Architettura degli Elaboratori

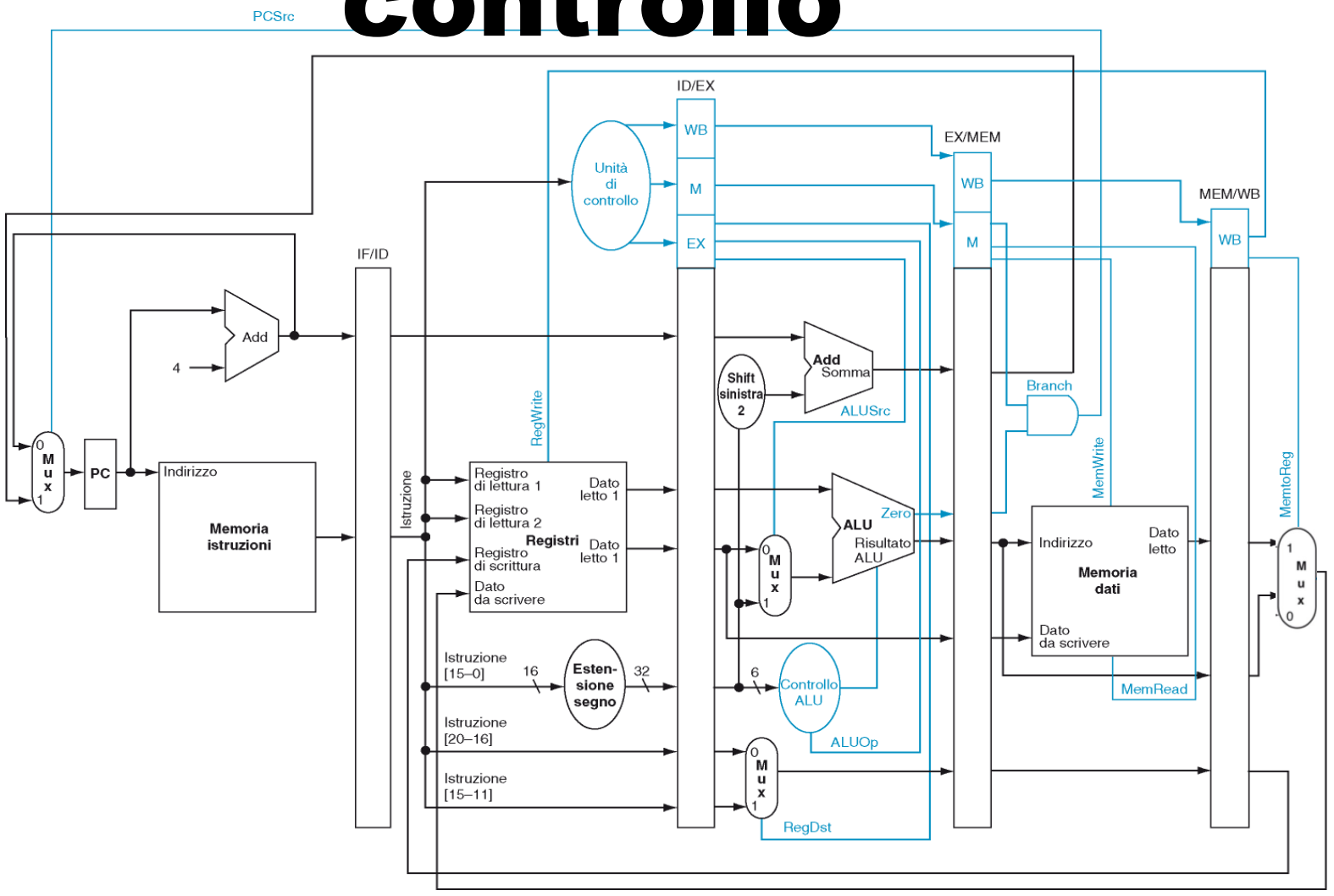
Pipelining:
Gestione degli Hazard

Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Abbiamo studiato
  - Una seconda implementazione per un sottoinsieme dell'IS del MIPS, basata su pipeline
  - In particolare, abbiamo visto come realizzare l'Unità di Elaborazione con pipeline e l'Unità di Controllo ad essa associata
- Obiettivo di oggi
  - Identificare le criticità (hazard) che sorgono in una pipeline quando non è possibile eseguire l'istruzione successiva nel ciclo di clock successivo
  - Descrivere brevemente alcune soluzioni hardware per risolvere tali criticità

Unità di elaborazione con pipeline e controllo



Hazard

- In una pipeline si possono verificare situazioni in cui l'istruzione successiva non può essere eseguita nel ciclo di clock successivo
- Tali situazioni sono dette hazard (o criticità, o conflitti) e possono essere di tre tipi
  - Hazard strutturali
    - Quando stadi diversi della pipeline necessitano delle stesse risorse hardware nello stesso ciclo di clock
  - Hazard sui dati
    - Quando un'istruzione dipende dal risultato di un'istruzione precedente che non è stata ancora completata
  - Hazard sul controllo
    - Quando è necessario prendere una decisione sulla prossima istruzione da eseguire prima che la condizione sia valutata

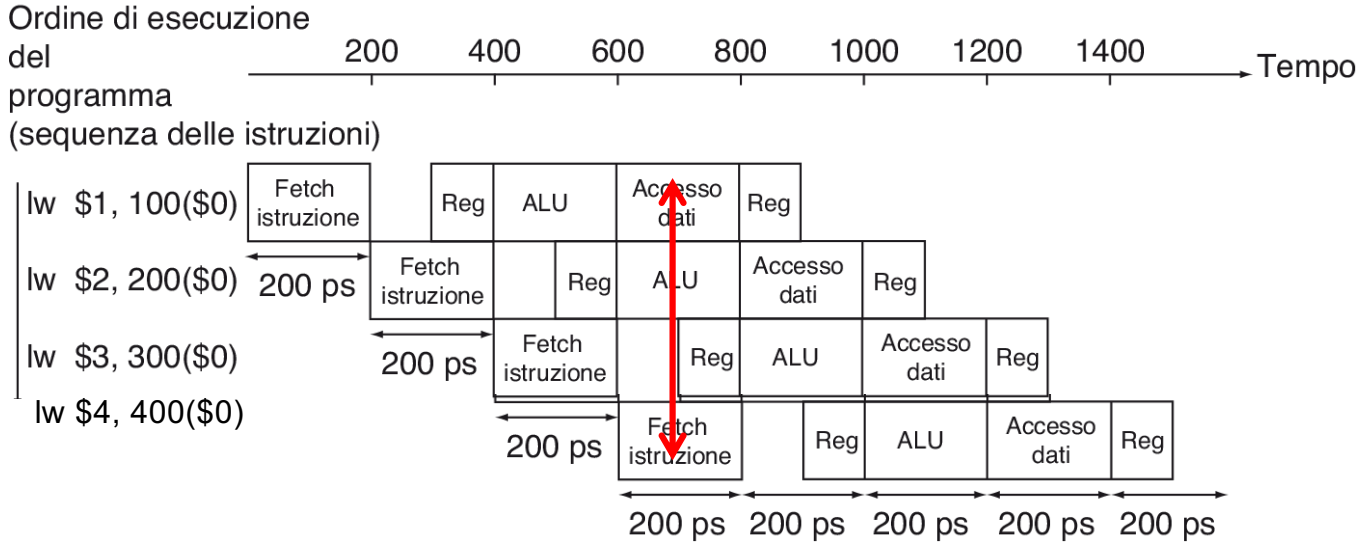
Hazard strutturali

- Si verificano quando stadi diversi della pipeline necessitano delle stesse risorse hardware nello stesso ciclo di clock

- Nel caso del MIPS
  - Se memoria istruzioni e memoria dati non fossero separate
  - Se il register file fosse usato nello stesso ciclo di clock per un accesso in lettura da un'istruzione e uno in scrittura da un'altra istruzione senza opportuni accorgimenti
  - Queste due problematiche hanno portato ad opportune scelte progettuali nella realizzazione del MIPS

