Laborator 07

Procesorul MIPS – versiune pe 16 biți, cu un tact de ceas pe instrucțiune

Unitatea de Instruction Execute EX / Unitatea de Memorie MEM / Unitatea Write-Back WB

0. Resurse minimale necesare !

Din laboratorul 6 (terminat acasă): Proiectul test_env, cu unitățile IF, ID și UC implementate, legate, cu mecanismul de testare, fără erori.

1. Obiective

Studiul, proiectarea, implementarea și testarea, pentru procesorul MIPS, pe 16 biți, un tact de ceas / instrucțiune (single-cycle):

- **Unitatea de execuție a instrucțiunii, EX**
- **Unitatea de memorie, MEM**
- **Unitatea de scriere a rezultatului, WB**
- **Restul conexiunilor necesare pentru terminarea implementării MIPS 16**

2. Descrierea procesorului MIPS, simplificat pe 16 biți - continuare

(!) Lectura obligatorie a cursurilor 3 și 4 pentru a înțelege activitățile din acest laborator. Cunoașterea în detaliu a noțiunilor din laboratoarele 4, 5 și 6!

Ciclul de execuție a unei instrucțiuni MIPS are următoarele etape / faze (curs 4):

- IF – Instruction Fetch
- ID/OF – Instruction Decode / Operand Fetch
- EX – Execute
- MEM – Memory
- WB – Write Back

Implementarea proprie a procesorului MIPS-16, pe care o continuați în acest laborator (și ideal o veți termina în timpul laboratorului curent), este partiționată în 5 componente (entități noi) pentru calea de date. Aceste componente se vor declara, instanția și lega în proiectul test_env, în entitatea principală (test_env, probabil…). Utilitatea acestei partiționări o veți înțelege în laboratoarele viitoare, când se va implementa versiunea pipeline a procesorului MIPS 16!

În acest laborator veți proiecta, descrie în VHDL, și implementa / testa unitatea de execuție a instrucțiunii Instruction Execute – EX, unitatea de memorie – MEM, unitatea de scriere a rezultatului Write Back – WB, precum și restul conexiunilor necesare pentru versiunea proprie a procesorului MIPS 16 cu ciclu unic de ceas pe instrucțiune.

Căile de date ale procesorului MIPS (versiunea 32 de biți) sunt prezentate mai jos, împreună cu unitatea de control (și semnalele aferente). Pentru a evita aglomerarea schemei, semnalele de control nu au mai fost legate explicit la destinații, dar se pot identifica ușor după nume.

