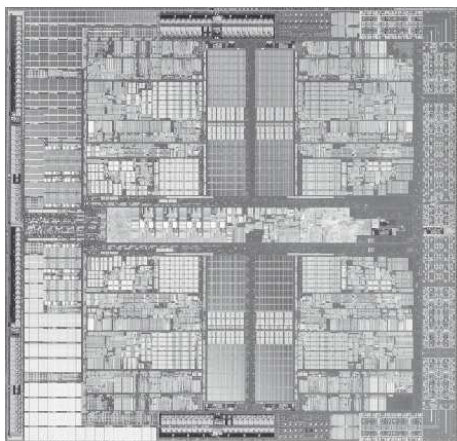




**Università degli Studi  
di Cassino**



**Corso di  
Calcolatori Elettronici  
Pipeline**

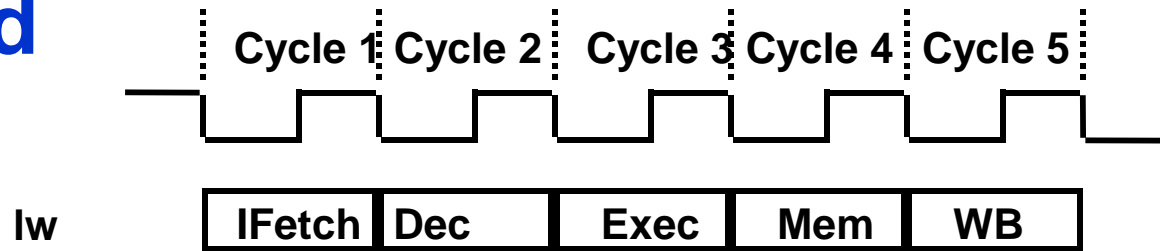
**Anno Accademico 2010/2011  
Francesco Tortorella**

# 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

# I 5 passi dell'istruzione

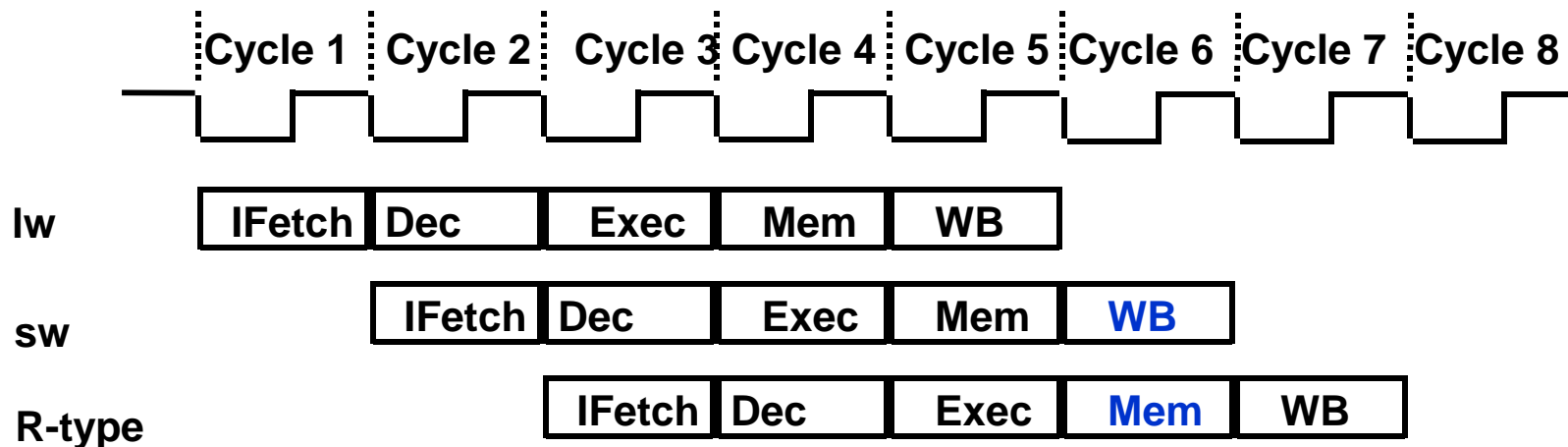
## Load



- **IFetch**: Instruction Fetch e aggiornamento PC
- **Dec**: Registers Fetch e decodifica istruzione
- **Exec**: Esecuzione(R-type) ; calcolo dell'indirizzo di memoria
- **Mem**: Read/write dei dati da/verso la memoria dati
- **WB**: Scrittura del risultato nel register file

# Processore MIPS Pipelined

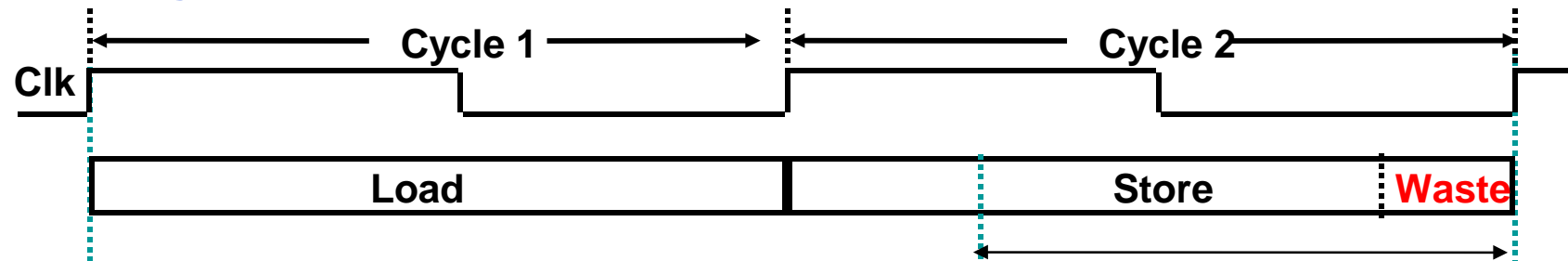
- Avvia l'istruzione successiva mentre la corrente è ancora in esecuzione
  - Migliora il **throughput** - totale di lavoro eseguito per unità di tempo (numero medio di istruzioni per secondo o per ciclo di clock)
  - Non riduce la **latenza** (tempo trascorso dall'inizio di un'istruzione fino al completamento)



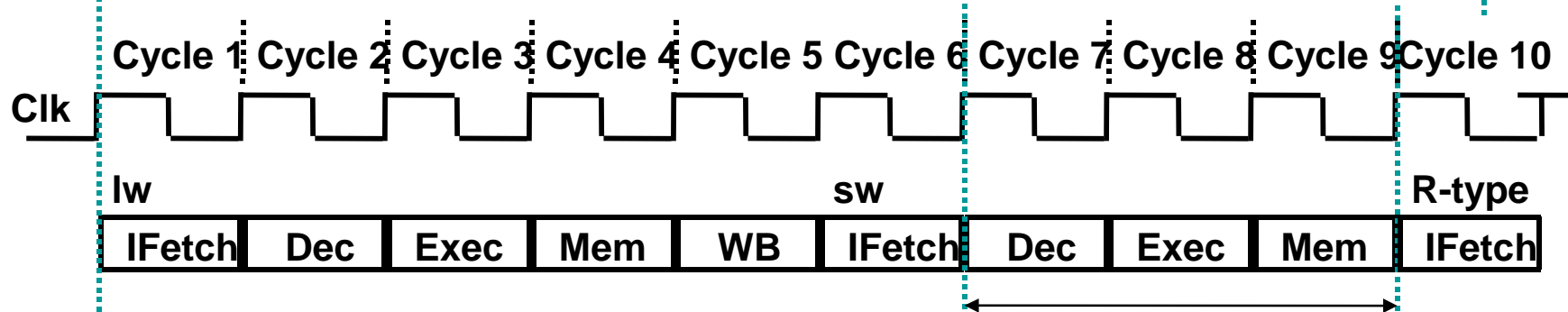
- Il ciclo di clock del pipeline è limitato dal passo più lento
- Per alcune istruzioni, alcuni passi sono cicli sprecati

# Confronto tra le implementazioni

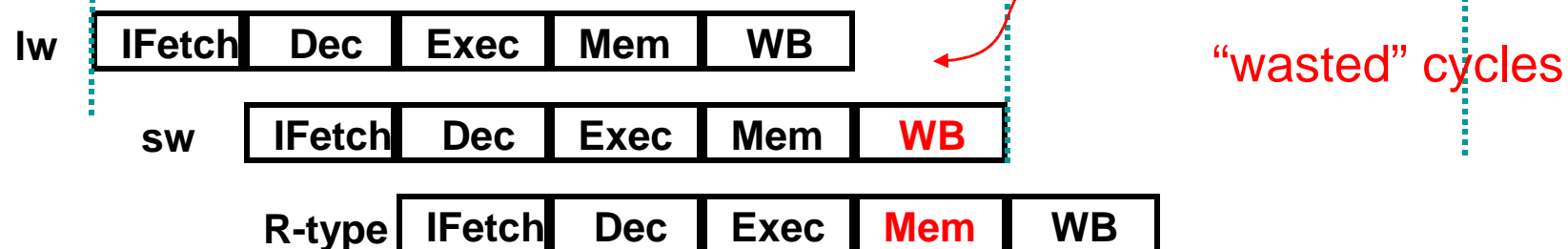
Ciclo singolo:



Multiciclo:

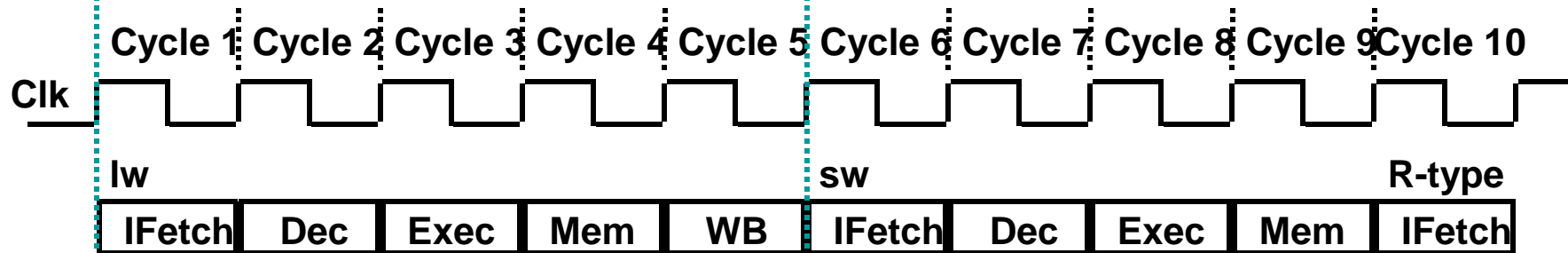


Pipeline:

