Pipeline - Banda de asamblare

– Curs 11_1 –

Banda de asamblare este o tehnică în care mai multe instrucțiuni sunt executate simultan.

Se utilizează intens în procesoarele moderne:

- ❑ AMD Opteron X4 (Barcelona)
- ❑ procesoarele Intel

Exemplu:

Un ciclu de spălare pentru rufe versiunea non-pipe:

– Introducerea rufelor în mașina de spălat

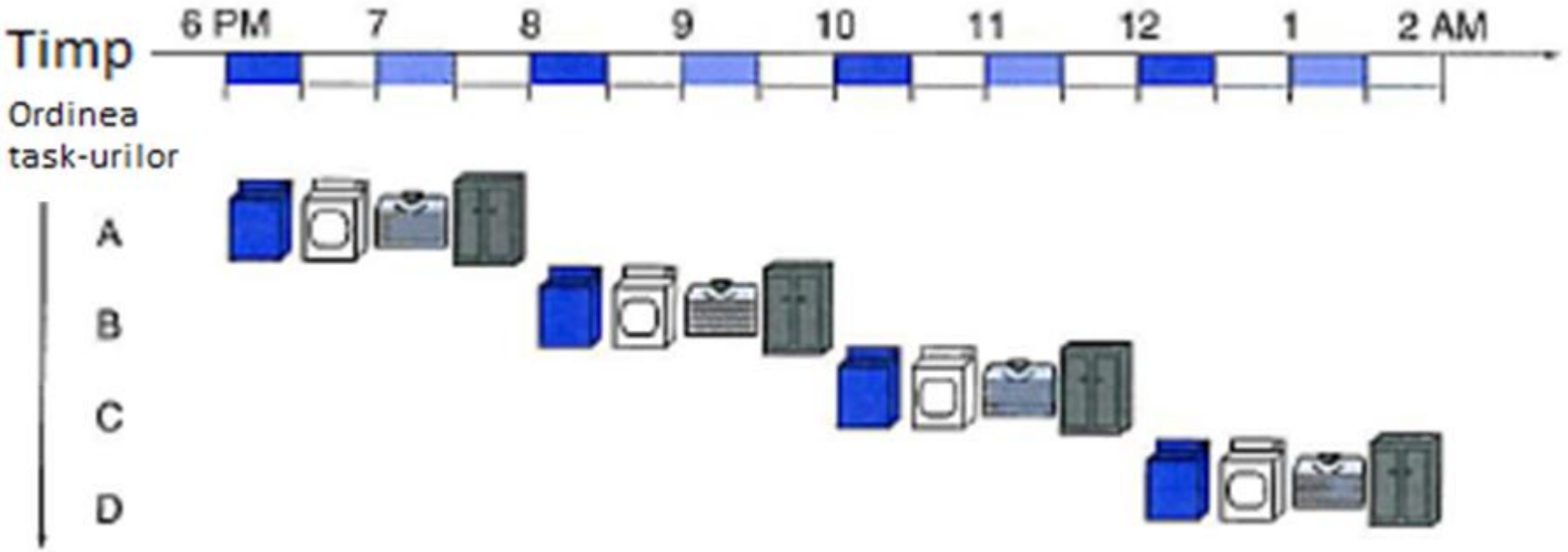
–După terminarea ciclului de spălare, se introduc rufele în uscator

