



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	4
2. Felhasználói dokumentáció	6
2.1. A feladat ismertetése	6
2.2. Minimum rendszerkövetelmények	7
2.3. Telepítés	7
2.4. Indítás	7
2.5. Kompatibilis kazetták	7
2.6. Főmenü	8
2.6.1. Kazetta kiválasztása	9
2.6.2. Irányítási beállítások megtekintése	9
2.7. Játék alatt elérhető funkciók	10
2.7.1. Irányítás	10
2.7.2. Mentés	11
2.7.3. Betöltés	12
2.7.4. A játék megállítása	13
2.7.5. Visszatérés a főmenübe	13
2.7.6. Képernyőképek	14
3. Fejlesztői dokumentáció I: A NES működésének ismertetése	15
3.1. A NES felépítése	15
3.2. Órajel-frekvenciák	16
3.3. A központi feldolgozóegységhez kapcsolódó fogalmak	16
3.3.1. Opcód	16
3.3.2. Regiszterek	18
3.3.3. Memórialap	18

3.3.4. Hívási verem	18
3.3.5. Megszakítás	19
3.3.6. Opkód argumentum helye, fajtája	19
3.3.7. Címzési módok	20
3.3.8. Memóriatérkép	21
3.3.9. Memóriatükrözés	22
3.3.10. Státusz flagek	22
3.3.11. Utasításkészlet	23
3.4. Kazetták	24
3.4.1. Az iNES fájlformátum	24
3.5. A képfeldolgozó egység	25
3.5.1. Színpalletta	25
3.5.2. Paletta indexek	26
3.5.3. Alakzattáblázat	26
3.5.4. Rétegek	28
3.5.5. Névtáblázatok	28
3.5.6. Attribútumtáblázatok	28
3.5.7. Háttéreltolás	29
3.5.8. OAM (Object Attribute Memory)	29
3.5.9. Regiszterek	29
3.5.10. Memóriatérkép	31
3.5.11. A háttér kirajzolásának egyszerű algoritmusa	31
3.5.12. Sprite réteg kirajzolása	34
4. Fejlesztői dokumentáció II: Megvalósítás	35
4.1. A feladat specifikációja	35
4.2. Fejlesztői környezet	35
4.3. Megvalósítási terv	36
4.3.1. NES emuláció	36
4.3.2. Felhasználói felület	36
4.4. Az emulációt magába záró monád	38
4.5. Adatrepräsentáció	38
4.6. A CPU emulációja	40

4.6.1. Az utasításkészlet implementálása	40
4.6.2. Egy processzorutasítás végrehajtása	40
4.7. A PPU emulációja	41
4.8. A CPU és a PPU szinkronizációja	41
4.9. Megszakítások	42
4.10. iNES fájlok betöltése	42
4.11. Mapperek megvalósítása	42
4.12. Mentés és betöltés	42
4.13. Inputkezelés	42
4.14. A grafikus felhasználói felület	42
4.14.1. Állapotmegjelenítés	42
4.14.2. Állapotátmenetek	42
4.15. Teljesítmény	42
5. Tesztelés	43
6. Bővítési lehetőségek	44
Irodalomjegyzék	45
Ábrajegyzék	46
Táblázatjegyzék	47
Forráskódjegyzék	48

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:

