



Stall on load

Prof. N. Alberto Borghese
Dipartimento di Informatica
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

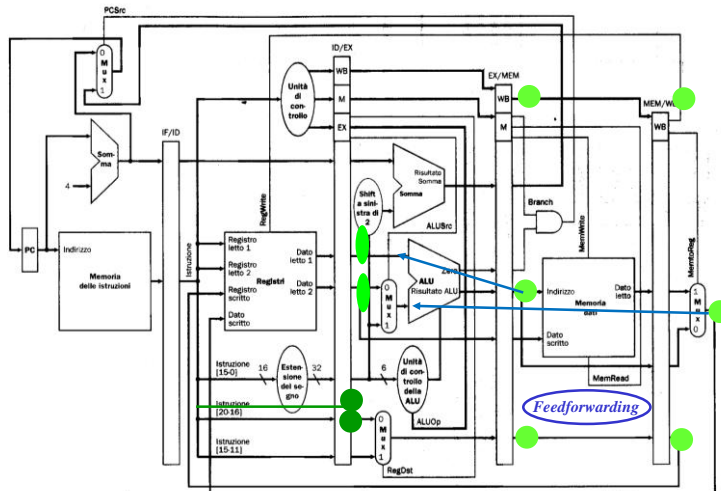


CPU con propagazione



sub \$s2, \$s1, \$s3
 add \$t2, \$s2, \$s5
 or \$t3, \$s6, \$s2

Dipendenze che non creano hazard



A.A. 2024-2025

imi.it



Data Path e criticità



Nella CPU a singolo ciclo, produzione, elaborazione e restituzione del dato avvengono nello stesso ciclo.

Nella CPU con pipeline, **produzione**, **elaborazione** e **restituzione** del dato avvengono in cicli diversi. Ad esempio: add \$t0, \$t1, \$t2.

Produzione del dato: fase di decodifica.

Elaborazione del dato: fase di calcolo.

Restituzione del dato: fase di WB

Questo provoca gli hazard sui dati nelle pipeline perché il dato serve alle istruzioni successive (viene consumato) prima della sua restituzione (nel register file).

Sono hazard predicibili già in fase di decodifica, perché tutta l'informazione richiesta per identificare la dipendenza è disponibile in quella fase.

A.A. 2024-2025

4/40

http://borghese.di.unimi.it



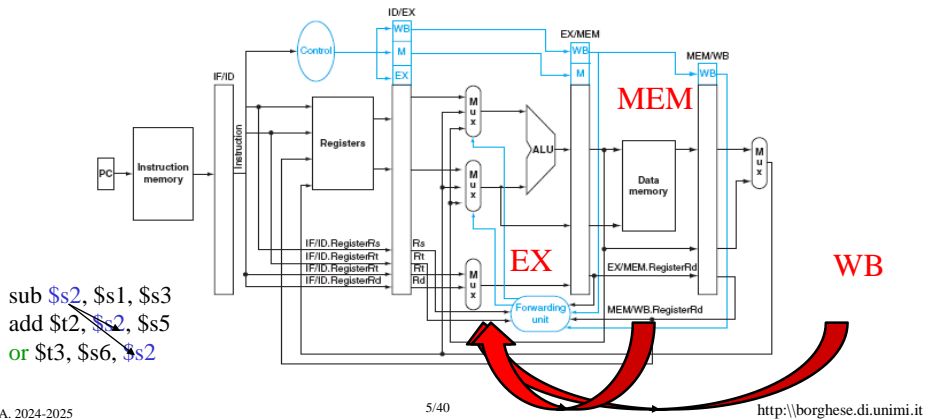
Soluzione mediante propagazione



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 # $t2 = vett[10] + $s5
```

```
or $t3, $s6, $s2 # $t3 = $s6 OR $s2
```

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

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

Supponiamo all'inizio:

```
Registri tutti a 0;
```

```
vett[i] = 12  $\forall$  i
```

Alla fine deve risultare:

```
$t2 = 12 = vett[10];
```

```
$t3 = 12 = vett[10];
```

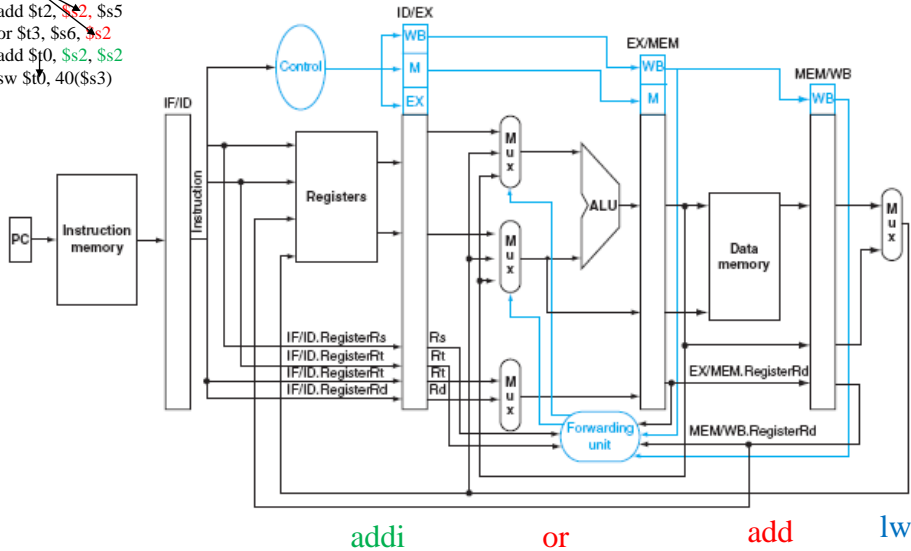
```
vett[10] = 24;
```



Le criticità sono risolte con il bypass?



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



Hazard sui dati: lw



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

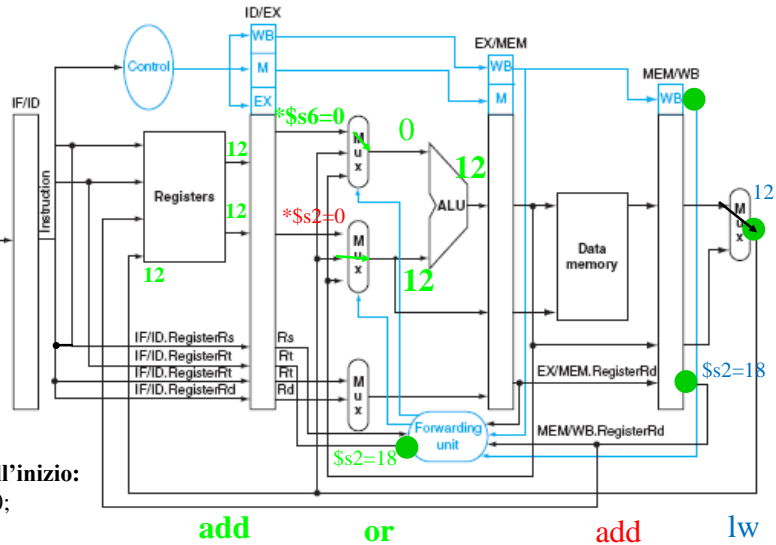


```

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



Le criticità sono risolte con il bypass?



```

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

```

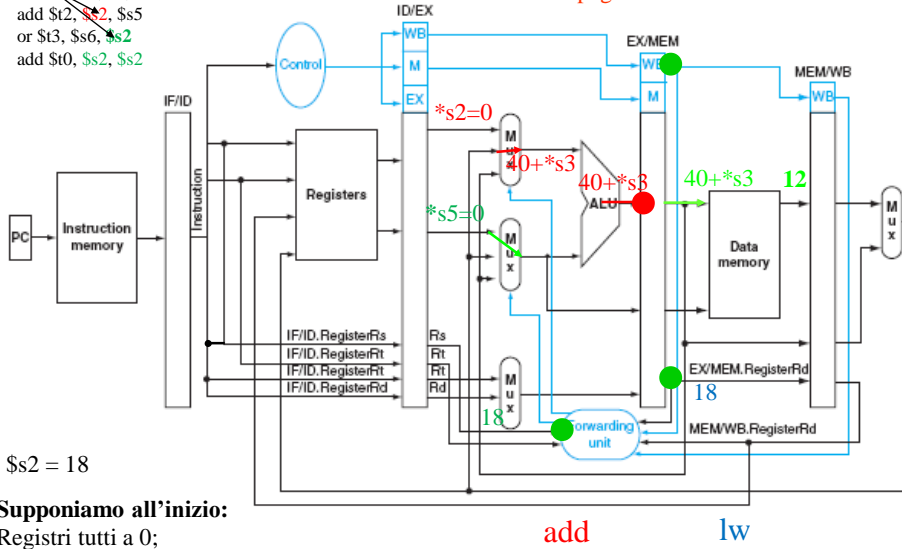


La criticità sulla prima add



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

Propaga dalla fase di memoria



\$s2 = 18

Supponiamo all'inizio:

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

11/40

Il bypass non risolve!



Hazard sui dati: lw, rilevamento della criticità



lw \$s2, 40(\$s3)	IF	ID	EX \$s3+ 40	MEM <\$s3+40>	WB s->\$s2			
add \$t2, \$s2, \$s5		IF	ID	EX \$s2 and \$s5	MEM s->\$t2			
or \$t3, \$s6, \$s2			IF	ID	EX \$s6 or \$s2	MEM	WB (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 ₀	t ₁	t ₂	t ₃	t ₄	t ₅	t ₆	t ₇
.....								
lw \$s2, 40(\$s3)	FF (Mem, ALU)	DECOD (RF)	EXEC (ALU)	MEM (MEM)	WB (RF)			
nop (add \$t2, \$s2, \$s5)		bolla (FF)	Bolla (DEC)	bolla (EXEC)	bolla (MEM)	bolla (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 1 ciclo di clock **per attendere che la lw abbia caricato dalla memoria il valore corretto di \$s2** e sia quindi disponibile in pipeline. Devo **quindi bloccare anche le istruzioni successive per 1 ciclo di clock.**

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.

