Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Jesenovec

Emulator igralne konzole Game Boy

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Za operacijski sistem MS Windows izdelajte emulator za igralno konzolo Game Boy. Opišite njegovo zgradbo in način delovanja, njegovo delovanje pa preizkusite z izbranimi igricami, za katere ugotovite, do katere mere delujejo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
LCD	liquid-crystal display	tekočekristalni zaslon
IR	infrared	infrardeče
ROM	read-only memory	bralni pomnilnik
RAM	random-access memory	bralno-pisalni pomnilnik
PC	program counter	programski števec
SP	stack pointer	skladovni kazalec
SGB	Super Game Boy	Super Game Boy
MBC	memory bank controller	krmilnik pomnilniških modulov
OAM	object attribute memory	tabela lastnosti objektov
DMA	direct memory access	neposredni dostop do pomnilnika

Povzetek

Naslov: Emulator igralne konzole Game Boy

Avtor: Klemen Jesenovec

Diplomsko delo obravnava problem študije stojne opreme in izdelave emulatorja za igralno konzolo Game Boy. Problema smo se lotili s preučevanjem dokumentacije Game Boya in njegovih lastnosti. Za implementacijo emulatorja smo zaradi njegove učinkovitosti in bližini strojne opreme izbrali jezik C++. Problem smo razbili na podprobleme in implementirali vsak podsistem Game Boya posebej, ki smo jih nato združili v celoto. Končen rezultat diplomske naloge je delujoč emulator Game Boya.

Ključne besede: emulator, Game Boy, C++.

Abstract

Title: Game Boy Emulator

Author: Klemen Jesenovec

This thesis describes the problem of hardware study and implementation of an emulator for the Game Boy video game console. The work began with studying Game Boy's documentation and its hardware properties. The C++ language was chosen for the implementation due to its efficiency and it being close to hardware. We divided the problem into sub-problems and implemented each Game Boy subsystem separately, which we then combined into a whole. The final result of the thesis is a working Game Boy emulator.

Keywords: emulator, Game Boy, C++.

Poglavje 1

Uvod

Game Boy [2] je prenosna igralna konzola, ki jo je leta 1989 izdalo japonsko podjetje Nintendo. Prvi model je prišel le v sivi barvi, prikazan je na sliki 1.1.

Slika 1.1: Prvi model igralne konzole Game Boy (DMG).

Visok je 148, širok 90 in debel 32 milimetorv. Zelen zaslon LCD brez

osvetljenega ozadja ima nastavljiv kontrast in premore štiri odtenke zelenkaste barve. Možno je tudi nastavljanje glasnosti zvoka in priključitev slušalk na 3.5 mm zvočni priključek. Igre so v obliki kaset in se jih vstavi v režo na zadnjem delu Game Boya. Kaseta igre Tetris je prikazana na sliki 1.2. Game Boy podpira tudi serijsko povezavo z drugo konzolo. Uporabnik z njim upravlja z osmimi gumbi.

Nintendu je uspelo prodati kar 118 milijonov konzol [7]. Tako je Game Boy, v času pisanja tega diplomskega dela, tretja najbolje prodajana igralna konzola vseh časov. Pred njim je le Sonyjev Playstation 2 z več kot 155 milijoni prodanih konzol in Nintendov Nintendo DS, ki je nasledil Game Boyevega naslednika in je bil prodan 154 milijonkrat.

V letih po izidu prvega modela igralne konzole, je izšlo še nekaj izboljšav [12]:

- Game Boy Light je bil preoblikovan izvirnik z manjšo obliko in bolj učinkovitim delovanjem, ter je tako porabil manj energije.
- Game Boy Pocket, ki je bil izdan le na japonskem, je bil prav tako preoblika, ki so mu dodali še osvetlitev ozadja.
- Game Boy Color je bila zadnja izboljšava, ki je konzoli dodala tudi zaslon v barvah in nekaj novih zmogljivosti strojne opreme kot so hitrejše delovanje, večja količina spomina, nove možnosti prikazovanja, ipd.

Game Boy so izdelovali vse do leta 2003, ko ga je nasledil Game Boy Advance. Seznam najbolj prodajanih iger vključuje [6]:

- Tetris (35 milijonov prodanih kopij),
- Pokémon Red, Green in Blue (31 milijonov prodanih kopij),
- Super Mario Land (18 milijonov prodanih kopij),
- Kirby's Dream Land (5 milijonov prodanih kopij).

Slika 1.2: Kaseta igre Tetris.

V tej diplomski nalogi se bomo posvetili predvsem lastnostim strojne opreme izvirnega modela Game Boya s poudarkom na izdelavi emulatorja, ki bo tekel na osebnih računalnikih. Vsa izvorna koda nastalega emulatorja je prosto dostopna na GitHubu [5].

V drugem poglavju sledi primerjava sorodnih del. V naslednih poglavjih pa so podrobnejše opisane lastnosti strojne opreme in njihove implementacije v emulatorju napisanem v programskem jeziku C++. V poglavju 3 se nahaja splošni opis strojne opreme in emulatorja. Temu sledijo poglavja, ki podrobneje opišejo podsisteme. V poglavju 4 centralno procesno enoto, v 5. pomnilnik, v 6. grafični koprocesor, v 7. tipkovnico in v 8. poglavju je opisan časovnik. V 9. poglavju je opisano delovanje emulatorja. Temu sledi le še zaključek.

Poglavje 2

Sorodna dela

Emulator [1] je programska oprema, ki omogoči gostujočemu računalniškemu sistemu, da se obnaša kot drug računalniški sistem t. i. gost. To omogoča gostujočemu računališkemu sistemu poganjanje programske opreme, ki je bila napisana za računalniški sistem gosta. To ima kar nekaj prednosti, kot so

- dodatne zmogljivosti, ki jih izvirni računalniški sistem ni imel;
- omogočanje poganjanja programske opreme na drugačnih sistemih;
- ohranjanje stare programske opreme na novejših sistemih.

Seveda pa imajo emulatorji tudi nekaj hib:

- legalne težave s kršitvijo avtorskih pravic;
- gostiteljski sistemi morajo biti bolj zmogljivi od sistema gosta.

Od Game Boyevega izida do danes je bilo narejenih že kar nekaj emulatorjev. Seznam pomembnejših obstoječih emulatorjev vključuje:

- BGB je zaprtokodni emulator, poznan po svoji natančnosti in vizualnim razshroščevalnikom.
- Gambatte je zelo natančen emulator, ki pa je tudi odprtokoden.
- Coffee GB je emulator napisan v Javi.

- GEM je emulator Game Boya, ki uporablja dinamično prevajanje.

Seznam podobnih emulatorjev, ki niso neposredno povezani z Game Boyem, vključuje:

- Oracle Virtual Box je prosta in odprtokodna programsa oprema za virtualizacijo osebnih računalnikov. Uporablja dinamično prevajanje.
- VMware Workstation Player je prosta, ampak ne odprtokodna, programska oprema za virtualizacijo osebnih računalnikov.
- DosBox je prosto dostopen emulator, ki emulira računalnike kompatibilne z IBMovem osebnim računalnikom in poganja DOS operacijski sistem.

2.1 Statično tolmačenje in dinamično prevajanje

Emulatorje lahko delimo v dve večji skupini glede na metodo izvajanja programske kode gosta. Emulatorji s s statičnim tolmačenjem kodo programske opreme gosta izvajajo ukaz za ukazom, tako da ukaze berejo in izvajajo programsko. Po drugi strani emulatorji z dinamičnim prevajanjem ob samem delovanju emulatorja kodo gosta prevedejo v strojno kodo, ki se izvaja na gostiteljskem računalniškem sistemu. Spodaj je naštetih nekaj prednosti in slabosti obeh metod.

`statično tolmačenje`	`dinamično prevajanje`
lažja implementacija	težje za implementacijo
počasnejše izvajanje	programi se izvajajo hitreje
niso vezani na arhitekturo gostitelja	potrebno je znanje o arhitekturi gostitelja

Tabela 2.1: Primerjava statičnega tolmačenja in dinamičnega prevajanja.

Spodaj je v tabeli naštetih nekaj programov, ki uporabljajo različne metode.

`statično tolmačenje`	`dinamično prevajanje`
DosBox	Oracle Virtual Box
Jpcsp	PPSSPP
Coffee GB	PCSX2
	RPCS3

Tabela 2.2: Primeri programov, ki uporabljajo različne metode.

V tem diplomskem delu smo se zaradi hitrejše implementacije odločili za emulator s statičnim tolmačenjem, saj to še vedno zadošča za realistično izvajanje programov za igralno konzolo Game Boy.

