



# Hazard sul controllo

Prof. Alberto Borghese  
Dipartimento di Scienze dell'Informazione  
[borgnese@dsi.unimi.it](mailto:borgnese@dsi.unimi.it)

Università degli Studi di Milano

Riferimento al Patterson: 4.8



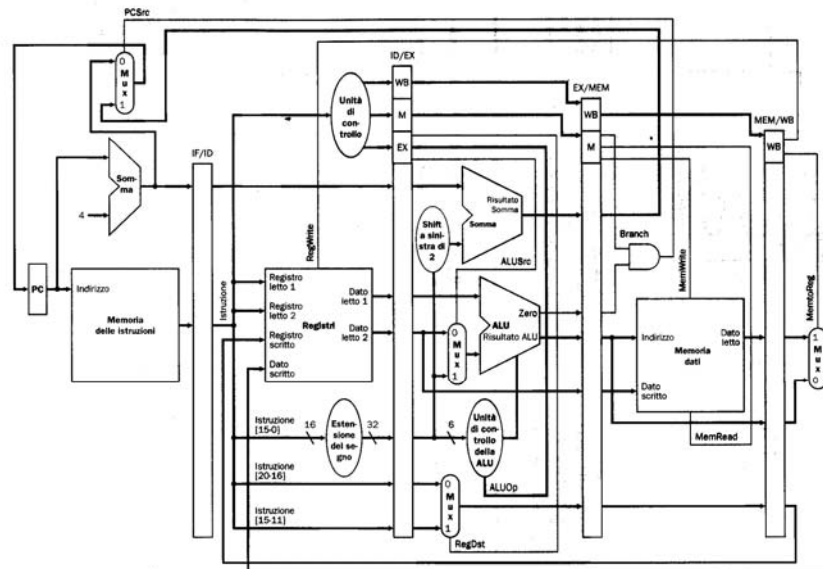
# Sommario

**Modifica delle CPU per la gestione degli hazard sul controllo**

Riorganizzazione del codice (delay slot)



# CPU con pipeline



## Esempio di Hazard sul controllo



sub \$s2, \$s1, \$s3	IF	ID	EX \$1-\$3	MEM	WB s->\$2			
beq \$t2, \$6, 24		IF	ID	EX Zero if (\$s2 == \$s5)	MEM	WB		
or \$t7, \$s6, \$s7			IF	ID	EX	MEM	WB	
add \$t4, \$s8, \$s8				IF	ID	EX	MEM	WB
and \$s5, \$s6, \$s7					IF	ID	EX	MEM
add \$t0, \$t1, \$t2						IF	ID	EX

**In caso di salto:** dovrei avere disponibile all'istante in cui inizia l'esecuzione dell'istruzione or l'indirizzo dell'istruzione add e non eseguire la or, la add e la and.

NB L'indirizzo scritto nel PC corretto deve essere disponibile prima dell'inizio della fase di fetch. Ho 3 istruzioni sbagliate in pipeline.



## Soluzioni alla criticità nel controllo



Modifiche strutturali per l'anticipazione dei salti.  
&  
Riordinamento del codice (delayed branch).



## Modifica della CPU



Obbiettivi:

- Identificare l'hazard durante la fase ID di esecuzione della branch.
- Scartare una sola istruzione.

800:	sub \$s2, \$s1, \$s3	IF	ID	EX \$s1- \$s3	MEM	WB s->\$2			
804:	beq \$t2, \$s6, tag		IF	ID	EX Zero if (\$s2 == \$s5)	MEM	WB		
<del>808:</del>	<del>or \$t7, \$s6, \$s7</del>			<del>IF</del>	<del>ID</del>	<del>EX</del>	<del>MEM</del>	<del>WB</del>	
....									
tag:	add \$t4, \$s8, \$s8				IF	ID	EX	MEM	WB
tag +4:	and \$s5, \$s6, \$s7					IF	ID	EX	MEM
Tag +8	add \$t0, \$t1, \$t2						IF	ID	EX

## Anticipazione del salto

800:	sub \$s2, \$s1, \$s3	IF	ID	EX \$s1- \$s3	MEM	WB s->\$2			
804:	beq \$t2, \$s6, Tag		IF	ID	EX Zero if (\$s2 == \$s5)	MEM	WB		
808:	or \$t7, \$s6, \$s7			IF	ID	EX	MEM	WB	
812:	add \$t4, \$s8, \$s8				IF	ID	EX	MEM	WB
816:	and \$s5, \$s6, \$s7					IF	ID	EX	MEM
....									
Tag	add \$t0, \$t1, \$t2						IF	ID	EX

**In caso di salto:** dovrei avere disponibile all'istante in cui inizia l'esecuzione dell'istruzione or l'indirizzo dell'istruzione add e non eseguire la or, la add e la and.

NB L'indirizzo scritto nel PC corretto deve essere disponibile prima dell'inizio della fase di fetch. Ho 3 istruzioni sbagliate in pipeline.

L'indirizzo è già pronto al termine della fase di EX, posso quindi risparmiare un ciclo di clock.  
Ho 2 istruzioni da eliminare.

A.A. 2009-2010 // 10 http://homes.dsi.unimi.it/~borgnese

## Come identificare l'Hazard nella fase ID

If (rs == rt) & (branch = 1) then  
hazard

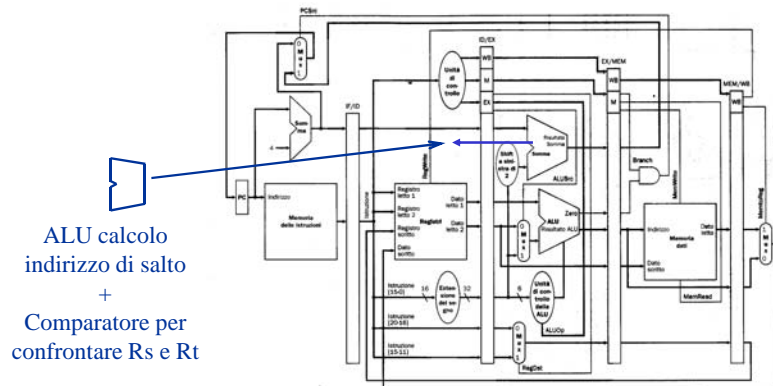
Hazard: Indirizzo successivo sarà  $PC + 4 + \text{Offset} * 4$ , ma ho già caricato (fetch) l'istruzione a  $PC + 4$ ....

Traduco in un circuito logico la condizione che produce hazard

A.A. 2009-2010 8/18 http://homes.dsi.unimi.it/~borgnese



## Come identificare l'Hazard nella fase ID



**Anticipazione della valutazione della branch:** Modifica della CPU nella gestione dei salti: anticipazione del calcolo dell'indirizzo di salto.

- HW aggiuntivo: un comparatore all'uscita del Register File.
- Anticipazione del sommatore .



## Soluzione dell'Hazard sul controllo



Stallo della pipeline.

Dalla fase ID alla WB la beq non fa nulla.

Nella fase di IF è l'istruzione successiva che viene trasferita nell'IR (IF/ID), mentre in caso di salto dovrebbe essere trasferita l'istruzione all'indirizzo di salto.

Quindi:



Occorre annullare l'istruzione nel registro IF/ID, ed inserire una bubble.

Occorre scrivere l'indirizzo di salto nel PC.

	FF	DEC	EX	MEM	WB
t	or	beq	sub	...	...
t+1	and	<i>nop</i>	beq	sub	...

La or deve essere scartata.



## Come scartare un'istruzione

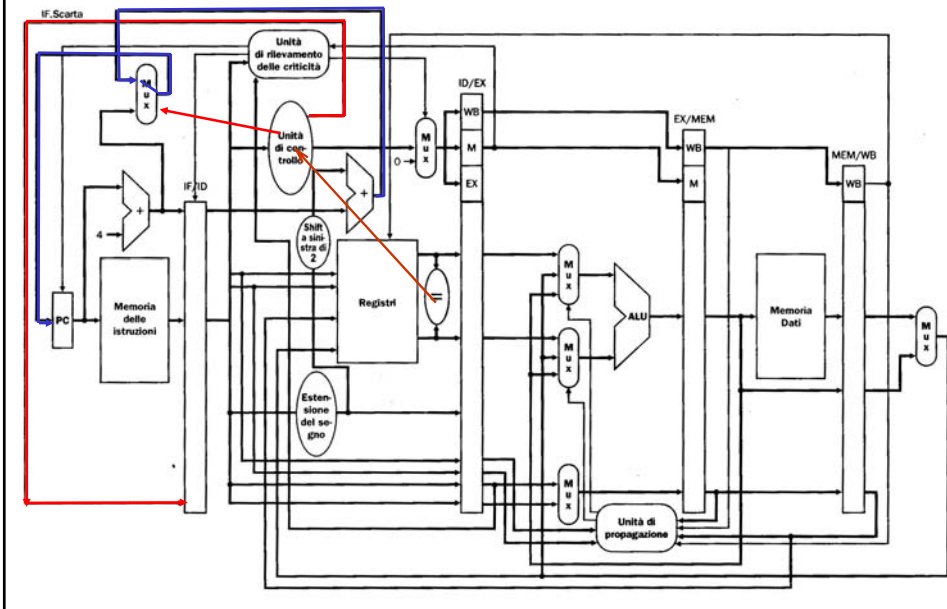


Si carica nel registro IF/ID un'istruzione nulla.

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<code>sll \$s1, \$s2, 7</code>	000000	X	10010	10001	00111 (7)	000000
$\$s1 = \$s2 = \$zero, shmt = 0$						
Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<code>sll \$zero, \$zero, 0</code>	000000	00000	00000	00000	00000 (0)	000000



## CPU con pipeline completa della gestione degli hazard sul controllo





## Sommario



Modifica delle CPU per la gestione degli hazard sul controllo

Riorganizzazione del codice (delay slot)



## Riordinamento del codice



Decisione ritardata (ci si affida al compilatore).

Aggiunta di un "branch delay slot"

- l'istruzione successiva ad un salto condizionato viene sempre eseguita
- contiamo sul compilatore/assemblatore per mettere dopo l'istruzione di salto una istruzione che andrebbe comunque eseguita indipendentemente dal salto (ad esempio posticipo un'istruzione precedente la branch).



## Salto incondizionato



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

	400:	add \$s0, \$s1, \$s2	400:	j 80004
	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.

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



## Salto incondizionato – soluzione II



Prendo l’istruzione dalla destinazione del salto.

	400:	add \$s0, \$s1, \$s2	400:	add \$s0, \$s1, \$s2
	404:	j 80000	404:	j 80004
Label	408:	and \$s1, \$s2, \$s3	408:	or \$t0, \$t1, \$t2
			412:	and \$s2, \$s2, \$s3
	80000:	or \$t0, \$t1, \$t2	80004:	sub \$t3, \$t4, \$t5
	80004:	sub \$t3, \$t4, \$t5		

Riempio tutti gli slot di esecuzione.





## Esempio di delayed branch



Originale	From target	From before
<i>sub \$t5, \$t8, \$s8</i>	<i>sub \$t5, \$t8, \$s8</i>	<i>add \$s4, \$t0, \$t1</i>
<i>add \$s4, \$t0, \$t1</i>	<i>add \$s4, \$t0, \$t1</i>	<i>beq \$s5, \$s6, salto</i>
<i>beq \$s5, \$s6, salto</i>	<i>beq \$s5, \$s6, salto</i>	<i>sub \$t5, \$t8, \$s8</i>
<i>and \$s0, \$s0, \$s1</i>	<i>add \$t5, \$t4, \$t3</i>	<i>and \$s0, \$s0, \$s1</i>
salto:	<i>and \$s0, \$s0, \$s1</i>	salto:
<i>add \$t5, \$t4, \$t3</i>	salto:	<i>add \$t5, \$t4, \$t3</i>
<i>add \$t6, \$t7, \$t7</i>	<i>add \$t6, \$t7, \$t7</i>	<i>add \$t6, \$t7, \$t7</i>

L'istruzione *add \$t5, \$t4, \$t3* o *sub \$t5, \$t8, \$s8* viene comunque eseguita, il salto (se richiesto) avviene all'istante successivo.

Riempio quindi con queste istruzioni lo slot dopo la banch, denominato branch delay slot.

Controllo di non inserire Hazard sui dati

L'istruzione target può essere posizionata prima o dopo a seconda che la beq salti prima o dopo.



## Esempio di riorganizzazione del codice



if (a == b)	if (a == b)
{	{
s2 = s0 + s1;	s2 = s0 + s1;
}	s3 = s4 + s5;
	}
	else
	{
	s3 = s4 + s5;
	}
s3 = s4 + s5;	
salta: s6 = 2;	s6 = 2;



## Esempio di riorganizzazione del codice - II

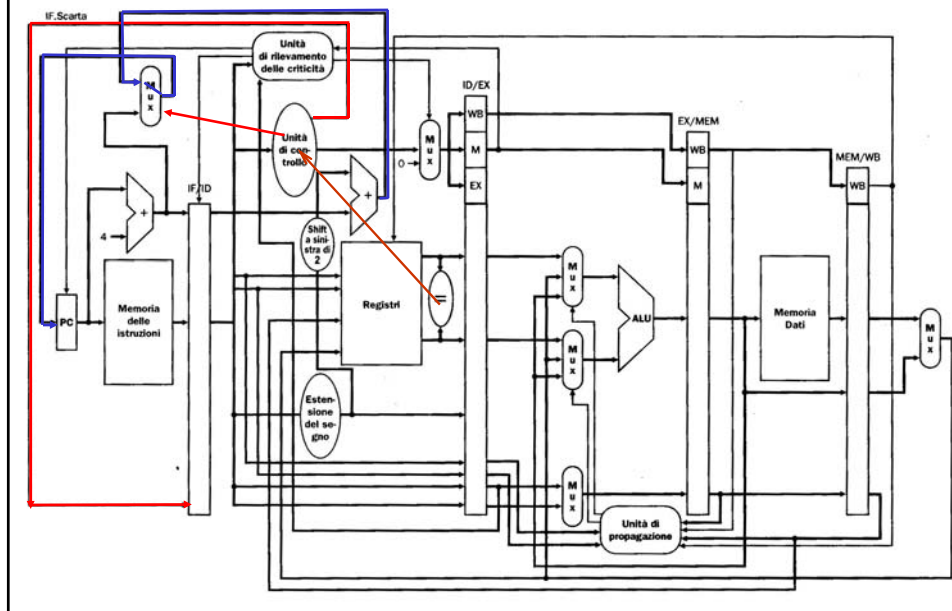


```
if (a == b)
{
    s2 = s0 + s1;
}
else
{
    t2 = t0 + t1;
}
s5 = s4 + s3;
t5 = 2;
```

```
if (a == b)
{
    s2 = s0 + s1;
    s5 = s4 + s3;
}
else
{
    t2 = t0 + t1;
    s5 = s4 + s3;
}
t5 = 2;
```



## CPU con pipeline completa della gestione degli hazard.

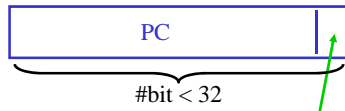




# Branch prediction buffer



Branch prediction ad esempio tramite: branch prediction buffer (4 kbyte nel Pentium 4).



Bit meno significativi del PC

Bit che indica se l'ultima volta il salto era stato eseguito o meno.

### Problema:

Previsione relativa ad una beq con gli stessi bit meno significativi del PC. E' un problema?

```

beq $t0, $t1, SALTA
add $s0, $s1, $s2
sub $s3, $s4, $s5
or  $s6 $s7, $s8
SALTA: and $t2, $t3, $t4

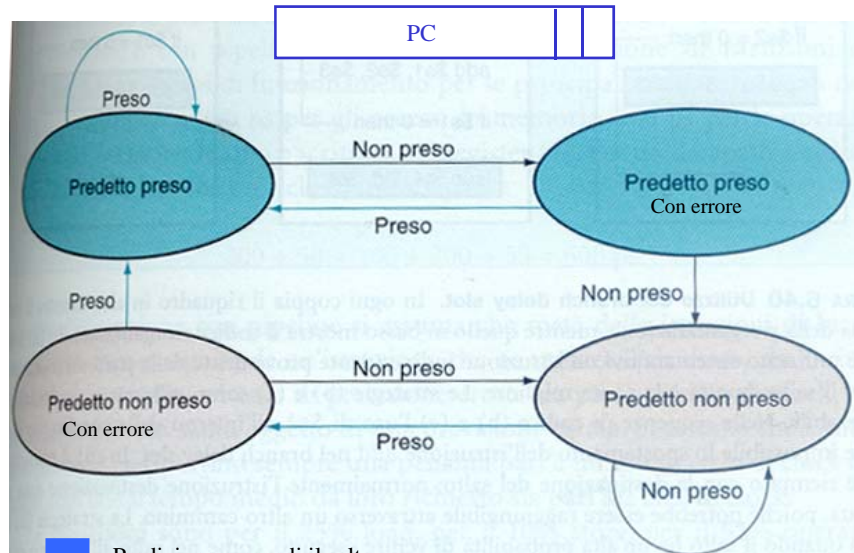
```

In questo caso suppongo di non dovere saltare. Procedo in sequenza. Se la previsione è sbagliata, devo annullare la add e saltare a SALTA.

Algoritmi di ottimizzazione dello scheduling per previsione ottima del salto.



# Branch Prediction buffer a 2 bit





## Evoluzioni della branch prediction



- 1) **Correlating predictors.** Comportamento locale e globale dei salti. Tipicamente 2 predittori a 2 bit. Viene scelto il predittore che correla meglio con la storia del salto.
- 2) **Tournament predictors.** Vengono utilizzati predittori multipli a 1 o 2 bit, e per ogni branch viene selezionato il predittore migliore. Il selettore selezione quale dei due bit di informazione utilizzare, in base alla loro accuratezza di predizione. Solitamente viene utilizzato un predittore che analizza informazioni locali (di contesto), un altro che analizza informazioni globali (di contesto). Informazioni di contesto possono ad esempio essere contenute di registri.... Il codice di selezione per il selettore viene memorizzato nel branch prediction buffer.



## Sommario



Modifica delle CPU per la gestione degli hazard sul controllo

Riorganizzazione del codice (delay slot)