A.A. 2024-2025

15/40

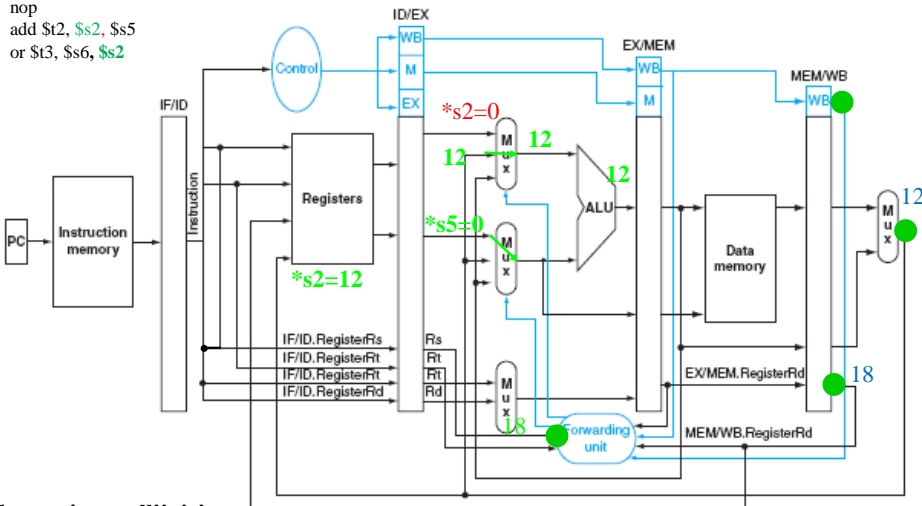
<http://borghese.di.unimi.it>



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?

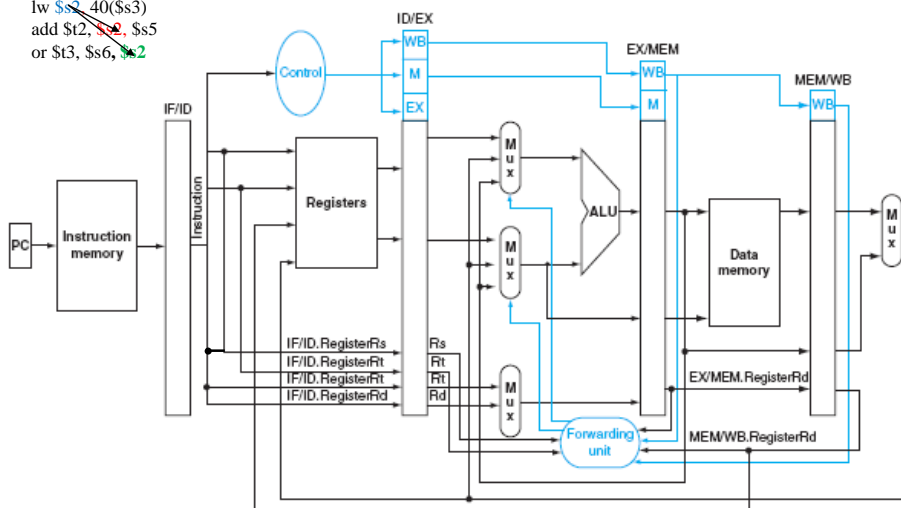
- 1) Rilevazione
- 2) Soluzione



Rilevazione della criticità



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



“Se l’istruzione corrente è una lw e uno dei registri operando dell’istruzione successiva è uguale al registro di scrittura della lw (rt), allora occorre inserire una nop”



Analisi della dipendenza - DEC

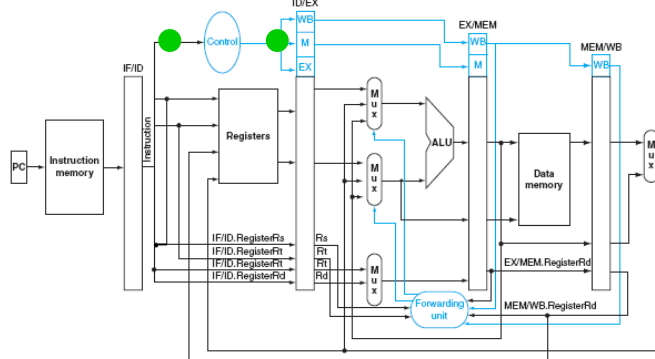


“L’istruzione corrente è una lw” – in fase di DEC

- Codice operativo
- Segnale di controllo caratteristico: MemRead = 1.

Entrambi i segnali vengono generati nella fase DECODE («the sooner the better»).

- MemRead su 1 bit e viene già trasportato in pipeline e copre tutte le varianti di load
- Codice operativo su 6 bit si ferma nella fase di DECODE.



Sommario



Identificazione delle criticità che richiedono stallo

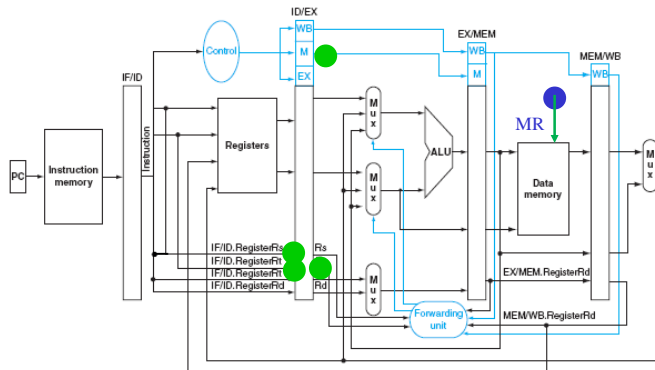
Soluzione della criticità mediante stallo



Analisi della dipendenza - EXE



Posso capire che si tratta di un'istruzione di lw nelle fasi DEC, EX, MEM.
 La criticità riguarda la lw e l'istruzione successiva. L'istruzione successiva deve essere almeno nella fase di DEC.
 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”



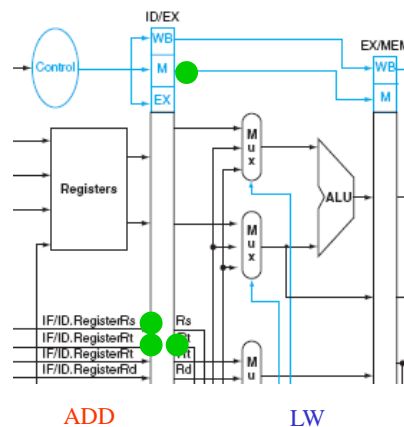
Implementazione del rilevamento della criticità della lw



IF (ID/EX.MemRead)
 AND
 [(IF/ID.RegistroRt == ID/EX.RegistroRt) OR
 (IF/ID.RegistroRs == ID/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**).





