Laborator 10

Procesorul MIPS – versiune pe 16 biți, implementare de tip pipeline - evaluare

0. Resurse / cunoștințe minimale necesare !

0.1. Cunoștințe

Procesorul MIPS pipeline, versiunea pe 16 biți, implementată în laboratorul 9, prezentată în curs 9: schemă, principii de funcționare, modul de execuție a fiecărei instrucțiuni implementate pe procesor. Ce sunt hazardurile (explicate pe scurt în laboratorul curent)?

0.2. Resurse

Proiectul test_env conținând implementarea proprie a procesorului MIPS 16 pipeline.

1. Obiective

Studiul, proiectarea, implementarea și testarea:

- **Procesorului MIPS 16, versiunea pipeline, cu programul modificat prin inserarea de NoOp, fără hazarduri**

2. Transformarea procesorului MIPS 16 cu ciclu unic într-un procesor de tip pipeline – recapitulare din laboratorul 9

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

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.

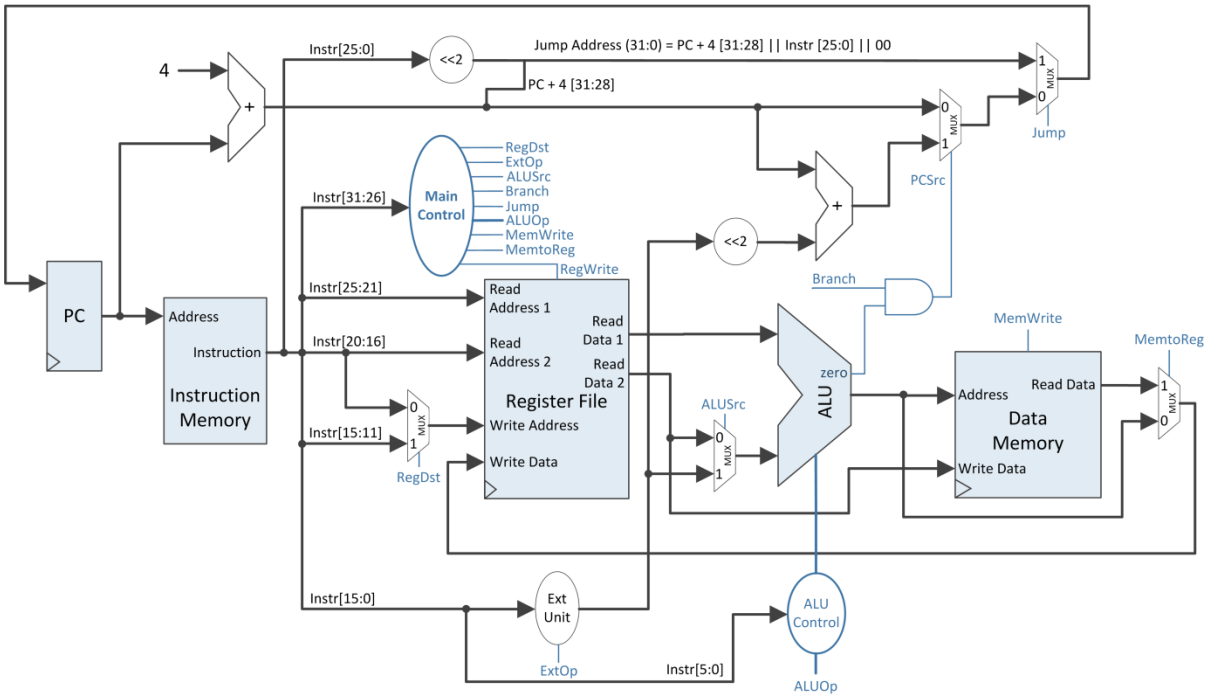

Figura 1: MIPS 32 cu ciclu unic de ceas

Pentru a reduce durata ciclului de ceas, soluția este secționarea căii critice cu elemente de stocare sincrone (regiștrii). Acești regiștri se interpun practic între unitățile procesorului MIPS 32, care coincid și cu fazele de execuție ale instrucțiunii: IF, ID, EX, MEM și WB. Astfel, se pot executa până la 5 instrucțiuni consecutive din program, fiecare aflându-se într-una din cele 5 faze de execuție, în funcție de ordinea de succesiune. Unitățile din varianta pipeline vor fi referite în continuare ca **etaje.**

Căile de date, și partea de control pentru procesorul MIPS 32 în varianta pipeline sunt prezentate în figura 5.

