-   D0 FB        BNE   $080C
0811-   E6 01        INC   $01
0813-   CA           DEX
0814-   D0 F6        BNE   $080C

; show hi-res graphics screen 1
0816-   2C 52 C0     BIT   $C052
0819-   2C 57 C0     BIT   $C057
081C-   2C 54 C0     BIT   $C054
081F-   2C 50 C0     BIT   $C050

; save slot number (x16)
0822-   A6 2B        LDX   $2B
0824-   86 08        STX   $08
```

```
; decrypt rest of boot0 and store it in
; zero page (starting at $60)
0826-   EA           NOP
0827-   EA           NOP
0828-   A0 00        LDY   #$00
082A-   EA           NOP
082B-   EA           NOP
082C-   B9 50 08     LDA   $0850,Y
082F-   EA           NOP
0830-   EA           NOP
0831-   49 A5        EOR   #$A5
0833-   EA           NOP
0834-   EA           NOP
0835-   99 60 00     STA   $0060,Y
0838-   EA           NOP
0839-   EA           NOP
083A-   C8           INY
083B-   D0 EF        BNE   $082C
083D-   EA           NOP
083E-   EA           NOP

; reset stack pointer
083F-   A2 FF        LDX   #$FF
0841-   EA           NOP
0842-   EA           NOP
0843-   EA           NOP
0844-   9A           TXS
0845-   EA           NOP
0846-   EA           NOP

; and exit
0847-   60           RTS
```

Wait, what?

Here's what: we decrypted $B0 bytes and stored them in zero page starting at $60. But that means $10 bytes were also stored in $0100..$010F. Then we reset the stack pointer, then we "returned." The stack pointer wrapped around to $00, and whatever ended up at $0100 serves as a "return" address (minus 1, as usual).

Let's find out what that is.

```
*9600<C600.C6FFM

; set up callback after decryption loop
96F8-   A9 4C       LDA   #$4C
96FA-   8D 45 08    STA   $0845
96FD-   A9 0A       LDA   #$0A
96FF-   8D 46 08    STA   $0846
9702-   A9 97       LDA   #$97
9704-   8D 47 08    STA   $0847

; start the boot
9707-   4C 01 08    JMP   $0801

; callback is here -- copy decrypted
; code/data to graphics page so it
; survives a reboot
970A-   A0 00       LDY   #$00
970C-   B9 60 00    LDA   $0060,Y
970F-   99 60 20    STA   $2060,Y
9712-   C8          INY
9713-   D0 F7       BNE   $970C

; turn off the slot 6 drive motor
9715-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9718-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE1,A$9600,L$11B
*9600G
...reboots slot 6...
...reboots slot 5...
]BSAVE BOOT0 0060-015F,A$2060,L$100
]CALL -151

*2100.210F

2100- B3 00 5F 00 FF 03 00 04
2108- FF 8B FE 07 FF 03 FF 5F
```

This is what ends up at $0100. The first two bytes are $B3/$00, so execution continues at $00B4. At the next RTS, it will jump to $0060, then $0400, then $0401, &c.

So what's at $00B4?

```
*20B4L

; no idea what this is doing, but I'm
; sure it'll become clear soon enough
20B4-   A2 D4        LDX   #$D4
20B6-   86 00        STX   $00
20B8-   E8           INX
20B9-   86 01        STX   $01
20BB-   E8           INX
20BC-   86 02        STX   $02
20BE-   E8           INX
20BF-   86 03        STX   $03
20C1-   A9 04        LDA   #$04
20C3-   AA           TAX
20C4-   60           RTS
```

Another RTS. Now we jump to $0060.