Poglavje 3

Pregled strojne opreme

Game Boy je kot večino ostalih računalnikih sistemov sestavljen iz centralne procesne enote, pomnilnika in vhodno izhodnih naprav.

Ima po meri narejeno centralno procesno enoto, ki je podobna Zilogovem Z80. Ima 8-bitno aritmetično logično enoto in 16-bitno vodilo. Urina frekvenca znaša 4.194304 MHz. Vgajen je tudi časovnik, ki lahko proži prekinitve. Poleg časovne prekinitve lahko prekinitve prožijo tudi video enota, gumbi in serijska povezava.

Notranji pomnilnik je razdeljen na delovni, video in specializiran pomnilnik. Poleg notranjega pomnilnika se v kaseti nahaja tudi zunanji pomnilnik. Na tej se nahaja tudi ROM z igro, ki je lahko dostopen po pomnilniških modulih. Kasete imajo lahko na sebi tudi dodatne funkcije kot so ura, IR senzorji, ipd.

Poleg centralne procesne enote ima Game Boy tudi grafični koprocesor, ki skrbi za izris grafike na zaslon. Ta je 160x144 točk velik LCD zaslon, ki pa nima osvetljenega ozadja in je sposoben prikazovati 4 barve. Osvežuje se s frekvenco 59.73 Hz.

Game Boy za ustvarjanje zvoka uporablja posebno zvočno enoto. Ima štiri kanale, ki jih lahko posebej zmeša na levi ali desni zvočnik. Prva dva kanala lahko proizvedeta kvadratne valove in imata kontrolo za glasnost, frekvenco in dinamični pomik teh. Tretji kanal proizvaja valove po meri, ki so shranjeni

v specializiranemu pomnilniku. Zadnji kanal pa proizvaja naključen šum.

Uporabnik s konzolo upravlja s pomočjo osmih gumbov: štiri smerne tipke, tipka A, tipka B, SELECT in START.

Game Boy napajajo štiri AA baterije. Električna poraba se razlikuje od igre do igre, saj so razvijalci uporabljali različne trike za varčevanje z energijo. Ti vključujejo izklapljanje zvočne enote, ko ni v uporabi, ali pa spretno uporabljanje ustavljenega načina delovanja. Tako lahko s polnimi baterijami Game Boy zdrži 20–35 ur igranja [8, 9, 12, 11].

3.1 Sestava emulatorja

Emulator smo napisali v programskem jeziku C++ z uporabo programskega orodja Microsoft Visual Studio 2017. Preveden je s prevajalnikom, ki podpira C++11 standard. Uporabili smo samo eno zunanjo knjižnjico za potrebe izrisa in branja tipkovnice. Zaradi prenosljivosti in odprokodnosti smo se odločili za Simple DirectMedia Layer (SDL) [10].

Emulator je sestavljen modularno. Glavni razred `Board`, ki predstavlja tiskano vezje, vsebuje posamezne module, ki predstavljajo centralno procesno enoto, grafični koprocesor in pomnilnik. Te moduli implementirajo svoje funkcionalnosti, ki jih nadzoruje in usklajuje glavni razred. Implementacija vsakega od modulov je podrobneje razložena v naslednjih poglavjih.

Poleg razreda `Board` obstaja še razred z imenom `Renderer`, ki poskrbi za izrisovanje in časovno usklajevanje izrisa z emulacijo izvajanja.

Poglavje 4

Centralna procesna enota

4.1 Registri in zastavice

Game Boyeva centralna procesna enota ima osem 8-bitnih registrov in dva 16-bitna regista. Predstavljeni so v tabeli 4.1, kjer en stolpec predstavlja širino osmih bitov.

A	F
B	C
D	E
H	L
PC	
SP	

Tabela 4.1: Registri.

En par 8-bitnih registrov se lahko uporablja kot 16-bitni register, tako dobimo registre `AF`, `BC`, `DE` in `HL`. Register `A` je akumulator in se uporablja za vse aritmetično logične operacije. Register F v zgornjih štirih bitih drži štiri zastavice. Po vrsti od sedmega do četrtega bita so:

- `Z`: ničelna zastavica, ki se postavi, ko je rezultat operacije enak 0;

- `N`: negativna zastavica, ki se postavi ob odštevalni operaciji;
- `H`: polprenosna zastavica, ki se postavi, če pride do prenosa iz tretjega na četrti bit;
- `C`: prenosna zastavica, ki se postavi ob prenosu iz sedmega bita.

Zastavice se ne spreminjajo ob vsakem ukazu, saj nekateri ukazi pustijo določene zastavice nespremenjene. Zastavici `N` in `H` se uporabljata samo pri ukazu DAA. Spodnji biti regista F so vedno postavljeni na 0.

Register `PC` je programski števec in kaže na naslov naslednjega ukaza. Register `SP` je skladovni kazalec, ki kaže na vrh sklada. Sklad se širi proti nižjim naslovom, skladovni kazalec pa kaže na zadnji vstavljen element (t. i. *full descending* sklad).

4.2 Ukazi

Game Boy pozna 501 ukazov, ki so lahko različne dolžine. V tabelah 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 in 4.10 so predstavljeni vsi ukazi razvrščeni pa njihovi vrsti [3, 8, 9].

Vrste operandov so lahko: d8 (takojšni 8-bitni podatek), d16 (takojšni 16-bitni podatek), a8 (8-bitni del naslova), a16 (16-bitni naslov), sd8 (8-bitni predznačen podatek), r8 (8-bitni register), r16 (16-bitni register). Podatek v oklepajih predstavlja naslov.

Zastavice so označene v enakem vrstnem redu kod zgoraj. Oznaka `0` pomeni, da se zastavica postavi na 0, `1` pa na 1. Zastavica označena s črko pomeni, da se postavi glede na rezultat ukaza (npr. zastavica `C`, če je pri seštevanju prišlo do preliva). Oznaka `-` pomeni, da ukaz zastavice ne spreminja.

Dolžina ukaza je označena v bajtih, trajanje pa v urinih periodah. Nekateri ukazi imajo več trajanj, saj trajajo različno dolgo odvisno od izpolnjenosti pogoja.

mnemonik	operandi	zastavice	dolžina	trajanje
`ld`	`r8, r8`	`----`	1	4
`ld`	`r8, d8`	`----`	2	8
`ld`	`r8, (HL)`	`----`	1	8
`ld`	`(HL), r8`	`----`	1	8
`ld`	`(HL), d8`	`----`	2	12
`ld`	`A, (r16)`	`----`	1	8
				Samo z `BC` in `DE`
`ld`	`A, (a16)`	`----`	3	16
`ld`	`(r16), A`	`----`	1	8
				Samo z `BC` in `DE`
`ld`	`(a16), A`	`----`	3	16
`ldh`	`A, (a8)`	`----`	2	12
				Naloži iz naslova 0xFF00 + a8
`ldh`	`(a8), A`	`----`	2	12
				Naloži na naslov 0xFF00 + a8
`ldh`	`A, (C)`	`----`	1	8
				Naloži iz naslova 0xFF00 + `C`
`ldh`	`(C), A`	`----`	1	8
				Naloži na naslov 0xFF00 + `C`
`ldi`	`(HL), A`	`----`	1	8
`ldi`	`A, (HL)`	`----`	1	8
Ukaz `ldi` po nalaganju inkrementira register `HL`				
`ldd`	`(HL), A`	`----`	1	8
`ldd`	`A, (HL)`	`----`	1	8
Ukaz `ldd` po nalaganju dekrementira register `HL`				

Tabela 4.2: 8-bitni ukazi za nalaganje.

mnemonik	operandi	zastavice	dolžina	trajanje
`ld`	`r16, d16`	`----`	3	12
		Samo z ***BC****,* ***DE****,* ***HL*** *in* ***SP***		
`ld`	`SP, HL`	`----`	1	8
`push`	`r16`	`----`	1	16
`pop`	`r16`	`znhc`	1	12
		Zastavice se postavijo ob nalaganju ***AF***		

Tabela 4.3: 16-bitni ukazi za nalaganje.

mnemonik	operandi	zastavice	dolžina	trajanje
`daa`		`z-0c`	1	4
	Popravi akumulator po pravilu BCD aritmetike			
`cpl`		`z0hc`	1	4
	Naredi eniški komplement akumulatorja			
`inc`	`r8`	`z0h-`	1	4
`inc`	`(HL)`	`z0h-`	1	12
`dec`	`r8`	`z1h-`	1	4
`dec`	`(HL)`	`z1h-`	1	12

