Emulating Game Boy in Java

Tomek Rękawek, trekawek@gmail.com

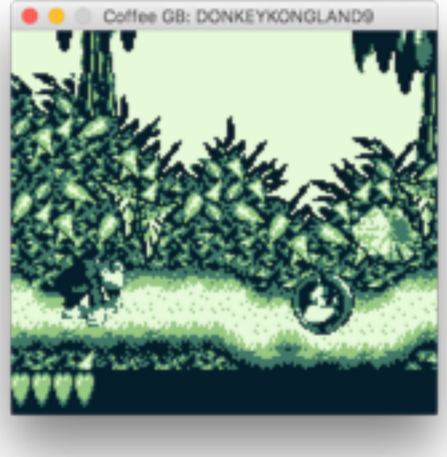
Image: http://clipground.com/image-post/25612-game-boy-clipart-2.jpg.html

Nintendo Game Boy

- DMG-001 (Dot Matrix Game)
- Released in 1989
- Sharp LR35902 CPU
    - Z80-based, 4.19 MHz
- 8 kB RAM + 8 kB VRAM
- 160x144, 4 shades of grey
- 118 690 000 sold units

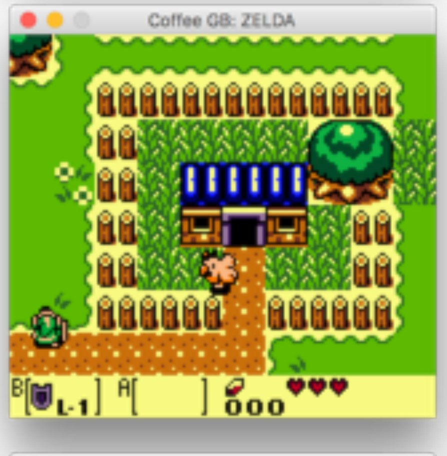
Image: https://pl.wikipedia.org/wiki/Game_Boy_(konsola)#/media/File:Nintendo_Gameboy.jpg

Game Boy titles



- Ports from other 8-, 16- and 32- (!) consoles and computers

- Many exclusives



MAT received
TOTODILE!

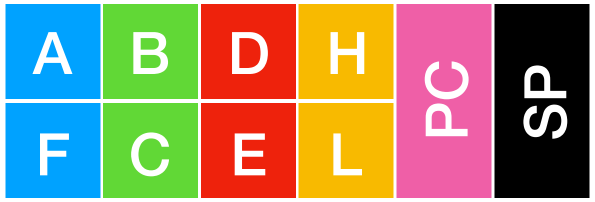
Yet another emulator?

Image: Tomek's home office

- Retrocomputers are magical
- Emulators are magical too
- A chance to learn everything about a simple computer
- Implementing all the subsystems is addicting

CPU

- Z80 based, 4.19 MHz
- 245 basic instructions
- 256 extended bit operations (prefix: 0xCB)
- 64 kB addressable space
  - RAM + ROM + IO
- Registers:

