SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori
Gestione dei control-hazard nella pipeline

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

- Spostare Jump nella fase IF
- Come gestire i control hazard
  - Eliminare le istruzioni non più necessarie
- Come ridurre l'impatto del control hazard
  - Anticipare la decisione di salto
  - Ritardare l'esecuzione del salto
  - «prevedere» la destinazione del salto
- Soluzione esercizio

Argomenti

- Spostare Jump nella fase IF
- Come gestire i control hazard
  - Eliminare le istruzioni non più necessarie
- Come ridurre l'impatto del control hazard
  - Anticipare la decisione di salto
  - Ritardare l'esecuzione del salto
  - «prevedere» la destinazione del salto
- Soluzione esercizio

Jump:

Argomenti

- Spostare Jump nella fase IF
- Come gestire i control hazard
  - Eliminare le istruzioni non più necessarie
- Come ridurre l'impatto del control hazard
  - Anticipare la decisione di salto
  - Ritardare l'esecuzione del salto
  - «prevedere» la destinazione del salto
- Soluzione esercizio

- **Jump:**
- l'istruzione viene riconosciuta nella fase ID mentre un'altra viene caricata
- quindi il Jump implica almeno uno stallo (l'istruzione successiva)

Argomenti

- Spostare Jump nella fase IF
- Come gestire i control hazard
  - Eliminare le istruzioni non più necessarie
- Come ridurre l'impatto del control hazard
  - Anticipare la decisione di salto
  - Ritardare l'esecuzione del salto
  - «prevedere» la destinazione del salto
- Soluzione esercizio

Jump:

l'istruzione viene riconosciuta nella fase ID mentre un'altra viene caricata

quindi il Jump implica almeno uno stallo (l'istruzione successiva)

Control-hazard:

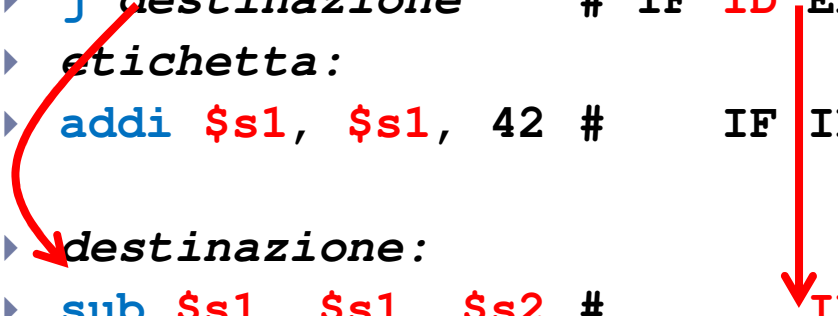
quando deve essere fatto un salto la pipeline è mezza piena e

le istruzioni seguenti già caricate vanno eliminate dalla pipeline

Un inutile stallo per il Jump!

- La decisione di eseguire il Jump viene presa dalla CU nella fase ID
- - nel frattempo un'altra istruzione è stata già caricata e va eliminata (SEMPRE)

```
j destinazione     # IF ID EX MM WB
etichetta:
addi $s1, $s1, 42 #    IF ID EX MM WB ist. da eliminare

destinazione:
sub $s1, $s1, $s2 #       IF ID EX MM WB
```



Un inutile stallo per il Jump!

- La decisione di eseguire il Jump viene presa dalla CU nella fase ID
- - nel frattempo un'altra istruzione è stata già caricata e va eliminata (SEMPRE)

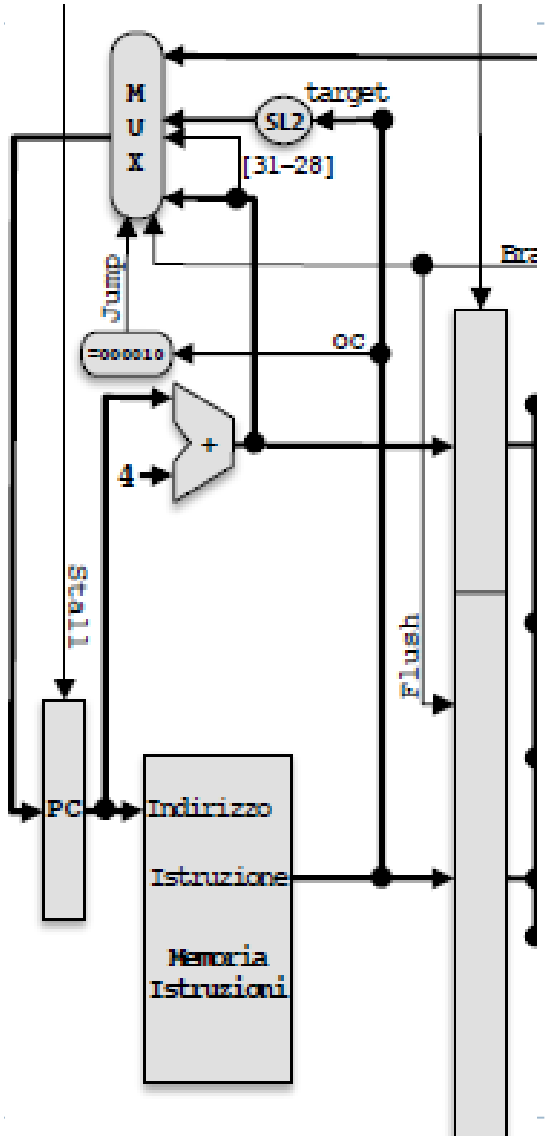
```
j destinazione     # IF ID EX MM WB
etichetta:
addi $s1, $s1, 42 #    IF ID EX MM WB ist. da eliminare

destinazione:
sub $s1, $s1, $s2 #       IF ID EX MM WB
```

- Per **eliminare l'istruzione** con uno stallo, dobbiamo (nella fase ID):

- annullare l'istruzione inutile (**una bolla che continuerà senza fare nulla**) (ovvero basta **azzerare i segnali di controllo MemWrite e RegWrite e IF/ID)**
- **e aggiornare il PC** perchè possa leggere la destinazione al prossimo colpo di clock

