

# **Architettura dei Calcolatori e Sistemi Operativi**

## **MIPS - Architettura**

---

*Chair*

**Prof. C. Brandoles**

e-mail: carlo.brandolese@polimi.it  
phone: +39 02 2399 3492  
web: home.dei.polimi.it/brandole

---

**Politecnico di Milano**

*Teaching Assistant*

**D. Iezzi**

e-mail: domenico.iezzi [at] polimi [dot] it  
material: [github.com/NoMore201/polimi\\_cr\\_acso\\_2019](https://github.com/NoMore201/polimi_cr_acso_2019)

# Outline

---

- **Formato Istruzioni**

- Istruzioni per formato

- **Architettura a singolo ciclo**

- Funzionamento
  - Segnali di controllo

- **Architettura pipelined**

- Funzionamento
  - Esecuzione sequenziale vs pipelined
  - Conflitti all'interno della pipeline
    - Conflitti strutturali
    - Conflitti sui dati
    - Conflitti di controllo

# Formato istruzioni

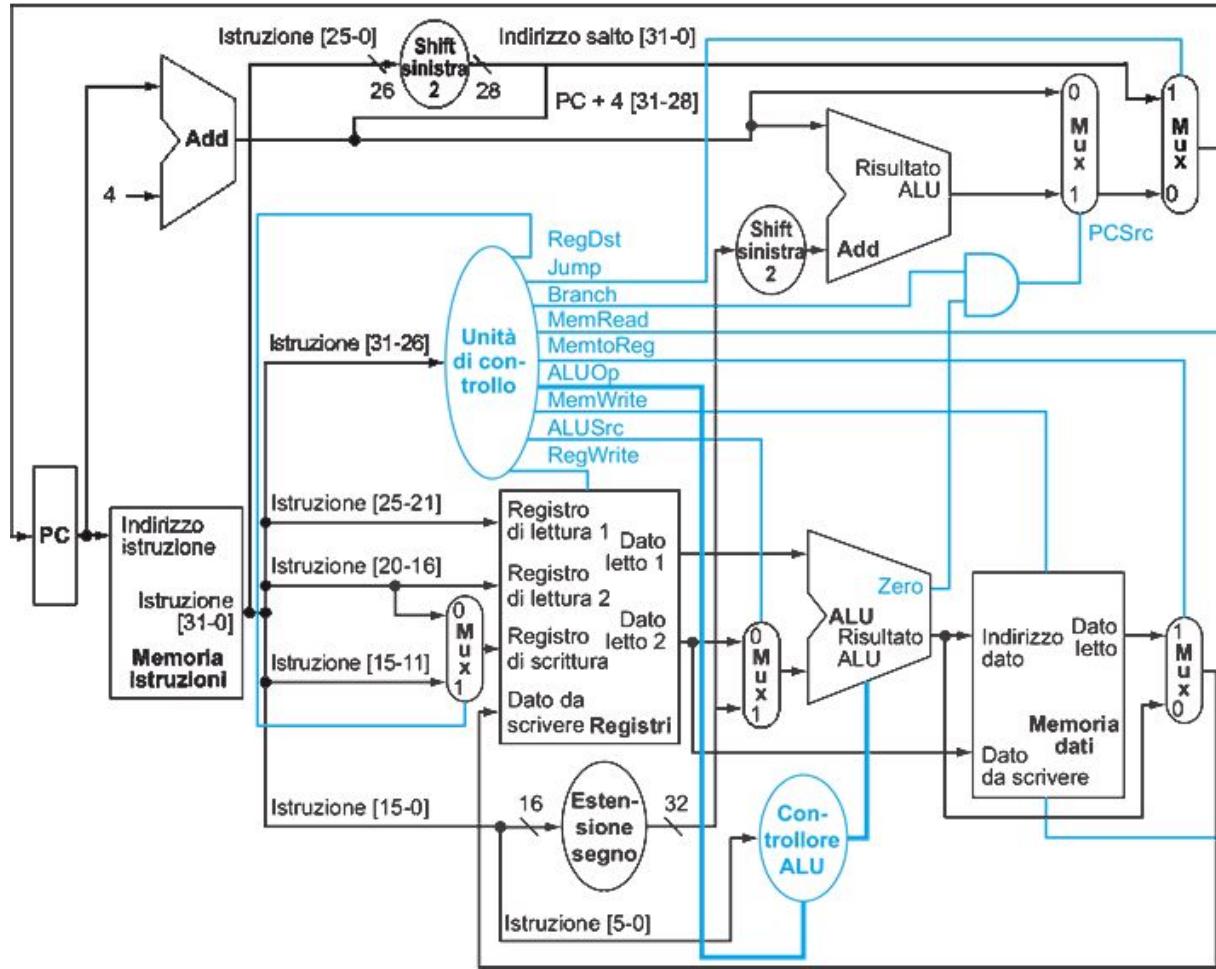
- Esistono 3 tipi di formati istruzioni nel processore MIPS