```
*2060L

2060-   86 3E       STX   $3E
2062-   85 3A       STA   $3A
2064-   A6 3E       LDX   $3E
2066-   86 40       STX   $40
2068-   A0 00       LDY   #$00
206A-   A5 3A       LDA   $3A
206C-   84 3C       STY   $3C
206E-   85 3D       STA   $3D
```

X and A came in with $04, and Y ended up at $00 after the decryption loop at $082C, so the zero page end up as

```
  $00 = $D4
  $01 = $D5
  $02 = $D6
  $03 = $D7
  ...
  $08 = slot number x16 (e.g. $60)
  ...
  $3A = $04
  ...
  $3C = $00
  $3D = $04
  $3E = $04
  ...
  $40 = $04
```

```
; slot number x16
2070-   A6 08       LDX   $08

; subroutine just reads a nibble
2072-   20 AE 00    JSR   $00AE
```

```
; Ah, that zero page initialization at
; $00B4 makes sense now. Those values
; constitute a custom prologue to read
; the rest of track $00: "D4 D5 D6"
2075-   C5 00       CMP   $00
2077-   D0 F9       BNE   $2072
2079-   20 AE 00    JSR   $00AE
207C-   C5 01       CMP   $01
207E-   D0 F5       BNE   $2075
2080-   20 AE 00    JSR   $00AE
2083-   C5 02       CMP   $02
2085-   D0 F5       BNE   $207C

; decode 4-4 encoded sector data
2087-   BD 8C C0    LDA   $C08C,X
208A-   10 FB       BPL   $2087
208C-   2A          ROL
208D-   85 3F       STA   $3F
208F-   BD 8C C0    LDA   $C08C,X
2092-   10 FB       BPL   $208F
2094-   25 3F       AND   $3F

; store in $0400 (text page)
2096-   91 3C       STA   ($3C),Y
2098-   C8          INY
2099-   D0 EC       BNE   $2087
209B-   0E 00 C0    ASL   $C000

; and a one-nibble prologue
209E-   BD 8C C0    LDA   $C08C,X
20A1-   10 FB       BPL   $209E
20A3-   C5 03       CMP   $03
20A5-   D0 BD       BNE   $2064

; increment page
20A7-   E6 3D       INC   $3D
```

```
; decrement sector count
20A9-   C6 40        DEC   $40
20AB-   D0 DA        BNE   $2087
20AD-   60           RTS
```

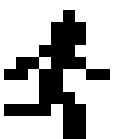
So we're reading 4 sectors into $0400, then "returning" again. According to the stack, execution continues at $0400, which I don't have yet.

```
*9600<C600.C6FFM

; set up callback #1
96F8-   A9 4C        LDA   #$4C
96FA-   8D 45 08     STA   $0845
96FD-   A9 0A        LDA   #$0A
96FF-   8D 46 08     STA   $0846
9702-   A9 97        LDA   #$97
9704-   8D 47 08     STA   $0847

; start the boot
9707-   4C 01 08     JMP   $0801

; (callback #1) set up callback #2
; after reading into text page by
; directly modifying the stack page
970A-   A9 14        LDA   #$14
970C-   8D 04 01     STA   $0104
970F-   A9 97        LDA   #$97
9711-   8D 05 01     STA   $0105

; "RTS" to continue the boot
9714-   60           RTS
```

```
; (callback #2) copy text page to
; graphics page so it survives a
; reboot
9715-   A2 04       LDX   #$04
9717-   A0 00       LDY   #$00
9719-   B9 00 04    LDA   $0400,Y
971C-   99 00 24    STA   $2400,Y
971F-   C8          INY
9720-   D0 F7       BNE   $9719
9722-   EE 1B 97    INC   $971B
9725-   EE 1E 97    INC   $971E
9728-   CA          DEX
9729-   D0 EE       BNE   $9719

; turn off slot 6 drive motor
972B-   AD E8 C0    LDA   $C0E8

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

*BSAVE TRACE2,A$9600,L$131
*9600G
...reboots slot 6...
...reboots slot 5...
]BSAVE BOOT1 0400-07FF,A$2400,L$400
```

Nestled in the code-on-the-text-page (and made visible by the boot tracing) is a message from the distant past:

HI FROM COMOX !!! R.G. AND M.G.



Chapter 2
In Which We Kinda Sorta Comprehend What's Going On, But Not Really

The last instruction we executed was an RTS (at $00AD). Once again, we look to the stack to see where execution continues, and it continues at $0400. I have $0400..$07FF in memory at $2400, so we go.

