



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	8
2.6.2. Irányítási beállítások megtekintése	9
2.7. Játék alatt elérhető funkciók	10
2.7.1. Irányítás	10
2.7.2. Mentési mappa kiválasztása	10
2.7.3. Mentés	11
2.7.4. Betöltés	12
2.7.5. A játék megállítása	12
2.7.6. Visszatérés a főmenübe	13
2.7.7. Képernyőképek	14
3. Fejlesztői dokumentáció	15
3.1. A NES felépítése	15
3.2. Órajel-frekvenciák	16
3.3. A központi feldolgozóegységhez kapcsolódó fogalmak	16
3.3.1. Opciók	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 argumentumok fajtái	19
3.3.7. Címzési módok	19
3.3.8. Memóriatérkép	21
3.3.9. Memóriatükrözés	21
3.3.10. Státusz flagek	22
3.3.11. Utasításkészlet	22
3.4. Kazetták	23
3.4.1. Az iNES fájl formátum	24
3.5. A képfeldolgozó egységhez kapcsolódó fogalmak	24
3.5.1. Színpalette	24
3.5.2. Paletta indexek	25
3.5.3. Alakzattáblázat	25
3.5.4. Rétegek	26
3.5.5. Névtáblázat	26
3.5.6. Attribútum táblák	27
3.5.7. Háttéreltolás	27
3.5.8. OAM	27
3.5.9. Regiszterek	27
3.5.10. Memóriatérkép	29
3.6. A kirajzolási folyamat	29
3.6.1. Időzítések	29
3.6.2. A háttér kirajzolása	30
3.6.3. Sprite kiértékelés	30
3.7. Az emulátor megvalósítása	30
3.7.1. Adatreprézentáció	30
3.7.2. Monád	30
3.8. A processzor megvalósítása	30
3.9. A képfeldolgozó megvalósítása	30
3.10. Inputkezelés megvalósítása	30

3.11. A grafikus felhasználói felület	30
3.12. Interakció a felhasználóval	30
3.12.1. UML Use-Case diagram	30
3.13. Tesztelés	30
Irodalomjegyzék	31
Ábrajegyzék	31
Táblázatjegyzék	32
Forráskódjegyzék	33

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. Bemeneti eszközöként 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:

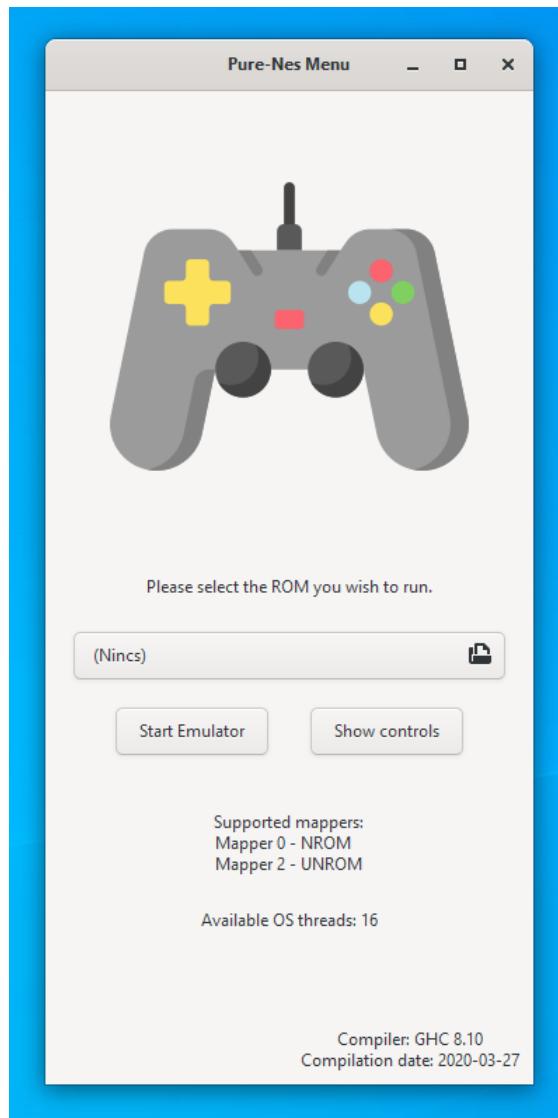
```
1 | pure-nes-1.0$ stack exec pure-nes
```

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. Ezeket Mapper-nek nevezzük. A későbbi játékok előszeretettel használták ki ezt a lehetőséget. A legtöbb speciális áramkör a megnövelt ROM kezeléséhez volt szükséges, de akadt olyan is, amelyik további hangcsatornákat adott hozzá a hangfeldolgozó egységhez. Az emulátorom jelenleg a 0 és 2 típusú Mapper-eket támogatja, ezáltal 351 játékot képes elindítani.

