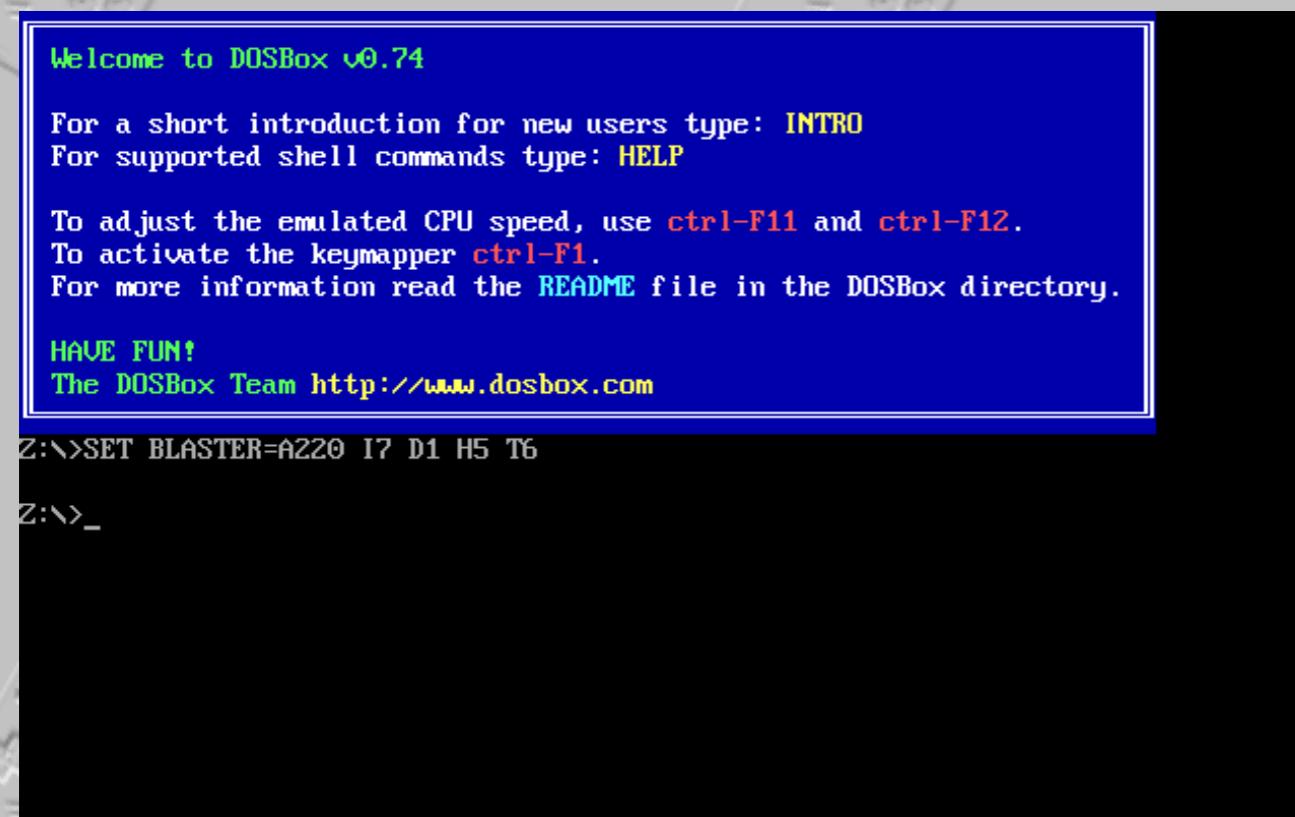


GAME BOY EMULATION



EMULATION OVERVIEW

“hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest)”



GAME BOY ARCHITECTURE

- Split up software components
- Simulate parts, test individually, “wire” together

