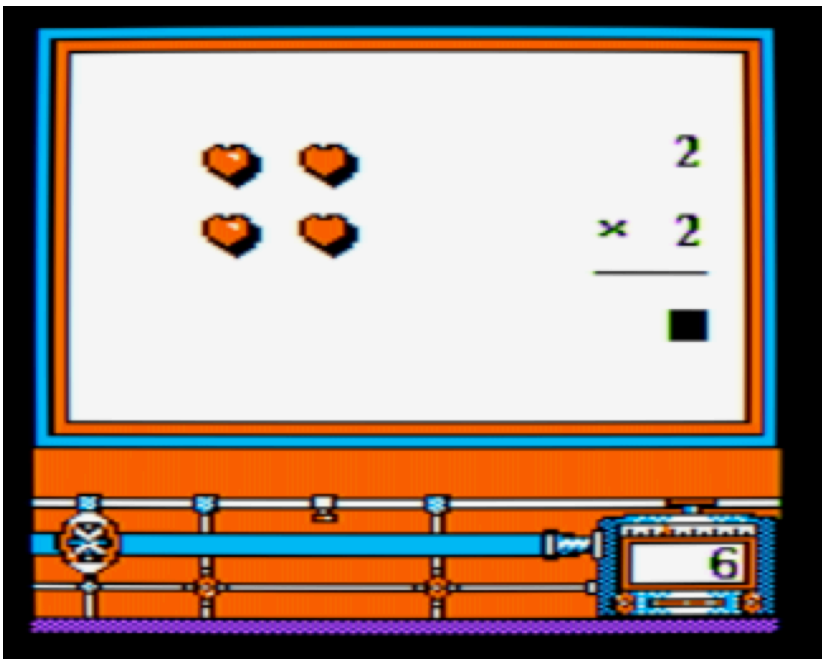
Stickybear Math 2



2016-02-22

Contents

```
-----------Stickybear Math 2-----------
A 4am crack                   2016-02-22
---------------------------------------

Name: Stickybear Math 2
Genre: educational
Year: 1986
Authors: Richard Hefter, Susan Dubicki
Publisher: Optimum Resource, Inc.
Media: single-sided 5.25-inch floppy
OS: custom with DOS 3.3 bootloader
Previous cracks: none of 5.25" version
Similar cracks:
  #586 Stickybear Parts of Speech
  #585 Map Skills
  #336 Car Builder
```

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  read error on first pass

Locksmith Fast Disk Backup
  fails to read T01,S0F; copy displays
  title screen, asks for player's name,
  then hangs

EDD 4 bit copy (no sync, no count)
  no read errors, but copy once again
  hangs after asking for player's name

Disk Fixer
  unable to read T01,S0F by any obvious
  combination of parameters

Why didn't COPYA work?
  intentionally bad sector on track $01

Why didn't Locksmith FDB work?
  probably a nibble check during boot
  that centers around the bad sector

Why didn't my EDD copy work?
  ditto

Next steps:

  1. Trace the boot
  2. Find the nibble check and skip it
  3. There is no step 3 (I hope)

Chapter 1
Hello, I'd Like To Have An Argument On The Stack, Please

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

]PR#5
...
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0
CAPTURING BOOT1
...reboots slot 6...
...reboots slot 5...
SAVING BOOT1
SAVING RWTS

]BLOAD BOOT1,A$2600
]CALL -151

*B600<2600.2EFFM ; move RWTS into place
                 ; (but don't overwrite
                 ; Diversi-DOS @ $BF00)
*B700L

; clear hi-res graphics screen 1
B700-   A9 00       LDA   #$00
B702-   A8          TAY
B703-   85 10       STA   $10
B705-   A2 20       LDX   #$20
B707-   86 11       STX   $11
B709-   91 10       STA   ($10),Y
B70B-   C8          INY
B70C-   D0 FB       BNE   $B709
B70E-   E6 11       INC   $11
B710-   CA          DEX
B711-   D0 F6       BNE   $B709
```

```
; and show it (now blank)
B713-   8D 50 C0    STA   $C050
B716-   8D 57 C0    STA   $C057
B719-   8D 54 C0    STA   $C054
B71C-   8D 52 C0    STA   $C052

