



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSI NYELVEK ÉS FORDÍTÓPROGRAMOK

TANSZÉK

NES játékkonzol emulációja Haskellben

Témavezető:

Poór Artúr
egyetemi tanársegéd

Szerző:

Suhajda Tamás József
programtervező informatikus BSc

Budapest, 2020

Az eredeti szakdolgozati / diplomamunka témabejelentő helye.

Tartalomjegyzék

1. Bevezetés	5
2. Felhasználói dokumentáció	7
2.1. A feladat ismertetése	7
2.2. Minimum rendszerkövetelmények	8
2.3. Telepítés	8
2.4. Indítás	8
2.5. Kompatibilis kazetták	8
2.6. Főmenü	9
2.6.1. Kazetta kiválasztása	10
2.6.2. Irányítási beállítások megtekintése	10
2.7. Játék alatt elérhető funkciók	11
2.7.1. Irányítás	11
2.7.2. Mentés	12
2.7.3. Betöltés	13
2.7.4. A játék megállítása	14
2.7.5. Visszatérés a főmenübe	14
2.7.6. Képernyőképek	15
3. Fejlesztői dokumentáció I: A NES működésének ismertetése	16
3.1. A NES felépítése	16
3.2. Órajel-frekvenciák	17
3.3. A központi feldolgozóegységhez kapcsolódó fogalmak	17
3.3.1. Opcód	17
3.3.2. Regiszterek	19
3.3.3. Memórialap	19

3.3.4. Hívási verem	19
3.3.5. Megszakítás	20
3.3.6. Opkód argumentum helye, fajtája	20
3.3.7. Címzési módok	21
3.3.8. Memóriatérkép	22
3.3.9. Memóriatükrözés	23
3.3.10. Státusz flagek	23
3.3.11. Utasításkészlet	24
3.4. Kazetták	25
3.4.1. Az iNES fájlformátum	25
3.5. A képfeldolgozó egység	26
3.5.1. Színpalletta	26
3.5.2. Paletta indexek	26
3.5.3. Alakzattáblázat	27
3.5.4. Rétegek	28
3.5.5. Névtáblázatok	28
3.5.6. Attribútumtáblázatok	29
3.5.7. Háttéreltolás	29
3.5.8. OAM (Object Attribute Memory)	29
3.5.9. Regiszterek	30
3.5.10. Memóriatérkép	31
3.5.11. A háttér kirajzolásának egyszerű algoritmusa	32
3.5.12. Sprite réteg kirajzolása	35
4. Fejlesztői dokumentáció II: Megvalósítás	36
4.1. A feladat specifikációja	36
4.2. Fejlesztői környezet	36
4.3. Megvalósítási terv	37
4.3.1. NES emuláció	37
4.3.2. Felhasználói felület	37
4.4. Az emulációt magába záró monád	39
4.5. Adatrepräsentáció	39
4.6. Mentés és betöltés	40

4.7.	Főmodulok áttekintése	41
4.7.1.	Nes.Cartridge.Memory	41
4.7.2.	Nes.Cartridge.INES.Parser	41
4.7.3.	Nes.Cartridge.Mappers	42
4.7.4.	Nes.Emulation.Monad	42
4.7.5.	Nes.CPU.Emulation	43
4.7.6.	Nes.PPU.Emulation	44
4.7.7.	Nes.Emulation.Controls	45
4.7.8.	Nes.Emulation.MasterClock	45
4.7.9.	Communication	46
4.7.10.	Emulator.JoyControls	46
4.7.11.	Emulator.Framerate	46
4.7.12.	Emulator.CrtShader	46
4.7.13.	Emulator.Window	47
4.8.	Egy processzorutasítás végrehajtása	48
4.9.	A CPU és a PPU szinkronizációja	49
4.10.	A grafikus felhasználói felület	49
4.10.1.	Állapotmegjelenítés	49
4.10.2.	Állapotátmenetek	50
4.11.	Teljesítmény	50
5.	Tesztelés	51
5.1.	CPU tesztek	51
5.1.1.	Nestest	51
5.1.2.	Instr_test_v5	51
5.1.3.	Instr_Misc	52
5.2.	PPU tesztek	52
5.2.1.	PPU_vbl_nmi	52
5.2.2.	PPU_sprite_hit és PPU_sprite_overflow	52
6.	Összegzés	53
	Irodalomjegyzék	54

Ábrajegyzék	55
Táblázatjegyzék	56
Forráskódjegyzék	57

1. fejezet

Bevezetés

A játékkonzol-emulátorok feladata, hogy egy kompatibilitási réteget képezzenek a modern x86 és ARM architektúrájú processzorok, valamint az elavult konzolokra megjelent játékok között, hogy azokat bárki zavartalanul élvezhesse a konzol birtoklása nélkül. Az emulációt végző programnak szoftveresen kell megvalósítania az eredeti konzol számítási egységei által nyújtott primitív utasításokat és a komponensek közötti kommunikációt, hogy a játékok az elvárt viselkedés szerint működjenek.

Az emulátorok világában a hardverközeli, elsősorban teljesítményre kihegyezett nyelvek használata (pl. C++) az elterjedt, mivel ezen a területen a program gyorsasága kulcsfontosságú. Ezeknél a nyelveknél az explicit memória kezelés és a vékony absztrakciós réteg megkönnyíti az optimalizációt, azonban ennek a kód átláthatósága látja a kárát. Ezzel szemben a funkcionális nyelvek erős kifejezőképessége és moduláris felépítést előnyben részesítő paradigmája az emulátorfejlesztésnél számos helyzetben könnyítik meg a programozó dolgát. A szakdolgozatom célja, hogy korszerű eszközök segítségével egy fejlesztőbarát, de mégis hatékony emulátort implementáljak funkcionális nyelven. A megfelelő teljesítménnyről a Haskell nyelv egyeduralkodó fordítója, a GHC gondoskodik, ami az egyik legfejlettebb fordítóprogram, ami funkcionális nyelvhez készült. Agresszív optimalizálási stratégiáján túl az is mellette szól, hogy a legfrissebb kiadása immár tartalmaz egy valós idejű alkalmazásokhoz szánt, alacsony késleltetésű szemétgyűjtőt.

A Nintendo Entertainment System (NES) generációjának legsikeresebb konzolja, több mint 61 millió darab kelt el belőle világszerte. Emiatt kezdettől fogva nagy volt az igény az emulátorokra és mára a hardver nem publikus részei is fel lettek

térképezve. Választásom azért erre a rendszerre esett, mert az internetről elérhető dokumentációk és leírások birtokában nincs szükség a konzol beszerzésére, a hardver viselkedésének felderítésére.

2. fejezet

Felhasználói dokumentáció

2.1. A feladat ismertetése



2.1. ábra. Nintendo Entertainment System (1985)

A program feladata a NES kazetták futtatása. A felhasználó grafikus felület segítségével kiválaszthatja a futtatni kívánt, iNES formátumú kazetta fájlt, majd annak betöltése után az emulátor belekezd a videókimenet előállításába (a hangfeldolgozást nem emulálja a program). Bemeneti eszközként billentyűzet vagy kontroller használható. A futtatás során lehetőség van az emulátor állapotának fájlként való mentésére és betöltésére. Az emulációt többféle módon is befolyásolhatja a felhasználó, amikbe beletartozik a játék szüneteltetése, adott mennyiségű CPU utasítás végrehajtása, valamint a teljes képkockánként történő léptetés.

A programot azoknak ajánlom, akik egy letisztult kezelőfelülettel rendelkező, több platformon is elérhető NES emulátorral szeretnék játszani kedvenc játékaikat.

2.2. Minimum rendszerkövetelmények

Processzor: Intel Core 2 Duo E8400 @ 3.0 GHz

Memória: 2 GB DDR2 @ 800 MHz

Videókártya: OpenGL 4.3 kompatibilis videókártya

Tárhely: 161 MB

Operációs rendszer: Linux, Windows

A program a fenti konfigurációt történő tesztelés során problémamentesen működött.

2.3. Telepítés

- Windows: A programot egy telepítőfájl segítségével telepíthetjük.
- Linux:

```
1 | $ tar xvf pure-nes-1.0.tar.gz
2 | $ cd pure-nes-1.0
3 | $ ./configure
4 | $ make
```

2.4. Indítás

- Windows: Kattintsunk a program parancsikonjára a Start Menüben vagy az asztalon.
- Linux:

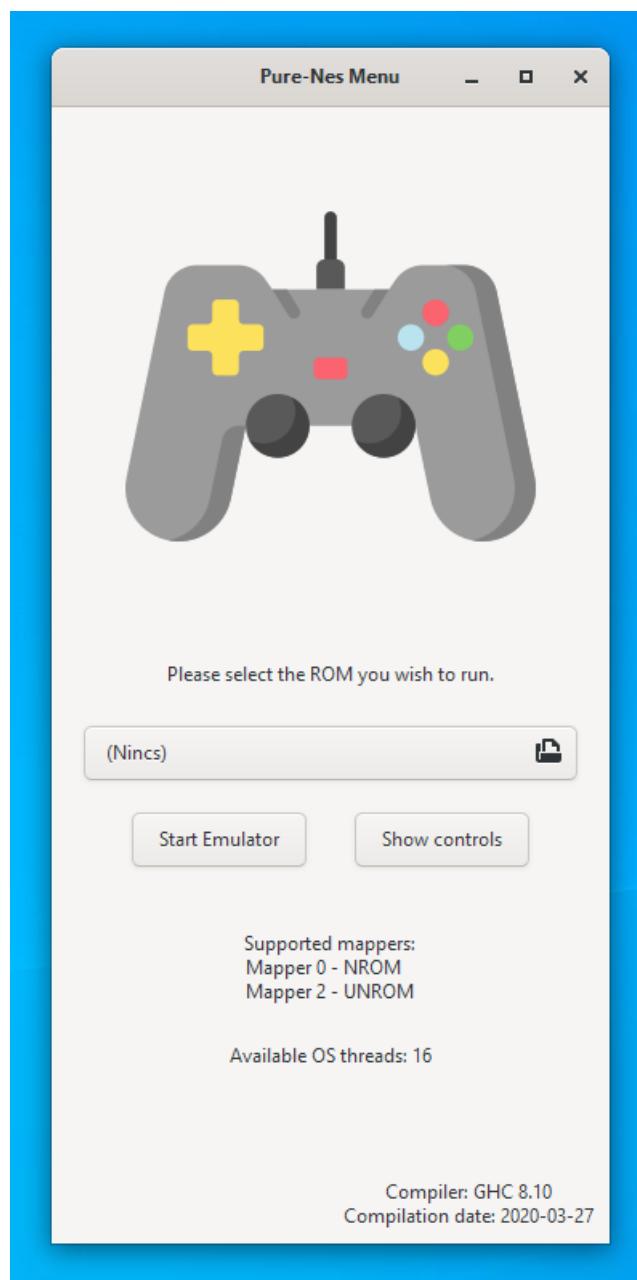
```
1 | pure-nes-1.0$ stack exec pure-nes
```

2.5. Kompatibilis kazetták

A NES megtervezésekor fontos cél volt, hogy a konzol életciklusa hosszú legyen, de ezt nem volt könnyű gazdaságos módon elérni. Azt a megoldás találták ki, hogy a

kazetta a játék mellett tartalmazzon speciális integrált áramköröket, amik igény szerint bővíthették a hardveres erőforrásokat. A kiegészítőegységek egy konkrét összeállítását *mapper*-nek nevezzük. A későbbi játékok előszeretettel használták ki ezt a lehetőséget. A legtöbb speciális áramkör a megnövelt háttértár kezeléséhez volt szükséges, de akadt olyan is, amelyik további hangcsatornákat adott hozzá a hangfeldolgozó egységhez. Az emulátorom jelenleg a 0, 2 és 3 azonosítójú mapper-eket használó kazettákat támogatja, ezáltal 471 játékot képes elindítani [**ROM lista**].

