


Forwarding, hazard sul controllo

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

Università degli Studi di Milano

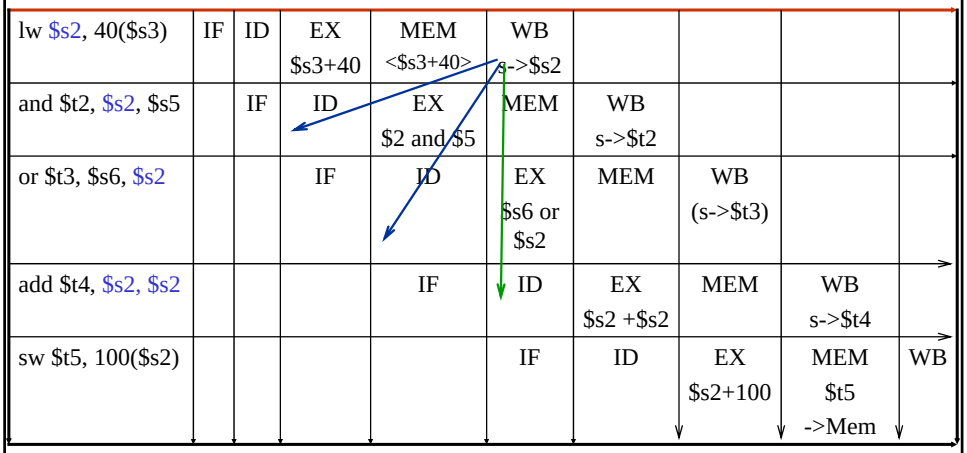

Sommario

Modifiche alla CPU per la gestione di criticità sui dati, istruzioni di lw.

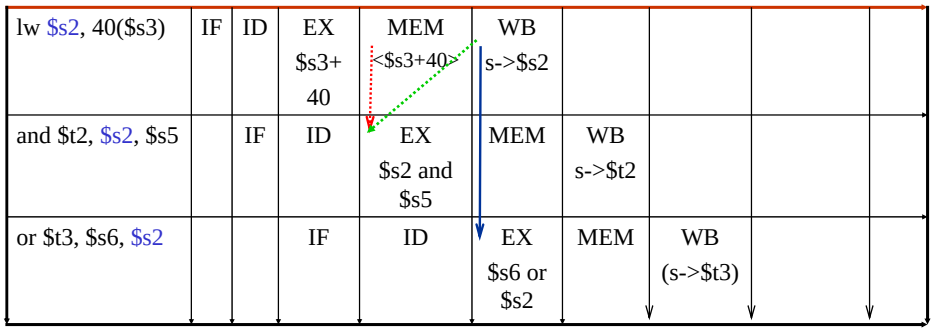
Hazard sul controllo

Trend delle pipeline

Hazard sui dati: lw



Hazard sui dati: lw, rilevamento della criticità



Il dato corretto per $s2 è pronto nella lw solamente alla fine della fase MEM, ed è perciò utilizzabile solamente a partire dall'inizio della fase di WB.

Rilevo la criticità (dato non corretto) su and quando and inizia la fase di EX. In questo caso il dato corretto non è ancora stato prodotto dalla lw.

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.

