



# Stall on load

Prof. N. Alberto Borghese  
Dipartimento di Informatica  
[alberto.borghese@unimi.it](mailto:alberto.borghese@unimi.it)

Università degli Studi di Milano

Riferimento al Patterson: 4.7, 4.8



# Sommario

Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

Hazard sul controllo





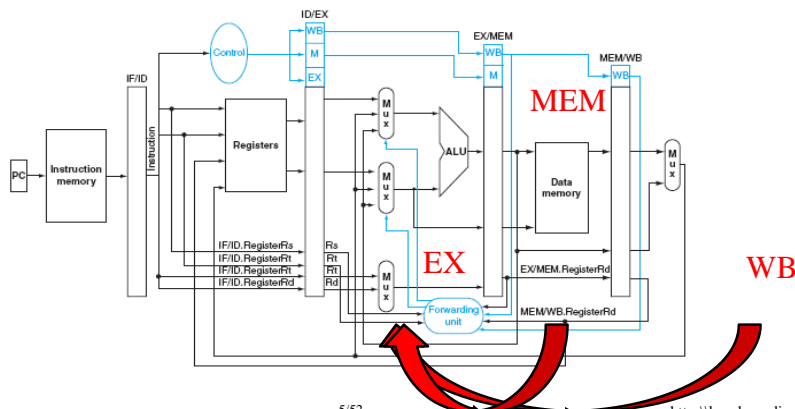
## Soluzione mediante propagazione



Si attiva quando c'è una criticità sui **dati** originata da una **dipendenza**.

Se il dato critico è già disponibile all'interno della pipe-line si preleva e si propaga (mediante bypass) all'indietro all'istruzione che ne ha bisogno.

*NB. L'unità di propagazione non ha conoscenza semantica: prende il contenuto di alcuni bus e lo elabora mediante una funzione logica, attivando opportunamente MuxA e MuxB. Non "sa" quello che sta facendo.*



## Codice con hazard sui dati



**In linguaggio C:**

```
t2 = vett[10] + s5;
```

```
t3 = s6 || vett[10];
```

```
vett[10] = vett[10]*2
```

**In linguaggio Assembler MIPS (\*s3 = vett[0]):**

```
lw $s2, 40($s3) # vett[10] -> $s2
```

```
add $t2, $s2, $s5
```

```
or $t3, $s6, $s2
```

```
add $t0, $s2, $s2 # vett[10]*2
```

```
sw $t0, 40($s3) # $s2 -> vett[10]
```

**Supponiamo all'inizio:**

Registri tutti a 0;

$vett[i] = 12 \forall i$

**Alla fine deve risultare:**

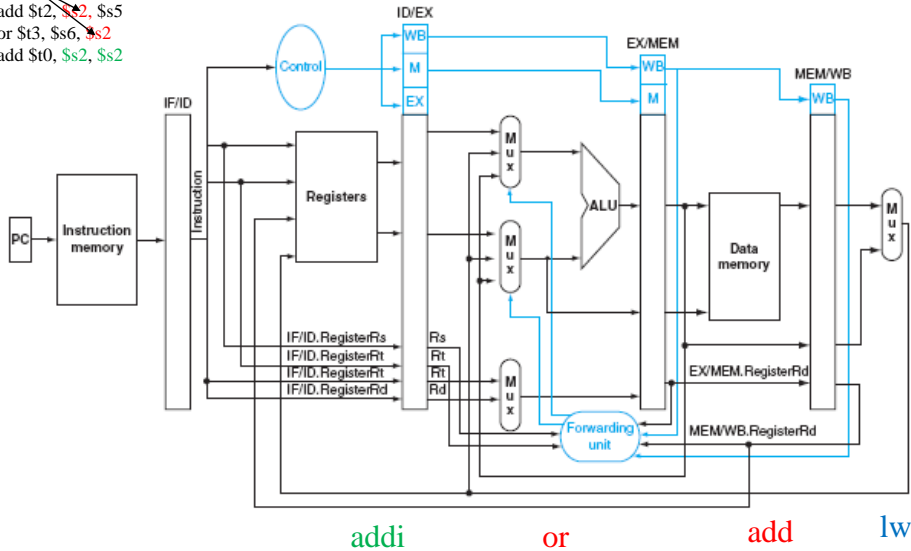
$\$t2 = 12; \$t3 = 12; vett[10] = 24;$



## Le criticità sono risolte con il bypass?



lw \$s2, 40(\$s3)  
 add \$t2, \$s2, \$s5  
 or \$t3, \$s6, \$s2  
 add \$t0, \$s2, \$s2



## Hazard sui dati: lw



|                      |    |    |                     |                      |                     |                |               |                    |    |
|----------------------|----|----|---------------------|----------------------|---------------------|----------------|---------------|--------------------|----|
| lw \$s2, 40(\$s3)    | IF | ID | EX<br>\$s3+40       | MEM<br>\$s2=vett[10] | WB<br>\$s2=12       |                |               |                    |    |
| add \$t2, \$s2, \$s5 | IF | ID | EX<br>\$s2 and \$s5 | MEM                  | MEM<br>\$t2=0       |                |               |                    |    |
| or \$t3, \$s6, \$s2  |    | IF | ID                  | EX<br>\$s6 or \$s2   | MEM<br>\$t3=12      |                |               |                    |    |
| add \$t0, \$s2, \$s2 |    |    | IF                  | ID                   | EX<br>\$s2<br>+\$s2 | MEM<br>\$s2=24 |               |                    |    |
| sw \$t0, 40(\$s3)    |    |    |                     |                      | IF                  | ID             | EX<br>\$s3+40 | MEM<br>Vett[10]=24 | WB |

**Supponiamo all'inizio:**

Registri tutti a 0;  
 vett[i] = 12  $\forall i$   
 vett[0] -> \$s3

In precedenza con la propagazione risolvevo l'hazard su add e or. E ora?



# La criticità su or

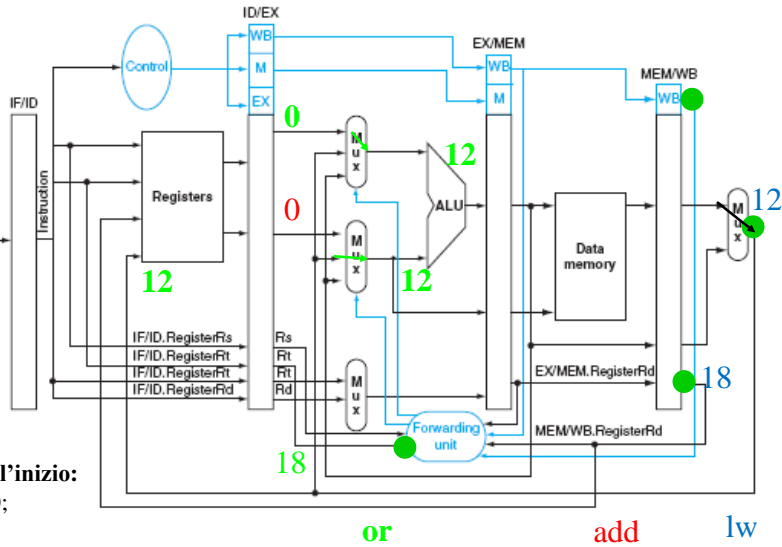


```

lw $s2, 40($s3)
add $t2, $s2, $s5
or $t3, $s6, $s2
add $t0, $s2, $s2

```

Supponiamo all'inizio:  
 Registri tutti a 0;  
 $vett[i] = 12 \forall i$   
 $vett[0] \rightarrow \$s3$



