Arhitectura Calculatoarelor

Curs 10: Pipeline avansat: Planificarea dinamică a execuției

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

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

Pipeline – Clasificarea hazarduri de date

- Considerăm două instrucțiuni **Inst1** și **Inst2**;
- **Inst1** este înainte de **Inst2,** se execută în mod pipeline.

1. **RAW** (Read After Write), dependență reală – discutată la pipeline clasic

   Inst1 **R2 ← R1 + R3**
   Inst2 **R6 ← R2 + R3**

   - **Inst2** poate citi (incorect) o valoare veche (R2) înainte de scrierea rezultatului actual în **Inst1**
   - Hazardul cel mai comun → soluția: înaintare sau așteptare

2. **WAW** (Write After Write), dependență de ieșire (de nume) – apare la pipeline avansat

   Inst1 **R2 ← R1 x R3**
   Inst2 **R2 ← R4 + R7**

   - **Inst2** scrie rezultatul în R2, înainte de **Inst1**
   - Nu se respecta ordinea scrierii, în R2 se va păstra valoarea scrisa de **Inst1**, nu de **Inst2**.
   - Acest hazard apare numai în pipeline-uri care permit scrierea rezultatului în mai multe etaje, sau permit rularea unei instrucțiuni chiar dacă instrucțiunea precedentă este în stare de așteptare (stall)
     - WAW nu are loc în pipeline MIPS cu 5 etaje, fiindcă scrierea (în regiștri) se efectuează numai în etajul 5 – WB, indiferent de instrucțiune

Pipeline – Clasificarea hazarduri de date

3. WAR (Write After Read), anti-dependență (de nume) – apare la pipeline avansat

Inst1 **R1 ← R2 x R3**
Inst2 **R3 ← R4 + R5**

- **Inst2** scrie in R3 (destinație) înainte de citirea R3 (sursa) de către **Inst1**, astfel **Inst1** citește o valoare eronată
- Nu are loc în MIPS pipeline cu 5 etaje, fiindcă citirea (din regiștri) are loc numai în etajul 2 (ID), iar scrierea (în regiștri) are loc numai in etajul 5 (WB), indiferent de instrucțiune
- WAR poate sa aibă loc dacă unele instrucțiuni scriu rezultatul în etaje apropiate de intrare în pipeline, iar altele citesc operandul sursă mai târziu, în etaje apropiate de ieșirea din pipeline

Urmează în curs:

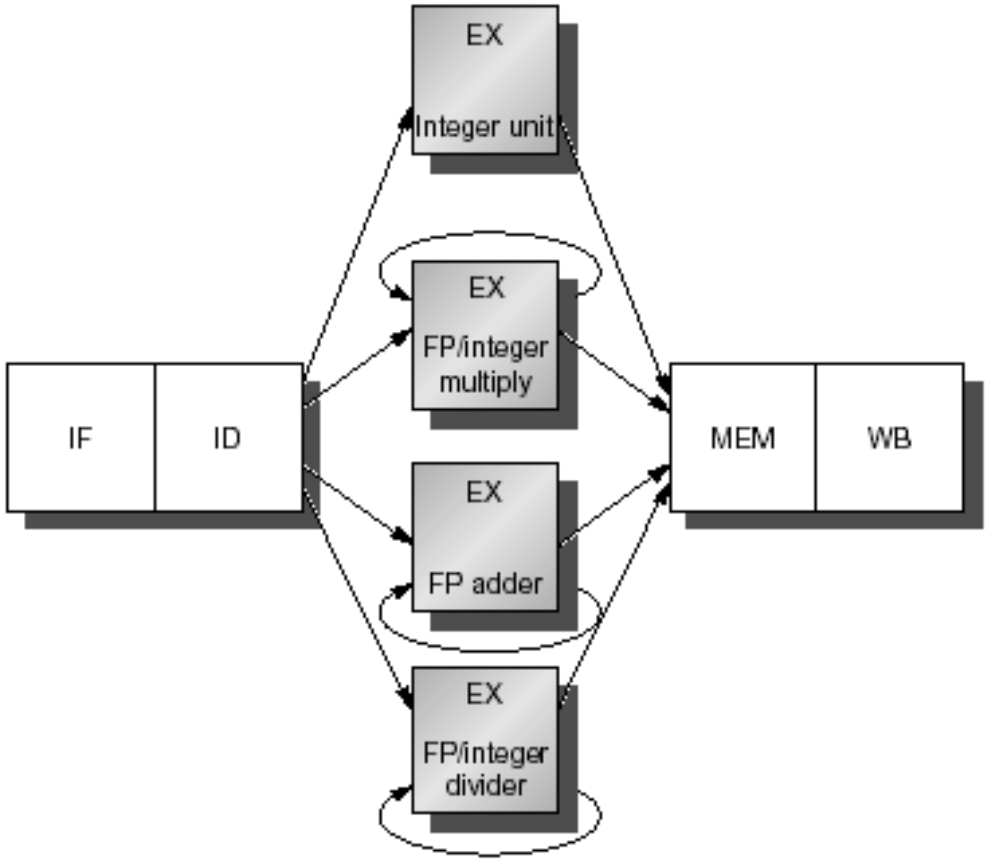
- Arhitecturi de tip pipeline complexe (**multi-ciclu sau pipeline în etajul EX**), cu soluții pentru tratarea/evitarea hazardurilor
- Tehnologii și concepte care se regăsesc sub diferite forme în procesoarele moderne
- Se urmează în principal descrierea din [1] care poate fi consultată opțional

