


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

Modifica delle CPU per la gestione degli hazard sul controllo

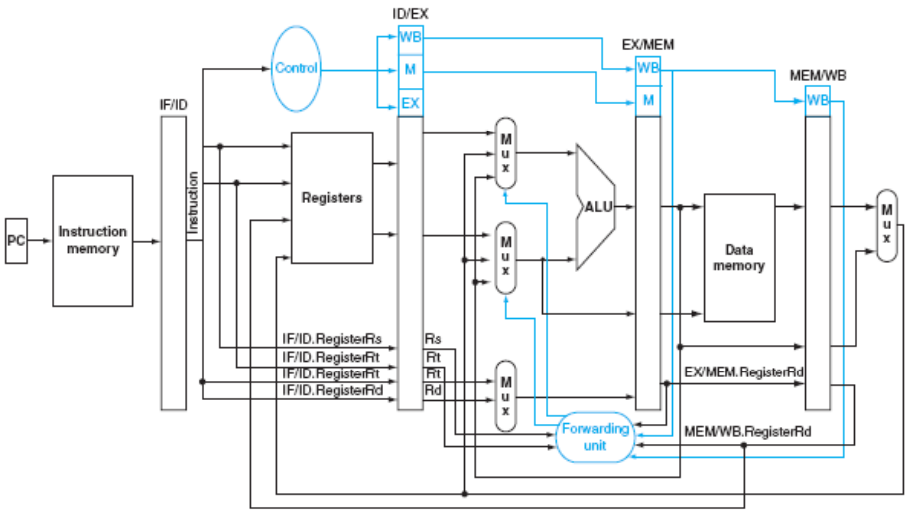
Riorganizzazione del codice (delay slot)

Sommario

Soluzione delle criticità sui dati senza stallo

Soluzione delle criticità sui dati con stallo

CPU con unità di propagazione

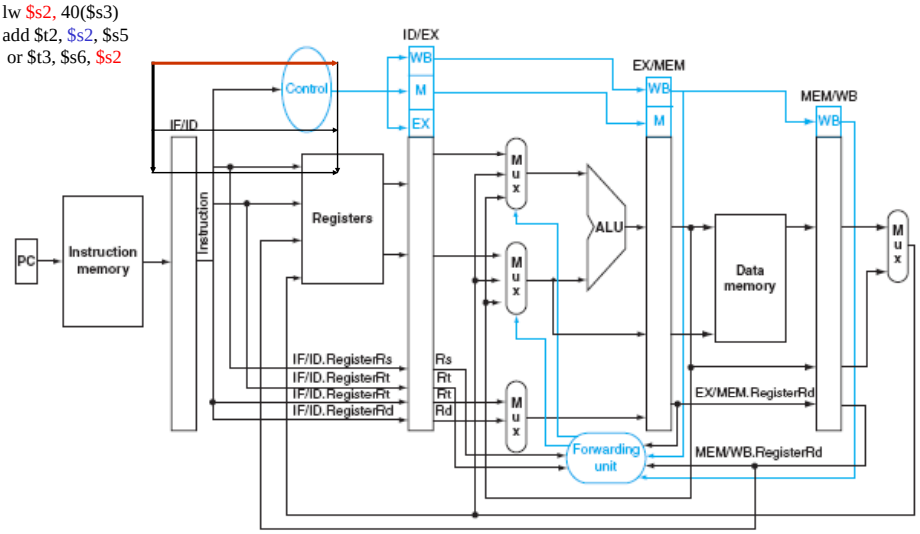


Hazard sui dati: lw

lw $s2, 40($s3)	IF	ID	EX $s3+40	MEM <$s3+40>	WB s->$s2				
add $t2, $s2, $s5		IF	ID	EX $2 and $5	MEM	WB s->$t2			
or $t3, $s6, $s2			IF	ID	EX $s6 or $s2	MEM	WB (s->$t3)		
add $t4, $s2, $s2				IF	ID	EX $s2 +$s2	MEM	WB s->$t4	
sw $t5, 100($s2)					IF	ID	EX $s2+100	MEM $t5 ->Mem	WB

Criticità a 2 passi è risolta?

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



or add lw

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 $s5	MEM	WB s->$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'inzio della fase di WB.

Rilevo la criticità (dato non corretto) 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.

La criticità (dato non corretto) si manifesta 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.

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		Buco (FF)	Buco (DEC)	Buco (EXEC)	Buco (MEM)	Buco (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 messsa in stallo.

