**Proiect FIC  
(milestone 3)**

1. **Introducere**

**Temă proiect:** Calculator într-un calculator de buzunar

**Nume Echipă:** Robocop

**Membrii Echipei:**

1. Nicoleta Ștefănică (hardware developer)
2. Cosmina Țunea (hardware developer)
3. Viktor Ungur (software developer)
4. Eliana Șușară (tester)
5. Robert Vinițchi (project manager)

**Bibliografie:**

* Digital Design and Computer Architecture, Second Edition, David M. Harris, Sarah L. Harris

1. **Hardware**

**Formatul Adresei:**

1. Load and store

|  |  |  |  |
| --- | --- | --- | --- |
| [15:10] | [9:8] | [7:6] | [5:0] |
| opcode | reg | addr | Imm |
| <6> | <2> | <2> | <6> |

1. Branch

|  |  |
| --- | --- |
| [15:10] | [9:0] |
| opcode | address |
| <6> | <10> |

1. ALU Registrii

|  |  |  |  |
| --- | --- | --- | --- |
| [15:10] | [9:8] | [7:6] | [5:0] |
| opcode | src1 | src2 | unused |
| <6> | <2> | <2> | <6> |

1. ALU Immediate

|  |  |  |
| --- | --- | --- |
| [15:10] | [9:8] | [7:0] |
| opcode | src | imm |
| <6> | <2> | <8> |

**Codificare Opcode:**

|  |  |  |
| --- | --- | --- |
| Opcode în zecimal | Opcode în binar | Operație |
| 0 | 000000 | ADD reg. |
| 1 | 000001 | SUB reg. |
| 2 | 000010 | LSR reg. |
| 3 | 000011 | LSL reg. |
| 4 | 000100 | MOV reg. |
| 5 | 000101 | RSR reg. |
| 6 | 000110 | RSL reg. |
| 7 | 000111 | MUL reg. |
| 8 | 001000 | DIV reg. |
| 9 | 001001 | MOD reg. |
| 10 | 001010 | AND reg. |
| 11 | 001011 | OR reg. |
| 12 | 001100 | XOR reg. |
| 13 | 001101 | NOT reg. |
| 14 | 001110 | CMP reg. |
| 15 | 001111 | TST reg. |
| 16 | 010000 | INC reg. |
| 17 | 010001 | DEC reg. |
| 18 | 010010 | BRZ |
| 19 | 010011 | BRN |
| 20 | 010100 | BRC |
| 21 | 010101 | BRO |
| 22 | 010110 | BRA |
| 23 | 010111 | JMP |
| 24 | 011000 | RET |
| 25 | 011001 | LW |
| 26 | 011010 | SW |
| 27 | 011011 | POW reg |
| 28 | 011100 | FACT reg |
| 32 | 100000 | ADD imm. |
| 33 | 100001 | SUB imm. |
| 34 | 100010 | LSR imm. |
| 35 | 100011 | LSL imm. |
| 36 | 100100 | MOV imm. |
| 37 | 100101 | RSR imm. |
| 38 | 100110 | RSL imm. |
| 39 | 100111 | MUL imm. |
| 40 | 101000 | DIV imm. |
| 41 | 101001 | MOD imm. |
| 42 | 101010 | AND imm. |
| 43 | 101011 | OR imm. |
| 44 | 101100 | XOR imm. |
| 45 | 101101 | NOT imm. |
| 46 | 101110 | CMP imm. |
| 47 | 101111 | TST imm. |
| 48 | 110000 | INC imm. |
| 49 | 110001 | DEC imm. |
| 59 | 111011 | POW reg. |
| 60 | 111100 | FACT imm. |

**Operații suportate:**

|  |  |  |
| --- | --- | --- |
| Număr | ALU control | Operație |
| 0 | 00000 | ADD |
| 1 | 00001 | SUB |
| 2 | 00010 | LSR |
| 3 | 00011 | LSL |
| 4 | 00100 | MOV |
| 5 | 00101 | RSR |
| 6 | 00110 | RSL |
| 7 | 00111 | MUL |
| 8 | 01000 | DIV |
| 9 | 01001 | MOD |
| 10 | 01010 | AND |
| 11 | 01011 | OR |
| 12 | 01100 | XOR |
| 13 | 01101 | NOT |
| 14 | 01110 | CMP |
| 15 | 01111 | TST |
| 16 | 10000 | INC |
| 17 | 10001 | DEC |

**A diagram of a machine

Description automatically generated**

**Schemă hardware:**

**Explicarea schemei hardware**

**PC:**

* PC indică adresa din memorie a instrucțiunii ce urmează să fie executată, acesta este incrementat cu 2 deoarece se lucreaza cu date pe 16 biti,deci avem nevoie de 2 bytes pentru a stoca fiecare instructiune in memorie.
* Aceasta poate fi o adresa adusa din memorie prin instructiunea RET, o adresa de branch, JUMP sau instructiunea urmatoare
* Bazat pe opcode-ul instrucțiunii, Unitatea Centrală (CU) trimite semnale către multiplexoarele (mux-urile) sistemului și către Unitatea Liniară de Algebra (ALU).