\$s2 = 18



# La criticità sulla prima add



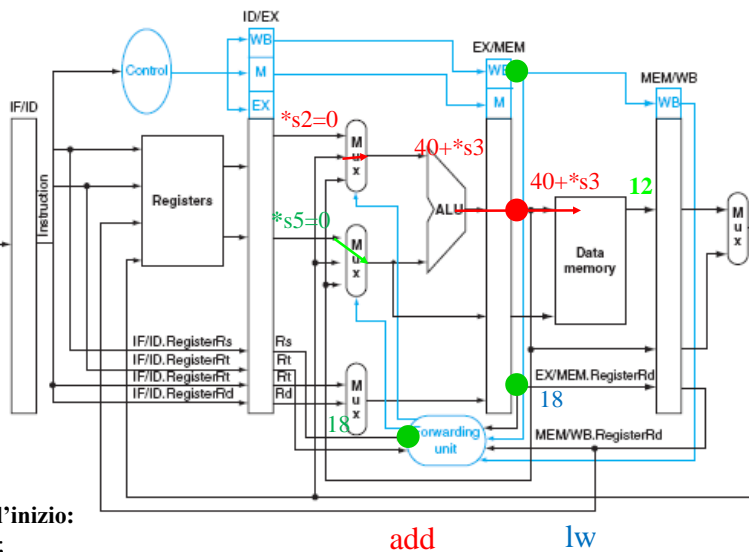
```

lw $s2, 40($s3)
add $t2, $s2, $s5
or $t3, $s6, $s2
add $t0, $s2, $s2

```

\$s2 = 18

Supponiamo all'inizio:  
 Registri tutti a 0;  
 $vett[i] = 12 \forall i$   
 $vett[0] \rightarrow \$s3 = 0 \rightarrow$



## Hazard sui dati: lw, rilevamento della criticità

|                      |    |    |                   |                        |                       |               |                 |  |  |
|----------------------|----|----|-------------------|------------------------|-----------------------|---------------|-----------------|--|--|
| lw \$s2, 40(\$s3)    | IF | ID | EX<br>\$s3+<br>40 | MEM<br><\$s3+40>       | WB<br>s->\$s2         |               |                 |  |  |
| add \$t2, \$s2, \$s5 |    | IF | ID                | EX<br>\$s2 and<br>\$s5 | MEM                   | WB<br>s->\$t2 |                 |  |  |
| or \$t3, \$s6, \$s2  |    |    | IF                | ID                     | EX<br>\$s6 or<br>\$s2 | MEM           | WB<br>(s->\$t3) |  |  |

Il dato corretto per \$s2 è pronto nella lw solamente alla fine della fase MEM, ed è perciò utilizzabile solamente a partire dall'inizio della fase di WB.

Al più tardi, posso risolvere la criticità su **or** quando **or inizia la fase di EX**. In questo caso il dato corretto si trova all'inizio della fase WB della lw e **può essere propagato** (il bypass risolve il problema).

Al più tardi, posso risolvere la criticità su **and** quando **and inizia la fase di EX**. In questo caso il dato corretto **non è ancora stato prodotto** dalla lw. **Non posso risolvere questo hazard con il bypass.**

## Soluzione mediante stallo

|                      | t <sub>0</sub>      | t <sub>1</sub> | t <sub>2</sub> | t <sub>3</sub>  | t <sub>4</sub> | t <sub>5</sub> | t <sub>6</sub> | t <sub>7</sub> |
|----------------------|---------------------|----------------|----------------|-----------------|----------------|----------------|----------------|----------------|
| ....                 |                     |                |                |                 |                |                |                |                |
| lw \$s2, 40(\$s3)    | FF<br>(Mem,<br>ALU) | DECOD<br>(RF)  | EXEC<br>(ALU)  | MEM<br>(MEM)    | WB<br>(RF)     |                |                |                |
| nop                  |                     | bolla<br>(FF)  | Bolla<br>(DEC) | bolla<br>(EXEC) | bolla<br>(MEM) | bolla<br>(WB)  |                |                |
| add \$t2, \$s2, \$s5 |                     |                | FF             | DEC             | EXEC           | MEM            |                |                |

I buchi (o bubble) inducano degli istanti di clock in cui non può essere eseguita l'istruzione successiva → **La pipeline va messa in stallo.**

Devo **bloccare l'esecuzione della and** per un ciclo di clock **per attendere che la lw abbia caricato dalla memoria il valore corretto di \$s2** e sia quindi disponibile in pipeline.

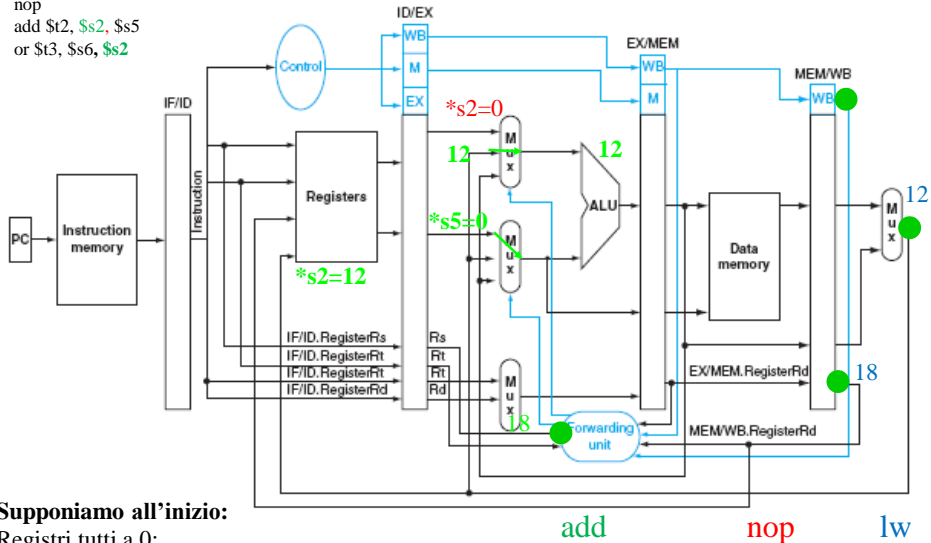
Il cammino in blu invece indica l'istante a partire dal quale il cammino di propagazione interna dei dati consente di risolvere l'hazard. Per la add è richiesto 1 ciclo di stallo.



## Stallo dell'istruzione di add



lw \$s2, 40(\$s3)  
 nop  
 add \$t2, \$s2, \$s5  
 or \$t3, \$s6, \$s2



Supponiamo all'inizio:

Registri tutti a 0;  
 vett[i] = 12  $\forall i$   
 vett[0] -> \*\$s3

In questa situazione, l'unità di propagazione può risolvere l'hazard



## Questions



Come mai il compilatore non ha inserito un nop?  
 perchè la coppia lw / add si è formata durante l'esecuzione (salti)  
 perchè il compilatore ha fallito...

....

La CPU deve garantire comunque la correttezza dell'esecuzione sempre.

Come fa la CPU a inserire una nop?

- 1) Rilevazione
- 2) Soluzione





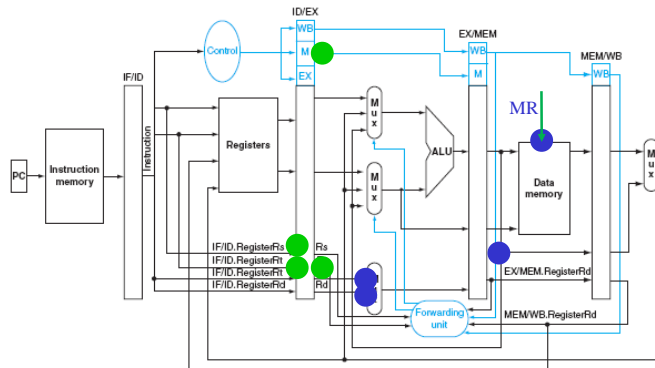


## Analisi della dipendenza - EXE



Posso capire che si tratta di un'istruzione di lw nelle fasi DEC, EX, MEM.

Posso capire se il registro rt della lw è uguale a uno dei registri operando dell'istruzione successiva a partire dalla fase di DEC dell'istruzione dipendente e dalla fase di EXE dell'istruzione di lw.