Devo bloccare l'esecuzione della and e ripeterla un ciclo dopo, quando è possibile utilizzare il valore corretto del registro $s2

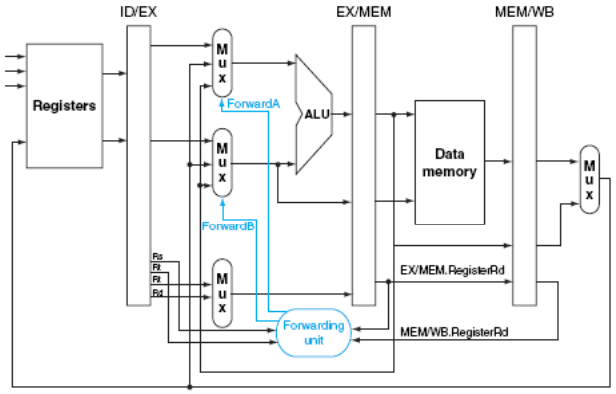
Il cammino in blu invece indica il cammino di propagazione interna dei dati che consente l'esecuzione corretta della add dopo 1 ciclo di stallo.

Rilevamento della criticità della lw

IF [(ID/EX.MemRead)] → Read in fase di EX
AND
{[(IF/ID.RegistroRt) == ID/EX.RegistroRt] OR
[(IF/ID.RegistroRs) == IF/EX.RegistroRt]}
THEN "Metti in stallo la pipeline"

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

Cerco di capire il prima possibile se c'è una criticità per la quale occorra mettere in stallo la pipeline.

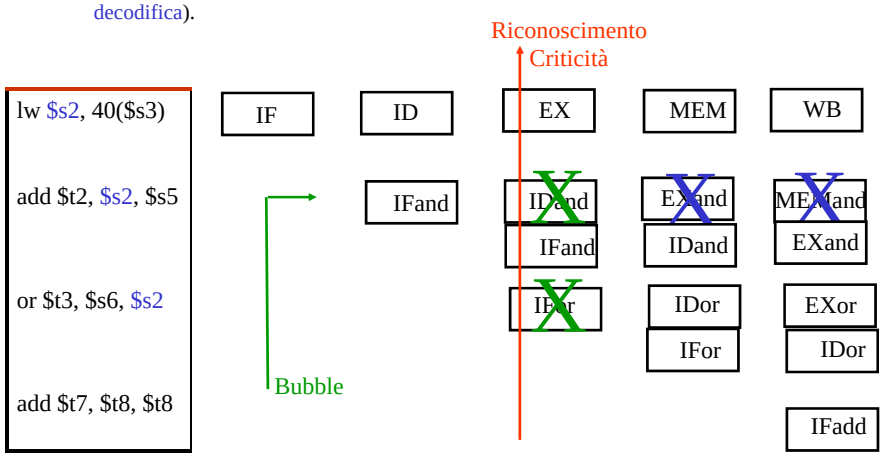


OR AND LW

Stallo della pipeline

Azioni:

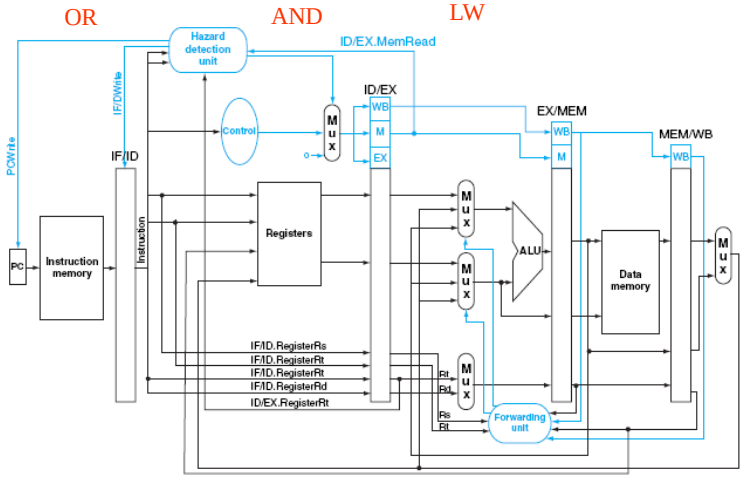
- Annullare i segnali di controllo generati nella fase ID per l'esecuzione dell'istruzione (successiva alla lw).
- Ripetere la lettura e la decodifica delle 2 istruzioni successive (ripetere la fase di fetch e decodifica).