- Istruzioni R
- Istruzioni I
- Istruzioni J

Nome	Campi						Commenti
Dimensione del campo	6 bit	5 bit	5 bit	5bit	5bit	6 bit	Tutte le istruzioni MIPS sono a 32 bit
Formato R	opcode	rs	rt	rd	shamt	fun	Formato istruzioni aritmetico-logiche a 3 registri
Formato I	opcode	rs	rt	immediate			Formato istruzioni aritmetico-logiche con immediato, salto condizionato e load/store
Formato J	opcode	indirizzo destinazione					Formato istruzioni salto incondizionato

# Architettura a singolo ciclo

- Schema semplificato dell'architettura MIPS a singolo ciclo
  - Supporto della maggior parte delle funzioni

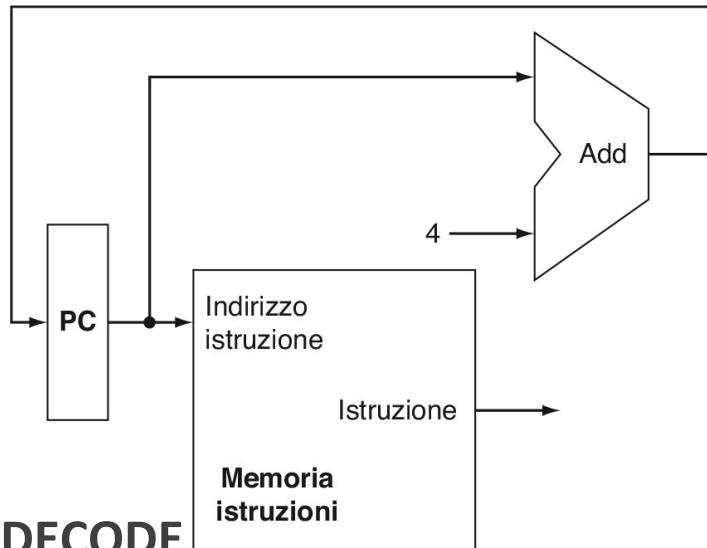


# Funzionamento (I)

- Il funzionamento dell'architettura MIPS si divide in 5 parti principali:

- FETCH**

- Caricamento dell'istruzione nel PC dalla memoria istruzioni
- Incremento del PC