Tabela 4.4: 8-bitni aritmetično logični ukazi.

mnemonik	operandi	zastavice	dolžina	trajanje
`add`	`A, r8`	`z0hc`	1	4
`add`	`A, d8`	`z0hc`	2	8
`add`	`A, (HL)`	`z0hc`	1	8
`adc`	`A, r8`	`z0hc`	1	4
`adc`	`A, d8`	`z0hc`	2	8
`adc`	`A, (HL)`	`z0hc`	1	8
`sub`	`A, r8`	`z1hc`	1	4
`sub`	`A, d8`	`z1hc`	2	8
`sub`	`A, (HL)`	`z1hc`	1	8
`sbc`	`A, r8`	`z1hc`	1	4
`sbc`	`A, d8`	`z1hc`	2	8
`sbc`	`A, (HL)`	`z1hc`	1	8
Ukaza **`adc`**/**`sbc`** *prištejeta/odštejeta še zastavico* **`C`**				
`and`	`r8`	`z010`	1	4
`and`	`d8`	`z010`	2	8
`and`	`(HL)`	`z010`	1	8
`xor`	`r8`	`z000`	1	4
`xor`	`d8`	`z000`	2	8
`xor`	`(HL)`	`z000`	1	8
`or`	`r8`	`z000`	1	4
`or`	`d8`	`z000`	2	8
`or`	`(HL)`	`z000`	1	8
`cp`	`r8`	`z1hc`	1	4
`cp`	`d8`	`z1hc`	2	8
`cp`	`(HL)`	`z1hc`	1	8

Tabela 4.5: 8-bitni aritmetično logični ukazi.

mnemonik	operandi	zastavice	dolžina	trajanje
`add`	`HL, r16`	`-0hc`	1	8
`inc`	`r16`	`----`	1	8
`dec`	`r16`	`----`	1	8
	V zgornjih treh ukazih samo ***BC****,* ***DE****,* ***HL*** *in* ***SP***			
`add`	`SP, sd8`	`00hc`	2	16
`ld`	`HL, SP+sd8`	`00hc`	2	12

Tabela 4.6: 16-bitni aritmetično logični ukazi.

mnemonik	operandi	zastavice	dolžina	trajanje
`bit`	`0--7, r8`	`z01-`	2	8
`bit`	`0--7, (HL)`	`z01-`	2	12
			Ukaz ***`bit`*** *testira bit*	
`set`	`0--7, r8`	`----`	2	8
`set`	`0--7, (HL)`	`----`	2	16
`res`	`0--7, r8`	`----`	2	8
`res`	`0--7, (HL)`	`----`	2	16

Tabela 4.7: Enobitni ukazi.

mnemonik	operandi	zastavice	dolžina	trajanje
`rlca`		`000c`	1	4
`rla`		`000c`	1	4
`rrca`		`000c`	1	4
`rra`		`000c`	1	4
		Zgornji štiri ukazi rotirajo akumulator		
		Ukazi s črko c rotirajo tudi čez zastavico `C`		
`rlc`	`r8`	`z00c`	2	8
`rlc`	`(HL)`	`z00c`	2	16
`rl`	`r8`	`z00c`	2	8
`rl`	`(HL)`	`z00c`	2	16
`rrc`	`r8`	`z00c`	2	8
`rrc`	`(HL)`	`z00c`	2	16
`rr`	`r8`	`z00c`	2	8
`rr`	`(HL)`	`z00c`	2	16
`sla`	`r8`	`z00c`	2	8
`sla`	`(HL)`	`z00c`	2	16
		Ukaz `sla` *je aritmetični pomik levo*		
`swap`	`r8`	`z00c`	2	8
`swap`	`(HL)`	`z00c`	2	16
		Ukaz `swap` *zamenja zgornje in spodnje štiri bite.*		
`sra`	`r8`	`z00c`	2	8
`sra`	`(HL)`	`z00c`	2	16
		Ukaz `sra` *je aritmetični pomik desno*		
`srl`	`r8`	`z00c`	2	8
`srl`	`(HL)`	`z00c`	2	16
		Ukaz `srl` *je logični pomik desno*		

Tabela 4.8: Rotacije in pomiki.

mnemonik	operandi	zastavice	dolžina	trajanje
		Pogoji so lahko: **`z`**, **`nz`**, **`c`** *in* **`nc`**		
`jp`	`a16`	`----`	3	16
`jp`	`HL`	`----`	1	4
`jp`	`pogoj, a16`	`----`	3	16/12
`jr`	`PC+sd8`	`----`	2	12
`jr`	`pogoj, PC+sd8`	`----`	2	12/8
`call`	`a18`	`----`	3	24
`call`	`pogoj, a18`	`----`	3	24/12
	Ob ukazu **`call`** *se na sklad zapiše vsebina registra* **`PC`**			
`ret`		`----`	1	16
`ret`	`pogoj`	`----`	1	20/8
`reti`		`----`	1	16
	Ob ukazu **`ret/reti`** *se iz sklada prepišeta dva bajta v* **`PC`**			
		Ukaz **`reti`** *ob vrnitvi omogoči prekinitve*		
`rst`	`n`	`----`	1	16
	Ukaz **`rst`** *je klic na naslove: 0x00, 0x08, ... , 0x38*			

Tabela 4.9: Skočni ukazi.

mnemonik	operandi	zastavice	dolžina	trajanje
`ccf`		`-00c`	1	4
		Ukaz **`ccf`** *komplementira zastavico* **`C`**		
`scf`		`-001`	1	4
		Ukaz **`scf`** *počisti zastavico* **`C`**		
`nop`		`----`	1	4
`halt`		`----`	1	4
		Ukaz **`halt`** *začne nizkoenergijski način*		
`stop`		`----`	1	4
		Ukaz **`stop`** *začne zelo nizkoenergijski način*		
`di`		`----`	1	4
		Ukaz **`di`** *onemogoči prekinitve*		
`ei`		`----`	1	4
		Ukaz **`ei`** *omogoči prekinitve*		

Tabela 4.10: Kontrolni ukazi.

4.3 Prekinitve

Game Boy pozna 5 prekinitev, ki si po prioriteti od najvišje do najnižje sledijo takole [9]:

1. Prekinitev V-blank, ki se zahteva, ko grafični koprocesor nariše sliko. PC se postavi na 0x40.

2. Prekinitev LCD STAT, ki se zahteva ob različnih pogojih grafičnega koprocesorja. Pogoje lahko nastavimo s pomočjo registrov grafičnega koprocesorja. PC se postavi na 0x48.

3. Časovna prekinitev, ki se zahteva, ko časovnk doseže svojo nastavljeno vrednost. PC se postavi na 0x50.

4. Serijska prekinitev, ki se zahteva, ko je poslan/prejet en bajt podatkov. PC se postavi na 0x58.

5. Prekinitev tipkovnice, ki se zahteva, ko je na eni od prej vklopljenima linijima za tipko nizek signal. PC se postavi na 0x60.

Vsaka prekinitev ima v registru `IE` (*Interrupt Enable*) na naslovu 0xFFFF bit, ki jo omogoči. Prekinitvi z najvišjo prioriteto pripada bit z najmanjšo težo. Prekinitev se zahteva tako, da se v registru `IF` (*Interrupt Flag*) na naslovu 0xFF0F postavi njen bit na 1.

Prekinitev se servisira samo, če sta njena bita v registru `IE` in registru `IF` postavljena na 1 in je hkrati nastavljen tudi `IME` (*Interrupt Master Enable Flag*), ki ga vklopimo/izklopimo z ukazoma `ei` in `di`. Če se prekinitev začne izvajati, se avtomatsko pobriše njen bit v registru `IF` in izklopi `IME`. Prav tako se register `PC` porine na sklad. V nasprotnem primeru ostane bit v registru `IF` postavljen in tako prekinitev čaka na izvajanje.

Prekinitve lahko gnezdimo, tako da ob vstopu v prekinitveno servisni program postavimo `IME` na 1 z ukazom `ei`. Možno je tudi ročno zahtevati prekinitev s postavitvijo ustreznega bita v registru `IF`.

S pomočjo prekinitev izstopimo iz nizkoenergijskih načinov.

