Game Boy w języku C#
Podstawy emulacji niskopoziomowej

Szymon Sandura
.NET Developer / Tech Lead

iteo | www.iteo.com

Zagadnienia

- Czym jest emulator?
- Emulator konsoli Nintendo Gameboy
  - Architektura
  - Przykłady implementacji
- Pokaz emulatora

Czym jest emulator?

Emulator to program lub sprzęt, który imituje działanie innego programu lub sprzętu.

Pojęcie jest dość otwarte i często się przenika z takimi pojęciami jak

- *Symulator*
- *Maszyna wirtualna*
- *Port*
- *Adapter*
- *Warstwa kompatybilności (ang. compatibility layer)*

WINE - **W**ine **I**s **N**ot **E**mulator


GAME
&
WATCH


ニンテンドークラシックミニ
FAMILY COMPUTER
ファミリーコンピュータ


Nintendo
ENTERTAINMENT
SYSTEM


SUPER FAMICOM


GAME BOY


SUPER NINTENDO
ENTERTAINMENT SYSTEM


NINTENDO 64


GAME BOY ADVANCE


NINTENDO
GAMECUBE


GAME BOY
PLAYER


e-reader


NINTENDO DS lite


Wii


new
NINTENDO 3DS XL


Wii U


NINTENDO
SWITCH


PlayStation


PocketStation


PS2
PlayStation.2


PS3
PlayStation.3


PSP
PlayStation.Portable


PSP
PlayStation.Portable


PS4


PSVITA


PS5


SEGA


Dreamcast


JAGUAR
64-BIT


SNK
NEOGEO


XBOX 360


XBOX ONE


SERIES
X|S


GAME PASS

Korzyści emulatorów konsol

- Zachowanie historii gier
- Możliwość zrozumienia gier i sprzętu
- **Usprawnienia graficzne** (płynność, rozdzielczość, wybór ekranu, mody graficzne)
- Możliwość wykorzystania **dowolnego kontrolera** (np. klawiatury), **save state'ów**, **online**

Emulacja generalnie jest legalna ale nie wolno...

- Udostępniać kopii gier (np. pliki ROM, ISO)
- Udostępniać BIOSu konsoli, kluczy deszyfrujących itp.
- Naruszać patentów, znaków towarowych
- Przyciągać uwagi Nintendo :)

Techniki emulacji?

- **LLE (Low-Level Emulation)**
- HLE (High-Level Emulation)

Obsługa CPU:

- **Interpreter**
- Kompilator (JIT, AOT)

LLE - Emulacja niskopoziomowa

- Szczegółowa implementacja architektury i komunikacji między komponentami
- Wysoka precyzja odwzorowania
- Spory narzut wydajności...

Dlaczego emulacja jest kosztowna?

- Procesor Gameboya 4Mhz ~= 4 miliony cykli na sekundę, do **miliona instrukcji na sekundę**

- Nasz procesor musi wykonać X instrukcji na każdą instrukcję, bo...
  - Nie piszemy w Assemblerze
  - Różnice w architekturze

HLE - High Level Emulation

- Skupia się na efekcie końcowym, a nie na detalach architektury i precyzji
- GPU, Dźwięk, BIOS, ale **nie CPU!**
- Pomost między natywnymi API
  - Przykład: Playstation zleciło swojemu API narysowanie trójkąta w 3D
  - Wywołujemy najbardziej zbliżoną metodę w OpenGL/Vulkan/DirectX, która rysuje trójkąt
- Potencjalnie prędkość bliska natywnej

Nintendo Game Boy

- Sprzedawany w latach 1989 - 2003 (**118,93 mln** sztuk)
- **CPU** Sharp LR35902 **4,19 Mhz**
- Dedykowany układ **Pixel Processing Unit** (**PPU**, prymitywne GPU), **8KB VRAM**
- Ekran LCD 2,5 cala **160x144px ~59,7Hz**
- Port szeregowy (**Multiplayer**...i **drukarka?**)
- **4 kanały** audio z wyjściem mono/stereo

Game Boy Printer (1998)

Game Boy Camera (1998)

- 0,1 Mpx!

CPU

- **4,19 Mhz** ~= 4 miliony cykli na sekundę
- Operacje zajmują od 4-20 cykli

- **8-bitowy**

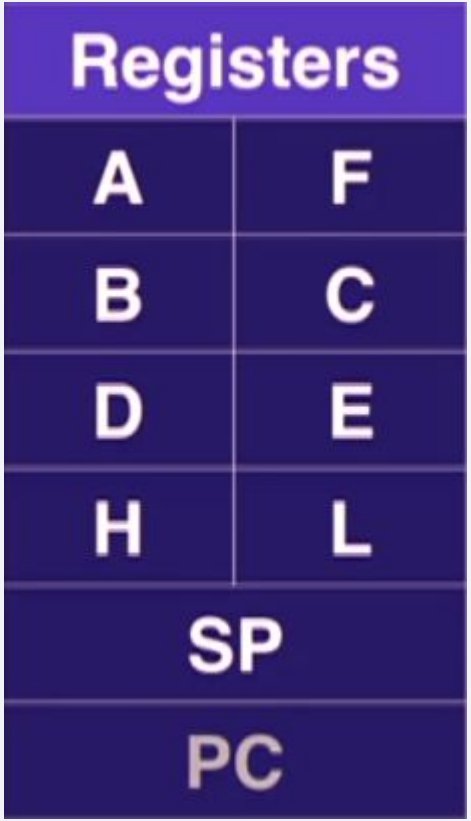
- Wspiera adresację pamięci do 64KB (indeksy od **0** do **65536**)

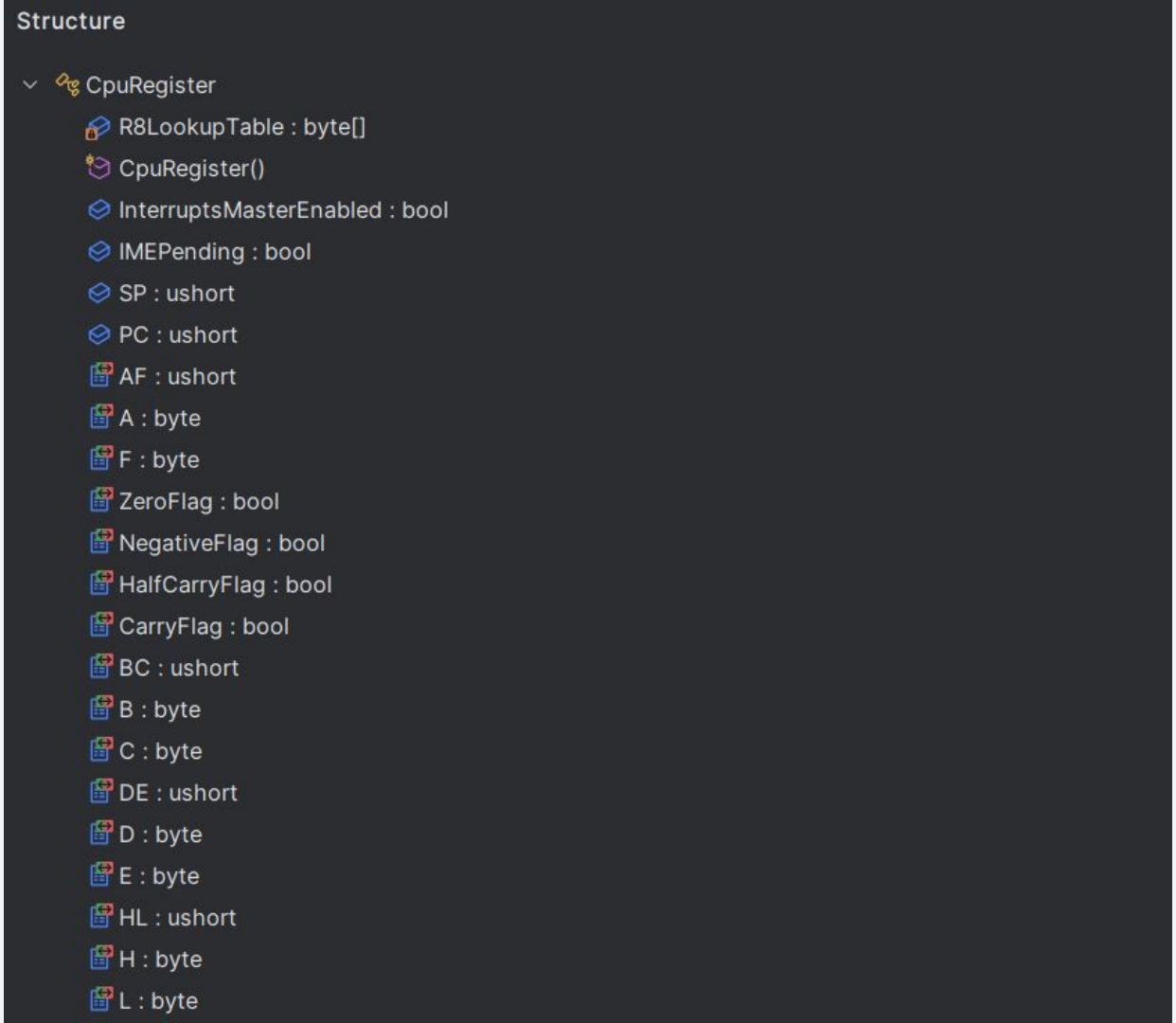
- Wspiera **przerwania** (ang. **Interrupts**)

