


Hazard sul controllo

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

Università degli Studi di Milano

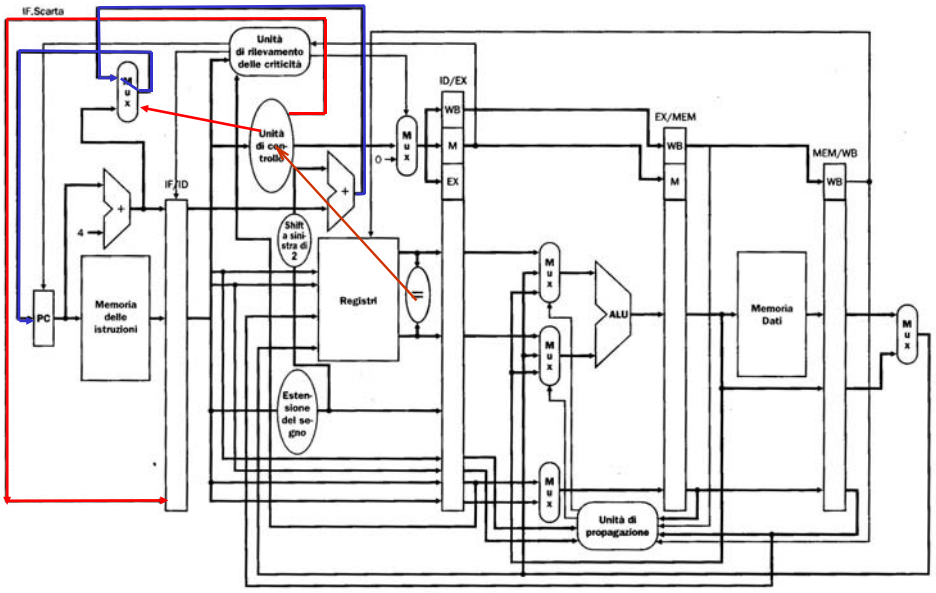
Riferimento al Patterson: 4.7, 4.8


Sommario

Riorganizzazione del codice (delay slot)

Esercizi

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).

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'assemblatere (ri)ordina il codice.

Esempio di riorganizzazione del codice per le istruzioni di branch

```
        if (a = = b)
        {       s2 = s0 + s1;
        }

        s3 = s4 + s5;
salta:  s6 = 2;
```

```
if (a = = b)
{       s2 = s0 + s1;
        s3 = s4 + s5;
}
else
{       s3 = s4 + s5;
}

s6 = 2;
```


Esempio di riorganizzazione del codice - II

```
if (a = = b)
{       s2 = s0 + s1;

}
else
{
        t2 = t0 + t1;

}
s5 = s4 + s3;
t5 = 2;
```

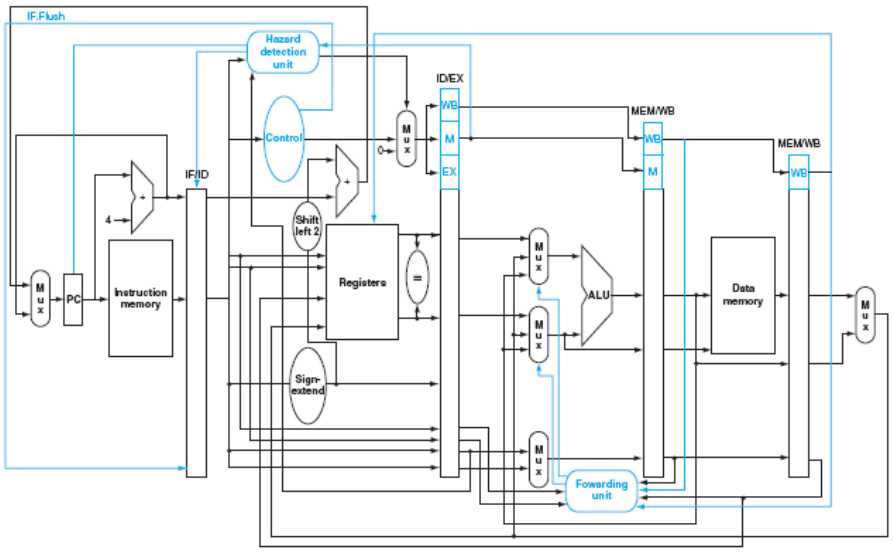
```
if (a = = b)
{       s2 = s0 + s1;
        s5 = s4 + s3;
}
else
{
        t2 = t0 + t1;
        s5 = s4 + s3;
}
t5 = 2;
```

Esempio di delayed branch

Originale	From target	From before
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*	*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*	*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*

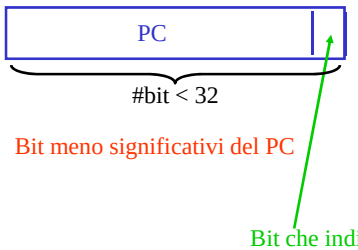
L'istruzione add $t5, $t4, $t3 o sub $t5, $t8, $s8 viene comunque eseguita, il salto (se richiesto) avviene all'istante successivo.
Riempio quindi con queste istruzioni lo slot dopo la banch, denominato branch delay slot.
Controllo di non inserire Hazard sui dati
L'istruzione target può essere posizionata prima o dopo a seconda che la beq salti prima o dopo.