Obs. 1. Instrucțiunile folosite pentru exemple vor fi pentru numere reprezentate în virgulă mobilă (VM), dublă precizie => MULD – multiply Double, ADDD, DIVD, LD (Load…) SD (Store…)

2. Registrele întregi vor fi Rx (x=0,1,2…) iar cele de virgulă mobilă Fx (x=0,1,2…)

Pipeline avansat: multi-ciclu de lungime variabilă

- MIPS pipeline extins: etajul EX cu unități funcționale (UF) pentru operații de tip virgulă mobilă (VM)
- Operațiile în virgulă mobilă au durate de execuție diferite => necesită multi-ciclu pentru a evita un ciclu lung de ceas
- Durata de execuție (latența) se specifică prin numărul de perioade de ceas



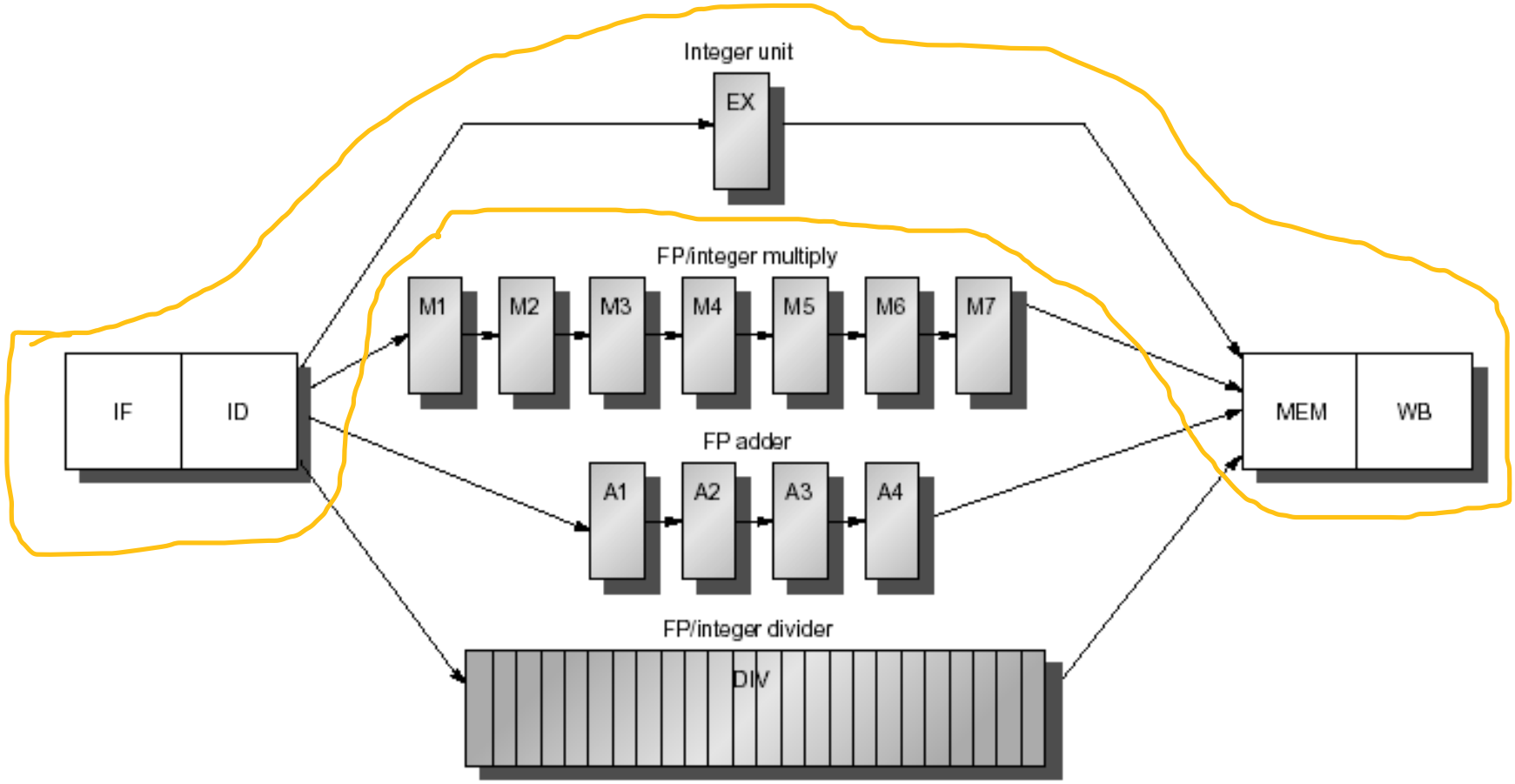
MIPS pipeline cu 4 UF în etajul EX: una pentru întregi și 3 VM