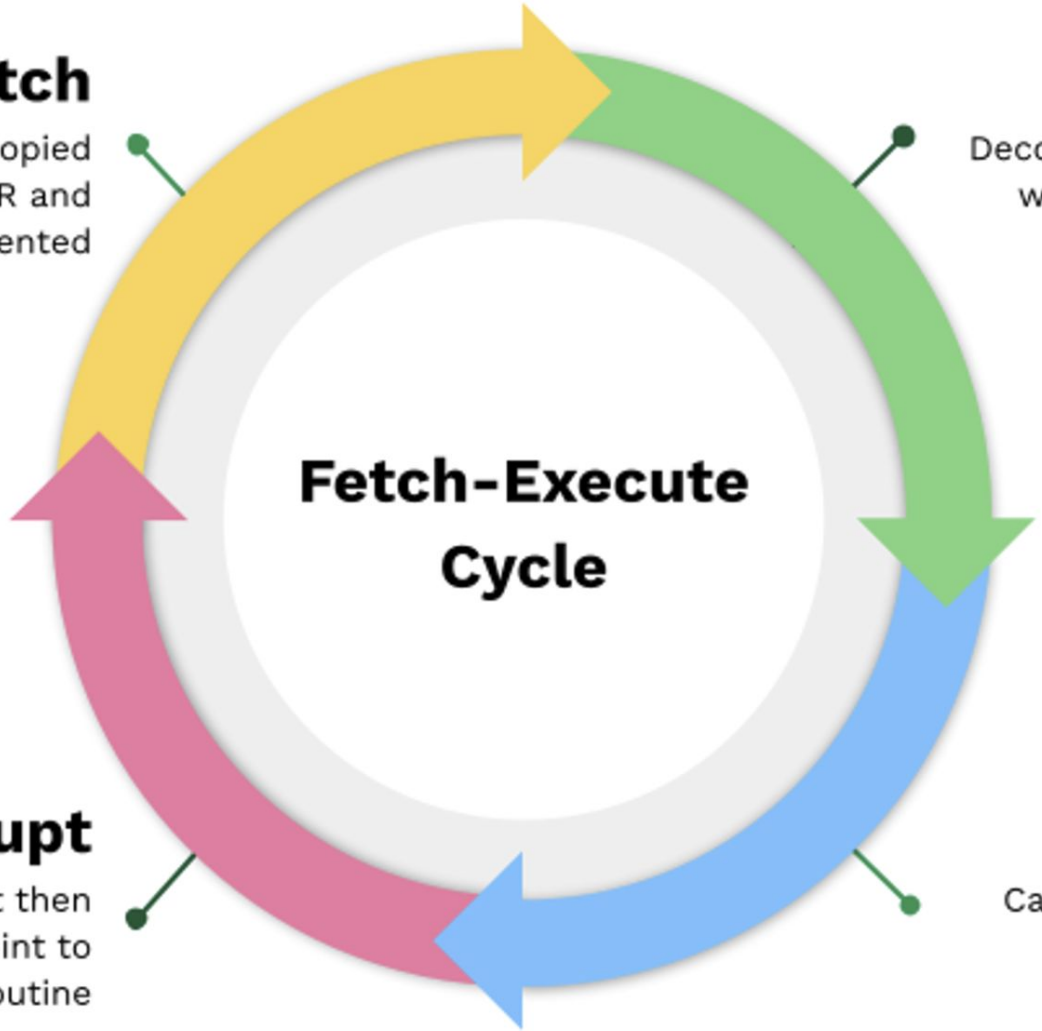
- Zawiera liczne **rejestry**

CPU - Rejestry

- **8** rejestrów 8-bitowych...
  A, F, B, C, D, E, H, L
- ...albo 4 rejestry 16-bitowe
  AF, BC, DE, HL
- SP - Stack Pointer, adres ostatniego elementu stosu (16-bit)
- **PC - Program Counter**, adres najbliższej instrukcji do wykonania (16-bit)




Structure
CpuRegister
R8LookupTable : byte[]
CpuRegister()
InterruptsMasterEnabled : bool
IMEPending : bool
SP : ushort
PC : ushort
AF : ushort
A : byte
F : byte
ZeroFlag : bool
NegativeFlag : bool
HalfCarryFlag : bool
CarryFlag : bool
BC : ushort
B : byte
C : byte
DE : ushort
D : byte
E : byte
HL : ushort
H : byte
L : byte


Fetch
The next instruction is copied from memory into the CIR and PC incremented
Decode
Decodes the instruction to work out what it means
Fetch-Execute Cycle
Interrupt
If interrupt flag set then change PC to point to Interrupt Service Routine
Execute
Carry out the instruction

```
Set8BitIncrementCarryFlags(value);
Register.SetRegisterByR8(r8, value.Add(1));
return (instructionBytesLength: 1, durationTStates: 4);
```

```
private void Set8BitIncrementCarryFlags(byte a)
{
    Register.NegativeFlag = false;
    Register.ZeroFlag = (byte)(a + 1) == 0;
    Register.HalfCarryFlag = (a & 0xF) + 1 > 0xF;
}
```

Eric's Blog

WTF is the DAA instruction?

January 30, 2018

Źródło: https://ehaskins.com/2018-01-30%20Z80%20DAA/

Ile takich metod musimy zaprogramować?

16 x 16 = 256...