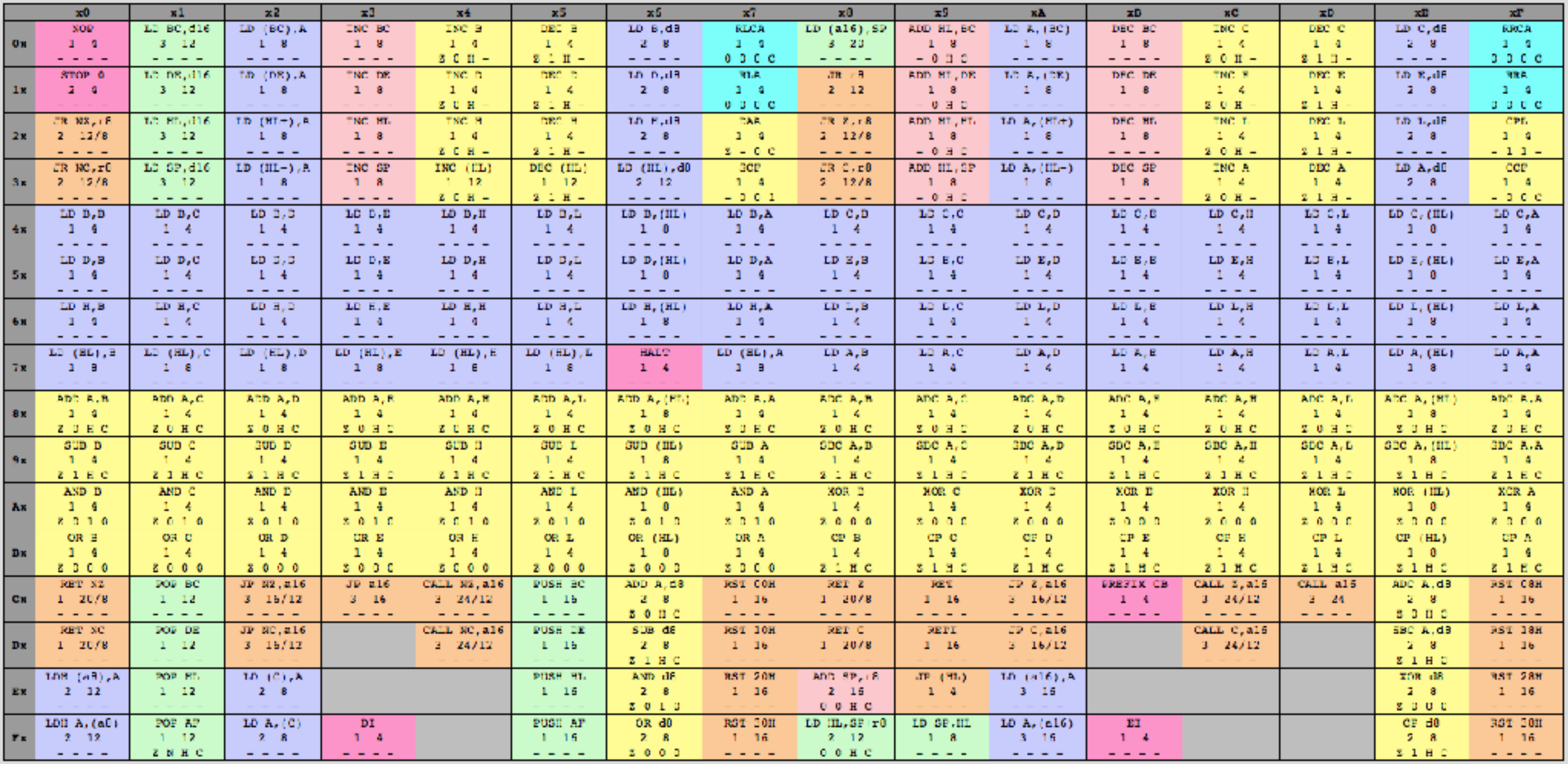


A B D H PC SP

F C E L

Image: http://www.vgmpf.com/Wiki/index.php/DMG-CPU_B

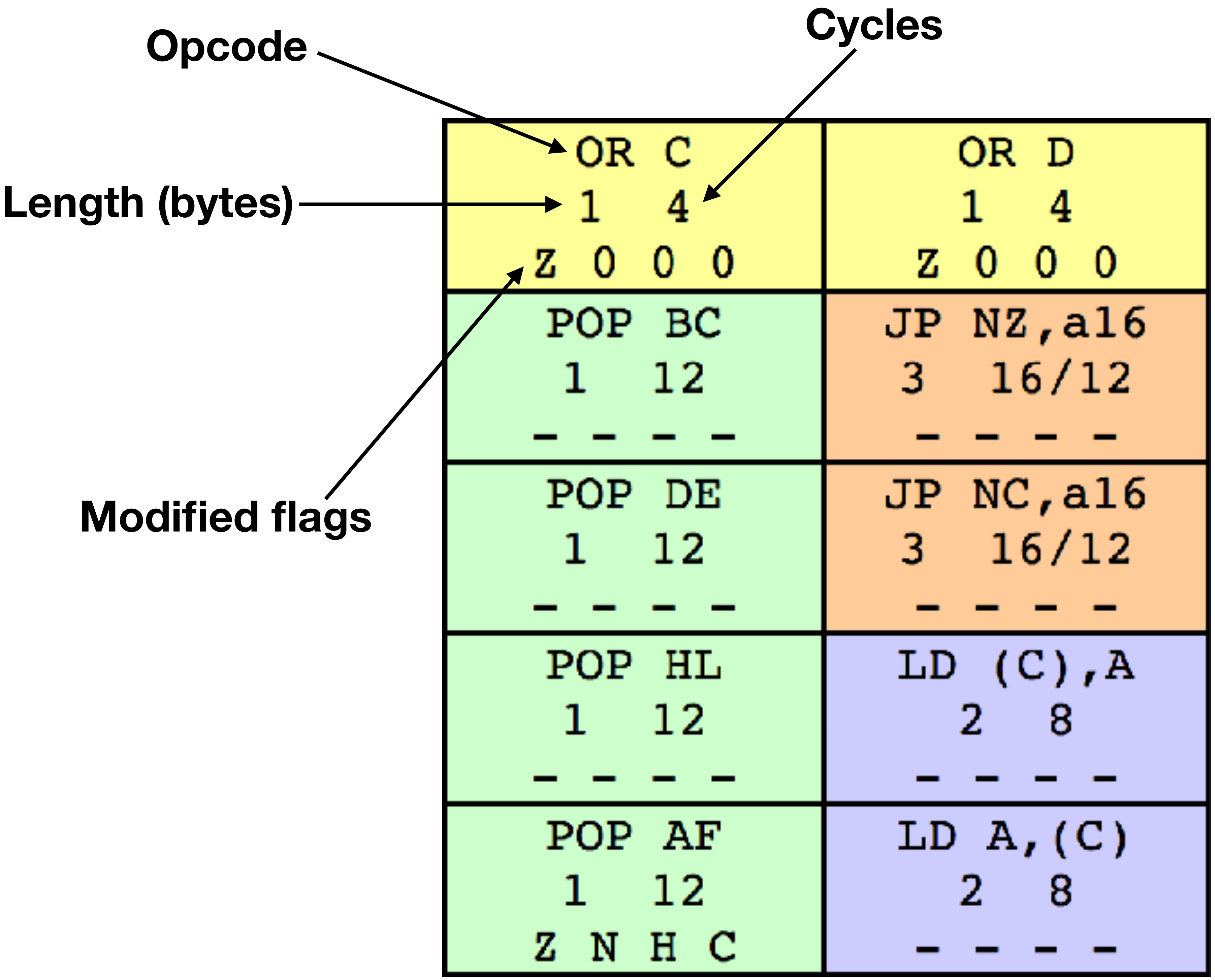
CPU instructions set



	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC,d16 3 12 - - - -	LD (BC),A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B,d8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD (a16),SP 3 20 - - - -	ADD HL,BC 1 8 - 0 H C	LD A,(BC) 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C,d8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP 0 2 4 - - - -	LD DE,d16 3 12 - - - -	LD (DE),A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D,d8 2 8 - - - -	RLA 1 4 0 0 0 C	JR r8 2 12 - - - -	ADD HL,DE 1 8 - 0 H C	LD A,(DE) 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E,d8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ,r8 2 12/8 - - - -	LD HL,d16 3 12 - - - -	LD (HL+),A 1 8 - - - -	INC HL 1 8 - - - -	INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H,d8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z,r8 2 12/8 - - - -	ADD HL,HL 1 8 - 0 H C	LD A,(HL+) 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L,d8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC,r8 2 12/8 - - - -	LD SP,d16 3 12 - - - -	LD (HL-),A 1 8 - - - -	INC SP 1 8 - - - -	INC (HL) 1 12 Z 0 H -	DEC (HL) 1 12 Z 1 H -	LD (HL),d8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C,r8 2 12/8 - - - -	ADD HL,SP 1 8 - 0 H C	LD A,(HL-) 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A,d8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B,B 1 4 - - - -	LD B,C 1 4 - - - -	LD B,D 1 4 - - - -	LD B,E 1 4 - - - -	LD B,H 1 4 - - - -	LD B,L 1 4 - - - -	LD B,(HL) 1 8 - - - -	LD B,A 1 4 - - - -	LD C,B 1 4 - - - -	LD C,C 1 4 - - - -	LD C,D 1 4 - - - -	LD C,E 1 4 - - - -	LD C,H 1 4 - - - -	LD C,L 1 4 - - - -	LD C,(HL) 1 8 - - - -	LD C,A 1 4 - - - -
5x	LD D,B 1 4 - - - -	LD D,C 1 4 - - - -	LD D,D 1 4 - - - -	LD D,E 1 4 - - - -	LD D,H 1 4 - - - -	LD D,L 1 4 - - - -	LD D,(HL) 1 8 - - - -	LD D,A 1 4 - - - -	LD E,B 1 4 - - - -	LD E,C 1 4 - - - -	LD E,D 1 4 - - - -	LD E,E 1 4 - - - -	LD E,H 1 4 - - - -	LD E,L 1 4 - - - -	LD E,(HL) 1 8 - - - -	LD E,A 1 4 - - - -
6x	LD H,B 1 4 - - - -	LD H,C 1 4 - - - -	LD H,D 1 4 - - - -	LD H,E 1 4 - - - -	LD H,H 1 4 - - - -	LD H,L 1 4 - - - -	LD H,(HL) 1 8 - - - -	LD H,A 1 4 - - - -	LD L,B 1 4 - - - -	LD L,C 1 4 - - - -	LD L,D 1 4 - - - -	LD L,E 1 4 - - - -	LD L,H 1 4 - - - -	LD L,L 1 4 - - - -	LD L,(HL) 1 8 - - - -	LD L,A 1 4 - - - -
7x	LD (HL),B 1 8 - - - -	LD (HL),C 1 8 - - - -	LD (HL),D 1 8 - - - -	LD (HL),E 1 8 - - - -	LD (HL),H 1 8 - - - -	LD (HL),L 1 8 - - - -	HALT 1 4 - - - -	LD (HL),A 1 8 - - - -	LD A,B 1 4 - - - -	LD A,C 1 4 - - - -	LD A,D 1 4 - - - -	LD A,E 1 4 - - - -	LD A,H 1 4 - - - -	LD A,L 1 4 - - - -	LD A,(HL) 1 8 - - - -	LD A,A 1 4 - - - -
8x	ADD A,B 1 4 Z 0 H C	ADD A,C 1 4 Z 0 H C	ADD A,D 1 4 Z 0 H C	ADD A,E 1 4 Z 0 H C	ADD A,H 1 4 Z 0 H C	ADD A,L 1 4 Z 0 H C	ADD A,(HL) 1 8 Z 0 H C	ADD A,A 1 4 Z 0 H C	ADC A,B 1 4 Z 0 H C	ADC A,C 1 4 Z 0 H C	ADC A,D 1 4 Z 0 H C	ADC A,E 1 4 Z 0 H C	ADC A,H 1 4 Z 0 H C	ADC A,L 1 4 Z 0 H C	ADC A,(HL) 1 8 Z 0 H C	ADC A,A 1 4 Z 0 H C
9x	SUB B 1 4 Z 1 H C	SUB C 1 4 Z 1 H C	SUB D 1 4 Z 1 H C	SUB E 1 4 Z 1 H C	SUB H 1 4 Z 1 H C	SUB L 1 4 Z 1 H C	SUB (HL) 1 8 Z 1 H C	SUB A 1 4 Z 1 H C	SBC A,B 1 4 Z 1 H C	SBC A,C 1 4 Z 1 H C	SBC A,D 1 4 Z 1 H C	SBC A,E 1 4 Z 1 H C	SBC A,H 1 4 Z 1 H C	SBC A,L 1 4 Z 1 H C	SBC A,(HL) 1 8 Z 1 H C	SBC A,A 1 4 Z 1 H C
Ax	AND B 1 4 Z 0 1 0	AND C 1 4 Z 0 1 0	AND D 1 4 Z 0 1 0	AND E 1 4 Z 0 1 0	AND H 1 4 Z 0 1 0	AND L 1 4 Z 0 1 0	AND (HL) 1 8 Z 0 1 0	AND A 1 4 Z 0 1 0	XOR B 1 4 Z 0 0 0	XOR C 1 4 Z 0 0 0	XOR D 1 4 Z 0 0 0	XOR E 1 4 Z 0 0 0	XOR H 1 4 Z 0 0 0	XOR L 1 4 Z 0 0 0	XOR (HL) 1 8 Z 0 0 0	XOR A 1 4 Z 0 0 0
Bx	OR B 1 4 Z 0 0 0	OR C 1 4 Z 0 0 0	OR D 1 4 Z 0 0 0	OR E 1 4 Z 0 0 0	OR H 1 4 Z 0 0 0	OR L 1 4 Z 0 0 0	OR (HL) 1 8 Z 0 0 0	OR A 1 4 Z 0 0 0	CP B 1 4 Z 1 H C	CP C 1 4 Z 1 H C	CP D 1 4 Z 1 H C	CP E 1 4 Z 1 H C	CP H 1 4 Z 1 H C	CP L 1 4 Z 1 H C	CP (HL) 1 8 Z 1 H C	CP A 1 4 Z 1 H C
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ,a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ,a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A,d8 2 8 Z 0 H C	RST 00H 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z,a16 3 16/12 - - - -	PREFIX CB 1 4 - - - -	CALL Z,a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A,d8 2 8 Z 0 H C	RST 08H 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC,a16 3 16/12 - - - -		CALL NC,a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB d8 2 8 Z 1 H C	RST 10H 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C,a16 3 16/12 - - - -		CALL C,a16 3 24/12 - - - -		SBC A,d8 2 8 Z 1 H C	RST 18H 1 16 - - - -
Ex	LDH (a8),A 2 12 - - - -	POP HL 1 12 - - - -	LD (C),A 2 8 - - - -			PUSH HL 1 16 - - - -	AND d8 2 8 Z 0 1 0	RST 20H 1 16 - - - -	ADD SP,r8 2 16 0 0 H C	JP (HL) 1 4 - - - -	LD (a16),A 3 16 - - - -				XOR d8 2 8 Z 0 0 0	RST 28H 1 16 - - - -
Fx	LDH A,(a8) 2 12 - - - -	POP AF 1 12 Z N H C	LD A,(C) 2 8 - - - -	DI 1 4 - - - -		PUSH AF 1 16 - - - -	OR d8 2 8 Z 0 0 0	RST 30H 1 16 - - - -	LD HL,SP+r8 2 12 0 0 H C	LD SP,HL 1 8 - - - -	LD A,(a16) 3 16 - - - -	EI 1 4 - - - -			CP d8 2 8 Z 1 H C	RST 38H 1 16 - - - -

Image: http://www.pastraiser.com/cpu/gameboy/gameboy_opcodes.html

Instruction features



Opcode

Cycles

Length (bytes)

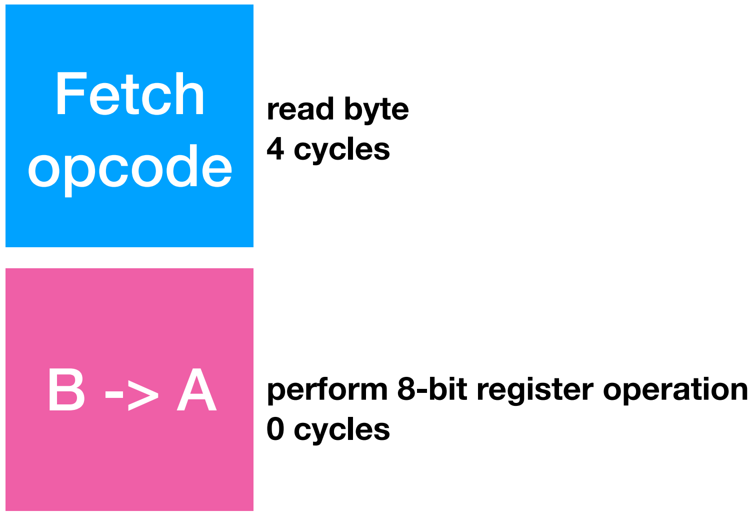
Modified flags

OR C 1 4 Z 0 0 0	OR D 1 4 Z 0 0 0
POP BC 1 12 - - - -	JP NZ,a16 3 16/12 - - - -
POP DE 1 12 - - - -	JP NC,a16 3 16/12 - - - -
POP HL 1 12 - - - -	LD (C),A 2 8 - - - -
POP AF 1 12 Z N H C	LD A,(C) 2 8 - - - -

