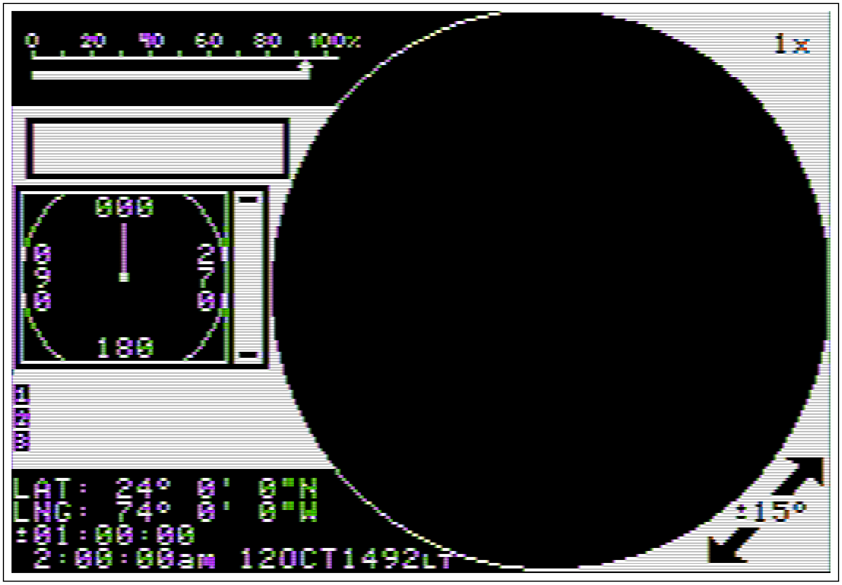
The Observatory



2015-10-22

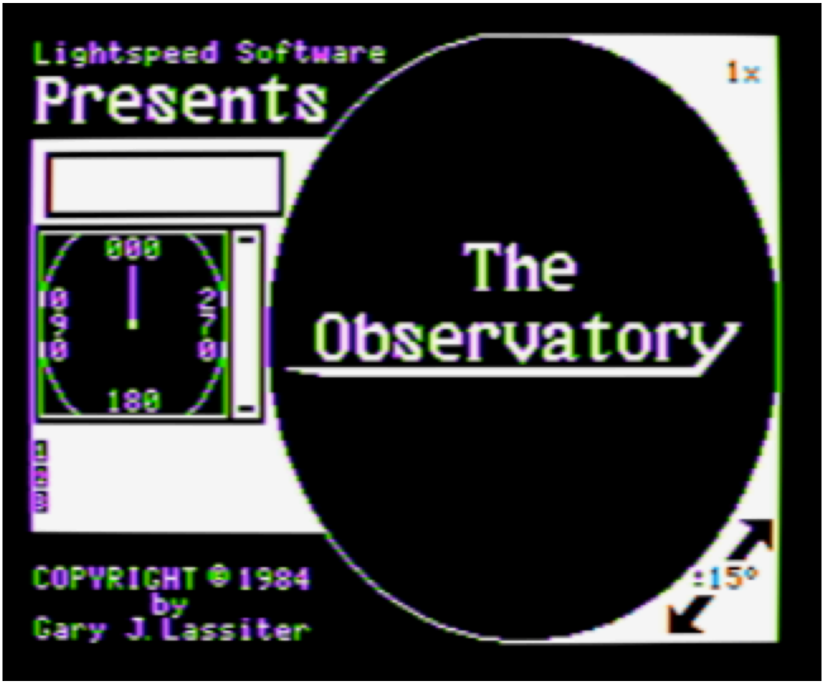
4am
to deprotect
and preserve

Contents

```
------------The Observatory-------------
a san inc crack               2015-10-22
--------------------. updated 2015-10-24
                    |___________________

Name: The Observatory
Genre: educational
Year: 1984
Authors: Gary J. Lassiter
Publisher: Lightspeed Software
Media: single-sided 5.25-inch floppy
OS: custom
Previous cracks: none

      ____________________________
     {                            }
     { "If you're going through    }
     {  hell, keep going."        }
     {                            }
     {  -variously misattributed  }
     {____________________________}
```

Prologue
From Each According To His Ability

This crack was a joint venture between me (4am) and qkumba of san inc, in the sense that he burned through the copy protection like flash paper, and I muttered "But that's insane!" over and over while I wrote these docs. Everyone seemed OK with this division of labor.



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)
  copy loads several tracks, then hangs
  with the drive motor running

Copy ][+ nibble editor
  T00 -> custom epilogues
  T01-T10 -> weird, 4-and-4 encoded?
  T12-T20 -> custom prologues
    ("AA D5 96" / "AA D5 AD")

Disk Fixer
  ["O" -> "Input/Output Control"]
    set CHECKSUM ENABLED to "NO"
  T00 readable, looks entirely custom
  (not DOS 3.3, not ProDOS, not Pascal)
  no sign of a disk catalog or OS

Why didn't COPYA work?
  so many reasons

Why didn't Locksmith FDB work?
  ditto

Why didn't my EDD copy work?
  I don't know. Maybe a nibble check
  during boot? Could also be reading
  from half or quarter tracks at some
  point. (The original disk has that
  "rapid fire" sound during the second
  half of the boot.)
```

Next steps:

1. Trace the boot
2. ???

Chapter 1
Look At This Stuff,
It's All An Exact Replica

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

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

]BLOAD BOOT0,A$800
]CALL-151

*800

; the disk controller PROM reads two
; sectors from disk, at $0800 and $0900
; (most disks have a $01 here)
0800- 02

*801L

; jump over denibbilisation table
0801-   78              SEI
0802-   4C 6F 08        JMP   $086F

*86FL

; enable RAM bank 1 in languard card
; this has the (unpleasant) side-effect
; of making the machine hang if anyone
; attempts to use the ROM (like an evil
; hacker breaking to monitor)
086F-   AD 8B C0        LDA   $C08B
0872-   AD 8B C0        LDA   $C08B
```

```
; prepare to write to $B400+
0875-   A9 00          LDA   #$00
0877-   85 00          STA   $00
0879-   A9 B4          LDA   #$B4
087B-   85 01          STA   $01

; sector 3
087D-   A9 03          LDA   #$03
087F-   85 04          STA   $04
```

After reusing the PROM to load the two boot sectors, the program proceeds to reproduce the behaviour of said PROM to load additional sectors. I have to assume the author knew about the $Cx5C entrypoint, so why duplicate work?

Anyway, this is an exact replica of the built-in PROM: standard address and data prologues, but no epilogue check. (Track $00 uses non-standard epilogue bytes.)

```
0881-   18          CLC
0882-   08          PHP
0883-   BD 8C C0    LDA   $C08C,X
0886-   10 FB       BPL   $0883
0888-   49 D5       EOR   #$D5
088A-   D0 F7       BNE   $0883
088C-   BD 8C C0    LDA   $C08C,X
088F-   10 FB       BPL   $088C
0891-   C9 AA       CMP   #$AA
0893-   D0 F3       BNE   $0888
0895-   78          SEI
0896-   BD 8C C0    LDA   $C08C,X
0899-   10 FB       BPL   $0896
089B-   C9 96       CMP   #$96
089D-   F0 09       BEQ   $08A8
089F-   28          PLP
08A0-   90 DF       BCC   $0881
08A2-   49 AD       EOR   #$AD
08A4-   F0 1D       BEQ   $08C3
08A6-   D0 D9       BNE   $0881
.
.  &c.
.

; load every second sector because
; denibbilisation is slow and we miss
; sectors while we're doing it
0906-   E6 01       INC   $01
0908-   E6 04       INC   $04
090A-   E6 04       INC   $04
090C-   A6 2B       LDX   $2B

; read until end of track
090E-   A5 04       LDA   $04
0910-   C9 11       CMP   #$11
0912-   D0 DC       BNE   $08F0
```

```
; erase hi-res screen 2
0914-   A9 40       LDA   #$40
0916-   85 03       STA   $03
0918-   A9 00       LDA   #$00
091A-   85 02       STA   $02
091C-   A8          TAY
091D-   91 02       STA   ($02),Y
091F-   C8          INY
0920-   D0 FB       BNE   $091D
0922-   E6 03       INC   $03
0924-   A6 03       LDX   $03
0926-   E0 60       CPX   #$60
0928-   D0 F3       BNE   $091D

; show hi-res screen 2 (now blank)
092A-   AD 50 C0    LDA   $C050
092D-   AD 52 C0    LDA   $C052
0930-   AD 55 C0    LDA   $C055
0933-   AD 57 C0    LDA   $C057

; continue to boot1
0936-   4C 00 B4    JMP   $B400
```