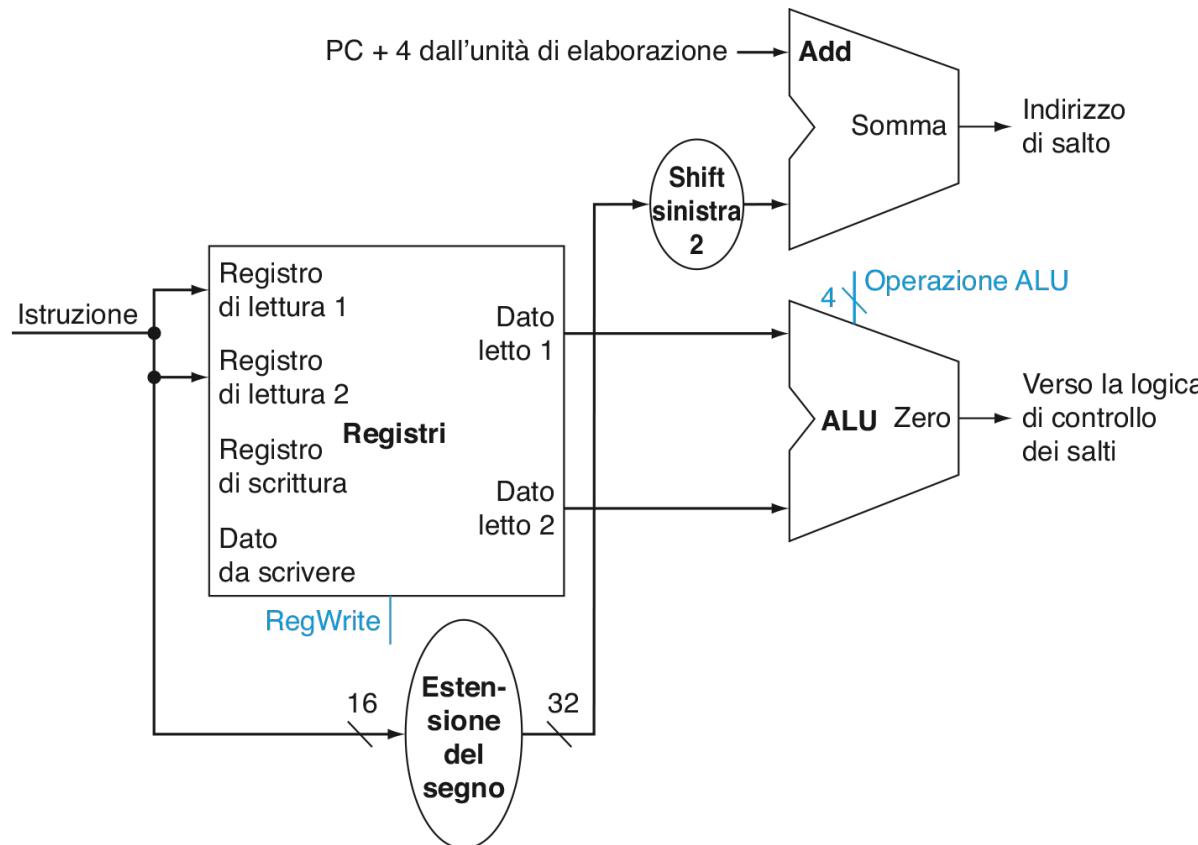
- DECODE**

- Lettura dei dati da Register File
- Preparazione di eventuali altri operandi per la ALU ( Ex. Immediati )
- Preparazione dei segnali di controllo per l'esecuzione dell'istruzione

# Funzionamento (II)

## – EXECUTE

- Calcolo del risultato dell'operazione
- Calcolo indirizzo di salto



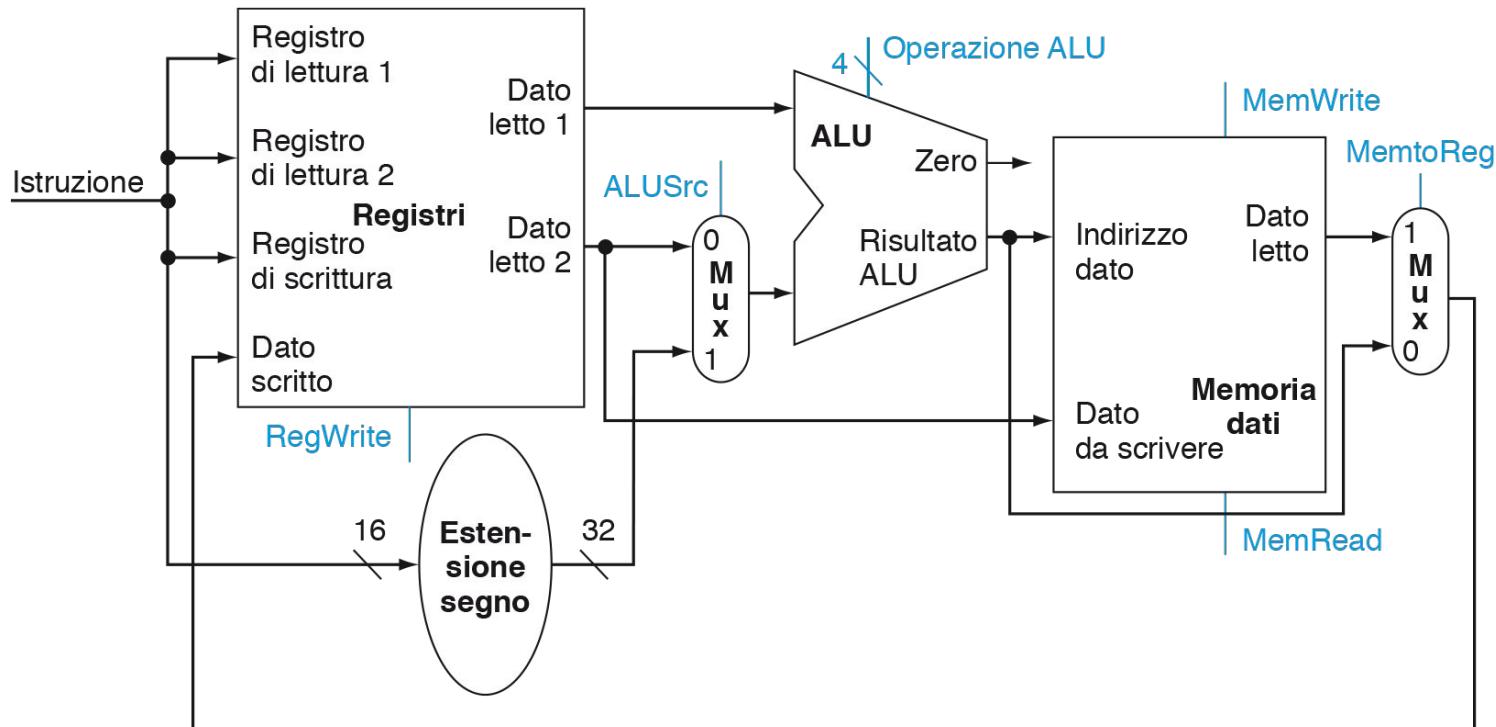
# Funzionamento (III)