```
]CALL -151
*2400L

2400-   60          RTS

I swear I am not making this up.

Looking to the stack once more, we see
that execution continues at $0401.

2401-   EA          NOP
2402-   EA          NOP
2403-   20 E0 07    JSR   $07E0
2406-   EA          NOP
2407-   EA          NOP

*27E0L

; zap RAM bank 1 in the language card
27E0-   AD 81 C0    LDA   $C081
27E3-   AD 81 C0    LDA   $C081
27E6-   A0 00       LDY   #$00
27E8-   A9 D0       LDA   #$D0
27EA-   84 00       STY   $00
27EC-   85 01       STA   $01
27EE-   B1 00       LDA   ($00),Y
27F0-   91 00       STA   ($00),Y
27F2-   C8          INY
27F3-   D0 F9       BNE   $27EE
27F5-   E6 01       INC   $01
27F7-   D0 F5       BNE   $27EE
27F9-   AD 80 C0    LDA   $C080
27FC-   60          RTS
```

```
*2408L

; read/write RAM bank 2
2408-   AD 83 C0    LDA   $C083
240B-   AD 83 C0    LDA   $C083

; move some code to $0200
240E-   A0 00       LDY   #$00
2410-   B9 00 07    LDA   $0700,Y
2413-   99 00 02    STA   $0200,Y
2416-   C8          INY
2417-   D0 F7       BNE   $2410

*2700L

; standard Broderbund Badlands -- put a
; debugging character in the upper-left
; corner of the screen, play a sound,
; wipe memory, and reboot
2700-   A9 D2       LDA   #$D2
2702-   2C A9 D0    BIT   $D0A9
2705-   2C A9 CC    BIT   $CCA9
2708-   2C A9 A1    BIT   $A1A9
270B-   48          PHA
270C-   20 E0 02    JSR   $02E0
270F-   20 2F FB    JSR   $FB2F
2712-   20 58 FC    JSR   $FC58
2715-   20 84 FE    JSR   $FE84
2718-   68          PLA
2719-   8D 00 04    STA   $0400
271C-   A0 00       LDY   #$00
271E-   98          TYA
271F-   99 00 BF    STA   $BF00,Y
2722-   C8          INY
2723-   D0 FA       BNE   $271F
2725-   CE 21 02    DEC   $0221
2728-   AD 21 02    LDA   $0221
272B-   AA          TAX
272C-   2C 30 C0    BIT   $C030
[...]
```

```
272F-   EA          NOP
2730-   EA          NOP
2731-   EA          NOP
2732-   C9 08       CMP   #$08
2734-   B0 E6       BCS   $271C
2736-   8D F3 03    STA   $03F3
2739-   8D F4 03    STA   $03F4
273C-   AD FF 02    LDA   $02FF
273F-   4A          LSR
2740-   4A          LSR
2741-   4A          LSR
2742-   4A          LSR
2743-   09 C0       ORA   #$C0
2745-   E9 00       SBC   #$00
2747-   48          PHA
2748-   A9 FF       LDA   #$FF
274A-   48          PHA
274B-   60          RTS
```

So, uh, let's try not to end up there.