	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 - - - -

...+256 = 512

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z 0 0 C	RLC C 2 8 Z 0 0 C	RLC D 2 8 Z 0 0 C	RLC E 2 8 Z 0 0 C	RLC H 2 8 Z 0 0 C	RLC L 2 8 Z 0 0 C	RLC (HL) 2 16 Z 0 0 C	RLC A 2 8 Z 0 0 C	RRC B 2 8 Z 0 0 C	RRC C 2 8 Z 0 0 C	RRC D 2 8 Z 0 0 C	RRC E 2 8 Z 0 0 C	RRC H 2 8 Z 0 0 C	RRC L 2 8 Z 0 0 C	RRC (HL) 2 16 Z 0 0 C	RRC A 2 8 Z 0 0 C
1x	RL B 2 8 Z 0 0 C	RL C 2 8 Z 0 0 C	RL D 2 8 Z 0 0 C	RL E 2 8 Z 0 0 C	RL H 2 8 Z 0 0 C	RL L 2 8 Z 0 0 C	RL (HL) 2 16 Z 0 0 C	RL A 2 8 Z 0 0 C	RR B 2 8 Z 0 0 C	RR C 2 8 Z 0 0 C	RR D 2 8 Z 0 0 C	RR E 2 8 Z 0 0 C	RR H 2 8 Z 0 0 C	RR L 2 8 Z 0 0 C	RR (HL) 2 16 Z 0 0 C	RR A 2 8 Z 0 0 C
2x	SLA B 2 8 Z 0 0 C	SLA C 2 8 Z 0 0 C	SLA D 2 8 Z 0 0 C	SLA E 2 8 Z 0 0 C	SLA H 2 8 Z 0 0 C	SLA L 2 8 Z 0 0 C	SLA (HL) 2 16 Z 0 0 C	SLA A 2 8 Z 0 0 C	SRA B 2 8 Z 0 0 0	SRA C 2 8 Z 0 0 0	SRA D 2 8 Z 0 0 0	SRA E 2 8 Z 0 0 0	SRA H 2 8 Z 0 0 0	SRA L 2 8 Z 0 0 0	SRA (HL) 2 16 Z 0 0 0	SRA A 2 8 Z 0 0 0
3x	SWAP B 2 8 Z 0 0 0	SWAP C 2 8 Z 0 0 0	SWAP D 2 8 Z 0 0 0	SWAP E 2 8 Z 0 0 0	SWAP H 2 8 Z 0 0 0	SWAP L 2 8 Z 0 0 0	SWAP (HL) 2 16 Z 0 0 0	SWAP A 2 8 Z 0 0 0	SRL B 2 8 Z 0 0 C	SRL C 2 8 Z 0 0 C	SRL D 2 8 Z 0 0 C	SRL E 2 8 Z 0 0 C	SRL H 2 8 Z 0 0 C	SRL L 2 8 Z 0 0 C	SRL (HL) 2 16 Z 0 0 C	SRL A 2 8 Z 0 0 C
4x	BIT 0,B 2 8 Z 0 1 -	BIT 0,C 2 8 Z 0 1 -	BIT 0,D 2 8 Z 0 1 -	BIT 0,E 2 8 Z 0 1 -	BIT 0,H 2 8 Z 0 1 -	BIT 0,L 2 8 Z 0 1 -	BIT 0,(HL) 2 16 Z 0 1 -	BIT 0,A 2 8 Z 0 1 -	BIT 1,B 2 8 Z 0 1 -	BIT 1,C 2 8 Z 0 1 -	BIT 1,D 2 8 Z 0 1 -	BIT 1,E 2 8 Z 0 1 -	BIT 1,H 2 8 Z 0 1 -	BIT 1,L 2 8 Z 0 1 -	BIT 1,(HL) 2 16 Z 0 1 -	BIT 1,A 2 8 Z 0 1 -
5x	BIT 2,B 2 8 Z 0 1 -	BIT 2,C 2 8 Z 0 1 -	BIT 2,D 2 8 Z 0 1 -	BIT 2,E 2 8 Z 0 1 -	BIT 2,H 2 8 Z 0 1 -	BIT 2,L 2 8 Z 0 1 -	BIT 2,(HL) 2 16 Z 0 1 -	BIT 2,A 2 8 Z 0 1 -	BIT 3,B 2 8 Z 0 1 -	BIT 3,C 2 8 Z 0 1 -	BIT 3,D 2 8 Z 0 1 -	BIT 3,E 2 8 Z 0 1 -	BIT 3,H 2 8 Z 0 1 -	BIT 3,L 2 8 Z 0 1 -	BIT 3,(HL) 2 16 Z 0 1 -	BIT 3,A 2 8 Z 0 1 -
6x	BIT 4,B 2 8 Z 0 1 -	BIT 4,C 2 8 Z 0 1 -	BIT 4,D 2 8 Z 0 1 -	BIT 4,E 2 8 Z 0 1 -	BIT 4,H 2 8 Z 0 1 -	BIT 4,L 2 8 Z 0 1 -	BIT 4,(HL) 2 16 Z 0 1 -	BIT 4,A 2 8 Z 0 1 -	BIT 5,B 2 8 Z 0 1 -	BIT 5,C 2 8 Z 0 1 -	BIT 5,D 2 8 Z 0 1 -	BIT 5,E 2 8 Z 0 1 -	BIT 5,H 2 8 Z 0 1 -	BIT 5,L 2 8 Z 0 1 -	BIT 5,(HL) 2 16 Z 0 1 -	BIT 5,A 2 8 Z 0 1 -
7x	BIT 6,B 2 8 Z 0 1 -	BIT 6,C 2 8 Z 0 1 -	BIT 6,D 2 8 Z 0 1 -	BIT 6,E 2 8 Z 0 1 -	BIT 6,H 2 8 Z 0 1 -	BIT 6,L 2 8 Z 0 1 -	BIT 6,(HL) 2 16 Z 0 1 -	BIT 6,A 2 8 Z 0 1 -	BIT 7,B 2 8 Z 0 1 -	BIT 7,C 2 8 Z 0 1 -	BIT 7,D 2 8 Z 0 1 -	BIT 7,E 2 8 Z 0 1 -	BIT 7,H 2 8 Z 0 1 -	BIT 7,L 2 8 Z 0 1 -	BIT 7,(HL) 2 16 Z 0 1 -	BIT 7,A 2 8 Z 0 1 -
8x	RES 0,B 2 8 - - - -	RES 0,C 2 8 - - - -	RES 0,D 2 8 - - - -	RES 0,E 2 8 - - - -	RES 0,H 2 8 - - - -	RES 0,L 2 8 - - - -	RES 0,(HL) 2 16 - - - -	RES 0,A 2 8 - - - -	RES 1,B 2 8 - - - -	RES 1,C 2 8 - - - -	RES 1,D 2 8 - - - -	RES 1,E 2 8 - - - -	RES 1,H 2 8 - - - -	RES 1,L 2 8 - - - -	RES 1,(HL) 2 16 - - - -	RES 1,A 2 8 - - - -
9x	RES 2,B 2 8 - - - -	RES 2,C 2 8 - - - -	RES 2,D 2 8 - - - -	RES 2,E 2 8 - - - -	RES 2,H 2 8 - - - -	RES 2,L 2 8 - - - -	RES 2,(HL) 2 16 - - - -	RES 2,A 2 8 - - - -	RES 3,B 2 8 - - - -	RES 3,C 2 8 - - - -	RES 3,D 2 8 - - - -	RES 3,E 2 8 - - - -	RES 3,H 2 8 - - - -	RES 3,L 2 8 - - - -	RES 3,(HL) 2 16 - - - -	RES 3,A 2 8 - - - -
Ax	RES 4,B 2 8 - - - -	RES 4,C 2 8 - - - -	RES 4,D 2 8 - - - -	RES 4,E 2 8 - - - -	RES 4,H 2 8 - - - -	RES 4,L 2 8 - - - -	RES 4,(HL) 2 16 - - - -	RES 4,A 2 8 - - - -	RES 5,B 2 8 - - - -	RES 5,C 2 8 - - - -	RES 5,D 2 8 - - - -	RES 5,E 2 8 - - - -	RES 5,H 2 8 - - - -	RES 5,L 2 8 - - - -	RES 5,(HL) 2 16 - - - -	RES 5,A 2 8 - - - -
Bx	RES 6,B 2 8 - - - -	RES 6,C 2 8 - - - -	RES 6,D 2 8 - - - -	RES 6,E 2 8 - - - -	RES 6,H 2 8 - - - -	RES 6,L 2 8 - - - -	RES 6,(HL) 2 16 - - - -	RES 6,A 2 8 - - - -	RES 7,B 2 8 - - - -	RES 7,C 2 8 - - - -	RES 7,D 2 8 - - - -	RES 7,E 2 8 - - - -	RES 7,H 2 8 - - - -	RES 7,L 2 8 - - - -	RES 7,(HL) 2 16 - - - -	RES 7,A 2 8 - - - -
Cx	SET 0,B 2 8 - - - -	SET 0,C 2 8 - - - -	SET 0,D 2 8 - - - -	SET 0,E 2 8 - - - -	SET 0,H 2 8 - - - -	SET 0,L 2 8 - - - -	SET 0,(HL) 2 16 - - - -	SET 0,A 2 8 - - - -	SET 1,B 2 8 - - - -	SET 1,C 2 8 - - - -	SET 1,D 2 8 - - - -	SET 1,E 2 8 - - - -	SET 1,H 2 8 - - - -	SET 1,L 2 8 - - - -	SET 1,(HL) 2 16 - - - -	SET 1,A 2 8 - - - -
Dx	SET 2,B 2 8 - - - -	SET 2,C 2 8 - - - -	SET 2,D 2 8 - - - -	SET 2,E 2 8 - - - -	SET 2,H 2 8 - - - -	SET 2,L 2 8 - - - -	SET 2,(HL) 2 16 - - - -	SET 2,A 2 8 - - - -	SET 3,B 2 8 - - - -	SET 3,C 2 8 - - - -	SET 3,D 2 8 - - - -	SET 3,E 2 8 - - - -	SET 3,H 2 8 - - - -	SET 3,L 2 8 - - - -	SET 3,(HL) 2 16 - - - -	SET 3,A 2 8 - - - -
Ex	SET 4,B 2 8 - - - -	SET 4,C 2 8 - - - -	SET 4,D 2 8 - - - -	SET 4,E 2 8 - - - -	SET 4,H 2 8 - - - -	SET 4,L 2 8 - - - -	SET 4,(HL) 2 16 - - - -	SET 4,A 2 8 - - - -	SET 5,B 2 8 - - - -	SET 5,C 2 8 - - - -	SET 5,D 2 8 - - - -	SET 5,E 2 8 - - - -	SET 5,H 2 8 - - - -	SET 5,L 2 8 - - - -	SET 5,(HL) 2 16 - - - -	SET 5,A 2 8 - - - -
Fx	SET 6,B 2 8 - - - -	SET 6,C 2 8 - - - -	SET 6,D 2 8 - - - -	SET 6,E 2 8 - - - -	SET 6,H 2 8 - - - -	SET 6,L 2 8 - - - -	SET 6,(HL) 2 16 - - - -	SET 6,A 2 8 - - - -	SET 7,B 2 8 - - - -	SET 7,C 2 8 - - - -	SET 7,D 2 8 - - - -	SET 7,E 2 8 - - - -	SET 7,H 2 8 - - - -	SET 7,L 2 8 - - - -	SET 7,(HL) 2 16 - - - -	SET 7,A 2 8 - - - -