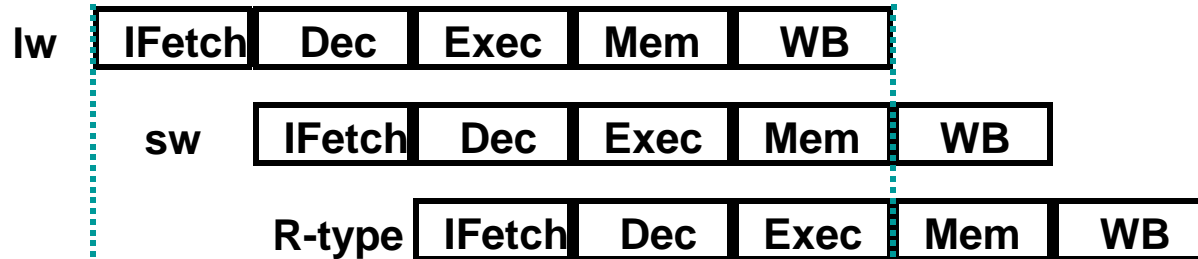


# Throughput e Latenza

Muticiclo:



Pipeline:

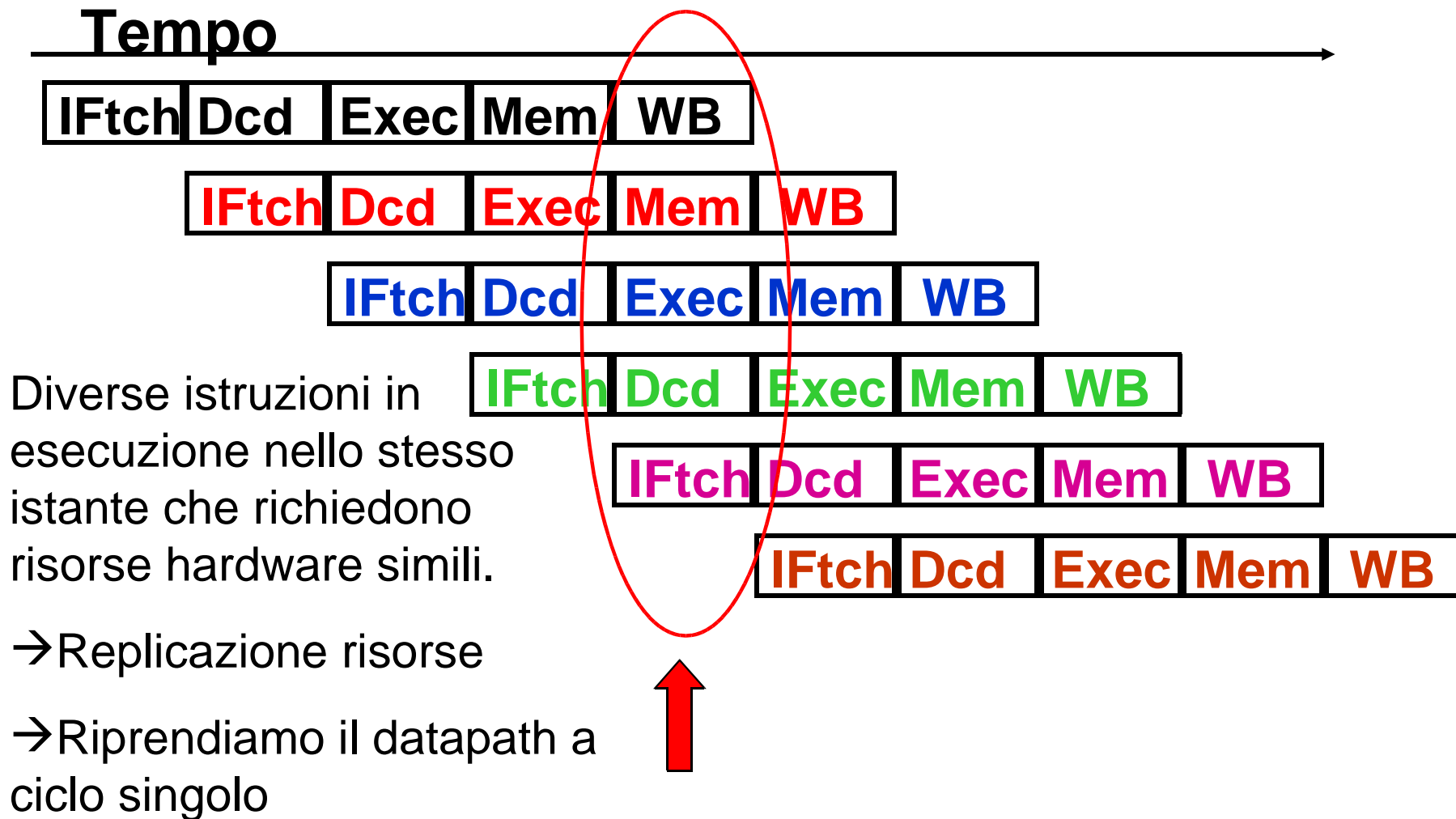


- **Latenza** 5 cicli di clock per entrambi
- **Throughput** 1 per ciclo di clock (IPC) per la pipeline  
1/5 IPC per il multiciclo

# Realizzazione in Pipelining del MIPS

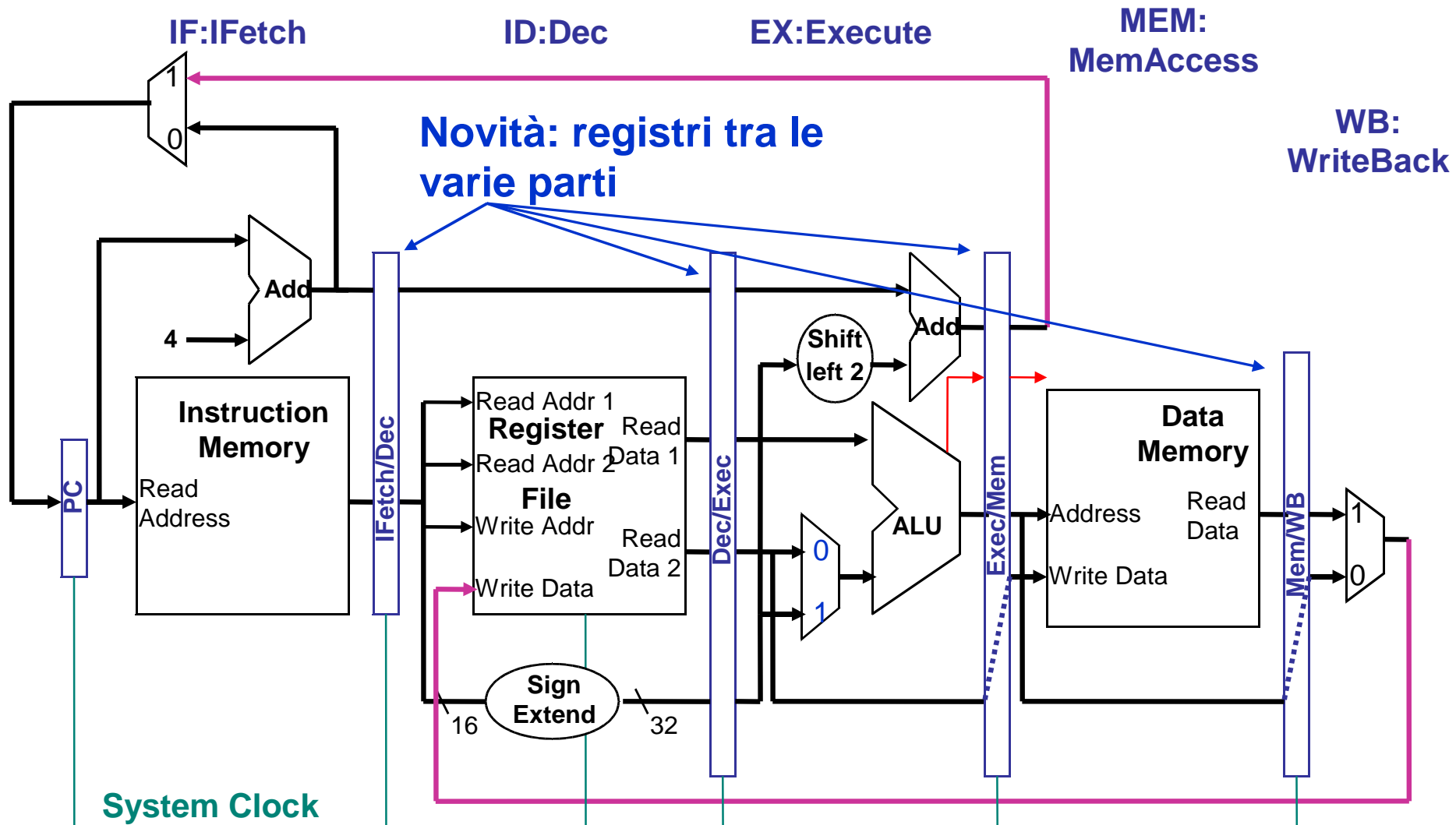
- **Aspetti dell'ISA MIPS che favoriscono il pipelining:**
  - Tutte le istruzioni hanno la stessa lunghezza (32 bit)
    - Più semplice il fetch nel 1° passo e il decode nel 2° passo
  - Pochi formati di istruzione (3) con **simmetria** tra i formati
    - Possibile cominciare la lettura del register 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
- **Possibili problemi**
  - **Alee strutturali:** che cosa succede se è presente un'unica memoria ?
  - **Alee di controllo:** che cosa succede in corrispondenza di salti ?
  - **Alee sui dati:** che cosa succede se gli operandi sorgente di un'istruzione dipendono dal risultato di un'istruzione precedente ?