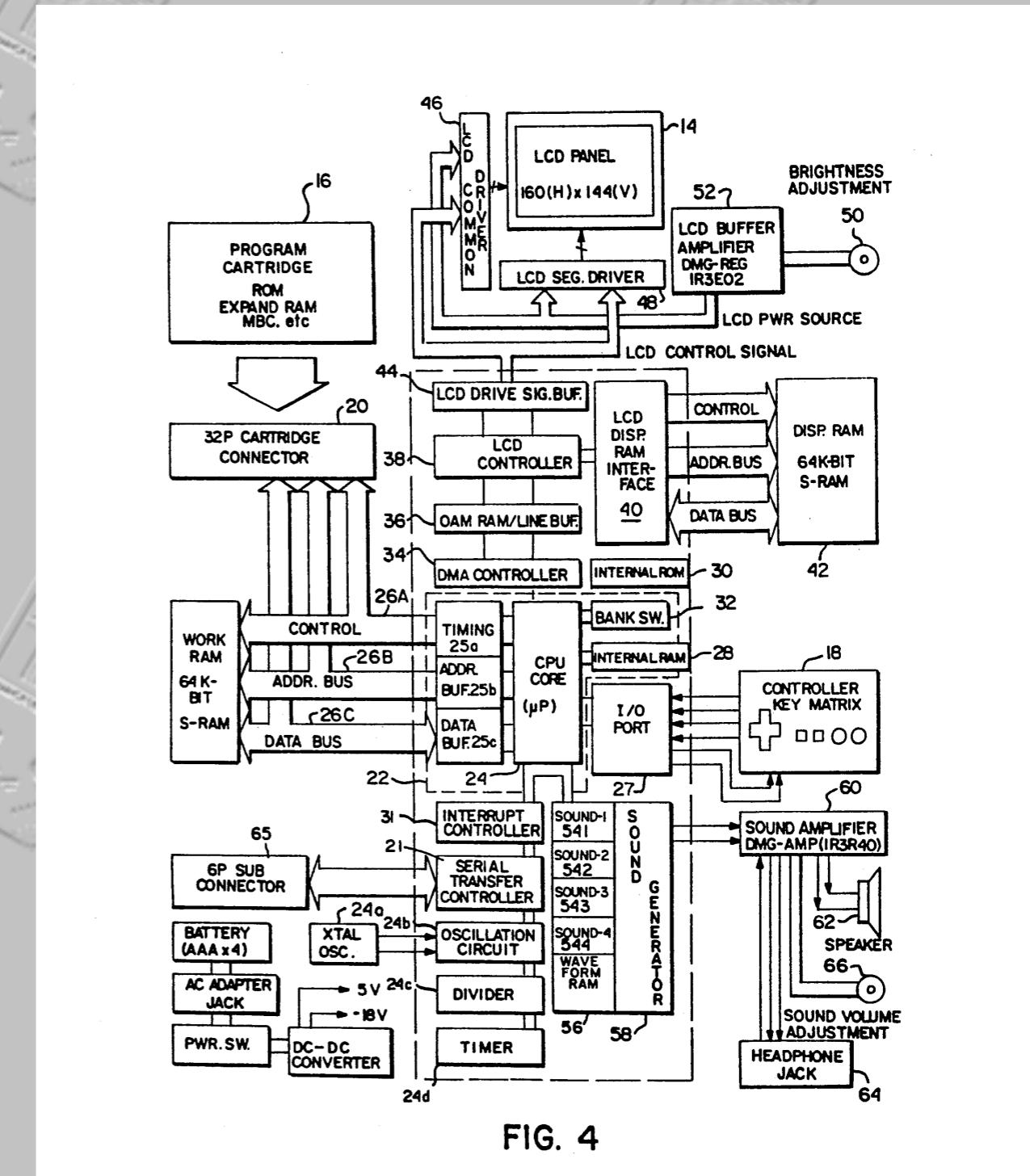


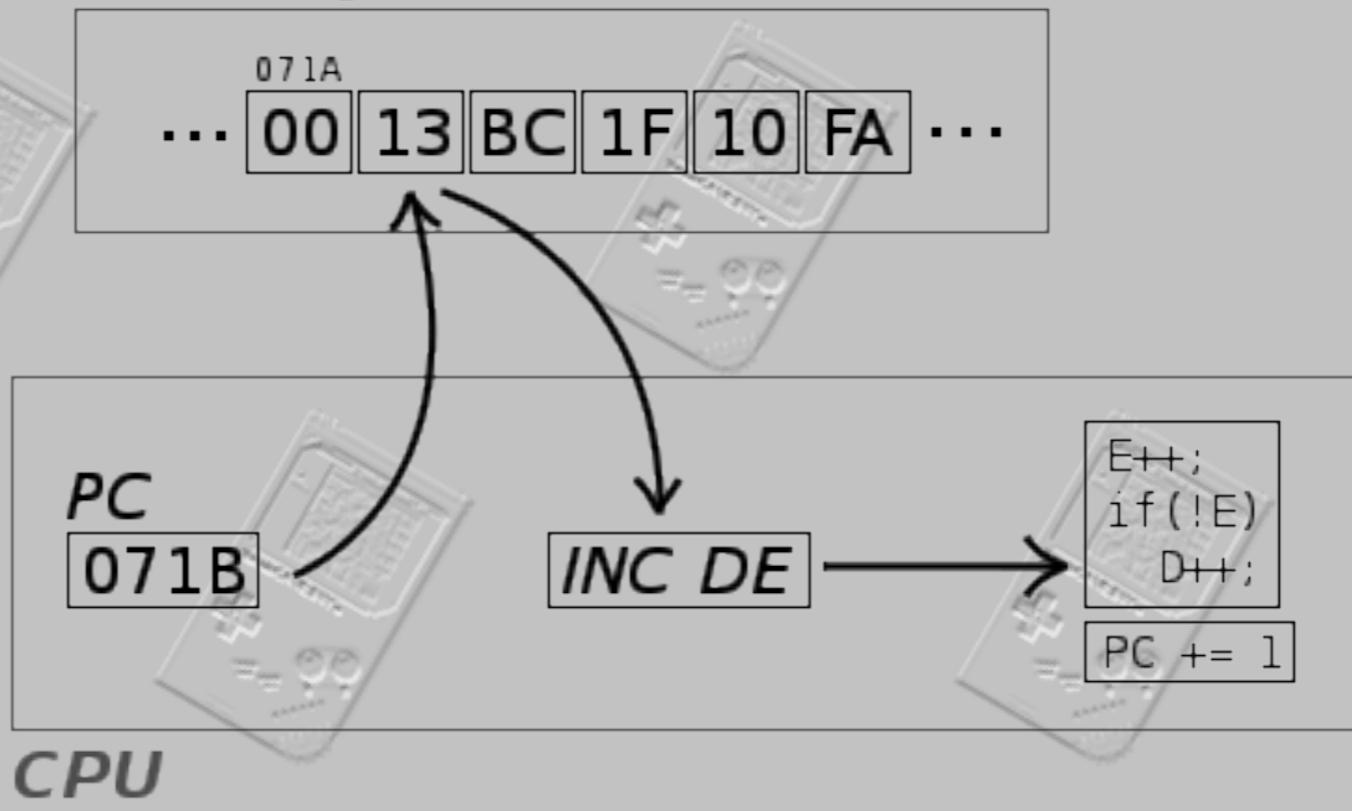
FIG. 4

WHAT'S A ROM?

"I've been downloading them 100% legally for years!"

```
00000100: 00c3 5001 ced 6666 cc0d 000b 0373 0083 ...P...ff....s...
00000110: 000c 000d 0008 111f 8889 000e dccc 6ee6 .....n.....
00000120: dddd d999 bbbb 6763 6e0e eccc dddc 999f .....gcn.....
00000130: bbb9 333e 5445 5452 4953 0000 0000 0000 ..3>TETRIS.....
00000140: 0000 0000 0000 0000 0000 0001 000b 89b5 .....
```

Memory



SHARP LR35902

- Frankenstein Zilog Z80 / Intel 8080
- 8 8-bit registers, 2 16-bit
- Whopping 4.19MHz clock speed
- 245 non-prefix instructions, 255 CB-prefixed



CPU IMPLEMENTATION

- CPU
 - registers
 - instructions
 - CB-prefixed instructions
 - executors

REGISTERS

```
type Registers struct {  
    a byte // Accumulator  
    // Flags  
    // ZNHC 0000  
    // Z = zero, N = subtract, H = half carry, C = carry  
    f byte  
  
    b byte  
    c byte  
  
    d byte  
    e byte  
  
    h byte  
    l byte  
  
    sp []byte // Stack pointer  
    PC []byte // Program counter  
}
```

Also contains methods
to modify flags

INSTRUCTIONS

```
type Instruction struct {
    Mnemonic string
    // Number of T cycles instruction takes to execute
    // Divide by 4 to get number of M cycles
    TCycles uint16
    NumOperands uint16
    Executor     func() int // Executes appropriate function
}
```

```
0x00: Instruction{"NOP", 4, 1, func() int { return 0 }},
0x01: Instruction{"LD BC,i16", 12, 3, func() int { gbcpu.LDrrnn(&gbcpu.Regs.b, &gbcpu.Reg.c); return 0 }},
0x02: Instruction{"LD (BC),A", 8, 1, func() int { gbcpu.LDaar(&gbcpu.Reg.b, &gbcpu.Reg.c, &gbcpu.Reg.a); return 0 }},
0x03: Instruction{"INC BC", 8, 1, func() int { gbcpu.INCrr(&gbcpu.Reg.b, &gbcpu.Reg.c); return 0 }},
0x04: Instruction{"INC B", 4, 1, func() int { gbcpu.INCr(&gbcpu.Reg.b); return 0 }},
```

EXECUTORS

```
// XORr -> e.g. XOR B
// Bitwise XOR of reg into A
// Flags: Z000
func (gbcpu *GBCPU) XORr(reg *byte) {
    gbcpu.Reg.s.a ^= *reg

    // Check for zero
    if gbcpu.Reg.s.a == 0x00 {
        gbcpu.Reg.setZero()
    } else {
        gbcpu.Reg.clearZero()
    }

    // Set flags
    gbcpu.Reg.clearSubtract()
    gbcpu.Reg.clearHalfCarry()
    gbcpu.Reg.clearCarry()
}
```