; hmm, a JSR followed by garbage code
B71F-   20 5D B6    JSR   $B65D
B722-   01 0E       ORA   ($0E,X)
B724-   00         BRK
B725-   08         PHP
B726-   18         CLC

B727-   4C 00 08    JMP   $0800
```

```
*B65DL

; Ah! This subroutine uses the stack to
; pass in multiple arguments and store
; them in zero page
B65D-   68          PLA
B65E-   85 F0       STA   $F0
B660-   68          PLA
B661-   85 F1       STA   $F1
B663-   A0 01       LDY   #$01
B665-   B1 F0       LDA   ($F0),Y
B667-   85 54       STA   $54
B669-   C8          INY
B66A-   B1 F0       LDA   ($F0),Y
B66C-   85 55       STA   $55
B66E-   C8          INY
B66F-   B1 F0       LDA   ($F0),Y
B671-   85 58       STA   $58
B673-   C8          INY
B674-   B1 F0       LDA   ($F0),Y
B676-   85 59       STA   $59
B678-   C8          INY
B679-   B1 F0       LDA   ($F0),Y
B67B-   85 67       STA   $67
B67D-   18          CLC
B67E-   A5 F0       LDA   $F0
B680-   69 05       ADC   #$05
B682-   A8          TAY
B683-   A5 F1       LDA   $F1
B685-   69 00       ADC   #$00
```

```
; then munges the stack pointer to
; "return" to the next real instruction
; after the parameters
B687-   48          PHA
B688-   98          TYA
B689-   48          PHA
B68A-   A9 00       LDA   ##$00
B68C-   85 5A       STA   $5A
B68E-   85 5B       STA   $5B
B690-   85 5E       STA   $5E
B692-   85 53       STA   $53
B694-   A9 01       LDA   ##$01
B696-   85 50       STA   $50
B698-   85 52       STA   $52
B69A-   85 60       STA   $60
B69C-   85 62       STA   $62
B69E-   A9 60       LDA   ##$60
B6A0-   85 51       STA   $51
B6A2-   85 5F       STA   $5F
B6A4-   A9 00       LDA   ##$00
B6A6-   85 57       STA   $57
B6A8-   A9 61       LDA   ##$61
B6AA-   85 56       STA   $56
B6AC-   A9 EF       LDA   ##$EF
B6AE-   85 63       STA   $63
B6B0-   A9 D8       LDA   ##$D8
B6B2-   85 64       STA   $64
B6B4-   A5 67       LDA   $67
B6B6-   D0 01       BNE   $B6B9
B6B8-   60          RTS
[...]
```

```
B6B9-   A9 01        LDA   #$01
B6BB-   85 5C        STA   $5C
B6BD-   A9 00        LDA   #$00
B6BF-   A0 50        LDY   #$50
B6C1-   20 B5 B7     JSR   $B7B5
B6C4-   E6 59        INC   $59
B6C6-   C6 67        DEC   $67
B6C8-   C6 55        DEC   $55
B6CA-   10 E8        BPL   $B6B4
B6CC-   A9 0F        LDA   #$0F
B6CE-   85 55        STA   $55
B6D0-   E6 54        INC   $54
B6D2-   D0 E0        BNE   $B6B4
```

This is a multi-sector read loop. It calls the regular $B7B5 entry point to read sectors (at $B6C1). Interestingly, the RWTS parameter table is actually on zero page, starting at $50. That would mean that the parameters passed in (on the stack, after the JSR $B65D) are

1. start track ($54)
2. start sector ($55)
3. start address - low ($58)
4. start address - high ($59)
5. sector count ($67)

It uses logical sectors (via the RWTS). Sectors count down from $0F to $00, and tracks are decremented once the sector wraps around to $0F. The target address is incremented monotonically, and the sector count is decremented until it hits 0.

Revisiting the caller with this new understanding...