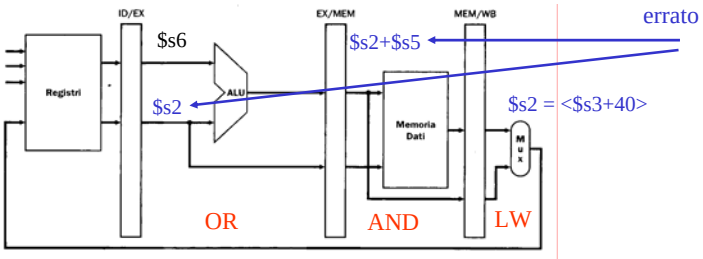
Esempio di Hazard sui dati: lw / add

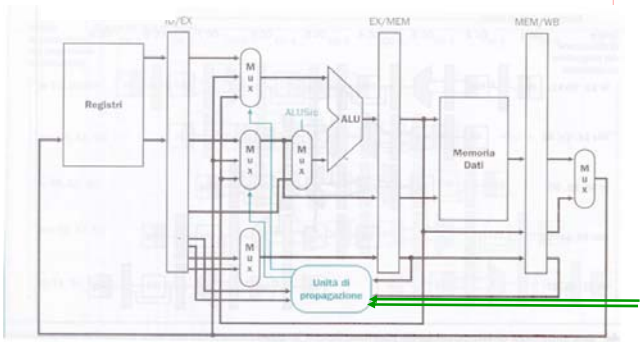
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
…..								
lw $t0, 8($s0)	FF (Mem, ALU)	DECOD (RF)	EXEC (ALU)	MEM (MEM)	WB (RF)			
add $t1, $t0, $s0		Buco (FF)	Buco (DEC)	Buco (EXEC)	Buco (MEM)	Buco (WB)		
add $t1, $t0, $s0			Buco	Buco	Buco	Buco	Buco	
add $t1, $t0, $s0				FF	DEC	EXEC	MEM	

I buchi (o bubble) inducano degli istanti di clock in cui non può essere eseguita l'istruzione successiva ➔ La pipeline va in stallo.

Hazard nei dati: lw, forwarding

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

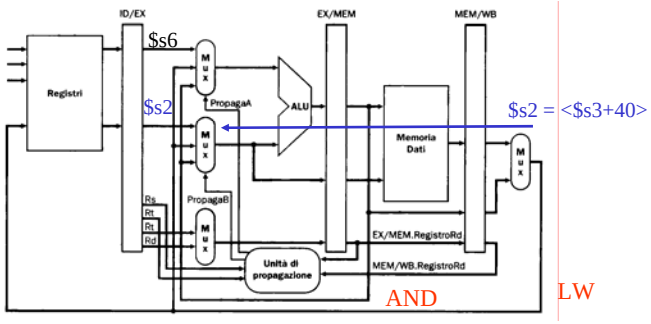




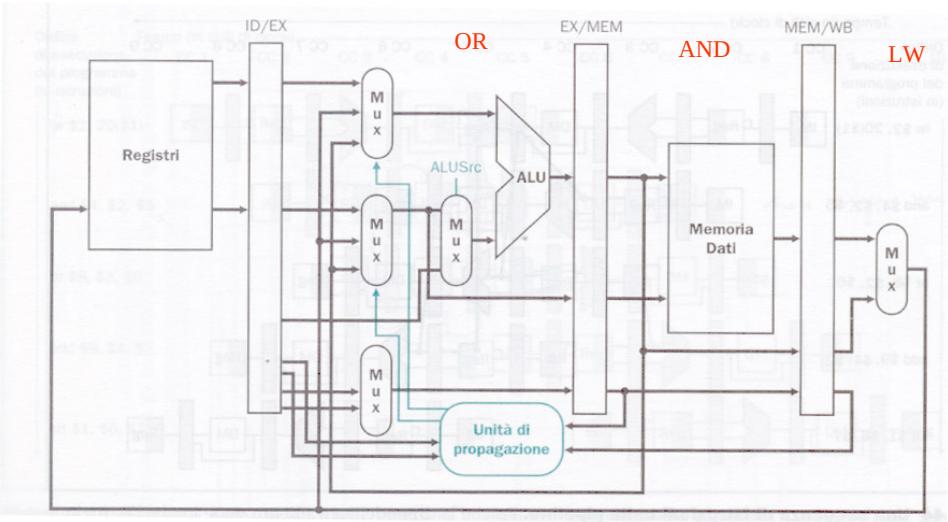
Hazard nei dati: lw, unità di propagazione