2.6. Főmenü



2.6.1. Kazetta kiválasztása

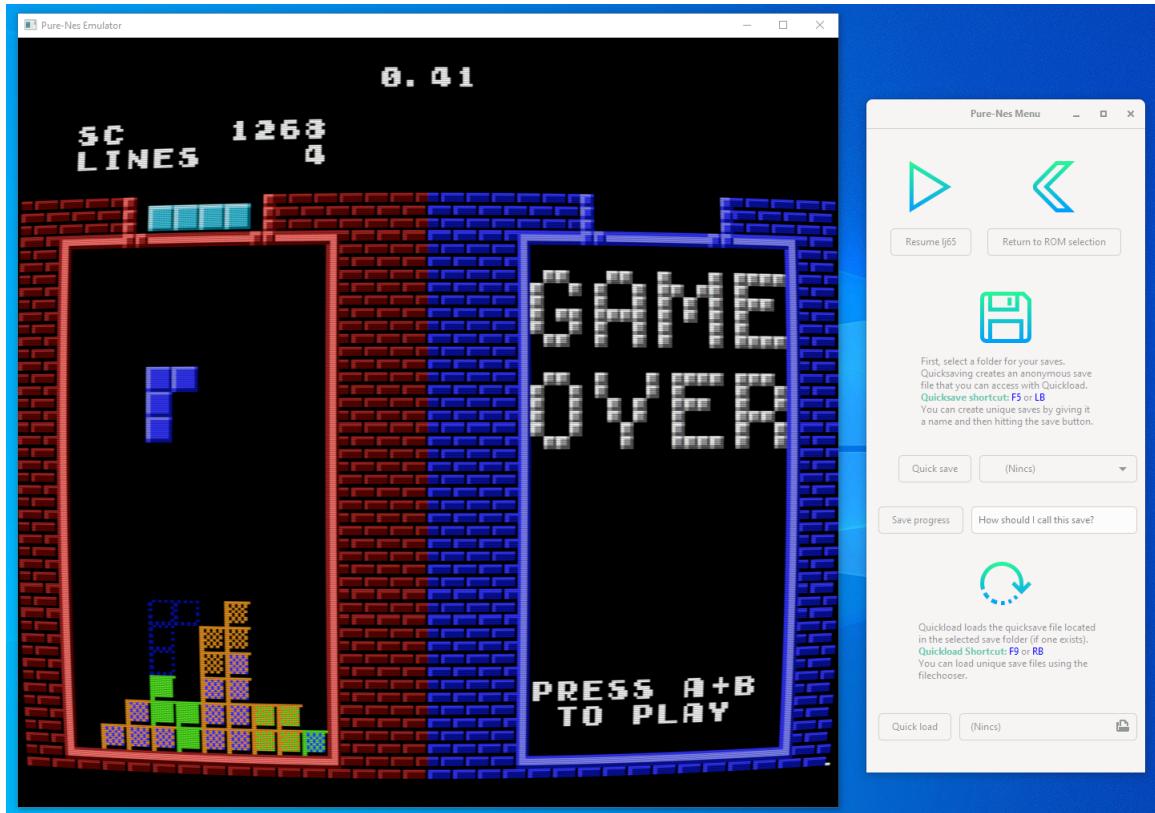
A fájlkiválasztó segítségével adjuk meg a futtatni kívánt kazettát vagy mentést, majd nyomjunk a *Start Emulator* gombra. A program hibaüzenettel jelzi, ha a kazetta nem kompatibilis vagy a fájl formátuma nem megfelelő.

2.6.2. Irányítási beállítások megtekintése

A főmenü *Show controls* gombjára kattintva megtekinthetjük a gombhozzárendeléseket.



2.7. Játék alatt elérhető funkciók



2.2. ábra. A felhasználói felület játék közben

2.7.1. Irányítás

NES kontroller gomb	Billentyű	Kontroller
A	1	2. gomb
B	2	3. gomb
Select	3	0. gomb
Start	4	1. gomb
Up	Fel nyíl	DPad Fel
Down	Lefele nyíl	DPad Le
Left	Balra nyíl	DPad Bal
Right	Jobbra nyíl	DPad Jobb

2.1. táblázat. Játékok irányításához használt gombok

A 2.1 táblázatban látható módon vannak a fizikai gombok (a billentyűzeten vagy a kontrolleren) az emulált NES virtuális gombjainak megfeleltetve. Például, ha a játékon belül a *Select* gombot szeretnénk lenyomni, akkor a billentyűzeten a 3-as, vagy a kontrolleren a 0. gombot kell lenyomnunk. A Joystick vezérlők gombkiosztása elszokott térfi, ami azt jelenti, hogy a fizikailag ugyanott található gomboknak más az azonosítója. A felhasználó feladata, hogy szükség esetén egy másik programmal módosítsa eszközének kiosztását úgy, hogy az megegyezzen az emulátor szerint elvárta.

Az 2.2 táblázatban felsorolt gombokkal lehet az emulációt irányítani és testre szabni. A szüneteltetés leállítja az emulált komponenseket és elérhetővé teszi a léptetési funkciókat. Képesek vagyunk a teljes rendszert egy processzorutasítással léptetni. minden léptetésnél láthatjuk a sztenderd kimeneten, hogy milyen utasítások fognak soron következni. Ha ennél gyorsabb ütemben szeretnénk léptetni az emulációt, akkor használjuk a képkockánti léptetést. A képre alapértelmezett rákerül egy katódsugárcsöves megjelenítőket szimuláló effektus, de ezt a funkciót bármikor kikapcsolhatjuk.

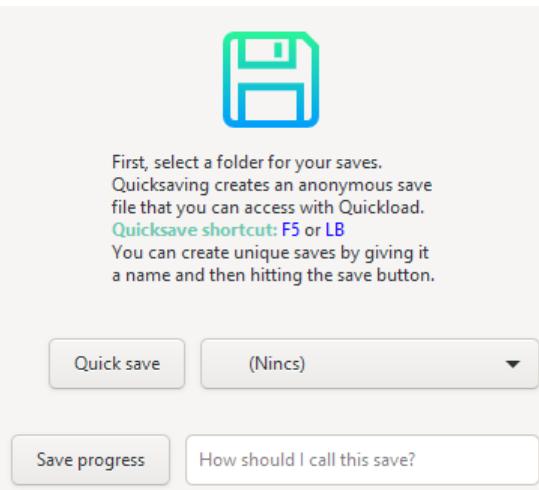
Emulátor funkció	Billentyű
Teljes képernyős nézet ki/be	R
Katódsugárcsöves képernyő-effektus ki/be	T
Szüneteltetés ki/be	Space
Egy processzorutasítás végrehajtása (szüneteltetés alatt)	C
Léptetés a következő képkockára (szüneteltetés alatt)	F

2.2. táblázat. Az emuláció vezérlése

2.7.2. Mentés

A felhasználónak ki kell jelölnie egy mappát, amit a program a mentések kezelésére használ. A program hibaüzenetet ad, ha enélkül próbálunk menteni vagy gyors betöltést végrehajtani.

A mentések a virtuális gép teljes állapota mellett a praktikusság érdekében a játék másolatát is tartalmazzák, tehát egy mentés betöltéséhez nem kell megőrizni a játék eredeti példányát.



Gyorsmentés

A gyorsmentési funkcióval egy gombnyomásra elmenthetjük állásunkat. A mentés *quick.purenes* néven jön létre a mappában. Ha már létezik ilyen fájl, az felül lesz írva. Mentés létrehozása: **Quicksave** gomb vagy **F5** billentyű vagy a **4.** gomb a kontrolleren. A mentés sikereségét egy pipa vagy kereszt jelzi a kezelőfelületen a mentés ikon mellett. Ha a kontroller rendelkezik rezgőmotorral, akkor a sikeres mentés rezgéssel is jelezve lesz.

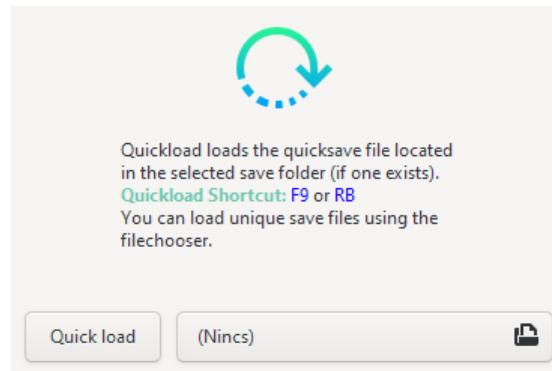
Egyedi mentés létrehozása

Adhatunk nevet a mentéseknek, ehhez írjuk be a nevet a szövegmezőbe és nyomjunk a *Save* gombra. A mentés *{név}.purenes* néven jön létre a mappában. A *quick* nevet nem adhatjuk a mentésnek.

2.7.3. Betöltés

Gyorsbetöltés

A gyorsbetöltési funkció játék közben érhető el, miután kiválasztottuk a mentéseket tároló mappát. A **Quickload** gombbal, vagy az **F9** billentyűvel, vagy a kontroller **6.** gombjával visszatölthetjük a korábban létrehozott gyorsmentést. A betöltés sikeresége a mentéshez hasonló módon jelenik meg a kezelőfelületen. Sikertelen betöltés abból adódhat, hogy a mappában nem található gyorsmentés, vagy a mentési fájl sérült. A sikeres betöltést itt is rezgés fogja követni.



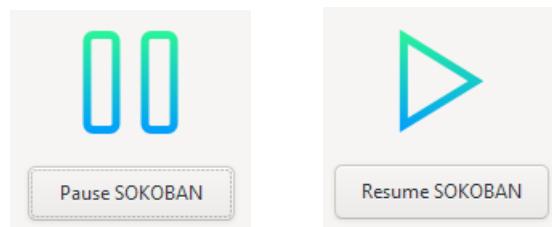
Egyedi mentés betöltése

Ez a funkció arra szolgál, hogy a gyorsmentésen kívül más mentéseket is be tudjunk tölteni. A fájlkiválasztó segítségével válasszuk ki a betölteni kívánt mentést.

Figyelem: Mindkét betöltési módnál az aktuális munkamenet elveszik. Ha ezt el szeretnénk kerülni, akkor mentsünk előtte.

2.7.4. A játék megállítása

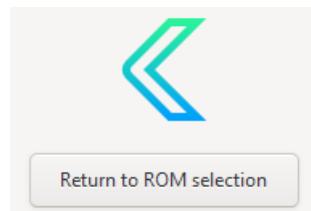
A játék futása bármikor felfüggeszthető a *Pause* gombbal, majd ezután folytat-ható a *Resume* gombbal.



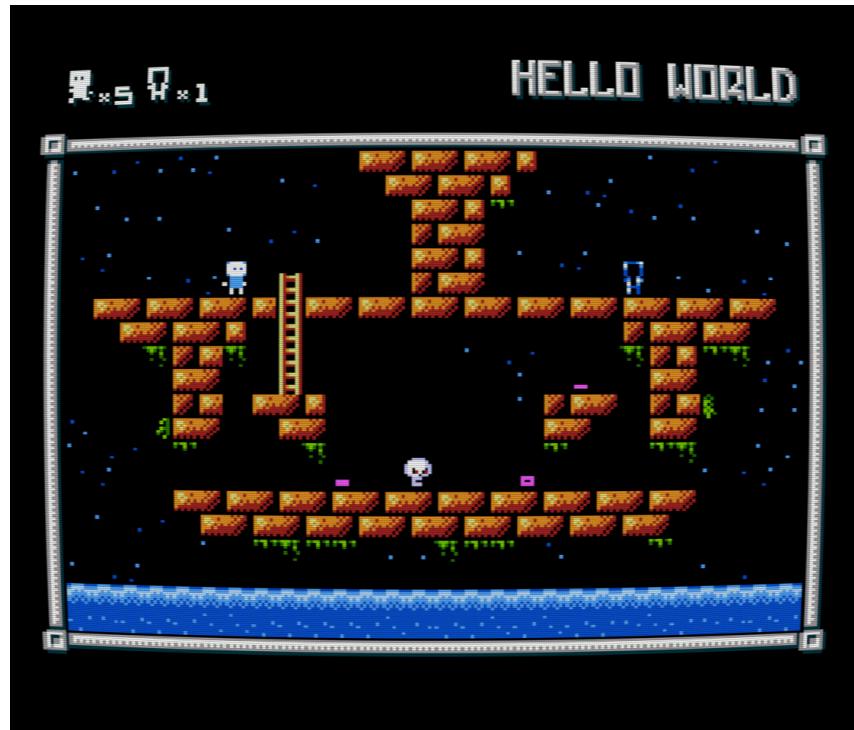
2.7.5. Visszatérés a főmenübe

A *Return To ROM selection* gomb bezárja a játékot és a felhasználói felületen visszanavigál minket a főmenübe.

Figyelem: Az állásunk elveszik, ha előtte nem mentünk.



