Peeks, Pokes and Pirates

Disk Layout

A 5.25-inch floppy disk has 35 tracks, numbered $00 to $22 (hex). The format of each track is disk-specific. Most disks split each track into 16 “sectors,” but older disks use 13 sectors per track. Some games use 12, 11, or 10. Newer games can squeeze up to 18 sectors in a single track! Just figuring out how data is stored on disk can be a challenge.

Disk Control

Disk control is through “soft-switches,” not function calls:

$C080-7,X move drive arm (phase 0 off/on, phase 1 off/on... until 3)
$C088,X turn off drive motor
$C089,X turn on drive motor
$C08C,X read raw nibble from disk
$C08D,X reset data latch (used in desync nibble checks)
(X = boot slot x $10)

Disk Boot

A disk is booted in stages, starting from ROM:

$C600 ROM finds track 0 and reads sector 0 into **$800**
$0801 RAM re-uses part of **$C600** code to read more sectors (usually into **$B600+**)
$B700 RAM uses RWTS at **$B800+** to read rest of disk

tip: **$C600** is read-only. But the code there is surprisingly flexible; It will run at **$9600**, **$8600**, even **$1600**. If you copy it to RAM, you can insert your own code before jumping to **$0801**.

Prologue And Epilogue

Many protected disks start with DOS 3.3 and change prologue/epilogue values. Here's where to look:

	0x	read	write
	D5	$B955	$BC7A
prologue	AA	$B95F	$BC7F
/	96	$B96A	$BC84
ADDRESS			
\	DE	$B991	$BCAE
epilogue	AA	$B99B	$BCB3
	EB	-----	$BCB8

	0x	read	write
	D5	$B8E7	$B853
prologue	AA	$B8F1	$B858
/	AD	$B8FC	$B85D
DATA			
\	DE	$B935	$B89E
epilogue	AA	$B93F	$B8A3
	EB	-----	$B8A8

Know Your Tools

Every pirate needs:

- a NIBBLE EDITOR for inspecting raw nibbles and determining disk structure (Copy II Plus, Nibbles Away, Locksmith)
- a SECTOR EDITOR for searching, disassembling, patching sector-based disks (Disk Fixer, Block Warden, Copy II Plus)
- a DEMUFFIN TOOL for converting disks to a standard format (Advanced Demuffin, Super Demuffin)
- a FAST DISK COPIER for backing up your work-in-progress! (Locksmith Fast Disk Backup, FASTDSK, Disk Muncher)

Common Code Obfuscation

Apples have a built-in “monitor” and naive disassembler. Confusing this disassembler is not hard!

Self-modifying code

```
BB03- 4E 06 BB   LSR  $BB06     <- modifies the next instruction
BB06- 71 6E      ADC  ($6E),Y
BB08- 0A         ASL
BB09- BB         ???
```

By the time **$BB06** is executed...

```
BB03- 4E 06 BB   LSR  $BB06
BB06- 38         SEC            <- the code has changed!
BB07- 6E 0A BB   ROR  $BB0A
```

Branches into the middle of an instruction

```
AEB5- A0 02      LDY  #$02
AEB7- 8C EC B7   STY  $B7EC
AEBA- 88         DEY
AEBB- 8C F4 B7   STY  $B7F4
AEBE- 88         DEY
AEBF- F0 01      BEQ  $AEC2     <- Y = 0 here, so this branches...
AEC1- 6C 8C F0   JMP  ($F08C)
AEC4- B7         ???
AEC5- 8C EB B7   STY  $B7EB

AEBF- F0 01      BEQ  $AEC2
AEC1- 6C
AEC2- 8C F0 B7   STY  $B7F0     <- ...to here (JMP is never executed)
AEC5- 8C EB B7   STY  $B7EB
```

Manual stack manipulation

```
0800- A9 51      LDA  #$0F      <- push address to stack ($0FFF)
0802- 48         PHA
0803- A9 8E      LDA  #$FF
0805- 48         PHA
0806- 20 5D 6A   JSR  $080C     <- call subroutine (also pushes to stack)
0809- 4C 00 08   JMP  $0800
080C- 68         PLA            <- remove address pushed by JSR
080D- 68         PLA
080E- 60         RTS            <- "return" to $0FFF+1 = $1000
```

JMP at **$0809** is never executed! Execution continues at **$1000**.

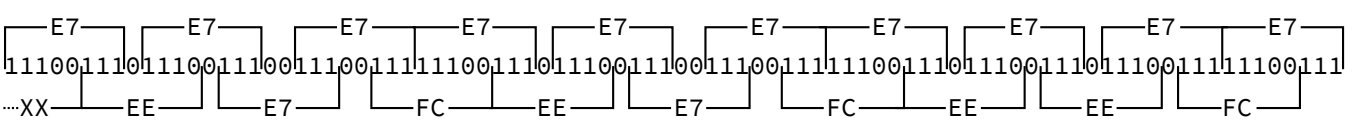
Undocumented opcodes

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

$74 is an undocumented 6502 opcode that does nothing, but takes a one-byte operand. Here is what actually executes:

```
0801- 74 4C      DOP  $4C,X
0803- B0 1C      BCS  $0821     <- actually a branch-on-carry (not a JMP)
```

JMP at **$0802** is never executed!



4am to deprotect and preserve