Pipeline avansat: multi-ciclu de lungime variabilă

- EX Integer pentru: adrese load, store; operații întregi ALU; condiții pentru ramificări
- EX FP/Integer multiply - înmulțire, EX FP Adder - adunare, EX FP/Integer divider - împărțire
- Cel mult o instrucțiune se emite într-un ciclu – **procesor scalar**
- Toate instrucțiunile trec prin etajele standard IF, ID, EX, MEM, WB
- Operațiile VM se execută în mai multe cicluri în EX
- După EX, prin MEM și WB se termină execuția
- Deoarece UF VM sunt fără pipeline: nu se poate lansa o instrucțiune nouă până când instrucțiunea precedentă (de același tip! Hazard structural) nu părăsește EX
- Daca o instrucțiune nu poate intra în EX, toate instrucțiunile următoare vor aștepta (stall)

…Rezultă necesitatea folosirii conceptului de **pipeline și pentru unitățile funcționale din etajul EX**!

Pipeline avansat: pipeline în pipeline (în fostul EX)



Pipeline în pipeline: UF VM din EX cu pipeline (mai puțin DIV!), pot fi în curs de execuție mai multe operații VM de același tip

Presupuneri:

- Înmulțitorul și sumatorul VM au pipeline de 7 și 4 etaje
- UF de Împărțire nu are pipeline, durata de execuție 24 de perioade

Pipeline avansat: pipeline în pipeline (EX)

- Structura pipeline impune registre de pipeline adiționale pentru etajele interne de pipeline din etajul referit anterior ca EX (ex. A1/A2, A2/A3, A3/A4).
- Registrul ID/EX trebuie adaptat pentru conectarea ID la EX, DIV, M1, si A1
- Înaintarea datelor (forwarding) se face similar ca la pipeline clasic
  - se compară dacă registrul de destinație în oricare dintre registrele EX/MEM, A4/MEM, M7/MEM, D/MEM, sau MEM/WB este sursă pentru o instrucțiune VM
  - daca da, MUX corespunzător selectat pentru înaintarea datelor
- Întârzierea între instrucțiuni după emiterea unei operații VM și utilizarea rezultatului fără așteptare (RAW) este numărul de etaje minus 1
- Ex. între MULD **F1**, F3, F4 și ADDD F5, **F1**, F6 sunt necesare 6 cicluri de întârziere (sau alte 6 instrucțiuni independente)

Probleme în pipeline-uri cu întârzieri mari

- UF de împărțire este fără pipeline, poate avea loc hazard structural;
  - Trebuie detectat și se va întârzia emiterea tuturor instrucțiunilor
- Instrucțiunile au durate de execuție variabilă, numărul scrierilor necesare în registre destinație într-un ciclu poate fi mai mare de 1 - necesită arbitraj
- Sunt posibile hazarduri WAW, fiindcă instrucțiunile nu ajung în ordine la etajul WB
- Hazarduri WAR nu sunt posibile, citirile din regiștri numai în etajul ID
- Instrucțiunile se pot termina într-o ordine diferită față de lansare, probleme cu excepții
- Datorita duratei mari a operațiilor, întârzierile de hazard RAW vor fi mai frecvente
- Soluția în procesoare moderne => planificare dinamică