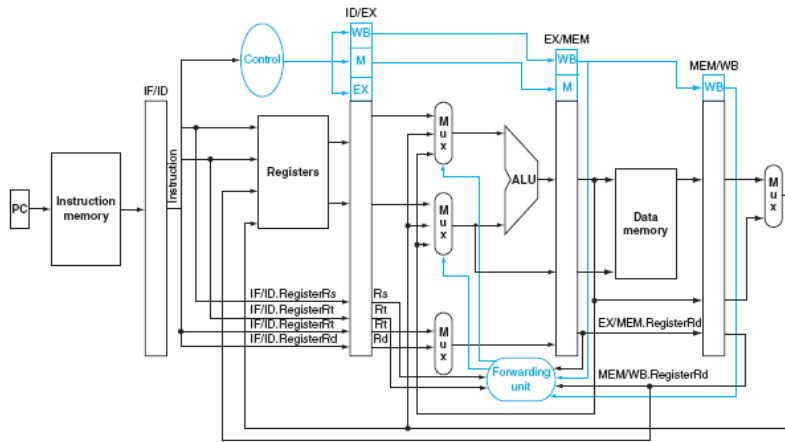
Perché è necessario controllare Mread?



sub \$s2, \$s3, \$s2
 add \$t2, \$s2, \$s5
 or \$t3, \$s6, \$s2

sub \$s7, \$s3, \$s2
 add \$t2, \$s2, \$s5
 or \$t3, \$s6, \$s2

Non presentano stalli o sono risolvibili con la propagazione



Implementazione dello stallo



IF (ID/EX.MemWrite)
 AND

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

THEN "Stallo della Add"

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

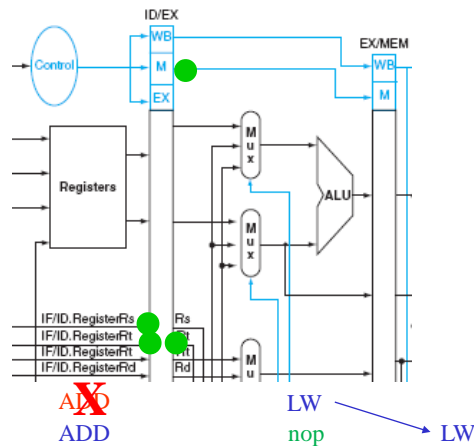
Occorre inserire una nop in fase di Exe, tra la lw e la add.

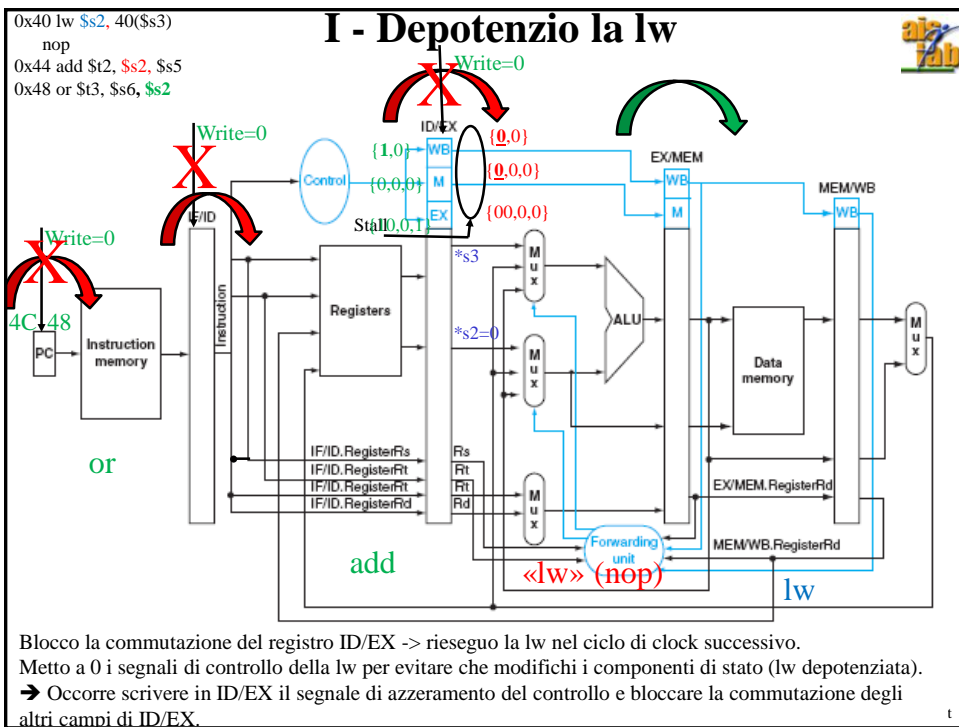
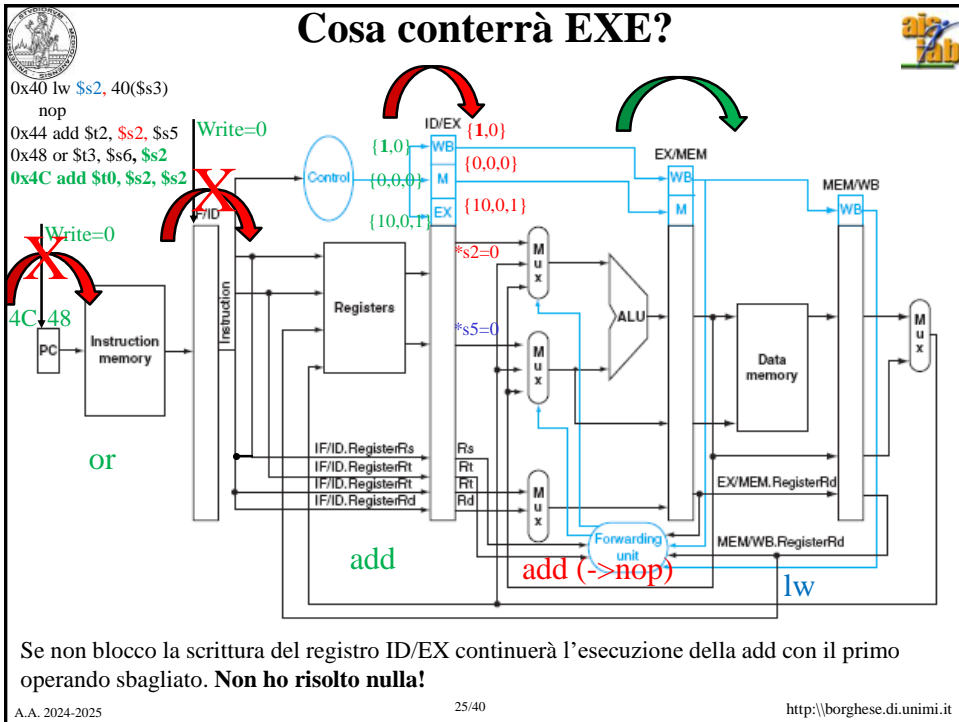
- Lascio continuare l'esecuzione della lw
- Blocco per un ciclo di clock l'esecuzione della add (e delle istruzioni successive)