```
; read $18 sectors into $0800 starting
; from T01,S0E
B71F-   20 5D B6    JSR   $B65D
B722-  [01 0E 00 08 18]

; continue in the code we just read
B727-   4C 00 08    JMP   $0800
```

That's where I'll interrupt the boot.

Chapter 2
Bootus Interruptus

```
*9600<C600.C6FFM

; set up callback #1 after boot0 loads
; boot1
96F8-   A9 4C       LDA   #$4C
96FA-   8D 4A 08    STA   $084A
96FD-   A9 0A       LDA   #$0A
96FF-   8D 4B 08    STA   $084B
9702-   A9 97       LDA   #$97
9704-   8D 4C 08    STA   $084C

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

; (callback #1) set up callback #2
; instead of jumping to newly loaded
; code at $0800
970A-   A9 4C       LDA   #$4C
970C-   8D 27 B7    STA   $B727
970F-   A9 1C       LDA   #$1C
9711-   8D 28 B7    STA   $B728
9714-   A9 97       LDA   #$97
9716-   8D 29 B7    STA   $B729

; continue the boot
9719-   4C 00 B7    JMP   $B700
```

```
; (callback #2) copy newly loaded
; code to higher memory so it doesn't
; get overwritten by the HELLO program
; on my work disk
971C-   A2 18       LDX   #$18
971E-   A0 00       LDY   #$00
9720-   B9 00 08    LDA   $0800,Y
9723-   99 00 28    STA   $2800,Y
9726-   C8          INY
9727-   D0 F7       BNE   $9720
9729-   EE 22 97    INC   $9722
972C-   EE 25 97    INC   $9725
972F-   CA          DEX
9730-   D0 EE       BNE   $9720

; reboot to my work disk
9732-   4C 00 C5    JMP   $C500

*BSAVE TRACE,A$9600,L$135

*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT2 0800,A$2800,L$1800
]CALL -151

*800<2800.3FFFM

*800L

0800-   4C 98 09    JMP   $0998

Woohoo! It worked.
```

```
*998L

; A quick investigation reveals that
; this is also a multi-sector read loop
; that takes parameters on the stack,
; just like the one at $B65D. So this
; reads 6 sectors into $8401 starting
; at T05,S0F.
0998-   20 20 16    JSR   $1620
099B-  [05 0F 01 84 06]

; Read 17 sectors into $89C1 starting
; at T06,S0F.
09A0-   20 20 16    JSR   $1620
09A3-  [06 0F C1 89 17]

; call somewhere in the middle of the
; code we just read
09A8-   20 39 8F    JSR   $8F39
```

Once again, time to interrupt the boot.

Chapter 3
In Which It Gets Crazy

One wrinkle: I normally store my boot trace programs at $9600, but this disk is loading code into that memory page. So this trace will need to start at $A600 instead. This is possible because my work disk uses Diversi-DOS 64K, which moves DOS to the language card and frees up virtually all of main memory for my own programs. Handy!

```
*A600<C600.C6FFM

; set up callback #1 after boot0, then
; start the boot
A6F8-   A9 4C       LDA   #$4C
A6FA-   8D 4A 08    STA   $084A
A6FD-   A9 0A       LDA   #$0A
A6FF-   8D 4B 08    STA   $084B
A702-   A9 A7       LDA   #$A7
A704-   8D 4C 08    STA   $084C
A707-   4C 01 08    JMP   $0801

; (callback #1) set up callback #2 and
; continue the boot
A70A-   A9 4C       LDA   #$4C
A70C-   8D 27 B7    STA   $B727
A70F-   A9 1C       LDA   #$1C
A711-   8D 28 B7    STA   $B728
A714-   A9 A7       LDA   #$A7
A716-   8D 29 B7    STA   $B729
A719-   4C 00 B7    JMP   $B700
```

```
; (callback #2) set up callback #3 and
; continue the boot
A71C-   A9 4C       LDA   #$4C
A71E-   8D A8 09    STA   $09A8
A721-   A9 2E       LDA   #$2E
A723-   8D A9 09    STA   $09A9
A726-   A9 A7       LDA   #$A7
A728-   8D AA 09    STA   $09AA
A72B-   4C 00 08    JMP   $0800

; (callback #2) copy the code from
; $89C1 down to lower memory so it
; survives a reboot
A72E-   A2 17       LDX   #$17
A730-   A0 00       LDY   #$00
A732-   B9 C1 89    LDA   $89C1,Y
A735-   99 C1 29    STA   $29C1,Y
A738-   C8          INY
A739-   D0 F7       BNE   $A732
A73B-   EE 34 A7    INC   $A734
A73E-   EE 37 A7    INC   $A737
A741-   CA          DEX
A742-   D0 EE       BNE   $A732

; and reboot to my work disk
A744-   4C 00 C5    JMP   $C500

*BSAVE TRACE2,A$A600,L$147

Ew, that feels weird, though.

