Emulator pentru Nintendo Entertainment System

LUCRARE DE LICENȚĂ

Coordonator științific:  
Andrei Stan

Absolvent:  
Rotaru Cristian

**Cuprins**

Introducere ...................................................................................................................................... 3  
Abrevieri ......................................................................................................................................... 4  
Capitolul 1: Structura hardware NES .............................................................................................. 5  
 §1.1 Unitatea centrală de procesare (CPU) ......................................................................... 5  
 §1.1.1 Descrierea pinout a procesorului .................................................................. 6  
 §1.1.2 Regiștrii procesorului ................................................................................... 6  
 §1.1.3 Moduri de adresare ....................................................................................... 7  
 §1.1.4 Setul de instrucțiuni ..................................................................................... 8  
 §1.1.5 Sistemul de întreruperi ............................................................................... 13  
 §1.2 Unitatea de procesare video (PPU) ............................................................................ 14  
 §1.2.1 Descrierea pinout a procesorului grafic ...................................................... 14  
 §1.2.2 Spațiul de adresare ..................................................................................... 15  
 §1.2.3 Registrele PPU ........................................................................................... 15  
 §1.2.4 Modulul DMA ........................................................................................... 18  
 §1.2.5 Paleta de culori ........................................................................................... 18  
 §1.2.6 Tabelele de afișare ...................................................................................... 19  
 §1.2.7 Derularea fundalului .................................................................................. 20  
 §1.2.8 Procesarea sprite-urilor .............................................................................. 22  
 §1.2.9 Randarea .................................................................................................... 23  
 §1.3 Unitatea de procesare audio (APU) ........................................................................... 25

**Introducere**

Scopul acestui proiect este crearea unui program (emulator) care să permită rularea jocurilor create pentru consola de jocuri **NINTENDO ENTERTAINMENT SYSTEM** (NES).

**NINTENDO ENTERTAINMENT SYSTEM** este o consolă de jocuri bazată pe procesorul **MOS6502**. Aceasta a fost lansată pentru prima dată în anul 1983 în Japonia, urmând să fie lansată ulterior în SUA și Europa. Consola **NES** a avut un succes foarte mare, fiind, până în prezent, cea mai vândută consolă din lume, cu peste 60 de milioane de unități vândute.

Pentru această platformă au fost create peste 700 de jocuri (lansate oficial), unele dintre ele fiind considerate legendare: „Super Mario Bros”, „Donkey Kong”, „Legend of Zelda”, „Tetris”, „PAC-MAN”, etc.

Deși consola nu mai este disponibilă pe piață, unii oameni ar dori să aibă posibilitatea de a juca jocurile create pentru această platformă. Această problemă poate fi soluționată prin crearea unui emulator, care să simuleze comportamentul hardware al consolei, permițând astfel, rularea jocurilor pe alte platforme.

Programul creat în cadrul acestui proiect suportă un număr mare de jocuri create pentru consola **NES**.

**Abrevieri**

**AD** – Audio Device

**ALU** – Unitate Aritmetico-Logică

**APU** – Audio Processing Unit

**BCD** – Binary-Coded Decimal

**CIC** – Checking Integrated Circuit

**CISC** – Complex Instruction-Set Computing

**CHR** – Character

**CPU** – Central Processing Unit

**CR** – Cartridge Reader

**CS** – Chip Select

**CVBS** – Composite Video Baseband Signal

**GC** – Game Controller

**MB** – Memory Bus

**MM** – Memory Mapper

**NES** – Nintendo Entertainment System

**NTSC** – National Television System Committe

**OAM** – Object Attribute Memory

**OPC** – Operation Code

**PAL** – Phase Alternating Line

**PPU** – Picture Processing Unit

**PRG** - Program

**RAM** – Random Access Memory

**R/W** – Read / Write

**RO** – Read Only

**RW** – Rendering Window

**WO** – Write Only

**Capitolul 1: Structura hardware NES**

Componentele principale ale unui sistem **NES** sunt: unitatea de procesare centrală (**CPU**) și audio (**APU**) încorporate în cipul **Ricoh RP2A03** (sau **RP2A07**, în dependență de regiune), unitatea de procesare grafică (**PPU**), memoria **RAM**, memoria **V-RAM** și cipul de securitate (**CIC**). Alte componente importante sunt plasate pe cartridge: memoria de program (**PRG**), memoria de elemente grafice (**CHR**), **RAM** extern (opțional), mapper de memorie (opțional) și un alte cip de securitate (**CIC**).

La consolă trebuie conectate 1 sau 2 controlere și un televizor ce suportă intrări **CVBS** și audio.

Schema funcțională a sistemului este descrisă în *figura 1.1*.



*figura 1.1 (schema funcțională a sistemului NES)*

**§1.1 Unitatea centrală de procesare (CPU)**

Nucleul de procesare a consolei **NES** este bazat pe procesorul **MOS6502**, diferența dintre acestea fiind lipsa modului **BCD** la instrucțiunile de adunare și scădere. **MOS6502** este un microprocesor de tip **CISC** pe 8 biți cu un bus de adrese pe 16 biți, capabil să lucreze la frecvențe de până la 3MHz.

Unitatea de procesare este integrată în capsula **Ricoh RP2A03** (**RP2A07**), care conține și procesorul audio.

**§1.1.1 Descrierea pinout a procesorului**



*figura 1.2 (RP2A03 / RP2A07 pinout)*

**[AD1 / AD2]** – Ieșiri audio (*vezi add\_ref*)

**RST** – Pin de reset activ pe 0

**[A00:A15]** – Bus de adrese. Menține adresa țintă pe parcursul unui ciclu de citire/scriere.

**[D0:D7]** – Bus de date. La citire, datele sunt citite de pe acești pini. La scriere, datele sunt plasate pe acești pini.

