



Pipeline – criticità e forwarding

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson: 4.5, 4.6



Sommario

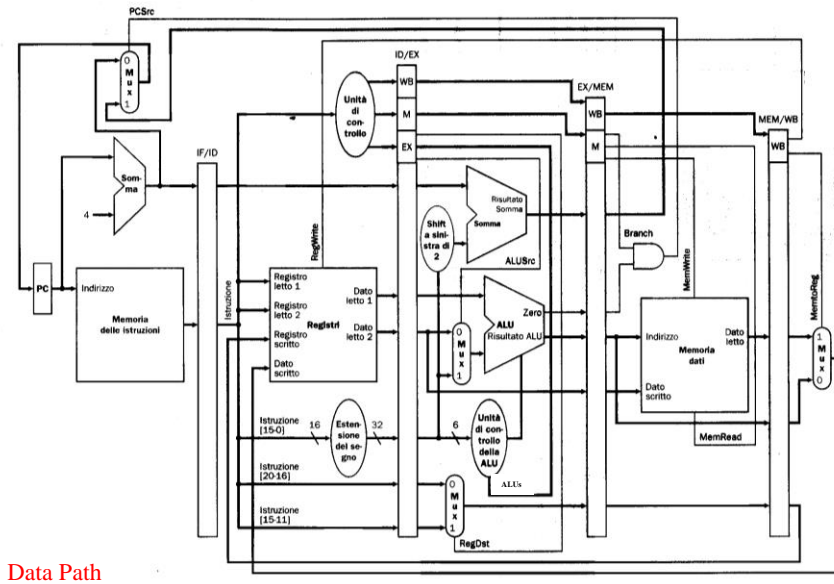
Criticità in una pipeline

Hazard sui dati

Propagazione



Pipeline - struttura



Data Path



Gli stadi di esecuzione



IF – Instruction Fetch

ID – Instruction Decode (e lettura register file)

EX – Esecuzione o calcolo dell'indirizzo di memoria.

MEM – Accesso alla memoria dati.

WB – Write Back (scrittura del risultato nel register file).

NB: I registri al termine di ogni fase prendono il nome dalle 2 fasi:

IF/ID	ID/EX	EX/MEM	MEM/WB
-------	-------	--------	--------

Perchè non c'è un registro WB/IF?

Il data-path e il control path procedono da sx a dx.



Il ruolo dei registri di pipeline



Ciascuno stadio produce un risultato. La parte di risultato che serve agli stadi successivi deve essere memorizzata in un registro insieme alle altre informazioni utili per l'esecuzione da quello stadio **fino alla fine**.

Il registro mantiene l'informazione anche se lo stadio in questione riutilizza l'unità funzionale.

Esempio: l'istruzione letta viene salvata nel registro IF/ID (cf. Instruction Register) e lì rimane per un ciclo di clock anche se nella fase di fetch transita un'altra istruzione.



Criticità (hazard)



Un'istruzione non può essere eseguita nel ciclo di clock immediatamente successivo a quello dell'istruzione precedente (mancano i dati necessari alla lavorazione di un qualche suo stadio).

Strutturali:

- Dovrei utilizzare la stessa unità funzionale due volte nello stesso ciclo di clock (e.g. se non avessi duplicato la ALU nella fase di EXE).

Controllo:

- Dovrei prendere una decisione (sull'istruzione successiva) prima che l'esecuzione dell'istruzione corrente sia terminata (e.g. Istruzioni successive a una branch).

Dati:

- Dovrei eseguire un'istruzione in cui uno dei dati è il risultato dell'esecuzione di un'istruzione precedente.



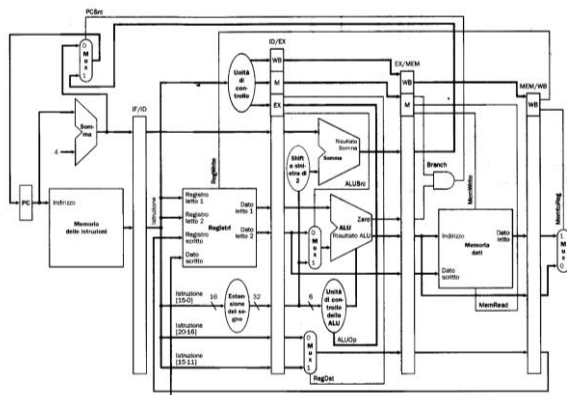
Soluzione delle criticità strutturali



Le criticità strutturali sono risolte con la duplicazione (suddivisione) delle unità funzionali.

Triplicazione delle ALU

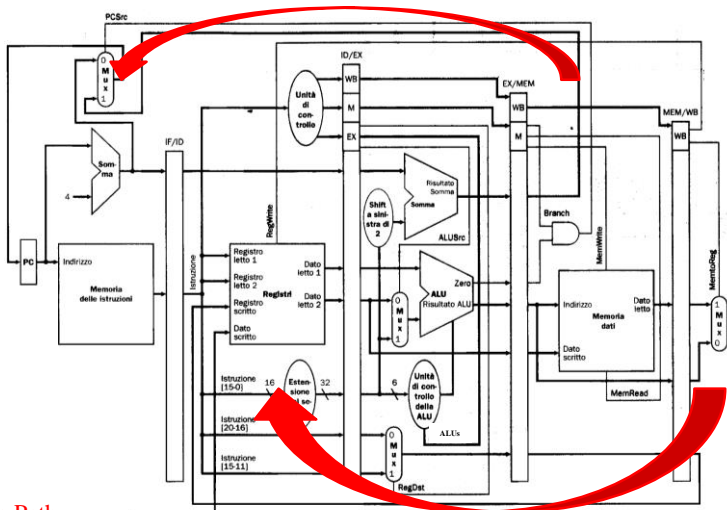
Duplicazione della Memoria (stessa memoria ma separazione della memoria dati dalla memoria istruzioni).



A.A. 2020-2021



Pipeline – criticità o hazard



Data Path

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

A.A. 2020-2021

borghese.di.unimi.it



Sommario



Criticità in una pipeline

Hazard sui dati

Propagazione



Esempio di hazard sui dati



In linguaggio C:

$s0 = t3 - (t1 + t2);$

In linguaggio assembler:

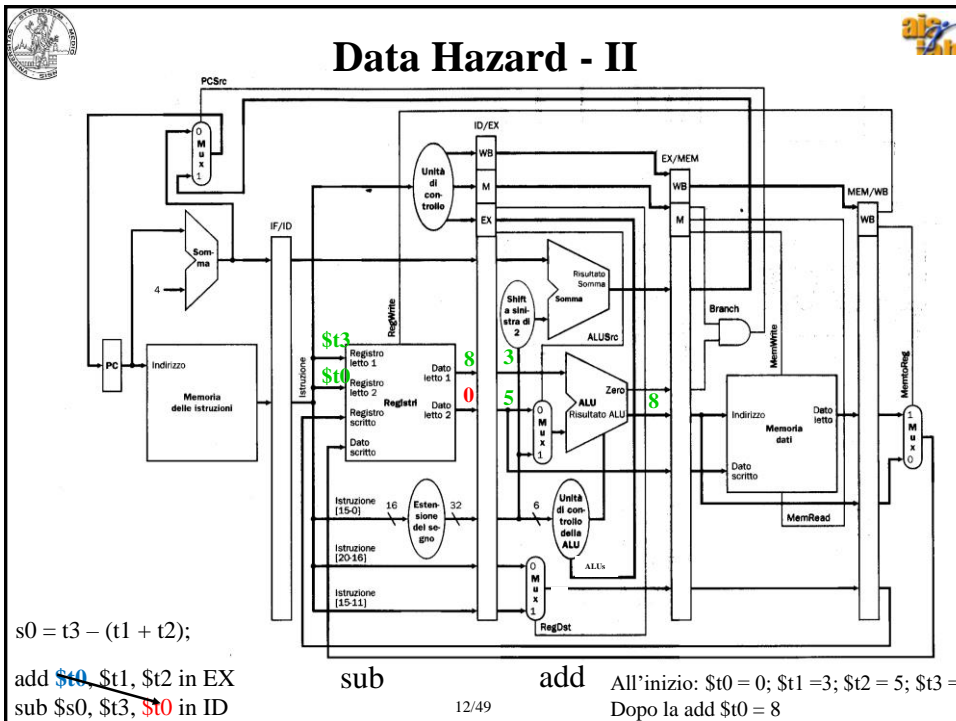
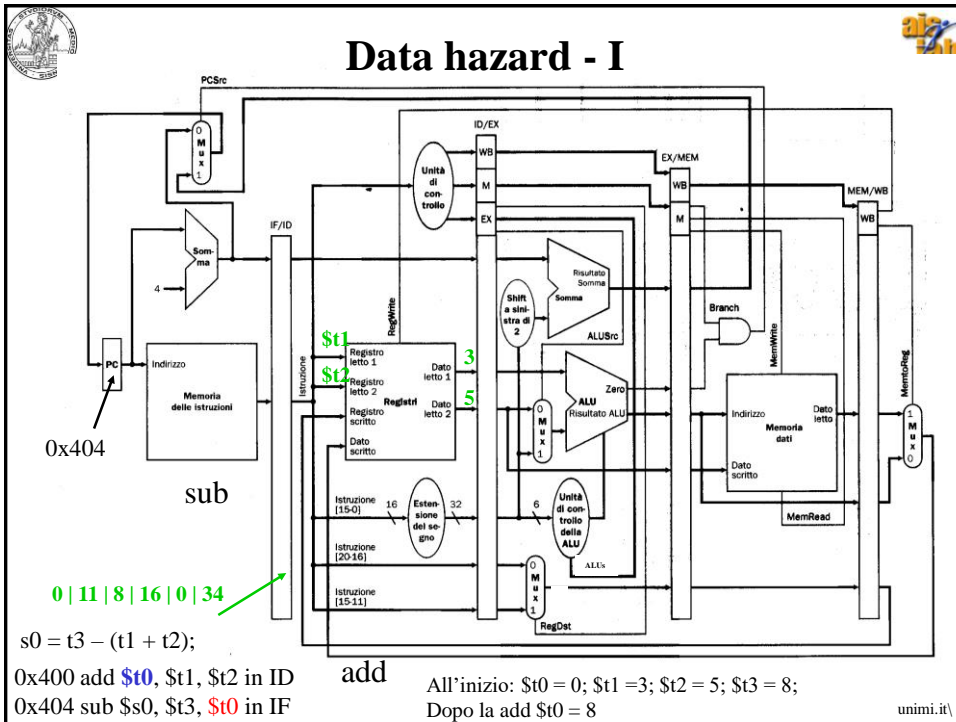
add \$t0, \$t1, \$t2

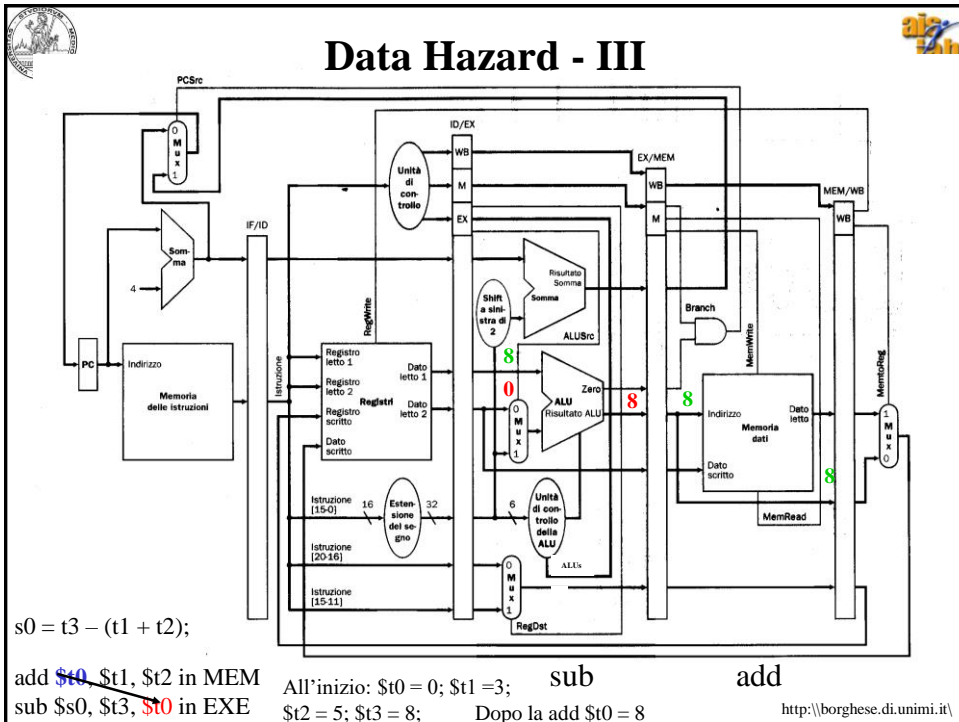
sub \$s0, \$t3, \$t0

Supponiamo all'inizio:

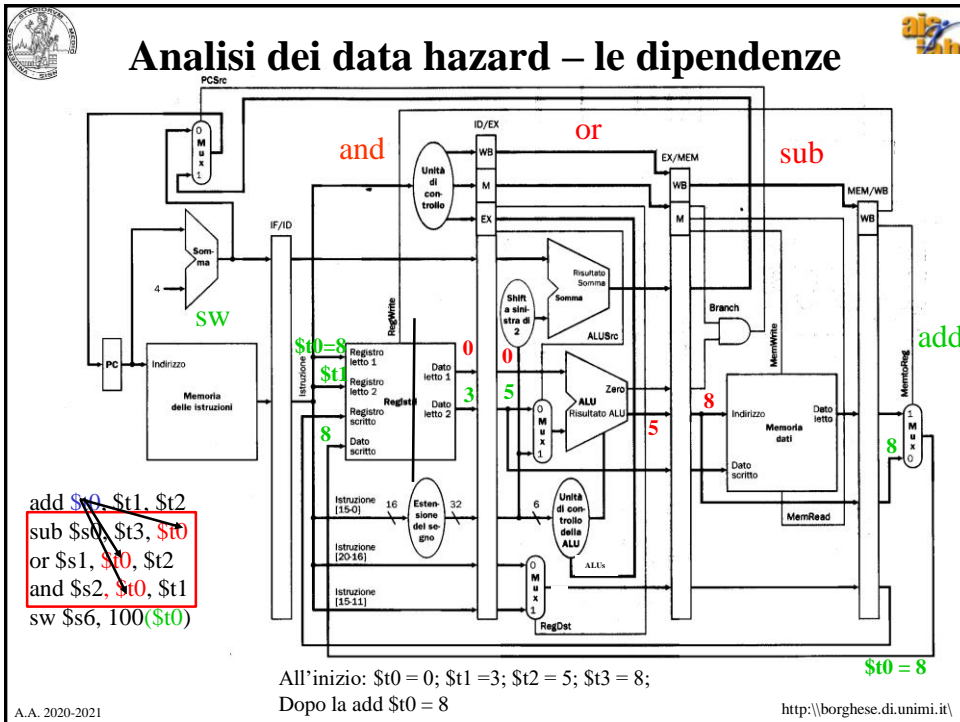
$\$t0 = 0; \$t1 = 3; \$t2 = 5; \$t3 = 8;$

Alla fine deve risultare: $\$s0 = 8 - (3+5) = 0;$





- ## Come affrontare gli hazard
- Si può risolvere l'hazard...
 - ...*aspettare*
 - si mette lo stadio 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.**
 - Es: 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 valore prodotto dall'istruzione – "value prediction"). Se la predizione si rivela corretta: tutto ok. Se si rivela sbagliata: **roll-back.**



Data Hazard – le dipendenze

add \$t0, \$t1, \$t2	IF	ID	EX t1+t2	MEM	WB write t0				
sub \$s0, \$t3, \$t0		IF	ID	EX \$t3 - \$t0	MEM	WB			
or \$s1, \$t0, \$t2			IF	ID	EX \$t0 or \$t2	MEM	WB		
and \$s2, \$t0, \$t5				IF	ID	EX \$t0 & \$t5	MEM	WB	
sw \$s6, 100(\$t0)					IF	ID	EX \$t0 +100	MEM	WB

Le dipendenze sono **critiche (hazard)** quando il dato serve prima nel tempo di quando viene scritto nel RF

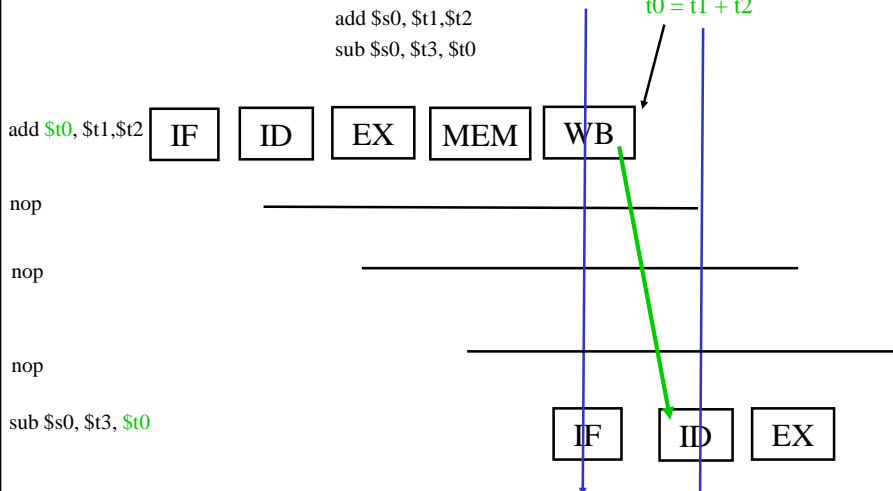
f)



Soluzione 1 – «aspettare» - stallo



Esempio:



Non carico istruzioni per 3 cicli di clock, la pipeline è **in stallo** per 3 cicli di clock, si inseriscono 3 bolle (**bubbles**) nella pipeline. E' una soluzione costosa in termini di throughput.



Soluzione 2 – «prevenire gli hazard» - riorganizzando il codice

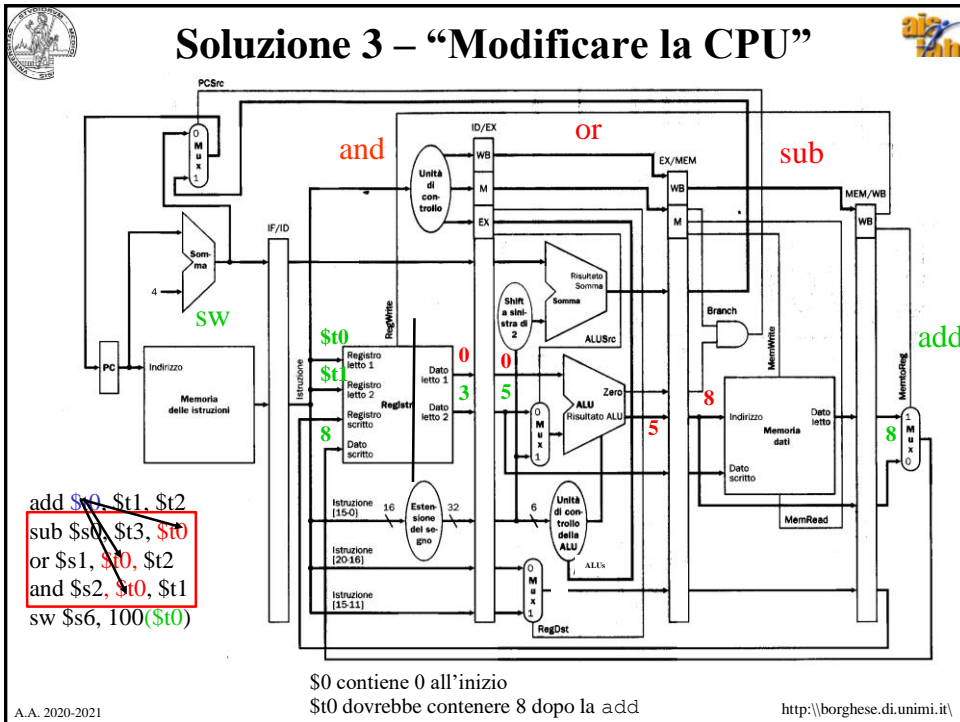


<p>add \$t0, \$t1, \$t2 sub \$s0, \$t3, \$t0 or \$s1, \$t0, \$t2 and \$s2, \$t0, \$t1 sw \$s6, 100(\$t0) add \$s5, \$t1, \$t3 addi \$s3, \$t4, 100 or \$s4, \$t5, \$t6</p>	<p>add \$t0, \$t1, \$t2 nop nop nop nop sub \$s0, \$t3, \$t0 or \$s1, \$t0, \$t2 and \$s2, \$t0, \$t1 sw \$s6, 100(\$t0) add \$s5, \$t1, \$t3 addi \$s3, \$t4, 100 or \$s4, \$t5, \$t6</p>	<p>add \$t0, \$t1, \$t2 add \$s5, \$t1, \$t3 nop addi \$s3, \$t4, 100 or \$s4, \$t5, \$t6 sub \$s0, \$t3, \$t0 or \$s1, \$t0, \$t2 and \$s2, \$t0, \$t1 sw \$s6, 100(\$t0)</p>
--	--	--

Con lo stallo spreco di 3 cicli di clock (in modo che la fase IF dell'istruzione *sub \$s0, \$t3, \$t0* vada a coincidere con la fase di WB della *add \$s0, \$t1, \$t2*). **Situazione troppo frequente perché la soluzione sia accettabile.**

Il codice viene riorganizzato in fase di compilazione. Non sempre è possibile o efficace.

Esecuzione fuori ordine.



Data Hazard – le dipendenze

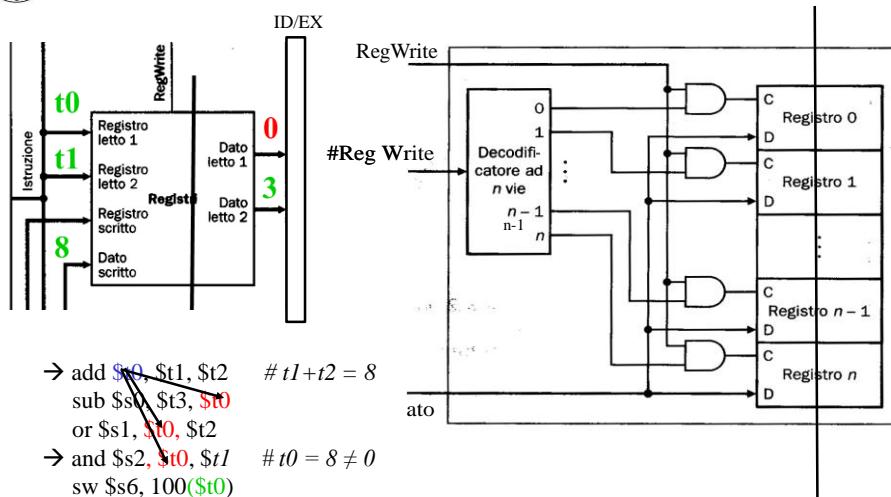
add \$t0, \$t1, \$t2	IF	ID	EX t1+t2	MEM	WB write t0				
sub \$s0, \$t0, \$t3		IF	ID	EX \$2 and \$5	MEM	WB s->\$t2			
or \$s1, \$t2, \$t0			ID	EX \$6 or \$2	MEM	WB (s->\$t3)			
and \$s2, \$t0, \$t5				IF	ID	EX \$2 & \$3	MEM	WB s->\$t4	
sw \$s6, 100(\$t0)					IF	ID	EX \$2+100	MEM \$t5	WB ->Mem

Il dato in \$s2 viene scritto nel Register File nella fase di WB della sub, è pronto al clock successivo. Non è ancora pronto quando viene effettuata la decodifica della and, della or e della add successiva.

ii.it



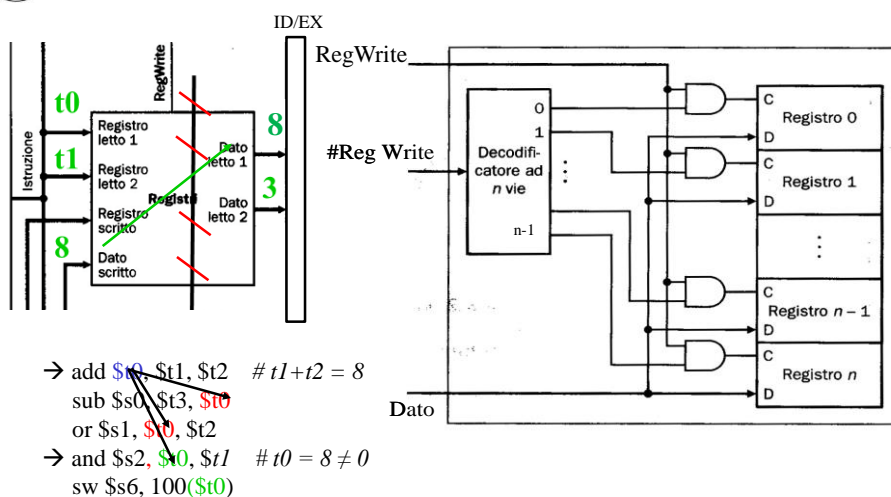
Analisi dell'hazard sulla and



Il contenuto del registro letto \$t0 si ferma nel registro ID/EX
 Il dato scritto si ferma nella "pancia" dei flip-flop.



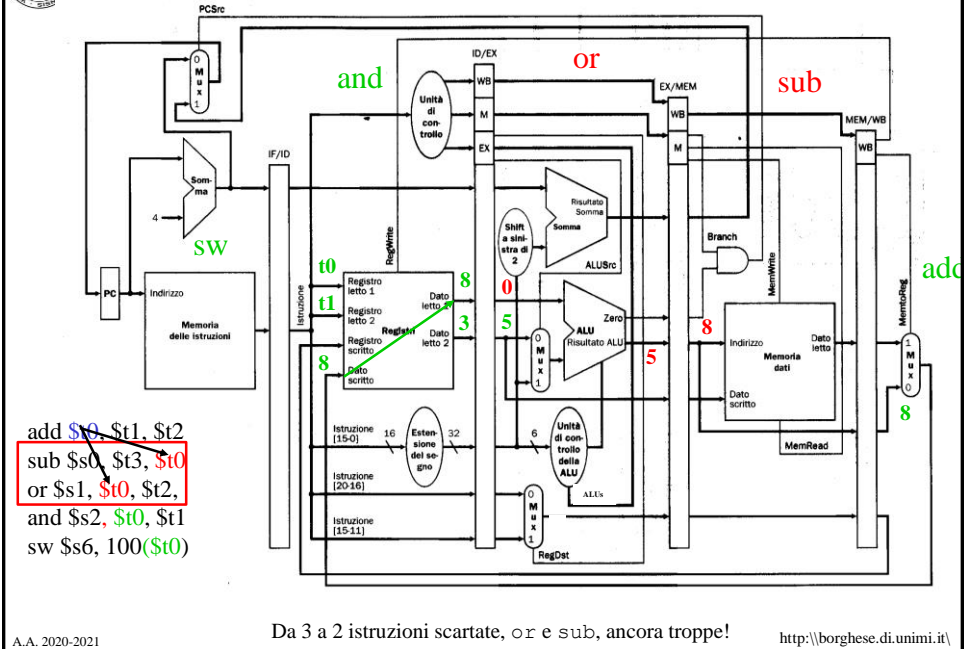
Soluzione dell'hazard sulla and



Il contenuto aggiornato di \$t0 (=8) è già nel register file (nella parte master). E' sufficiente lasciarlo passare in uscita. Dobbiamo utilizzare **latch** di tipo D e non flip-flop come nel caso della CPU singolo ciclo



Soluzione 3 – “Modificare la CPU”



Sommario



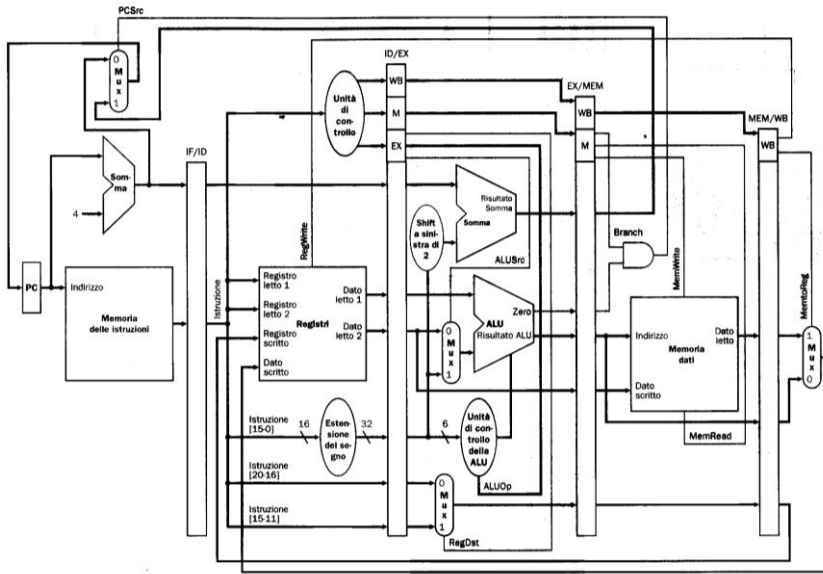
Criticità in una pipeline

Hazard sui dati

Propagazione



CPU con pipeline



Data Hazard – le dipendenze



add \$t0, \$t1, \$t2	IF	ID	EX t1+t2	MEM	WB write t0				
sub \$s0, \$t0, \$t3		IF	EX \$2 and \$5	MEM	WB s->\$t2				
or \$s1, \$t2, \$t0			IF	ID \$6 or \$2	EX \$6 or \$2	MEM	WB (s->\$t3)		
and \$s2, \$t0, \$t5				IF	ID \$2 & \$3	EX \$2 & \$3	MEM	WB s->\$t4	
sw \$s6, 100(\$t0)					IF	ID	EX \$2+100	MEM \$t5	WB ->Mem

Le dipendenze sono **critiche (hazard)** quando il dato serve prima nel tempo



Soluzione architetturale della criticità sui dati



La criticità nei dati ha a che fare essenzialmente con la **disponibilità di dati corretti**.

- 1) **Identificazione** della criticità (funzione del tipo di istruzione e dei registri coinvolti).
- 2) **Correzione** della situazione: **propagazione a ritroso** dei dati richiesti (negli stadi della pipeline = in avanti nel tempo) *su data-path alternativi*



Data Hazard – analisi



add \$t0, \$t1, \$t2	IF	ID	EX t1+t2	MEM	WB write t0				
sub \$s0, \$t0, \$t3		IF	ID	EX \$2 and \$5	MEM	WB s->\$t2			
or \$s1, \$t2, \$t0			IF	ID	EX \$6 or \$2	MEM	WB (s->\$t3)		
and \$s2, \$t0, \$t5				IF	ID	EX \$2 & \$3	MEM	WB s->\$t4	
sw \$s6, 100(\$t0)					IF	ID	EX \$2+100	MEM \$t5	WB ->Mem

Diagram annotations: A red arrow points from the MEM stage of the first instruction to the ID stage of the second instruction. A green arrow points from the WB stage of the first instruction to the ID stage of the fifth instruction. A blue dashed vertical line is drawn through the MEM stage of the first instruction, with a blue arrow pointing down to the text "\$t0 corretto" in the MEM stage of the fifth instruction.

Alla fine della fase di EXE nella pipeline e' presente il valore corretto di \$t0 che e' \$t1 + \$t2

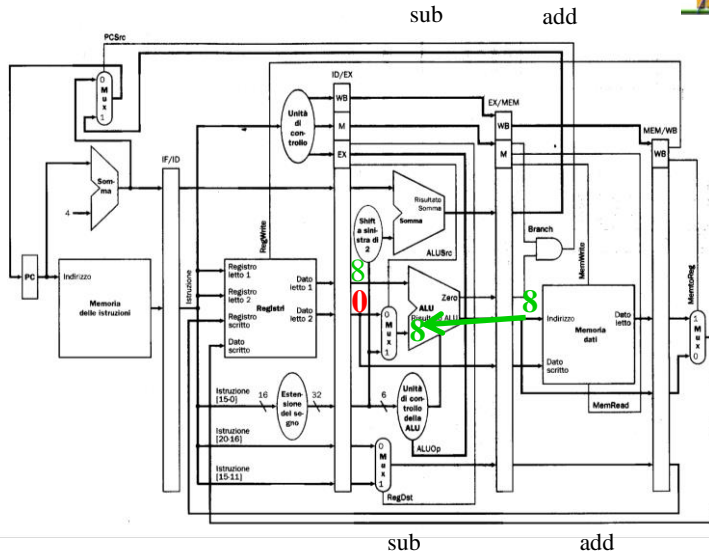


Soluzione della criticità in MEM



add \$t0, \$t1, \$t2
sub \$s0, \$t3, \$t0

*\$t1 + *\$t2 = 8



Cammino di **retro-propagazione** (feed-forward nel tempo) del risultato della ALU. Il cammino non deve essere utilizzato sempre.

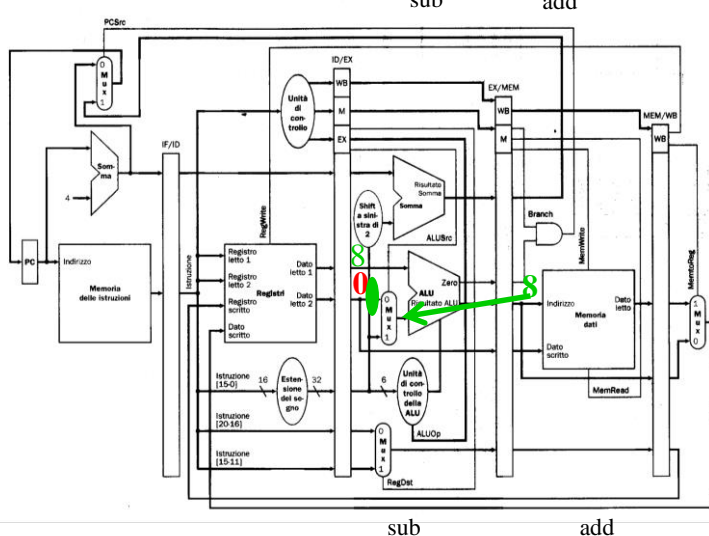


Propagazione da MEM



add \$t0, \$t1, \$t2
sub \$s0, \$t3, \$t0

*\$t1 + *\$t2 = 8



Operando 2 = ID/EX.rt (if not hazard)
EX/MEM.rd (if hazard)

Chi controlla il MUX_B? Come?

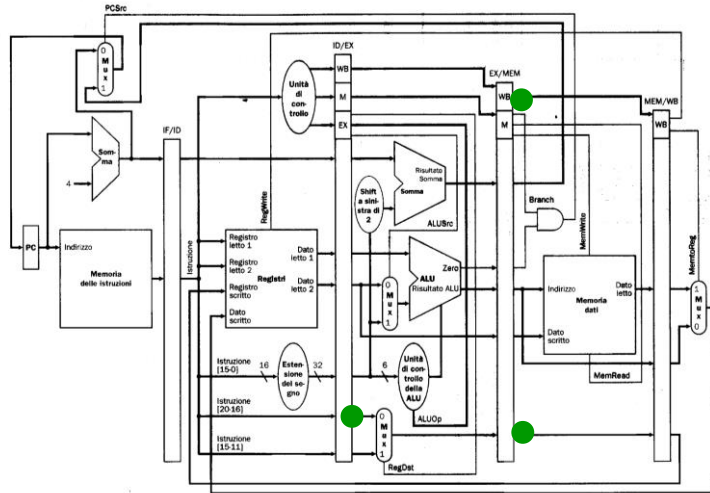


Identificazione delle criticità in MEM



add \$t0, \$t1, \$t2
sub \$s0, \$t3, \$t0

if (EX/MEM.RegistroRd = ID/EX.RegistroRt) and
(EX/MEM.RegWrite == 1) then
“Hazard on operando2”

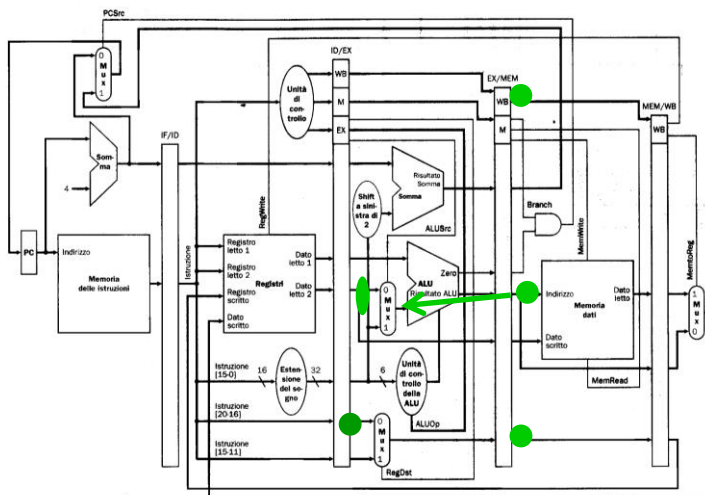


Putting the picture together



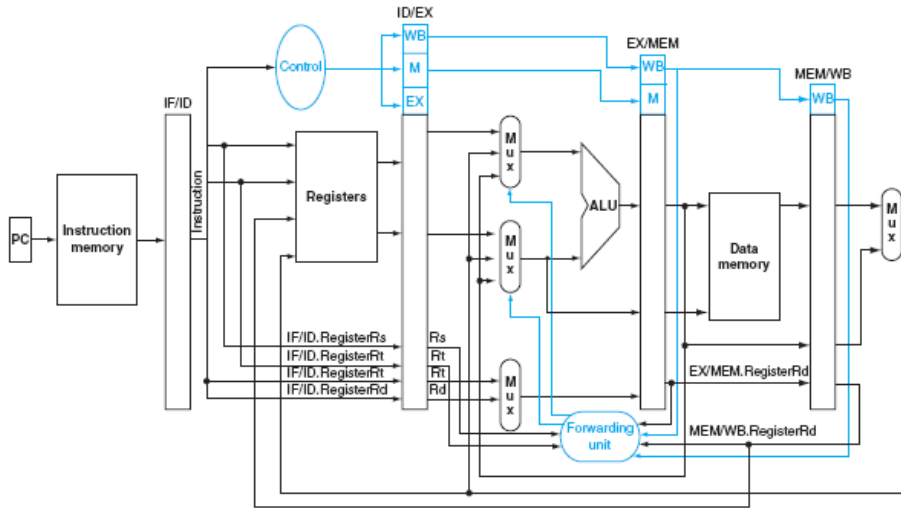
add \$t0, \$t1, \$t2
sub \$s0, \$t3, \$t0

IF ((EX/MEM.RegistroRd == ID/EX.RegistroRt) &&
(EX/MEM.RegisterWrite)) then operando2 = EX/MEM.data;

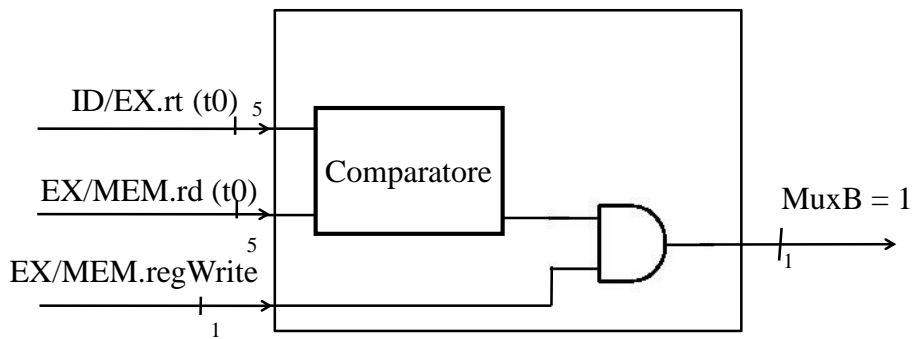




CPU con unità di propagazione



Unità di controllo della propagazione



IF ((MEM/WB.RegistroRd == ID/EX.RegistroRt) &&
 (MEM/WB.RegisterWrite) Then MuxB = 1; # Operando2 = EX/MEM.data

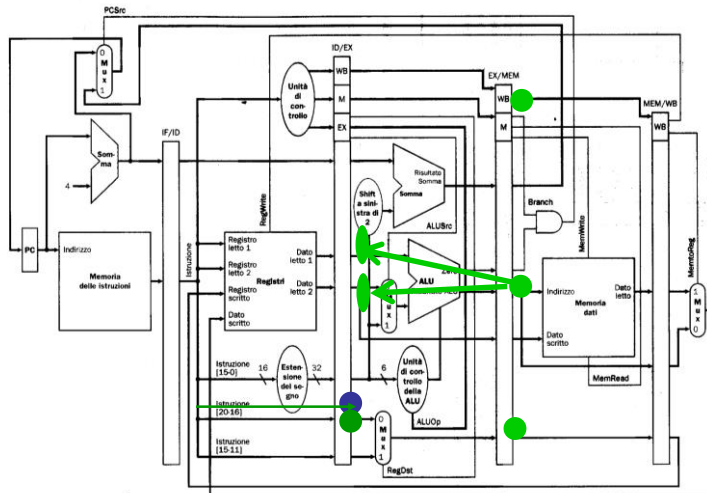


Punti salienti per un hazard in MEM



add \$t0, \$t1, \$t2
sub \$s0, \$t0, \$t0

IF ((EX/MEM.RegistroRd == ID/EX.RegistroRs) &&
(EX/MEM.RegisterWrite) then **operando1 = EX/MEM.data;**
IF ((EX/MEM.RegistroRd == ID/EX.RegistroRt) &&
(EX/MEM.RegisterWrite) then **operando2 = EX/MEM.data;**



A.A. 2020-2021

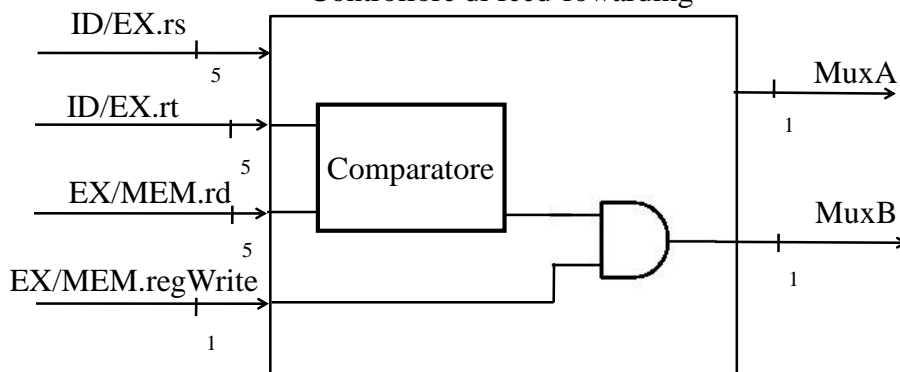
mi.it\



Unità di controllo della propagazione nello stadio MEM



Controllore di feed-forwarding



A.A. 2020-2021

36/49

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

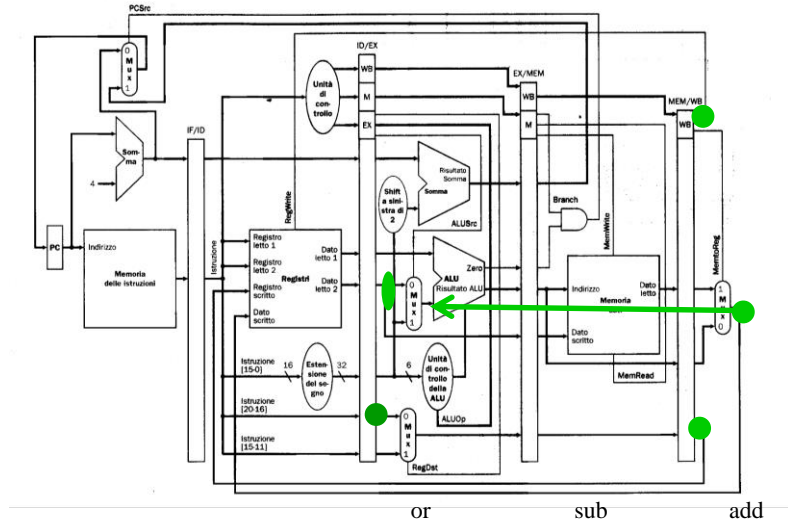


Punti salienti per un hazard in WB



add \$t0, \$t1, \$t2
 sub \$s0, \$t0, \$t3
 or \$s1, \$t2, \$t0

IF ((MEM/WB.RegistroRd == ID/EX.RegistroRt) &&
 (MEM/WB.RegisterWrite)) then **operando2 = MEM/WB.data;**

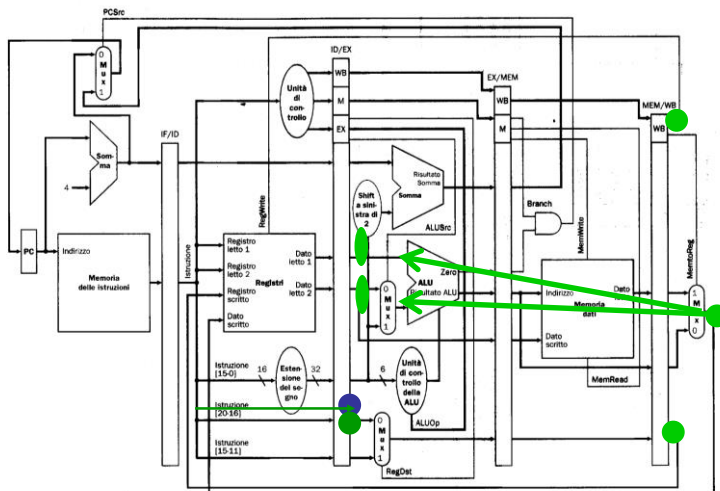


Propagazione completa da WB



add \$t0, \$t1, \$t2
 sub \$s0, \$t0, \$t3
 or \$s1, \$t0, \$t0

IF ((MEM/WB.RegistroRd == ID/EX.RegistroRs) &&
 (MEM/WB.RegisterWrite)) then **operando1 = MEM/WB.data;**
 IF ((MEM/WB.RegistroRd == ID/EX.RegistroRt) &&
 (MEM/WB.RegisterWrite)) then **operando2 = MEM/WB.data;**





Relazione tra forwarding e contenuto del registro ID/EX



Nel normale funzionamento, il registro ID/EX contiene quanto letto dal Register File.

Quando abbiamo forwarding, quello che viene letto dal registro ID/EX nella **fase di esecuzione** viene sovrascritto da quanto letto dal registro EX/MEM o MEM/WB.

Nel registro EX/MEM è contenuto il risultato dell'operazione eseguita all'istante precedente.

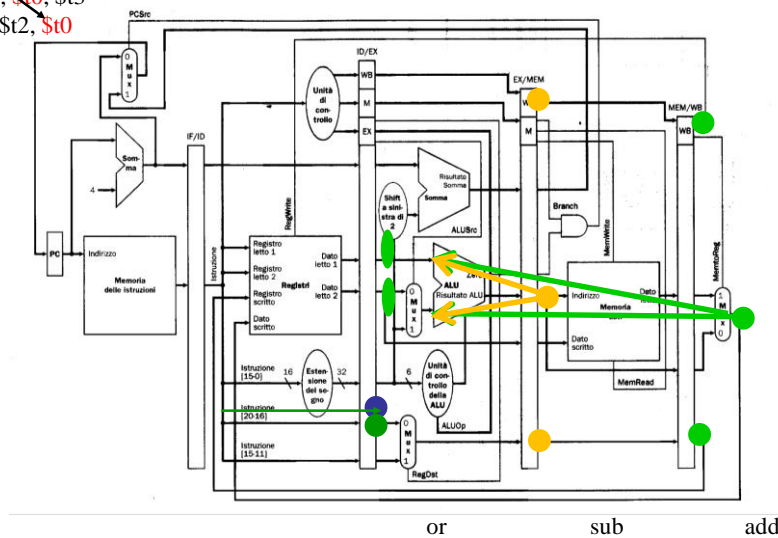
Nel registro MEM/WB è contenuto il risultato dell'operazione eseguita 2 istanti precedenti.



Punti salienti per un hazard in MEM o WB



add \$t0, \$t1, \$t2
 sub \$s0, \$t0, \$t3
 or \$s1, \$t2, \$t0





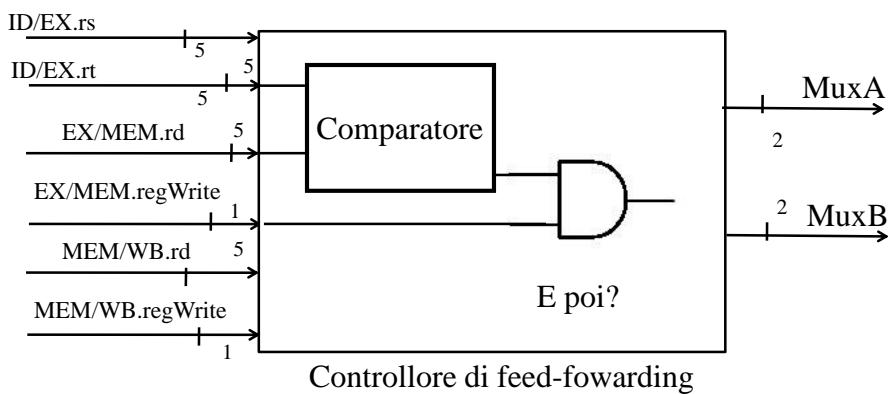
Controllo Mux ingresso alla ALU



Controllo Multiplexer	Registro Sorgente	Funzione
PropagaA = 00	ID/EX	Il primo operando della ALU proviene dal Register File
PropagaA = 01	EX/MEM	Il primo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaA = 10	MEM/WB	Il primo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.
PropagaB = 00	ID/EX	Il secondo operando della ALU proviene dal Register File
PropagaB = 01	EX/MEM	Il secondo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaB = 10	MEM/WB	Il secondo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.



Unità di controllo della propagazione



```

add $t0, $t1, $t2
sub $s0, $r0, $t3
or $s1, $t2, $t0

```

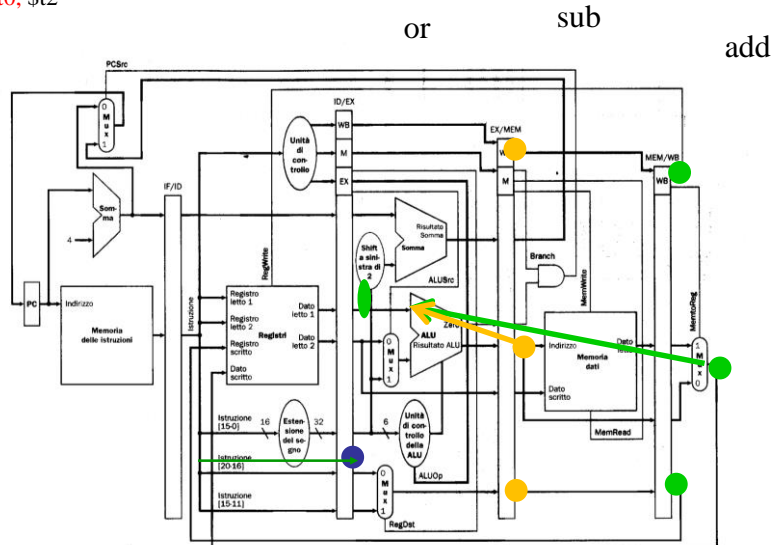


Hazard in MEM e WB



add \$t0, \$t1, \$t2
 sub \$s1, \$t0, \$t3
 or \$s1, \$t0, \$t2

Operando 1 della or sarà = EX/MEM.data o a MEM/WB.data?



A.A. 2020-2021

mi.it



Unità di controllo del forwarding



Deve controllare che la criticità sia effettiva (che l'istruzione precedente scriva il RegisterFile).

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.