# Quale datapath ?





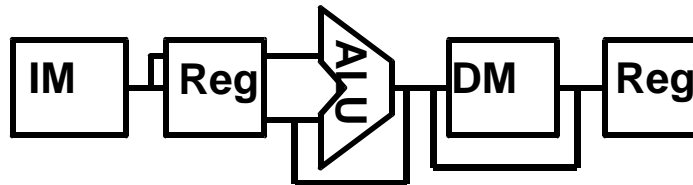
# Modifiche al datapath per il Pipeline



# Datapath per il pipeline

- 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)

# Rappresentazione grafica del pipeline MIPS

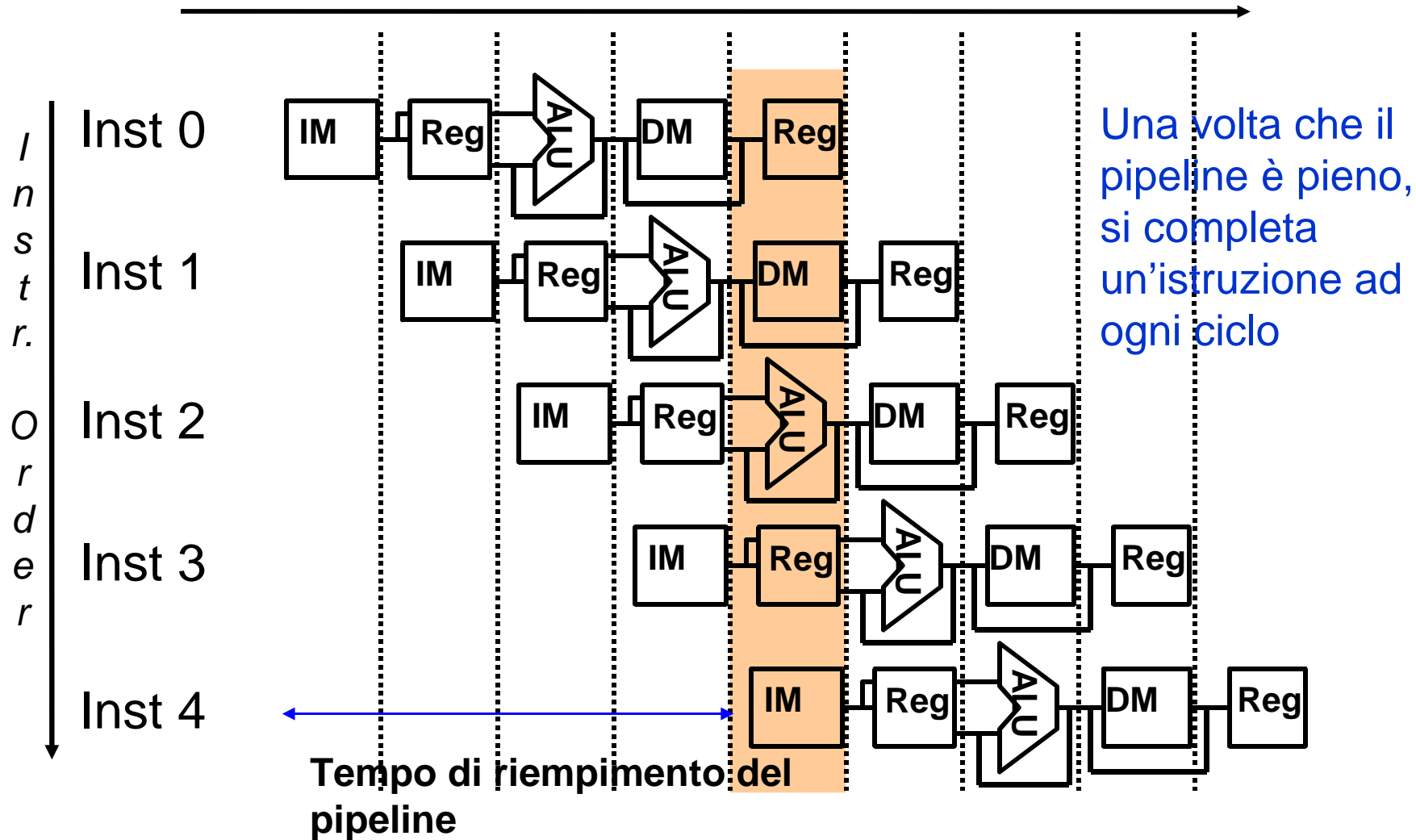


E' di aiuto per rispondere a domande quali:

- Quanti cicli sono necessari per eseguire un certo codice ?
- Che cosa sta facendo l'ALU durante il ciclo 4?
- C'è un'alea ? Come può essere risolta ?

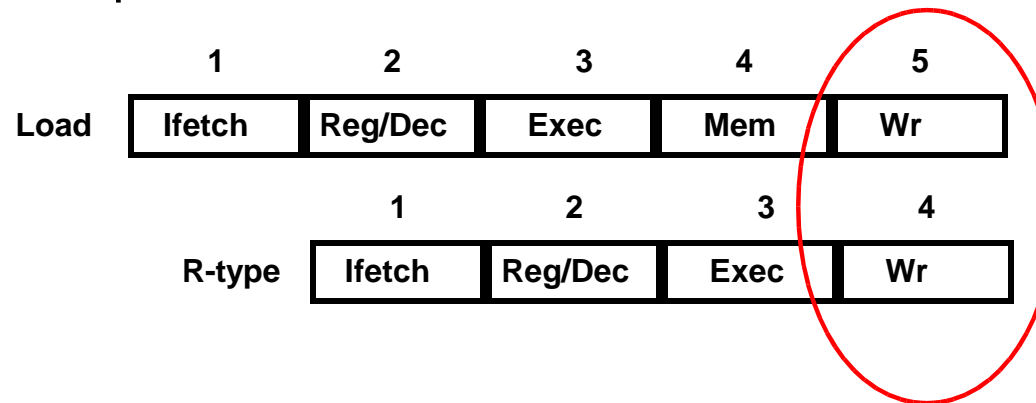
# Il pipeline ottimizza il throughput

*Tempo (cicli di clock)*

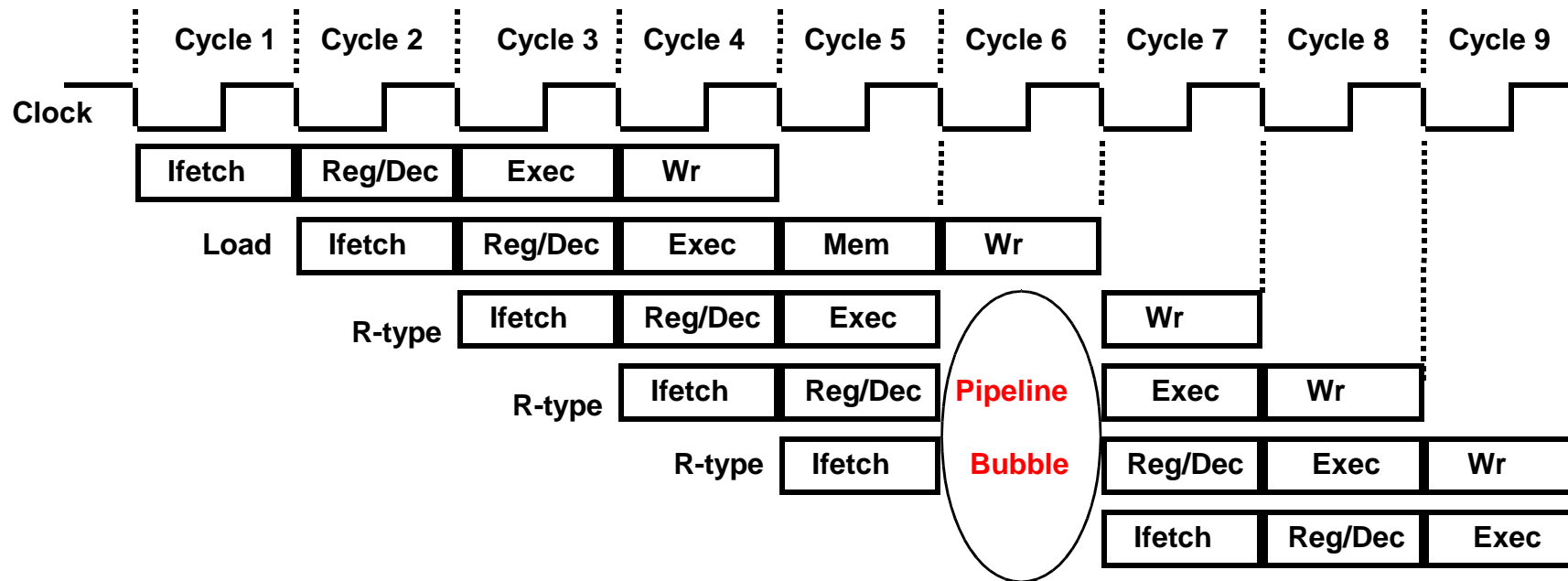


# Osservazione importante

- Ogni unità funzionale può essere usata solo una volta da ciascuna istruzione (ci sono 5 istruzioni in esecuzione contemporanea)
- Condizione necessaria, ma non sufficiente
- L'uso di unità funzionali in passi diversi da parte di istruzioni diverse potrebbe generare alee. Esempio:
  - Load usa il Write Port del Register File durante il 5° passo
  - Le istruzioni R-type usano il Write Port del Register File durante il 4° passo
- Due modi possibili per risolvere l'alea