**CLK** – Intrare de clock (*vezi add\_ref*)

**TST** – Mod de test activ pe 1. La activare, regiștrii de test devin accesibili programelor ce rulează.

**M2** – „signals ready”. Semnalează perifericelor că datele sunt disponibile pe **[D0:D7]**.

**IRQ** – Pin de întrerupere activ pe palierul negativ. (*vezi add\_ref*)

**NMI** – Pin de întrerupere nemascabilă activ pe frontul negativ. (*vezi add\_ref*)

**R/W** – Indică dacă operația este de scriere sau de citire. 1 = citire, 0 = scriere.

**[OE1 / OE2]** – Activează/dezactivează semnalele de output pentru controlerele de joc.

**[OUT0:OUT2]** – Portul de expansiune al procesorului (registrul $4016).

**§1.1.2 Regiștrii procesorului**

Nucleul de procesare conține un registru acumulator (**A**), 2 regiștri de indexare (**X**, **Y**), un program counter (**PC**), un stack pointer (**SP**) și un registru de flag-uri (**P**). Regiștrii **A**, **X**, **Y**, **SP** și **P** sunt pe 8 biți, iar registrul **PC** este pe 16 biți.

Registrul acumulator este folosit de unitatea aritmetico-logică pentru stocarea rezultatelor operațiilor în modul de adresare **accumulator** (*vezi add\_ref*).

Regiștrii **X** și **Y** sunt folosiți în anumite moduri de adresare ale instrucțiunilor. Aceștia pot fi folosiți pentru numărarea iterațiilor în bucle sau la accesul la blocuri de memorie alocate continuu.

Registrul **SP** este folosit pentru urmărirea vârfului stivei la operațiunile **push** și **pop**. Adresa din memoria a vârfului stivei este calculat astfel: $(SP) | 0x0100.

Registrul **PC** indică adresa instrucțiunii curente ce trebuie executată.

Registrul de flag-uri conține informații despre starea procesorului. Acesta poate fi modificat de unitatea aritmetico-logică și de sistemul de întreruperi. Valorile anumitor biți din acest registru determină rezultatul instrucțiunilor de salt condiționat (*vezi add\_ref*).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | O | - | B | D | I | Z | C |

*figura 1.3 (registrul de flag-uri)*

**C**(carry) – este setat dacă ultima operațiune **ALU** a efectuat transport către exterior.

**Z**(zero) – este setat dacă ultima operațiune **ALU** a returnat 0.

**I**(inhibit) – este setat de fiecare dată când are loc o întrerupere. Dacă este setat, întreruperile **IRQ** sunt inhibate (*vezi add\_ref*).

**D**(decimal) – dacă este setat, operațiile de adunare și scădere rulează în modul **BCD**. (nu se aplică în cazul procesorului din consola **NES**)

**B**(break) – este setat la sfârșitul unei întreruperi de tip **BRK** (*vezi add\_ref*).

**O**(overflow) – este setat de **ALU** în cazul unei erori de overflow.

**N**(negative) – este setat dacă ultima operațiune **ALU** a returnat un număr negativ.

**§1.1.3 Moduri de adresare**

Procesorul **MOS6502** are un spațiu de adrese pe 16 biți ($0000-$FFFF). Maparea memorie este descrisă în *tabelul 1.1*.

|  |  |  |  |
| --- | --- | --- | --- |
| Adresa de început | Adresa de sfârșit | Dimensiunea | Descriere |
| $0000 | $07FF | 2KB | **RAM** intern |
| $0800 | $1FFF | 6KB | Pointează spre spațiul $0000-$07FF (se repetă la fiecare 2KB) |
| $2000 | $2007 | 8B | Regiștri **PPU** |
| $2008 | $3FFF | 8KB – 8B | Pointează spre regiștrii **PPU** (se repetă la fiecare 8B) |
| $4000 | $401F | 32B | Regiștri **APU** și **I/O** |
| $4020 | $5FFF | 8160B | Spațiu nefolosit |
| $6000 | $7FFF | 8K | **RAM** extern |
| $8000 | $FFFF | 32K | **PRG ROM** |

*tabelul 1.1 (spațiul de adrese al procesorului)*

Vectorii de întreruperi și de reset se găsesc la adresele: (*vezi add\_ref*)

* $FFFA-$FFFB – vectorul **NMI**
* $FFFC-$FFFD – vectorul de reset
* $FFFE-$FFFF – vectorul **BRK** / **IRQ**

Spațiul de memorie este împărți în pagini de 256B (mapabile pe 8 biți). Dacă în timpul unei operațiuni de citire pentru încărcarea unui registru de 16 biți (*vezi add\_ref*) este necesară schimbarea paginii, operațiunea va dura cu un ciclu mai mult.

Instrucțiunile procesorului pot accesa memoria prin intermediul a 13 moduri de adresare descrise în *tabelul 1.2*.

|  |  |  |
| --- | --- | --- |
| Denumire | Asamblare | Descriere |
| Implicit | OPC | Instrucțiunea nu are operand. Destinația rezultatului este implicită. |
| Accumulator | OPC A | Operația este executată asupra registrului **A**. |
| Immediate | OPC #$*BB* | Operandul sursă este o valoare imediată pe 8 biți. Rezultatul se scrie în unul din registrele procesorului în dependență de instrucțiune. |
| Zero Page | OPC $*LL* | Operandul este luat din pagina 0 a memoriei **RAM** de la o adresă dată pe 8 biți. |
| Absolute | OPC $*LLHH* | Operandul este luat de la o adresă dată pe 16 biți. |
| Relative | OPC $*BB* | Este folosit de instrucțiunile de salt. Operandul reprezintă un offset, valoare pe 8 biți cu semn. |
| Indirect | OPC ($*LLHH*) | Este folosit de o instrucțiune de salt necondiționat. Operandul reprezintă o adresă pe 16 biți la care se găsește valoarea ce urmează a fi încărcată în registrul **PC**. |
| Zero Page indexed by X | OPC $*LL*,X | Operandul se află la adresa ($*LL* + X) & 0xFF. |
| Zero Page indexed by Y | OPC $*LL*,Y | Operandul se află la adresa ($*LL* + Y) & 0xFF. |
| Absolute indexed by X | OPC $*LLHH*,X | Operandul se află la adresa $*LLHH* + X. |
| Absolute indexed by Y | OPC $*LLHH*,Y | Operandul se află la adresa $*LLHH* + Y. |
| Indexed indirect | OPC ($*LL*,X) | Operandul se obține în 3 pași:  baseAddr=read($*LL*)+X  addr=read(baseAddr)|read((baseAddr+1)&0xFF)<<8  operand=read(addr) |
| Indirect indexed | OPC ($*LL*),Y | Operandul se obține în 3 pași:  baseAddr=read($*LL*)  addr=(read(baseAddr)|(read(baseAddr+1)&0xFF)<<8)+Y  operand=read(addr) |