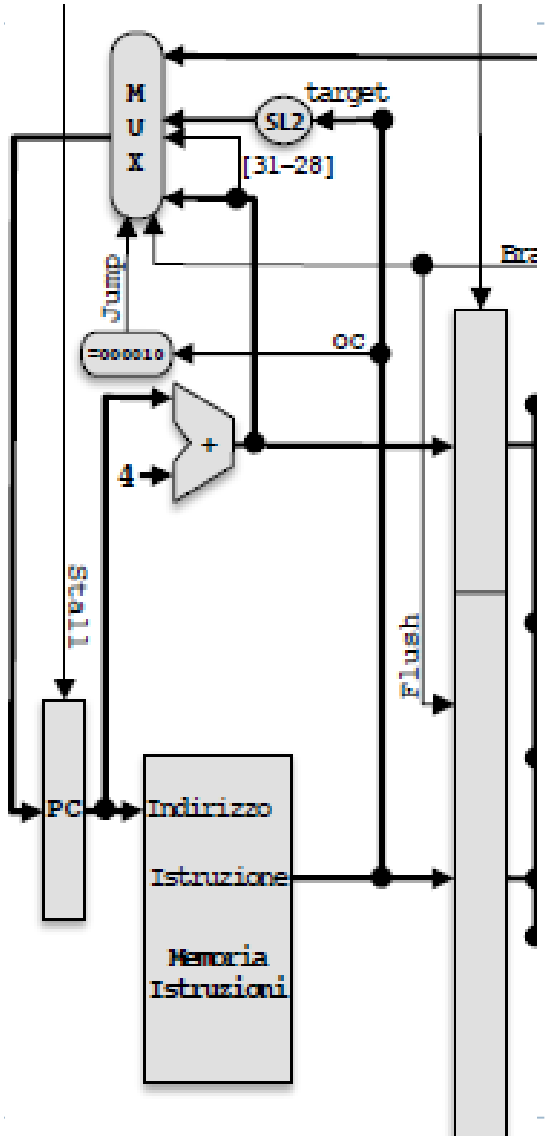
Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:



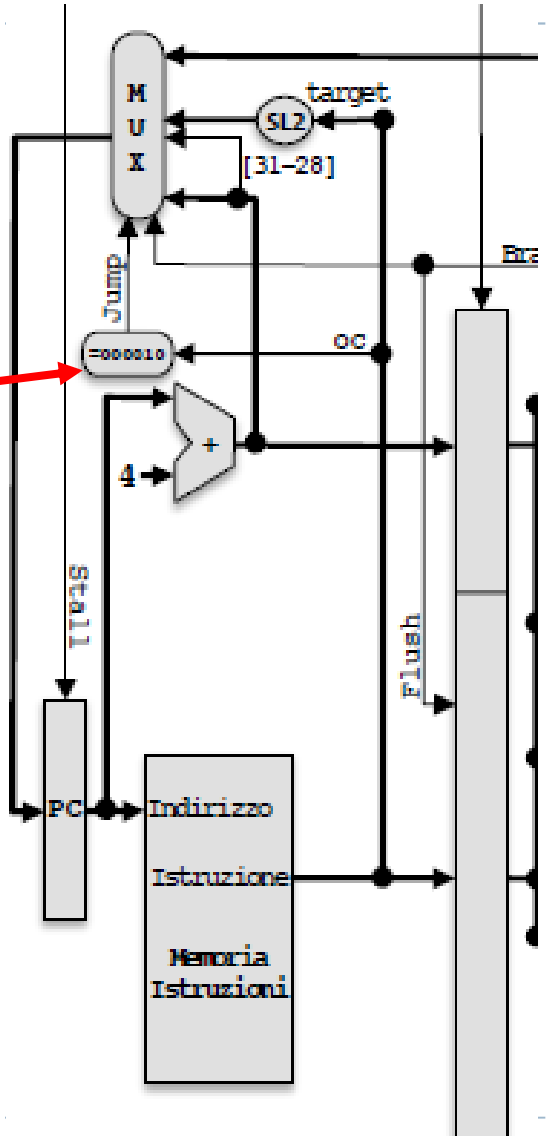
Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:
- **- anticipare il riconoscimento della istruzione**
  (che normalmente fa la CU in fase ID)
- - basta un comparatore col valore dell'Opcode della J (000010)



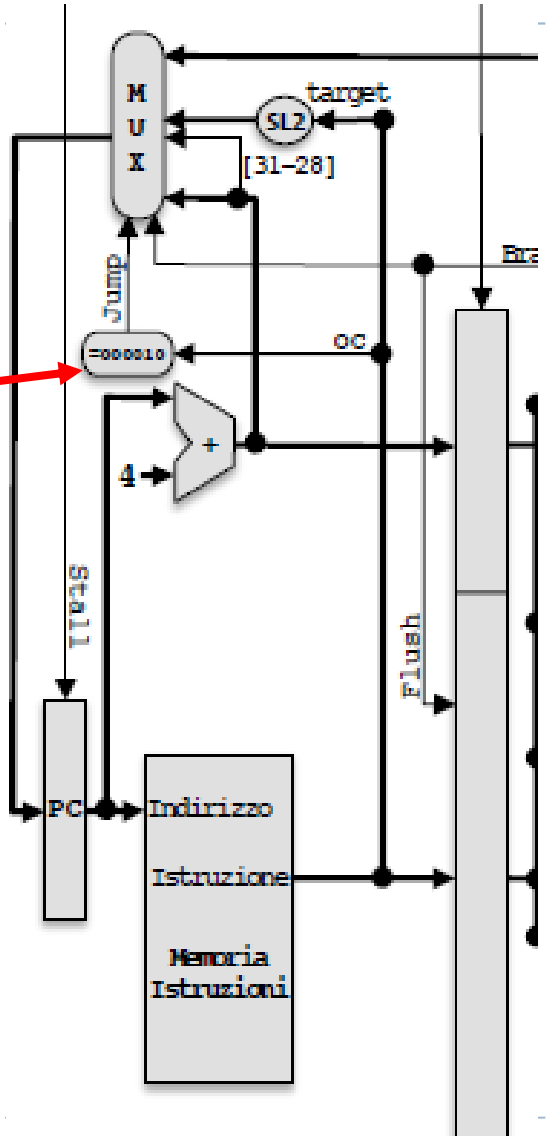
Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:
- **- anticipare il riconoscimento della istruzione**
  (che normalmente fa la CU in fase ID)
- - basta un comparatore col valore dell'Opcode della J (000010)