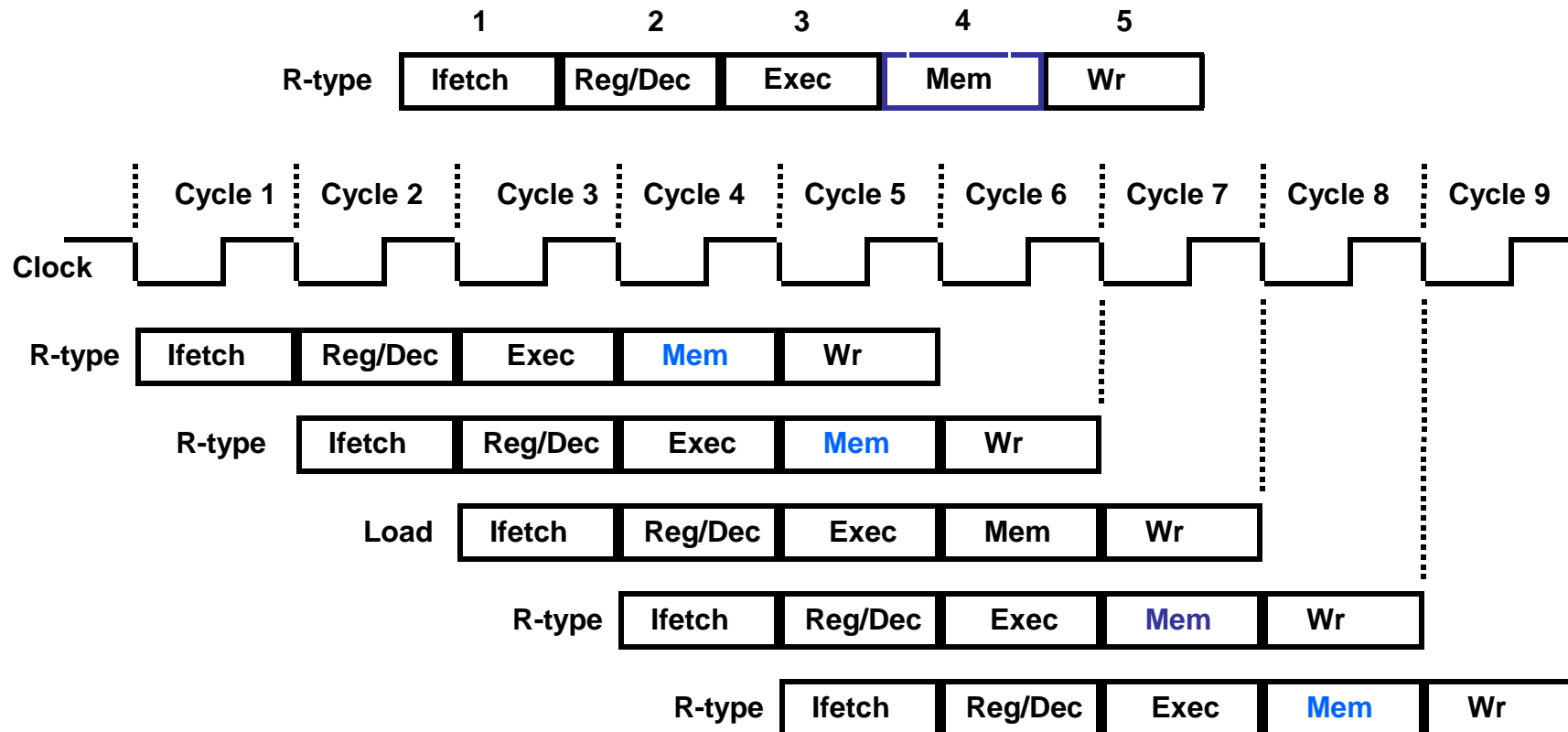
# Soluzione 1: “Bolla” nel Pipeline



- Si inserisce una “bolla” nel pipeline per evitare due write nello stesso ciclo (nel ciclo procede solo la prima istruzione impegnata nel write) :
  - Logica di controllo complessa
  - Si perde la possibilità di avviare altre istruzioni

# Soluzione 2: 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



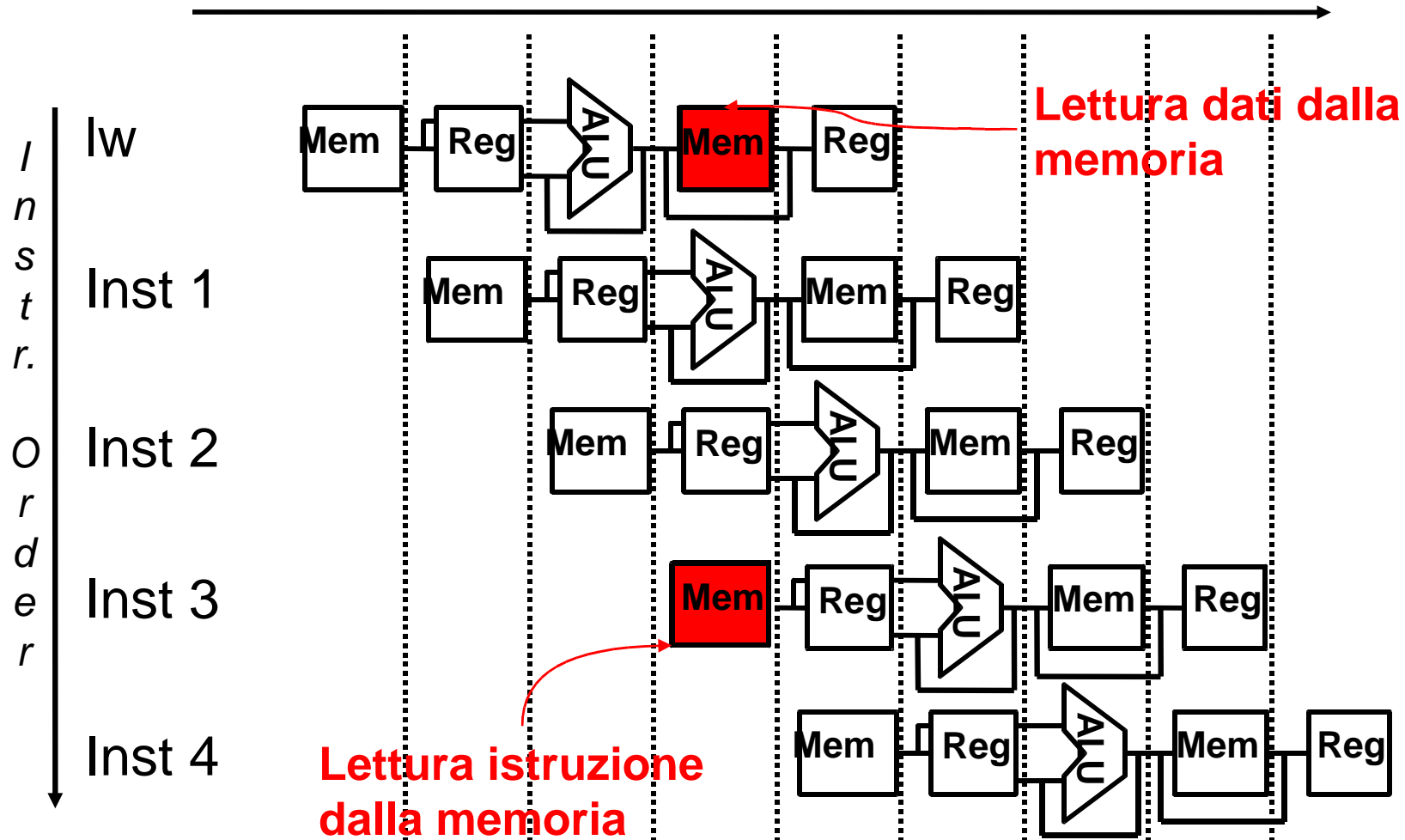
# Altri problemi dal pipelining ?

- **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

*Tempo (cicli di clock)*

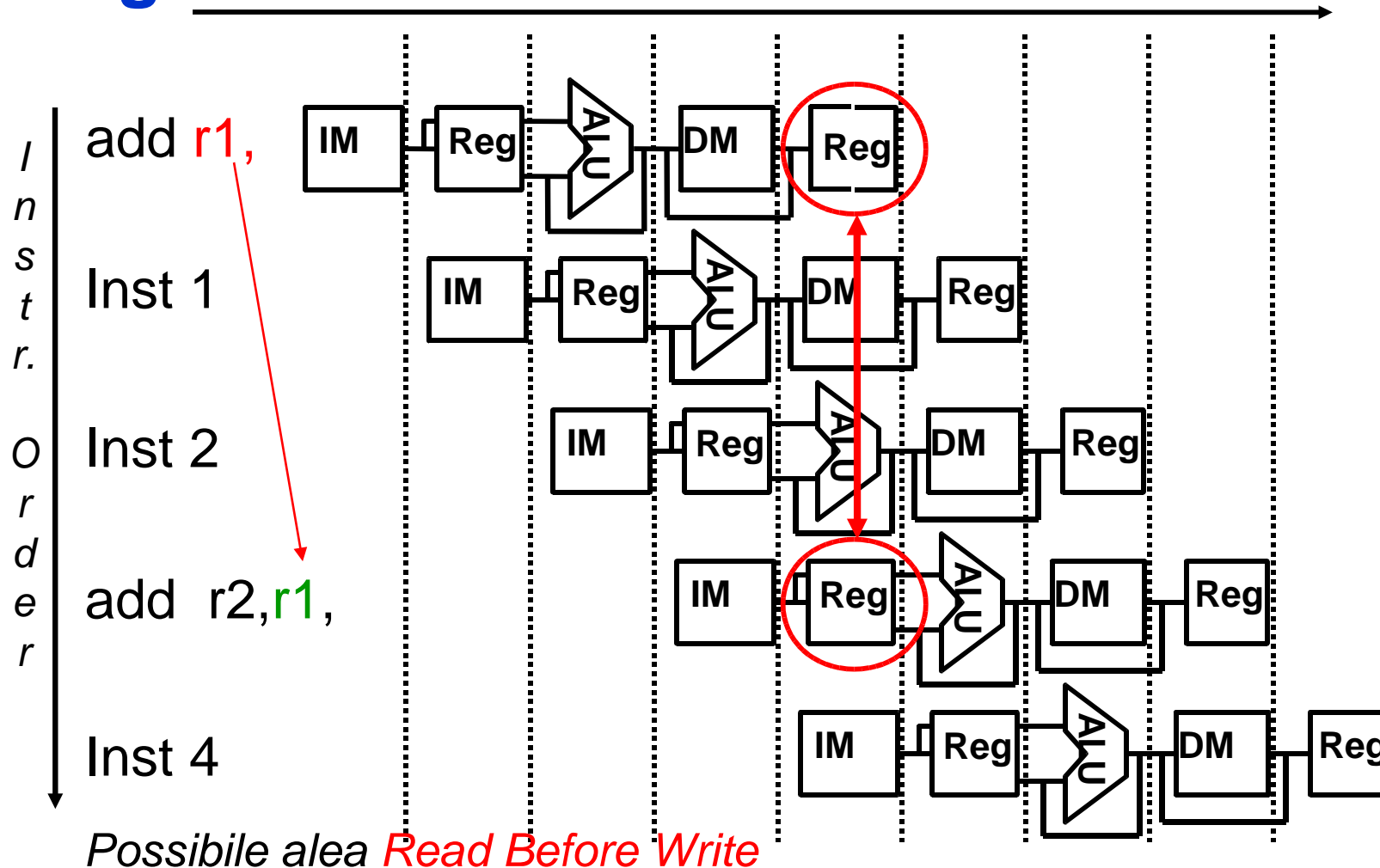


# Alea strutturale 1: memoria unica

- 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

Tempo (cicli di clock)

