Documentatie proiect – implementare MIPS 32 cu pipeline

1. Descriere generală

Proiectul constă în implementarea unui procesor MIPS cu pipeline pe 5 stadii (IF, ID, EX, MEM, WB), realizat în limbaj VHDL și sintetizat pe o placă FPGA Basys3. Scopul principal este simularea și verificarea funcționării corecte a unei arhitecturi de tip RISC, respectând principiile execuției pipelinate.

Procesorul suportă setul de instrucțiuni de bază MIPS, extins cu patru instrucțiuni suplimentare: bne (branch if not equal), xor, sra (shift right arithmetic) și ori (or immediate). Acestea au fost integrate prin modificări în unitatea de control și în ALU, asigurând compatibilitatea completă cu schema de pipeline.

Funcționarea procesorului este vizibilă în timp real prin intermediul afișajului cu 7 segmente de pe placa Basys3, iar interacțiunea utilizatorului este posibilă cu ajutorul butoanelor fizice ale plăcii, permițând o testare pas-cu-pas a instrucțiunilor.

2. Componente funcționale

Implementarea este structurata modular, utilizand urmatoarele fisiere:

2.1 Ifetch.vhd – Instruction Fetch

ROL: aduce instructiunea curenta din memoria de instructiuni

Elemente:

- Registru PC (Program Counter) care se incrementeaza cu 4 la fiecare ciclu de ceas
- Memorie ROM ce contine instructiunile programului

Ofera ca iesire instructiunea catre etapa de decodificare si noua valoare a PC-ului.

2.2 ID.vhd – Instruction Decode

ROL: decodifica instructiunea, extrage campurile importante (OpCode, registre, valori imediate)

Primeste instructiunea de 32 de biti de la IF si semnalele de control de la UC si trimite catre EX operanzii si semnalele relevante.

2.3 RegFile.vhd – Register File

- Contine 32 de registre de cate 32 de biti
- Suporta citirea a doua registre si scrierea intr-unul singur in acelasi ciclu de ceas

2.4 UC.vhd – Unit of Control

ROL: genereaza semnale de control in functie de opcode-ul instructiunii

Converteste instructiunea in semnale precum: RegWrite, ALUSrc, MemRead, Branch etc. Astfel permite selectarea si coordonarea fluxului de date

2.5 MUX.vhd – Multiplexor

Un multiplexor parametrizat pentru a putea fi folosit oriunde este nevoie a alege intre mai multe surse de date.

Exemple:

- Selectarea intre valoarea imediata si registru pentru ALU
- Selectarea intre valoarea din memorie si rezultatul ALU pentru scrierea in registru

2.6 EX.vhd – Execute

- Contine ALU care executa operatii aritmetice si logice
- Primeste datele de la ID si controleaza semnalul Zero pentru instructiunea de tip branch
- Efectueaza si calculul adresei de memorie pentru lw/sw

2.7 MEM.vhd – Memory Access

Simuleaza o memorie RAM pentru date.

Permite:

- Citirea: pentru lw, datele sunt trimise catre WB
- Scrierea: pentru sw, datele din registru sunt stocate in memorie

2.8 MPG.vhd – Debounce pentru butoane (Multiple Pulse Generator)

ROL: elimina efectul de bouncing la apasarea butoanelor fizice de pe placa. De asemenea este folosit pentru a inregistra tranzitii clare de la utilizator (ex: trecerea la urmatoarea instructiune)

2.9 View_instructions_on_basys.vhd – Afisare instructiuni

Converteste instructiunea MIPS intr-un format afisabil pe cele 4 afisoare cu 7 segmente.

Utilizeaza un switch pentru a selecta partea inferioara (biti 15 – 0) sau superioara (biti 31 – 16) ai instructiunii, astfel intregul cuvant de 32 de biti sa poata fi vizualizat in doua parti.

Usureaza depanarea si monotorizarea instrucituniilor direct pe placa.

2.10 Test_env.vhd – Mediu de test

Integreaza toate componentele de mai sus intr-o schema de ansamblu si permite simularea si verificarea functionarii corecte a instructiuniilor.

3. Probleme intampinate

- **Hazarduri structurale și de date**: Inițial, lipsea forwarding-ul, iar unele instrucțiuni produceau erori. A fost necesară inserarea de **NoOp-uri** sau implementarea forwarding-ului pentru testare corectă.

- **Branch delay slot**: Gestionarea precisă a salturilor condiționate a necesitat sincronizarea semnalelor între ID și IF.

- **Afișaj Basys3**: Inițial, semnalele de la SSD nu se actualizau corect. Corectarea formatului binar în zecimal a fost necesară.

- **Testare în Mediu Real**: S-au făcut ajustări în `test_env.vhd` pentru a configura corect push-button-ul ca semnal de ceas lent.

4. Semnale de control

Unitatea de control este responsabila pentru generarea semnalelor care ghideaza executia fiecarei instructiuni. Aceste semnale influenteaza selectarea datelor, activarea scrierilor si citirilor in registre/memorie si determinarea salturilor conditionate.

SEMNAL	DESCRIERE
RegWrite	Activeaza scrierea in registrul destinatie in etapa WB
MemRead	Permite citirea din memoria de date (util pentru instructiunea lw)
MemWrite	Permite scrierea in memoria de date (util pentru instructiunea sw)
MemtoReg	Selecteaza daca valoarea scrisa in registru vine din memorie sau din ALU

ALUSrc	Alege intre operandul 2 din registru si o valoare imediata pentru ALU
ALUOp	Codifica tipul de operatie aritmetica/logica
Branch	Activeaza logica de salt conditionat
Jump	Activeaza saltul neconditionat
Br_ne	Semnal specific pentru branch daca nu sunt egale. Se activeaza daca operand1 != operand2

5. Viitoare imbunatatiri

Implementarea hazardurilor de date cu forwarding automat

În prezent, hazardurile de date sunt tratate manual (inserare de nop-uri). Adăugarea unei unități de data forwarding ar permite rezolvarea automată a conflictelor între instrucțiuni consecutive, îmbunătățind performanța și acuratețea execuției.

Unitate de detecție și rezolvare a hazardurilor de control

Adăugarea unei logici dedicate pentru detecția și gestionarea hazardurilor cauzate de instrucțiunile de salt (beq, bne) ar permite evitarea execuției incorecte fără inserarea manuală de instrucțiuni nop.

Extinderea setului de instrucțiuni

Suportul pentru instrucțiuni adiționale precum lui, slti, andi, jal sau mult ar permite rularea unor programe mai complexe și apropierea de un subset complet MIPS.

Afișare mai detaliată pe 7 segmente sau LCD

Afișajul ar putea fi extins pentru a vizualiza codul instrucțiunii curente, conținutul registrelor sau adresele accesate, în loc doar de rezultatul final, oferind astfel un debug vizual mai eficient.

Interfață software pentru încărcarea programelor

Dezvoltarea unei interfețe grafice simple pentru generarea și încărcarea programelor în memoria procesorului ar îmbunătăți experiența de utilizare și ar elimina necesitatea modificării manuale a codului în VHDL.

6. RTL schematic generat in vivado