Image: http://www.pastraiser.com/cpu/gameboy/gameboy_opcodes.html

Instruction cycles

LD A, B



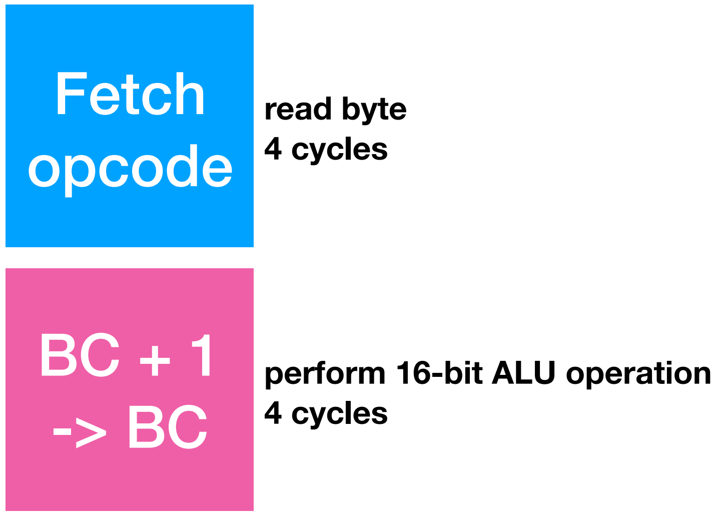
Fetch opcode

read byte
4 cycles

B -> A

perform 8-bit register operation
0 cycles

INC BC



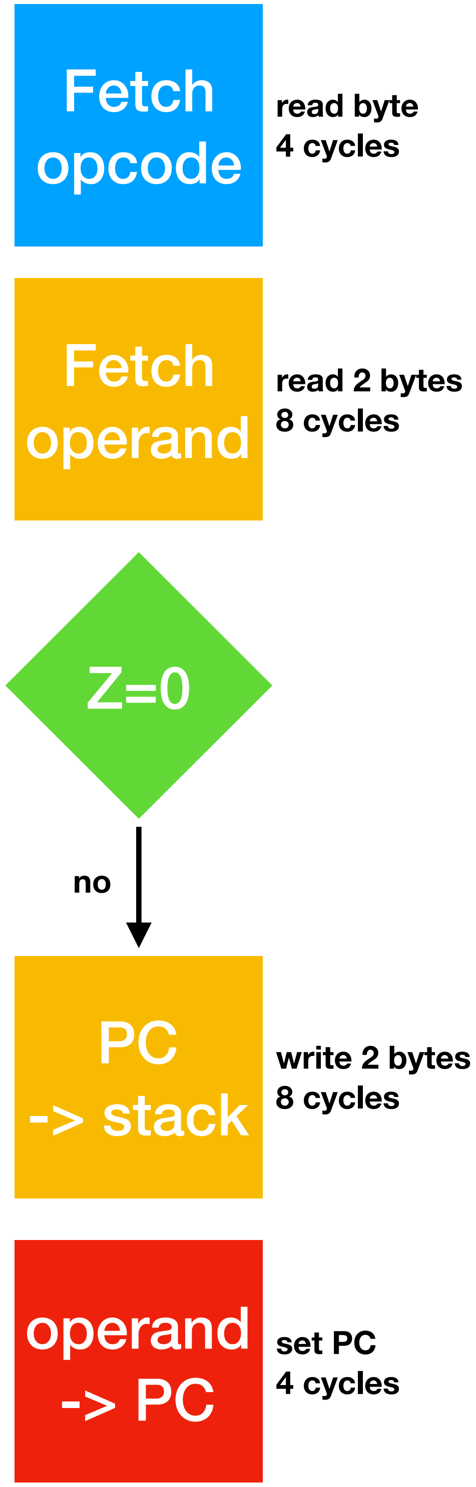
Fetch opcode

read byte
4 cycles

BC + 1 -> BC

perform 16-bit ALU operation
4 cycles

CALL NZ, a16



Fetch opcode

read byte
4 cycles

Fetch operand

read 2 bytes
8 cycles

Z=0

no

PC -> stack

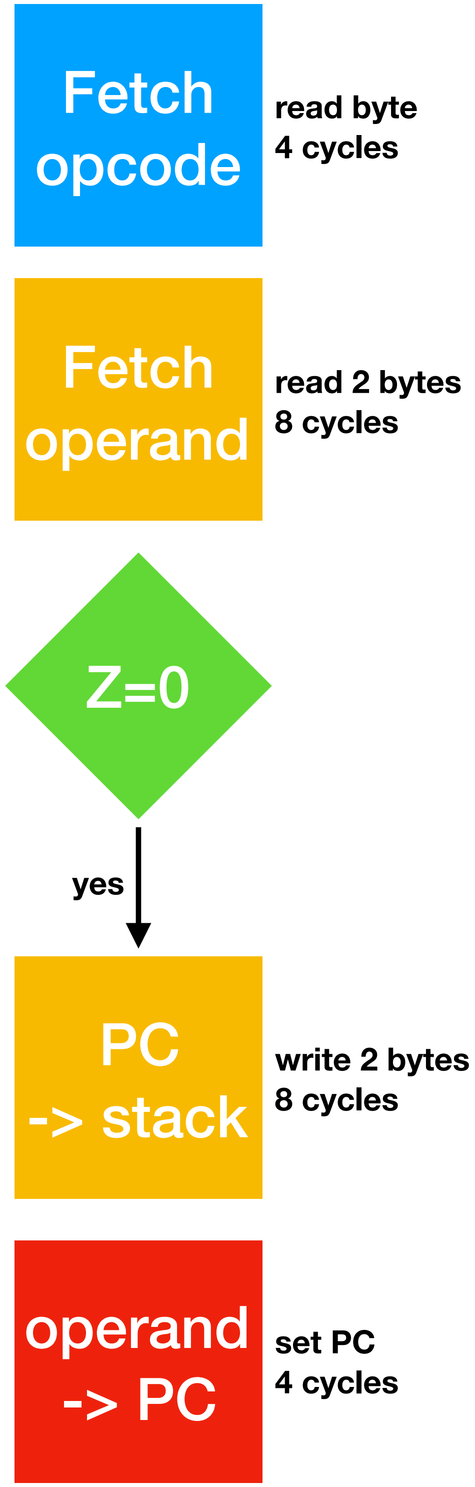
write 2 bytes
8 cycles

operand -> PC

set PC
4 cycles

CPU instructions DSL

CALL NZ, a16



Fetch opcode

read byte
4 cycles

Fetch operand

read 2 bytes
8 cycles

Z=0

yes

PC -> stack

write 2 bytes
8 cycles

operand -> PC

set PC
4 cycles

```
regCmd(opcodes, 0xC4, "CALL NZ,a16")
        .proceedIf("NZ")
        .extraCycle()
        .load("PC")
        .push()
        .load("a16")
        .store("PC");
```

https://github.com/trekawek/coffee-gb/blob/master/src/main/java/eu/rekawek/coffeegb/cpu/Opcodes.java

CPU as state machine

```
private final AddressSpace addrSpace;

public CPU(AddressSpace addrSpace) {
    this.addrSpace = addrSpace;
}

public void tick() {
    if (divider++ == 4) { divider = 0; }
    else { return; }

    switch (state) {
        case OPCODE:
        // fetch opcode

        case RUNNING:
        // run a single 4-cycle operation

        case …:
    }
}
```

https://github.com/trekawek/coffee-gb/blob/master/src/main/java/eu/rekawek/coffeegb/cpu/Cpu.java

Alternative CPU-driven approach

```
static void call_cc_a16(GB_gameboy_t *gb, uint8_t opcode)
{
    uint16_t call_addr = gb->pc - 1;
    if (condition_code(gb, opcode)) {
        GB_advance_cycles(gb, 4);
        gb->registers[GB_REGISTER_SP] -= 2;
        uint16_t addr = GB_read_memory(gb, gb->pc++);
        GB_advance_cycles(gb, 4);
        addr |= (GB_read_memory(gb, gb->pc++) << 8)
        GB_advance_cycles(gb, 8);
        GB_write_memory(gb, gb->registers[GB_REGISTER_SP] + 1, (gb->pc) >> 8);
        GB_advance_cycles(gb, 4);
        GB_write_memory(gb, gb->registers[GB_REGISTER_SP], (gb->pc) & 0xFF);
        GB_advance_cycles(gb, 4);
        gb->pc = addr;
        GB_debugger_call_hook(gb, call_addr);
    }
    else {
        GB_advance_cycles(gb, 12);
        gb->pc += 2;
    }
}
```

https://github.com/LIJI32/SameBoy/blob/master/Core/z80_cpu.c

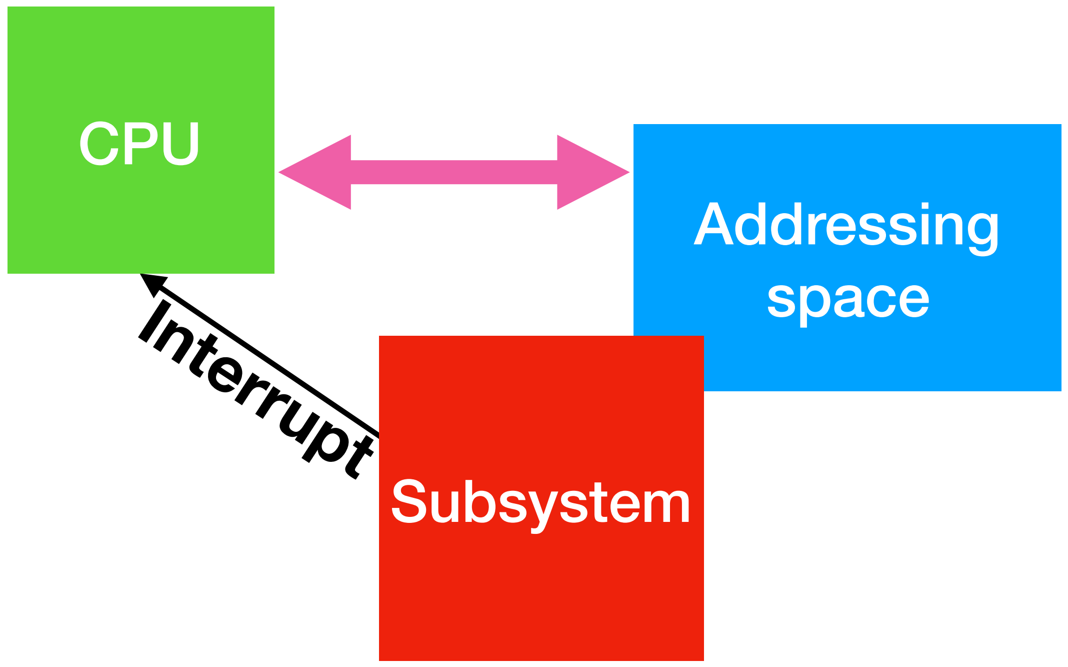
Game Boy bootstrap

- Scrolls “Nintendo” logo
- Checks if cartridge contains the same trademarked logo
- Great way to check if the initial CPU implementation *somehow* work
- Requires CPU, memory and PPU line register