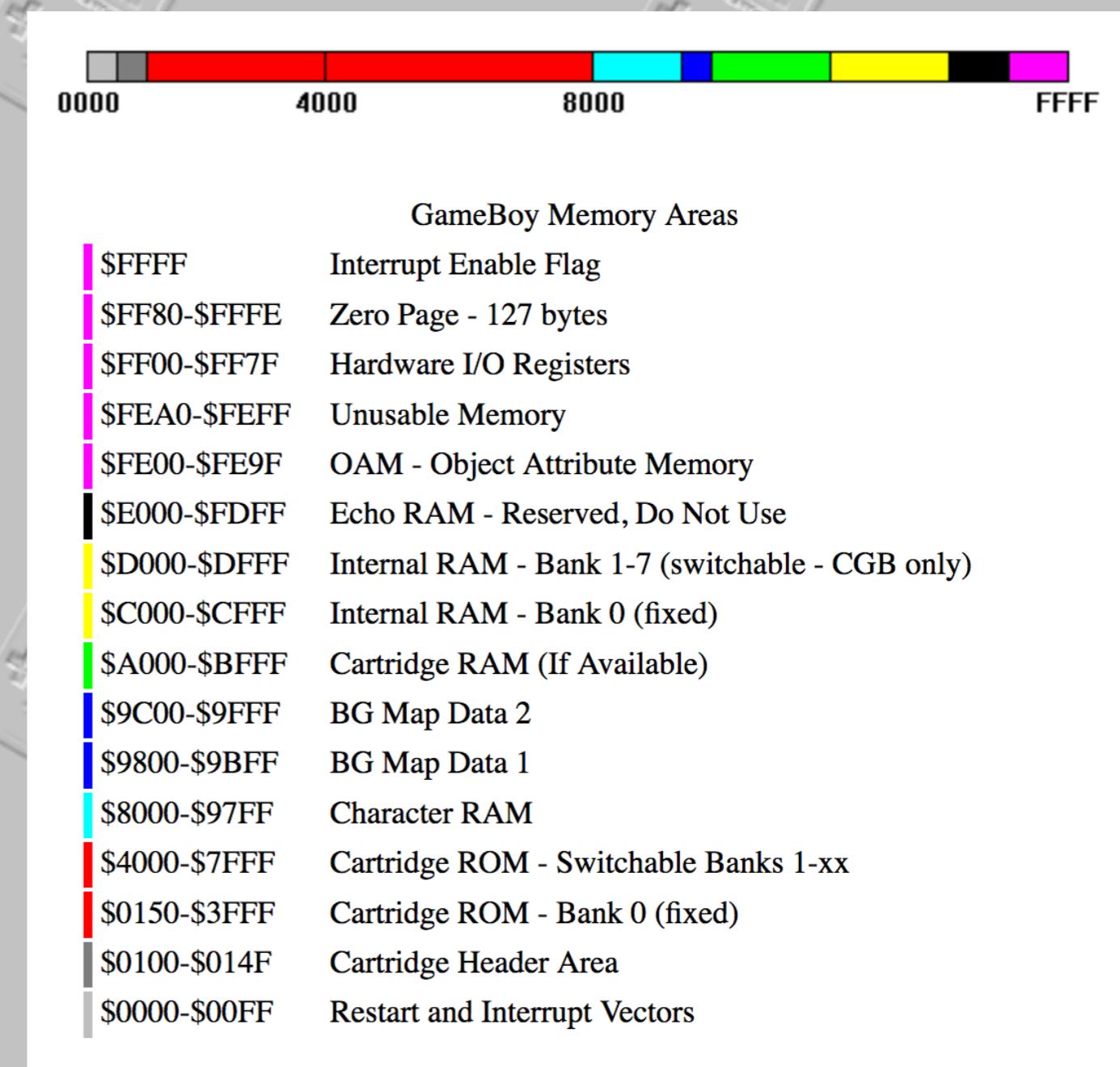
```
// LDSPnn -> e.g. LD SP,i16
// Loads 16 bit value from next 2 bytes into SP
// Flags: none
func (gbcpu *GBCPU) LDSPnn() {
    operands := gbcpu.getOperands(2)
    gbcpu.Reg.s.sp = operands
}

// CALLaa -> e.g. CALL $028B
// Pushes the addr at PC+3 to the stack
// Jumps to the address specified by next 2 bytes
func (gbcpu *GBCPU) CALLaa() {
    operands := gbcpu.getOperands(2)
    nextInstr := gbcpu.sliceToInt(gbcpu.Reg.s.PC) + 3
    nextInstrBytes := make([]byte, 2)
    binary.LittleEndian.PutUint16(nextInstrBytes, nextInstr)
    gbcpu.pushByteToStack(nextInstrBytes[1])
    gbcpu.pushByteToStack(nextInstrBytes[0])
    gbcpu.Reg.s.PC = operands

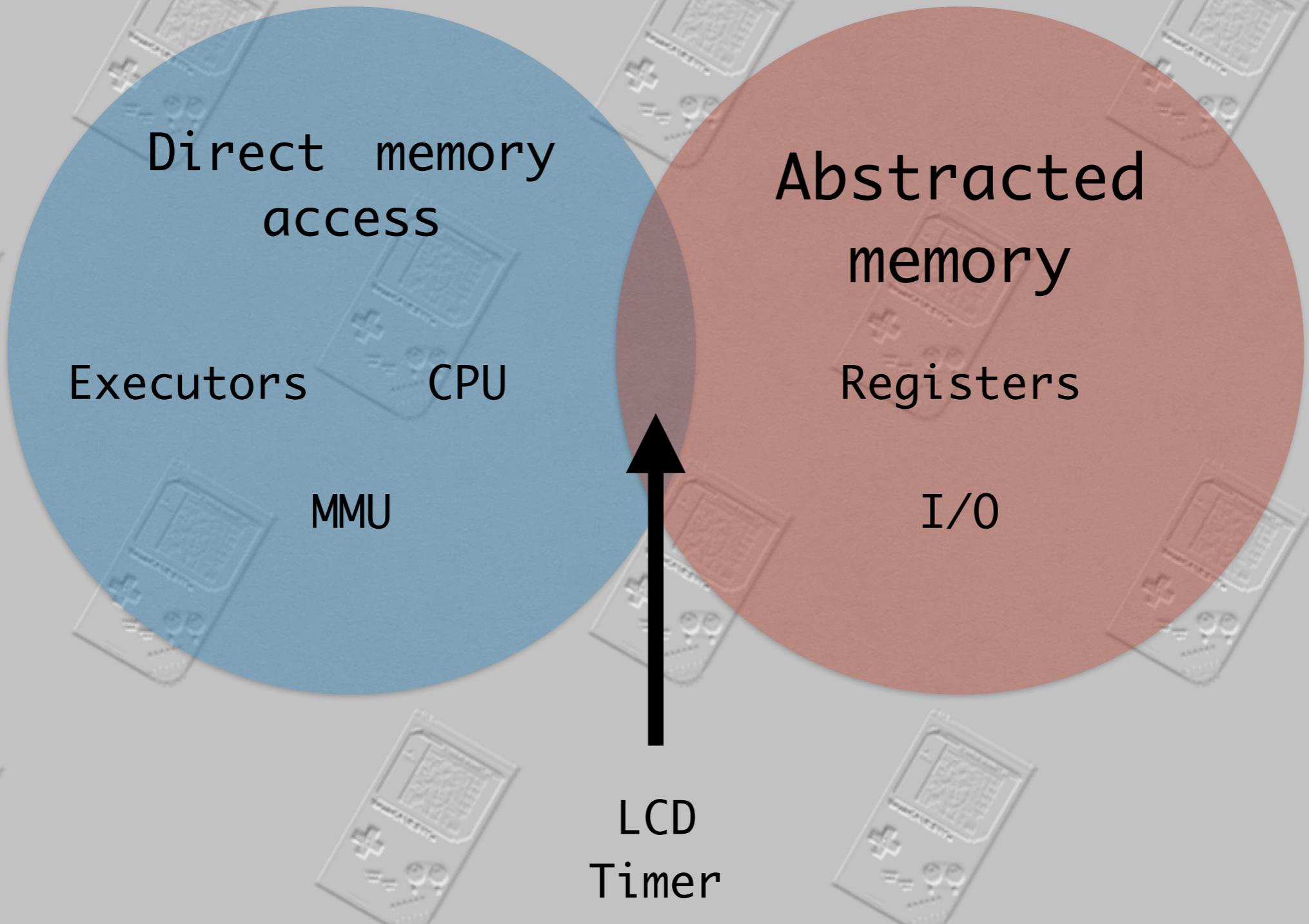
    gbcpu.Jumped = true
}
```

MEMORY

- Straightforward representation as a flat byte array
- Load ROM into correct location, begin fetch-decode-dispatch loop
- Can ignore all memory dealing with video & I/O



REPRESENTATION



BENEFITS

```
wb: function(addr, val) {
    switch(addr&0xF000)
    {
        // ROM bank 0
        // MBC1: Turn external RAM on
        case 0x0000: case 0x1000:
            switch(MMU._carttype)
            {
                case 1:
                    MMU._mbc[1].ramon = ((val&0xF)==0xA)?1:0;
                    break;
            }
            break;

        // MBC1: ROM bank switch
        case 0x2000: case 0x3000:
            switch(MMU._carttype)
            {
                case 1:
                    MMU._mbc[1].rombank &= 0x60;
                    val &= 0x1F;
                    if(!val) val=1;
                    MMU._mbc[1].rombank |= val;
                    MMU._romoffs = MMU._mbc[1].rombank * 0x4000;
                    break;
            }
            break;
    }
}
```

```
func (gbmmu *GBMMU) WriteData(addr uint16, data byte) {
    if addr == 0xFF00 {
        GbIO.SetCol(data)
    } else if addr == 0xFF0F {
        // TODO What do writes here really do? Ignored or bit set?
        // gbmmu.Memory[0xFF0F] |= (1 << 0)
    } else if addr == 0xFF41 {
        // TODO Same as above
    } else if addr >= 0x0000 && addr <= 0x150 {
        // Don't allow writes to invalid locations
    } else if addr == 0xFF46 {
        spriteAddr := int(data) * 256
        for i := range gbmmu.Memory[0xFE00:0xEA0] {
            gbmmu.Memory[0xFE00+i] = gbmmu.Memory[spriteAddr+i]
        }
    } else if addr == 0xFF07 {
        gbmmu.Memory[addr] += data
    } else {
        gbmmu.Memory[addr] = data
    }
}
```