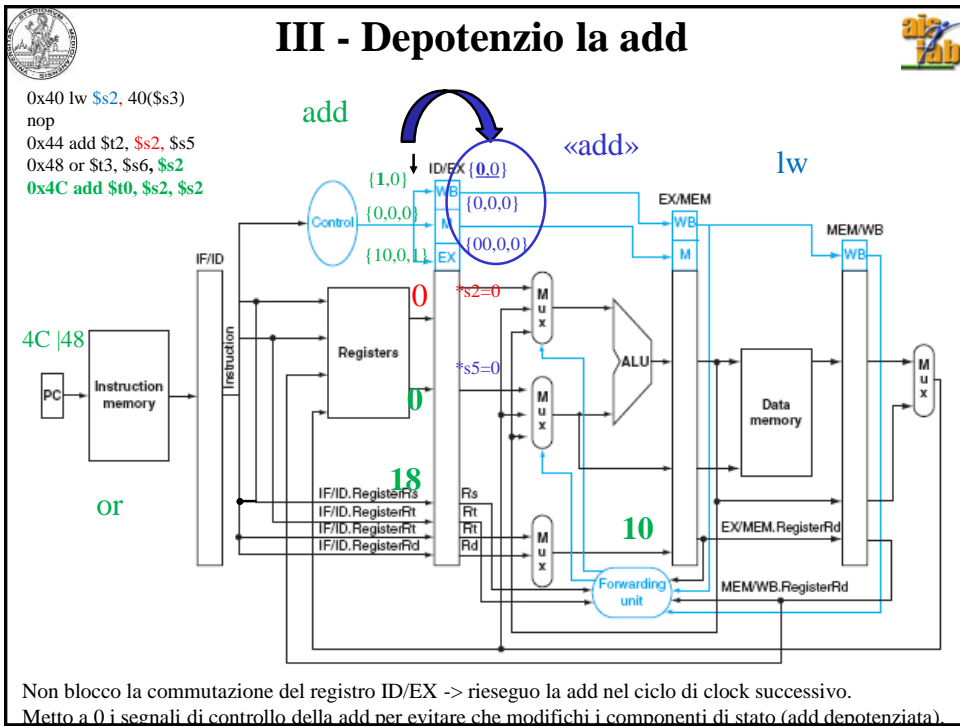
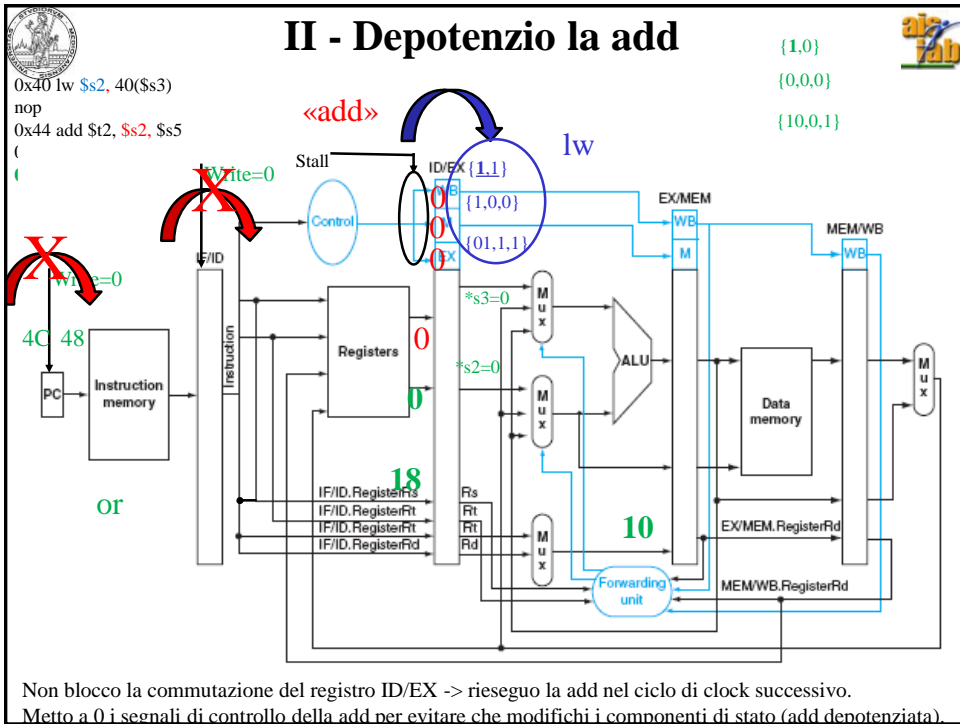
Inserisco la nop il prima possibile, tra la fase di MEM della lw e la fase di ID della add (*modifico il codice*).