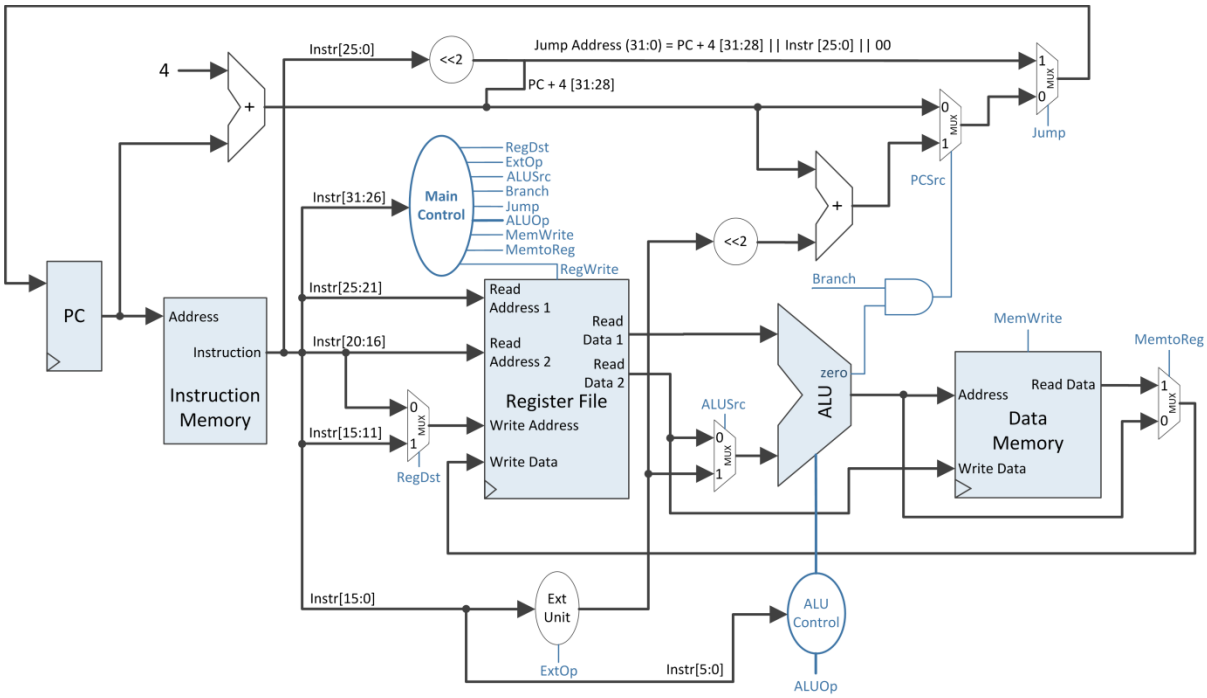

Figura 1: MIPS 32 cu ciclu unic de ceas

Mai jos sunt revizitate cele 3 formate de instrucțiuni pentru MIPS 16, descrise în laboratoarele anterioare:

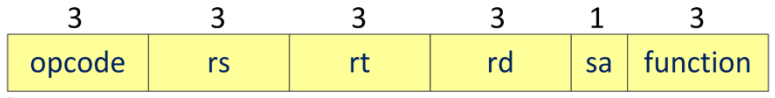

Figura 2: Instrucțiune de tip R

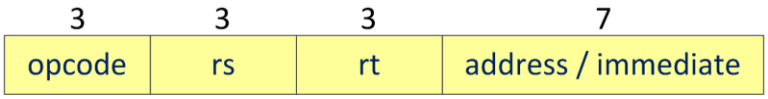

Figura 3: Instrucțiune de tip I

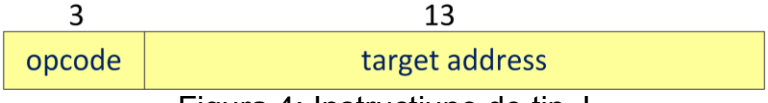

Figura 4: Instrucțiune de tip J

Unitatea de execuție EX conține următoarele elemente principale:

- Unitatea aritmetico-logică (ALU)
- Unitatea locală de control ALU
- Multiplexor
- Deplasare la stânga cu 2 și sumatorul pentru calculul adresei de salt (branch)

Consultați laboratoarele 2 și 4 pentru caracteristicile acestor elemente pentru procesorul MIPS 16.

Căile de date MIPS 32 pentru unitatea EX sunt prezentate în figura 5.