Planificarea dinamică a execuției

- **schemă hardware de exploatarea paralelismul instrucțiunilor**
- De ce prin hardware, în timpul rulării?
  1. Nu toate dependențele reale pot fi depistate în timpul compilării
  2. Codul nu depind de implementarea fizică (aceeași ISA poate avea implementări multiple)
  3. Compilator mai simplu

Ideea de bază: Sa permitem rularea instrucțiunilor care urmează după un „stall”

DIVD F0, F2, F4

ADDD F10, F0, F8

SUBD F12, F8, F14

- **Execuția Out-of-order:** dacă o instrucțiune nu poate fi executată, instrucțiunile următoare nu vor fi blocate (nu stall)

Planificarea dinamică a execuției: Tomasulo

Algoritmul Tomasulo - algoritm de planificare dinamică îmbunătățită:

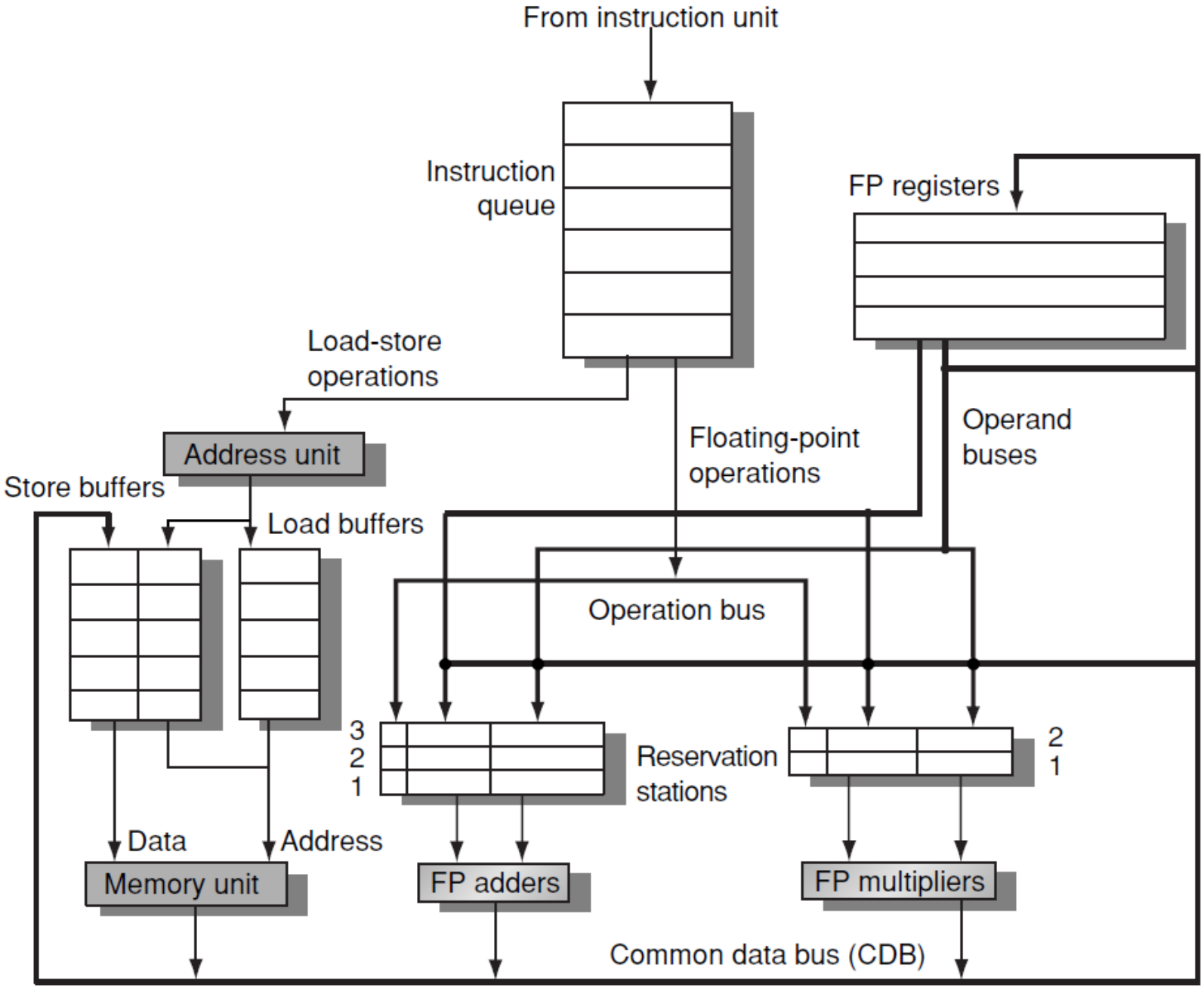
- Pentru IBM 360/91 (1966)
- Scop: Planificarea dinamică performantă a execuției pentru a elimina nevoia unor compilatoare speciale
- De ce este relevant?
  - Dezvoltat în Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604, …
  - Conceptele introduse sunt prezente în procesoarele actuale, cu diferite modificări

Planificarea dinamică a execuției: Tomasulo

Algoritmul Tomasulo - caracteristici

- Comandă și registre distribuite la unitățile funcționale
- Registre buffer pentru unități funcționale – “reservation stations” (RS); păstrează operanzii instrucțiunilor care așteaptă pentru lansare (issue)
  - In RS se aduc operanzii imediat când sunt disponibili, se elimină necesitatea de a citi operanzi numai din Blocul de registre
  - Instrucțiunile în așteptare desemnează UF care vor produce sursele lor
  - Daca au loc scrieri succesive într-un registru, numai ultima va actualiza registrul
- Registrele în instrucțiuni sunt înlocuite implicit de valori sau indicatori la RS; conceptul se numește Register Renaming (redenumirea registrelor)
  - Înlătură hazardurile WAR, WAW
  - Sunt mai multe stații de rezervare decât registre, astfel mai multe posibilități de optimizare decât pentru compilator
- Rezultatele sunt emise (difuzate) prin Common Data Bus (CDB)
- Load și Stores tratate ca unități funcționale cu reservation stations
- Magistrala normală de date constă din data + destinație (“go to” bus)
- Common Data Bus (CDB): data + sursa (“come from” bus)
  - 64 biți de date + 4 biți adresa UF sursă
  - Destinația preia datele de pe CDB când apare adresa UF (sursa așteptată)
  - Astfel se difuzează datele; mai multe destinații sunt servite simultan

Planificarea dinamică a execuției: Tomasulo



MIPS : Unitate cu operații FP folosind algoritmul Tomasulo

Planificarea dinamică a execuției: Tomasulo

- Instrucțiunile VM sunt depuse de unitatea IF într-o coadă de așteptare
- Stațiile de rezervare includ operația, operanzii și informația folosită pentru detectarea și rezolvarea hazardurilor
- Bufere de LOAD păstrează adresele pentru citirile din memorie
- Bufere de STORE păstrează adresele de memorie pentru instrucțiuni care așteaptă operanzii de scris in memorie
- Toate rezultatele de la UF VM sau de la unitatea LOAD se difuzează prin CDB la Blocul de registre VM, la Stațiile de Rezervare și la bufere STORE
- Sumatoarele VM executa adunări și scăderi, înmulțitoarele VM execută înmulțire și împărțire

Planificarea dinamică a execuției: Tomasulo

Algoritmul Tomasulo: 3 faze

1. **Issue – Lansarea** unei instrucțiuni din coada VM.
   - Dacă este o operație VM, se lansează dacă există o **SR** (Statie de Rezervare) corespunzătoare și se trimit operanzii la SR dacă se găsesc în registre
   - Daca operația este load sau store, poate fi lansată daca există un bufer disponibil
   - Daca nu există SR sau bufer gol → hazard structural și instrucțiunea așteaptă până la eliberarea SR sau a unui bufer
   - Acest pas realizează și redenumirea regiștrilor

2. **Executie** – Dacă operanzii nu sunt disponibili, se monitorizează CDB
   - Când un operand corespunzător apare pe CDB, se înregistrează în stația de rezervare potrivită (cea unde el este așteptat de altă instrucțiune)
   - Când ambii operanzi sunt disponibili, se execută operația
   - Acest pas testează și rezolvă prin așteptare hazardurile RAW

3. **Write result** – terminarea execuției (WB)
   - Când rezultatul este disponibil, se difuzează prin CDB; SR, Registrele VM sau buferele de STORE în așteptare pot prelua operandul așteptat
   - Stația de rezervare care a emis rezultatul devine disponibilă.