O diferență notabilă pe schemă, față de gruparea în unități pe care ați făcut-o deja pentru MIPS 16 cu ciclu unic, este că multiplexorul care selectează adresa registrului destinație este plasat în EX, nu în ID cum îl aveți în varianta proprie. Există 2 soluții posibile:

1. Îl lăsați în ID, în acest caz semnalul de control RegDst nu se mai transmite prin ID/EX, ci se ia direct de la UC. UC și ID se află în același etaj de pipeline.
2. Îl mutați în EX, modificând porturile de intrare / ieșire ale unităților ID și EX, și transmiteți RegDst conform schemei.

În rest, semnalul de control MemRead se va ignora, la fel ca la MIPS 16 cu ciclu unic, pentru a evita modificări suplimentare în căile de date.

Un alt semnal care nu apare explicit pe schema, dar a fost folosit și la MIPS 16 cu ciclu unic este semnalul de 1 bit sa. Acest semnal trebuie transmis prin registrul intermediar ID/EX către unitatea EX (unde e legat deja la ALU).

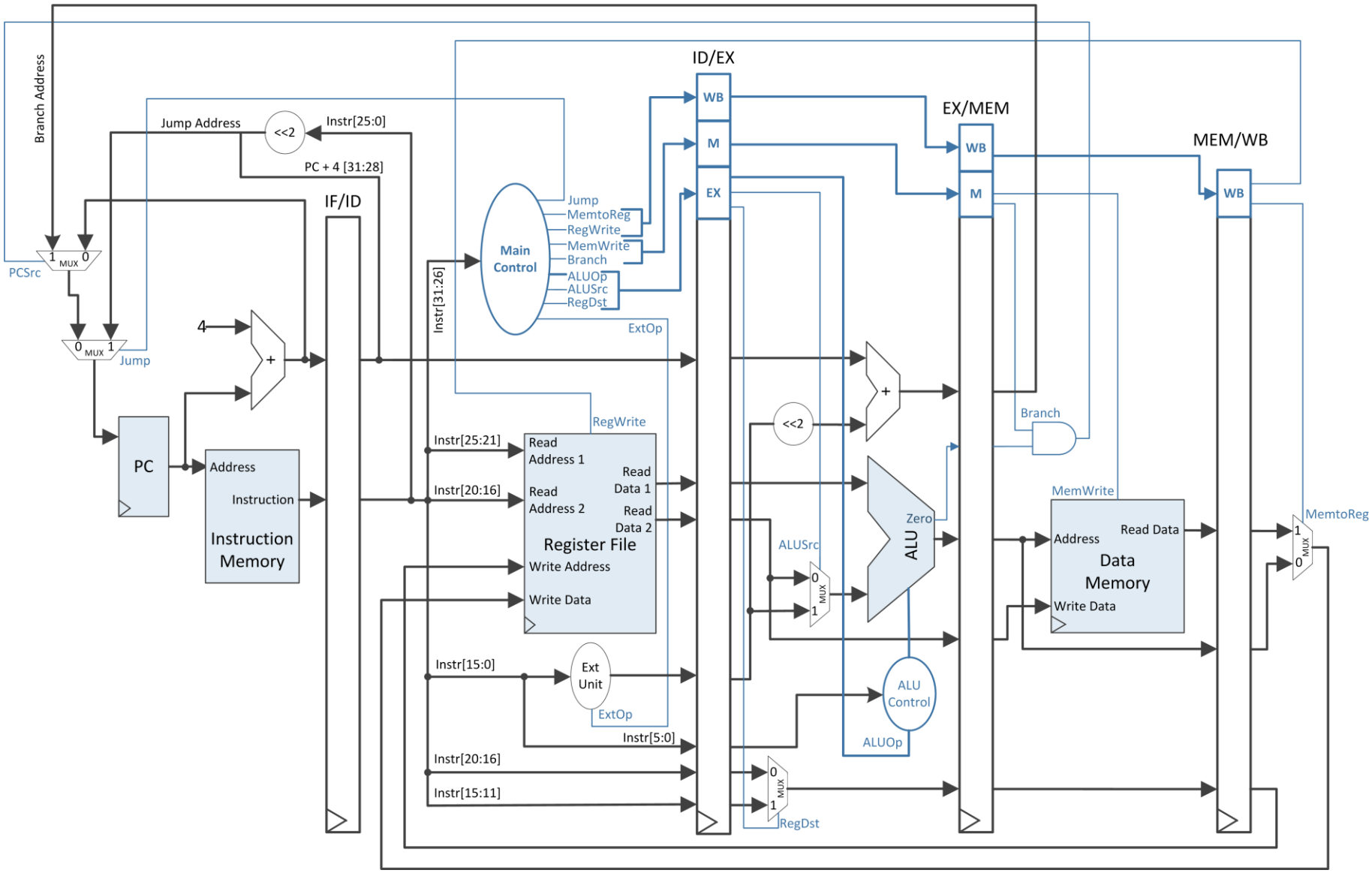

Figura 2: Procesorul MIPS 32, implementare de tip pipeline, obținut prin secționarea căilor de date MIP 32 cu ciclu unic, din figura 1

3. Hazarduri în procesorul MIPS

Hazardurile sunt situații în care o instrucțiune următoare nu poate fi executată în următoarea perioadă de ceas. Există trei tipuri de hazard, în funcție de cauza lor:

1. **Hazard structural (dependență de resurse)**
   - Două instrucțiuni încearcă să folosească simultan o resursă, în scopuri diferite → constrângeri de resurse
2. **Hazard de date (dependenta de date)**
   - Încercare de a folosi date înainte să fie disponibile
   - Pentru o instrucțiune care a ajuns în faza ID, operanzii sunt în curs de prelucrare în alte etaje de pipeline
3. **Hazard de control/comandă (dependenta de condiții, control)**
   - Decizia despre ramificarea unui program nu se știe înainte de evaluarea condițiilor salturilor și calcularea adresei de ramificare pentru PC
   - Trecerea prin pipeline a instrucțiunilor care influențează PC (ramificări, salturi)

