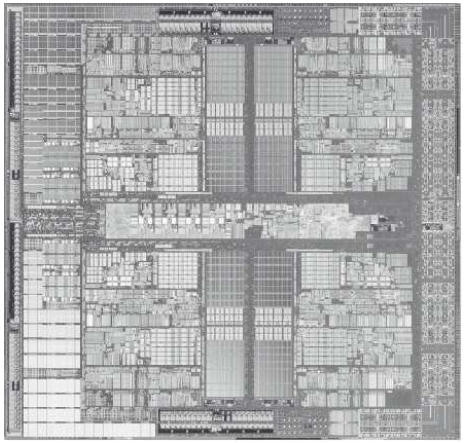
Università degli Studi di Cassino e del Lazio Meridionale



Corso di Calcolatori Elettronici

Pipeline

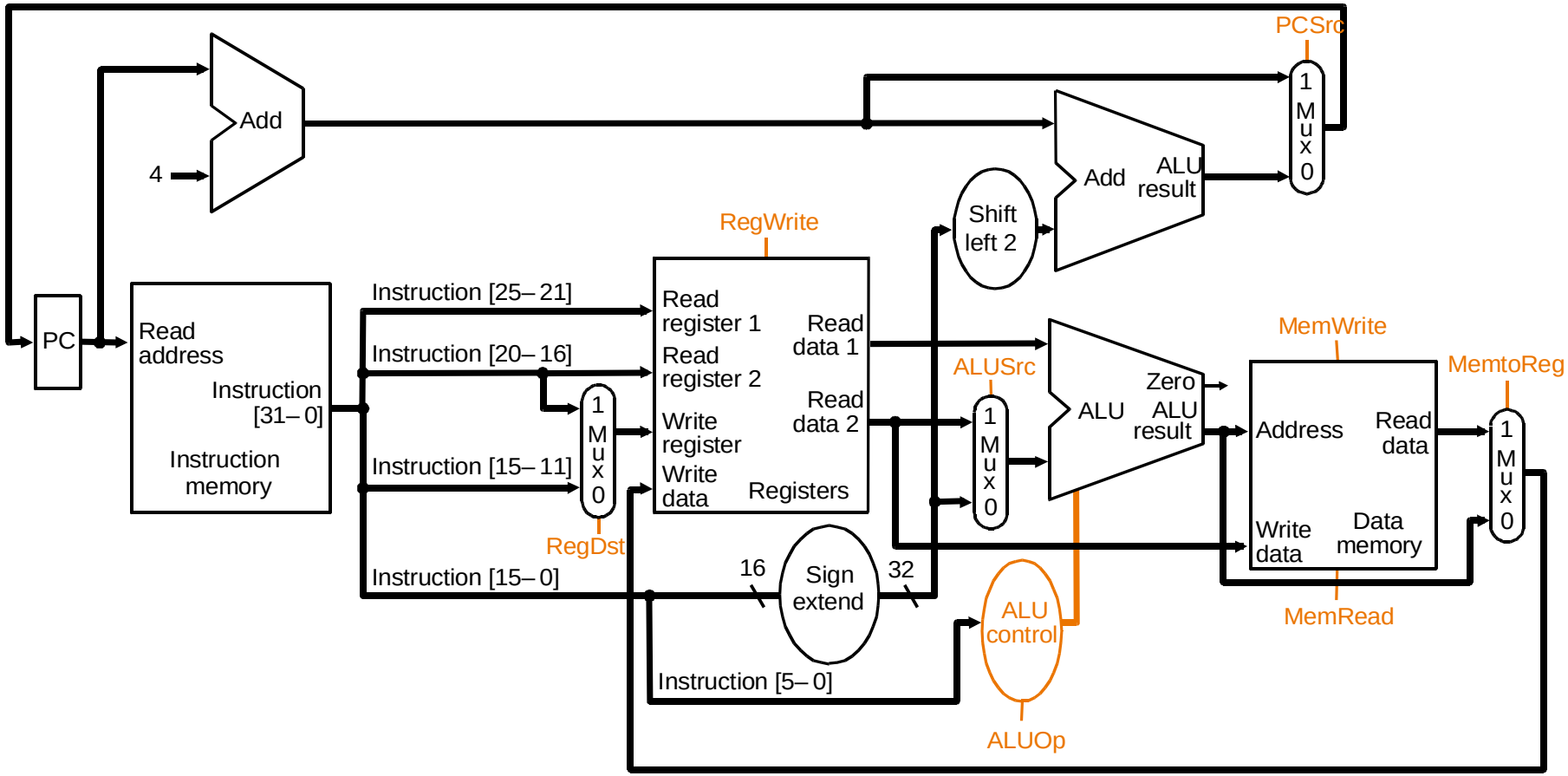
Anno Accademico 2011/2012

Francesco Tortorella

Progettazione del datapath

- Prima soluzione: d.p. a ciclo singolo
  - Semplice da realizzare
  - Condizionato dal worst case (istruzione più lenta)
  - Ogni unità funzionale è usata una volta sola per ciclo
- Si può dare di più ?
- Pipeline
- Da dove partiamo ?

Il datapath a ciclo singolo

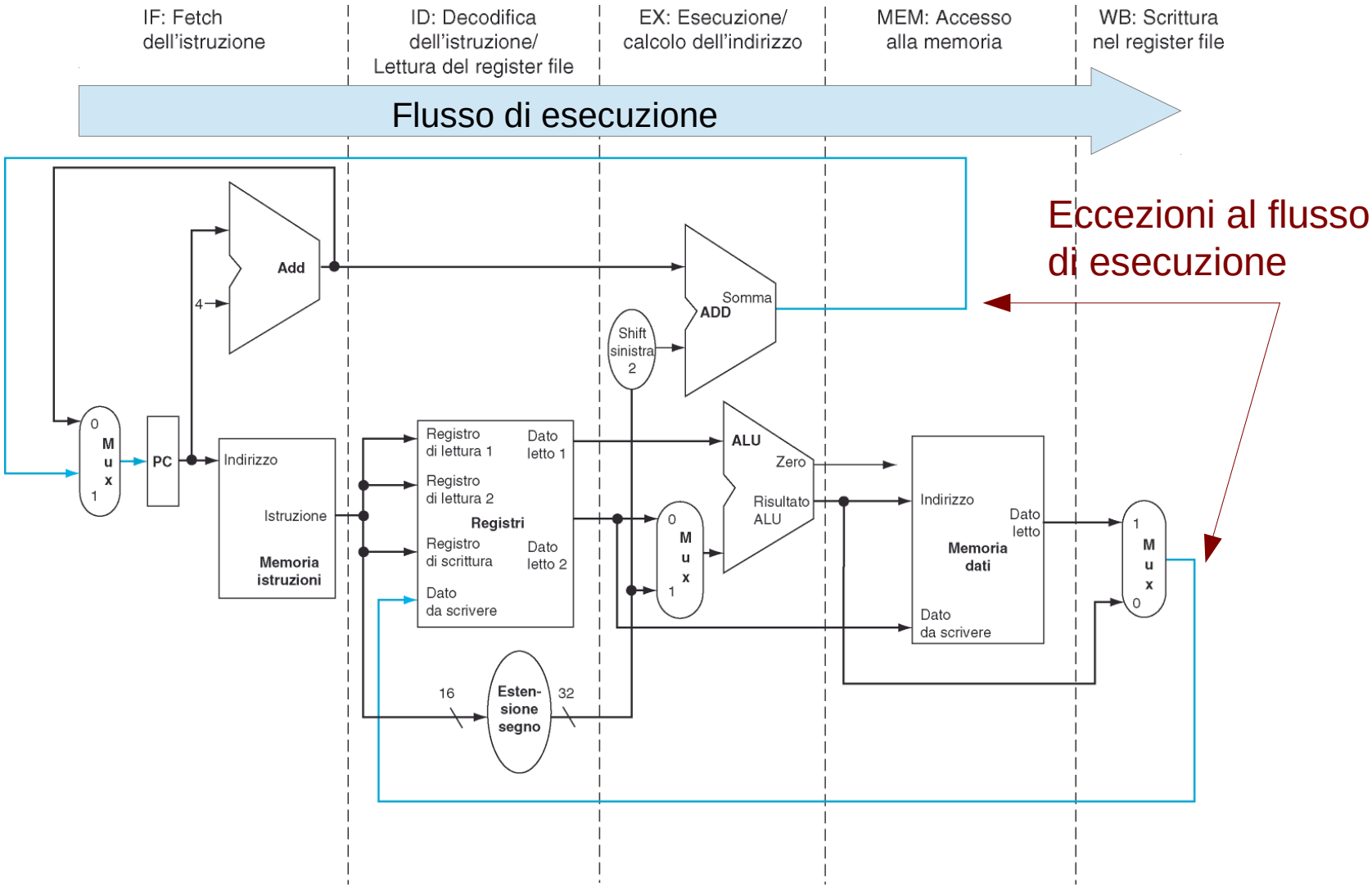


5 fasi per l'esecuzione di un'istruzione

- IFetch: Instruction Fetch e aggiornamento PC
- Dec: Registers Fetch e decodifica istruzione
- Exec:
  Esecuzione (R-type)
  Calcolo dell'indirizzo di memoria (I-type)
- Mem:
  Lettura dato dalla memoria (lw)
  Scrittura dato in memoria (sw)
- WB: Scrittura del risultato nel register file

Eseguite da tutte le istruzioni ?

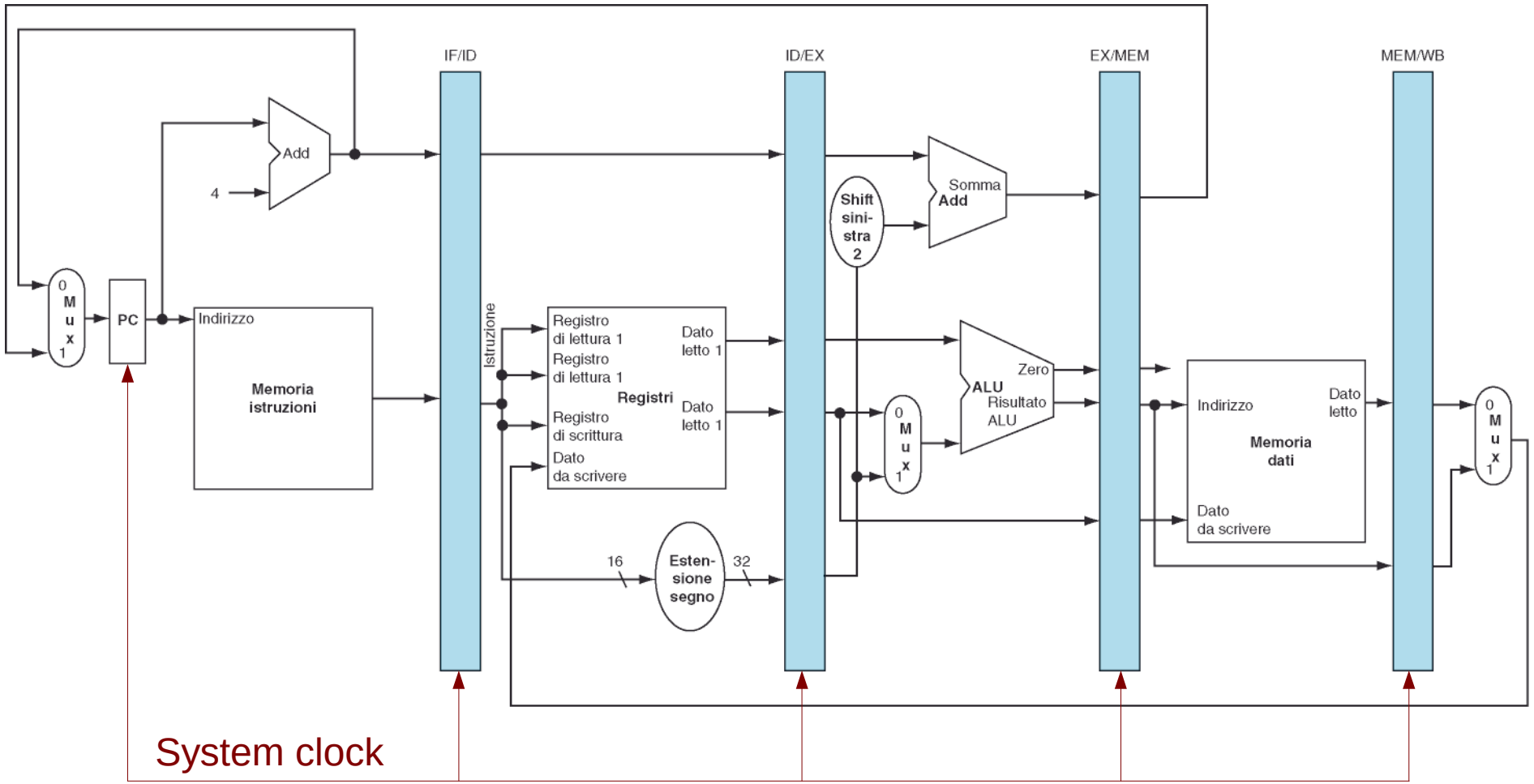
Dove si realizzano le varie fasi ?



Dal ciclo singolo al pipeline

- Come permettere l'utilizzo dell'hardware da più istruzioni in contemporanea ?
- Si inseriscono registri tra i vari stadi
    - Mantengono le informazioni prodotte nel ciclo precedente
- Si ottiene una pipeline a 5 stadi
    - Frequenza di clock potenzialmente 5 volte più alta

Datapath basato su pipeline



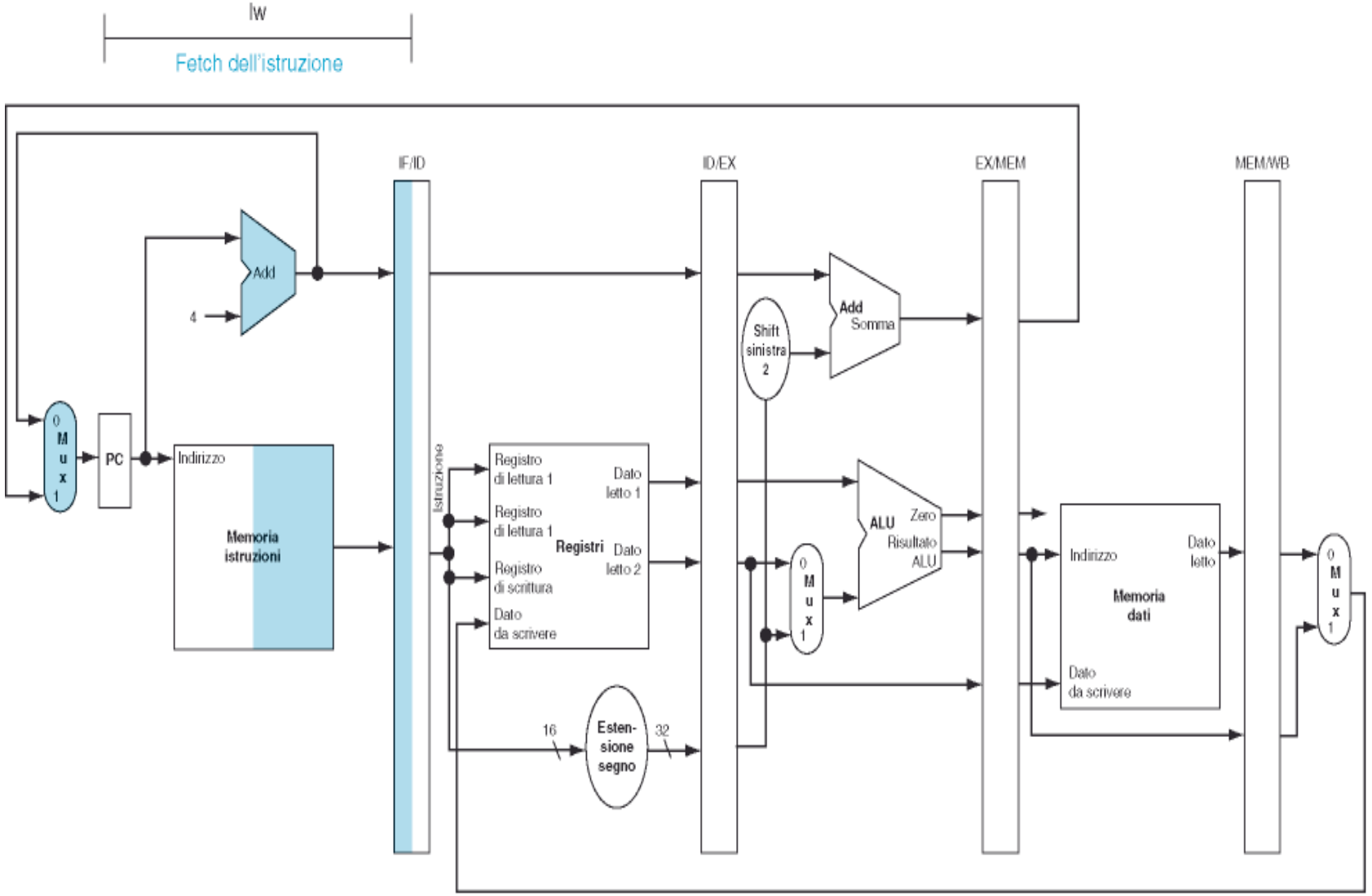
Prime considerazioni

- I registri sono presenti tra le varie parti del datapath, ognuna delle quali realizza una particolare fase.

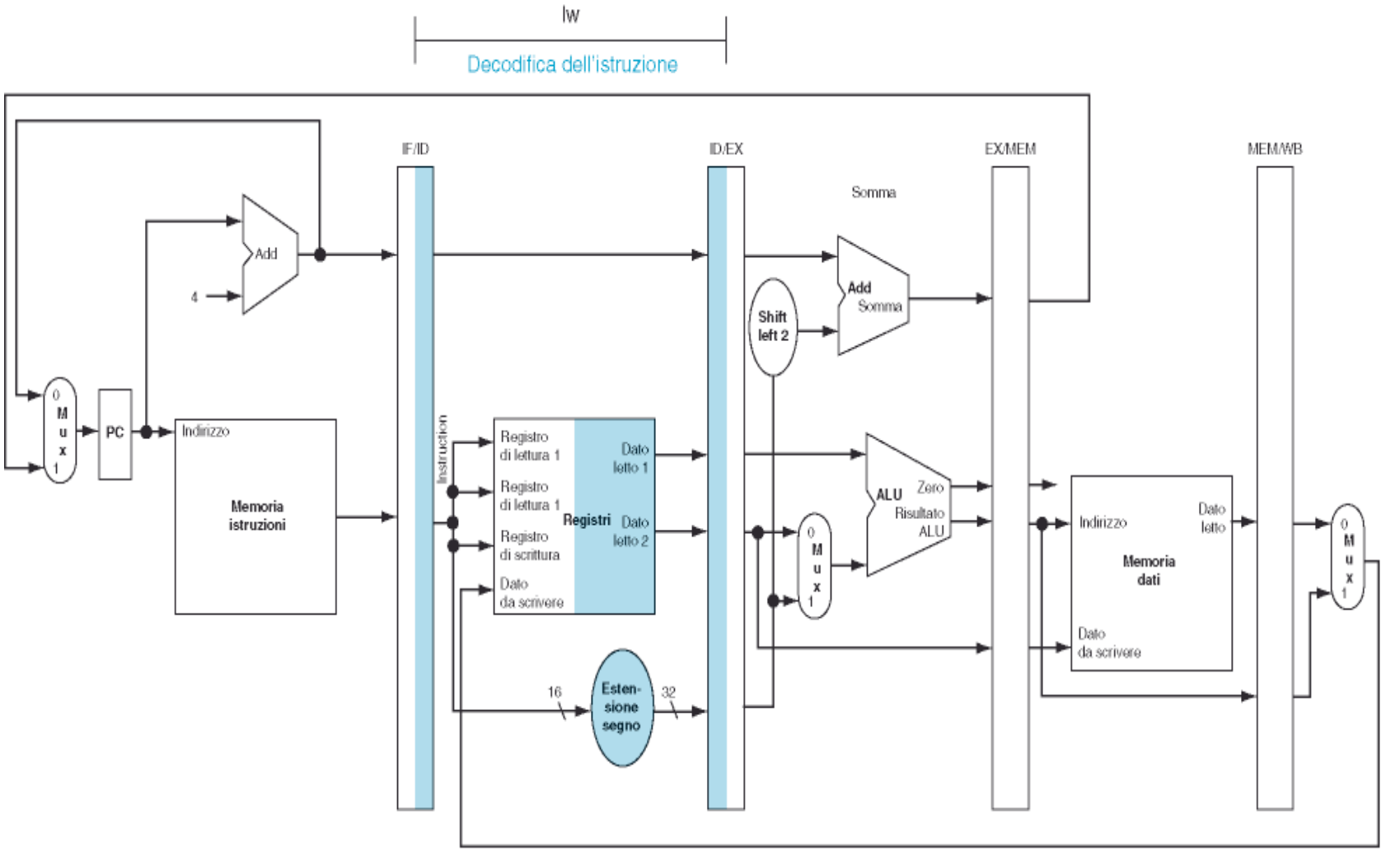
- I registri separano una sezione dall'altra, evitando così che le diverse istruzioni in esecuzione interferiscano tra loro.