Planificarea dinamică a execuției: Tomasulo

Avantajele algoritmului Tomasulo:

1. Logica de detecție hazard este distribuită
2. Eliminarea așteptărilor pentru hazardurile WAW și WAR
3. Reducerea hazardurilor structurale – prin mai multe SR-uri decât UF

- Primul avantaj este datorită SR-urilor distribuite și folosirea CDB
  - Daca mai multe instrucțiuni așteaptă operand de la aceeași SR, îl vor putea citi de pe CDB în același timp.
  - In Scoreboard rezultatele sunt scrise în registre, abia apoi pot fi citite de instrucțiunile în așteptare
- Hazardurile WAW și WAR se elimină prin redenumirea registrelor, folosind stațiile de rezervare și plasarea operanzilor în SR imediat după producerea lor.

În ansamblu Tomasulo implică: In-order issue; out-of-order execute & completion (commit)

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul (se presupune că nu există alte hazarduri cu instrucțiuni anterioare secvenței):

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**							
2	**LD F2 45+R3**							
3	**MULD F0 F2 F4**							
4	**SUBD F8 F6 F2**							
5	**DIVD F10 F0 F6**							
6	**ADDD F6 F8 F2**							

- stE: start Execution
- eE: end Execution
- str, raw: hazard type
- cdb: cdb conflict
- numărul din coloana hazard arată numărul instrucțiunii implicate

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1					
2	**LD F2 45+R3**							
3	**MULD F0 F2 F4**							
4	**SUBD F8 F6 F2**							
5	**DIVD F10 F0 F6**							
6	**ADDD F6 F8 F2**							

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2			
2	**LD F2 45+R3**		2					
3	**MULD F0 F2 F4**							
4	**SUBD F8 F6 F2**							
5	**DIVD F10 F0 F6**							
6	**ADDD F6 F8 F2**							

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		
2	**LD F2 45+R3**		2		3			
3	**MULD F0 F2 F4**		3					
4	**SUBD F8 F6 F2**							
5	**DIVD F10 F0 F6**							
6	**ADDD F6 F8 F2**							

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		
3	**MULD F0 F2 F4**		3	**2**				
4	**SUBD F8 F6 F2**		4					
5	**DIVD F10 F0 F6**							
6	**ADDD F6 F8 F2**							

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	**2**				
4	**SUBD F8 F6 F2**		4	**2**				
5	**DIVD F10 F0 F6**		5					
6	**ADDD F6 F8 F2**							

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	2	6			
4	**SUBD F8 F6 F2**		4	2	6			
5	**DIVD F10 F0 F6**		5	**3**				
6	**ADDD F6 F8 F2**		6					

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	2	6			
4	**SUBD F8 F6 F2**		4	2	6	7		8
5	**DIVD F10 F0 F6**		5	**3**				
6	**ADDD F6 F8 F2**		6	**4**				

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	2	6			
4	**SUBD F8 F6 F2**		4	2	6	7		8
5	**DIVD F10 F0 F6**		5	**3**				
6	**ADDD F6 F8 F2**		6	4	9	10		11

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	2	6	15		16
4	**SUBD F8 F6 F2**		4	2	6	7		8
5	**DIVD F10 F0 F6**		5	**3**				
6	**ADDD F6 F8 F2**		6	4	9	10		11

Planificarea dinamică a execuției: Tomasulo

Exemplu – Aplicarea metodei Tomasulo**:**

Constrângeri:

- Durate: VM Add – 2, Multiply – 10, Divide – 40, Integer Ex – 1 perioade
- **Etaj** **Hazardul testat**
  Issue (Iss): Structural
  Execute: RAW
  Write result (Wr): CDB conflict

Pentru o secvență de instrucțiuni se completează tabelul, pentru fiecare perioadă de ceas:

#	**Instrucțiune**	str	**Iss**	raw	**stE**	**eE**	cdb	**Wr**
1	**LD F6 34+R2**		1		2	3		4
2	**LD F2 45+R3**		2		3	4		5
3	**MULD F0 F2 F4**		3	2	6	15		16
4	**SUBD F8 F6 F2**		4	2	6	7		8
5	**DIVD F10 F0 F6**		5	3	17	56		57
6	**ADDD F6 F8 F2**		6	4	9	10		11

Planificarea dinamică a execuției: Tomasulo

Tomasulo – dezavantaje

- Complex
- Multe memorări asociative (CDB) la viteze mari
- Performanță limitată de Common Data Bus
- CDB-uri multiple → mai multe logică în unitățile funcționale pentru memorări paralele.