Dato preso dalla fase WB:
IF (ID/EX.RegistroRs == MEM/WB.RegistroRd) AND (MEM/WB.RegWrite)
ID/EX.RegistroRs = MEM/WB.RegistroRd)
IF (ID/EX.RegistroRt == MEM/WB.RegistroRd) AND (MEM/WB.RegWrite)
ID/EX.RegistroRt = MEM/WB.RegistroRd)

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

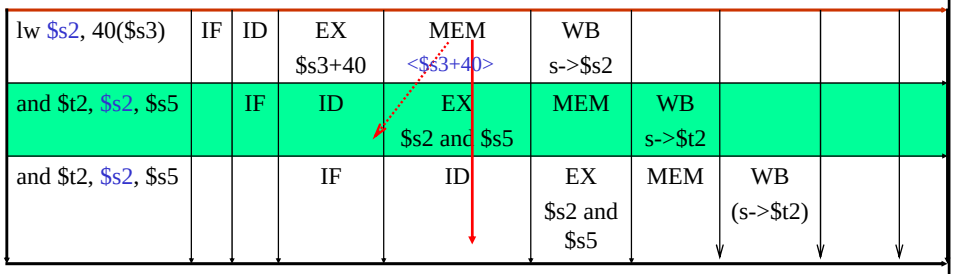


Il forwarding non è sufficiente



Nessuna modifica ma risolve solamente uno dei due problemi della lw.

Hazard sui dati: lw, stallo

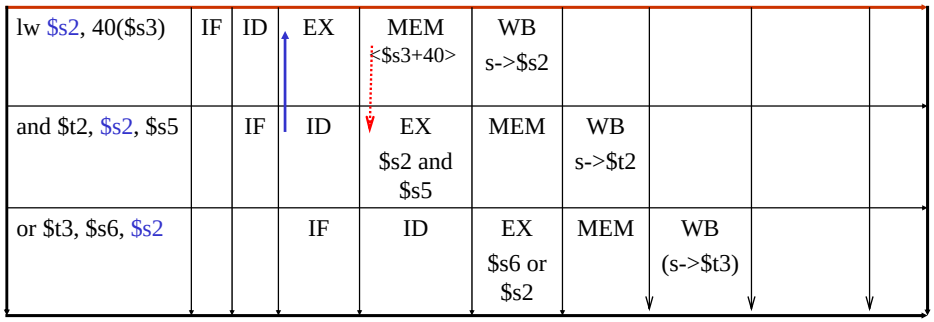


Il dato corretto per $s2 è pronto nella lw solamente alla fine della fase MEM, ed è perciò utilizzabile solamente a partire dall'inizio della fase di EX.

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

Stallo della pipeline

Rilevamento della criticità sulla lw



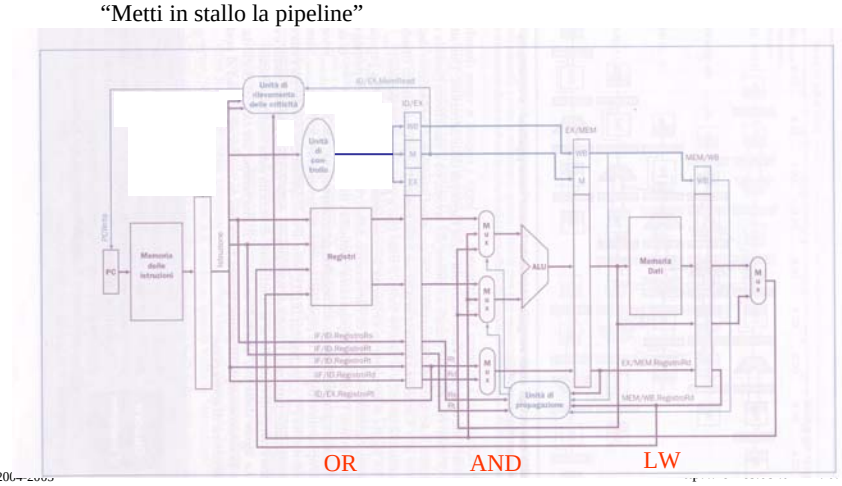
Il dato corretto per $s2 è pronto nella lw solamente alla fine della fase MEM, ed è perciò utilizzabile solamente a partire dall'inizio della fase di WB.