Supported Core Features

Registers

A	F
B	C
D	E
H	L
SP	
PC	

r8

a (hl)
b c
d e
h l

r16

bc
de
hl
sp

Load/Store

```
ld r8, d8
ld r8, r8
ld a, (bc)
ld a, (de)
ld a, (a16)*
ld (bc), a
ld (de), a
ld (a16), a*
ld r16, d16
```

Stack

```
ld sp, hl
push r16
pop r16
```

Artih/Logical

```
add  | a, d8
adc  | a, r8
sub  |
sbc  |
and  |
xor  |
or   |
cp   |
inc  | r8
dec  |
daa
cpl
add hl, r16
inc r16
dec r16
```

Rotate

```
rlca
rla
rrca
rra
```

Control Flow

```
jp a16
jp hl
jp f, a16
call a16
call f, a16
ret
ret f
rst n
```

Misc

```
ccf
scf
nop
halt
di
ei
```

* different opcode than 8080

```
public byte ExecuteNextOperation()
{
    if (IsHalted)
    {
        return 1;
    }

    var opCode = MemoryController.ReadByte(address: Register.PC);
    var operationBlock = (opCode & 0b11000000) >> 6;

    var operationSize = operationBlock switch
    {
        0x0 => ExecuteBlock0(ref opCode),
        0x1 => ExecuteBlock1(ref opCode),
        0x2 => ExecuteBlock2(ref opCode),
        0x3 => ExecuteBlock3(ref opCode),
        _ => throw new NotImplementedException($"Operation {opCode:X2} not implemented")
    };

    if (Register.IMEPending && opCode != 0xFB)
    {
        Register.IMEPending = false;
        Register.InterruptsMasterEnabled = true;
    }

    Register.PC += operationSize.instructionBytesLength;

    return (byte)(operationSize.durationTStates + (interruptHandled ? 5 : 0));
}
```

CPU - Przerwania

- **“Zdarzenia”** obsługiwane jak najszybciej to możliwe

- Gameboy posiada zamkniętą listę 5 przerwań
  - 1 związane z Joypadem (**klasyk**)
  - 2 związane z cyklem życia LCD i PPU
  - 1 z timerem
  - 1 z portem szeregowym

- Obsługę zdarzenia definiuje twórca programu umieszczając swoją procedurę pod jednym z adresów: 64, 72, 80, 88, 96

Liczniki (ang. Timers) **- TIMA, DIV**

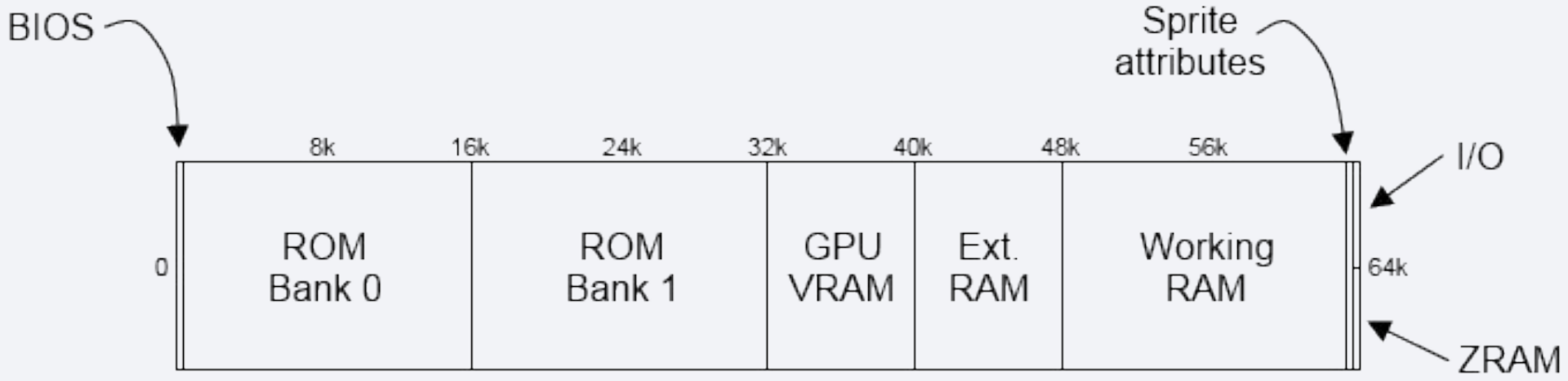
DIV Timer

```
public class DivTimer(MemoryController memoryController)
{
    private int _dividerCycleCounter;

    1 usage
    internal void CheckAndIncrementTimer(ref byte tStates)
    {
        _dividerCycleCounter += tStates;

        if (_dividerCycleCounter >= Cycles.DividerCycles)
        {
            _dividerCycleCounter -= Cycles.DividerCycles;
            memoryController.IncrementByte(Constants.DIVRegister);
        }
    }
```

Mapowanie pamięci

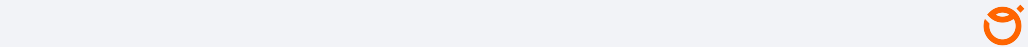


Źródło: https://gbdev.io/pandocs/Memory_Map.html

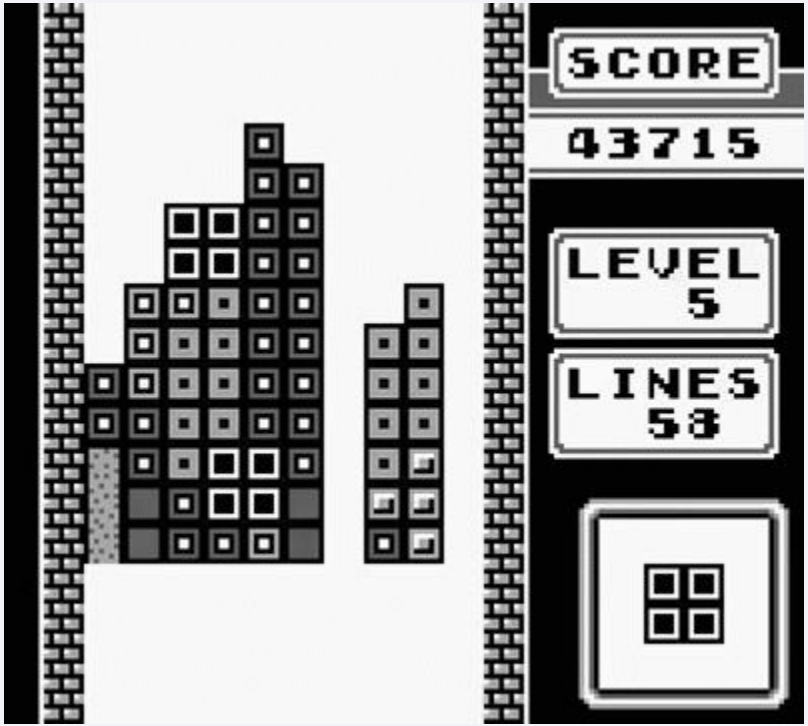
Mapowanie pamięci

```
public byte[] MemorySpace = new byte[65535]; //64 KB
```