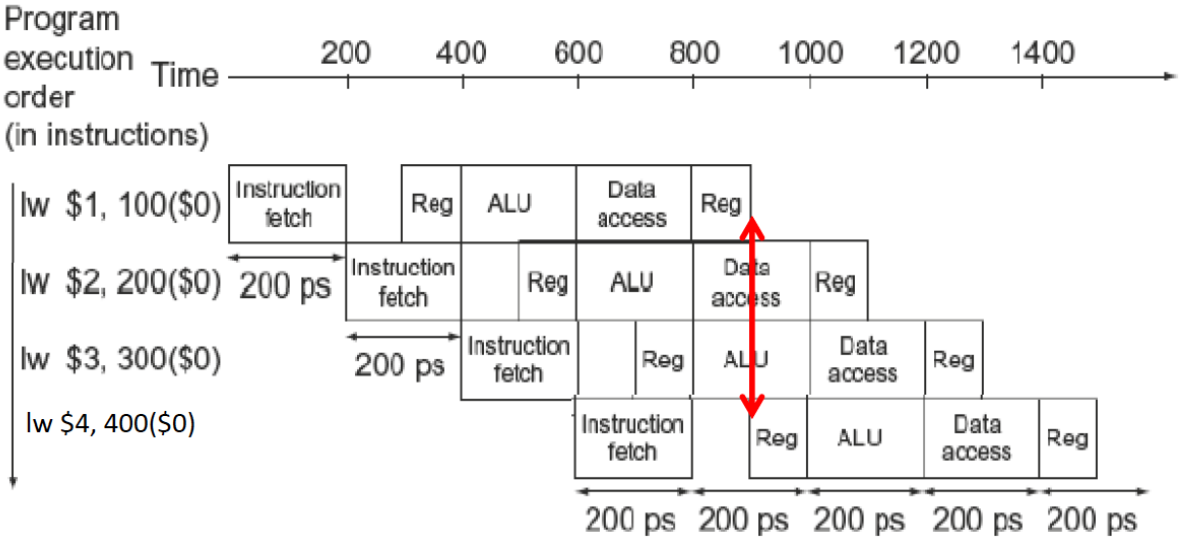
Hazard strutturali

- Supponiamo che memoria istruzioni e memoria dati non siano separate
- Supponendo di avere 4 istruzioni lw in sequenza
  - Si avrebbe un hazard strutturale tra la prima e la quarta istruzione
  - La prima istruzione deve accedere ai dati in memoria mentre la quarta istruzione deve essere prelevata dalla stessa memoria nello stesso ciclo di clock



Hazard strutturali

- Supponiamo che il register file sia usato nello stesso ciclo di clock per un accesso in lettura da un'istruzione e uno in scrittura da un'altra istruzione
- Per evitare l'hazard strutturale
  - La scrittura del register file avviene nella prima metà del ciclo di clock
  - La lettura del register file avviene nella seconda metà del ciclo di clock



Hazard sui dati

- Si verificano quando un'istruzione dipende dal risultato di un'istruzione precedente che non è stata ancora completata
- Esempio

  add $2, $1, $3

  sub $12, $2, $5

  - Uno degli operandi sorgente di sub ($2) è prodotto da add, che è ancora nella pipeline
- Esempio

  lw $2, 20($1)

  sub $4, $2, $5

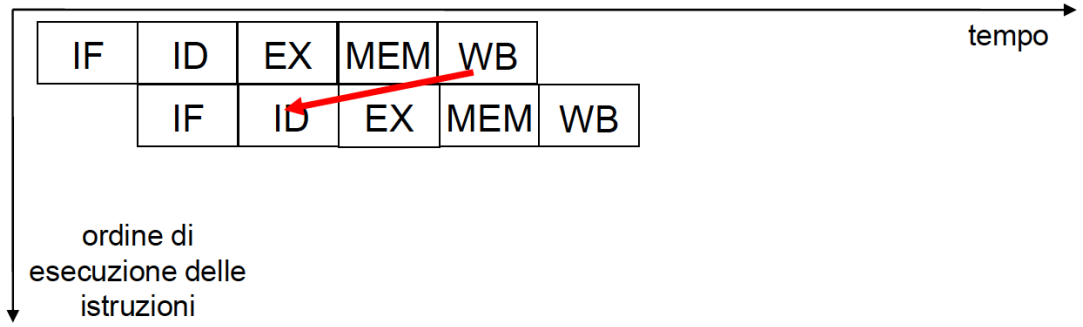
  - Uno degli operandi sorgente di sub ($2) è prodotto da lw, che è ancora nella pipeline

Hazard sui dati

- Esempio

  add $2, $1, $3

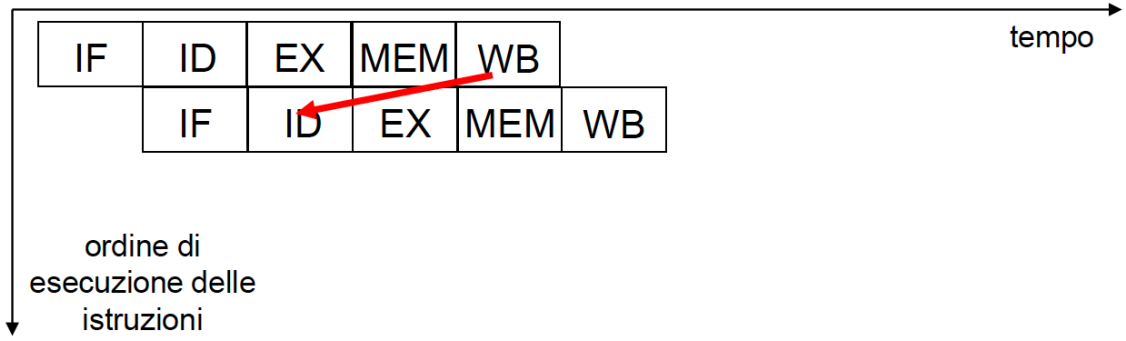
  sub $12, $2, $5



- Esempio

  lw $2, 20($1)

  sub $4, $2, $5



Hazard sui dati

- Esistono due tipi di soluzioni per gestire gli hazard sui dati
  - Soluzioni di tipo hardware
    - Propagazione o scavalcamento (forwarding o bypassing)
    - Inserimento di stalli (o bolle) nella pipeline
  - Soluzioni di tipo software
    - Inserimento di istruzioni "nop" (no operation)
    - Riordino delle istruzioni

Propagazione

- Idea: propagare i dati in avanti, non appena sono disponibili, verso le unità che li richiedono
- Se ad esempio si devono eseguire le due istruzioni

```
add $2, $1, $3
sub $12, $2, $5
```

non appena la ALU calcola la somma tra $1 e $3, il risultato viene subito messo a disposizione della sub

- Ciò può essere fatto mediante l'aggiunta di un circuito di propagazione che fornisce, in un punto dell'esecuzione, il dato mancante (preso da una risorsa interna)