–Rufele uscate se calcă

–Rufele călcate se pun într-un loc pentru utilizarea lor viitoare

–Se repornește cu ciclul de spălare

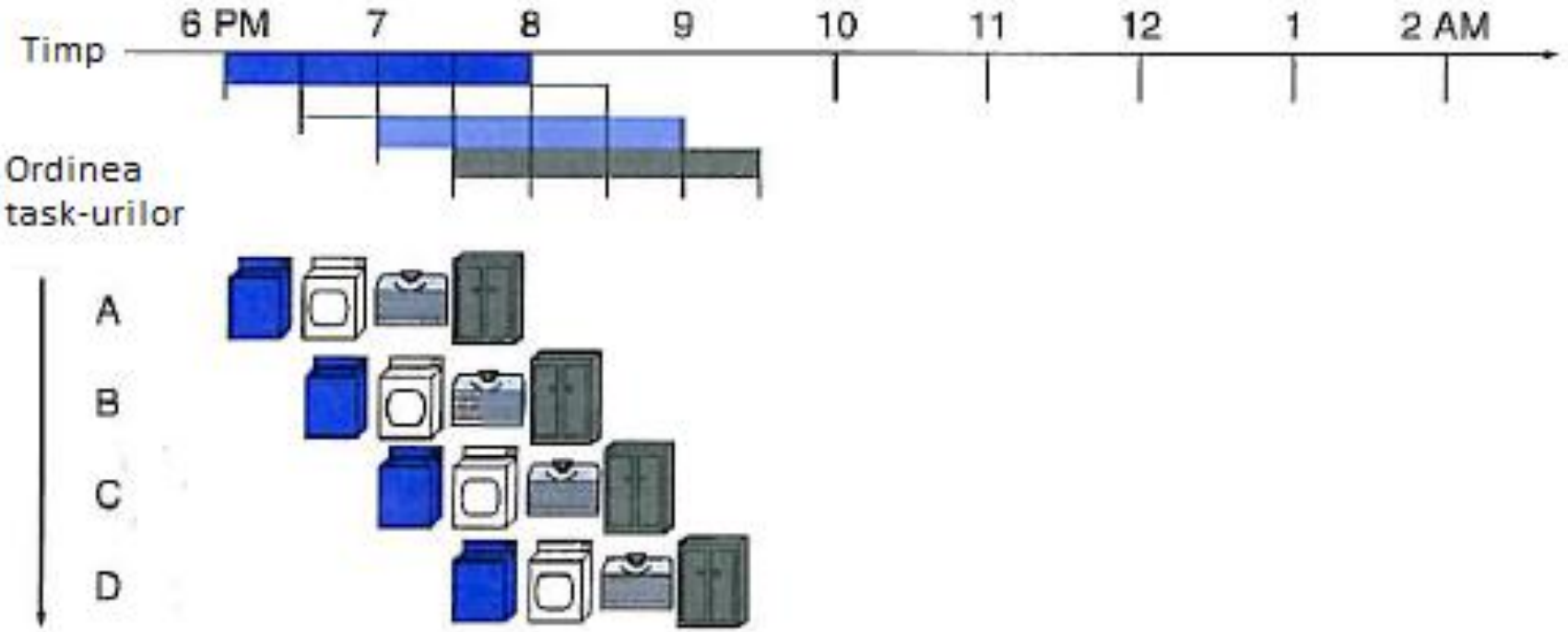
Cât timp este necesar pentru un ciclu de spălare ?



Versiunea bandă de asamblare

- După terminarea primului ciclu de spălare și încărcarea uscatorului cu rufe, se poate iniția un nou ciclu de spălare
- Se scot rufele din uscător și se începe procesul de călcare a lor, se mută rufele spălate în uscător și se pune o nouă încărcătură în mașina de spălat
- Se pun rufele la loc sigur și se continuă procesul

Cât timp durează versiunea pipeline ?



Observații:

- Perioada de ceas alocată fiecărei operații trebuie să fie egală
- Banda de asamblare îmbunătățește performanța/ randamentul (*throughput*) sistemului
- Timpul total pentru executarea tuturor task-urilor este mai mic
- Varianta pipeline este mult mai rapidă decât varianta non-pipe

Banda de asamblare la MIPS

•Pas 1 – Citire instrucțiune din memorie

•Pas 2 – Citirea registrelor cât timp se decodifică instrucțiunea.