Źródło: https://gbdev.io/pandocs/Memory_Map.html



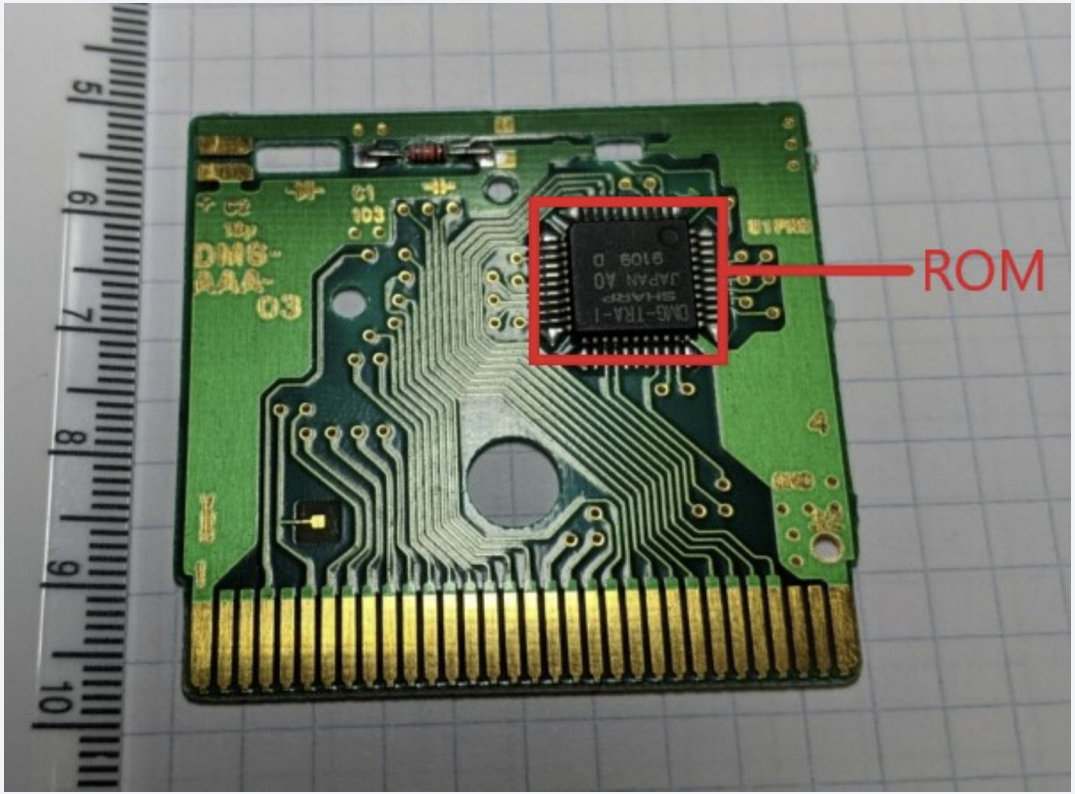
Tetris



Pokemon Red

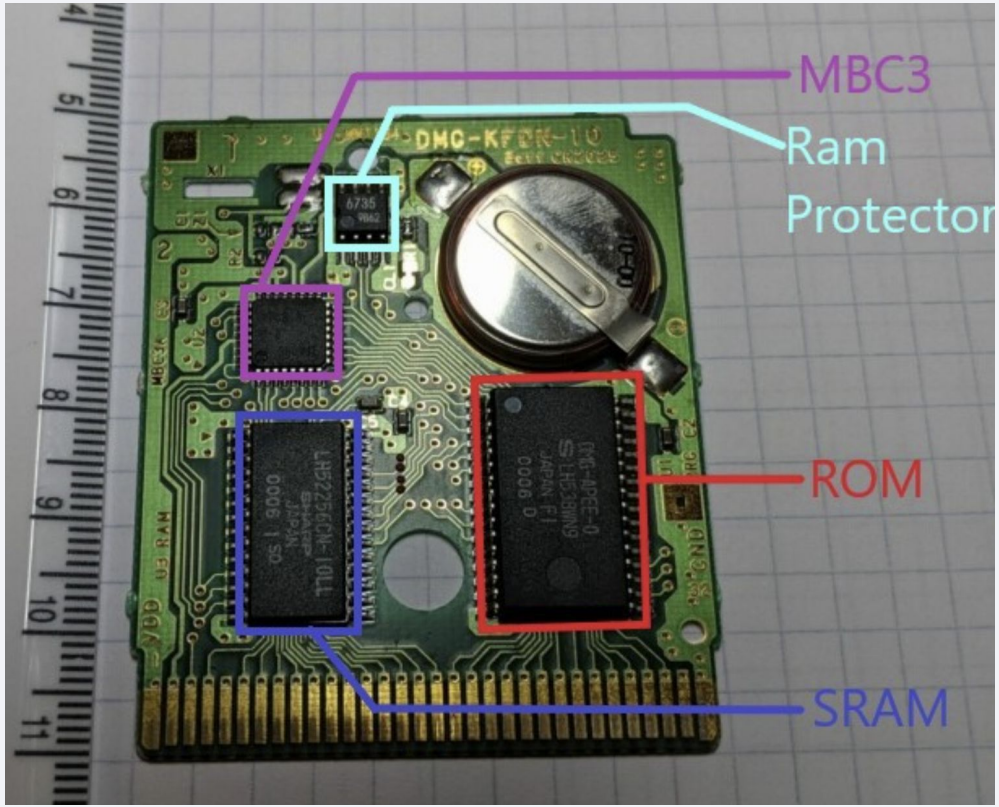

Exception Unhandled
System.ArgumentOutOfRangeException: 'Index was out of range. Must be non-negative and less than the size of the collection. (Parameter 'index')'
© Tutlane.com
Show Call Stack | View Details | Copy Details | Start Live Share session...
Exception Settings

Tak wygląda Tetris (32 KB)...



Źródło: https://b13rg.github.io/Gameboy-Bank-Switching/

Ale Pokemony Red (2 MB) już tak...



Źródło: https://b13rg.github.io/Gameboy-Bank-Switching/

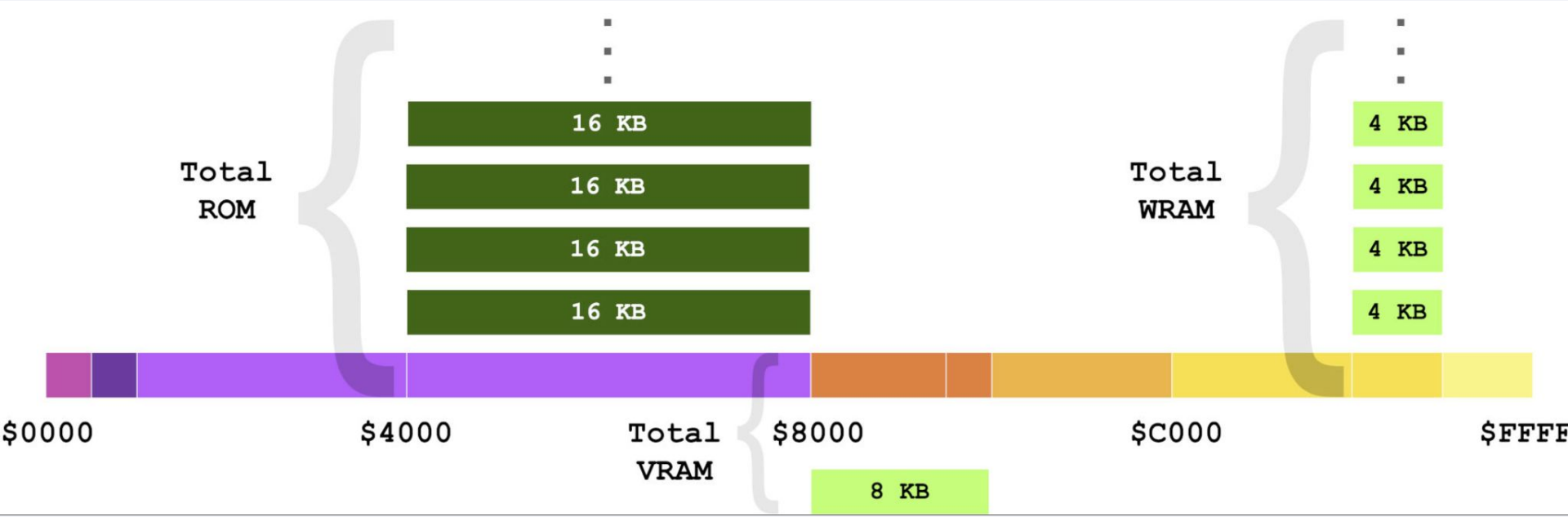
MBC - Memory Bank Controller

- **MBC** pozwala na obsługę **wielu banków pamięci** na tym samym adresie

- Dodatkowe funkcje:
  - sterowanie **zegarem czasu rzeczywistego**
  - **zapis save'ów**
  - obsługa urządzeń (kamera, czujnik ruchu, wibracje)

RTC - Zegar czasu rzeczywistego

Mapowanie pamięci, ale tym razem z MBC



Źródło: https://gbdev.io/pandocs/Memory_Map.html

Mapowanie pamięci w kodzie

```
private void InitializeMemoryMap()
{
    for (int i = 0; i <= 65535; i++)
    {
        _memoryMap[i] = i switch
        {
            <= BankAddress.RomBankNnEnd => RomBankNn,
            <= BankAddress.VramEnd => Vram,
            <= BankAddress.ExternalRamEnd => RomBankNn,
            <= BankAddress.Wram0End => Wram0,
            <= BankAddress.Wram1End => Wram1,
            <= 0xEFFF => Wram0, //Echo of Wram0
            <= 0xFDFF => Wram1, //Echo of Wram1
            <= BankAddress.OamEnd => Oam,
            <= BankAddress.NotUsableEnd => NotUsable,
            <= BankAddress.IoRegistersEnd => IoRegisters,
            <= BankAddress.HRamEnd => HRam,
            _ => InterruptEnableRegister
        };
    }
}
```

