


Hazard e forwarding

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

Università degli Studi di Milano

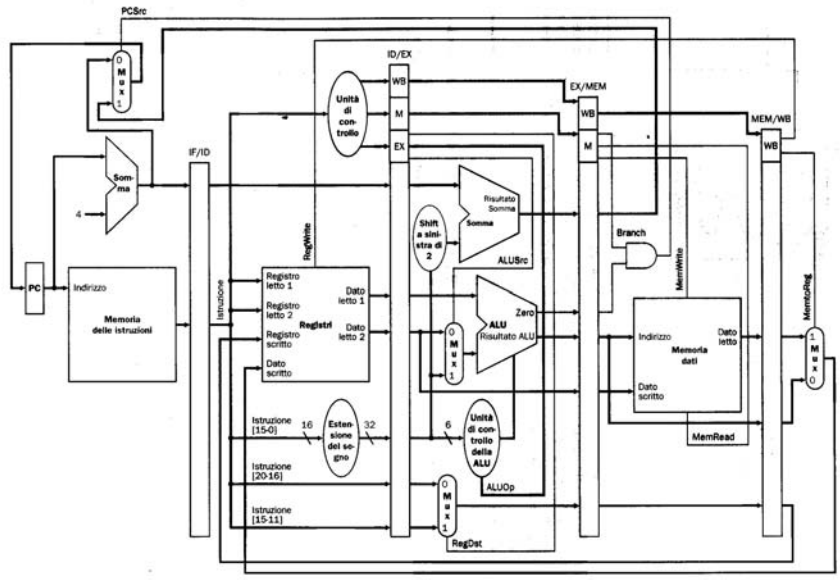
Riferimento al Patterson: 6.4 e 6.5


Sommario

Gli Hazard di una pipeline

Soluzione delle criticità sui dati

CPU con pipeline



Criticità (hazard)

Un'istruzione non può essere eseguita nel ciclo di clock immediatamente successivo a quella precedente (mancano i dati necessari alla lavorazione di un qualche suo stadio).

Strutturali:

- Dovrei utilizzare la stessa unità funzionale due volte nello stesso ciclo di clock (e.g. se non avessi duplicato la memoria)..

Controllo:

- Dovrei prendere una decisione (sull'istruzione successiva) prima che l'esecuzione dell'istruzione corrente sia terminata (e.g. Istruzioni successive ad una branch).

Dati:

- Dovrei eseguire un'istruzione in cui uno dei dati è il risultato dell'esecuzione di un'istruzione precedente.

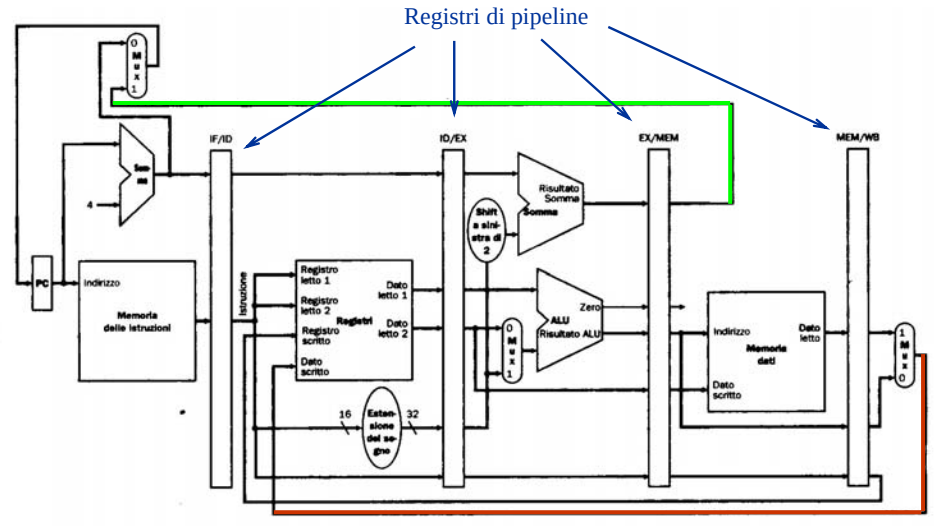
Esempio:

```
add $s0, $t1,$t1
add $s2, $s0, $t3
```