“The sooner the better”



## Implementazione del rilevamento della criticità della lw



IF (ID/EX.MemRead)

AND

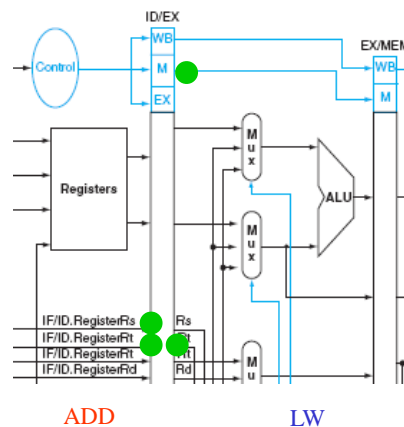
[(IF/ID.RegistroRt == ID/EX.RegistroRt) OR  
(IF/ID.RegistroRs == IF/EX.RegistroRt)]

THEN “Inserisci una nop”

EX – lw \$s2, 40(\$s3)

DEC – add \$t2, \$s2, \$s5

Inserire un nop → Fare **aspettare** l'istruzione di ADD (e quelle successive) un ciclo di clock (**STALLO della add e delle istruzioni successive**).



ADD

LW



# Sommario



Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

Hazard sul controllo



# Implementazione dello stallo



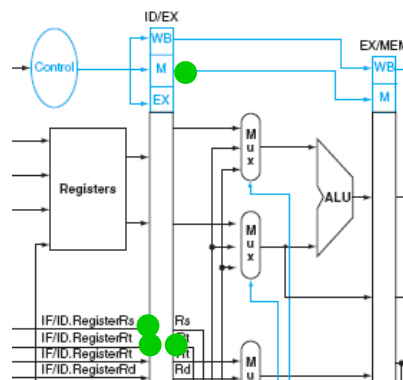
IF (ID/EX.MemWrite)  
 AND  
 [(IF/ID.RegistroRt == ID/EX.RegistroRt) OR  
 (IF/ID.RegistroRs == ID/EX.RegistroRt)]  
 THEN "Inserisci una nop"

EX     – lw \$s2, 40(\$s3)  
 DEC    – add \$t2, \$s2, \$s5

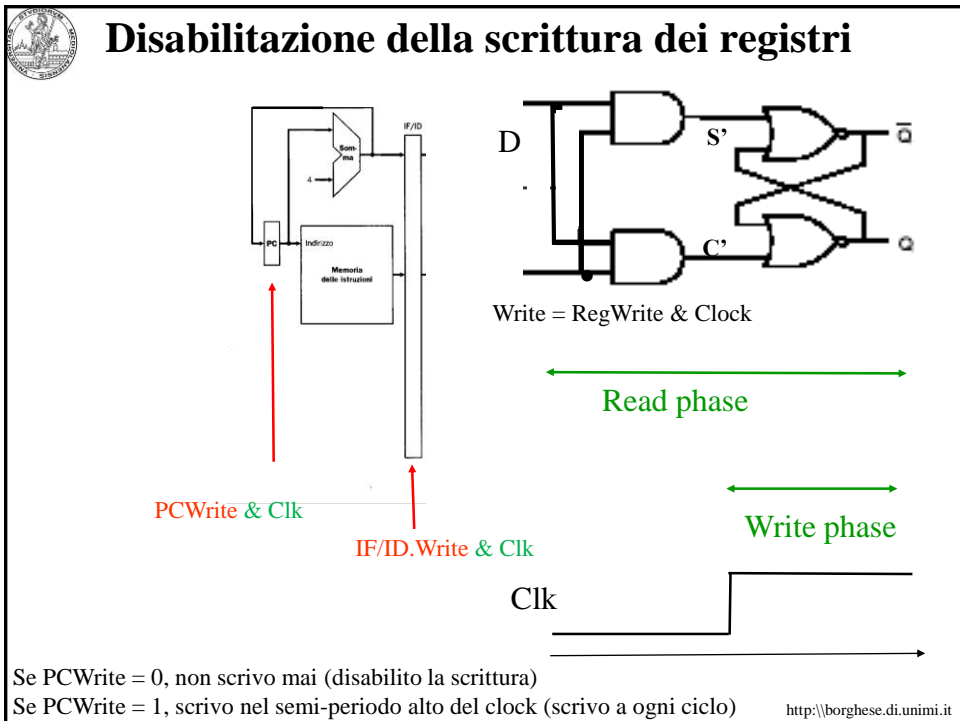
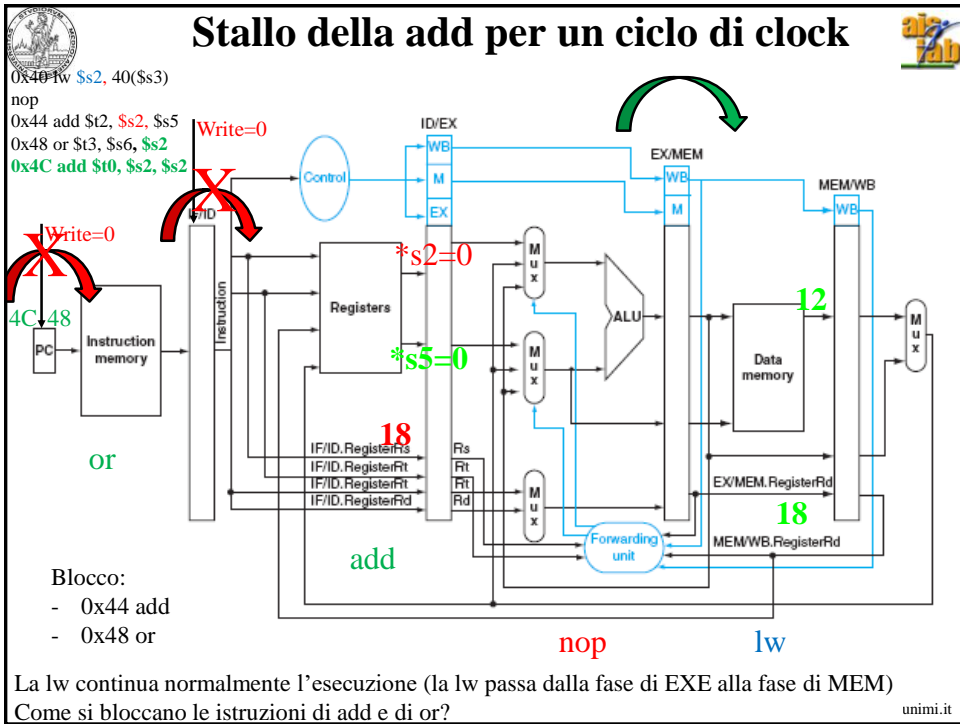
**La nop va inserita tra la lw e la add.**

- Lascio continuare l'esecuzione della lw
- Blocco per un ciclo di clock l'esecuzione della add (e delle istruzioni successive)



Inserisco la nop il prima possibile, nella fase di EXE della lw.



|        |     |     |    |
|--------|-----|-----|----|
| Time 1 | ADD | LW  |    |
| Time 2 | ADD | nop | LW |

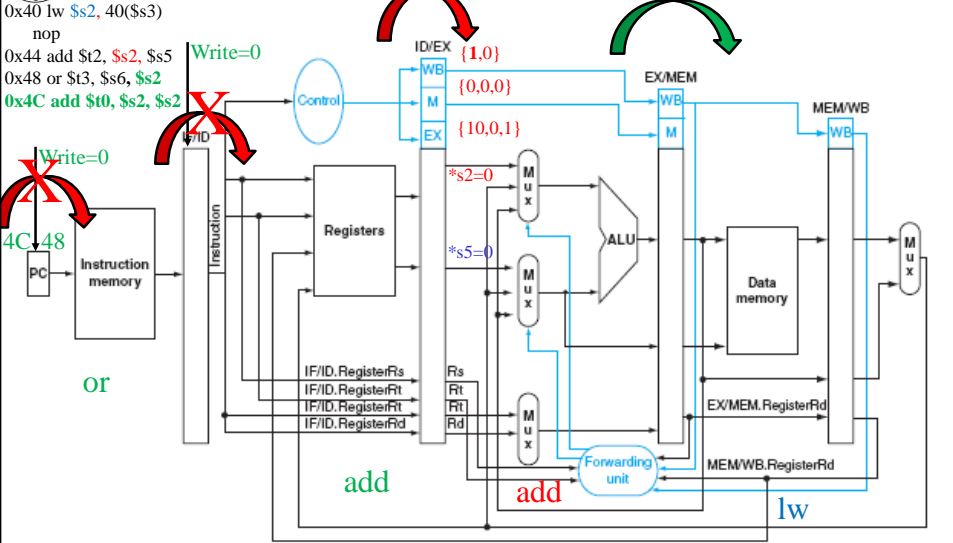


## Cosa conterrà EXE?

```



0x40 lw $s2, 40($s3)
nop
0x44 add $t2, $s2, $s5
0x48 or $t3, $s6, $s2
0x4C add $t0, $s2, $s2
  