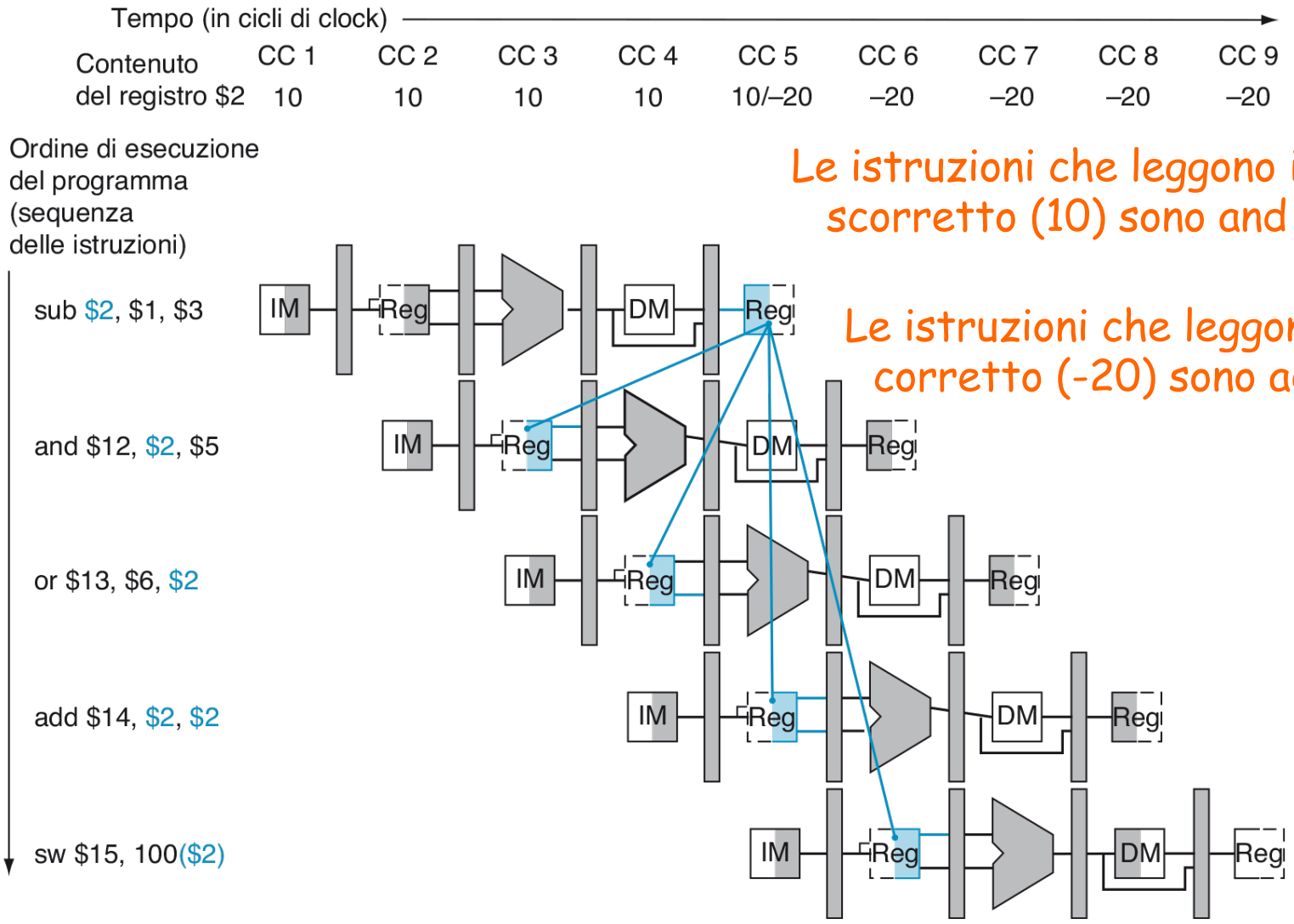
Propagazione

- Consideriamo la seguente sequenza di istruzioni

```
sub $2, $1, $3 #sub scrive nel registro $2
and $12, $2, $5 #il primo operando ($2) dipende da sub
or $13, $6, $2 #il secondo operando ($2) dipende da sub
add $14, $2, $2 #entrambi gli operandi ($2 e $2) dipendono da sub
sw $15, 100($2) #il registro base ($2) dipende da sub
```

- Le ultime quattro istruzioni dipendono tutte dal risultato della prima, che scrive nel registro $2
  - Supponiamo che il registro $2 contenga il valore 10 prima della sub e il valore -20 dopo la sub
  - Vorremmo che le istruzioni successive utilizzassero il dato corretto (-20)

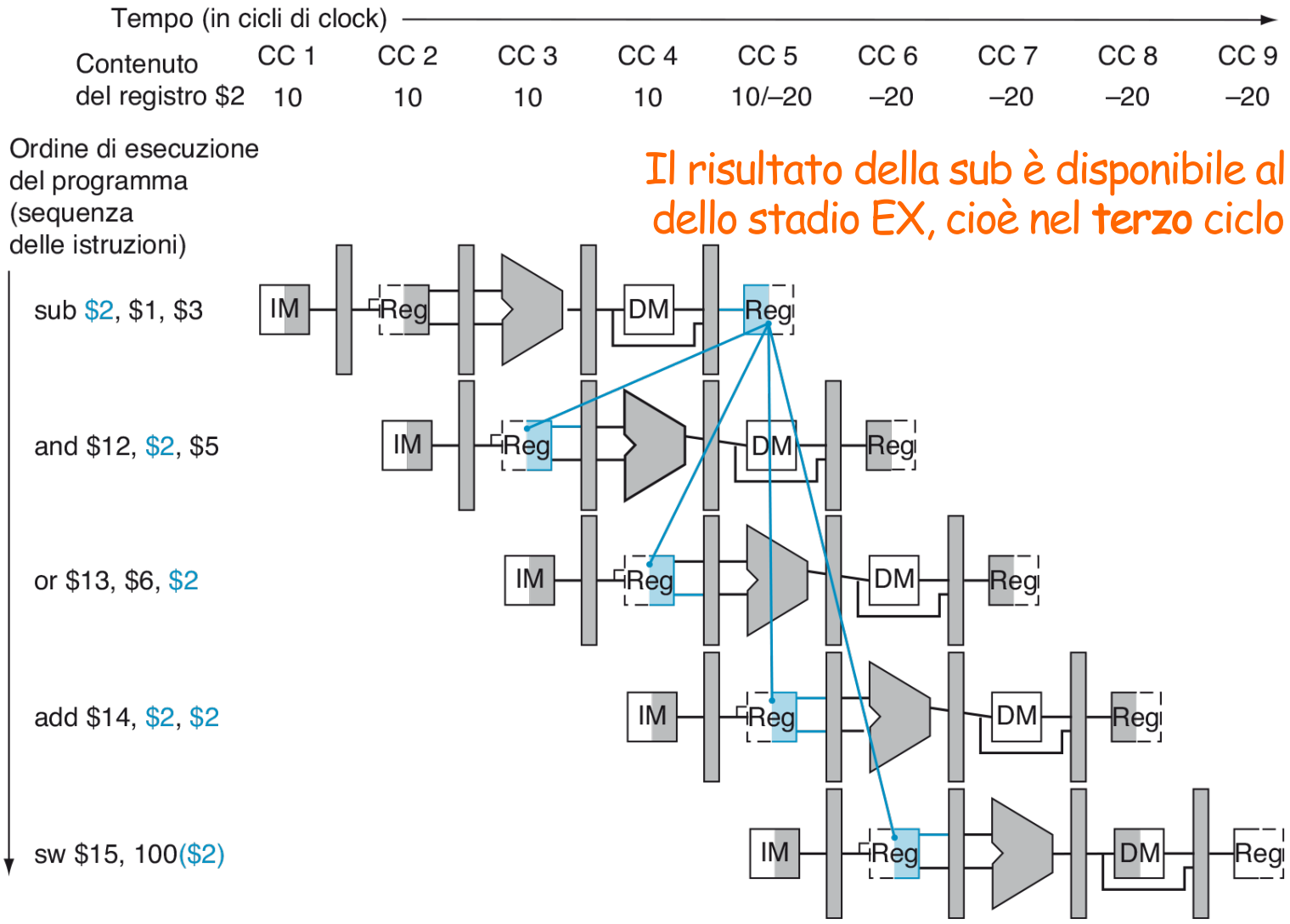
Propagazione



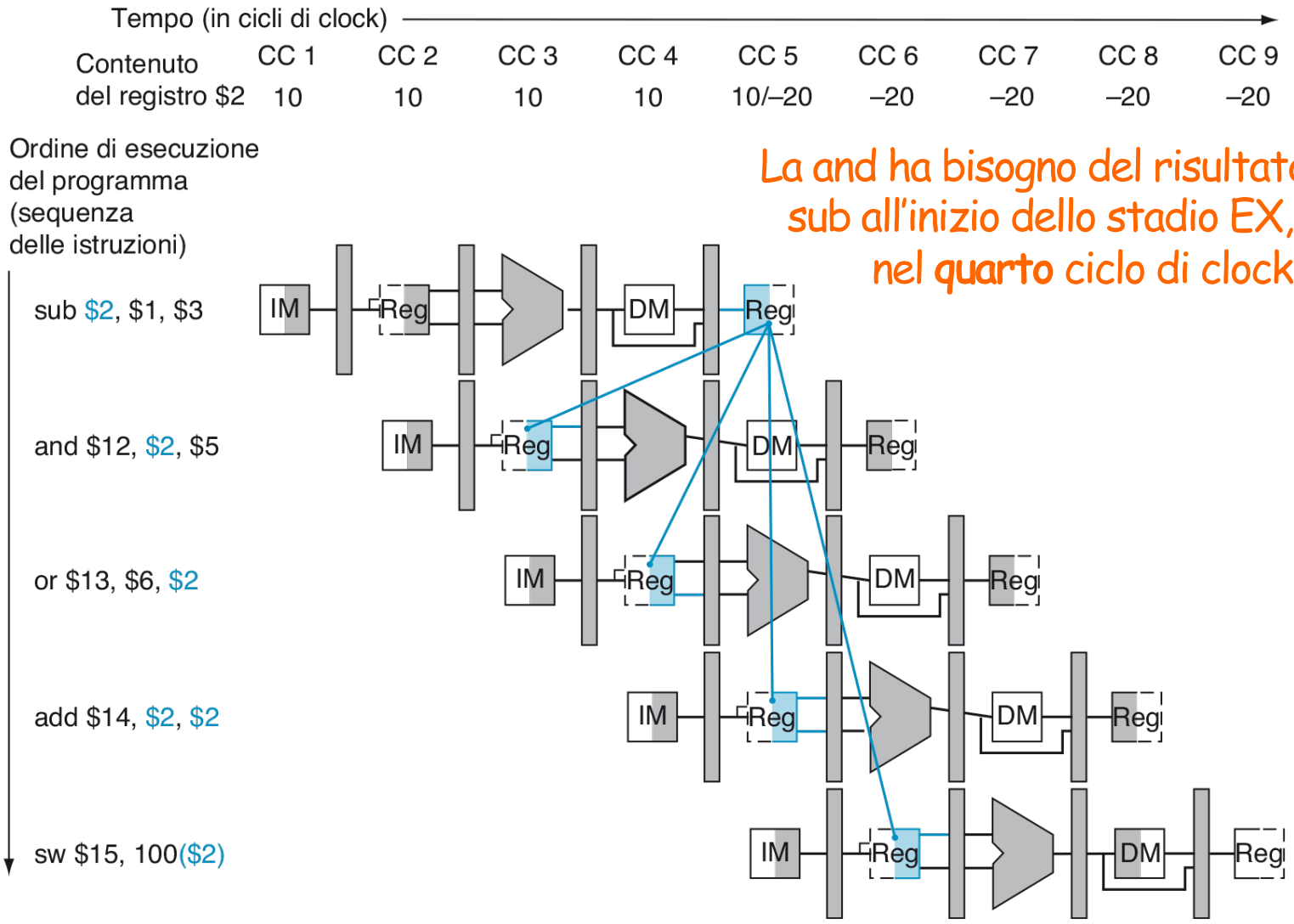
Le istruzioni che leggono il dato scorretto (10) sono and e or