2.7.6. Képernyőképek



2.3. ábra. Alter Ego

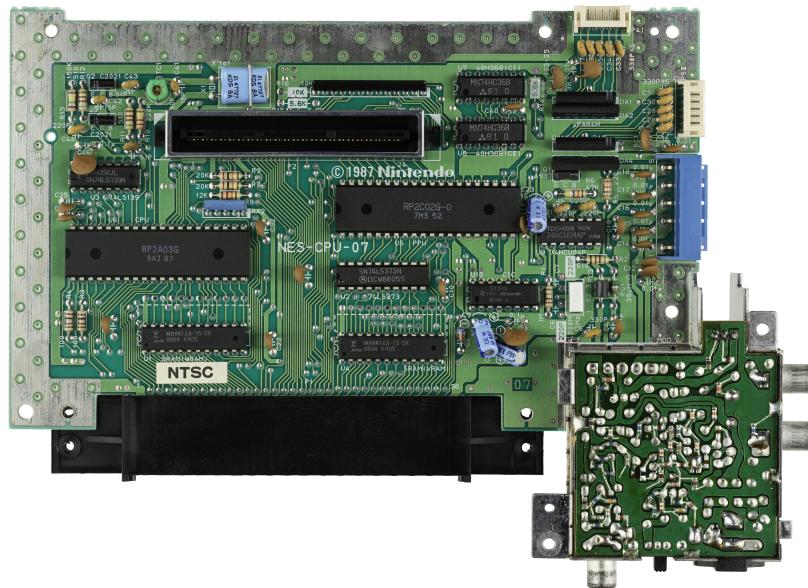


2.4. ábra. Lawn Mover

3. fejezet

Fejlesztői dokumentáció I: A NES működésének ismertetése

3.1. A NES felépítése



3.1. ábra. A NES alaplapja

A NES emulációjához a következő komponenseket kell megismernünk és megvalósítanunk:

Ricoh RP2A03:

A hangchipet és központi feldolgozóegységet (CPU) tartalmazó integrált áram-

kör. Utóbbi nem más, mint az Apple II-ben és Commodore 64-ben használt 8-bites MOS Technology 6502.

Ricoh RP2C02:

A képfeldolgozó egység, rövid nevén PPU (Picture Processing Unit).

NROM, UNROM és CNROM:

Az emulátor által támogatott három kazettatípus.

Sztenderd NES kontroller:

A konzol alapértelmezett beviteli eszköze.

3.2. Órajel-frekvenciák

A párhuzamosan működő komponenseket az órajelek hangolják össze. Az órajel-frekvencia határozza meg, hogy egy másodperc alatt hány atomi műveletet végez el egy komponens. minden komponens rendelkezik egy saját órajelfrekvenciával, amit egy központi órajelből származtatnak.

- Központi órajel-frekvencia: $f = \frac{236.25 \text{ MHz}}{11} \sim 21.477272 \text{ MHz}$
- CPU órajel-frekvencia: $\frac{f}{12} \sim 1.789773 \text{ MHz}$
- PPU órajel-frekvencia: $\frac{f}{4} \sim 5.369318 \text{ MHz}$

3.3. A központi feldolgozóegységhez kapcsolódó fogalmak

Megjegyzés. A hexadecimális értékeket \$ prefix-el jelölöm.

3.3.1. Opkód

Egy opkód a 6502 esetében csupán egyetlen bájt, amiből az utasításdekódoló egyértelműen meg tudja határozni a végrehajtandó utasítást és annak címzési módját. Ezt a hozzárendelést az opkódmátrix írja le. Az emulációhoz emellett azt is tárolni kell az opkódmátrixban, hogy a végrehajtandó művelet hány CPU-órajel alatt fejeződik be, ugyanis csak ennek ismeretében tudjuk a CPU-t és a PPU-t precízen egymáshoz szinkronizálni.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	BRK 7	ORA izx 6	KIL	SLO izx 8	NOP zp 3	ORA zp 3	ASL zp 5	SLO zp 5	PHP 3	ORA imm 2	ASL 2	ANC imm 2	NOP abs 4	ORA abs 4	ASL abs 6	SLO abs 6
1x	BPL rel 2*	ORA izy 5*	KIL	SLO izy 8	NOP zpx 4	ORA zpx 4	ASL zpx 6	SLO zpx 6	CLC 2	ORA aby 4*	NOP 2	SLO aby 7	NOP abx 4*	ORA abx 4*	ASL abx 7	SLO abx 7
2x	JSR abs 6	AND izx 6	KIL	RLA izx 8	BIT zp 3	AND zp 3	ROL zp 5	RLA zp 5	PLP 4	AND imm 2	ROL 2	ANC imm 2	BIT abs 4	AND abs 4	ROL abs 6	RLA abs 6
3x	BMI rel 2*	AND izy 5*	KIL	RLA izy 8	NOP zpx 4	AND zpx 4	ROL zpx 6	RLA zpx 6	SEC 2	AND aby 4*	NOP 2	RLA aby 7	NOP abx 4*	AND abx 7	ROL abx 7	RLA abx 7
4x	RTI 6	EOR izx 6	KIL	SRE izx 8	NOP zp 3	EOR zp 3	LSR zp 5	SRE zp 5	PHA 3	EOR imm 2	LSR	ALR imm 2	JMP abs 3	EOR abs 4	LSR abs 6	SRE abs 6
5x	BVC rel 2*	EOR izy 5*	KIL	SRE izy 8	NOP zpx 4	EOR zpx 4	LSR zpx 6	SRE zpx 6	CLI 2	EOR aby 4*	NOP 2	SRE aby 7	NOP abx 4*	EOR abx 4*	LSR abx 7	SRE abx 7
6x	RTS 6	ADC izx 6	KIL	RRA izx 8	NOP zp 3	ADC zp 3	ROR zp 5	RRA zp 5	PLA 4	ADC imm 2	ROR 2	ARR imm 2	JMP ind 5	ADC abs 4	ROR abs 6	RRA abs 6
7x	BVS rel 2*	ADC izy 5*	KIL	RRA izy 8	NOP zpx 4	ADC zpx 4	ROR zpx 6	RRA zpx 6	SEI 2	ADC aby 4*	NOP 2	RRA aby 7	NOP abx 4*	ADC abx 4*	ROR abx 7	RRA abx 7
8x	NOP imm 2	STA izx 6	NOP imm 2	SAX izx 6	STY zp 3	STA zp 3	STX zp 3	SAX zp 3	DEY 2	NOP imm 2	TXA 2	XAA imm 2	STY abs 4	STA abs 4	STX abs 4	SAX abs 4
9x	BCC rel 2*	STA izy 6	KIL	AHX izy 6	STY zpx 4	STA zpx 4	STX zpy 4	SAX zpy 4	TYA 2	STA aby 5	TXS 2	TAS aby 5	SHY abx 5	STA abx 5	SHX aby 5	AHX aby 5
Ax	LDY imm 2	LDA izx 6	LDX imm 2	LAX izx 6	LDY zp 3	LDA zp 3	LDX zp 3	LAX zp 3	TAY 2	LDA imm 2	TAX 2	LAX imm 2	LDY abs 4	LDA abs 4	LDX abs 4	LAX abs 4
Bx	BCS rel 2*	LDA izy 5*	KIL	LAX izy 5*	LDY zpx 4	LDA zpx 4	LDX zpy 4	LAX zpy 4	CLV 2	LDA aby 4*	TSX 2	LAS aby 4*	LDY abx 4*	LDA aby 4*	LDX aby 4*	LAX aby 4*
Cx	CPY imm 2	CMP izx 6	NOP imm 2	DCP izx 8	CPY zp 3	CMP zp 3	DEC zp 5	DCP zp 5	INY 2	CMP imm 2	DEX 2	AXS imm 2	CPY abs 4	CMP abs 4	DEC abs 6	DCP abs 6
Dx	BNE rel 2*	CMP izy 5*	KIL	DCP izy 8	NOP zpx 4	CMP zpx 4	DEC zpx 6	DCP zpx 6	CLD 2	CMP aby 4*	NOP 2	DCP aby 7	NOP abx 4*	CMP abx 4*	DEC abx 7	DCP abx 7
Ex	CPX imm 2	SBC izx 6	NOP imm 2	ISC izx 8	CPX zp 3	SBC zp 3	INC zp 5	ISC zp 5	INX 2	SBC imm 2	NOP 2	SBC imm 2	CPX abs 4	SBC abs 4	INC abs 6	ISC abs 6
Fx	BEQ rel 2*	SBC izy 5*	KIL	ISC izy 8	NOP zpx 4	SBC zpx 4	INC zpx 6	ISC zpx 6	SED 2	SBC aby 4*	NOP 2	ISC aby 7	NOP abx 4*	SBC abx 4*	INC abx 7	ISC abx 7

3.2. ábra. A 6502 opkódmátrixa

Ha meg szeretnénk találni egy adott opkódhoz a hozzá tartozó információt, akkor szükségünk lesz az opkód hexadecimális alakjára, ami legfeljebb kétszámjegyű lehet. A nagyobb helyiértékű számjegy a keresett cella sorát, a kisebbik pedig az oszlopát írja le. Példaként a 3.2 ábrán láthatjuk, hogy a \$30 opkódhoz a BMI utasítás tartozik relatív címzéssel. Azok az opkódok csillaggal vannak jelölve, amiknek a futási ideje megnőhet az aktuális argumentumuktól függően. A szürkével jelölt opkódokhoz hivatalosan nincs utasítás rendelve. Ezeket a nem dokumentált „illegális” opkódokat a processzor későbbi verzióinak hagyta fent. A tervezők nem tiltották meg azon-

ban ezeknek a használatát, egyszerűen csak nem definiálták a viselkedésüket. Ennek ellenére több olyan illegális opkód is belekerült a dizájnba, ami később hasznosnak bizonyult. A játékfejlesztők próbálkozások útján felfedezték, hogy melyek azok az opkódok, amiknek a viselkedése determinisztikus és néhány speciális feladat esetén érdemes őket használni. Ritka ugyan, de van olyan játék, ami ezeket az opkódokat is használja, ezért ezeknek az opkódoknak az emulációját is megvalósítottam.

3.3.2. Regiszterek

A regiszter a processzor leggyorsabban elérhető memóriája. A gyártási költségek alacsonyan tartása végett csak 6 regiszter került a processzorba. minden regiszter mellett zárójelben fel van tüntetve annak mérete bitekben megadva.

A (8): Akkumulátor, az aritmetikai műveletek eredményei ebbe kerülnek.

X (8) és Y (8): Index regiszterek, indirekt címzésnél használjuk őket. Ciklusok esetén a ciklusváltozót érdemes ezekben tárolnunk.

S (8): Verem mutató. A verem tetejének a kezdőcímtől vett eltolását tárolja.

P (8): Státusz regiszter, ami 7 darab flag bitet tárol.

PC (16): Programszámláló. A következő opkód memóriacímét tárolja. Méretéből következik, hogy a processzor teljes címtartománya 64 KiB nagyságú.

3.3.3. Memórialap

Az i . lap egy 256 bájtos egybefüggő memóriarész, ami a $[i \cdot \$100, (i + 1) \cdot \$100)$ címtartományon helyezkedik el.

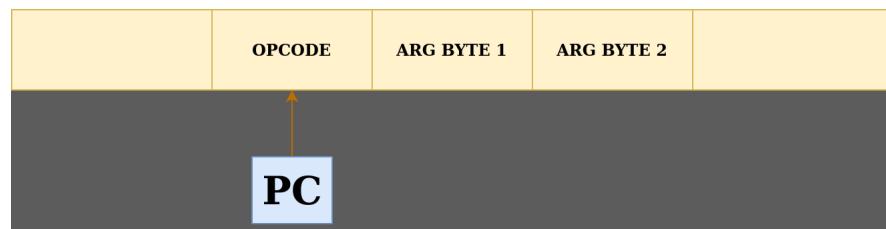
3.3.4. Hívási verem

Az egymásba ágyazott eljárásokat a processzor hardveresen támogatja, amihez egy vermet használ. A verem az 1. lapon található, és a kisebb címek felé nő. Eljárás hívásakor a verem tetejére kerül az aktuális programszámláló értéke, visszatéréskor pedig a veremről levett címre állítjuk be a programszámláló értékét.

3.3.5. Megszakítás

A komponensek kommunikációjának egyik módja a hardveres megszakítás. A 6502 chip egy darab maszkolható (*IRQ*) és egy nem maszkolható (*NMI*) megszakítási lábbal rendelkezik. A megszakítási vektorokkal a program beállíthatja, hogy egy bizonyos megszakításra milyen szubrutinnal kíván reagálni. A vektorok 2 bájtos tárolók, amik a kezelő szubrutin címét tartalmazzák. Az NMI-hez tartozó vektor a \$FFFA és a \$FFFB címeken, míg az IRQ-hoz tarozó vektor az \$FFFE és a \$FFFF címeken található. A 6502 „kicsi az elején” bájtsorrendű (little-endian) processzor, ezért a címeknek minden az alsó 8 bitjét tárolja az alacsonyabb címen, a felső 8 bitjét pedig a magasabb címen. A program dönthet úgy, hogy a maszkolható megszakítást figyelmen kívül hagyja (a kezelő szubrutin nem hívódik meg), ehhez az *IRQ Disable* flag-et be kell állítania a státusz regiszterben. A nem maszkolható megszakítás esetén erre nincsen lehetőség, a végrehajtás mindenkorban a kezelő szubrutinhoz ugrik. A nem maszkolható lábhoz a képfeldolgozó, a maszkolhatóhoz a hangchip van kötve.