*tabelul 1.2 (moduri de adresare)*

**§1.1.4 Setul de instrucțiuni**

Setul oficial de instrucțiuni al procesorului **MOS6502** este constituit din 151 de instrucțiuni codificate pe 8 biți. Fiecare instrucțiune poate fi urmată de un parametru pe 8 sau 16 biți. Întreg setul este descris în *tabelul 1.3*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Adresare | Asamblare | Cod | Descriere |
| BRK | Implicit | BRK | 00 | Cauzează o întrerupere de tip **BRK**. (*vezi add\_ref*) |
| PHP | Implicit | PHP | 08 | Pune pe stivă registrul de flag-uri. |
| PLP | Implicit | PLP | 28 | Ia de pe stivă registrul de flag-uri. |
| PHA | Implicit | PHA | 48 | Pune pe stivă registrul acumulator (**A**). |
| PLA | Implicit | PLA | 68 | Ia de pe stivă registrul acumulator (**A**). |
| CLC | Implicit | CLC | 18 | Setează flag-ul *carry* (**C**) pe 0. |
| SEC | Implicit | SEC | 38 | Setează flag-ul *carry* (**C**) pe 1. |
| CLI | Implicit | CLI | 58 | Setează flag-ul *inhibit* (**I**) pe 0. |
| SEI | Implicit | SEI | 78 | Setează flag-ul *inhibit* (**I**) pe 1. |
| CLD | Implicit | CLD | D8 | Setează flag-ul *decimal* (**D**) pe 0. |
| SED | Implicit | SED | F8 | Setează flag-ul *decimal* (**D**) pe 1. |
| CLV | Implicit | CLV | B8 | Setează flag-ul *overflow* (**O**) pe 0. |
| TAX | Implicit | TAX | AA | Copie valoarea registrului **A** în registrul **X**.  Flag-uri afectate: **N**, **Z** |
| TAY | Implicit | TAY | A8 | Copie valoarea registrului **A** în registrul **Y**.  Flag-uri afectate: **N**, **Z** |
| TSX | Implicit | TSX | BA | Copie valoarea registrului **SP** în registrul **X**.  Flag-uri afectate: **N**, **Z** |
| TXA | Implicit | TXA | 8A | Copie valoarea registrului **X** în registrul **A**.  Flag-uri afectate: **N**, **Z** |
| TYA | Implicit | TYA | 98 | Copie valoarea registrului **Y** în registrul **A**.  Flag-uri afectate: **N**, **Z** |
| TXS | Implicit | TXS | 9A | Copie valoarea registrului **X** în registrul **SP**. |
| NOP | Implicit | NOP | EA | Nici o operațiune. |
| INX | Implicit | INX | E8 | Incrementează valoarea registrului **X**. |
| INY | Implicit | INY | C8 | Incrementează valoarea registrului **Y**. |
| DEX | Implicit | DEX | CA | Decrementează valoarea registrului **X**. |
| DEY | Implicit | DEY | 88 | Decrementează valoarea registrului **Y**. |
| JSR | Absolute | JSR $*LLHH* | 20 | Salt la subrutină. Salvează pe stivă valoarea registrului **PC**+1 și sare la adresa $*LLHH*. |
| RTS | Implicit | RTS | 60 | Revenire din subrutină. Citește de pe stivă valoarea registrului **PC**. |
| RTI | Implicit | RTI | 40 | Revenire din întrerupere. Citește de pe stivă valoarea registrului **PC** și a registrului de flag-uri. |
| JMP | Indirect | JMP ($*LLHH*) | 6C | Salt necondiționat la o nouă locație. |
| Absolute | JMP $*LLHH* | 4C |
| LDA | Immediate | LDA #$*BB* | A9 | Încarcă o valoare în registrul **A**.  Flag-uri afectate: **N**, **Z** |
| Zero Page | LDA $*LL* | A5 |
| Zero Page indexed by X | LDA $*LL*,X | B5 |
| Absolute | LDA $*LLHH* | AD |
| Absolute indexed by X | LDA $*LLHH*,X | BD |
| Absolute indexed by Y | LDA $*LLHH*,Y | B9 |
| Indexed indirect | LDA ($*LLHH*,X) | A1 |
| Indirect indexed | LDA ($*LLHH*),Y | B1 |
| LDX | Immediate | LDX #$*BB* | A2 | Încarcă o valoare în registrul **X**.  Flag-uri afectate: **N**, **Z** |
| Zero Page | LDX $*LL* | A6 |
| Zero Page indexed by Y | LDX $*LL*,Y | B6 |
| Absolute | LDX $*LLHH* | AE |
| Absolute indexed by Y | LDX $*LLHH*,Y | BE |
| LDY | Immediate | LDY #$*BB* | A0 | Încarcă o valoare în registrul **Y**.  Flag-uri afectate: **N**, **Z** |
| Zero Page | LDY $*LL* | A4 |
| Zero Page indexed by X | LDY $*LL*,X | B4 |
| Absolute | LDY $*LLHH* | AC |
| Absolute indexed by X | LDY $*LLHH*,X | BC |
| STA | Zero Page | STA $*LL* | 85 | Stochează valoarea registrului **A**. |
| Zero Page indexed by X | STA $*LL*,X | 95 |
| Absolute | STA $*LLHH* | 8D |
| Absolute indexed by X | STA $*LLHH*,X | 9D |
| Absolute indexed by Y | STA $*LLHH*,Y | 99 |
| Indexed indirect | STA ($*LLHH*,X) | 81 |
| Indirect indexed | STA ($*LLHH*),Y | 91 |
| STX | Zero Page | STX $*LL* | 86 | Stochează valoarea registrului **X**. |
| Zero Page indexed by Y | STX $*LL*,Y | 96 |
| Absolute | STX $*LLHH* | 8E |
| STY | Zero Page | STY $*LL* | 84 | Stochează valoarea registrului **Y**. |
| Zero Page indexed by X | STY $*LL*,X | 94 |
| Absolute | STY $*LLHH* | 8C |
| CMP | Immediate | CMP #$*BB* | C9 | Compară o valoare din memorie cu valoarea acumulatorului (**A**).  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | CMP $*LL* | C5 |
| Zero Page indexed by X | CMP $*LL*,X | D5 |
| Absolute | CMP $*LLHH* | CD |
| Absolute indexed by X | CMP $*LLHH*,X | DD |
| Absolute indexed by Y | CMP $*LLHH*,Y | D9 |
| Indexed indirect | CMP ($*LLHH*,X) | C1 |
| Indirect indexed | CMP ($*LLHH*),Y | D1 |
| CPX | Immediate | CPX #$*BB* | E0 | Compară o valoare din memorie cu valoarea registrului **X**.  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | CPX $*LL* | E4 |