4.4 Implementacija v emulatorju

Centralno procesno enoto smo implementirali v razredu z imenom `CPU`. Registre smo implementirali v strukturi `regs`, ki vključuje anonimne unije, v katerih so anonimne strukture. Struktura je razvidna na sliki 4.1. Na ta način lahko dosežemo enostavno naslavljanje 8-bitnih in 16-bitnih registrov (npr. `regs.bc` ali `regs.h`).

```
// Registers
struct {
    union {
        struct { byte f; byte a; };
        word af;
    };
    union {
        struct { byte c; byte b; };
        word bc;
    };
    union {
        struct { byte e; byte d; };
        word de;
    };
    union {
        struct { byte l; byte h; };
        word hl;
    };
    word pc, sp;
}regs;
```

Slika 4.1: Implementacija registrov v razredu `CPU`.

Za izvajanje ukazov smo implementirali nekaj pomožnih funkcij, ki lahko izvajajo več operacijskih kod glede na vrsto ukaza. Podpisi funkcij so prikazani na sliki 4.2. Funkcija `add(byte)` je na sliki 4.3.

```
byte incByte(byte);
byte decByte(byte);
byte rlc(byte);
byte rrc(byte);
byte rl(byte);
byte rr(byte);
word addWords(word, word);
void jumpRelative(sbyte);
void daa();
void add(byte);
void adc(byte);
void sub(byte);
void sbc(byte);
void and(byte);
void xor(byte);
void or(byte);
void cp(byte);
word addWordSbyte(word, sbyte);
byte sla(byte);
byte sra(byte);
byte srl(byte);
byte swap(byte);
void bit(byte, byte);
byte res(byte, byte);
byte set(byte, byte);
```

Slika 4.2: Podpisi pomožnih funkcij v razredu `CPU`.

```
void CPU::add(byte op1) {
    regs.f = 0x00;
    if((regs.a & 0xF) + (op1 & 0xF) > 0xF) regs.f |= 0x20;
    if(regs.a + op1 > 0xFF) regs.f |= 0x10;
    regs.a += op1;
    if(regs.a == 0x00) regs.f |= 0x80;
}
```

Slika 4.3: Implementacija funkcije `add(byte)`.

Glavna funckija razreda `CPU` je `exec(byte)`, ki glede na operacijsko kodo podano kot argumet, izvede ukaz in nastavi čas izvajanja v urinih periodah.

V njej se nahaja `switch` stavek, ki se glede na operacijsko kodo odloči, kaj se izvede. Del implementacije je razviden iz slike 4.4.

```
case 0xD5: // PUSH DE
    regs.sp -= 2;
    mem -> writeWord(regs.sp, regs.de);
    clocks += 16;
    break;
case 0xD6: // SUB d8
    sub(mem -> readByte(regs.pc++));
    clocks += 8;
    break;
case 0xD7: // RST 10H
    regs.sp -= 2;
    mem -> writeWord(regs.sp, regs.pc);
    regs.pc = 0x10;
    clocks += 16;
    break;
```

Slika 4.4: Del implementacije funkcije `exec(byte)`.

Razred `Board`, ki kontrolira izvajanje, pred vsakim izvedenim ukazom kliče funkcijo `handleInterrupts()`, ki jo implementira razred `CPU`.

Ta funkcija po vrstnem redu prioritet prekinitev preveri, če je prišlo do prekinitve in na sklad porine programski števec, ga ustrezno popravi, počisti `IME` in po potrebi izstopi iz nizkoenergijskega načina delovanja.

Poglavje 5

Pomnilnik

Game Boyeva centralno procesna enota ima 16-bitno naslovno vodilo. Tako lahko naslovi 64K pomnilniških besed. Splošna pomnilniška slika je predstavljena na tabeli 5.1 [9].

naslov	velikost	opis
`0x0000`–`0x3FFF`	16 KB	ničti pomnilniški modul ROM-a (na kaseti)
`0x4000`–`0x7FFF`	16 KB	n-ti pomnilniški modul ROM-a (na kaseti)
`0x8000`–`0x9FFF`	8 KB	grafični pomnilnik
`0xA000`–`0xBFFF`	8 KB	zunanji pomnilnik (na kaseti, lahko več pomnilniških modulov)
`0xC000`–`0xDFFF`	8 KB	delovni pomnilnik
`0xE000`–`0xFDFF`	7680 B	preslikava delovnega pomnilnika (se ne uporablja)
`0xFE00`–`0xFE9F`	160 B	tabela lastnosti za 40 poseblih grafičnih ploščic
`0xFEA0`–`0xFEFF`	96 B	se ne uporablja
`0xFF00`–`0xFF7F`	128 B	vhodno izhodni registri (niso vsi dostopni)
`0xFF80`–`0xFFFE`	127 B	višji pomnilnik (možnost hitrejšega dostopa)
`0xFFFF`	1 B	`IE` register

Tabela 5.1: Splošna pomnilniška slika.

Na ničtem pomnilniškem modulu, ki se nahaja na kaseti v ROM-u, je na naslovih `0x100`–`0x14F` zaglavje kasete, v kateri se nahajajo meta podatki. Opisani so v tabeli 5.2 [9].

naslov	opis
`0x100`–`0x103`	vstopna točka, kjer se začne izvajanje programa
`0x104`–`0x133`	Nintendov logo, ki ga Game Boy preveri
`0x134`–`0x143`	ime igre (neuporabljeni bajti so postavljeni na 0)
`0x144`–`0x145`	nova licenčna koda
`0x146`	SGB zastavica
`0x147`	tip kasete
`0x148`	velikost ROM-a
`0x149`	velikost zunanjega pomnilnika
`0x14A`	koda destinacije
`0x14B`	stara licenčna koda
`0x14C`	verzija igre
`0x14D`	kontrolna vsota zaglavja (se preveri)
`0x14E`–`0x14F`	globalna kontrolna vsota (se ne preveri)

Tabela 5.2: Zaglavje kasete.

Ničti pomnilniški modul se vedno nahaja na prvih 16 KB ROM-a. Na naslednjih 16 KB je lahko preostali ROM ali pa nek pomnilniški modul ROM-a, če je igra večja od 32 KB. Pomnilniške module izbiramo tako, da pišemo na naslove `0x0000`–`0x7FFF`. Ta pisanja prebere krmilnik pomnilniških modulov (`MBC`, *Memory Bank Controller*), ki se nahaja v kaseti in skrbi za menjavo pomnilniških modulov. Obstaja več vrst krmilnikov pomnilniških modulov. Katerega igra uporablja, je zapisano v zaglavju kasete na naslovu `0x147`. Podrobneje so opisani spodaj.

V grafični pomnilnik se shranjujejo grafične ploščice, ki jih želimo izrisati. Poleg ploščic se tu nahajata še dve mapi, ki poskrbita za pravilno postavitev ploščic.

Zunanji pomnilnik se nahaja na kaseti in se ponavadi uporablja za shranjevanje stanja igre, tabele z najboljšimi dosežki ali preslikavo registrov zunanje strojne opreme. Ta pomnilnik je ponavadi podprt z baterijo, tako da se podatki ne izbrišejo po izklopu Game Boya.

Delovni pomnilnik je namenjen splošni uporabi. Tukaj je ponavadi postavljen tudi sklad programa.

Na naslednjih 7680 bajtih v pomnilniku se nahaja preslikava delovnega pomnilnika, ki se ne uporablja.

Na naslovih `0xFE00`–`0xFE9F` se nahaja `OAM` (*Object Attribute Memory*), ki vsebuje informacije za 40 posebnih grafičnih ploščic, ki se lahko prosto pomikajo po ekranu.

Sledi še nekaj neuporabljenega prostora in vhodno izhodni registri.

Proti koncu pomnilniške slike se nahaja višji pomnilnik, do katerega lahko dostopamo s posebnimi ukazi, ki v ROM-u zavzamejo manj prostora od navadnih ukazov za dostop do pomnilnika.

5.1 Krmilniki pomnilniških modulov

Za igre manjše od 32 KB krmilnik ni potreben, saj lahko celotno igro postavimo na prvih 32 KB v pomnilniški sliki. Velika večina iger pa uporablja enega od spodaj naštetih krmilnikov pomnilniških modulov [9]:

- Krmilnik `MBC1`. Ta krmilnik podpira 125 pomnilniških modulov (vsak 32. pomnilniški modul ni podprt). Tako je lahko velikost ROM-a malo manjša od 2 MB. RAM je lahko velik največ 32 KB.

- Krmilnik `MBC2`. Podpira do 16 pomnilniških modulov. Velikost ROM-a znaša največ 256 KB. Ima poseben RAM, ki je vgrajen kar v krmilnik. Ima 512 besed po 4 bite.

