

Arhitectura Calculatoarelor

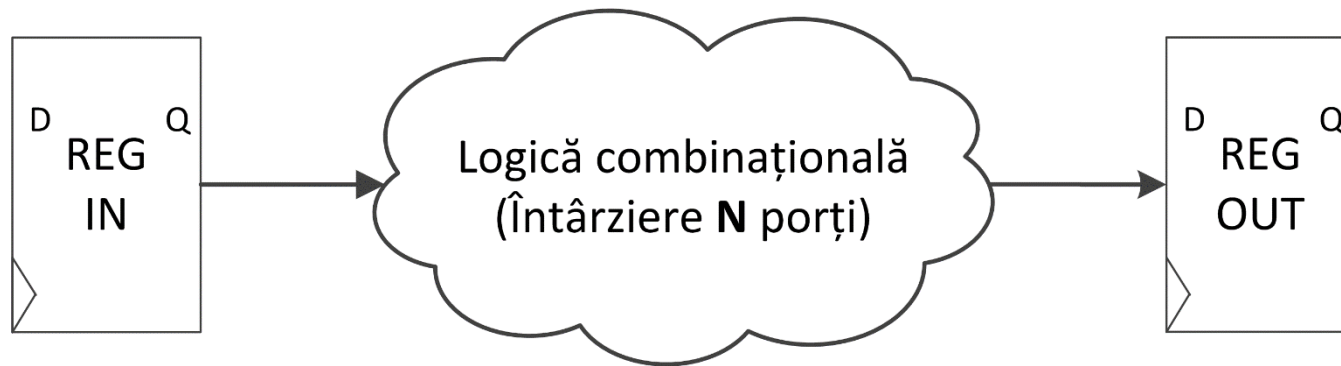
Curs 8: Proiectarea MIPS pipeline

E-mail: florin.oniga@cs.utcluj.ro

Web: <http://users.utcluj.ro/~onigaf>, secțiunea Teaching/AC

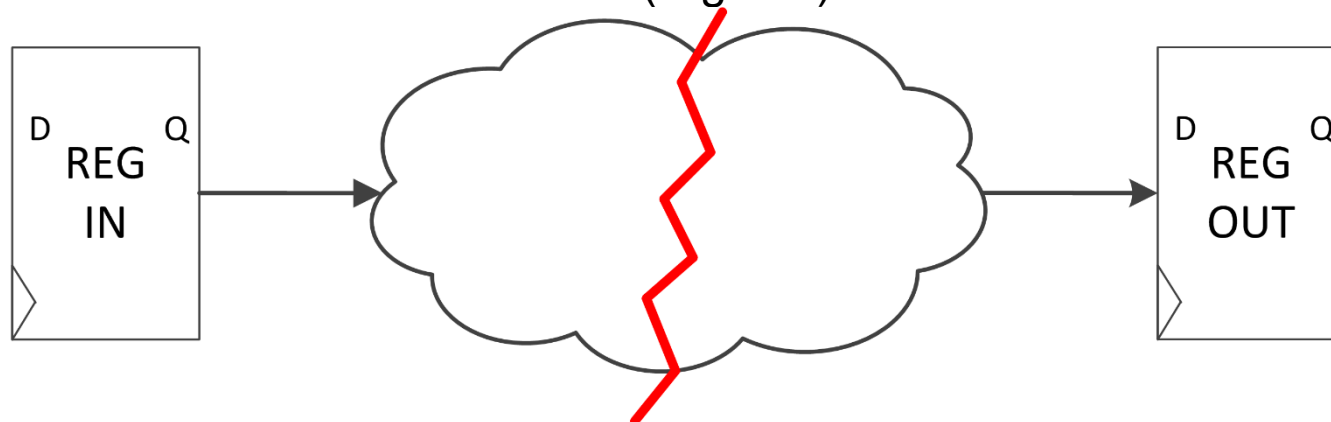
Problema căilor critice lungi

Calea critică limitează frecvența maximă de ceas la care poate funcționa corect circuitul!



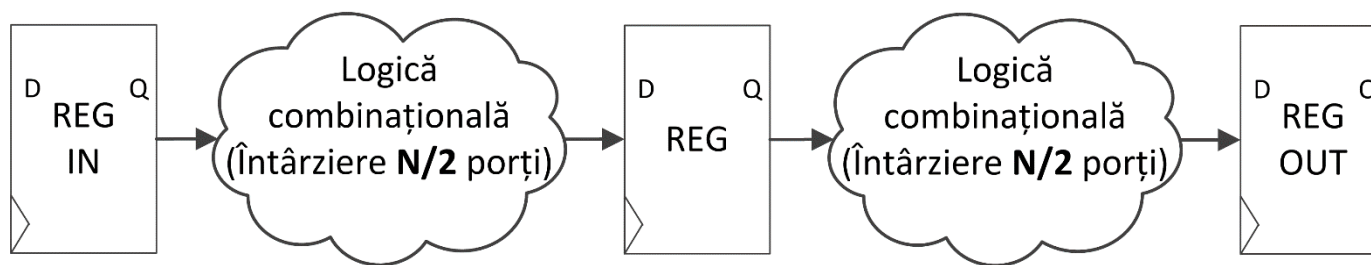
- Perioada de ceas trebuie să acopere minim întârzierea echivalentă de N porți (în realitate plus timpi setup/hold pentru registre)
- Durata pentru furnizarea unui rezultat nou $\approx N \times (\text{întârziere/poartă}) = D$
- Productivitatea este (nr. rezultate / interval de timp) $\text{Prod}_{\text{init}} = 1/D$

Soluția pentru reducerea perioadei de ceas (\Leftrightarrow frecvență mai mare) este secționarea căii critice cu elemente de stocare sincrone (registre)



Secționarea căii critice

Pipeline (linie de asamblare): un set de elemente / etaje de prelucrare a datelor, separate prin registre și conectate în serie; ieșirea unui etaj servește ca intrare pentru următorul.



Pipeline cu 2 etaje

- Perioada de ceas trebuie să acopere minim întârzierea echivalentă de $N/2$ porți
- Durata pentru furnizarea unui rezultat nou $\approx (N/2) \times (\text{întârziere/poartă}) = D/2$
- Productivitatea potențială este dublă: $2/D = 2 \cdot (1/D) = 2 \cdot \text{Prod}_{\text{init}}$



Pipeline cu 3 etaje

- Perioada de ceas trebuie să acopere minim întârzierea echivalentă de $N/3$ porți
- Durata pentru furnizarea unui rezultat nou $\approx (N/3) \times (\text{întârziere/poartă}) = D/3$
- Productivitatea potențială este triplă: $3/D = 3 \cdot (1/D) = 3 \cdot \text{Prod}_{\text{init}}$

Secționarea căii critice – performanță vs cost

Mărind numărul de etaje crește la nesfârșit productivitatea?

- **Nu**, deoarece registrele intermediare introduc întârzieri suplimentare, cumulative.

Dacă durata fără pipeline este D , iar durata introdusă de un registru intermediar este D_{reg} , atunci productivitatea pentru $k / k+1$ etaje este

$$Prod(k) = \frac{1}{\frac{D}{k} + (k-1)D_{reg}} \quad Prod(k+1) = \frac{1}{\frac{D}{k+1} + kD_{reg}}$$

Productivitate începe să scadă $Prod(k+1) < Prod(k)$ atunci când

$$D_{reg} > \frac{D}{k} - \frac{D}{k+1}$$

În realitate apar și probleme de cost suplimentar, se realizează un echilibru între cost și performanță pentru a determina nivelul optim de etaje K_{opt} :

- $K < K_{opt}$ – proiect “underpipelined”
- $K > K_{opt}$ – proiect “overpipelined”

Procesoare pipeline – Aspecte generale

- Pipeline-urile permit exploatarea paralelismului la nivel de instrucțiuni (**ILP**), efectuând în paralel, în etaje diferite, operații pentru instrucțiuni diferite
- „Pipelining” nu îmbunătățește **durata de execuție** a instrucțiunilor individuale; crește **productivitatea** => se reduce durata de execuție a programului în ansamblu
- Se reduce durata perioadei semnalului de ceas
- **Caz Ideal:** Durata de execuție fără pipeline/Numărul etajelor de pipeline
- **În realitate:**
 - Etajele nu sunt perfect balansate, timpi pierduți
 - Viteza „Pipeline”-ului este limitată de etajul cel mai lent
 - Apar întârzieri în plus datorită hazardurilor (așteptare – stall)
 - Timpi setup/hold pentru registrele intermediare
- Pipelining reduce durata de execuție medie (aparentă!) per instrucțiune.
- În general, pipelining-ul nu este vizibil pentru programator (eventual pt. compilator).

Procesoare pipeline – Aspecte generale

Proiectantul pipeline-ului trebuie să

- Balanseze duratele de execuție în diferite etaje.
- Să reducă efectul hazardurilor

Caracteristici ISA MIPS care favorizează pipelining-ul

- Toate instrucțiunile au aceeași lungime
- Număr redus de formate de instrucțiuni, locații fixe pentru registre în instrucțiuni
- Operanți de memorie apar numai în operațiile specifice: load și store

Ce îngreunează pipelining-ul?

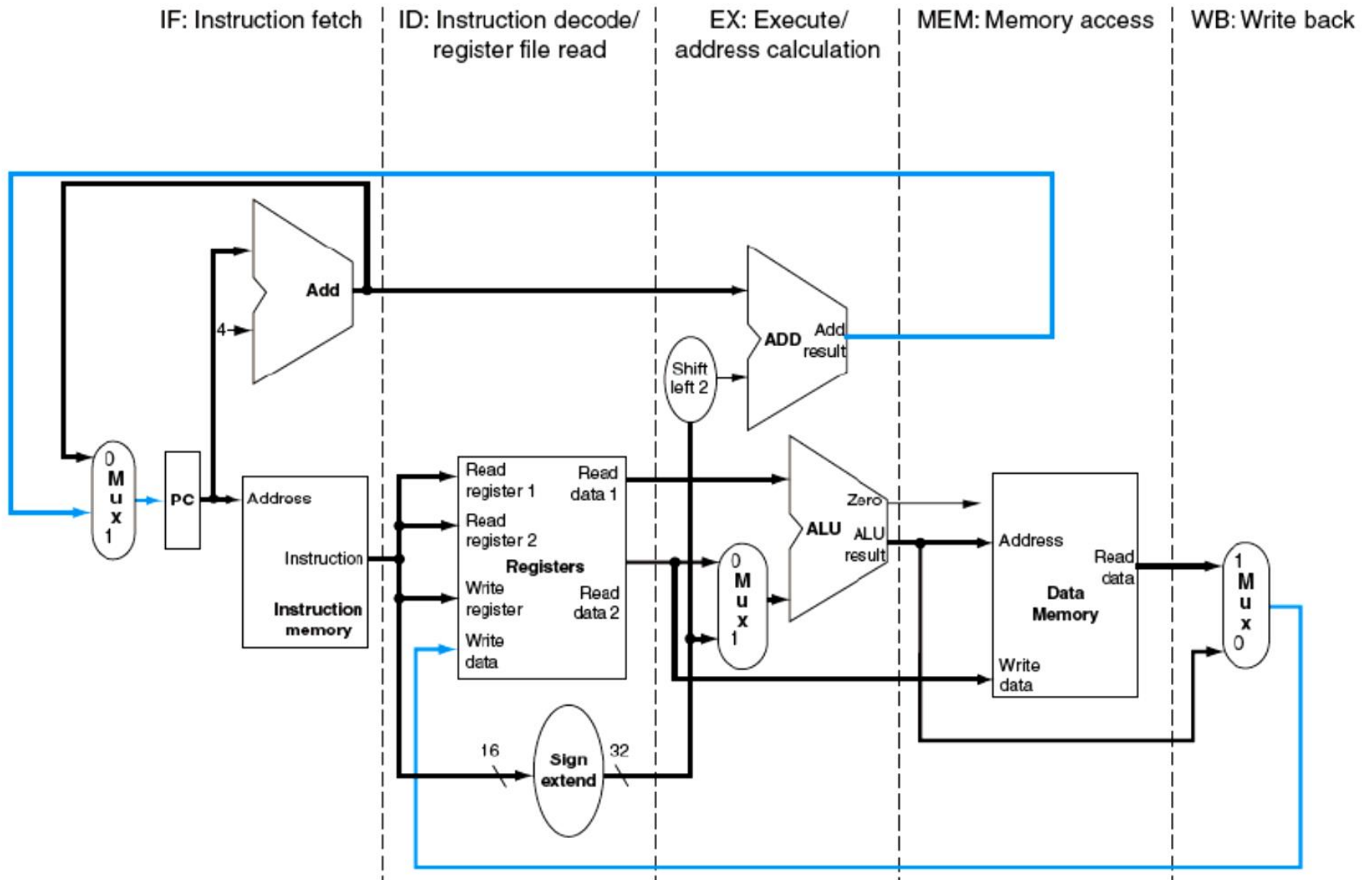
- Hazarduri Structurale: ex. presupunem ca avem o singură memorie
- Hazarduri de Control: tratarea instrucțiunilor de ramificare
- Hazarduri de Date: instrucțiunile pot depinde de rezultatele instrucțiunilor precedente

Urmează în curs => construim* un pipeline simplu pornind de la procesorul MIPS cu ciclu unic, **evidențiem problemele** uzuale care apar, și **căutăm soluții** pentru aceste probleme;