| Absolute | CPX $*LLHH* | EC |
| CPY | Immediate | CPY #$*BB* | C0 | Compară o valoare din memorie cu valoarea registrului **Y**.  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | CPY $*LL* | C4 |
| Absolute | CPY $*LLHH* | CC |
| BIT | Zero Page | BIT $*LL* | 24 | Biții 7 și 6 ai operandului sunt copiați în biții 7 și 6 ai registrului de flag-uri. Dacă operandul este 0, se setează flag-ul **N**. |
| Absolute | BIT $*LLHH* | 2C |
| ADC | Immediate | ADC #$*BB* | 69 | Adună valoarea acumulatorului (**A**) cu o valoare din memorie și cu bitul de *carry* (**C**) și stochează rezultatul în acumulator.  Flag-uri afectate: **O**, **N**, **Z**, **C** |
| Zero Page | ADC $*LL* | 65 |
| Zero Page indexed by X | ADC $*LL*,X | 75 |
| Absolute | ADC $*LLHH* | 6D |
| Absolute indexed by X | ADC $*LLHH*,X | 7D |
| Absolute indexed by Y | ADC $*LLHH*,Y | 79 |
| Indexed indirect | ADC ($*LLHH*,X) | 61 |
| Indirect indexed | ADC ($*LLHH*),Y | 71 |
| SBC | Immediate | SBC #$*BB* | E9 | Din valoarea acumulatorului (**A**) scade o valoare din memorie și bitul de *carry* (**C**).  Flag-uri afectate: **O**, **N**, **Z**, **C** |
| Zero Page | SBC $*LL* | E5 |
| Zero Page indexed by X | SBC $*LL*,X | F5 |
| Absolute | SBC $*LLHH* | ED |
| Absolute indexed by X | SBC $*LLHH*,X | FD |
| Absolute indexed by Y | SBC $*LLHH*,Y | F9 |
| Indexed indirect | SBC ($*LLHH*,X) | E1 |
| Indirect indexed | SBC ($*LLHH*),Y | F1 |
| ASL | Accumulator | ASL A | 0A | Se realizează o deplasare la stânga. Bitul 7 al operandului este copiat în *carry* (**C**).  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | ASL $*LL* | 06 |
| Zero Page indexed by X | ASL $*LL*,X | 16 |
| Absolute | ASL $*LLHH* | 0E |
| Absolute indexed by X | ASL $*LLHH*,X | 1E |
| LSR | Accumulator | LSR A | 4A | Se realizează o deplasare aritmetică la dreapta. Bitul 0 al operandului este copiat în *carry* (**C**).  Flag-uri afectate: **Z**, **C** |
| Zero Page | LSR $*LL* | 46 |
| Zero Page indexed by X | LSR $*LL*,X | 56 |
| Absolute | LSR $*LLHH* | 4E |
| Absolute indexed by X | LSR $*LLHH*,X | 5E |
| ROL | Accumulator | ROL A | 2A | Se realizează o rotire la stânga cu *carry* (**C**).  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | ROL $*LL* | 26 |
| Zero Page indexed by X | ROL $*LL*,X | 36 |
| Absolute | ROL $*LLHH* | 2E |
| Absolute indexed by X | ROL $*LLHH*,X | 3E |
| ROR | Accumulator | ROR A | 6A | Se realizează o rotire la dreapta cu *carry* (**C**).  Flag-uri afectate: **N**, **Z**, **C** |
| Zero Page | ROR $*LL* | 66 |
| Zero Page indexed by X | ROR $*LL*,X | 76 |
| Absolute | ROR $*LLHH* | 6E |
| Absolute indexed by X | ROR $*LLHH*,X | 7E |
| ORA | Immediate | ORA #$*BB* | 09 | Realizează operația *sau logic* între acumulator și operand.  Flag-uri afectate: **N**, **Z** |
| Zero Page | ORA $*LL* | 05 |
| Zero Page indexed by X | ORA $*LL*,X | 15 |
| Absolute | ORA $*LLHH* | 0D |
| Absolute indexed by X | ORA $*LLHH*,X | 1D |
| Absolute indexed by Y | ORA $*LLHH*,Y | 19 |
| Indexed indirect | ORA ($*LLHH*,X) | 01 |
| Indirect indexed | ORA ($*LLHH*),Y | 11 |
| AND | Immediate | AND #$*BB* | 29 | Realizează operația *și logic* între acumulator și operand.  Flag-uri afectate: **N**, **Z** |
| Zero Page | AND $*LL* | 25 |
| Zero Page indexed by X | AND $*LL*,X | 35 |
| Absolute | AND $*LLHH* | 2D |
| Absolute indexed by X | AND $*LLHH*,X | 3D |
| Absolute indexed by Y | AND $*LLHH*,Y | 39 |
| Indexed indirect | AND ($*LLHH*,X) | 21 |
| Indirect indexed | AND ($*LLHH*),Y | 31 |
| EOR | Immediate | EOR #$*BB* | 49 | Realizează operația *sau exclusiv* între acumulator și operand.  Flag-uri afectate: **N**, **Z** |
| Zero Page | EOR $*LL* | 45 |
| Zero Page indexed by X | EOR $*LL*,X | 55 |
| Absolute | EOR $*LLHH* | 4D |
| Absolute indexed by X | EOR $*LLHH*,X | 5D |
| Absolute indexed by Y | EOR $*LLHH*,Y | 59 |
| Indexed indirect | EOR ($*LLHH*,X) | 41 |
| Indirect indexed | EOR ($*LLHH*),Y | 51 |
| INC | Zero Page | INC $*LL* | E6 | Incrementează cu 1 valoarea operandului.  Flag-uri afectate: **N**, **Z** |
| Zero Page indexed by X | INC $*LL*,X | F6 |
| Absolute | INC $*LLHH* | EE |
| Absolute indexed by X | INC $*LLHH*,X | FE |
| DEC | Zero Page | DEC $*LL* | C6 | Decrementează cu 1 valoarea operandului.  Flag-uri afectate: **N**, **Z** |
| Zero Page indexed by X | DEC $*LL*,X | D6 |
| Absolute | DEC $*LLHH* | CE |
| Absolute indexed by X | DEC $*LLHH*,X | DE |
| BEQ | Relative | BEQ $*BB* | F0 | Salt condiționat de **Z**=1. |
| BNE | Relative | BNE $*BB* | D0 | Salt condiționat de **Z**=0. |
| BMI | Relative | BMI $*BB* | 30 | Salt condiționat de **N**=1. |
| BPL | Relative | BPL $*BB* | 10 | Salt condiționat de **N**=0. |
| BCS | Relative | BCS $*BB* | B0 | Salt condiționat de **C**=1. |
| BCC | Relative | BCC $*BB* | 90 | Salt condiționat de **C**=0. |
| BVS | Relative | BVS $*BB* | 70 | Salt condiționat de **O**=1. |
| BVC | Relative | BVC $*BB* | 50 | Salt condiționat de **O**=0. |