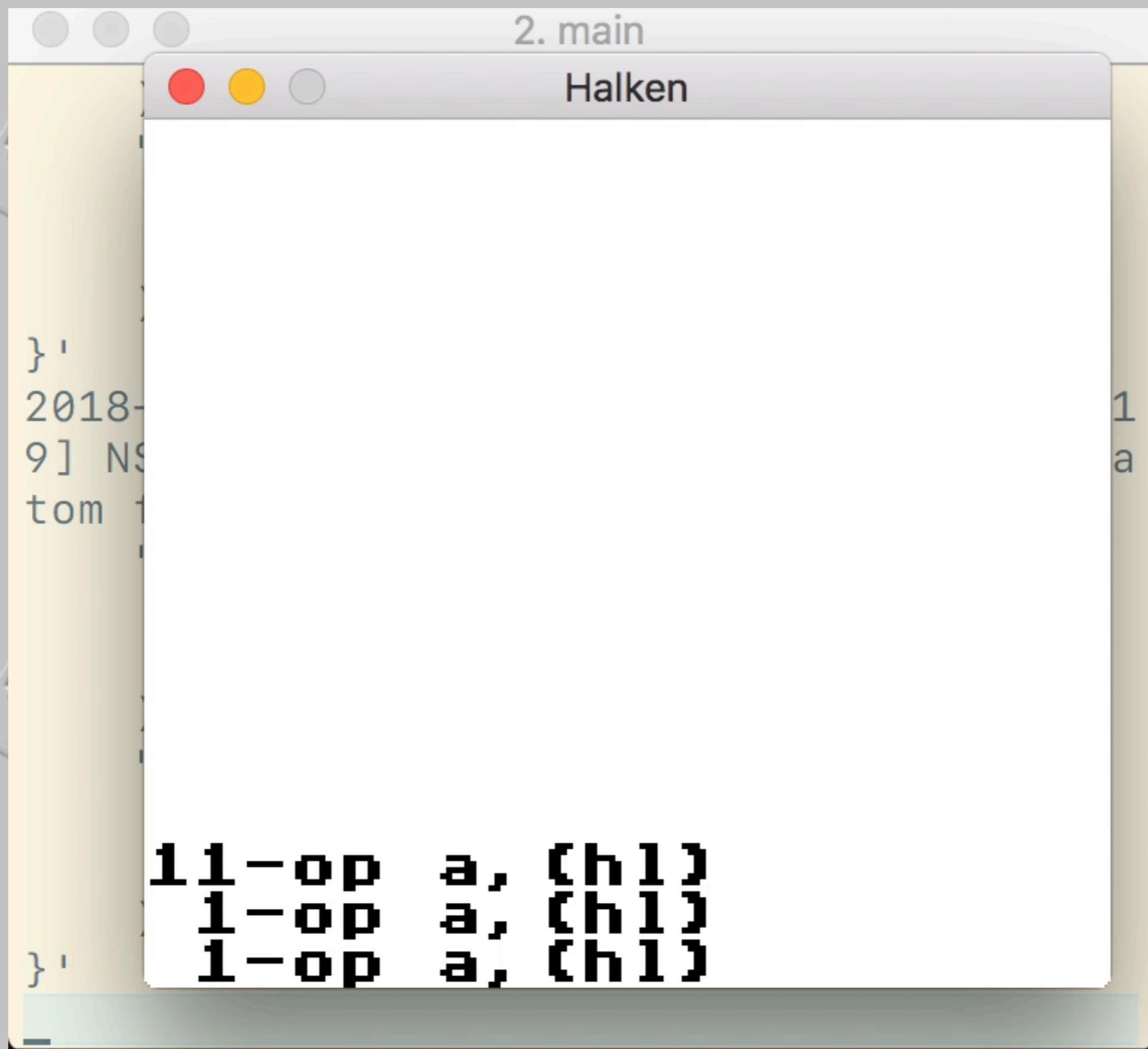
PAINFUL DEBUGGING

bgb debugger - Z:\Users\nsm\Downloads\tetris.gb

File Search Run Debug Window Execution profiler

ROM0:00FF FF	rat 38	; 4 4	af= 00A0 lcdc=D3 <input checked="" type="checkbox"/> z		
ROM0:0100 00	nop	; 1 5	bc= 0000 stat=81 <input type="checkbox"/> n		
ROM0:0101 C3 50 01	jp 0150	; 4 9	de= 0410 ly= 90 <input checked="" type="checkbox"/> h		
ROM0:0104 CE ED 66 66+	db CE,ED,66,66,CC,0D,00,08,03,73,00,83,00,0C, ROM0:0114 00 08 11 1F+	00,08,11,1F,88,89,00,0E,DC,CC,6E,E6,DD,DD, ROM0:0124 88 88 67 63+	88,88,67,63,6E,0E,EC,CC,DD,DC,99,9F,88,89, ROM0:0134 54 45 54 52+ ROM0:0143 00 ROM0:0144 00 00 ROM0:0146 00 ROM0:0147 00 ROM0:0148 00 ROM0:0149 00 ROM0:014A 00 ROM0:014B 01 ROM0:014C 00 ROM0:014D 08 ROM0:014E 89 85 ROM0:0150 C3 88 02 ROM0:0153 CD 28 2A ROM0:0156 F0 41 ROM0:0158 E6 03 ROM0:015A 20 FA ROM0:015C 46 ROM0:015D F0 41 ROM0:015F E6 03 ROM0:0161 00 FF	"TETRIS00000000" db 00 ; DMG - classic gameboy db 00,00 ; new license db 00 ; SGB flag: not SGB capable db 00 ; cart type: ROM db 00 ; rom size: 32 KiB db 00 ; ram size: 0 B db 00 ; destination code: Japanese db 01 ; old licensee: not SGB capable db 00 ; mask ROM version number db 08 ; header check (OK) db 89,85 ; global check (okay) jp 0288 ; 4 14 call 2A28 ; 6 20 ld a,(ff00+41) ;lcd stat ; 3 23 and a,03 ; 2 25 jr nz,0156 ; 2 27 ld b,(hl) ; 2 29 ld a,(ff00+41) ;lcd stat ; 3 32 and a,03 ; 2 34 -- 0150 ; 2 26	WRA1:D027 0000 WRA1:D025 0000 WRA1:D023 0000 WRA1:D021 0000 WRA1:D01F 0000 WRA1:D01D 0000 WRA1:D018 0000 WRA1:D019 0000 WRA1:D017 0000 WRA1:D015 0000 WRA1:D013 0000 WRA1:D011 0000 WRA1:D00F 0000 WRA1:D00D 0000 WRA1:D00B 0000 WRA1:D009 0000 WRA1:D007 0000 WRA1:D005 0000 WRA1:D003 0000 WRA1:D001 0000 WRA0:CFFF 0000 WRA0:CFFD 036F █ WRA0:CFFB 6920 WRA0:CFF9 0000 WRA0:CFF7 0410 WRA0:CFF5 0410 WRA0:CFF3 658F WRA0:CFF1 03E6 WRA0:CFEF 025A WRA0:CFED 0000 WRA0:CFE8 0000 WRA0:CFE9 0000
ROM0:0000 C3 88 02 00 00 00 00 C3 88 02 FF FF FF FF FF Ä< Ä<					
ROM0:0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:0020 FF FF FF FF FF FF 87 E1 5F 16 00 19 5E 23 †á ... ^#					
ROM0:0030 56 D5 E1 E9 FF FF FF FF FF FF FF FF FF FF FF FF VÖáé.....					
ROM0:0040 C3 FD 01 FF FF FF FF C3 12 27 FF FF FF FF FF Äý..... Ä. '.....					
ROM0:0050 C3 12 27 FF FF FF FF C3 7E 01 FF FF FF FF FF Ä. '..... Ä~.....					
ROM0:0060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:0070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:0080 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:0090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:00A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:00B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					
ROM0:00C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF 					