Situazione della pipeline con stallo

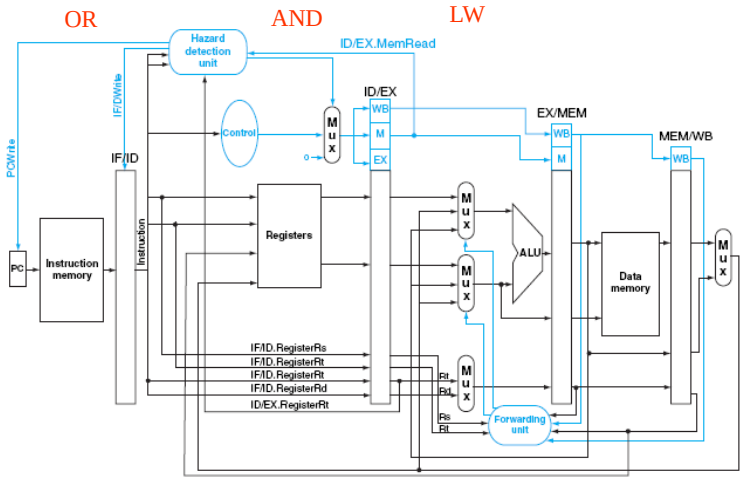
OR AND ?? LW

OR AND LW



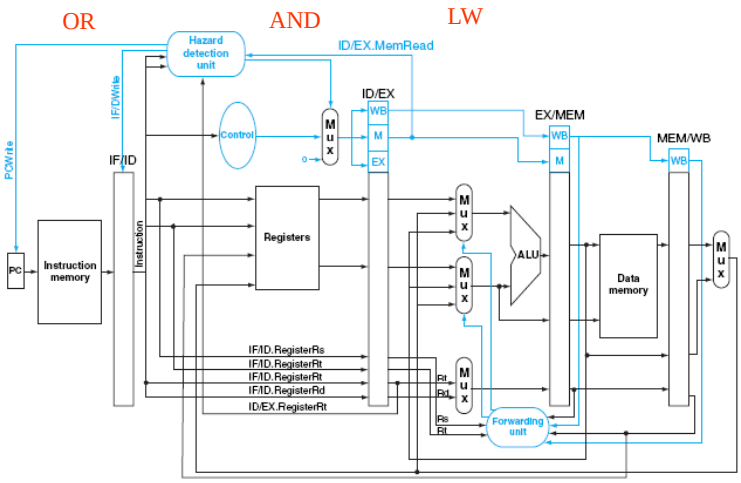
Annullamento dell'istruzione

OR AND LW



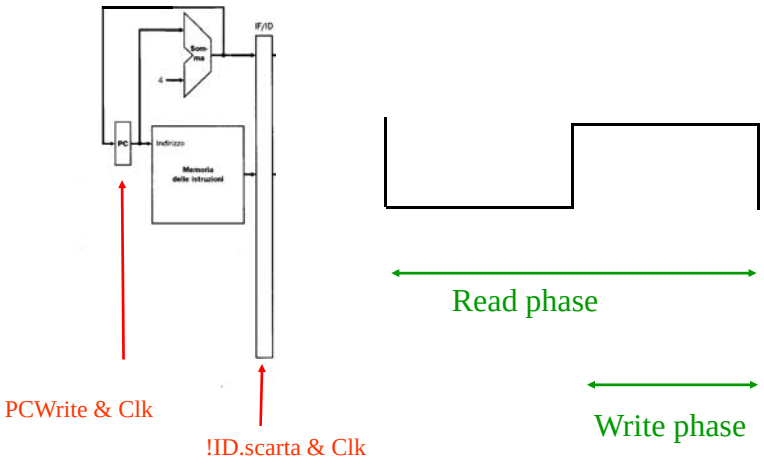
Annullamento dei segnali di controllo associati. Perchè invece non annullo la scrittura dei registri ID/EX, EX/MEM e MEM/WB?

Ripetizione delle fasi ID e IF delle due istruzioni successive



Disabilitazione della scrittura del PC e del registro IF/ID nella fase di Exec della lw.

Disabilitazione della scrittura dei registri



Ipotesi: L'unità di controllo della criticità è in grado di prendere una decisione in tempo utile (prima dell'inizio della fase di Write).

Hazard sui dati della lw

1) Rilevamento della criticità

IF [(ID/EX.MemRead)] AND {[(IF/ID.RegistroRt) == ID/EX.RegistroRt] OR
[(IF/ID.RegistroRs) == IF/EX.RegistroRt]}

2) Correzione del problema -> stallo

2a) faccio eseguire l'istruzione in ID con segnali di controllo a 0: esecuzione fasulla.
2b) inibisco la scrittura dei registri ID e PC.