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 ka-

zetta nem kompatibilis vagy a fájl formátuma nem megfelelő.

2.6.2. Irányítási beállítások megtekintése

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

2.7. Játék alatt elérhető funkciók

2.7.1. Irányítás

Virtuális kontroller	Billentyű	Joystick
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 Joystick vezérlők gombkiosztása el szokott térti, 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.

Emulátor funkció	Billentyű
Teljes képernyős nézet ki/be	R
Katódsugárcsöves képernyő-effektus ki/be	T
Szüneteltetés ki/be	Space
100 processzor utasítás végrehajtása szüneteltetett állapotban	C
A következő képkocka kiszámolása szüneteltetett állapotban	F

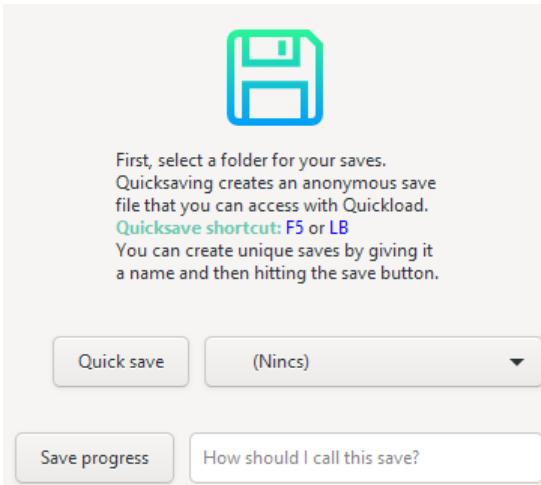
2.2. táblázat. Az emuláció vezérlése

2.7.2. Mentési mappa kiválasztása

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.

2.7.3. Mentés

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



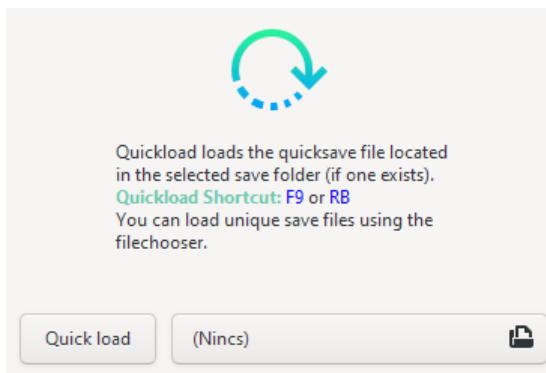
Gyors mentés

A gyors mentési 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: **Quick Save** gomb vagy **F5** billentyű. A mentés sikerességét 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.

2.7.4. Betöltés



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

Gyors betöltés

A gyors betöltési funkció játék közben érhető el, miután kiválasztottuk a mentéseket tároló mappát. A **Quick load** gombbal, vagy az **F9** billentyűvel visszatölthetjük a korábban létrehozott gyorsmentést. A betöltés sikeresége a mentéshez hasonló módon jelenik meg a kezelőfelületen. A sikeres betöltést szintén rezgés fogja követni.

Lehetséges hibaüzenet

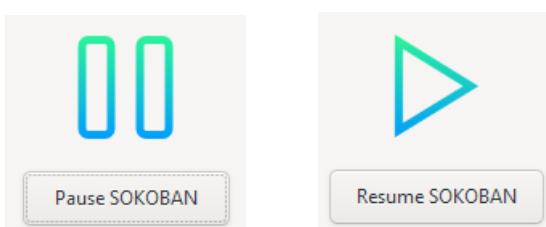
- A kiválasztott mappában nem található gyorsmentés. Megoldás: hozzunk létre egy gyorsmentést.

Egyedi mentés betöltése

A fájlkiválasztó segítségével válasszuk ki a betölteni kívánt mentést.