Rilevo questa criticità il prima possibile in modo da mettere in stallo prima possibile la pipeline: nello stadio di decodifica dell'istruzione AND.

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"

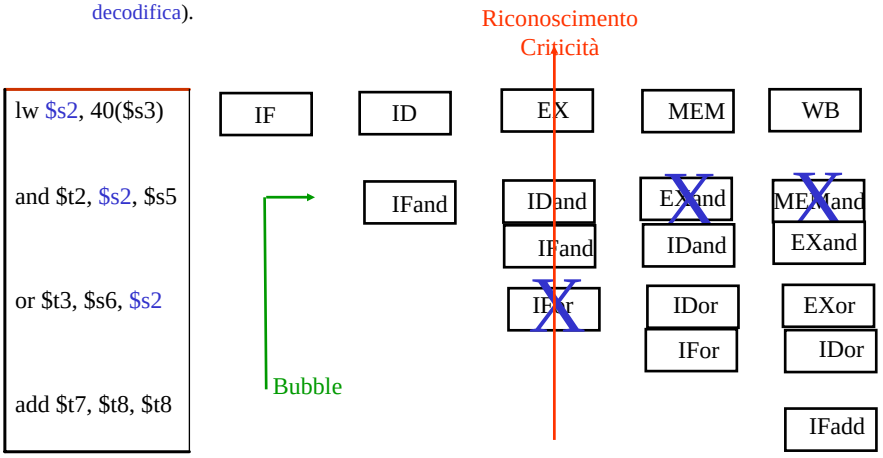
EX – lw $s2, 40($s3)
DEC – and $t2, $s2, $s5



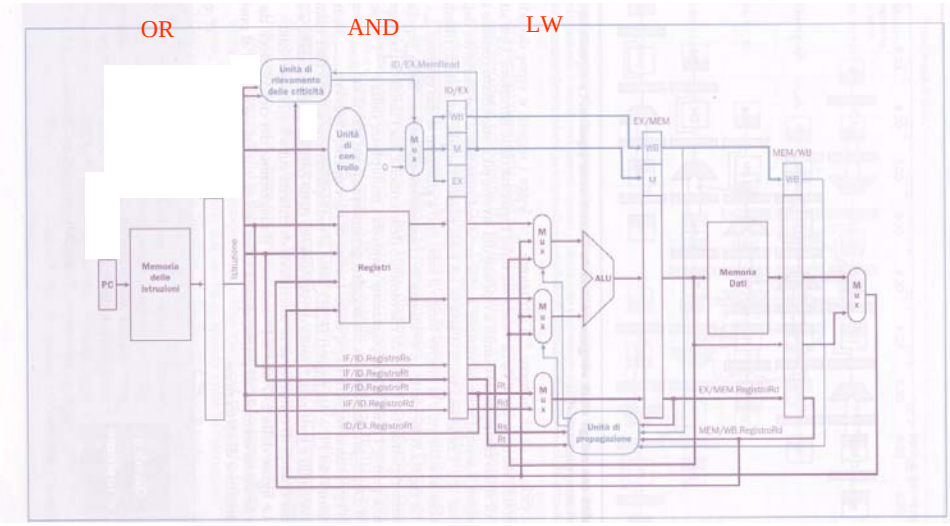
Stallo della pipeline

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).