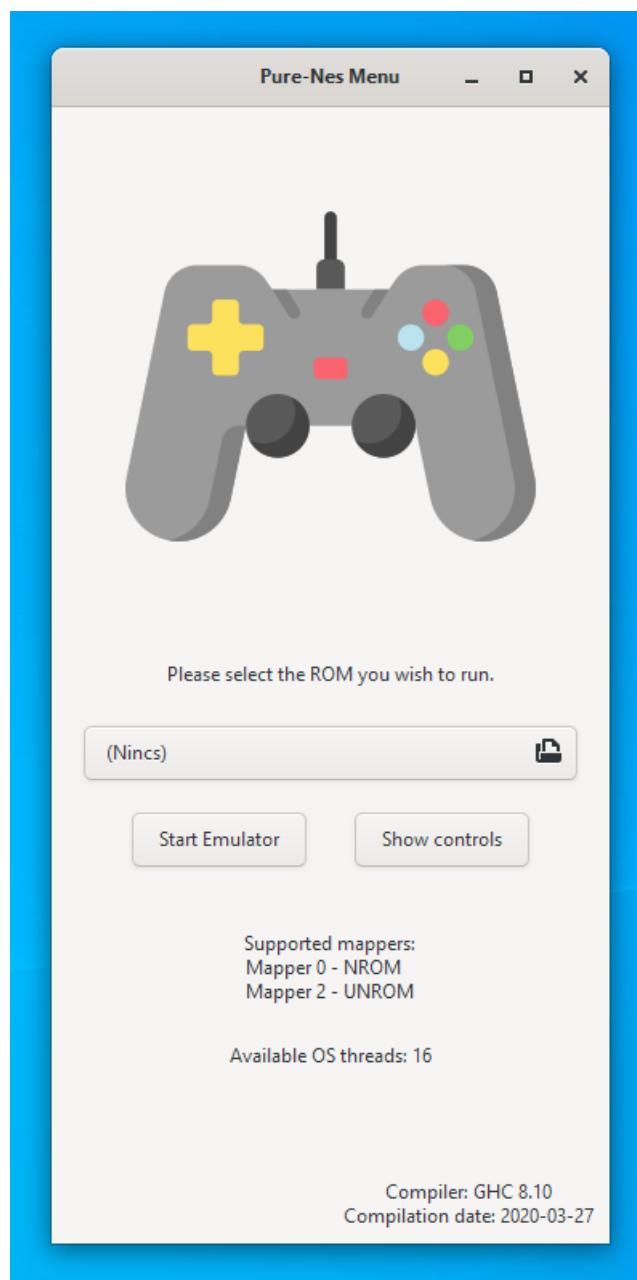
```
1 | pure-nes-1.0$ stack exec pure-nes
```

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

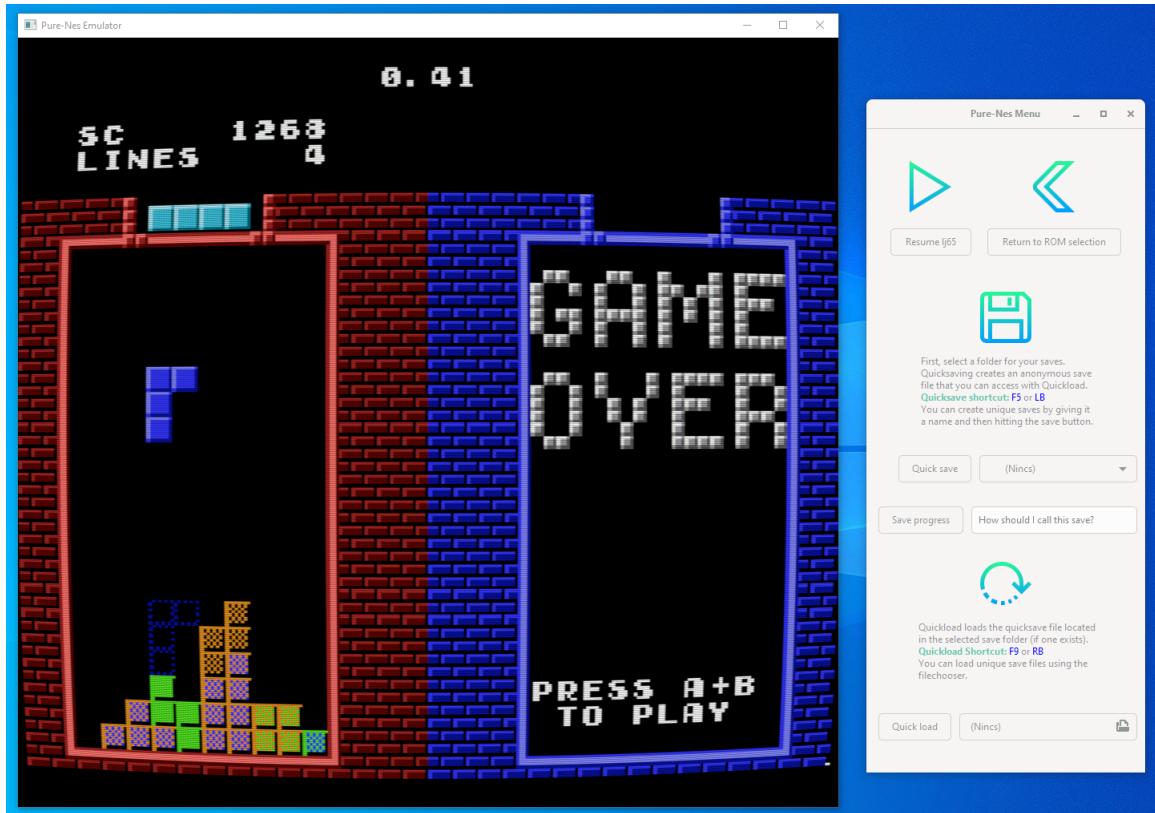
A fájlkiválasztó segítségével adjuk meg a futtatni kívánt kazettát vagy mentést, majd nyomjunk a *Start Emulator* gombra. A program hibaüzenettel jelzi, ha a kazetta nem kompatibilis vagy a fájl formátuma nem megfelelő.