```
*2419L

; set reset vector to The Badlands
2419-   A9 02       LDA   #$02
241B-   8C FC FF    STY   $FFFC
241E-   8D FD FF    STA   $FFFD
2421-   8C F2 03    STY   $03F2
2424-   8D F3 03    STA   $03F3
2427-   49 A5       EOR   #$A5
2429-   8D F4 03    STA   $03F4

; also set input and output vectors to
; The Badlands
242C-   A0 03       LDY   #$03
242E-   A9 02       LDA   #$02
2430-   84 36       STY   $36
2432-   85 37       STA   $37
2434-   84 38       STY   $38
2436-   85 39       STA   $39
```

```
; also the BRK vector
2438-   8C F0 03    STY   $03F0
243B-   8D F1 03    STA   $03F1
243E-   A9 00       LDA   #$00
2440-   85 0A       STA   $0A
2442-   A6 2B       LDX   $2B
2444-   8E FF 02    STX   $02FF

; and continue elsewhere
2447-   4C 00 05    JMP   $0500

*2500L

2500-   A9 00       LDA   #$00
2502-   85 FF       STA   $FF
2504-   A9 1A       LDA   #$1A
2506-   20 C0 05    JSR   $05C0

*25C0L

; This is the subroutine that positions
; the drive head over a given quarter
; track. I've done my best to comment
; it, but quite honestly, I don't fully
; comprehend it.
;
; input: accumulator holds the phase of
;         the nearest whole track
25C0-   A2 13       LDX   #$13

; there's an "LDX #$0A" hidden in here
25C2-   2C A2 0A    BIT   $0AA2

; modify some code later (in the wait
; loop subroutine)
25C5-   8E 27 06    STX   $0627

; store the phase
25C8-   8D 49 06    STA   $0649
```

```
; if the drive head is already here,
; just exit
25CB-   C5 FF       CMP   $FF
25CD-   F0 54       BEQ   $2623

; figure out if we need to move the
; drive head forwards or backwards
25CF-   A9 00       LDA   #$00
25D1-   8D 4A 06    STA   $064A
25D4-   A5 FF       LDA   $FF
25D6-   8D 4B 06    STA   $064B
25D9-   38          SEC
25DA-   ED 49 06    SBC   $0649
25DD-   F0 35       BEQ   $2614
25DF-   B0 06       BCS   $25E7
25E1-   49 FF       EOR   #$FF
25E3-   E6 FF       INC   $FF
25E5-   90 04       BCC   $25EB
25E7-   69 FE       ADC   #$FE
25E9-   C6 FF       DEC   $FF
25EB-   CD 4A 06    CMP   $064A
25EE-   90 03       BCC   $25F3
25F0-   AD 4A 06    LDA   $064A
25F3-   C9 0C       CMP   #$0C
25F5-   B0 01       BCS   $25F8
25F7-   A8          TAY
25F8-   38          SEC

; trigger the appropriate stepper motor
25F9-   20 18 06    JSR   $0618

; wait exactly the right amount of time
25FC-   B9 31 06    LDA   $0631,Y
25FF-   20 26 06    JSR   $0626

; trigger another stepper motor
2602-   AD 4B 06    LDA   $064B
2605-   18          CLC
2606-   20 1A 06    JSR   $061A
2609-   B9 3D 06    LDA   $063D,Y
260C-   20 26 06    JSR   $0626
```

```
; increment the motor index (gets ANDed
; with #$03, so just increment it
; without end)
260F-   EE 4A 06    INC   $064A

; always branches
2612-   D0 C0        BNE   $25D4

; called from $05DD to move the head
; one more time and exit
2614-   20 26 06    JSR   $0626
2617-   18           CLC
2618-   A5 FF        LDA   $FF

; main entry point for stepper motor
; mover subroutine
261A-   29 03        AND   #$03
261C-   2A           ROL
261D-   05 2B        ORA   $2B
261F-   AA           TAX
2620-   BD 80 C0     LDA   $C080,X
2623-   A6 2B        LDX   $2B
2625-   60           RTS

; wait loop (modified repeatedly above)
2626-   A2 13        LDX   #$13
2628-   CA           DEX
2629-   D0 FD        BNE   $2628
262B-   38           SEC
262C-   E9 01        SBC   #$01
262E-   D0 F6        BNE   $2626
2630-   60           RTS