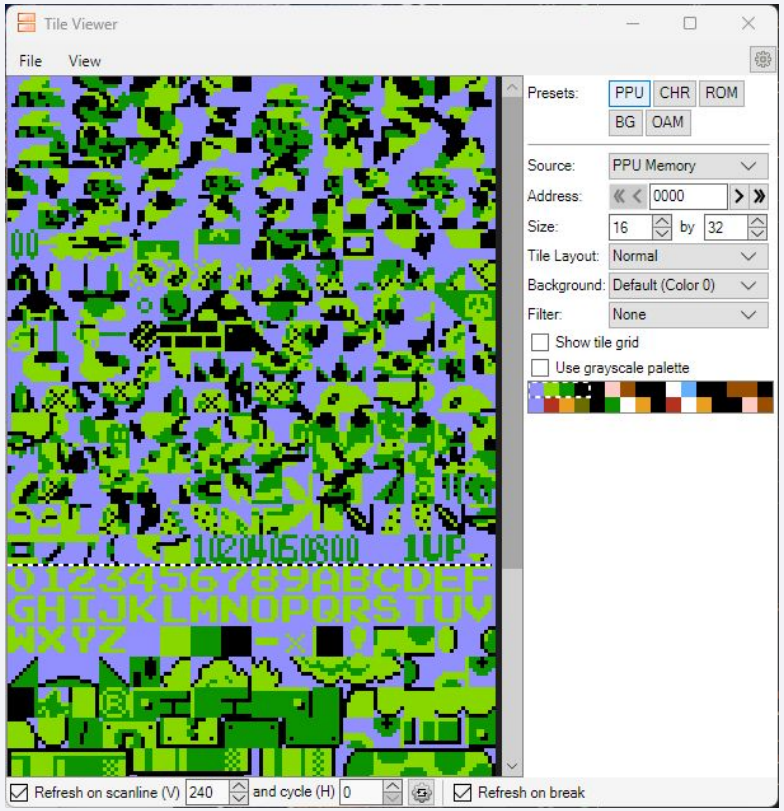
PPU - Pixel Processing Unit

- **Renderuje** z pamięci VRAM **kafelki (ang. Tiles)**, a nie pojedyncze piksele
  - **1 Tile = 8x8 pikseli**
  - Oszczędność pamięci **VRAM!**

- Rysuje 3 typy rzeczy na ekranie
  - **Background** (tło)
  - **Window** (okno)
  - **Objects** (obiekty częściej **Sprites**)

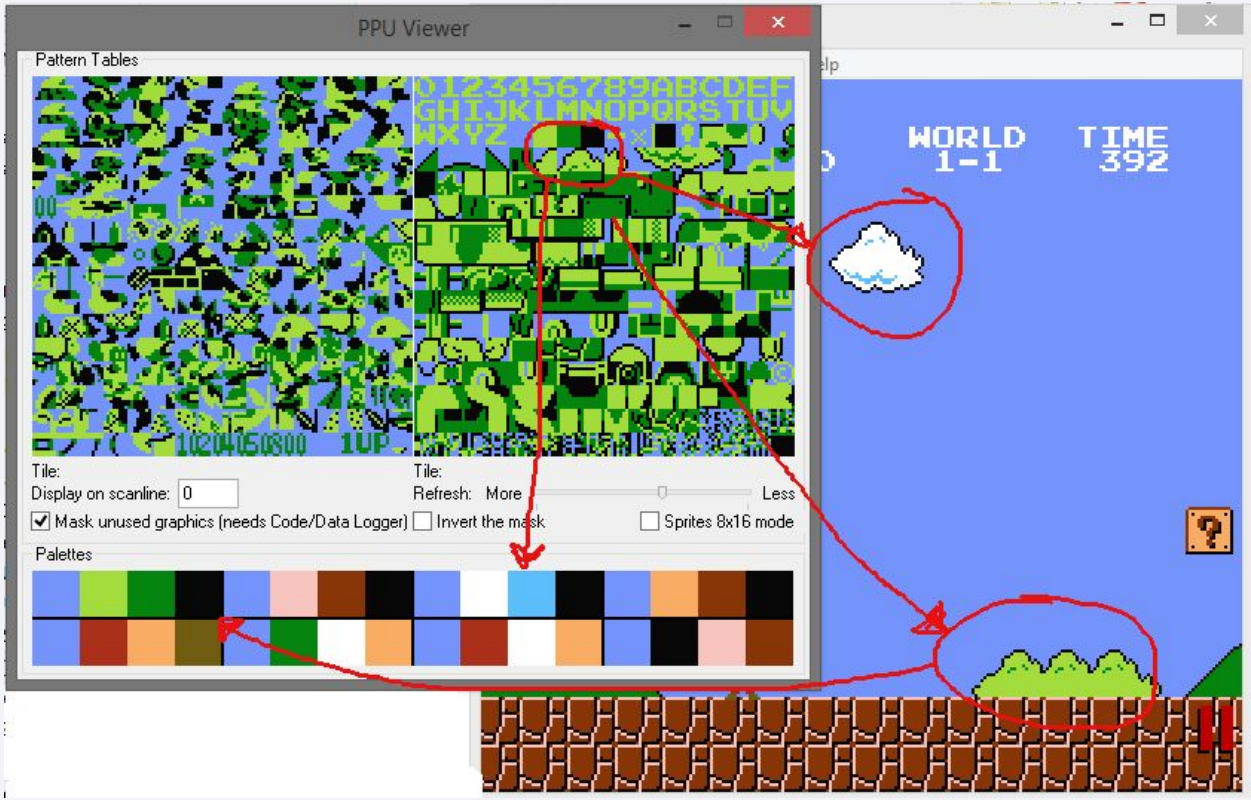
- **Rysowanie** odbywa się **rząd po rzędzie**!
  144 linii (ang. **scanlines**) na jedną klatkę

Tiles / Kafelki zamiast pikseli



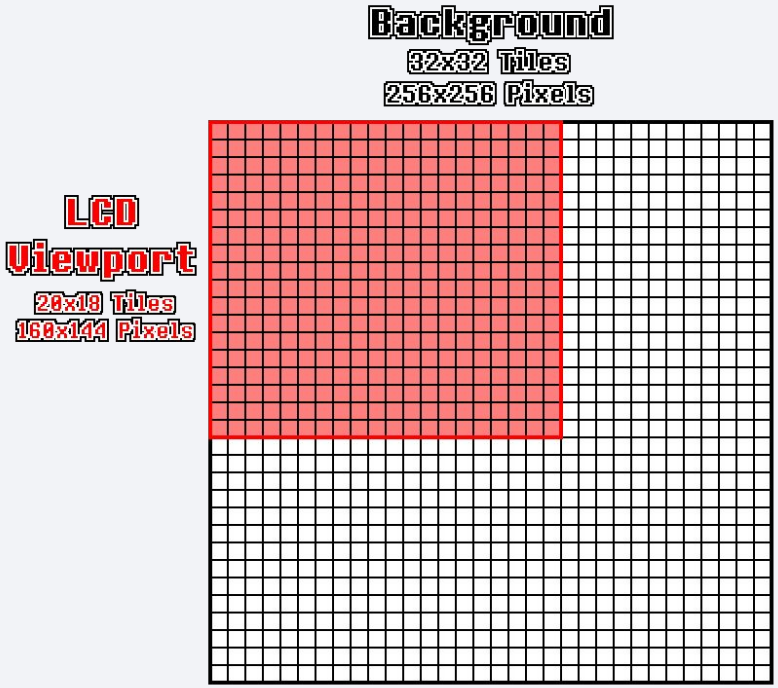
Źródło: https://nesdoug.com/

Super Mario na konsoli NES/Pegasus



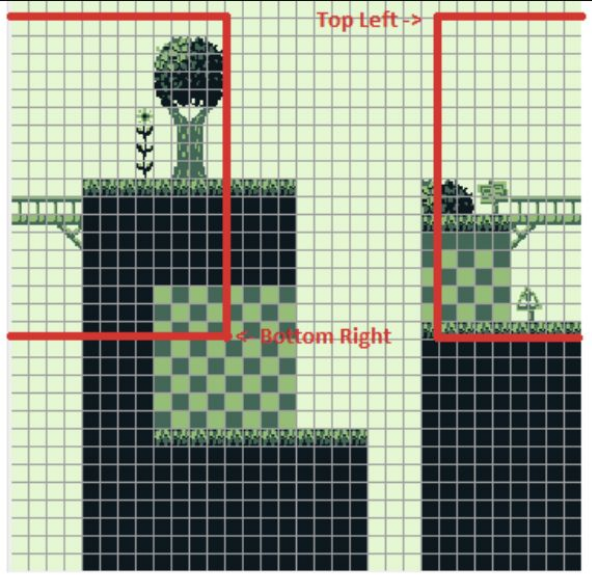
Źródło: https://nesdoug.com/

Rysowanie tła (Background)