Le istruzioni che leggono il dato corretto (-20) sono add e sw

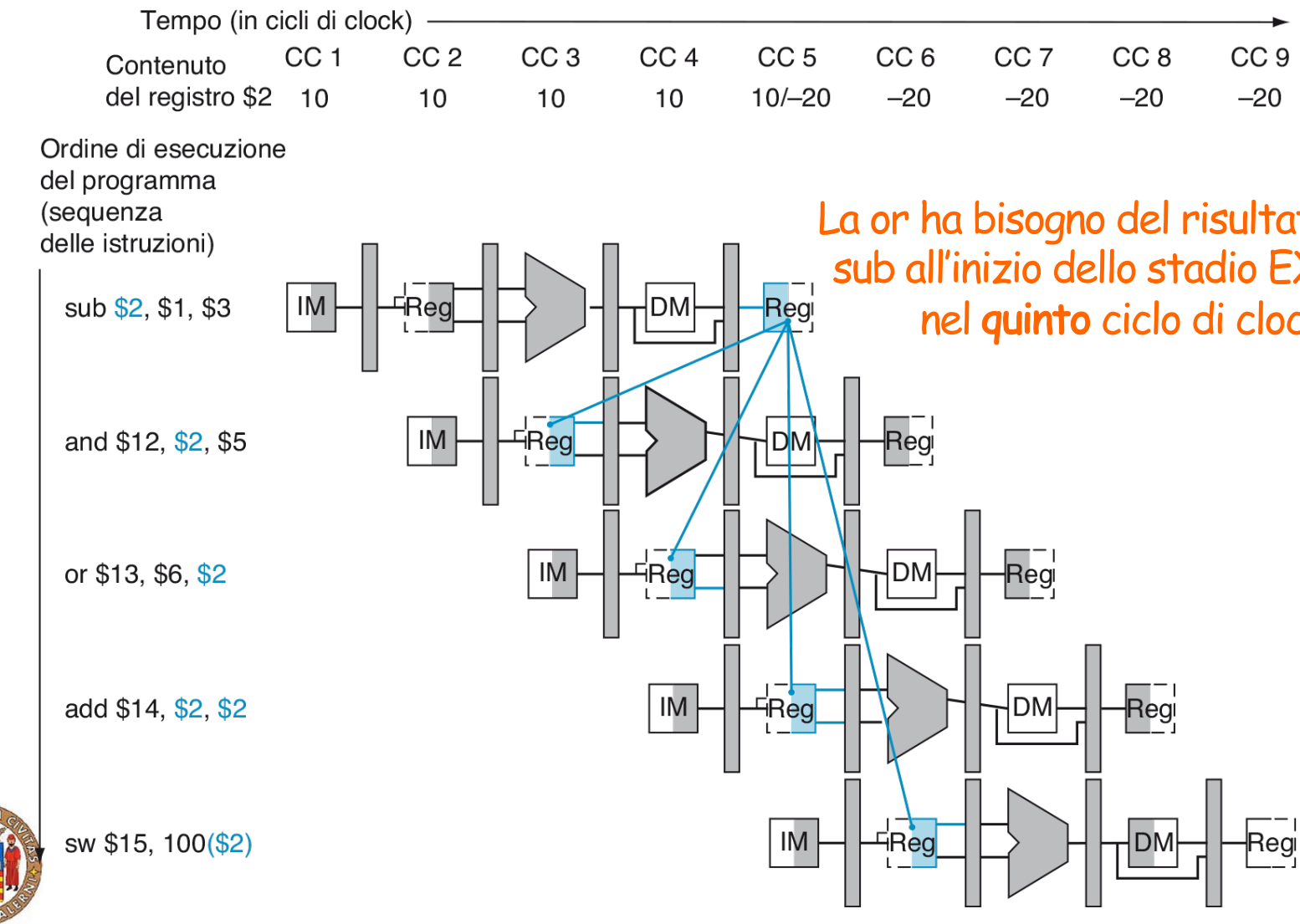
Propagazione



Propagazione



Propagazione



Propagazione

- Il frammento di codice può essere eseguito propagando il dato non appena esso è disponibile
  - Non c'è necessità di attendere che il dato venga scritto nel register file
  - Basta leggere il dato dal registro EX/MEM e propagarlo in avanti, portandolo alle unità che ne hanno bisogno
  - Nell'esempio, la and ha bisogno del dato all'ingresso della ALU
- C'è necessità di rilevare gli hazard, prima di propagare i dati

Rilevare gli hazard

- Le condizioni che generano hazard sui dati sono di 4 tipi diversi
  1. EX/MEM.RegistroRd = ID/EX.RegistroRs
  2. EX/MEM.RegistroRd = ID/EX.RegistroRt
  3. MEM/WB.RegistroRd = ID/EX.RegistroRs
  4. MEM/WB.RegistroRd = ID/EX.RegistroRt

- Quando si verificano i primi due tipi di hazard?
  - Quando uno dei due operandi (rs, rt) di una istruzione che si trova nello stadio ID e deve entrare nello stato EX (ed è quindi presente nel registro ID/EX) dipende dal risultato (rd) di una istruzione precedente che viene prodotto nello stadio EX (ed è quindi presente nel registro EX/MEM)

Rilevare gli hazard

- Le condizioni che generano hazard sui dati sono di 4 tipi diversi
  1. EX/MEM.RegistroRd = ID/EX.RegistroRs
  2. EX/MEM.RegistroRd = ID/EX.RegistroRt
  3. MEM/WB.RegistroRd = ID/EX.RegistroRs
  4. MEM/WB.RegistroRd = ID/EX.RegistroRt

- Quando si verificano gli ultimi due tipi di hazard?
  - Quando uno dei due operandi (rs, rt) di una istruzione che si trova nello stadio ID e deve entrare nello stato EX (ed è quindi presente nel registro ID/EX) dipende dal risultato (rd) di una istruzione precedente che viene prodotto nello stadio MEM (ed è quindi presente nel registro MEM/WB)