2.7.5. A játék megállítása

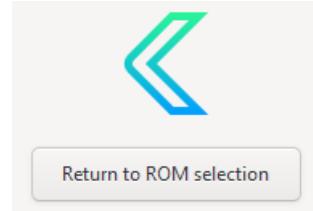
A játék futása bármikor felfüggeszthető a *Pause* gombbal, majd ezután folytatatható a *Resume* gombbal.



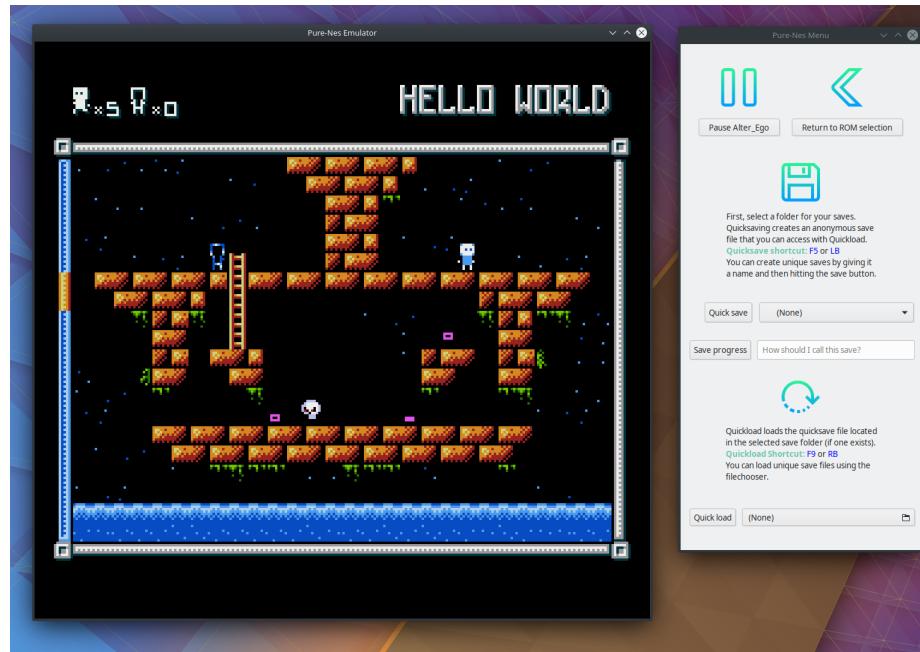
2.7.6. Visszatérés a főmenübe

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

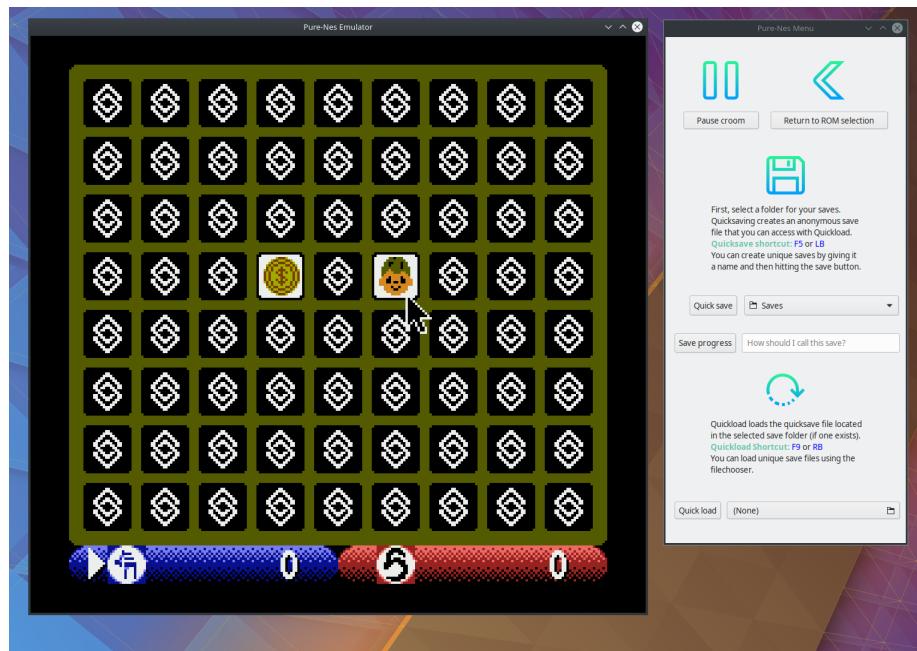
Figyelem: Az állásunk elveszik, ha előtte nem mentünk.



