



Hazard sul controllo

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson: 4.7, 4.8



Sommario

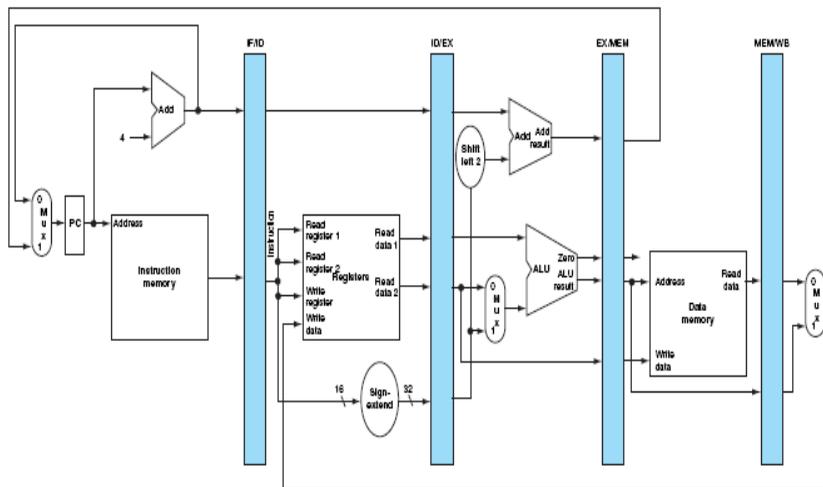
Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

Introduzione agli hazard sul controllo



CPU con pipeline



Esempio di Hazard sul controllo



sub \$s2, \$s1, \$s3	IF	ID	EX \$1-\$3	MEM	WB s->\$2			
beq \$t2, \$s6, 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 eseguir 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.

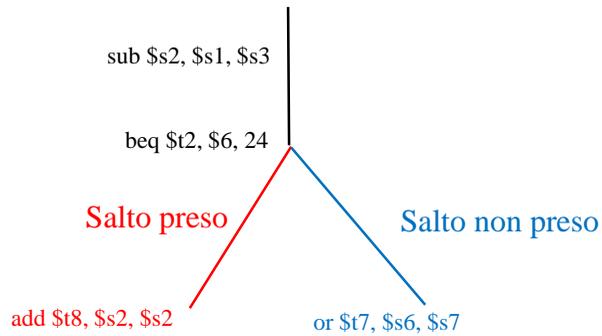
NB Il PC è master/slave per cui occorre che l'indirizzo sia pronto prima dell'inizio della fase di fetch.



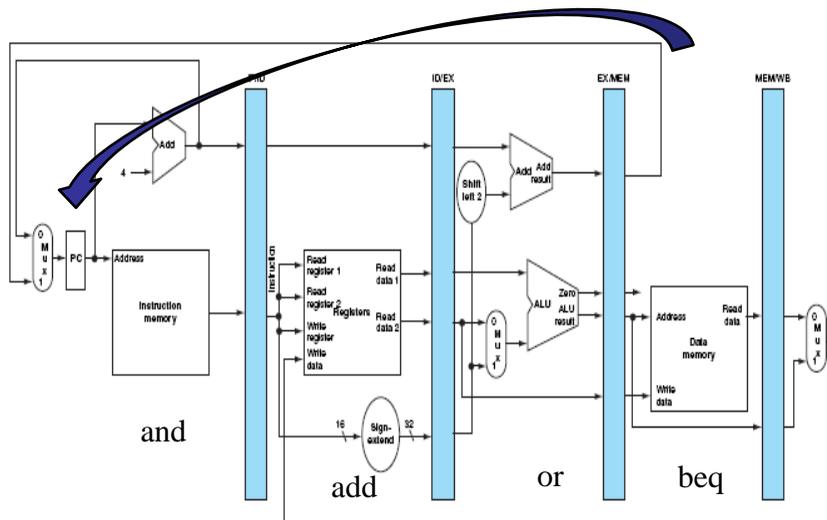
Hazard sul controllo



sub \$s2, \$s1, \$s3
beq \$t2, \$6, 24
or \$t7, \$s6, \$s7
add \$t4, \$s8, \$s8
and \$s5, \$s6, \$s7
add \$t0, \$t1, \$t2
sw \$s3, 24(\$t1)
addi \$t7, \$s6, 10
add \$t8, \$s2, \$s2
and \$s5, \$s6, \$s7
add \$t0, \$t1, \$t2



Criticità sul controllo



Il valore del PC può essere aggiornato nella fase di Memoria
 Le istruzioni in fase di IF, ID, EX potrebbero non dovere essere eseguite.



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			
804:	beq \$t2, \$s6, tag		IF	ID	EX Zero if (\$s2 == \$s5)	MEM	WB		
808:	or \$t7, \$s6, \$s7			IF	ID	EX	MEM	WB	
.....									
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



Come identificare l'Hazard nella fase ID



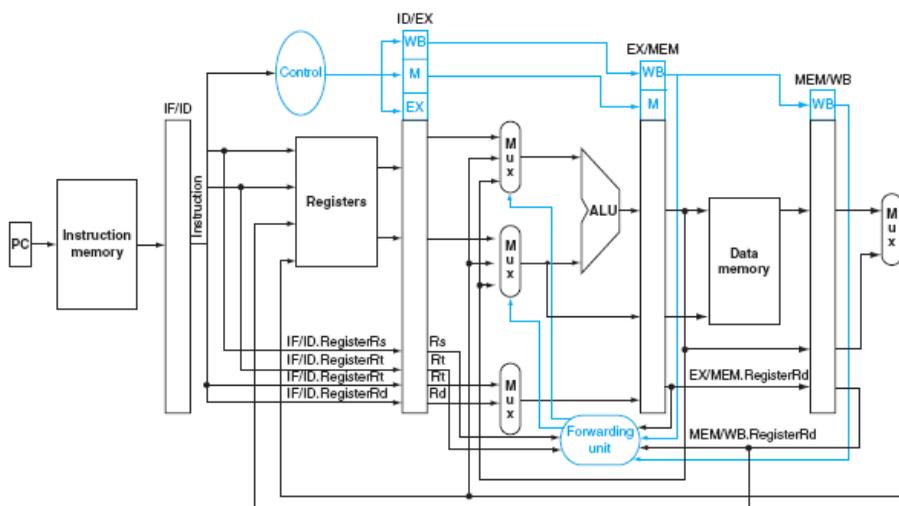
If $(rs == rt) \ \& \ (branch = 1)$ then
hazard

Hazard: Indirizzo successivo sar  $PC + 4 + Offset * 4$, ma ho gi  caricato (fetch) l'istruzione a $PC + 4 \dots$

Traduco in un circuito logico la condizione che produce hazard



CPU con unit  di propagazione





Come scartare un'istruzione



Si carica nel registro IF/ID un'istruzione nulla. Un'istruzione con tutti 0, è un'istruzione di tipo R, ha quindi RegWrite = 1 e modifica un elemento di stato. Ma:

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

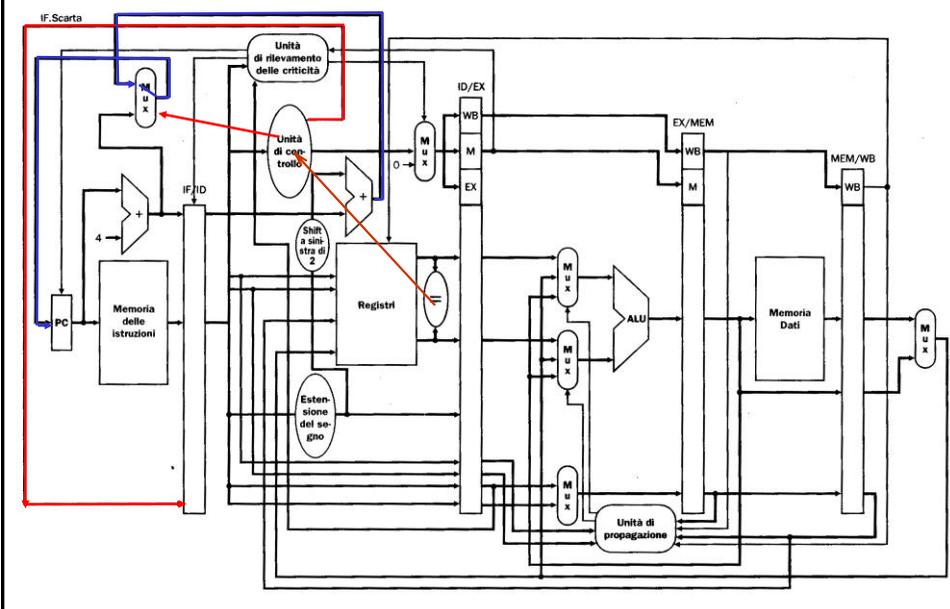
$\$s1 = \$s2 = \$zero$, shmt = 0

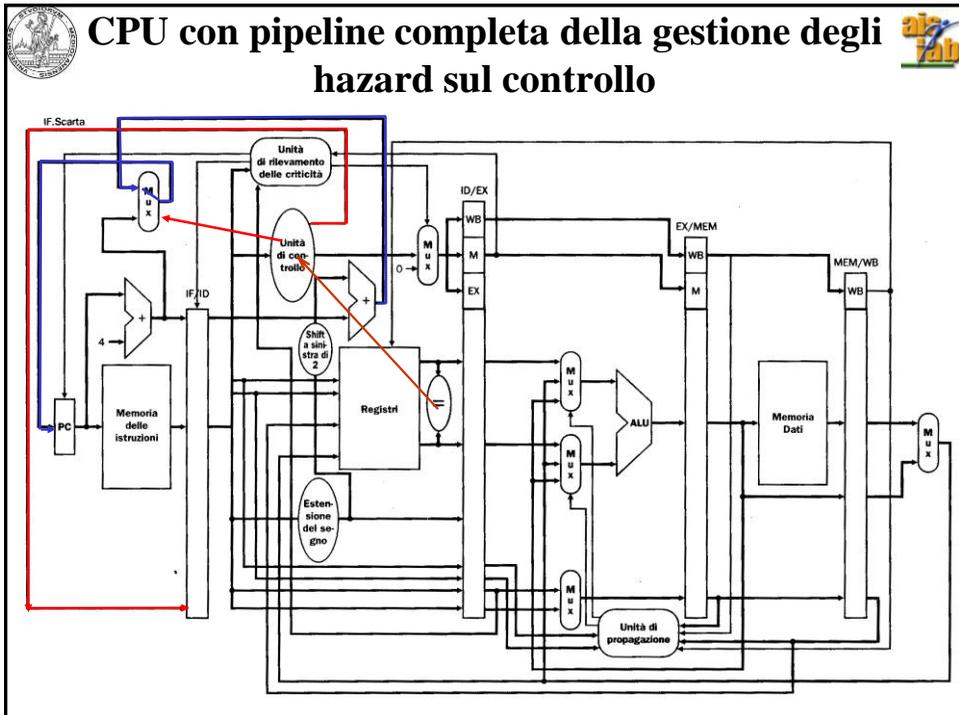
Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<code>sll \$zero, \$zero, 0</code>	000000	00000	00000	00000	00000 (0)	000000

Il registro \$zero non viene utilizzato come registro destinazione



CPU con pipeline completa della gestione degli hazard sul controllo





Gestione della criticità

Soluzione HW:
 Decisione ritardata. Ci si affida all'hardware della CPU per gestire l'eliminazione delle istruzioni (flush).

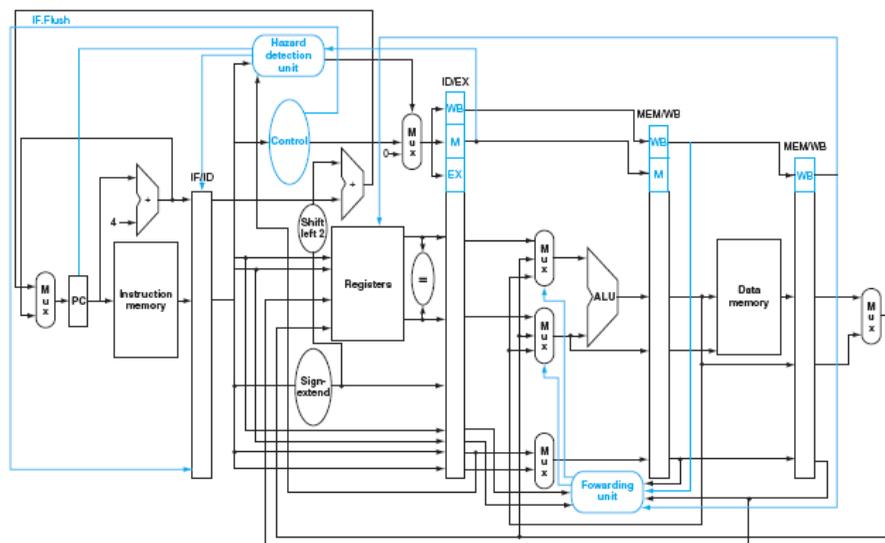
Soluzione SW:
 Aggiunta di un "branch delay slot", un'istruzione successiva a quella di salto che viene sempre eseguita indipendentemente dall'esito della branch.

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).

A.A. 2018-2019 16/22 <http://borghese.di.unimi.it/>



CPU con pipeline completa della gestione degli hazard.



Sommario



Identificazione delle criticità che richiedono stallo

Soluzione della criticità mediante stallo

Introduzione agli hazard sul controllo



Sommario

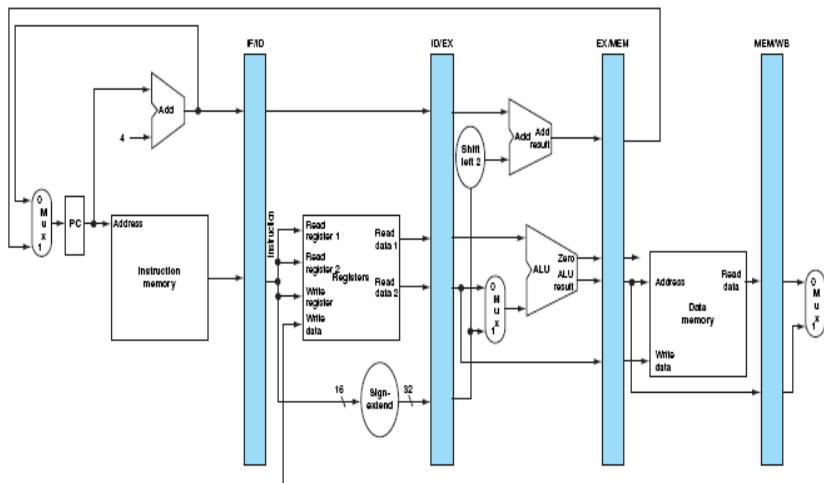


Riorganizzazione del codice (delay slot)

Esercizi



CPU con pipeline





Esempio di Hazard sul controllo



sub \$s2, \$s1, \$s3	IF	ID	EX \$1-\$3	MEM	WB s->\$2			
beq \$t2, \$s6, 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.

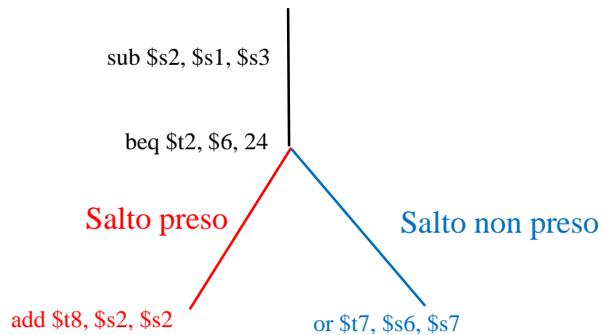
NB Il PC è master/slave per cui occorre che l'indirizzo sia pronto prima dell'inizio della fase di fetch.



Hazard sul controllo

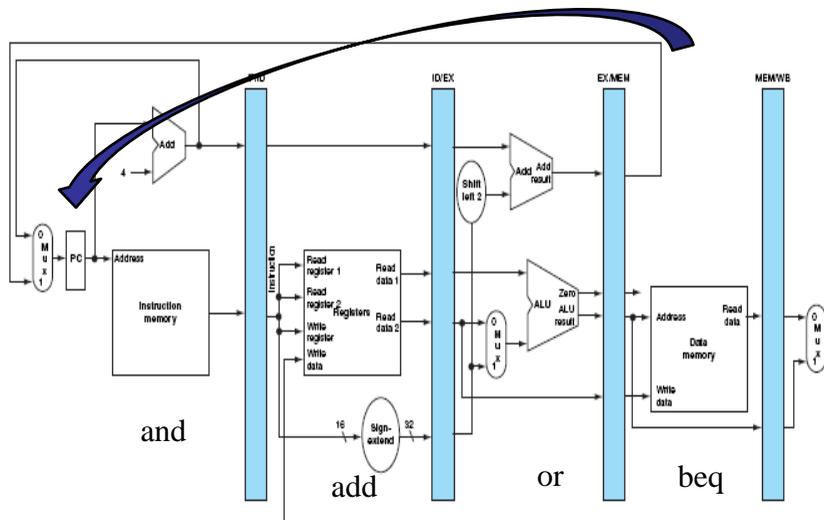


sub \$s2, \$s1, \$s3
beq \$t2, \$s6, 24
or \$t7, \$s6, \$s7
add \$t4, \$s8, \$s8
and \$s5, \$s6, \$s7
add \$t0, \$t1, \$t2
sw \$s3, 24(\$t1)
addi \$t7, \$s6, 10
add \$t8, \$s2, \$s2
and \$s5, \$s6, \$s7
add \$t0, \$t1, \$t2





Criticità sul controllo



Il valore del PC può essere aggiornato nella fase di Memoria

Le istruzioni in fase di IF, ID, EX potrebbero non dovere essere eseguite.

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



Soluzioni alla criticità nel controllo



Modifiche strutturali per l'anticipazione dei salti.

&

Riordinamento del codice (delayed branch).



Esempio di riorganizzazione del codice per le istruzioni di branch



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



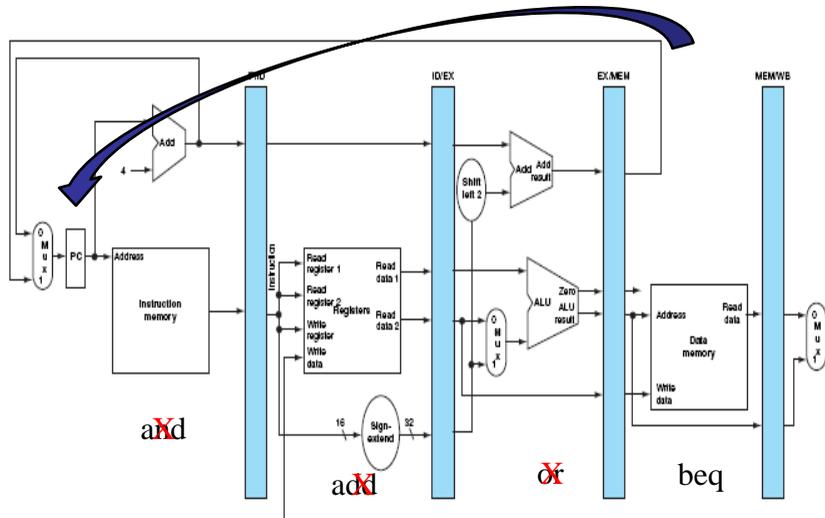
Esempio di riorganizzazione del codice - II



<pre>if (a == b) { s2 = s0 + s1; } else { t2 = t0 + t1; } s5 = s4 + s3; t5 = 2;</pre>	<pre>if (a == b) { s2 = s0 + s1; s5 = s4 + s3; } else { t2 = t0 + t1; s5 = s4 + s3; } t5 = 2;</pre>
--	--



Criticità sul controllo



beq è un salto ritardato, rispetto al codice, il salto viene generato 3 cicli dopo.

Questi 3 cicli di ritardo vengono chiamati **delay slot**.

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



Esempio di delayed branch (1 solo delay slot)



Originale

```
sub $t5, $t8, $s8
add $s4, $t0, $t1
beq $s5, $s6, salto
and $s0, $s0, $s1
```

salto:

```
add $t5, $t4, $t3
add $t6, $t7, $t7
```

From target

```
sub $t5, $t8, $s8
add $s4, $t0, $t1
beq $s5, $s6, salto
add $t5, $t4, $t3
and $s0, $s0, $s1
```

salto:

```
add $t6, $t7, $t7
```

From before

```
add $s4, $t0, $t1
beq $s5, $s6, salto
sub $t5, $t8, $s8
and $s0, $s0, $s1
```

salto:

```
add $t5, $t4, $t3
add $t6, $t7, $t7
```

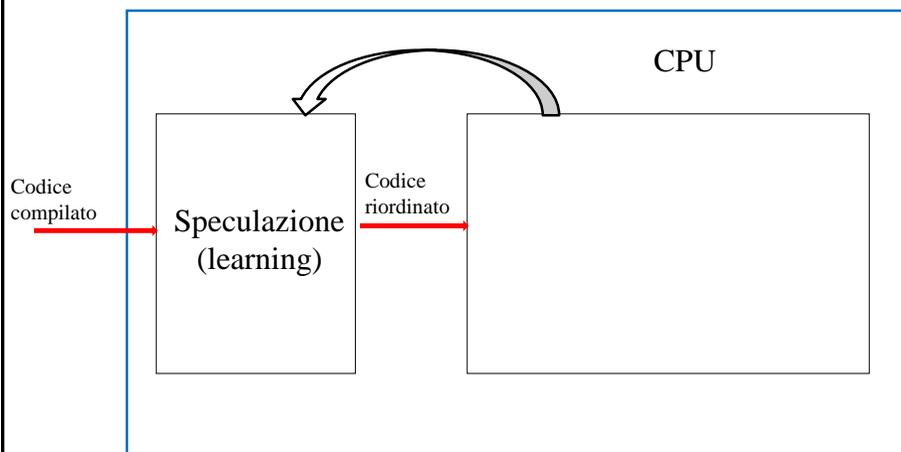
L'istruzione `add $t5, $t4, $t3` o `sub $t5, $t8, $s8` viene comunque eseguita, il salto (se richiesto) avviene all'istante successivo.

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

Controllo di non inserire Hazard sui dati



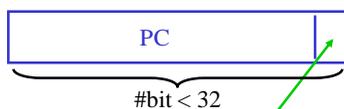
CPU con Predizione



Branch prediction buffer



*Branch prediction ad esempio tramite: branch prediction buffer (4 kbyte nel Pentium 4)
Si trova nel circuito di speculazione.*



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?

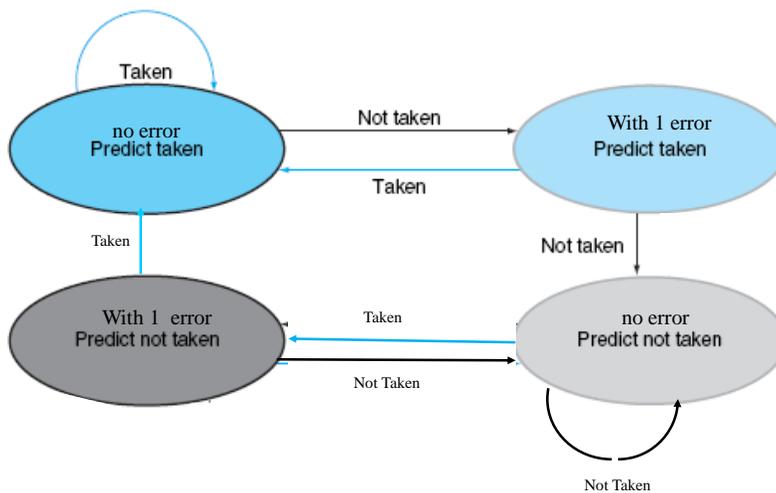
```
START:  beq $t0, $t1, SALTA    for (t0=0; t0<=t1; t0++)
        add $s0, $s1, $s2    {
        sub $s3, $s4, $s5
        addi $t0 $t0, 1
        j  START
SALTA:  and $t2, $t3, $t4    }
```

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 predizione ottima del salto.



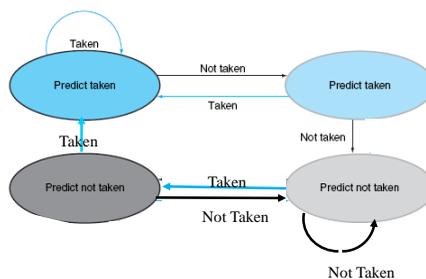
Branch Prediction buffer a 2 bit



Sintesi della FSM della Branch Prediction

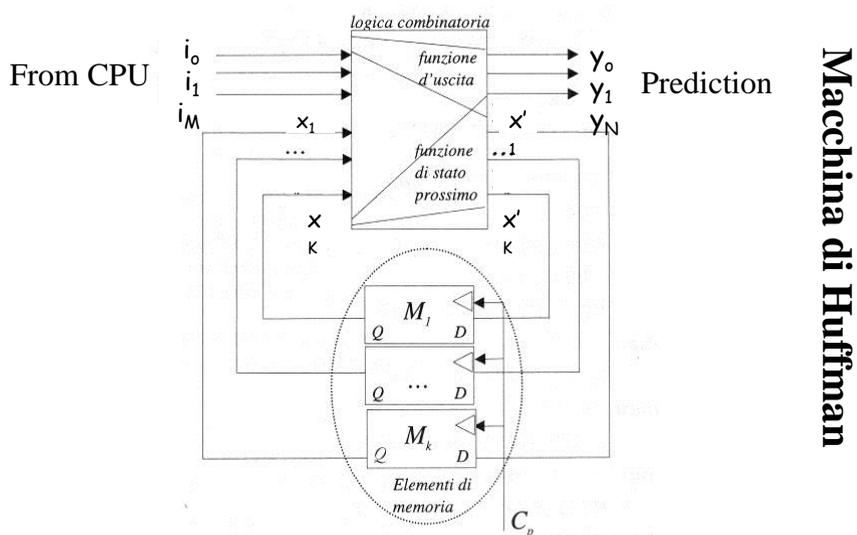


	I=Taken	I=NotTaken	Y
S=Taken 0 errors	Taken 0 errors	Taken 1 error	Predict taken
S = Taken 1 error	Taken 0 errors	Not Taken 1 error	Predict taken
S=Not Taken 0 errors	Not Taken 1 error	Not Taken 0 errors	Predict Not taken
S = Not Taken 1 error	Taken 1 error	Not Taken 0 errors	Predict Not taken





Implementazione del BPB



Gli elementi di memoria sono contenuti nel Branch Prediction Buffer



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 seleziona 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 in registri.... Il codice di selezione per il selettore viene memorizzato nel branch prediction buffer.



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.

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 – soluzione II



Prendo l’istruzione dalla destinazione del salto.

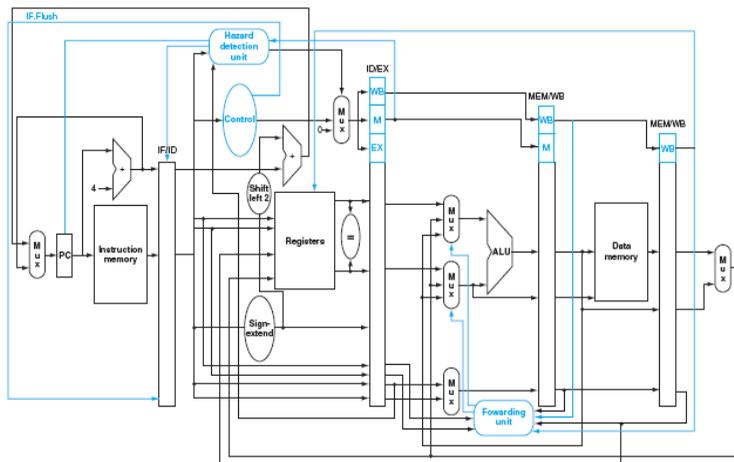
400:	add \$s0, \$s1, \$s2	400:	add \$s0, \$s1, \$s2
404:	j 80000	404:	j 80004
Label 408:	and \$s1, \$s2, \$s3	408:	or \$t0, \$t1, \$t2
		412:	and \$s2, \$s2, \$s3
80000:	or \$t0, \$t1, \$t2	80004:	sub \$t3, \$t4, \$t5
80004:	sub \$t3, \$t4, \$t5		

Riempio tutti gli slot di esecuzione.

L’assemblatore (ri)ordina il codice.



Esercizio



Data la CPU sopra, specificare il contenuto di TUTTE le linee (dati e controllo) quando è in esecuzione il seguente segmento di codice:

0x400 addi \$t3, \$t1, 32

0x404 sub \$t4, \$t1, \$t1

0x408 add \$t1, \$t2, \$t3

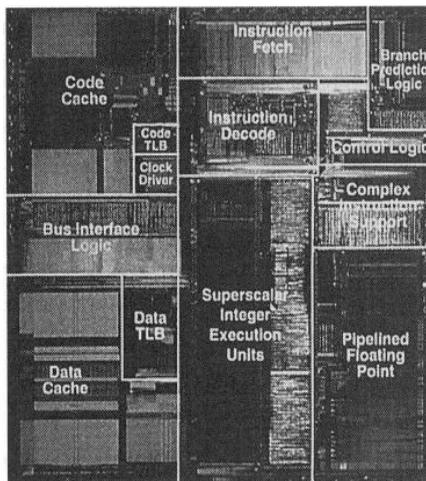
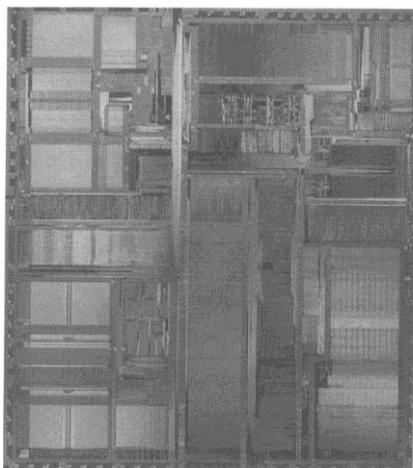
0x 40C beq \$t1, \$s0, 40

0x410 sw \$s2, 64(\$s0)

quando l'istruzione di addi si trova in fase di WB. Specificare sullo schema (con colore o con tratto grosso) quali linee, all'interno dei diversi stadi, trasportino dati e segnali di controllo utili all'esecuzione dell'istruzione, riferendosi alla situazione in cui l'istruzione di addi è in fase di WB.



Il pentium





Sommario



Riorganizzazione del codice (delay slot)

Esercizi