## – MEMORY

- Caricamento di eventuali dati da memoria

## – WRITE BACK

- Scrittura nel Register File



# Architettura a singolo ciclo – Segnali di controllo

- I segnali di controllo sono i punti di controllo dell'architettura MIPS
  - L'unità di controllo genera questi segnali in base all'istruzione da eseguire
  - Il controllore ALU è un sottosistema dell'unità di controllo

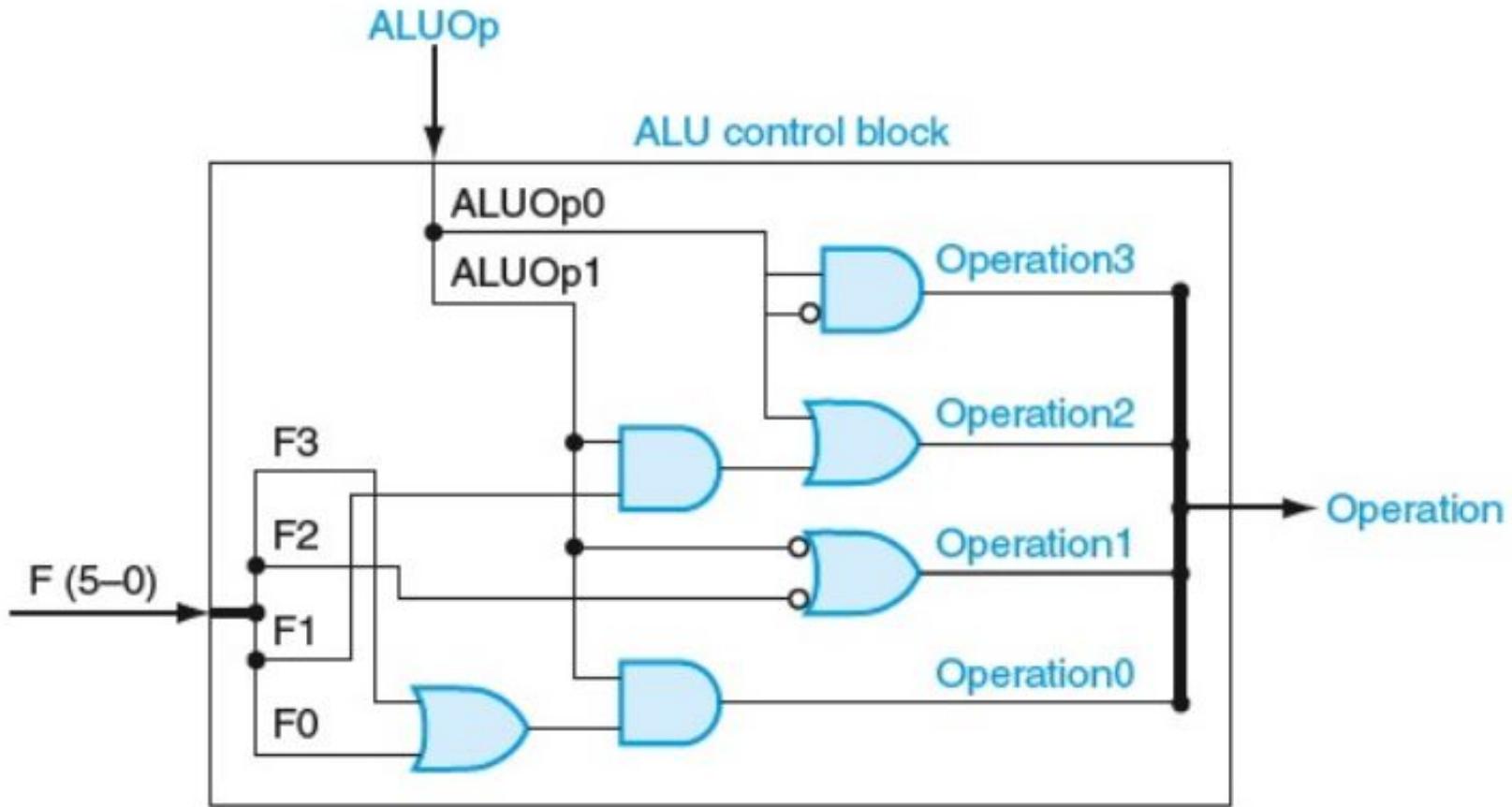
Nome del segnale	Effetto quando non asserito	Effetto quando asserito
RegDst	Il numero del registro di scrittura proviene dal campo rt (bit 20-16)	Il numero del registro di scrittura proviene dal campo rd (bit 15-11)
RegWrite	Nulla	Il dato viene scritto nel register file nel registro individuato dal numero 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