** descrierea din curs urmează referința [1], capitolul 6, cu anumite modificări*

Proiectarea MIPS pipeline

Pornim de la căile de date MIPS cu ciclu unic, cu cele 5 etape de execuție



Proiectarea MIPS pipeline

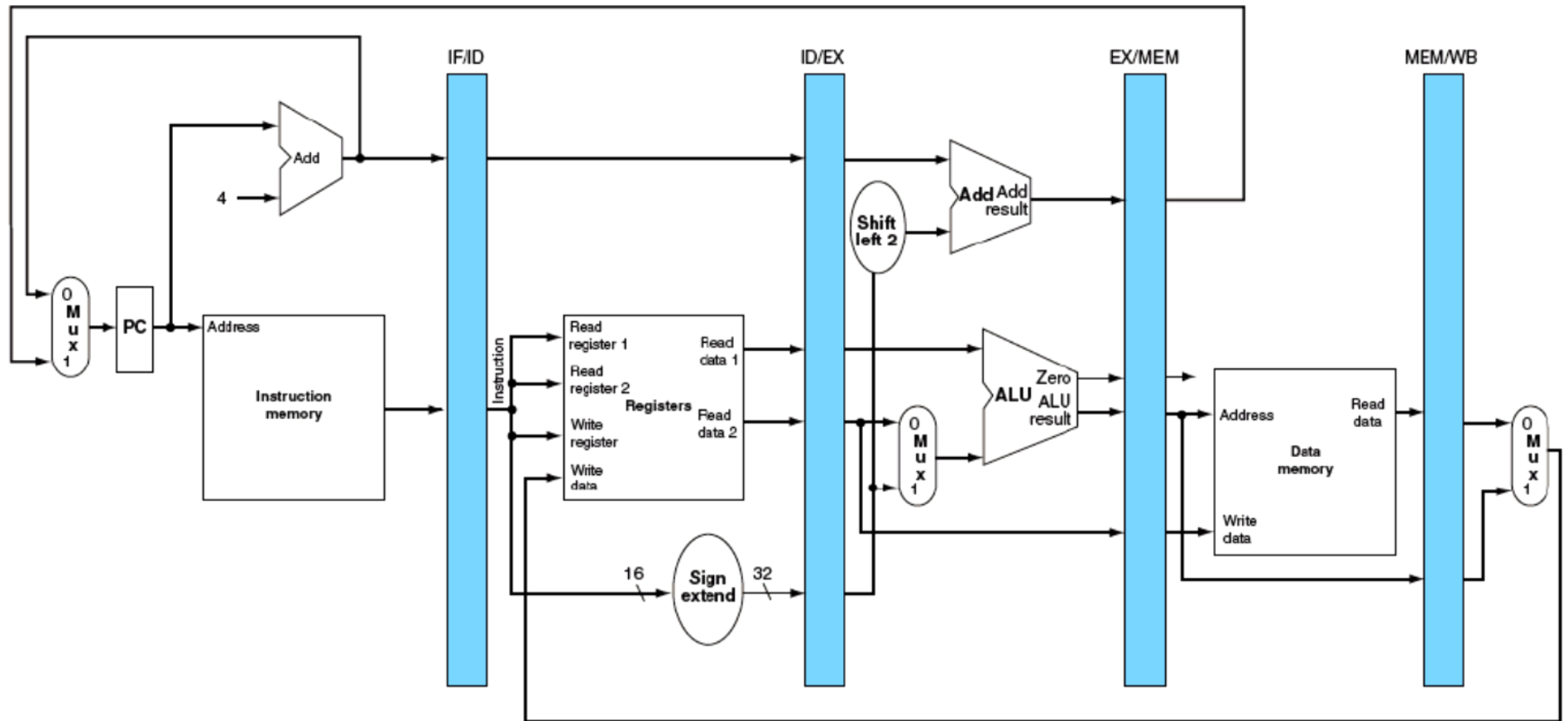
Ideea de bază

- Căile de date MIPS cu ciclu unic secționare în 5 etaje: IF, ID, EX, MEM, WB
- Până (!) la 5 instrucțiuni vor fi în execuție în fiecare ciclu de ceas
- Flux normal de date de la stânga la dreapta
- Căile de la dreapta la stânga (write-back în blocul de registre și în PC la ramificări) introduc complicații în pipeline (forwarding, branch delay)

Cum se împart căile de date în etaje?

- Se introduc registre între etaje (scrieri pe front crescător de ceas).
- Registrele transferă valori de date și semnale de control de la etaj la etaj.
- PC → teoretic este echivalent cu un registru pipeline înainte de etajul IF → un registru pipeline pentru fiecare etaj. Practic PC este parte a IF!

Proiectarea MIPS pipeline



MIPS cu ciclu unic, după inserarea de registre între etajele funcționale

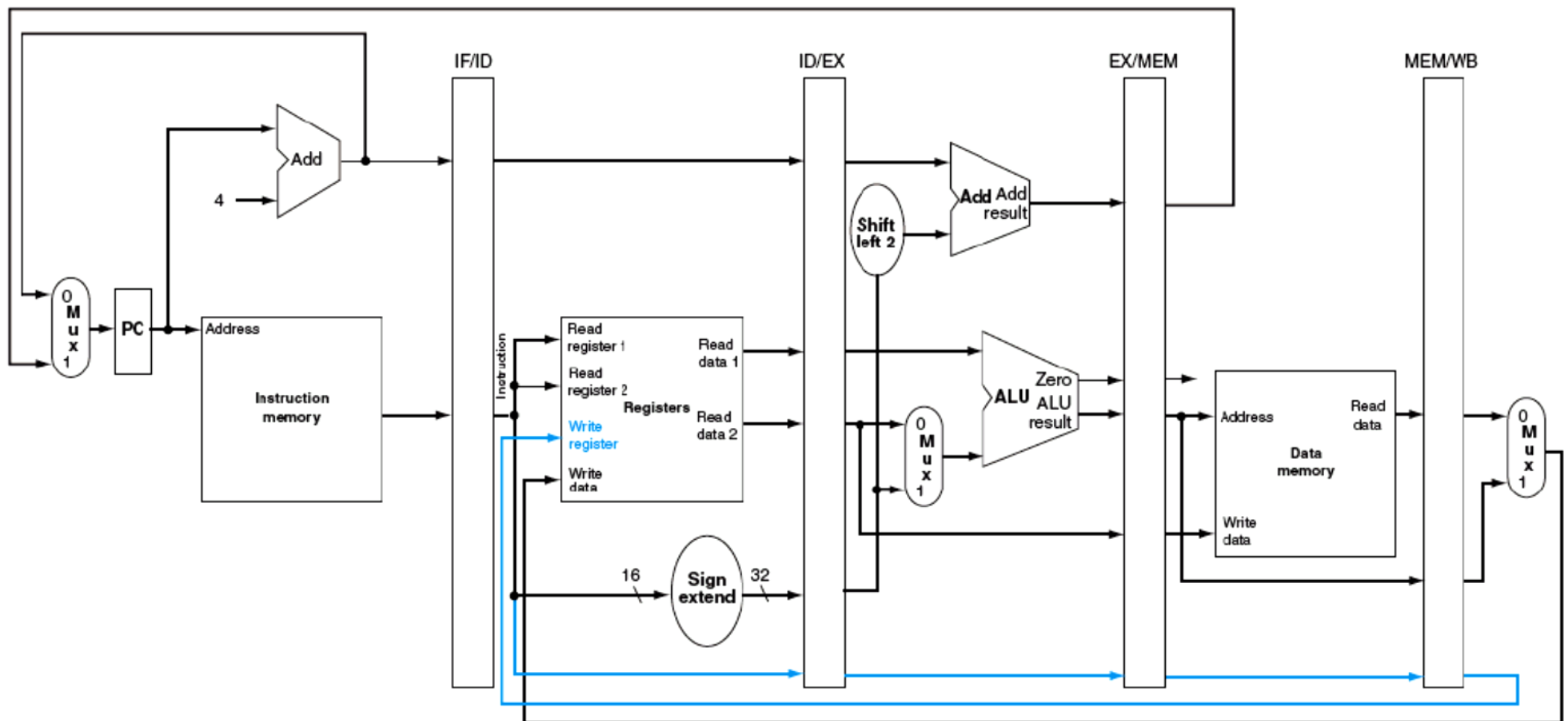
- Exista vreo problemă, chiar dacă se executa instrucțiuni independente?
- La ce tip de instrucțiuni apare problema?

Proiectarea MIPS pipeline

Răspuns

- Tip R, la blocul de registre: Write address și Write data de la instrucțiuni diferite

Pipeline corectat: Adresa de destinație și Data corespunzătoare trec împreună prin pipeline



MIPS, pipeline corectat pentru blocul de registre

Proiectarea MIPS pipeline - operații în etaje

IF – Aducerea instrucțiunii

- PC – Adresa pentru Memoria de Instrucțiuni (IM);
 - **IF/ID.IR \leftarrow IM[PC]**; se citește instrucțiunea de executat
- Adresa PC incrementată cu 4
 - **IF/ID.PC \leftarrow PC + 4**
- Se scrie în PC adresa PC incrementată sau adresa de ramificare
 - **PC \leftarrow PC + 4 sau PC \leftarrow EX/MEM.PC (\Leftrightarrow PC + 4 + S_Ext(Imm) \ll 2)**

ID (ID&OF) – Decodificarea instrucțiunii și citirea operanzilor

- OF: Se citesc operanzii din Blocul de Registre \rightarrow în jumătatea a doua a perioadei de ceas, asincron, apar valorile actualizate.
 - **ID/EX.A \leftarrow RF[rs]**
 - **ID/EX.B \leftarrow RF[rt]**
- Câmpul imediat se extinde cu Semn sau cu Zero
 - **ID/EX.Imm \leftarrow S_Ext(Imm) sau ID/EX.Imm \leftarrow Z_Ext(Imm)**
- Se transmit adresele care pot fi folosite ca adrese de scriere în Blocul de Registre
 - **ID/EX.WriteRegRd \leftarrow rd, ID/EX.WriteRegRt \leftarrow rt**
 - Vor intra în mux-ul din EX
- PC incrementat se transmite la etajul următor
 - **ID/EX.PC \leftarrow IF/ID.PC** (\Leftrightarrow PC + 4)

Proiectarea MIPS pipeline - operații în etaje

EX – Etajul de execuție

- Se calculează adresa efectivă pentru instrucțiunile care accesează **Memoria**
 - **EX/MEM.ALUResult** \leftarrow **ID/EX.A** + **ID/EX.Imm** (\Leftrightarrow **RF[rs]** + **S_Ext(Imm)**)
- Se transmite Data de stocat pentru instrucțiunea **SW**
 - **EX/MEM.WriteData** \leftarrow **ID/EX.B** (\Leftrightarrow **RF[rt]**)
- Instrucțiuni tip R, operații tip ALU:
 - **EX/MEM.ALUResult** \leftarrow **ID/EX.A** (**ALUOp-func**) **ID/EX.B**
(\Leftrightarrow **RF[rs]** **ALUOp-func** **RF[rt]**)
- Instrucțiuni tip I, operații tip ALU:
 - **EX/MEM.ALUResult** \leftarrow **ID/EX.A** (**ALUOp-codop**) **ID/EX.Imm** – aritmetice
 - **EX/MEM.ALUResult** \leftarrow **ID/EX.A** (**ALUOp-codop**) **ID/EX.Imm** – logice
- Instrucțiuni de Ramificare, test de egalitate în ALU
 - **EX/MEM.Zero** \leftarrow **ALUZero** (\Leftrightarrow **ID/EX.A** – **ID/EX.B** = 0 ?)
- Adresa de ramificare se calculează în Sumatorul suplimentar
 - **EX/MEM.PC** \leftarrow **ID/EX.PC** + **ID/EX.Imm** \ll 2 (\Leftrightarrow **PC** + 4 + **S_Ext(Imm)** \ll 2)
- Se transmite mai departe adresa potențială de scriere în Blocul de Registre
 - **EX/MEM.Writereg** \leftarrow **ID/EX.WriteRegRd** or **ID/EX.WriteRegRt**

Proiectarea MIPS pipeline - operații în etaje

MEM – Memoria

- **LW:** Se citește din memoria de date, conform adresei calculată în etajul EX
 - $\text{MEM/WB.LMD} \leftarrow \text{DM} [\text{EX/MEM.ALUResult}]$
 - **LMD-Load Memory Data register**
- **SW:** Se scrie în memoria de date, conform adresei calculată în etajul EX, valoarea citită din RF[rt] în etajul ID
 - $\text{DM}[\text{EX/MEM.ALUResult}] \leftarrow \text{EX/MEM.WriteData}$
- Se transmite mai departe adresa de scriere în Blocul de Registre
 - $\text{MEM/WB.WriteReg} \leftarrow \text{EX/MEM.WriteReg}$
- Se transmit rezultatele operațiilor ALU / adresa de ramificare
 - $\text{MEM/WB.ALUResult} \leftarrow \text{EX/MEM.ALUResult}$
 - **BEQ:** $(\text{EX/MEM.Zero}) \rightarrow \text{PC} \leftarrow \text{EX/MEM.PC} \quad (\Leftrightarrow \text{PC} + 4 + \text{S_Ext(Imm)} \ll 2)$

WB – Write back

- **Operații tip ALU pentru instrucțiunile tip R și I:**
 - Rezultatul obținut în ALU se scrie în Blocul de Registre.
 - $\text{RF}[\text{MEM/WB.WriteReg}] \leftarrow \text{MEM/WB.ALUResult}$ (fie în RF[rd], fie în RF[rt])
- **LW :** Valoarea citită din Memoria de Date se scrie în Blocul de Registre
 - $\text{RF}[\text{MEM/WB.WriteReg}] \leftarrow \text{MEM/WB.LMD}$ (în RF[rt])
- Scriere în Blocul de Registre → pe frontul **descrescător** al ciclului de ceas, este posibilă citirea valorii noi în același ciclu (citirea este asincronă!)