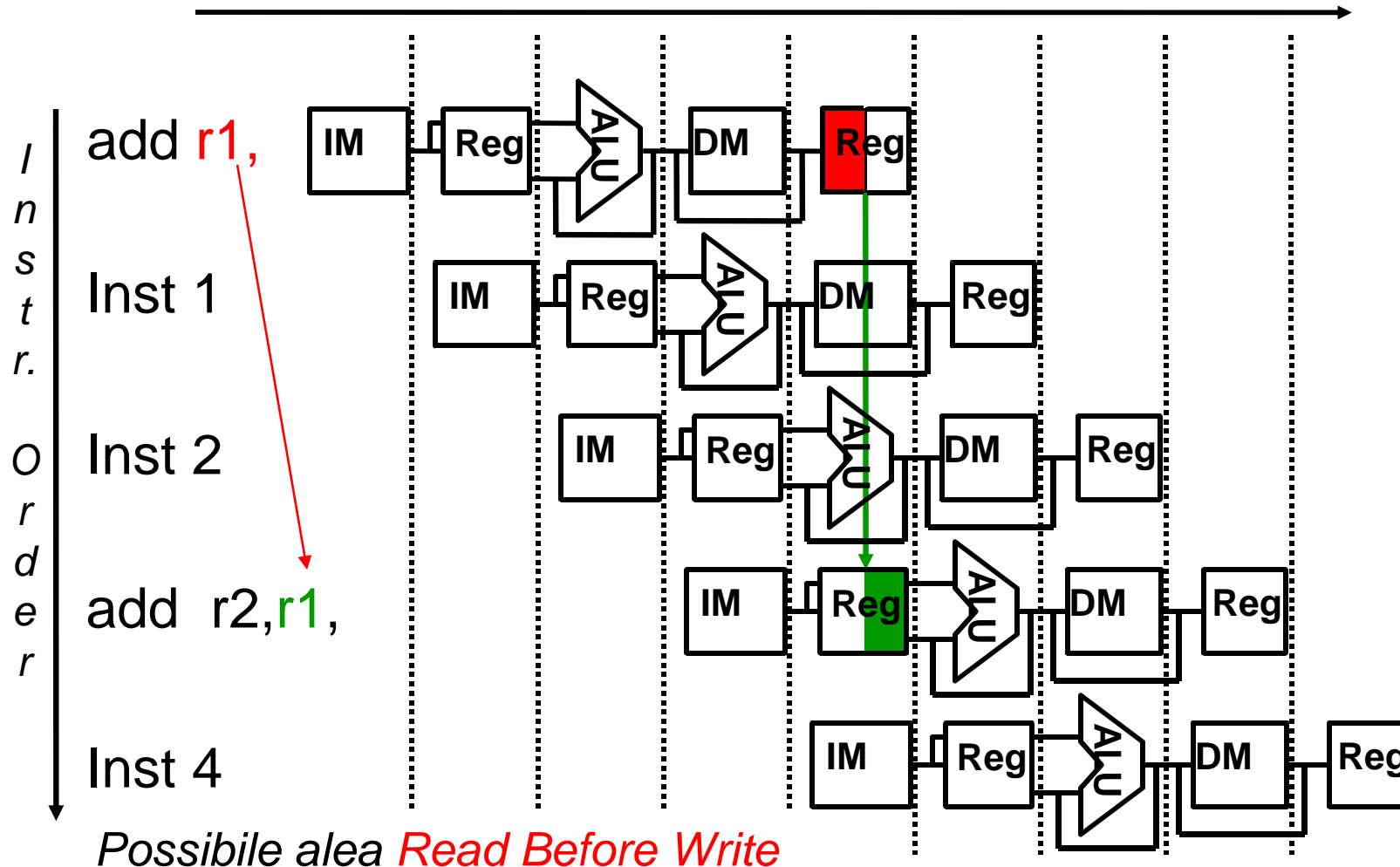


## Alea strutturale 2: accesso ai registri

- 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

Tempo (cicli di clock)