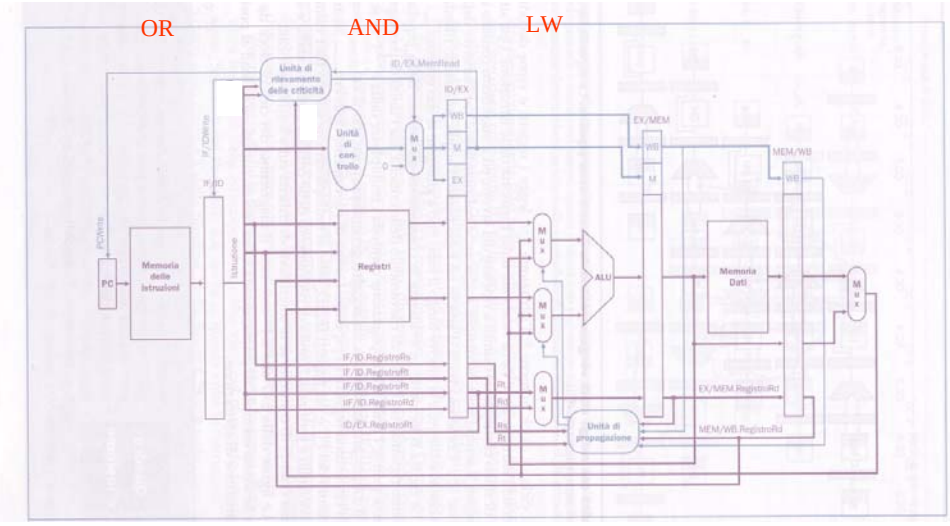


Annullamento dell'istruzione in fase ID (and)



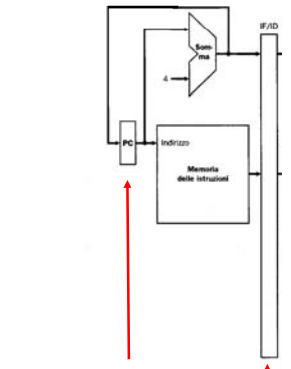
Annullamento dei segnali di controllo associati. Perchè invece non annullo la scrittura dei registri ID/EX, EX/MEM e MEM/WB?

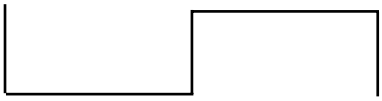
Ripetizione delle fasi ID e IF delle due istruzioni successive



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

Disabilitazione della scrittura dei registri





Read phase

Write phase

PCWrite & Clk

!ID.scarta & Clk

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

Rilevamento della criticità sulla lw

lw $s2, 40($s3)	IF	ID	EX	MEM	WB s->$s2				
and $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 questa criticità il prima possibile in modo da mettere in stallo prima possibile la pipeline: nello stadio di decodifica dell'istruzione AND.

Potrei rilevare la criticità anche nello stadio EX dell'istruzione AND. Quale svantaggio avrei?

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.

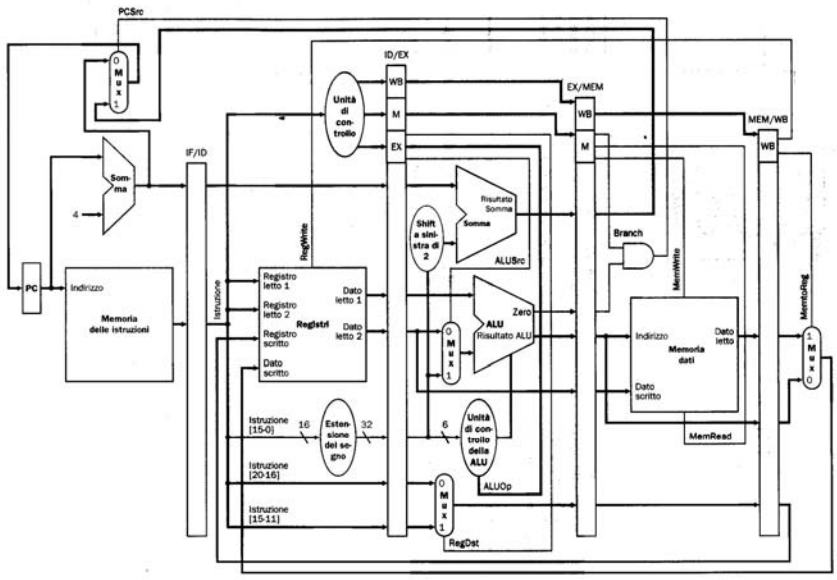
Sommario

Modifiche alla CPU per la gestione di criticità sui dati, istruzioni di lw.