Proiectarea MIPS pipeline - operații în etaje

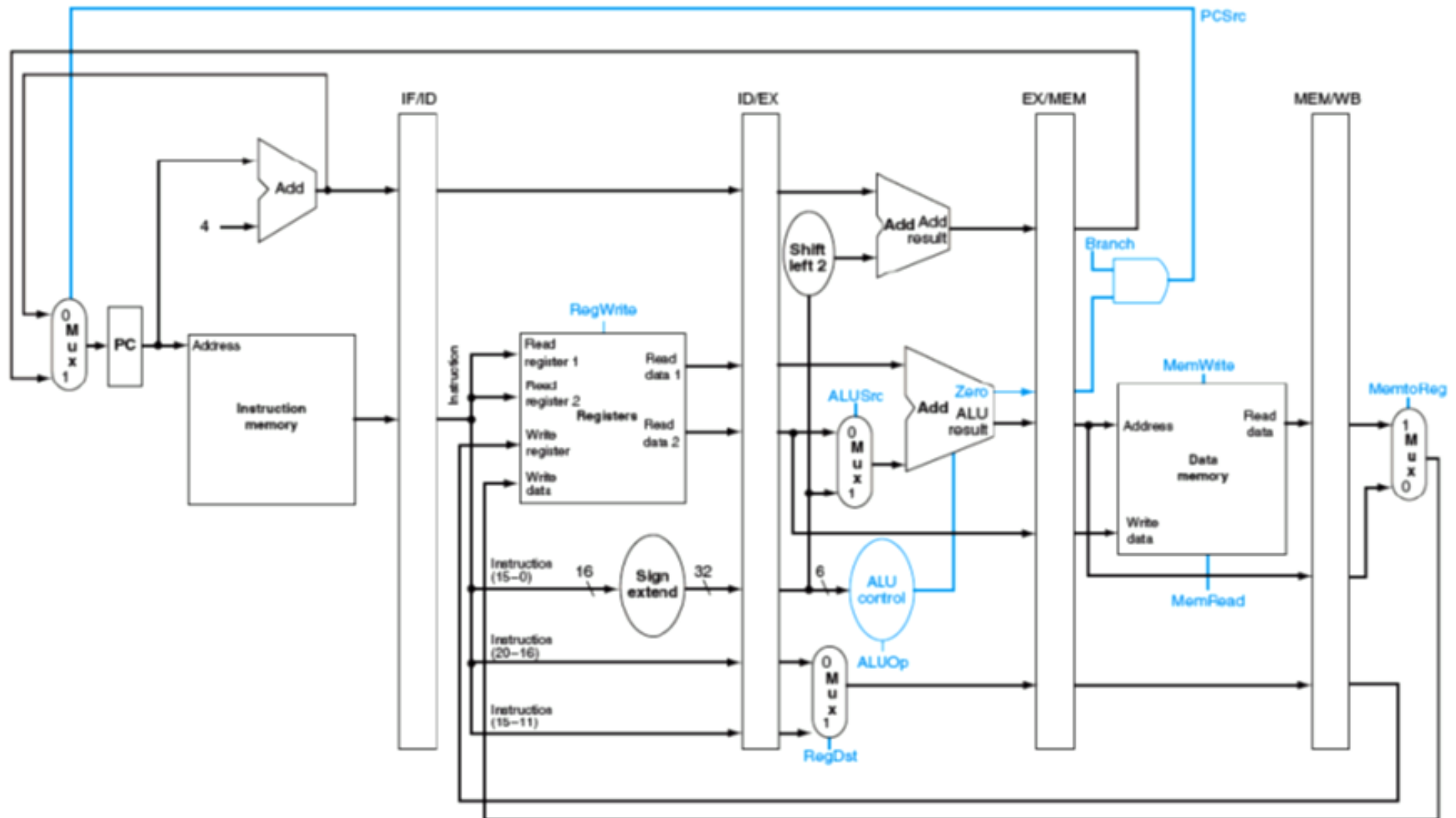
Descriere RTL, sumar

	R – R (Tip R)	R–I (Tip I, aritm.)	LW	SW	BR
IF	IF/ID.IR \leftarrow IM[PC] IF/ID.PC \leftarrow PC+ 4 PC \leftarrow PC+4 or (EX/MEM.Zero&Branch) \rightarrow PC \leftarrow EX/MEM.PC				
ID	ID/EX.A \leftarrow RF[rs] ID/EX.B \leftarrow RF[rt] ID/EX.Imm \leftarrow S_Ext(Imm) or ID/EX.Imm \leftarrow Z_Ext(Imm) ID/EX.WriteRegRd \leftarrow rd ID/EX.WriteRegRt \leftarrow rt ID/EX.PC \leftarrow IF/ID.PC				
EX	EX/MEM. ALUResult \leftarrow ID/EX.A (ALUOp-Func) ID/EX.B	EX/MEM. ALUResult \leftarrow ID/EX.A (ALUOp-codop) ID/EX.Imm	EX/MEM. ALUResult \leftarrow ID/EX.A + ID/EX.Imm	EX/MEM. ALUResult \leftarrow ID/EX.A + ID/EX.Imm	EX/MEM.Zero \leftarrow ALUZero (ID/EX.A – ID/EX.B)
	EX/MEM.WriteReg \leftarrow ID/EX.WriteRegRd or ID/EX.WriteRegRt			EX/MEM.WriteData \leftarrow ID/EX.B	EX/MEM.PC \leftarrow ID/EX.PC + ID/EX.Imm << 2
MEM	MEM/WB. ALUResult \leftarrow EX/MEM.ALUResult	MEM/WB.ALUResult \leftarrow EX/MEM.ALUResult	MEM/WB.LMD \leftarrow DM [EX/MEM. ALUResult]	DM[EX/MEM.ALUResult] \leftarrow EX/MEM.WriteData	EX/MEM.Zero \rightarrow Branch
	MEM/WB.WriteReg \leftarrow EX/MEM.WriteReg				
WB	RF[MEM/WB.WriteReg] \leftarrow MEM/WB.ALUResult	RF[MEM/WB.WriteReg] \leftarrow MEM/WB.ALUResult	RF[MEM/WB.WriteReg] \leftarrow MEM/WB.LMD		

Ce trebuie transmis, pe lângă date, prin registrele intermediare, pentru ca aceste transferuri RTL să se execute corect ?

Proiectarea MIPS pipeline - Control

Specificarea semnalelor de control. Ce se comandă în fiecare etaj?



MIPS pipeline cu semnale de control

Proiectarea MIPS pipeline - Control

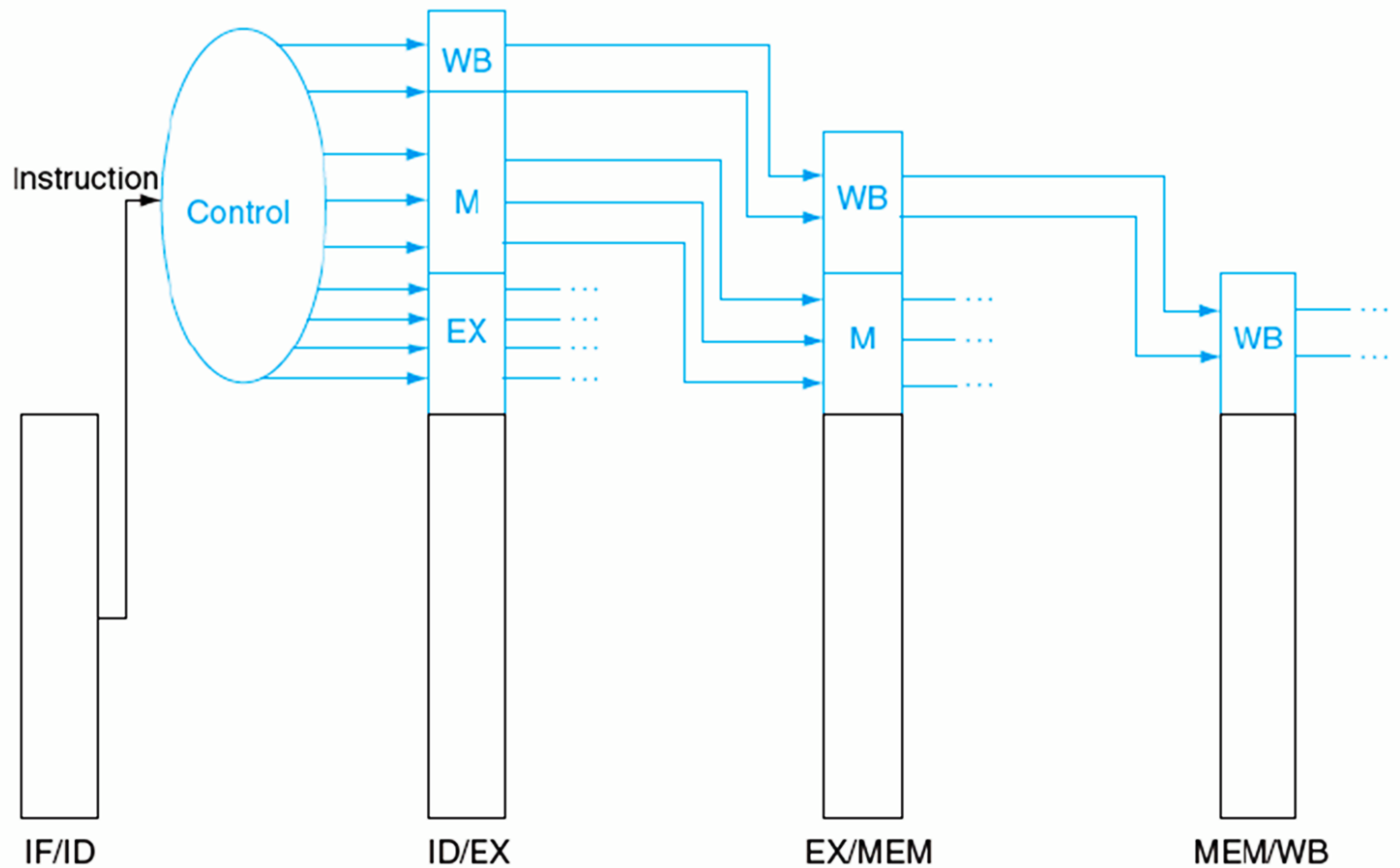
Instr	EX Stage			MEM Stage			WB Stage	
	RegDst	ALUOp	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
R	1	10	0	0	0	0	1	0
lw	0	00	1	0	1	0	1	1
sw	X	00	1	0	0	1	0	X
beq	X	01	0	1	0	0	0	X

Valorile semnalelor de control pentru diferite instrucțiuni, în diferite etaje (IF, ID comune)

Semnalele de control pentru Pipeline trebuie să “călătorească” împreună cu rezultatele intermediare:

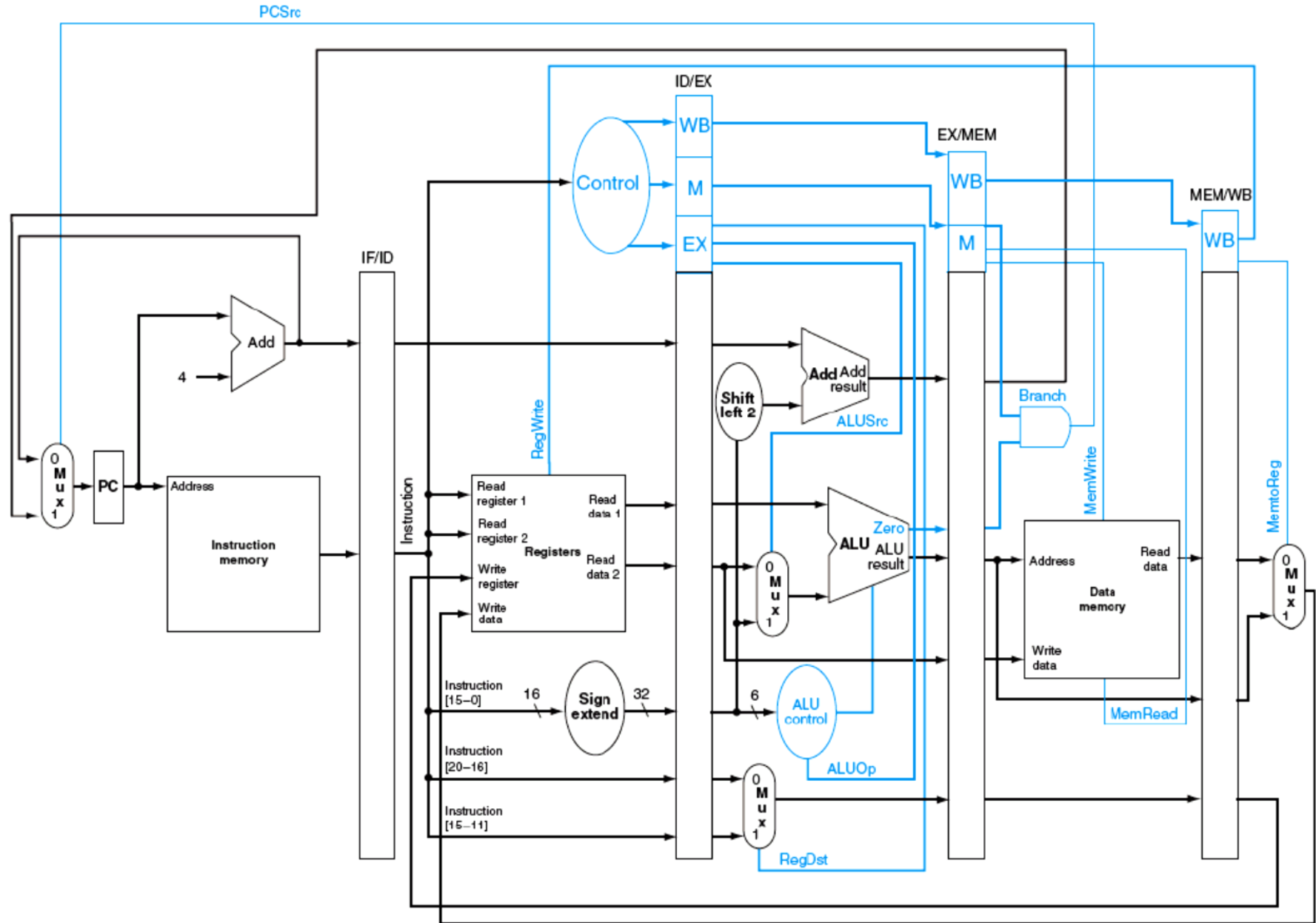
- Unitatea de control principală generează semnalele de control în etajul ID
- Pentru o anumită instrucțiune aflată în etajul ID:
 - Semnalele de comandă pentru Ex (RegDst, ALUOp, ALUSrc) – necesare după 1 ciclu
 - Semnalele de comandă pentru MEM (MemWrite, Branch, – necesare după 2 cicluri
 - Semnalele de comandă pentru WB (MemtoReg, RegWrite) – necesare după 3 cicluri
- **Practic, semnalele de control se transmit, ca și datele, de la etaj la etaj**

