Componenta Instruction Fetch – IFetch

Primește la intrare flag-ul Jump, PCSrc și adresa de jump și branch, iar în funcție de acestea pune la ieșire instructiunea de la adresa curentă și valoarea PC + 4.

- Dacă Jump = 1, atunci PC = Jump Address;
- Dacă Jump = 0, atunci:
    - Dacă PCSrc = 1, atunci PC = Branch Address;
    - Dacă PCSrc = 0, atunci PC = PC+4.

Componenta Instruction Decode – ID

Primește la intrare flag-urile RegWrite, RegDst, și ExtOp și primii 26 de biți mai puțini semnificativi din instrucțiunea curentă Instr(25:0) și valoarea de scris în registru WD.

- Dacă RegWrite = 1 atunci în registru se scrie valoarea de la WD, la adresa data de multiplexor.
- Daca RegDst = 1 atunci se scrie in memorie la adresa pentru registrul destinație.
- Daca RegDst = 0 atunci se scrie in memorie la adresa pentru registrul target.
- Dacă ExrOp = 1 atunci se iau primii 16 mai putini semnificativi biți din Instr și se face extindere cu semn la 32 de biți.
- Dacă ExrOp = 0 atunci se iau primii 16 mai putini semnificativi biți din Instr și se face extindere cu 0 la 32 de biți

Ieșirea RD1 semnifica valoarea din registrul sursa.

Ieșirea RD1 semnifica valoarea din registrul target.

Ieșirea func reprezintă campul function pentru instrucțiunile de tip R.

Ieșirea sa reprezintă numărul de biți cu care se face shift-area în cazul operațiilor de shiftare.

Componenta Unitatea de Control – UC

Primește la intrare primii 6 cei mai semnificativi biți din instrucțiune, iar in funție de codul instrucțiunii setează flag-urile: RegDst, ExtOp, ALUSrc, BranchOnEqual, BranchOnGreaterThanZero, BranchOnGreaterThanOrEqualToZero, Jump, MemWrite, MemtoReg, RegWrite și valoarea pentru ALUOp.

Componenta Instruction Execute – EX

Primește la intrare flag-ul ALUSrc, și valorile din registrul sursa, registrul target, valoarea imediată extensa cu semn, numărul de biți pentru shiftare, functia pentru operațiile de tip R, ALUOp pentru operațiile de alt tip, adresa PC + 4.

- Dacă ALUSrc = 1 atunci pe intrarea din ALU se pune valoarea imediată.
- Dacă ALUSrc = 0 atunci pe intrarea din ALU se pune valoarea din registrul target.

În funcție de funcția data pentru operațiile de tip R sau ALUOp, ALU execute operția dorită, dă pe ieșirea ALURes rezultatul, setează flag-urile: Zero, GreaterThanZero, GreaterOrEqualToZero și calculează adresa de branch.

Componenta Unitatea de Memorie – MEM

Primește la intrare flag-ul MemWrite, valoarea ALURes și valoarea registrului target.

- Dacă MemWrite = 1 permite stocarea valorii din regisrul target, la adresa din ALURes.

Are rolul de a stoca valorile pe 32 de biți, scrierea având loc doar pe frontal de ceas, iar citirea e asincrona.

La ieșire avem pe MemData valoarea din memorie stocată la adresa data de ALURes, iar pe ALURes valoarea primită inițial.

Componenta Write-Back – WB

Reprezintă multiplexorul cu selecția data de flag-ul MemtoReg:

• Dacă MemtoReg = 0, se trimite ALUResOut la componenta ID;

• Dacă MemtoReg = 1, se trimite MemData la componenta ID.

Toate componetele prezentate mai sus funcționează, iar intregul proiect a fost testat pe placă, dar și în simulator și funcționează.

Instrucțiunile alese:

Logical XOR

- Asamblare: xor $d, $s, $t
- RTL: $d ← $s ^ $t; PC ← PC + 4;
- Format binar: 000000 sssss ttttt ddddd 00000 100110
- Valorile semnalelor de control:
    - RegDst <= '1';
    - ExtOp <= '0';
    - ALUSrc <= '0';
    - BranchOnEqual <= '0';
    - BranchOnGreaterThanZero <= '0';
    - BranchOnGreaterThanOrEqualToZero <= '0';
    - Jump <= '0';
    - MemWrite <= '0';
    - MemtoReg <= '0';
    - RegWrite <= '1';

Realizează operația de xor între valorile din registrul sursă și registrul target și se incrementează program counter-ul. Rezultatul se pune în registrul destinație.

Shift-Right Arithmetic

- Asamblare: sra $d, $t, h
- RTL: $d ← $t >> h; PC ← PC + 4;
- Format binar: 000000 00000 ttttt ddddd hhhhh 000011
- Valorile semnalelor de control:
    - RegDst <= '1';
    - ExtOp <= '0';
    - ALUSrc <= '0';
    - BranchOnEqual <= '0';
    - BranchOnGreaterThanZero <= '0';
    - BranchOnGreaterThanOrEqualToZero <= '0';
    - Jump <= '0';
    - MemWrite <= '0';
    - MemtoReg <= '0';
    - RegWrite <= '1';

Realizează operația de shiftare pentru valoarea din registrul target și se incrementează program counter-ul. Rezultatul se pune în registrul destinație.

Branch on Greater Than Zero

- Asamblare: bgtz $s, offset
- RTL: if $s > 0 then PC ← PC + 4 + (SE(offset) << 2) else PC ← PC + 4;
- Format binar: 000111 sssss 00000 oooooooooooooooo
- Valorile semnalelor de control:
  - RegDst <= '1';
  - ExtOp <= '0';
  - ALUSrc <= '0';
  - BranchOnEqual <= '0';
  - BranchOnGreaterThanZero <= '1';
  - BranchOnGreaterThanOrEqualToZero <= '0';
  - Jump <= '0';
  - MemWrite <= '0';
  - MemtoReg <= '0';
  - RegWrite <= '1';

Verifică dacă valoarea din registrul sursa este strict mai mare ca 0, iar daca da aduna la program counter offset-ul, echivalent cu numărul de instrucțiuni peste care să sară. Se incrementează program counter-ul.

Branch on Greater than or Equal to Zero

- Asamblare: bgez $s, offset
- RTL: if $s >= 0 then PC ← PC + 4 + (SE(offset) << 2) else PC ← PC + 4;
- Format binar: 000111 sssss 00000 oooooooooooooooo
- Valorile semnalelor de control:
  - RegDst <= '0';
  - ExtOp <= '1';
  - ALUSrc <= '0';
  - BranchOnEqual <= '0';
  - BranchOnGreaterThanZero <= '0';
  - BranchOnGreaterThanOrEqualToZero <= '1';
  - Jump <= '0';
  - MemWrite <= '0';
  - MemtoReg <= '0';
  - RegWrite <= '0';

Verifică dacă valoarea din registrul sursa mai mare sau egală cu 0, iar daca da aduna la program counter offset-ul, echivalent cu numărul de instrucțiuni peste care să sară. Se incrementează program counter-ul.