Backing up...
```

```
*2509L

; this is the multi-track read loop
2509-   A0 00       LDY   #$00
250B-   84 80       STY   $80
250D-   84 81       STY   $81

; get address from table (see below)
250F-   B9 3E 05    LDA   $053E,Y
2512-   F0 08       BEQ   $251C

; read one track of data (not shown)
2514-   20 1F 05    JSR   $051F

; increment the address table index
2517-   A4 81       LDY   $81
2519-   C8          INY

; always branches
251A-   D0 F1       BNE   $250D

; execution continues after all tracks
; are read
251C-   4C 4C 06    JMP   $064C
```

Here is the address table:

```
*253E.254D

253E-                     08 10
2540- 18 60 68 70 78 80 88 90
2548- 98 A0 A8 B0 B8 00
```

We already zeroed out both graphics pages. This read loop fills up most everything else, from $0800 to $BFFF in main memory.

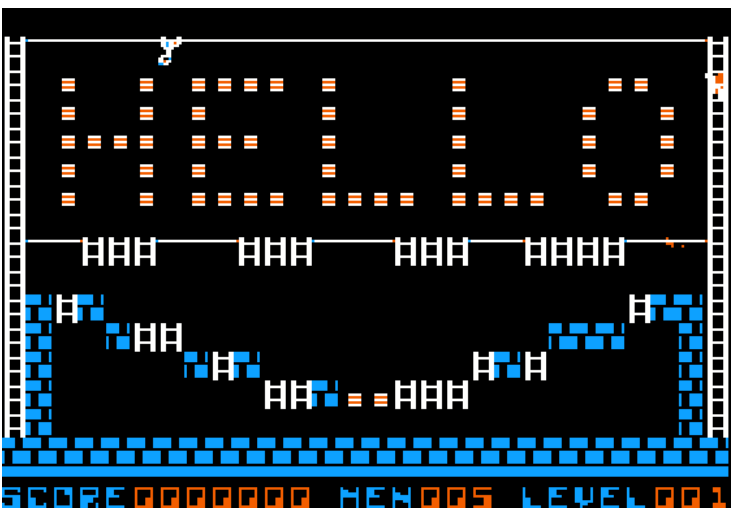
Execution continues at $064C.

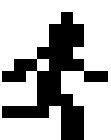
```
*264CL

; move drive head to track $22
264C-   A9 22       LDA   #$22
264E-   20 C0 05    JSR   $05C0

; turn off drive motor
2651-   BD 88 C0    LDA   $C088,X

; calculate a checksum (actually two of
; them, in $90 and $91) of all the data
; we just read
2654-   A0 00       LDY   #$00
2656-   84 90       STY   $90
2658-   84 91       STY   $91
265A-   84 92       STY   $92
265C-   84 81       STY   $81

; re-use address table to only include
; pages we read from disk when
; calculating the checksum
265E-   B9 3E 05    LDA   $053E,Y
2661-   F0 20       BEQ   $2683
2663-   85 93       STA   $93
2665-   A2 08       LDX   #$08
2667-   A0 00       LDY   #$00
2669-   B1 92       LDA   ($92),Y
266B-   45 90       EOR   $90
266D-   85 90       STA   $90
266F-   B1 92       LDA   ($92),Y
2671-   18          CLC
2672-   65 91       ADC   $91
2674-   85 91       STA   $91
2676-   C8          INY
2677-   D0 F0       BNE   $2669
2679-   E6 93       INC   $93
267B-   CA          DEX
267C-   D0 EB       BNE   $2669
267E-   A4 81       LDY   $81
2680-   C8          INY
2681-   D0 D9       BNE   $265C
```

```
; valid checksum?
2683-   A5 90        LDA   $90
2685-   CD FE 04     CMP   $04FE