2.7.7. Képernyőképek



2.2. ábra. Alter Ego by Shiru

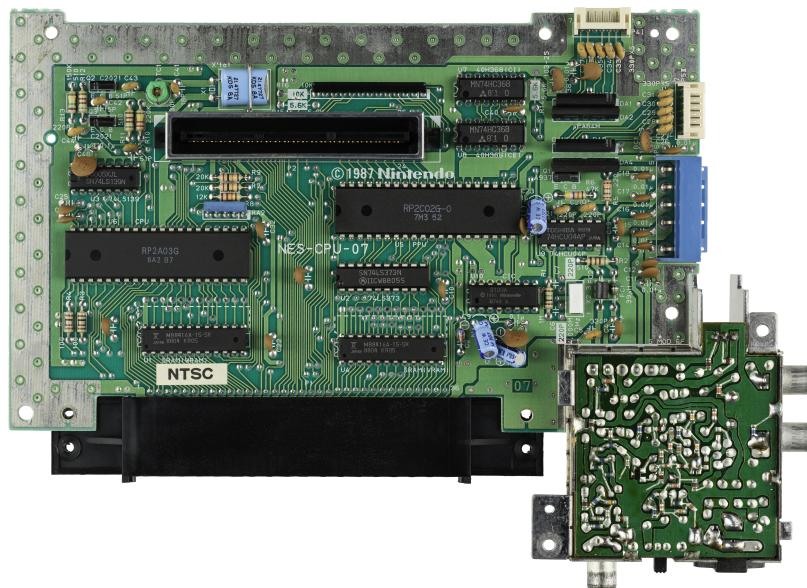


2.3. ábra. Concentration Room by Damian Yerrick

3. fejezet

Fejlesztői dokumentáció

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 tartalmazó integrált áramkör. 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 és UNROM: Az emulátor által támogatott két 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 központi órajel alatt fejeződik be, ugyanis ebből tudjuk megállapítani, hogy a processzor utasításával párhuzamosan hány órajelet kell kapnia a képfeldolgozónak.

	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 4*	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 abx 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 két számjegyű 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 futási ideje csak végrehajtás során dönthető el, az aktuális argumentumok alapján. 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 azonban ezeknek a használatát, egyszerűen csak nem definiálták a viselkedésüket. Ennek ellenére több olyan illegális opkód is belekerült a dizájnba, ami később hasznosnak bizonyult. A fejlesztők 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, így ajánlott ezeket is emulálni.

3.3.2. Regiszterek

A regiszter a processzor leggyorsabban elérhető memóriája. A gyártási költségek alacsonyan tartása végett csak 6 regiszter került a processzorba.

A: Akkumulátor, az aritmetikai műveletek eredményei ebbe kerülnek.

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

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

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

PC: Programszámláló. A következő opkód memóriacímét tárolja. A többi regiszterrel ellentétben ez nem 8, hanem 16 bites. Ebből következik, hogy a processzor címtartományának mérete 64 KiB.

3.3.3. Memórialap

Az i . lap a $[i \cdot \$FF, (i + 1) \cdot \$FF]$ címtartományon elhelyezkedő, 256 bájtos egybefüggő memóriarész.

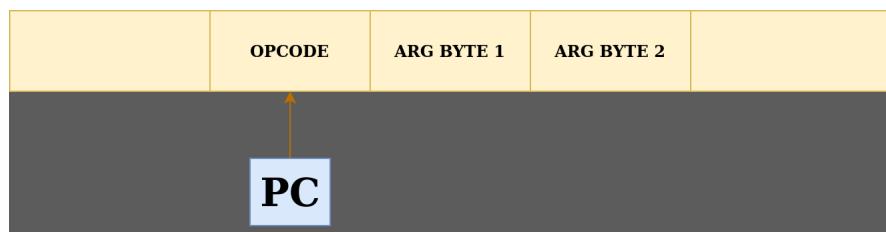
3.3.4. Hívási verem

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 vektorral a program megszabhatja, hogy adott megszakítás esetén a vezérlés melyik szubrutinhoz ugorjon. Például ha van egy szubrutin, amit NMI esetén futtatni szeretnénk, akkor a **\$FFFA** és a **\$FFFB** címekre be kell írni a szubrutin címét, a kevésbé szignifikáns bájttal kezdve. A program dönthet úgy, hogy a maszkolható megszakítást figyelmen kívül hagyja, ehhez a megfelelő flag-et be kell állítania a státusz regiszterben. A nem maszkolható megszakítást 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 argumentumok fajtái