Esempio di Hazard sui dati

sub $s2, $s1, $s3	IF	ID	EX $1-$3	MEM	WB s->$2				
add $t2, $s2, $s5		IF	ID	EX $2 and $5	MEM	WB s->$t2			
or $t3, $s6, $s2			IF	ID	EX $6 or $2	MEM	WB (s->$t3)		
and $t4, $s2, $s2				IF	ID	EX $2 +$2	MEM	WB s->$t4	
sw $t5, 100($s2)					IF	ID	EX $2+100	MEM $t5 ->Mem	WB

CPU con pipeline



Soluzione delle criticità strutturali

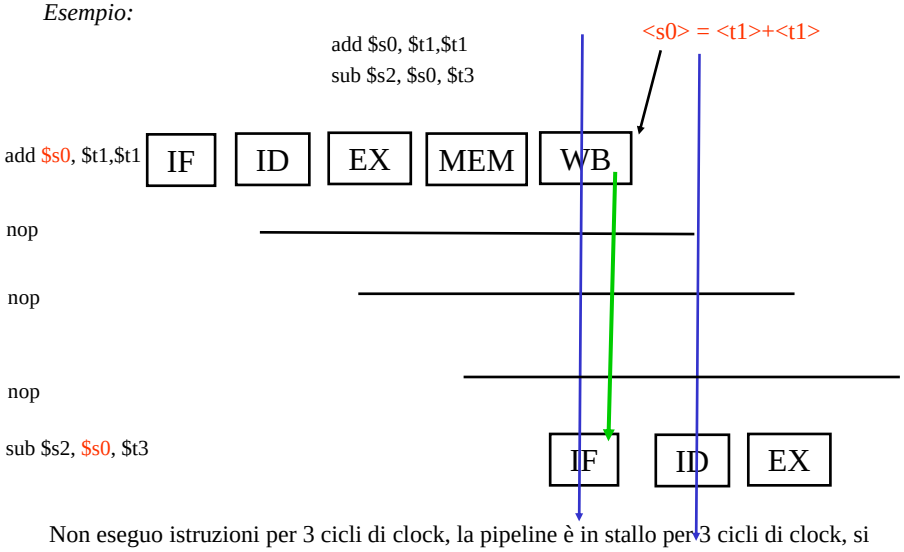
Le criticità strutturali sono risolte con la duplicazione (suddivisione) delle unità funzionali.

Triplicazione delle ALU

Duplicazione della Memoria (stessa memoria ma separazione della memoria dati dalla memoria istruzioni).

Soluzione mediante stallo

Esempio:



Non eseguo istruzioni per 3 cicli di clock, la pipeline è in stallo per 3 cicli di clock, si formano 3 bolle (bubbles) nel funzionamento della pipeline.

Hazard nei dati: soluzione tramite compilatore

add $s0, $t1,$t1
nop
nop
nop
sub $s1, $s0, $s0
and $t2, $t0 $t1
or $t5, $t3, $t4
add $s2, $s7, $t7
sw $3, 100($t0)

add $s0, $t1,$t1
and $t2, $t0 $t1
or $t5, $t3, $t4
add $s2, $s7, $t7
sub $s2, $s0, $t3
sw $15, 100($s2)

Spreco di 3 cicli di clock (in modo che la fase IF dell'istruzione *sub $s2, $s0, $t3* vada a coincidere con la fase di WB della *add $s0, $t1,$t1*.

Situazione troppo frequente perché la soluzione sia accettabile.

Il codice viene riorganizzato in fase di compilazione. Non sempre è possibile o efficace.