•Pas 3 – Execuția operației sau calcularea unei adrese

•Pas 4 – Accesarea unui operand în memoria de date

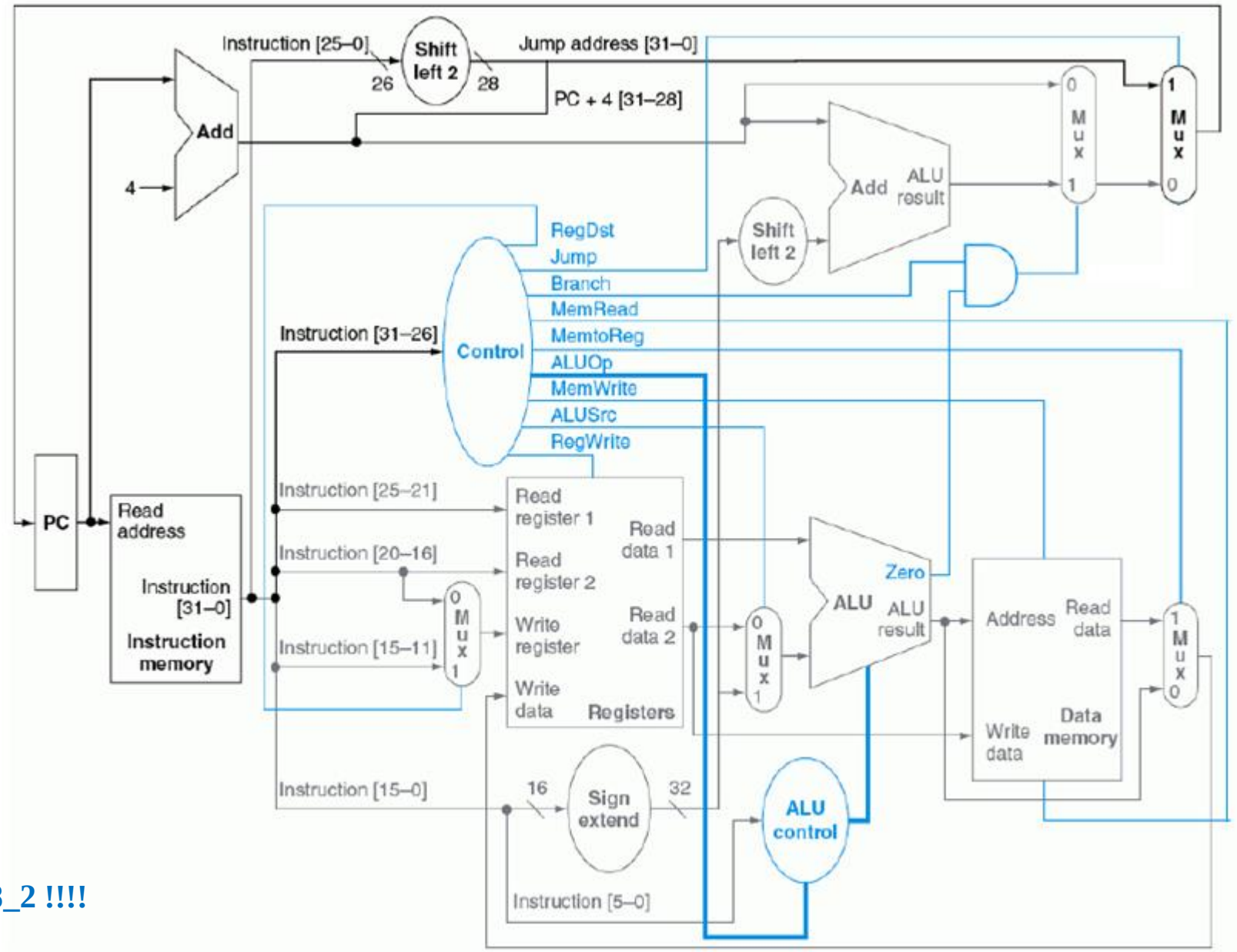
•Pas 5 – Scrierea rezultatului într-un registru

Obs: Din formatul instrucțiunilor MIPS rezultă posibilitatea apariției simultane a citirii și a decodificării.

Pipe-ul va fi discutat având în vedere 8 instrucțiuni: lw (load word), sw (store word), add, sub, AND, OR, slt (setează mai mic decât), beq (salt pe egalitate)

REMEMBER FROM COURSES

8_1 / 8_2 !!!!



Procesorului MIPS care operează într-un singur ciclu de ceas

Exemplu MIPS

Să se compare rezultatele versiunilor non-pipe și pipe pentru:

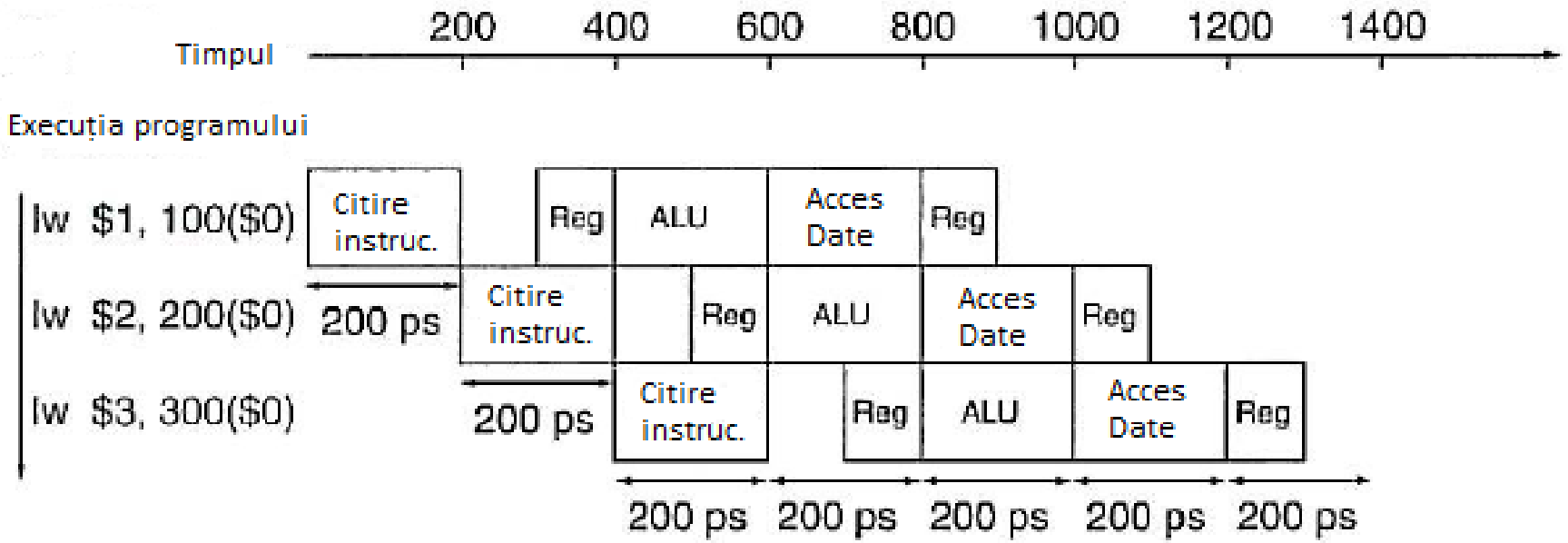
- Acces la memorie 200ps
- Operație ALU 200ps
- Citirea unui registru sau scrierea lui 100ps
- În versiunea single-cycle, fiecare instrucțiune va dura exact un ciclu de ceas.