3.3.6. Opkód argumentum helye, fajtája



3.3. ábra. Opkód argumentumainak helye

A 3.3 ábra szemlélteti, hogy az opkód argumentumok közvetlenül az opkód mögött, a $PC+1$ és $PC+2$ címeken helyezkedhetnek el a memóriában. Címezési módtól függően változhat az argumentumok száma 0, 1 és 2 bájt között. Ha az opkód rendelkezik argumentummal vagy argumentumokkal, akkor azok a következő fajtáiuk lehetnek:

- 2 bájt, ami abszolút memóriacímet ábrázol kicsi-az-elején bájtsorrenddel
- 1 bájt, ami relatív eltolást ír le
- 1 bájt, ami a művelet közvetlen operandusa

3.3.7. Címzési módok

Egy opkód után azoknál a címzési módoknál áll argumentum, amiknél a művelet elvégzéséhez szükséges operandus nem regiszterben, hanem a memóriában van. A címzési módok azt határozzák meg, hogy az argumentumból hogyan kell kiszámolni az operandus effektív 16 bites memóriacímét. Az alábbiakban felsorolt címzési módoknál a zárójelben az opkódmátrixbeli név (amennyiben van) és az argumentum bájtok száma található.

Akkumulátor mód (0): nincs argumentum, az utasítás az **A** regiszter értékét módosítja.

Azonnali mód (imm, 1): az utasítás operandusa maga az argumentum. Jele: #

Példa: az LDA #\$0 utasítás nullára állítja az **A** regisztert.

Abszolút mód (abs, 2): az argumentum az operandus effektív címe.

0. lap mód (zp, 1): A CPU kevés regiszterét azzal ellensúlyozták, hogy ennek a speciális módnak köszönhetően a nulladik lapot hatékonyabban lehet címezni, mint a többöt. Mivel a 0. lap mérete 256 bájt, ezért teljes cím helyett elég egyetlen bájt a címzéséhez. A kisebb paraméter gyorsabban beolvasható és egyúttal a kódmeretet is csökkenti.

Indexelt 0. lap mód (zpx, zpy, 1): Hasonlóan most is csak a 0. lapot tudjuk címezni, de az argumentumhoz hozzáadjuk valamelyik index regiszter értékét. Az operandus címének kiszámítása: $(arg1 + index) \bmod 256$

Indexelt abszolút mód (abx, aby, 2): Az argumentumok egy teljes memóriacímét alkotnak, amihez hozzáadjuk a megadott index regiszter értékét.

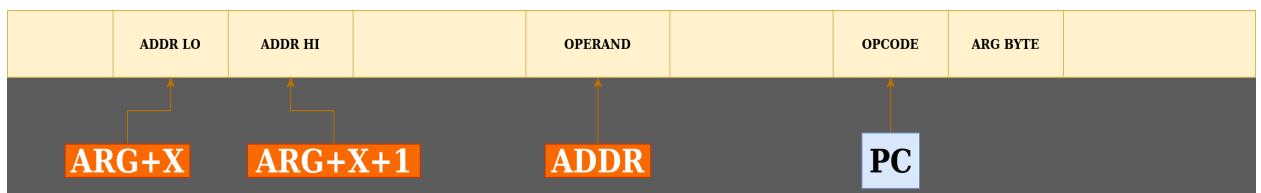
Implicit mód (0): nincs szükség argumentumra, mert az utasítás regisztereikkkel dolgozik.

Relatív mód (rel, 1): Az elágazási utasítások használják ezt a címzési módot. Elágazásoknál, ha a feltétel teljesül, akkor az argumentummal el kell tolni a programszámlálót. Az elágazási utasítások feltételeit lásd az *Utasításkészlet* szekciójában.

Indirekt mód (ind, 2): Erre a címzési módra a JMP utasításnál van szükség. A két argumentum bájt együtt egy teljes memóriacímet alkot, legyen ez m . Az operandus címét az m és $m+1$ címeken találjuk, így tehát az operandus értékét a következő módon kapjuk meg:

$$\text{operand} := \text{read}((\text{read}(m + 1) << 8) \mid \text{read}(m))$$

Indexelt indirekt mód (izx, 1): Az X regisztert összeadjuk az argumentummal, így egy 0. lapon található címet kapunk. Erről a címről kell az operandus effektív címét kiolvasni.



3.4. ábra. Indexelt indirekt címzés

Indirekt indexelt mód (izy, 1): Az argumentum egy 0. lapon található címre mutat, amit ha összeadunk az Y regiszter értékével, akkor megkapjuk az operandus effektív címét.

Az abx, aby és izy indexelt címzési módok és bizonyos utasítások kombinációjánál előfordulhat, hogy a véleges operandus cím és az indexelés előtt álló cím különböző memórialapra esik. Ebben az esetben 1 órajelciklussal tovább tart az utasítás végrehajtása. Emellett az elágazási utasítások (relatív címzés) több ciklus alatt fejeződnek be, ha az ugrási feltétel teljesül. Ha az ugrás előtti PC értéke és az ugrási cél más lapra esik, akkor +2, ellenkező esetben csak +1 órajelciklussal kell számolni.

3.3.8. Memóriatérkép

A memóriatérkép leírja a címtér felosztását a komponensek között. A memóriatérképből meg tudjuk állapítani, hogy egy adott címen található bájt kiolvasásához vagy írásához melyik komponens hardveres logikáját kell alkalmazni.

Tartomány	Eszköz
\$0000 – \$07FF	CPU RAM
\$0800 – \$1FFF	CPU RAM tükrözése
\$2000 – \$2007	PPU regiszterek
\$2008 – \$3FFF	PPU regiszterek tükrözése
\$4000 – \$4017	APU és IO regiszterek
\$4018 – \$401F	APU és IO regiszterek tükrözése
\$4020 – \$FFFF	Kazetta

3.3.9. Memóriatükrözés

Memóriatükrözésről beszélünk, amikor fizikailag ugyanazt a memóriaterületet több memóriacímről is el tudjuk érni. Ha egy címtartomány x bájtonként tükrözve van, akkor minden olyan a és b memóriacím ugyanarra a bájtra mutat, ami a tartományba vagy a tükrözésébe esik és $a \equiv b \pmod{x}$. A CPU RAM 2 KiB-onként tükrözve van, amiből következik, hogy a **\$0001**, **\$0801**, **\$1001**, **\$1801** címek ugyanarra a bájtra mutatnak. Egy címtartomány tükrözése lehet nagyobb mint az eredeti tartomány (például a CPU RAM esetében a tükrözési tartomány háromszor nagyobb).

3.3.10. Státusz flagek

0. bit: C = Carry

Összeadások, forgatások, eltolások során a leeső biteket jelzésére.

1. bit: Z = Zero

Aritmetikai művelet eredménye vagy mozgatott bájt 0.

2. bit: I = IRQ Disable

Az 1-re állításával maszkoljuk az IRQ-t.

3. bit: D = Decimal mode

A NES nem használja ezt a flag-et.

4. bit: B = BRK Command

Az 1-re állításával generálhatunk egy szoftveres megszakítást.

6. bit: V = Overflow

Aritmetikai műveleteknél a túlcordulás vagy alulcsordulás jelzésére. A BIT utasítás speciálisan használja.

7. bit: N = Negative

A manipulált bájt negatív, vagyis a legnagyobb helyiértékű bitje 1.

3.3.11. Utasításkészlet

Aritmetikai és logikai egység (ALU)

ADC: Összeadás; **SBC:** Kivonás; **AND:** Logikai ÉS művelet; **ASL:** Bájt balra elcsúsztatása; **LSR:** Bájt jobbra elcsúsztatása; **ORA:** Logikai VAGY; **EOR:** Kizárasos VAGY; **INC,** **INX,** **INY:** növelés eggyel; **DEC,** **DEX,** **DEY:** csökkentés eggyel; **ROL,** **ROR:** bájt forgatása;

Összehasonlítás, tesztelés

CMP, CPX, CPY:

A megadott címen található bájt és rendre az A, X és Y regiszterekben található bájt között az = és a >= reláció kiértékelése

BIT: Az operandus bájt maszkolása (&) az A regiszter tartalmával, az eredmény szerint a Z,N,V flag-ek frissítése

Veremműveletek

PHA: Az akkumulátor regiszter értékének felrakása a veremre

PHP: A státusz regiszter értékének felrakása a veremre

PHA: Az akkumulátor regiszter új értékének levétele a veremről

PHP: A státusz regiszter új értékének levétele a veremről

Vezérlés

JMP: Vezérlés áthelyezése egy megadott memóriacímhez, avagy a programszámláló átállítása erre a címre

JSR: Szubrutin hívás (visszatérési cím elmentése + **JMP**)

RTS: Visszatérés szubrutinból (visszatérési cím olvasása + **JMP**)

BRK: Szoftveresen generált megszakítás

RTI: Visszatérés megszakításkezelőből

Flag manipuláció (a utasításnév harmadik betűje a változtatott flag-et jelöli)

CLC, CLD, CLI, CLV, SED, SEC, SEI

(C = Clear, S = Set)

Elágazások: vezérlés áthelyezése akkor, ha teljesül a feltétel az utasítás által vizsgált státusz flag-re

**BCC(C=0), BCS(C=1), BNE(Z=0), BEQ(Z=1), BPL(N=0),
BMI(N=1), BVC(V=0), BVS(V=1)**

Bájtok mozgatása regiszterek és a memória között

LDA, LDX, LDY, STA, STX, STY

(L = Load, S = Store)

A név harmadik betűje a használt regisztert jelöli.

Bájtok mozgatása regiszterek között

TAX, TAY, TSX, TXA, TXS, TYA

A név második betűje a forrásregisztert, a harmadik a célregisztert jelöli.

3.4. Kazetták

A kazettákon logikai szempontból kétfajta memória található: PRG és CHR. A PRG jellemzően csak olvasható memória (ROM), ahol a processzor által végrehajtandó utasítások, vagy a játéklogika kapott helyett. Mérete NROM esetében 16 KiB vagy 32 KiB, UNROM-nál pedig 64 KiB vagy 128 KiB. A 8 KiB méretű A CHR ROM/RAM memória, mely 8 KiB méretű és játék sprite-jait tárolja, amik 8*8 pixeles kis képekből állnak. Ezeket a kis képeket ezentúl alakzatoknak fogom hívni.

3.4.1. Az iNES fájlformátum

Az iNES fájlformátumot egy korai NES emulátor vezette be a NES játékok bináris formában történő terjesztésére. A fájl elején található 16 bájtos fejléc többek között a mapper áramkör azonosítóját, a képfeldolgozó névtábláinak tükrözési módját, valamint a PRG ROM, PRG RAM és a CHR méretét határozza meg. A fejléc után ezek tartalma található.

3.5. A képfeldolgozó egység

3.5.1. Színpaletta

A képpontok végső RGB színkódjának meghatározása többféle módon is lehetséges. A gyors, de kevésbé pontos módszer, ha egy fix palettával dolgozunk. A paletta hozzárendel egy \$0 és \$3F közötti azonosítóhoz egy színkódot, így tehát 55 különböző színt tudunk megjeleníteni. A lassabb, de pontosabb módszer, hogy az analóg NTSC videójelre történő konverziót is emuláljuk. Az emulátoromnál a gyorsabb módszert implementáltam.

3.5.2. Paletta indexek

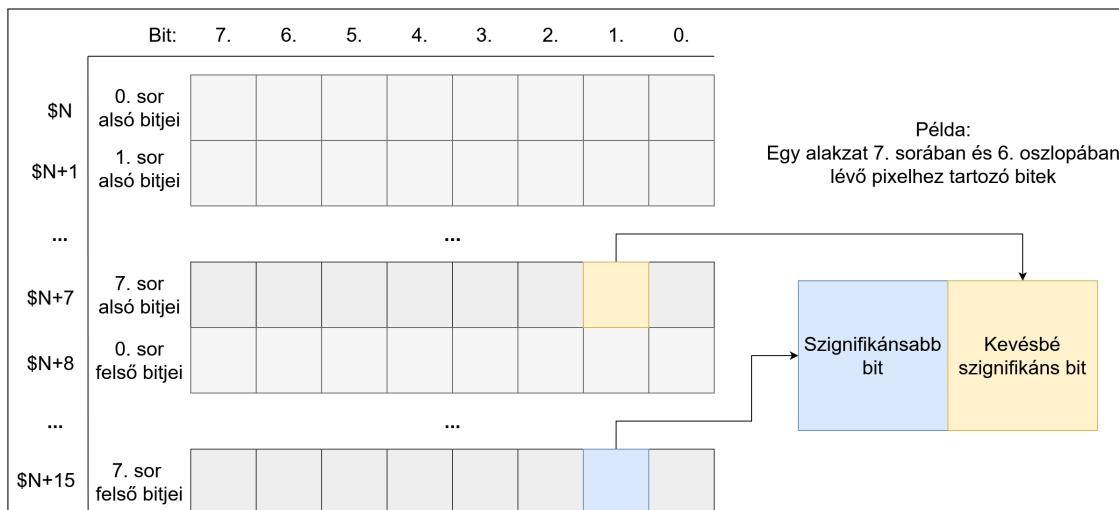
A \$3F00 – \$3F1F címtartományon található paletta indexben kerülnek eltárolásra a használatban lévő színek színpalettabeli azonosítói, csoportokba (palettákba) rendezve. A hardveres limitációk miatt a háttteret 16x16 pixeles cellákra osztották fel. Egy cellán belül kizárolag 4 féle szín fordulhatott elő. Ezeket a 4 színből álló színkombinációkat kissé megtévesztő módon szintén palettáknak nevezik. További limitáció, hogy a paletták 4. színe nem változtatható, mert ezek minden a \$3F00 címet (a háttérszínt) tükrözik.