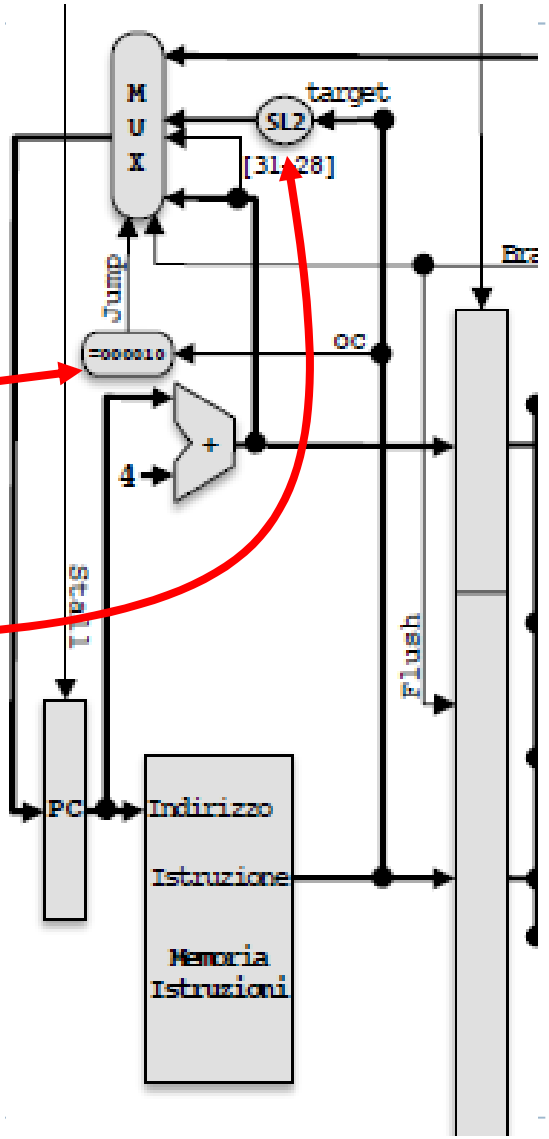
Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:
- **- anticipare il riconoscimento della istruzione**
  (che normalmente fa la CU in fase ID)
- - basta un comparatore col valore dell'Opcode della J (000010)
- **- spostare la logica di aggiornamento del PC alla fase IF**
- - **SL2** dei 26 bit meno significativi della istruzione
  …
- … e aggiunta dei 4 bit più significativi di PC+4



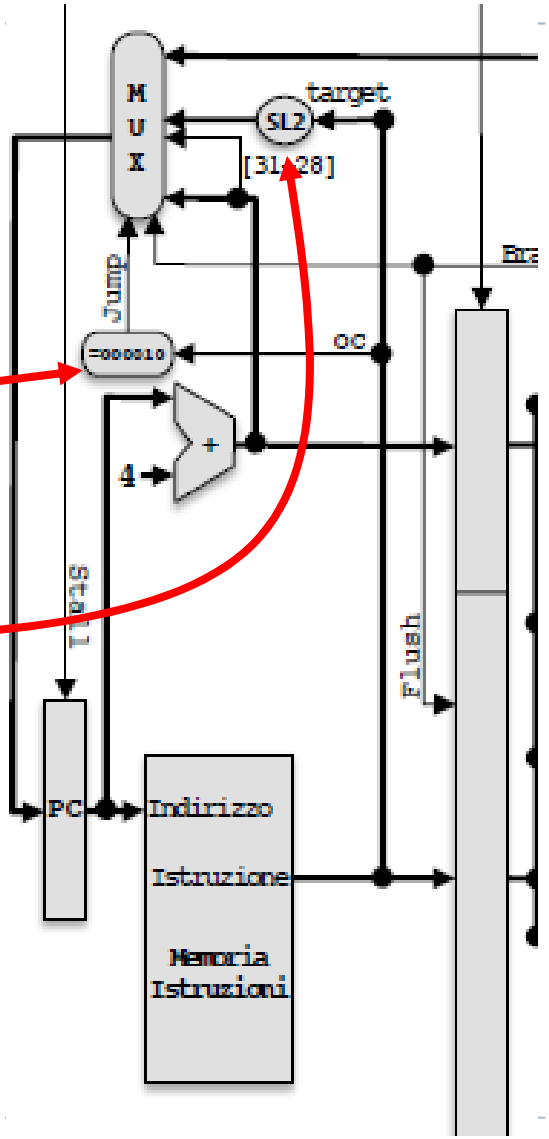
Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:
- **- anticipare il riconoscimento della istruzione**
  (che normalmente fa la CU in fase ID)
- - basta un comparatore col valore dell'Opcode della J (000010)
- **- spostare la logica di aggiornamento del PC alla fase IF**
- - **SL2** dei 26 bit meno significativi della istruzione
  …
- … e aggiunta dei 4 bit più significativi di PC+4



Anticipare il Jump alla fase ID

- Per saltare con la J già nella fase IF dobbiamo:
- **- anticipare il riconoscimento della istruzione**
  (che normalmente fa la CU in fase ID)
- - basta un comparatore col valore dell'Opcode della J (000010)
- **- spostare la logica di aggiornamento del PC alla fase IF**
- - **SL2** dei 26 bit meno significativi della istruzione
  …
- … e aggiunta dei 4 bit più significativi di PC+4
- Risultato: **la Jump anticipata NON introduce più stalli**



Control hazards (beq)

- **La istruzione beq usa la ALU per fare il confronto, per cui:**

- salta dopo la fase EXE, nella fase MEM
- quando salta, le 2 istruzioni seguenti (già caricate) vanno «eliminate»
- ha bisogno degli argomenti nella fase EXE, quindi il normale forwarding va bene
- può aver bisogno di uno stallo solo se preceduto da lw (come ogni istruz. di tipo R)

Control hazards (beq)

- **La istruzione beq usa la ALU per fare il confronto, per cui:**

- salta dopo la fase EXE, nella fase MEM
- quando salta, le 2 istruzioni seguenti (già caricate) vanno «eliminate»
- ha bisogno degli argomenti nella fase EXE, quindi il normale forwarding va bene
- può aver bisogno di uno stallo solo se preceduto da lw (come ogni istruz. di tipo R)

- **Come eliminare le (2) istruzioni dalla pipeline:**

- IF/ID.Istruzione viene azzerata e diventa NOP
- ID/EXE.MemWrite e ID/EXE.RegWrite vengono azzerate per non modificare nulla

Control hazards (beq)