CPU con pipeline completa della gestione degli hazard.



Branch prediction buffer

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



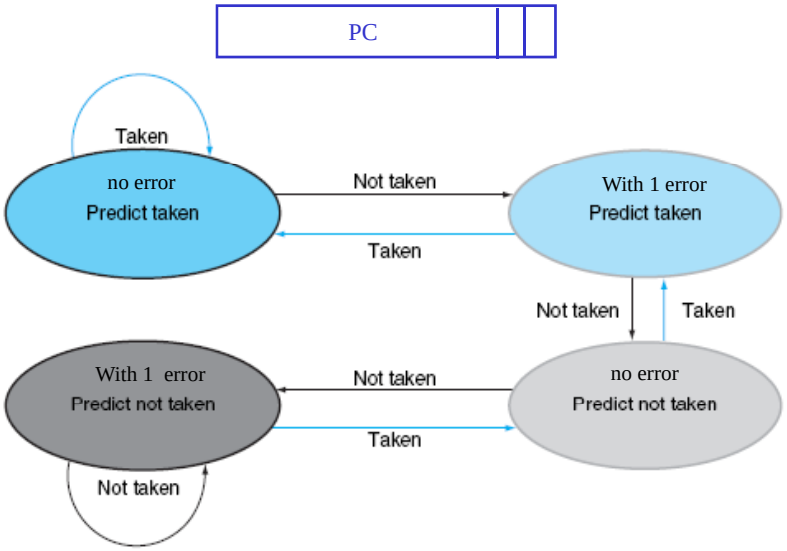
Problema:
Previsione relativa ad una beq con gli stessi bit meno significativi del PC.
E' un problema?

```
        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 predizione 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.

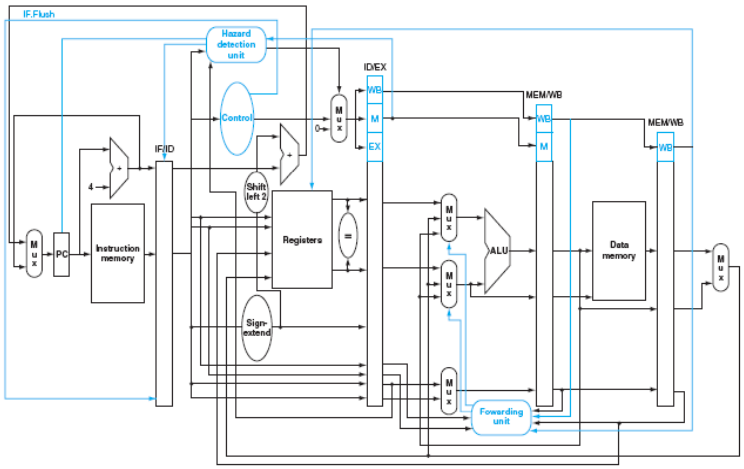
2) **Tournament predictors.** Vengono utilizzati predittori multipli a 1 o 2 bit, e per ogni branch viene selezionato il predittore migliore. Il selettore selezione quale dei due bit di informazione utilizzare, in base alla loro accuratezza di predizione. Solitamente viene utilizzato un predittore che analizza informazioni locali (di contesto), un altro che analizza informazioni globali (di contesto). Informazioni di contesto possono ad esempio essere contenuto di registri…. Il codice di selezione per il selettore viene memorizzato del branch prediction buffer.

Sommario

Riorganizzazione del codice (delay slot)

Esercizi

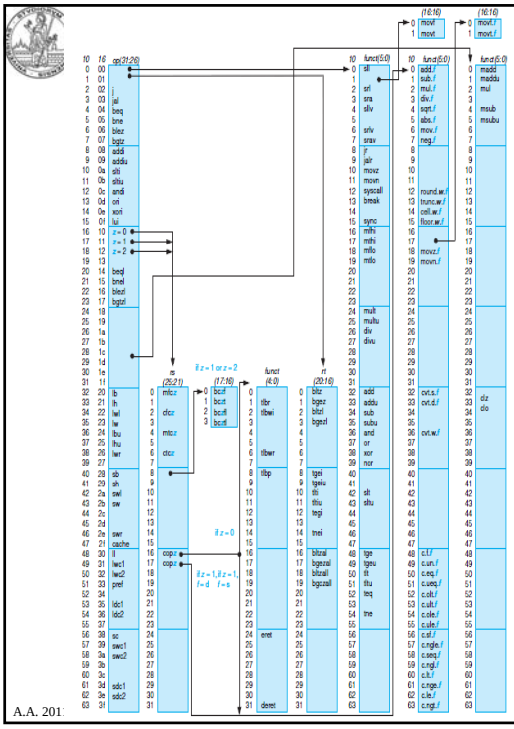
Esercizio



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.

I registri del Register File

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)



I codici operativi

Sommario

Riorganizzazione del codice (delay slot)

Esercizi