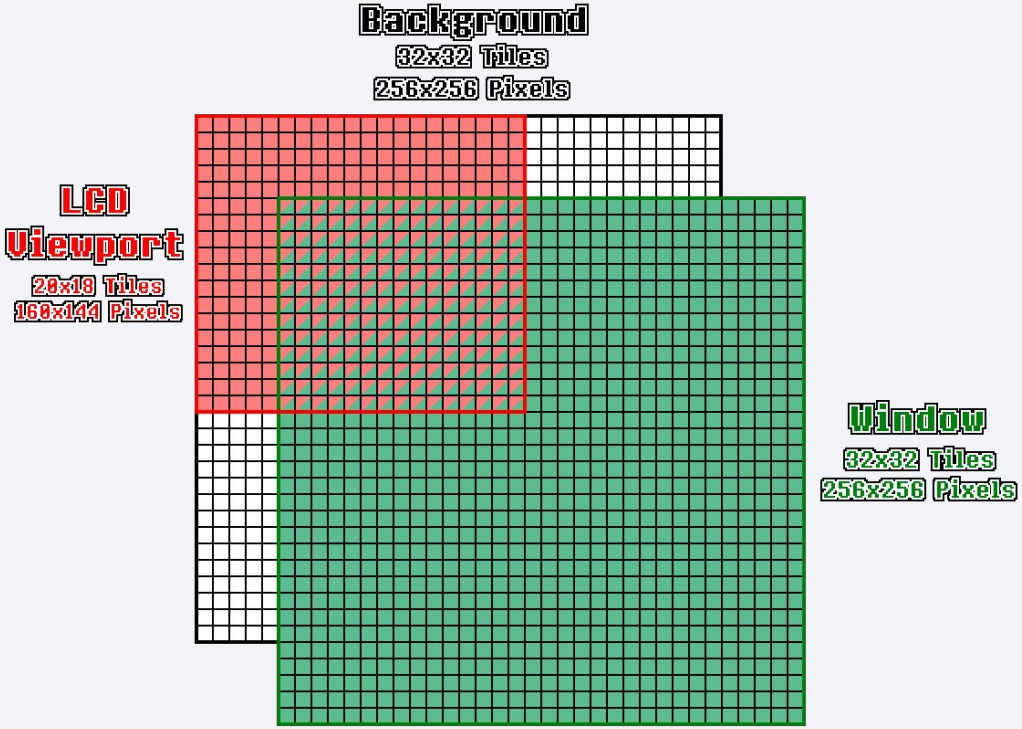
Źródło: https://hacktix.github.io/GBEDG/ppu/

Background Tilemap:



Output:

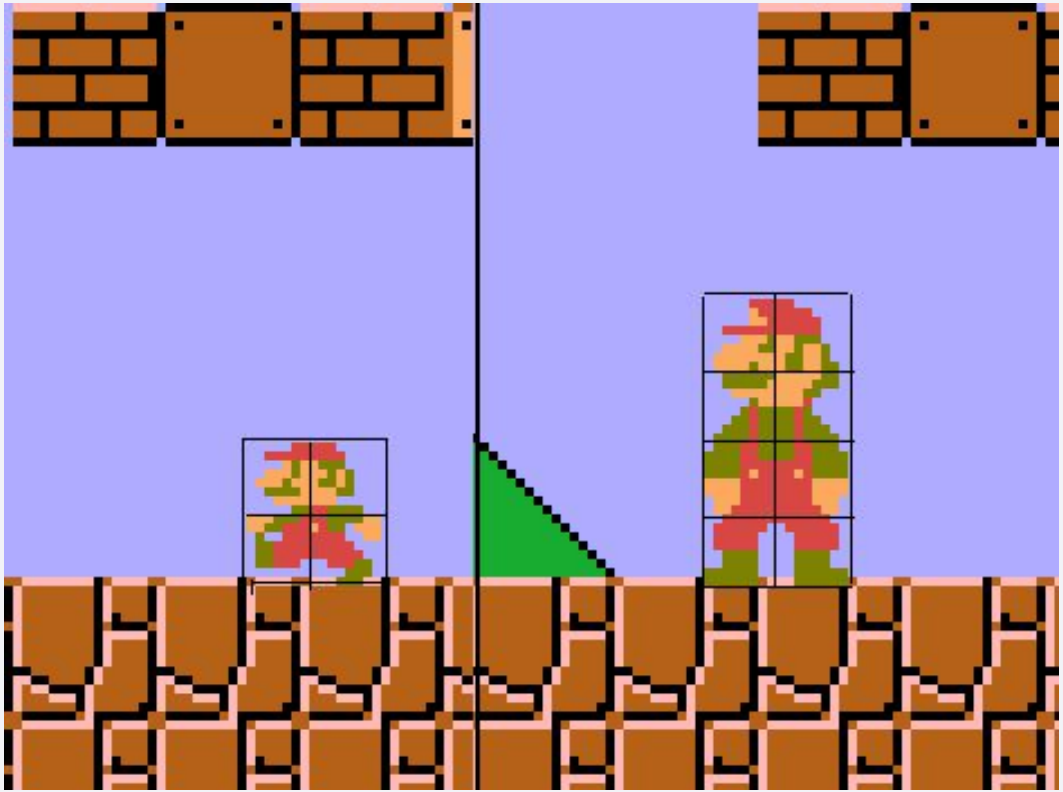
Rysowanie okna (**Window**)



Źródło: https://hacktix.github.io/GBEDG/ppu/


NIDORAN♂
:L3
HP:
PIKACHU
:L50
HP:
135/135
PIKACHU
used THUNDERBOLT!