- **La istruzione beq usa la ALU per fare il confronto, per cui:**

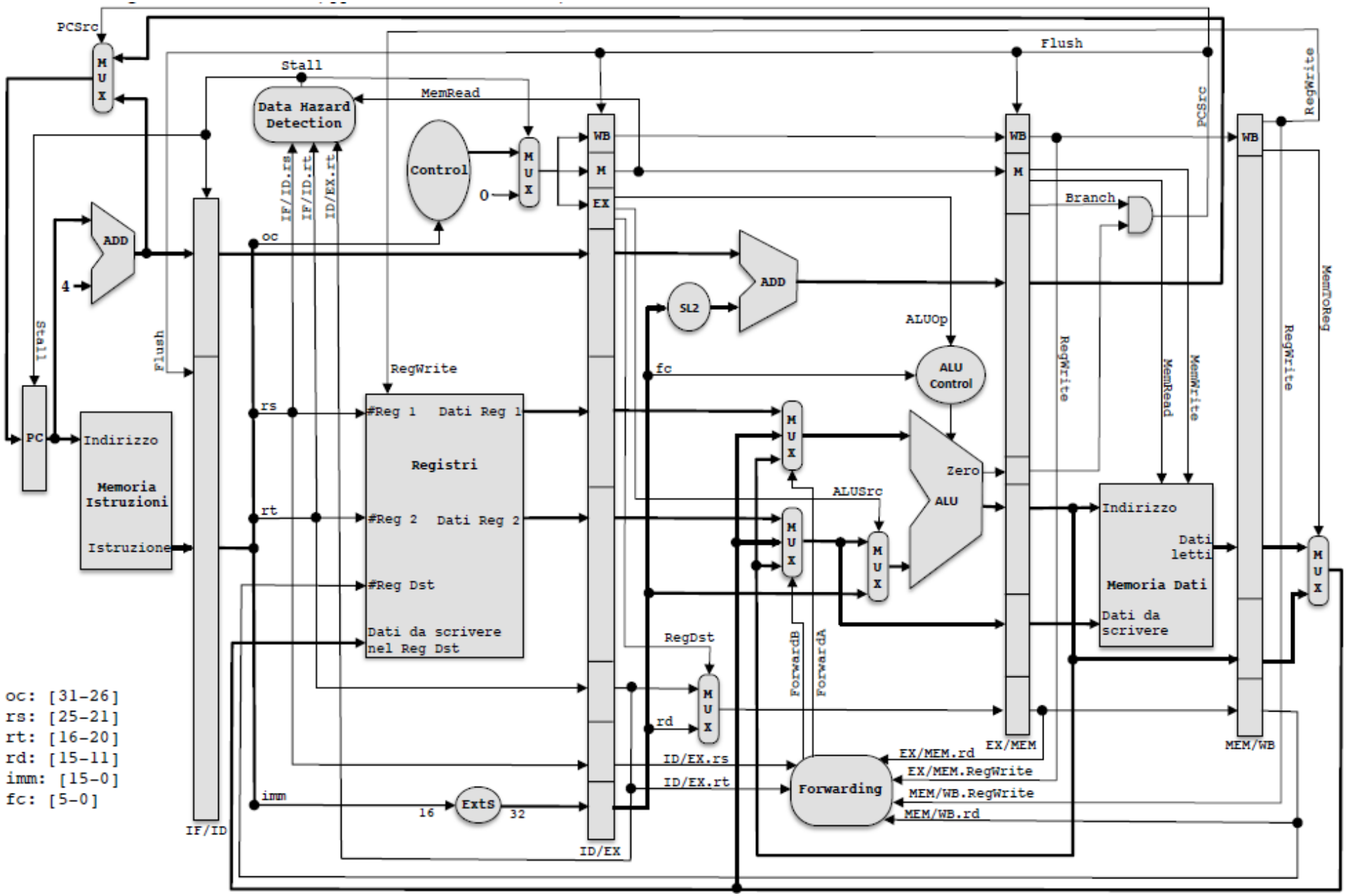
- salta dopo la fase EXE, nella fase MEM
- quando salta, le 2 istruzioni seguenti (già caricate) vanno «eliminate»
- ha bisogno degli argomenti nella fase EXE, quindi il normale forwarding va bene
- può aver bisogno di uno stallo solo se preceduto da lw (come ogni istruz. di tipo R)

