**Procesorul MIPS, ciclu unic**

**– versiune pe 32 biți –**

**1.Descrierea instructiunulor alese**

a. ANDI – AND Immediate

* **Descriere**: ȘI logic între un registru și o valoare imediată, cu rezultatul în alt registru
* **RTL:** $t <- $s & ZE(imm); PC <- PC + 4;
* **Sintaxă:** andi $t, $s, imm
* **Format:** 001100 sssss ttttt iiiiiiiiiiiiiiii

**OPCODE rs rt val imm zero extended**

b. ORI – bitwise OR Immediate

* **Descriere**: SAU logic între un registru și o valoare imediată, memorează rezultatul în alt registru
* **RTL:** $t <- $s | ZE(imm); PC <- PC + 4;
* **Sintaxă:** ori $t, $s, imm
* **Format:** 001101 sssss ttttt iiiiiiiiiiiiiiii

**OPCODE rs rt val imm zero extended**

c. XOR – bitwise eXclusive-OR

* **Descriere**: SAU-Exclusiv logic între două registre, memorează rezultatul în alt registru
* **RTL:** $d <- $s ^ $t; PC <- PC + 4;
* **Sintaxă:** xor $d, $s, $t
* **Format:** 000000 sssss ttttt ddddd 00000 000110

**OPCODE rs rt rd shamt funct**

d. SRA – Shift-Right Arithmetic

* **Descriere**: Deplasare aritmetică la dreapta cu h pozitii pentru un registru, rezultatul este memorat în altul. Se repetă valoarea bitului de semn.
* **RTL:** $d <- $t >> h; PC <- PC + 4;
* **Sintaxă:** sra $d, $t, h
* **Format:** 000000 00000 ttttt ddddd hhhhh 000111

**OPCODE rs rt rd shamt funct**

**2.Tabel pentru semnalele de control ale instructiunilor alese**

**A screenshot of a computer

AI-generated content may be incorrect.**

**3.Programul ales pentru a fi executat pe procesor**

Se cere să se determine **câte valori pozitive și impare** există într-un șir de N elemente întregi, stocat în memorie începând cu adresa 8. Valoarea N (numărul de elemente) se găsește la adresa de memorie 4. Rezultatul (numărul de valori pozitive și impare) trebuie scris în memorie la adresa 0.

**Pseudocodul rezolvarii:**

1. Citește N (numărul de elemente) din adresa 4 în registrul $t0.

2. Inițializează un contor ($t1) cu 0.

3. Setează adresa de început a șirului ($t2) cu 8.

4. Dacă N == 0, sari direct la sfârșit.

5. Pentru fiecare element din șir:

6. Încarcă elementul curent în $t3.

Află dacă este **pozitiv**:

7. Folosește SRA (Shift Right Arithmetic) pentru a obține semnul (bitul 31).

8. Dacă semnul este 1(negativ), **sari la 12**

Află dacă este **impar**:

10. Folosește ANDI cu 1, dacă rezultatul este 1, e impar.

11. Dacă este impar, **incrementează contorul** ($t1).

Mergi la următorul element:

12. Incrementează adresa ($t2 += 4)

13. Decrementează N ($t0 -= 1)

14. Sari la 4

15. La final, salvează rezultatul ($t1) în memoria de la adresa 0.

**Cod in asamblare:**

01: LW $t0, 4($zero) - $t0 = N

02: ADDI $t1, $zero, 0 - $t1 = 0 (counter)

03: ADDI $t2, $zero, 8 - $t2 = 8 (array base)

04: BEQ $t0, $zero, 10 - if $t0 == 0 -> jump to end

05: LW $t3, 0($t2) -$t3 = Mem[$t2]

06: SRA $t4, $t3, 31 - $t4 = $t3 >>a 31 (sign bit)

07: BEQ $t4, $zero, 1 - if $t4 == 0 (positive) -> skip

08: J 11 - jump to check odd

09: ANDI $t5, $t3, 1 - $t5 = $t3 & 1 (check odd)

10: BEQ $t5, $zero, 1 - if even -> skip next

11: ADDI $t1, $t1, 1 - $t1++

12: ADDI $t2, $t2, 4 - $t2 += 4 (next element)

13: ADDI $t0, $t0, -1 - $t0--

14: J 3 - jump back to 3

15: SW $t1, 0($zero) - store result in Mem[0]

**SCHEMA RTL**

A diagram of a circuit board

AI-generated content may be incorrect.

**4.Alte observatii**

* Codul a fost testat atat in modulul de simulare, cat si pe placa.
* Toate elementele sunt functionale.