Rilevare gli hazard

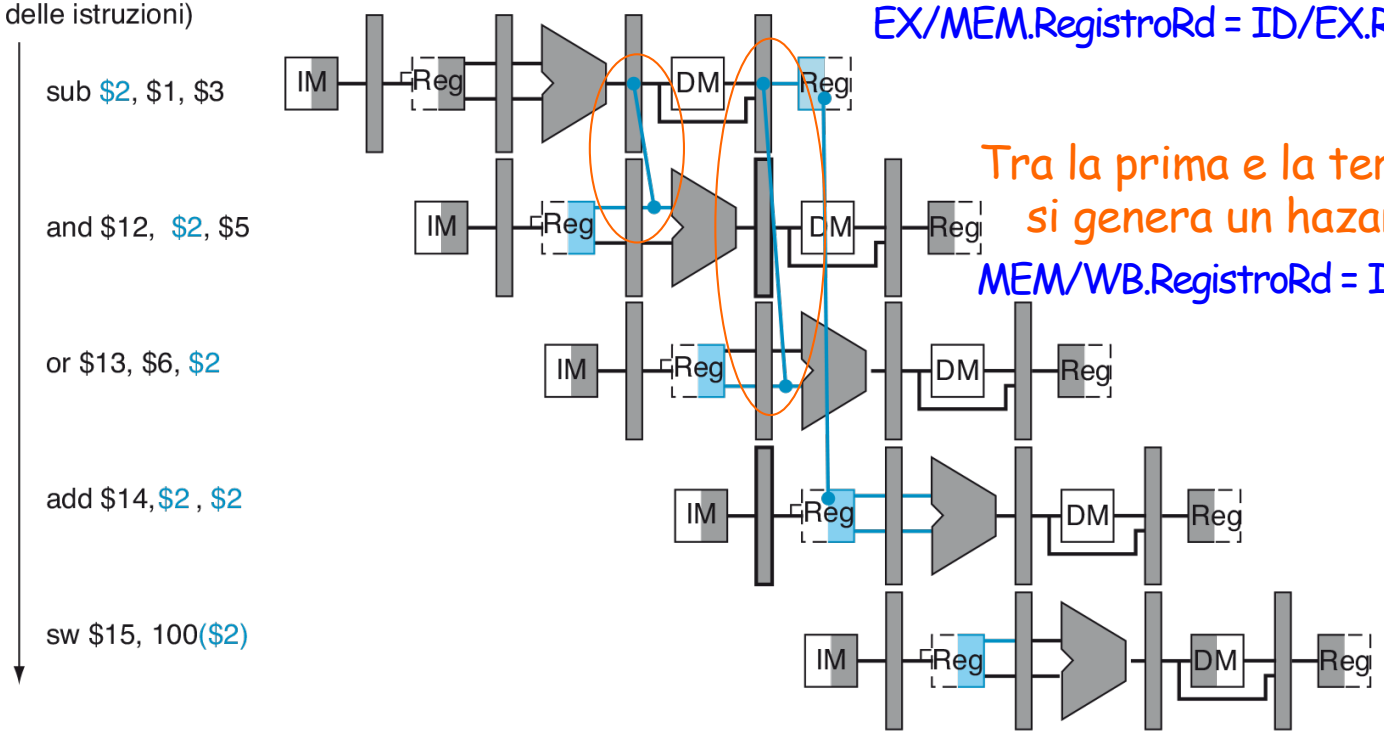
- Dato che non tutte le istruzioni scrivono il register file, come possiamo evitare di propagare dati quando non è necessario?
- Possiamo controllare se il segnale di controllo RegWrite è asserito (scrittura register file)
- Se RegWrite=0 non c'è necessità di propagazione

Propagazione

Tempo (in cicli di clock) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Contenuto del registro $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Contenuto del registro EX/MEM:	X	X	X	–20	X	X	X	X	X
Contenuto del registro MEM/WB:	X	X	X	X	–20	X	X	X	X

Ordine di esecuzione del programma (sequenza delle istruzioni)



Tra le prime due istruzioni si genera un hazard di tipo 1

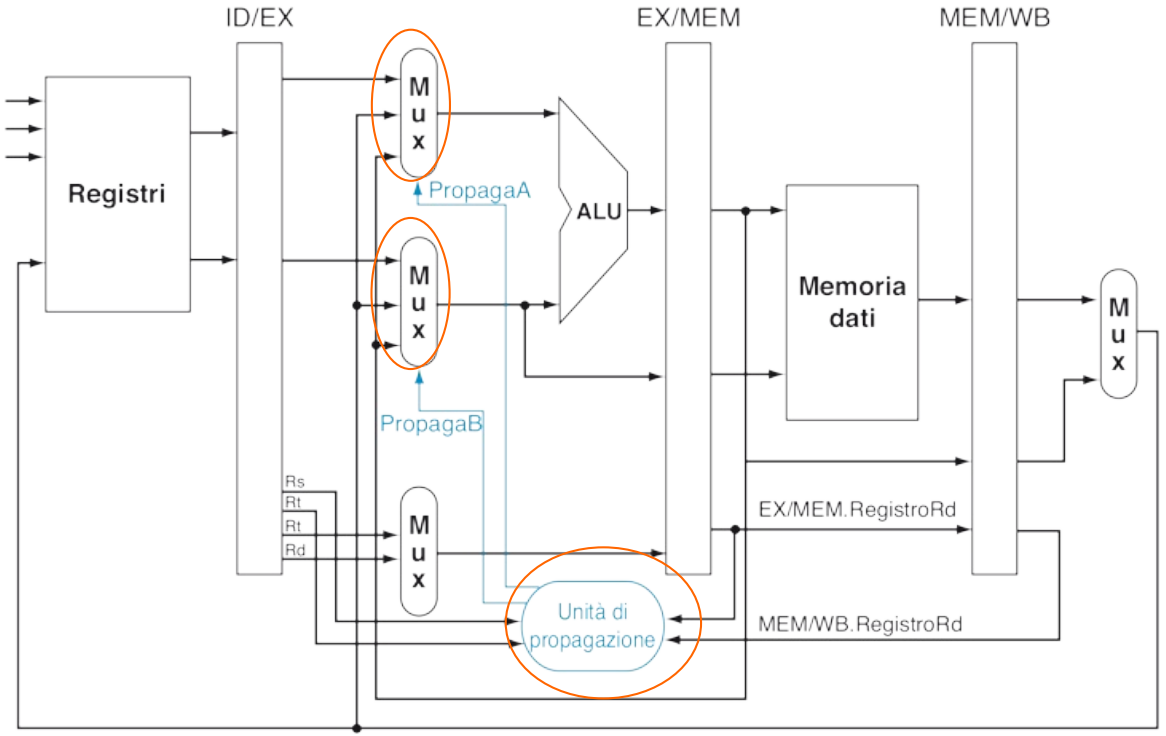
EX/MEM.RegistroRd = ID/EX.RegistroRs

Tra la prima e la terza istruzione si genera un hazard di tipo 4

MEM/WB.RegistroRd = ID/EX.RegistroRt

Propagazione

- Per consentire agli input dell'ALU di provenire da uno qualsiasi dei registri di pipeline ID/EX, EX/MEM, MEM/WB, vengono aggiunti dei multiplexer
  - I segnali di controllo per tali multiplexer (PropagaA e PropagaB) sono generati da una Unità di Propagazione