E' attiva nella fase di esecuzione (EX) ed implementa le seguenti funzioni:

Dato preso dalla fase MEM:

IF (ID/EX.RegistroRs == EX/MEM.RegistroRd) AND (EX/MEM.RegWrite)

ALUSrcA = <EX/MEM.Data>

IF (ID/EX.RegistroRt == EX/MEM.RegistroRd) AND (EX/MEM.RegWrite)

ALUSrcB = <EX/MEM.Data>

Dato preso dalla fase WB:

IF (ID/EX.RegistroRs == MEM/WB.RegistroRd) AND (MEM/WB.RegWrite)

ALUSrcA = <MEM/WB.Data>

IF (ID/EX.RegistroRt == MEM/WB.RegistroRd) AND (MEM/WB.RegWrite)

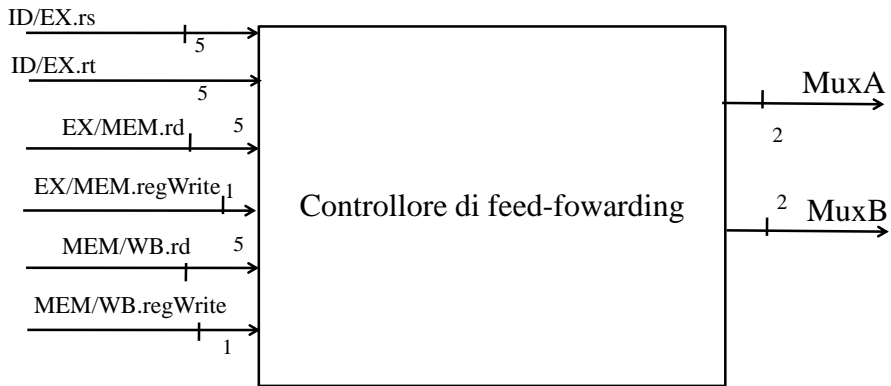
ALUSrcB = <MEM/WB.Data>

Cosa succede se è rilevata una criticità su RS (RT) sia con la fase di MEM che di WB?

4



Unità di controllo della propagazione

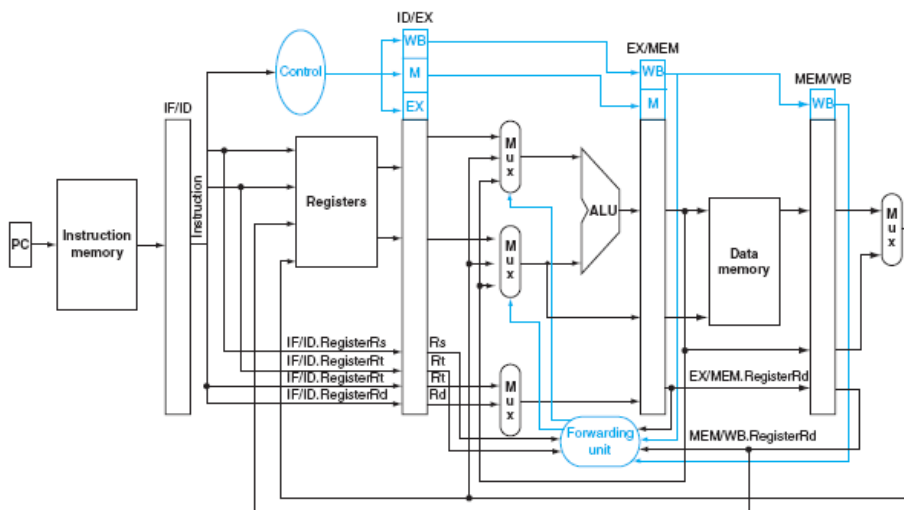


add \$t0, \$t1, \$t2
 sub \$t0, \$t0, \$t3
 or \$s1, \$t2, \$t0

ID/EX.rs = EX/MEM.rd & EX/MEM.write	ID/EX.rs = MEM/WB.rd & MEM/WB.write	MuxA
0	0	00
0	1	10
1	0	01
1	1	01

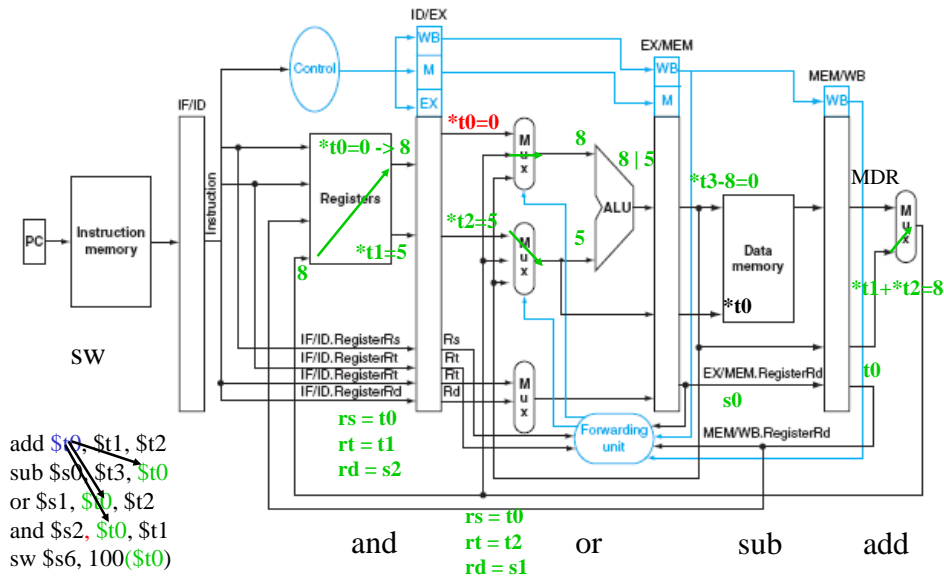


CPU con unità di propagazione

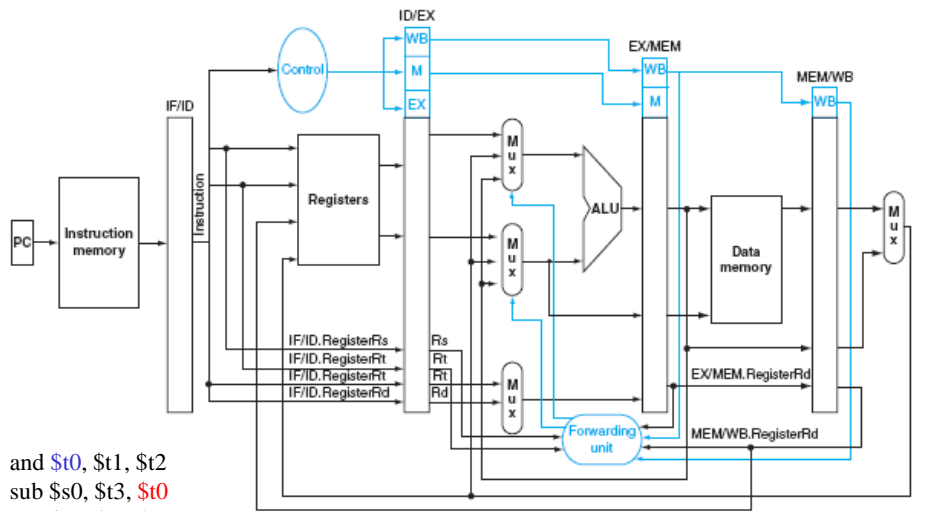




Esecuzione con forwarding



Esercizio



Scrivere il contenuto di tutti i bus quando l'istruzione di add è in fase di EXE, MEM e WB. Ipotizzate $*t1 = 2$, $*t2 = 3$, $*t0 = 0$



Sommario



Criticità in una pipeline

Hazard sui dati

Propagazione