```



Se non blocco la scrittura del registro ID/EX continuerà l'esecuzione della add con il primo operando sbagliato. **Non ho risolto nulla!**

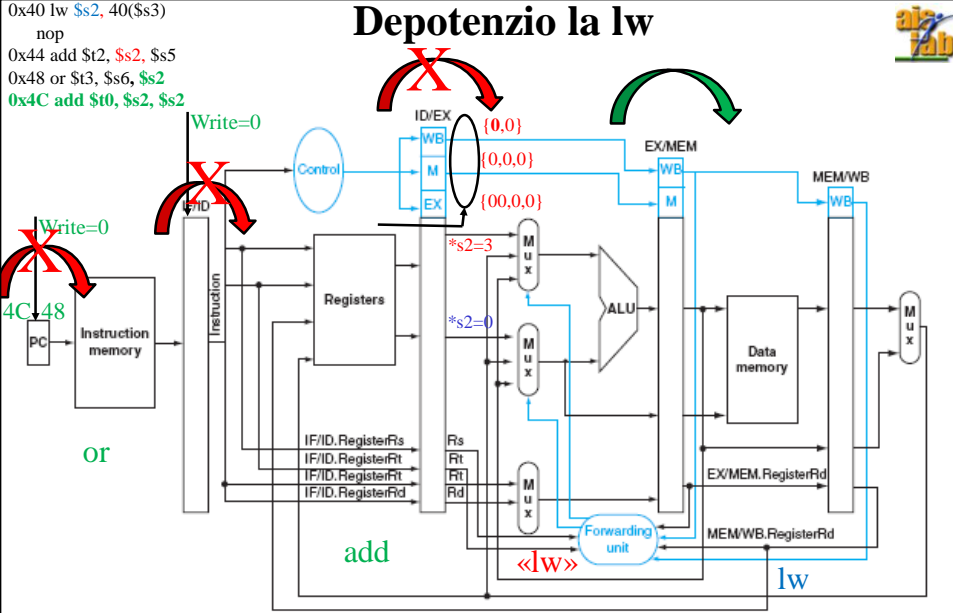
A.A. 2020-2021
23/52
<http://borghese.di.unimi.it>

## Depotenzio la lw

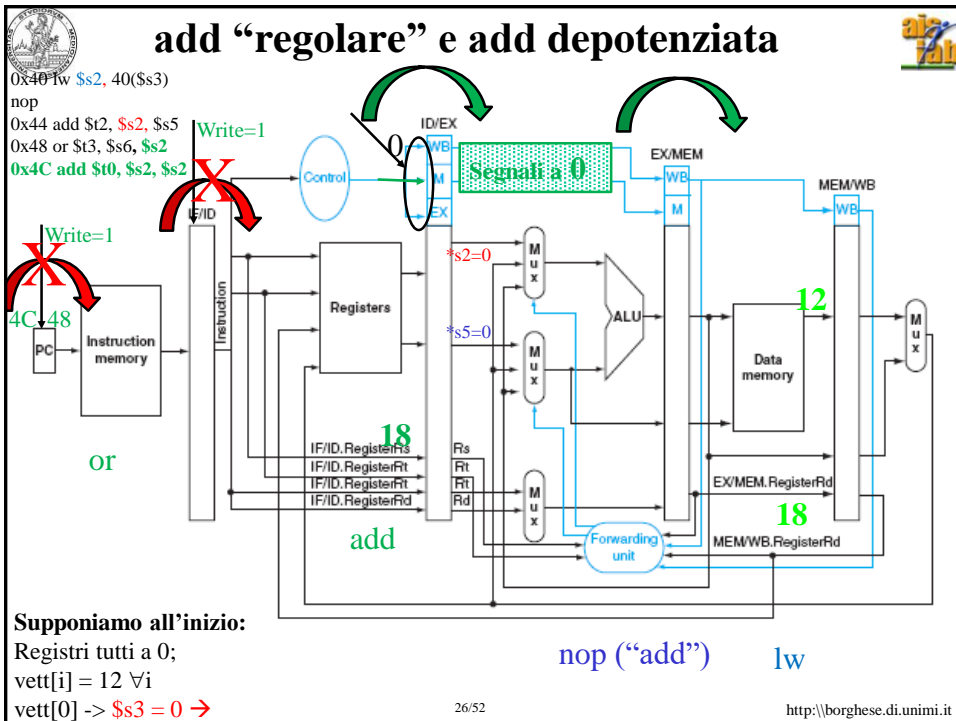
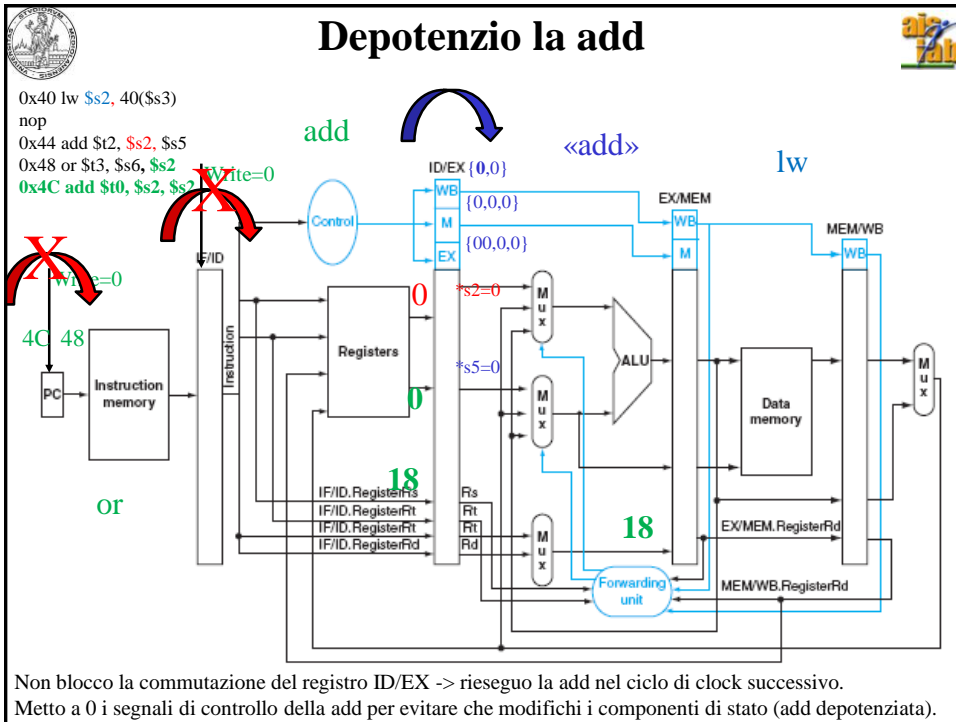
```

0x40 lw $s2, 40($s3)
nop
0x44 add $t2, $s2, $s5
0x48 or $t3, $s6, $s2
0x4C add $t0, $s2, $s2
  
```



Blocco la commutazione del registro ID/EX -> rieseguo la lw nel ciclo di clock successivo. Metto a 0 i segnali di controllo della lw per evitare che modifichi i componenti di stato (lw depotenziata). → Occorre scrivere in ID/EX il segnale di azzeramento del controllo.