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.

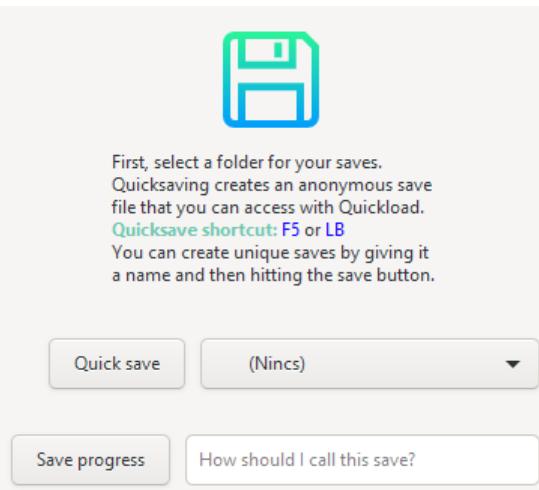
Emulátor funkció	Billentyű
Teljes képernyős nézet ki/be	R
Katódsugárcsöves képernyő-effektus ki/be	T
Szüneteltetés ki/be	Space
100 processzor utasítás végrehajtása (szüneteltetés alatt)	C
Léptetés a következő képkockára (szüneteltetés alatt)	F

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.

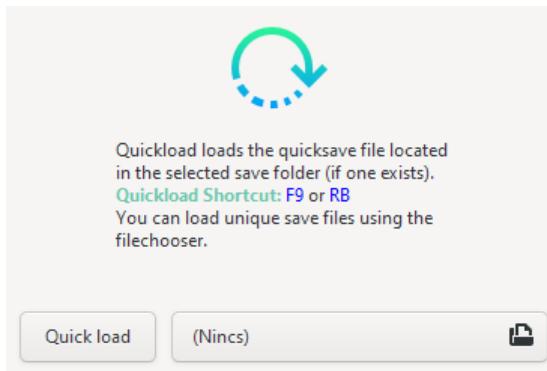
Egyedi mentés létrehozása

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.



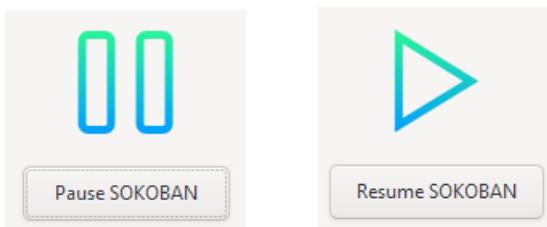
Egyedi mentés betöltése

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

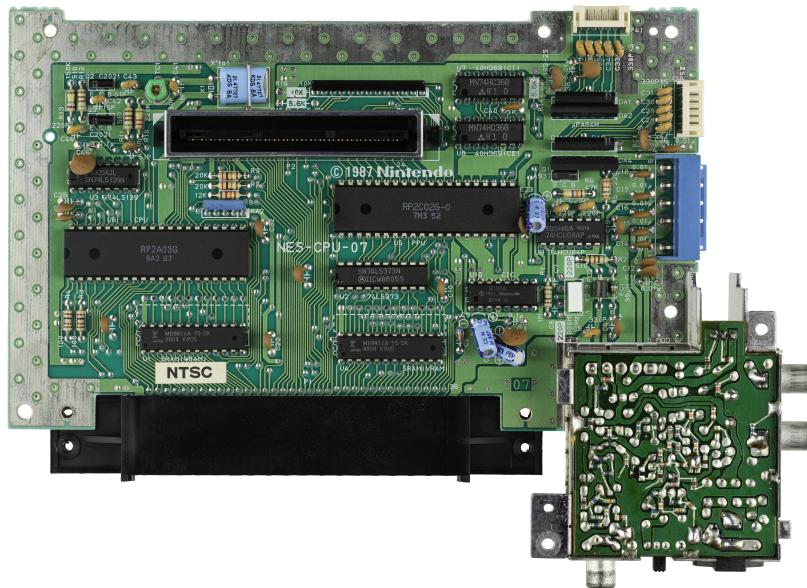


2.4. ábra. Lawn Mover

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.