- Il flusso dei dati è da sinistra verso destra con due eccezioni significative:
  - WB scrive il risultato nel file register, nel mezzo del data path
  - Uno dei possibili valori per l'aggiornamento del PC proviene dalla sezione MEM
- Solo le istruzioni successive possono essere influenzate da questi movimenti dati in "direzione opposta" al flusso consueto:
  - WB -> ID provoca data hazards (alea sui dati)
  - MEM -> IF provoca control hazards (alea sul controllo)

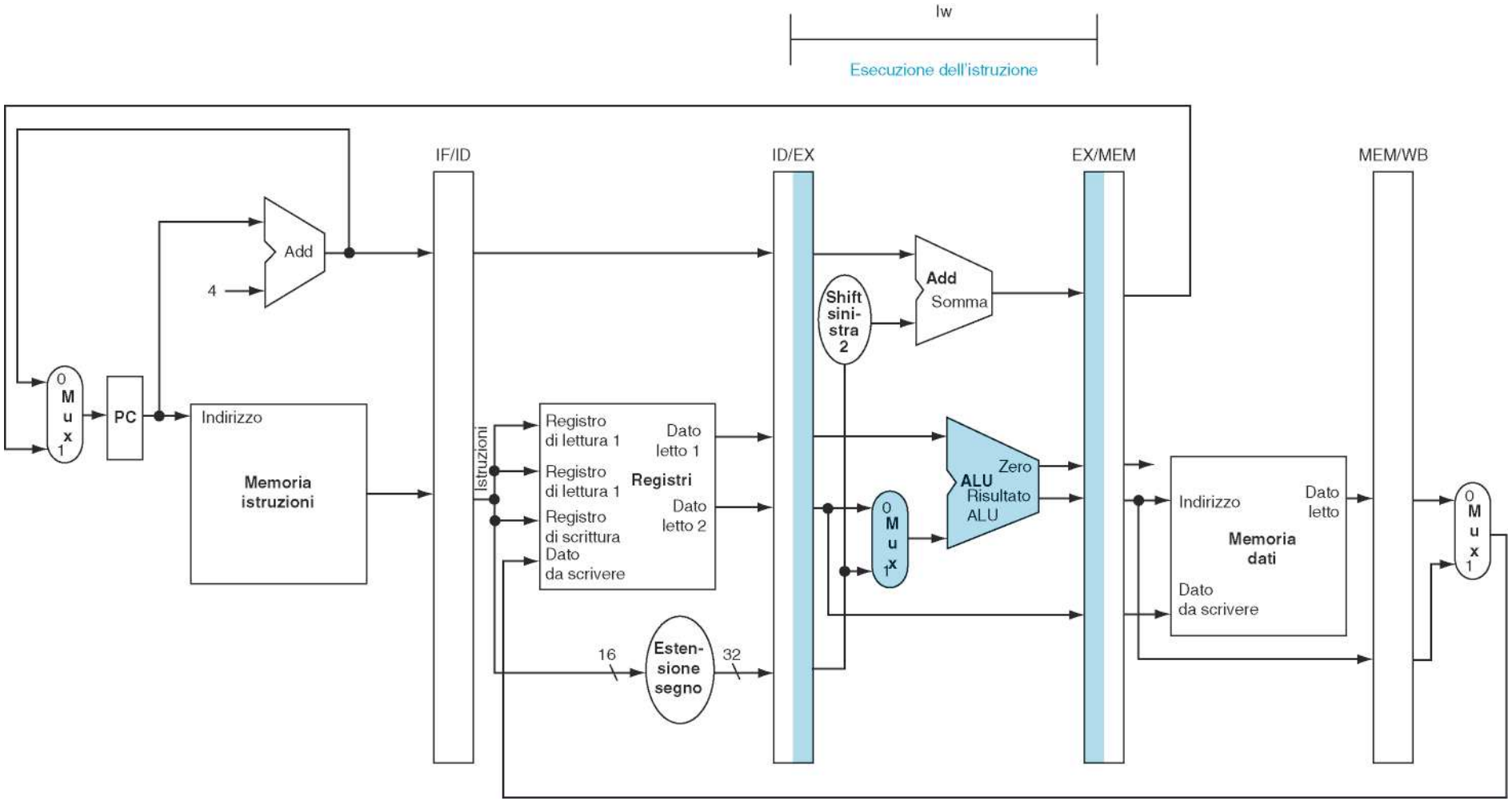
Esecuzione di lw



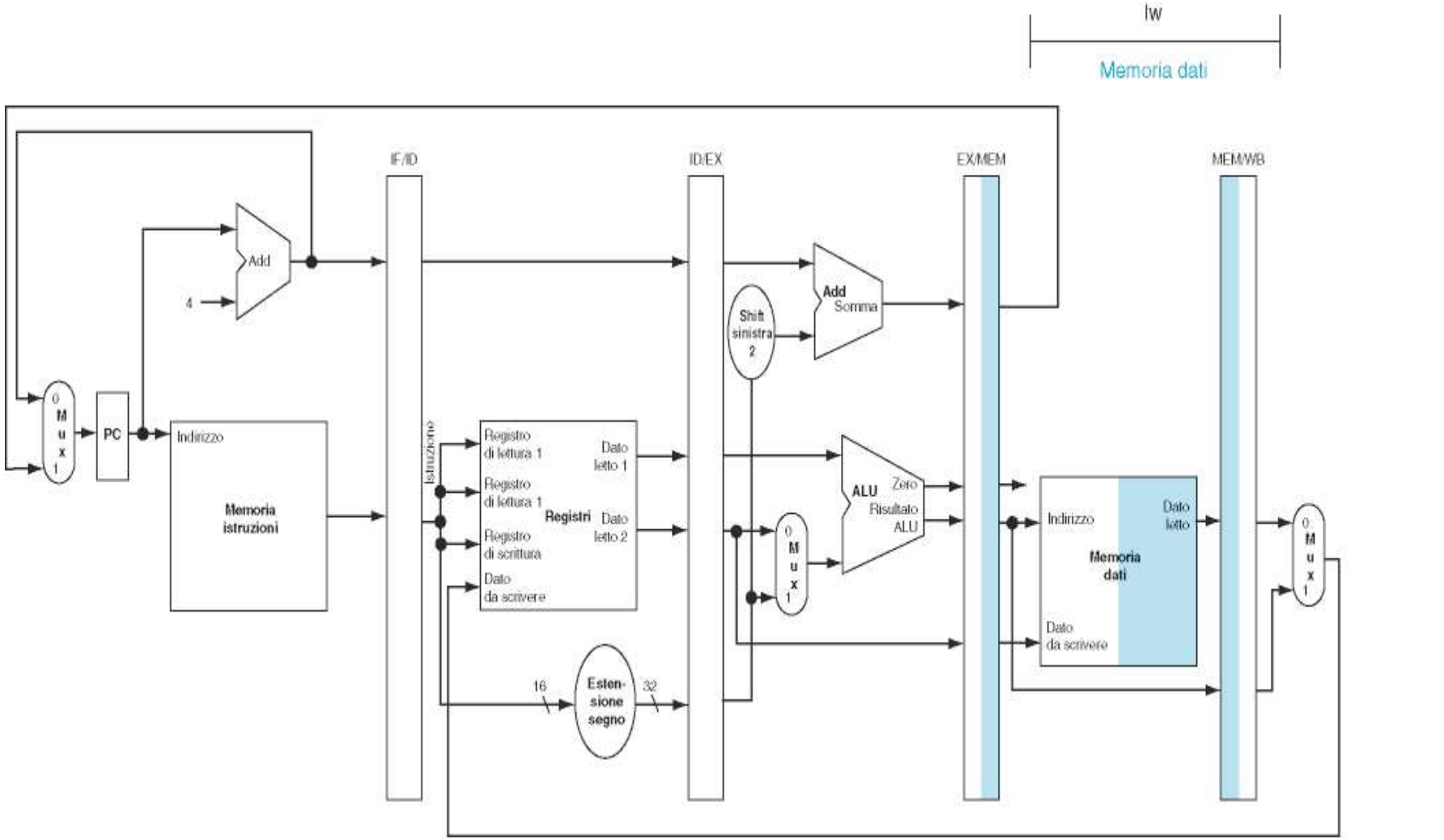
Esecuzione di lw



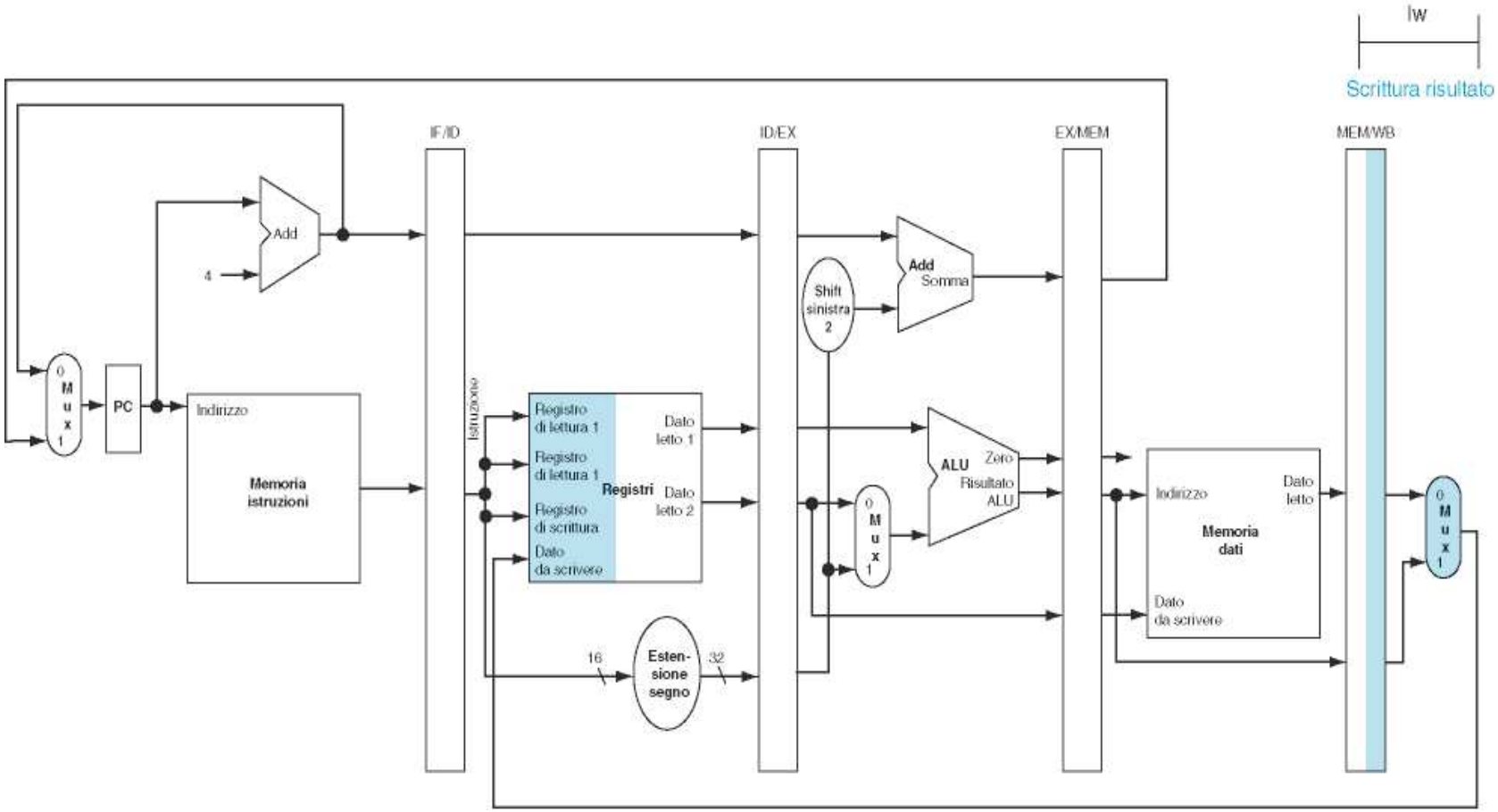
Esecuzione di lw



Esecuzione di lw



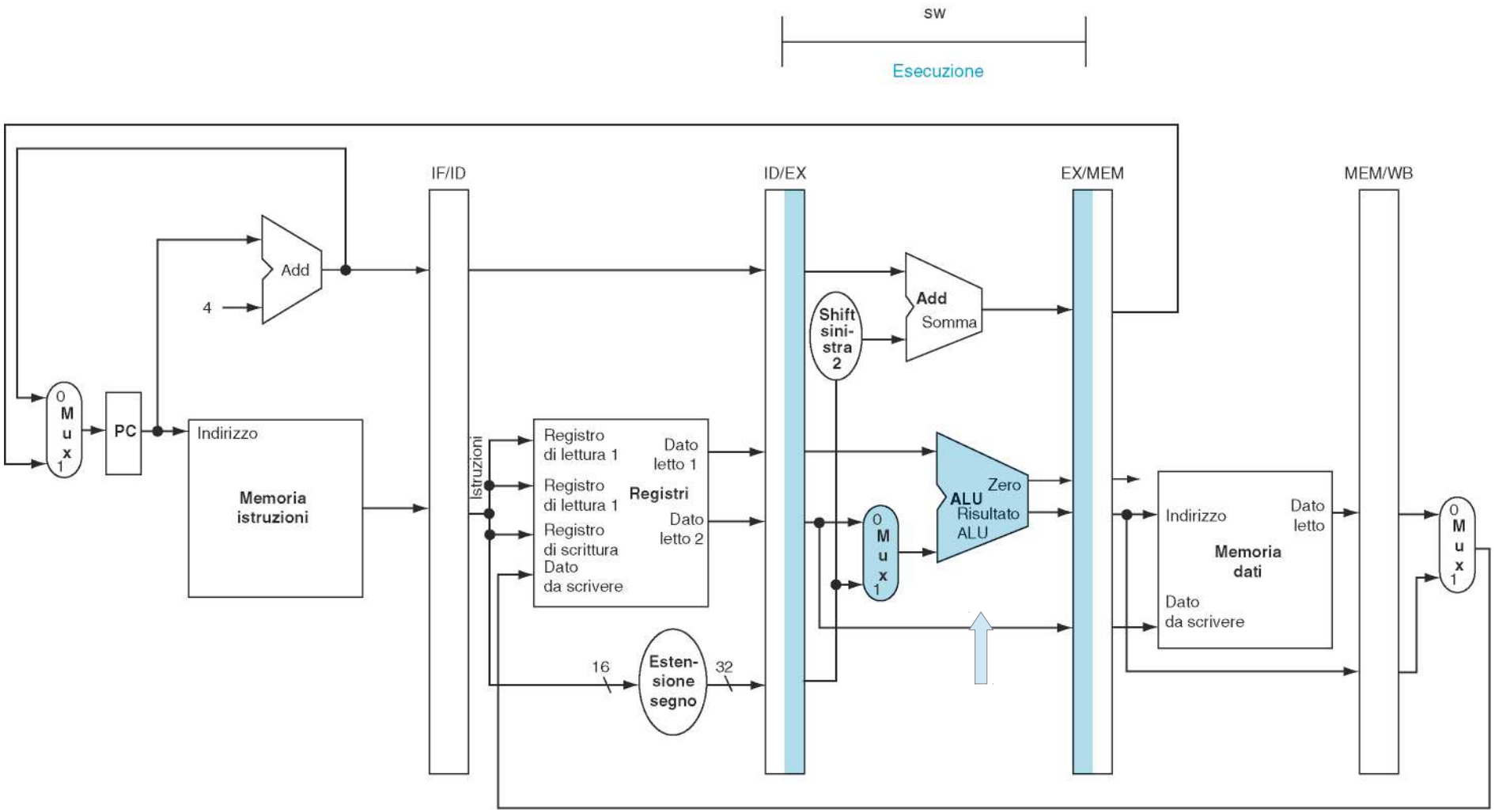
Esecuzione di lw



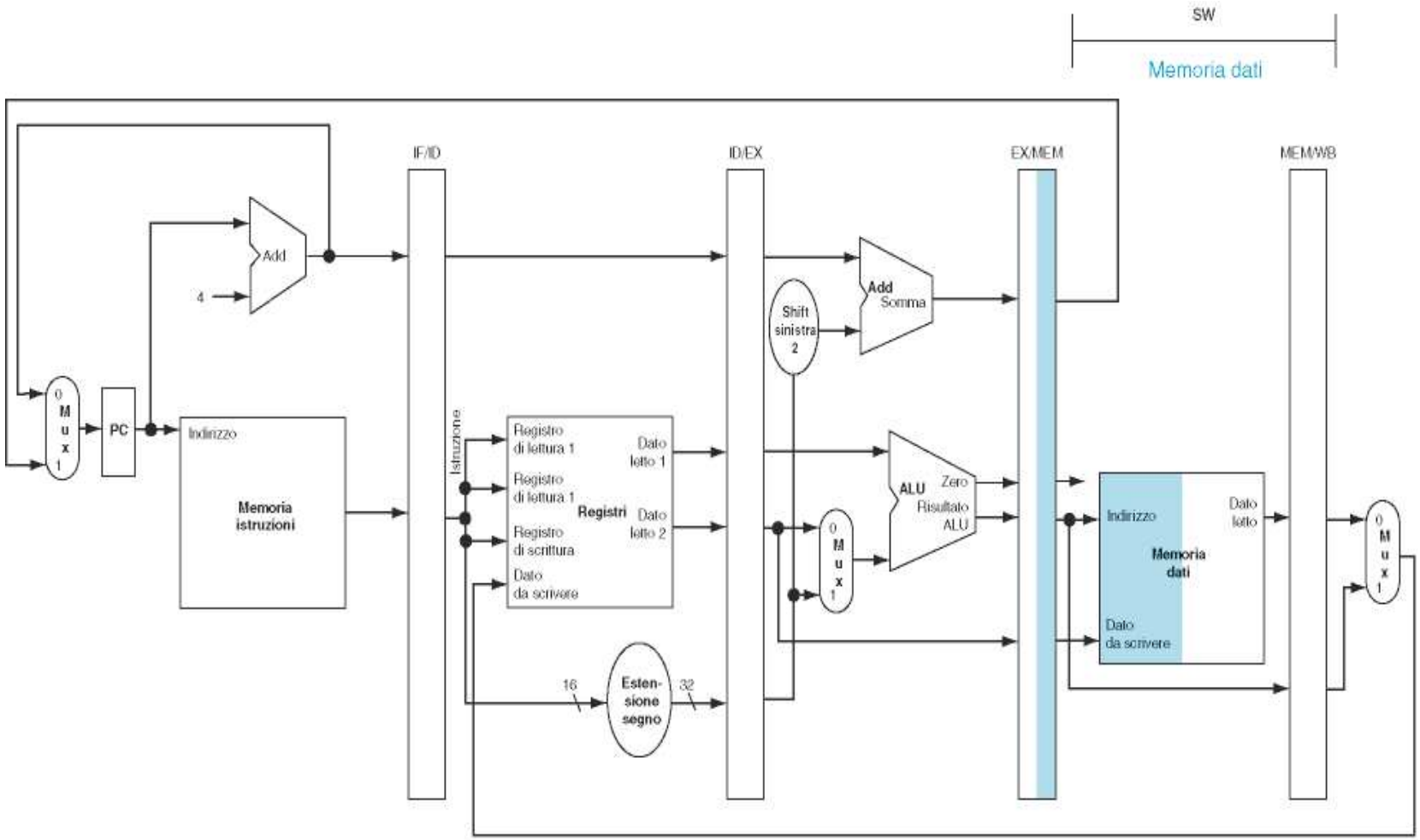
Esecuzione di sw

- Che cosa cambia ?
  - IF, ID uguali
- In EX viene inoltrato il contenuto di Rt
- In MEM viene scritto il valore in memoria
- Che cosa succede in WB ?