*tabelul 1.3 (instrucțiunile procesorului 6502)*

**§1.1.5 Sistemul de întreruperi**

Nucleul de **MOS6502** suportă 3 tipuri de întreruperi: **BRK**, **NMI** și **IRQ**.

Întreruperea **IRQ** are loc pe palierul negativ al pinului **IRQ** (vezi *add\_ref*). La fiecare ciclu **CPU** se verifică nivelul pinului de întrerupere și se activează întreruperea pe nivelul logic 0, dacă flag-ul *inhibit* (**I**) nu este setat. Întreruperea **IRQ** este cea mai puțin prioritară.

Întreruperea **NMI** este cauzată de detecția unui front negativ pe pinul **NMI**. Această întrerupere nu ține cont de valoare flag-ului *inhibit* și poate întrerupe procesorul în timpul execuției unei alte rutine de tratare a întreruperii.

Întreruperea de tip **BRK** (break) este cauzată de instrucțiunea cu același nume (**BRK**, vezi *add\_ref*). Procesorul își cauzează singur o întrerupere. La fel ca întreruperea **NMI** aceasta se execută chiar dacă flag-ul *inhibit* este setat.

**BRK** și **NMI** au aceeași prioritate și se pot întrerupe una pe cealaltă.

Vectorii de întreruperi și reset sunt situați la sfârșitul memoriei de program la adresele:

* $FFFA-$FFFB – vectorul **NMI**
* $FFFC-$FFFD – vectorul de reset
* $FFFE-$FFFF – vectorul **BRK** / **IRQ**

La aceste adrese se găsesc adresele de început a rutinelor de tratare a întreruperilor și, în cazul vectorului reset, adresa de început a programului. Întreruperile **BRK** și **IRQ** folosesc același vector. Pentru ca procesorul să poată determina care întrerupere a avut loc, la activarea întreruperii **BRK** se setează flagul **B** (*break*).

**§1.2 Unitatea de procesare video (PPU)**

Unitatea de procesare grafică folosită în consola **NES** este **Ricoh RP2C02** (**RP2C07**). Acesta un semnal video **CVSB** cu o rezoluție de 240 de linii a câte 256 de pixeli. Frecvența de cadre este de aproximativ 60fps pentru varianta **NTSC** și aproximativ 50fps pentru varianta **PAL**.

**§1.2.1 Descrierea pinout a procesorului grafic**



*figura 1.4 (RP2C02 / RP2C07 pinout)*

**R/W** – Semnal de la procesor care determină dacă următorul ciclu va fi de citire sau de scriere în memoria registrelor.

**[CPU D0:D7]** – Semnalele bus-ului de date prin care se comunică cu procesorul central.

**[CPU A0:A2]** – Semnale care selectează adresa la care va avea loc operațiunea de citire/scriere.

**CS** – Semnal generat de un decodor de adrese pentru spațiul $2000-$3FFF.

**[EXT0:EXT3]** – Pot fi folosiți ca mod alternativ de selecție a culorii de fundal (vezi *add\_ref*).

**CLK** – Intrare de clock (vezi *add\_ref*).

**INT** – Ieșire de întrerupere. Este conectată la pinul **NMI** al procesorului (vezi *add\_ref*).

**RST** – Pin de reset activ pe 0.

**WR** – Semnal de scriere în **V-RAM**.

**RD** – Semnal de citire din **V-RAM**.

**[PPU A8:A13]** – Biții cei mai semnificativi ai bus-ului de adrese.

**[PPU AD0:AD7]** – Bus-ul de date multiplexat cu cei mai puțin semnificativi biți ai bus-ului de adrese.

**ALE** *(address latch enable)* – Când este activ, menține cei mai puțin semnificativi 8 biți din bus-ul de adrese pentru a permite pinilor **[PPU AD0:AD7]** să fie folosiți pentru transferul de date.