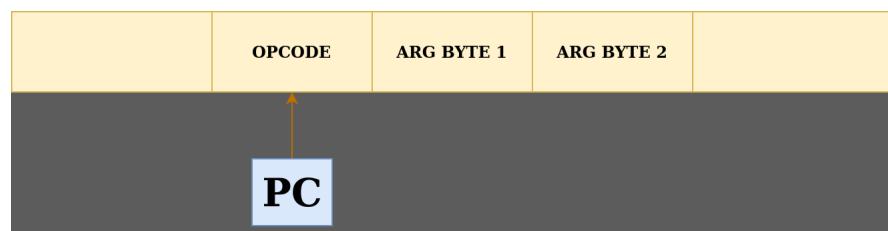
3.3.4. Hívási verem

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 opkód mátrixbeli 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 többit. 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.

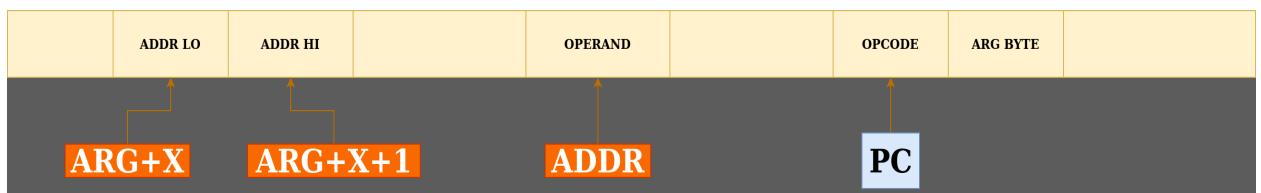
Implicit mód (0): nincs szükség argumentumra, mert az utasítás regisztereikkel dolgozik.

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: Kizárástos VAGY

INC, INX,INY: növelés eggel

DEC, DEX, DEY: csökkentés eggel

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, vagy a programszámláló átállítása erre a címre

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

RTS: Visszatérés szubrutinból (visszatérési cím kiolvasása + **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, avagy 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

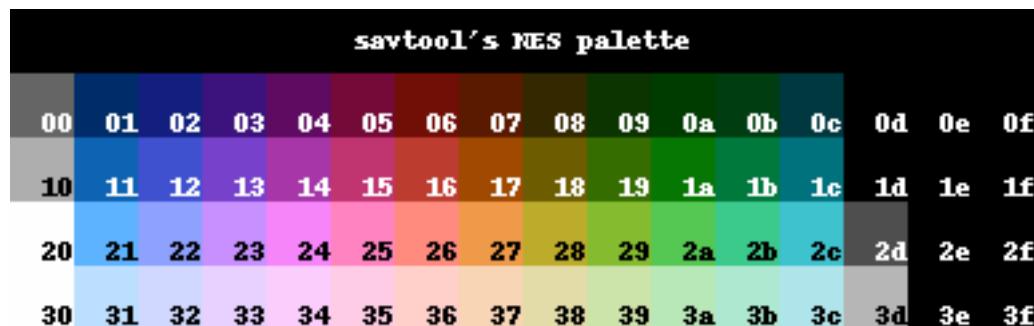
Az iNES fájlformátumot egy korai NES emulátor vezette be a NES játékok bináris formában történő terjesztésére. A fájl elején található 16 bájtos fejléc többek

között a mapper áramkör azonosítóját, a képfeldolgozó névtábláinak tükrözési módját, valamint a PRG ROM és a CHR memóriák típusát (ROM/RAM) és 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. ábra. A 2C02 színpalettája

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 háttérét 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.