- **Come eliminare le (2) istruzioni dalla pipeline:**

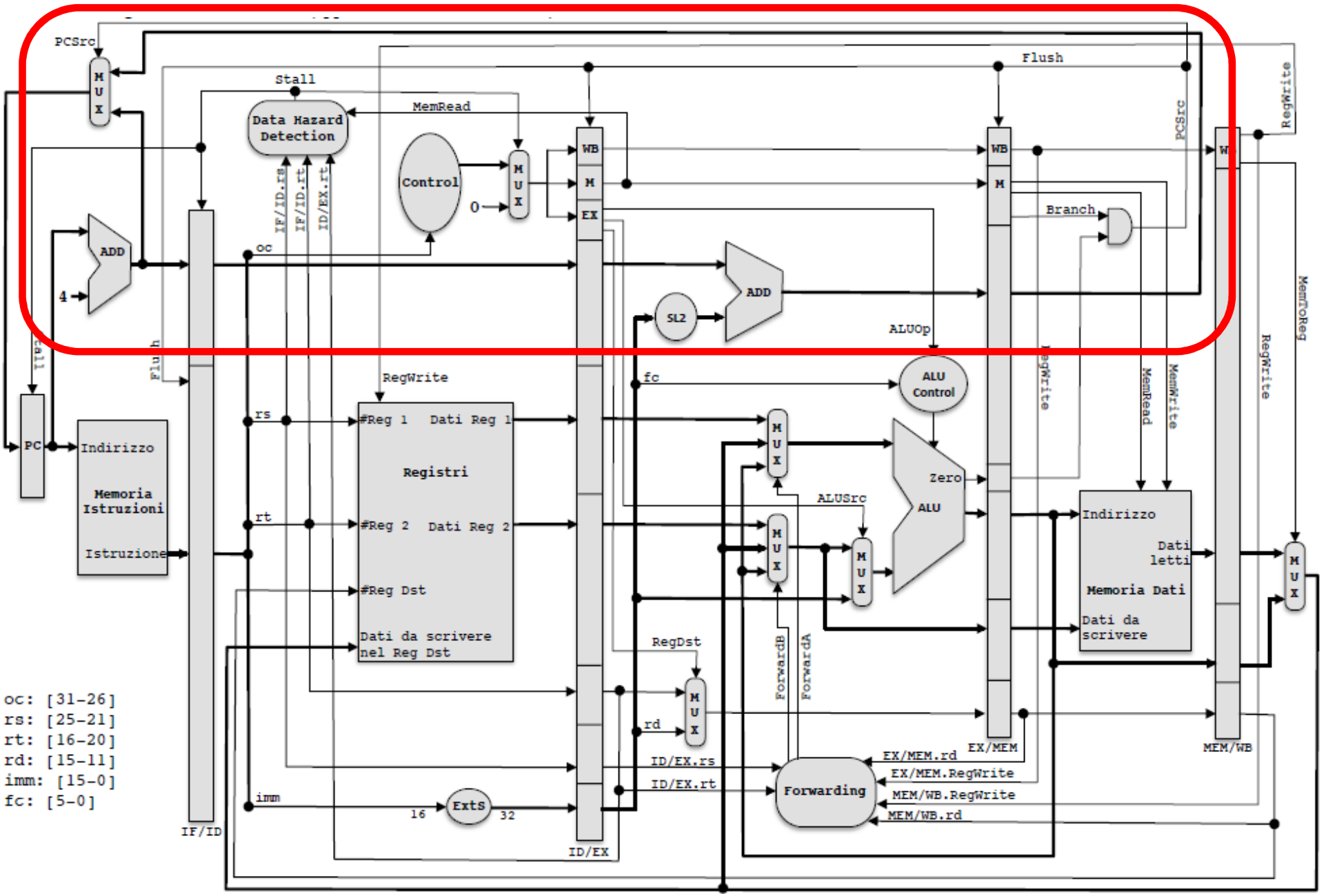
- IF/ID.Istruzione viene azzerata e diventa NOP
- ID/EXE.MemWrite e ID/EXE.RegWrite vengono azzerate per non modificare nulla

- **È possibile anticipare la decisione di salto alla fase ID ma solo se NON si usa la ALU:**

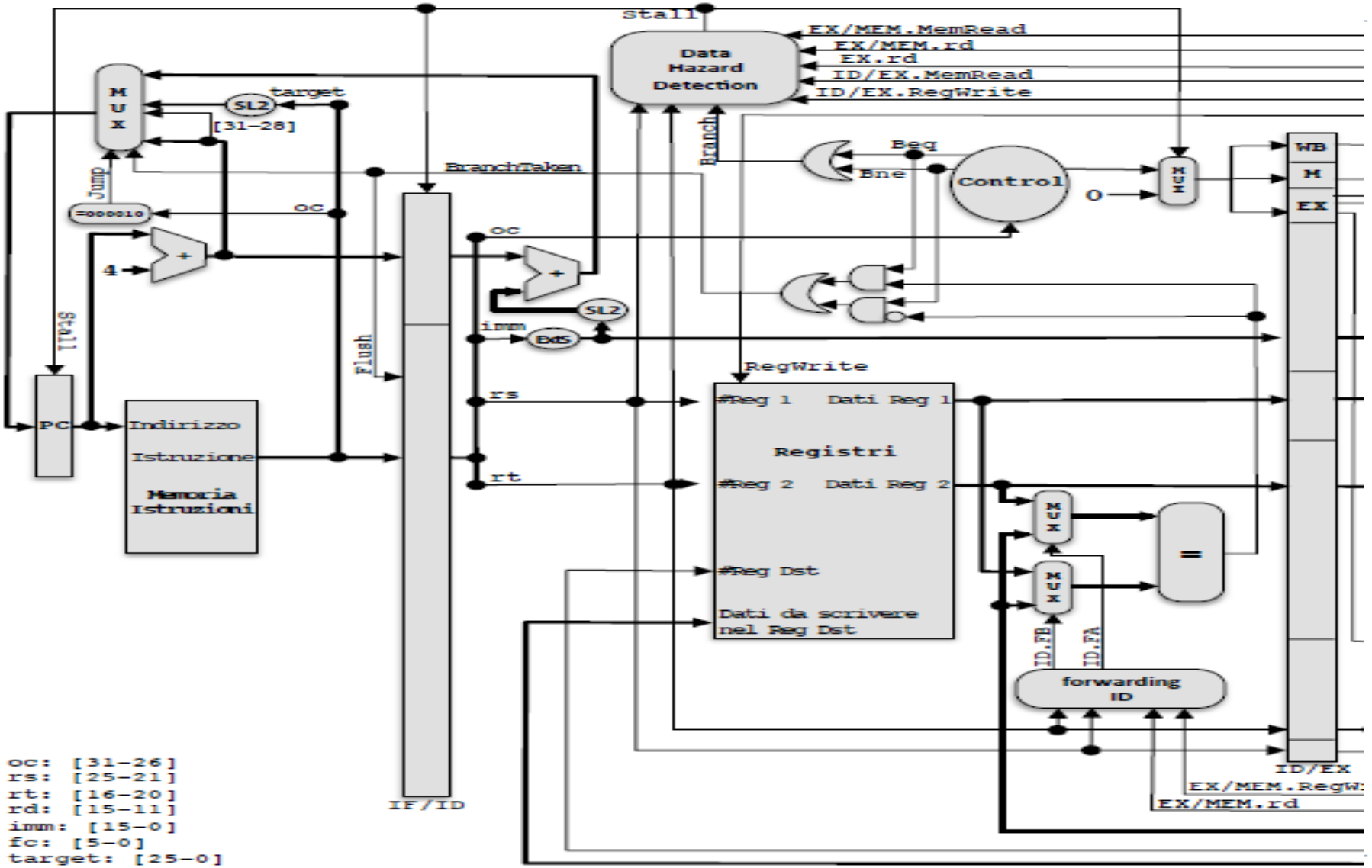
- inserendo un comparatore tra i due argomenti letti dal blocco registri
- spostando la logica di salto e il calcolo del salto relativo dalla fase EXE alla fase ID
- sarà necessaria una unità di forwarding apposta per la fase ID