Hazard sul controllo (branch hazard)

Trend delle pipeline

CPU con pipeline



Soluzioni alla criticità nel controllo

Modifiche strutturali per l'anticipazione dei salti.

Riordinamento del codice (delayed branch).

Hazard sul controllo

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.

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

Modifiche strutturali

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
tag:	and $s5, $s6, $s7					IF	ID	EX	MEM
tag +4	add $t0, $t1, $t2						IF	ID	EX

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

L'indirizzo è già pronto al termine della fase di EX. Scrivo il PC durante la fase di EX, il nuovo indirizzo diventa disponibile solamente per l'istruzione che è in fase IF dopo la EX della beq. Ho 2 istruzioni da eliminare.

e

Esecuzione speculativa

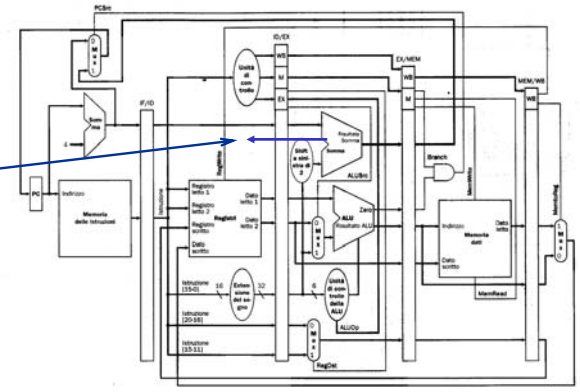
Mettere in stallo ad ogni branch una pipeline è troppo costoso.

Si assume che branch salti (o non salti), cf. cicli.

Se in fase di EXE ci si accorge che è stato commesso un errore, occorre eliminare (flush the instructions) le istruzioni in pipe nelle fasi IF ed ID.

Occorre modificare la CPU.

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.

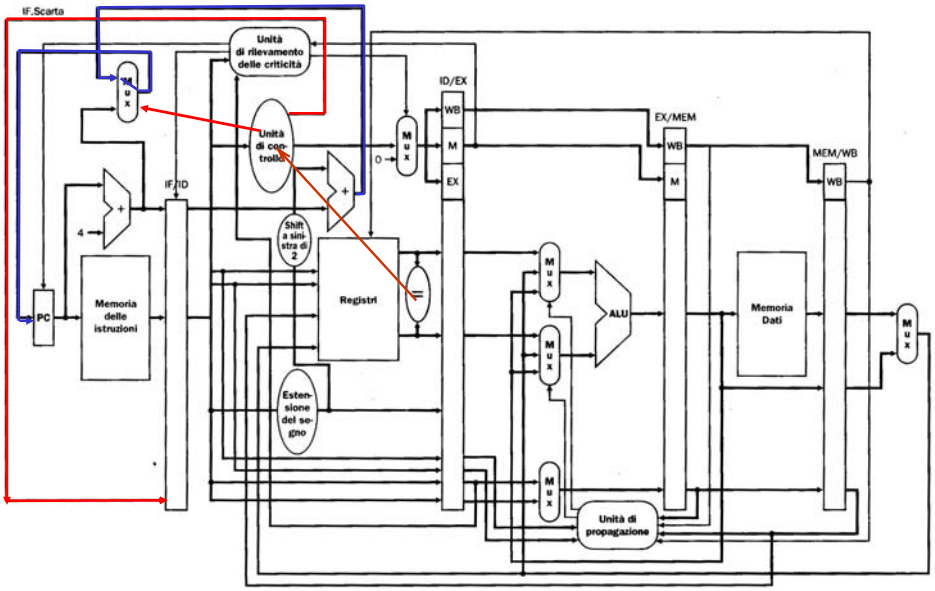
Dalla fase ID alla WB la beq non fa nulla.

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.

CPU con pipeline completa della gestione degli hazard.