; yup
2688-   F0 03        BEQ   $268D

; nope, jump to The Badlands (will show
; a different debugging character on
; screen, but still wipes memory and
; reboots)
268A-   4C 06 02     JMP   $0206

; verify second checksum
268D-   A5 91        LDA   $91

; valid?
268F-   CD FF 04     CMP   $04FF

; nope, off to The Badlands with you
2692-   D0 F6        BNE   $268A

; initialize zero page (not sure why;
; used by game code somehow)
2694-   A9 4C        LDA   #$4C
2696-   85 23        STA   $23

; initialize input and output vectors
; to $B7B5 (WTF?)
2698-   A0 B5        LDY   #$B5
269A-   A9 B7        LDA   #$B7
269C-   84 36        STY   $36
269E-   85 37        STA   $37
26A0-   84 38        STY   $38
26A2-   85 39        STA   $39
```

```
; initialize some parameters of a
; DOS 3.3-shaped RWTS (WTF?)
26A4-   A5 2B        LDA   $2B
26A6-   4A           LSR
26A7-   4A           LSR
26A8-   4A           LSR
26A9-   4A           LSR
26AA-   A8           TAY
26AB-   A9 22        LDA   #$22
26AD-   99 78 04     STA   $0478,Y
26B0-   A5 2B        LDA   $2B
26B2-   8D E9 B7     STA   $B7E9
26B5-   8D F7 B7     STA   $B7F7

; start game
26B8-   4C 00 60     JMP   $6000
```

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





Chapter 3
In Which We Fake It 'Til We Make It

```
*9600<C600.C6FFM

; set up callback #1
96F8-   A9 4C       LDA   #$4C
96FA-   8D 45 08    STA   $0845
96FD-   A9 0A       LDA   #$0A
96FF-   8D 46 08    STA   $0846
9702-   A9 97       LDA   #$97
9704-   8D 47 08    STA   $0847

; start the boot
9707-   4C 01 08    JMP   $0801

; (callback #1) set up callback #2
970A-   A9 14       LDA   #$14
970C-   8D 04 01    STA   $0104
970F-   A9 97       LDA   #$97
9711-   8D 05 01    STA   $0105

; continue the boot
9714-   60          RTS

; (callback #2) set up unconditional
; break at $06B8 once entire game is in
; memory
9715-   A9 59       LDA   #$59
9717-   8D B9 06    STA   $06B9
971A-   A9 FF       LDA   #$FF
971C-   8D BA 06    STA   $06BA
971F-   60          RTS

*BSAVE TRACE3,A$9600,L$120
*9600G
...reboots slot 6...
<beep>
```

```
*2000<800.1FFFM
*C500G
...
]BSAVE CLR.OBJ 0800-1FFF,A$2000,L$1800
]BRUN TRACE3
...reboots slot 6...
<beep>

*C500G
...
]BSAVE CLR.OBJ 6000-7FFF,A$6000,L$2000
]BRUN TRACE3
...reboots slot 6...
<beep>

*2000<8000.BFFFM
*C500G
...
]BSAVE CLR.OBJ 8000-BFFF,A$2000,L$4000
```

Examining my treasure trove, it appears that the following ranges are all zero:

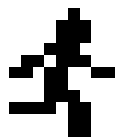
- $0800..$0EFF
- $1F00..$1FFF
- $2000..$5FFF (this was cleared during boot and remained untouched)

Furthermore, $B800..$BFFF contains a full DOS 3.3-shaped RWTS! I'm guessing this is used to load level data from those almost-normal tracks (T03-T0C). $8000..$BFFF is in memory at $2000, so the RWTS starts at $5800.