NICE DEBUGGING



The image shows a terminal window titled "Halken" with the identifier "2. main". The window contains assembly code and some log output. The assembly code at the bottom is:

```
11-op a, (h1)
1-op a, (h1)
1-op a, (h1)
```

The log output above the assembly code includes:

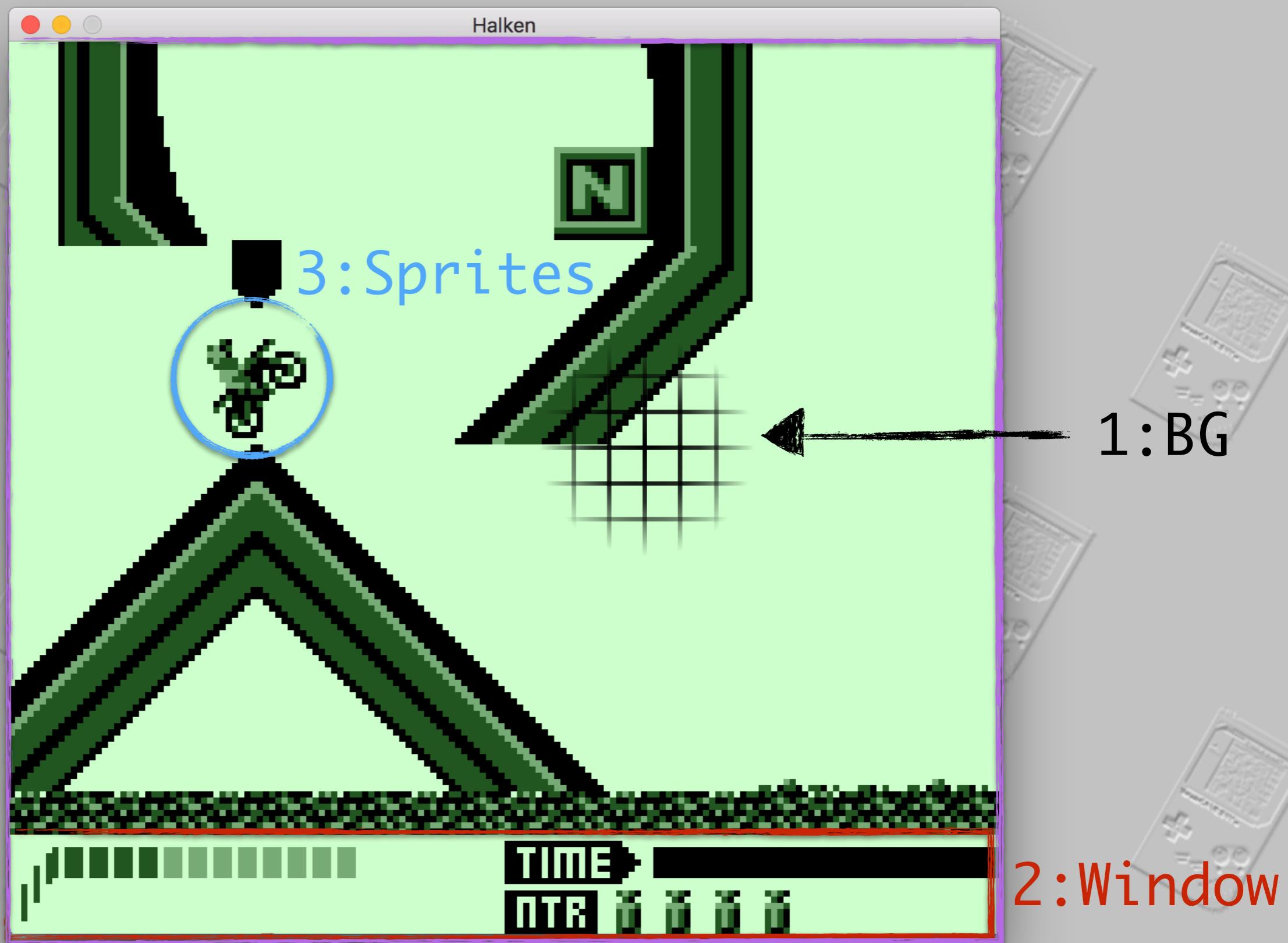
```
}'
2018-
9] NS
tom
```

THE LCD

- **160x144 pixels**
- **20x18 tiles**,
each **8x8 pixels**
- **256x256**
“background”
- **40 sprites at once**
- **4 colors, shades of gray**



GRAPHICS "LAYERS"



TILE MAPS



8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E,8E
8E,8E,8E,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5A,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B,5B
5B,5B,5B,5C,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5D,80,81,82,83,90,91,92,81,82,83,90,6C,6D,6E,6F
70,71,72,5E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5D,84,85,86,87,93,94,95,85,86,87,93,73,74,75,76
77,78,2F,5E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5D,2F,88,89,2F,96,97,98,88,89,2F,96,79,7A,7B,7C
7D,7E,2F,5E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5D,2F,8A,8B,2F,8E,8F,6B,8A,8B,2F,8E,7F,66,67,68
69,6A,2F,5E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
5F,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60
60,60,60,61,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,3C,3C,3C,3C,3C,3C,3C,3C,3C,3C,3C,3C,3C,3C,3D,3E
3C,3C,3C,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,8C,8C,62,63,8C,8C,3A,8C,8C,8C,8C,3A,42,43
3B,8C,8C,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,3A,8C,64,65,8C,8C,8C,3B,8C,8C,8C,8C,44,45
8C,8C,8C,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,8C,8C,8C,8C,8C,8C,8C,8C,8C,8C,8C,46,47,48
49,3F,40,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,8C,8C,8C,8C,3A,8C,8C,8C,8C,53,54,8C,4A,4B,4C
4D,42,43,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F
8E,8C,8C,8C,8C,8C,8C,8C,54,55,56,57,4E,4F,50
51,52,45,8E,2F,2F,2F,2F,2F,2F,2F,2F,2F
41,41,41,41,41,41,41,41,41,41,41,41,41,41,41
41,41,41,41,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
2F,2F,59,19,15,0A,22,0E,1B,2F,2F,99,19,15,0A
22,0E,1B,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
2F,2F,9A,9A,9A,9A,9A,9A,9A,2F,2F,2F,9A,9A,9A
9A,9A,9A,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F
2F,2F,2F,2F,33,30,31,32,31,2F,34,35,36,37,38,39
2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F,2F