Image: https://pl.wikipedia.org/wiki/Game_Boy_(konsola)#/media/File:Nintendo_Gameboy.jpg

Game Boy subsystems

- Pixel Processing Unit
- Audio Processing Unit
- Timer
- Joypad
- Memory Bank Controller
- Serial Port
- Direct Memory Access



Skeleton of a subsystem implementation

```java
public class Timer implements AddressSpace {

    public Timer(InterruptManager irq) {
        //…
    }

    public void tick() {
        //…
    }
}

public interface AddressSpace {
    boolean accepts(int address);
    void setByte(int address, int value);
    int getByte(int address);
}
```

Interrupts

- Global IME flag enables / disables interrupts
- It can modified with EI, DI, RETI
- More granular control is possible with the 0xFFFF
- Interrupt is enabled when:
  - 0xFF0F bit goes from 0 to 1
  - The same bit in 0xFFFF is 1
  - IME is enabled
- Normally 0xFF0F bits are set by hardware, but it’s possible to set them manually
- Implementation:
  - extra states for the CPU state machine
  - **InterruptManager** class as a bridge between hardware and CPU (as particular 0xFF0F bits belongs to different subsystems)

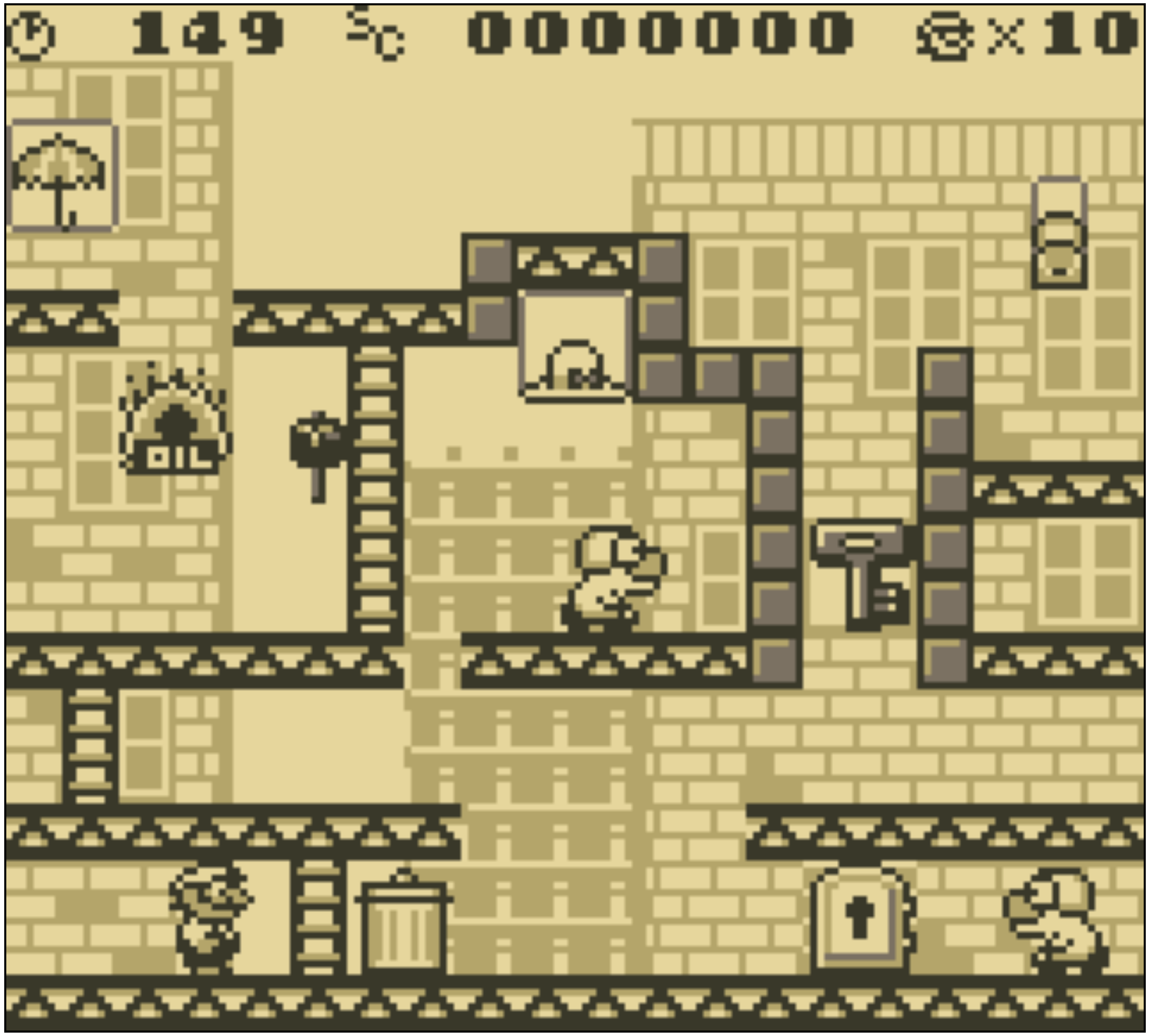
Interrupt Flag (0xFF0F)

4	Joypad	0x60
3	Serial	0x58
2	Timer	0x50
1	LCD STAT	0x48
0	V-Blank	0x40

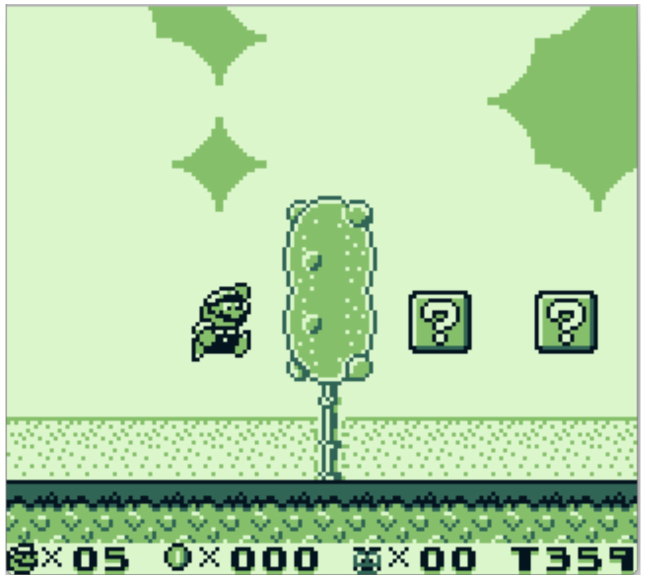
Interrupt Enable (0xFFFF)

Wild INTERRUPT appeared!

Pixel Processing Unit



Game Boy graphics



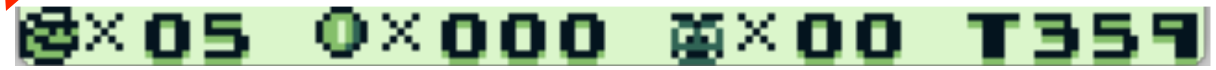
10 x

Y	X	Tile

7	Priority
6	Y flip
5	X flip
4	Palette

Sprite Attribute Table (0xFE00)

WX / WY
(0xFF4A / 0xFF4B)

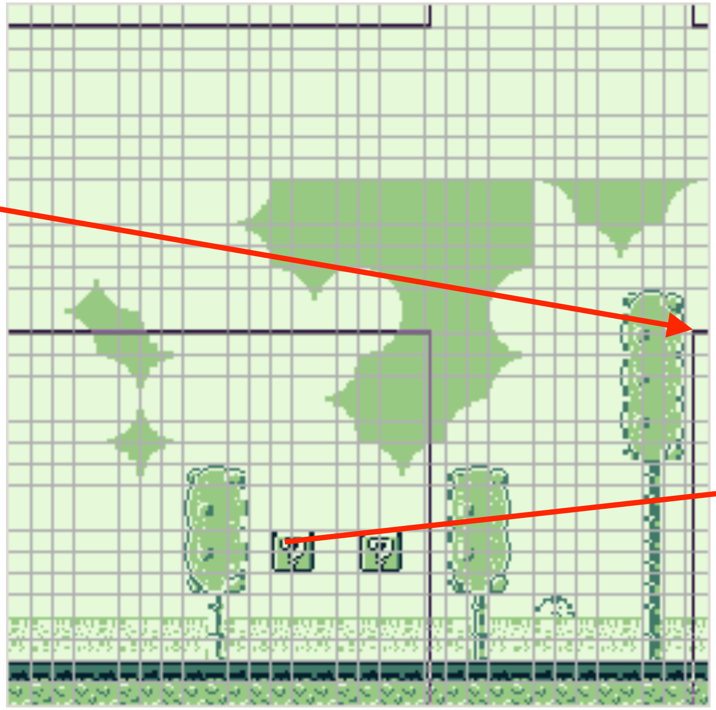


Window tile map (0x9800 / 0x9c00)

SCX / SCY
(0xFF42 / 0xFF43)

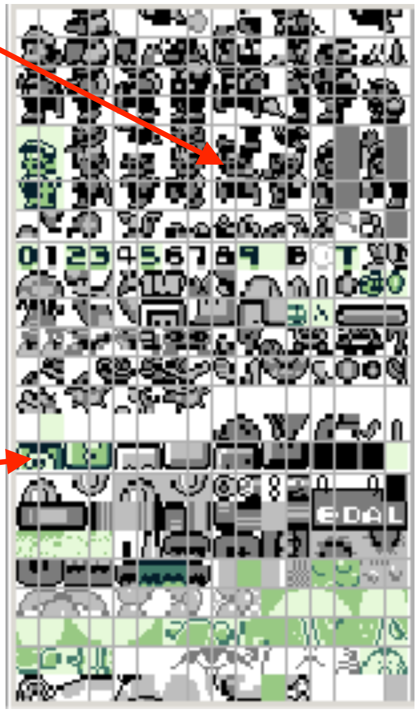
00	01	10	11

1-byte palettes:
BGP, OBP0, OBP1



Background tile map (0x9800 / 0x9C00)