# Architettura a singolo ciclo – Controllore ALU

- Il controllore ALU decide come deve operare la ALU in base all'istruzione da eseguire. Il segnale ALUOp (a due linee) indica che tipo di operazione dovrà essere effettuata (**load/store**, **branch** o aritmetica)

Operation	ALUOp	Funct	Output
Load/Store	00	XXXXXX	0010
BEQ	01	XXXXXX	0110
ADD	10	100000	0010
SUB	10	100010	0110
AND	10	100100	0000
OR	10	100101	0001

# Architettura a singolo ciclo – Controllore ALU



# Esercizio 1

---

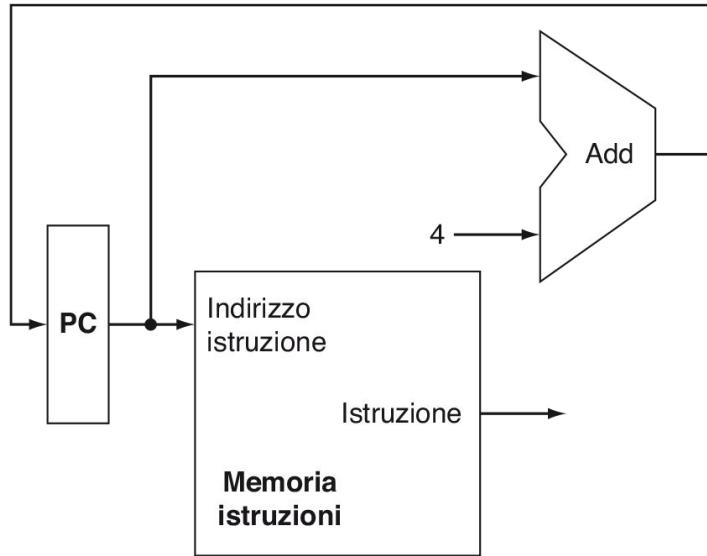
- Si suppone che i blocchi logici richiesti per implementare l'unità di elaborazione di un processore abbiano le latenze riportate nella tabella seguente

Mem-I	Add	Mux	ALU	Registri	Mem-D	Estensione segno	Shift Sx 2
200 ps	70 ps	20 ps	90 ps	90 ps	250 ps	15 ps	10 ps

1. Se richiedessimo al processore di prelevare soltanto le istruzioni una dopo l'altra, quale sarebbe il periodo del clock?
2. Si consideri un'unità di elaborazione completa che debba eseguire solamente istruzioni di un unico tipo: *salti incondizionati relativi al PC*. Quale sarebbe il cammino critico di questa unità di elaborazione?
3. Quale sarebbe il cammino critico per *salti condizionati relativi al PC*?