Load/store disambiguation – Decuplarea Load/Store; prin buferele de adresă

- Un element important pentru funcționarea algoritmului Tomasulo
- Store bufer memorează adresele și datele de memorat, pentru ca UCP să nu aștepte pentru scrieri în memorie
- Load și Store pot fi executate în ordinea schimbată față de apariția lor în program, dacă accesează adrese diferite
- La fiecare Load sunt examinate adresele în buferul de Store
- Daca o adresă de Store corespunde cu adresa Load, se așteaptă până când buferul Store primește valoarea de scris în memorie (se va scrie în memorie).

Planificarea dinamică a execuției: Tomasulo

Sumar

- Stațiile de Rezervare: redenumire la un set mai mare de regiștri + bufer pentru operanzi sursă
  - Previne ca numărul redus al regiștrilor (uz general) să limiteze performanța
  - Elimină hazardurile WAR, WAW
  - Permite despachetarea buclelor în HW
- Contribuții cheie ale algoritmului Tomasulo pentru preluate în calculatoare moderne
  - Dynamic scheduling – Planificare dinamică
  - Register renaming – Redenumirea regiștrilor
  - Load/store disambiguation – Decuplarea acceselor la memorie Load/Store
- Schema Tomasulo este aplicată pentru arhitecturi pipeline pentru care planificarea instrucțiunilor de către compilator este dificilă, sau numărul regiștrilor este redus
  - Avantajele Tomasulo față de planificarea de către compilator pentru un pipeline cu o singură instrucțiune lansată (**procesor scalar**), sunt probabil mai mici decât costul implementării
  - Dar, datorită lansării simultane a mai multor instrucțiuni (**superscalar**) și a nevoii de îmbunătățire a performanței pentru coduri greu de planificat static, contribuțiile cheie ale algoritmului Tomasulo sunt aplicate în calculatoarele moderne.

Paralelism la nivel de instrucțiuni / bucle

- **Instruction Level Parallelism ILP**: suprapunerea execuției instrucțiunilor nedependente
- **Paralelism la nivelul buclelor** – o oportunitate pentru exploatarea ILP
  - O metoda uzuală pentru creșterea paralelismului disponibil – exploatarea paralelismului între iterațiile unei bucle
  - Acest tip de paralelism deseori se numește **paralelism la nivelul buclelor**
- **Planificare de bază în pipeline și despachetarea buclelor**
  - Presupunem pipeline MIPS standard cu evaluarea ramificării în etajul ID (vezi curs 9, beq), „branch delay” de 1 ciclu
  - Presupunem în etajul EX ca sunt UF-uri cu pipeline, replicate (de câte ori este necesar) astfel orice operație poate fi lansată la fiecare ciclu de ceas, fără hazarduri structurale

Întârzierile operațiilor VM folosite

Instrucțiune care produce un rezultat	Instrucțiune care folosește un rezultat	Latența în cicluri de ceas
FP ALU op	Altă FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Integer op	Integer op	0

- Latența arată numărul perioadelor de ceas pentru evitarea așteptărilor

Paralelism la nivel de instrucțiuni / bucle

Exemplu: adunarea aceleiași valori la elementele unui șir din memorie

```
Loop:
 LD      F0, 0(R1)      F0 = vector element
 ADDD    F4, F0, F2     add scalar in F2
 SD      0(R1), F4      store result
 SUBI    R1, R1, #8     decrement pointer 8 bytes (DWord)
 BNEZ    R1, Loop       branch R1 != zero
 NOP                    branch delay slot
```

FP Loop: Hazarduri ?

Fără planificare bucla va fi executată astfel:

```
     Loop:
1    LD      F0, 0(R1)      F0 = vector element
2    stall
3    ADDD    F4, F0, F2     add scalar in F2
4    stall
5    stall
6    SD      0(R1), F4      store result
7    SUBI    R1, R1, #8     decrement pointer 8 bytes (DW)
8    BNEZ    R1, Loop       branch R1 != zero
9    stall                  branch delay slot
```

Durata 9 perioade / iterație: Rescriem codul pentru minimizarea întârzierilor.

