**Documentație MIPS32 pipeline**

**Student: Ardelean Mitică-Mario**

**Grupa: 30224**

**Problema aleasă:**

“Să se determine dacă un șir de N elemente este progresie aritmetică (diferența dintre oricare 2 elemente consecutive este constantă). Șirul este stocat în memorie începând cu adresa A (A≥12). A și N se citesc de la adresele 4, respectiv 8. Rezultatul (1=true / 0=false) se va scrie în memorie la adresa 0.”

**Regiștrii și instrucțiuni alese:**

* Regiștrii:
* $1 -> contorul buclei
* $2 -> adresa A
* $3 -> numărul de elemente N
* $4 -> rația progresiei aritmetice înainte de a intra în buclă
* $5 -> indexul locației de memorie
* $6 -> A(i)
* $7 -> A(i+1)
* $8 -> rația progresiei aritmetice în buclă
* $10 -> rezultatul final(1 dacă șirul e progresie aritmetică, 0 dacă nu e)
* Instrucțiuni alese:
* Add -> Adună două registre și memorează rezultatul în al treilea
* Sub -> Scade două registre și memorează rezultatul în al treilea
* Addi -> Adună un registru cu o valoare imediată și memorează rezultatul în alt registru
* Lw -> Un cuvânt din memorie este încărcat într-un registru
* Sw -> Valoarea unui registru este stocată în memorie la o anumită adresă
* Beg -> Salt condiționat dacă este egalitate între două registre
* Bne -> Salt condiționat dacă două registre sunt diferite
* J -> Salt la adresă absolută
* NoOP -> Nicio operație

**Elemente funcționale:**

* MPG
* iFetch
* UC
* ID
* EX
* MEM
* SSD
* test\_env
* Blocurile de registre iFetch/ID, ID/EX, EX/MEM și MEM/WB

**Elemente nefuncționale:** -

**Probleme întâmpinate:**

Inițial, atunci când se executau instrucțiunile BEQ, BNE sau J, acestea nu îmi săreau la porțiunea de cod care trebuie să se execute, din cauza hazardurilor.

După remedierea acesteia, nu am mai întâmpinat alte probleme.

**Testare:**

Codul acestui MIPS pipeline a fost testat atât pe placă(prin test\_env), cât și în simulator(tot prin test\_env). În ambele cazuri, codul a mers după așteptări, neavând probleme nici pe placă, nici în simulator.

**Instrucțiunile alese suplimentar:**

* XOR
* Descriere: SAU-Exclusiv între două registre, memorează rezultatul în alt registru
* Cod ASM: XOR $d, $s, $t
* Cod RTL: $d <- $s^$t; PC <- PC + 4;
* Cod mașină: B”000000\_sssss\_ttttt\_ddddd\_00000\_000011”
* Valori semnale control: RegDst – 1, RegWrite – 1, ALUOp – 00, celelalte - 0
* SLT
* Descriere: Dacă $s < $t, $d este inițializat cu 1, altfel cu 0
* Cod ASM: SLT $d, $s, $t
* Cod RTL: PC <- PC + 4; if $s < $t then $d <- 1 else $d <- 0
* Cod mașină: B”000000\_sssss\_ttttt\_ddddd\_00000\_110000”
* Valori semnale control: RegDst – 1, RegWrite – 1, ALUOp – 00, celelalte - 0
* BNE
* Descriere: Salt condiționat dacă două registre sunt diferite
* Cod ASM: BNE $s, $t, offset
* Cod RTL: if $s ≠ $t then PC <- (PC + 4) + (SE(offset) << 2) else PC <- PC + 4;
* Cod mașină: B”000010\_sssss\_ttttt\_oooooooooooooooo”
* Valori semnale control: ExtOp – 1, Br\_ne – 1, ALUOp – 10, celelalte - 0
* BGTZ
* Descriere: Salt condiționat dacă un registru este mai mare ca 0
* Cod ASM: BGTZ $s, offset
* Cod RTL: If $s > 0 then PC <- (PC + 4) + (SE(offset) << 2) else PC <- PC + 4;
* Cod mașină: B”000001\_sssss\_00000\_oooooooooooooooo”
* Valori semnale control: ExtOp – 1, Br\_gtz – 1, ALUOp – 10, celelalte – 0
* NoOP
* Descriere: Nicio operație
* Cod ASM: ADD $d, $s, $t
* Cod RTL: $d <- $s + $t; PC <- PC + 4;
* Cod mașină: B”000000\_00000\_00000\_00000\_00000\_100000
* Valori semnale control: RegDst – 1, RegWrite – 1, ALUOp - 00