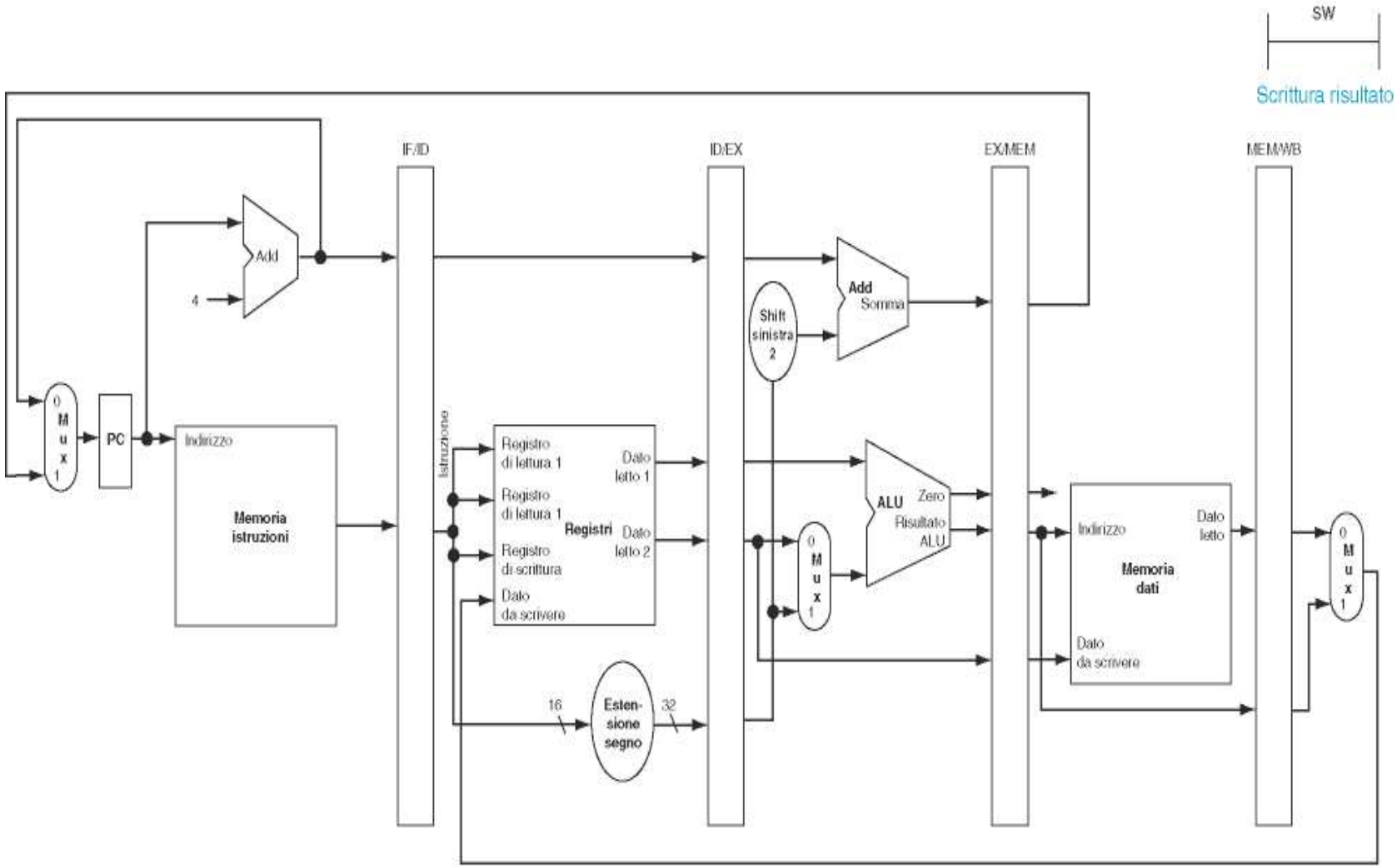
Esecuzione di sw (EX)



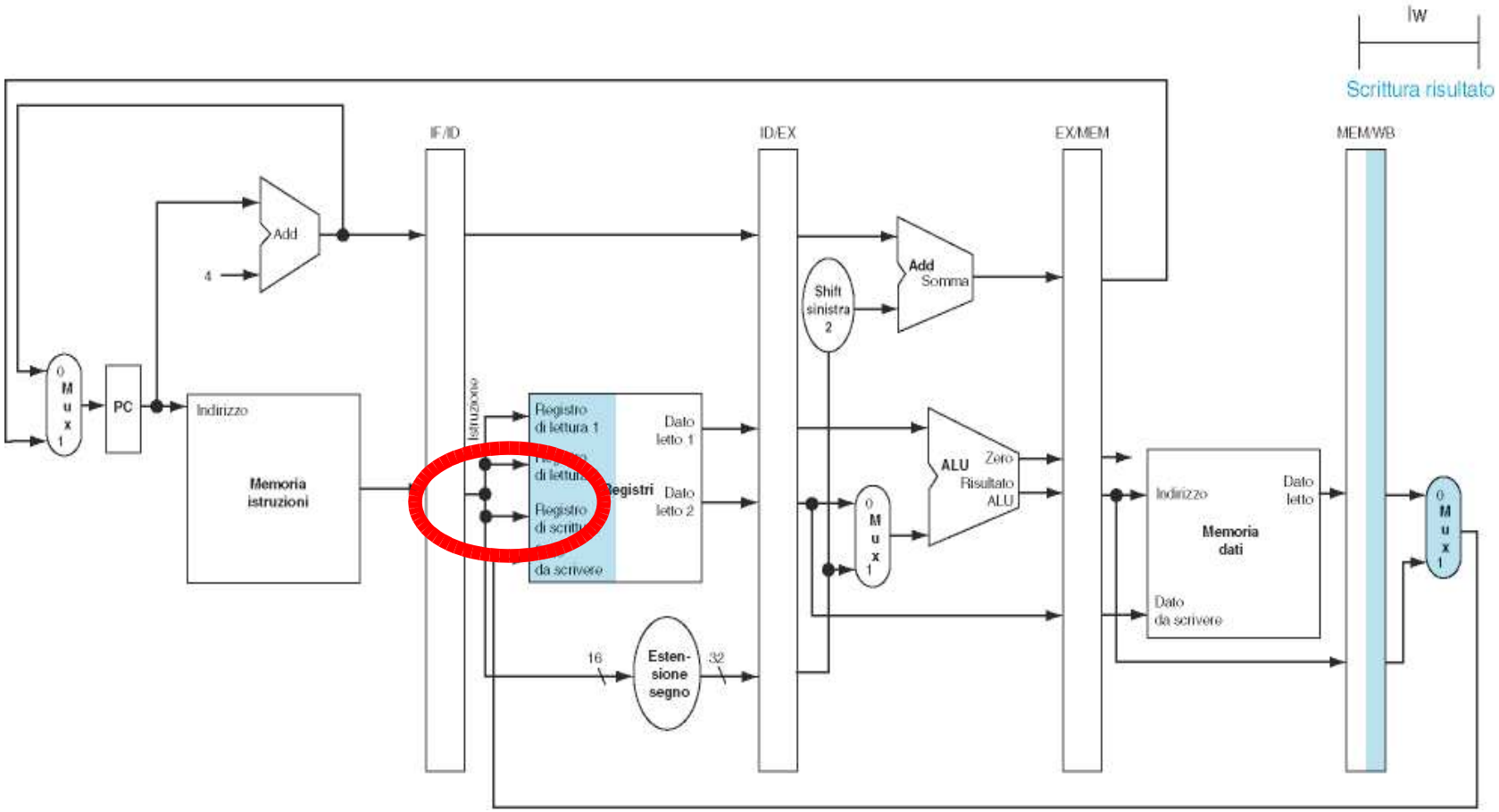
Esecuzione di sw (MEM)



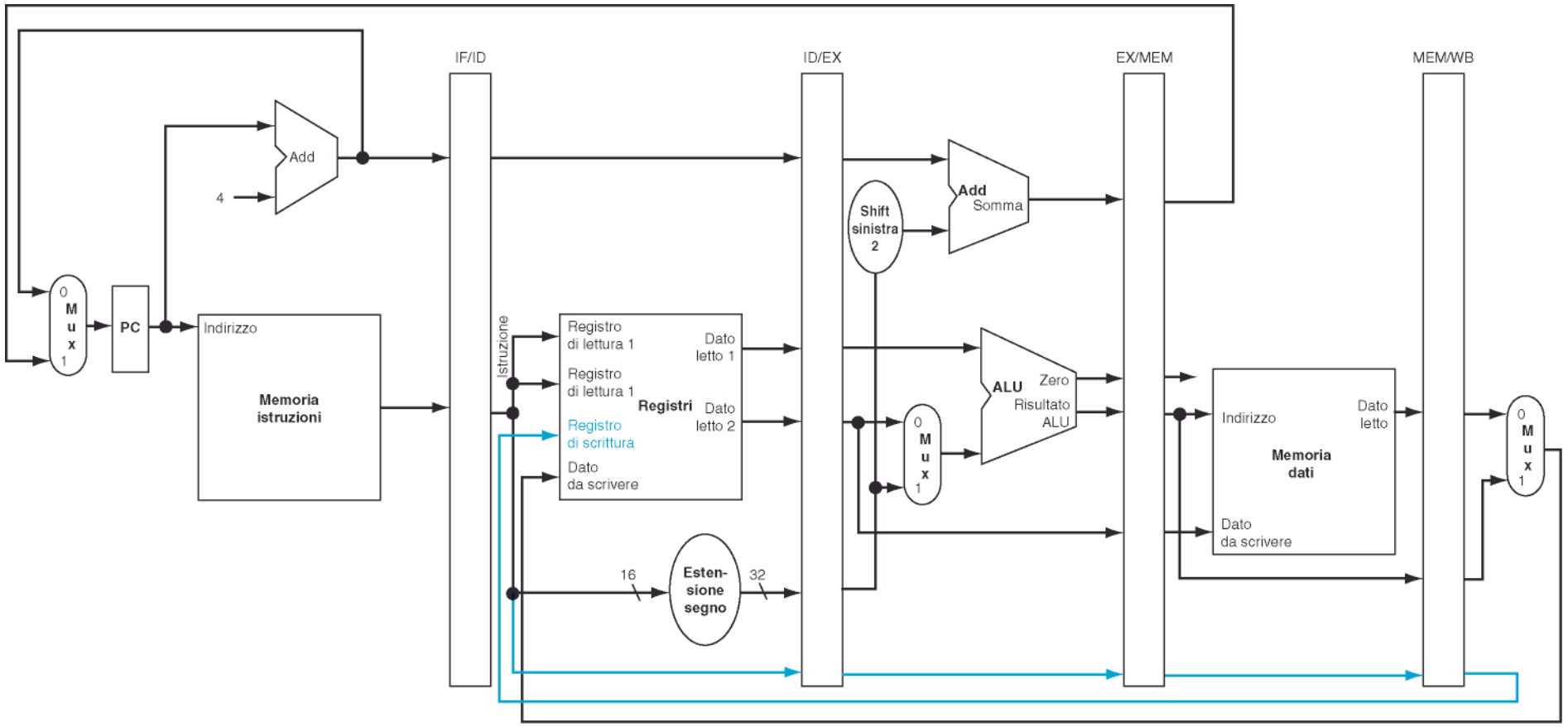
Esecuzione di sw (WB)



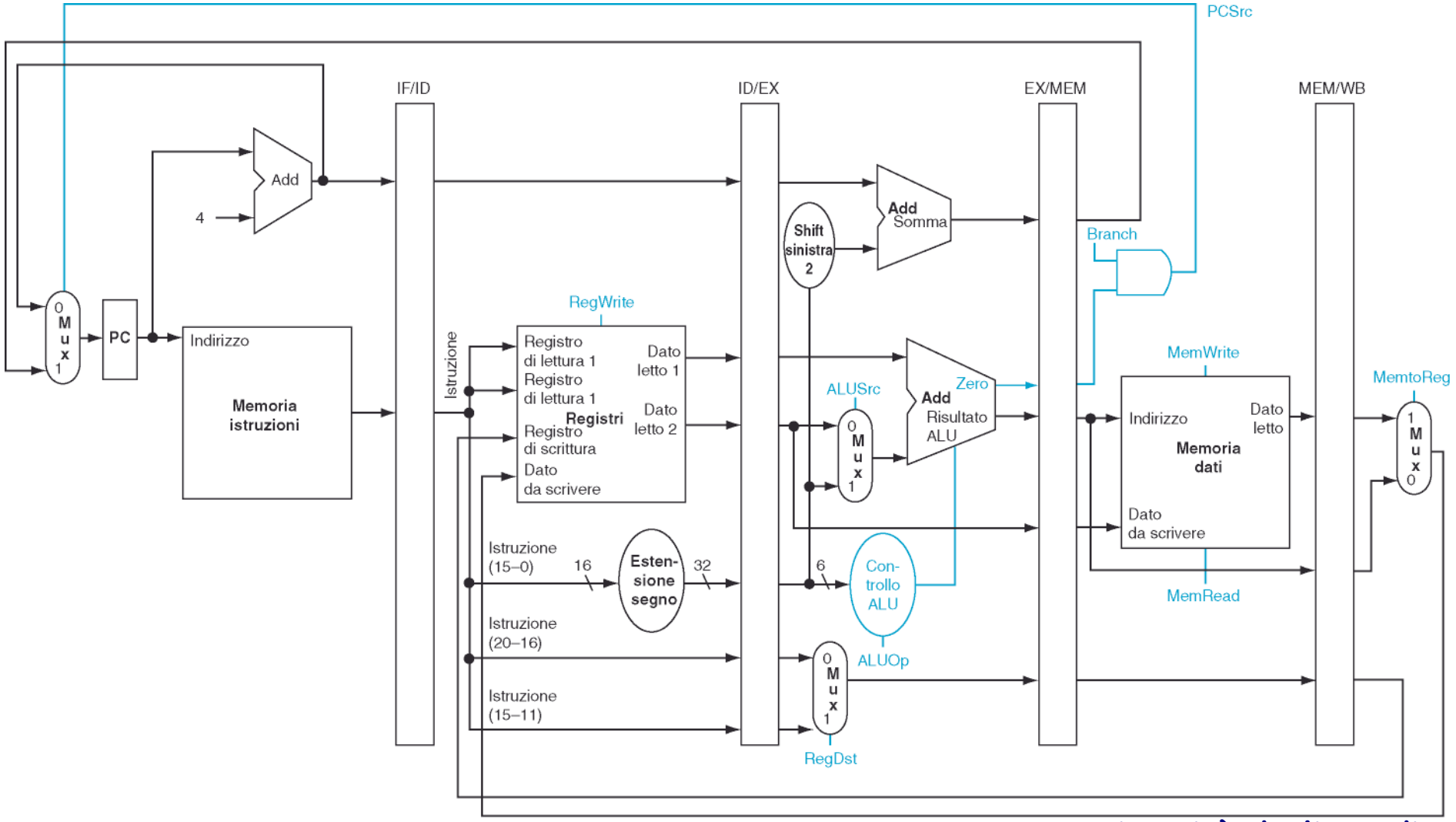
Problemi nell'esecuzione di lw ?



Datapath corretto per lw



Unità di controllo della pipeline



Segnali di controllo

Nome del segnale	Effetto quando non asserito (0)	Effetto quando asserito (1)
RegDst	Il numero del registro di scrittura proviene dal campo rt (bit 20-16)	Il numero del registro di scrittura proviene dal campo rd (bit 15-11)
RegWrite	Nulla	Il dato viene scritto nel registro (del register file) individuato dal numero del registro di scrittura
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita del register file (Dato letto 2)	Il secondo operando della ALU proviene dall'estensione del segno dei 16 bit meno significativi dell'istruzione
PCSrc	Nel PC viene scritta l'uscita del sommatore che calcola il valore di PC + 4	Nel PC viene scritta l'uscita del sommatore che calcola l'indirizzo di salto
MemRead	Nulla	Il dato della memoria nella posizione puntata dall'indirizzo viene inviato in uscita sulla linea «dato letto»
MemWrite	Nulla	Il contenuto della memoria nella posizione puntata dall'indirizzo viene sostituito con il dato presente sulla linea «dato scritto»
MemtoReg	Il dato inviato al register file per la scrittura proviene dalla ALU	Il dato inviato al register file per la scrittura proviene dalla Memoria Dati

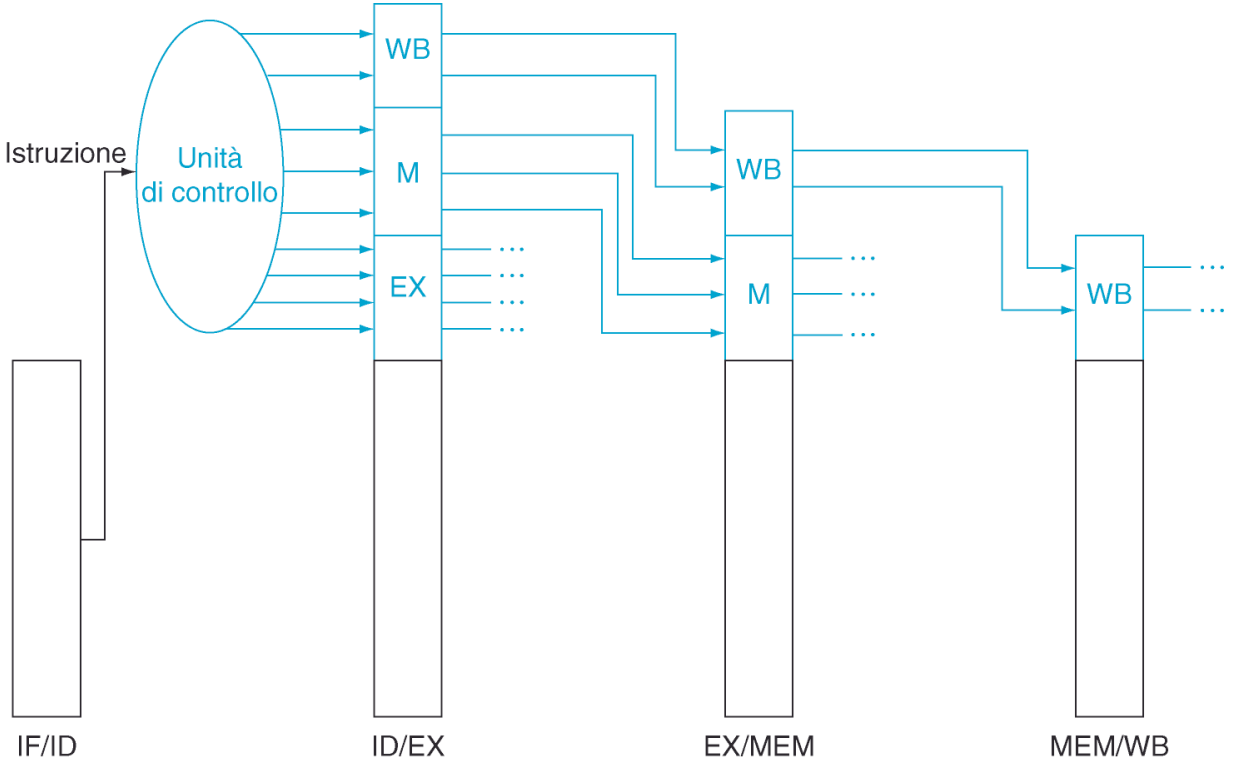
Istruzioni / Segnali di controllo

Istruzione	Segnali di controllo dello stadio di esecuzione/calcolo dell'indirizzo				Segnali di controllo dello stadio di accesso alla memoria dati			Segnali di controllo dello stadio di *Write-back*	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Read Mem	Write Mem	RegWrite	Memto-Reg
Formato-R	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

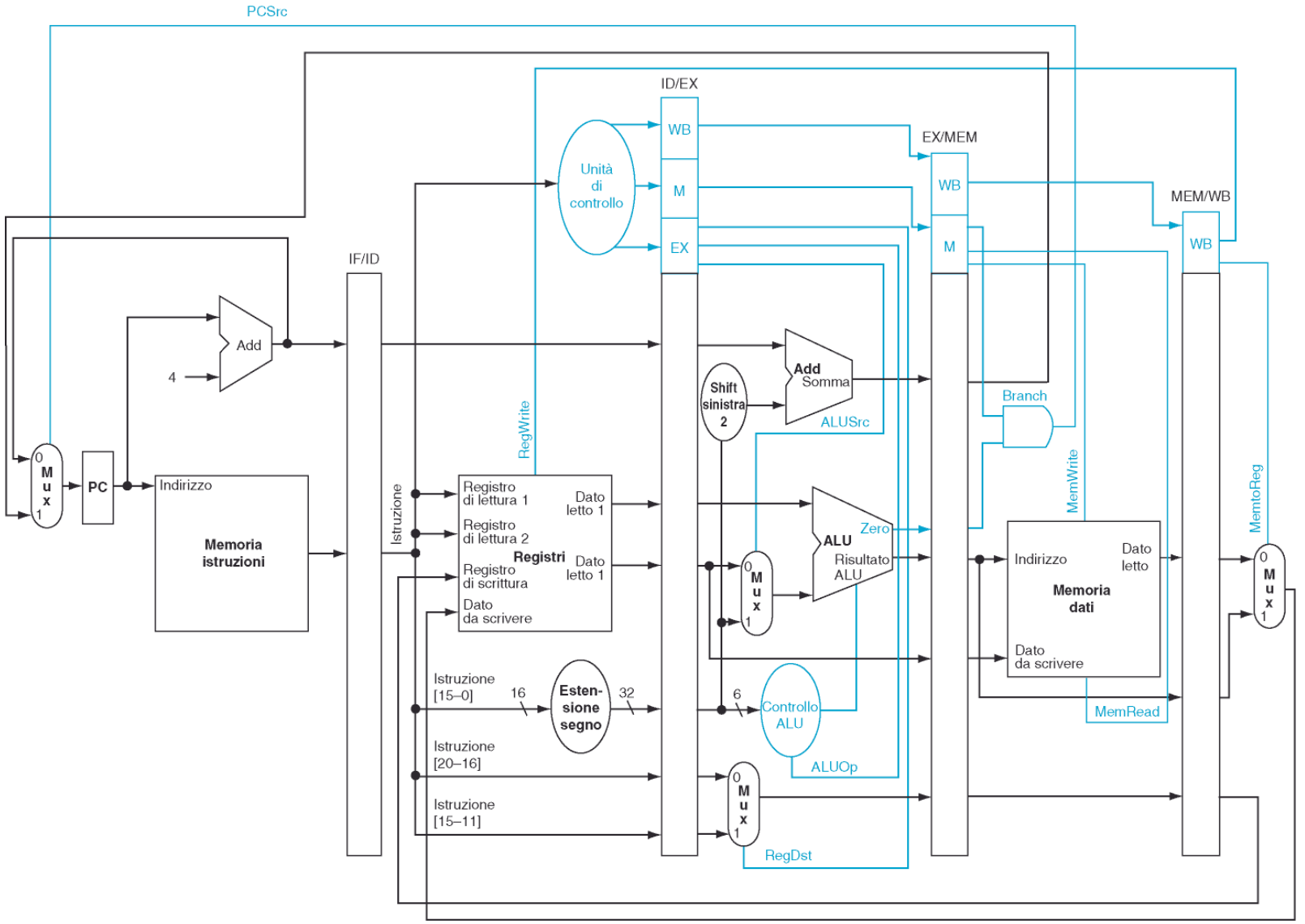
- Sono raggruppati secondo lo stadio in cui sono attivi
- I primi due stadi non contengono segnali di controllo
  - Perchè ?