Tile id: 0xF0

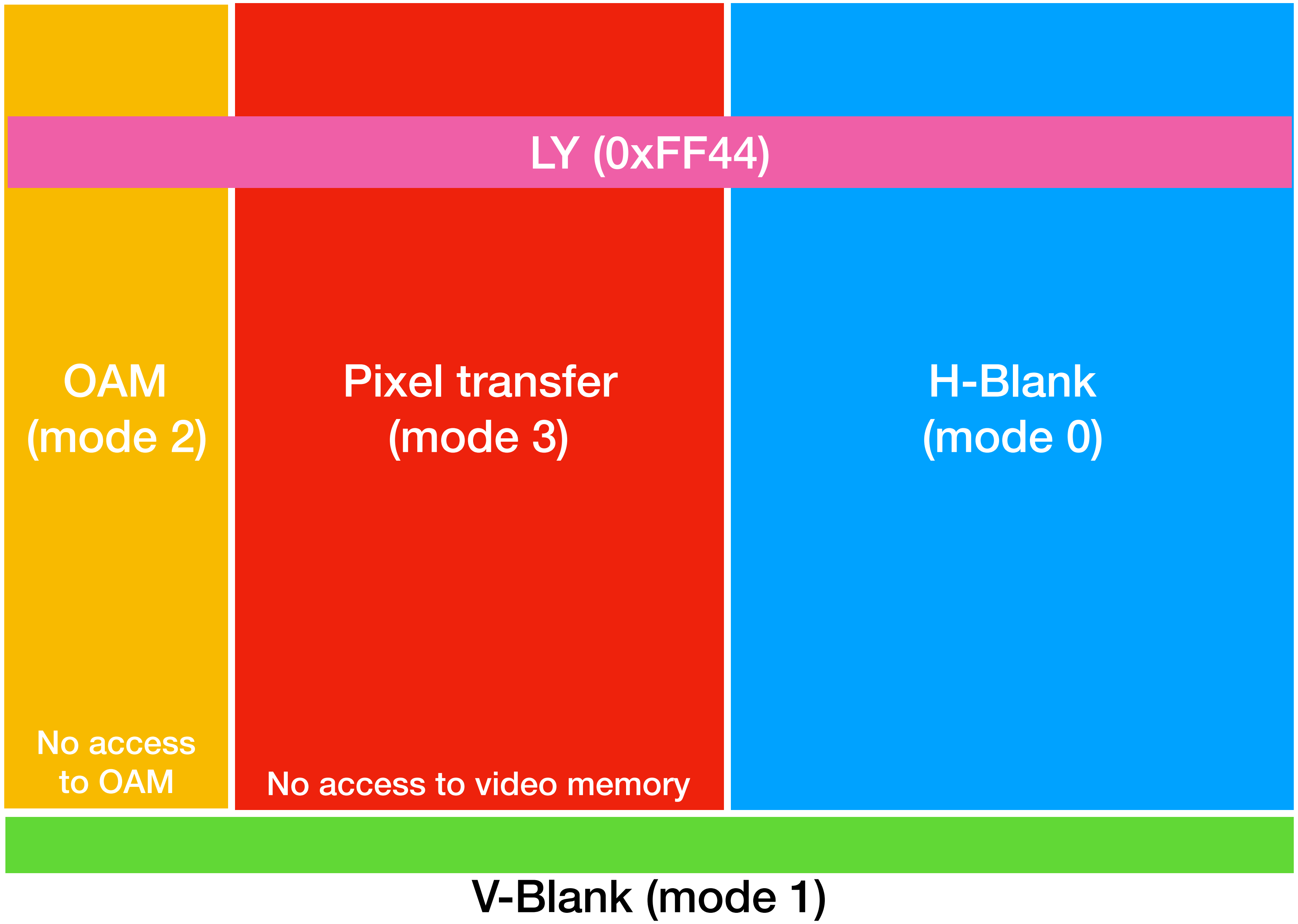


Tile data (0x8000 / 0x9000)

7	LCD enable
6	Window tile map select
5	Window display enable
4	BG & window tile data Select
3	Background tile map select
2	Sprite height (8 / 16)
1	Sprite enabled
0	BG & window display priority

LCDC (0xFF40)

Transferring data to LCD



LY (0xFF44)

OAM
(mode 2)

No access
to OAM

Pixel transfer
(mode 3)

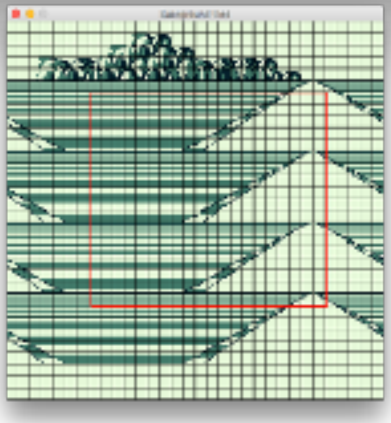
No access to video memory

H-Blank
(mode 0)

V-Blank (mode 1)

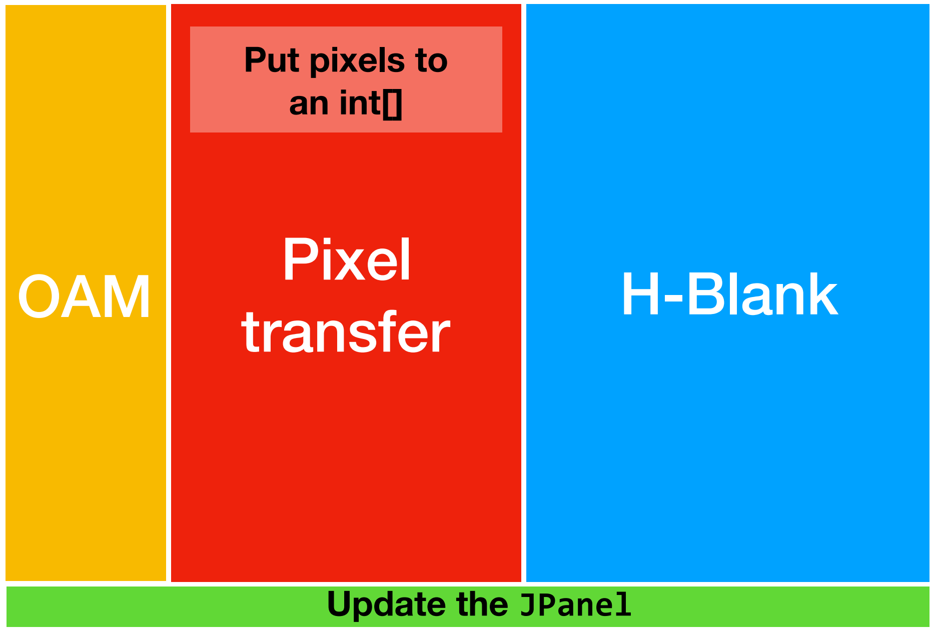
6	LYC=LY interrupt	RW
5	OAM interrupt	RW
4	V-Blank interrupt	RW
3	H-Blank interrupt	RW
2	LYC=LY?	RO
1	Current mode	RO
0		RO

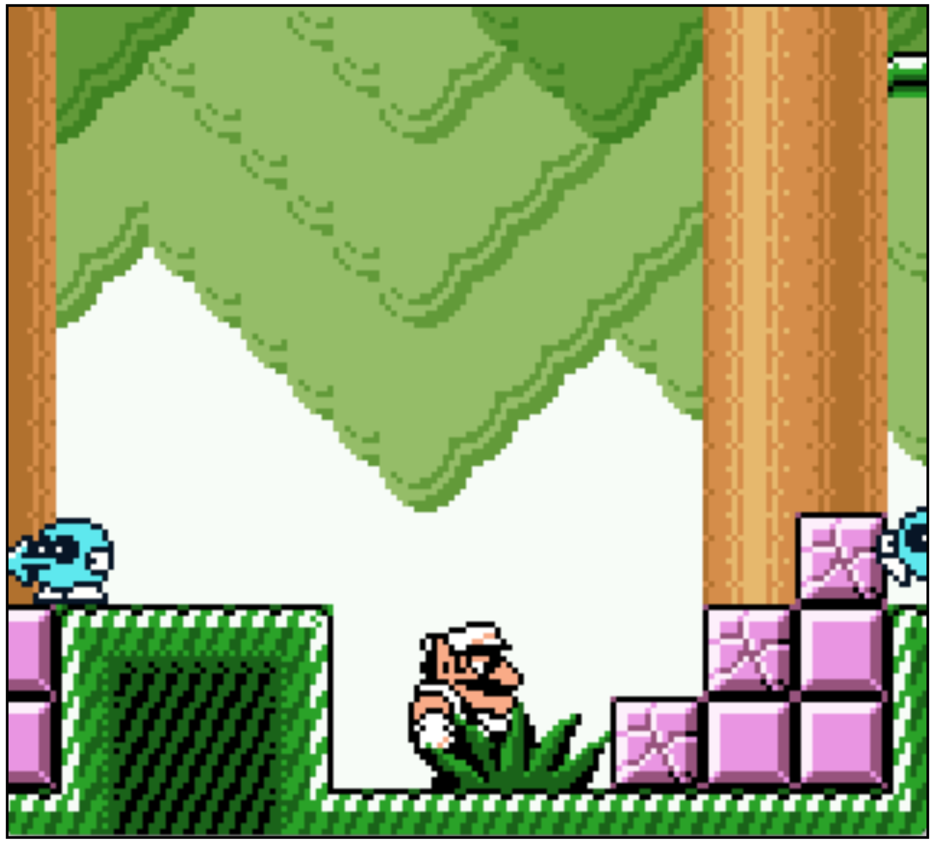
STAT (0xFF41)