```
]BSAVE RWTS,A$5800,L$800
]BRUN ADVANCED DEMUFFIN 1.5

[S6,D1=original disk]
[S6,D2=formatted blank disk]
[S5,D1=my work disk]
```

```
[Q to switch to slot 5]

[R to load a new RWTS module]
  --> At $B8, load "RWTS" from drive 1

[6 to switch to slot 6]

[C to convert disk]

[press Y to change default values]

                  --v--

ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
========================================


INPUT ALL VALUES IN HEX


SECTORS PER TRACK? (13/16) 16

START TRACK: $03          <-- change this
START SECTOR: $00
END TRACK: $0C            <-- change this
END SECTOR: $0F

INCREMENT: 1

MAX # OF RETRIES: 0

COPY FROM DRIVE 1
TO DRIVE: 2
========================================
16SC $03,$00-$0C,$0F BY$01 S6,D1->S6,D2

                  --^--

And here we go...
```

```
                    --v--

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 $03,$00-$0C,$0F BY$01 S6,D1->S6,D2

                    --^--

[S6,D1=DOS 3.3 master disk]
[S5,D1=my work disk]

]PR#6
...
]CALL -151
*2800<B800.BFFFM
*BSAVE DOS33 RWTS,A$2800,L$800,S5,D1
```



Chapter 4
If You Wish To Play A Game, You Must First Create The Universe

I have the level data (untouched) on tracks $03-$0C. Now I'm going to write the game code to tracks $0D-$13. Track $0D will be $F00..$1EFF. Tracks $0E-$13 will be $6000..$BFFF. Instead of the original disk's RWTS at $B800..$BFFF, I'll substitute the standard DOS 3.3 RWTS that I captured from the DOS 3.3 master disk. Then I'll add a bootloader on track $00 to tie it all together.

```
[S6,D1=demuffin'd disk with T03-T0C]
[S5,D1=my work disk]

]PR#5
...

]BLOAD CLR.OBJ 0800-1FFF,A$800
]BLOAD CLR.OBJ 6000-7FFF,A$1F00
]BLOAD CLR.OBJ 8000-BFFF,A$3F00
]BLOAD DOS33 RWTS,A$7700
]CALL -151

; page count (decremented)
0300-   A9 70       LDA   #$70
0302-   85 FF       STA   $FF

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

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

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

; Convert logical to physical sector.
; 4boot reads tracks in physical sector
; order.
031E-   B9 40 03    LDA   $0340,Y
0321-   8D 8D 03    STA   $038D

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

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

; logical to physical sector mapping
*340.34F

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

; RWTS parameter table, pre-initialized
; with slot 6, drive 1, track $0D,
; sector $00, address $0F00, and RWTS
; write command ($02)
*388.397

0388- 01 60 01 00 0D 00 FB F7
0390- 00 0F 00 00 02 00 00 60
```

```
*BSAVE MAKE,A$300,L$98
*300G
...write write write...
```

Now I have the entire game on tracks $0D-$13 of a standard format disk.

The bootloader (which I've named 4boot) lives on track $00. T00S00 is boot0, which reuses the disk controller ROM routine to load boot1, which lives on sectors $0C-$0E and is loaded into $0900..$0B00.

Boot0 looks like this:

```
; decrement sector count
0801-   CE 12 08    DEC   $0812

; branch once we've read enough sectors
0804-   30 0B       BMI   $0811

; increment physical sector to read
0806-   E6 3D       INC   $3D

; $0880 is a sparse table of $C1..$C6,
; so this sets up the proper jump to
; the disk controller ROM based on the
; slot number
0808-   BD 80 08    LDA   $0880,X
080B-   8D 10 08    STA   $0810

; read a sector (exits via $0801)
080E-   4C 5C 00    JMP   $005C
```

```
; sector read loop exits to here (from
; $0804) -- note: by the time execution
; reaches here, $0812 is $FF, so this
; just resets the stack
0811-   A2 03       LDX   #$03
0813-   9A          TXS

; set up zero page (used by RWTS) and
; push an array of addresses to the
; stack at the same time
0814-   A2 17       LDX   #$17
0816-   BD 78 08    LDA   $0878,X
0819-   95 E8       STA   $E8,X
081B-   48          PHA
081C-   CA          DEX
081D-   D0 F7       BNE   $0816
081F-   60          RTS