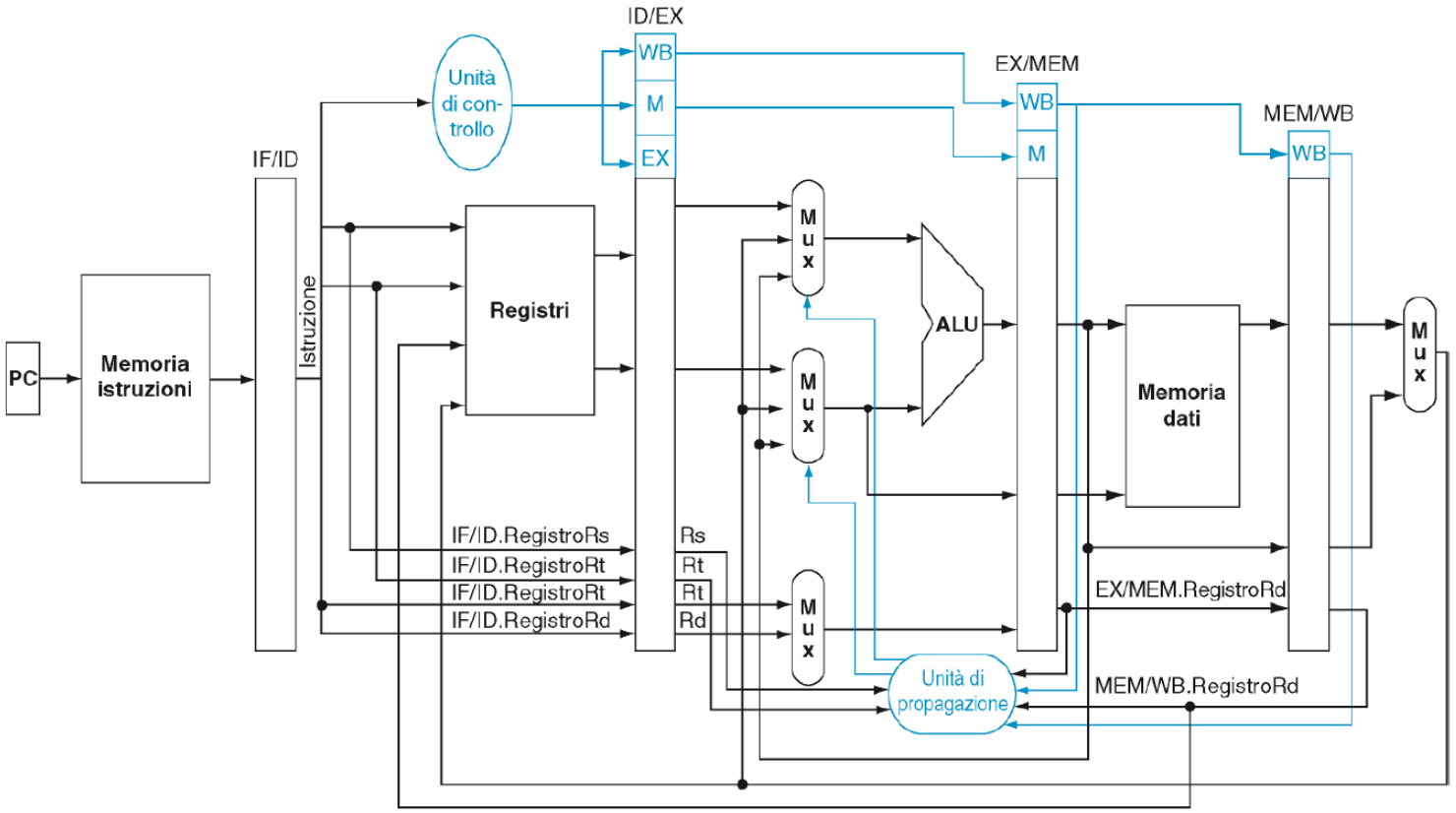
Propagazione

Segnali di controllo per i multiplexer

Controllo multiplexer	Sorgente	Spiegazione
PropagaA = 00	ID/EX	Il primo operando della ALU proviene dal register file.
PropagaA = 10	EX/MEM	Il primo operando della ALU viene propagato dal risultato della ALU nel ciclo di clock precedente.
PropagaA = 01	MEM/WB	Il primo operando della ALU viene propagato dalla memoria dati o da un precedente risultato della ALU.
PropagaB = 00	ID/EX	Il secondo operando della ALU proviene dal register file.
PropagaB = 10	EX/MEM	Il secondo operando della ALU viene propagato dal risultato della ALU nel ciclo di clock precedente.
PropagaB = 01	MEM/WB	Il secondo operando della ALU è propagato dalla memoria dati o da un precedente risultato della ALU.

Propagazione

Unità di elaborazione e di controllo dopo le modifiche apportate per risolvere gli hazard tramite la propagazione



Propagazione e stallo

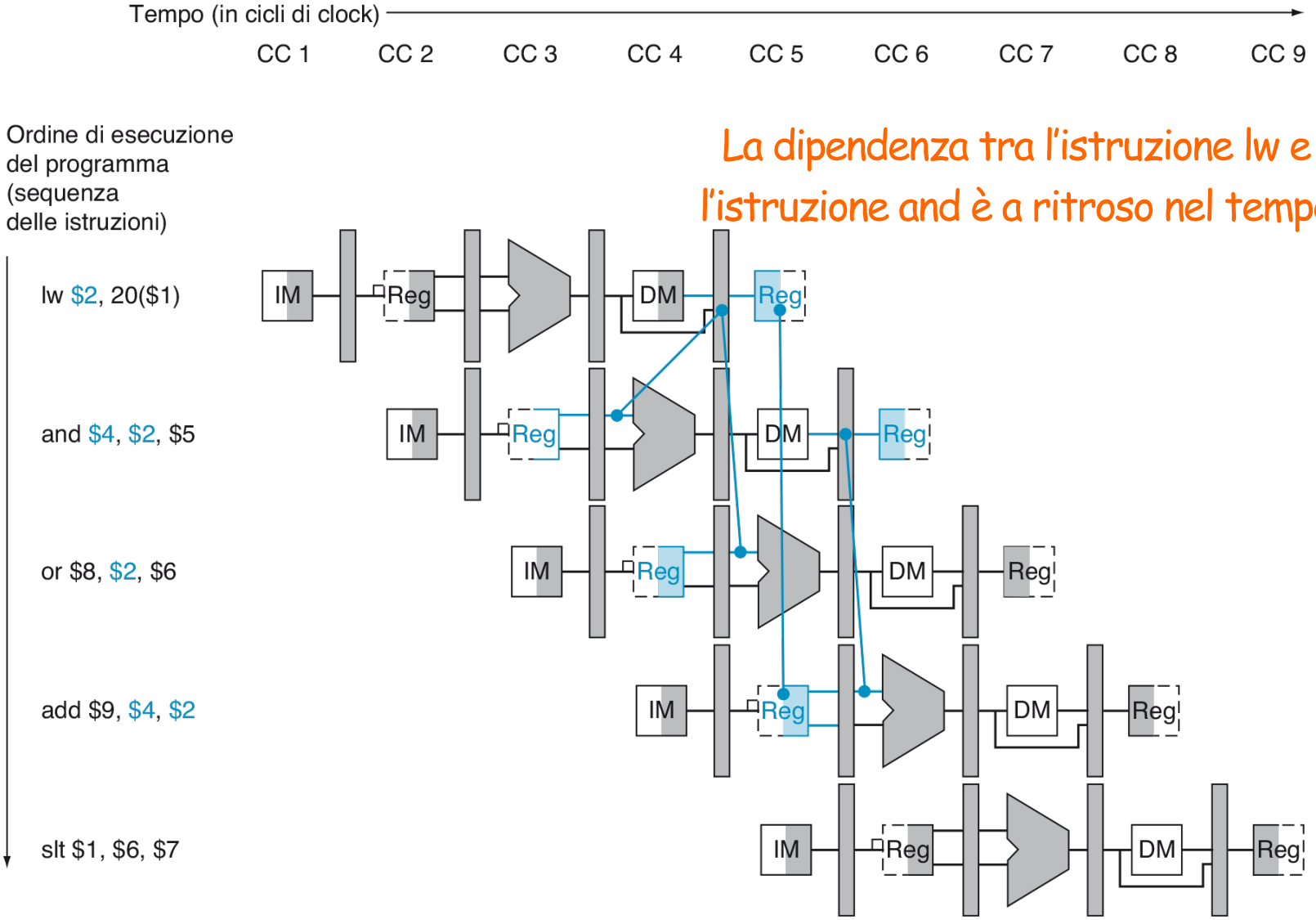
- Consideriamo la seguente sequenza di istruzioni

```
lw $2, 20($1) #viene caricata nel registro $2 una word della memoria
and $4, $2, $5 #il primo operando ($2) dipende da lw
or $8, $2, $6 #il primo operando ($2) dipende da lw
add $9, $4, $2 #il primo operando ($4) dipende da and,
                #il secondo operando ($2) dipende da lw
slt $1, $6, $7 #il registro $1$ viene posto a 1 se il contenuto
                #del primo operando è minore del contenuto
                #del secondo operando
```

- La seconda istruzione dipende dalla prima

Il dato caricato nel registro $2 non è ancora disponibile quando viene richiesto dalla and