- Krmilnik `MBC3`. Podpira do 128 pomnilniških modulov. ROM je tako lahko velik 2 MB. Velikost RAM-a je 32 KB. Krmilnik podpira tudi dodaten časovnik `RTC` (*Real Time Clock*).

- Krmilnik `MBC5`. Podpira do 64 pomnilniških modulov (8 MB ROM-a). Podprtih je do 128 KB RAM-a. Poleg dodatnega časovnika `RTC` podpira ta krmilnik tudi motor za haptične informacije.

5.2 Implementacija v emulatorju

Implementacija pomnilnika se nahaja v razredu `Memory`. Ta razred vsebuje tabele za zaglavje kasete, delovni pomnilnik, vhodno izhodne registre in višji pomnilnik. Člane razreda lahko vidimo na sliki 5.1.

```
std::string filepath = "";
byte cartrigeHeader[0x50];  // Address 0x100 - 0x14F

// controller class with rom and ram
MBCBase* mbc = nullptr;

// VRAM, OAM and LCD registers
LCD* lcd = nullptr;

Joypad joypad;
Timer timer;

byte workRam[0x2000];

byte IOPorts[0x80];
byte highRam[0x80];
```

Slika 5.1: Razred `Memory`.

Grafični pomnilnik, grafični registri in `OAM` so implementirani v razredu `LCD`, ki realizira grafični koprocesor. Za branje registrov tipkovnice je tu razred `Joypad`. Časovnik nadzoruje `Timer`.

Razred `Memory` ima po par metod za branje in pisanje. Metode, ki imajo v imenu besedo `get` ali besedo `set`, se uporabljajo za interno nastavljanje pomnilnika. Metode, ki imajo v imenu besedo `read` ali besedo `write`, pa se uporabljajo za simulacijo pravih branj in pisanj.

Za zunanji pomnilnik (ROM in RAM) skrbijo razredi, ki implemetirajo

krmilnike pomnilniških modulov. Tu nam v pomoč pride abstraktni razred `MBCBase`, po katerem dedujejo vsi razredi, ki implementirajo krmilnike pomnilniških modulov. Tako imamo v razredu `Memory` lahko le kazalec na bazni razred. Razred `MBCBase` je prikazan na sliki 5.2.

```
class MBCBase {
protected:
    byte romBnkNum; // Number of ROM banks on chip
    byte ramSize;   // Number of RAM banks on chip

    byte **rom;     // ROM banks
    byte **ramExt;  // External RAM banks
public:
    virtual byte getByte(word addr) = 0;
    virtual void setByte(word addr, byte data) = 0;
    virtual byte readByte(word addr) = 0;
    virtual void writeByte(word addr, byte data) = 0;

    MBCBase(const byte *header, std::string filepath);
    ~MBCBase();
};
```

Slika 5.2: Razred `MBCBase`.

Preden se lahko igra začne izvajati, se mora pomnilnik inicializirati. To se zgodi v konstruktorju razreda `Memory`, ki prejme pot do datoteke, ki vsebuje ROM igre. Ta prebere zaglavje kasete in glede na tip kasete inicializira razred ustreznega krmilnika pomnilniških modulov. Konstruktor tega razreda iz zaglavja razbere velikost ROM-a in RAM-a. Nato rezervira dovolj prostora za vse pomnilniške module in shrani vsebino vseh pomnilniških modulov ROM-a. Tako je igra pripravljena na izvajanje. Ob zapiranju emulatorja se viri počistijo s pomočjo destruktorjev.

Poglavje 6

Grafični koprocesor

Grafični koprocesor izrisuje na zaslon, ki je širok 160 in visok 144 točk. Zaradi omejitev spomina uporablja tehniko ploščic, kjer je najmanjša izrisljiva enota ena 8x8 točk velika ploščica.

Sposoben je izrisati pomikajoče ozadje. V spominu je mapa, ki določa 256 točk široko kvadratno ozadje. Ker je velikost zaslona manjša od velikosti tega ozadja, lahko zaslon "pomikamo"po tem ozadju in tako dosežemo zamik ozadja. To se določi s točko, od katere se bo desno in dol ozadje izrisalo na zaslon. Če zaslon preseže desno ali spodnjo mejo ozadja, se izris nadaljuje na levi oziroma na vrhu ozadja.

Nad navadnim ozadjem lahko izrišemo še eno pokrivajoče ozadje, ki se imenuje okno. Ta se vedno izriše od izbrane točke na ekranu dol in desno.

Nad tem lahko na eno sliko izrišemo do 40 posebnih ploščic, ki se prosto pomikajo nad ozadjem oziroma oknom. Na eno vrstico ekrana lahko zaradi omejitev grafičnega cevovoda izrišemo le do 10 posebnih ploščic.

Grafični koprocesor teče ločeno od centralne procesne enote. Centralna procesna enota do njega dostopa s pomočjo kontrolnih registrov in dostopom do grafičnega pomnilnika [9].

6.1 Grafični cevovod

Grafični koprocesor simulira delovanje zaslona s katodno cevjo. Na ekran izrisuje vrstico za vrstico. Izris ene vrstice traja 456 urinih period. Na začetku vrstice pregleda `OAM` tabelo za posebne ploščioce, ki bodo izrisane v tej vrstici, kar traja 60 urinih period. Potem točko za točko izriše vrstico.

Tu delujeta prevzemnik ploščic in točkovna vrsta. V točkovni vrsti se hranijo informacije o kodi barve in izviru točke (ozadje, okno ali zaporedna številka posebne ploščice). Ta vrsta vsako urino periodo na LCD porine eno točko, a samo če je v njej več kot osem točk. V nasprotnem primeru se točkovna vrsta ustavi. Preden gre točka na LCD, se določi njena barva s pomočjo paletnih registrov.

Prevzemnik ploščic vsakih 6 urinih period prevzame eno vrstico ploščice iz grafičnega pomnilnika. 2 periodi porabi za branje naslova ploščice. Še po 2 periodi porabi za prevzem enega bajta infomacij iz pomnilika. Te informacije potem postavi na konec vrste.

Če je potreben izris okna, se vrsta pobriše in prevzemnik začne prevzemati ploščice okna. Tu mora vrsta počakati na prevzem podatkov, preden lahko porine na LCD naslednjo točko. Če je potreben izris posebne ploščice, prevzemnik prevzame pravo ploščico in jo pomeša med prvih osem točk v vrsti. Tako lahko dosežemo učinek prozorne barve. To je tudi razlog, da mora biti v vrsti vedno vsaj osem točk.

Izris točk traja vsaj 172 urinih period. Ta čas se lahko podaljša, če moramo na vrstici izrisati tudi okno ali posebne ploščice. Preostali čas grafični koprocesor simulira periodo `HBLANK`, kjer naj bi se žarek v pravem zaslonu s katodno cevjo pomikal na začetek naslednje vrstice.

Ko je izrisanih vseh 144 vrstic, grafični koprocesor simulira periodo `VBLANK`, kjer naj bi se žarek v pravem zaslonu s katodno cevjo pomikal na začetek ekrana. To traja 4560 urinih period, kar je enako času, ki ga grafični koprocesor potrbuje, da izriše 10 vrstic [9, 11].

6.2 Registri

V tabeli 6.1 so našteti registri grafičnega koprocesorja [9].

naslov	ime	opis
0xFF40	LCDC	kontrolni register
0xFF41	STAT	statusni register
0xFF42	SCY	pomik Y ozadja
0xFF43	SCX	pomik X ozadja
0xFF44	LY	trenutna vrstica
0xFF45	LYC	primerjalni register LY
0xFF4A	WY	pomik Y okna
0xFF4B	WX	pomik X okna
0xFF47	BGP	paleta ozadja
0xFF48	OBP0	prva paleta posebnih ploščic
0xFF49	OBP1	druga paleta posebnih ploščic
0xFF46	DMA	register za začetek DMA prenosa

Tabela 6.1: Registri grafičnega koprocesorja.

bit	opis	pomen 1	pomen 0
7	napajanje zaslona	izklopi	vklopi
6	mesto mape okna	0x9800–0x9BFF	0x9C00–0x9FFF
5	vklop okna	izklopi	vklopi
4	mesto ploščic ozadja in okna	0x8800–0x97FF	0x8000–0x8FFF
3	mesto mape ozadja	0x9800–0x9BFF	0x9C00–0x9FFF
2	velikost posebnih ploščic	8x8	8x16
1	vklop posebnih ploščic	izklopi	vklopi
0	vklop ozadja	izklopi	vklopi

Tabela 6.2: Pomen bitov registra LCDC.