Bucla VM cu întârzieri (stall)

Paralelism la nivel de instrucțiuni / bucle

Bucla VM reorganizată pentru minimizarea întârzierilor (se reordonează instrucțiunile independente pentru a minimiza așteptarea):

```
1   Loop:
2    LD      F0, 0(R1)
3    stall
4    ADDD F4, F0, F2
5    SUBI   R1, R1, #8
6    BNEZ  R1, Loop      delayed branch
     SD      8(R1), F4   de ce 8(R1) ? Pentru a compensa efectul lui
                         SUBI la accesarea locației de memorie
```

- Întârzierea după BNEZ a fost înlocuita cu SD cu adresa schimbată
- Rezultă 6 perioade / iterație

Paralelism la nivel de instrucțiuni / bucle

Despachetarea buclei de 4 ori (metoda directă)

```
1   Loop:
2   LD      F0, 0(R1)       după LD 1 ciclu stall
3   ADDD    F4, F0, F2      dupa ADDD 2 cicluri stall
4   SD      0(R1), F4       eliminat SUBI & BNEZ
5   LD      F6, -8(R1)
6   ADDD    F8, F6, F2
7   SD      -8(R1), F8      eliminat SUBI & BNEZ
8   LD      F10, -16(R1)
9   ADDD    F12, F10, F2
10  SD      -16(R1), F12    eliminat SUBI & BNEZ
11  LD      F14, -24(R1)
12  ADDD    F16, F14, F2
13  SD      -24(R1), F16
14  SUBI    R1, R1, #32     se scade 4*8
15  BNEZ    R1, LOOP
    NOP
```

- Așteptări după: LD – 1 ciclu, ADDD – 2 cicluri
- 15 + 4 x (1+2) = 27 cicluri, sau 6.8 per iterație
- Presupunem lungimea șirului multiplă de 4 elemente (8 octeți/element);
- Câți regiștri sunt necesari?

  3 vs. 8

- Aceasta versiune despachetată este mai lentă decât versiunea planificată originală

Paralelism la nivel de instrucțiuni / bucle

Bucla despachetată care minimizează așteptările necesită reordonarea instrucțiunilor independente

```
1    Loop:
2     LD     F0, 0(R1)
3     LD     F6, -8(R1)
4     LD     F10, -16(R1)
5     LD     F14, -24(R1)
6     ADDD   F4, F0, F2
7     ADDD   F8, F6, F2
8     ADDD   F12, F10, F2
9     ADDD   F16, F14, F2
10    SD     0(R1), F4
11    SD     -8(R1), F8
12    SD     -16(R1), F12
13    SUBI   R1, R1, #32
14    BNEZ   R1, LOOP
      SD     8(R1), F16      8-32 = -24 efectul
                             SUBI anterior
```

- Presupuneri făcute la relocarea instrucțiunilor?
- OK pentru plasarea SD după SUBI; schimbarea conținutului R1 (corectat prin offset)
- OK pentru mutarea LD înainte de SD: se citesc date corecte?
- 14 cicluri de ceas, adică 3.5 per iterație

- Folosirea registrelor noi este echivalentă funcțional cu „redenumirea” registrelor de la Tomasulo, fiind făcută de către programator sau compilatorul care optimizează

Referințe

[1] Computer Architecture - A Quantitative Approach, 4th edition J.L. Hennessy, D.A. Patterson Elsevier, 2007, si editii recente

[2] CS152 Computer Architecture and Engineering Lecture 14, 15: Designing a Pipeline Processor March 8, 1995 Dave Patterson (patterson@cs) and Shing Kong (shing.kong@eng.sun.com) Slides available on http://http.cs.berkeley.edu/~patterson

[3] CS152: Computer Architecture and Engineering Introduction to Pipelining October 15, 1997 Dave Patterson (http.cs.berkeley.edu/~patterson).

[4] Lectures 2: Review of Pipelines and Caches, Prof. David A. Patterson Computer Science 252 Fall 1996

[5] Computer Organization and Design, 3d edition D.A. Patterson, J.L. Hennessy Elsevier, 2005, FMP 6.14-1, For More Practice Understanding Pipelines by Drawing Them

[6] Introduction to Advanced Pipelining Professor David A. Patterson Computer Science 252, Spring 1998

- CA2e ch4 - Advanced Pipelining and Instruction Level Parallelism
- Lecture 4:Tomasulo Algorithm and Dynamic Branch Prediction Professor David A. Patterson Computer Science 252 Spring 1998