PPU emulation issues

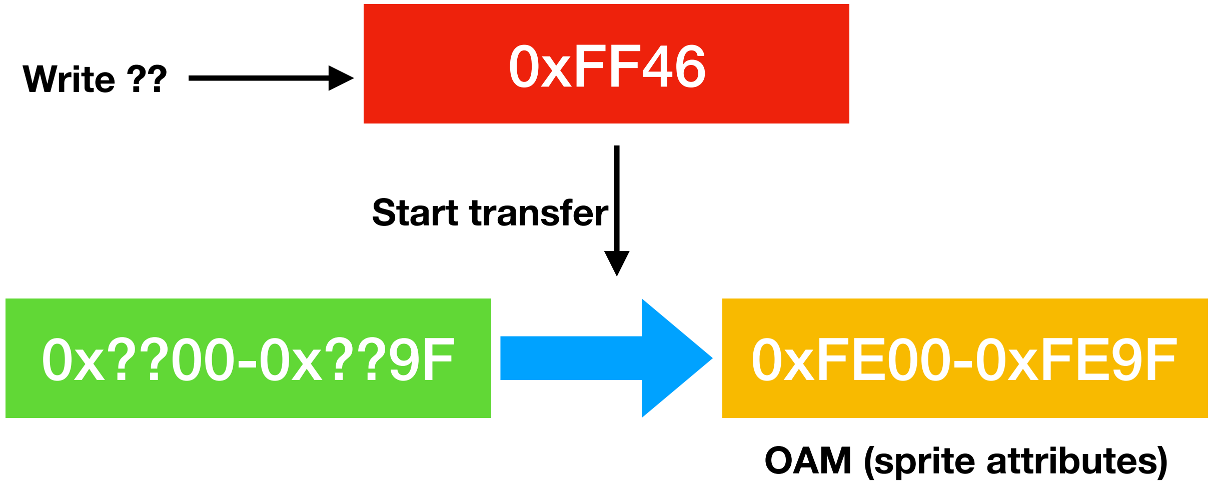
- STAT mode timing
- IRQ timing
- Performance
- Color accuracy
- Sprite RAM bug
- Complex priorities





Direct Memory Access

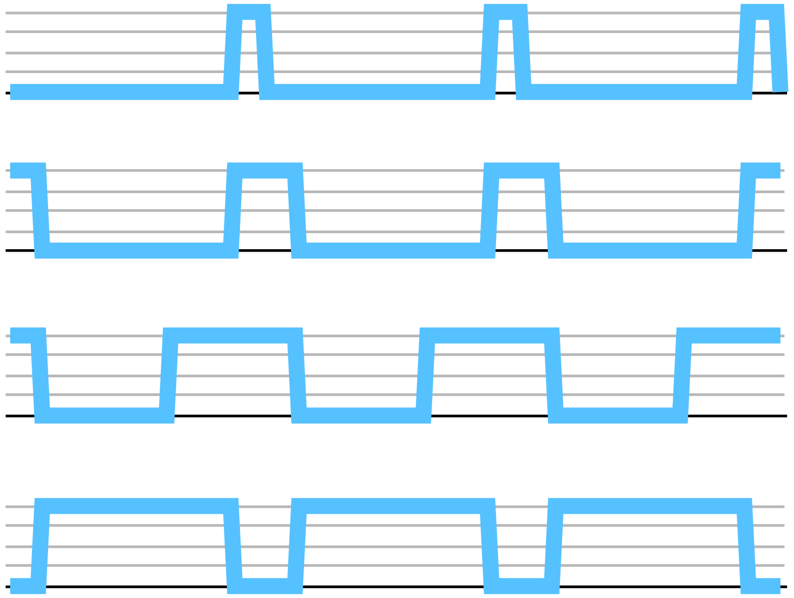
- 0xFF46 - DMA register
- Allows to copy the sprite attributes from RAM or ROM in the “background”
- Takes 648 clock cycles
- During this time CPU can only access 0xFF80-0xFFFE



Audio Processing Unit

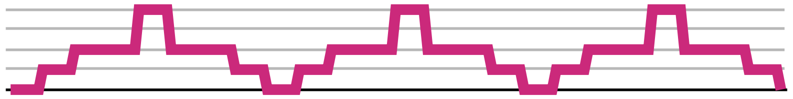
Channels 1 and 2

Square wave forms



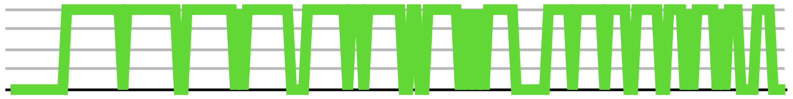
Channel 3

Custom wave form



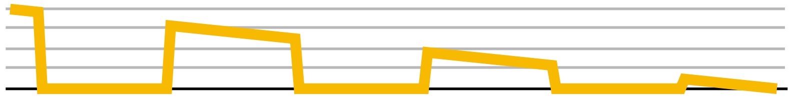
Channel 4

Random (noise)

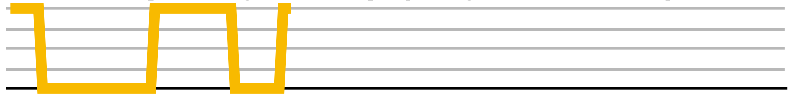


Effects

Volume envelope



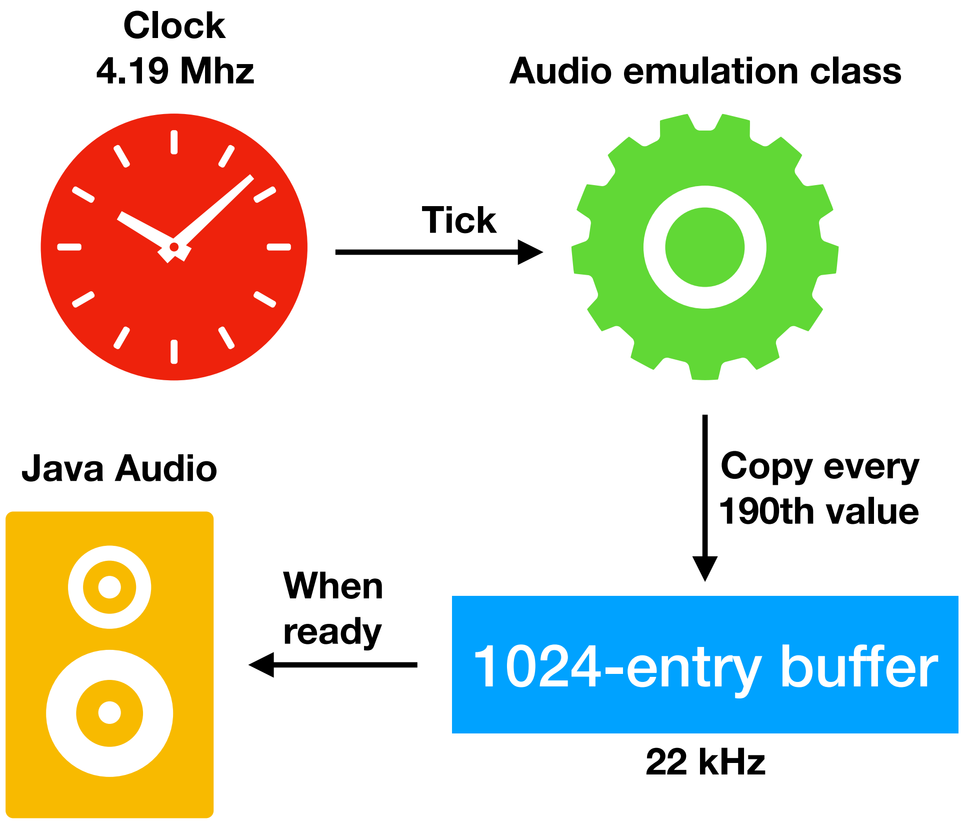
Frequency sweep (only channel 1)



- 4 channels
- each with different capabilities
- common properties:
  - length
  - volume
- registers: 0xFF10-0xFF26
- custom wave form: 32 x 4-bit
- master volume, balance
- “trigger” bit to restart the channel
- timed by the main 4.19 MHz clock

Playing the sound

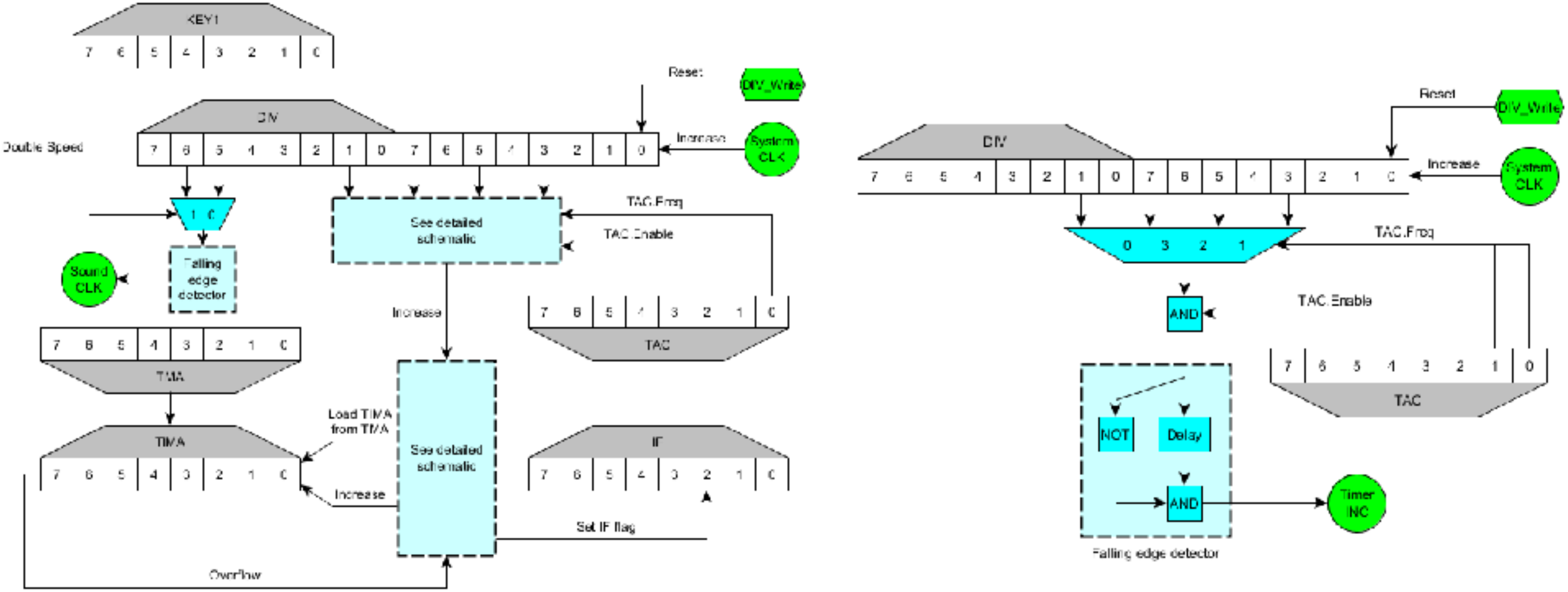
- At every *tick*, each channel provides a value 0-15
- They are mixed together (average), so the whole audio subsystem provides a single value for the channel
- The Game Boy clock speed is 4.19 MHz
- We want the emulator sampling rate to be 22 kHz
- Every (4 190 000 / 22 050) ~ 190 ticks we're adding the current sound value to the buffer
- Once the buffer is full (1024) entries, we're playing it in the main Game Boy loop as a 22 kHz sample
- It takes 0.0464 second to play the buffer - it's exactly how long it should take to run 194 583 Game Boy ticks
- We're playing sound AND synchronising the emulation - no need to extra Thread.sleep()
- The Java audio system won't allow to run the emulation too fast