Tartomány	Paletta
\$3F00	Univerzális háttérszín
\$3F01 – \$3F03	0. háttér paletta
\$3F05 – \$3F07	1. háttér paletta
\$3F09 – \$3F0B	2. háttér paletta
\$3F0D – \$3F0F	3. háttér paletta
\$3F11 – \$3F13	0. sprite paletta
\$3F15 – \$3F17	1. sprite paletta
\$3F19 – \$3F1B	2. sprite paletta
\$3F1D – \$3F1F	3. sprite paletta

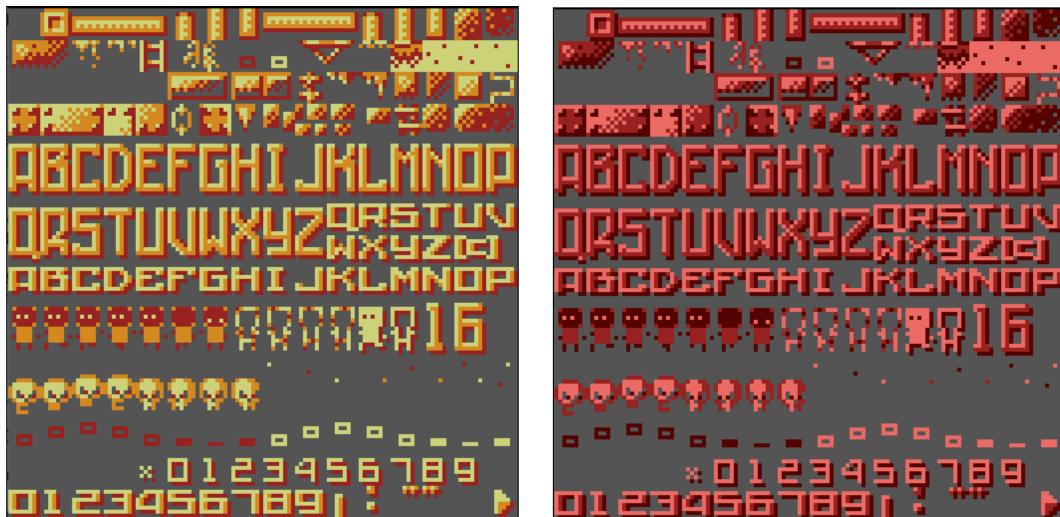
3.1. táblázat. A paletta indexek címei

3.5.3. Alakzattáblázat

A CHR memóriában található két alakzattáblázat tárolja az alakzatokat egy speciális formátumban. Ha az alakzatokat úgy reprezentálnánk, hogy minden pixelre eltárolnánk egy színkódot, akkor pixelenként 6 bitre lenne szükségünk (mivel 55 színkóból választhatunk). Ehelyett pixelenként csak 2 bitet (egy szignifikáns és egy kevésbé szignifikáns bitet) tárolnunk, amik együtt egy, a palettaindexben található palettán belüli színt azonosítanak. Az attribútumtábla segítségével fogjuk később meghatározni, hogy a vizsgált pixelhez melyik paletta tartozik. Mivel a paletta index írható és olvasható memória is, így a program futási időben, dinamikusan változthatja, hogy egy paletta milyen színeket tartalmaz, ezáltal pár lépésben átszínezheti az összes alakzatot, ami az adott palettát használja. Egy alakzattáblázat 16x16 darab alakzatot tartalmaz és egy alakzat 8x8 pixeles, ebből adódóan a táblázat teljes mérete $16 \cdot 16 \cdot (8 \cdot 8 \cdot 2) \div 8 = 4096$ bájt. Egy alakzat pixeljeinek bitjeit $(8 \cdot 8 \cdot 2) \div 8 = 16$ egymást követő bájt tárolja a 3.5 ábrán szemléltetett módon. Először 8 bájton keresztül az alakzat sorainak kevésbé szignifikáns bitjei, majd ezután újabb 8 bájton át a szignifikánsabb bitek sorakoznak. Az oszlopok és bitek sorszámozása fel van cserélve, tehát az i . oszlophoz egy bájton belül a $7 - i$. bit tartozik.



3.5. ábra. Egy alakzat reprezentálása



3.6. ábra. Az Alter Ego játék 0. alakzattáblázata különböző palettákkal kirajzolva

3.5.4. Rétegek

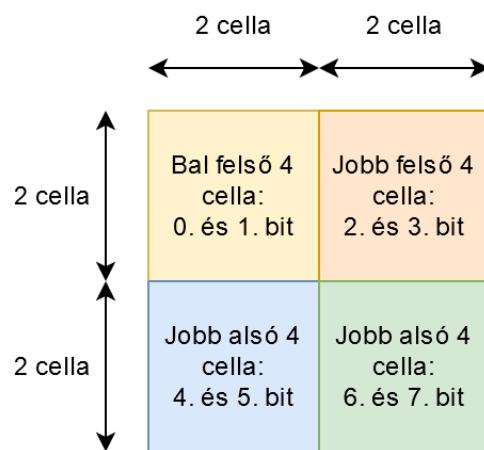
A képfeldolgozó két réteget képes kezelní hardveresen, ezek a háttér és a sprite rétegek. Általában a képkocka azon részei tartoznak a háttérhez, amik ritkán változnak (például feliratok, számlálók, mozdulatlan képek) és a kirajzolásukhoz nem szükséges további transzformáció (eltolás, tükrözés). A háttérben az alakzatok 30 sorból és 32 oszlobból álló négyzetrácsot alkotnak (ebből és az alakzatok méretéből ered a NES 256*240 pixeles felbontása). A háttér rétegben egyesével nem lehet alakzatokat eltolni, csak az egész háttér eltolása lehetséges. A sprite rétegben nagyobb flexibilitás áll rendelkezésre, ugyanis itt egyenként, pixel pontosságú eltolással, valamint horizontális és/vagy vertikális tükrözéssel rajzolhatjuk ki az alakzatokat. A sprite rétegnél legfeljebb 64 alakzat szerepelhet egy képkockán (ez a megkötés a később ismertetett OAM méretéből adódik).

3.5.5. Névtáblázatok

A háttérréteg négyzetrácsának elrendezését a névtáblázatok tárolják. Egy névtáblázat a háttér 960 darab alakzatcellájának mindegyikéhez eltárolja a cellába rajzolandó alakzat sorszámát. Ez a sorszám relatív, ugyanis minden az alakzattáblázaton belül értendő, amit a CONTROLLER regiszterrel a program kiválasztott.

3.5.6. Attribútumtáblázatok

Minden névtáblához tartozik egy 64 bájtos attribútumtáblázat, ahol minden bájt egy 4×4 alakzatcellából álló terület palettáit határozza meg. A bájtok 4 darab 2 bites részre vannak felosztva, ahol mindegyik rész egy 2×2 cellából álló terület palettájának sorszámát (0-3) kódolja el. Ennek a reprezentációnak a következménye, hogy a háttér 4 cellából álló csoportjai mindig egy palettán osztoznak. A 16 cellát lefedő bájt felosztása a 4 cellás területek között a 3.7 ábrán van szemléltetve.



3.7. ábra. Attribútumbájtok felosztása

3.5.7. Háttéreltolás

A háttéreltolással pixelpontossággal megszabhatjuk, hogy a háttér mely része legyen látható a játékos számára. Az eltolás miatt van szükség több névtáblára, ugyanis a lecsúszó háttérrész nem az aktív, hanem háttértükrözéstől függően valamelyik másik névtáblából lesz kiolvasva.

3.5.8. OAM (Object Attribute Memory)

A sprite réteg elrendezését leíró 256 bájtos memória. 64 darab alakzatról tárol információt, amik a következők:

- X és Y koordináta
- A sprite réteg aktív alakzattáblázatán belüli sorszám
- Tükörzés
- Paletta sorszám

- Prioritás a háttérrel szemben

3.5.9. Regiszterek

A CPU és a PPU az alább látható 9 darab egy bájtos regiszter segítségével tud egymással kommunikálni. A regisztereket a CPU a zárójelekben található címeken tudja elérni. Az olvasató regiszterek **R**, az írhatók **W** betűvel vannak megjelölve.

CONTROLLER(\$2000, W):

A kirajzolás vezérlésére szolgáló regiszter. Beállítható vele a következő képkockánál használandó táblázatok indexe.

- 0-1. bit: Aktív névtáblázat indexe
2. bit: VRAM cím inkrementálási mód
3. bit: Aktív alakzattáblázat indexe a sprite rétegnél
4. bit: Aktív alakzattáblázat indexe a háttér rétegnél
5. bit: Sprite méret (8x8 vagy 8x16 pixel)
6. bit: Az emuláció során nem használt bit
7. bit: NMI generálása a PPU tétlen periódusának kezdetén

MASK(\$2001, W):

A rétegek egyenkénti ki/bekapcsolása és speciális effektusok (például szürkeárnyalat) vezérelhetők vele.

STATUS(\$2002, R):

A kirajzolás alatt bekövetkező eseményeket jelzi a PPU a CPU-nak ezzel a regiszterrel. Ilyen esemény például a sprite túlcordulás, ami akkor áll fent, ha több mint a 8 alakzat kerülne egy sorra a sprite rétegen.

OAMADDR(\$2003, W) és OAMDATA(\$2004, R/W):

A processzor ezen két regiszter segítségével képes új adatokkal feltölteni az OAM memóriát. Az alábbi kód részlet azt szemlélteti, hogy az *adatok* tömb tartalmát hogyan kell a CPU memóriájából az OAM-ba átmásolni egy megadott címtől kezdve. Az OAMDATA írása után az OAMADDR automatikusan inkrementálódik a másolás gyorsításának érdekében.

```

1 OAMADDR := 8 bites OAM cím
2 for i in 1..adatok.hossz
3   OAMDATA := adatok[i]

```

PPUSCROLL(\$2005, W):

Beállíthatjuk vele, hogy a hátteret hány pixellel szeretnénk arrébbcsúsztatni (Horizontális tükrözésnél vízszintesen, vertikális tükrözésnél függőlegesen).

PPUADDR(\$2006, W) és PPUDATA(\$2007, R/W):

A névtáblák frissítésére szolgálnak. Hasonlóan kell őket használni, mint az OAMADDR és OAMDATA regisztereket.

OAMDMA(\$4014, W):

Az OAM memória frissítésének egy alternatív, gyorsabb módja a Direct Memory Access (DMA). Ekkor a processzorban található dedikált hardver másolja át az adatokat egyenesen a CPU RAM-ból az OAM memóriába. A másolás megkezdéséhez annak a memórialapnak a sorszámát kell beírni a regiszterbe, ahol az átmásolandó adatok találhatók.