Proiectarea MIPS pipeline - Control



Liniile de control pentru ultimele trei etaje se propagă la fel ca datele

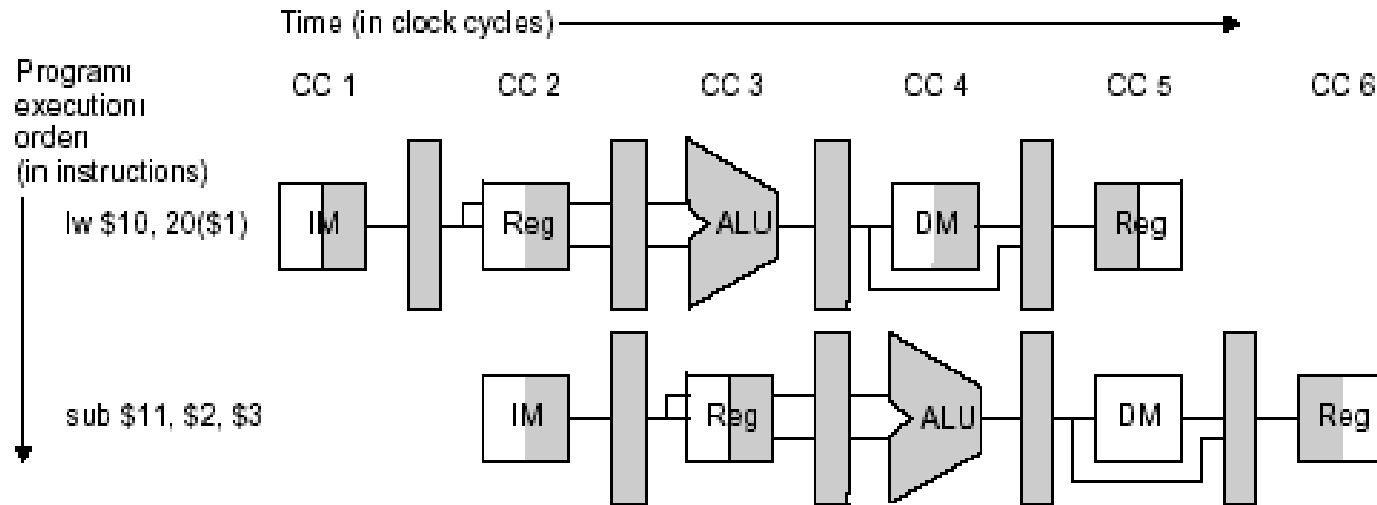
Proiectarea MIPS pipeline - Control



MIPS pipeline căile de date cu unitatea de control (principală și secundară ALU)

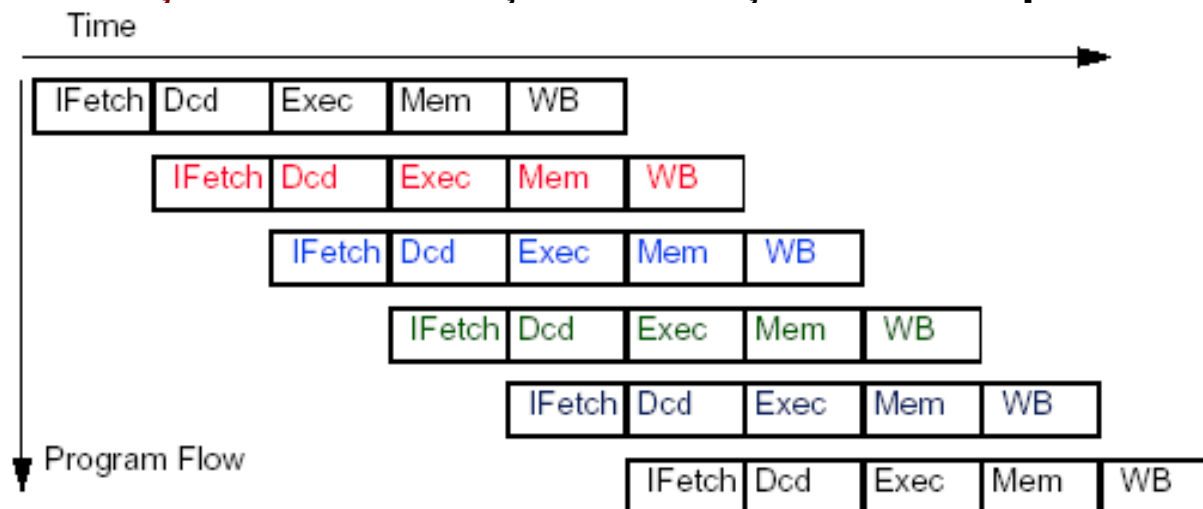
Proiectarea MIPS pipeline - Reprezentare

Reprezentarea grafică a execuției instrucțiunilor în Pipeline



- **Utilitate:** Câte cicluri durează executarea unui cod? Ce face UAL in ciclul 4? Etc.

Reprezentarea convențională a execuției instrucțiunilor în Pipeline



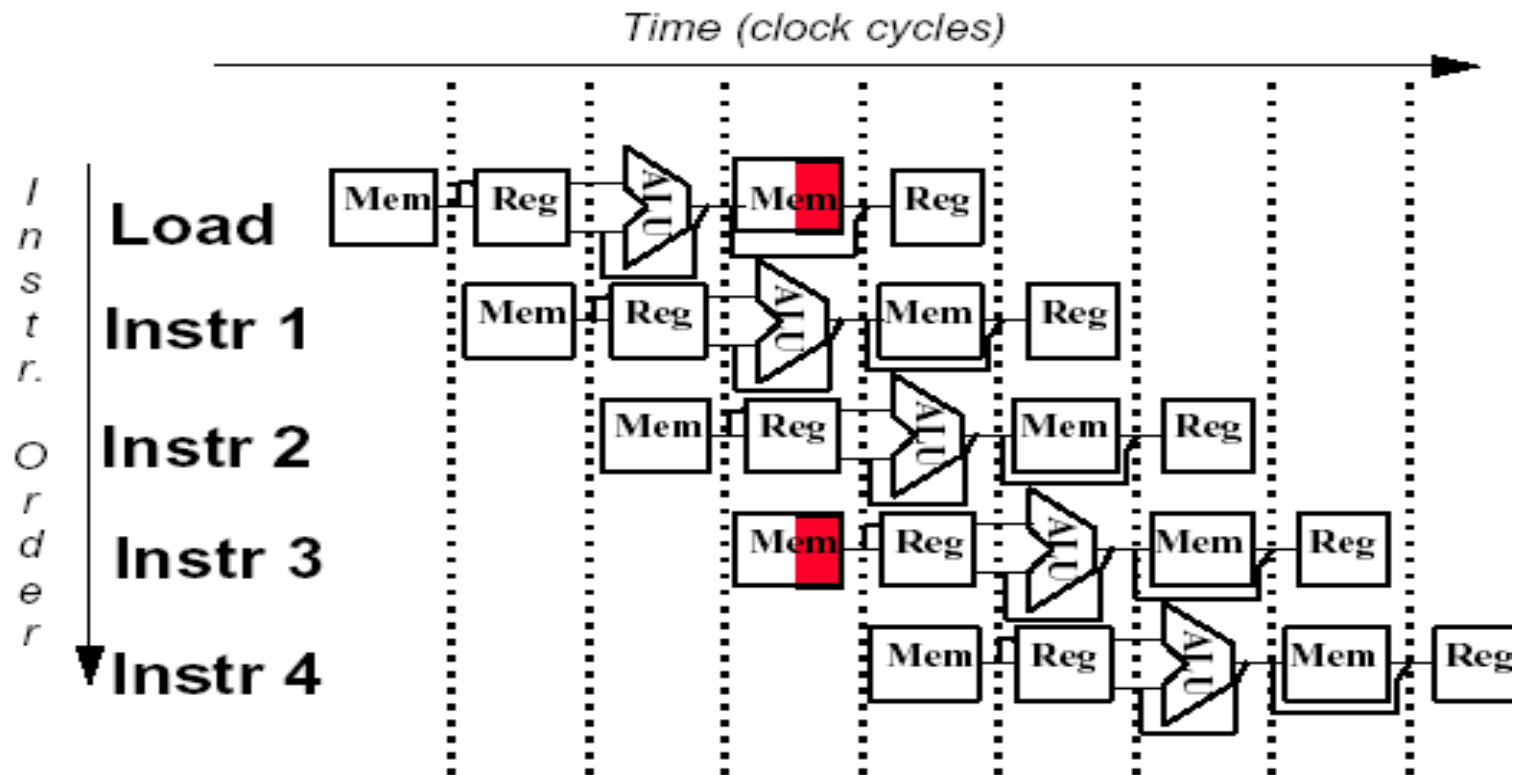
Proiectarea MIPS pipeline – Hazarduri

- **Hazardurile** = situații în care o instrucțiune următoare nu poate fi executată în următoarea perioadă de ceas
- **Trei tipuri de hazarduri**
 - **Hazard structural (dependență de resurse)**
 - Două instrucțiuni încearcă să folosească simultan o resursă, în scopuri diferite → constrângeri de resurse
 - **Hazard de date (dependența 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
 - **Hazard de control/comandă (dependența 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)
- **Hazardurile pot fi rezolvate întotdeauna prin așteptare**
 - Controlul pipeline trebuie să detecteze hazardurile și să le rezolve
 - O soluție comună este „stall” – oprirea parțială a pipeline-ului până la dispariția hazardului, introducând unul sau mai multe goluri “bubbles” (NOOPs, stalls) în pipeline

Proiectarea MIPS pipeline – Hazarduri

Hazard Structural

- Scenariu: O singură memorie → Hazard Structural



Acces simultan la memorie: Load si Instr 3, IF

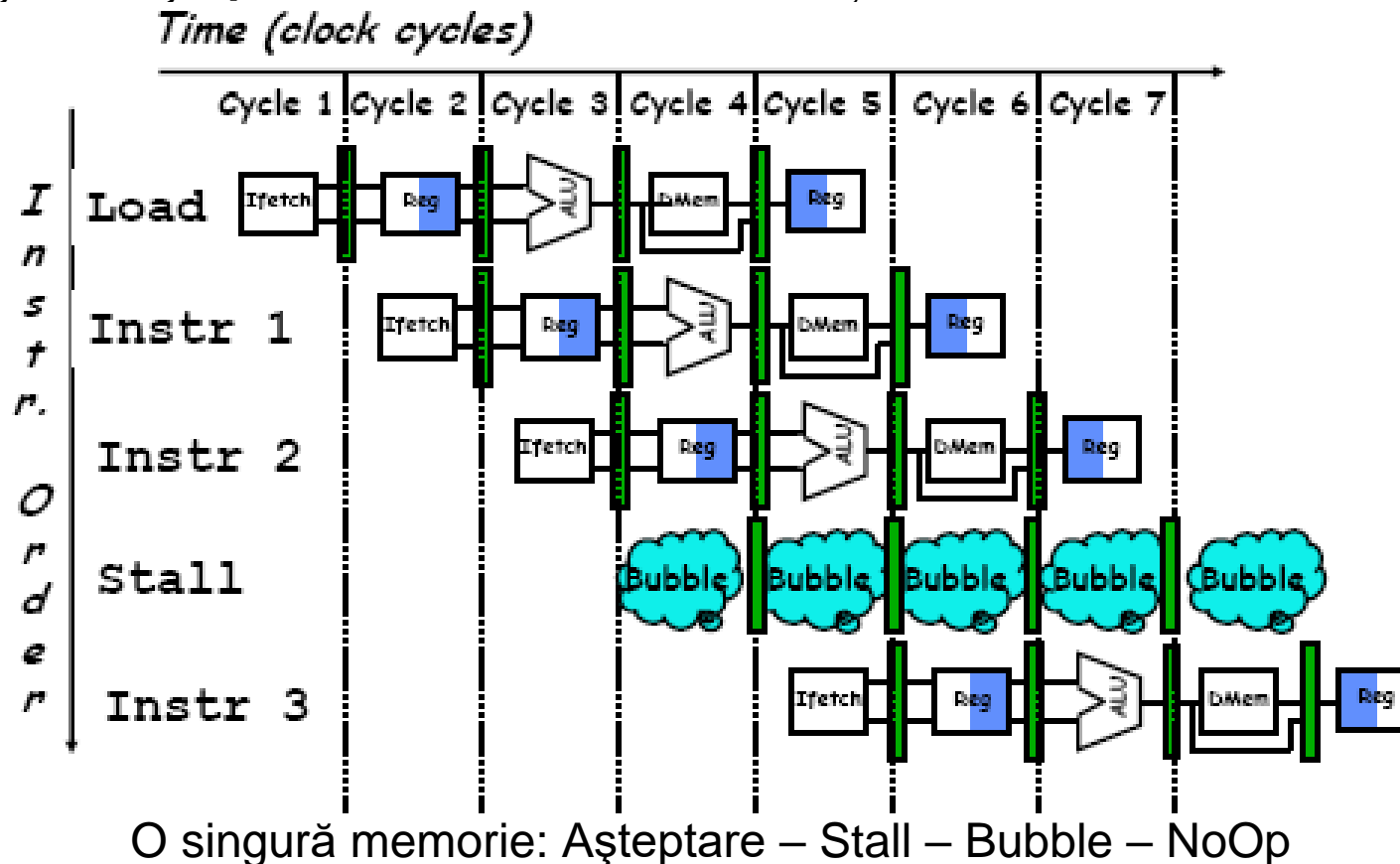
Ce soluție se poate adopta?