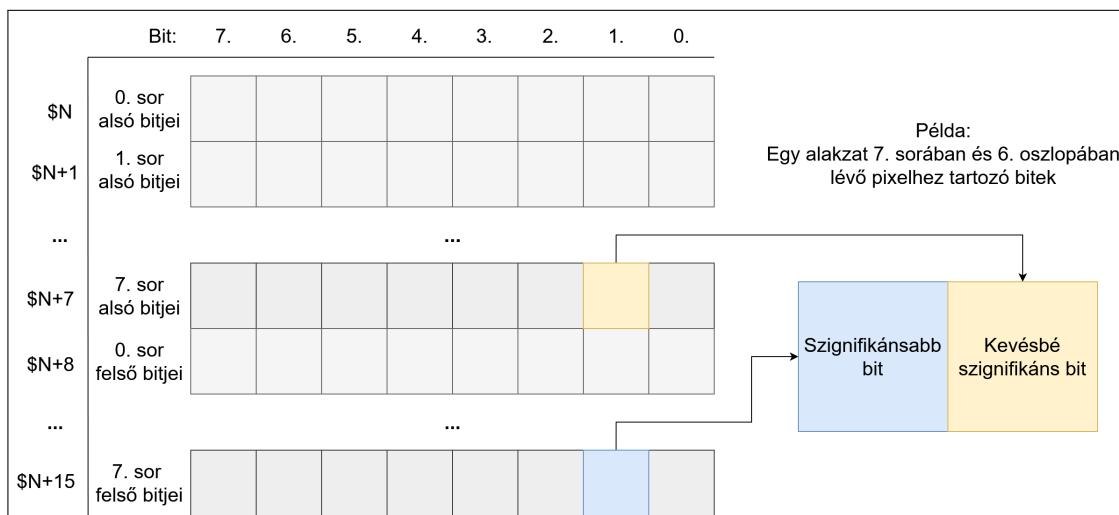
Tartomány	Paletta
$\$3F00$	Univerzális háttérszín
$\$3F01 - \$3F03$	0. háttér paletta
$\$3F05 - \$3F07$	1. háttér paletta
$\$3F09 - \$3F0B$	2. háttér paletta
$\$3F0D - \$3F0F$	3. háttér paletta
$\$3F11 - \$3F13$	0. sprite paletta
$\$3F15 - \$3F17$	1. sprite paletta
$\$3F19 - \$3F1B$	2. sprite paletta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

3.1. táblázat. A paletta RAM struktúrája

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 16×16 darab alakzatot tartalmaz és egy alakzat 8×8 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 3.6 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.



3.6. ábra. Egy alakzat reprezentálása



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

3.5.4. Rétegek

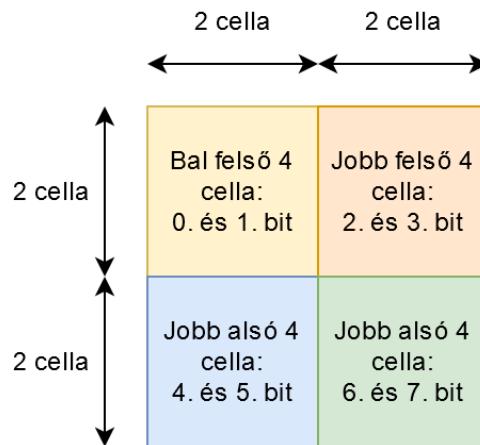
A képfeldolgozó két réteget képes kezelní hardveresen, ezek a háttér és a sprite rétegek. Általában a képkocka azon részei tartoznak a háttérhez, amik ritkán változnak (például feliratok, számlálók, mozdulatlan képek) és a kirajzolásukhoz nem szükséges további transzformáció (eltolás, tükrözés). A háttérben az alakzatok 30 sorból és 32 oszloból 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 mindenek közül 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 mindenkorban egy palettán osztoznak. A 16 cellát lefedő bájt felosztása a 4 cellás területek között a 3.8 ábrán van szemléltetve.



3.8. 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 háttértükörözéstől 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 rétegben.

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 // Az eredményül kapott pixeleket tároló kétdimenziós tömb
21 RGB_Kód pixelek[256][240]
22
23 procedure HáttérKirajzol
24 begin
25     // A CONTROLLER regiszterrel a választott névtáblázat
26     // és alakzattáblázat kezdőcímének meghatározása
27     cím aktívNévtáblazat := $2000 + (CONTROLLER & 0b11) * $400
28     cím aktívAlakzatTáblázat := bit(CONTROLLER, 4) * $1000
29
30     // Végigiterálás a háttér celláin
31     for cellaSor in 0..29
32         for cella0szlop in 0..31
33             begin
34                 // A cellához tartozó névtáblabájt indexének kiszámolása
35                 byte NTB_Eltolás := cellaSor * 32 + cella0szlop
36
37                 // A névtáblabájt kiolvasása
38                 byte NTB := olvas(aktívNévtáblázat + NTB_Eltolás)
```