3.5.10. Memóriatérkép

Tartomány	Paletta
\$0000 – \$0FFF	0. Alakzattáblázat
\$1000 – \$1FFF	1. Alakzattáblázat
\$2000 – \$23FF	0. Névtáblázat
\$2400 – \$27FF	1. Névtáblázat
\$2800 – \$2BFF	2. Névtáblázat
\$2C00 – \$2FFF	3. Névtáblázat
\$3000 – \$3EFF	A 0-3. névtáblázatok tükrözése
\$3F00 – \$3F1F	Paletta indexek
\$3F20 – \$3FFF	Paletta indexek tükrözése

3.2. táblázat. A képfeldolgozó memóriatérképe

3.5.11. A háttér kirajzolásának egyszerű algoritmusa

Ezen a ponton már minden részletet ismerünk ahhoz, hogy megérthessük a háttérkirajzolás logikáját. A következő pszeudokód azt szemlélteti, hogy a fent említett táblázatokat és a paletta indexet hogyan kell együtt használni a háttér kiszámolásához. Az egyszerűség kedvéért a háttéreltolást itt nem veszem figyelembe. Sajnos ez az algoritmus ebben a formában emulációra nem alkalmas, mert nehézzé teszi a processzor párhuzamos, megfelelő szinkronizációval történő futtatását.

```

1
2 // Egy bájt indexedik bitjének kiolvasása (0/1)
3 byte bit(byte bájt, int index)
4 begin
5     return (bájt >> index) & 1
6 end
7
8 // Az alábbi függvényel olvasunk a PPU címterében
9 byte olvas(cím)
10
11 type RGB_Kód = (byte, byte, byte)
12
13 // Az alábbi függvény a paletta index és a színpaletta felhasználás
14 // ával
15 // visszaadja, hogy egy adott sorszámú palettán belül az indexedik
16 // színnek mi
17 // az RGB kódja
18 RGB_Kód színKeresés(byte palettaSorszám, byte index)
19
20
21 // Az eredményül kapott pixeleket tároló kétdimenziós tömb
22 RGB_Kód pixelek[256][240]
23
24 procedure HáttérKirajzol
25 begin
26     // A CONTROLLER regiszterrel a választott névtáblázat
27     // és alakzattáblázat kezdőcímének meghatározása
28     cím aktívNévtáblazat      := $2000 + (CONTROLLER & 0b11) * $400
29     cím aktívAlakzatTáblazat := bit(CONTROLLER, 4) * $1000
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
629
629
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1
```

```
28 // Végigiterálás a háttér celláin
29 for cellaSor in 0..29
30     for cellaOszlop in 0..31
31         begin
32             // A cellához tartozó névtáblabájt indexének kiszámolása
33             byte NTB_Eltolás := cellaSor * 32 + cellaOszlop
34
35             // A névtáblabájt kiolvasása
36             byte NTB := olvas(aktívNévtáblázat + NTB_Eltolás)
37
38             // A cellához tartozó attribútumbájt eltolása a névtábla
39             // kezdőcíméhez viszonyítva.
40             cím ATB_Eltolás :=
41                 // Átlépjük a 960 névtáblabájtot
42                 $3C0 +
43
44                 // Átlépjük az előző cellasorok attribútumbájtjait
45                 // Emlékeztető: egy sorban 32 alakzat van amik
46                 // négyesével osztoznak a bájton
47                 (cellaSor div 4) * 8 +
48
49                 // Átlépjük a jelenlegi cellasorban az előző attribútumbá-
50                 // jtokat
51                 (cellaOszlop div 4)
52
53             // Az attribútumbájt kiolvasása
54             byte ATB := olvas(aktívNévtáblázat + ATB_Eltolás)
55
56             // Az attribútumbájtnak a cellához tartozó 2 bites része lesz
57             // a palettaszínszám.
58             // Ki kell számolni, hogy a cella melyik (bal felső, jobb
59             // felső, stb.)
60             // kvadránsába esik a bájt által lefedett 4*4-es területnek,
61             // ugyanis így kapjuk meg,
62             // hogy a bájt melyik 2 bitjére van szükségünk
63             byte kvadráns := (cellaSor & 0b10)*2 + (cellaOszlop & 0b10)
64             byte palettaSorszám :=
65                 (ATB >> kvadráns) & 0b11
```

```

63 // Az alakzat kezdőcíme az alakzattáblában
64 // Segítség: egy alakzat (8*8*2)/8 = 16 bájtot foglal
65 cím alakzatCím := aktívAlakzattáblázat + NTB*16
66
67 // Végigiterálás a cella 8*8 pixeles területén
68 for pixelSor in 0..7
69 begin
70     cím alakzatSor := alakzatCím + pixelSor
71
72     // A sor alsó bitjei
73     byte alakzatLSB := olvas(alakzatSor)
74
75     // A sor felső bitjei
76     byte alakzatMSB := olvas(alakzatSor + 8)
77
78     for pixel0szlop 0..7
79     begin
80         // A pixelhez tartozó 2 bittel a palettán belüli szín
81         meghatározása
82         byte palettaIndex :=
83             (bit(alakzatMSB, 7-pixel0szlop) << 1) | bit(alakzatLSB,
84             7-pixel0szlop)
85
86         // A pixel X koordinátája a képernyőn (0-255)
87         byte X := cella0szlop * 8 + pixel0szlop
88
89         // A pixel Y koordinátája a képernyőn (0-239)
90         byte Y := cellaSor * 8 + pixelSor
91
92         // Az RGB kód kikeresése és beállítása
93         pixelek[X][Y] :=
94             színKeresés(palettaSorszám, palettaIndex)
95     end
96 end

```

3.5.12. Sprite réteg kirajzolása

A kirajzolás algoritmusának annyiban változik, hogy a névtáblázatok és attribútumtáblázatok helyett az OAM memóriára hagyatkozva határozzuk meg az alakzatsorszámokat és palettaszámokat. A 240 sor mindegyikénél a kirajzolás megkezdése előtt ki kell értékelni, hogy melyek azok az alakzatok, amik a következő soron láthatóak. Ezeknek az adatait egy pufferbe, u.n. másodlagos OAM-ba kell helyezni (legfeljebb 8 OAM-bejegyzés fér bele). minden pixelnél megnézzük, hogy van-e olyan alakzat a másodlagos OAM-ban, ami arra a pixelre esik. Ha több is van, akkor a kisebb indexű alakzat élvez nagyobb prioritást.

4. fejezet

Fejlesztői dokumentáció II: Megvalósítás

4.1. A feladat specifikációja

A feladat a NES játékkonzol CPU-ját, PPU-ját és kazettáit valós időben emulálni képes szoftver implementálása, ahol a játékos a játék irányítása mellett képes a virtuális gép állapotának elmentésére és betöltésére, valamint az emuláció vezérlésére. A játékokat billentyűzettel vagy kontrollerrel lehet irányítani. A program felhasználói felülettel is rendelkezik, amivel szintén tudunk mentéseket kezelní és szüneteltethetjük az emulációt.

4.2. Fejlesztői környezet

A programot Haskell nyelven írtam és a GHC 8.10.1-es verziójával fordítottam. Szerkesztőprogramként a Visual Studio Code-ot használtam. A Haskell-ben írt független könyvtárak telepítéséhez a Stack eszközt használtam. Az SDL2 és a GTK könyvtárakat Linux-on a beépített csomagkezelővel, Windows-on az MSYS2 konzol csomagkezelőjével telepítettem.

4.3. Megvalósítási terv

4.3.1. NES emuláció

Minden komponenshez (CPU, PPU, kazetták) tartozik egy *Memory* modul, ami definiálja a komponens emulációja során használt rekordokat. A *Serialization* modulok ezen rekordok mentését/betöltését tartalmazzák. A CPU és a PPU *Emulation* moduljai tartalmazzák a hozzájuk tartozó rekordokon végezhető műveleteket.

A *Monad* modul definiálja az emulációs monádot és az abban elérhető primitív műveleteket. A *MasterClock* modul összefogja a komponenseket és szinkronizálja azok emulációját. A *Controls* modul a NES sztenderd kontrollerének emulációját tartalmazza.

A *JoyControls* modul a fizikai kontrollerek kezelését végzi (újonnan csatlakoztatott kontroller és annak rezgőmotorjának inicializálása, gombnyomások megfeleltései parancsoknak).

A *Window* modul az SDL2 multimédia-könyvtárat és az OpenGL-t használva megjeleníti az előállított képkockákat, kezeli a billentyűzet gomblenyomásait és futtatja a parancsokat feldolgozó ciklust. A *CrtShader* modulban a CRT effektusért felelős OpenGL shaderek találhatók. Végül, de nem utolsó sorban a *Framerate* modul függvényeivel tudjuk szabályozni a képkockaszámot (fix vagy korlátozatlan számú képkocka/másodperc).

A *Communication* modul definiálja azokat az eseményeket, amiket az emulációs ablak a felhasználói felületnek küldhet (mentés/betöltés eredménye, megjelenítendő figyelmeztetés/hibaüzenet), illetve azokat a parancsokat, amiket attól fogadhat (szüneteltetés, mentés/betöltés, leállítás, stb.).

4.3.2. Felhasználói felület

A GTK (GIMP Toolkit) egy nyílt forráskódú, kezelőfelületek tervezésére szolgáló szoftvercsomag. Az API-ja javarésztl imperatív stílusú, például a widget-ek létrehozását követően mellékhatásos függvényekkel tudjuk beállítani az attribútumokat és az eseménykezelőket. A Haskellben elérhető *gi-gtk-declarative* könyvtár ezzel szemben egy deklaratív GTK API-t ad a kezünkbe, aminek segítségével röviden és tömörén tudjuk összetett vezérlőelemek megjelenését leírni. A *gi-gtk-declarative*-

app-simple erre építve definiál felhasználói felületek vezérléséhez egy egyszerű architektúrát, ahol a felületet állapotgépnek tekintjük és az állapotátmeneteket események váltják ki. A felület létrehozásához elég a kezdőállapotot, az állapotmegjelenítő (*state → widget*) függvényt és az állapotátmenet-függvényt definiálni.

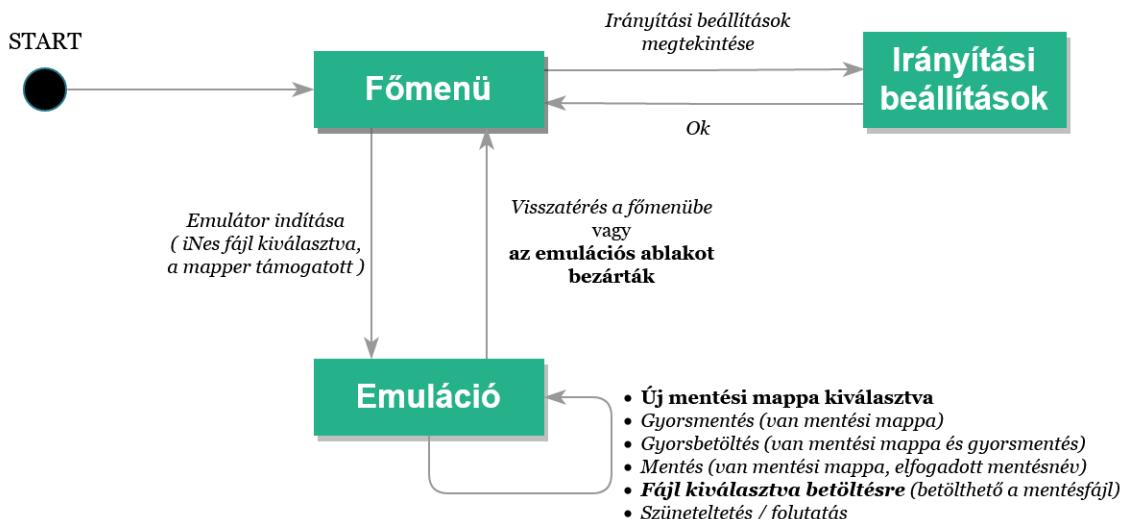
A felhasználói felület moduljai

State: A felhasználói felület állapotai (Főmenü, Irányítási beállítások, Emuláció, Üzenet). minden állapothoz tartozik egy rekord, ami a megjelenítéshez és az állapotátmenetekhez szükséges információkat tárolja.

InGame: Az emulációs állapotnál használt megjelenítőfüggvényt tartalmazza.

Window: A teljes felület megjelenítőfüggvényét tartalmazza, ami minden állapotot lefed.

Logic: Az állapotátmenet-függvényt tartalmazza, ami a felhasználói felület állapotváltozásaikor képes olyan mellékhatásokat is végrehajtani, mint például parancsok küldése az emulációs ablaknak.



4.1. ábra. A felület reakciói az eseményekre

A 4.1 ábra az állapotátmenet-függvényt hivatott vizualizálni. A dőlt betűs részek gombnyomásokat, a kövér betűs részek eseményeket jelölnek. Ha egy eseménynél a zárójelben lévő feltétel nem teljesül, akkor az *Üzenet* állapotba lépünk, ahol megjelenítjük a hibaüzenetet/figyelmeztetést. Innen az *Ok* gomb lenyomására visszatérünk

az előző állapotba. A felület bármelyik állapotban bezárható, ekkor a teljes program terminál (az emulációs ablak is bezárul).