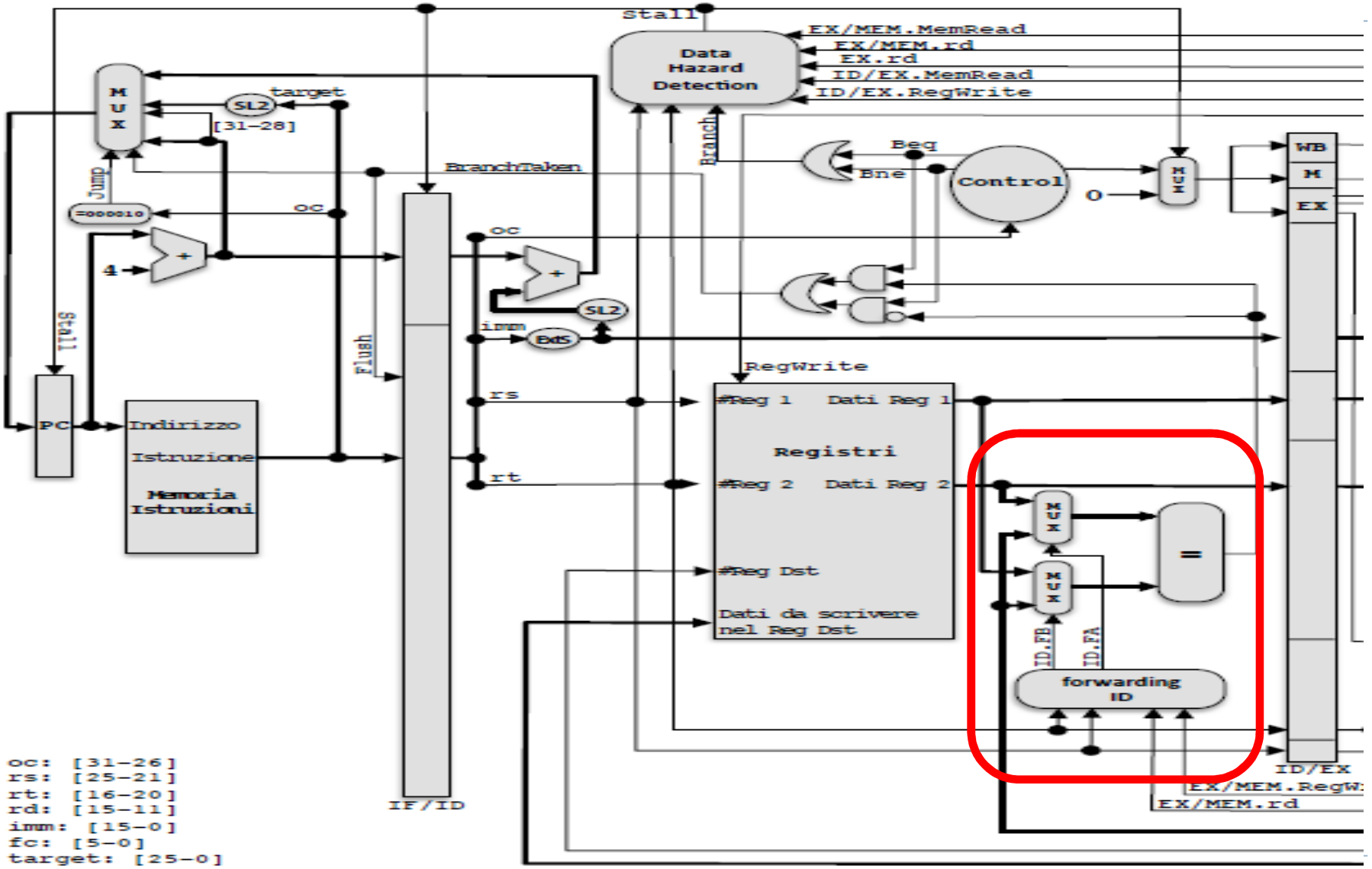
oc: [31-26]
rs: [25-21]
rt: [16-20]
rd: [15-11]
imm: [15-0]
fc: [5-0]


PCSrc
MUX
Stall
Data Hazard Detection
MemRead
Control
0
IF/ID.rs
IF/ID.rt
ID/EX.rt
WB
M
EX
Flush
RegWrite
PCSrc
ADD
4
oc
SL2
ADD
ALUOp
Branch
MemToReg
RegWrite
MemRead
MemWrite
RegWrite
fc
ALU Control
PC
Indirizzo
Memoria Istruzioni
Istruzione
rs
rt
#Reg 1
Dati Reg 1
Registri
#Reg 2
Dati Reg 2
#Reg Dst
Dati da scrivere nel Reg Dst
ALUSrc
Zero
ALU
Indirizzo
Dati letti
Memoria Dati
Dati da scrivere
RegDst
rd
ForwardB
ForwardA
ID/EX.rs
ID/EX.rt
EX/MEM.rd
EX/MEM.RegWrite
MEM/WB.RegWrite
MEM/WB.rd
Forwarding
EX/MEM
MEM/WB
imm
16
ExtS
32
IF/ID
ID/EX
oc: [31-26]
rs: [25-21]
rt: [16-20]
rd: [15-11]
imm: [15-0]
fc: [5-0]