Come inserisco una nop?

Time 1
 Time 2

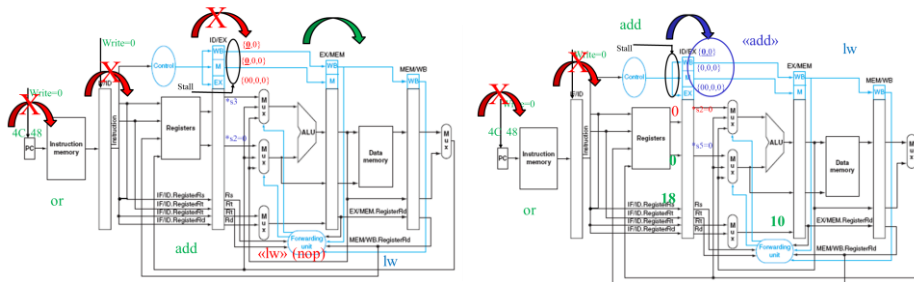








Soluzione per lo stallo



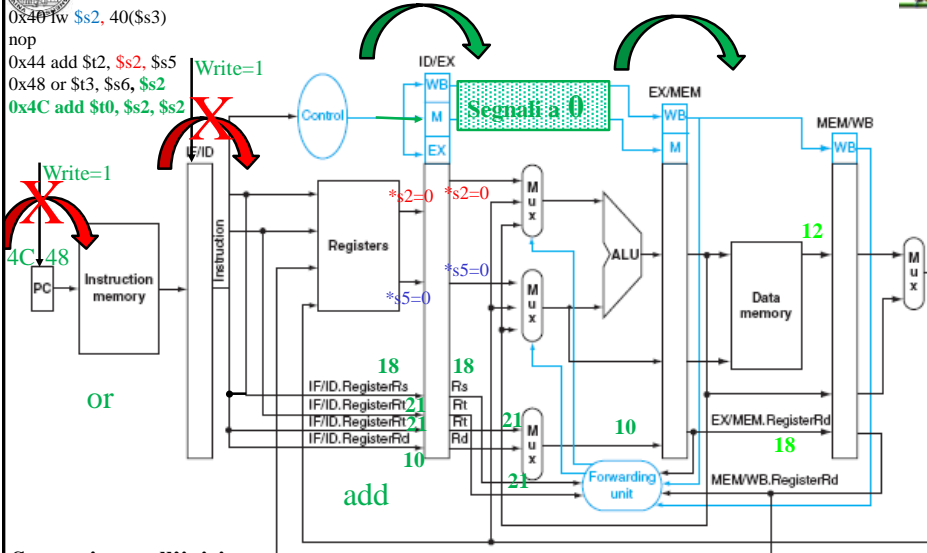
Depotenzio la lw

Depotenzio la add

Decisione migliore perchè presa un ciclo di clock prima e implementazione più semplice.