Timpii pentru versiunea fără bandă de asamblare

Instrucțiunea	Citire instr.	Citire reg.	Operație ALU	Acces date	Scriere reg.	Timp total
lw	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
sw	200 ps	100 ps	200 ps	200 ps		700 ps
Add, sub, AND, OR, slt	200 ps	100 ps	200 ps		100 ps	600 ps
beq	200 ps	100 ps	200 ps			500 ps

Ceasul va trebui să fie suficient de lent pentru a permite execuția oricărei instrucțiuni ⇨ valoarea ceasului va fi de 800 ps

Varianta bandă de asamblare



În cazul ideal creșterea vitezei este egală cu numărul de stagii al benzii de asamblare.

CONCLUZII

•Toate instrucțiunile MIPS au aceeași lungime

•MIPS are doar câteva formate de instrucțiuni care au câmpul – registru sursă – în aceași poziție pentru fiecare instrucțiune.

•Operanzii în memorie apar doar în instrucțiunile sw

•Operanzii trebuie să fie aliniați în memorie

Această simetrie presupune că al doilea stagiu de pipe poate începe citirea registrului în același timp în care hardware-ul determină tipul instrucțiunii citite. Dacă nu ar exista această simetrie stagiul 2 de pipe ar trebui împărțit, rezultând astfel 6 stagii de pipe.

Această restricție presupune utilizarea stagiului de execuție pentru calcularea adresei de memorie și apoi accesarea memoriei în următorul stagiu.

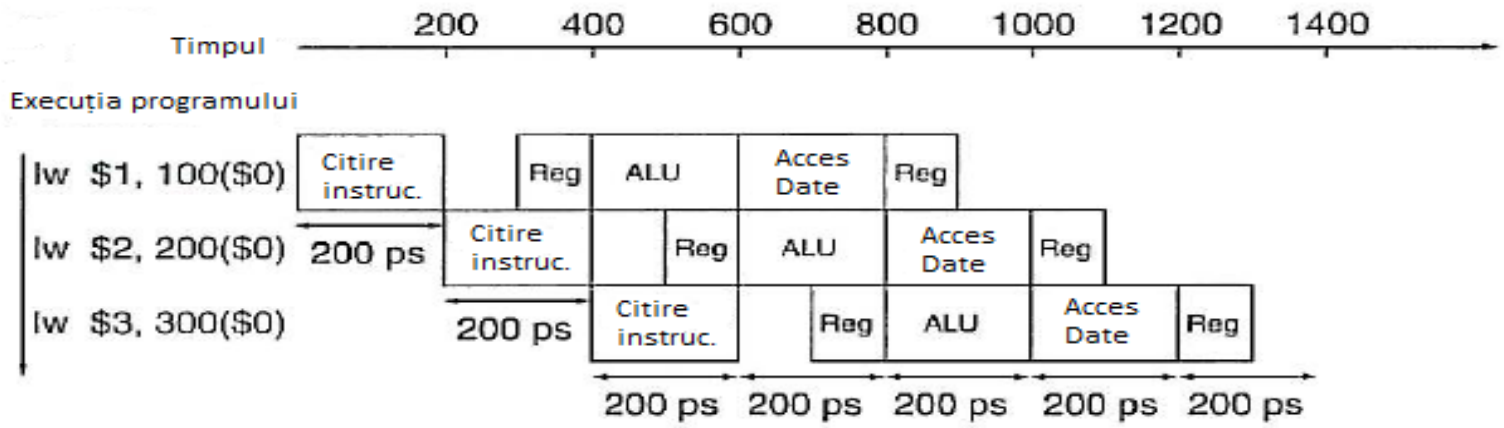
Datele cerute pot fi transferate între procesor și memorie într-un singur stagiu de pipe.

Hazardul în pipeline

Hazard – situația în care într-un pipe, următoarea instrucțiune nu poate fi executată în următorul ciclu de ceas.