```

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 (cella0szlop 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 palettaszám.
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 + (cella0szlop & 0b10)
64 byte palettaSorszám :=
65 (ATB >> kvadráns) & 0b11
66
67 // Az alakzat kezdőcíme az alakzattáblában
68 // Segítség: egy alakzat (8*8*2)/8 = 16 bájtot foglal
69 cím alakzatCím := aktívAlakzattáblázat + NTB*16
70
71 // Végigiterálás a cella 8*8 pixeles területén
72 for pixelSor in 0..7
73 begin
74     cím alakzatSor := alakzatCím + pixelSor
75
76     // 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 algoritmusában 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 játékkonzolt 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 kezelni 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 fügőségi könyvtárak telepítéséhez a Stack eszközt használtam. Az SDL2-öt és a GTK-t 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 megfeleltétele 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.

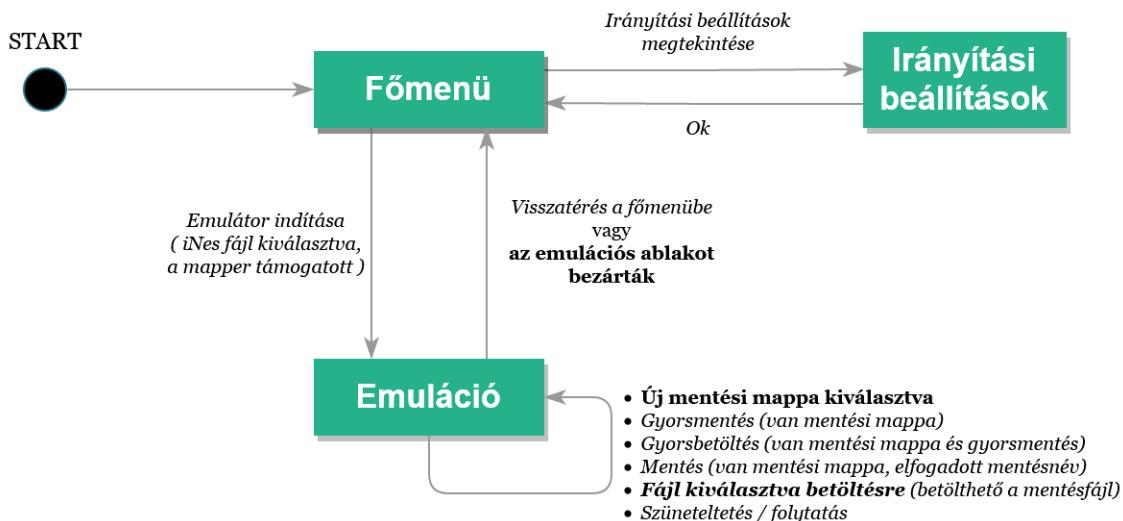
A felhasználói felület moduljai

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.



4.1. ábra. A felület reakciói az eseményekre

A 4.1 ábra az állapotátmenet-függvényt hivatott vizualizálni. 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).