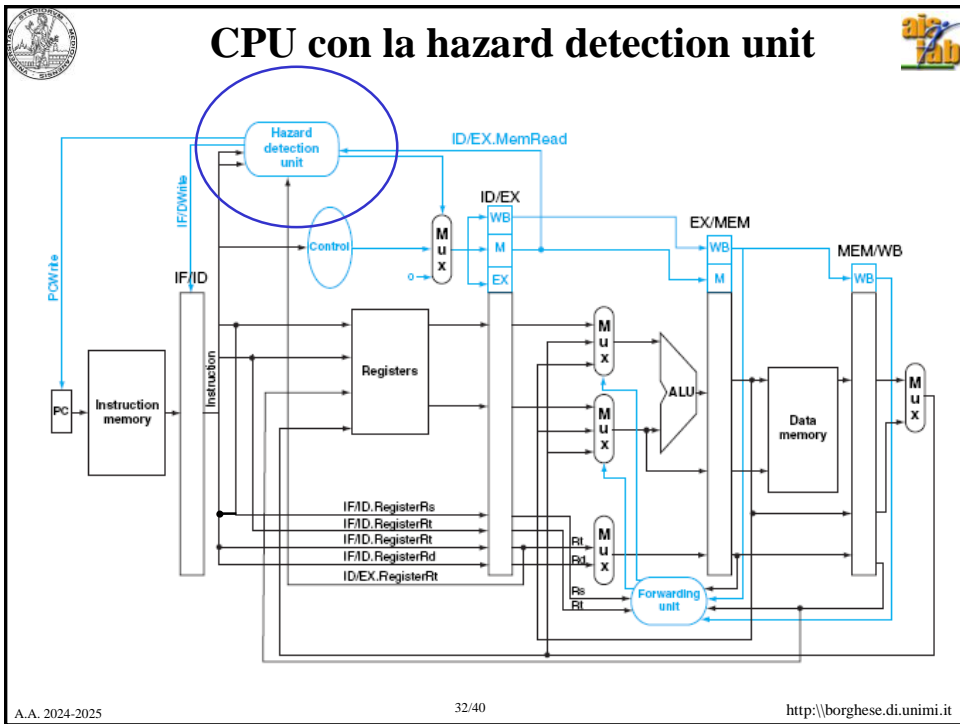
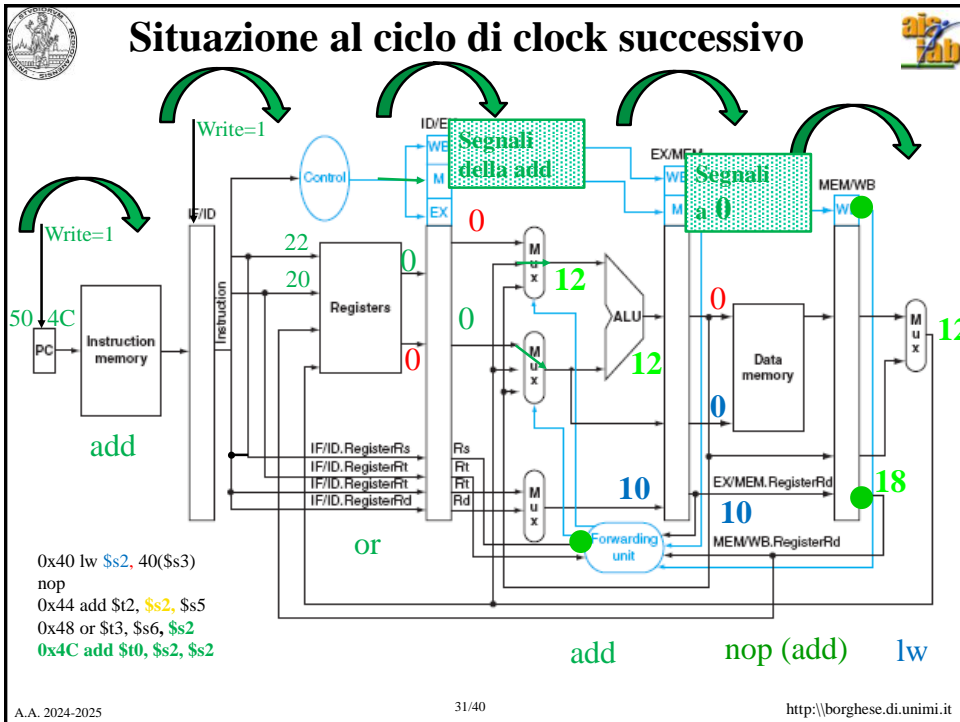
add "regolare" e add depotenziata



Supponiamo all'inizio:

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

nop ("add") lw





Hazard nei dati: soluzioni



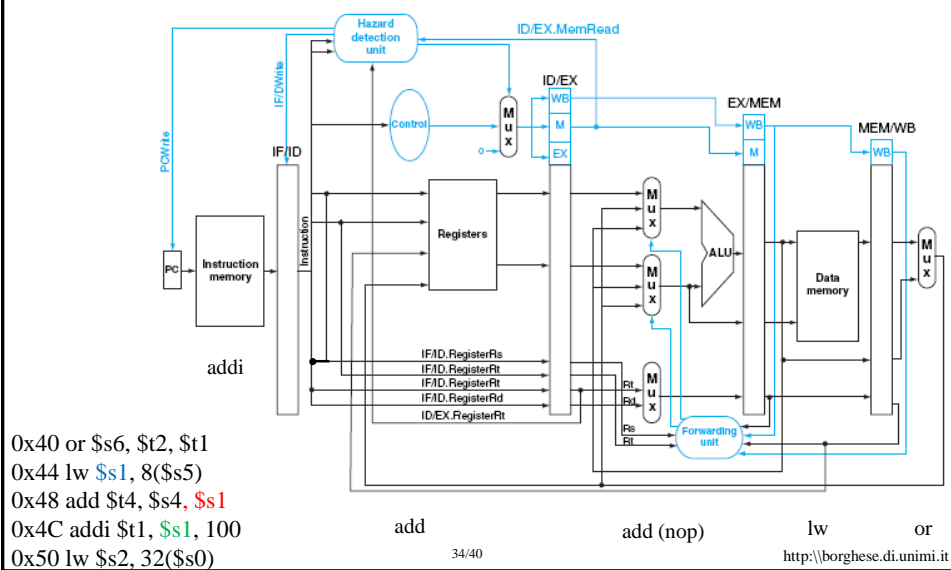
- Buona scrittura del codice (il programmatore deve conoscere la macchina per scrivere un buon codice!).
- Compilatore efficiente (che riordini il codice).
- Architettura che renda disponibile i dati appena pronti alla fase di esecuzione.
- Accettare uno stallo (non sempre si può evitare).