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

Chapter 2
Another Day, Another RWTS

```
*1600<C600.C6FFM

; set up callback instead of jumping to
; $B400
16F8-   A9 05       LDA   #$05
16FA-   8D 37 09    STA   $0937
16FD-   A9 17       LDA   #$17
16FF-   8D 38 09    STA   $0938

; start the boot
1702-   4C 01 08    JMP   $0801

; callback is here --
; copy boot1 to the hi-res screen so it
; survives a reboot to my work disk
1705-   A0 00       LDY   #$00
1707-   B9 00 B4    LDA   $B400,Y
170A-   99 00 20    STA   $2000,Y
170D-   C8          INY
170E-   D0 F7       BNE   $1707
1710-   EE 09 16    INC   $1609
1713-   EE 0C 16    INC   $160C
1716-   AD 09 16    LDA   $1609
1719-   C9 BB       CMP   #$BB
171B-   D0 EA       BNE   $1707
171D-   AD 81 C0    LDA   $C081
1720-   AD 51 C0    LDA   $C051
1723-   4C 00 C5    JMP   $C500

*BSAVE TRACE1,A$1600,L$126
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1,A$2000,L$700
]BLOAD BOOT1,A$2400
```

I'm going to leave boot1 at $2400.
Relative branches will look correct,
but absolute addresses will be off by
$9000.

```
]CALL -151

*2400L

2400-   4C C1 B5    JMP   $B5C1

*25C1L

; get boot slot (x16)
25C1-   A6 2B       LDX   $2B

; pretend that track was 1
25C3-   A9 01       LDA   #$01
25C5-   85 02       STA   $02

; request track 0
25C7-   A9 00       LDA   #$00
25C9-   85 14       STA   $14
25CB-   85 03       STA   $03
25CD-   20 B6 B6    JSR   $B6B6
```

```
*26B6L

; ordinary step routine for moving the
; drive arm to a specified phase
; (2 phases = 1 track)
; zp$02 has "current" phase
; zp$03 has new phase
26B6-   A5 02         LDA   $02
26B8-   C5 03         CMP   $03
26BA-   F0 1C         BEQ   $26D8
26BC-   90 04         BCC   $26C2
26BE-   C6 02         DEC   $02
26C0-   C6 02         DEC   $02
26C2-   E6 02         INC   $02
26C4-   A5 02         LDA   $02
26C6-   48            PHA

; hit proper stepper motor (not shown)
26C7-   38            SEC
26C8-   20 D9 B6      JSR   $B6D9

; wait routine (not shown)
26CB-   A9 56         LDA   #$56
26CD-   20 E3 B6      JSR   $B6E3
26D0-   68            PLA

; hit other stepper motor (not shown)
26D1-   18            CLC
26D2-   20 D9 B6      JSR   $B6D9

; branch back to see if we need to move
; any further
26D5-   4C B6 B6      JMP   $B6B6
26D8-   60            RTS
```

Continuing from $B5D0...

```
; prepare to read to $2000+
25D0-   A9 20          LDA   #$20
25D2-   85 15          STA   $15

; request track $12
25D4-   A9 24          LDA   #$24
25D6-   85 03          STA   $03
25D8-   20 B6 B6       JSR   $B6B6

; start with sector #$0f
25DB-   A9 0F          LDA   #$0F
25DD-   85 04          STA   $04
25DF-   20 39 B8       JSR   $B839

*2839L
```

Another RWTS! Counting the one embedded in boot0, this is RWTS #2. It reads 6-and-2 encoded sector data with a modified address prologue ("AA D5 96"), modified data prologue ("AA D5 AD"), no epilogue checking, and no address field parsing except for the sector number.

```
; get boot slot (x16)
2839-   A6 2B          LDX   $2B
```

```
; match "AA D5 96" address prologue
283B-    BD 8C C0    LDA   $C08C,X
283E-    10 FB       BPL   $283B
2840-    C9 AA       CMP   #$AA
2842-    D0 F5       BNE   $2839
2844-    78          SEI
2845-    BD 8C C0    LDA   $C08C,X
2848-    10 FB       BPL   $2845
284A-    C9 D5       CMP   #$D5
284C-    D0 F2       BNE   $2840
284E-    78          SEI
284F-    BD 8C C0    LDA   $C08C,X
2852-    10 FB       BPL   $284F
2854-    C9 96       CMP   #$96
2856-    D0 E8       BNE   $2840
```

```
; no volume, track, or checksum here --
; just the sector number
2858-   78          SEI
2859-   BD 8C C0    LDA   $C08C,X
285C-   10 FB       BPL   $2859
285E-   2A          ROL
285F-   85 10       STA   $10
2861-   BD 8C C0    LDA   $C08C,X
2864-   10 FB       BPL   $2861
2866-   25 10       AND   $10
2868-   85 10       STA   $10
286A-   A4 04       LDY   $04
286C-   B9 EF B6    LDA   $B6EF,Y
286F-   C5 10       CMP   $10
2871-   D0 C6       BNE   $2839
2873-   A0 20       LDY   #$20
2875-   88          DEY
2876-   F0 C1       BEQ   $2839

; match "AA D5 AD" data prologue
2878-   BD 8C C0    LDA   $C08C,X
287B-   10 FB       BPL   $2878
287D-   49 AA       EOR   #$AA
287F-   D0 F4       BNE   $2875
2881-   78          SEI
2882-   BD 8C C0    LDA   $C08C,X
2885-   10 FB       BPL   $2882
2887-   C9 D5       CMP   #$D5
2889-   D0 F2       BNE   $287D
288B-   A0 56       LDY   #$56
288D-   BD 8C C0    LDA   $C08C,X
2890-   10 FB       BPL   $288D
2892-   C9 AD       CMP   #$AD
2894-   D0 E7       BNE   $287D
```

```
; standard 6-and-2 decoding
2896-    A9 00          LDA   #$00
2898-    88             DEY
2899-    84 10          STY   $10
289B-    BC 8C C0       LDY   $C08C,X
289E-    10 FB          BPL   $289B
28A0-    59 C1 B4       EOR   $B4C1,Y
28A3-    A4 10          LDY   $10
28A5-    99 2C 00       STA   $002C,Y
28A8-    D0 EE          BNE   $2898
28AA-    84 10          STY   $10
28AC-    BC 8C C0       LDY   $C08C,X
28AF-    10 FB          BPL   $28AC
28B1-    59 C1 B4       EOR   $B4C1,Y
28B4-    A4 10          LDY   $10
28B6-    91 14          STA   ($14),Y
28B8-    C8             INY
28B9-    D0 EF          BNE   $28AA
28BB-    BC 8C C0       LDY   $C08C,X
28BE-    10 FB          BPL   $28BB
28C0-    D9 C1 B4       CMP   $B4C1,Y
28C3-    F0 03          BEQ   $28C8
28C5-    4C 39 B8       JMP   $B839
28C8-    A0 00          LDY   #$00
28CA-    A2 56          LDX   #$56
28CC-    CA             DEX
28CD-    30 FB          BMI   $28CA
28CF-    B1 14          LDA   ($14),Y
28D1-    56 2C          LSR   $2C,X
28D3-    2A             ROL
28D4-    56 2C          LSR   $2C,X
28D6-    2A             ROL
28D7-    91 14          STA   ($14),Y
28D9-    C8             INY
28DA-    C0 00          CPY   #$00
28DC-    D0 EE          BNE   $28CC

; no epilogue or checksum byte
28DE-    60             RTS
```

Continuing from $B5E2...

```
; increment address and exit when zero
25E2-   E6 15       INC   $15
25E4-   A5 15       LDA   $15
25E6-   F0 28       BEQ   $2610

; special case page $40 to show the
; title screen as soon as it becomes
; available
25E8-   C9 40       CMP   #$40
25EA-   D0 0F       BNE   $25FB
25EC-   AD 50 C0    LDA   $C050
25EF-   AD 52 C0    LDA   $C052
25F2-   AD 54 C0    LDA   $C054
25F5-   AD 57 C0    LDA   $C057
25F8-   4C 05 B6    JMP   $B605

; special case page $B4 to skip to $D0
; (i.e. RAM bank 1, which we switched
; on at $086F)
25FB-   C9 B4       CMP   #$B4
25FD-   D0 06       BNE   $2605
25FF-   A9 D0       LDA   #$D0
2601-   85 15       STA   $15
2603-   D0 04       BNE   $2609

; decrement sector
2605-   C6 04       DEC   $04
2607-   10 D6       BPL   $25DF

; increment track
2609-   E6 03       INC   $03
260B-   E6 03       INC   $03

; unconditional jump back for more
260D-   4C D8 B5    JMP   $B5D8
```

```
; Execution continues here (from $B5E6)
; when the target page hits $00. That's
; not as weird as it sounds. Remember,
; we were reading into $D000...$FFFF in
; RAM bank 1, so the target page was
; going to hit $00 eventually. Now it
; has, and now we enable the other RAM
; bank (#2) and read one more track.
2610-   AD 83 C0     LDA   $C083
2613-   AD 83 C0     LDA   $C083

; restart on sector $0F
2616-   A9 0F        LDA   #$0F
2618-   85 04        STA   $04

; to $D000 (but in RAM bank 2)
261A-   A9 D0        LDA   #$D0
261C-   85 15        STA   $15

; move to next whole track
261E-   E6 03        INC   $03
2620-   E6 03        INC   $03
2622-   20 B6 B6     JSR   $B6B6

; read a sector
2625-   20 39 B8     JSR   $B839

; increment address
2628-   E6 15        INC   $15

; decrement sector and repeat for the
; whole track
262A-   C6 04        DEC   $04
262C-   10 F7        BPL   $2625
```

Now we've filled $2000..$B3FF in main memory, $D000..$FFFF in RAM bank 1, and $D000..$DFFF in RAM bank 2. And we're not done yet. But I can interrupt the boot to capture what we've read so far.

Chapter 3
Capture All The Things!

```
*1600<C600.C6FFM

; set up callback #1 after boot0 loads
; boot1 at $B400
16F8-   A9 05       LDA   #$05
16FA-   8D 37 09    STA   $0937
16FD-   A9 17       LDA   #$17
16FF-   8D 38 09    STA   $0938

; start the boot
1702-   4C 01 08    JMP   $0801

; callback #1 is here --
; set up callback #2 at $B5FF (after
; everything has been loaded into main
; memory at $2000..$B3FF)
1705-   A9 4C       LDA   #$4C
1707-   8D FF B5    STA   $B5FF
170A-   A9 17       LDA   #$17
170C-   8D 00 B6    STA   $B600
170F-   A9 17       LDA   #$17
1711-   8D 01 B6    STA   $B601

; continue the boot
1714-   4C 00 B4    JMP   $B400

; callback #2 is here --
; reset memory softswitches, show text
; page, and reboot to my work disk
1717-   AD 81 C0    LDA   $C081
171A-   AD 51 C0    LDA   $C051
171D-   4C 00 C5    JMP   $C500

*BSAVE TRACE2,A$1600,L$120
*1600G
...reboots slot 6...
...reboots slot 5...
```

```
]BSAVE THE OBSERVATORY.OBJ 2000-7FFF,
 A$2000,L$6000

]CALL -151

*1600<C600.C6FFM

; set up callback #1 after boot0
16F8-    A9 05          LDA   #$05
16FA-    8D 37 09       STA   $0937
16FD-    A9 17          LDA   #$17
16FF-    8D 38 09       STA   $0938

; start the boot
1702-    4C 01 08       JMP   $0801

; callback #1 is here --
; set up callback #2 after main memory
; is full
1705-    A9 4C          LDA   #$4C
1707-    8D FF B5       STA   $B5FF
170A-    A9 17          LDA   #$17
170C-    8D 00 B6       STA   $B600
170F-    A9 17          LDA   #$17
1711-    8D 01 B6       STA   $B601

; continue the boot
1714-    4C 00 B4       JMP   $B400
```

```
; copy $8000..$B3FF to lower memory so
; it survives a reboot
1717-   A0 00       LDY   #$00
1719-   B9 00 80    LDA   $8000,Y
171C-   99 00 20    STA   $2000,Y
171F-   C8          INY
1720-   D0 F7       BNE   $1719
1722-   EE 1B 17    INC   $171B
1725-   EE 1E 17    INC   $171E
1728-   AD 1B 17    LDA   $171B
172B-   C9 B4       CMP   #$B4
172D-   D0 EA       BNE   $1719

; normalize the environment and reboot
; to my work disk
172F-   AD 81 C0    LDA   $C081
1732-   AD 51 C0    LDA   $C051
1735-   4C 00 C5    JMP   $C500

*BSAVE TRACE3,A$1600,L$138
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE THE OBSERVATORY.OBJ 8000-B3FF,
 A$2000,L$3400
```

Now to save those chunks that were
loaded into the RAM banks...

```
]CALL -151

*1600<C600.C6FFM
```

```
; set up callback #1 and start the boot
16F8-   A9 05       LDA   #$05
16FA-   8D 37 09    STA   $0937
16FD-   A9 17       LDA   #$17
16FF-   8D 38 09    STA   $0938
1702-   4C 01 08    JMP   $0801

; callback #1 is here --
; set up callback #2 after we've loaded
; the chunk into $D000..$FFFF
1705-   A9 4C       LDA   #$4C
1707-   8D 10 B6    STA   $B610
170A-   A9 17       LDA   #$17
170C-   8D 11 B6    STA   $B611
170F-   A9 17       LDA   #$17
1711-   8D 12 B6    STA   $B612

; continue the boot
1714-   4C 00 B4    JMP   $B400

; callback #2 is here --
; copy code from RAM bank to graphics
; page in main memory so it survives a
; reboot
1717-   A0 00       LDY   #$00
1719-   B9 00 D0    LDA   $D000,Y
171C-   99 00 20    STA   $2000,Y
171F-   C8          INY
1720-   D0 F7       BNE   $1719
1722-   EE 1E 17    INC   $171E
1725-   EE 1B 17    INC   $171B
1728-   D0 EF       BNE   $1719

; reboot to my work disk
172A-   AD 81 C0    LDA   $C081
172D-   AD 51 C0    LDA   $C051
1730-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE4,A$1600,L$133
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE THE OBSERVATORY.OBJ D000-FFFF,
 A$2000,L$3000

]CALL -151

*1600<C600.C6FFM

; set up callback #1 and start the boot
16F8-    A9 05          LDA   #$05
16FA-    8D 37 09       STA   $0937
16FD-    A9 17          LDA   #$17
16FF-    8D 38 09       STA   $0938
1702-    4C 01 08       JMP   $0801

; callback #1 is here --
; set up callback #2 after we load into
; RAM bank 2
1705-    A9 4C          LDA   #$4C
1707-    8D 2E B6       STA   $B62E
170A-    A9 17          LDA   #$17
170C-    8D 2F B6       STA   $B62F
170F-    A9 17          LDA   #$17
1711-    8D 30 B6       STA   $B630

; continue the boot
1714-    4C 00 B4       JMP   $B400
```

```
; callback #2 is here --
; copy the code in RAM bank 2 down to
; main memory so it survives a reboot
1717-    A0 00          LDY   #$00
1719-    B9 00 D0       LDA   $D000,Y
171C-    99 00 20       STA   $2000,Y
171F-    C8             INY
1720-    D0 F7          BNE   $1719
1722-    EE 1B 17       INC   $171B
1725-    EE 1E 17       INC   $171E
1728-    AD 1B 17       LDA   $171B
172B-    C9 E0          CMP   #$E0
172D-    D0 EA          BNE   $1719

; reboot to my work disk
172F-    AD 81 C0       LDA   $C081
1732-    AD 51 C0       LDA   $C051
1735-    4C 00 C5       JMP   $C500

*BSAVE TRACE5,A$1600,L$138
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE THE OBSERVATORY.OBJ D000-DFFF,
 A$2000,L$1000
```

Chapter 4
Two's Company, Three's A Crowd

Continuing from $B62E...

```
; back to RAM bank 1
262E-    AD 8B C0     LDA   $C08B
2631-    AD 8B C0     LDA   $C08B

; seek to track 0
2634-    A9 00        LDA   #$00
2636-    85 03        STA   $03
2638-    20 B6 B6     JSR   $B6B6

; sector #$0D
263B-    A9 0D        LDA   #$0D
263D-    85 04        STA   $04
263F-    20 00 B8     JSR   $B800

*2800L
```

Holy crap, it's a third RWTS!

```
; match standard address prologue
; ("D5 AA 96")
2800-    A6 2B        LDX   $2B
2802-    BD 8C C0     LDA   $C08C,X
2805-    10 FB        BPL   $2802
2807-    C9 D5        CMP   #$D5
2809-    D0 F5        BNE   $2800
280B-    78           SEI
280C-    BD 8C C0     LDA   $C08C,X
280F-    10 FB        BPL   $280C
2811-    C9 AA        CMP   #$AA
2813-    D0 F2        BNE   $2807
2815-    78           SEI
2816-    BD 8C C0     LDA   $C08C,X
2819-    10 FB        BPL   $2816
281B-    C9 96        CMP   #$96
281D-    D0 E8        BNE   $2807
```

```
; parse standard address field, ignore
; everything except sector number
281F-   A0 03       LDY   #$03
2821-   BD 8C C0    LDA   $C08C,X
2824-   10 FB       BPL   $2821
2826-   38          SEC
2827-   2A          ROL
2828-   85 10       STA   $10
282A-   BD 8C C0    LDA   $C08C,X
282D-   10 FB       BPL   $282A
282F-   25 10       AND   $10
2831-   88          DEY
2832-   D0 ED       BNE   $2821

; loop until we find the sector we want
2834-   C5 04       CMP   $04
2836-   D0 C8       BNE   $2800
2838-   60          RTS
```

Continuing from $B642...

```
; seek to track 1
2642-   A9 02       LDA   #$02
2644-   85 03       STA   $03
2646-   20 B6 B6    JSR   $B6B6

; prepare to read to $0200+
2649-   A9 02       LDA   #$02
264B-   85 15       STA   $15
264D-   06 02       ASL   $02
264F-   06 03       ASL   $03
2651-   A5 03       LDA   $03
2653-   85 06       STA   $06

2655-   20 50 BA    JSR   $BA50
```

```
*2A50L

; match standard address prologue
; ("D5 AA 96")
2A50-   A6 2B       LDX   $2B
2A52-   BD 8C C0    LDA   $C08C,X
2A55-   10 FB       BPL   $2A52
2A57-   C9 D5       CMP   #$D5
2A59-   D0 F5       BNE   $2A50
2A5B-   78          SEI
2A5C-   BD 8C C0    LDA   $C08C,X
2A5F-   10 FB       BPL   $2A5C
2A61-   C9 AA       CMP   #$AA
2A63-   D0 F2       BNE   $2A57
2A65-   78          SEI
2A66-   BD 8C C0    LDA   $C08C,X
2A69-   10 FB       BPL   $2A66
2A6B-   C9 96       CMP   #$96
2A6D-   D0 E8       BNE   $2A57

; parse one 4-and-4 encoded value and
; match it against the sector we want
; (this is NOT a full address field)
2A6F-   78          SEI
2A70-   BD 8C C0    LDA   $C08C,X
2A73-   10 FB       BPL   $2A70
2A75-   38          SEC
2A76-   2A          ROL
2A77-   85 00       STA   $00
2A79-   BD 8C C0    LDA   $C08C,X
2A7C-   10 FB       BPL   $2A79
2A7E-   25 00       AND   $00
2A80-   85 00       STA   $00
2A82-   A5 06       LDA   $06
2A84-   C5 00       CMP   $00
2A86-   D0 C8       BNE   $2A50
```

```
; match standard data field prologue
; ("D5 AA AD")
2A88-   A6 2B       LDX   $2B
2A8A-   BD 8C C0    LDA   $C08C,X
2A8D-   10 FB       BPL   $2A8A
2A8F-   C9 D5       CMP   #$D5
2A91-   D0 F5       BNE   $2A88
2A93-   78          SEI
2A94-   BD 8C C0    LDA   $C08C,X
2A97-   10 FB       BPL   $2A94
2A99-   C9 AA       CMP   #$AA
2A9B-   D0 F2       BNE   $2A8F
2A9D-   78          SEI
2A9E-   BD 8C C0    LDA   $C08C,X
2AA1-   10 FB       BPL   $2A9E
2AA3-   C9 AD       CMP   #$AD
2AA5-   D0 E8       BNE   $2A8F

; now read 4-and-4 encoded sector data
2AA7-   A0 00       LDY   #$00
2AA9-   BD 8C C0    LDA   $C08C,X
2AAC-   10 FB       BPL   $2AA9
2AAE-   38          SEC
2AAF-   2A          ROL
2AB0-   85 00       STA   $00
2AB2-   BD 8C C0    LDA   $C08C,X
2AB5-   10 FB       BPL   $2AB2
2AB7-   25 00       AND   $00
2AB9-   91 14       STA   ($14),Y
2ABB-   C8          INY
2ABC-   D0 EB       BNE   $2AA9

; no epilogue, no checksum
2ABE-   60          RTS
```

OK, that was weird.

```
Continuing from $B658...

; increment phase
2658-   E6 03       INC   $03
265A-   20 BB B9    JSR   $B9BB

*29BBL

; advance drive arm in single phase
; (half-track) increments
29BB-   A5 02       LDA   $02
29BD-   C5 03       CMP   $03
29BF-   D0 01       BNE   $29C2
29C1-   60          RTS
29C2-   4A          LSR
29C3-   90 14       BCC   $29D9
29C5-   A5 03       LDA   $03
29C7-   85 02       STA   $02
29C9-   4A          LSR
29CA-   48          PHA
29CB-   38          SEC
29CC-   20 D9 B6    JSR   $B6D9
29CF-   A9 46       LDA   #$46
29D1-   20 E3 B6    JSR   $B6E3
29D4-   68          PLA
29D5-   18          CLC
29D6-   4C D9 B6    JMP   $B6D9
29D9-   A5 03       LDA   $03
29DB-   85 02       STA   $02
29DD-   4A          LSR
29DE-   48          PHA
29DF-   38          SEC
29E0-   20 D9 B6    JSR   $B6D9
29E3-   68          PLA
29E4-   18          CLC
29E5-   69 01       ADC   #$01
29E7-   48          PHA
29E8-   38          SEC
29E9-   20 D9 B6    JSR   $B6D9
[...]
```

```
29EC-   A9 46         LDA   #$46
29EE-   20 E3 B6      JSR   $B6E3
29F1-   68            PLA
29F2-   48            PHA
29F3-   18            CLC
29F4-   20 D9 B6      JSR   $B6D9
29F7-   68            PLA
29F8-   38            SEC
29F9-   E9 01         SBC   #$01
29FB-   18            CLC
29FC-   4C D9 B6      JMP   $B6D9
```

Cool, so we're reading from consecutive half-tracks. The data has to be laid out in a spiral on the physical disk, otherwise there would be too much cross-track interference. (That's why "whole" tracks are the distance away from each other that they are.) That's a neat trick.

Continuing from $B65D...

```
; Increment address until $2000.
265D-   E6 15         INC   $15
265F-   A5 15         LDA   $15
```

Astute readers will notice that we are only reading one sector from each track (in 4-and-4 encoding, no less) before moving on to the next consecutive half track.

Less astute readers should not feel bad about themselves if they did not notice that. This is really, really weird. Like, unique across every disk I've ever examined. I've literally never seen anything like this.

```
; loop until we've filled $0200..$1FFF
; in main memory
2661-   C9 20          CMP   #$20
2663-   D0 EC          BNE   $2651

; position on next whole track
2665-   A5 02          LDA   $02
2667-   4A             LSR
2668-   90 05          BCC   $266F
266A-   E6 03          INC   $03
266C-   20 BB B9       JSR   $B9BB
266F-   46 02          LSR   $02
2671-   46 03          LSR   $03

; jump to next stage
2673-   4C 03 B4       JMP   $B403
```

And that's where I can interrupt the boot again.

```
*1600<C600.C6FFM

; set up callback #1 after boot0 and
; start the boot
16F8-   A9 05          LDA   #$05
16FA-   8D 37 09       STA   $0937
16FD-   A9 17          LDA   #$17
16FF-   8D 38 09       STA   $0938
1702-   4C 01 08       JMP   $0801
```

```
; callback #1 is here --
; set up callback #2 and move ourselves
; to higher memory so we don't get
; overwritten by the data being read
; from disk
1705-   A9 1D       LDA   #$1D
1707-   8D 74 B6    STA   $B674
170A-   A9 BE       LDA   #$BE
170C-   8D 75 B6    STA   $B675
170F-   A0 00       LDY   #$00
1711-   B9 00 17    LDA   $1700,Y
1714-   99 00 BE    STA   $BE00,Y
1717-   C8          INY
1718-   D0 F7       BNE   $1711

; continue the boot
171A-   4C 00 B4    JMP   $B400

; callback #2 is here --
; copy $0200..$1FFF to $2000+ so it
; survives a reboot
171D-   A0 00       LDY   #$00
171F-   B9 00 02    LDA   $0200,Y
1722-   99 00 20    STA   $2000,Y
1725-   C8          INY
1726-   D0 F7       BNE   $171F
1728-   EE 21 BE    INC   $BE21
172B-   EE 24 BE    INC   $BE24
172E-   AD 21 BE    LDA   $BE21
1731-   C9 20       CMP   #$20
1733-   D0 EA       BNE   $171F

; restore environment and reboot to my
; work disk
1735-   AD 81 C0    LDA   $C081
1738-   AD 51 C0    LDA   $C051
173B-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE6,A$1600,L$13E
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE THE OBSERVATORY.OBJ 0200-1FFF,
 A$2000,L$1E00
```

To recap:

RWTS #1 (used to read track $00) was relatively normal -- no epilogue check, but otherwise standard.

RWTS #2 (used to read tracks $12-$20) had modified prologues but no epilogue.

RWTS #3 (used to read tracks $01-$10) initially looked like it used standard prologues and no epilogue, but then it pivoted into reading 4-and-4 encoded sectors from consecutive half-tracks.

And we're not done yet.

Chapter 5
Please Not Another RWT---okay

Continuing from $B403...

```
*2403L

; seek to track $11
2403-   A9 22       LDA   #$22
2405-   85 03       STA   $03
2407-   20 B6 B6    JSR   $B6B6

; sector 0
240A-   A9 00       LDA   #$00
240C-   85 04       STA   $04

; read a sector
240E-   A5 2B       LDA   $2B
2410-   85 E9       STA   $E9
2412-   20 00 B9    JSR   $B900

*2900L
```

RWTS #4 (so not kidding):

```
; zp$E9 is the boot slot (x16) (set at
; $B410, just before this call)
2900-   A6 E9       LDX   $E9
```

```
; standard "D5 AA 96" address prologue
2902-   BD 8C C0    LDA   $C08C,X
2905-   10 FB       BPL   $2902
2907-   C9 D5       CMP   #$D5
2909-   D0 F5       BNE   $2900
290B-   78          SEI
290C-   BD 8C C0    LDA   $C08C,X
290F-   10 FB       BPL   $290C
2911-   C9 AA       CMP   #$AA
2913-   D0 F2       BNE   $2907
2915-   78          SEI
2916-   BD 8C C0    LDA   $C08C,X
2919-   10 FB       BPL   $2916
291B-   C9 96       CMP   #$96
291D-   D0 E8       BNE   $2907

; parse a normal address field, but
; ignore everything but the sector
291F-   A0 03       LDY   #$03
2921-   BD 8C C0    LDA   $C08C,X
2924-   10 FB       BPL   $2921
2926-   38          SEC
2927-   2A          ROL
2928-   85 10       STA   $10
292A-   BD 8C C0    LDA   $C08C,X
292D-   10 FB       BPL   $292A
292F-   25 10       AND   $10
2931-   85 10       STA   $10
2933-   88          DEY
2934-   D0 EB       BNE   $2921
2936-   A4 04       LDY   $04
2938-   B9 EF B6    LDA   $B6EF,Y

; loop until we find the sector we want
293B-   C5 10       CMP   $10
293D-   D0 C1       BNE   $2900
```

```
; standard "D5 AA AD" data prologue
293F-   A0 20       LDY   #$20
2941-   88          DEY
2942-   D0 03       BNE   $2947
2944-   4C 00 B9    JMP   $B900
2947-   BD 8C C0    LDA   $C08C,X
294A-   10 FB       BPL   $2947
294C-   49 D5       EOR   #$D5
294E-   D0 F1       BNE   $2941
2950-   78          SEI
2951-   BD 8C C0    LDA   $C08C,X
2954-   10 FB       BPL   $2951
2956-   C9 AA       CMP   #$AA
2958-   D0 F2       BNE   $294C
295A-   A0 56       LDY   #$56
295C-   BD 8C C0    LDA   $C08C,X
295F-   10 FB       BPL   $295C
2961-   C9 AD       CMP   #$AD
2963-   D0 E7       BNE   $294C

; 6-and-2 decoding
2965-   A9 00       LDA   #$00
2967-   88          DEY
2968-   84 10       STY   $10
296A-   BC 8C C0    LDY   $C08C,X
296D-   10 FB       BPL   $296A
296F-   59 C1 B4    EOR   $B4C1,Y
2972-   A4 10       LDY   $10
2974-   99 2C 00    STA   $002C,Y
2977-   D0 EE       BNE   $2967
2979-   84 10       STY   $10
297B-   BC 8C C0    LDY   $C08C,X
297E-   10 FB       BPL   $297B
2980-   59 C1 B4    EOR   $B4C1,Y
2983-   A4 10       LDY   $10
```

```
; The target address is never modified
; by the caller. This entire RWTS is
; hard-coded to read a single sector
; from T11,S00 into $BF00.
2985-   99 00 BF    STA   $BF00,Y
2988-   C8          INY
2989-   D0 EE       BNE   $2979
298B-   BC 8C C0    LDY   $C08C,X
298E-   10 FB       BPL   $298B
2990-   D9 C1 B4    CMP   $B4C1,Y
2993-   F0 03       BEQ   $2998
2995-   4C 00 B9    JMP   $B900
2998-   A0 00       LDY   #$00
299A-   A2 56       LDX   #$56
299C-   CA          DEX
299D-   30 FB       BMI   $299A
299F-   B9 00 BF    LDA   $BF00,Y
29A2-   56 2C       LSR   $2C,X
29A4-   2A          ROL
29A5-   56 2C       LSR   $2C,X
29A7-   2A          ROL
29A8-   99 00 BF    STA   $BF00,Y
29AB-   C8          INY
29AC-   C0 00       CPY   #$00
29AE-   D0 EC       BNE   $299C

; no epilogue, no checksum
29B0-   60          RTS
```

Continuing from $B415...

```
; don't know what this does yet
2415-   20 BA 1E    JSR   $1EBA
```

Time to interrupt the boot. Again.

```
*1600<C600.C6FFM
```

```
; set up callback #1 and start the boot
16F8-   A9 05       LDA   #$05
16FA-   8D 37 09    STA   $0937
16FD-   A9 17       LDA   #$17
16FF-   8D 38 09    STA   $0938
1702-   4C 01 08    JMP   $0801

; callback #1 is here --
; set up callback #2, move ourselves
; out of the way, and continue the boot
1705-   A9 1D       LDA   #$1D
1707-   8D 16 B4    STA   $B416
170A-   A9 BE       LDA   #$BE
170C-   8D 17 B4    STA   $B417
170F-   A0 00       LDY   #$00
1711-   B9 00 17    LDA   $1700,Y
1714-   99 00 BE    STA   $BE00,Y
1717-   C8          INY
1718-   D0 F7       BNE   $1711
171A-   4C 00 B4    JMP   $B400

; callback #2 is here --
; copy $BF00 page to lower memory so it
; survives a reboot
171D-   A0 00       LDY   #$00
171F-   B9 00 BF    LDA   $BF00,Y
1722-   99 00 20    STA   $2000,Y
1725-   C8          INY
1726-   D0 F7       BNE   $171F

; restore environment and reboot
1728-   AD 81 C0    LDA   $C081
172B-   AD 51 C0    LDA   $C051
172E-   4C 00 C5    JMP   $C500
```

```
*BSAVE TRACE7,A$1600,L$131
*1600G
...reboots slot 6...
...reboots slot 5...

]BSAVE THE OBSERVATORY.OBJ BF00-BFFF,
 A$2000,L$100

]CATALOG

DISK VOLUME 254

 A 002 Z
 B 004 BOOT0
 B 003 TRACE1
 B 009 BOOT1
 B 003 TRACE2
 B 098 THE OBSERVATORY.OBJ 2000-7FFF
 B 003 TRACE3
 B 054 THE OBSERVATORY.OBJ 8000-B3FF
 B 003 TRACE4
 B 050 THE OBSERVATORY.OBJ D000-FFFF
 B 003 TRACE5
 B 018 THE OBSERVATORY.OBJ D000-DFFF
 B 003 TRACE6
 B 032 THE OBSERVATORY.OBJ 0200-1FFF
 B 003 TRACE7
 B 003 THE OBSERVATORY.OBJ BF00-BFFF
```

As you can see, this disk fills pretty much all the memory you can fill on a 64K machine. We could have saved some code by, you know, using any one of the four disk read routines more than once, but I'm not bitter.

Chapter 6
Oh What Fresh Hell Is This

```
]BLOAD THE OBSERVATORY.OBJ 0200-1FFF,
 A$2200
]CALL -151
```

Again, I can't load this entire chunk in its actual place (since it includes the input buffer and text page), so absolute addresses will be +$2000.

Continuing at $1EBA...

```
*3EBAL

3EBA-   A5 ED          LDA   $ED
3EBC-   48             PHA
3EBD-   A9 22          LDA   #$22
3EBF-   85 5F          STA   $5F
3EC1-   4A             LSR
3EC2-   4A             LSR
3EC3-   85 ED          STA   $ED
3EC5-   C6 ED          DEC   $ED
3EC7-   30 53          BMI   $3F1C
3EC9-   20 69 1F       JSR   $1F69

*3F69L
```

Oh look, it's yet another disk read routine.

```
; boot slot (x16)
3F69-   A6 E9          LDX   $E9
```

```
; match "D5 AA 96" prologue
3F6B-   BD 8C C0      LDA   $C08C,X
3F6E-   10 FB         BPL   $3F6B
3F70-   C9 D5         CMP   #$D5
3F72-   D0 F5         BNE   $3F69
3F74-   78            SEI
3F75-   BD 8C C0      LDA   $C08C,X
3F78-   10 FB         BPL   $3F75
3F7A-   C9 AA         CMP   #$AA
3F7C-   D0 F2         BNE   $3F70
3F7E-   78            SEI
3F7F-   BD 8C C0      LDA   $C08C,X
3F82-   10 FB         BPL   $3F7F
3F84-   C9 96         CMP   #$96
3F86-   D0 E8         BNE   $3F70

; parse standard address field, ignore
; everything except sector number
3F88-   A0 03         LDY   #$03
3F8A-   BD 8C C0      LDA   $C08C,X
3F8D-   10 FB         BPL   $3F8A
3F8F-   38            SEC
3F90-   2A            ROL
3F91-   85 EB         STA   $EB
3F93-   BD 8C C0      LDA   $C08C,X
3F96-   10 FB         BPL   $3F93
3F98-   25 EB         AND   $EB
3F9A-   88            DEY
3F9B-   D0 ED         BNE   $3F8A

; loop until we find sector $00
3F9D-   C9 00         CMP   #$00
3F9F-   D0 C8         BNE   $3F69
```

```
; seek to track $10
3FA1-   A9 20       LDA   #$20
3FA3-   85 60       STA   $60
3FA5-   A5 5F       LDA   $5F
3FA7-   C5 60       CMP   $60
3FA9-   F0 1C       BEQ   $3FC7
3FAB-   90 04       BCC   $3FB1
3FAD-   C6 5F       DEC   $5F
3FAF-   C6 5F       DEC   $5F
3FB1-   E6 5F       INC   $5F
3FB3-   A5 5F       LDA   $5F
3FB5-   48          PHA

; hit stepper motor (not shown, but
; keep in mind this is NOT calling a
; ROM routine -- we filled both RAM
; banks with custom code and it's still
; active)
3FB6-   38          SEC
3FB7-   20 ED FF    JSR   $FFED

; wait (not shown)
3FBA-   A9 56       LDA   #$56
3FBC-   20 65 78    JSR   $7865
3FBF-   68          PLA

; hit other stepper motor (not shown)
3FC0-   18          CLC
3FC1-   20 ED FF    JSR   $FFED

; jump back to move more, if necessary
3FC4-   4C A5 1F    JMP   $1FA5
3FC7-   60          RTS
```

That's it. We never read anything. We just position the drive. Highly suspicious, no?

Continuing from $1ECC...

```
; manual stack push (hmm)
3ECC-   A9 00         LDA   #$00
3ECE-   48            PHA

; zp$5F and zp$60 had the current phase
; (set and used during the drive arm
; move routine at $1FA1), but now we're
; multiplying them by 2 for reasons
; unknown
3ECF-   06 5F         ASL   $5F
3ED1-   06 60         ASL   $60

; and copying to zp$EF
3ED3-   A5 60         LDA   $60
3ED5-   85 EF         STA   $EF

; still the boot slot (x16)
3ED7-   A6 E9         LDX   $E9
3ED9-   20 71 78      JSR   $7871
```

I don't know what that does yet, but it's taking the boot slot (x16) in the X register, so I'm guessing it's disk-related. (I'm beginning to wonder if this program does anything other than fiddle with the disk.)

Oh hey, I have that chunk on my work disk. Let's go code spelunking.

```
*BLOAD THE OBSERVATORY.OBJ 2000-7FFF,
 A$2000
```

(Absolute addresses are correct again.)

```
*7871L
```

Look ma, another disk read routine.
(I've honestly lost count by now.)

```
; match "D5 AA 96" prologue
7871-   BD 8C C0     LDA   $C08C,X
7874-   10 FB        BPL   $7871
7876-   C9 D5        CMP   #$D5
7878-   D0 F7        BNE   $7871
787A-   BD 8C C0     LDA   $C08C,X
787D-   10 FB        BPL   $787A
787F-   C9 AA        CMP   #$AA
7881-   D0 F3        BNE   $7876
7883-   BD 8C C0     LDA   $C08C,X
7886-   10 FB        BPL   $7883
7888-   C9 96        CMP   #$96
788A-   D0 EA        BNE   $7876

; get a single 4-and-4 encoded value
788C-   BD 8C C0     LDA   $C08C,X
788F-   10 FB        BPL   $788C
7891-   38           SEC
7892-   2A           ROL
7893-   85 F0        STA   $F0
7895-   BD 8C C0     LDA   $C08C,X
7898-   10 FB        BPL   $7895
789A-   25 F0        AND   $F0

; compare it to the phase that we set
; at $1ED5
789C-   C5 EF        CMP   $EF

; loop until they match
789E-   D0 D1        BNE   $7871
```

```
; overwriting zp$EF and zp$F0 (!) to
; hold an address
78A0-   A9 96       LDA   #$96
78A2-   85 EF       STA   $EF
78A4-   A9 6A       LDA   #$6A
78A6-   85 F0       STA   $F0
78A8-   A0 07       LDY   #$07

; take a single uninitialized value
78AA-   A5 8B       LDA   $8B

; and store it at that address
78AC-   91 EF       STA   ($EF),Y
```

(That was complete misdirection. The initial value was never set, and the address is overwritten later anyway.)

```
; match "D5 AA AD" prologue
78AE-   A0 00       LDY   #$00
78B0-   BD 8C C0    LDA   $C08C,X
78B3-   10 FB       BPL   $78B0
78B5-   C9 D5       CMP   #$D5
78B7-   D0 F7       BNE   $78B0
78B9-   BD 8C C0    LDA   $C08C,X
78BC-   10 FB       BPL   $78B9
78BE-   C9 AA       CMP   #$AA
78C0-   D0 F3       BNE   $78B5
78C2-   BD 8C C0    LDA   $C08C,X
78C5-   10 FB       BPL   $78C2
78C7-   C9 AD       CMP   #$AD
78C9-   D0 EA       BNE   $78B5
```

```
; get another 4-and-4 encoded value
78CB-   BD 8C C0    LDA   $C08C,X
78CE-   10 FB       BPL   $78CB
78D0-   38          SEC
78D1-   2A          ROL
78D2-   85 EF       STA   $EF
78D4-   BD 8C C0    LDA   $C08C,X
78D7-   10 FB       BPL   $78D4
78D9-   25 EF       AND   $EF

; store it in zp$EF
78DB-   85 EF       STA   $EF

; 256 times (overwriting zp$EF each
; time, so I guess only the last value
; is actually important)
78DD-   88          DEY
78DE-   D0 EB       BNE   $78CB
78E0-   60          RTS
```

Continuing from $1EDC...

```
*BLOAD THE OBSERVATORY.OBJ 0200-1FFF,
 A$2200
```

(Absolute addresses are +$2000.)

```
; using the value we pushed to the
; stack (at $1ECC) as the index...
3EDC-   68          PLA
3EDD-   AA          TAX

; put the magic byte into an array (the
; last of the 256 bytes we decoded)
3EDE-   A5 EF       LDA   $EF
3EE0-   95 6F       STA   $6F,X
```

```
; increment and prepare to do it all
; over again
3EE2-   E8              INX
3EE3-   8A              TXA
3EE4-   48              PHA

; if we've stored a total of $1C magic
; bytes, we're done (whew)
3EE5-   C9 1C           CMP   #$1C
3EE7-   F0 08           BEQ   $3EF1

; otherwise move the drive arm to the
; previous half-track (not shown) and
; jump back to do it again
3EE9-   C6 60           DEC   $60
3EEB-   20 FE 18        JSR   $18FE
3EEE-   4C D3 1E        JMP   $1ED3
```

Execution continues here (from $1EE7) once we've plucked one magic byte from each of $1C consecutive half-tracks. Seriously.

```
; now seek to nearest whole track
3EF1-   A5 5F           LDA   $5F
3EF3-   4A              LSR
3EF4-   90 05           BCC   $3EFB
3EF6-   C6 60           DEC   $60
3EF8-   20 FE 18        JSR   $18FE
3EFB-   46 5F           LSR   $5F
3EFD-   68              PLA

; seek to track $11 again
3EFE-   A9 22           LDA   #$22
3F00-   85 60           STA   $60
3F02-   20 A5 1F        JSR   $1FA5
```

```
; now verify the magic byte array
3F05-   A0 00        LDY   #$00
3F07-   A2 07        LDX   #$07
3F09-   B9 6F 00     LDA   $006F,Y
3F0C-   DD EC 15     CMP   $15EC,X

; if bytes don't match, branch back to
; $1EC5, which decrements zp$ED and
; tries again, but eventually branches
; forward to $1F1C regardless
3F0F-   D0 B4        BNE   $3EC5
3F11-   CA           DEX
3F12-   D0 02        BNE   $3F16
3F14-   A2 07        LDX   #$07
3F16-   C8           INY
3F17-   C0 1C        CPY   #$1C
3F19-   D0 EE        BNE   $3F09
3F1B-   68           PLA

; execution always continues here
; (success falls through, failure
; branches to here from $1EC7)
3F1C-   85 ED        STA   $ED

; turn off drive (finally)
3F1E-   A6 E9        LDX   $E9
3F20-   BD 88 C0     LDA   $C088,X
3F23-   60           RTS
```

We will reach the end of this routine even if the protection fails, but there is a side effect later that relies on the magic byte array being correct.

```
*BLOAD THE OBSERVATORY.OBJ 2000-7FFF,
 A$2000
```

(Absolute addresses are correct again.)

```
*5CEDL

; index into magic byte array
5C3D-   A0 04       LDY   #$04

; initial value
5C3F-   A9 80       LDA   #$80
5C41-   85 F2       STA   $F2

; verify one value from the magic byte
; array
5C43-   B9 79 00    LDA   $0079,Y
5C46-   C9 34       CMP   #$34
5C48-   D0 02       BNE   $5C4C

; if verification passes, zp$F2 will
; end up with $20, otherwise $40!
5C4A-   46 F2       LSR   $F2
5C4C-   46 F2       LSR   $F2
```

And later, in a galaxy far far away...

```
E2D9-   A5 F1       LDA   $F1
E2DB-   85 ED       STA   $ED
E2DD-   A5 F2       LDA   $F2
E2DF-   85 EE       STA   $EE
...
E308-   91 ED       STA   ($ED), Y
```

...we will end up writing to the wrong memory location and damaging the code if the magic byte verification failed.

Chapter 7
Nuke It From Orbit,
It's The Only Way To Be Sure

At this point, I'm strongly tempted to burn this entire disk to the ground and rebuild it from scratch with a custom bootloader.

[takes deep breath]

[takes another]

[falls asleep]

[wakes up with a fresh perspective]

OK, here's what we have:

- Altered prologues and epilogues (T00, T12-T20) - easy enough to normalize.

- Half-track stepping, one sector per half-track, 4-and-4 encoding - also easy to normalize. I've captured all the data that was read; I can put it back in any format I choose. There's plenty of room.

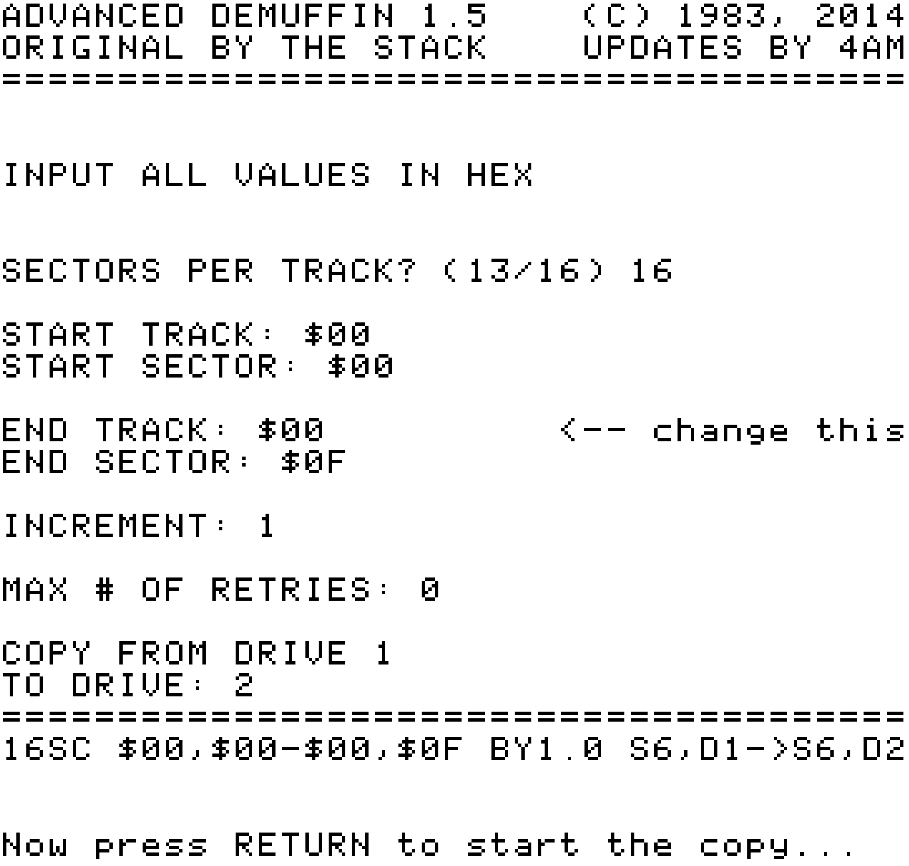
- Magic bytes, one per half-track - not required at all. The correct bytes are in memory; we can just copy them from one memory address to another.

- Magic byte verification - leave it intact. It'll pass once we copy the magic byte array into position.

This is doable. Let's do this.

First, I can use Advanced Demuffin to
convert the relatively normal tracks.
Starting with the bog standard DOS 3.3
on my work disk, I'll make one change
to ignore epilogue bytes.

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

]PR#5
...
]CALL -151

*3800<B800.BFFFM   ; copy DOS
*3942:18           ; ignore epilogues
*BSAVE RWTS 0,A$3800,L$800
*BRUN ADVANCED DEMUFFIN

]BRUN ADVANCED DEMUFFIN 1.5

[press "5" to switch to slot 5]

[press "R" to load a new RWTS module]
  --> At $B8, load "RWTS 0"

[press "6" to switch to slot 6]

[press "C" to convert disk]

--> CHANGE DEFAULT VALUES? Y
```

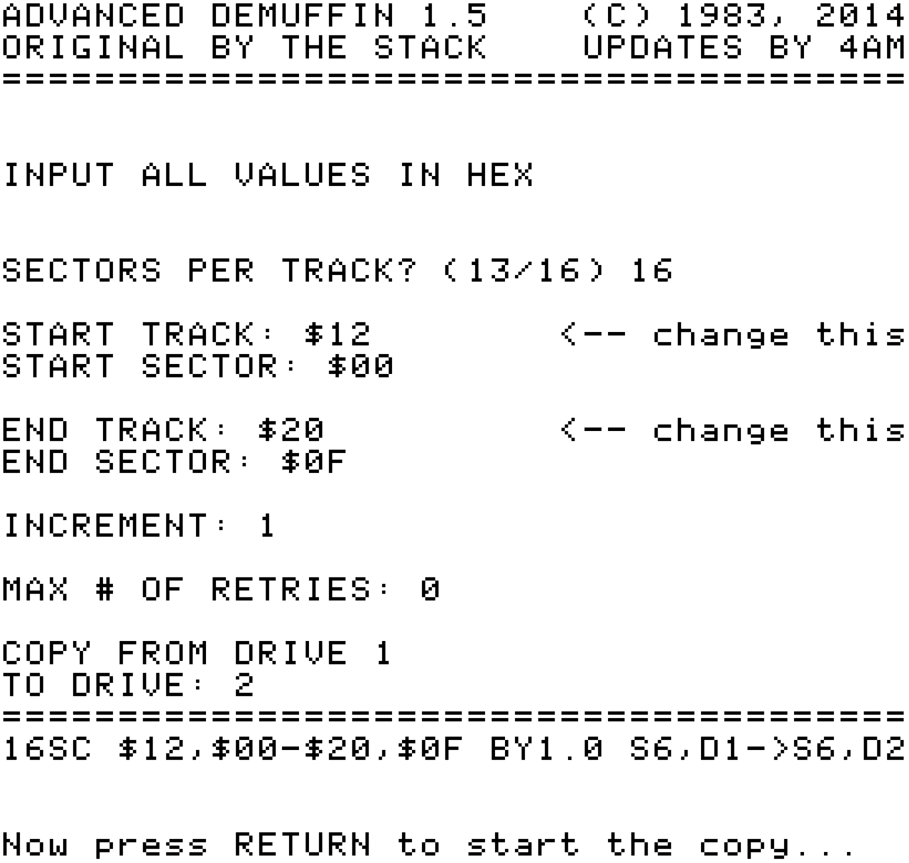


```
ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======================================


INPUT ALL VALUES IN HEX


SECTORS PER TRACK? (13/16) 16

START TRACK: $00
START SECTOR: $00

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

INCREMENT: 1

MAX # OF RETRIES: 0

COPY FROM DRIVE 1
TO DRIVE: 2
=======================================
16SC $00,$00-$00,$0F BY1.0 S6,D1->S6,D2


Now press RETURN to start the copy...
```

```
--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 $00,$00-$00,$0F BY1.0 S6,D1->S6,D2

--^--
```

One track down, 34 to go!

```
]PR#5
...
]CALL -151

*3800<B800.BFFFM  ; copy DOS
*3942:18          ; was "38"
*3955:AA          ; was "D5"
*395F:D5          ; was "AA"
*38E7:AA          ; was "D5"
*38F1:D5          ; was "AA"

*BSAVE RWTS 12+,A$3800,L$800
*BRUN ADVANCED DEMUFFIN 1.5

[press "5" to switch to slot 5]

[press "R" to load a new RWTS module]
  --> At $B8, load "RWTS 12+"

[press "6" to switch to slot 6]

[press "C" to convert disk]

--> CHANGE DEFAULT VALUES? Y
```



```
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: $12        <-- change this
START SECTOR: $00

END TRACK: $20          <-- change this
END SECTOR: $0F

INCREMENT: 1

MAX # OF RETRIES: 0

COPY FROM DRIVE 1
TO DRIVE: 2
=======================================
16SC $12,$00-$20,$0F BY1.0 S6,D1->S6,D2


Now press RETURN to start the copy...
```

```
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 $12,$00-$20,$0F BY1.0 S6,D1->S6,D2
```

--^--

The RWTS that reads tracks $12-$20 is on track $00, so let's patch that now. Both the address and data prologue swap the order of the first two nibbles:

```
T00,S02,$41 change "AA" to "D5"
T00,S02,$4B change "D5" to "AA"
T00,S02,$7E change "AA" to "D5"
T00,S02,$88 change "D5" to "AA"
```

For T11,S00, I just used my Disk Fixer sector editor to copy the sector. The RWTS that reads it is liberal enough to read a standard format (since it just ignores the epilogues), so no patches are required.

Fun(*) fact: T11,S00 is the only sector on the disk that is read/write. The program prompts you for coordinates (latitude and longitude) and writes them back to disk on T11,S00. The write routine uses standard prologue and epilogue bytes, so no patches are required.

Next up: the weird 4-and-4 encoded data on tracks $01-$10. The code that reads the data and moves the drive arm is on T00,S04, starting at offset $3B. (It's loaded into $B63B in memory; I listed it earlier at $263B.)

Instead of 4-and-4 encoding, I'll use a standard (6-and-2) encoding; instead of one sector per consecutive half-track, I'll read two sectors from each whole track. Which is still weird, but it'll be minimally invasive, given the code we already have.

(*) not guaranteed, actual fun may vary

So this is the new disk layout:

```
T01,S01 -> $0200
T01,S00 -> $0300
T02,S01 -> $0400
T02,S00 -> $0500
.
. &c.
.
```

And this is the new code (on T00,S04):

```
; start on sector $01
263B-   A9 01       LDA   #$01
263D-   85 04       STA   $04
263F-   20 00 B8    JSR   $B800

; seek to track $01 and prepare to read
; into $0200+ (like the original)
2642-   A9 02       LDA   #$02
2644-   85 03       STA   $03
2646-   A9 02       LDA   #$02
2648-   85 15       STA   $15
264A-   20 B6 B6    JSR   $B6B6

; read sector using 6-and-2 encoding,
; using one of the many RWTS routines
; available in memory (I'm not bitter!)
264D-   20 39 B8    JSR   $B839

; decrement sector (we'll read two
; sectors per whole track, then skip to
; the next whole track)
2650-   C6 04       DEC   $04
2652-   10 09       BPL   $265D

; sector $01 again
2654-   A9 01       LDA   #$01
2656-   85 04       STA   $04
```

```
; increment phase twice to advance to
; the next whole track
2658-   E6 03         INC   $03
265A-   E6 03         INC   $03

; a spare byte (!)
265C-   EA            NOP

; increment address until $2000
265D-   E6 15         INC   $15
265F-   A5 15         LDA   $15

; loop until we've filled $0200..$1FFF
; in main memory
2661-   C9 20         CMP   #$20
2663-   D0 E5         BNE   $264A

; prevent seek from wandering away
2665-   A5 02         LDA   $02
2667-   4A            LSR
2668-   90 09         BCC   $2673
```

(I could lie and say that I automated writing out this data in the right pattern across tracks $01-$10, but I didn't. I just used a sector editor to write each page where this code would look for it. It's not that much data.)

That's covers all the disk reading. For the "magic byte" verification, I can change the comparison loop into a copy loop. (This is now on T06,S00.)

```
; copy the magic byte array into the
; proper place on zero page (instead of
; verifying)
1F05-   A0 00         LDY   #$00
1F07-   A2 07         LDX   #$07
1F09-   BD EC 15      LDA   $15EC,X    <--
1F0C-   99 6F 00      STA   $006F,Y    <--
1F0F-   D0 00         BNE   $1F11
```

Finally, I made a one-byte patch on T0C,S00 to disable the drive stepping. Now it'll actually read $1C bytes from disk, but all from the same track (and then ignore them and copy the correct bytes from memory). Which would make no sense if we were building this program from scratch, but we're not; we're patching a hostile codebase in a minimally invasive way.

Anyway, this code (to check whether we need to move the drive arm):

```
18FE-   A5 5F         LDA   $5F
1900-   C5 60         CMP   $60
1902-   D0 01         BNE   $1905      <--
1904-   60            RTS
```

becomes this:

```
18FE-   A5 5F         LDA   $5F
1900-   C5 60         CMP   $60
1902-   D0 00         BNE   $1904      <--
1904-   60            RTS
```

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

Infernum post nos.

Changelog

2015-10-24

- corrected bug in original software
  that could write to the wrong track
  when saving state

2015-10-22

- initial release