Distribuzione dei segnali di controllo

- I segnali vengono determinati durante la fase ID
- Vengono usati nelle tre fasi successive: vanno quindi inoltrati fra gli stadi.

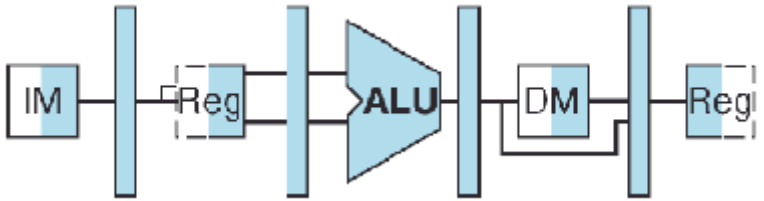


Datapath pipelined completo (quasi ...)



Rappresentazione grafica della pipeline

- Necessario rappresentare la situazione della pipeline in un certo istante
  - Quali istruzioni si stanno eseguendo ? In quale stadio?
- I diagrammi visti finora rappresentano la situazione in un solo ciclo di clock (*diagrammi a singolo ciclo*)
- Sono necessari diagrammi che si riferiscano a più cicli di clock (*diagrammi a più cicli*)
- A questo scopo si ado … e sintetica del datapath



Rappresentazione grafica della pipeline

Consideriamo la sequenza di cinque istruzioni:

```
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
```

E supponiamo di voler considerare la loro esecuzione sul datapath pipelined al progredire del clock

Avanzamento delle istruzioni

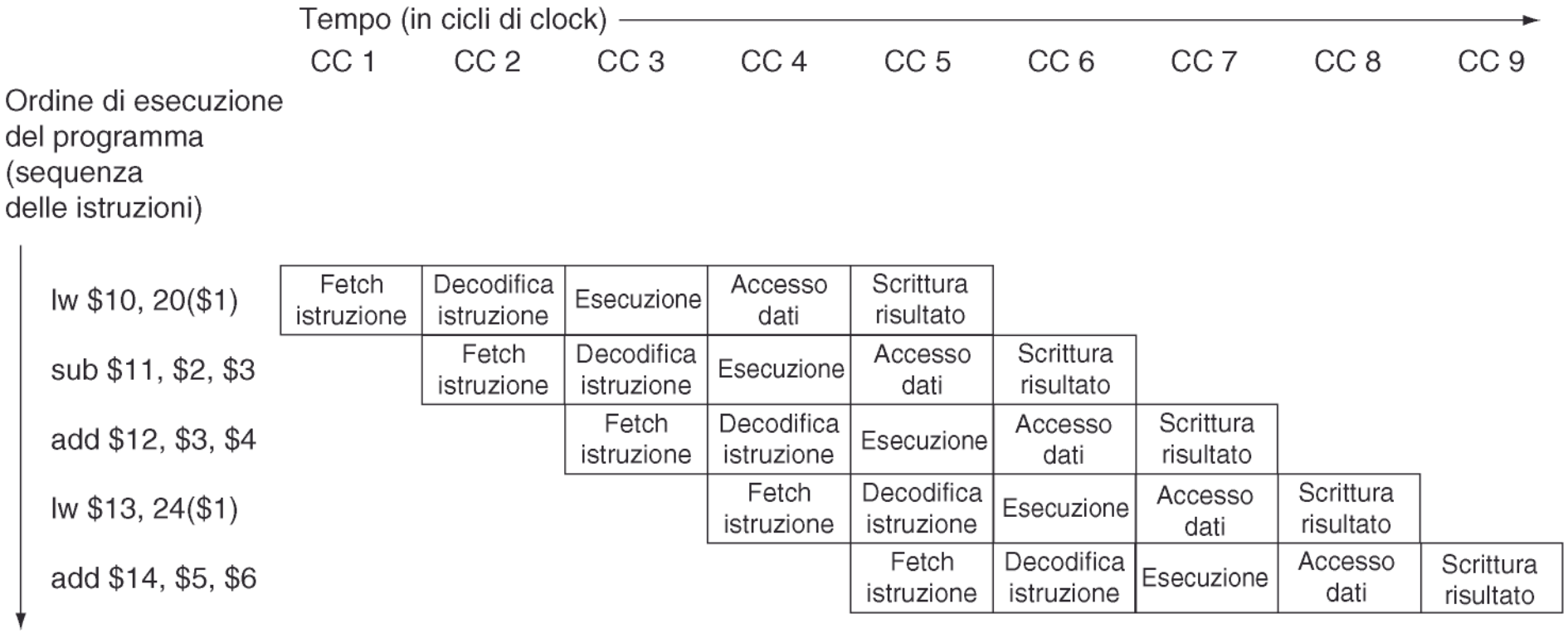
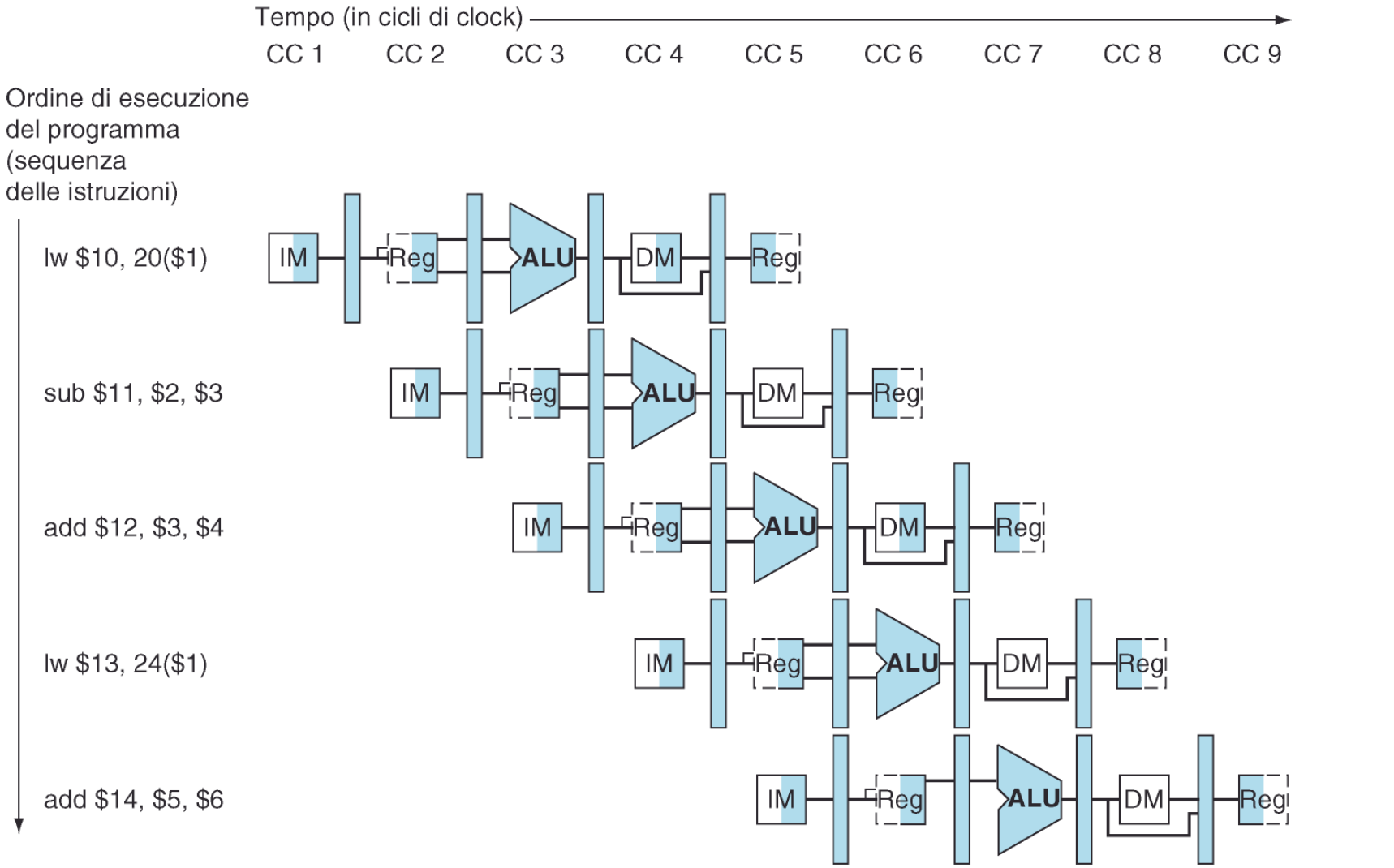
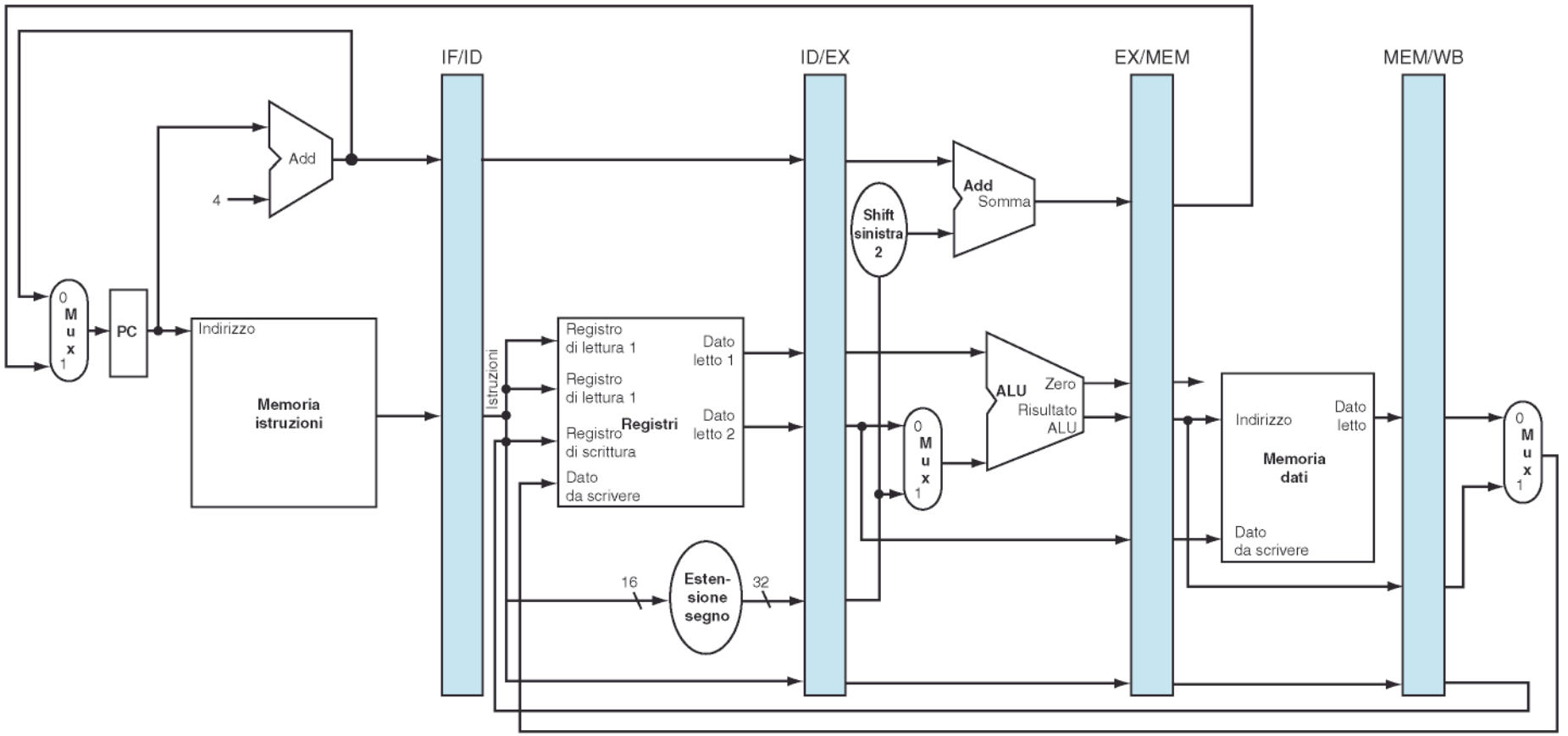


Diagramma a più cicli

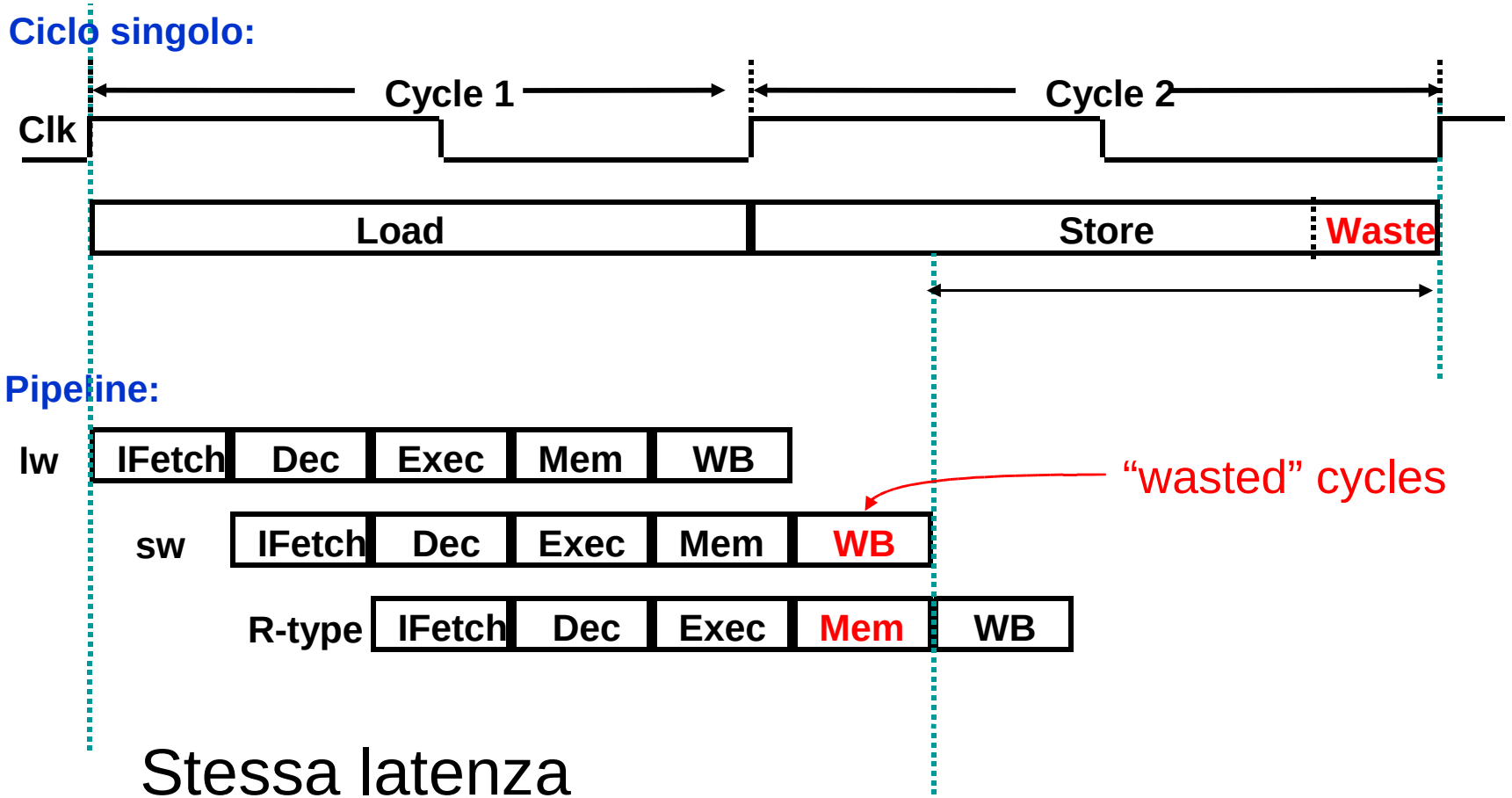


Situazione al ciclo 5 (CC5)

add \$14, \$5, \$6	lw \$13, 24 (\$1)	add \$12, \$3, \$4	sub \$11, \$2, \$3	lw \$10, 20(\$1)
Fetch istruzione	Decodifica istruzione	Esecuzione	Memoria dati	Scrittura risultato