I. Hazardul structural – o instrucțiune planificată nu poate fi executată în propriul ciclu de ceas deoarece hardware-ul nu suportă combinația de instrucțiuni planificate pentru execuție.

Unde poate apărea un hazard structural ?



Daca am avea o singura memorie, hazardul structural ar apărea în ciclul 4 când dorim să accesăm datele din memorie, dar și sa extragem o nouă instrucțiune.

Hazardul de date

II. Hazardul de date – instrucțiunea planificată spre execuție nu poate fi executată în ciclul de ceas deoarece datele necesare execuției nu sunt încă disponibile.

add $s0, $t0, $t1

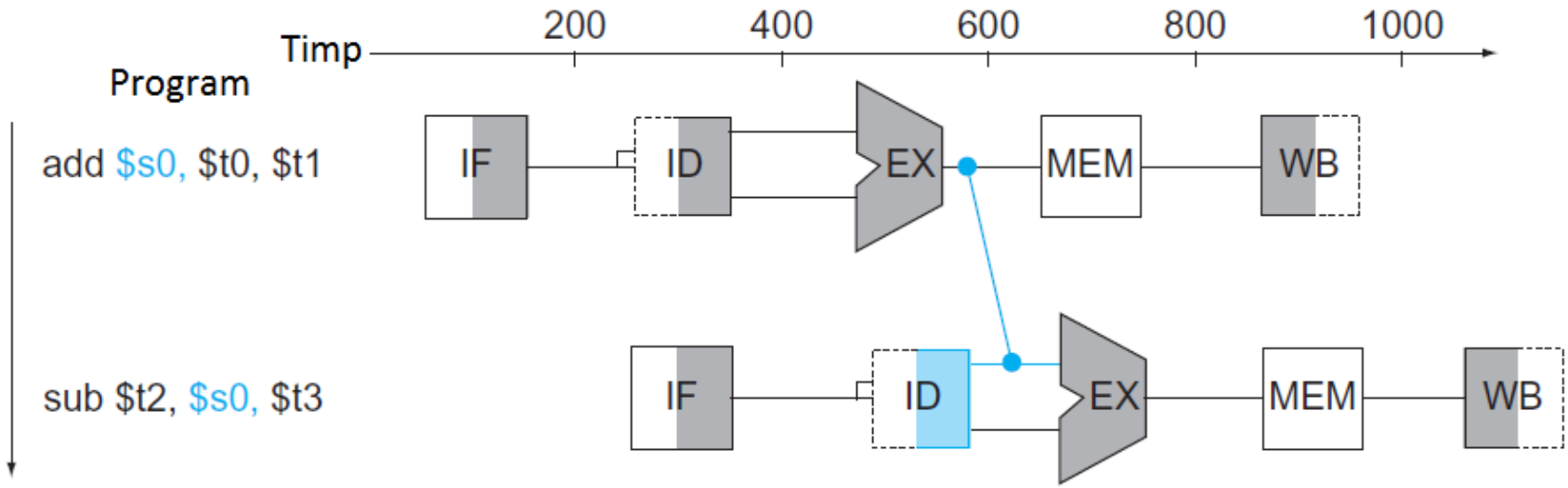
sub $t2, $s0, $t3

(Un client solicită ridicarea rufelor, dar ele nu se regăsesc în dulap. Va trebui timp să căutăm în tot dulapul.)

Rezultatul adunării va fi disponibil abia în ciclul 5 de ceas de unde rezultă că sunt necesari 3 ciclii de ceas de așteptare. O soluție ar fi să tratăm aceste situații prin intermediul compilatorului, dar aceste situații sunt atât de dese încât nu ne putem baza pe compilator să le rezolve.