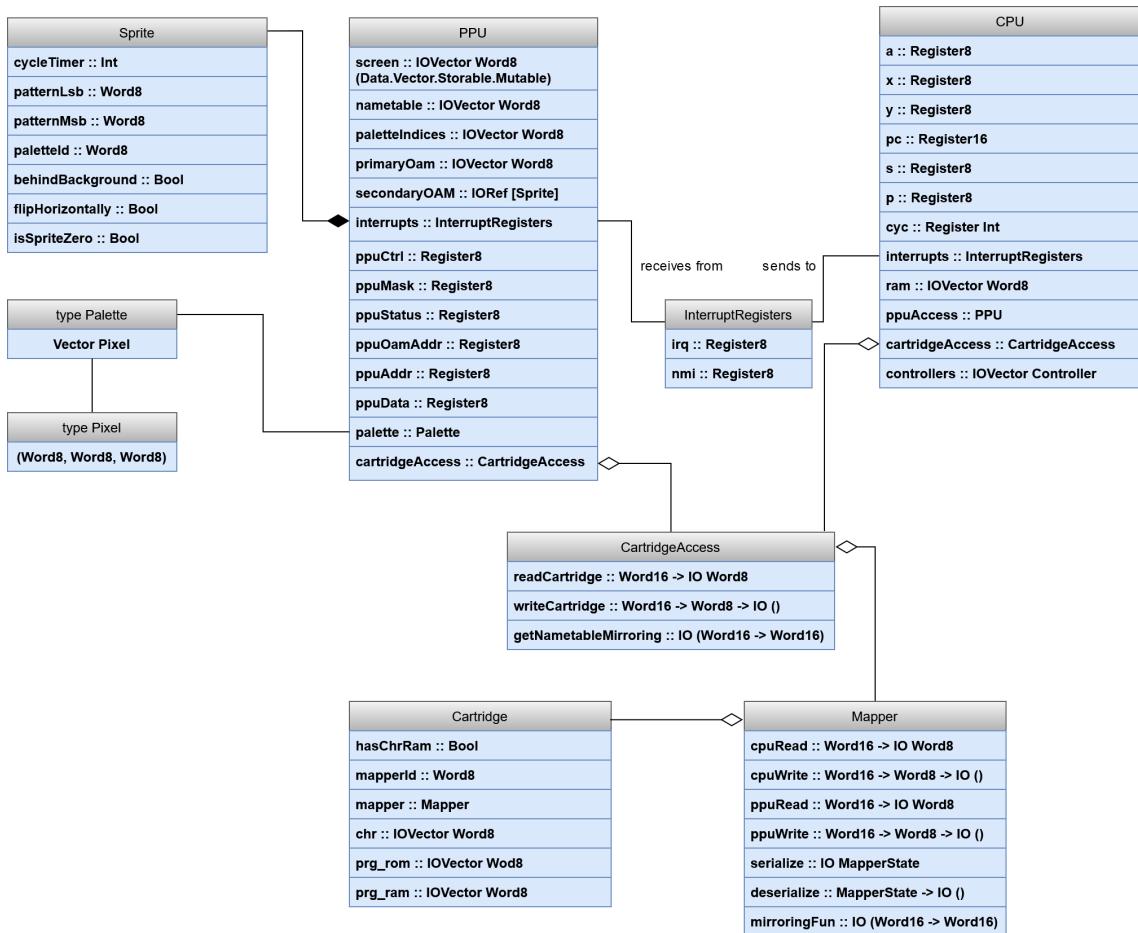
4.4. Az emulációt magába záró monád

A CPU, a PPU, valamint a teljes NES emulációjához érdemes egy monádot bevezetni, hogy ne kelljen mindenhol explicit paraméterként átadni a komponens rekordját. A ReaderT monádtranszformer miatt ez bármely ponton elérhető (olvasható) lesz. A *runEmulator* függvényel lefuttathatjuk az emuláció mellékhatásait. Az *emulateSubcomponent* függvény célja, hogy a komponensek emulációját egyesítve lehetséges legyen a teljes rendszer emulációja.

```
1 import Control.Monad.Reader
2
3 newtype Emulator component value
4   = Emulator (ReaderT component IO value)
5   deriving
6     (Functor, Applicative, Monad, MonadIO, MonadReader component)
7
8 runEmulator :: c -> Emulator c v -> IO v
9 runEmulator component (Emulator effect)
10  = runReaderT effect component
11
12 emulateSubcomponent :: (c -> s) -> Emulator s v -> Emulator c v
13 emulateSubcomponent subcomponent (Emulator effect)
14  = Emulator (withReaderT subcomponent effect)
```

4.5. Adatreprézentáció

A teljesítmény fontos szempont volt, ezért a regisztereket változtatható és u.n. *unboxed* (közvetlenül az értéket tárolják és nem egy mutatót az értékre) referenciaikkal ábrázoltam. Az egyes memóriarészleteket (CPU RAM, PRG, CHR, stb...) külön-külön, változtatható és unboxed vektorokkal reprezentálom.



4.2. ábra. Az emulációnál használt rekordok

A CPU és a PPU osztozik a megszakítási "regisztereken", ezért a PPU megszakításokat tud küldeni a CPU-nak.

A kazetta memóriáját a *Cartridge* rekord tárolja. Ehhez közvetlenül nem tudunk hozzáférni, mert a mapper típusa szabja meg, hogy hogyan kell a virtuális címeket fizikai címekre átfordítani. A CPU és a PPU a *CartridgeAccess* rekord függvényeit használva tudja olvasni és írni a kazettát. Ezeket minden komponens személyre szabva kapja meg, így tehát a CPU esetében a *readCartridge* függvény a mapper *cpuRead* függvényének felel meg.

4.6. Mentés és betöltés

Ennek a funkciónak a megvalósítása sokféleképp lehetséges. Én azt a megoldást választottam, hogy azokhoz a rekordokhoz, amik változtatható referenciákat és vektorokat tartalmaznak, definiáltam egy "tiszta" rekordot, ami csak a komponens

tárolandó adattagjait tartalmazza (például a PPU framebuffer-ét felesleges tárolni). A tiszta rekordokhoz automatikusan generálni lehet szerializáló műveleteket (erre a *cereal* könyvtárat használtam), így tehát csak annyi dolgom volt, hogy rekordok közti konverziót implementáljam.

```
1 data SerializedPPU = SerializedPPU {
2 ...
3     secondaryOam      :: [Sprite],
4     ppuCtrl           :: Word8,
5     ppuMask           :: Word8,
6 ...
7 } deriving (Generic, Serialize)
```

4.7. Főmodulok áttekintése

4.7.1. Nes.Cartridge.Memory

A modul a Cartridge, Mapper és CartridgeAccess rekordok definícióját és a hozzájuk kapcsolódó függvényeket, valamint a névtáblázattükörözési függvényeket tartalmazza.

- **getCPUAccess :: Cartridge → CartridgeAccess**

Hozzáférés kérése a kazettához a CPU számára.

- **getPPUAccess :: Cartridge → CartridgeAccess**

Hozzáférés kérése a kazettához a PPU számára.

- **horizontalMirroring :: Word16 → Word16**

Horizontális tükrözés.

- **verticalMirroring :: Word16 → Word16**

Vertikális tükrözés.

4.7.2. Nes.Cartridge.INES.Parser

- **iNESloader :: Get INES**

Az iNES fájlokat betöltő parser.

- **tryLoadingINES :: FilePath → IO INES**

Az iNES parser futtatása a megadott fájlon, hiba esetén kivétel dobása.

- **assembleCartridge :: INES → IO Cartridge**

Az iNES fájl tartalmából az emuláció során használt kazettareprezentáció előállítása.

- **loadCartridge :: FilePath → IO Cartridge**

Adott elérési útvonalról egy kazetta betöltése.

```
1 import Control.Monad (>=>)
2
3 loadCartridge = tryLoadingINES >=> assembleCartridge
```

4.7.3. Nes.Cartridge.Mappers

Ez a modul a mapper-ek implementációit tartalmazza. Ha új mapper-t szeretnénk hozzáadni az emulátorhoz, akkor elég pusztán itt definiálni számára egy konstruktort. A kazetta összeszerelésénél a mapper konstruktora megkapja a félkész kazettát és visszaadja az írásra/olvasásra szolgáló függvényeket tartalmazó rekordot, amik a megfelelő logikával fordítják át a címeket. A konstruktorkok azért nem tisztítanak a függvényeket, hogy tetszőleges belső állapotot létre lehessen hozni bennük. (belépő regiszterekhez referenciák létrehozása)

- **mappersById :: Map Word8 (Cartridge → IO Mapper)**

- **nrom :: Cartridge → IO Mapper**

- **unrom :: Cartridge → IO Mapper**

- **cnrom :: Cartridge → IO Mapper**

4.7.4. Nes.Emulation.Monad

Az emuláció során használt primitív műveletek.

- **powerUpNes :: Cartridge → IO Nes**

"Behelyezzük" a kazettát és inicializáljuk a komponenseket.

- **useMemory :: (comp → ref) → (ref → IO val) → Emulator comp val**
Kényelmi függvény arra, hogy a komponens rekordjának egy adott mezőjével műveletet végezzünk.

```
1 |   useMemory memory action = asks memory >>= liftIO . action
```

- **readMemory és writeMemory** Komponens vektorral reprezentált memóriájának közvetlen olvasása és írása (a memóriatérkép nincs figyelembe véve).
- **readCartridgeWithAccessor és writeCartridgeWithAccessor** A kazettához a CPU-nak és a PPU-nak is saját hozzáférése van, amit ezekkel a függvényekkel tudnak használni.

```
1 |   readCartridgeWithAccessor ::  
2 |     (component -> CartridgeAccess) ->  
3 |     Word16 ->  
4 |     Emulator component Word8  
5 |   writeCartridgeWithAccessor ::  
6 |     (component -> CartridgeAccess) ->  
7 |     Word16 ->  
8 |     Word8 ->  
9 |     Emulator component ()
```

4.7.5. Nes.CPU.Emulation

A 6502 teljes utasításkészletét tartalmazó modul. Néhány segédfüggvény, eljárás:

- **readReg :: Prim a ⇒ (CPU → Register a) → Emulator CPU a**
A megadott regiszter olvasása. A regiszterek csak primitív, kicsomagolható típusokat tárolhatnak.
- **read :: Word16 → Emulator CPU Word8**
Olvasás a memóriából. A memóriatérkép és a cím alapján dönti el, hogy honnan (RAM, PPU regiszterek, stb...) kell olvasni.
- **write :: Word16 → Word8 → Emulator CPU ()**
Írás a memóriába. Hasonlóan itt is a memóriatérképet kell figyelembe venni.

- **push :: Word8 → Emulator CPU ()**

Bájt felrakása a stack-re.

- **pushAddress :: Word16 → Emulator CPU ()**

Cím felrakása a stack-re. Mivel a stack a kisebb címek felé nő, ezért először a szignifikánsabb bájtot kell felrakni, hogy tartsuk a kicsi-az-elején bájtsorrendet.

```
1  pushAddress addr = do
2      let (high, low) = splitWord16 addr
3      push high
4      push low
```

- **nmi :: Emulator CPU ()**

NMI megszakítás érkezésekor meghívott eljárás. Elmenti a programszámlálót és a státusz-regisztert a stack-re és a megszakítási vektor által mutatott címen folytatja a végrehajtást.

- **processInterrupts :: Emulator CPU ()**

Az interrupt-ok nem érkeznek meg azonnal a processzorhoz, hanem időre van szükségük. Ennél a virtuális gépnél az interrupt regiszterek tárolják, hogy még hány utasítást kell végrehajtani a megszakításig. A processInterrupts eljárás csökkenti a számlálókat és ha valamelyik számláló eléri a nullát, akkor meghívja a megfelelő megszakítást feldolgozó eljárást.

- **oamDma :: Word8 → Emulator CPU ()**

Adott memórialapról másolás DMA-val a PPU OAM-ra.

4.7.6. Nes.PPU.Emulation

- **cpuReadRegister és cpuWriteRegister**

A Nes.CPU.Emulation modul számára kiexportált függvények, amivel a CPU kommunikálhat a PPU-val.

- **getColor :: Word8 → Word8 → Emulator PPU Pixel**

Visszaadja, hogy adott palettán belül az *i*. színnek mi az RGB kódja.

- **clock :: Emulator PPU ()**

A PPU állapotának léptetésére használt függvény. A PPU sorfolytonosan halad végig a képernyőn és állítja elő a pixeleket, amik vagy a háttérrétegből vagy a sprite-rétegből kerülnek ki. Emellett a háttérréteg következő celláinak adatait is előre betölti. A vertikális váltás (a CRT TV elektronágúja a jobb alsó sarokból visszaáll a bal felső sarokba) ideje alatt tétlen (lásd az időzítési diagramot).

4.7.7. Nes.Emulation.Controls

- **press :: Button → Controller → Controller**

Ezzel a függvénnyel nyomhatunk le egy gombot a virtuális kontrolleren.

- **release :: Button → Controller → Controller**

Ezzel a függvénnyel engedhetünk fel egy gombot a virtuális kontrolleren.

- **read :: Controller → (Word8, Controller)**

A CPU ezzel a függvénnyel kérdezheti le a kontroller gombjainak állapotát. Az *i.* meghívása a függvénynek visszaadja az *i.* gomb állapotát. (0 - felengedve, 1 - lenyomva) Az állapotváltoztató hatás a típusból egyértelműen látható.

- **write :: Word8 → Controller → Controller**

A CPU a kontroller írásával vissza tudja állítani a kontrollert az alaphelyzetbe (a 0. gombot fogja újra a *read* művelet lekérdezni).