Confronto tra le implementazioni

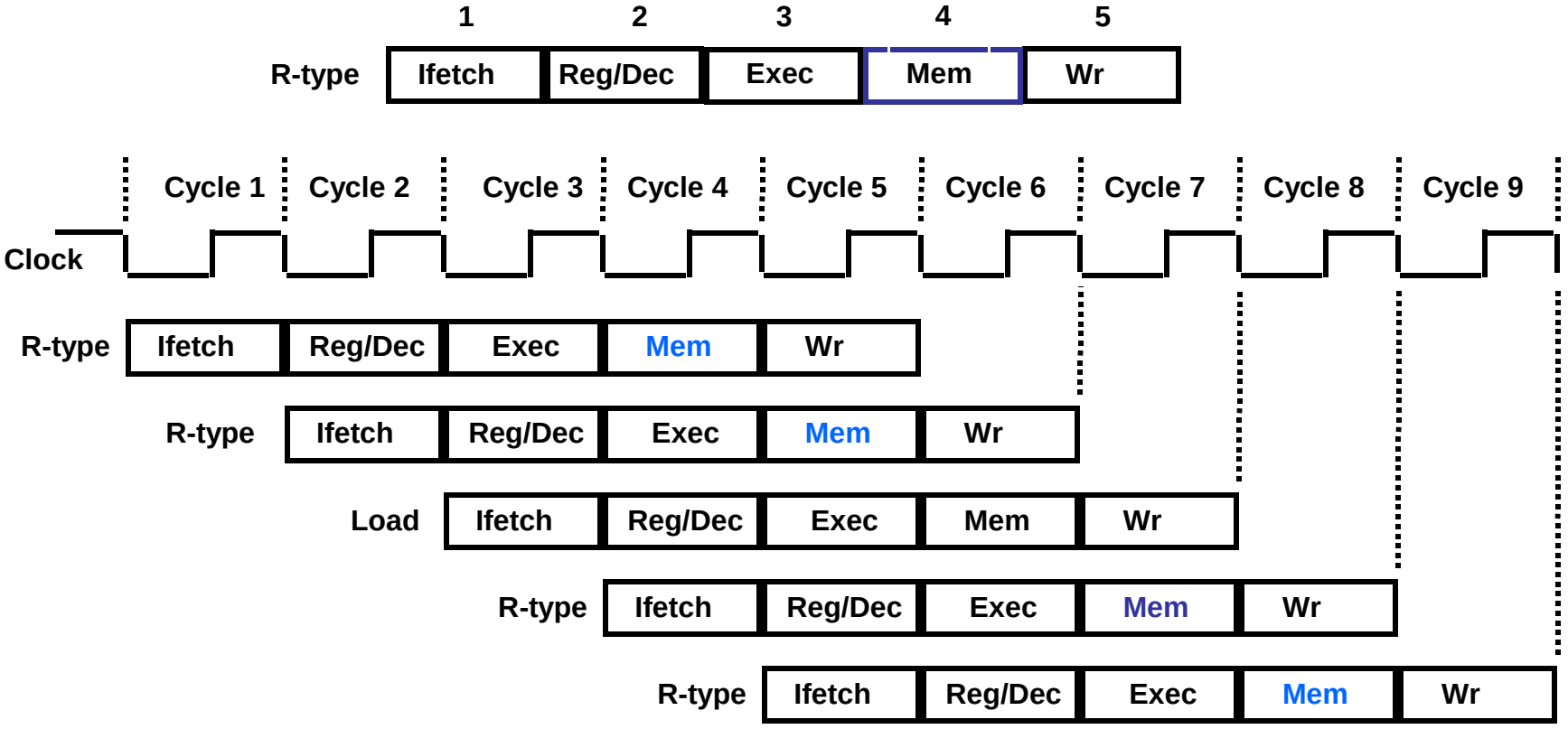


Stessa latenza

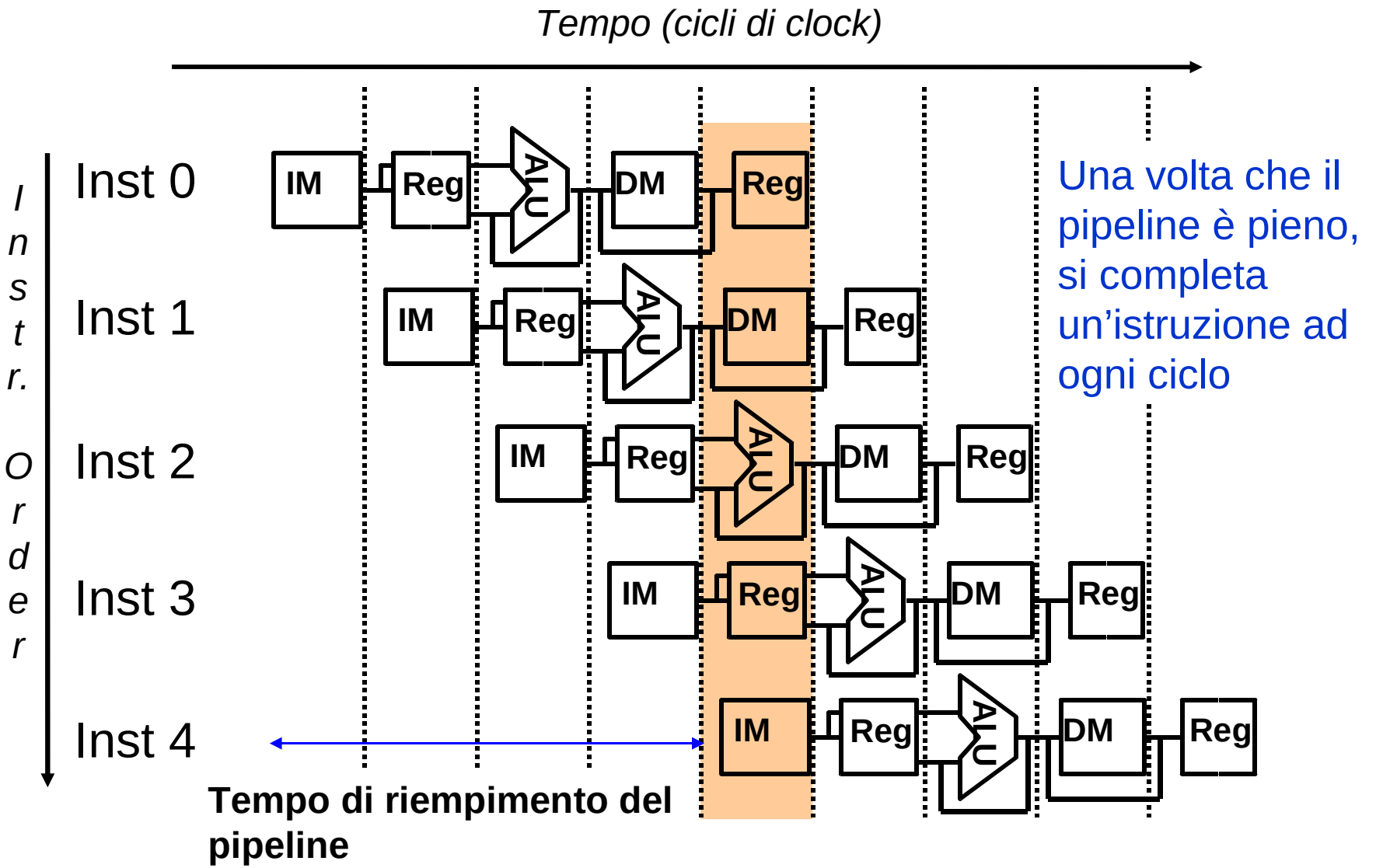
Throughput diverso

Ritardo del Write

- Si ritarda di un ciclo il write delle istruzioni R-type
  - Tutte le istruzioni accedono al Write Port del R.F. nel 5° passo →nessun conflitto
  - Il 4° passo (Mem) per le istr. R-type è in effetti un passo NOP



Il pipeline ottimizza il throughput



Come l'ISA MIPS favorisce il pipelining

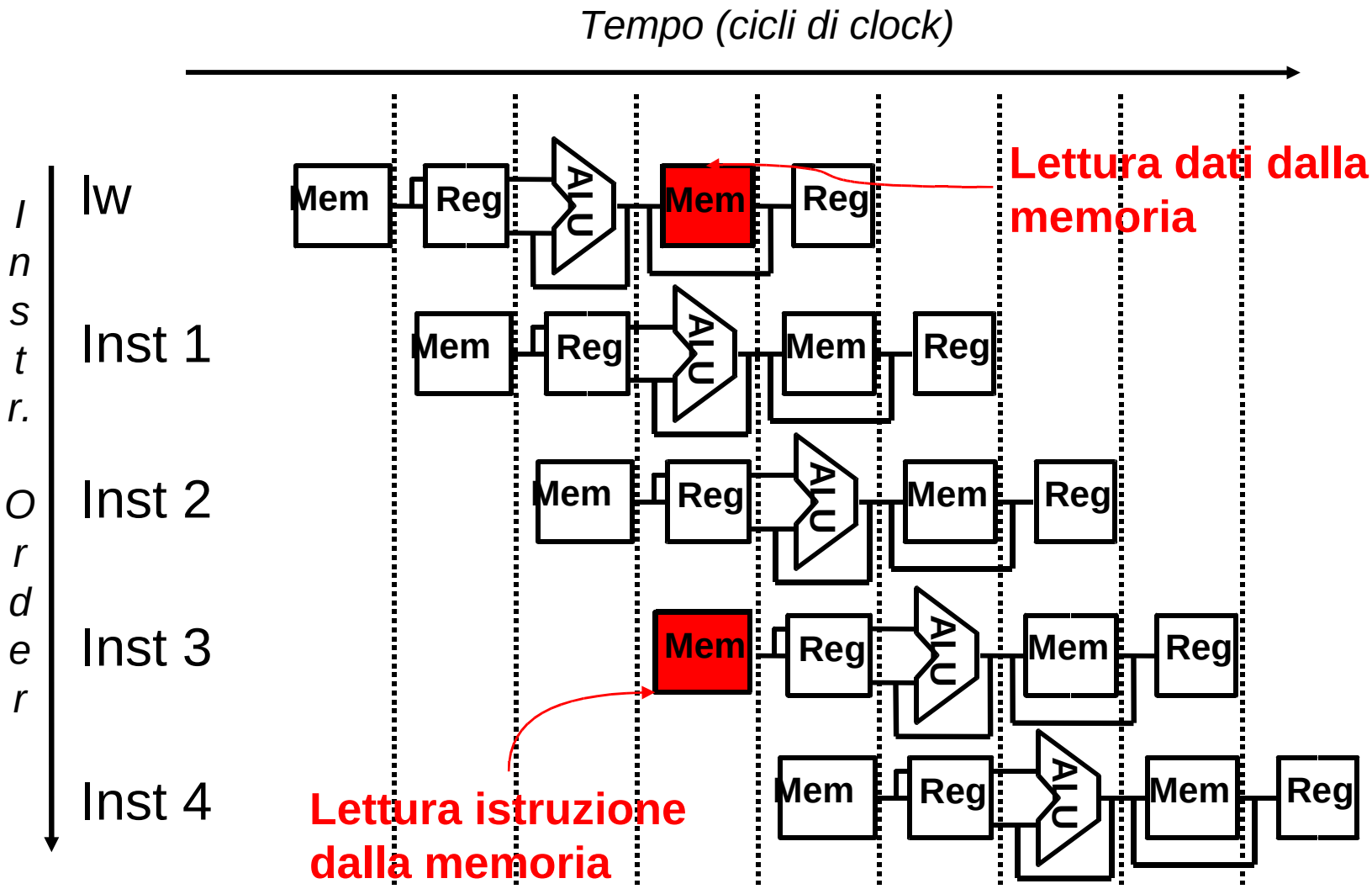
- Tutte le istruzioni hanno la stessa lunghezza
  - Più semplice il fetch nel 1° passo e il decode nel 2° passo
- Pochi formati di istruzione (3) con simmetria tra i formati
  - Possibile cominciare la lettura del register file nel 2° passo
- Le operazioni in memoria limitate alle istruzioni di load/store
  - Possibile usare il passo di execute per calcolare gli indirizzi
- Ogni istruzione MIPS scrive al più un risultato e lo fa verso la fine del pipeline

Problemi dal pipelining ?

- **Alee**
  - Alee strutturali: tentativo di usare contemporaneamente la stessa risorsa da parte di due istruzioni differenti
  - Alee sui dati : tentativo di usare i dati prima che siano disponibili
    - Gli operandi sorgente di un'istruzione sono prodotti da un'istruzione precedente che è ancora nel pipeline
    - Istruzione di Load seguita immediatamente da un'istruzione che richiede l'operando del load come sorgente per un'operazione sull'ALU
  - Alee sul controllo: tentativo di prendere una decisione prima che la condizione sia stata valutata
    - Istruzioni di salto

- È sempre possibile eliminare le alee inserendo dei tempi morti ("bolle"), ma si abbassa l'efficienza

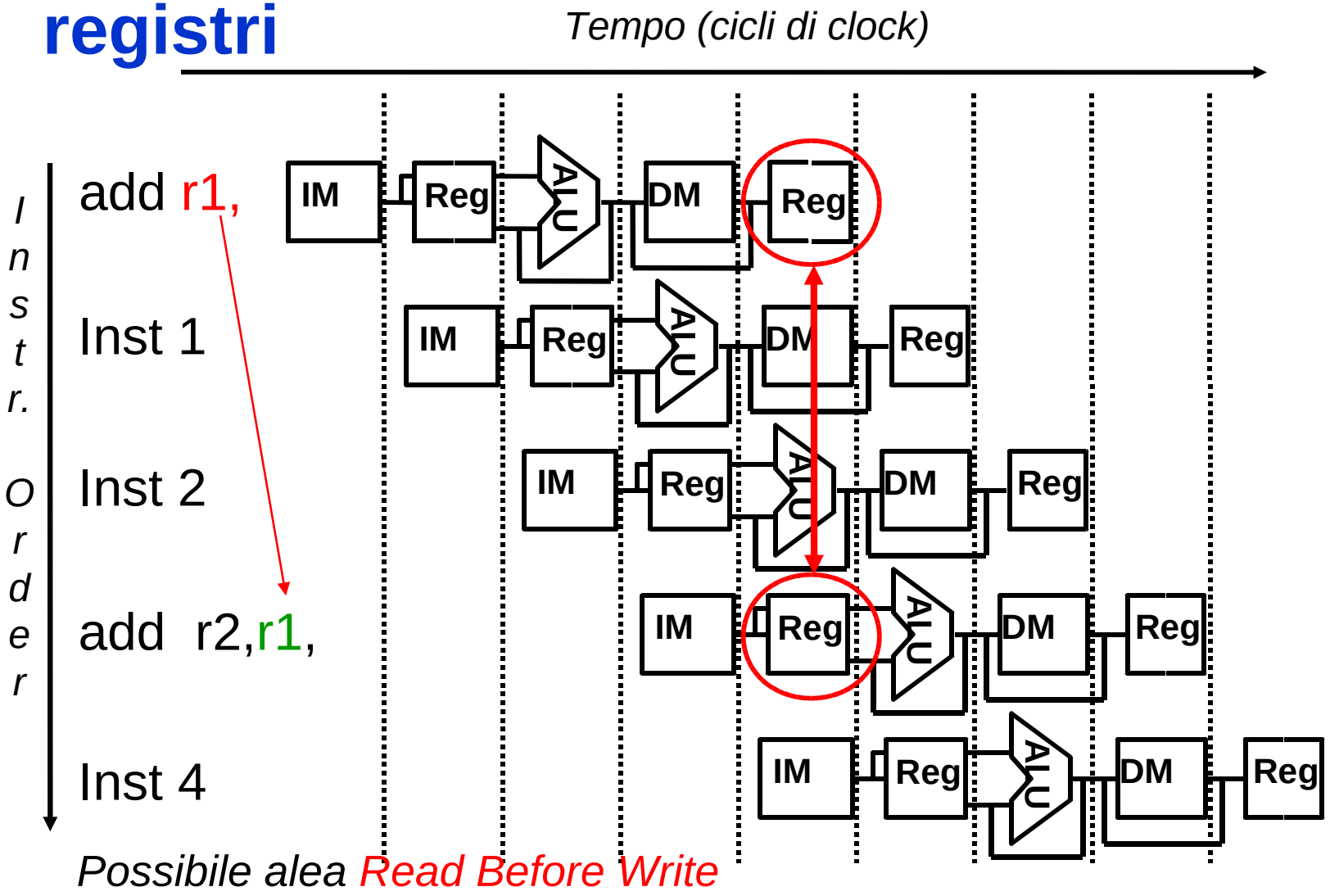
Alea strutturale 1: memoria unica



Alea strutturale 1: memoria unica

- Soluzione:
  - Creazione di una seconda memoria irrealizzabile e inefficiente
  - Ne simuliamo la presenza tramite una memoria cache separata in cache istruzioni e cache dati.

Alea strutturale 2: accesso ai registri

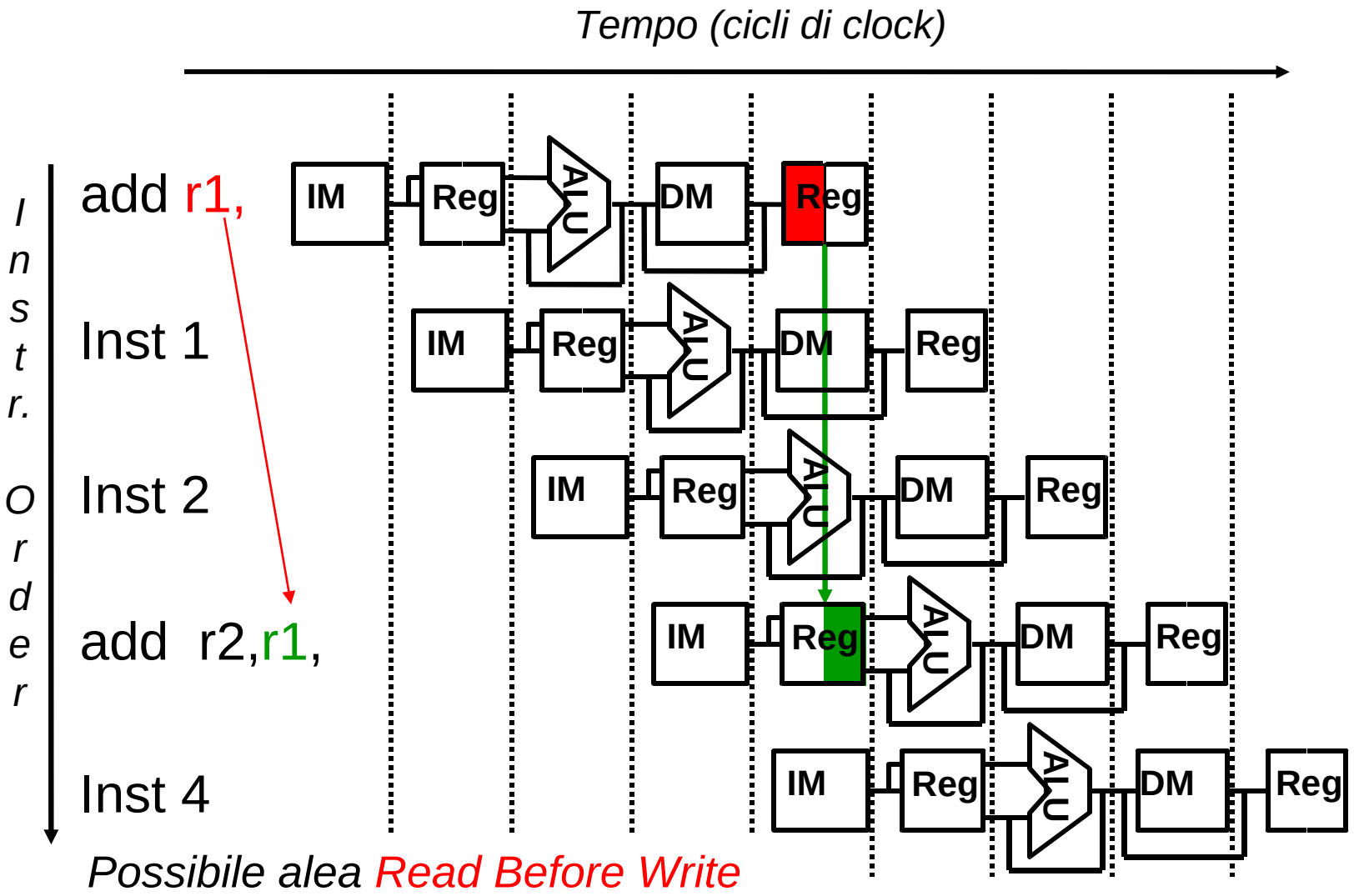


Possibile alea Read Before Write

Alea strutturale 2: accesso ai registri

- L'accesso ai registri è molto veloce: richiede meno di metà del tempo necessario ad un'operazione ALU
- Soluzione: si introduce la convenzione
  - Scrittura sui registri durante la prima metà del ciclo di clock
  - Lettura dai registri durante la seconda metà del ciclo di clock
  - Risultato : possibile realizzare senza problemi un Read ed un Write durante lo stesso ciclo di clock

Alea strutturale 2: accesso ai registri



Possibile alea Read Before Write

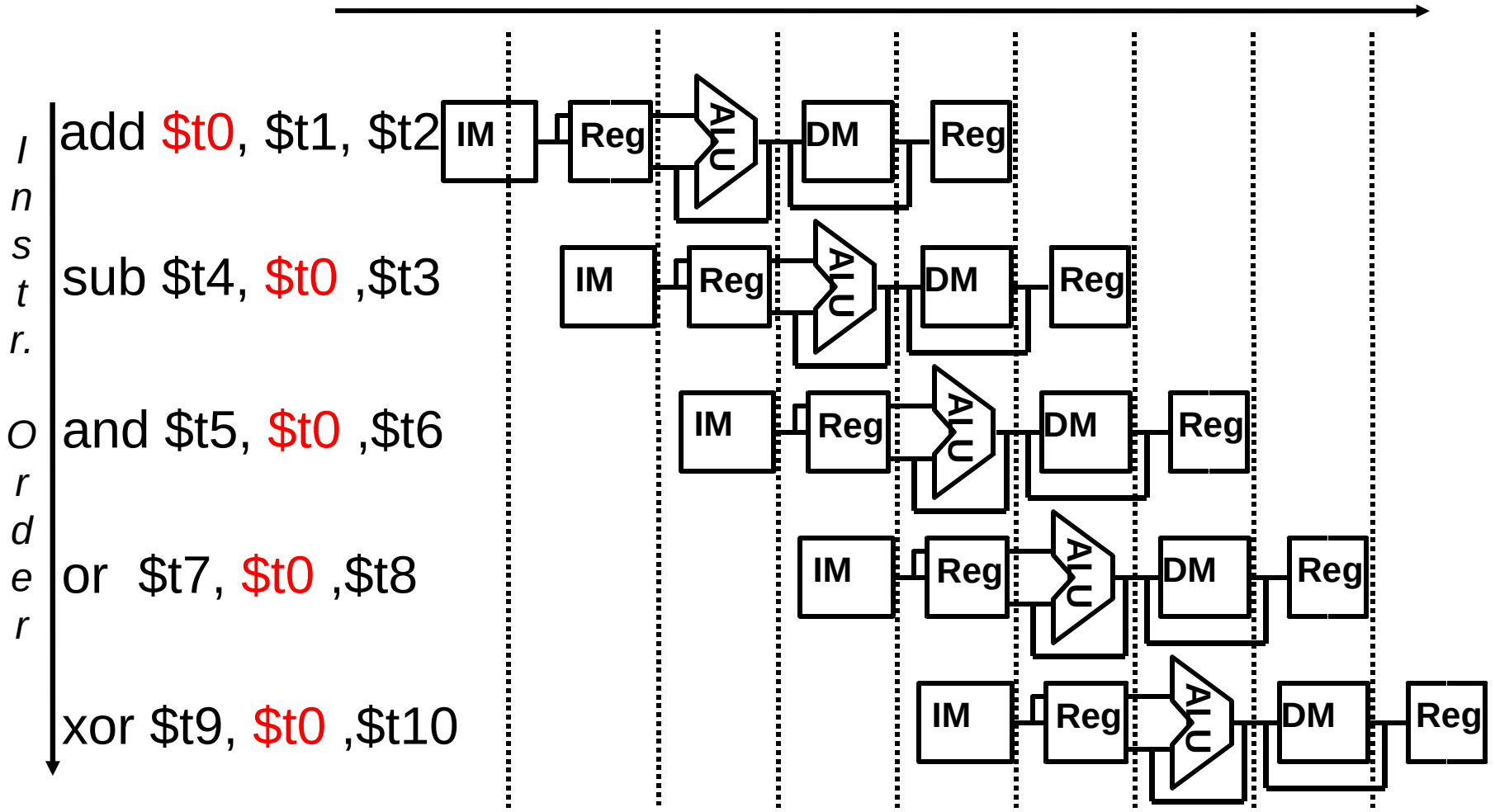
Alea sui dati

- Consideriamo la sequenza di istruzioni

```
add $t0, $t1, $t2
sub $t4, $t0 ,$t3
and $t5, $t0 ,$t6
or  $t7, $t0 ,$t8
xor $t9, $t0 ,$t10
```