forwarding, bypassing

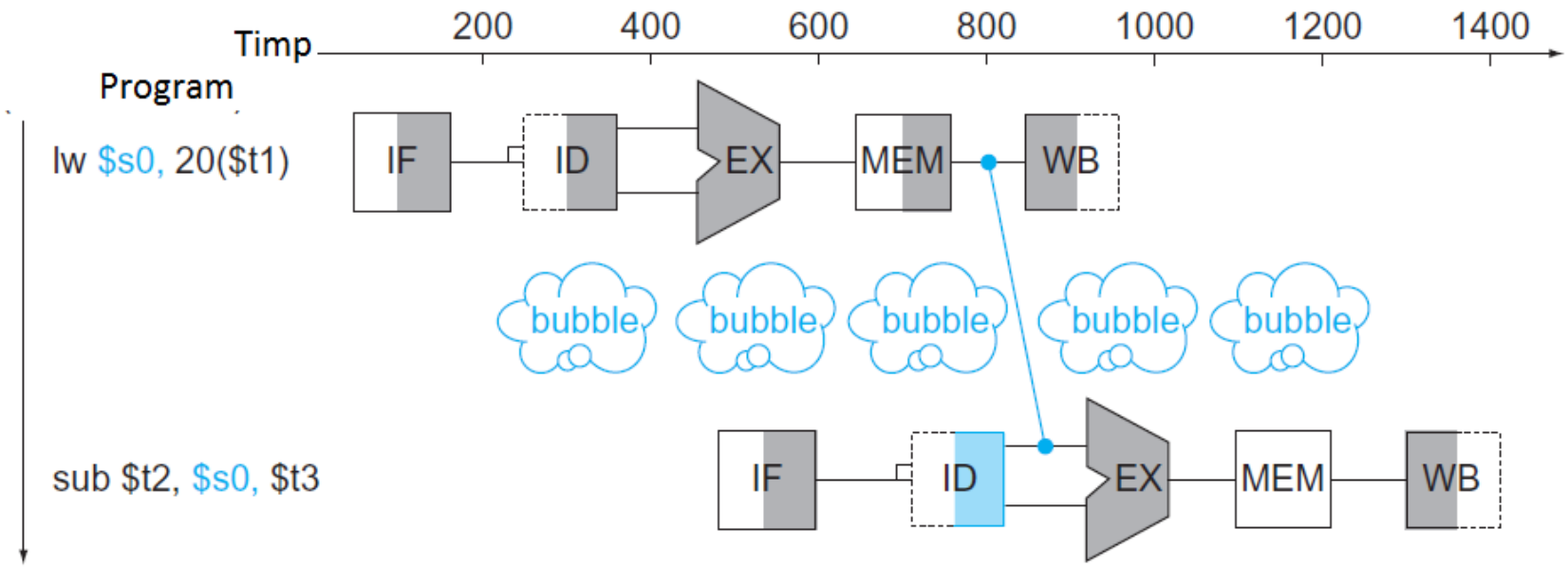
Forwarding



Valoarea registrului $s0 este replasată cu valoarea corecta.
Nu se poate face forward la valoarea produsă de MEM deoarece ar trebui să ne întoarcem în timp.

Notări:
IF = instruction fetch
ID = decodificare instrucțiune/citire register
EX = execuție operație ALU
MEM = acces la memorie
WB = scriere în registre

Hazard de date la încărcare



Valoarea conținută de MEM poate fi sau nu folosită de către EX. Pentru a putea determina dacă valoarea este necesară va trebui să așteptăm un ciclu de ceas, adică introducem un stall. Chiar dacă folosim tehnica FORWARDING, tot va trebui să stăm un ciclu de ceas.

Conceptul de “pipeline stall” este echivalent cu “bubble”.

Exemplu

Se consideră următorul program C:

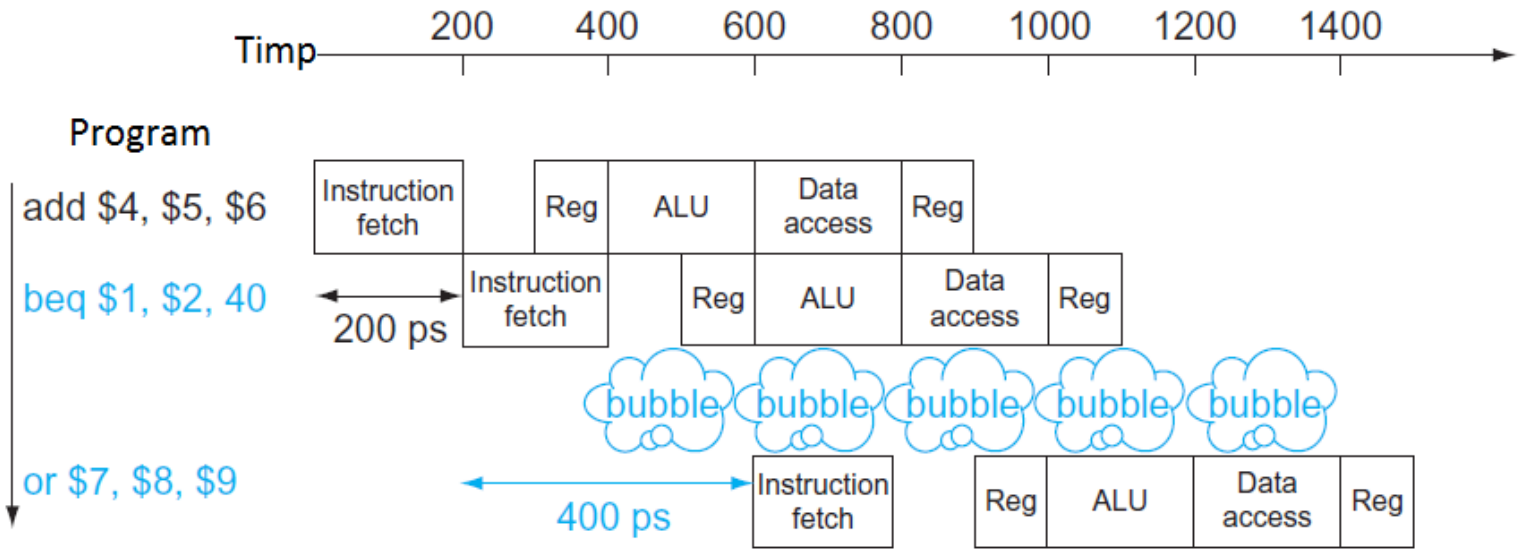
```
A = B + E;

C = B + F;
```

```
lw $t1, 0($t0);

lw $t2, 4($t0);

add $t3, $t1, $t2;

sw $t3, 12($t0);

 lw $t4, 8($t0);

add $t5, $t1, $t4;

sw $t5, 16($t0)
```