Proiectarea MIPS pipeline – Hazarduri

Hazard Structural

➤ O singură memorie

- **Soluția cu așteptare** reduce viteza de execuție



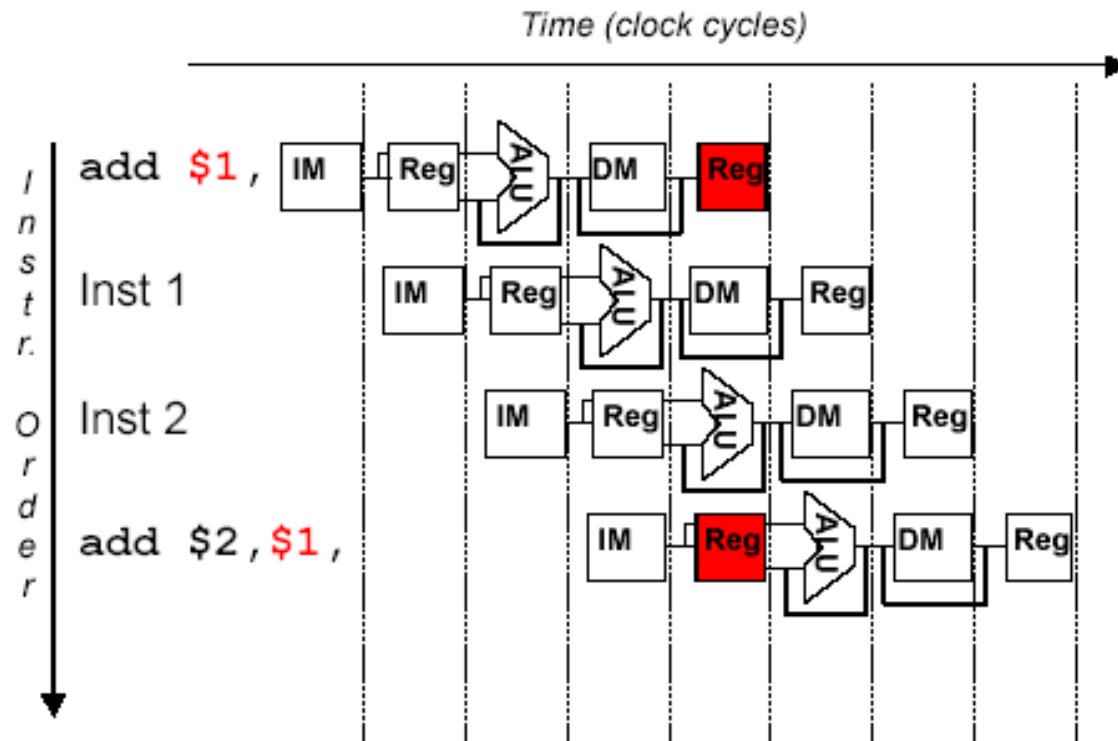
Soluția uzuală:

- Memorii (cache în procesoare reale !) independente pentru instrucțiuni și date, IM și DM, deja moștenite din MIPS cu ciclu unic

Proiectarea MIPS pipeline – Hazarduri

Hazard Structural

- Acces simultan la Blocul de registre (în acest caz hazardul fiind de fapt de date!)
 - Citire operanzi în faza ID și scrierea rezultatului în faza WB



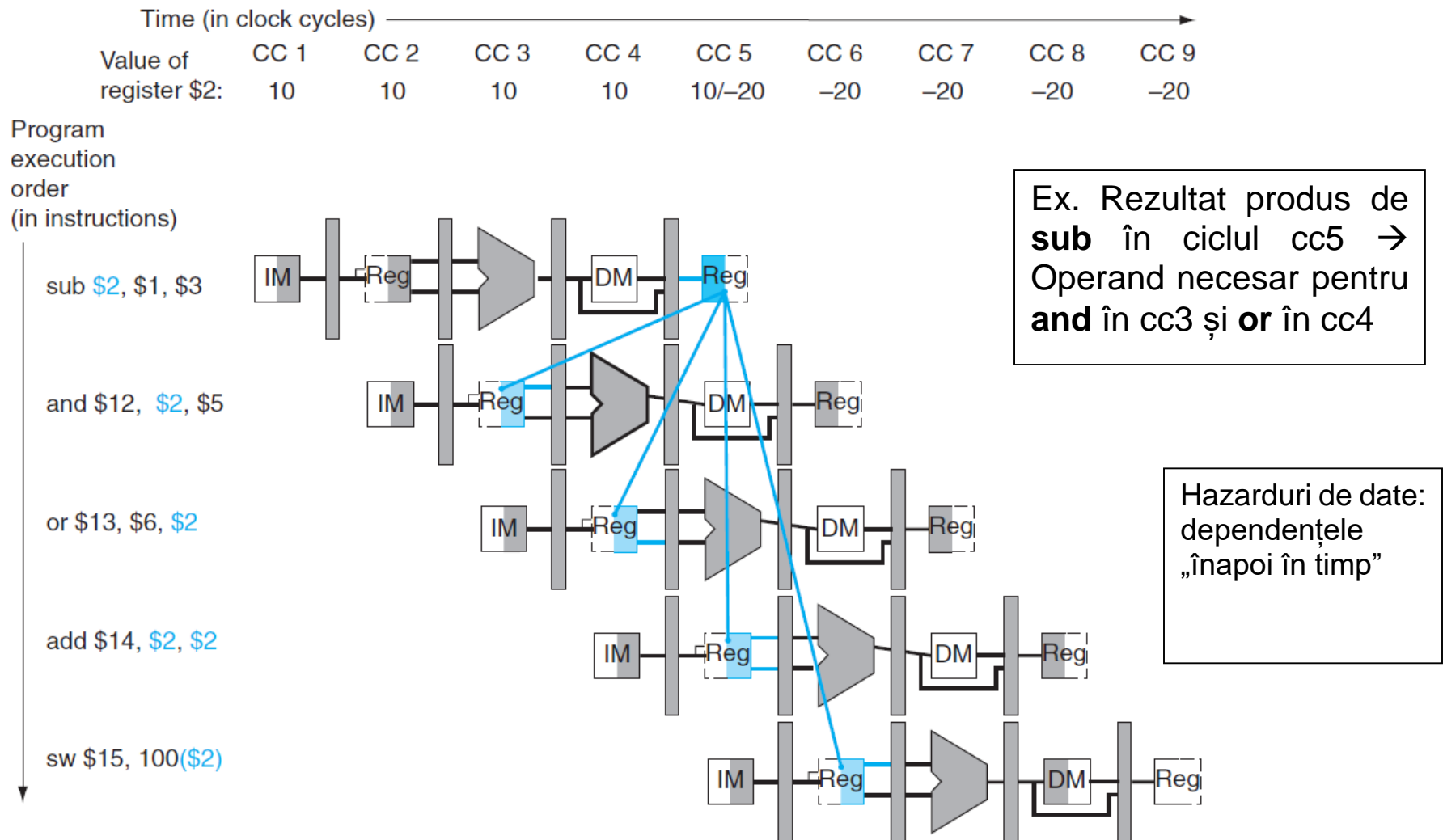
Acces simultan la blocul de regiștri în etajele de WB și ID

- **Soluție prin** modificarea blocului de regiștri RF din MIPS cu ciclu unic:
 - **Scriere în RF pe frontul descrescător** al perioadei de ceas!
 - Citirea din RF este asincronă, se face până la sfârșitul perioadei de ceas

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Forwarding (Înaintare)

- Se lansează o instrucțiune în pipeline, a cărei operanzi vor fi produși de instrucțiunile precedente, în curs de execuție în pipeline (ne-finalizate)
- Folosirea registrelor cu valori în curs de calculare poate cauza hazarduri de date



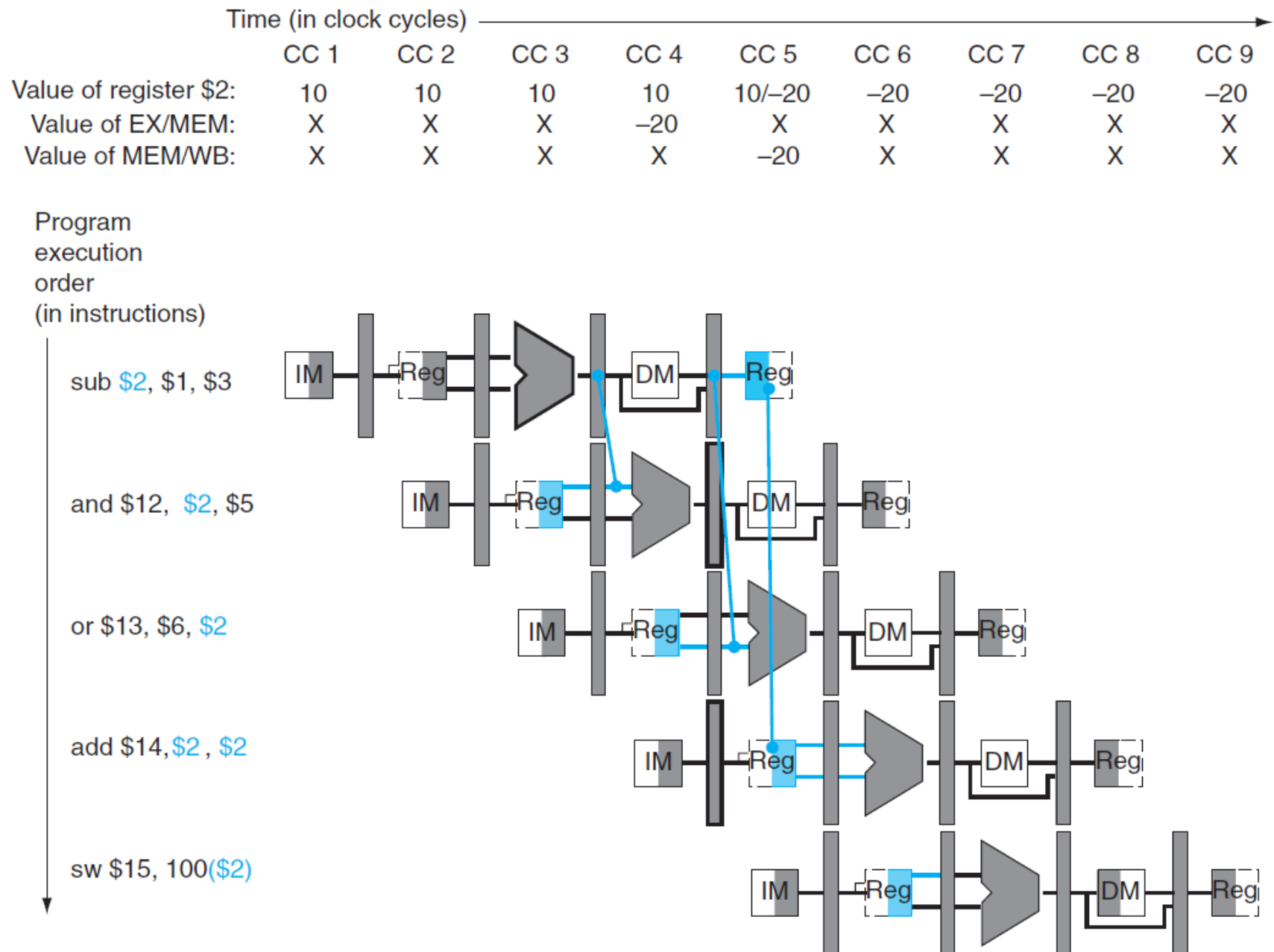
Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Forwarding (Înaintare)

- Apare la instrucțiunea din ID, iar operandul în curs de prelucrare este în alte etaje de pipeline
- Dependențele care "merg înapoi în timp" cauzează hazarduri de date
- Numele uzual al acestui hazard de date: **RAW – Read After Write Hazard**, dependență reală de date
- **Soluție software** (programator sau compilator-asamblor): inserare în program de instrucțiuni **NOP**; Reduce viteza programului.
- **Soluția hardware: „Înaintarea datelor” – Forwarding**

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Forwarding (Înaintare)



Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Forwarding (Înaintare)

- **Forwarding**: Folosirea rezultatelor din etajele EX, MEM, înainte de scrierea lor în Blocul de Registre în WB
- „Înaintarea” rezultatului produs de **sub (EX/MEM), (MEM/WB)** la intrarea **UAL** pentru **and** și **or**, în **cc4** și **cc5**
- „Înaintarea” implicită pentru **add** prin scriere pe front descrescător / Citire asincronă la blocul de registre

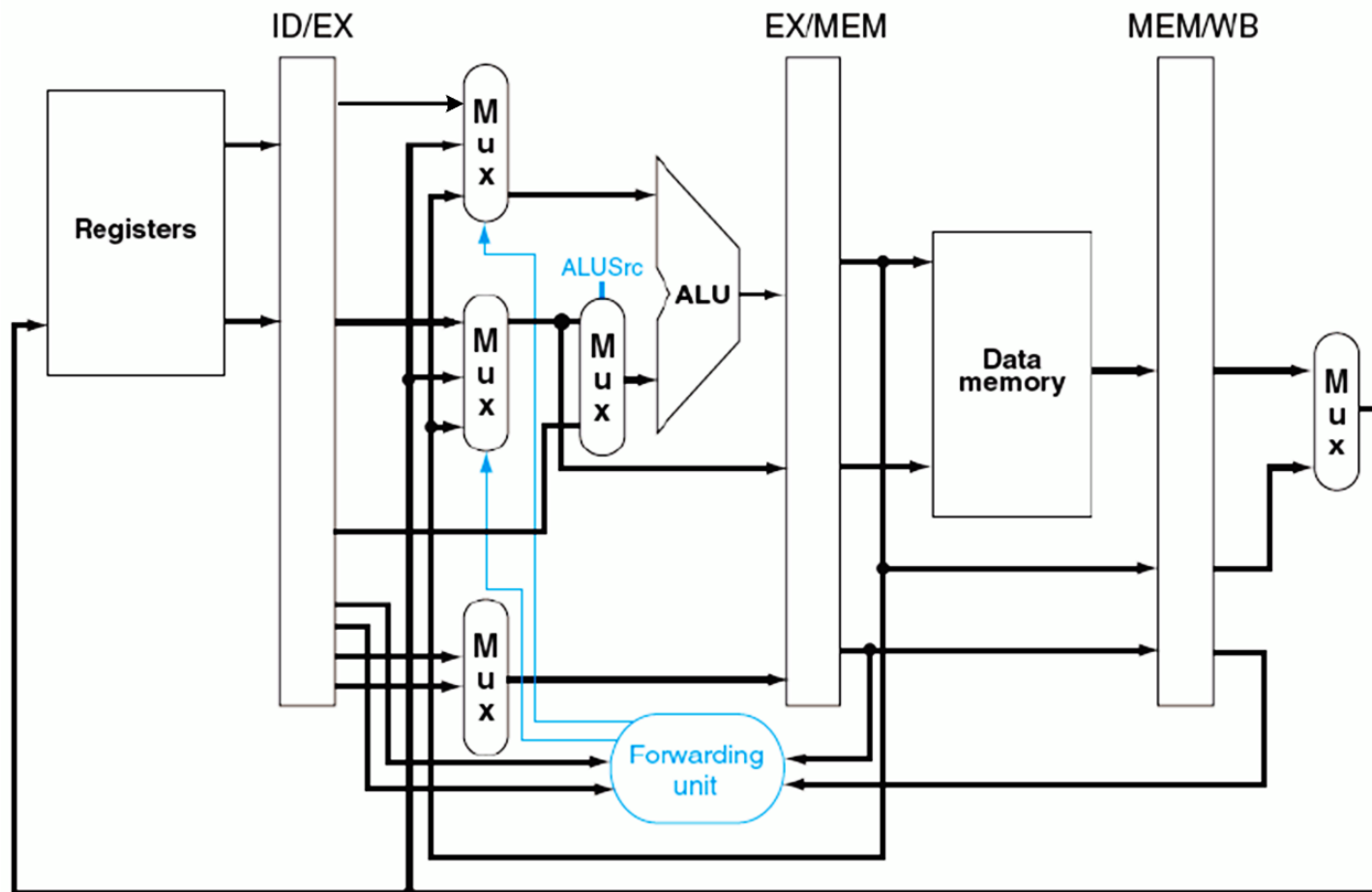
Discutat până acum **ce** trebuie făcut...

...**Cum** trebuie făcut? Urmează...