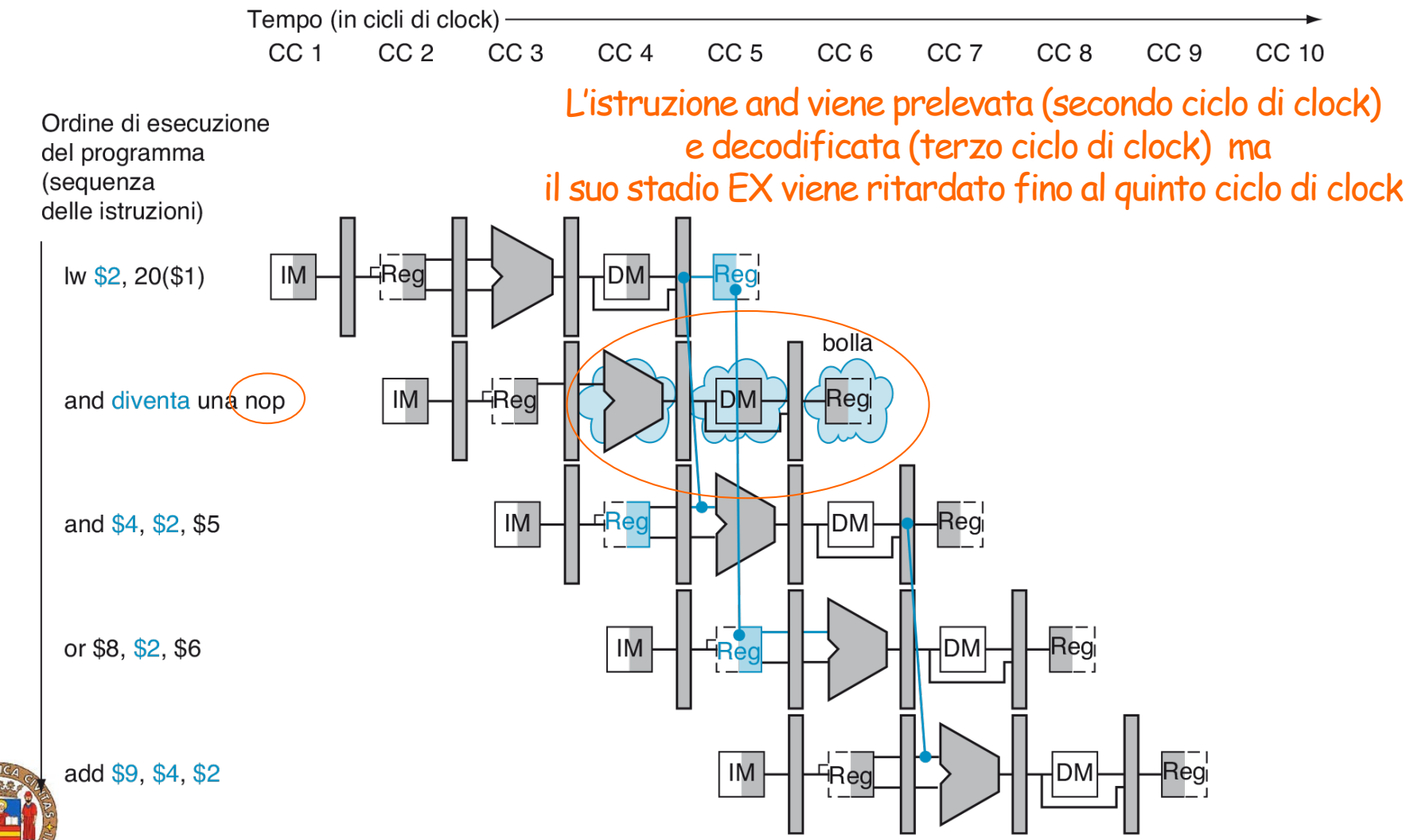
Propagazione e stallo



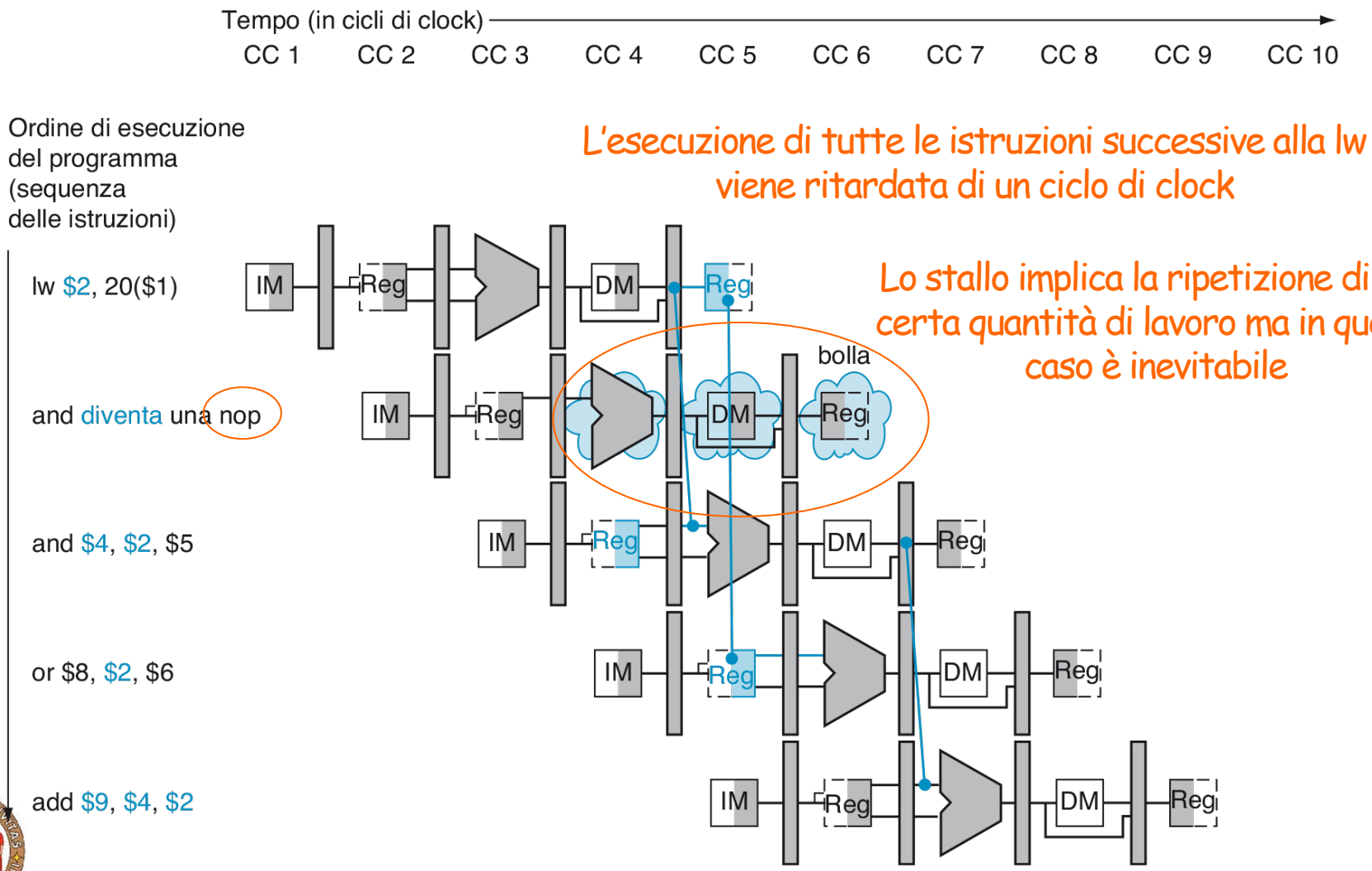
Propagazione e stallo

- La sola propagazione è insufficiente a risolvere questo tipo di criticità
- E' quindi necessaria, oltre all'Unità di Propagazione, anche una Hazard Detection Unit
  - Compito di tale unità è quello di controllare se si è in presenza del tipo di hazard descritto nelle slide precedenti
  - In tal caso, la pipeline viene messa in pausa (stallo)
  - Tecnicamente, la messa in stallo si ottiene mediante l'aggiunta di una bolla (equivalente hardware di una istruzione che non fa nulla: nop)

Propagazione e stallo



Propagazione e stallo



L'esecuzione di tutte le istruzioni successive alla lw viene ritardata di un ciclo di clock

Lo stallo implica la ripetizione di una certa quantità di lavoro ma in questo caso è inevitabile

Rilevare gli hazard

- Per rilevare il tipo di hazard appena descritto, bisogna innanzitutto verificare che l'istruzione nello stadio EX sia una lw
  - Basta verificare che il segnale di controllo ID/EX.MemRead sia asserito

- Se ciò accade, bisogna controllare se il registro di scrittura della lw coincide con uno dei registri sorgente nello stadio ID, cioè se si presenta una delle due condizioni seguenti
  - ID/EX.RegistroRt = IF/ID.RegistroRs
  - ID/EX.RegistroRt = IF/ID.RegistroRt