3.3. ábra. Opkód argumentumainak helye

- Abszolút memóriacím (16 bit)
- Eltolást leíró bájt
- Közvetlen operandusként szolgáló bájt

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. A címzési mód mögött álló zárójelben az opkódmátrixbeli név (amennyiben van) és az argumentumok 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): a paraméter 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 többet. Mivel a 0. lap mérete 256 bájt, ezért teljes cím helyett elég egyetlen bájt a címzéséhez. A kisebb paraméter gyorsabban beolvasható és egyúttal a kódméretet 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: $(arg1 + index) \bmod 256$

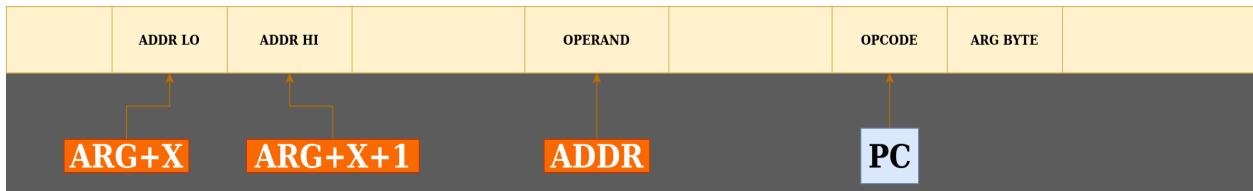
Indexelt abszolút mód (abx, aby, 2): Az argumentum egy teljes memóriacím, amihez hozzáadjuk a megadott index regiszter értékét.

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

Relatív mód (rel, 1): Elágazásoknál ha a feltétel teljesül, akkor az argumentummal el kell tolni a programszámlálót.

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 innen kell kiolvasnunk, vagyis a következőképpen kapjuk meg:
 $(read(m + 1) << 8) \mid read(m)$

Indexelt indirekt mód (izz, 1): Az X regisztert összeadjuk az argumentummal, így egy 0. lapon található címet kapunk. Ezen címen az alsó, a következőn pedig a felső 8 bitje található az operandus effektív memóriacímének.



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 végleges operandus cím és az indexelés előtt álló cím különböző memórialapra esik. Ebben az esetben 1 órajelciklussal tovább tart az utasítás végrehajtása.

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.

3.3.10. Státusz flagek

0. bit: C = Carry
1. bit: Z = Zero
2. bit: I = IRQ Disable
3. bit: D = Decimal mode
4. bit: B = BRK Command
6. bit: V = Overflow
7. bit: N = Negative

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: Kizárással VAGY

INC, INX, INY: növelés eggel

DEC, DEX, DEY: csökkentés eggel

ROL, ROR: bájt forgatása

Összehasonlítás

CMP, CPX, CPY

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 kiolvasása + **JMP**)

BRK: Szoftveresen generált megszakítás

RTI: Visszatérés megszakításkezelőből

Flag manipuláció

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)

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 két fajta 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 CHR ROM/RAM memória a játék sprite-jait tárolja, amik 8*8 pixeles kis képekből állnak. Ezeket a kis képeket ezentúl alakzatoknak fogom hívni.

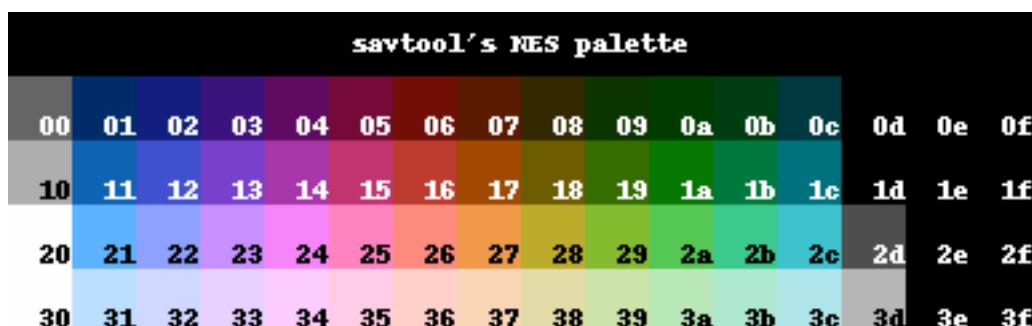
3.4.1. Az iNES fájl formátum

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 egységhez kapcsolódó fogalmak

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, amivel a régi katódsugárcsöves TV-k jellegzetes színtorzítását is vissza tudjuk adni. Az emulátoromnál a gyorsabb módszert implementáltam.