Codul MIPS considerând că toate variabilele sunt deja în memorie, adresabile de la adresa de baza $t0

III. Hazardul de control

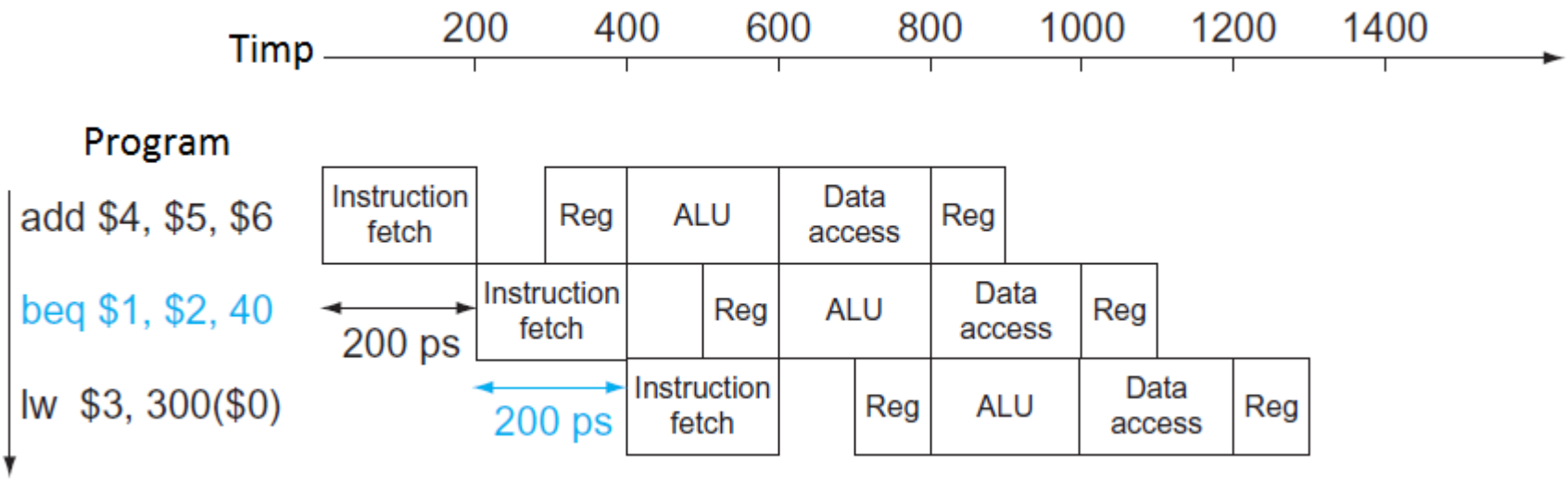
Apar când este necesar să luăm o decizie bazată pe rezultatul unei instrucțiuni cât timp alte instrucțiuni sunt în execuție.