*9600G
...crashes...

Sorry, force of habit.
```

```
*A600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT2 89C1,A$29C1,A$1700
]CALL -151

*89C1<29C1.40C0M

*8F39L

; slow to 1 MHz (IIgs)
8F39-   AD 36 C0    LDA   $C036
8F3C-   29 7F       AND   #$7F
8F3E-   8D 36 C0    STA   $C036

; black screen border (IIgs)
8F41-   AD 34 C0    LDA   $C034
8F44-   29 F0       AND   #$F0
8F46-   8D 34 C0    STA   $C034

; reset vector jumps to monitor (?!?)
; (strange but true)
8F49-   A9 69       LDA   #$69
8F4B-   8D F2 03    STA   $03F2
8F4E-   A9 FF       LDA   #$FF
8F50-   8D F3 03    STA   $03F3
8F53-   A9 00       LDA   #$00
8F55-   8D F4 03    STA   $03F4
8F58-   8D 02 C0    STA   $C002
8F5B-   8D 04 C0    STA   $C004
8F5E-   8D 0C C0    STA   $C00C

; read 1 sector from T18,S0F into $4000
8F61-   20 20 16    JSR   $1620
8F64-  [18 0F 00 40 01]
```

```
; copy it to the text page (well that's
; vaguely suspicious)
8F69-   A2 00       LDX   #$00
8F6B-   86 D4       STX   $D4
8F6D-   BD 00 40    LDA   $4000,X
8F70-   9D 01 05    STA   $0501,X
8F73-   E8          INX
8F74-   E0 FF       CPX   #$FF
8F76-   F0 03       BEQ   $8F7B
8F78-   4C 6D 8F    JMP   $8F6D
8F7B-   BD 00 40    LDA   $4000,X
8F7E-   9D 01 05    STA   $0501,X

8F81-   20 89 8B    JSR   $8B89

*8B89L

8B89-   A9 20       LDA   #$20
8B8B-   85 5E       STA   $5E

; read $20 sectors into $2000, starting
; at T03,S0F (this is the title screen)
8B8D-   20 20 16    JSR   $1620
8B90-  [03 0F 00 20 20]

; memory move (not shown)
8B95-   20 98 16    JSR   $1698

; read $13 sectors into $A390, starting
at T16,S0C
8B98-   20 20 16    JSR   $1620
8B9B-  [16 0C 90 A3 13]
[...]
```

```
8BA0-   8D 57 C0    STA   $C057
8BA3-   8D 54 C0    STA   $C054
8BA6-   8D 52 C0    STA   $C052
8BA9-   8D 50 C0    STA   $C050
8BAC-   8D 10 C0    STA   $C010
.
.
.
```

The rest of this routine is non-disk-related. It's responsible for cycling through the credits screens, then it clears the screen and asks for the player to enter their name. Then it returns to the caller. It was called from $09A8. I need to pop the stack (literally and figuratively) and continue the trace from there.

Chapter 4
In Which It Gets Crazier

```
Backing up to $09A8...

*BLOAD BOOT2 0800,A$800
*9A8L

; this filled more memory, but it
; eventually returns, even on my non-
; working copy
09A8-   20 39 8F    JSR   $8F39

; read 3 sectors into $8701, starting
; from T05,S09
09AB-   20 20 16    JSR   $1620
09AE-  [05 09 01 87 03]

; and call into it
09B3-   20 07 87    JSR   $8707
```

And that's where I need to, you guessed it, interrupt the boot.

One tiny wrinkle: at $8B98, we read $13 sectors at $A390, which means I am running out of space to put my own code that will not be overwritten by the time I need to call back to it. So I'll need to get creative.

```
*A600<C600.C6FFM
.
. set up callback #1 and #2 (not shown,
. same as previous trace)
.
; (callback #2) set up callback #3
A71C-   A9 4C       LDA   #$4C
A71E-   8D B3 09    STA   $09B3
```

```
; yeah, the callback is at $100 -- the
; bottom of the stack
A721-   A9 00       LDA   #$00
A723-   8D B4 09    STA   $09B4
A726-   A9 01       LDA   #$01
A728-   8D B5 09    STA   $09B5

; copy my code to $100
A72B-   A0 20       LDY   #$20
A72D-   B9 40 A7    LDA   $A740,Y
A730-   99 00 01    STA   $0100,Y
A733-   88          DEY
A734-   10 F7       BPL   $A72D

; continue the boot
A736-   4C 00 08    JMP   $0800
...