Sprites / Objects

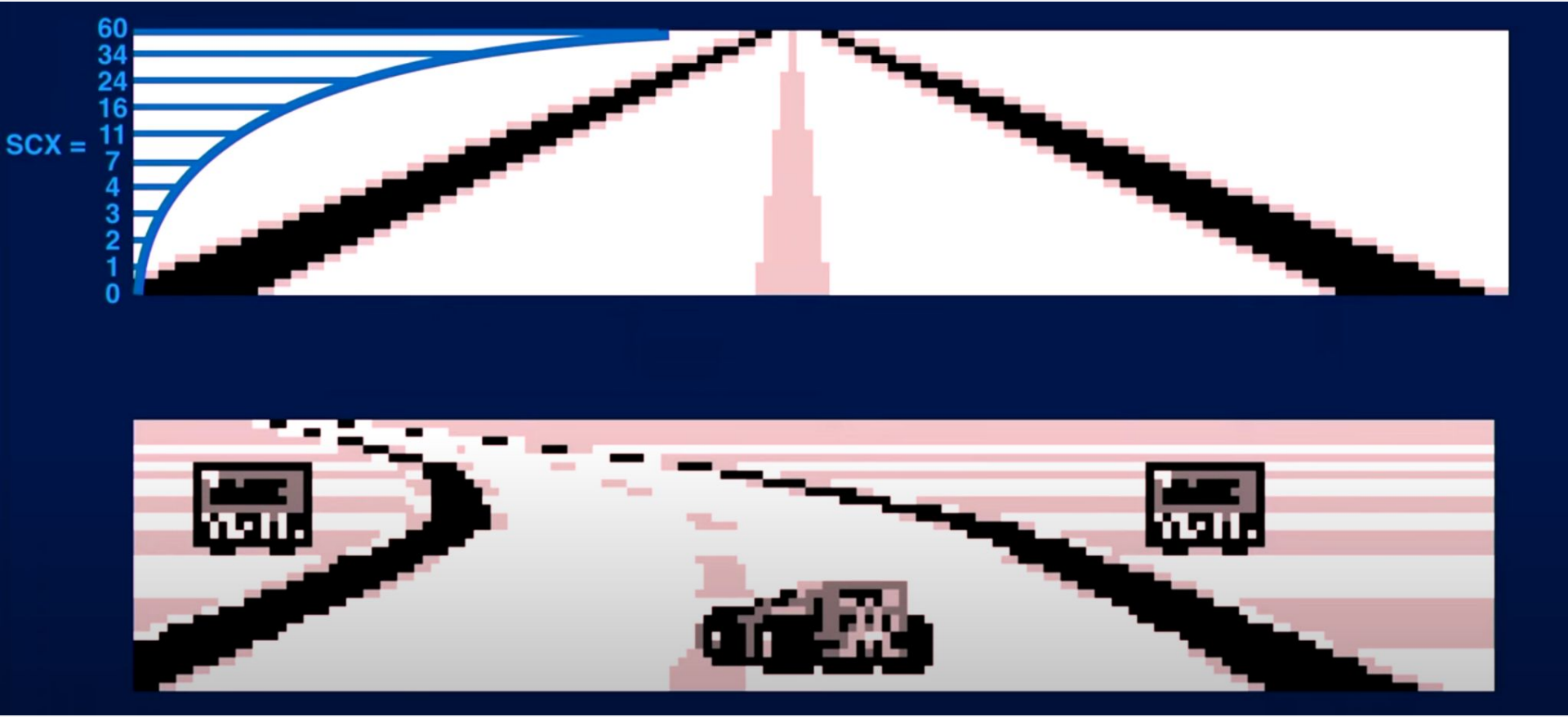


Źródło: https://nesdoug.com/

Cykl życia PPU



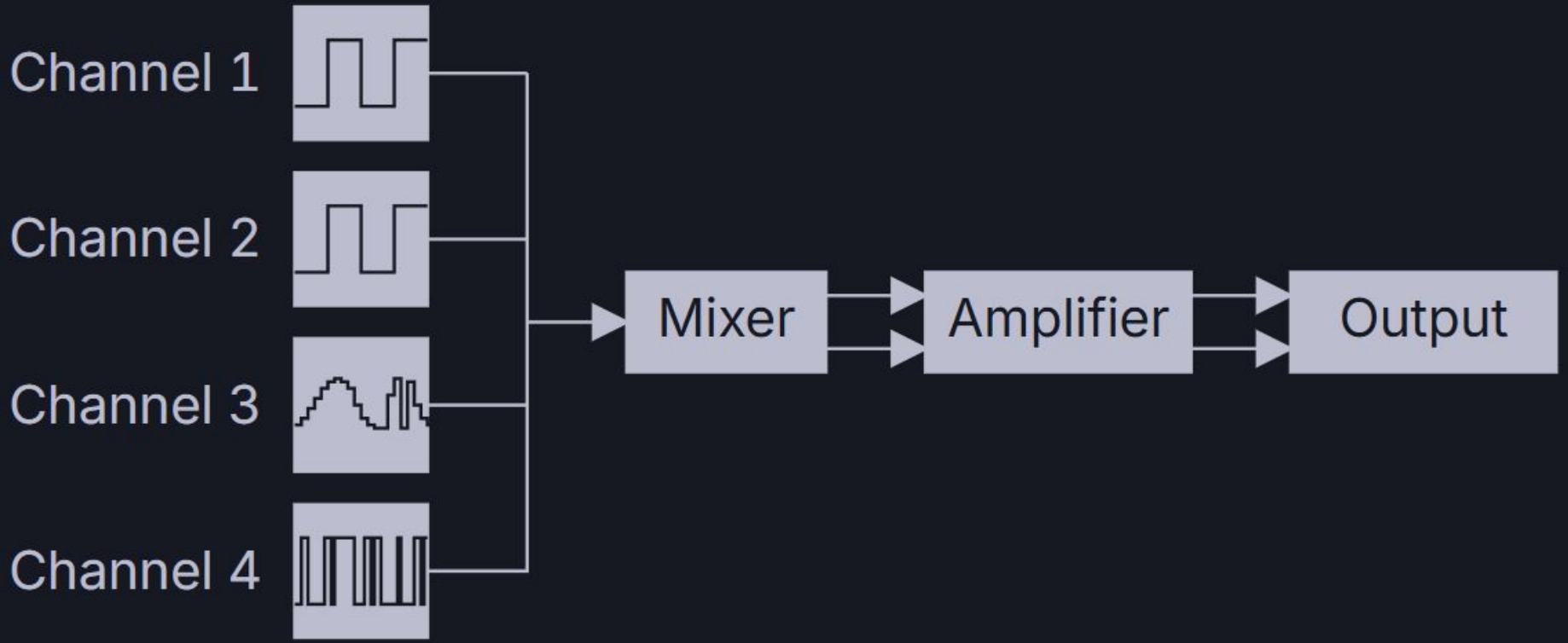
Źródło: https://gbdev.io/



SCX =
60
34
24
16
11
7
4
3
2
1
0

APU - Audio Processing Unit

- **PPU - Zbiór statycznych stanów**
- **APU - Zbiór timerów (+ stanów)**
- **4 niezależne kanały**
  - Square Wave 1 - Generator Fali Prostokątnej
  - Square Wave 2 - Generator Fali Prostokątnej
  - Wave - Własne próbki
  - Noise - Generator pseudolosowego szumu

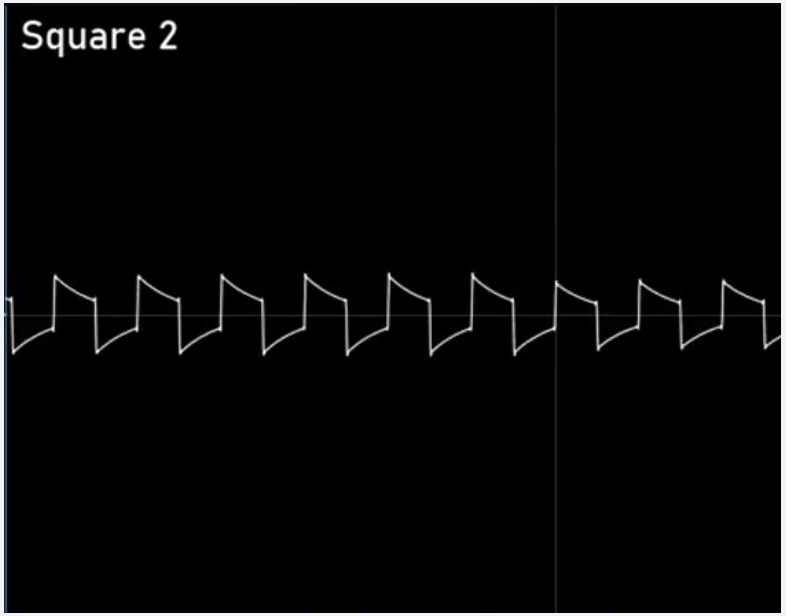
Architecture



APU - Square 1/2

- Generator fali prostokątnej
- DAC + HPF

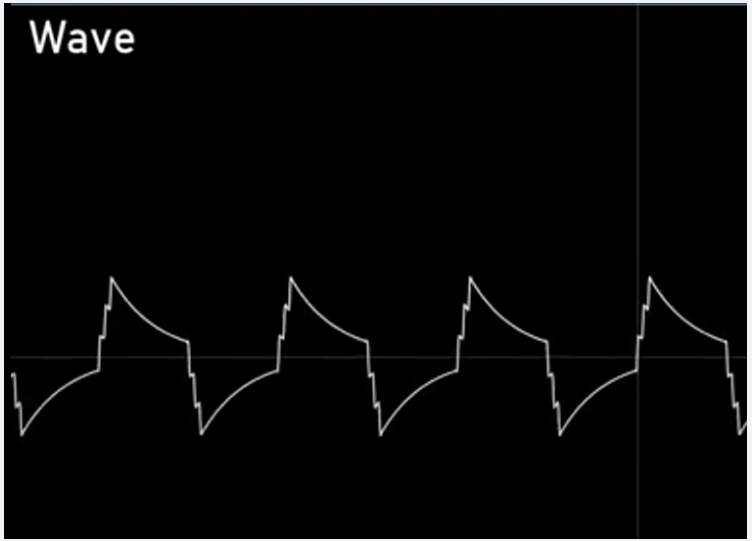
- Modulacja:
  - Amplitudy
  - Amplitudy w czasie
  - Częstotliwości
  - Czasu wykonania
  - *(Square 1)* Częstotliwości w czasie tzw. Envelope



APU - Wave 3

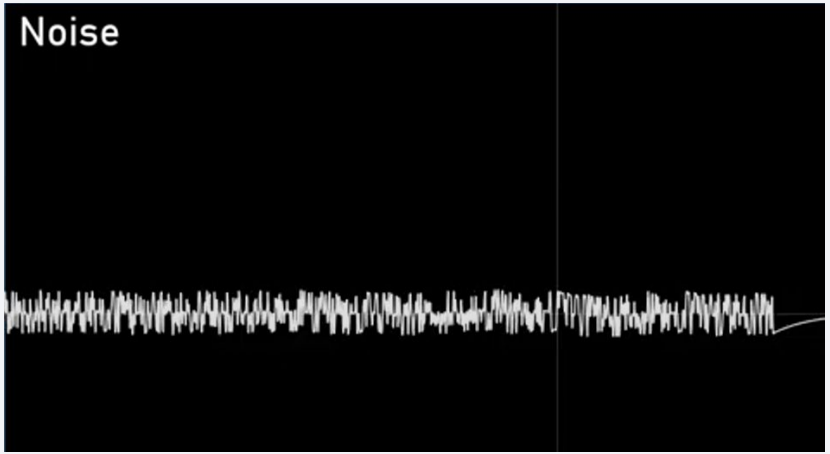
- Generator dowolnie zaprogramowanej fali
- 32 próbki po 4 bity (0-15)

- Modulacja:
  - Amplitudy (głośność)
  - Częstotliwości
  - Czasu wykonania



APU - Noise 4

- Generator pseudolosowego szumu
- Efekty perkusyjne/dźwiękowe



Szymon Sandura
.NET Senior Developer / Tech Lead

szymon.sandura@iteo.com

iteo.com