Sommario

Gli Hazard di una pipeline

Soluzione delle criticità sui dati

Criticità nei dati

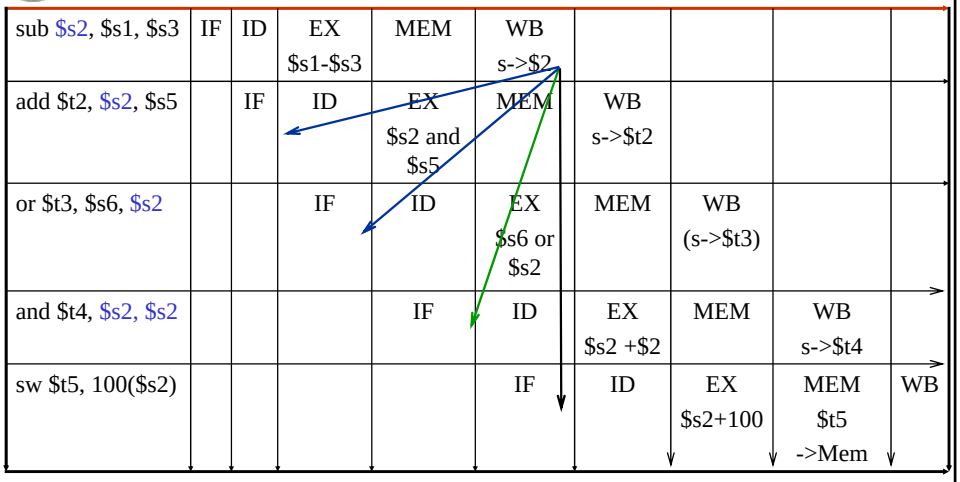
Dovrei eseguire un'istruzione in cui uno dei dati è il risultato dell'esecuzione di un'istruzione precedente.

Soluzione mediante due tecniche:

Riorganizzazione del codice (compilatore).

Propagazione (forwarding) o scavalcamento (bypassing).

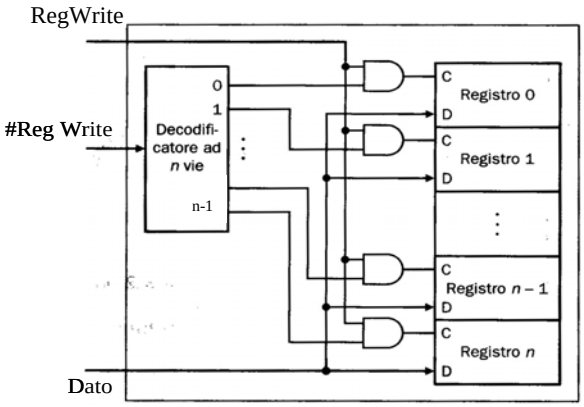
Hazard sui dati



Con le frecce sono indicate le dipendenze, in blu gli hazard (tra sub e and, sub e add).

Il dato in $s2 viene scritto nel Register File nella fase di WB della sub, è pronto al clock successivo. Non è ancora pronto quando viene effettuata la decodifica della and, della or e della add successiva.

E' una criticità il Register File?



No, a patto che utilizziamo Latch di tipo D e non flip-flop come nel caso della CPU singolo ciclo

Hazard sui dati

sub $s2, $s1, $s3	IF	ID	EX $s1-$s3	MEM	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)		
and $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

Con le frecce sono indicate le dipendenze, in blu gli hazard (tra sub, e and e or), dopo la modifica del RegisterFile: il dato è disponibile in lettura, già nella prima parte del clock.
Il dato in $s2 viene scritto nel Register File nella fase di WB della sub, è pronto al clock successivo. Non è ancora pronto quando viene effettuata la decodifica della and e della or successiva.

Soluzione architetturale della criticità sui dati

La criticità nei dati ha a che fare essenzialmente con la disponibilità di dati corretti.

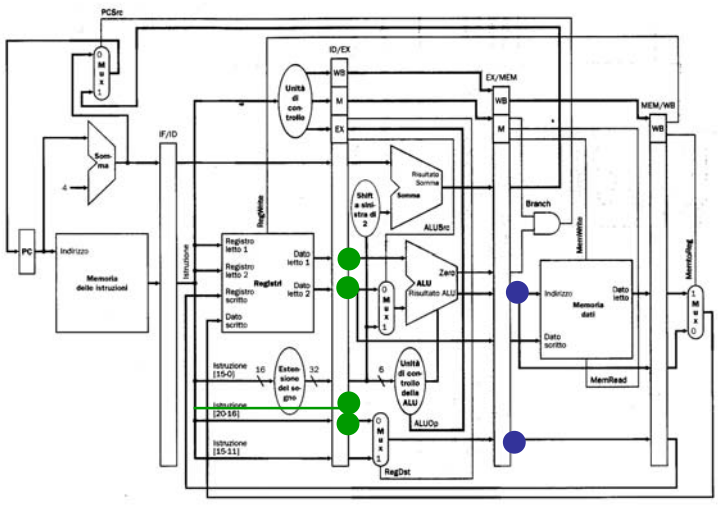
Identificazione della criticità (funzione del tipo di istruzione e dei registri coinvolti).

Correzione della situazione: propagazione a ritroso (negli stadi della pipeline = in avanti nel tempo) su data-path alternativi dei dati richiesti.

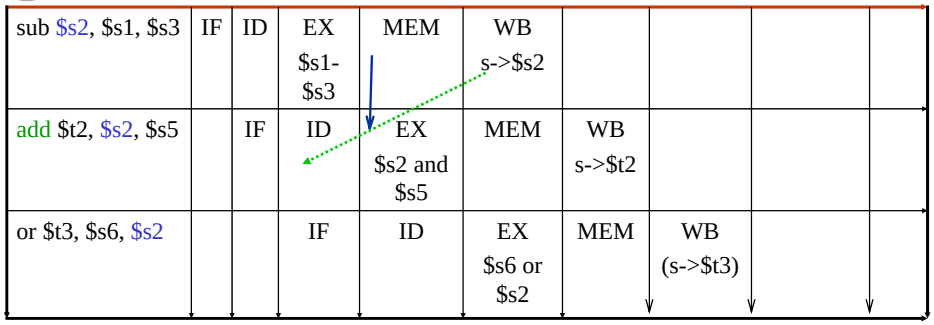
Identificazione delle criticità – EX/MEM

sub $s2, $s1, $s3
add $t2, $s2, $s5

1a. EX/MEM.RegistroRd = ID/EX.RegistroRs
1b. EX/MEM.RegistroRd = ID/EX.RegistroRt



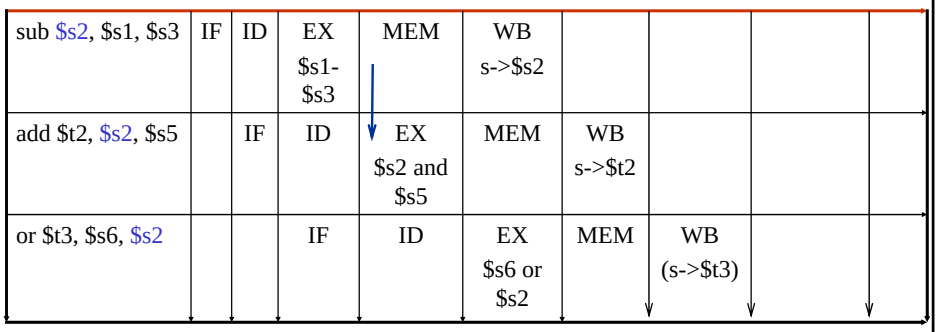
Hazard sui dati: rilevamento della criticità



Rilevo la criticità (dato non corretto) su and quando and inizia la fase di EX. In questo caso il dato corretto si trova all'inizio della fase MEM della sub.

Questo modo di rilevare la criticità consente di ottenere i datapath più brevi all'interno della CPU.

Hazard sui dati: formalizzazione della criticità

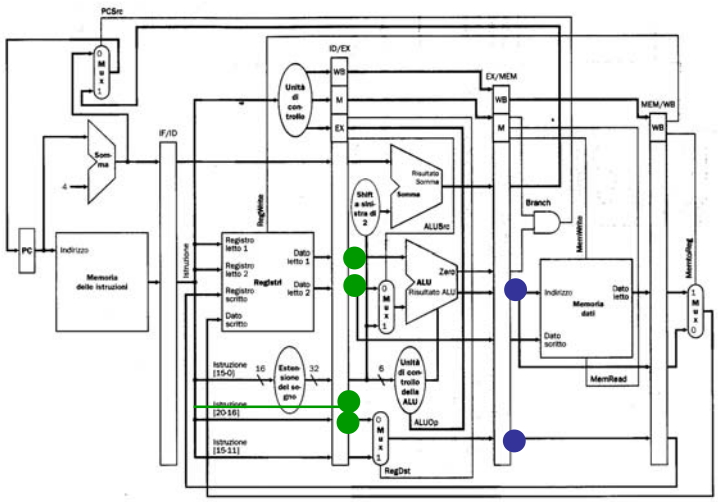


IF ((EX/MEM.RegistroRd == ID/EX.RegistroRs) ||
((EX/MEM.RegistroRd == ID/EX.RegistroRt)) &&
(EX/MEM.RegisterWrite)

Abbiamo identificato il problema, dobbiamo ora risolverlo

sub $s2, $s1, $s3
add $t2, $s2, $s5

IF ((EX/MEM.RegistroRd == ID/EX.RegistroRs) ||
((EX/MEM.RegistroRd == ID/EX.RegistroRt)) &&
(EX/MEM.RegisterWrite)



Hazard sui dati: feed-forwarding

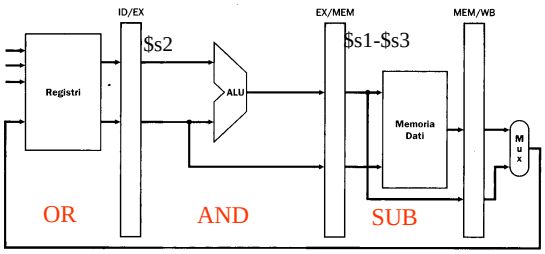
sub $s2, $s1, $s3	IF	ID	EX $s1-$s3	MEM	WB s->$s2				
add $t2, $s2, $s5		IF	ID	EX $s2 and $s5	MEM	WB s->$t2			
or $t3, $s6, $s2			IF	ID	EX $s6 or $s2	MEM	WB (s->$t3)		

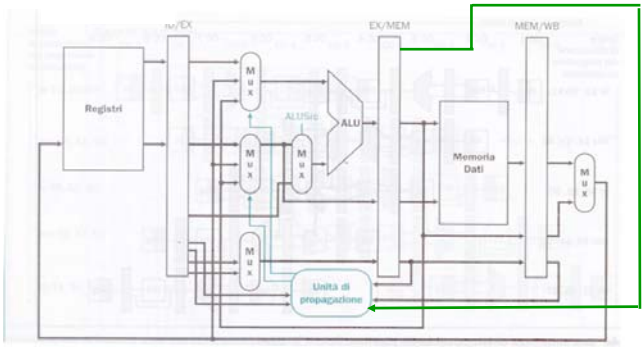
IF (EX/MEM.RegistroRd == ID/EX.RegistroRs) && (EX/MEM.RegisterWrite)
ALUSrcA = <EX/MEM.RegistroRd>

IF (EX/MEM.RegistroRd == ID/EX.RegistroRt) && (EX/MEM.RegisterWrite)
ALUSrcB = <EX/MEM.RegistroRd>

Hazard nei dati: forwarding

sub $s2, $s1, $s3
add $t2, $s2, $s5
or $t3, $s6, $s2



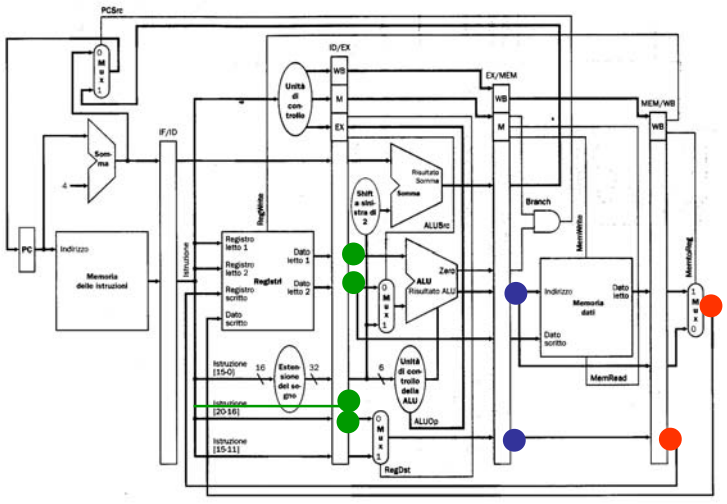


"RegWrite = 1"

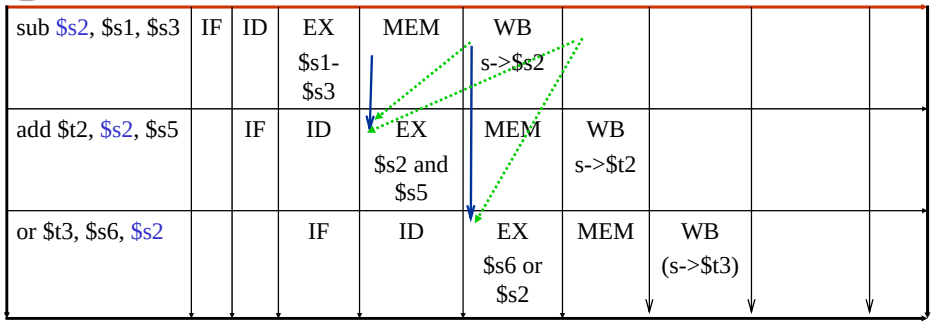
Identificazione delle criticità – MEM/WB

sw $s1, 100($t1)
sub $s2, $s1, $s3
add $t2, $s2, $s5
or $t3, $s6, $s2

2a. MEM/WB.RegstroRd = ID/EX.RegistroRs
2b. MEM/WB.RegistroRd = ID/EX.RegistroRt



Hazard sui dati: rilevamento della criticità

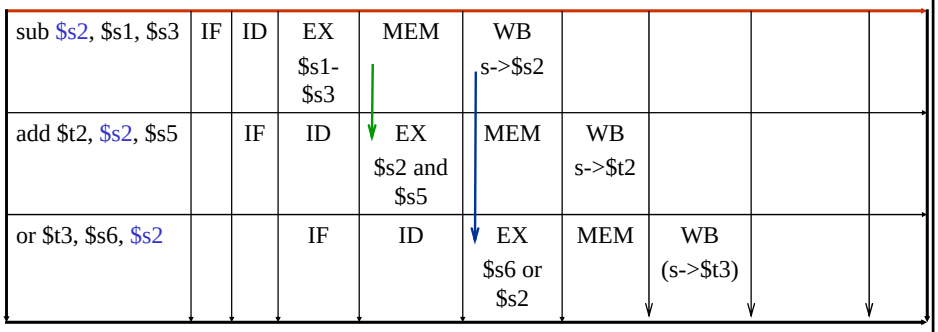


Rilevo la criticità (dato non corretto) su and quando and inizia la fase di EX. In questo caso il dato corretto si trova all'inizio della fase MEM della sub.

➔ 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 sub.

Questo modo di rilevare la criticità consente di ottenere i datapath più brevi all'interno della CPU.

Hazard sui dati: formalizzazione della criticità



IF ((MEM/WB.RegistroRd == ID/EX.RegistroRs) ||
(MEM/WB.RegistroRd == ID/EX.RegistroRt)) &&
(MEM/WB.RegisterWrite)

Hazard sui dati: feed-forwarding

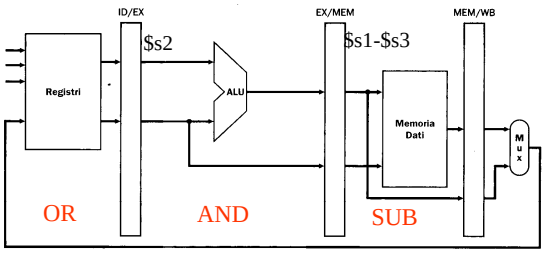
sub $s2, $s1, $s3	IF	ID	EX $s1- $s3	MEM	WB s->$s2				
add $t2, $s2, $s5		IF	ID	EX $s2 and $s5	MEM	WB s->$t2			
or $t3, $s6, $s2			IF	ID	EX $s6 or $s2	MEM	WB (s->$t3)		