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.

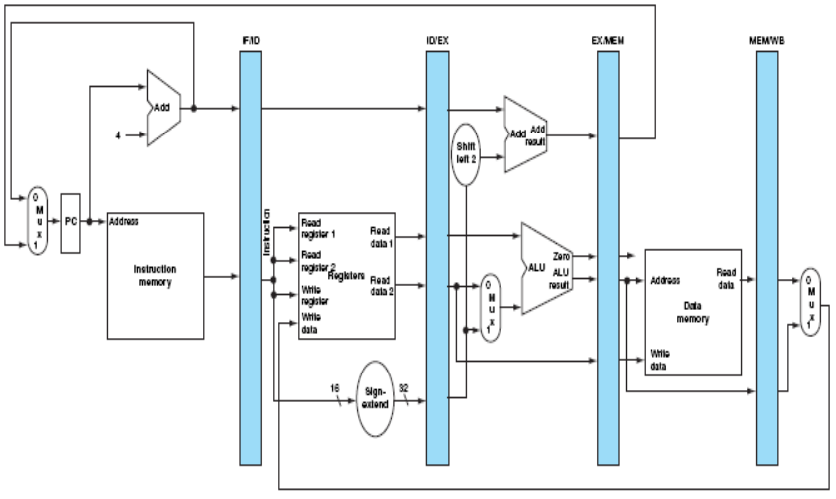
•Accettare uno stallo (non sempre si può evitare).

Sommario

Soluzione delle criticità sui dati senza stallo

Soluzione delle criticità sui dati con stallo

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.

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

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		
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

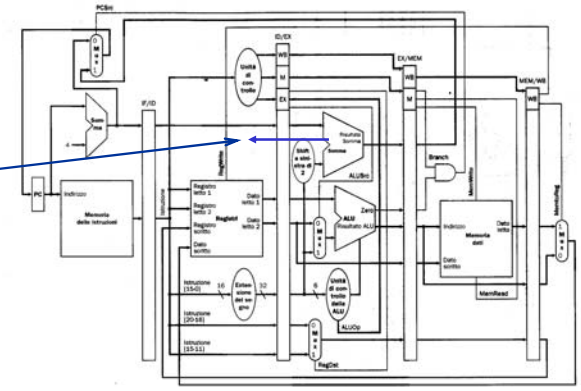
Come identificare l'Hazard nella fase ID

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

Hazard: Indirizzo successivo sarà PC + 4 + Offset * 4, ma ho già caricato (fetch) l'istruzione a PC +4…..

Traduco in un circuito logico la condizione che produce hazard

Come identificare l'Hazard nella fase ID



ALU calcolo indirizzo di salto
+
Comparatore per confrontare Rs e Rt

Anticipazione della valutazione della branch: Modifica della CPU nella gestione dei salti: anticipazione del calcolo dell'indirizzo di salto.
•HW addizionale: 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	*nop*	beq	sub	…

La or deve essere scartata.

Come scartare un'istruzione

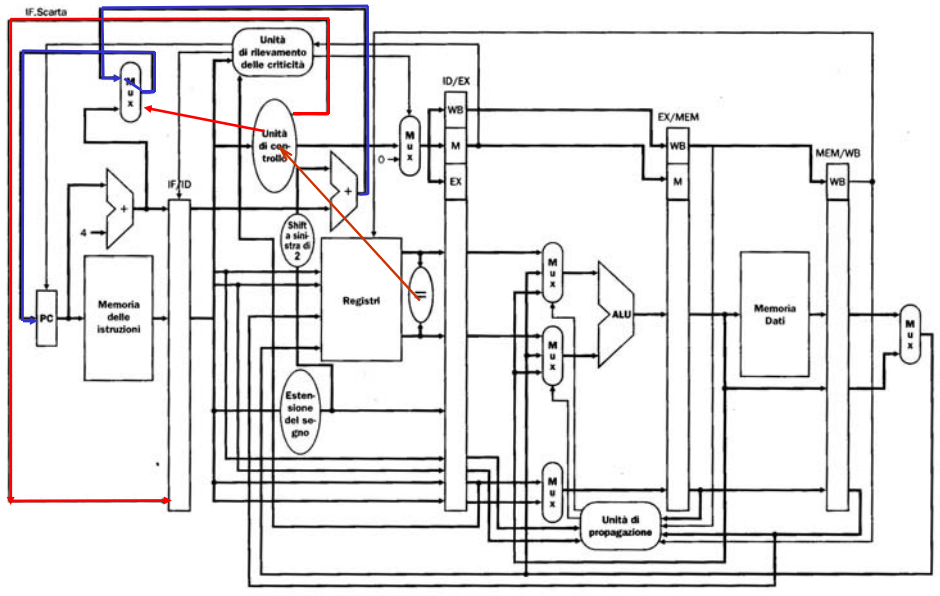
Si carica nel registro IF/ID un'istruzione nulla.

