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

**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

**CVBS** – Composite Video Baseband Signal

**GC** – Game Controller

**MB** – Memory Bus

**MM** – Memory Mapper

**NES** – Nintendo Entertainment System

**OPC** – Operation Code

**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**