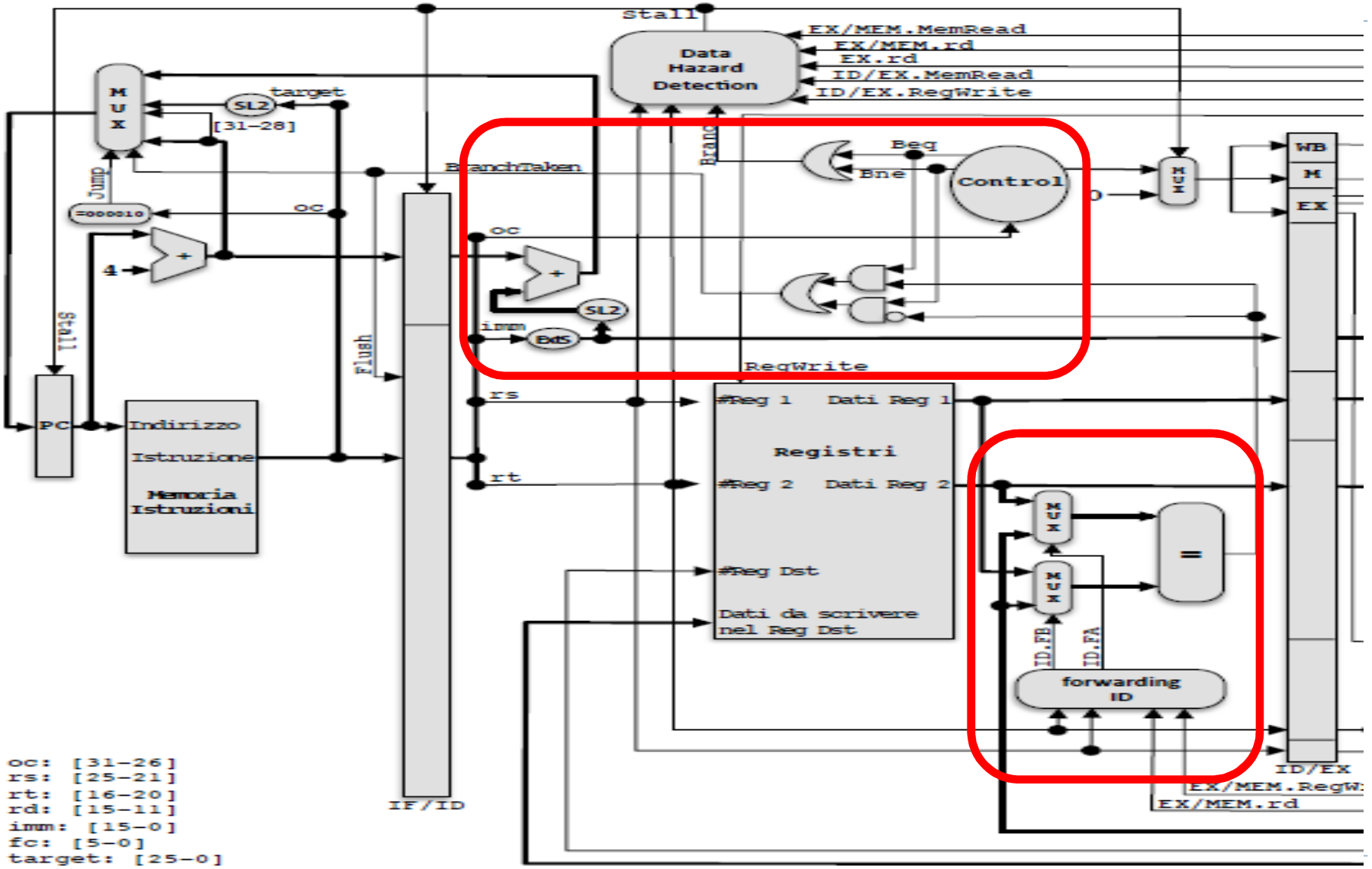
Anticipiamo beq alla fase ID



Anticipiamo beq alla fase ID



Anticipiamo beq alla fase ID

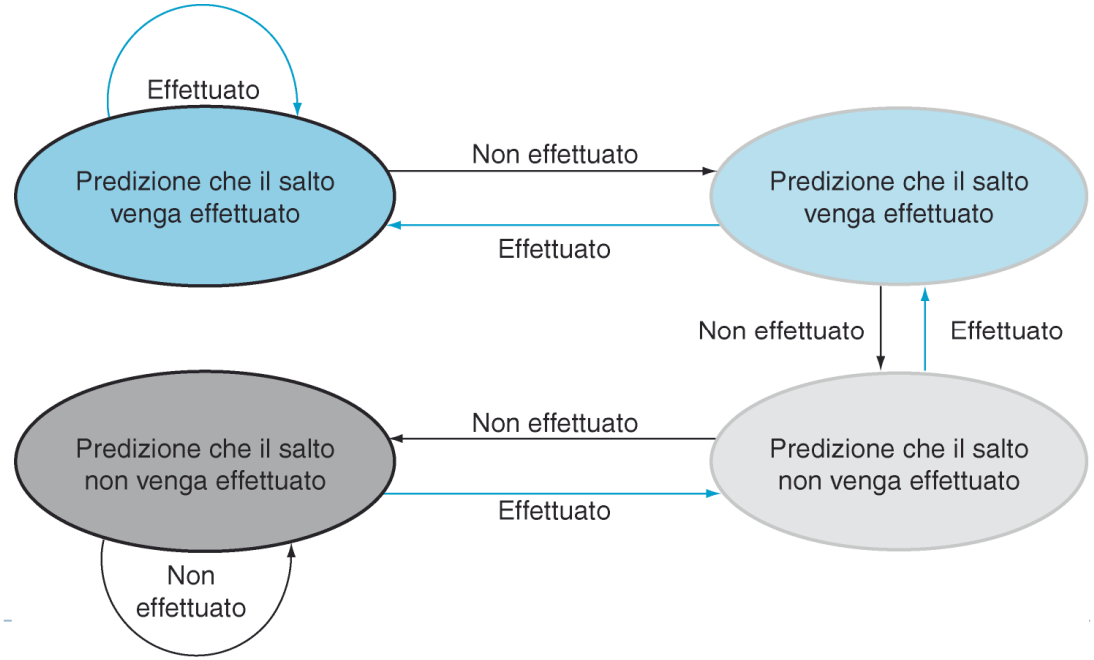


Impatto della beq

- Se la beq usa la ALU e il salto viene fatto dopo EXE:
  - 2 stalli dopo solo se il salto viene eseguito (non eliminabili)
  - 1 stallo prima se preceduta da lw (data-hazard eliminabile riordinando il codice)
- Se la beq è spostata nella fase ID:
  - 1 stallo dopo solo se il salto viene eseguito (eliminabile con il salto ritardato)
  - fino a 2 stalli prima se preceduta da R o lw (data-hazard eliminabile riordinando il codice)
- L'impatto dipende dal tipo di codice che usa la beq, soprattutto nei cicli:
  - se il test è all'inizio, seguito dal corpo del ciclo, la beq fa **un solo salto alla fine del ciclo**
  - se il test è alla fine, dopo il corpo, la beq fa **tanti salti** ed in un solo caso continuerà
- La CPU vista finora presume che il salto NON venga fatto (branch not taken)
- Se si potesse «prevedere» il salto si potrebbero caricare le istruzioni giuste ed evitare stalli