Proiectarea MIPS pipeline – Hazarduri

Detectarea hazardurilor pentru „Înaintare”

- Hazardul trebuie detectat înainte de execuție (=ALU)
- Detectarea egalităților adreselor Rs, Rt din etajul EX cu Rd/Rt din etajele MEM și WB
- În caz de hazard, datele se înaintează la intrarea UAL



Unitatea de Forwarding și MUX-uri adăugate pentru înaintare

Proiectarea MIPS pipeline – Hazarduri

Detectarea hazardurilor pentru „Înaintare”

- Trei surse pentru fiecare MUX
 - ID/EX: date obișnuite din Blocul de Registre (00)
 - EX/MEM: date înaintate – rezultatul operației precedente din UAL (10)
 - MEM/WB: date înaintate din Memorie sau rezultatul pre-precedent din UAL (01)
- Notății: ID/EX.RegisterRs = Câmpul de adresă Rs pentru blocul de registre în registrul de pipeline ID/EX, etc.
- Se detectează condiția de hazard (este Rs sau Rt din EX același cu Rd din MEM sau WB), după care
 - „Înaintarea” se execută numai dacă este necesar, se verifică și semnalul RegWrite
- ⇒ **Logica de control** pentru înaintare (Forwarding Unit)

Proiectarea MIPS pipeline – Hazarduri

Logica de controlul pentru înaintare

1. **MEM hazard** (poate apărea la instrucțiuni consecutive, distanță 1):

if (EX/MEM.RegWrite and

(EX/MEM.RegisterRd != 0) (fără înaintare pentru reg. nr=0) și

(EX/MEM.RegisterRd = ID/EX.RegisterRs)); (condiție de hazard)

ForwardA = 10

if (EX/MEM.RegWrite and

(EX/MEM.RegisterRd != 0) and

(EX/MEM.RegisterRd = ID/EX.RegisterRt)); (condiție de hazard)

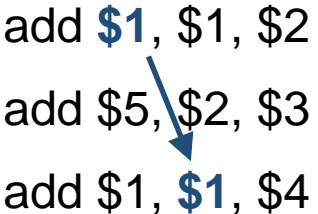

ForwardB = 10

➤ Înaintarea datelor de la instrucțiunea precedentă la intrările ALU. Nu se face înaintare pentru registrul nr. 0

Proiectarea MIPS pipeline – Hazarduri

Logica de controlul pentru înaintare

2. **WB hazard** (apare pentru instrucțiuni la distanță 2):

Problema tipică, hazard WB	Problema atipică, hazard MEM
 <p>add \$1, \$1, \$2 add \$5, \$2, \$3 add \$1, \$1, \$4</p>	 <p>add \$1, \$1, \$2 add \$1, \$1, \$3 add \$1, \$1, \$4</p>

Trebuie înaintat rezultatul cel mai recent:

- hazardul MEM, dacă există, are prioritate la rezolvare
- hazardul WB se rezolvă doar dacă nu există hazard MEM cu instrucțiunea anterioară (distanță 1)

⇒ rezultă logica de control

Proiectarea MIPS pipeline – Hazarduri

Logica de controlul pentru înaintare

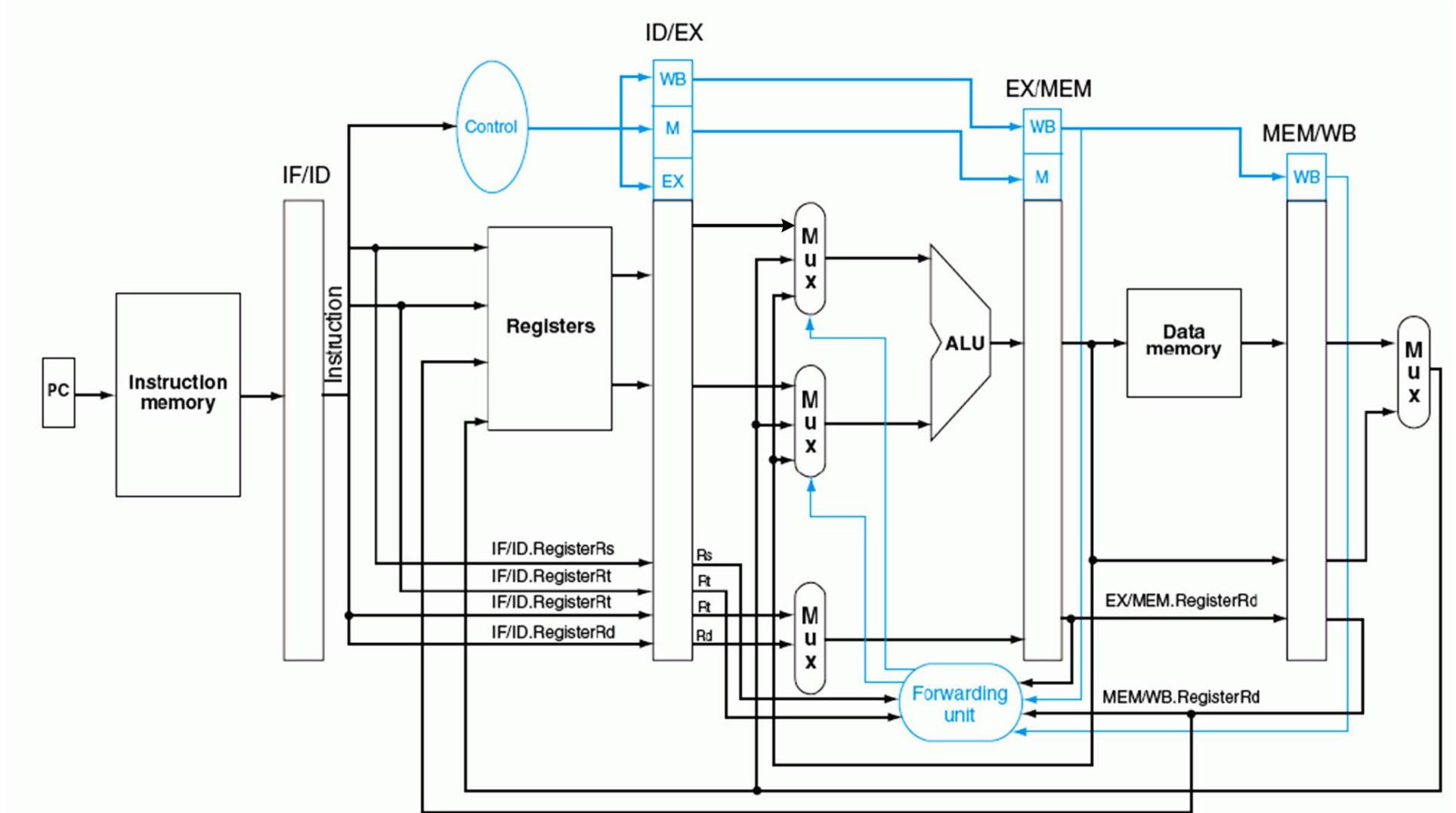
WB hazard:

```
if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd != 0)
    and (EX/MEM.RegisterRd != ID/EX.RegisterRs)      (nu este hazard MEM!)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs));      (condiție hazard WB)
    ForwardA = 01

if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd != 0)
    and (EX/MEM.RegisterRd != ID/EX.RegisterRt)      (nu este hazard MEM!)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt));      (condiție hazard WB)
    ForwardB = 01
```

➤ Înaintarea rezultatului cel mai recent la intrările ALU

Proiectarea MIPS pipeline – Hazarduri



MIPS pipeline cu Înaintare (fără unele detalii...)

(!) **Până acum** s-a discutat soluția cu Forwarding/Înaintare pentru hazard de date între instrucțiuni aritmetice/logice (tip R sau Tip I)

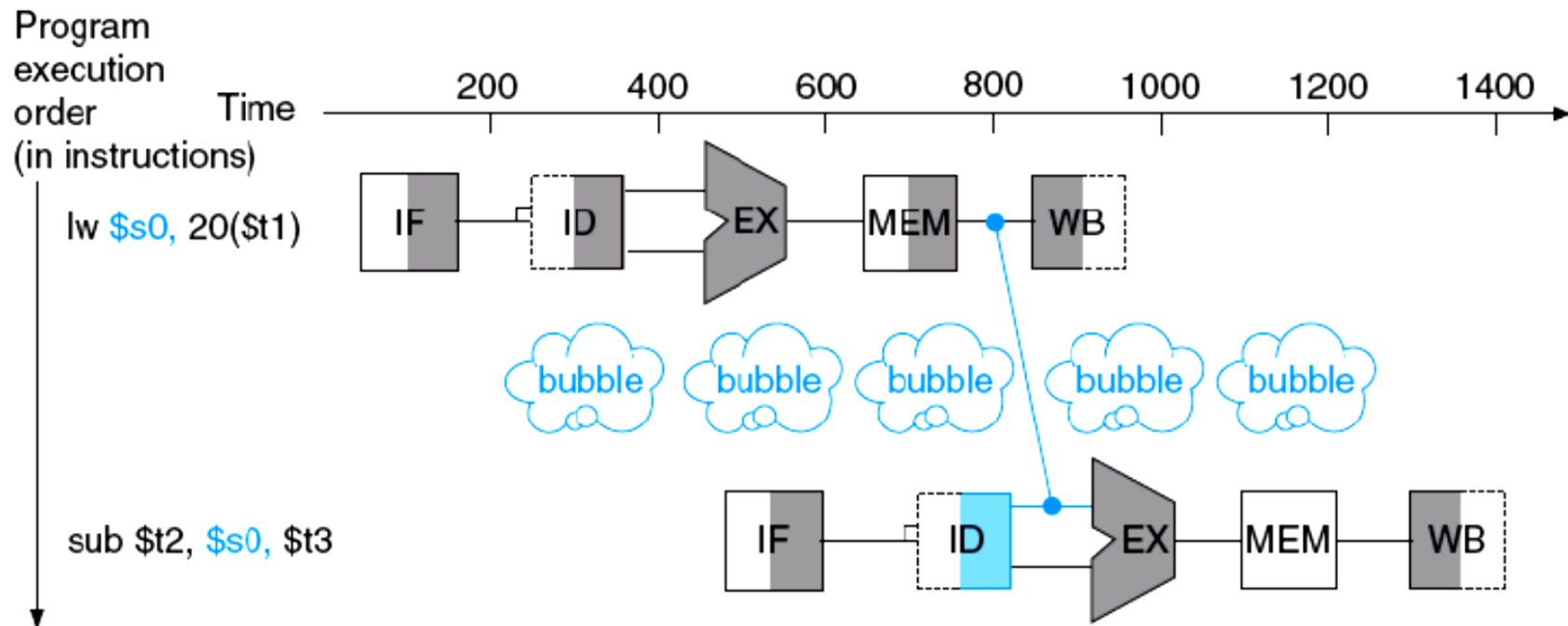
Proiectarea MIPS pipeline – Hazarduri

- Tehnica de înaintare se poate implementa și pentru alte operații în procesor. Ex. calculul adresei de memorie la lw/sw. Detalii în bibliografie!

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)

- Unele hazarduri **nu** se pot rezolva **decât** prin așteptare, **doar înaintarea nu ajută!**

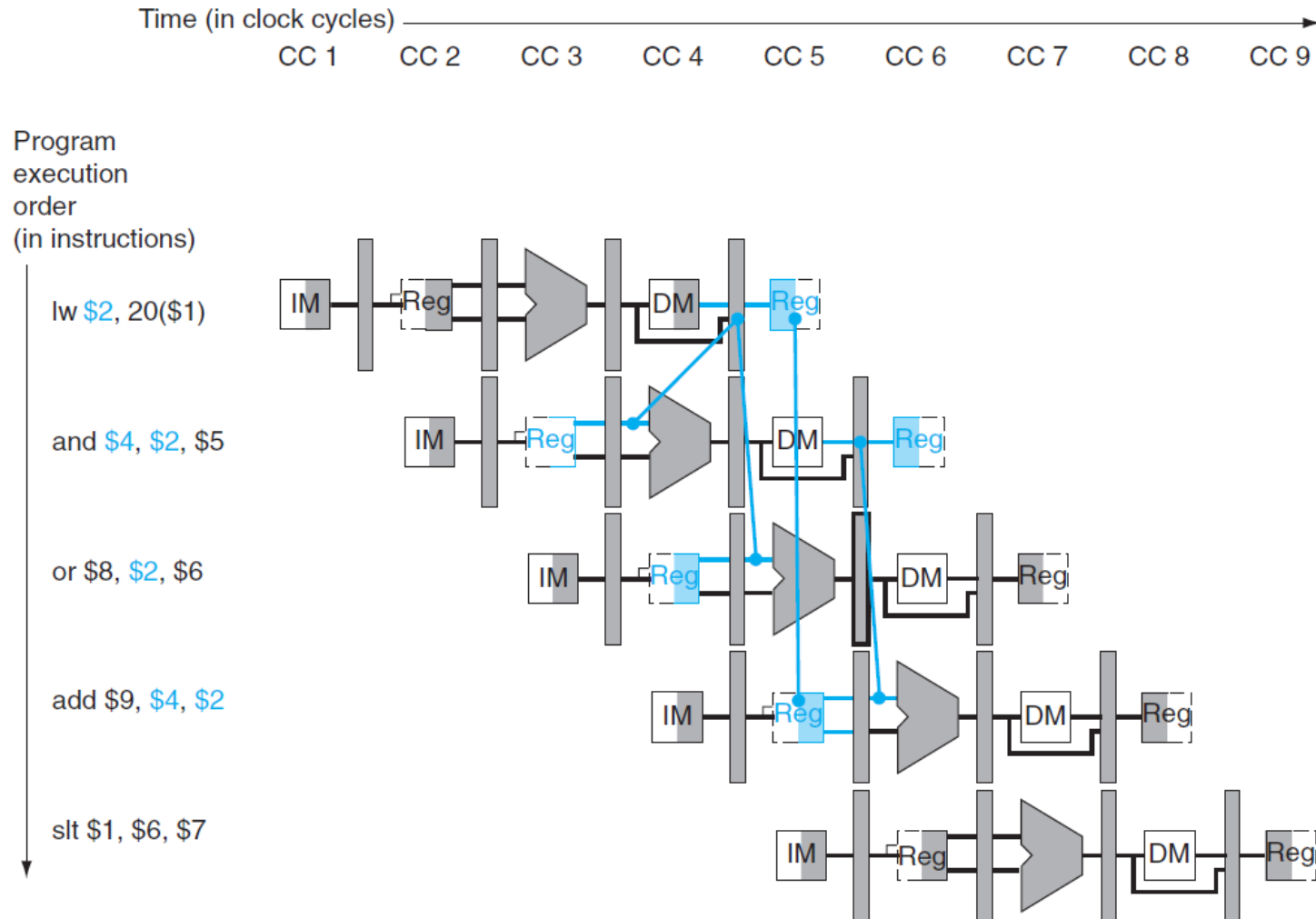


Instrucțiuni tip R dependente de LW → așteptare (stall sau bubble)