**1.2.2 Spațiul de adresare**

Procesorul grafic are un spațiu de adrese pe 14 biți ($0000-$3FFF), care este separat de bus-ul de memorie al procesorului central. Acesta poate fi accesat direct de către **PPU** sau de către **CPU**, prin intermediul regiștrilor procesorului grafic (vezi *add\_ref*).

|  |  |  |  |
| --- | --- | --- | --- |
| Adresa de început | Adresa de sfârșit | Dimensiunea | Descriere |
| $0000 | $0FFF | 4KB | Tabela de șabloane 0. Este situată în prima jumătate a memoriei **CHR** de pe cartridge. |
| $1000 | $1FFF | 4KB | Tabela de șabloane 1. Este situată în a doua jumătate a memoriei **CHR** de pe cartridge. |
| $2000 | $23FF | 1KB | Tabela de afișare (Nametable) 0. Situată în **V-RAM**. |
| $2400 | $27FF | 1KB | Tabela de afișare (Nametable) 1. Situată în **V-RAM**. |
| $2800 | $2BFF | 1KB | Tabela de afișare (Nametable) 2. Situată în **V-RAM**. |
| $2C00 | $2FFF | 1KB | Tabela de afișare (Nametable) 3. Situată în **V-RAM**. |
| $3000 | $3EFF | 3840B | Pointează spre spațiul $2000-$2EFF. |
| $3F00 | $3F1F | 32B | Memorie **RAM** pentru indecșii culorilor (Palette **RAM**). |
| $3F20 | $3FFF | 224B | Pointează spre spațiul $3F00-$3F1F repetat de 7 ori. |

*tabelul 1.4 (spațiul de adresare PPU)*

Procesorul grafic are o memorie **V-RAM** de 2KB în care se stochează tabelele de afișare. Odată ce, evident, nu există suficient spațiu pentru 4 tabele, acestea sunt suprapuse (vezi *add\_ref*).

Pe lângă acest spațiu de adresare, modulul **PPU** mai are o memorie de 256 octeți pentru a stoca sprite-urile.

De menționat că există un silicon-bug care cauzează ca orice încercare de a scrie la adresa $3F10 să rezulte într-o scriere la adresa $3F00. Adresa $3F10 poate fi scrisă prin intermediul adreselor din spațiul $3F20-$3FFF care pointează spre aceeași zonă de memorie. Bug-ul nu se manifestă la citire.

**§1.2.3 Registrele PPU**

Unitatea de procesare grafică expune 8 registre de 8 biți în bus-ul de memorie al procesorului central. Acestea apar în spațiul de adresare al procesorului la adresele $2000-$2007, dar pot fi accesate și prin intermediul altor adrese (vezi *add\_ref*).

Procesorul grafic are un bus intern care este folosit la comunicarea cu procesorul. Acesta se comportă ca un latch din cauza proprietăților capacitive ale firelor din acest bus. Încercările de citire a regiștrilor **WO** sau a biților nefolosiți din regiștri rezultă în citirea unei valori ce a fost plasată pe bus la o operațiune anterioară.

Registrele **PPU** sunt descrise în *tabelul 1.5*.

|  |  |  |  |
| --- | --- | --- | --- |
| Denumire | Adresă | Acces | Descriere |
| PPUCTRL | $2000 | **WO** | Registru de control. |
| PPUMASK | $2001 | **WO** | Controlează ce elemente vor fi randate și modul de randare. |
| STATUS | $2002 | **RO** | Descrie starea curentă (din momentul citirii) a modulului **PPU**. |
| OAMADDR | $2003 | **R/W** | Adresa de citire/scriere a memoriei de sprite-uri. |
| OAMDATA | $2004 | **R/W** | Permite citirea/scrierea memoriei de sprite-uri. |
| PPUSCROLL | $2005 | **WO** | Poziția de derulare (scroll) orizontal și vertical. Sunt necesare două scrieri. |
| PPUADDR | $2006 | **WO** | Aici se scrie adresa din bus-ul **PPU** pe care procesorul central vrea să o acceseze. Sunt necesare două scrieri. |
| PPUDATA | $2007 | **R/W** | Folosit pentru transferul de date dintre **CPU** și bus-ul de memorie **PPU**. |

*tabelul 1.5 (registrele PPU)*

Registrul **PPUCTRL**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| V | P | H | B | S | I | N1 | N0 |

*figura 1.5 (registrul PPUCTRL)*

**[N1:N0]** – Adresa de început pentru tabela de afișare (0=$2000; 1=$2400; 2=$2800; 3=$2C00).

**I** – Setează cu cât va fi incrementată adresa din bus-ul **PPU** după fiecare operațiune de citire sau scriere realizată de **CPU**. (0=incrementare cu 1; 1=incrementare cu 32).

**S** – Selectează tabela de șabloane pentru sprite-urile de dimensiune 8x8. Pentru sprite-urile 8x16 acest bit este ignorat. (0=tabela 0; 1=tabela 1) (vezi *add\_ref*).

**B** – Selectează tabela de șabloane pentru fundal. (0=tabela 0; 1=tabela 1) (vezi *add\_ref*).

**H** – Setează dimensiunea de afișare a sprite-urilor. (0=8x8; 1=8x16) (vezi *add\_ref*).

**P** – Setează modul de funcționare a pinilor **EXT**. (0=citește indexul paletei de pe **EXT**; 1=scrie indexul paletei pe **EXT**).

**V** – Activează/dezactivează întreruperea **NMI** de la sfârșitul fiecărui cadru (la intrarea în *vertical blank*) (0=**NMI** dezactivat; 1=**NMI** activat).

Setarea bitului **V** atunci când modulul **PPU** se află în starea *vertical blank* va cauza generarea imediată a unei întreruperi **NMI**. Acest lucru poate cauza randarea greșită a următorului cadru.

Registrul **PPUMASK**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B | G | R | s | b | M | m | g |

*figura 1.6 (registrul PPUMASK)*

**g** – Dacă este setat, la ecran vor fi afișate doar tonuri de gri.

**m** – Setează dacă pe banda de 8 pixeli de la marginea ecranului va fi afișat fundalul. (0=ascunde; 1=afișează).

**M** – Setează dacă pe banda de 8 pixeli de la marginea ecranului vor fi afișate sprite-urile. (0=ascunde; 1=afișează).

**b** – Afișare fundal. Dacă nu este setat se va afișa negru în locul pixelilor de fundal.

**s** – Afișare sprite-uri.

**R** – Dacă este setat vor fi evidențiate culorile roșii.

**G** – Dacă este setat vor fi evidențiate culorile verzi.

**B** – Dacă este setat vor fi evidențiate culorile albastre.

Registrul **STATUS**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| V | S | O | - | - | - | - | - |

*figura 1.7 (registrul STATUS)*

**O** – (*sprite everflow*) Se setează dacă a se încearcă afișarea a mai mult de 8 sprite-uri pe o singură linie a ecranului.