; This code gets copied to $0100 and
; executed after the game loads 3
; sectors into $8701. For my own sanity
; I've unrolled the loop to eliminate
; any self-modifying code.
A740-   A2 03       LDX   #$03
A742-   A0 00       LDY   #$00
A744-   B9 01 87    LDA   $8701,Y
A747-   99 01 27    STA   $2701,Y
A74A-   B9 01 88    LDA   $8801,Y
A74D-   99 01 28    STA   $2801,Y
A750-   B9 01 89    LDA   $8901,Y
A753-   99 01 29    STA   $2901,Y
A756-   C8          INY
A757-   D0 EB       BNE   $A744

; reboot to my work disk
A759-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE3,A$A600,L$15C
*A600G
...reboots slot 6...
...displays title screen, runs through
   credits, asks for player name...
...reboots slot 5...

]BSAVE BOOT2 8701,A$2701,L$300
]CALL -151

*8701<2701.2A00M
*8707L

; hmm
8707-   A9 60       LDA   #$60
8709-   8D 2D 88    STA   $882D
870C-   A9 1E       LDA   #$1E
870E-   85 51       STA   $51

; double hmm
8710-   A9 D8       LDA   #$D8
8712-   8D 2B 88    STA   $882B
8715-   A9 20       LDA   #$20
8717-   8D 2A 88    STA   $882A
871A-   A9 02       LDA   #$02
871C-   8D 2C 88    STA   $882C

; wipe $46 bytes of memory at $1E00
; (zp$51 was just set at $870E)
871F-   A9 00       LDA   #$00
8721-   85 50       STA   $50
8723-   A0 64       LDY   #$64
8725-   A9 00       LDA   #$00
8727-   91 50       STA   ($50),Y
8729-   88          DEY
872A-   D0 FB       BNE   $8727

872C-   20 C4 87    JSR   $87C4
```

```
*87C4L

; read 2 sectors into $22D8, starting
; at T02,S06
87C4-   20 20 16    JSR   $1620
87C7-  [02 06 D8 22 02]

; copy them (well, most of them) into
; lower memory, starting at $02D8
87CC-   A0 00       LDY   #$00
87CE-   B9 D8 22    LDA   $22D8,Y
87D1-   99 D8 02    STA   $02D8,Y
87D4-   C8          INY
87D5-   D0 F7       BNE   $87CE
87D7-   A0 90       LDY   #$90
87D9-   B9 D6 23    LDA   $23D6,Y
87DC-   99 D6 03    STA   $03D6,Y
87DF-   88          DEY
87E0-   D0 F7       BNE   $87D9

87E2-   20 2A 88    JSR   $882A
87E5-   60          RTS

*882AL

; and... crash?
882A-   00          BRK
882B-   00          BRK
882C-   00          BRK
882D-   00          BRK
```

Ah! But all is not as it seems. Recall that we literally *just* modified the memory at $822A..$822D (at $8707). This is the code that ends up there:

```
*822A:20 D8 02 60

*822AL

822A-   20 D8 02    JSR   $02D8
822D-   60          RTS
```

...which is the code we just moved, that we read from T02,S06.

Highly suspect.

Let's interrupt at $87E2 and see what sneaky code ends up in the input buffer ($02D8), the page 3 vectors ($03D6), and overflowing onto the text page.

Chapter 5
On A Clear Day You Can Trace Forever

```
*A600<C600.C6FFM
.
. set up callbacks 1, 2, and 3 (not
. shown, same as previous trace)
.
; copy my next callback to $0100 and
; continue the boot
A72B-   A0 20       LDY   #$20
A72D-   B9 40 A7    LDA   $A740,Y
A730-   99 00 01    STA   $0100,Y
A733-   88          DEY
A734-   10 F7       BPL   $A72D
A736-   4C 00 08    JMP   $0800
...