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sll $s1, $s2, 7`	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
`sll $zero, $zero, 0`	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)

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.

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 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

```
        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

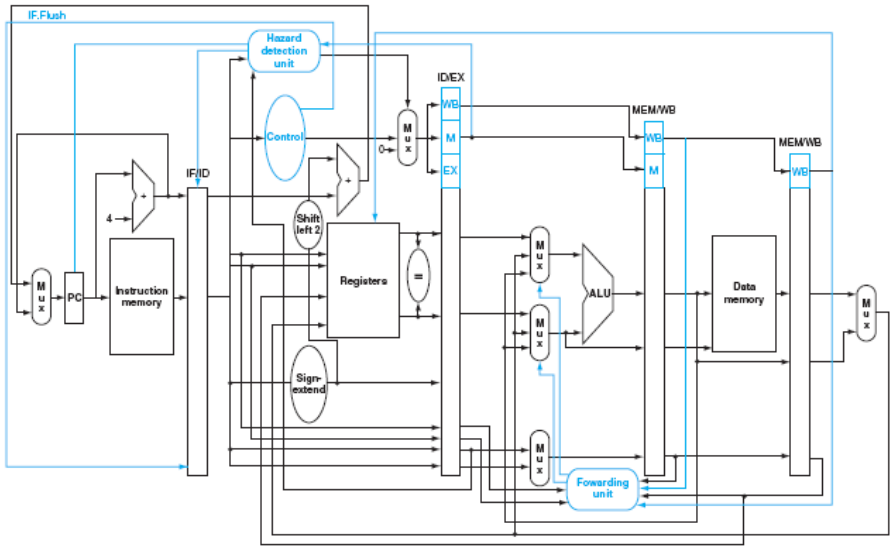
```
        sub $t5, $t8, $s8
        add $s4, $t0, $t1
        beq $s5, $s6, salto
        add $t5, $t4, $t3
        and $s0, $s0, $s1
salto:
        add $t6, $t7, $t7
```

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
```

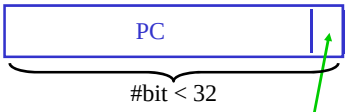
L'istruzione add $t5, $t4, $t3 o sub $t5, $t8, $s8 viene comunque eseguita, il salto (se richiesto) avviene all'istante successivo.
Riempio quindi con queste istruzioni lo slot dopo la banch, denominato branch delay slot.
Controllo di non inserire Hazard sui dati
L'istruzione target può essere posizionata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



Bit meno significativi del PC

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

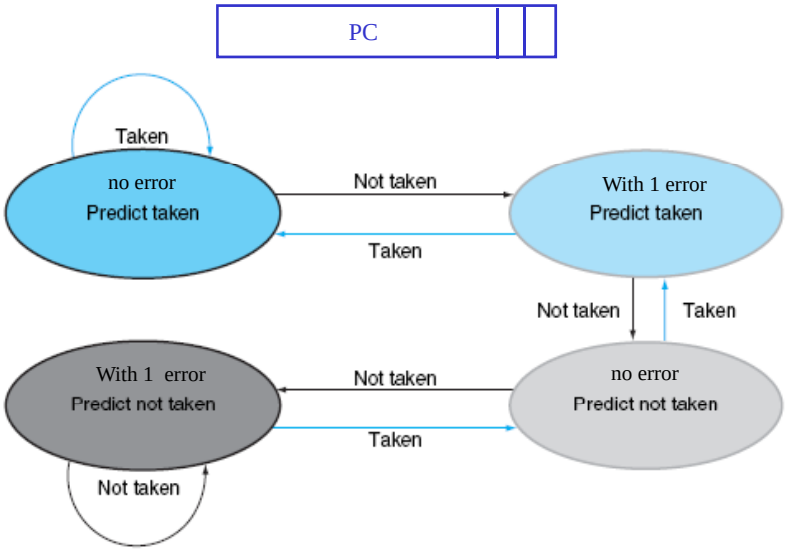
Problema:
Previsione relativa ad una beq con gli stessi bit meno significativi del PC.
E' un problema?

```
        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 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

Modifica delle CPU per la gestione degli hazard sul controllo

Riorganizzazione del codice (delay slot)