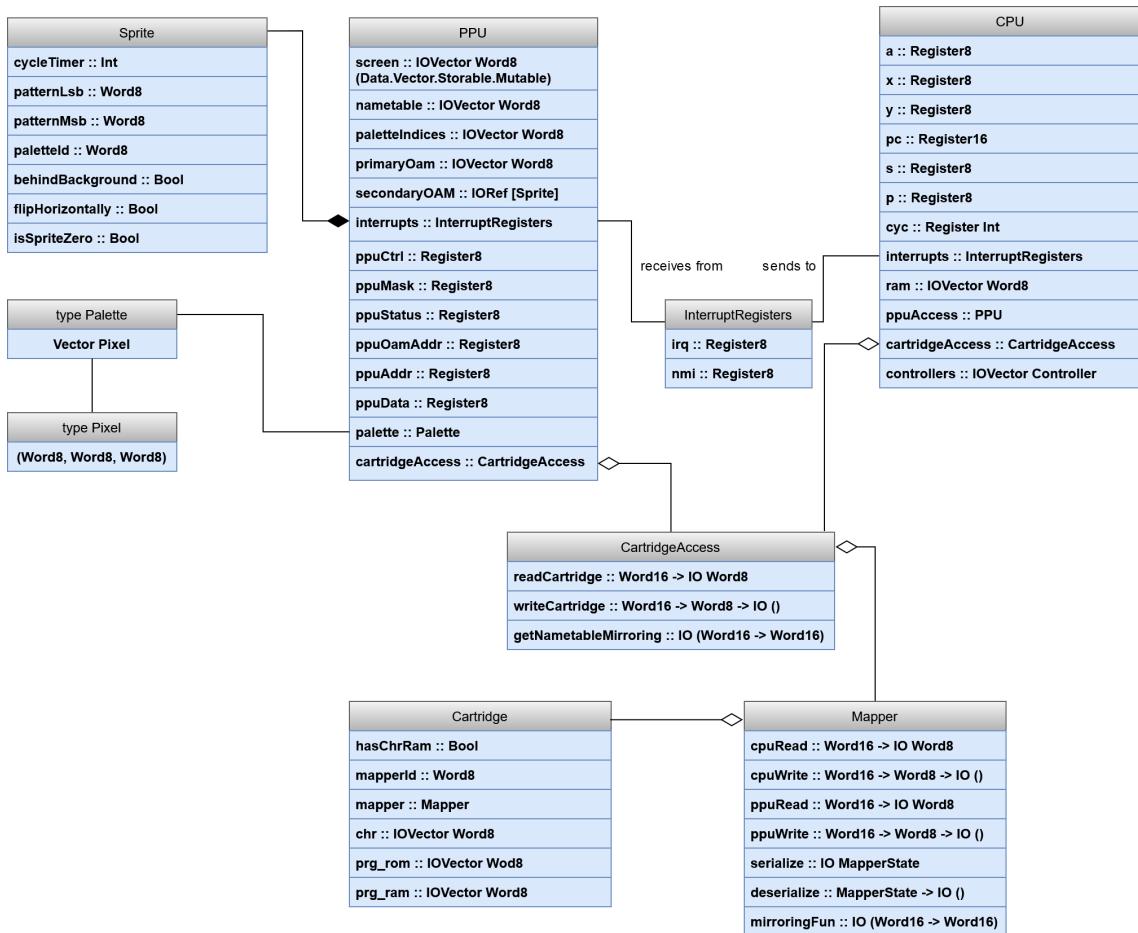
4.4. Az emulációt magába záró monád

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) referenciákkal 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 regisztereiken, 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. A CPU emulációja

4.6.1. Az utasításkészlet implementálása

4.6.2. Egy processzorutasítás végrehajtása

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 opkódroz 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 opkódhoz hozzá kellene adni a címzési módot). A címzési módokat megvalósító függvények arra számítanak, hogy a PC már az opkódot követő bajtra 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* függvénnyel 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 <&> (\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 <&> decodeOpcode) >>= runInstruction
```

4.7. A PPU emulációja

4.8. 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
```

4.9. Megszakítások

4.10. iNES fájlok betöltése

4.11. Mapperek megvalósítása

4.12. Mentés és betöltés

4.13. Inputkezelés

4.14. A grafikus felhasználói felület

4.14.1. Állapotmegjelenítés

4.14.2. Állapotátmenetek

4.15. Teljesítmény

5. fejezet

Tesztele

5.1. Egységtesztek

5.2. Vizuális tesztele

6. fejezet

Bővítési lehetőségek

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)	6
2.2.	A felhasználói felület játék közben	10
2.3.	Alter Ego	14
2.4.	Lawn Mover	14
3.1.	A NES alaplapja	15
3.2.	A 6502 opkód mátrixa	17
3.3.	Opkód argumentumainak helye	19
3.4.	Indexelt indirekt címzés	21
3.5.	A 2C02 színpalettája	25
3.6.	Egy alakzat reprezentálása	27
3.7.	Az Alter Ego játék 0. alakzattablázata különböző palettákkal kirajzolva	27
3.8.	Attribútumbájtok felosztása	29
4.1.	A felület reakciói az eseményekre	37
4.2.	Az emulációnál használt rekordok	39

Táblázatok jegyzéke

2.1. Játékok irányításához használt gombok	10
2.2. Az emuláció vezérlése	11
3.1. A paletta RAM struktúrája	26
3.2. A képfeldolgozó memóriatérképe	31

Forráskódjegyzék