**Operarea cu Registre:**

* În cazul instrucțiunilor ce implică lucrul cu registre, opcode-urile determină extragerea valorilor corespunzătoare din Register File.
* Valorile extrase pot fi supuse unor diverse operații, cum ar fi: ADD, LW, ADDI, și altele.

**Actualizarea Flag-urilor si a memoriei:**

* ALU actualizează Flags Register și Memoria (daca este cazul) în funcție de rezultatele obținute în urma execuției instrucțiunilor.

**Scrierea în Registrii:**

* În cazul instrucțiunilor care generează rezultate ce trebuie stocate, valorile obținute din ALU sau memorie sunt scrise înapoi în Register File, actualizând conținutul registrelor implicate.

**Descrierea operațiilor**

**LOAD**

Pentru a face load, prima dată trebuie să accesăm adresa la care se găsește valoarea. Registrul sursă este specificat prin biții 6 și 7, file register citind valoarea din acest registru și o trimite către Data Memory, care acceasează adresa primită și citește conținutul, pe care îl trimite pentru a fi scris în Register File. MemToReg selectează valoarea citită din memorie, iar aceasta este scrisă în registrul destinație specificat de biții 8:9.

**STORE**

În cazul în care instrucțiunea adusă din Instruction Memory este una de Store, biții 7:6 selectează registrul(X,Y sau AC) în care se găsește adresa la care vom stoca datele în memorie, iar biții 8:9 selectează registrul(X,Y sau AC) în care se găsesc datele ce vor fi stocate. Datele ce trebuie stocate în memorie sunt conectate la portul WD (Write Data) de la Memory Data, în timp ce adresa corespunzătoare este transmisă la portul A (Adresă).

**JUMP**

La instrucțiunea de jump, având în vedere că opcode-ul ocupă 6 biți, ne mai rămân 10 biți pentru a construi adresa la care se face saltul. Adresele sunt word-aligned, ceea ce inseamnă că ele sunt un multiplu de 2 și putem face o shiftare la stânga cu un bit pentru a obține o adresă validă. Pentru a ajunge la 16 biți din care este formată o adresă completă, concatenăm pe primele poziții cei 5 cei mai semnificativi biți ai PC + 2. Alegerea adresei de jump cu ajutorul multiplexorului este condiționată de semnalul SJMP.

**BRANCH**

În cazul unui branch, pentru a se verifica dacă se respectă condiția pusă, de exemplu BRZ, Control Unit activează bitul Z , acesta o să aibă valoarea 1, iar ceilalți biți vor fi setați la 0. Biții furnizați de către Unitatea de Control sunt apoi comparați cu cei din Flags Register pentru a verifica îndeplinirea condiției specificate. În cazul în care condiția este satisfăcută, semnalul "Branch Sgl" va fi setat la 1, determinând astfel Program Counter-ul să preia adresa instrucțiunii de branch.

**STACK POINTER**

Stack pointer-ul reține adresa de vârf a stivei, decrementându-se cu 2 de fiecare dată când se face jump, sau incrementându-se cu 2 când se face return. Pentru return trebuie citită valoarea PC din memorie cu ajutorul stack pointer-ului care precizează adresa la care se găsește. Cu ajutorul multiplexorului care alege una dintre valorile venite de la ALU sau SP, activând semnalul Wadress, este citit PC din memorie si este transmis mai departe către multiplexoare pentru a se alege noua valoare PC. Primul multiplexor întâlnit face alegerea între adresa obținută pentru branch și cea de return venită din memorie. Totodată, pentru a reține PC atunci când se face jump, o putem scrie în memorie, deoarece în Data Memory pot fi scrise atât date, cât și PC-ul datorită unui multiplexor condiționat de semnalul PC/DATA.

**N factorial**

A diagram of a block diagram

Description automatically generated

Counterul este inițial egal cu numărul pentru care se calculează factorial, făcându-se o comparare cu numărul în sine, iar dacă este egal, se face înmulțire pentru prima dată cu 1 și numărul. Mai apoi counterul se decrementează cu 1 și înmulțirea se face în continuare între valoarea counterului și rezultatul obținut anterior, până când primul ajunge la 0, moment în care rezultatul final este transmis mai departe pentru a fi scris în registrul destinație.

**Ridicarea la putere**

A diagram of a computer

Description automatically generated

Inițializăm counter-ul la valoarea 0 și îl incrementăm cu 1 în fiecare iterație. Counter-ul este comparat cu puterea specificată. Dacă este zero, valoarea 1 este salvată în registrul "reg". În caz contrar, registrul primește rezultatul înmulțirii dintre baza specificată și valoarea rezultatului anterior. Atunci când counter-ul atinge valoarea puterii, demultiplexorul selectează ieșirea 1, iar rezultatul este trimis la ieșire. În caz contrar, procesul este repetat pentru calcularea următoarei valori.