- Nu se poate înainta până la terminarea citirii din memorie
- **Load word** poate cauza **hazard**:
 - Instrucțiunea de după LOAD citește registrul scris de LOAD
- Referit ca **Load data hazard**

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)

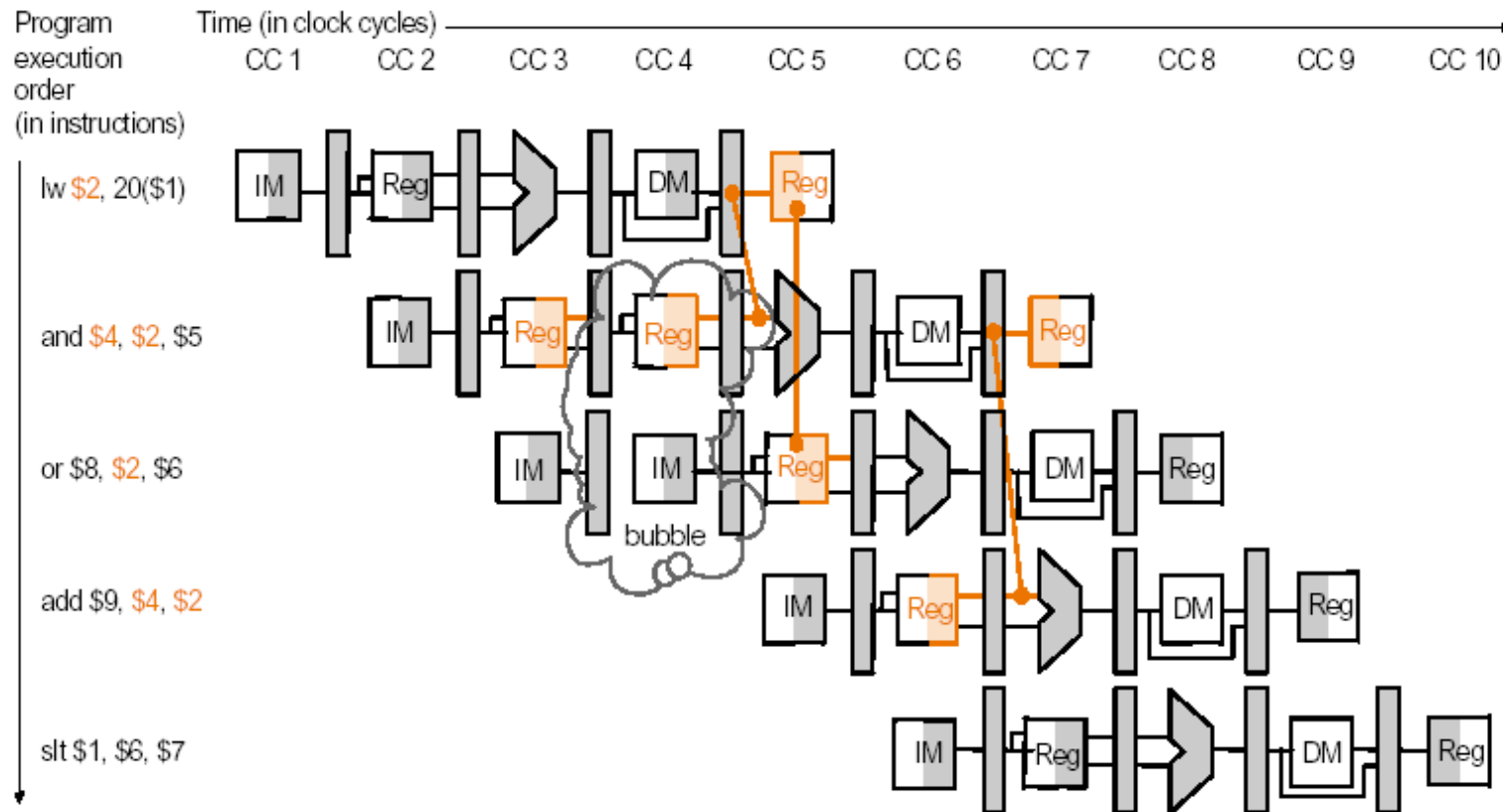


Hazardul între LW și AND nu se poate rezolva prin înaintare, AND trebuie să aștepte

Soluția: Se introduce un ciclu de așteptare (bubble / stall / NoOp) în ciclul 4, NOP în loc de AND =>

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)



În caz de dependență, după LW se introduce așteptare de un ciclu, nu se pot „înaînta” datele „înapoi” (în timp, cu 1 ciclu!)

- **Unitate de detectare de hazard în etajul ID** trebuie să introducă NoOp (stall) între lw și and
- Este echivalentul hardware al introducerii în program a unei instrucțiuni NoOp după lw

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)

Unitate de detectare de hazard

- Verifică dacă în etajul EX este instrucțiunea lw
- Dacă da, atunci dacă destinația din lw coincide cu una din sursele instrucțiunii din ID, se blochează pipeline-ul în etajele IF și ID (instrucțiunile sunt puse în așteptare, întârziate cu 1 ciclu de ceas)

Descrierea logicii de control din Unitatea de detectare hazard

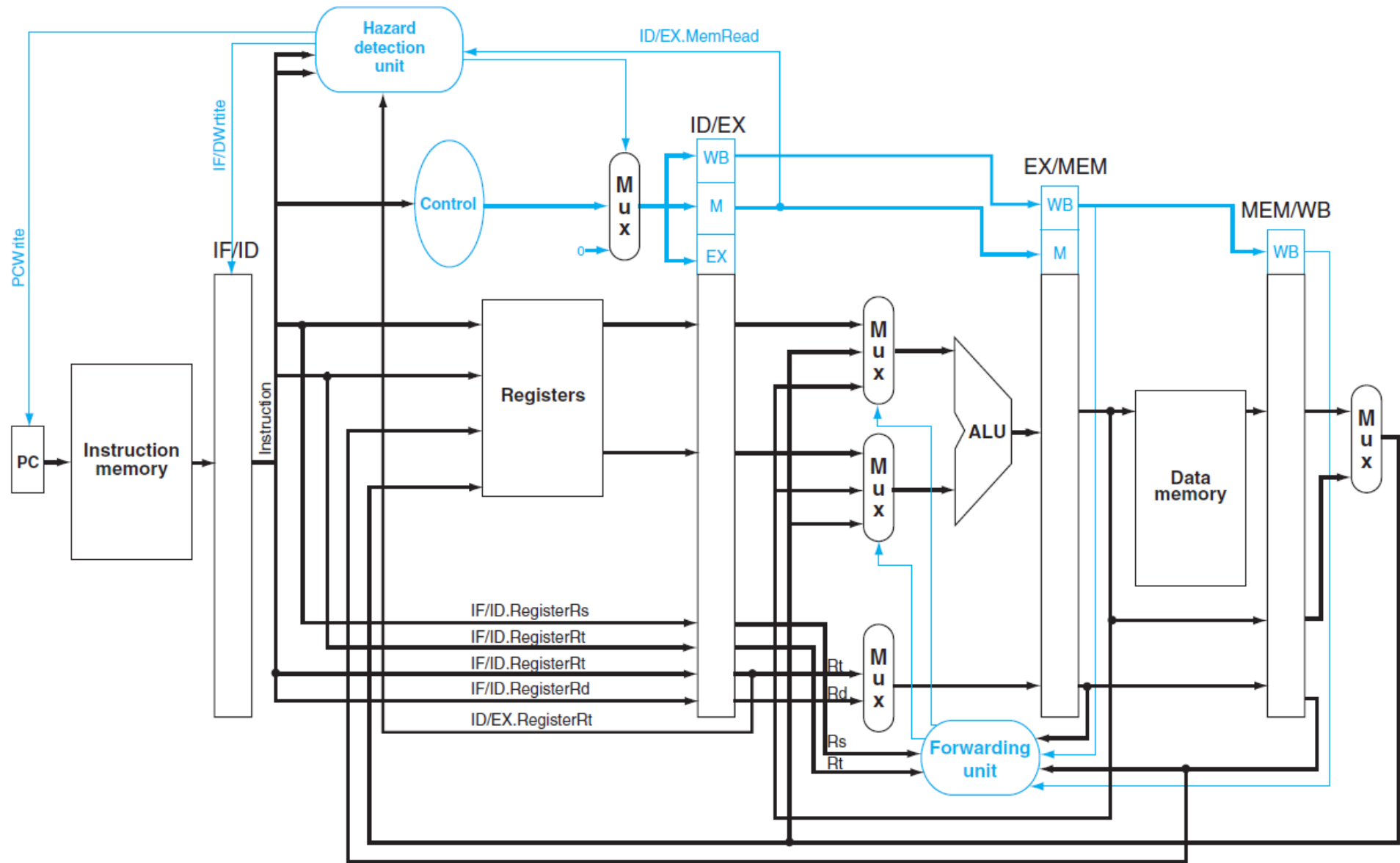
If (ID/EX.MemRead (LW INSTRUCTION?)
and ((ID/EX.RegisterRt = IF/ID.RegisterRs) (destinația în EX= sursă în ID?)
or (ID/EX.RegisterRt = IF/ID.RegisterRt)))
→ stall the pipeline

Un NoOp(stall/bubble) implementat hardware:

1. Blocarea PC și a registrelor pipeline anterioare etajului unde se află LW (adică registrul IF/ID) ca să își păstreze valoarea la următorul front de ceas,
2. Forțarea semnalelor de control din registrul ID/EX pe 0 (se face *FLUSH*) pentru a insera NoOp mai departe în pipeline

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)



Cai de date cu înaintare și detecție de hazard

Proiectarea MIPS pipeline – Hazarduri

Hazardul de date / tehnica de Stall (Așteptare)

Cum se implementează așteptarea/blocarea în caz de hazard (Stalling) ?

1. NoOp se introduce în pipeline în etajul ID prin forțarea semnalelor: **PCWrite = 0** și **IF/IDWrite = 0**, - se păstrează PC și instrucțiunea în IF/ID
2. Se lansează NoOp prin forțare '0' la câmpurile de control EX, MEM și WB din registrul ID/EX
3. Comanda '0' se transmite în etajele următoare de pipeline la fiecare ciclu de ceas și anulează scrierile în blocul de registre și în memoria de date, respectiv posibilitatea de executare branch sau jump. Adică exact ce ar face o instrucțiune NoOp!
 - Practic valoarea 0 trebuia transmisă doar pe RegWrite, MemWrite, Branch și Jump

Proiectarea MIPS pipeline – Hazarduri

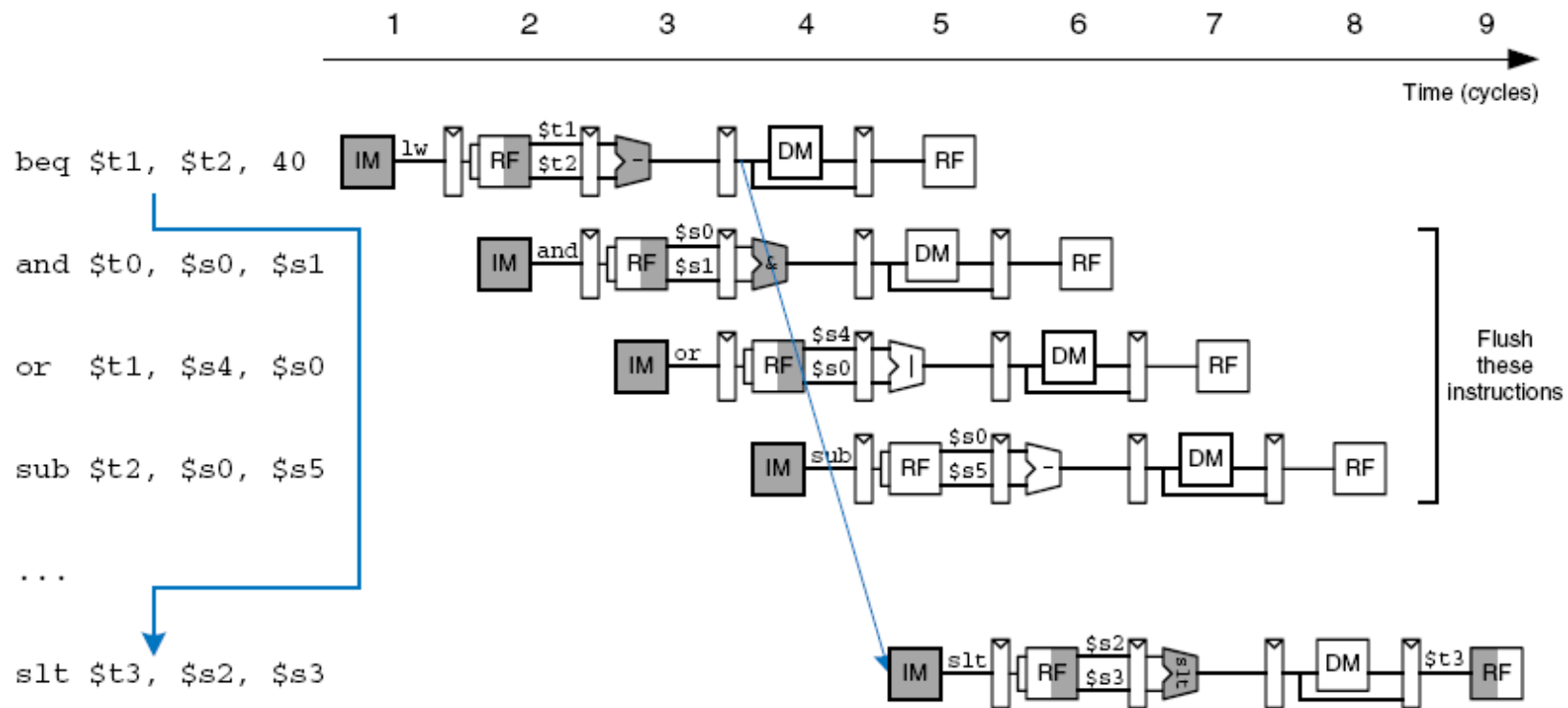
Soluții (în general) pentru hazardul tip load data:

- Hardware introduce NoOp (Stalls) – prezentat anterior
- Software (programatorul) inserează instrucțiuni independente
 - NoOp în lipsa de altceva mai util
- Soluții bazate pe Compilator (Asamblor) pentru evitarea hazardului tip Load
 - Compilatorul detectează dependenta si inserează NoOp
 - Compilatorul caută instrucțiuni independente pentru ocuparea pozițiilor dependente de Load (Load-delay slots)
 - Planificarea software, prin reordonarea instrucțiunilor, poate micșora numărul de hazarduri

Proiectarea MIPS pipeline – Hazarduri

Hazardul de control / Hazarduri de ramificări condiționate (**Branch hazard**)

- Condiția (decizia) de ramificare este evaluată abia în **etajul MEM**
- Următoarele 3 instrucțiuni după Branch sunt deja lansate (în IF, ID, EX)
- Dacă saltul se efectuează trebuie anulat efectul celor 3 instrucțiuni în pipeline



Ex. Condiția Beq este evaluată în ciclul 4, dacă este adevărată atunci

- Adresa țintei se scrie în PC la tranziția în ciclul 5, deci va începe instrucțiunea slt
- Ce se poate face pentru a reduce numărul de instrucțiuni lansate incert?