t







# Hazard nei dati: soluzioni



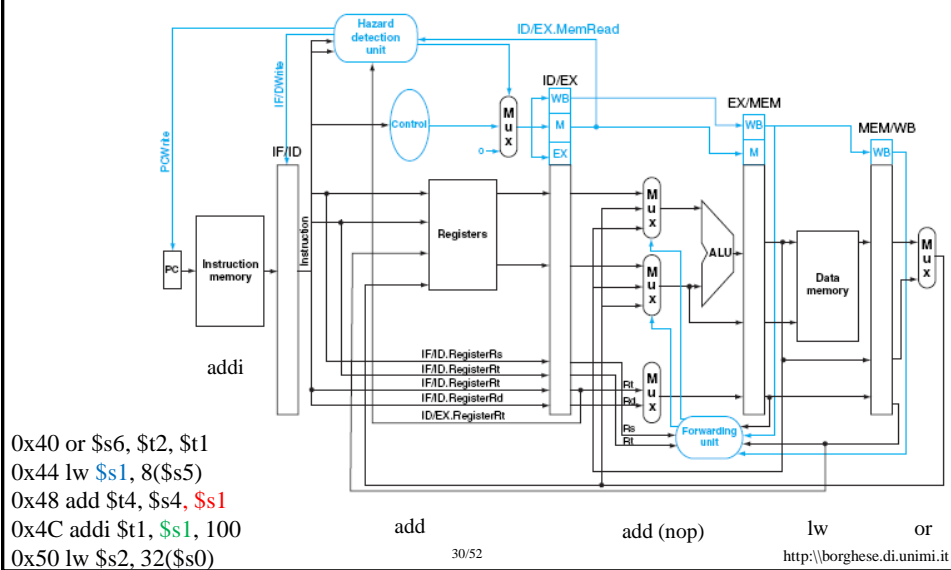
- Buona scrittura del codice (il programmatore deve conoscere la macchina per scrivere un buon codice!).
- Compilatore efficiente (che riordini il codice).
- Architettura che renda disponibile i dati appena pronti alla fase di esecuzione.
- Accettare uno stallo (non sempre si può evitare).



# Esempio



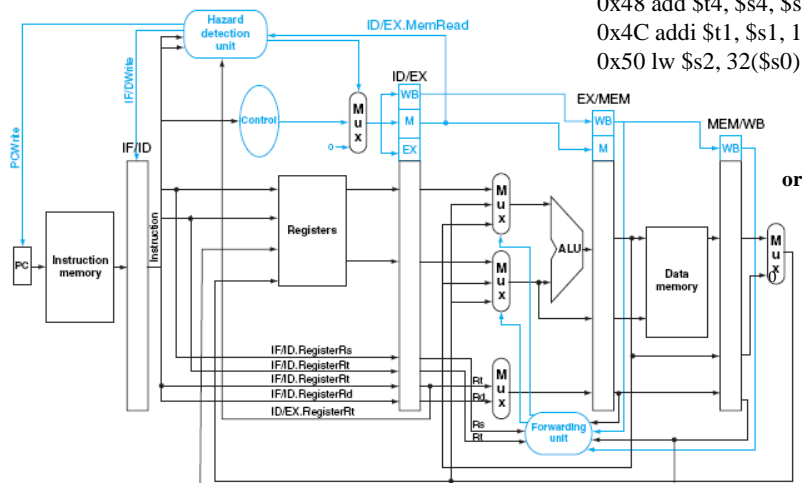
Specificare il contenuto di tutti i bus di questa pipeline quando è in esecuzione il seguente codice con l'istruzione or in WB:





## Esercizio – fase WB

0x40 or \$s6, \$t2, \$t1  
 0x44 lw \$s1, 8(\$s5)  
 0x48 add \$t4, \$s4, \$s1  
 0x4C addi \$t1, \$s1, 100  
 0x50 lw \$s2, 32(\$s0)

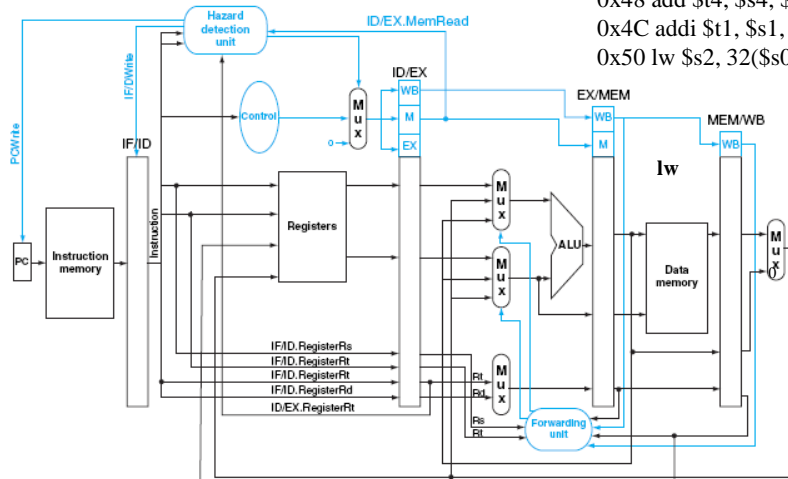


MEM/WB.mdr = ?  
 MEM/WB.data = t2 || t1  
 MEM/WB.RegScrittura = 22 = \$s6  
 MEM/WB.RegWrite = 1;  
 MEM/WB.Mem2Reg = 0



## Esercizio – fase MEM

0x40 or \$s6, \$t2, \$t1  
**0x44 lw \$s1, 8(\$s5)**  
 0x48 add \$t4, \$s4, \$s1  
 0x4C addi \$t1, \$s1, 100  
 0x50 lw \$s2, 32(\$s0)



EX/MEM.data = \*s5+8  
 EX/MEM.\*rt = \*s1  
 EX/MEM.MemR = 1  
 EX/MEM.RegWrite = 1;

MDR = MEM[\*s5+8]  
 EX/MEM.RegScrittura = 17 = \$s1  
 EX/MEM.MemW = 0  
 EX/MEM.Mem2Reg = 1

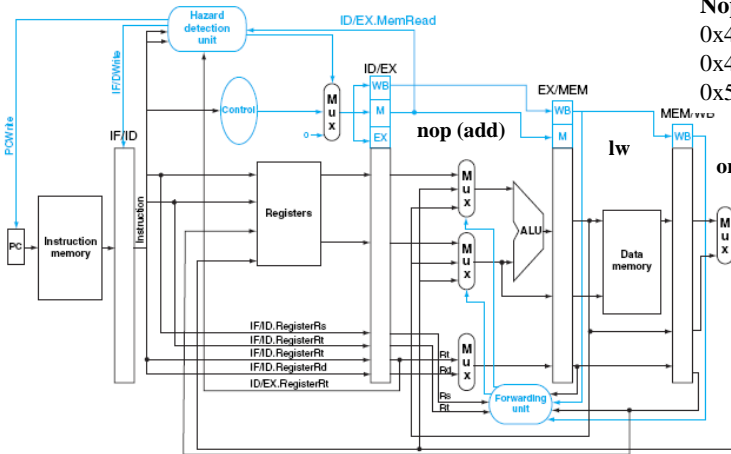
EX/MEM.Branch = 0





## Esercizio – fase EXE

0x40 or \$s6, \$t2, \$t1  
 0x44 lw \$s1, 8(\$s5)  
**Nop (add)**  
 0x48 add \$t4, \$s4, \$s1  
 0x4C addi \$t1, \$s1, 100  
 0x50 lw \$s2, 32(\$s0)



ID/EX.\*rs = \*s4

ID/EX.rs = 20

Operando1 = \*s4

ID/EX.ALUSrc = 0;

ID/EX.MemR = 0

ID/EX.RegWrite = 0

ID/EX.\*rt = \*s1

ID/EX.rt = 17

operando2 = \*s5 + 8 (propagazione attivata ma dato non corretto)