Timer

- Two registers: DIV & TIME
- DIV (0xFF04) - incremented at rate 16384 Hz (clock / 256)
- TIMA (0xFF05)
    - incremented at rate specified by TAC (0xFF07) (clock / 16, 64, 256, 1024)
    - when overflowed, reset to value in TMA (0xFF06)
    - when overflowed, triggers interrupt
- Seems easy to implement, but there's a number of bugs
    - eg. writing to TAC or DIV may increase the TIMA in some cases

Timer internals



- Implementing the timer as it was designed automatically covers all the edge cases and the bug-ish behaviour

- Opposite to implementing it according to the specification and then trying to add extra ifs to implement the discovered bugs

- But we rarely have such a detailed internal documentation

http://gbdev.gg8.se/wiki/articles/Timer_Obscure_Behaviour

Joypad input

- Joypad buttons are available as 0xFF00 bits 0-3
- Writing 0 to bit 4 enables 🟦
- Writing 0 to bit 5 enables 🟩
- Joypad interrupt - mostly useless, only to resume after STOP

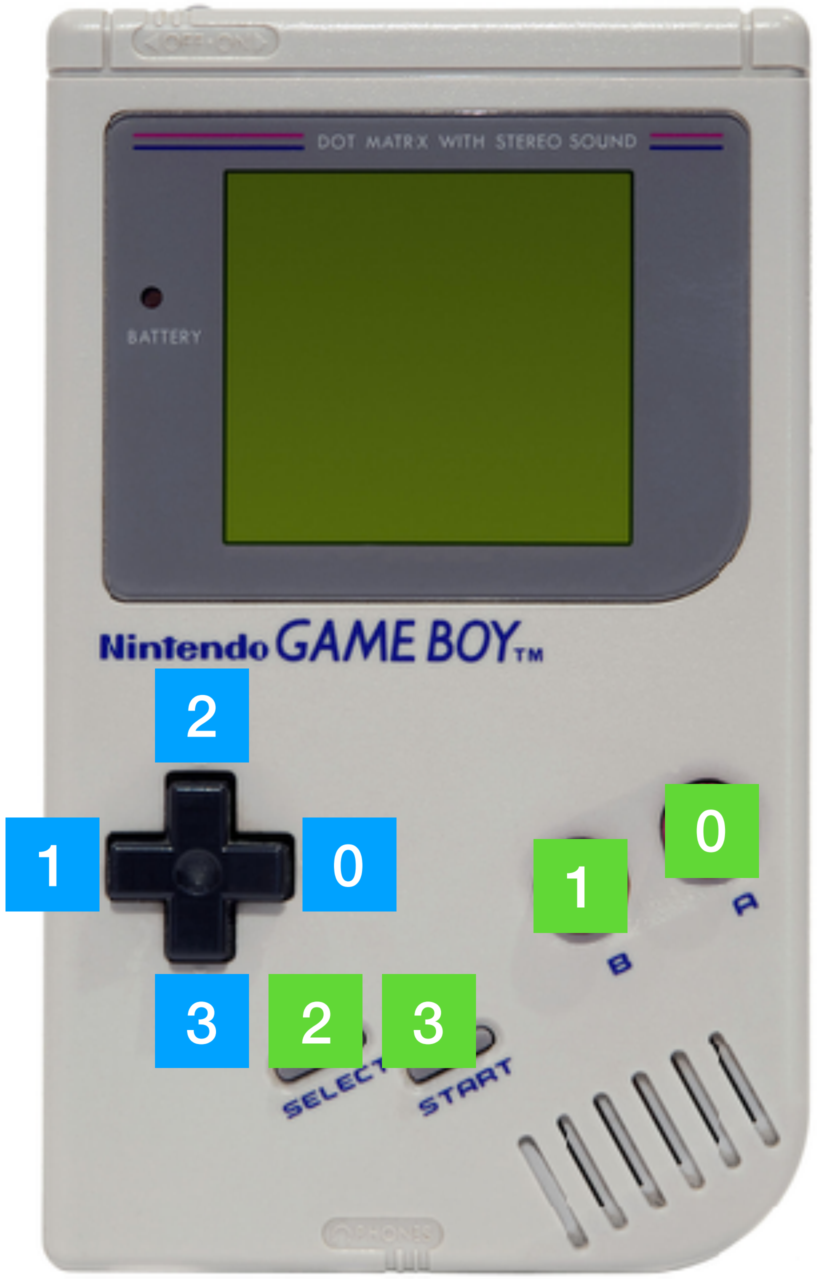
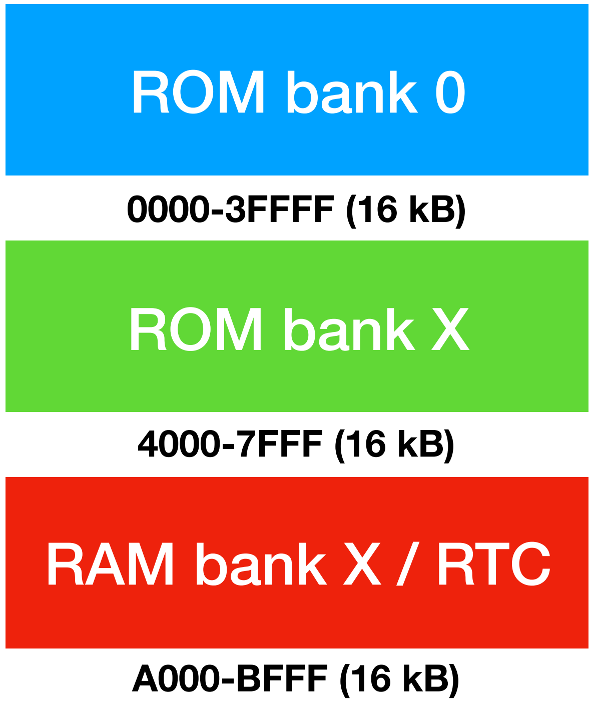


Image: https://pl.wikipedia.org/wiki/Game_Boy_(konsola)#/media/File:Nintendo_Gameboy.jpg

Memory Bank Controller

- Cartridge is available under first 48 kB
- Cartridge allows to switch ROM / RAM banks
- Battery-powered RAM is used for the save games
- A few different versions
    - MBC1 - 2 MB ROM, 32 kB RAM
    - MBC2 - 256 kB ROM, 512x4 bits RAM
    - MBC3 - 2 MB ROM, 32 kB RAM, clock
    - MBC5 - 8 MB ROM, 128 kB RAM
- Each MBC has a different semantics of switching memory banks, but usually the bank number should be written in the read-only ROM area



ROM bank 0

0000-3FFFF (16 kB)

ROM bank X

4000-7FFF (16 kB)

RAM bank X / RTC

A000-BFFF (16 kB)

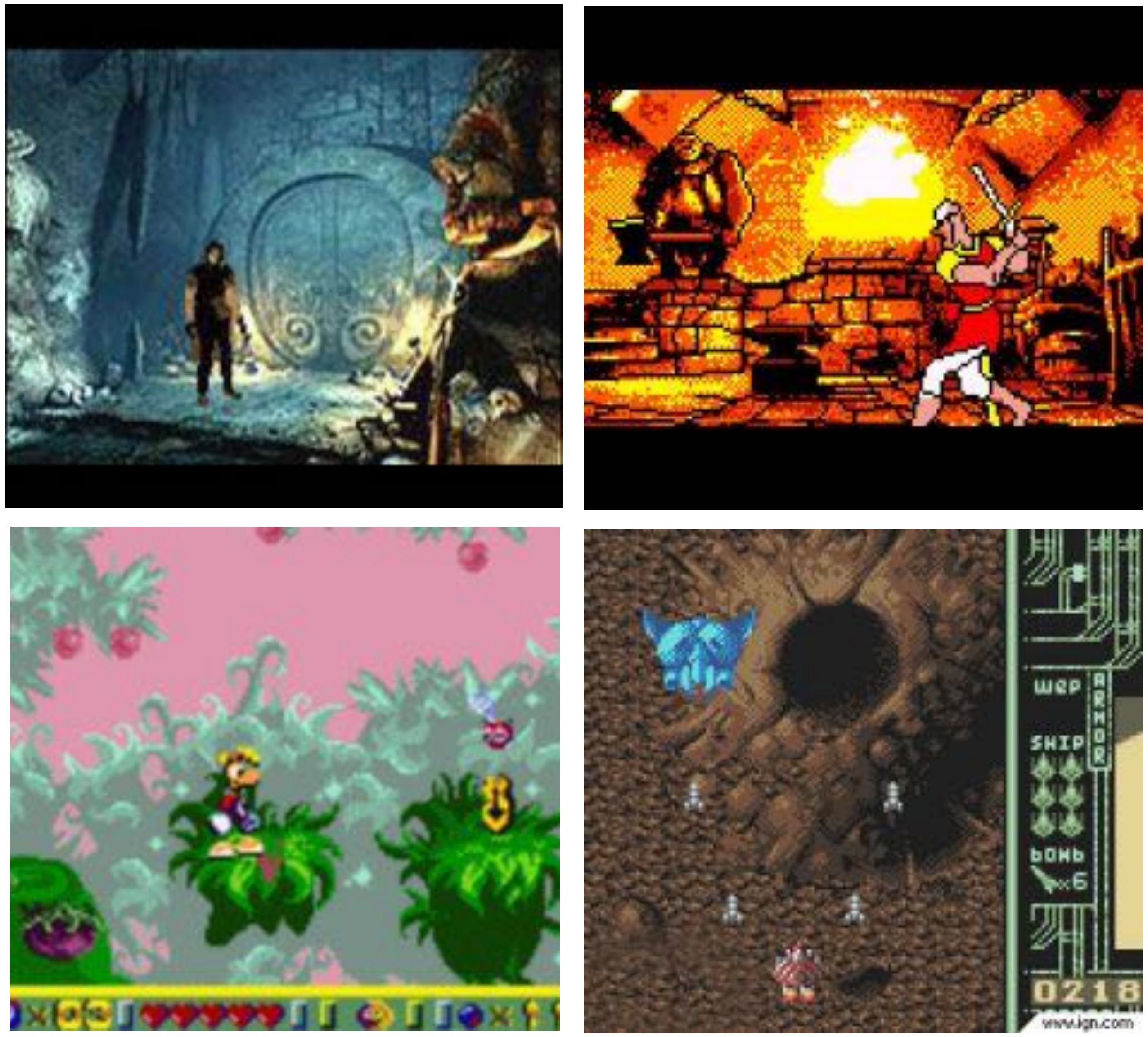
Pokemon Gold cartridge is a MBC3 with clock