; (callback #4) set up callback #5 and
; continue the boot
A740-   A9 4C       LDA   #$4C
A742-   8D E2 87    STA   $87E2

; remember, this chunk ends up at $0100
; (starting from $A740), so the next
; callback ends up at $0112
A745-   A9 12       LDA   #$12
A747-   8D E3 87    STA   $87E3
A74A-   A9 01       LDA   #$01
A74C-   8D E4 87    STA   $87E4
A74F-   4C 07 87    JMP   $8707

; (callback #5) copy the code we copied
; into the input buffer and text page
; to higher memory so it survives a
; reboot
A752-   A0 00       LDY   #$00
A754-   B9 D8 02    LDA   $02D8,Y
A757-   99 D8 22    STA   $22D8,Y
A75A-   B9 D8 03    LDA   $03D8,Y
A75D-   99 D8 23    STA   $23D8,Y
A760-   C8          INY
A761-   D0 F1       BNE   $A754
```

```
; and finally reboot to my work disk
A763-   4C 00 C5    JMP   $C500

*BSAVE TRACE4,A$A600,L$166
*A600G
...reboots slot 6...
...displays title screen, runs through
   credits, asks for player name...
...reboots slot 5...

]BSAVE BOOT2 02D8,A$22D8,L$200
]CALL -151

Since part of this ends up on the text
page, I'm going to leave it at $22D8.
Relative branches will look correct,
but absolute addresses will be off by
$2000.

]CALL -151

*22D8L

; whatever this is doing, it's going to
; keep doing it until it works (note
; the infinite loop if the carry bit
; is set)
22D8-   A0 3C       LDY   #$3C
22DA-   A9 04       LDA   #$04
22DC-   20 E2 02    JSR   $02E2
22DF-   B0 F7       BCS   $22D8
22E1-   60          RTS
```

```
; save registers
22E2-   48          PHA
22E3-   8A          TXA
22E4-   48          PHA
22E5-   98          TYA
22E6-   48          PHA

; do something
22E7-   20 F8 02    JSR   $02F8

; restore registers
22EA-   68          PLA
22EB-   A8          TAY
22EC-   68          PLA
22ED-   AA          TAX
22EE-   68          PLA

; read a sector via RWTS
22EF-   20 B5 B7    JSR   $B7B5

; do the same thing again (but save and
; restore the processor flags)
22F2-   08          PHP
22F3-   20 F8 02    JSR   $02F8
22F6-   28          PLP

; and return to caller with the flags
; from the RWTS call
22F7-   60          RTS
```

OK, this is a bit convoluted, but it's not intentionally obfuscated. It's just cautious. The heart of it is a standard RWTS call (at $02EF) which absolutely has to work, otherwise $02DF will keep branching back to $02D8 to try again. The RWTS parameter table is at $043C (passed in A and Y, set at $02D8). Before and after the RWTS call, we call a routine at $02F8. The RWTS parameter table is never modified, so it's just whatever was read from disk.

```
*243C.2450

243C- 01 60 01 00
2440- 01 0F 4D 04 00 1E 00 00
      ^^ ^^       ^^^^^
   track sector  address

2448- 01 00 00 60 01 00 01 EF
2450- D8
```

That's reading from slot 6, drive 1, track $01, sector $0F (aha! that's the unreadable sector), into $1E00.

```
*22F8L

; take two bytes from a data structure
; starting at $0321 and store them in
; zero page $02/$03
22F8-   A2 00       LDX   #$00
22FA-   A0 00       LDY   #$00
22FC-   BD 21 03    LDA   $0321,X
22FF-   E8          INX
2300-   85 02       STA   $02
2302-   BD 21 03    LDA   $0321,X
2305-   E8          INX
2306-   85 03       STA   $03
2308-   C5 02       CMP   $02
230A-   D0 05       BNE   $2311
230C-   C9 00       CMP   #$00
230E-   D0 01       BNE   $2311
2310-   60          RTS

; swap the byte at ($02),Y with the
; byte at $0321,X
2311-   BD 21 03    LDA   $0321,X
2314-   48          PHA
2315-   B1 02       LDA   ($02),Y
2317-   9D 21 03    STA   $0321,X
231A-   E8          INX
231B-   68          PLA
231C-   91 02       STA   ($02),Y