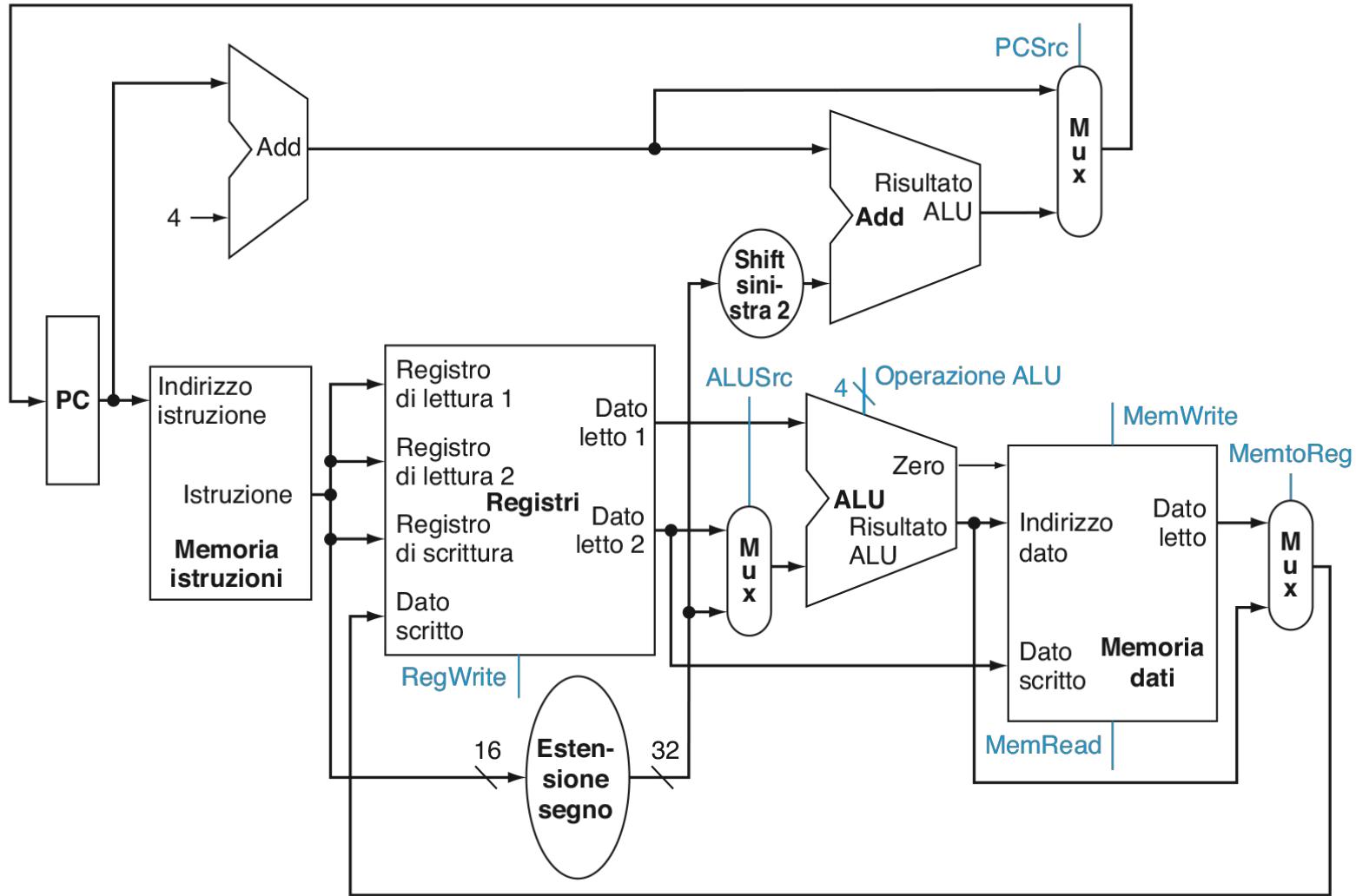
# Esercizio 1

- Riferimento per punto 1