Z registrom LCDC kontroliramo grafični koprocesor. Pomeni bitov tega registra so predstavljeni v tabeli 6.2 [9].

V registru STAT so podatki o stanju grafičnega koprocesorja. Poleg tega tu lahko nastavimo prekinitev LCD STAT grafičnega koprocesorja. Prekinitev V-blank se zahteva vedno ob vstopu v način VBLANK ne glede na stanje registra STAT. Pomeni nekaterih bitov tega registra so predstavljeni v tabeli 6.3 [9].

bit	opis	pomen 1	pomen 0
7	neuporabljen		
6	vklop LYC=LY prekinitve	izklopi	vklopi
5	vklop prekinitve načina 2	izklopi	vklopi
4	vklop prekinitve načina 1	izklopi	vklopi
3	vklop prekinitve načina 0	izklopi	vklopi
2	zastavica ujemanja	LYC≠LY	LYC=LY

Tabela 6.3: Pomen nekaterih bitov registra STAT.

V spodnjih dveh bitih registra STAT se nahaja številka načina delovanja grafičnega procesorja. Pomeni načinov [9]:

- Način 0 predstavlja periodo HBLANK. Centralna procesna enota lahko prosto dostopa do grafičnega pomnilnika in tablele OAM.
- Način 1 predstavlja periodo VBLANK. Centralna procesna enota lahko prosto dostopa do grafičnega pomnilnika in tablele OAM.
- V načinu 2 grafični procesor bere iz tabele OAM. Centralna procesna enota ne more dostopati do tabele OAM. Lahko pa dostopa do grafičnega pomnilnika.
- V načinu 3 grafični procesor bere iz tabele OAM in grafičnega pomnilnika. Dostop je centralni procesni enoti onemogočen.

Z registroma SCY in SCX nastavimo postavitev ozadja na zaslonu. V registru LY je številka trenutne vrstice. Ta register se vedno primerja z vsebino registra LYC za določanje zahteve prekinitve. Z registroma WY in WX nastavimo postavitev okna na zaslonu.

Registri BGP, OPB0 in OPB1 vsebujejo podatke o barvah. Vsak od teh registrov določa štiri barve. Po dva bita za vsako barvo, ki si po vrsti od 3 do 0 sledijo od bita 7 do bita 0. Za paleto ozadja so možne štiri barve. Barva 0 je bela, 1 svetlo siva, 2 temno siva in 3 črna. Za paleti za posebne ploščice so možne le tri barve, saj barvna koda 0 predstavlja prozorno.

Z vpisom bajta 0xXX v register DMA začnemo DMA prenos iz naslovov 0xXX00–0xXX9F v OAM tabelo na naslovih 0xFE00–0xFE9F. Prenos traja 160 µs. Medtem lahko centralna procesna enota dostopa le do višjega pomnilnika. Običajno se pred prenosom v višji pomnilnik kopira kratek program, ki počaka, da se prenos DMA konča.

6.3 Pomnilnik

V grafičnem pomnilniku se na naslovih 0x8000–0x97FF nahajajo podatki o ploščicah. Vsaka ploščica je velika 8x8 točk in ima barvno globino 2 bitov. Tako je v grafičnem pomnilniku prostora za 384 ploščic, ki se lahko uporabijo za ozadje, okno ali posebne ploščice. Razdeljene so na dve tabeli. Ena se nahaja na naslovih 0x8000–0x8FFF, kjer so ploščice oštevilčene z nepredznačenimi številkami 0–255. Druga se nahaja na naslovih 0x8800–0x97FF, kjer so ploščice oštevilčene s predznačenimi številkami -128–127.

Vsaka vrstica ploščice zasede 2 bajta. V prvem bajtu so spodnji biti barvih kod, v drugem pa zgornji. Barvne kode se glede na uporabljeno paleto preslikajo v odtenke sive.

Na naslovih 0x9800–0x9BFF in 0x9C00–0x9FFF se nahajata dve mapi, ki predstavljata sliko sestavljeno iz ploščic dimenzije 256x256 točk. Katerakoli mapa se lahko uporablja za prikaz ozadja ali za prikaz okna.

H grafičnemu pomnilniku sodi tudi `OAM` tabela, ki leži na naslovih `0xFE00`–`0xFE9F`. Velika je 160 bajtov in vsebuje informacije za prikaz 40 posebnih ploščic, za vsako ploščico 4 bajte. Pomen bajtov [9]:

- Bajt 0 določa pozicijo Y na ekranu. Vrednost 16 prikaže vrh ploščice na prvi vrstici zaslona.

- Bajt 1 določa pozicijo X na ekranu. Vrednost 8 prikaže levo stran ploščice na prvem stolpcu zaslona.

- Bajt 2 določa številko ploščice. Bajt 0xXX izbere ploščico na naslovu 0x8XX0. Če se ploščice prikazujejo v načinu 8x16, se izbereta dve ploščici. Zgornja z naslovom, kjer ima bajt 2 zadnji bit postavljen na 0, spodnja pa, kjer ima bajt 2 zadnji bit postavljen na 1.

- Bajt 3 določa lastnosti ploščice, kot je prikaz nad/pod ozadjem, zrcaljenje prek Y/X osi in uporabljena paleta.

6.4 Implementacija v emulatorju

Grafični procesor je implementiran v razredu `LCD`. V njem so tabele, ki hranijo grafični pomnilkik, `OAM`, registre in 2D tabelo `screen`, v katero se izrisuje slika. S pomočjo te tabele potem grafična knjižnica izriše sliko na osebnem računalniku. Nekateri člani razreda so vidni na slikah 6.1, 6.2 in 6.3.

```
// Registers
byte LCDCreg; // Mapped to 0xFF40
byte STATreg; // Mapped to 0xFF41
byte SCYreg; // Mapped to 0xFF42
byte SCXreg; // Mapped to 0xFF43
byte LYreg; // Mapped to 0xFF44
byte LYCreg; // Mapped to 0xFF45
byte DMAreg; // Mapped to 0xFF46
byte WYreg; // Mapped to 0xFF4A
byte WXreg; // Mapped to 0xFF4B
byte BGPreg; // Mapped to 0xFF47
byte OBP0reg; // Mapped to 0xFF48
byte OBP1reg; // Mapped to 0xFF49
```

Slika 6.1: Registri grafičnega koprocesorja.

```
byte screen[0x90][0xA0];
byte screenSourceData[0x90][0xA0];
byte VRAM[0x2000];
byte OAM[0x100];
```

Slika 6.2: Implementacija pomnilnika in tabele za izris.

```
byte columnRendering;
word clocksSpentInLine;

word dmaClocksLeft;
int frameCount;
bool screenRedrawn;
```

Slika 6.3: Spremenljivke za pomnenje stanja in prenos informacij.

Glavna funkcija razreda `LCD` je funkcija `run(int)`, ki prejme število urinih

period in emulira delovanje grafičnega koporcesorja. S pomožnimi funkcijami poskrbi za prenos DMA, izris slike, zahtevo prekinitev, nastavljanje statusnih registrov, itn.

Razred implementira tudi funkcije za branje in pisanje pomnilnika, ki so po enakem principu kot v razredu `Memory` razdeljene na te, ki se uporabljajo za interno nastavljanje in branje vrednosti ter te, ki emulirajo pravo branje in pisanje pomnilnika (npr. nedostopen grafični pomnilnik zaradi izrisovanja ali prenosa DMA). Te funkcije uporablja razred `Memory` za branje in pisanje grafičnega pomnilnika.

Podpisi funkcij razreda `LCD` so prikazani na sliki 6.4.

```
void init();
void turnOff();
void run(int);

void dmaTransfer(byte);

void displayBGLineTest();
void dumpVramTiles();
void dumpBackgorundTiles();

void renderBackgroundLine();
void renderWindowLine();
void renderSpritesLine();
void findSpritesOnLine(int*, int, int);

void setStatMode(byte);

void setByte(word, byte);
byte getByte(word);
void writeByte(word, byte);
byte readByte(word);
```

Slika 6.4: Podpisi funkcij razreda `LCD`.

Poglavje 7

Tipkovnica

Game Boyeva tipkovnica ima osem tipk, ki so urejene v 2x4 matriko. Centralna procesna enota enota lahko do stanja tipkovnice dostopa preko registra `P1`, ki je dostopen na naslovu 0xFF00. Funkcije bitov registra so predstavljene v tabeli 7.1 [9].

bit	opis	pomen 1	pomen 0
7	neuporabljen		
6	neuporabljen		
5	izbira funkcijskih tipk	neizbrane	izbrane
4	izbira smernih tipk	neizbrane	izbrane
3	stanje tipk `dol` ali `start`	nepritisnjena	pritisnjena
2	stanje tipk `gor` ali `select`	nepritisnjena	pritisnjena
1	stanje tipk `levo` ali `B`	nepritisnjena	pritisnjena
0	stanje tipk `desno` ali `A`	nepritisnjena	pritisnjena

Tabela 7.1: Pomen bitov registra `P1`.