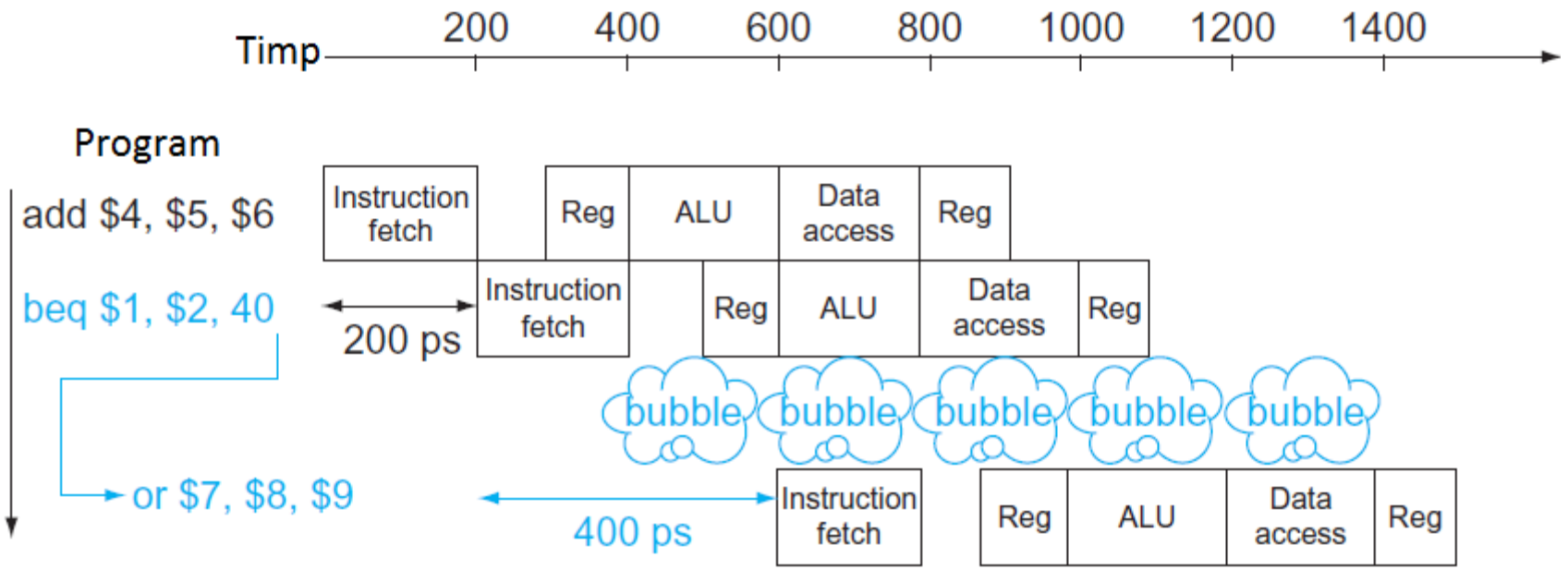
Se face extragere de instrucțiune iar în următorul ciclu de ceas trebuie să avem BEQ. După primirea acestei instrucțiuni începem imediat stall-urile și ele se termină în momentul în care știm adresa de la care trebuie să executăm următorul FETCH.

Predicția

- ✓ Procesoarele utilizează predicția atunci când au instucțiuni de ramificație.
- ✓ Metoda cea mai simplă este să presupunem că nu vom avea ramificații. Doar în cazul apariției lor, se vor lua în considerare – altfel pipe-ul va funcționa la viteza maximă.



Predicția nu este luată în considerare.



Predicția este luată în considerare.
Inserarea de stall-uri simplifică lucrurile cel puțin în ciclul de ceas imediat următor ramificației.

În cazul programelor scrise…

- În cazul ciclurilor de programare avem instrucțiuni care întorc execuția programului la începutul ciclului.

- În acest caz putem prezice că întotdeauna vom considera ramificațiile care ne conduc la o adresă anterioară.

Predicții hardware dinamice:

În acest caz putem schimba predicția în timpul execuției programului pe baza unui istoric. Se menține un istoric pentru fiecare decizie de ramificație luată și vom utiliza acest istoric pentru a lua o decizie la momentul curent de timp.

Decizii întârziate:

Ramificația întârziată execută întotdeauna următoarea instrucțiune secvențială iar ramificația se execută după terminarea execuției de întârziere.

Hazard RAW – read after write

Exemplul 1:

i: **R7 = R12 + R15**

i+1: **R8 = R7 – R12**

i+2: **R15 = R8 + R7**

- Instrucțiunea i+1 are o dependență RAW de instrucțiunea i, deoarece unul dintre registrele sale de intrare, R7, este registru de ieșire pentru instrucțiunea i
- Instrucțiunea i+2 are o dependență RAW cu instrucțiunea i, din același motiv
- Instrucțiunea i+2 are o dependență RAW cu instrucțiunea i+1 deoarece unul dintre registrele de instrare, R8, este registru de ieșire pentru instrucțiunea i+1
- R12 nu cauzează nici o dependență

Exemplul 2:

```
i1:  add  r1, r2, r3
i2:  lw   r4, 0(r1)
i3:  sw   12(r1), r4
```

r1 este citit de o instrucțiune care vine după o instrucțiune care scrie în r1

Hazard WAW – write after write

i1: sub R6, R1, R2

i2: add R6, R4, R5 // se încearcă scrierea reg R6 chiar daca încă nu s-a terminat instrucțiunea precedentă

Hazard WAR – write after read

i1: add R3, R1, R2

i2: sub R5, R4, R3 //RAW cauzat de utilizarea conținutului R3

I3: or R4, R0, R6 // se încearcă scrierea reg R4 chiar daca încă nu s-a terminat citirea conținutului său de către instrucțiunea precedentă

// WAR apare foarte rar, iar în acest exemplu se datorează faptului că avem un hazard RAW înaintea lui, iar i2 așteaptă încă operanzii.