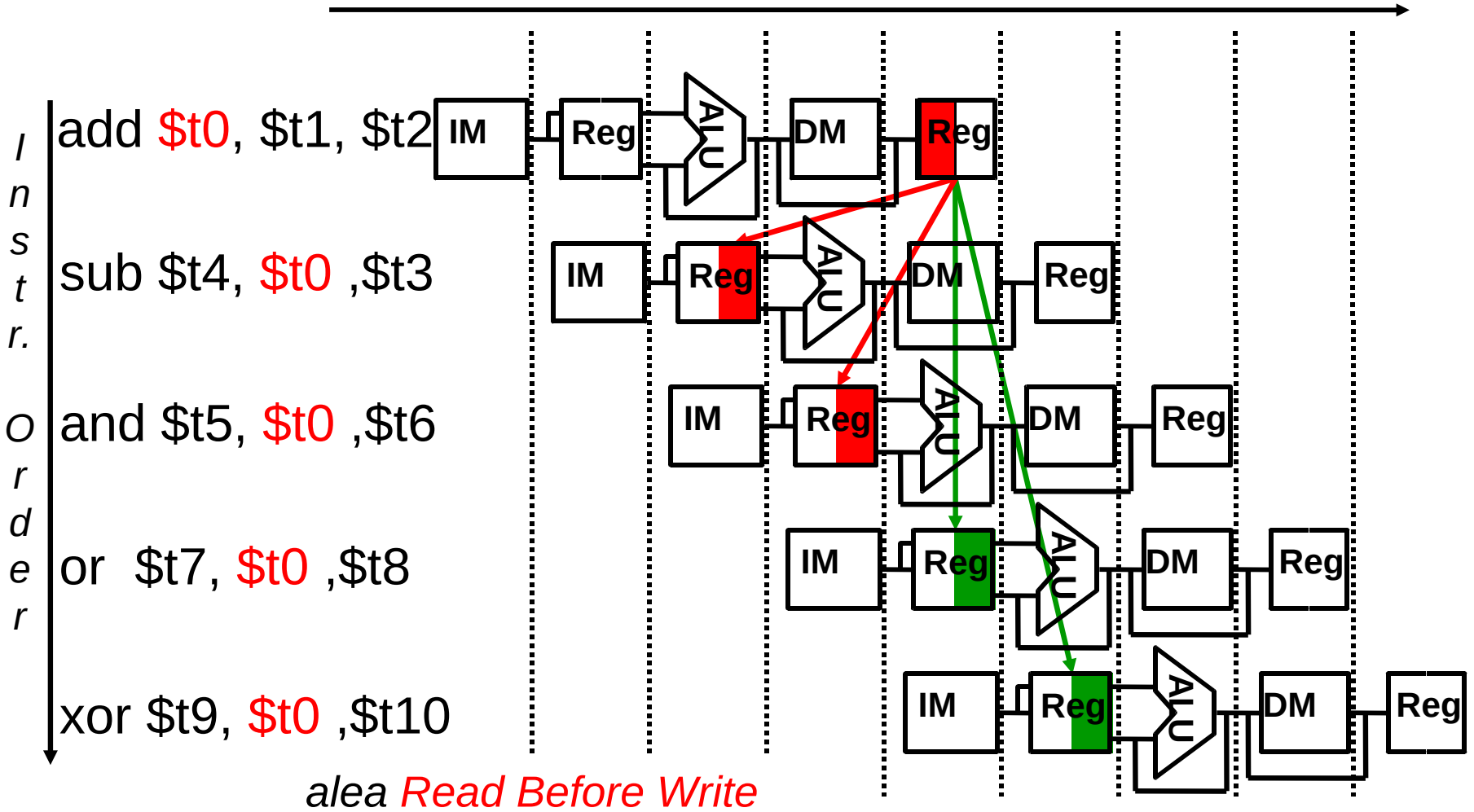
Alea sui dati

- L'influenza su fasi precedenti produce alee



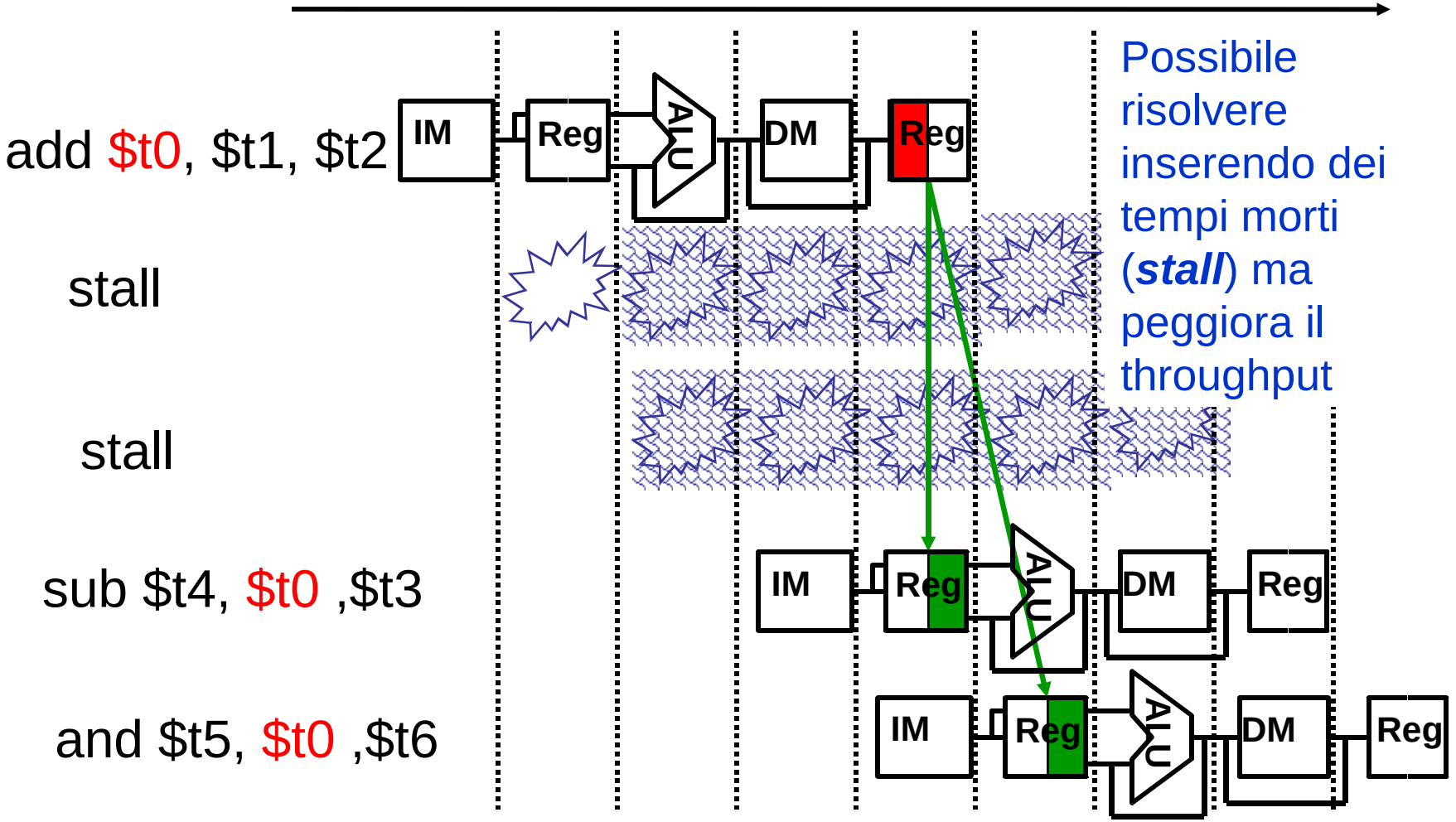
Alea sui dati

- L'influenza su fasi precedenti produce alee

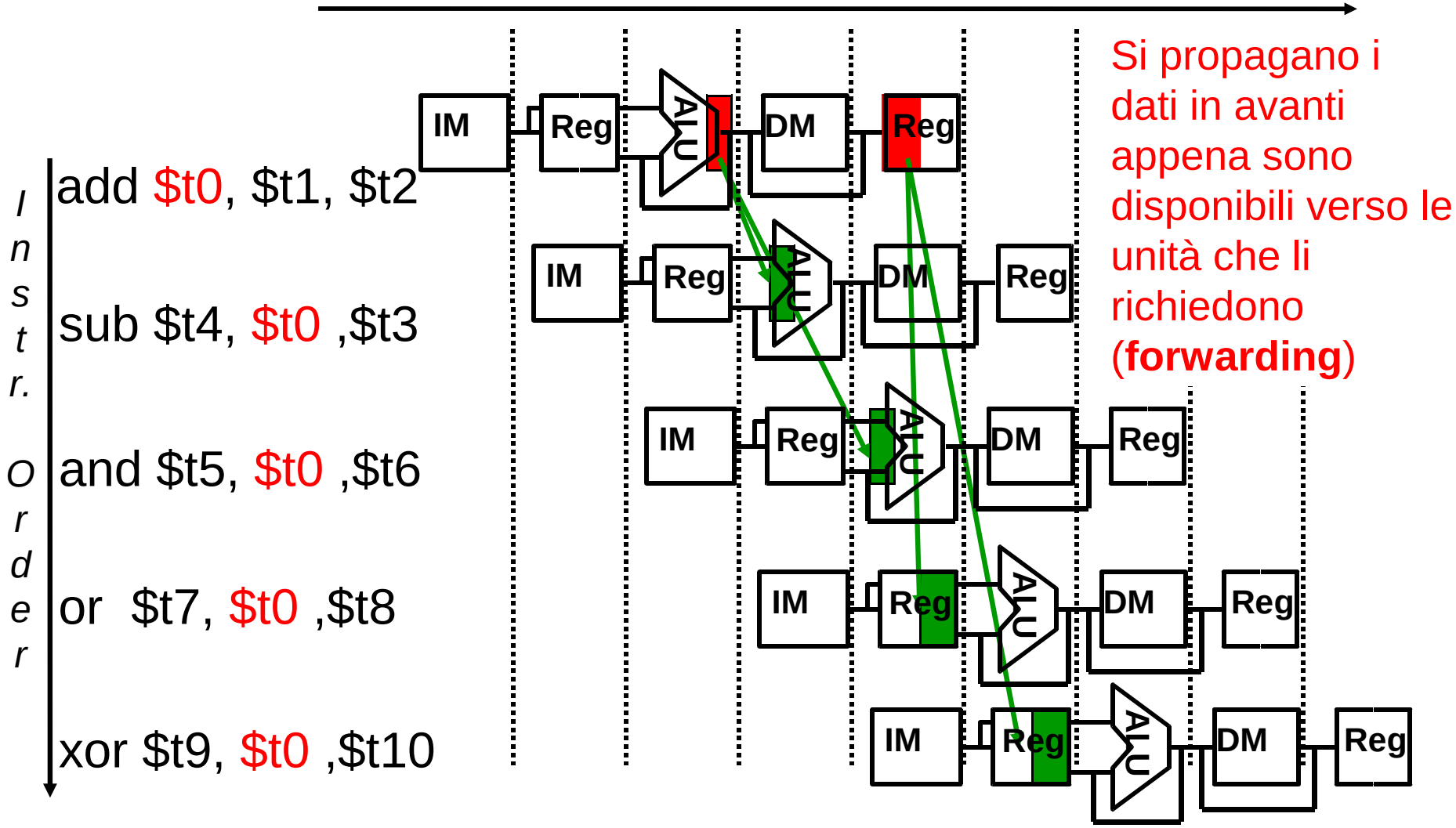


alea Read Before Write

Alea sui dati: soluzione 1

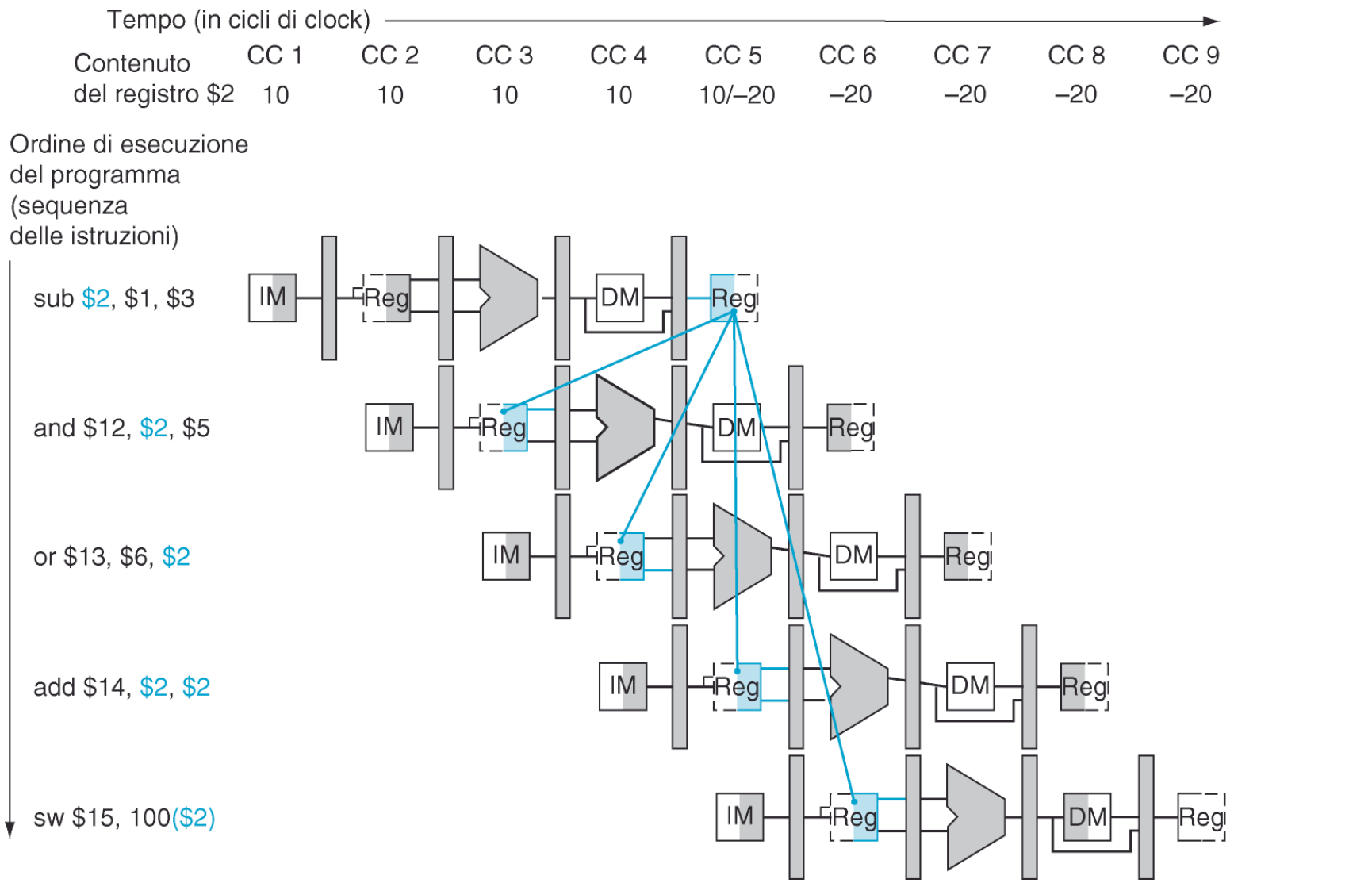


Alea sui dati: soluzione 2



Si propagano i dati in avanti appena sono disponibili verso le unità che li richiedono (**forwarding**)

Forwarding


Tempo (in cicli di clock)
Contenuto del registro $2
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
10 10 10 10 10/–20 –20 –20 –20 –20
Ordine di esecuzione del programma (sequenza delle istruzioni)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM

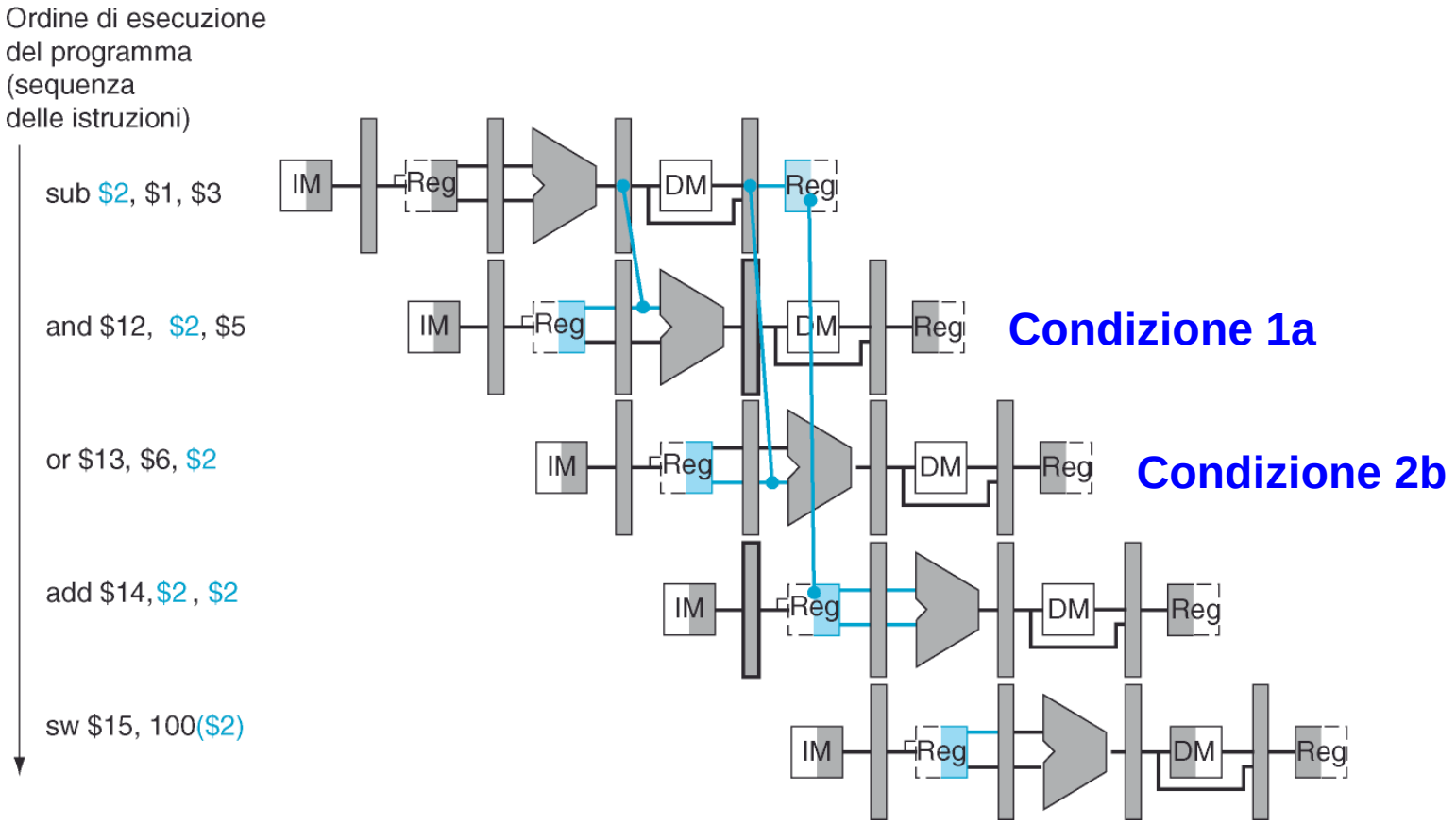
Forwarding

- Il risultato dell'istruzione sub è già disponibile al termine dello stadio EX (CC3).
- Le altre istruzioni hanno bisogno di quel dato negli stadi CC4 (and) e CC5 (or)
- Dove si trova il dato in quegli istanti ?
  - CC4: nel registro EX/MEM
  - CC5: nel registro MEM/WB
- Come è quindi possibile capire che è necessario il forwarding ?
  - 1a: EX/MEM.RegistroRd == ID/EX.Registro.Rs
  - 1b: EX/MEM.RegistroRd == ID/EX.Registro.Rt
  - 2a: MEM/WB.RegistroRd == ID/EX.Registro.Rs
  - 2b: MEM/WB.RegistroRd == ID/EX.Registro.Rt