Esempio



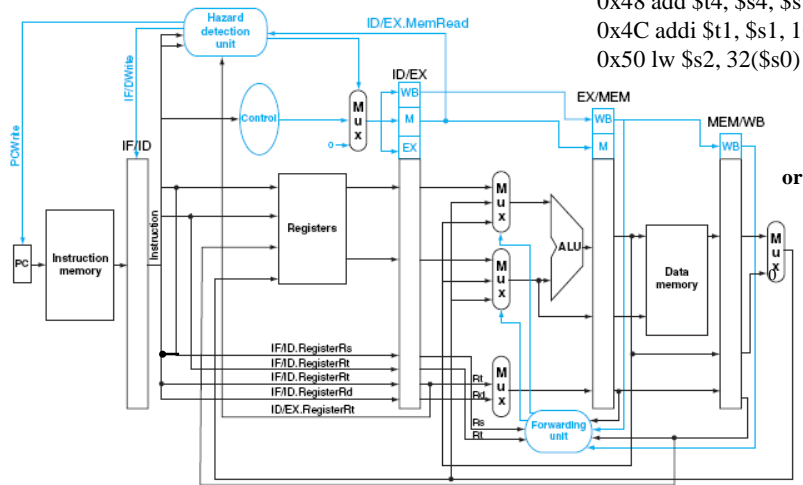
Specificare il contenuto di tutti i bus di questa pipeline quando è in esecuzione il seguente codice con l'istruzione or in WB:





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)

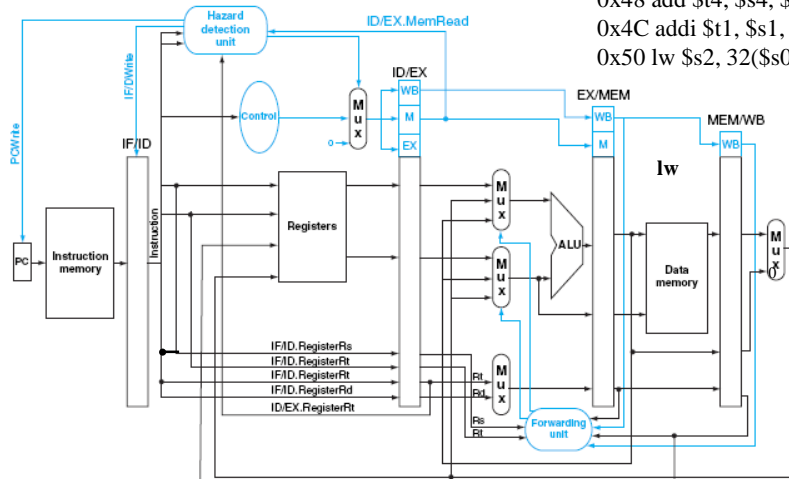


MEM/WB.mdr = ?
 MEM/WB.data = $t2 \parallel t1$
 MEM/WB.RegScrittura = 22 = \$s6
 MEM/WB.RegWrite = 1;
 MEM/WB.Mem2Reg = 0



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;

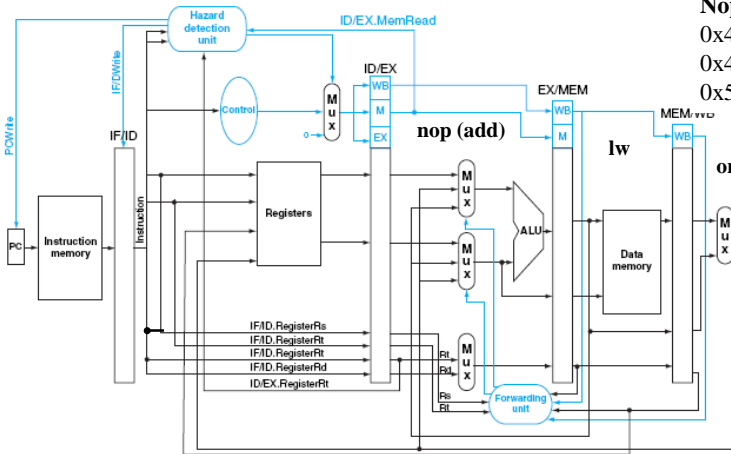
MDR = $MEM[*s5+8]$
 EX/MEM.RegScrittura = 17 = \$s1
 EX/MEM.MemW = 0
 EX/MEM.Mem2Reg = 1

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

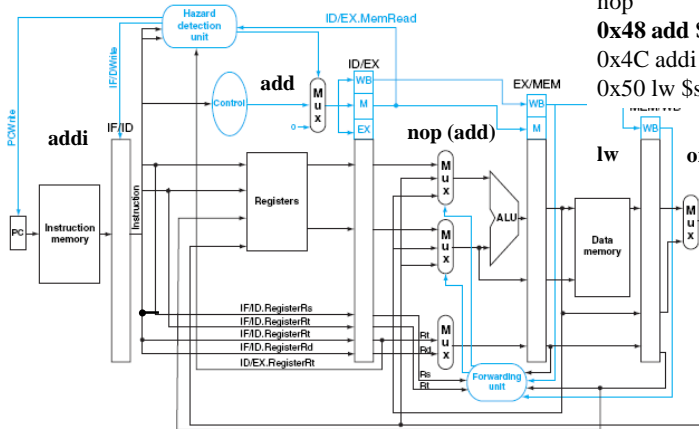
RegDst = 0

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)
```

Estendere la CPU perchè possa gestire hazard tra una `lw` e una `sw` e tra una `add` e una `addi`.



Sommario



Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo