# **ARHITECTURA CALCULATOARELOR**

# **PROCESORUL MIPS, CICLU UNIC**

# **VERSIUNE PE 16 BIȚI**

STUDENTĂ: ROTARIU LAURA-ALEXANDRA

GRUPA: 30224-2

## Descrierea elementelor funcționale și nefuncționale

Proiectul a fost testat pe placă și funcționează corespunzător. Se aprind led-urile de pe placă conform tabelului cu semnalele de control, sunt parcurse instrucțiunile în mod corect și sunt afișate datele așteptate în funcție de combinația dată de la switch-uri.

Programul MIPS parcurge un șir de 10 numere și pentru fiecare număr se face XOR cu un alt număr. Dacă rezultatul este zero, se face operația logică OR cu un alt număr și se adaugă la sumă, altfel se face AND cu un alt număr și apoi se adaugă rezultatul la sumă și după 10 iterații se salvează suma in memorie.

## Descrierea problemelor întâmpinate

Am întâmpinat probleme la implementarea instrucțiunii BEQ. Problema a fost rezolvată cu mici modificări în EX.vhd deoarece greșisem ceva minor la procesul unde primea flag-ul Zero valoarea corectă. De asemenea, în InstructionFetch.vhd, la scrierea instrucțiunii pentru BEQ, a fost necesar să pun registrul cu valoarea mai mare primul deoarece această instrucțiune presupune o scădere pentru a vedea dacă valorile din cele două registre primite sunt egale.

Nu am întâmpinat alte probleme.

1. Instrucțiunile

Instrucțiunile care pot fi executate de microprocesorul ciclu unic sunt de 3 tipuri:

* De tip R: am implementat instrucțiunile obligatorii ADD, SUB, SLL, SRL, AND și OR și am ales în plus instrucțiunile XOR și SLT:
* Instrucțiunea XOR realizează operația logică XOR pe valorile din două registre și pune rezultatul în alt registru. Rezultatul are valoarea logică 1 atunci când operanzii sunt diferiți și valoarea 0 când operanzii au aceeași valoare.

Scrierea în asamblare: XOR $d. $s, $t

RTL: $d <- $s ^ $t; PC <- PC + 1

Formatul binar: B”000\_sss\_ttt\_ddd\_0\_110”

* Instrucțiunea SLT compară două registre și dacă primul este mai mic decât al doilea, un alt registru primește valoarea 1, altfel primește valoarea 0.

Scrierea în asamblare: SLT $d. $s, $t

RTL: PC <- PC + 1; if $s < $t $d-< $1; else $d<-0

Formatul binar: B”000\_sss\_ttt\_ddd\_0\_111”

* De tip I: am implementat instrucțiunile obligatorii ADDI, LW, SW, BEQ și am ales în plus instrucțiunile ORI și ANDI:
* Instrucțiunea ORI realizează același lucru ca și instrucțiunea OR, adică operația logică OR, dar operația se face între registrul sursă și un imediat care va fi extins, iar rezultatul se pune în alt registru. Rezultatul are valoarea logică 1 atunci când cel puțin unul dintre operanzi are valoarea 1 și 0 când ambii operanzi sunt 0.

Scrierea în asamblare: ORI $t. $s, imm/offset

RTL: $t <- $s | SE(imm) ; PC <- PC + 1

Formatul binar: B”110\_sss\_ttt\_iiiiiii”

* Instrucțiunea ANDI realizează același lucru ca și instrucțiunea AND, adică operația logică AND, dar operația se face între registrul sursă și un imediat care va fi extins, iar rezultatul se pune în alt registru. Rezultatul are valoarea logică 1 atunci când ambii operanzi sunt 1 și valoarea 0 când cel puțin unul dintre operanzi are valoarea 0.

Scrierea în asamblare: ANDI $t. $s, imm/offset

RTL: $t <- $s & SE(imm) ; PC <- PC + 1

Formatul binar: B”101\_sss\_ttt\_iiiiiii”

* De tip J: am implementat instrucțiunea obligatorie jump(j).