# Alea sui dati

- Consideriamo la sequenza di istruzioni

add \$t0, \$t1, \$t2

sub \$t4, \$t0, \$t3

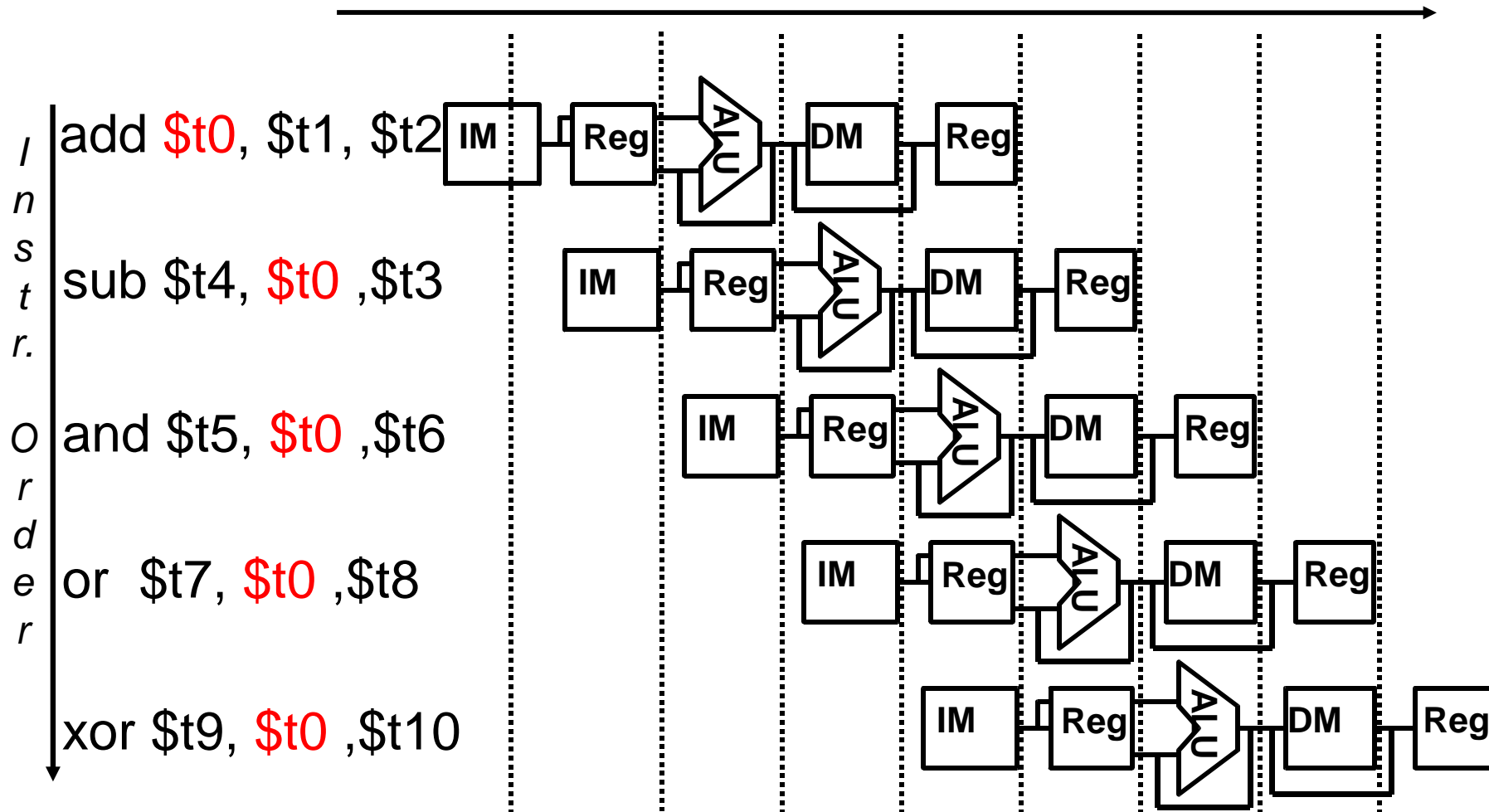
and \$t5, \$t0, \$t6

or \$t7, \$t0, \$t8

xor \$t9, \$t0, \$t10

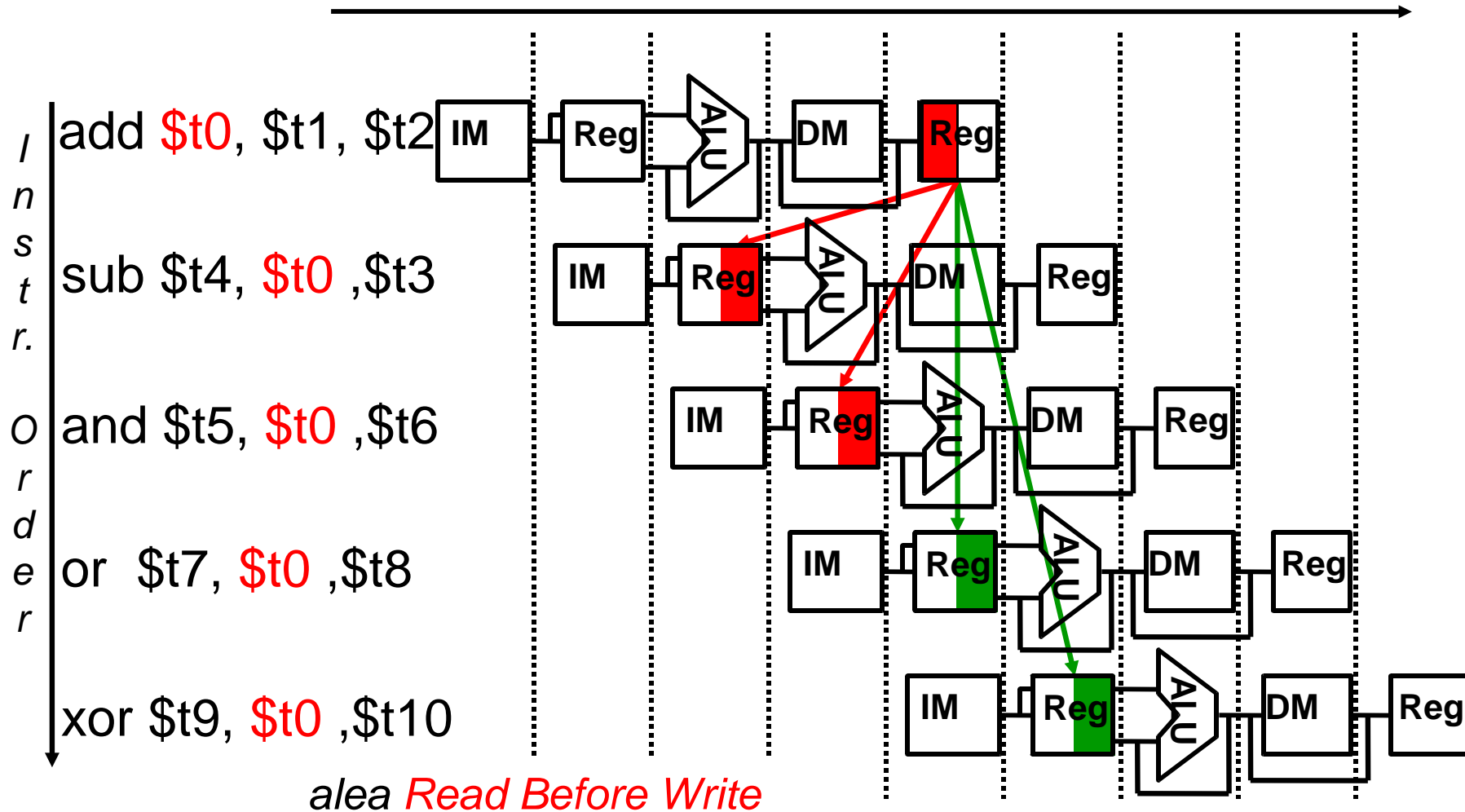
# Alea sui dati

- L'influenza su fasi precedenti produce alee



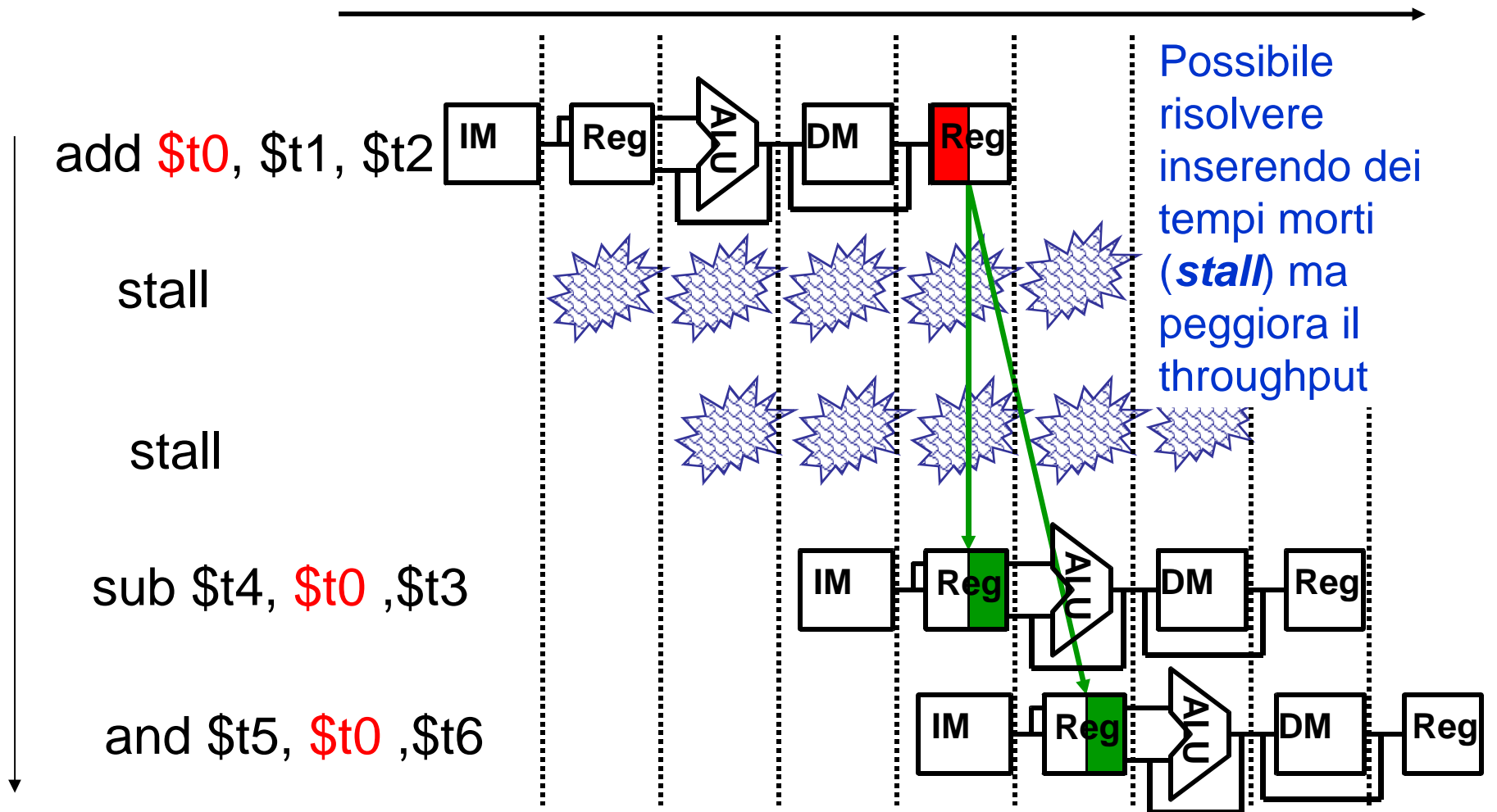
# Alea sui dati

- L'influenza su fasi precedenti produce alee

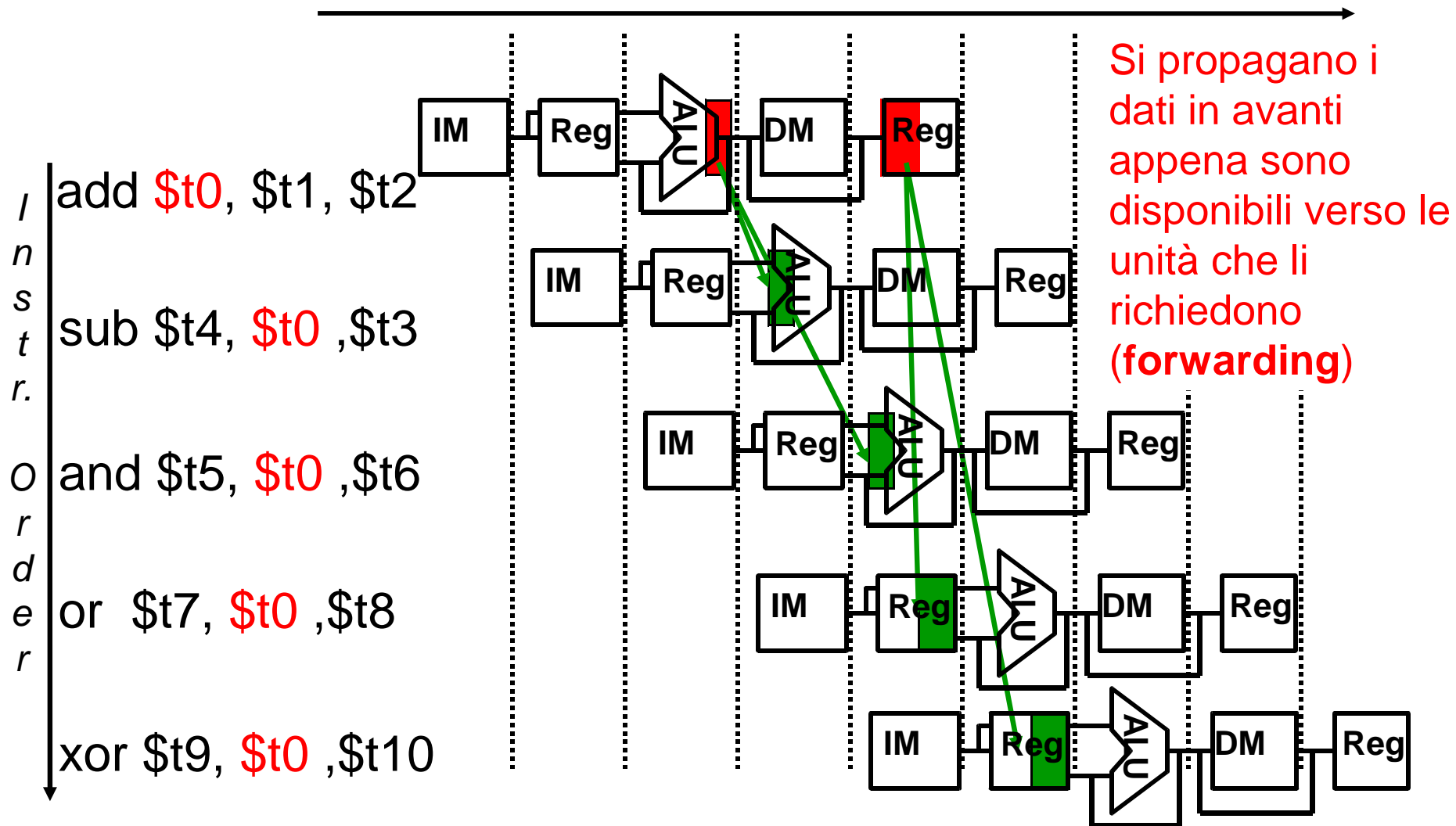