Proiectarea MIPS pipeline – Hazarduri

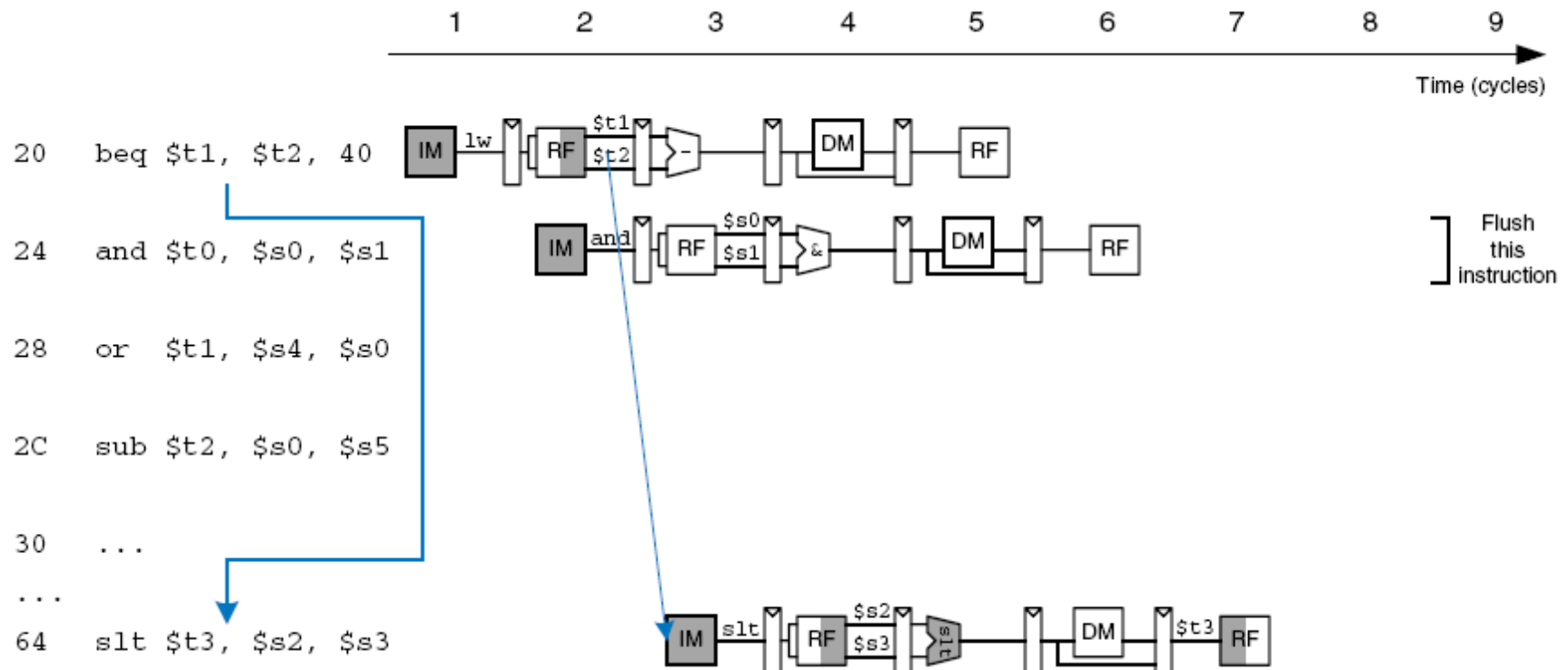
Hazardul de control / Hazarduri de ramificări condiționate (**Branch hazard**)

Reproiectare pipeline: reducerea numărului instrucțiunilor cu lansare incertă la 1

- Dacă tratarea ramificărilor se rezolvă mai aproape de intrare în pipeline, se vor anula mai puține instrucțiuni în caz de efectuare a saltului
- În pipeline MIPS simplu, selecția PC pentru ramificare are loc în etajul MEM
- **Putem plasa calcularea adresei și evaluarea condiției de ramificare în etajul ID →** numai o instrucțiune intră în pipeline după instrucțiunea de ramificare
- Testarea condiției de ramificare în etajul ID implică detectare de hazard și înaintare suplimentară (!); ramificarea poate depinde de o valoare în curs de prelucrare în pipeline
- Înaintarea datelor de la registrele de pipeline EX/MEM sau MEM/WB
- Stall (așteptare) dacă are loc un hazard de date nerezolvabil prin înaintare (ex. un lw înainte de branch, unde destinația este unul dintre registrele de comparat în branch)
- Branch Delay – Întârziere de ramificare numai 1 ciclu de ceas

Proiectarea MIPS pipeline – Hazarduri

Hazardul de control / Hazarduri de ramificări condiționate (**Branch hazard**)

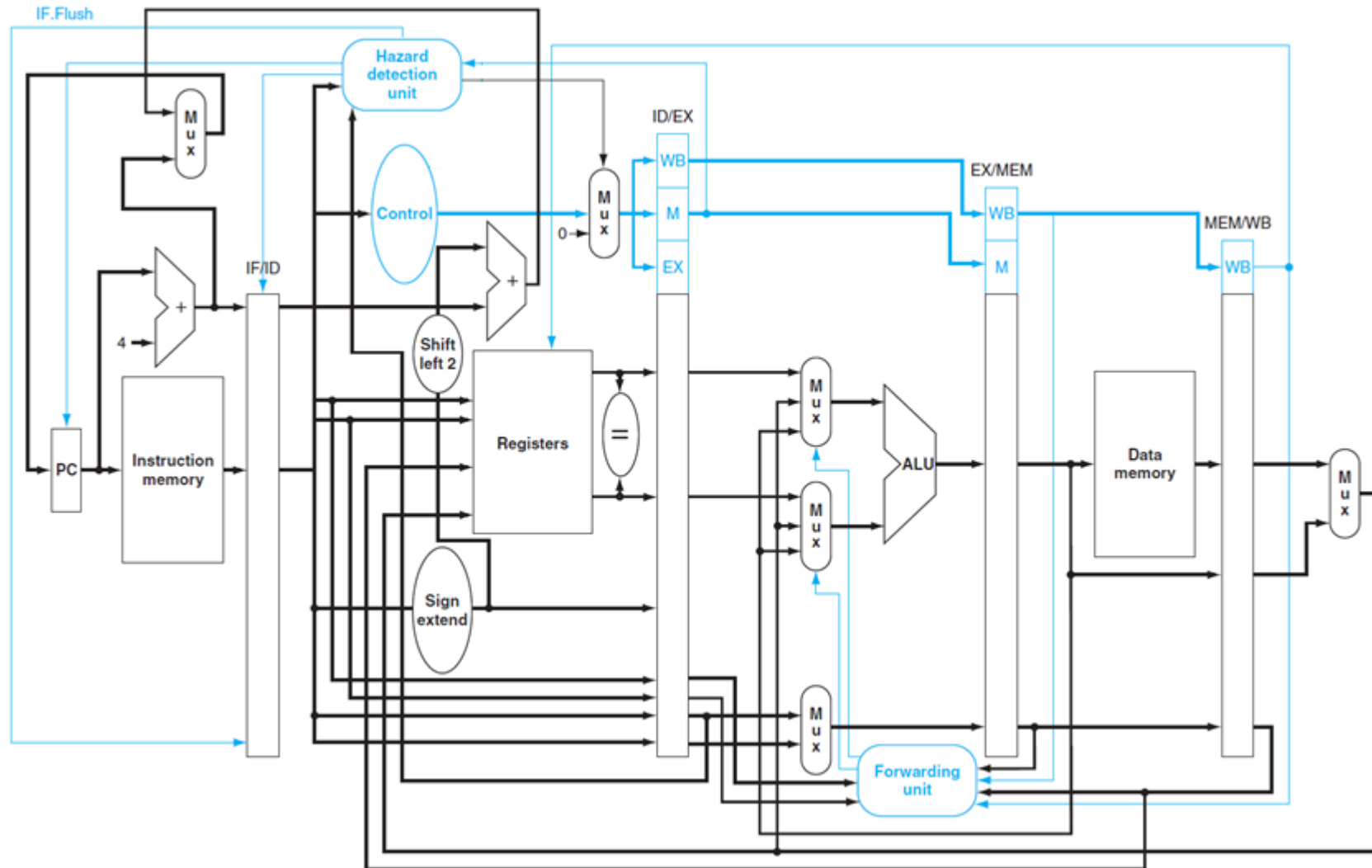


Instrucțiune de ramificare rezolvată în ID; – se anulează numai 1 instrucțiune în caz de salt

- Trebuie să adăugăm hardware pentru anularea (flush) instrucțiunilor care au intrat nejustificat în pipeline (în caz de salt)

Proiectarea MIPS pipeline – Hazarduri

Hazardul de control / Hazarduri de ramificări condiționate (**Branch hazard**)



MIPS; Detecție de hazard LW, Ramificare în ID, Flush pentru ramificare
Înaintarea în ID a operanzilor din instrucțiunea de branch (ramificare) nu este reprezentată
explicit pe schemă

Proiectarea MIPS pipeline – Hazarduri

Tratarea hazardurilor (sumar)

- **Flush (anulare)** – anulează o instrucțiune prin transformare implicită în NoOp, la nivel hardware, prin anularea registrului de pipeline de la etajul următor etajului curent în care a ajuns instrucțiunea
 - anularea se poate face complet (reset registru pipeline), pentru o implementare simplă, sau minimal, prin setarea pe zero doar a acelor semnale care controlează scrierea în blocul de registre și memorie, respectiv cele care controlează logica de salt
 - Flush pentru ramificare – anularea instrucțiunii următoare după salt condiționat – Clear registru IF/ID $\leftarrow 0$; (NoOp, dar fără așteptare)
- **Stall/Bubble/NoOp (așteptare)** - se păstrează conținutul în registrele de pipeline pentru instrucțiuni mai „tinere” (mai aproape de intrare în pipeline) și se așteaptă terminarea instrucțiunilor mai „bătrâne”;
 1. Se blochează scrierea în registrele pipeline precedente, (ex. invalidare scriere)
 2. Se transmite 0 (NoOp) pentru etajul următor celui curent unde e instrucțiunea cu dependență (echivalent cu flush, dar se și așteaptă, vezi pas 1)

Exemplu software NoOp = sll \$0, \$0, 0; (bubble)

Proiectarea MIPS pipeline – Hazarduri

- **Mai multe alternative pentru hazarduri de ramificare (cursurile următoare)**
- **Conceptul de Delayed Branch / Ramificare întârziată, lungime n**
 - Instrucțiunea de salt condiționat se execută (saltul!) după n instrucțiuni următoare
- **MIPS: Întârzierea de o instrucțiune ($n=1$, comparare în ID) permite rezolvarea corectă a saltului condiționat în pipeline cu 5 etaje**
 - Se definește un “branch delay slot” – instrucțiunea de după salt condiționat, care se va executa întotdeauna
 - Sarcina compilatorului de a găsi instrucțiuni utile pentru “branch delay slot”, pentru a evita detecția hazardurilor trebuie ca instrucțiunea utilă să fie selectată dintre cele dinaintea branch
 - În caz nefavorabil – NoOp

Probleme

1. Implementarea anumitor instrucțiuni pe pipeline, similar cu problemele din cursurile anterioare.

2. Diagrama pipeline cu și fără forwarding

```
lw $6, 4000($7)
add $9, $6, $3
or $2, $9, $6
lw $2, 2000($2)
add $3, $9, $2
sw $9, 2000($3)
```

3. CPI per iteratie in diverse conditii; Diagrama pipeline

```
loop: lw $6, 4000($7)
      add $9, $6, $3
      or $5, $9, $6
      lw $2, 2000($5)
      add $3, $9, $2
      subi $5, $5, 12
      sw $9, 2000($3)
      bne $9, $0, loop
```

4.

```
loop: lw $10, X($20)
      lw $11, Y($20)
      subu $10, $10, $11
```


sw Z(\$20), \$10
 addiu \$20, \$20, 4
 subu \$5, \$23, \$20
 bnez \$5, loop
 nop ; 1 delay slot

Diagrama pipeline; Branch ID/ Mem; cu/fara forwarding
 Numar de cicluri pentru 2 iteratii

5. Diagrama pipeline cu si fara forwarding

add \$1, \$5, \$3
 lw \$2, 0(\$1)
 add \$1, \$2, \$3
 sw \$1, 0(\$5)

Fara forwarding

<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>15</u>
IF	ID	EX	MEM	WB										
	IF	ID	ID	ID	EX	MEM	WB							
		IF	IF	IF	ID	ID	ID	EX	MEM	WB				
					IF	IF	IF	ID	ID	ID	EX	MEM	WB	

Cu forwarding

<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>15</u>
IF	ID	EX	(F1)MEM	WB										
	IF	ID	(F1)EX	MEM	(F2)WB									
		IF	ID	ID	(F2)EX	(F3)MEM	WB							
			IF	IF	ID	(F3)EX	MEM	WB						

Referințe

- [1] D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 3rd edition, ed. Morgan–Kaufmann, 2005. Capitolul 6.
- [2] MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
- [3] MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.
- [4] Material suport (**slide-urile cu executia unor secvente de program, slide 41-50, slide 66-75**) http://www.cs.fsu.edu/~zwang/files/cda3101/Fall2017/Lecture8_cda3101.pdf