**S** – Se activează după afișarea unui sprite ce are indexul 0 în tabela de șabloane.

**V** – Se setează în momentul în care procesorul grafic trece în starea *vertical blank*. Se resetează doar la citirea registrului.

Citirea acestui registru cauzează resetarea bitului **V** și a numărătoarelor de scrieri pentru registrele **PPUSCROLL** și **PPUADDR**.

Comportamentul bitului **O** este imprevizibil din cauza unor bug-uri. Poate raporta atât falsuri pozitive cât și negative.

Registrul **OAMADDR**:

În acest registru se scrie o adresă de 8 biți din memoria de sprite-uri care se dorește o fi accesată. Pentru copierea datelor despre sprite-uri, majoritatea jocurilor scriu în acest registru $00 și activează modulul **DMA** (vezi *add\_ref*).

Registrul **OAMDATA**:

Acest registru este folosit pentru interacționarea (scrierea/citirea) cu zona de memorie din memoria de sprite-uri selectată de registrul **OAMADDR**.

Registrul **PPUSCROLL**:

Acest registru este folosit pentru setarea poziției de derulare a imaginii de pe ecran (vezi *add\_ref*). Prima scriere în acest registru va seta derularea orizontală (pe axa X), iar a doua scriere va seta derularea verticală (pe axa Y).

Registrul **PPUADDR**:

În acest registru va fi scrisă adresa din bus-ul **PPU** pe care procesorul central vrea să o acceseze. Odată ce adresa țintă este pe 14 biți, iar registrul pe 8, vor fi nevoie de două scrieri pentru a introduce adresa completă. La prima scriere se vor copia cei mai semnificativi 6 biți ai adresei, iar la a doua ce mai puțin semnificativi 8. Accesarea zonei de memorie se realizează prin intermediul registrului **PPUDATA**. Fiecare operațiune cu registrul **PPUDATA** va cauza incrementarea adresei cu 1 sau cu 32, în dependență de valoarea bitului **I** din registrul **PPUCTRL**.

Registrul **PPUDATA**:

Prin intermediul acestui registru procesorul poate accesa bus-ul **PPU**. La fiecare operațiune realizată cu acest registru (scriere/citire) are loc incrementarea adresei din bus.

**§1.2.4 Modulul DMA**

Pentru copierea rapidă a datelor despre sprite-uri din bus-ul procesorului în memoria de sprite-uri a unității grafice este folosit un **DMA**. Acesta are un singur registru situat la adresa $4014 (**OAMDMA**) în spațiul de adresare al procesorului central.

Modulul se activează la scrierea unei valori de 8 biți în registrul **OAMDMA**. Valoarea scrisă reprezintă pagina din bus-ul de memorie al procesorului de unde se vor copia datele. Odată pornit, **DMA**-ul copie 256 de octeți (adică toată pagina) în memoria de sprite-uri a procesorului grafic începând cu adresa aflată în registrul **OAMADDR**.

Copierea durează 514 cicluri de ceas (256 de citire + 256 de scriere + 2 de inițializare și finalizare), timp în care procesorul nu execută instrucțiuni.

**§1.2.5 Paleta de culori**

Procesorul grafic **NES** poate genera 64 de culori, indexate de la 0 la 63.



*figura 1.8 (paleta de culori NES)*

Culorile generate pot avea anumite variații în dependență de versiunea consolei și de televizorul (monitorul) utilizat.

Pe ecran pot fi afișate, la un moment dat, maxim 25 de culori dintre cele afișate în *figura 1.8*. Indecșii culorilor ce se doresc a fi utilizate se scriu în memoria Palette **RAM** (vezi *add\_ref*) situată la adresele $3F00-$3F1F. Indecșii pentru culorile de fundal sunt situate la adresele $3F00-$3F0F, iar cei pentru culorile sprite-urilor la adresele $3F10-$3F1F.

Însemnătatea fiecărei adrese este descrisă în *tabelul 1.6*.

|  |  |
| --- | --- |
| Adrese | Însemnătate |
| $3F00 | Culoare universală de fundal. |
| $3F01-$3F03 | Paleta de fundal 0. |
| $3F04 | Neutilizată. Considerată drept transparent (se afișează $3F00). |
| $3F05-$3F07 | Paleta de fundal 1. |
| $3F08 | Neutilizată. Considerată drept transparent (se afișează $3F00). |
| $3F09-$3F0B | Paleta de fundal 2. |
| $3F0C | Neutilizată. Considerată drept transparent (se afișează $3F00). |
| $3F0D-$3F0F | Paleta de fundal 3. |
| $3F10 | Considerată a fi partea transparentă a sprite-ului. |
| $3F11-$3F13 | Paleta de sprite-uri 0. |
| $3F14 | Considerată a fi partea transparentă a sprite-ului. |
| $3F15-$3F17 | Paleta de sprite-uri 1. |
| $3F18 | Considerată a fi partea transparentă a sprite-ului. |
| $3F19-$3F1B | Paleta de sprite-uri 2. |
| $3F1C | Considerată a fi partea transparentă a sprite-ului. |
| $3F1D-$3F1F | Paleta de sprite-uri 3. |

*tabelul 1.6 (tabela indecșilor de culori)*

**§1.2.6 Tabelele de afișare**

Tabela de afișare este o zonă de memorie de 1024 de octeți folosită de **PPU** pentru randarea pixelilor de fundal. Procesorul grafic poate accesa 4 tabele virtuale de afișare. Fizic, memoria în care se află aceste tabele, este de doar 2048 de octeți, ceea ce înseamnă că pot exista maxim două tabele de afișare.

Cele 4 tabele pot fi accesate în bus-ul de adresare **PPU** în spațiul $2000-$2FFF (vezi *add\_ref*).

Tabelele de afișare virtuale sunt create prin suprapunerea adreselor. Nativ (fără *memory mapper*, vezi *add\_ref*), procesorul grafic suportă 4 tipuri de suprapunere: *horizontal mirroring*, *vertical mirroring*, *one-screen higher*, *one-screen lower*.