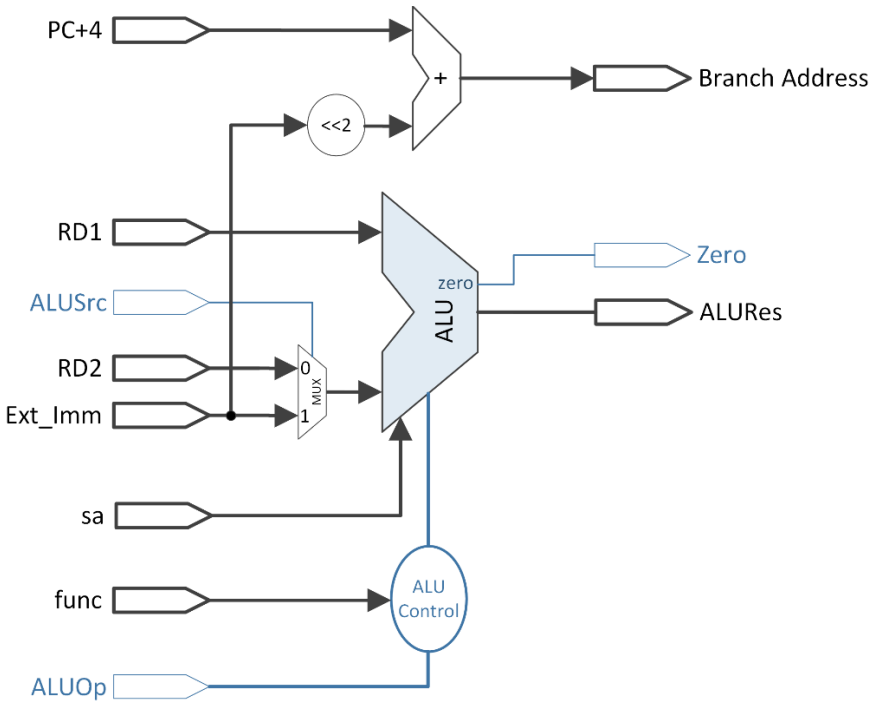

Figura 5: Unitatea ID, pentru MIPS 32, căile de date

Unitatea EX furnizează ca ieșire rezultatul ALU folosit pentru scrierea rezultatului instrucțiunilor artimetico-logice în blocul de regiștrii RF, sau ca adresă la Memoria de Date în cazul instrucțiunilor lw și sw. În plus, ALU furnizează pentru ieșire semnalul de stare Zero, care arată dacă rezultatul operației curente din ALU este egal cu 0 (Zero = 1 daca rezultatul este 0 sau 0 altfel). Pentru a ușura în viitor conversia procesorului MIPS 16 în implementarea pipeline, unitatea EX conține de asemenea calculul adresei de salt condiționat (branch).

Intrările unității EX sunt:

- Adresa următoarei instrucțiuni de executat, mod secvențial (PC + 4)

- Registru de la adresa rs, 32-biți, RD1
- Registru de la adresa rt, 32-biți, RD2
- Imediatul extins la 32-biți, Ext_Imm
- Câmpul func, pe 6 biți
- Câmpul sa, pe 5 biți
- Semnale de control:
  - ALUSrc – selecție între Read Data 2 și Ext_Imm pentru intrarea a doua din ALU
  - ALUOp – codul pentru operația ALU furnizat de unitatea principală de control UC

Ieșirile unității EX sunt:
- Adresa de salt Branch Address, 32-biți
- Rezultatul ALURes, 32-biți
- Semnalul Zero de 1 bit

Semnificația semnalelor de control este:
- ALUSrc = 0 → semnalul Read Data 2 merge pe intrarea a doua ALU
- ALUSrc = 1 → semnalul Ext_Imm merge pe intrarea a doua ALU
- ALUOp → este definit in unitatea principală de control UC în conformitate cu operațiile suportate (care se vor executa) de ALU.

Adresa de salt condiționat este calculată cu următoarea formulă:
Branch Address ← PC + 4 + S_Ext(Imm) << 2;

Semnalul Zero împreună cu semnalul de control Branch sunt folosite în exterior lui EX pentru calcula semnalul PCSrc, folosit în IF pentru a selecta valoarea care va merge spre contorul de program (PC+4 sau Branch Address).

Unitatea locală de control ALU definește operațiile de executat de către ALU prin codificarea lor în semnalul ALUCtrl. Pentru instrucțiuni de tip I, codificarea lui ALUCtrl depinde doar de valoarea semnalului ALUOp (setat din unitatea de control). Pentru instrucțiuni de tip R, ALUOp are aceeași valoare, iar valoarea lui ALUCtrl este definită pe baza valorii din intrarea func (câmpul function al instrucțiunii).

Unitatea de memorie MEM este simplă, conținând un singur element:
- Memoria de date (Data Memory)

Căile de date MIPS 32 pentru unitatea MEM sunt prezentate în figura 6.

Memoria de date este o memorie RAM cu citire asincronă (similar cu RF) și scriere sincronă pe front crescător de ceas, cu validare prin MemWrite. O descriere similară pentru RAM (! dar cu citire sincronă) a fost făcută în laboratorul 3.

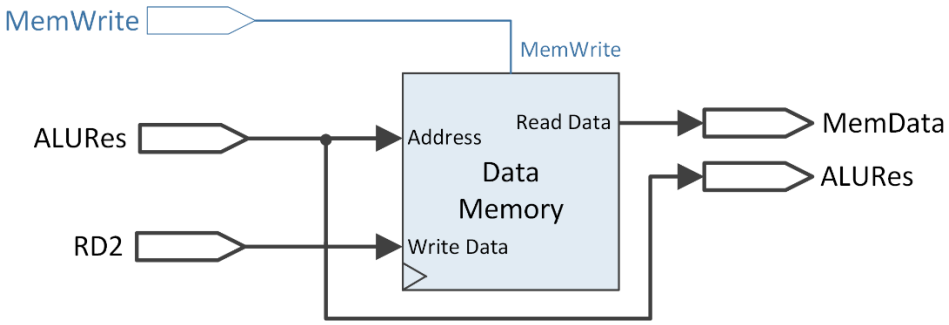

Figura 6: Unitatea de memorie MEM pentru MIPS 32, cu ciclu unic

Intrările unității MEM sunt:

- Semnalul de ceas (pentru scriere sincronă în memorie)
- Semnalul ALURes, 32 de biți – furnizează adresa pentru memorie (are efect doar pentru load sau store word)
- Semnalul RD2, 32 de biți – a doua ieșire din blocul de regiștrii RF (doar pentru instrucțiunea store word) care merge pe câmpul Write Data (portul de scriere) al memoriei
- Semnalul de control MemWrite

Ieșirile unității MEM sunt:

- MemData, 32 de biți, reprezintă datele de pe portul de citire Read Data al memoriei de date (doar pentru instrucțiunea load word)
- ALURes, 32 de biți, acest semnal reprezintă de asemenea rezultatul operațiilor aritmetico-logice, care trebuie stocat în blocul de regiștrii, deci trebuie furnizat la ieșirea unității MEM ca intrare pentru unitatea următoare, WB

Singurul semnal de control prezent în această unitate este MemWrite:

- MemWrite = 0 → Nu se scrie nimic în memoria de date
- MemWrite = 1 → Valoarea din semnalul RD2 este scrisă la adresa de memorie indicată de valoarea semnalului ALURes

Unitatea Write Back – WB este simplă, constând din ultimul multiplexor (dreapta) din Figura 1, iar restul componentelor neincluse până acum în unitățile descrise sunt: poarta SI pentru generarea semnalului PCSrc, și logica de calcul a adresei de salt Jump Address.

Semnalul de control pe multiplexorul din WB este MemtoReg, el selectând ce valoare se va scrie înapoi în blocul de regiștrii RF din unitatea ID:

- MemtoReg = 0 → valoare semnalului ALURes va ajunge pe intrarea Write Data a RF
- MemtoReg = 1 → valoare semnalului MemData va ajunge pe intrarea Write Data a RF

PCSrc ← Branch and Zero;

3. Activități practice

Citiți fiecare activitate în întregime, înainte să o începeți!

Resurse necesare (de avut la începerea laboratorului!):

- Rezultatele obținute din activitățile din laboratoarele 3, 4, 5 și 6!
- Proiectul test_env, cu unitățile IF, ID și UC implementate, legate, fără erori.

3.1. Proiectarea / implementarea unității EX

Ținând cont de descrierea unității EX din secțiunea 2, figura 5, descrieți o nouă componentă (entitate) pentru EX în proiectul test_env, ca parte a procesorului MIPS 16. Toate câmpurile de date sunt pe 16 biți!

Entitatea EX va conține elementele prezentate în figura 5, care nu se vor descrie cu entități suplimentare!

Folosiți o linie de cod VHDL (deplasarea cu 2 poziții nu mai e necesară… de ce?) pentru calculul adresei de salt Branch Address.

Folosiți un proces (cu case…) pentru descrierea ALU, asemănător cu descrierea făcută în laboratorul 2. Deplasamentul pe 1 bit, câmpul sa, este folosit doar pentru operații de deplasare logică sau aritmetică.

Folosiți un proces (tot cu case…) pentru implementarea unității locale de control ALU. Codificarea semnalului ALUCtrl depinde de cele 15 instrucțiuni suportate de varianta voastră a procesorului MIPS 16, și definește operațiile aritmetico-logice care trebuie suportate / cunoscute de către ALU. Indiciu: Pentru instrucțiunile de tip R, se poate face pe ramura ”when” corespunzătoare încă un case, după câmpul function...

Atenție! Aveți grijă la transformarea câmpurilor de date din figura 5 (MIPS 32) în versiunea proprie de implementare a MIPS pe 16 biți.

3.2. Proiectarea / implementarea unității MEM

Descrieți o nouă componentă MEM (entitate) pentru unitate de memorie, conform figurii 6. Folosiți implementarea RAM prezentată în laboratorul 3, și schimbați operația de citire astfel încât să devină asincronă. Descrieți totul doar comportamental în arhitectura entității MEM.

