**Procesorul MIPS - ciclu unic**

**- 32 biți –**

*Raport de activitate*

*Student: Ureche Simona Elena*

*Grupa: 30224*

**Componente:**

* **IFetch **
* Componenta functionala.
* Memoria ROM este initializata cu, codificarea binara a instructiunilor care urmaresc rezolvarea urmatoarei cerinte:

15. Să se determine dacă valorile unui șir de N elemente sunt ordonate crescător. Șirul este stocat în memorie începând cu adresa A (A≥12). A și N se citesc de la adresele 0, respectiv 4. Rezultatul (1=true / 0=false) se va scrie la adresa 8.

C: A computer screen shot of a program

Description automatically generated

Cod masina: A screenshot of a computer code

Description automatically generated

Assembly: A screenshot of a computer screen

Description automatically generated

A diagram of a computer program

Description automatically generated with medium confidence

- "IFetch" reprezinta unitatea de preluare a instrucțiunilor pentru procesor. Am intampinat o problema in cadrul instructiunilor de salt, care dupa mai multe incercari a fost rezolvata.

* **MPG **
* Componenta functionala.
* "MPG", servește drept mecanism simplu de generare a unui semnal de activare bazat pe un semnal de ceas și pe starea unui buton:

A diagram of a circuit

Description automatically generated

* **SSD **
* Componenta functionala.
* "SSD" (Seven Segment Display) este utilizat pentru afișarea cifrelor pe un display cu șapte segmente:

A diagram of a computer

Description automatically generated

* **UC **
* Componenta functionala.
* Arhitectura UC implementează un decodificator care are intrarea Instr și semnalele de control ca ieșiri. În fiecare ramură "when" a structurii case, se actualizează doar semnalele de control care trebuie să fie diferite de zero pentru instrucțiunea respectivă.

A screenshot of a computer

Description automatically generated

* **ID**
* Componenta functionala.
* „ID” este responsabilă pentru decodificarea instrucțiunilor primite și pregătirea datelor necesare pentru execuția acestora. Aceasta primește instrucțiuni și semnale de control și generează semnale corespunzătoare pentru citirea și scrierea în registre, extinderea imediatelor și gestionarea altor operații specifice instrucțiunilor.
* A diagram of a computer

  Description automatically generated
* **EX**
* Componenta functionala.
* "EX” este responsabilă pentru execuția operațiilor aritmetice și logice specifice instrucțiunilor, precum și pentru calcularea adresei de branch în cazul instrucțiunilor de tip branch.

A diagram of a computer program

Description automatically generated with medium confidence

* **MEM**
* Componenta functionala.

"mem" este responsabilă pentru gestionarea accesului la memoria de date a sistemului. Aceasta primește datele de la componenta de execuție (EX), inclusiv rezultatele operațiilor ALU și datele citite din registre, și efectuează operații de scriere în memorie dacă este necesar.

A diagram of a computer circuit

Description automatically generated

* Am intampinat probleme din cauza faptului ca nu am conectat AluResIn la AluResOut; Acestea au fost rezolvate ulterior.
* CODUL A FOST TESTAT PE PLACUTA.
* **Instrucțiuni suplimentare**

1. **Tip I:**
2. **Instrucțiunea ori (**bitwise OR Immediate)

* SAU logic între un registru și o valoare imediată, memorează rezultatul în alt registru;
* RTL: $t <- $s | ZE(imm); PC <- PC + 4;
* Sintaxa: ori $t, $s, imm
* Format:

opcode rs rt imm

|  |  |  |  |
| --- | --- | --- | --- |
| 000101 | sssss | ttttt | iiiiiiiiiiiiiiii |

* Semnale : **ExtOp = ‘1’, ALUSrc = ‘1’, RegWrite = ‘1’ si ALUOp = “011”** (restul 0)

1. **Instrucțiunea bne (**Branch on Not Equal)

* Salt condiționat dacă două registre sunt diferite;
* RTL: if $s ≠ $t then PC <- (PC + 4) + (SE(offset) << 2) else PC <- PC + 4;
* Sintaxa: bne $s, $t, offset
* Format:

opcode rs rt imm

|  |  |  |  |
| --- | --- | --- | --- |
| 000110 | sssss | ttttt | oooooooooooooooo |

* Semnale : **ExtOp = ‘1’, Br\_ne = ‘1’, si ALUOp = “010”** (restul 0)

1. **Tip R:**
2. **Instrucțiunea xor(**bitwise eXclusive-OR)

* SAU-Exclusiv logic între două registre, memorează rezultatul în alt registru;
* RTL $d <- $s ^ $t; PC <- PC + 4;;
* Sintaxa: xor $d, $s, $t
* Format:

opcode rs rt rd sa function

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | sssss | ttttt | ddddd | 00000 | 100010 |

* Semnale : **RegDst = ‘1’, RegWrite = ‘1’, RegWrite = ‘1’** (restul 0)

1. **Instrucțiunea slt(**Set on Less Than (signed))

* Dacă $s < $t, $d este inițializat cu 1, altfel cu 0;
* RTL PC <- PC + 4; if $s < $t then $d <- 1 else $d <- 0;
* Sintaxa: slt $d, $s, $t
* Format:

opcode rs rt rd sa function

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | sssss | ttttt | ddddd | 00000 | 010000 |

* Semnale : **RegDst = ‘1’, RegWrite = ‘1’, RegWrite = ‘1’** (restul 0)

**Observații:**

(+) – în ALU are loc o operație de adunare

(–) – în ALU are loc o operație de scădere

(<<) – în ALU are loc o deplasare logică la stânga cu o poziție

(>>) – în ALU are loc o deplasare logică la dreapta cu o poziție

(and) – în ALU are loc o operație de și-logic

(or) – în ALU are loc o operație de sau-logic

(xor) – în ALU are loc o operație de sau-exclusiv

(cmp) – în ALU are loc o operație de comparare (folosită doar în cazul instrucțiunii *Set on Less Than*)