TILE RENDERING

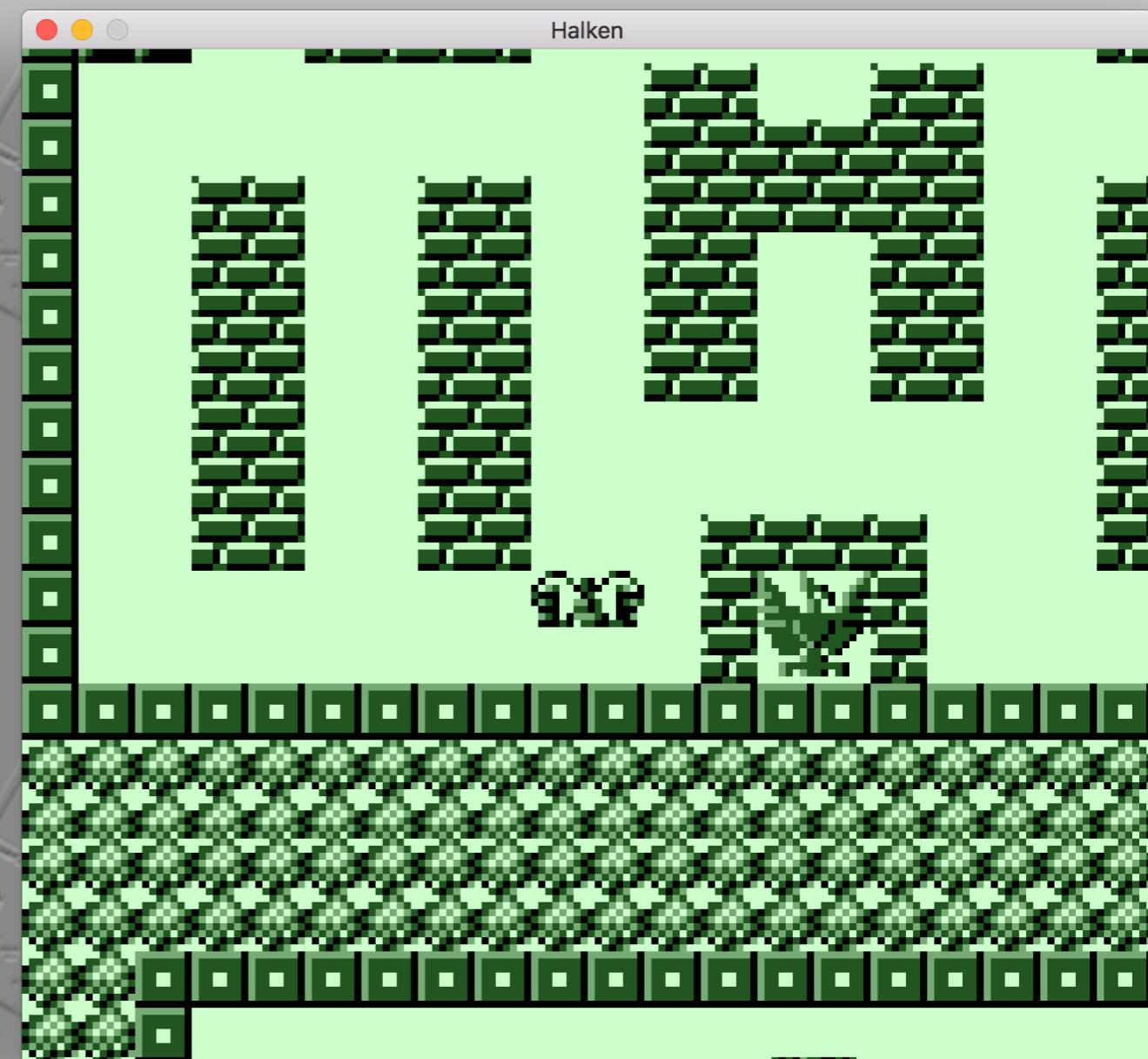
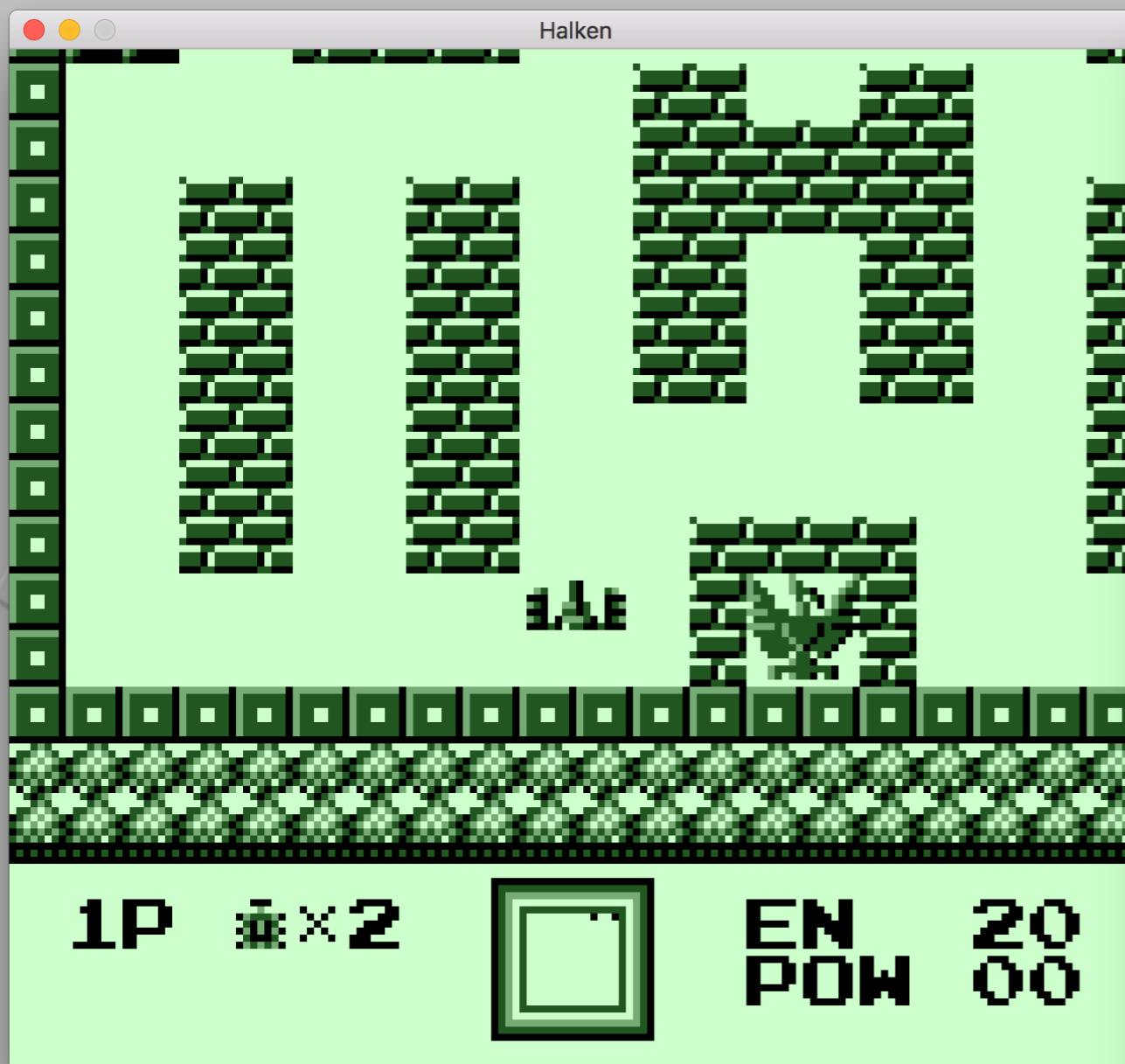
Tile:

.33333..
22...22.
11...11.
2222222. <-- digits represent
33...33. color numbers
22...22.
11...11.
.....

Image:

.33333..	->	01111100	->	7Ch
22...22.	->	01111100	->	7Ch
11...11.	->	00000000	->	00h
2222222.	->	11000110	->	C6h
11...11.	->	11000110	->	C6h
		00000000	->	00h
2222222.	->	00000000	->	00h
		11111110	->	FEh
33...33.	->	11000110	->	C6h
		11000110	->	C6h
22...22.	->	00000000	->	00h
		11000110	->	C6h
11...11.	->	11000110	->	C6h
		00000000	->	00h
.....	->	00000000	->	00h
		00000000	->	00h