Forwarding

Tempo (in cicli di clock) →

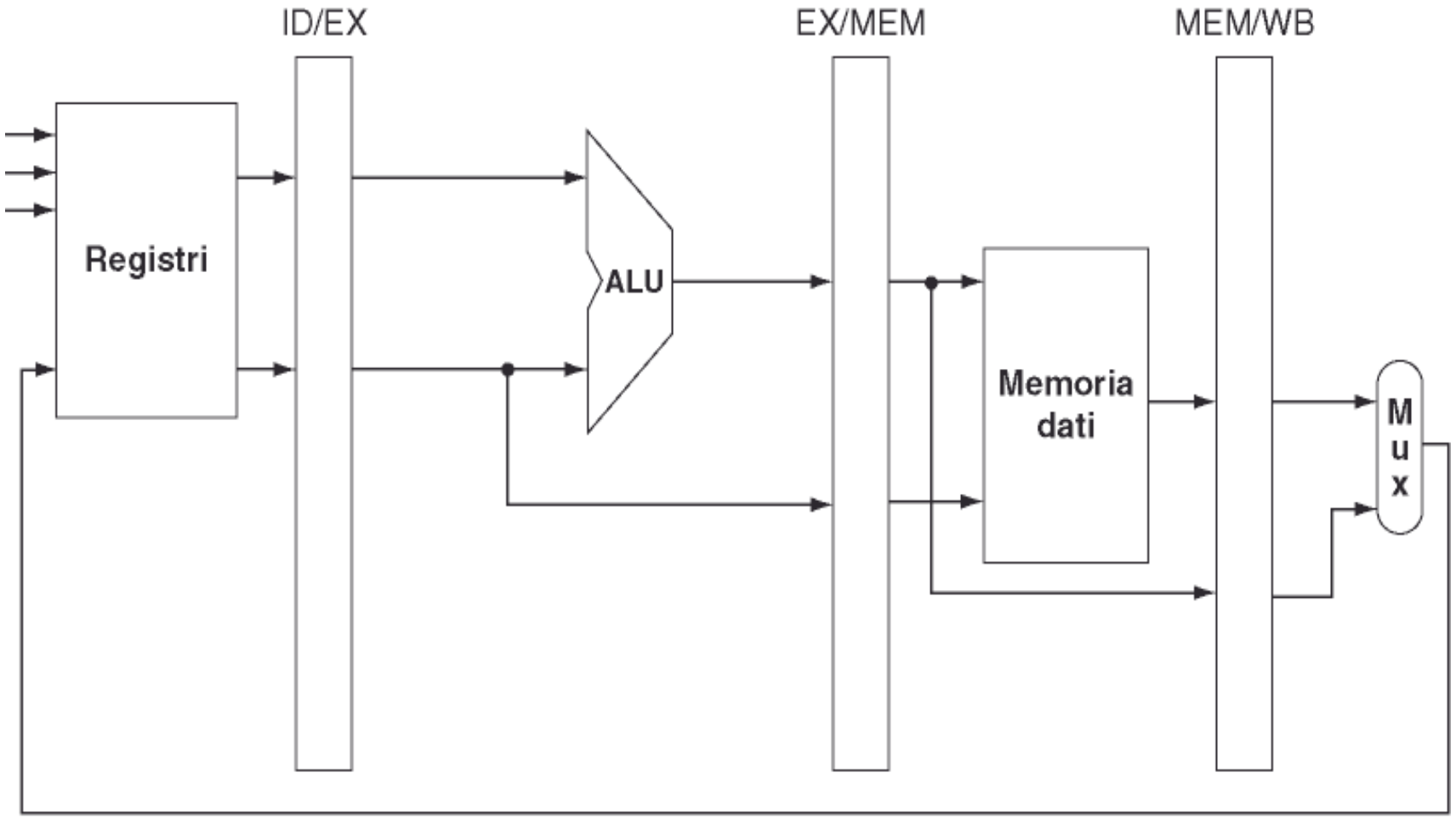
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Contenuto del registro $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Contenuto del registro EX/MEM:	X	X	X	–20	X	X	X	X	X
Contenuto del registro MEM/WB:	X	X	X	X	–20	X	X	X	X



Condizione 1a

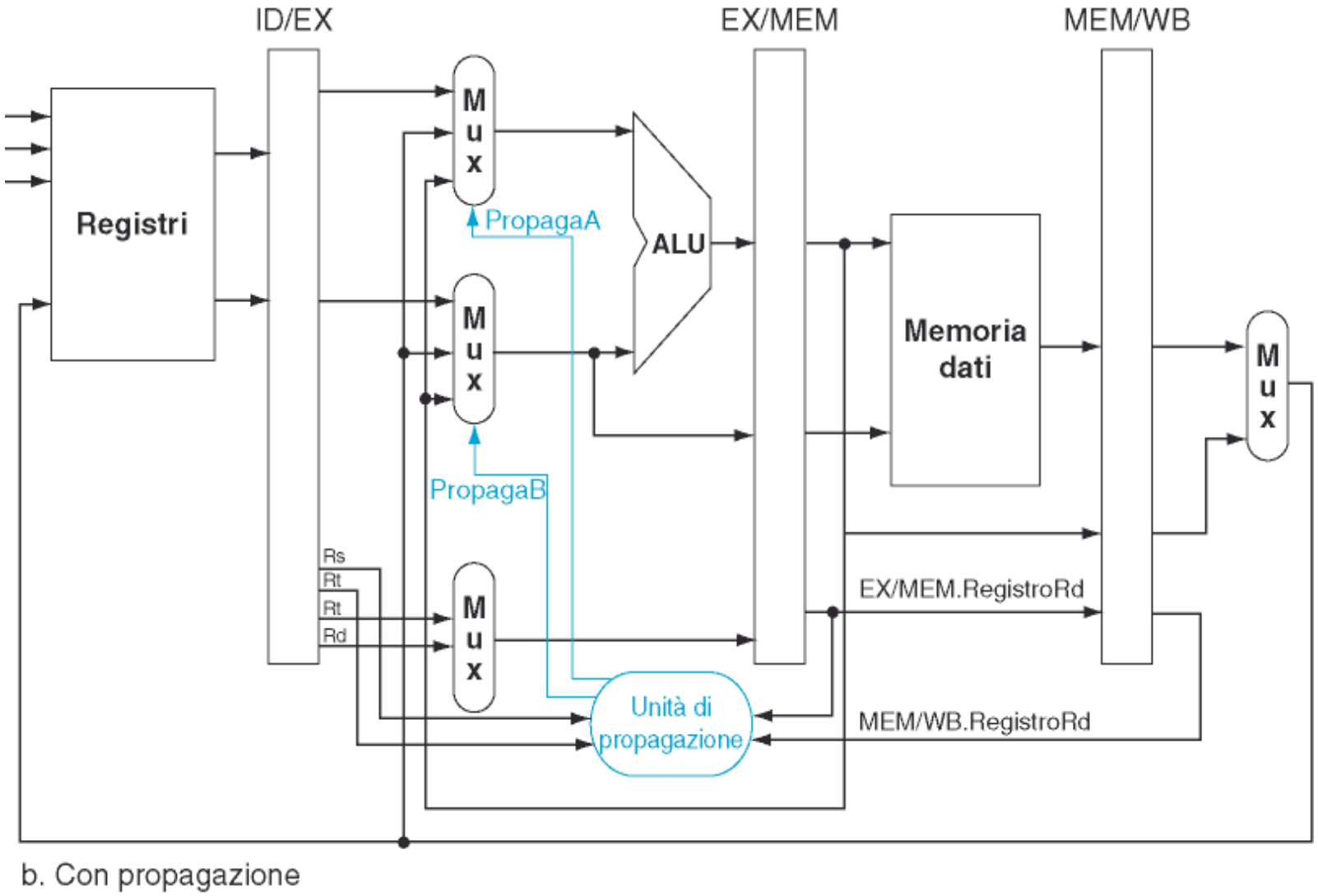
Condizione 2b

Forwarding: come si modifica il Datapath ?



a. Nessuna propagazione

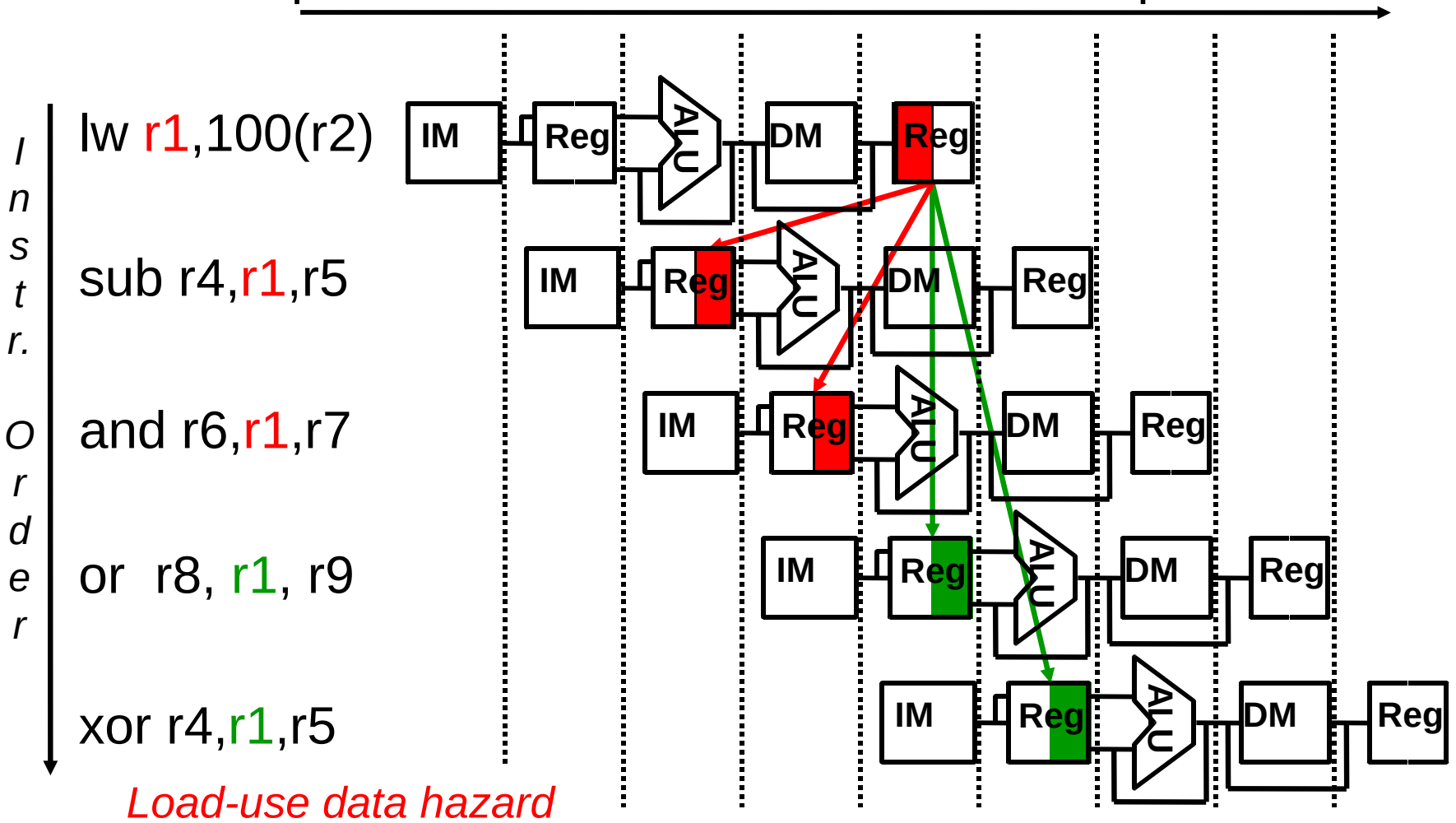
Forwarding: come si modifica il Datapath ?



b. Con propagazione

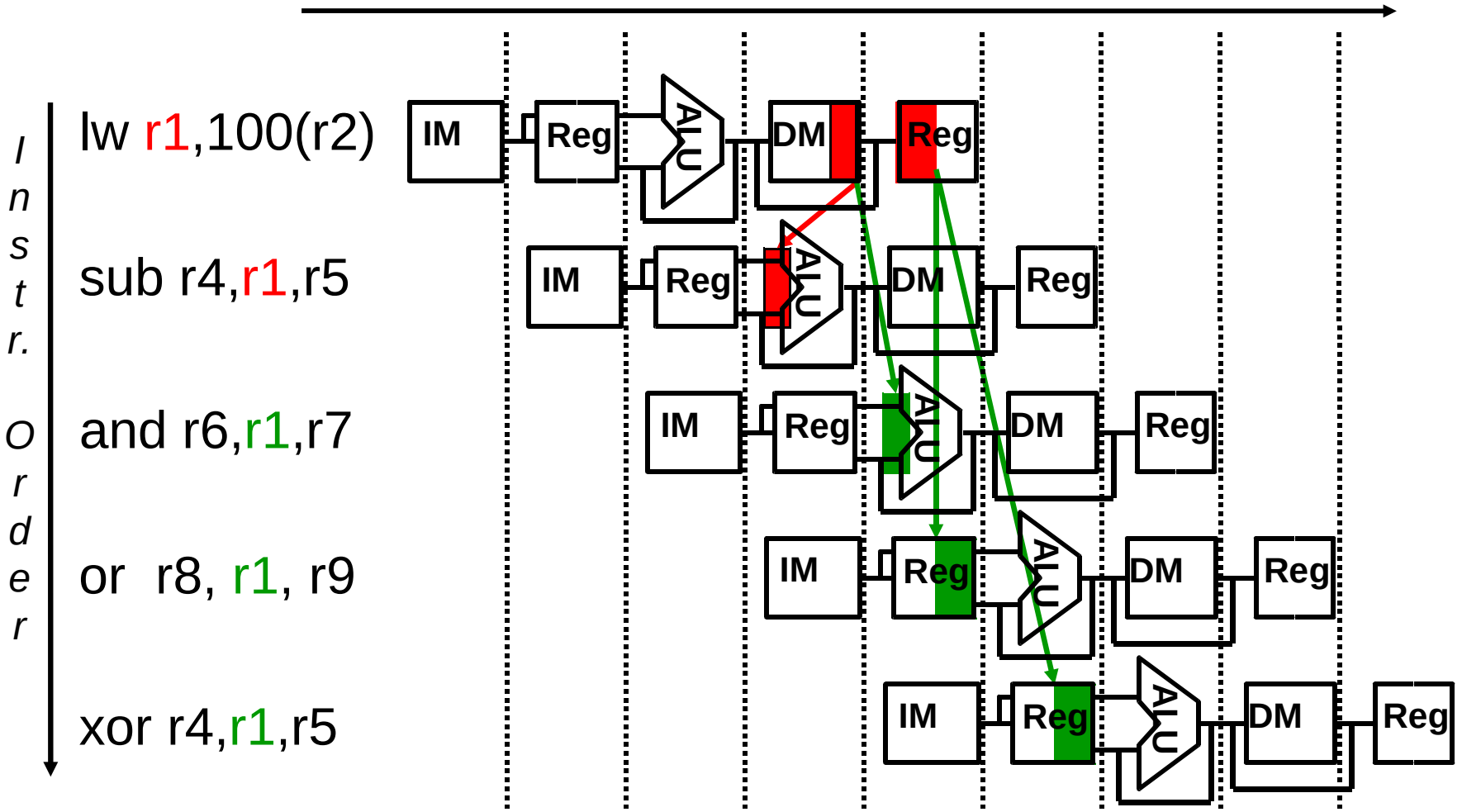
Alea sui dati da Load

Simile al precedente, ma con una difficoltà in più



Load-use data hazard

Alea sui dati da Load: forwarding



- Forwarding insufficiente. Necessario uno stall ?

Alea sui dati da Load

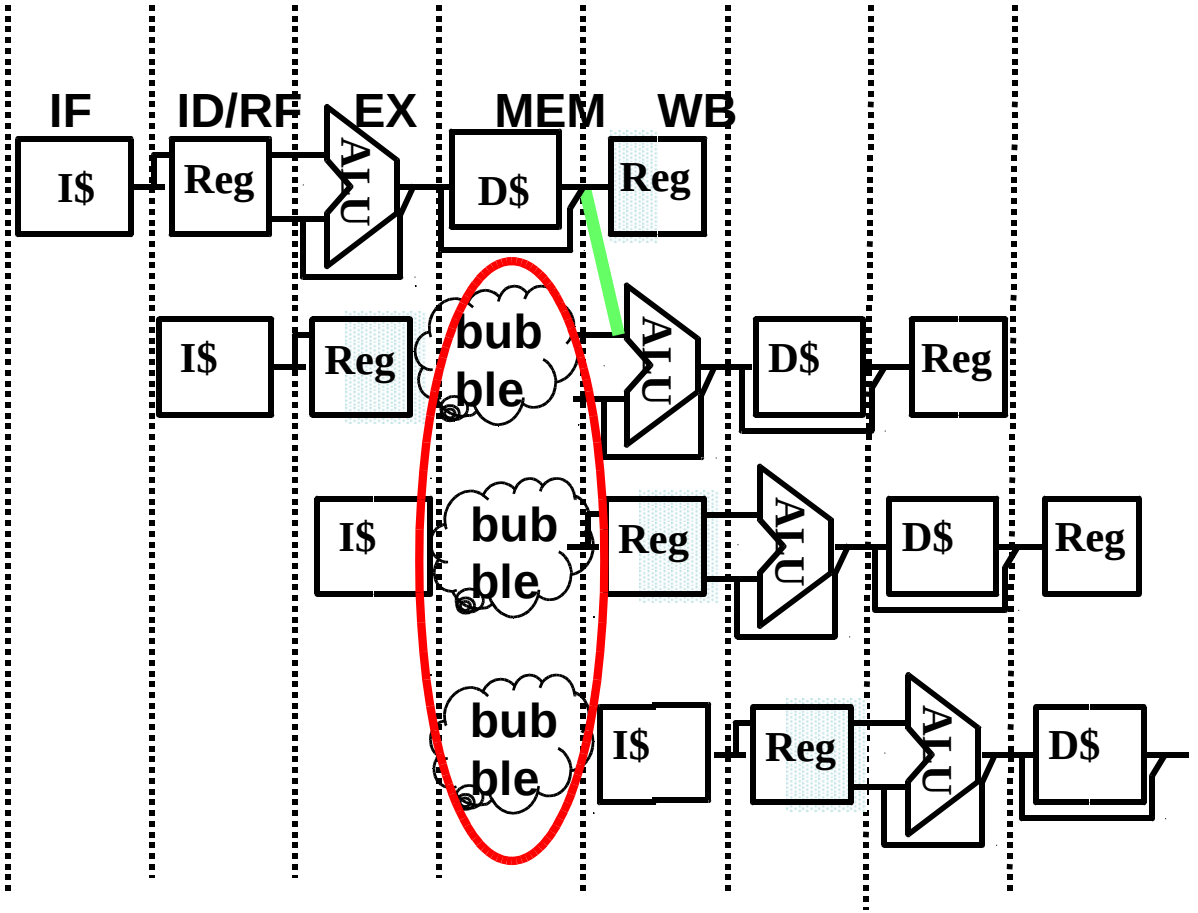
- Il controllo deve bloccare il pipeline
- interlock

lw $t0, 0($t1)

sub $t3,$t0,$t2

and $t5,$t0,$t4

or $t7,$t0,$t6



Alea sui dati da Load

- Lo spazio per l'istruzione dopo un load è chiamato "load delay slot"
- Se l'istruzione usa il risultato del load, allora il controllo dovrà forzare uno stall per un ciclo.
- Non necessario se il traduttore inserisce nello slot un'istruzione non dipendente dal load (reordering) → traduttore cosciente del pipelining
- Anche se questo non fosse possibile, si può risolvere ugualmente il problema inserendo un NOP nello slot
  - Chi inserisce il NOP ?