# Alea sui dati: soluzione 1

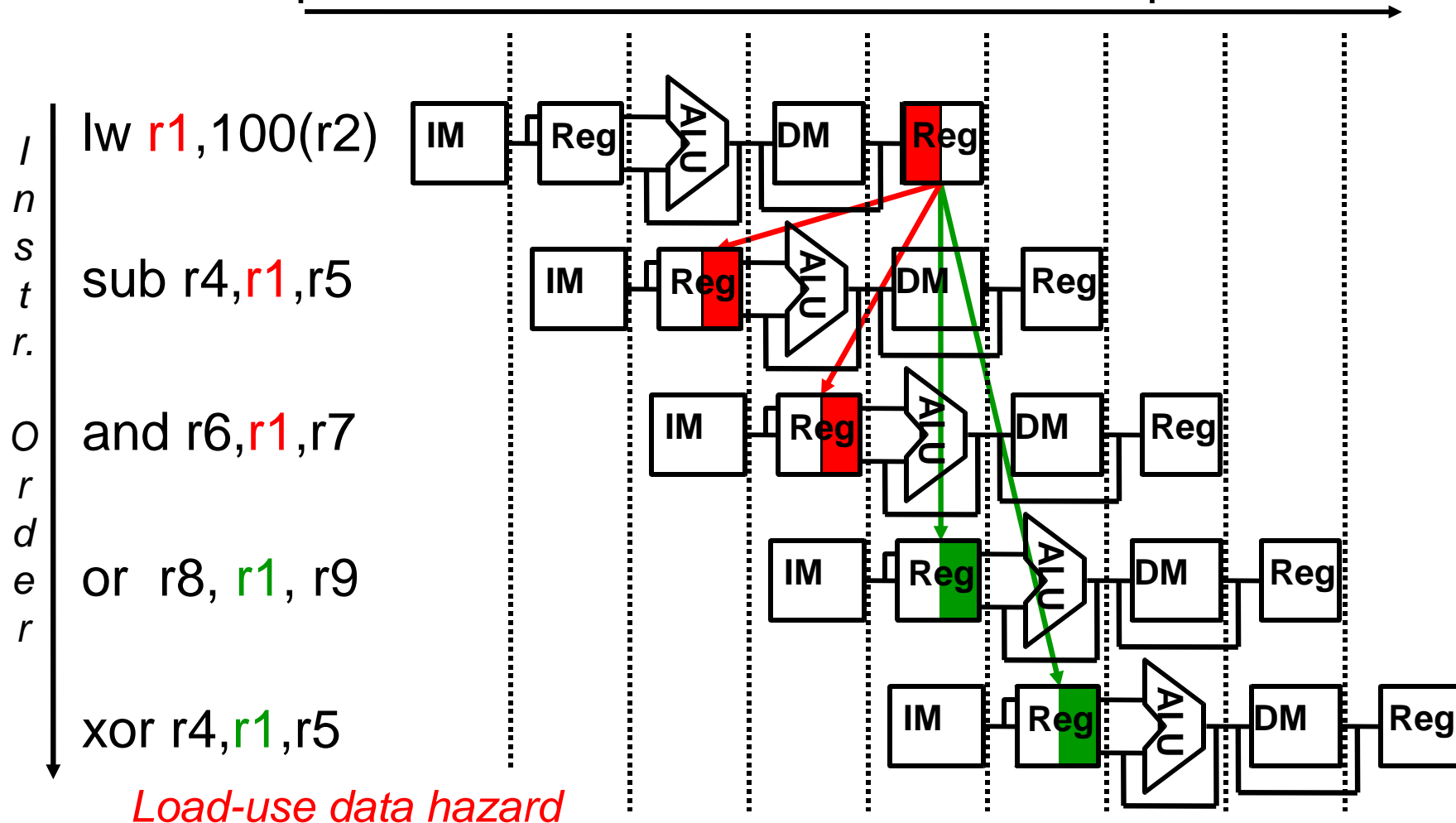


# Alea sui dati: soluzione 2

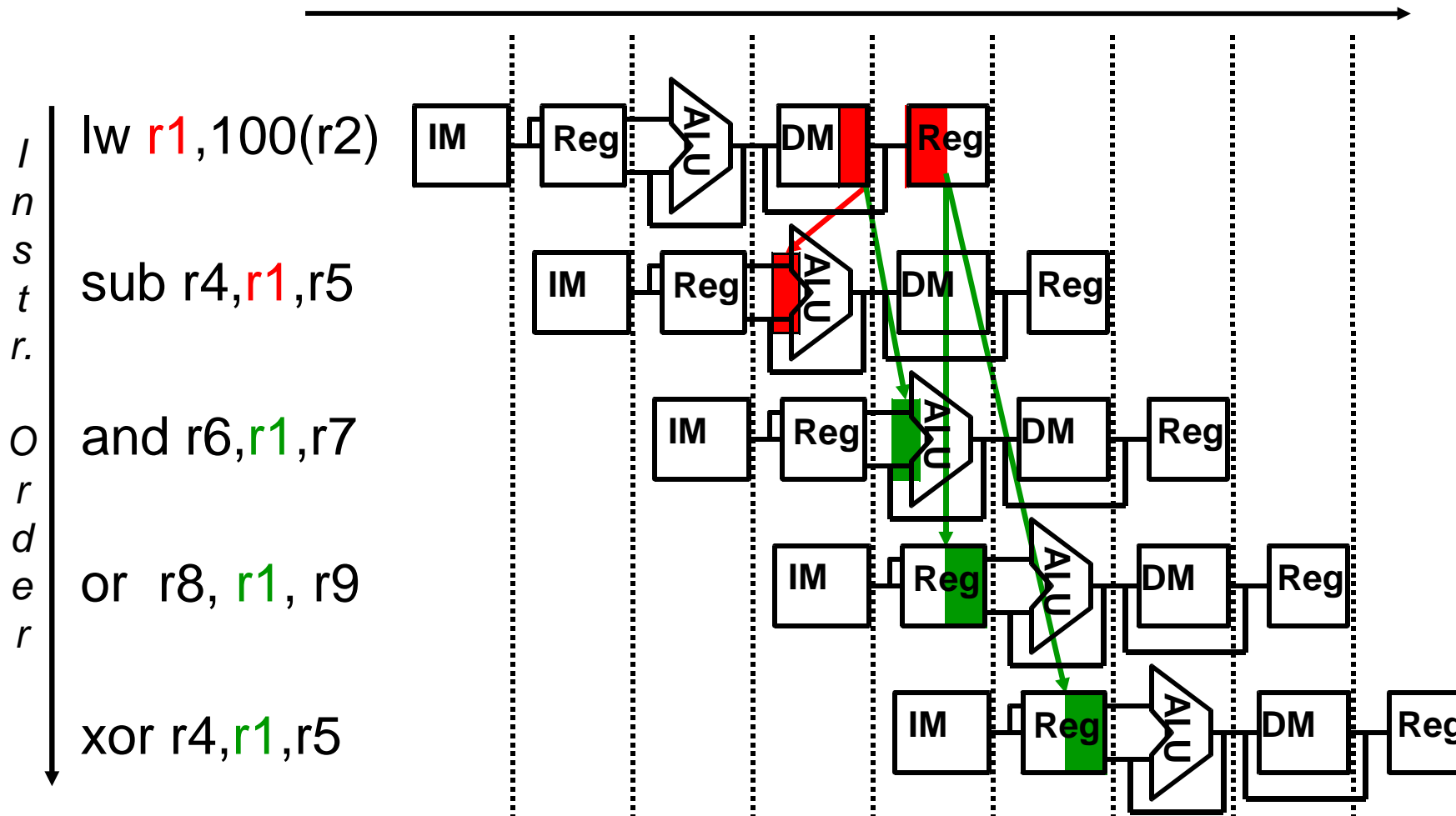


# Alea sui dati da Load

Simile al precedente, ma con una difficoltà in più



# Alea sui dati da Load: forwarding



- Forwarding insufficiente. Necessario uno stall ?

# Alea sui dati da Load

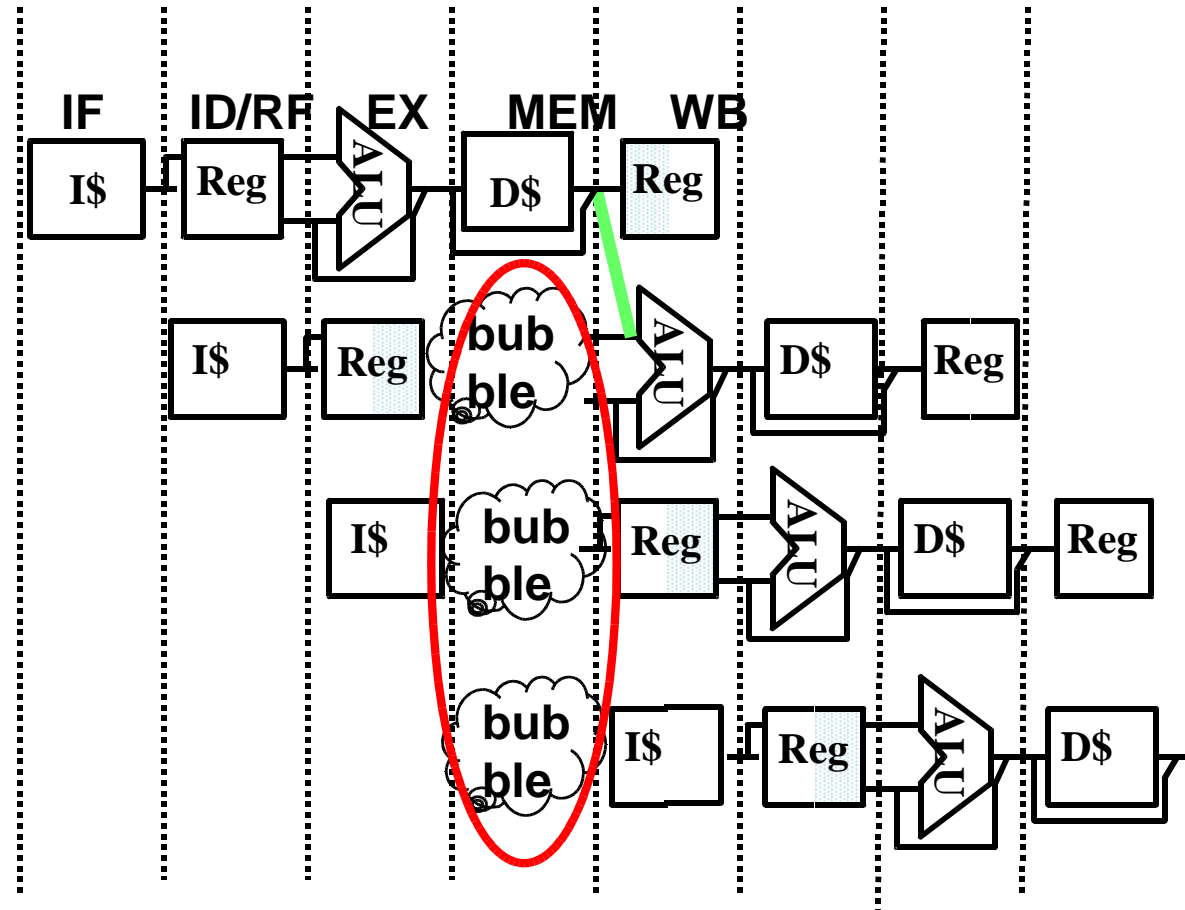
- Il controllo deve bloccare il pipeline
- interlock