S pomočje bitov 5 in 4 izberemo katere pritiske tipk želimo zaznavati. Na bitih 3–0 lahko potem preberemo, katere tipke so bile pritisnjene.

Tipkovnica zahteva prekinitev vsakič, ko se na bitih 3–0 pojavi sprememba stanja iz 1 na 0. To je lahko precej nezanesljivo, saj pri pritiskih

tipk prihaja do odbijanja signala in tako večih sprememb signala. Zato se prekinitev tipkovnice uporablja večinoma za izstop iz zelo nizkoenergijskega načina delovanja Game Boya.

7.1 Implementacija v emulatorju

Tipkovnica je implementirana v razredu `Joypad`, ki se nahaja v razredu `Memory`. Prikazan je na sliki 7.1.

```
class Joypad {
private:
    byte keystates;
    byte p1reg;

    bool interruptRequested;
public:
    Joypad();
    ~Joypad();

    void updateKeystates(const Uint8*);

    void setP1reg(byte);
    byte getP1reg();

    void writeP1reg(byte);
    byte readP1reg();

    bool isInterruptRequested();
    void setInterruptRequested(bool);
};
```

Slika 7.1: Razred `Joypad`.

Spremenljivka `keystates` hrani stanje o dejanskih pritiskih tipk na tipkovnici. Te posodobimo s pomočjo funkcije `updateKeystates(const Uint8*)`, ki poskrbi tudi za nastavljanje prekinitvine zahteve.

Za branje in pisanje registra `P1` so implementirane funkcije za interno nastavljanje vrednosti in funkcije za simuliranje pravih branj in pisanj. Funkcija `readP1reg()` je prikazana na sliki 7.2.

```
byte Joypad::readP1reg() {
    byte pressedkeys = 0xFF;
    if((p1reg & 0x20) == 0) pressedkeys &= keystates >> 4;
    if((p1reg & 0x10) == 0) pressedkeys &= keystates;
    return p1reg & 0xF0 | pressedkeys & 0x0F;
}
```

Slika 7.2: Funkcija `readP1reg()`.

Poglavje 8

Časovnik

Game Boy ima notranji 16-bitni časovnik, ki se povečuje z uro procesorja. S tem časovnikom si pomaga pri povečevanju glavnega časovnika. Za nadzor uporablja štiri registre [9]:

- `DIV` (0xFF04), ki je slika zgornjih osmih bitov notranjega časovnika. Pisanje na ta naslov postavi notranji časovnik na 0.
- `TIMA` (0xFF05) je 8-bitni glavni števec, ki se povečuje odvisno od stanja kontrolnega registra. Ko pride do preliva, se zahteva prekinitev. Takrat se v števec naloži vrednost registra `TMA`.
- `TMA` (0xFF06) je 8-bitni register. Njegova vrednost se naloži v `TIMA` ob prelivu.
- `TAC` (0xFF07) je kontrolni register. Spodnja dva bita določata frekvenco povečevanja `TIMA` registra. Bit 2 pa omogoči povečevanje `TIMA` registra.

Hitrost povečevanja `TIMA` registra se določi s pomočjo notranjega časovnika. Spodnja dva bita `TAC` registra izbereta en bit notranjega časovnika. Logični *in* tega bita in bita 2 registra `TAC` gre čez detektor negativne fronte. Ko je zaznana negativna fronta, se poveča vrednost `TIMA`. Možne frekvence so naštete v tabeli 8.1.

bit 1	bit 0	frekvenca
0	0	4096 Hz
0	1	262144 Hz
1	0	65536 Hz
1	1	16384 Hz

Tabela 8.1: Možne frekvence glede na stanje bitov v registru `TAC`.

8.1 Implementacija v emulatorju

Časovnik je implementiran v razredu `Timer`, ki se nahaja v razredu `Memory`. Prikazan je na sliki 8.1.

```
class Timer {
private:
    word divReg;     // Mapped to 0xFF04;
    byte timaReg;    // Mapped to 0xFF05
    byte tmaReg;     // Mapped to 0xFF06
    byte tacReg;     // Mapped to 0xFF07

    bool interruptRequested;
public:
    Timer();
    ~Timer();

    void run(int);

    void setByte(word, byte);
    byte getByte(word);
    void writeByte(word, byte);
    byte readByte(word);

    bool isInterruptRequested();
    void setInterruptRequested(bool);
};
```

Slika 8.1: Razred `Timer`.

Vsebuje vse registre časovnika. Glavna je funkcija `run(int)`, ki emulira

delovanje časovnika za določeno število urinih period. Skrbi za povečevanje notranjega časovnika, registra `TIMA` in zahtevanje prekinitev. Prikazana je na sliki 8.2.

```
void Timer::run(int clocks) {
    byte shiftValue;
    switch(tacReg & 0x03) {
        case 0: shiftValue = 9; break;
        case 1: shiftValue = 3; break;
        case 2: shiftValue = 5; break;
        case 3: shiftValue = 7;
    }

    for(int i = 0; i < clocks; i++) {
        byte oldDivBit = (divReg & shiftValue) >> shiftValue;
        divReg++;
        byte newDivBit = (divReg & shiftValue) >> shiftValue;
        byte oldTima = timaReg;

        // TIMA is increased
        if(oldDivBit == 1 && newDivBit == 0
          && (tacReg & 0x4) != 0) {
          timaReg++;
          if(timaReg < oldTima) { // Interrupt triggered
              interruptRequested = true;
              timaReg = tmaReg;
          }
        }
    }
}
```

Slika 8.2: Funkcija `run(int)`.

Za branje in pisanje registrov časovnika so implementirane funkcije za interno nastavljanje vrednosti in funkcije za simuliranje pravih branj in pisanj.

Poglavje 9

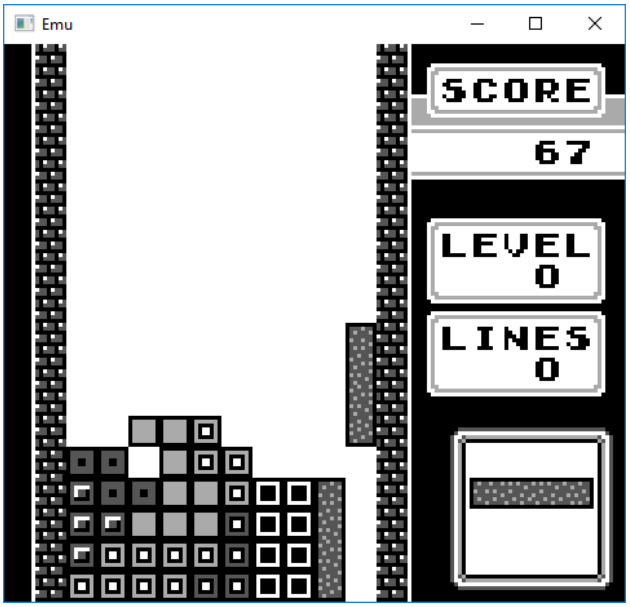
Delovanje emulatorja

Vstopna točka emulatorja je funkcija `main()`, ki prejme en argument, ki vsebuje pot do datoteke z vsebino ROM-a. Potem konstruira razred `Render`, ki poskrbi za inicializacijo tipkovnice, okna, grafike in razreda `Board`, ki vsebuje vse podsisteme Game Boya. Konstruktor razreda `Board` najprej konstruira razred `Memory`, ki kot argumet dobi pot do datoteke z vsebino ROM-a. Ta se glede na vsebino datoteke odloči za ustrezen krmilnik pomnilniških modulov in pripravi pomnilnik. Konstruktor razreda `Board` nato še poveže razrede med sabo (npr. nastavi kazalec `LCD* lcd` v razredu `Memory`).