*figura 1.9 (tabelele de afișare)*



*figura 1.9a (horizontal mirroring) figura 1.9b (vertical mirroring)*



*figura 1.9c (one-screen higher) figura 1.9d (one-screen lower)*

Tabele de căutare conțin indecșii blocurilor ce trebuie afișate. Blocurile au dimensiunea de 8x8 pixeli și pe ecran încap 30 rânduri a câte 32 blocuri. Fiecare tabelă de afișare conține o matrice cu 30 rânduri și 32 coloane de indecși pe 8 biți. La sfârșitul tabelei de afișare se află un vector de 64 de octeți numit tabela de atribute. Fiecare octet din tabela de atribute conține 4 indecși (de 2 biți) ai paletei de fundal ce trebuie folosită pentru fiecare bloc (vezi *add\_ref*).

**§1.2.7 Derularea fundalului**

Derulare reprezintă mișcarea ecranului (viewport-ului) peste tabelele de afișare. Acest lucru este realizat pentru a afișa secvențe din hărțile de joc care au dimensiuni mai mari decât mărimea ecranului.

Poziția și direcția de derulare este controlată de procesor prin intermediul registrului **PPUSCROLL** (vezi ­*add\_ref*). Modificarea poziției de derulare necesită două scrieri în acest registru: prima scriere modifică derularea orizontală (axa X), iar a doua scrierea pe cea verticală (axa Y). Valoarea de o biți scrisă în registru reprezintă ofsetul în pixeli peste care trebuie realizată derularea.

Pentru derularea continuă, procesorul rescrie în permanență partea din tabelele de afișare care nu se vede pe ecran. Efectul este demonstrat în *figura 1.10*.

În partea de sus a imaginilor este reprezentat conținutul tabelelor de afișare. Barele cafenii indică ce porțiune din tabelele de afișare sunt vizibile pe ecran. În partea de jos a imaginilor este reprezentat ce vede utilizatorul pe ecran.

 

*figura 1.10a (derularea fundalului) figura 1.10b (derularea fundalului)*

 

*figura 1.10c (derularea fundalului) figura 1.10d (derularea fundalului)*

Din perspectiva utilizatorului harta este derulată continuu spre dreapta. În memoria consolei, totuși, are loc modificarea permanentă a datelor dintr-un „buffer” circular.

Datele din tabelele de afișare și pozițiile de derulare sunt modificate atunci când procesorul grafic este în starea *vertical blank*. Procesorul este informat de trecerea în *vertical blank* prin intermediul unei întreruperi **NMI** (vezi *add\_ref*).

**§1.2.8 Procesarea sprite-urilor**

Datele despre sprite-uri sunt stocate într-o memorie internă a procesorului grafic de 256 de octeți. Acolo încap maxim 64 de sprite-uri și fiecare ocupă câte 4 octeți.

Octetul 0 conține poziția sprite-ului pe axa Y (ofsetul în pixeli de la partea de sus a ecranului la partea de sus a sprite-ului). Odată ce viewport-ul are o înălțime de 240 de pixeli, setarea acestui atribut pe oricare valoare între 240 și 255 va rezulta în ascunderea sprite-ului.

Octetul 1 reprezintă indexul șablonului ce se va afișa. Pentru sprite-urile de 8x8 pixeli se va citi un singur șablon din memoria **CHR**, iar pentru sprite-urile 8x16 pixeli sunt citite două șabloane consecutive începând cu o adresă pară (bitul 0 este ignorat).

Octetul 2 conține atributele pentru randarea sprite-ului. Semnificația biților este descrisă în *figura 1.11*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FV | FH | P | - | - | - | C1 | C0 |

*figura 1.11 (Atributele sprite-ului)*

**[C1:C0]** – Indexul paletei care se va folosi la colorarea sprite-ului.

**P** – Determină dacă sprite-ul va fi randat în fața sau în spatele fundalului (0=în față; 1=în spate).

**FH** – (*flip horizontally*) Dacă bitul este setat, sprite-ul va fi afișat ca în oglindă.

**FV** – (*flip vertically*) Dacă bitul este setat, sprite-ul va fi afișat cu susul în jos.

Oglindirea sprite-urilor nu modifică poziția catetei de încadrarea a acestora.

Octetul 3 conține poziția sprite-ului pe axa X (ofsetul în pixeli de la partea din stânga a ecranului la partea din stânga a sprite-ului).

Atunci când două sau mai multe sprite-uri se suprapun, prioritate are sprite-ul cu adresa cea mai mică în memoria **OAM**.

În timpul randării, în momentul în care se afișează primul pixel opac al sprite-ului cu indexul 0, se setează bitul **S** din registrul **STATUS** (vezi *add\_ref*) pentru a raporta procesorului la ce etapă din randare se află cadrul curent.



*figura 1.12 (suprapunerea sprite-urilor)*

**§1.2.9 Randarea**

Unitatea de procesare grafică randează cadre cu rezoluția de 256x240 pixeli la o frecvență de cadre de 60fps sau 50fps, pentru modelele **NTSC** și respectiv **PAL**. La fiecare ciclu de ceas, se scoate la ieșire culoarea unui pixel. Durata *horizontal blank* este de 84 de cicli, iar durata *vertical blank* este de 21 de linii. Sincronizarea cadrelor este descrisă în *figura 1.13*.



*figura 1.13 (sincronizarea cadrelor)*

Randarea propriu-zisă a cadrului se realizează pe 4 straturi (vezi *figura 1.14*).



*figura 1.14 (straturile vizibile)*

Stratul din spate reprezintă o culoare solidă luată de la adresa $3F00 (vezi *add\_ref*). Stratul cu verde reprezintă fundalul jocului, iar cele cu roșu și albastru sprite-urile. Aceste 3 straturi pot avea pixeli transparenți. Culoarea de ieșire este calculată, la fiecare moment de timp cu două multiplexoare de prioritate și unul de transparență (vezi *figura 1.15*).



*figura 1.15 (randarea)*

Din cauza modului în care funcționează această logică de prioritate, se poate întâmpla ca unii pixeli din sprite-urile din stratul de suprafață să nu se vadă. Acest fenomen apare atunci când în spatele acestui sprite se află un alt sprite cu index mai mic în memoria **OAM** dar are prioritate mai mică decât fundalul (vezi *figura 1.16*).



*figura 1.16 (greșeală de prioritate)*

**§1.3 Unitatea de procesare audio (APU)**