3.5. ábra. A 2C02 színpalettája

3.5.2. Paletta indexek

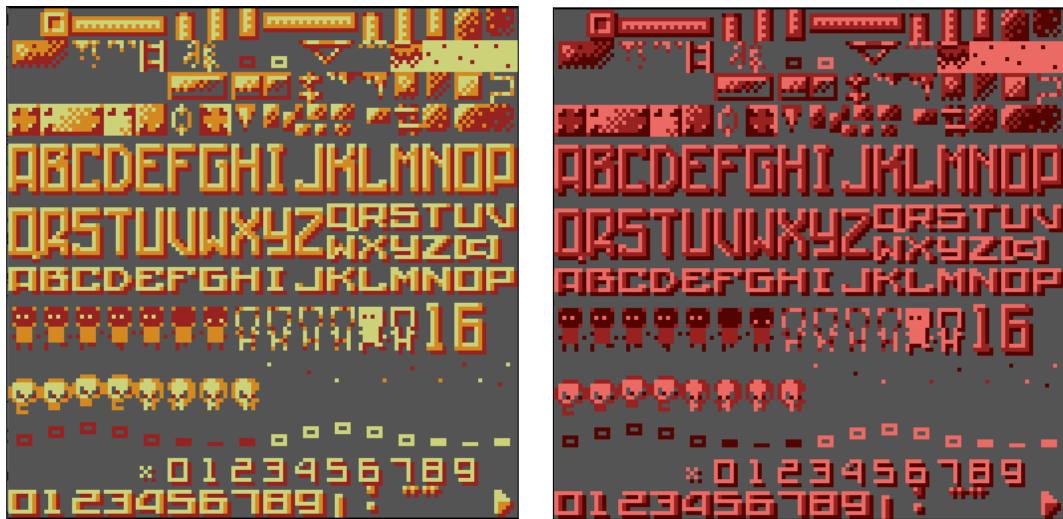
A $\$3F00 - \$3F1F$ címtartományon található memóriaterületen kerülnek eltárolásra a használatban lévő színek azonosítói. A hardveres limitációk miatt a hátteret 16x16 pixeles cellákra osztották fel. Egy cellán belül kizárolag 4 féle szín fordulhatott elő. Ezeket a 4 színből álló színkombinációkat kissé megtévesztő módon szintén palettáknak nevezzük.

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 az alakzatokat tárolja 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 tárolnunk, ami egy 4 színből álló palettán belül azonosít egy színt. Ezzel a megoldással amellett, hogy egyharmadára csökkent a képek mérete, egyben újrafelhasználhatóvá is váltak: a Super Mario Bros. játékban a felhők és bokrok képe ugyanaz, csupán másik palettával kirajzolva. Egy alakzattáblázat 16x16 darab alakzatot tartalmaz, ebből adódóan a táblázat teljes mérete $16 \cdot 16 \cdot (8 \cdot 8 \cdot 2) \div 8 = 4096$ bájt.



3.6. ábra. Az Alter Ego játék 0. mintázat táblája 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, forgatás). A háttérben az alakzatok 30 sorból és 32 oszloból álló négyzetrácsot alkotnak (innen 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 és tetszőleges elforgatottsággal rajzolhatjuk ki az alakzatokat. A sprite rétegnél legfeljebb 64 alakzat szerepelhet egy képkockán.

3.5.5. Névtáblázat

A háttér réteg elrendezését a névtáblázatok tárolják. A névtáblázat a háttér 960 darab alakzatcellájának mindenkorához tartozik, és minden korral együtt van a cellába rajzolandó alakzat sorszámát az aktív alakzattáblázatban.