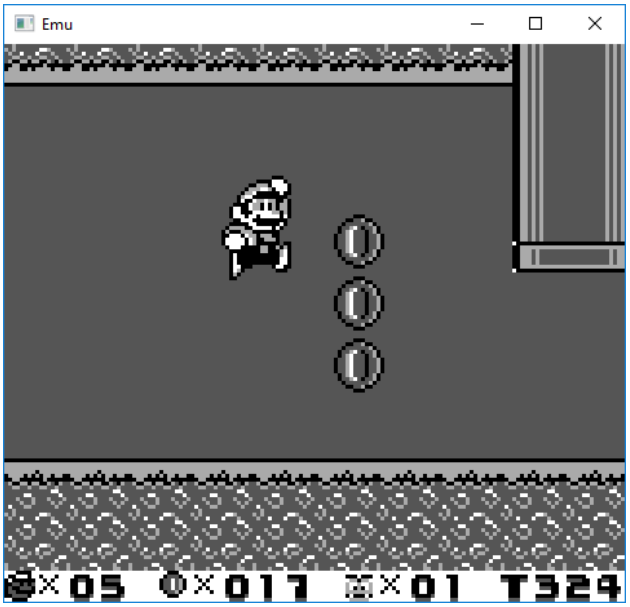
Ko so vsi podsistemi inicializirani, se program nadaljuje v glavni zanki, ki se nahaja v razredu `Render`. Tu je funkicja `mainLoop()`, ki skrbi za branje tipk s tipkovnice, nadzor razreda `Board` in izris na ekran, ko je razred `LCD` v svoj predpomnilnik izrisal eno sličico. Po vsaki iteraciji te zanke emulator spi toliko časa, da se hitrost izvajanja programa ujema s hitrostjo na pravem Game Boyu. Ker so današnji osebni računalniki veliko hitrejši od Game Boya, lahko emulacija teče v realnem času. Na računalniku s procesorjem Intel i7 2600 smo z merjenjem časa ugotovili, da emulator spi v povprečju 10% časa.

Na emulatorju smo testirali kar nekaj iger. Večino jih deluje pravilno in jih lahko igramo. Nekaj iger se odpre le do neke točke, nato pa pride do napake, ki onemogoči nadaljevanje izvajanja. Na slikah 9.1, 9.2, 9.3 in 9.4

je prikazanih nekaj delujočih iger.

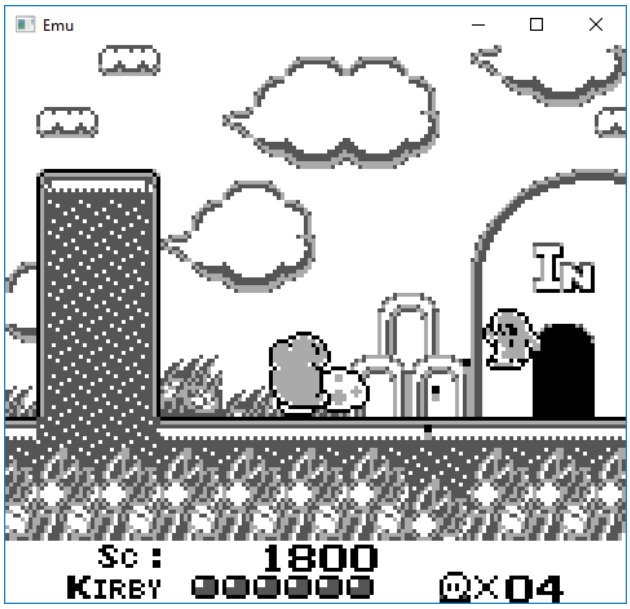


Slika 9.1: Igra Tetris.



Slika 9.2: Igra Super Mario Land 2: 6 Golden Coins.

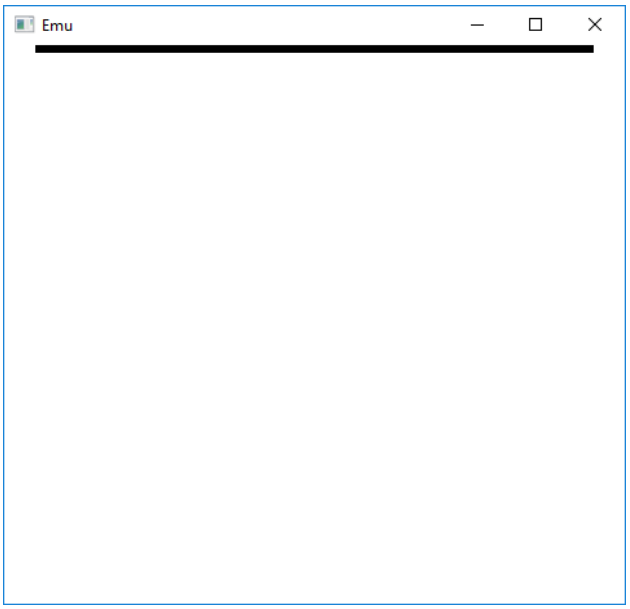
Slika 9.3: Igra Pokemon Red.



Slika 9.4: Igra Kirby's Dream Land.

Nekaj iger ne deluje zaradi nepodprtega krmilnika pomnilniških modulov. Pri večini nedelujočih iger pa gre bodisi za hrošče v emulatorju ali pa za hrošče v pravem Game Boyu, ki v emulatorju niso pravilno emulirani. Kar nekaj iger je takih, ki se zanašajo na te nepravilnosti Game Boya in delujejo normalno na pravem sistemu, emulatorji pa imajo z njimi težave.

Ena od znanih iger, ki povzroča težave, je Pinball Fantasies, ki na večini emulatorjev ne deluje pravilno. To naj bi bila posledica neujemanja časovnih intervalov [4]. Na izdelanem emulatorju se igra priže in ekran enkrat utripne. Potem pa se igra ustavi na skoraj praznem ekranu, ki je prikazan na sliki 9.5.



Slika 9.5: Izvajanje igre Pinball Fantasies obtiči kmalu po zagonu igre.

Poglavje 10

Zaključek

Glavna cilja pri izdelavi diplomske naloge sta bila delujoč emulator in učenje jezika C++.

Pred začetkom programiranja smo prebrali vso dokumentacijo, ki jo lahko najdemo. Nato smo sestavili načrt arhitekture, ki se je med programiranjem še kar nekajkrat spremenil. Podsisteme smo implementirali enega za drugim in jih nato povezali v celoto. S programiranjem uporabnega programa smo poglobili znanje jezika C++. Tako je bil dosežen drugi cilj.

Nastali emulator lahko igra precej iger, ki so bile narejene za originalni Game Boy. Seveda bi ga lahko še izboljšali z odstranjevanjem manjših hroščov in nepravilnosti emulacije. Prav tako emulatorju manjka še podpora za zvok in serijski vmesnik, ki bi ga lahko realizirali preko omrežja. Te dodatne funkcionalnosti se lahko enostavno dodajo v prihodnosti, saj je arhitektura programa sestavljena zelo modularno. Tako je bil uresničen tudi prvi cilj diplomske naloge.

Literatura

[1] Emulator. Dosegljivo: https://en.wikipedia.org/wiki/Emulator. [Dostopano: 10. 7. 2018].

[2] Game boy. Dosegljivo: https://en.wikipedia.org/wiki/Game_Boy. [Dostopano: 10. 7. 2018].

[3] Gameboy cpu (lr35902) instruction set. Dosegljivo: http://pastraiser.com/cpu/gameboy/gameboy_opcodes.html. [Dostopano: 15. 7. 2018].

[4] "holy grail"bugs in emulation. Dosegljivo: https://mgba.io/2017/05/29/holy-grail-bugs. [Dostopano: 20. 8. 2018].

[5] Izvorna koda. Dosegljivo: https://github.com/ExPie/GameBoyEmu. [Dostopano: 20. 8. 2018].

[6] List of best-selling game boy video games. Dosegljivo: https://en.wikipedia.org/wiki/List_of_best-selling_Game_Boy_video_games. [Dostopano: 10. 7. 2018].

[7] List of best-selling game consoles. Dosegljivo: https://en.wikipedia.org/wiki/List_of_best-selling_game_consoles. [Dostopano: 10. 7. 2018].

[8] Nintendo. *Game Boy Programming Manual*. 1999.

[9] Pan docs. Dosegljivo: http://bgb.bircd.org/pandocs.htm. [Dostopano: 10. 7. 2018].

[10] Simple directmedia layer. Dosegljivo: https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer. [Dostopano: 15. 7. 2018].

[11] Michael Steil. The ultimate game boy talk. Dosegljivo: https://www.youtube.com/watch?v=HyzD8pNlpwI. [Dostopano: 15. 7. 2018].

[12] Technical data. Dosegljivo: https://www.nintendo.co.uk/Support/Game-Boy-Pocket-Color/Product-information/Technical-data/Technical-data-619585.html. [Dostopano: 12. 7. 2018].