Aceste hazarduri au fost prezentate pe larg la curs, a cărui lectură este obligatorie în prealabil. În continuare se discută cele 3 tipuri de hazarduri pe exemple concrete de cod. Soluțiile optime, hardware, au fost discutate la curs, se bazează pe tehnici de înaintare sau așteptare/anulare. Pentru implementarea MIPS 16 pipeline, din motive de timp, se va implementa soluția software, modificând programul astfel încât să nu mai existe hazarduri.

Modificare se va face inserând instrucțiuni de tip NoOp (NoOperation) între instrucțiunile unde apare hazard! Practic, e varianta software a tehnicilor de așteptare.

Instrucțiunea NoOp este una dintre instrucțiunile pe care le aveți deja, care în urma execuției nu ar trebui să schimbe nimic în procesor – în blocul de regiștri, memoria de date, iar PC se incrementează normal (ex. sll $0, $0, 0, sau add $0,$0, $0, etc.).

3.1. Hazard structural

Hazardul structural apare atunci când, pe același ciclu, instrucțiuni aflate în pipeline în etaje diferite încearcă să folosească, aceeași componentă din procesor.

De interes pentru implementarea de la laborator este hazardul structural care apare la utilizarea blocului de regiștri, pentru instrucțiuni situate la distanță de 3. În exemplul următor presupunem ca instr1 și instr2 nu au hazard cu restul instrucțiunilor.

Hazard **structural la RF**
add **$1**, $1, $2
instr1
instr2
add $3, **$1**, $4

Diagrama pipeline (pe fiecare clock indicăm în ce etaj se află fiecare instrucțiune):

Instr\Clk	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	…
add **$1**, $1, $2	IF	ID	EX	MEM	**WB**				
instr1		IF	ID	EX	MEM	WB			
instr2			IF	ID	EX	MEM	WB		
add $3, **$1**, $4				IF	**ID**	EX	MEM	WB	

Pe durata ciclului de ceas CC5 valoarea nouă a lui $1, generată de prima instrucțiune, se află în etajul WB, nefiind scrisă încă în RF. Astfel, în etajul ID, a patra instrucțiune va citi valoarea veche a lui $1, la tranziția CC5->CC6 propagându-se în etajul EX valoarea veche (incorectă).