Atenție! Din cauza citirii asincrone, va rezulta o memorie RAM distribuită (vezi laboratorul 3), care nu se poate mapa pe module de block RAM ale plăcii. Pentru a evita un necesar de resurse mai mare decât suportă placa de dezvoltare, declarați memoria RAM cu mai puține locații (ex. 32, 64, 128, în loc de 2^{16}) iar la adresă folosiți biții cei mai puțin semnificativi din ALURes (câți sunt necesari).

3.3. Descrierea restului logicii necesare pentru procesor

În entitatea test_env declarați și instanțiați unitățile EX și ID. Conectați toate semnalele acestor unități în căile de date existente.

Atenție! Pentru memoria RAM, este necesar ca scrierea să fie controlată de la unul dintre butoane, la fel cum este controlată scrierea în registrul PC, respectiv în RF. Practic, în memoria RAM trebuie folosită ieșirea enable a aceluiași MPG folosit pentru activarea scrierii în PC (din unitatea IF) și în RF (din unitatea ID), ca o condiție suplimentară pe lângă testarea frontului de ceas și a lui MemWrite.

Adăugați multiplexorul pentru unitatea WB direct în arhitectura test_env.

Finalizați implementarea procesorului cu descrierea, în arhitectura test_env, a logicii de calcul a adresei de salt Jump Address (fără circuitul de deplasare cu 2 poziții… de ce?), și poarta SI pentru evaluarea semnalului PCSrc. Completați restul conexiunilor necesare în căile de date / control, conform Figurii 1 (Jump Address, Branch target address, write back la RF, etc.).

3.4. Aproape de destinație: testarea propriului procesor MIPS 16

În acest moment trebuie să aveți procesorul MIPS 16 implementat complet în proiectul test_env.

Pentru afișarea pe SSD a semnalelor relevante din căile de date, de la toate unitățile, trebuie să extindeți multiplexorul din laboratorul anterior:

- sw(7:5) = 000 → se afișează instrucțiunea pe SSD
- sw(7:5) = 001 → se afișează următoare valoare secvențială a PC (PC + 1), pe SSD
- sw(7:5) = 010 → se afișează RD1 pe SSD
- sw(7:5) = 011 → se afișează RD2 pe SSD
- sw(7:5) = 100 → se afișează Ext_Imm pe SSD
- sw(7:5) = 101 → se afișează ALURes pe SSD
- sw(7:5) = 110 → se afișează MemData pe SSD
- sw(7:5) = 111 → se afișează WD pe SSD

Pe ledurile plăcii se vor afișa semnalele de control ale UC. Există 8 semnale de 1 bit (cel puțin), plus ALUOp. Pe leduri se afișează toate semnalele de control definite pentru instrucțiunile implementate.

Dacă este necesar pentru depanarea procesorului MIPS (= trasarea atentă pas cu pas a programului scris în ROM) pe placa de dezvoltare, se pot afișa și alte semnale pe SSD / LED-uri: Branch Address, Jump Address, ALUCtrl, etc.

Acum, încărcați procesorul pe placa de dezvoltare, și executați pas cu pas (de la buton prin MPG) programul din memoria ROM. Verificați cu atenție la fiecare instrucțiune valorile prezente în semnalele relevante din căile de date, afișate pe SSD / LED-uri, pentru a stabili dacă instrucțiunea se execută corect. Verificați scrierea în memorie / RF urmărind valoarea locațiilor destinație din instrucțiunea curentă în instrucțiunile următoare unde ele apar ca operanzi sursă (le vedeți valorile din căile de date, pe SSD). **ETC…**

4. Referințe

- Cursurile 3 & 4 de la Arhitectura Calculatoarelor.
- MIPS® Architecture For Programmers, Volume I-A: Introduction to the MIPS32® Architecture, Document Number: MD00082, Revision 5.01, December 15, 2012
- MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set Manual, Revision 6.02
- MIPS32® Architecture for Programmers Volume IV-a: The MIPS16e™ Application-Specific Extension to the MIPS32™ Architecture, Revision 2.62.
    - Chapter 3: The MIPS16e™ Application-Specific Extension to the MIPS32® Architecture.