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

Problema: come ottimizzare il funzionamento ottenendo il minimo scarto.

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).

Esempio di riorganizzazione del codice

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

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

Esempio di delayed branch

Originale

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

```
                add $s4, $t0, $t1
                beq $s5, $s6, salto
                sub $t5, $t8, $s8
                add $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.
L'istruzione inserita: add $t5, $t4, $t3 o sub $t5, $t8, $s8 riempie un "branch delay slot".

Sommario

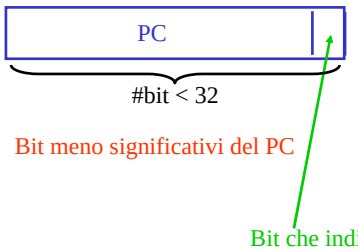
Modifiche alla CPU per la gestione di criticità sui dati, istruzioni di lw.

Hazard sul controllo

Trend delle pipeline

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.

Evoluzioni della branch prediction

1) **Correlating predictors.** Due bit. Informazione locale (branch) + informazione globale (di tutte le branch del modulo).

2) **Tournament predictors.** Vengono utilizzati predittori multipli, 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.

Pipeline più spinte

Instruction level parallelism (ILP). Come si può aumentare?

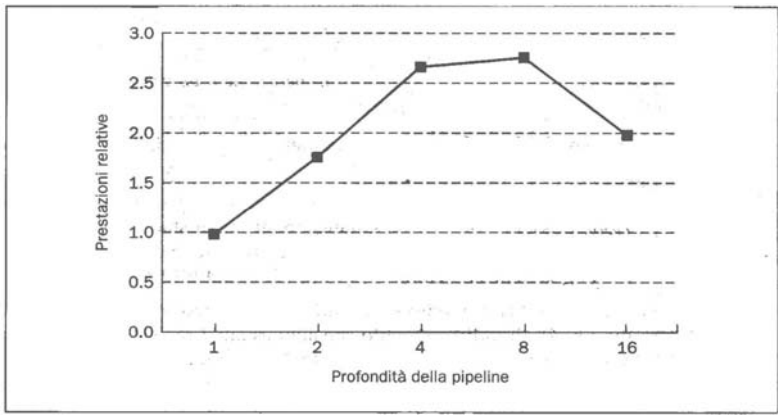
•Superpipelining (pipeline più lunga).

• Superscalar (componenti multipli).

• Scheduling dinamico.

Superpipeline

Pipeline più lunghe (e.g. Digital DecAlpha 21264, 5-7 stadi).
Teoricamente: guadagno in velocità proporzionale al numero di stadi.



Problemi:

•Criticità sui dati: stalli più frequenti.

•Criticità sul controllo: numero maggiore di stadi di cui annullare l'esecuzione.

•Maggior numero di registri: il clock non si riduce linearmente con il numero degli stadi.

Schedulazione avanzata in una pipeline

Duplico unità funzionali (e.g. ALU e ALUfp) ed eseguo più istruzioni in parallelo.

Ottengo più di 1 istruzione per ciclo di clock.

Per il MIPS: 4 porte di lettura, 2 porte di scrittura, 2 ALU (general purpose e base+offset)

CPI < 1 (da 3 ad 8 istruzioni inviate ad ogni ciclo di clock).

Vincoli su quali istruzioni possono essere inviate alla pipe-line contemporaneamente.

Due approcci:
• Static multiple issues (schedulazione decisa dal compilatore)
• Dynamic multiple issues (schedulazione decisa run-time dalla CPU).

Requisiti pipe-line superscalari

Impacchettamento delle istruzioni in **issue slot**: esaminando il codice, quante e quali istruzioni possono essere impacchettate in un ciclo di clock?

Occorre gestire gli hazard nei dati. Nei sistemi statici, questi hazard vengono analizzati dal compilatore, occorre prevedere dei sistemi (e.g. forwarding), nei sistemi dinamici.