Există două soluții posibile:

1. Recomandat: Se modifică blocul de regiștri astfel încât scrierea să se facă în mijlocul perioadei de ceas (se testează frontul descrescător clk=0 & clk'event sau falling_edge(clk)). În acest caz, citirea din RF fiind asincronă, în a doua jumătate a lui CC5 apare în ID noua valoare (corectă) a lui $1 și este propagată mai departe în etajul EX la tranziția CC5->CC6.
2. Se introduce o instrucțiune NoOp:

Fără hazard
add **$1**, $1, $2
instr1
instr2
NoOp
add $3, **$1**, $4

Atenție!: În explicarea hazardurilor următoare se consideră ca ați ales soluția 1, dacă ați mers pe varianta 2 țineți cont și unde este necesar introduceți câte un NoOp suplimentar, în plus față de cele inserate.

3.2. Hazard de date

Hazardul de date (tip Read After Write, sau Load data hazard) apare la acele instrucțiuni care folosesc ca operanzi sursă valori care sunt în curs de calculare de către instrucțiuni anterioare, nefiind actualizate încă.

(!)Pentru a lămuri exact unde apar aceste hazarduri, este utilă diagrama pipeline, precum și o bună stăpânire teoretică a execuției instrucțiunilor în pipeline (cum trec prin fiecare etaj, când se citesc operanzii, când se scrie destinația, vezi curs pentru detalii suplimentare).

Exemplul de mai jos conține majoritatea hazardurilor de date care ar trebui să le regăsiți în programul propriu din procesorul vostru MIPS 16 pipeline. Atenție, spre deosebire de programul vostru, exemplu de mai jos nu are un scop anume!

Instr. Nr.	Program
1	add $1, $2, $3
2	add $3, $1, $2
3	add $4, $1, $2
4	add $5, $3, $2
5	lw $3, 5($5)
6	add $4, $5, $3
7	sw $3, 6($5)
8	beq $3, $4, -6

Procesul de identificare a hazardurilor începe de la primele instrucțiuni, se rezolvă cu inserare de NoOp-uri, se continuă până la ultima instrucțiune. Dacă există îndoieli, diagrama de pipeline (pe foaie se pot folosi săgeți între etaje) poate ajuta la identificarea hazardurilor. Exemplu:

Instr\Clk	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
add $1, $2, $3	IF	ID	EX	MEM	**WB($1)**				
add $3, $1, $2		IF	**ID($1)**	EX	MEM	**WB($3)**			
add $4, $1, $2			IF	**ID($1)**	EX	MEM	WB		
add $5, $3, $2				IF	**ID($3)**	EX	MEM	**WB($5)**	
lw $3, 5($5)					IF	**ID($5)**	EX	MEM	**WB($3)**

Instr\Clk	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	…
add $5, $3, $2	IF	**ID($3)**	EX	MEM	**WB($5)**				
lw $3, 5($5)		IF	**ID($5)**	EX	MEM	**WB($3)**			
add $4, $5, $3			IF	**ID($3, $5)**	EX	MEM	**WB($4)**		
sw $3, 6($5)				IF	**ID($3)**	EX	MEM	WB	
beq $3, $4, -6					IF	**ID($4)**	EX	MEM	WB

Hazardurile se rezolvă pe rând, începând cu primul. Rezolvarea unui hazard între 2 instrucțiuni consecutive, rezolvă implicit eventuale hazarduri între prima instrucțiune și cea de la distanță +2. Exemplu: între instrucțiunea 1 și 2, și 1 si 3 există hazard de tip RAW (după $1). Se rezolvă mai întâi hazardul dintre 1 și 2. Conform diagramei instrucțiunea 2 ar trebui decalată (întârziată) cu 2 cicluri (să aibă faza ID pe ciclul CC5) => 2 NoOp. În acest moment toate instrucțiunile următoare se vor decala tot cu 2 cicluri, deci între 1 și 3 hazardul dispare de la sine.