; and loop back (will exit if the BNEs
; at $030A and $030E fail)
231E-   4C FC 02    JMP   $02FC
```

So we're swapping some bytes in memory, then (since we call the same routine again) swapping them back after the RWTS call at $02EF. This is driven by a data structure (starting at $0321) that contains an array of triples (two byte address + one byte value) followed by two null bytes.

Here is the data structure:

```
*2321.2334

2321- C2 B8 4C
      C3 B8 8E
      C4 B8 03
      DC B8 4C
      DD B8 35
      DE B8 03
      00 00
```

So we're swapping three bytes at $B8C2 (4C 8E 03) and three bytes at $B8DC (4C 35 03). That... is right in the middle of RWTS code. $B8C2 is the POSTNIBBLE routine that converts the 342 nibbles on disk into 256 bytes in memory. $B8DC is the READ routine. And we're completely replacing (well, redirecting) both of them to custom routines that we just read from disk, in order to read the unreadable sector on track $01.

```
Let's see what's at $038E and $0335.

*238EL

; new POSTNIBBLE routine
238E-   A0 87        LDY   #$87
2390-   20 AB 03     JSR   $03AB
2393-   88           DEY
2394-   20 AB 03     JSR   $03AB
2397-   0A           ASL
2398-   0A           ASL
2399-   0A           ASL
239A-   0A           ASL
239B-   85 00        STA   $00
239D-   20 AB 03     JSR   $03AB
23A0-   05 00        ORA   $00
23A2-   91 3E        STA   ($3E),Y
23A4-   98           TYA
23A5-   88           DEY
23A6-   C9 00        CMP   #$00
23A8-   D0 EA        BNE   $2394
23AA-   60           RTS
23AB-   98           TYA
23AC-   48           PHA
23AD-   C9 87        CMP   #$87
23AF-   D0 19        BNE   $23CA
23B1-   A0 55        LDY   #$55
23B3-   AD C0 FB     LDA   $FBC0
23B6-   F0 0C        BEQ   $23C4
23B8-   AE 8D 03     LDX   $038D
23BB-   BD 8D C0     LDA   $C08D,X
23BE-   BD 8E C0     LDA   $C08E,X
23C1-   10 01        BPL   $23C4
23C3-   88           DEY
[...]
```

```
23C4-   84 02        STY   $02
23C6-   A2 00        LDX   #$00
23C8-   86 01        STX   $01
23CA-   A4 02        LDY   $02
23CC-   A6 01        LDX   $01
23CE-   30 0C        BMI   $23DC
23D0-   BE FF BB     LDX   $BBFF,Y
23D3-   88           DEY
23D4-   D0 0A        BNE   $23E0
23D6-   A9 FF        LDA   #$FF
23D8-   85 01        STA   $01
23DA-   D0 04        BNE   $23E0
23DC-   BE 00 BB     LDX   $BB00,Y
23DF-   C8           INY
23E0-   84 02        STY   $02
23E2-   68           PLA
23E3-   A8           TAY
23E4-   BD 3C 03     LDA   $033C,X
23E7-   60           RTS

*2335L

; new READ routine
2335-   A0 20        LDY   #$20
2337-   8E 8D 03     STX   $038D
233A-   88           DEY
233B-   F0 4E        BEQ   $238B
233D-   BD 8C C0     LDA   $C08C,X
2340-   10 FB        BPL   $233D
2342-   49 D5        EOR   #$D5
2344-   D0 F4        BNE   $233A
2346-   EA           NOP
2347-   BD 8C C0     LDA   $C08C,X
234A-   10 FB        BPL   $2347
234C-   C9 AA        CMP   #$AA
234E-   D0 F2        BNE   $2342
2350-   A0 56        LDY   #$56
2352-   BD 8C C0     LDA   $C08C,X
2355-   10 FB        BPL   $2352
[...]
```