1. **Software**

Codul verilog este atașat acestui document.

1. **Testare**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. Test | Nume test | Pași parcurși în test | Rezultat așteptat | Rezultat primit | Pass/Fail |
| 1. | ALU\_ADD.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 15,-20,-8 | 15,-20,-8 | Pass |
| 2. | ALU\_AND.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0,-15,0,0,16 | 0,-15,0,0,16 | Pass |
| 3. | ALU\_CMP.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 100x,101x,001x,000x | 100x,101x,001x,000x | Pass |
| 4. | ALU\_DEC.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0,99,-33,-1 | 0,99,-33,-1 | Pass |
| 5. | ALU\_DIV.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0,9,1,0 | 0,9,1,0 | Pass |
| 6. | ALU\_INC.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 1,-31,101,6 | 1,-31,101,6 | Pass |
| 7. | ALU\_LSL.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 32,0,18 | 32,0,18 | Pass |
| 8. | ALU\_LSR.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 1,0,4 | 1,0,4 | Pass |
| 9. | ALU\_MOD.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 4,-13,-13,5,16 | 4,-13,-13,5,16 | Pass |
| 10. | ALU\_MOV.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 5,-3,1 | 5,-3,1 | Pass |
| 11. | ALU\_MUL.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 39,-160,9,176,-160 | 39,-160,9,176,-160 | Pass |
| 12. | ALU\_NOT.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 31,12,-1,-348 | 31,12,-1,-348 | Pass |
| 13. | ALU\_OR.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 9,27,-27,-1,-10 | 9,27,-27,-1,-10 | Pass |
| 14. | ALU\_RSL.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 1024,-13,18,16 | 1024,-13,18,16 | Pass |
| 15. | ALU\_RSR.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 1,-13,512,1024 | 1,-13,512,1024 | Pass |
| 16. | ALU\_SUB.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 5,-6,10,10 | 5,-6,10,10 | Pass |
| 17. | ALU\_TST.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0100,1000,0000,0100 | 0100,1000,0000,0100 | Pass |
| 18. | ALU\_XOR.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 14,-27,9,0,-26 | 14,-27,9,0,-26 | Pass |
| 19. | Tb\_adder.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 3,0,43981 | 3,0,43981 | Pass |
| 20. | Tb\_control\_unit.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | Multe rezultate->verificate->corecte | Multe rezultate->verificate->corecte | Pass |
| 21. | Tb\_data\_memory.v | Initializare input-uri cu diferite valori,generare clock si afisare output pentru a verifica rezultatul. | 42661,5757,65535 | 42661,5757,65535 | Pass |
| 22. | Tb\_decide\_branching.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 1,1,1,1,1,1,0 | 1,1,1,1,1,1,0 | Pass |
| 23. | Tb\_file\_register.v | Initializare input-uri cu diferite valori,generare clock,reset si afisare output pentru a verifica rezultatul. | 2-60078,3-6078,43981-43981 | 2-60078,3-6078,43981-43981 | Pass |
| 24. | Tb\_instruction\_memory.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 37123,37391,384 | 37123,37391,384 | Pass |
| 25. | Tb\_mux2.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 170,83 | 170,83 | Pass |
| 26. | Tb\_sign\_extend.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 65535,65408,117,65522 | 65535,65408,117,65522 | Pass |
| 27. | Tb\_shift\_left.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 254,0,234,228 | 254,0,234,228 | Pass |
| 28. | Tb\_sp\_adder.v | Initializare input-uri cu diferite valori,generare clock si afisare output pentru a verifica rezultatul. | 4117,4394 | 4117,4394 | Pass |
| 29. | Tb\_sp\_subtractor.v | Initializare input-uri cu diferite valori,generare clock si afisare output pentru a verifica rezultatul. | 4117,43936 | 4117,43936 | Pass |
| 30. | Tb\_clock\_counter.v | Generare clock,generare reset si afisare output pentru a verifica rezultatul. | Verificat in simulare->corect | Verificat in simulare->corect | Pass |
| 31. | Tb\_clock\_decrement.v | Initializare input cu diferite vaori,generare clock si reset si afisare output pentru a verifica rezultatul. | 1,0,65535,65534+varificat in simulare | 1,0,65535,65534+varificat in simulare | Pass |
| 32. | Tb\_demux2.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 43775,65423 | 43775,65423 | Pass |
| 33. | Tb\_comparator.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0,1 | 0,1 | Pass |
| 34. | Tb\_multiplier.v | Initializare input-uri cu diferite valori si afisare output pentru a verifica rezultatul. | 0,484,65315 | 0,484,65315 | Pass |