Între instrucțiunea 2 și 4 există hazard RAW după $3, care se poate rezolva conform diagramei pipeline prin decalarea cu 1 ciclu. Rezultă un NoOp inserabil fie după 2, fie înainte de 4.

Pe aceeași logică se continuă cu restul hazardurilor. Rezultă următoarea rescriere a programului, fără hazarduri:

Instr. Nr.	Program
1	add $1, $2, $3
2	NoOp
3	NoOp
4	add $3, $1, $2
5	NoOp
6	add $4, $1, $2
7	add $5, $3, $2
8	NoOp
9	NoOp
10	lw $3, 5($5)
11	NoOp
12	NoOp
13	add $4, $5, $3
14	NoOp
15	sw $3, 6($5)
16	beq $3, $4, -6

3.3. Hazard de control

Hazardurile de control apar la instrucțiuni de salt, unde instrucțiunile care urmează după instrucțiunea de salt intră implicit în execuție.

În cazul instrucțiunilor de salt condiționat (beq, bne, etc), în forma de bază a procesorului MIPS pipeline, vor intra implicit în execuție următoarele 3 instrucțiuni. Modalitatea simplă, dar nu cea mai eficientă (vezi curs), este să se insereze 3 NoOp în program după instrucțiunea de salt condiționat.

În cazul instrucțiunilor de salt necondiționat j, conform descrierii pipeline din figura 2, aceste instrucțiuni se finalizează (execută saltul) când sunt în etajul ID. Rezultă că intră în execuție doar instrucțiunea imediat următoare. Soluția simplă este să se insereze un NoOp după instrucțiunea j. O soluție mai bună (!) este să se pună instrucțiunea care era înainte de j, care practic oricum trebuia să se execute, însă cu condiția ca această instrucțiune să nu fie tot de salt (j, beq, etc.)

4. Activități practice

Citiți fiecare activitate în întregime, înainte să o începeți!

Resurse necesare (de avut la începerea laboratorului!):

- Proiect Xilinx cu test_env, care să conțină implementarea proprie a procesorului MIPS 16 pipeline, completă.

4.0. Analiza programului și eliminarea hazardurilor (hârtie / instrument de scris)

Pe baza exemplului prezentat în secțiunea 3, identificați hazardurile din programul vostru. Inserați instrucțiunile NoOp acolo unde este necesar. Desenați diagrama pipeline pentru cel puțin 5 instrucțiuni succesive din programul vostru (sau tot, dacă nu identificați ușor hazardurile).

Atenție, în urma introducerii de NoOp-uri va trebui să ajustați adresele din instrucțiunile de salt din program.

Actualizați programul corectat (cod mașină) în memoria ROM de instrucțiuni.

4.1. Testarea și evaluarea procesorului MIPS 16 pipeline

În acest moment se face testarea pe placa de dezvoltare.

Există două variante, ca nivel de detaliu:

1. Dacă transformarea în pipeline a fost corectă, fără greșeli la mapare, etc., atunci este suficient să urmăriți rezultatele finale ale programului vostru (trebuie sa fie identice cu rezultatele din varianta cu ciclu unic).
2. Dacă rezultatele nu coincid, atunci este cazul să faceți o trasare pas cu pas a programului. Folosiți același mecanism de afișare a câmpurilor intermediare din procesor ca la varianta cu ciclu unic (mux-ul pe switch-uri pentru afișorul SSD). Țineți cont că, spre deosebire de MIPS 16 cu ciclu unic, acum veți avea 5 instrucțiuni consecutive în execuție (unele NoOp), deci câmpurile vizualizate nu vor fi de la aceeași instrucțiune. Dacă este nevoie afișați alte câmpuri pe SSD, pentru depanare.

   În funcție de câmpurile din procesor (semnale de control, Jump Address, Branch Address, Read data 1,2, ALUResult, etc) care nu au valorile corecte la instrucțiunile urmărite, reveniți la descrierea VHDL pentru identifica și corecta erorile.

Procesorul se prezinta în acest laborator (nota maximă se obține dacă este funcțional corect, cu răspunsuri corecte la întrebări), Nota obținută în laboratorul 10 se va duplica pe laboratoarele 9 și 10!

4. Referințe

- Cursurile 3 & 4, și cursul 9 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.