Alea sui dati da Load

- Lo stall è equivalente ad un NOP

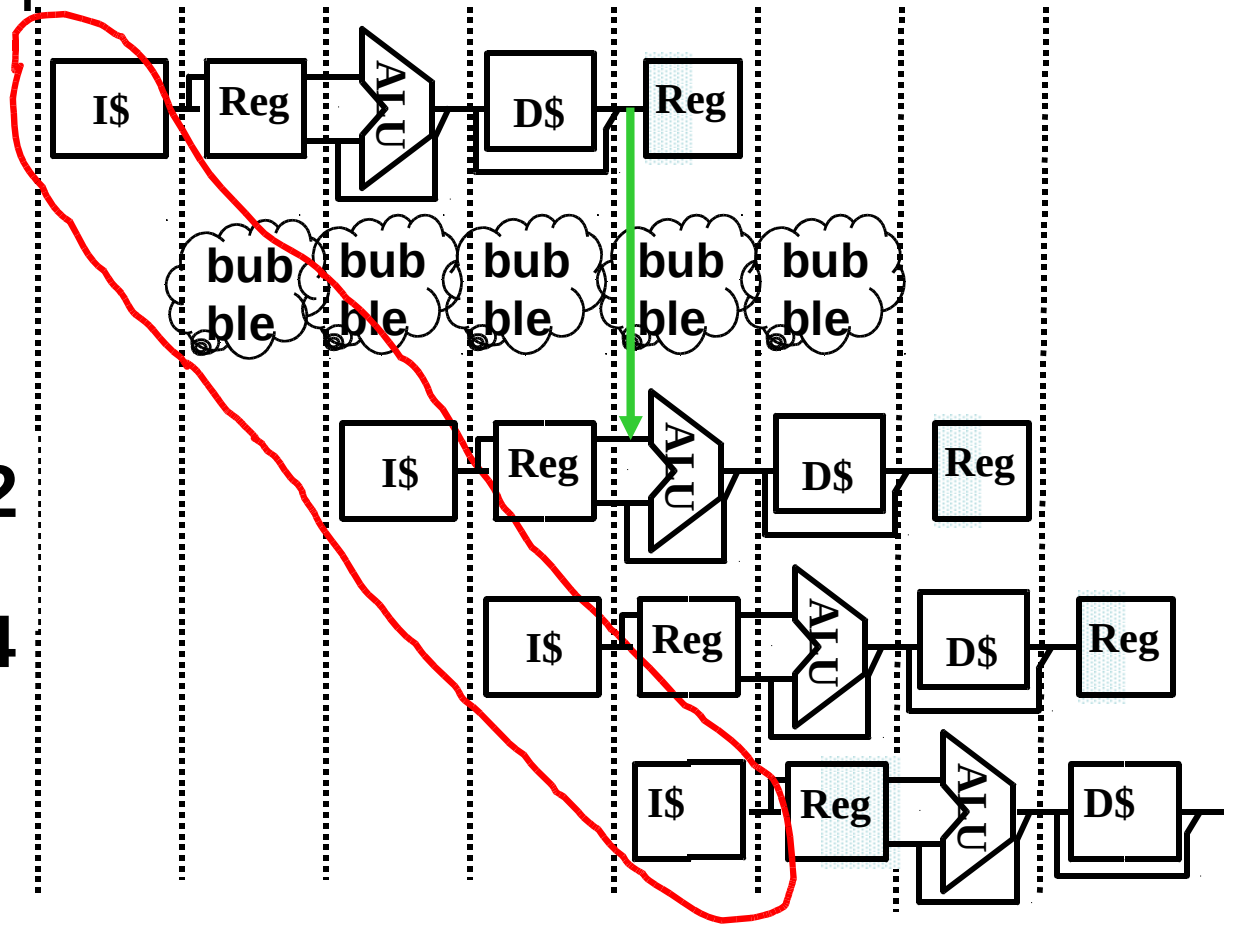
lw $t0, 0($t1)

nop

sub $t3,$t0,$t2

and $t5,$t0,$t4

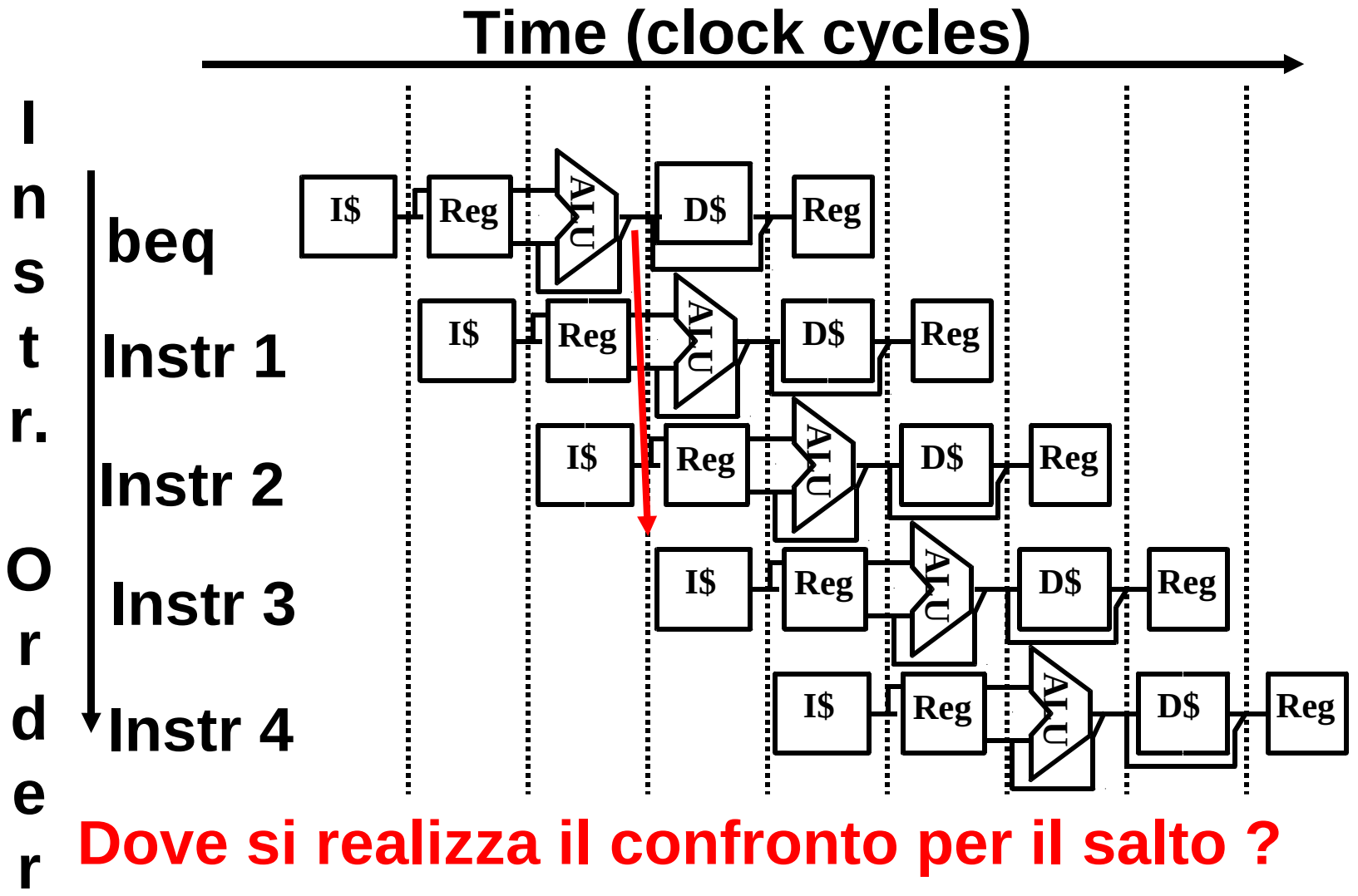
or $t7,$t0,$t6



Nota storica

- Il MIPS è stato il primo progetto di processore a non risolvere il load-use data hazard con interlock e stall
- Acronimo reale per il MIPS:
  Microprocessor without
  Interlocked
  Pipeline
  Stages

Alea di controllo: salto



Dove si realizza il confronto per il salto ?

Alea di controllo: salto

- L'unità per il confronto e la decisione è nel terzo stage (ALU stage)
  - Quindi altre due istruzioni dopo il beq entreranno nel pipeline, quale che sia il risultato del confronto

- Comportamento (desiderato) del salto
  - Se il confronto fallisce continua la normale esecuzione
  - Se il confronto è verificato, non eseguire altre istruzioni successive al salto e vai all'indirizzo specificato

Alea di controllo: salto. Soluzione 1

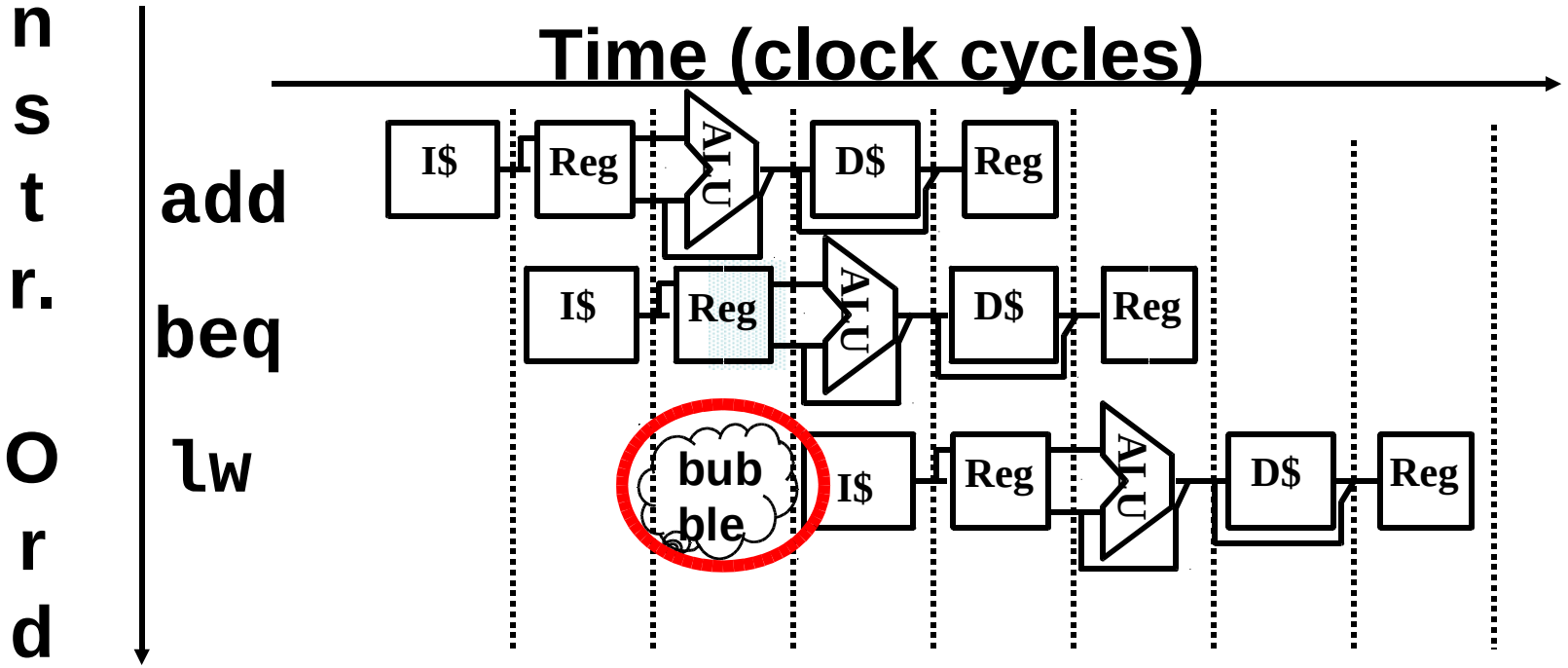
- Blocco del pipeline finchè il confronto non sia stato realizzato
  - Inserimento di NOP
  - Conseguenza: i salti richiedono 3 cicli di clock (assumendo che il confronto si faccia nel 3° stage)

Alea di controllo: salto. Soluzione 2.1

- Soluzione 2.1:
  - Anticipazione del confronto al passo 2
  - Appena l'istruzione è decodificata, si può decidere immediatamente e modificare il PC (se necessario)
  - In questo modo entra solo un'istruzione nel pipeline
    → necessario solo un NOP

Alea di controllo: salto. Soluzione 2.1

- Inserimento di una bolla



- 2 cicli per branch ⇒ ancora lento

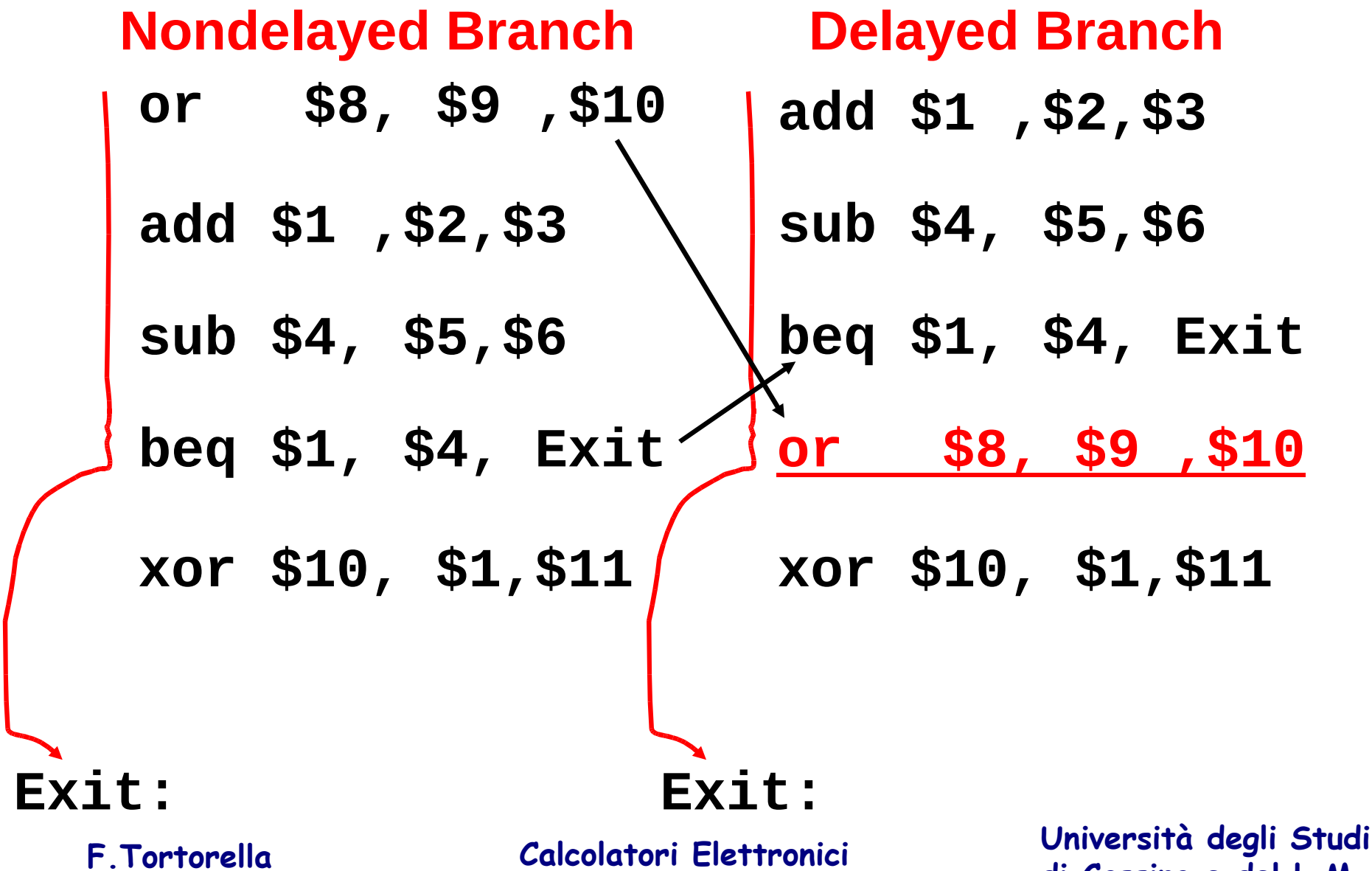
Alea di controllo: salto. Soluzione 2.2

- Soluzione 2.2 (ridefinizione del salto)
  - Vecchia definizione: in caso di salto, nessuna istruzione successiva al branch deve essere eseguita
  - Nuova definizione: in ogni caso, viene eseguita l'istruzione che segue immediatamente il branch (definita branch-delay slot)
- Questo approccio viene definito "Delayed Branch"

Note sul Branch-Delay Slot

- Caso peggiore
  - Inserimento di un NOP nello slot
- Caso migliore
  - È possibile trovare un'istruzione precedente al branch che possa essere posticipata al branch senza alterare il flusso di controllo (e dei dati)
  - Dipende dalle capacità del traduttore
- Anche i jump hanno un delay slot

Esempio: Nondelayed vs. Delayed Branch



Riassumendo ...

- Il pipelining migliora le prestazioni incrementando il throughput delle istruzioni: sfrutta l'Instruction Level Parallelism
  - Esegue più istruzioni in parallelo
  - Ogni istruzione ha la stessa latenza
- E' soggetto ad alee
  - Strutturali, di dati, di controllo
- Gli stalli riducono le prestazioni
  - Ma sono necessari per ottenere risultati corretti
- Il compilatore può riorganizzare il codice per evitare alee e stalli
  - Richiede conoscenza dell'architettura pipelined

Come aumentare l'ILP ?

- Esecuzione parallela (*Multiple Issue*) superscalare
  - Replica degli stadi di pipeline (pipeline multiple)
  - Si avviano più istruzioni nel ciclo di clock
- Parallelizzazione statica dell'esecuzione (*Static MI*)
  - Il compilatore raggruppa le istruzioni da eseguire insieme
  - Le impacchetta in insiemi di esecuzione (*issue slots*)
  - Individua ed evita le alee
- Parallelizzazione dinamica dell'esecuzione (*Dynam. MI*)
  - La CPU esamina il flusso di istruzioni e sceglie quali eseguire in ogni ciclo
  - Il compilatore può aiutare nel riordinare le istruzioni
  - La CPU risolve le alee utilizzando opportune tecniche a runtime.
  -

Parallelizzazione statica

- Il compilatore raggruppa le istruzioni in *pacchetti di esecuzione*
  - Gruppi di istruzioni che possono partire nello stesso ciclo
  - Determinati dalle risorse richieste
- Ogni pacchetto può essere visto come un'istruzione molto lunga che specifica molte operazioni contemporanee
  - **VLIW**: Very Long Instruction Word

Parallelizzazione statica

- Il compilatore deve impedire alee (se possibile)
- Riordina le istruzioni nei pacchetti
- Nessuna dipendenza all'interno del pacchetto
- Possibili dipendenze tra pacchetti
- Quando inevitabile, inserimento di NOP

Parallelizzazione dinamica

- Processori “superscalari”
- La CPU decide quante e quali istruzioni avviare in un ciclo (0, 1, 2, …)
    - Da evitare alee strutturali e di dati
- Non necessario lo scheduling da parte del compilatore
    - Anche se può aiutare …
    - La CPU deve garantire che l’esecuzione sia fedele a quanto definito nel programma
- Possibile l’esecuzione *fuori ordine* (out of order) delle istruzioni per evitare stalli (es. dovuti a lw)

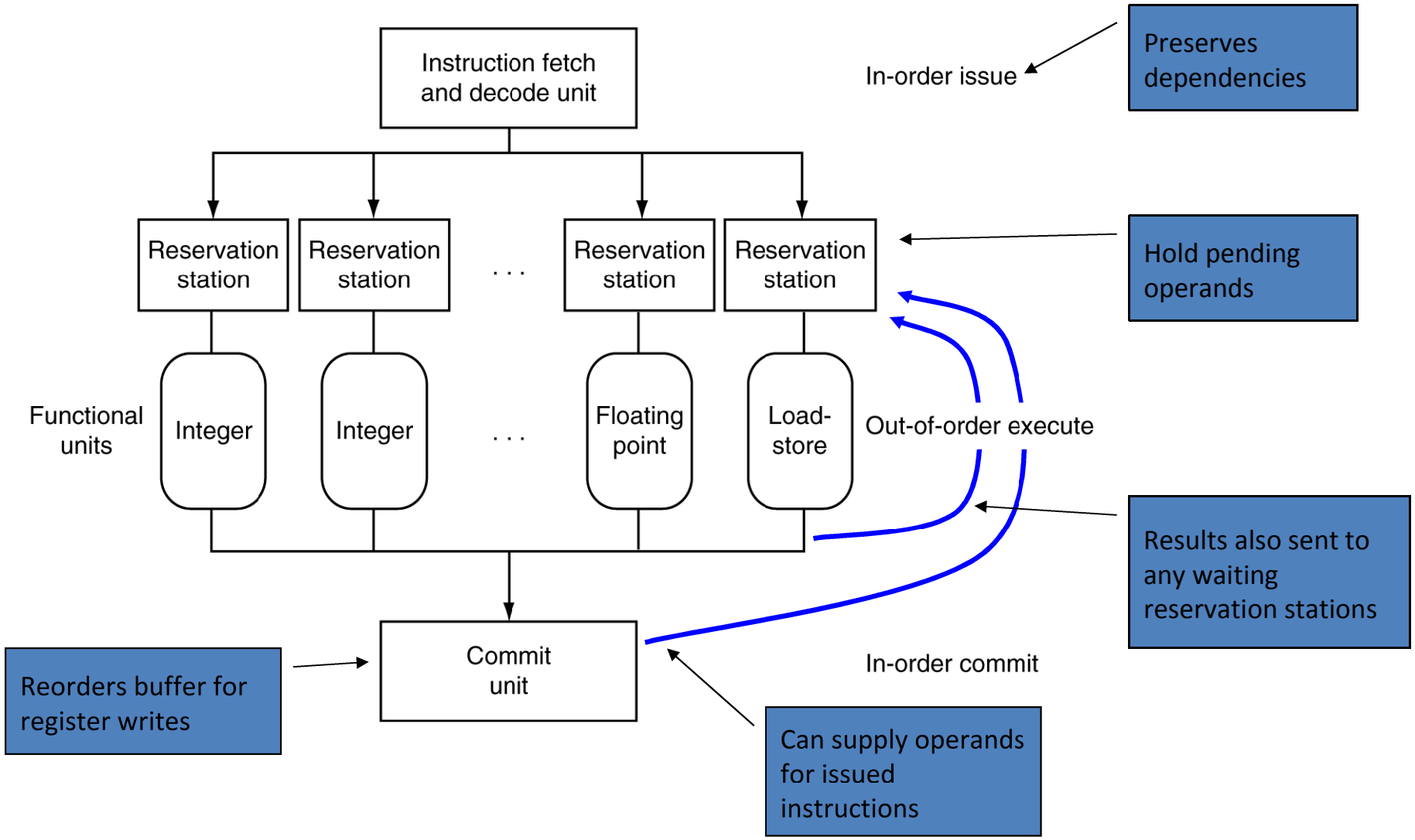
Speculazione (non economica …)

- Permette di “indovinare” le caratteristiche di un’istruzione
  - Inizia l’operazione appena possibile
  - Verifica se la previsione è stata corretta
  - Se OK, completa l’operazione; altrimenti trona indietro
- Esempi: branch prediction
- E’ comune alla parallelizzazione statica e dinamica.

Organizzazione di una CPU superscalare

- La pipeline è divisa in tre unità principali
- Una unità per l'instruction fetch e avvio dell'esecuzione
  - Istruzioni avviate *in ordine*
- Un gruppo di più unità funzionali in parallelo
  - Ogni unità ha una Reservation Station che mantiene gli operandi e memorizza l'operazione
  - Può eseguire istruzioni fuori ordine
- Una unità di consegna
  - Preleva i risultati dalle unità funzionali e li salva nei Reorder Buffers
  - Salva i risultati garantendo una consegna in ordine

Organizzazione di una CPU superscalare



Esempi

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
Core 2 Yorkfield	2008	2930 MHz	16	4	Yes	4	95W
Core i7 Gulftown	2010	3460 MHz	16	4	Yes	6	130W