*879.88F

0878-    88 FE 92 FE 2E FB 57
0880- FC 7B 0A FF 08 84 0A FF
0888- 08 FF B6 0F 01 00 00 00
```

These are pushed to the stack in reverse order, starting with $088F. When we hit the "RTS" at $0826, it pops the stack and jumps to $FE89, then $FE93, $FB2F, $FC58, $0A7C, $0900, $0A85, $0900, and $B700.

- $FE89, $FE93, $FB2F, and $FC58 are in ROM (IN#0, PR#0, TEXT, and HOME)
- $0A7C was loaded from sector $0D. It moves the drive head forward to track $0C.

- $0900 is the RWTS entry point. It advances the head one more track, then reads T0D at $0F00...$1EFF. (These values are stored in zero page, which we just set.)
- $0A85 resets the zero page values for the RWTS to read 6 more tracks at $6000.
- $0900 loads T0E-T13 to $6000..$BFFF
- $B700 is the final pre-game setup routine, which I adapted from the original late-stage bootloader. It jumps directly to the game start and never returns, so the other values on the stack are irrelevant.

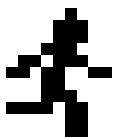
Here's the final pre-game setup routine at $B700:

```
; set up DOS 3.3 RWTS that's now in
; memory at $B800 (used for loading
; levels and saving high scores)
B700-   8E E9 B7    STX   $B7E9
B703-   8E F7 B7    STX   $B7F7
B706-   BD 88 C0    LDA   $C088,X
B709-   8A          TXA
B70A-   4A          LSR
B70B-   4A          LSR
B70C-   4A          LSR
B70D-   4A          LSR
B70E-   A8          TAY

; set current track so the RWTS doesn't
; grind the disk when it tries to load
; the first level
B70F-   A5 FD       LDA   $FD
B711-   99 78 04    STA   $0478,Y
```

```
; initialize zero page (copied verbatim
; from original disk bootloader)
B714-   A9 4C       LDA   #$4C
B716-   85 23       STA   $23

; yes, it really uses the I/O vectors
; at $36 and $38 to call the RWTS
B718-   A0 B5       LDY   #$B5
B71A-   A9 B7       LDA   #$B7
B71C-   84 36       STY   $36
B71E-   85 37       STA   $37
B720-   84 38       STY   $38
B722-   85 39       STA   $39

; clear both hi-res graphics screens
B724-   A2 40       LDX   #$40
B726-   A0 00       LDY   #$00
B728-   98          TYA
B729-   99 00 20    STA   $2000,Y
B72C-   C8          INY
B72D-   D0 FA       BNE   $B729
B72F-   EE 2B B7    INC   $B72B
B732-   CA          DEX
B733-   D0 F4       BNE   $B729

; show hi-res graphics screen 1
B735-   2C 54 C0    BIT   $C054
B738-   2C 57 C0    BIT   $C057
B73B-   2C 52 C0    BIT   $C052
B73E-   2C 50 C0    BIT   $C050

; jump to game start
B741-   4C 00 60    JMP   $6000
```

The entire boot takes less than five seconds.

Quod erat liberandum.



POSTSCRIPT: CHEAT CODES

I have *not* enabled the following cheats, but I have verified that they work. You can use any or all of them.

Zero page $7B holds the current level minus 1. It starts at 0.

Zero page $7D holds the number of lives remaining. It starts at 5. You gain one life every time you complete a level, but it maxes out at 255.

Start with 255 lives:
T0E,S00,$DD change "05" to "FF"

Infinite lives:
T0E,S07,$AB change "C6" to "24"

Make <Ctrl-R> advance to next level instead of restarting game:
T0F,S05,$B8 change "A9 01 85"
to "E6 7B E6"

EOF