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(REG_EX_MEM[37:6]);
    - Dacă PCSrc = 0, atunci PC = PC+4(REG_IF_ID[31:0]).

Componenta Instruction Decode – ID

Primește la intrare flag-urile RegWrite, RegDst(REG_ID_EX[13]), și ExtOp și primii 26 de biți mai puțini semnificativi din instrucțiunea curentă Instr(25:0)(REG_IF_ID[57:32]) 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 (REG_ID_EX[77:46]).

Ieșirea RD2 semnifica valoarea din registrul target (REG_ID_EX[109:78]).

Ieșirea func reprezintă campul function pentru instrucțiunile de tip R (REG_ID_EX[152:147]).

Ieșirea sa reprezintă numărul de biți cu care se face shift-area în cazul operațiilor de shiftare (REG_ID_EX[114:110]).

Componenta Unitatea de Control – UC

Primește la intrare primii 6 cei mai semnificativi biți din instrucțiune(REG_IF_ID[63:58]), 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(REG_ID_EX[12]), 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(REG_ID_EX[11:6]), ALU execute operția dorită, dă pe ieșirea ALURes(REG_EX_MEM[72:41]) rezultatul, setează flag-urile: Zero(REG_EX_MEM[3]), GreaterThanZero(REG_EX_MEM[4]), GreaterOrEqualToZero (REG_EX_MEM[5]) și calculează adresa de branch(REG_EX_MEM[37:36]).

Componenta Unitatea de Memorie – MEM

Primește la intrare flag-ul MemWrite(REG_EX_MEM[2]), valoarea ALURes (REG_EX_MEM[72:41]) și valoarea registrului target(REG_EX_MEM[104:73]).

- 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(REG_EX_MEM[33:2]) valoarea din memorie stocată la adresa data de ALURes, iar pe ALURes(REG_EX_MEM[65:34]) valoarea primită inițial.

Componenta Write-Back – WB

Reprezintă multiplexorul cu selecția data de flag-ul MemtoReg(REG_MEM_WB[0]):

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

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

Registrele intermediare:

REG_IF_ID[63:0]

REG_ID_EX[162:0]

REG_EX_MEM[110:0]

REG_MEM_WB[71:0]

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

Am întâmpinat o problemă în ceea ce privește scrierea in RegFile pe falling edge care nu functiona daca verificam ”if falling_edge(clk)”, dar merge daca verific ”if not rising_edge(clk)”, în rest nu am avut probleme.