ALUOp = 00

ID/EX.MemW = 0

ID/EX.Mem2Reg = 0

ID/EX.AluDataOut = \*s4 + \*s5 + 8

ID/EX.rd = 12

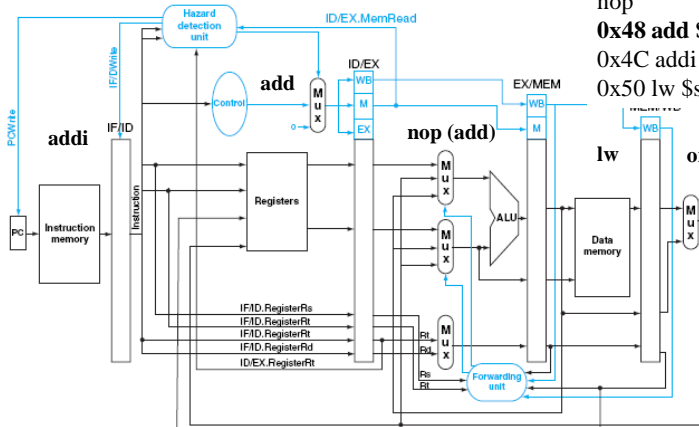
RegDst = 0

ID/EX.Branch = 0



## Esercizio – fase DEC

0x40 or \$s5, \$t2, \$t1  
 0x44 lw \$s1, 8(\$s5)  
 nop  
**0x48 add \$t4, \$s4, \$s1**  
 0x4C addi \$t1, \$s1, 100  
 0x50 lw \$s2, 32(\$s0)



IF/ID.rs = 20 = \$s4

Operando1 = \*s4

Segnali di controllo dell'istruzione ADD

ALUSrc = 1; ALUOp = "R"; RegDst = 1;

MemR = 0; MemW = 0; Branch = 0;

RegWrite = 1; Mem2Reg = 0;

IF/ID.rt = 17 = \$s1

operando2 = \*s1



## Esercizio



Specificare il contenuto di tutti i bus di questa pipeline quando è in esecuzione il seguente frammento di codice con l'istruzione or in fase di WB:

```
0x00000400 or $s5, $t2, $t1
0x00000404 sw $s1, 8($s0)
0x00000408 add $t4, $s5, $s1
0x0000040C addi $t1, $t4, 100
0x00000410 lw $s2, 32($s0)
```



## Sommario



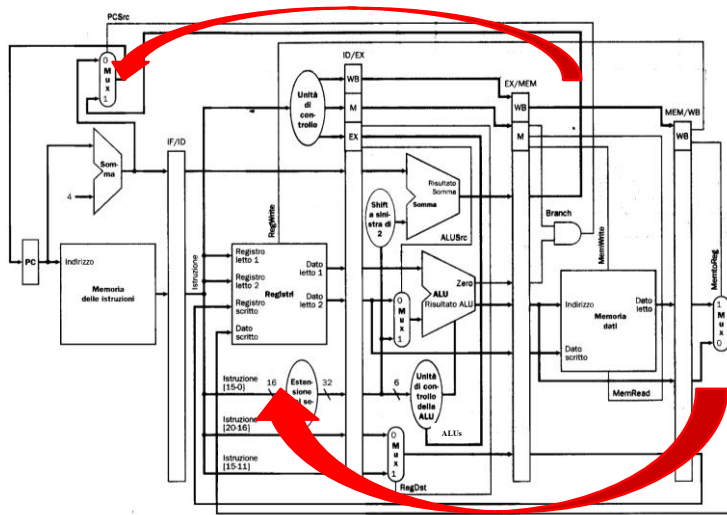
Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

**Hazard sul controllo**



## Pipeline – criticità o hazard



la fase di WB genera un cammino da dx (WB) a sx (DEC) sul data path  
beq genera un cammino da dx (MEM) a sx (FF) sul control path



## Come affrontare gli *hazard su controllo*

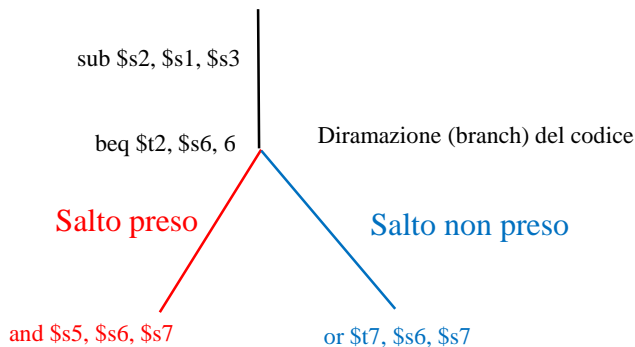


- Si può risolvere l'hazard...
  - ...*aspettare*
    - si mette lo stage opportuno dell'istruzione dipendente dalla precedente in pausa
    - il controllo della pipeline deve individuare il problema prima che avvenga! **Stallo**.
  - ...*prevenire*
    - Il compilatore, come ottimizzazione, può **riordinare le istruzioni** in modo che il risultato sia lo stesso ma non ci siano hazard
  - ...*modificare la CPU*
  - ...*scartare istruzioni*
    - si butta via l'attuale lavoro della pipeline e si ricomincia ("flushing" della pipeline)
    - è sufficiente individuare il problema DOPO che è avvenuto
    - Es: l'istruzione successiva (a sx) ha usato in lettura un registro che è stato appena modificato dall'istruzione precedente (dx)? **flush**. Oppure: l'istruzione precedente (a dx) ha effettuato un salto e quindi successive (a sx) sta lavorando con un PC e IR sbagliato? **flush**.
  - ...*prevedere*
    - attraverso alcuni meccanismi di **predizione** preposti, l'architettura stessa tenta di predire su base statistica i risultati rilevanti dell'istruzione in corso (es il PC – "branch prediction", o il valori prodotti – "value prediction"). Se la predizione si rivela giusta: tutto ok. Se si rivela sbagliata: **roll-back**



## Hazard sul controllo

|                            |
|----------------------------|
| 0x400 sub \$s2, \$s1, \$s3 |
| 0x404 beq \$t2, \$s6, 7    |
| 0x408 or \$t7, \$s6, \$s7  |
| 0x40C add \$t4, \$s8, \$s8 |
| 0x410 and \$s5, \$s6, \$s7 |
| 0x414 add \$t0, \$t1, \$t2 |
| 0x418 sw \$s3, 24(\$t1)    |
| 0x41C addi \$t7, \$s6, 10  |
| 0x420 add \$t8, \$s2, \$s2 |
| 0x424 and \$s5, \$s6, \$s7 |
| 0x418 add \$t0, \$t1, \$t2 |



## Esempio di Hazard sul controllo

|                            |    |    |               |                                 |              |     |     |     |
|----------------------------|----|----|---------------|---------------------------------|--------------|-----|-----|-----|
| 0x400 sub \$s2, \$s1, \$s3 | IF | ID | EX<br>\$1-\$3 | MEM                             | WB<br>s->\$2 |     |     |     |
| 0x404 beq \$t2, \$s6, 7    |    | IF | ID            | EX<br>Zero if<br>(\$s2 == \$s5) | MEM          | WB  |     |     |
| 0x408 or \$t7, \$s6, \$s7  |    |    | IF            | ID                              | EX           | MEM | WB  |     |
| 0x40C add \$t4, \$s8, \$s8 |    |    |               | IF                              | ID           | EX  | MEM | WB  |
| 0x410 and \$s5, \$s6, \$s7 |    |    |               |                                 | IF           | ID  | EX  | MEM |
| 0x414 add \$t0, \$t1, \$t2 |    |    |               |                                 |              | IF  | ID  | EX  |

Quando sono in fase di fetch dovrei avere a disposizione l'indirizzo corretto. In caso di salto questo potrebbe esser disponibile nella PIPELINE solo all'inizio della fase di WB della beq.

**In caso di salto:** Ho 3 istruzioni sbagliate in pipeline: or, la add e la and.

NB Il PC è master/slave per cui occorre che l'indirizzo sia pronto prima dell'inizio della fase di fetch.