Predizione dei salti

- Ad ogni istruzione di salto possiamo associare alcuni bit che «contano» (rispetto alla storia dei salti già fatti) se è più probabile che il salto venga fatto oppure no
- - **con un solo bit** si può rappresentare l'informazione «l'ultima volta il salto è stato fatto»
- nel realizzare un ciclo la previsione sarà sbagliata 2 volte: **entrando e uscendo**
- - **con due bit** si può realizzare la macchina a stati finiti qua sotto, che ha bisogno di due sbagli consecutivi per cambiare previsione e **quindi sbaglia meno** nei cicli a forte prevalenza di uno specifico tipo di scelta (fa una sola predizione errata)



Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:

Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:
  - Caso 1: **una istr. precedente** che non abbia dipendenze (anche indirette) con la beq

Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:
  - Caso 1: **una istr. precedente** che non abbia dipendenze (anche indirette) con la beq
  - NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo

Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:
  - Caso 1: **una istr. precedente** che non abbia dipendenze (anche indirette) con la beq
  - NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo

- Se non ci sono istruzioni precedenti senza dipendenze:

Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:
  - Caso 1: **una istr. precedente** che non abbia dipendenze (anche indirette) con la beq
  - NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo

- Se non ci sono istruzioni precedenti senza dipendenze:
  - Caso 2: **l'istruzione alla destinazione del salto**

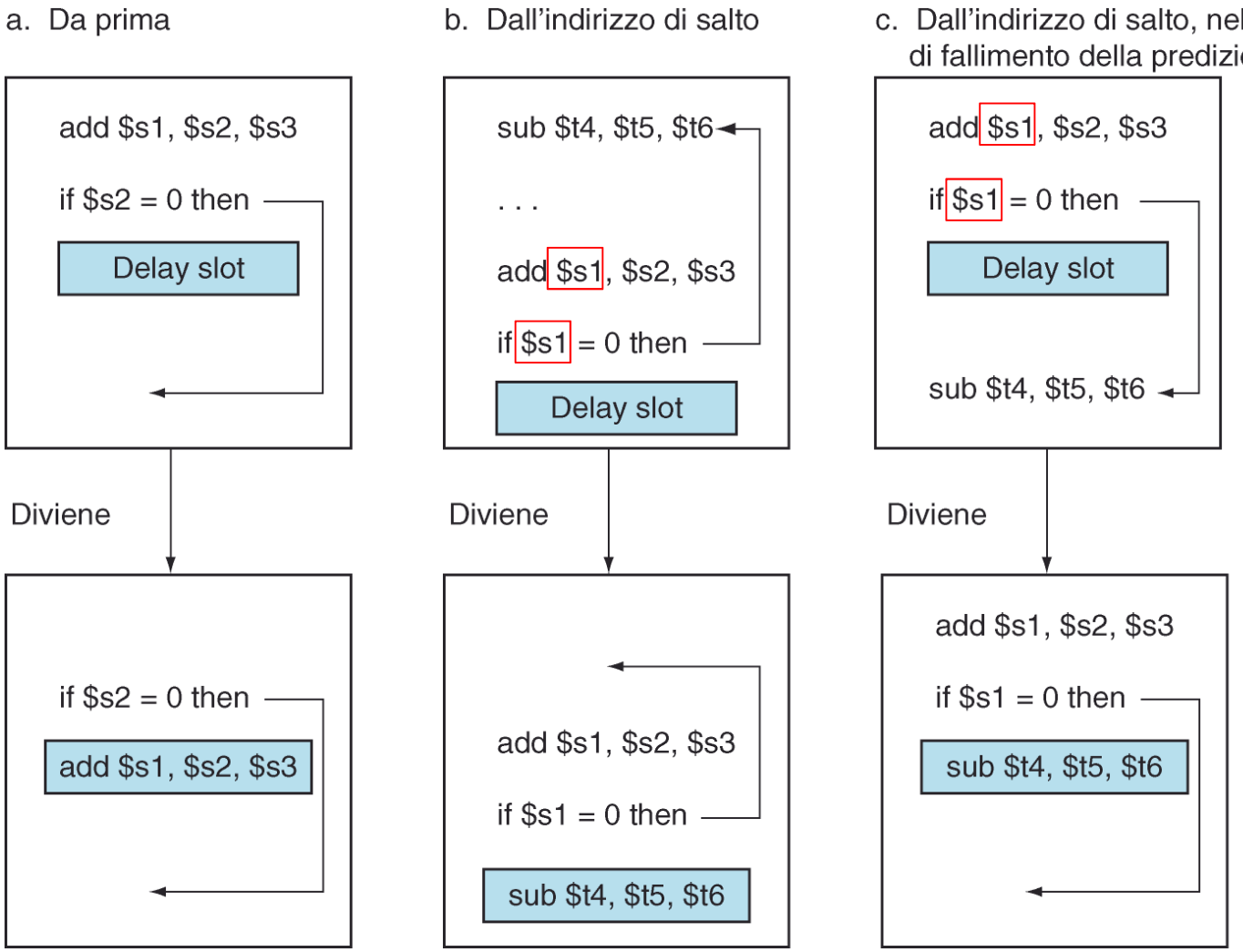
Ritardare il salto

- Oppure per recuperare il tempo perso dallo stallo si può **ritardare il salto**
  - ovvero **eseguire in ogni caso** l'istruzione che segue la beq (chiamata **delay slot**)
- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:
  - Caso 1: **una istr. precedente** che non abbia dipendenze (anche indirette) con la beq
  - NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo

- Se non ci sono istruzioni precedenti senza dipendenze:
  - Caso 2: **l'istruzione alla destinazione del salto**
  - NOTA: quando il salto NON viene fatto l'istruzione scelta (che viene sempre eseguita) non deve creare problemi:
    - ad esempio può calcolare un valore non più necessario nel codice seguente
    - l'importante è che non sia dannosa per l'esecuzione successiva

Riordinamento con salto ritardato



Soluzione: forwarding in MEM

- **Condizioni da verificare nella unità di forwarding MEM:**
  - EXE/MEM.MemWrite = 1 (è una sw)
  - AND MEM/WB.MemRead = 1 (la prec. è lw)
  - AND MEM/WB.RegWrite = 1
  - AND MEM/WB.RD = EXE/MEM.RT (lw $rt → sw $rt)
  - AND EXE/MEM != 0 (non è $zero)

- **Modifiche al datapath della fase MEM:**
  - aggiungere un **MUX** alla porta di scrittura in memoria, che permette di sostituire al valore di **EXE/MEM.DatoRT** il valore di **MEM/WB.DatoLetto**
  - **SOLO se le condizioni sono verificate**