Ricorso massiccio alla **speculazione** (= immaginare degli scenari). Ad esempio si può speculare sul risultato di una branch, e quindi eseguire le istruzioni di conseguenza. La speculazione si paga in termini di meccanismi per **controllare** se la speculazione è stata corretta e di **correggerla**. Il compilatore può utilizzare la speculazione per riordinare le istruzioni, la CPU per riordinare l'esecuzione run-time.

La soluzione ai problemi posti dalla speculazione è diversa:

- Il compilatore inserisce delle istruzioni di controllo e di correzione a speculazioni errate.
- La CPU mette in buffer i risultati dell'esecuzione speculativa fino a quando non si è potuto verificare la correttezza della speculazione. Nel caso di speculazione errata, la cancellazione del lavoro fatto viene ottenuta semplicemente svuotando i buffer.

Multiple issue statici (schedulazione statica)

Issue packet è l'insieme delle istruzioni inviate alla pipeline in un ciclo di clock = 1 grande istruzioni con tante operazioni e tanti operandi.

L'approccio ILP di fatto vede le istruzioni come una **VLIW** (Very Long Instruction Word). L'architettura INTEL IA-64, utilizza questo approccio che ha battezzato **EPIC** (Explicitly Parallel Instruction Computer): Itanium (2000), Itanium-2 (2002).

Dynamic issues (schedulazione dinamica)

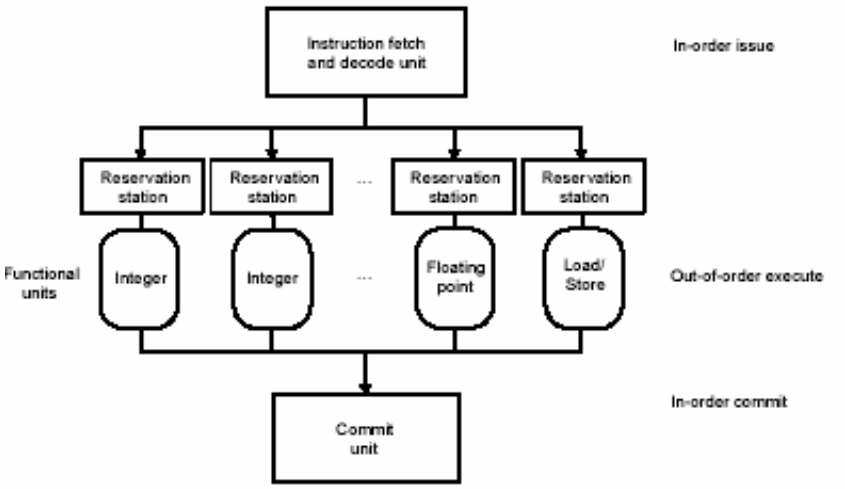
Questi processori sono detti anche **superscalari**.

La scelta di quali istruzioni inviare alla pipe-line viene eseguita durante l'esecuzione stessa. Dipende dalla compatibilità tra le varie istruzioni e da eventuali hazard su dati e controllo.

Per evitare stalli, le istruzioni possono essere scambiate di ordine.

Le componenti principali sono tre:
•Instruction fetch and issue
•Multiple functional units (fino a 10 nelle CPU di alto livello – 2004).
•Commit unit (unità di uscita).

Pipeline con schedulazione dinamica



Esistono diversi cammini paralleli (per la fase di esecuzione e memoria) dell'istruzione.

Principi della schedulazione dinamica

Obbiettivo: mettere in esecuzione istruzioni che non presentino criticità.

Le istruzioni vengono bufferizzate dalla **reservation station**, la quale gestisce la coda delle istruzioni che hanno bisogno della stessa unità funzionale.

Al termine dell'esecuzione la **reorder station**, provvede ad ordinare le istruzioni nella sequenza con la quale devono essere restituite.

NB Le istruzioni non sono eseguite sequenzialmente.

Sommario

Modifiche alla CPU per la gestione di criticità sui dati, istruzioni di lw.

Hazard sul controllo

Trend delle pipeline