## Soluzione 1 – «aspettare» - stallo



0x400 sub \$s2, \$s1, \$s3  
 0x404 beq \$t2, \$s6, 7  
 0x408 or \$t7, \$s6, \$s7  
 0x40C add \$t4, \$s8, \$s8  
 0x410 and \$s5, \$s6, \$s7  
 0x414 add \$t0, \$t1, \$t2  
 0x418 sw \$s3, 24(\$t1)  
 0x41C addi \$t7, \$s6, 10  
 0x420 add \$t8, \$s2, \$s2  
 0x424 and \$s5, \$s6, \$s7  
 0x428 add \$t0, \$t1, \$t2

0x400 sub \$s2, \$s1, \$s3  
 0x404 beq \$t2, \$s6, 7  
 nop  
 nop  
 nop  
 0x408 or \$t7, \$s6, \$s7  
 0x40C add \$t4, \$s8, \$s8  
 0x410 and \$s5, \$s6, \$s7  
 0x414 add \$t0, \$t1, \$t2  
 0x418 sw \$s3, 24(\$t1)  
 0x41C addi \$t7, \$s6, 10  
 0x420 add \$t8, \$s2, \$s2  
 0x424 and \$s5, \$s6, \$s7  
 0x428 add \$t0, \$t1, \$t2

Con lo stallo spreco di 3 cicli di clock (in modo che la fase IF dell'istruzione successiva vada a coincidere con la fase di WB dell'istruzione beq. **Situazione troppo frequente perché la soluzione sia accettabile.**



## Esempio di Hazard sul controllo



|                            |    |    |               |                                 |              |     |     |     |
|----------------------------|----|----|---------------|---------------------------------|--------------|-----|-----|-----|
| 0x400 sub \$s2, \$s1, \$s3 | IF | ID | EX<br>\$1-\$3 | MEM                             | WB<br>s->\$2 |     |     |     |
| 0x404 beq \$t2, \$s6, 7    |    | IF | ID            | EX<br>Zero if<br>(\$s2 == \$s5) | MEM          | WB  |     |     |
| nop                        |    |    | IF            | ID                              | EX           | MEM | WB  |     |
| nop                        |    |    |               | IF                              | ID           | EX  | MEM | WB  |
| nop                        |    |    |               |                                 | IF           | ID  | EX  | MEM |
| 0x408 or \$t7, \$s6, \$s7  |    |    |               |                                 |              | IF  | ID  | EX  |

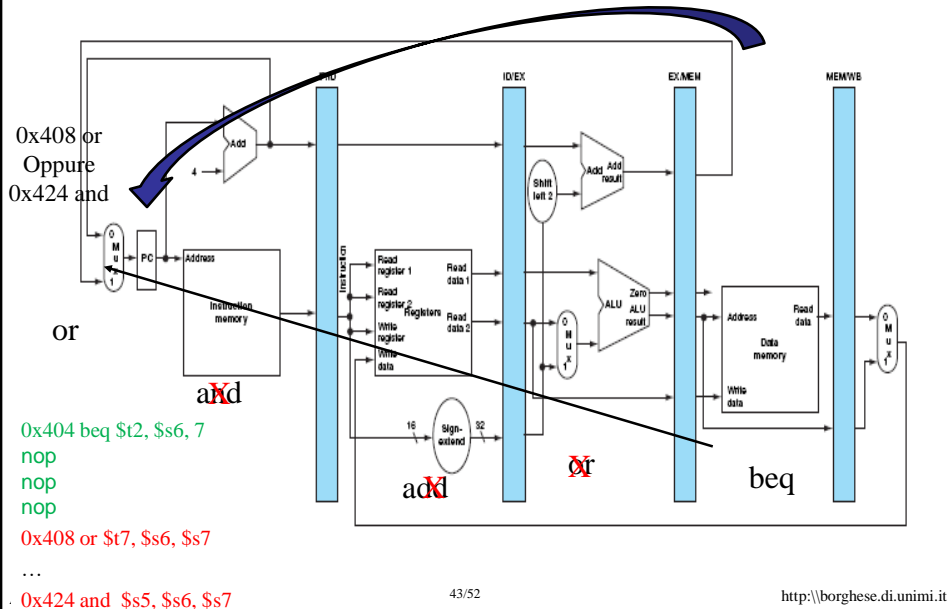
Quando sono in fase di fetch dovrei avere a disposizione l'indirizzo corretto. In caso di salto questo potrebbe esser disponibile nella PIPELINE solo all'inizio della fase di WB della beq.

**In caso di salto:** Ho 3 istruzioni sbagliate in pipeline ma **sono 3 nop**.

NB Il PC è master/slave per cui occorre che l'indirizzo sia pronto prima dell'inizio della fase di fetch.



## Soluzione con uno stallo di 3 cicli di clock



## Soluzione 2 – «prevenire»



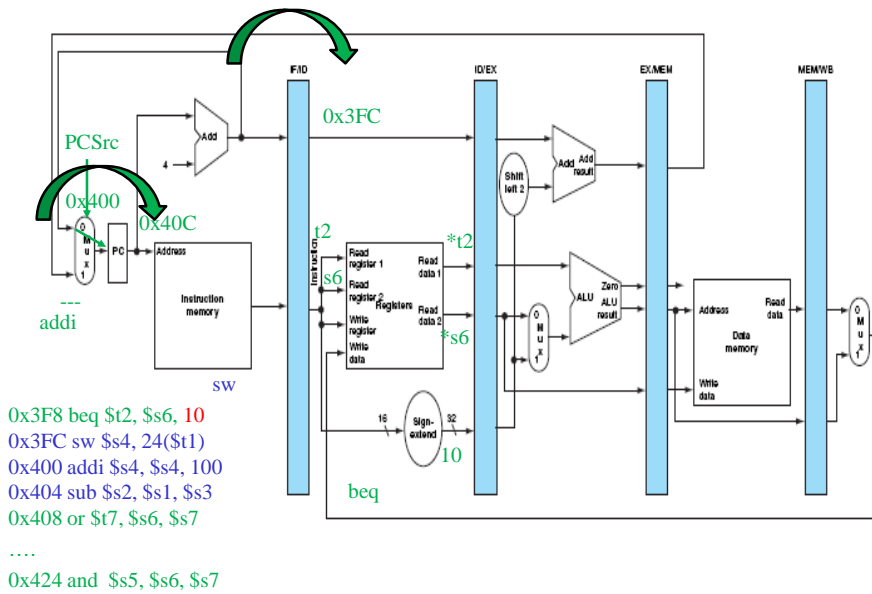
|                            |                            |
|----------------------------|----------------------------|
| 0x3F8 sw \$s4, 24(\$t1)    | 0x3F8 beq \$t2, \$s6, 10   |
| 0x3FC addi \$s4, \$s4, 100 | 0x3FC sw \$s4, 24(\$t1)    |
| 0x400 sub \$s2, \$s1, \$s3 | 0x400 addi \$s4, \$s4, 100 |
| 0x404 beq \$t2, \$s6, 7    | 0x404 sub \$s2, \$s1, \$s3 |
| 0x408 or \$t7, \$s6, \$s7  | 0x408 or \$t7, \$s6, \$s7  |
| 0x40C add \$t4, \$s8, \$s8 | 0x40C add \$t4, \$s8, \$s8 |
| 0x410 and \$s5, \$s6, \$s7 | 0x410 and \$s5, \$s6, \$s7 |
| 0x414 add \$t0, \$t1, \$t2 | 0x414 add \$t0, \$t1, \$t2 |
| 0x418 sw \$s3, 24(\$t1)    | 0x418 sw \$s3, 24(\$t1)    |
| 0x41C addi \$t7, \$s6, 10  | 0x41C addi \$t7, \$s6, 10  |
| 0x420 add \$t8, \$s2, \$s2 | 0x420 add \$t8, \$s2, \$s2 |
| 0x424 and \$s5, \$s6, \$s7 | 0x424 and \$s5, \$s6, \$s7 |
| 0x428 add \$t0, \$t1, \$t2 | 0x428 add \$t0, \$t1, \$t2 |
| ...                        | ...                        |
| ...                        | ...                        |