3.5.6. Attribútum táblák

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 kódolja el. Ennek a reprezentációnak a következménye, hogy a háttér 4 cellából álló csoportjai mindig egy palettán osztoznak.

3.5.7. Háttéreltolás

A háttéreltolással pixel pontossággal megszabhatjuk, hogy a háttér mely része legyen látható a játékos számára.

3.5.8. OAM

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

- X és Y koordináta
- Alakzattáblázatbeli sorszám
- Elforgatottság
- 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.

CONTROLLER(\$2000):

- 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):

- 0. bit: Szürkeárnyalatos effektus
- 1. bit: Háttér kirajzolása a bal szélső 8 pixel széles oszlopban
- 2. bit: Sprite-ok kirajzolása a bal szélső 8 pixel széles oszlopban
- 3. bit: Háttér kirajzolása
- 4. bit: Sprite-ok kirajzolása
- 5. bit: Piros szín intenzitásának növelése
- 6. bit: Zöld szín intenzitásának növelése
- 7. bit: Kék szín intenzitásának növelése

STATUS(\$2002):

- 5. bit: Sprite túlcsordulás
- 6. bit: Sprite 0 találat
- 7. bit: A PPU a tétlen periódusban van

OAMADDR(\$2003) és OAMDATA(\$2004):

A processzor ezen két regiszter segítségével képes új adatokkal feltölteni az OAM memóriát. Az alábbi kódrészlet azt szemlélteti, hogy az *adatok* tömb tartalmát hogyan kell a CPU memóriájából az OAM-ba átmásolni egy megadott címtől kezdve.

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

PPUSCROLL(\$2005):

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

PPUADDR(\$2006) és PPUDATA(\$2007):

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

OAMDMA(\$4014):

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.

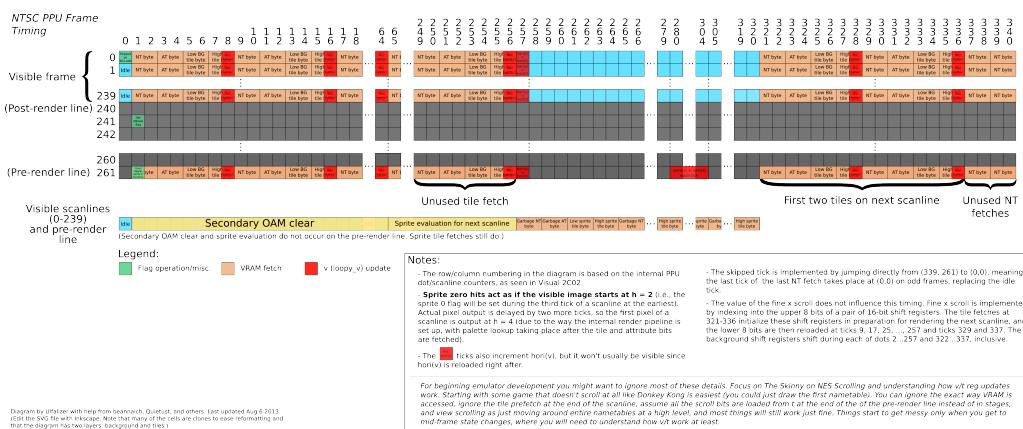
3.5.10. Memóriatérkép

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

3.6. A kirajzolási folyamat

3.6.1. Időzítések



3.6.2. A háttér kirajzolása

3.6.3. Sprite kiértékelés

3.7. Az emulátor megvalósítása

3.7.1. Adatreprézentáció

3.7.2. Monád

3.8. A processzor megvalósítása

3.9. A képfeldolgozó megvalósítása

3.10. Inputkezelés megvalósítása

3.11. A grafikus felhasználói felület

3.12. Interakció a felhasználóval

3.12.1. UML Use-Case diagram

3.13. Tesztelés

Ábrák jegyzéke

2.1.	Nintendo Entertainment System (1985)	6
2.2.	Alter Ego by Shiru	14
2.3.	Concentration Room by Damian Yerrick	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	24
3.6.	Az Alter Ego játék 0. mintázat táblája különböző palettákkal kirajzolva	26

Táblázatok jegyzéke

2.1. Játékok irányításához használt gombok	10
2.2. Az emuláció vezérlése	10
3.1. A paletta RAM struktúrája	25
3.2. A képfeldolgozó memóriatérképe	29

Forráskódjegyzék