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

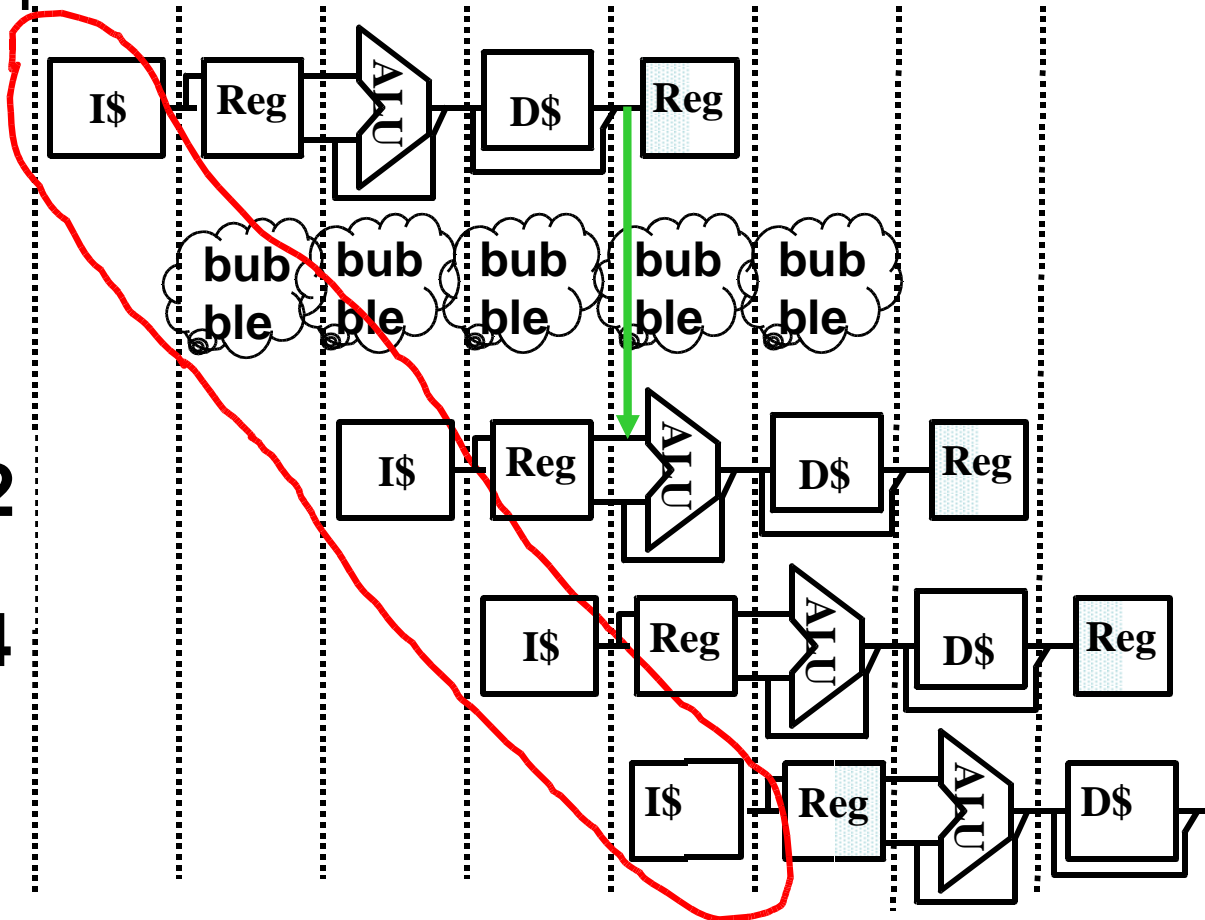
lw **\$t0**, 0(\$t1)

nop

sub \$t3,**\$t0**,\$t2

and \$t5,**\$t0**,\$t4

or \$t7,**\$t0**,\$t6

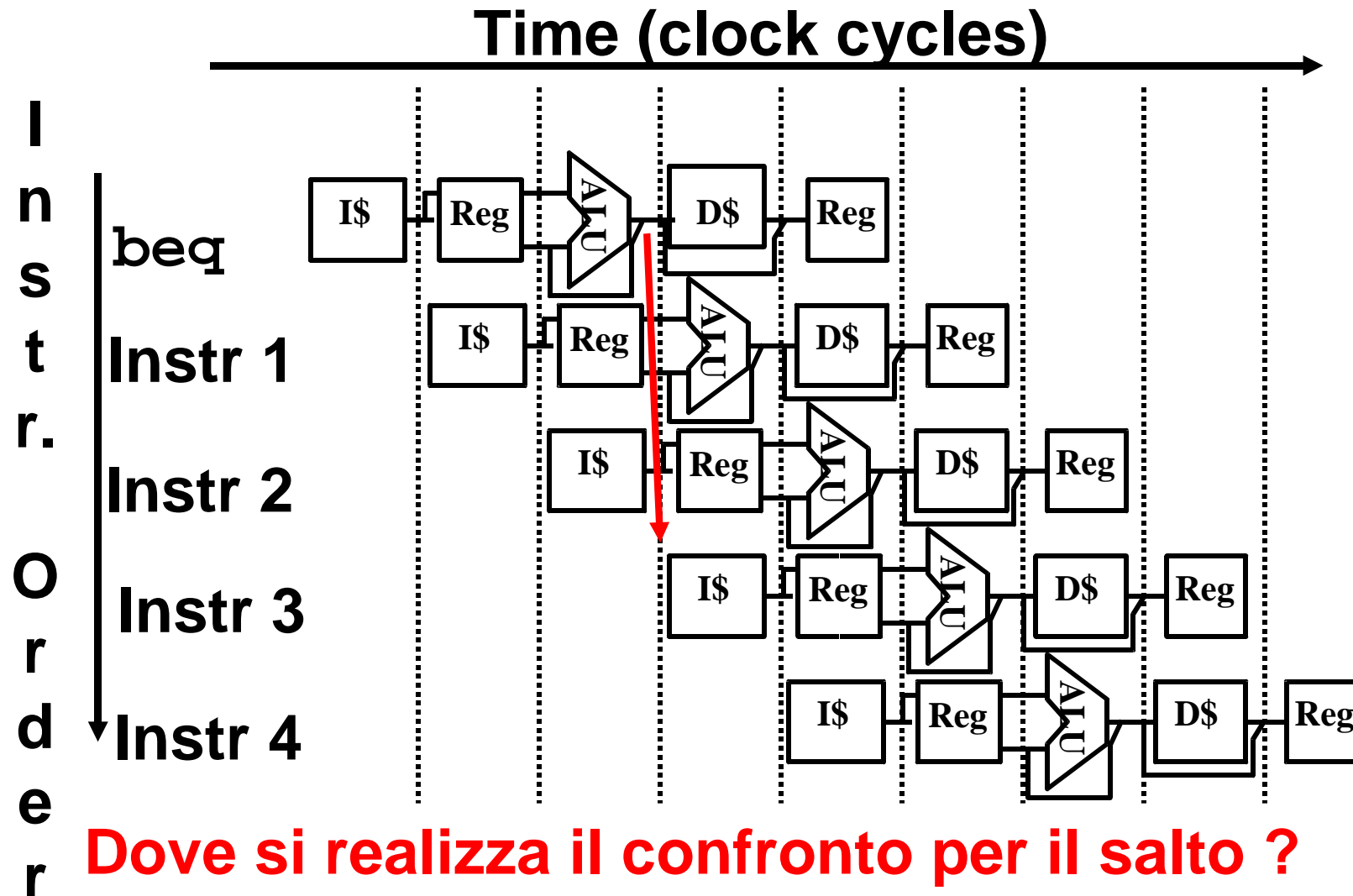


## Nota storica

- 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

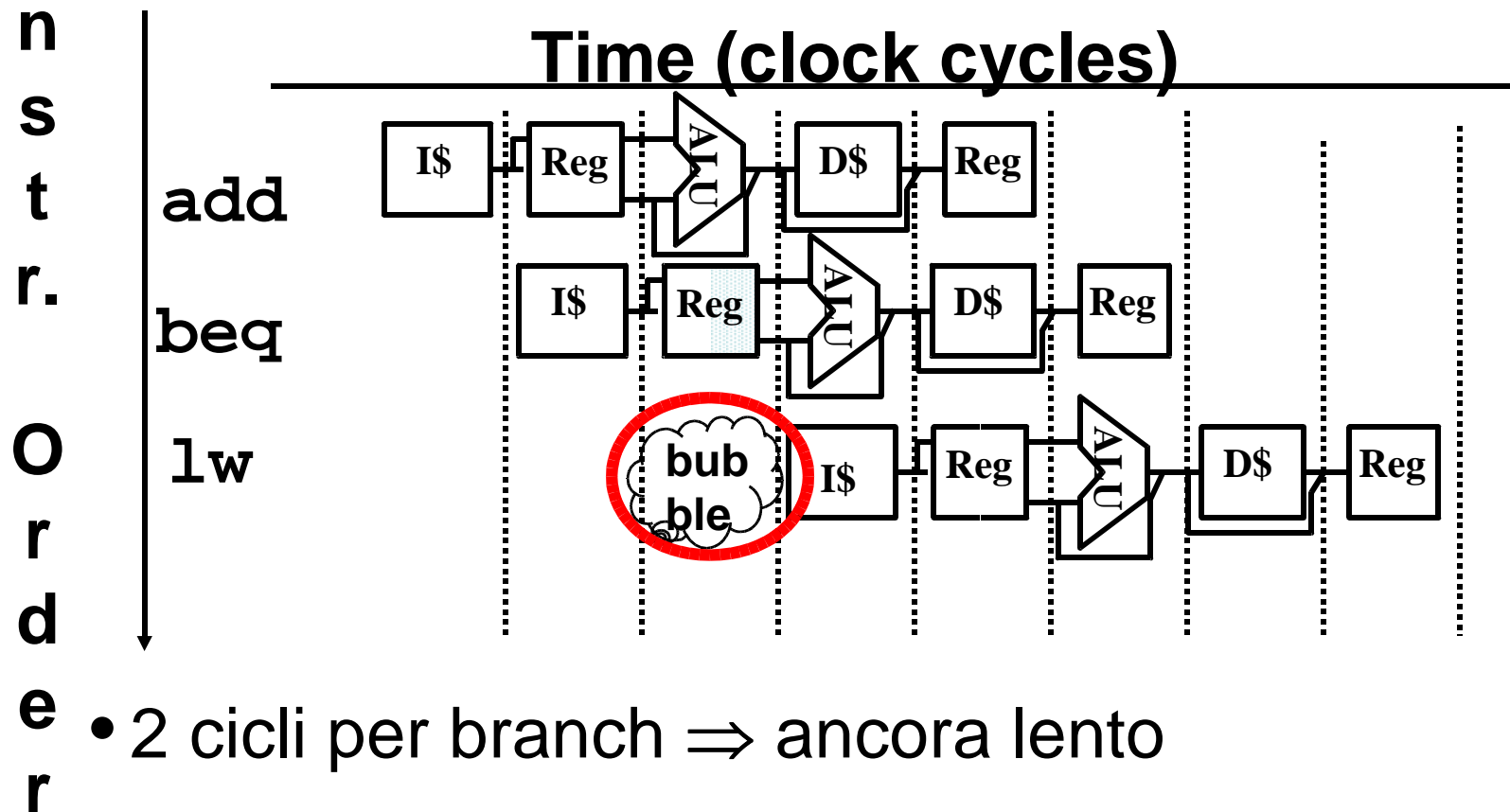
- Blocco del pipeline finchè il confronto non sia stato realizzato
  - Inserimento di NOP
  - Conseguenza: i salti richiedono 3 cicli di clock (assumendo che il confronto si faccia nel 3° stage)

# 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  $\Rightarrow$  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

## Nondelayed Branch

or \$8, \$9, \$10

add \$1, \$2, \$3

sub \$4, \$5, \$6

beq \$1, \$4, Exit

xor \$10, \$1, \$11

Exit:

F.Tortorella

## Delayed Branch

add \$1, \$2, \$3

sub \$4, \$5, \$6

beq \$1, \$4, Exit

or \$8, \$9, \$10

xor \$10, \$1, \$11

Exit:

Calcolatori Elettronici  
2010/2011

Università degli Studi  
di Cassino