Game Boy Color

- Color LCD display (32768 colors, 56 on screen)
- Double speed mode (8 MHz)
- HDMA (copy any data to VRAM)
- 28 kB of extra RAM
  - total: 32 kB
  - 7 switchable banks under 0xD000-0xDFFF
- 8 kB of extra VRAM
  - total: 16 kB

Color palettes

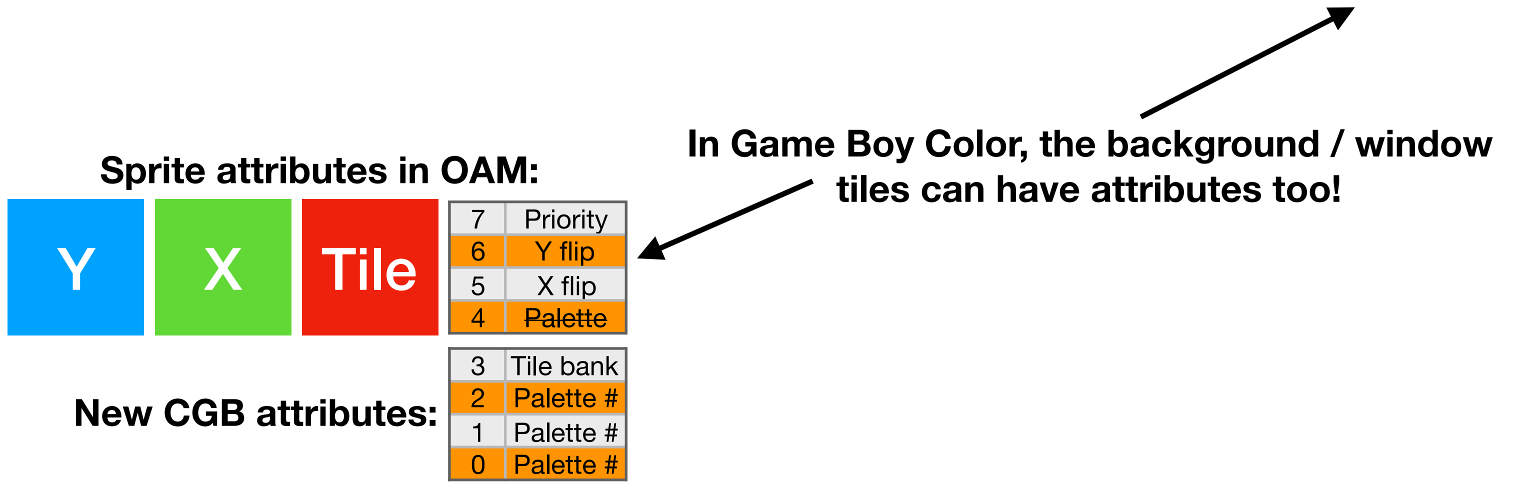
- 8 palettes for sprites and 8 for background
- Each palette: 4 colors
- Each color: 15-bit RGB
- Size of palette: 8 bytes
- Each sprite / background **tile** may choose their own palette
- Palettes can be changed between scanlines (!)



http://www.racketboy.com/retro/nintendo/gameboy/game-boy-games-that-pushed-the-limits-of-graphics-sound

Applying colors

VRAM bank 0 (as in GB classic)		VRAM bank 1	
Tile data	0x8000-0x87FF	Tile data	0x8000-0x87FF
Tile data	0x8800-0x8FFF	Tile data	0x8800-0x8FFF
Tile data	0x9000-0x97FF	Tile data	0x9000-0x97FF
Tile map	0x9800-0x9BFF	Tile attr	0x9800-0x9BFF
Tile map	0x9C00-0x9FFF	Tile attr	0x9C00-0x9FFF



Sprite attributes in OAM:

7	Priority
6	Y flip
5	X flip
4	Palette

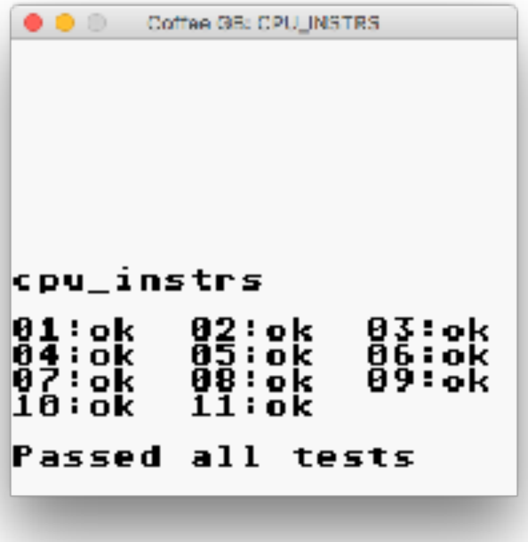
New CGB attributes:

3	Tile bank
2	Palette #
1	Palette #
0	Palette #

In Game Boy Color, the background / window tiles can have attributes too!

Testing

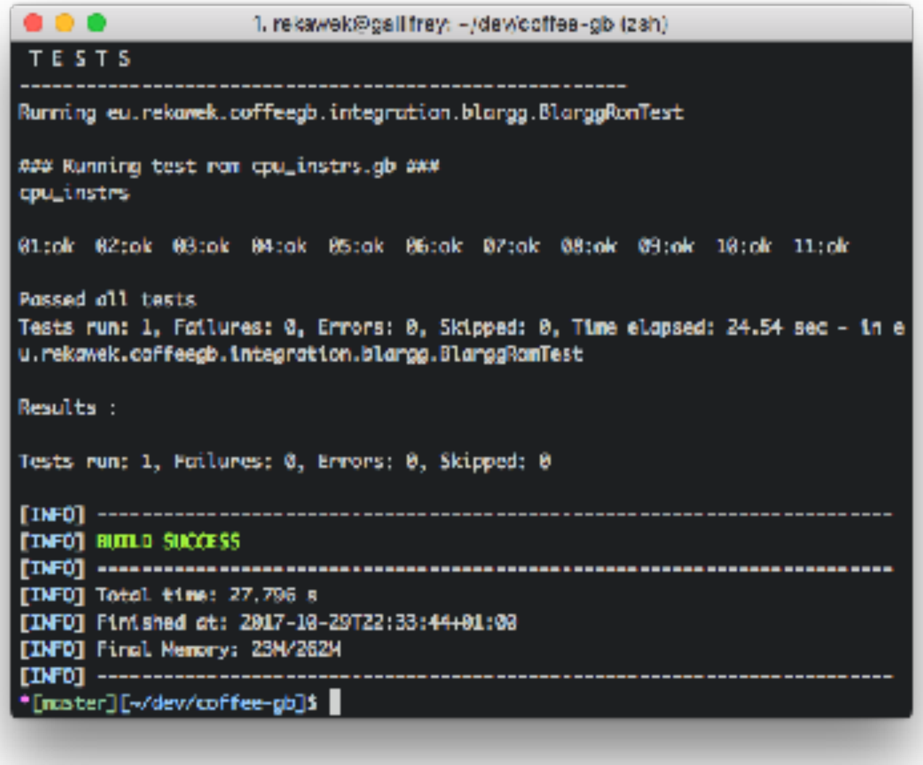
- Test ROMs:
  - blargg’s test ROMs:
    http://gbdev.gg8.se/wiki/articles/Test_ROMs
  - mooneye-gb test ROMs:
    https://github.com/Gekkio/mooneye-gb/tree/master/tests
- Coffee GB uses these for the integration testing
  - .travis.yml included
- Also: games (it’s good to have an experience from the real hardware)



```
cpu_instrs

01:ok  02:ok  03:ok
04:ok  05:ok  06:ok
07:ok  08:ok  09:ok
10:ok  11:ok

Passed all tests
```



```
 T E S T S
-------------------------------------------------------
Running eu.rekawek.coffeegb.integration.blargg.BlarggRomTest

### Running test rom cpu_instrs.gb ###
cpu_instrs

01:ok  02:ok  03:ok  04:ok  05:ok  06:ok  07:ok  08:ok  09:ok  10:ok  11:ok

Passed all tests
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 24.54 sec - in e
u.rekawek.coffeegb.integration.blargg.BlarggRomTest

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 27.796 s
[INFO] Finished at: 2017-10-29T22:33:44+01:00
[INFO] Final Memory: 23M/262M
[INFO] ------------------------------------------------------------------------
*[master][~/dev/coffee-gb]$
```

Lessons learned, plans

- Lessons learned, tips & tricks:
  - start with creating debugger
  - don’t use **byte**, use **int**
  - refactor aggressively, even prototypes should be clean
  - have automated tests
  - compare the execution with another emulator (BGB have nice debugger!)
- Plans:
  - cycle-accurate PPU implementation
  - improve the debugger
  - serial link support

Resources

- My emulator: https://github.com/trekawek/coffee-gb
- Ultimate Game Boy talk: https://www.youtube.com/watch?v=HyzD8pNlpwI
    - excellent, insightful, inspiring presentation
- Pan Docs: http://gbdev.gg8.se/wiki/articles/Pan_Docs
    - the most comprehensive description of the GB hardware
- Game Boy CPU manual: http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf
    - good but a bit inaccurate, aimed at developers
- Other accurate emulators:
    - Mooneye GB: https://github.com/Gekkio/mooneye-gb
    - Sameboy: https://sameboy.github.io/
    - BGB: http://bgb.bircd.org/

Thank you!

THANK YOU MARIO.

OH! DAISY