4.7.8. Nes.Emulation.MasterClock

- **syncCPUwithPPU :: Emulator CPU ()**

CPU utasítás végrehajtása, majd a megfelelő számú PPU órajel emulálása a szinkronizáció megőrzése érdekében.

- **emulateFrame :: Emulator Nes FrameBuffer**

Addig léptetjük a rendszert az előző függvénnyel, amíg nem végzünk a képkockával.

- **resetNes :: Emulator Nes ()**

Visszaállítja alaphelyzetbe a komponenseket.

4.7.9. Communication

4.7.10. Emulator.JoyControls

- **init :: ButtonMappings → IO JoyControlState**

Létrehozza azt a rekordot, amiben a gombhozzárendeléseket és csatlakoztatott kontrollereket tartjuk nyilván.

- **manageButtonEvent**

Visszaadja, hogy a gombnyomáshoz milyen parancsok tartoznak (gyorsmentés, gyorsbetöltés vagy játékirányítás).

- **manageHatEvent**

A DPAD Fel/Le/Balra/Jobbra/Középen eseményeit figyeli és kiadja a megfelelő parancsokat a virtuális kontroller módosítására. Mivel a virtuális kontrollernél nincs Középen állapota a DPAD-nak, ezért arra is figyelni kell, hogy ezt az állapotot "*engedd fel az előző DPAD gombot*" parancsként kell továbbküldeni.

- **manageDeviceEvent**

Figyeli a kontrollercsatlakoztatásokat és az eltávolításokat. Az új kontrollereket inicializálja és eltárolja a rekordban, a kihúzottakat eltávolítja.

4.7.11. Emulator.Framerate

- **uncapped :: MonadIO m ⇒ m Bool → m ()**

A megadott műveletet futtatja olyan gyorsan, amilyen gyorsan azt lehet addig, amíg a művelet igaz értékkel nem jelzi, hogy terminálni szeretne.

- **cappedAt :: MonadIO m ⇒ m Bool → Word32 → m ()**

A megadott műveletet futtatja a második paraméterben megadott frekvenciával (Hz), amíg a művelet igaz értékkel nem jelzi, hogy terminálni szeretne.

4.7.12. Emulator.CrtShader

- **newShader :: ShaderType → ByteString → IO Shader**

Új GLSL shader lefordítása forráskódból. Fordítási hiba esetén kiírja a hibautenetet majd kivételt dob.

- **createProgramFrom :: [Shader → IO Program]**

Linkeli és validálja a shader-ekből készített programot, hiba esetén kivételt dob.

- **getCrtShaderProgram :: IO Program**

A modul exportált eljárása, ami létrehozza a CRT effektust megvalósító shader programot.

```
1 import Control.Monad ((=<<), sequence)
2
3 getCrtShaderProgram =
4     createProgramFrom =<< sequence
5     [
6         newShader VertexShader crtVertexShader,
7         newShader FragmentShader crtFragmentShader
8     ]
```

4.7.13. Emulator.Window

- **translateSDLEvent :: SDL.EventPayload → Maybe Command**

Megadja egy beérkező eseményhez a hozzá tartozó parancsot.

- **acquireResources :: FilePath → CommResources → IO AppResources**

Inicializálja az SDL2 és OpenGL könyvtárakat, a létrehozott ablakot, renderert, kirajzolásnál használt textúrát, stb. visszaadja egy rekordban.

- **releaseResources :: AppResources → IO ()**

Felszabadítja a lefoglalt erőforrásokat és leállítja az SDL-t.

- **pollCommands :: AppResources → IO [Command]**

Lekérdezi az emulátornak adott parancsokat, amiket kiadhattak billentyűzettel, kontrollerrel vagy akár a felhasználói felülettel is.

- **executeCommand :: AppResources → Command → Emulator Nes ()**

Az eljárás végrehajtja a paraméterül kapott parancsot.

- **updateScreen :: AppResources → IOVector Word8 → Emulator Nes ()**

A paraméterül kapott vektor a megjelenítendő kép nyers pixeladatait tartalmazza RGB24 formátumban. A vektor tartalmát feltölti a videómemóriában

tárolt textúrába, majd ezután egy OpenGL (be van kapcsolva a CRT shader) vagy SDL (ki van kapcsolva) hívással kirajzolja a textúrát a képernyőre.

- **runEmulatorWindow :: FilePath → CommResources → IO ()** A modul exportált eljárása, ami egy ciklusban futtatja a parancsok végrehajtását és az emuláció léptetését. A grafikus felület ezt az eljárást indítja el egy új szalon.

4.8. Egy processzorutasítás végrehajtása

Ideje, hogy felhasználjuk a Nes.CPU.Emulation modulban implementált tömérdek mennyiségű utasítást. Az első lépés, hogy az utasítás opkódját kiolvassuk.

```
1 fetch :: Emulator CPU Opcode
2 fetch = readReg pc >>= read
```

Az opkódmátrix segítségével meg kell keresnünk az opkódhoz tartozó utasítást és ciklusszámot. Az opkódmátrix eltárolásának egy egyszerű módja a case elágazás.

```
1 data DecodedOpcode = DecodedOpcode {
2   instruction :: Emulator CPU (),
3   cycles       :: Int
4 }
5
6 decodeOpcode :: Opcode -> DecodedOpcode
7 decodeOpcode opcode = case opcode of
8   0x00 -> DecodedOpcode (implied >> brk) 7;
9   ...
```

Az implementációmánál a dekódolt opkód nem külön, hanem az utasítással "egy-beégetve" tartalmazza a címzési módot (így könnyebb volt kezelní azt, hogy nem minden opkód címzési módot ad vissza eredményül címet). A címzési módokat megvalósító függvények arra számítanak, hogy a PC már az opkódot követő bájtra mutat, ezért a végrehajtás előtt meg kell növelni a PC értékét eggyel. A *cycle* eljárással megnöveljük az eltelt ciklusok számát a statikusan ismert értékkel, az *elapsedCycles* függvény pedig megnézzük, hogy a végrehajtás után ténylegesen hány ciklus telt el. Így tehát a végső *runNextInstruction* eljárás:

```
1 import Data.Functor (&&gt;)
2
3 elapsedCycles :: Emulator CPU a -> Emulator CPU Int
4 elapsedCycles operation = do
5   cyclesBefore <- readReg cyc
6   operation
7   readReg cyc &&gt; (\cyclesAfter -> cyclesAfter - cyclesBefore)
8
9 runInstruction :: DecodedOpcode -> Emulator CPU ()
10 runInstruction DecodedOpcode{instruction, cycles} = do
11   modifyReg pc (+1)
12   instruction
13   cycle cycles
14
15 runNextInstruction :: Emulator CPU Int
16 runNextInstruction
17   = elapsedCycles (fetch &&gt; decodeOpcode >>= runInstruction)
```

4.9. A CPU és a PPU szinkronizációja

Az emuláció valójában szekvenciálisan történik, de arra oda kell figyelnünk, hogy egy processzorutasítás végrehajtása után megfelelő számú PPU léptetés történjen meg. Mivel a *runNextInstruction* visszaadja eredményül, hogy hány CPU órajel telt el elméletben az utasítás alatt, így már nem nehéz összehangolni a két komponenst annak tudatában, hogy a PPU órajel-frekvenciája háromszorosa a CPU órajel-frekvenciájának. A CPU-nak van hozzáférése a PPU-hoz, ezért ezt a CPU szintjén is meg tudjuk tenni.

```
1 syncCPUwithPPU :: Emulator CPU ()
2 syncCPUwithPPU = do
3   clocks <- CPU.processInterrupts >> CPU.runNextInstruction
4   directPPUAccess $ replicateM_ (clocks * 3) PPU.clock
```

4.10. A grafikus felhasználói felület

4.10.1. Állapotmegjelenítés

```

1 inGame :: State -> Widget Event
2 inGame Emulating{isRunning, romName} =
3     container Box
4     [#orientation := OrientationVertical, #valign := AlignCenter]
5     [
6         ...
7         , BoxChild defaultBoxChildProperties $
8             widget Button
9             [
10                 #label :=
11                     ((if isRunning then "Pause" else "Resume") <> romName)
12                 , on #clicked TogglePause
13             ]
14         ...
15     ]

```

4.10.2. Állapotátmenetek

```

1 update comms Emulating{savePath} ReturnToSelection
2     = Transition (Started Nothing savePath) (sendMsg comms Quit)

1 update _ e@Emulating{savePath = Nothing} QuickSavePressed
2 = Transition e (emit $ chooseSaveFolderFirst)

```

4.11. Teljesítmény

5. fejezet

Tesztelés

A fejlesztés a Test Driven Development (TDD) módszer szerint folyt. Mielőtt belekezdtem volna egy CPU utasítás vagy PPU eljárás implementálásába, a teszteket kibővítettem, hogy az új kódrészeket is lefedjék, ezáltal mindenkor tudtam, hogy jó irányba haladok-e.

Az emulátorok tesztelésére vannak kifejezetten erre a célra készített NES ROM-ok, amik tökéletes aprólékosággal ellenőrzik, hogy minden utasítás az elvárt módon változtatja-e a hardver belső állapotát. Azért döntöttem ezek használata mellett, mert ilyen pontosságú tesztek írása 6502 Assembly-ben rengeteg időt vett volna el és a saját tesztek lefedettsége meg se közelítette volna a professzionális tesztek által nyújtott lefedettséget.

5.1. CPU tesztek

5.1.1. Nestest

Ez volt az első teszt, amin a CPU átment. Részletesen végigteszti az összes utasítást (beleértve néhány illegálisat is). A utasítások címzési módját és sorrendjét is variálja. Összesen 8991 utasítást hajt végre.

5.1.2. Instr_test_v5

16 tesztet tartalmaz, amik a címzési módokat, stack műveleteket, elágazásokat és megszakításokat tesztelik. Remek hibaüzenetek ír a \$6004 címtől kezdve egy nullával

terminált sztring segítségével.

5.1.3. Instr_Misc

2 teszteset ellenőrzi az abszolút indexelt címzés és az elágazások szélsőséges eseteit.

5.2. PPU tesztek

5.2.1. PPU_vbl_nmi

Az itt található 5 tesztcsomag a vertical blanking period státusz-flaget (be/kikapcsolása megfelelő időben történik), az NMI megszakítást (bekövetkezik-e, a PPU regiszterek befolyásolják-e) és a páros-páratlan képkockák 1 PPU órajelciklusnyi hosszkülönbségét ellenőrzik.

5.2.2. PPU_sprite_hit és PPU_sprite_overflow

12 tesztcsomag, amik a sprite réteggel kapcsolatos státusz flag-eket ellenőrzik (Sprite 0 Hit, Sprite Overflow).

6. fejezet

Összegzés

A megszületett program sikeresen demonstrálja a Haskell nyelv alkalmasságát összetett, nagyfokú precizitást igénylő, de mégis hatékony alkalmazások készítésére. A magas szintű absztrakcióknak köszönhetően gyors fejlesztési ütemet tudtam tartani, így új funkciók implementálására és finomhangolására tudtam fordítani azt az időt, amit a hardverközeli nyelveknél a szegmentálási hibák kijavítása emészttet volna fel. Emellett elismerés jár a *gi-gtk-declarative* könyvtárnak, ami kísérleti stádiuma ellenére a gyakorlatban is jól alkalmazható eszköznek bizonyult a felhasználó felület megalkotásához.

Az emulátor remek kompatibilitással rendelkezik és az egyedi megjelenés, valamint a kényelmi funkciók miatt úgy gondolom, hogy jól használható eszköz bárki számára, aki meg szeretne ismerkedni a 80-as évek játékainak világával.

Irodalom

[1] *Master list of NES ROMs*

<http://tuxnes.sourceforge.net/nesmapper.txt>

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper.* (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[3] Knuth: Computers and Typesetting,

<http://www-cs-faculty.stanford.edu/~uno/abcde.html>

Ábrák jegyzéke

2.1.	Nintendo Entertainment System (1985)	7
2.2.	A felhasználói felület játék közben	11
2.3.	Alter Ego	15
2.4.	Lawn Mover	15
3.1.	A NES alaplapja	16
3.2.	A 6502 opkód mátrixa	18
3.3.	Opkód argumentumainak helye	20
3.4.	Indexelt indirekt címzés	22
3.5.	Egy alakzat reprezentálása	27
3.6.	Az Alter Ego játék 0. alakzattáblázata különböző palettákkal kirajzolva	28
3.7.	Attribútumbájtok felosztása	29
4.1.	A felület reakciói az eseményekre	38
4.2.	Az emulációnál használt rekordok	40

Táblázatok jegyzéke

2.1.	Játékok irányításához használt gombok	11
2.2.	Az emuláció vezérlése	12
3.1.	A paletta indexek címei	26
3.2.	A képfeldolgozó memóriatérképe	31

Forráskódjegyzék