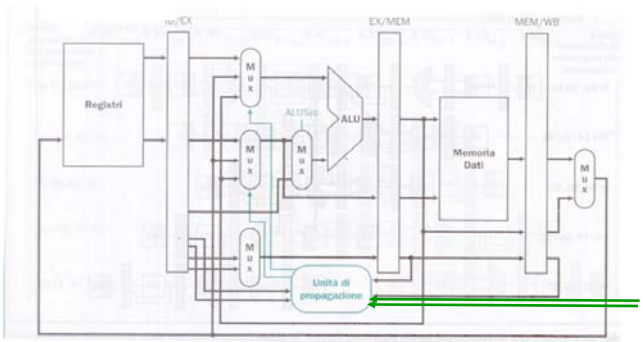
IF (MEM/WB.RegistroRd == ID/EX.RegistroRs) && (MEM/WB.RegisterWrite)
ALUSrcA = <MEM/WB.RegistroRd>

IF (MEM/WB.RegistroRd == ID/EX.RegistroRt) && (MEM/WB.RegisterWrite)
ALUSrcB = <MEM/WB.RegistroRd>

Hazard nei dati: forwarding

sub $s2, $s1, $s3
add $t2, $s2, $s5
or $t3, $s6, $s2





EX/MEM.RegWrite
MEM/WB.RegWrite

Relazione tra forwarding e contenuto del registro ID/EX

Nel normale funzionamento, il registro ID/EX contiene quanto letto dal Register File.

Quando abbiamo forwarding, quello che viene letto dal registro ID/EX nella fase di esecuzione viene sovrascritto da quanto letto dal registro EX/MEM o MEM/WB.

Nel registro EX/MEM è contenuto il risultato dell'operazione eseguita all'istante precedente.

Nel registro MEM/WB è contenuto il risultato dell'operazione eseguita 2 istanti precedenti.

Controllo Mux ingrsso alla ALU

Controllo Multiplexer	Registro Sorgente	Funzione
PropagaA = 00	ID/EX	Il primo operando della ALU proviene dal Register File
PropagaA = 01	EX/MEM	Il primo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaA = 10	MEM/WB	Il primo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.
PropagaB = 00	ID/EX	Il secondo operando della ALU proviene dal Register File
PropagaB = 01	EX/MEM	Il secondo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaB = 10	MEM/WB	Il secondo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.

Unità di controllo del forwarding

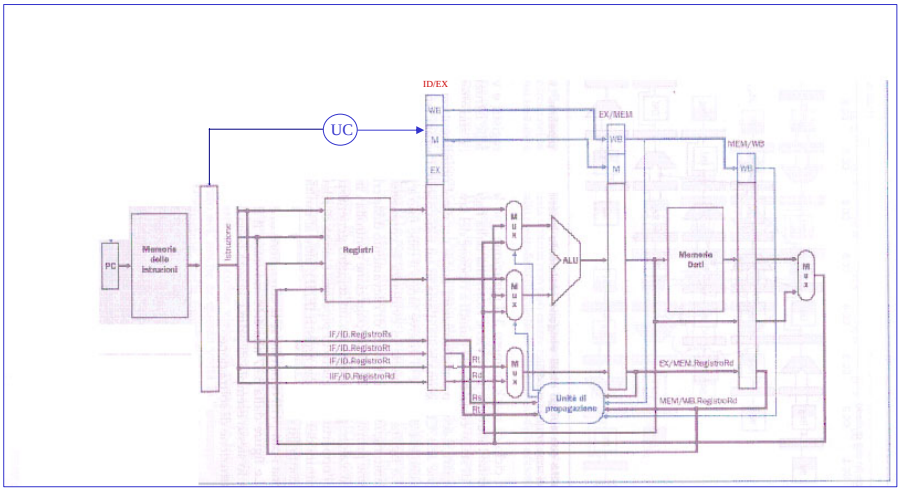
Deve controllare che la criticità sia effettiva (che l'istruzione precedente scriva il RegisterFile).

E' attiva nella fase di esecuzione (EX) ed implementa le seguenti funzioni:

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

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)

CPU con unità di propagazione



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

Gli Hazard di una pipeline

Soluzione delle criticità sui dati