```
2357-   EA          NOP
2358-   EA          NOP
2359-   9D 8D C0    STA   $C08D,X
235C-   88          DEY
235D-   84 26       STY   $26
235F-   BD 8C C0    LDA   $C08C,X
2362-   10 FB       BPL   $235F
2364-   BC 00 BA    LDY   $BA00,X
2367-   A4 26       LDY   $26
2369-   99 00 BC    STA   $BC00,Y
236C-   D0 EE       BNE   $235C
236E-   84 26       STY   $26
2370-   BD 8C C0    LDA   $C08C,X
2373-   10 FB       BPL   $2370
2375-   BC 00 BA    LDY   $BA00,X
2378-   A4 26       LDY   $26
237A-   99 00 BB    STA   $BB00,Y
237D-   C8          INY
237E-   D0 EE       BNE   $236E
2380-   AD 52 BC    LDA   $BC52
2383-   29 F0       AND   ##$F0
2385-   C9 90       CMP   ##$90
2387-   F0 02       BEQ   $238B
2389-   18          CLC
238A-   60          RTS
238B-   38          SEC
238C-   60          RTS
238D-   00          BRK
```

And that's how we're going to read the unreadable sector: we're going to let this code do it for us.

Chapter 6
In Which We Limp Across The Finish Line, Unbroken, And Live To Crack Another Day

```
*A600<C600.C6FFM
.
. set up callbacks 1, 2, and 3 (not
. shown, same as previous trace)
.
; (callback #4 -- ends up at $0100)
; set up callback #5 and continue the
; boot
A740-   A9 4C       LDA   #$4C
A742-   8D E2 87    STA   $87E2
A745-   A9 12       LDA   #$12
A747-   8D E3 87    STA   $87E3
A74A-   A9 01       LDA   #$01
A74C-   8D E4 87    STA   $87E4
A74F-   4C 07 87    JMP   $8707

; (callback #5 -- ends up at $0112)
; force routine at $02D8 to reboot to
; my work disk immediately after it
; reads the unreadable sector with the
; modified RWTS
A752-   A9 4C       LDA   #$4C
A754-   8D DF 02    STA   $02DF
A757-   A9 00       LDA   #$00
A759-   8D E0 02    STA   $02E0
A75C-   A9 C5       LDA   #$C5
A75E-   8D E1 02    STA   $02E1

; continue the boot
A761-   4C D8 02    JMP   $02D8

*BSAVE TRACE5,A$A600,L$164
*A600G
...reboots slot 6...
...displays title screen, runs through
   credits, asks for player name...
...reboots slot 5...
```

```
]BSAVE BOOT2 1E00,A$1E00,L$100
]CALL -151

*1E00L

1E00-   10 07        BPL   $1E09
1E02-   18           CLC
1E03-   A0 05        LDY   #$05
1E05-   71 75        ADC   ($75),Y
1E07-   A0 03        LDY   #$03
1E09-   49 FF        EOR   #$FF
1E0B-   18           CLC
1E0C-   69 01        ADC   #$01
1E0E-   10 07        BPL   $1E17
1E10-   18           CLC
1E11-   A0 05        LDY   #$05
1E13-   71 75        ADC   ($75),Y
1E15-   A0 03        LDY   #$03
1E17-   91 75        STA   ($75),Y
1E19-   4C D9 1D     JMP   $1DD9
1E1C-   20 B4 1D     JSR   $1DB4
1E1F-   A0 0F        LDY   #$0F
1E21-   A2 FF        LDX   #$FF
1E23-   D1 75        CMP   ($75),Y
1E25-   B0 02        BCS   $1E29
```

Well, I'm not sure what this is, but it is most decidedly something. This was not just a check to ensure that the unreadable sector was there. The data on that sector actually matters.

OK, no problem. I have the data now. I can write it back to T01,S0F on my non-working copy. Then, if I disable the RWTS swapper at $02F8, the program will just read T01,S0F with a standard RWTS.

```
08C0-   A9 08       LDA   #$08
08C2-   A0 E8       LDY   #$E8
08C4-   4C D9 03    JMP   $03D9

08E8- 01 60 01 FE 01 0F FB 08
                  ^^ ^^
                track sector

08F0- 00 1E 00 00 02 00 FE 60
      ^^^^^        ^^
   address     write

08F8- 01 00 00 00 01 EF D8 00

*BSAVE WRITE T01S0F,A$8C0,L$40
*BLOAD BOOT2 1E00,A$1E00

[S6,D1=non-working copy]

*8C0G
...write write write...

$02D8 was read from T02,S06 (at $87C4).
I want to put an "RTS" at $02F8 to
disable the RWTS swapper, so that's at
offset $20. I shouldn't need to change
anything else.

T02,S06,$20 change "A2" to "60"

]PR#6
...works...


Quod erat liberandum.
```

A 4am crack No. 609

EOF