WINDOW



Win. X offset->\$FE4B, Win. Y offset->\$FE4A

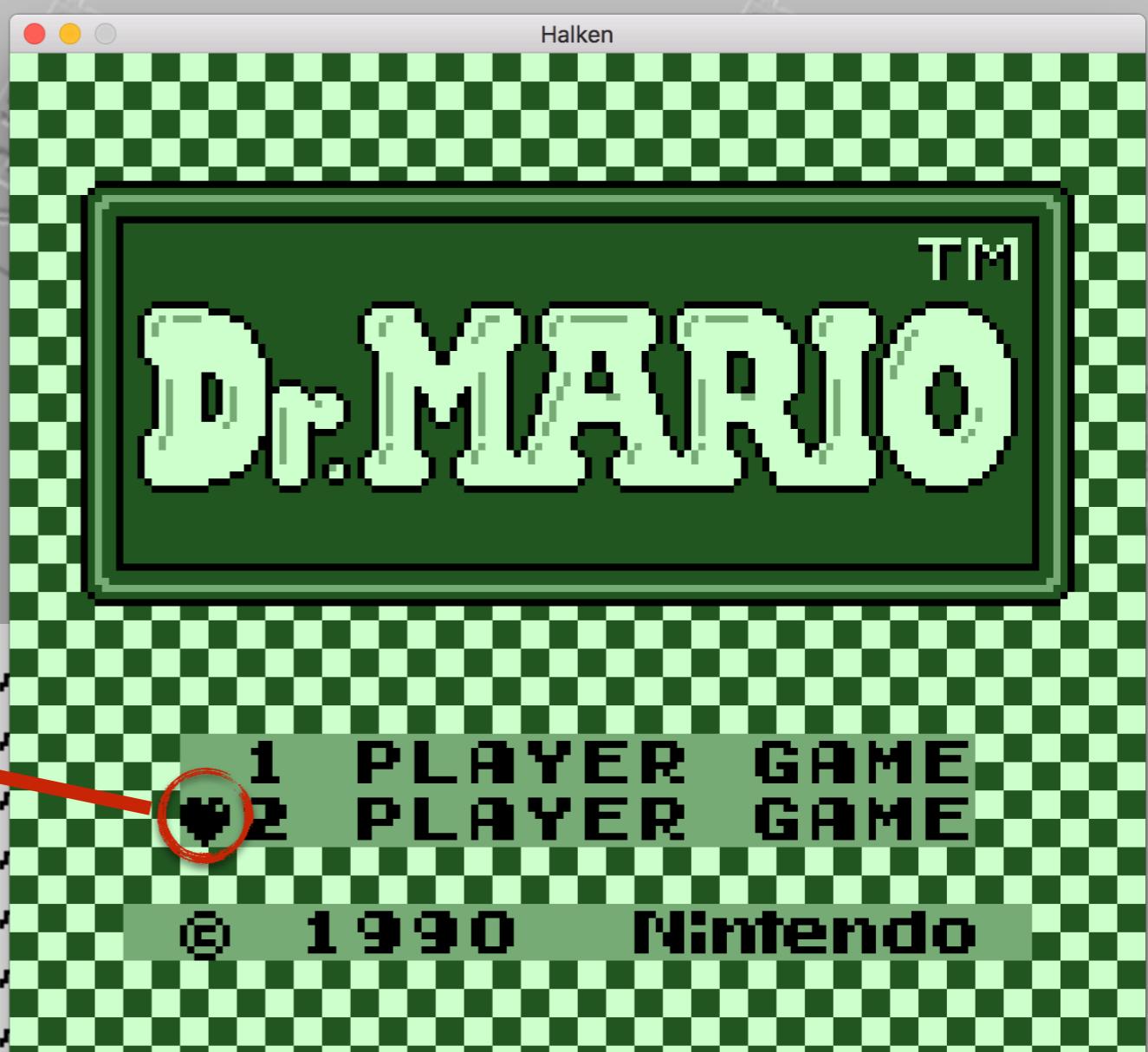
SPRITES

Attributes

- 0->Palette # (CGB)
- 1->Palette # (CGB)
- 2->Palette # (CGB)
- 3->VRAM bank (CGB)
- 4->Palette # (DMG)
- 5->X flip
- 6->Y flip
- 7->Above BG

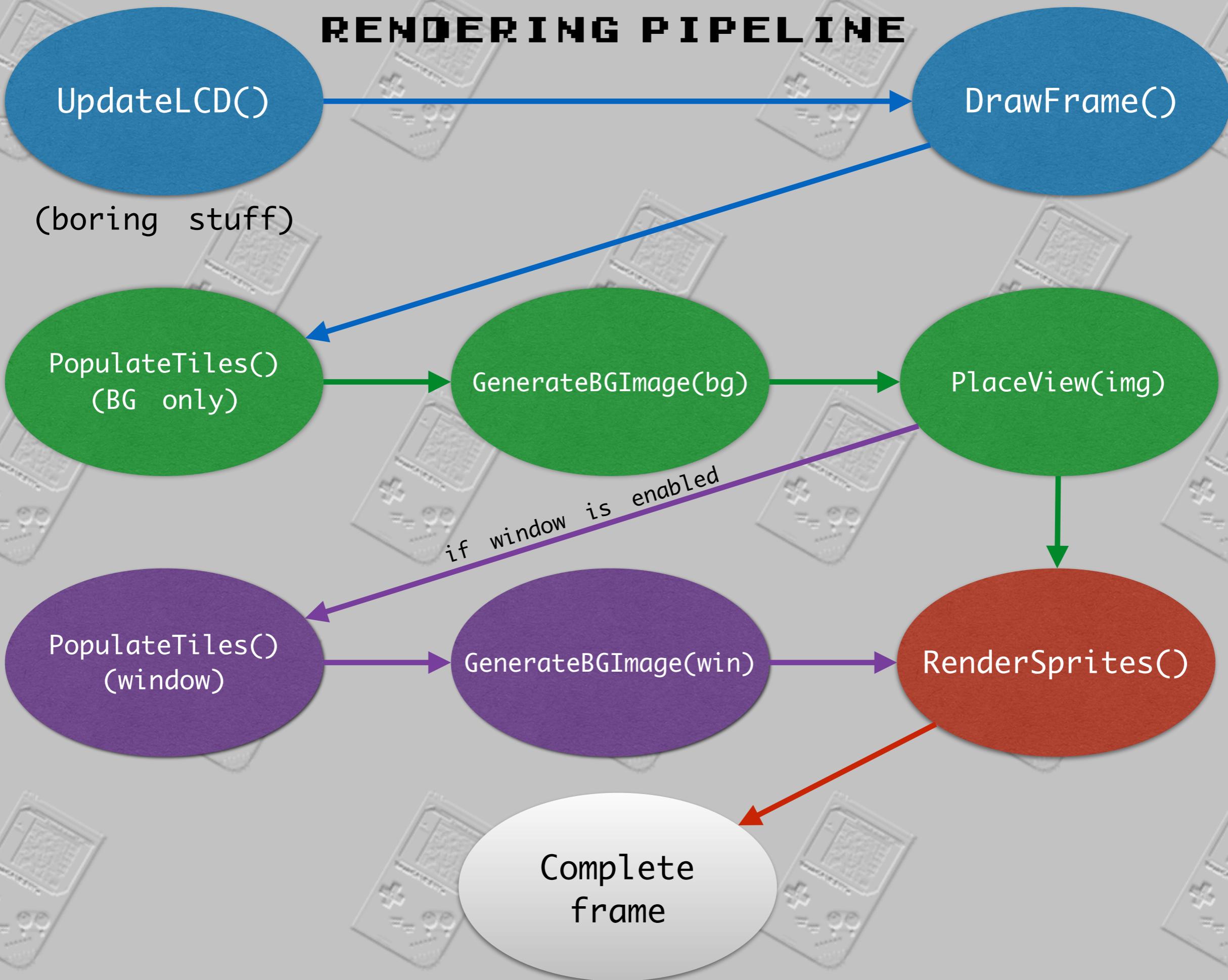
X
Y
ID

78,20,9B,00	01,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00, 00,00,00,00,00,00,00,00
-------------	---



