**Descriere funcționalitate MIPS32**

1. Elemente funcționale

* Procesorul MIPS32 a fost implementat cu unitatea top-level test\_env și toate unitățile componente necesare:
  + IFetch
  + ID
  + EX
  + UC
  + MEM
  + SSD
  + MPG
* Instrucțiunile implementate de 3 tipuri: R(8 instrucțiuni), I(7 instrucțiuni), J(o instrucțiune)
* Programul de test a fost realizat în ROM și include funcționalitatea de numărare a numerelor pozitive și pare dintr-un vector. (Am folosit pentru exemplu un vector de 4 elemente, 1 element impar, 1 element negativ, 2 elemente pare, pentru a atinge toate instrucțiunile)

2. Elemente nefuncționale și probleme întâmpinate

* Problema la SLL:   
  Inițial am pus offsetul greșit la instrucțiunea SLL (am folosit 4 în loc de 2), ceea ce ducea la calcularea incorectă a adreselor în memorie.   
  ➔ Rezolvare: am corectat shiftarea pe 2 poziții.
* Problema la Ext\_Imm:   
  Inițial extindeam mereu bitul de semn, fără să țin cont de ExtOp.   
  ➔ Rezolvare: am corectat extinderea folosind formula corectă din slide-urile de la laborator
* Confuzie între AND și ANDI:   
  Am folosit AND în loc de ANDI, ceea ce făcea ca procesorul să folosească valoarea din registrul $1 în loc de imediatul 1. Cum în registru nu aveam nimic (implicit 0), compararea pentru a determina paritatea nu funcționa.   
  ➔ Rezolvare: am schimbat codificarea și controlul pentru a folosi ANDI și am adăugat instrucțiunea aceasta în UC și EX.
* Încurcare semnale Branch/Br\_Gtz:   
  Am confundat semnalul de control pentru BR cu Br\_Gtz în instrucțiunea:

PCSrc <= (Branch and zero) when Br\_GTZ = '0' else(Br\_GTZ and gtz);

Am scris în loc de Br\_GTZ tot Branch, ceea ce ducea la comportament greșit la BGTZ.   
➔ Rezolvare: Am verificat semnalele în timpul execuției BGTZ și am corectat.

* Timing issues la implementare:   
  Teoretic, programul funcționează (simularea dă aceleași rezultate cu cele calculate de mine în traseul de execuție).   
  La implementare apar mici probleme de timing, cauzate de curse critice (din ce spune internetul).   
  ➔ Momentan nu afectează corectitudinea funcțională, dar nu am știut cum să rezolv asta.

3. Testare

* Testare efectuată:
  + Simulator: am testat complet în simulare și funcționează bine.
  + Plăcuță: înainte de vacanță am reușit să testez parțial instrucțiunile 0, 1, 2, 3 (ADD, LW, ADD, ADDI), dar instrucțiunea 5 (branch) nu era implementată correct în UC și EX în acel moment.
* Stare actuală:
  + În simulare programul rulează corect, inclusiv salturile condiționate (BEQ, BGTZ, J).
  + Pe placă, nu știu cum funcționează.

**Instrucțiuni**

**1. SLLV (Shift Left Logical Variable)**

* Tip: R-type
* Scriere în asamblare:

SLLV $d, $t, $s

* Funcționalitate: Deplasare logică la stânga pentru un registru, cu un număr de poziții indicat de alt registru, iar rezultatul este memorat într-un al treilea registru
* RTL:

$d<-$t<<$s;

PC<-PC+4;

* Binar: 000000\_sssss\_ttttt\_ddddd\_00000\_101010
* Semnale de control:
  + RegDst = 1
  + ExtOp = 0
  + ALUSrc = 0
  + ALUOp = 000 (R)
  + MemtoReg = 0
  + RegWrite = 1
  + MemWrite = 0
  + Branch = 0
  + Br\_Gtz = 0
  + Jump = 0
  + AluCtrl = 0110
  + Func = 000100

**2. SLT (Set on Less Than)**

* Tip: R-type
* Scriere în asamblare:

SLT $d, $s, $t

* Funcționalitate: Dacă $s < $t, $d este inițializat cu 1, altfel cu 0
* RTL:

PC <- PC + 4;

if $s < $t then $d <- 1

else $d <- 0;

* Binar: 000000\_sssss\_ttttt\_ddddd\_00000\_000100
* Semnale de control:
  + RegDst = 1
  + ExtOp = 0
  + ALUSrc = 0
  + ALUOp = 000 (R)
  + MemtoReg = 0
  + RegWrite = 1
  + MemWrite = 0
  + Branch = 0
  + Br\_Gtz = 0
  + Jump = 0
  + AluCtrl = 0111
  + Func = 101010

**3. SLTI (Set on Less Than Immediate)**

* Tip: I-type
* Scriere în asamblare:

SLTI $t, $s, imm

* Funcționalitate: Dacă $s este mai mic decât un imediat, $t este inițializat cu 1, altfel cu 0
* RTL:

PC <- PC + 4;

if $s < SE(imm) then $t <-1

else $t <-0;

* Binar: 001010\_sssss\_ttttt\_iiiiiiiiiiiiiiii
* Semnale de control:
  + RegDst = 0
  + ExtOp = 1
  + ALUSrc = 1
  + ALUOp = 011 (<)
  + MemtoReg = 0
  + RegWrite = 1
  + MemWrite = 0
  + Branch = 0
  + Br\_Gtz = 0
  + Jump = 0
  + AluCtrl = 0110
  + Func = X

**4. BGTZ (Branch on Greater Than Zero)**

* Tip: I-type
* Scriere în asamblare:

BGTZ $s, offset

* Funcționalitate: Salt condiționat dacă un registru este mai mare ca 0
* RTL:

If $s > 0 then PC <- (PC + 4) + (SE(offset) << 2)

else PC <- PC + 4;

* Binar: 000111\_sssss\_00000\_oooooooooooooooo
* Semnale de control:
  + RegDst = X
  + ExtOp = 1
  + ALUSrc = 1
  + ALUOp = 010 (-)
  + MemtoReg = 0
  + RegWrite = 0
  + MemWrite = 0
  + Branch = 0
  + Br\_Gtz = 1
  + Jump = 0
  + AluCtrl = 0001
  + Func = X

**5. ANDI (AND Immediate)**

* Tip: I-type
* Scriere în asamblare:

ANDI $t, $s, imm

* Funcționalitate: ȘI logic între un registru și o valoare imediată, cu rezultatul în alt registru
* RTL:

$t <- $s & ZE(imm);

PC <- PC + 4;

* Binar: 001100\_sssss\_ttttt\_iiiiiiiiiiiiiiii
* Semnale de control:
  + RegDst = 0
  + ExtOp = 0
  + ALUSrc = 1
  + ALUOp = 100 (&)
  + MemtoReg = 0
  + RegWrite = 1
  + MemWrite = 0
  + Branch = 0
  + Br\_Gtz = 0
  + Jump = 0
  + AluCtrl = 0100
  + Func = X