Branch delay slots

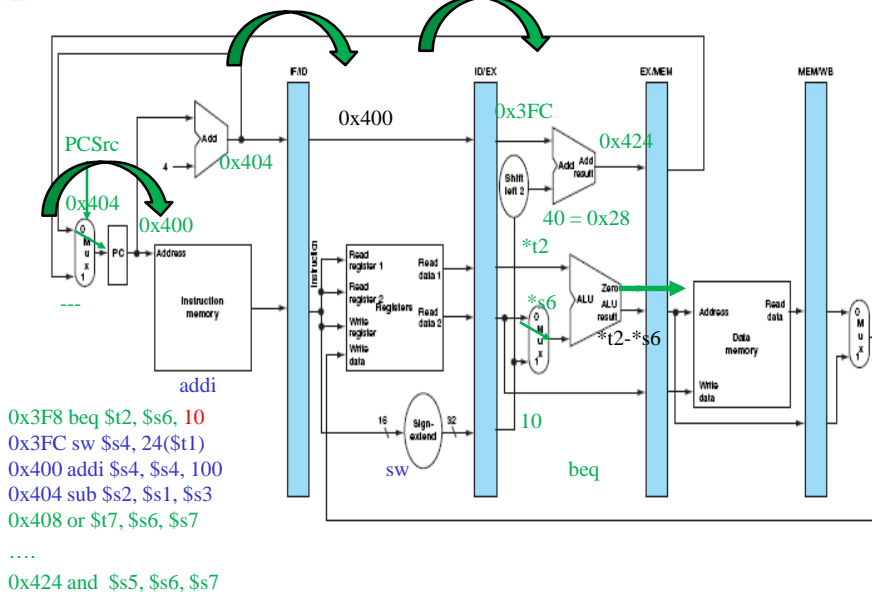
Le istruzioni immediatamente seguenti il salto vengono **sempre** eseguite (sia che il salto venga preso sia che non venga preso). +



# Soluzione con branch delay slots - I

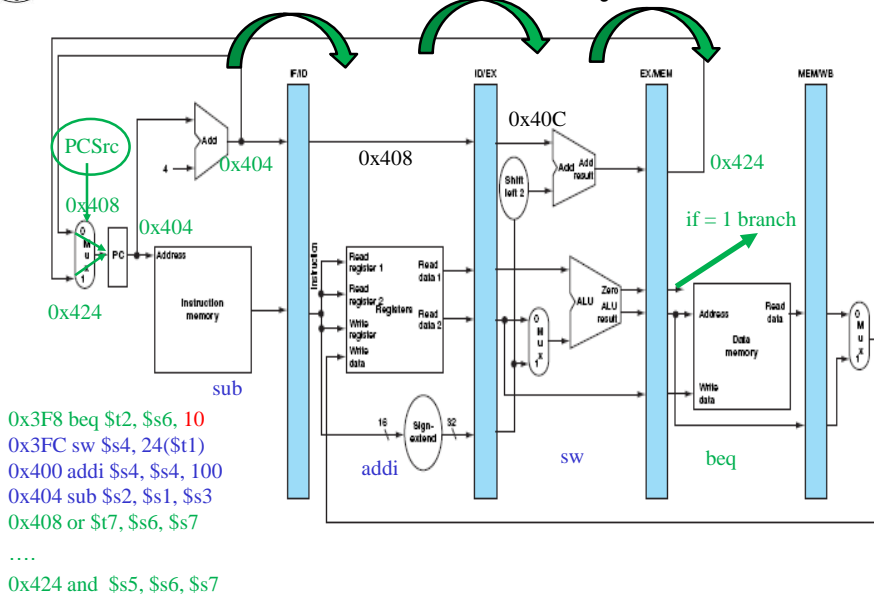


# Soluzione con branch delay slots - II

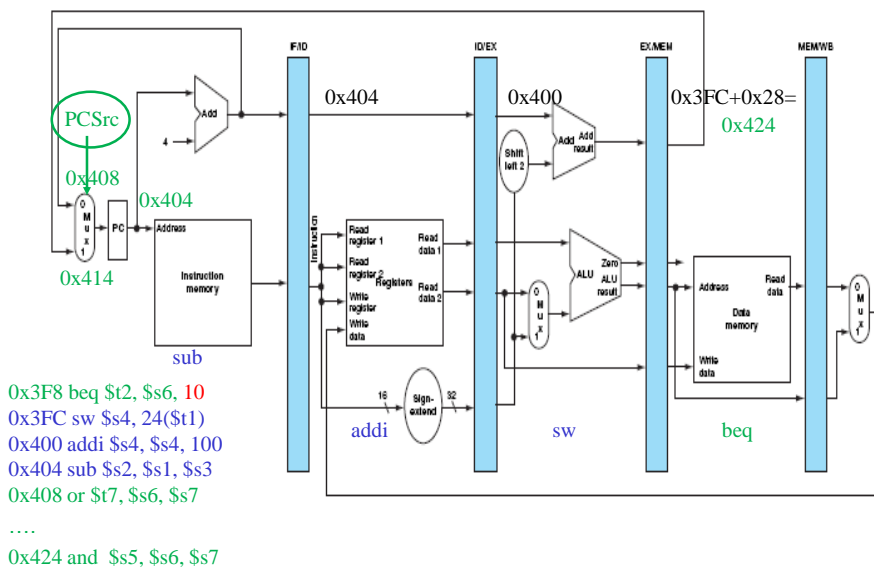




## Soluzione con branch delay slots - III



## Soluzione con branch delay slots







## Salto incondizionato – “prevenire” - I



Utilizzato all'interno dei cicli for / while. Non pone problemi. Si risolve con la riorganizzazione del codice.

|            |                      |        |                      |
|------------|----------------------|--------|----------------------|
| 396:       | addi \$t6,\$t6,20    | 396:   | j 80000              |
| 400:       | add \$s0, \$s1, \$s2 | 400:   | addi \$t6,\$t6,20    |
| 404:       | j 80000              | 404:   | add \$s0, \$s1, \$s2 |
| Label 408: | and \$s1, \$s2, \$s3 | 408:   | and \$s2, \$s2, \$s3 |
| 80000:     | or \$t0, \$t1, \$t2  | 80000: | or \$t0, \$t1, \$t2  |
| 80004:     | sub \$t3, \$t4, \$t5 | 80004: | sub \$t3, \$t4, \$t5 |

j “lavora” nella fase di decodifica. Viene eseguita un’istruzione prima del salto: delayed jump. **Riempio tutti gli slot di esecuzione.**

L’esecuzione avviene fuori ordine, ma l’utente non vede differenze.

Come viene modificata la CPU (parte di datapath e parte di controllo)?



## Salto incondizionato – “prevenire” - II



Prendo l’istruzione dalla destinazione del salto.

|        |                      |        |                      |
|--------|----------------------|--------|----------------------|
| 400:   | add \$s0, \$s1, \$s2 | 400:   | add \$s0, \$s1, \$s2 |
| 404:   | j 80000              | 404:   | j 80008              |
| 408:   | and \$s1, \$s2, \$s3 | 408:   | or \$t0, \$t1, \$t2  |
|        |                      | 412:   | addi \$t6, \$t7, 100 |
|        |                      | 416:   | and \$s2, \$s2, \$s3 |
| 80000: | or \$t0, \$t1, \$t2  |        |                      |
| 80004: | addi \$t6,\$t7,100   |        |                      |
| 80008: | sub \$t3, \$t4, \$t5 | 80008: | sub \$t3, \$t4,\$t5  |

Riempio tutti gli slot di esecuzione.

E’ il compilatore a riorganizzare il codice. Non sono richieste modifiche alla CPU.

Si può fare di meglio con una fase di fetch evoluta.



## Modifiche della CPU



Non sempre i delay slot si riescono a riempire

3 slot sono tanti

Il miglioramento dell'architettura è richiesto.



## Sommario



Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

Hazard sul controllo