Sprite OAM: \$FE00-\$FEA0

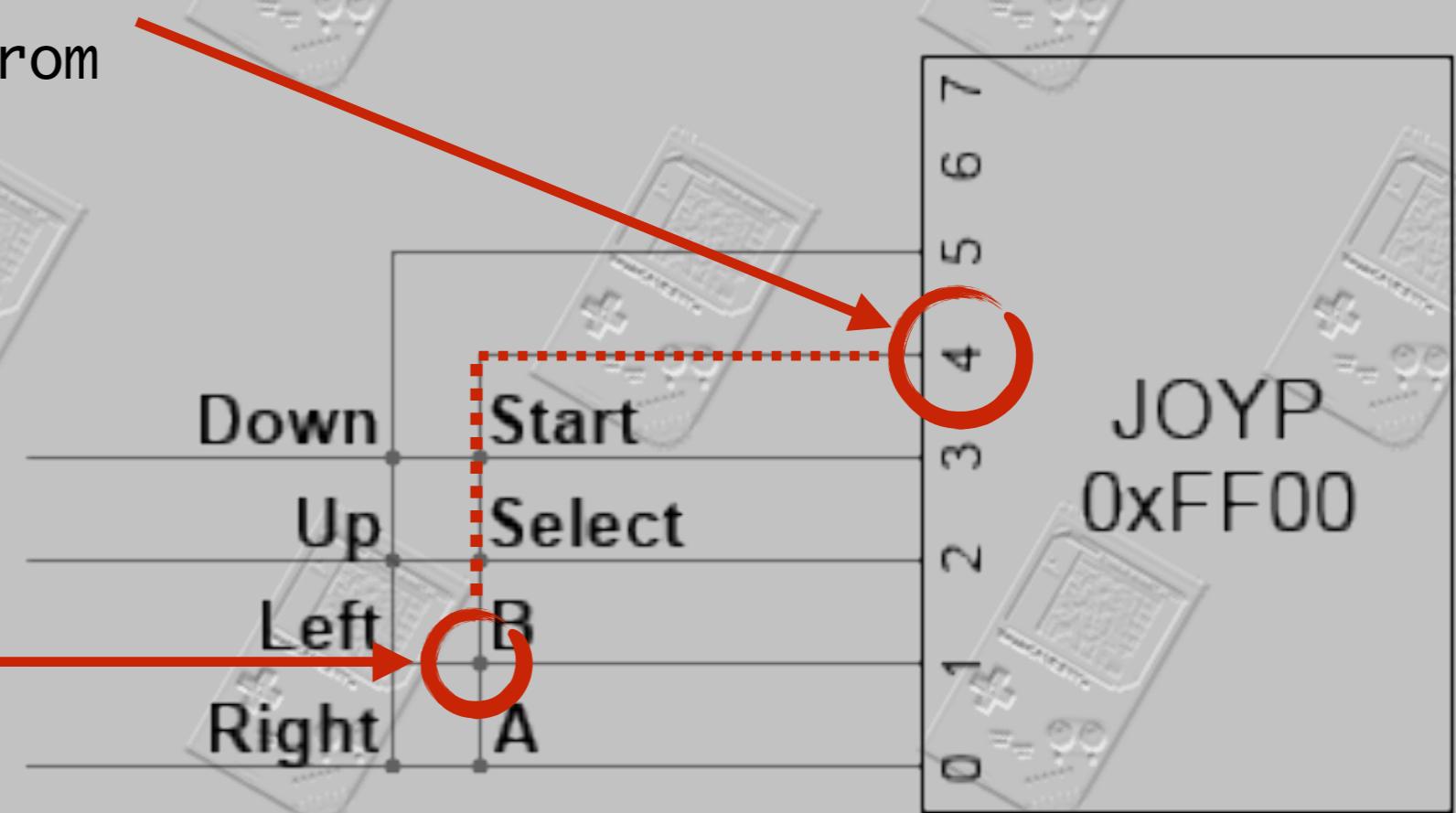
RENDERING PIPELINE



INPUTS

Hardware selects which column it wants input from

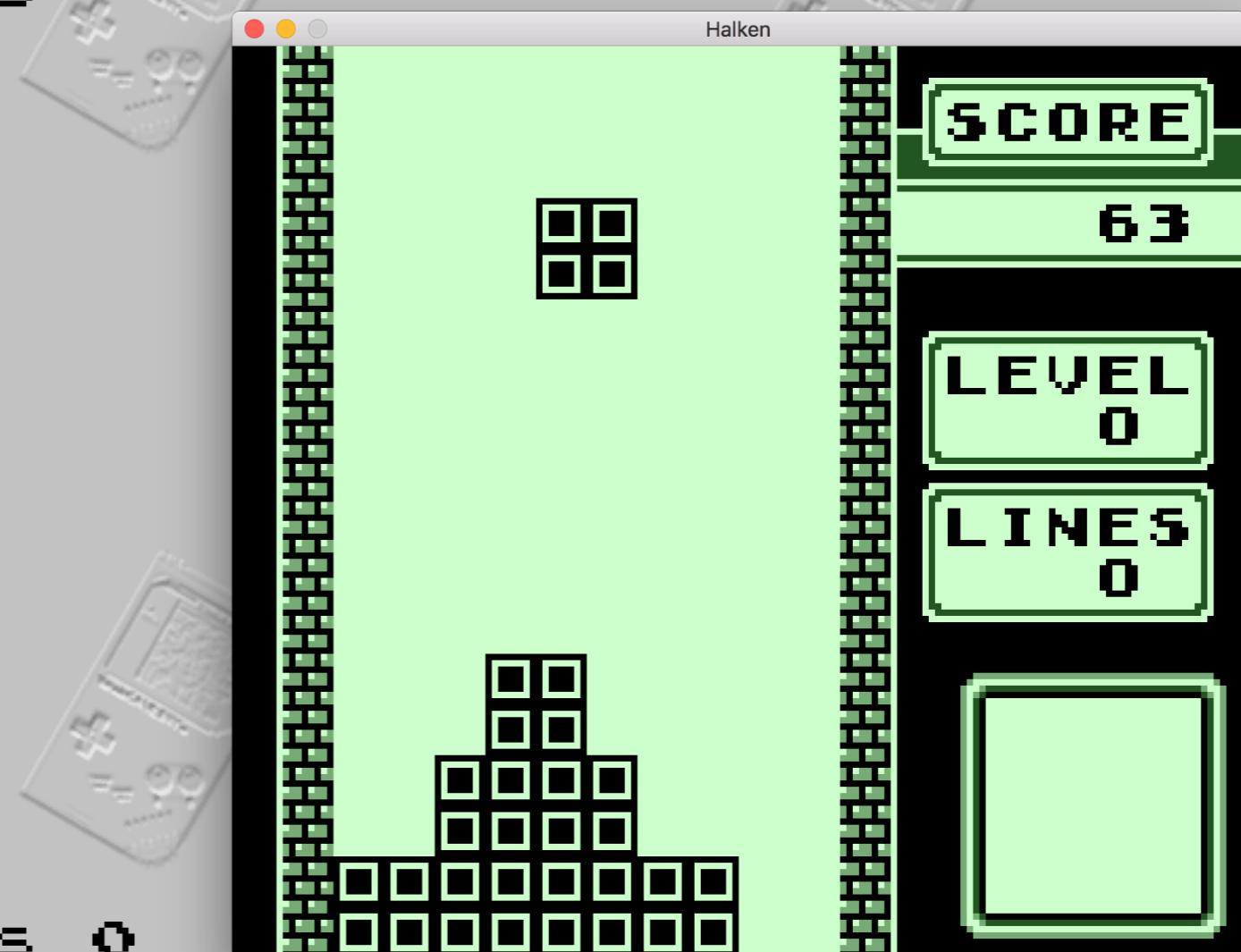
Last button pressed updated 60 times per second



cols = [0x0F, 0x0F]
Values altered by reading inputs
Selected column = index 0 or 1
Return value

TIMER

- \$FF04 = divider, acts as PRNG
- \$FF05 = counter, triggers timer interrupt when overflows
- \$FF06 = modulo, when counter overflows, resets. Used to generate PRNG in divider
- \$FF07 = control, bits 0 & 1 set speed, bit 2 sets timer on/off



WHAT'S DONE



- CPU
- Memory
- (most) Graphics
- (most) Interrupts
- Timer
- I/O

WHAT'S NOT



- Sound
- LCD STAT interrupt not entirely correct
- Sprites only use 1 color palette
- 8x16 sprites aren't implemented, get cut off
- ROM banking

WHAT I LEARNED

- a lot
- Excellent exercise in lower-level Programming, computer organization/architecture
- Some software projects take a LONG time to write
- Testing is extremely helpful, especially when someone else wrote the tests
- Hardware restrictions lead to creative programming