Rilevare gli hazard

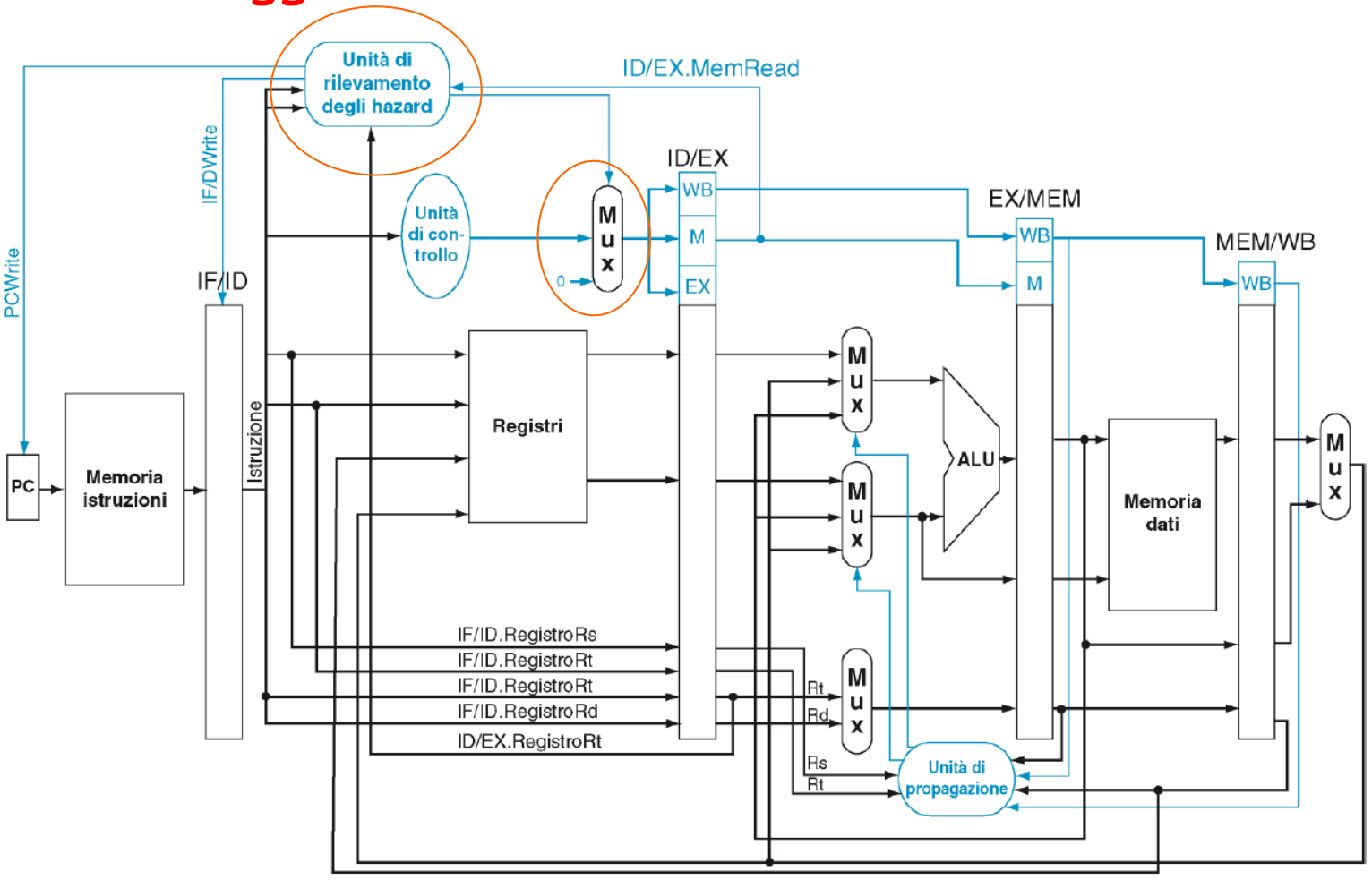
- In caso di verifica dell'hazard, la Hazard Detection Unit procede all'inserimento della bolla

- Ciò viene fatto impostando a zero i campi EX, MEM e WB del registro di pipeline ID/EX, mediante un multiplexer
  - Tali campi contengono i 9 segnali di controllo
  - Ad ogni ciclo i clock i segnali vengono trasportati in avanti, producendo l'effetto desiderato
    (nè la memoria dati, nè i registri vengono scritti)

- La Hazard Detection Unit controlla anche la scrittura del PC e del registro IF/ID
  - Necessario non incrementare il PC e non prelevare una istruzione successiva per non perdere l'istruzione corrente

Propagazione e stallo

Unità di elaborazione con pipeline con l'aggiunta della Hazard Detection Unit



Hazard sul controllo

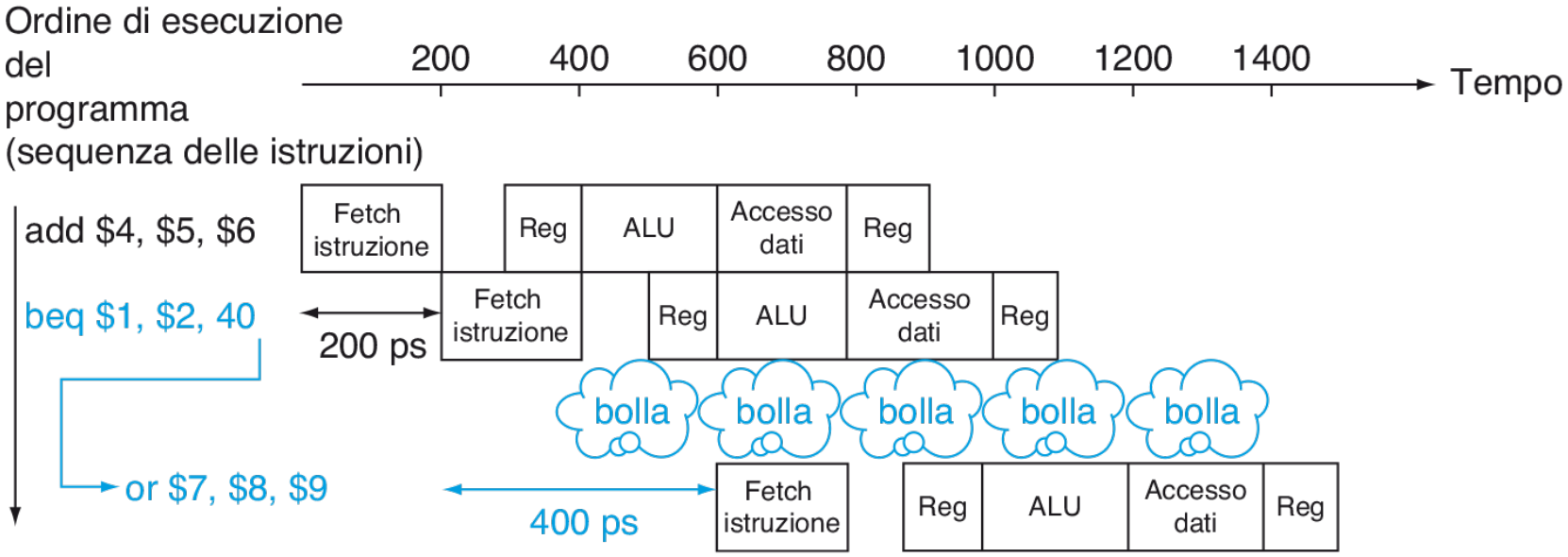
- Si verificano quando si tenta di prendere una decisione sulla prossima istruzione da eseguire prima che la condizione sia valutata
- Ad esempio, se si sta eseguendo una beq, come si fa a sapere in anticipo quale sarà la prossima istruzione da eseguire?
  - La lettura dalla memoria istruzioni dell'istruzione che segue la beq viene effettuata nel ciclo di clock successivo a quello in cui la beq è stata prelevata
  - La pipeline non ha alcun modo di sapere quale sarà l'istruzione successiva perché ha appena ricevuto la beq dalla memoria

Hazard sul controllo

- Le soluzioni possibili per gestire gli hazard sul controllo sono le seguenti
  - Inserimento di bolle
  - Anticipazione del confronto allo stadio ID
  - Utilizzo di tecniche di predizione del salto
    - Tecniche di predizione statica
    - Tecniche di predizione dinamica (non le descriveremo)

Hazard sul controllo

- L'inserimento di bolle consente di mettere in stallo la pipeline subito dopo il caricamento della beq
  - In tal modo si può attendere fino a quando non viene calcolato il risultato del confronto (avviene nel quarto stadio)



Hazard sul controllo

- L'anticipazione del confronto dallo stadio MEM allo stadio ID richiede di anticipare due azioni
  - Calcolo dell'indirizzo del salto
    - Facile da fare perché le informazioni necessarie sono contenute nel registro di pipeline IF/ID
    - Basta spostare il sommatore che calcola l'indirizzo del salto dallo stadio EX allo stadio ID
  - Valutazione del confronto sulla base del quale il salto verrà effettuato o meno
    - E' necessario hardware aggiuntivo per effettuare il confronto ed eventualmente modificare il PC
    - Poiché i dati su cui fare il confronto sono richiesti già nello stadio ID ma possono essere prodotti più tardi nella pipeline, può verificarsi un hazard sui dati

Hazard sul controllo

- La tecnica di predizione statica del salto è frequentemente usata per risolvere hazard sul controllo
  - Consiste nel predire che il salto non sarà eseguito continuando ad eseguire le istruzioni secondo il flusso normale
  - Se invece il salto doveva essere eseguito, bisogna eliminare le tre istruzioni presenti negli stadi IF, ID, EXE della pipeline
    - Ciò può essere fatto tramite nuovi segnali di controllo che azzerano i registri

Riepilogo e riferimenti

- Abbiamo analizzato le criticità (hazard) che si possono verificare in un'architettura basata su pipeline
  - I tipi di hazard nelle pipeline: [PH] par. 4.5 (seconda parte)
  - Hazard sui dati: propagazione e stallo: [PH] par. 4.7
  - Hazard sul controllo: [PH] par 4.8 (cenni)