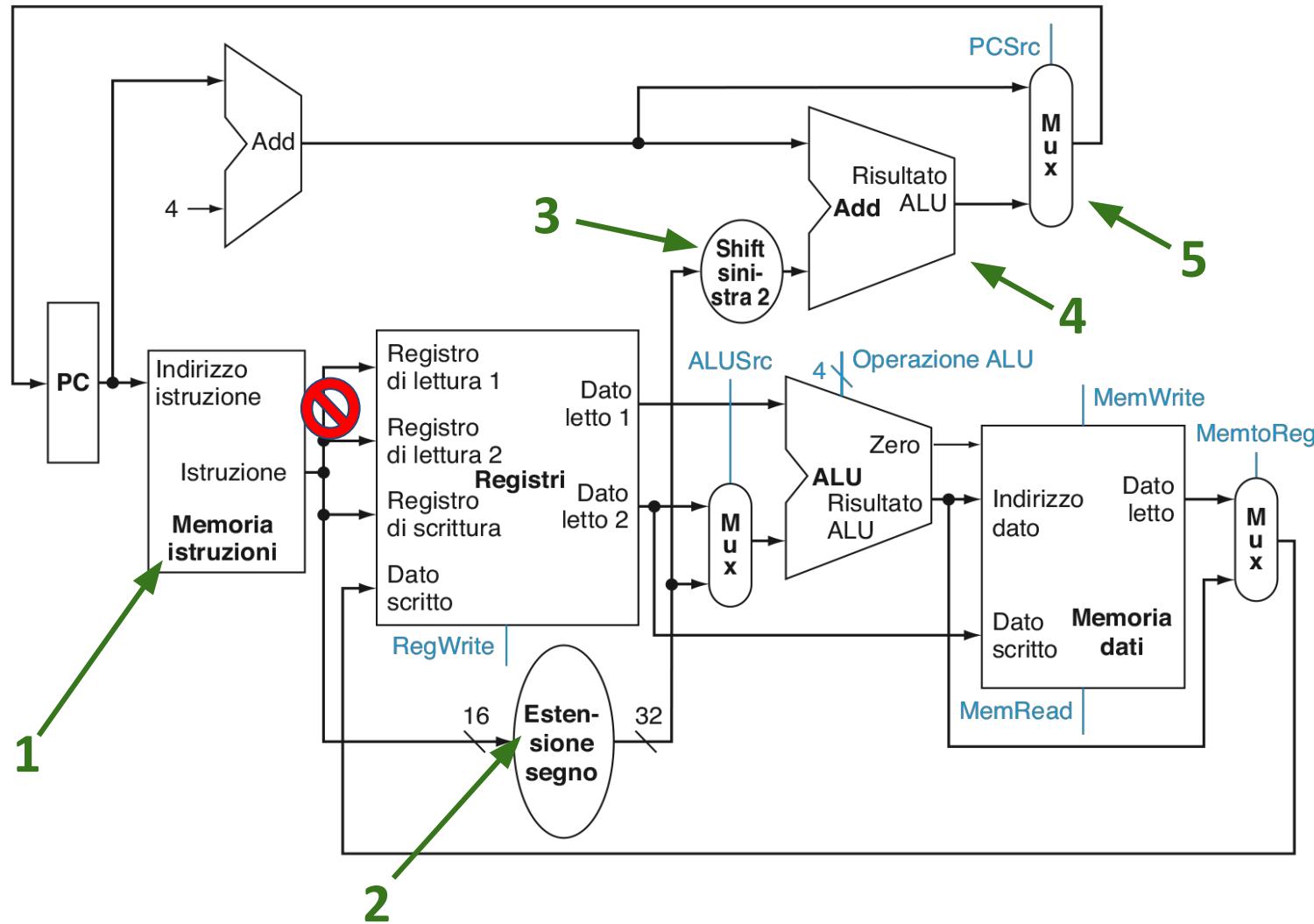
# Esercizio 1

## ■ Riferimento punto 2 e 3



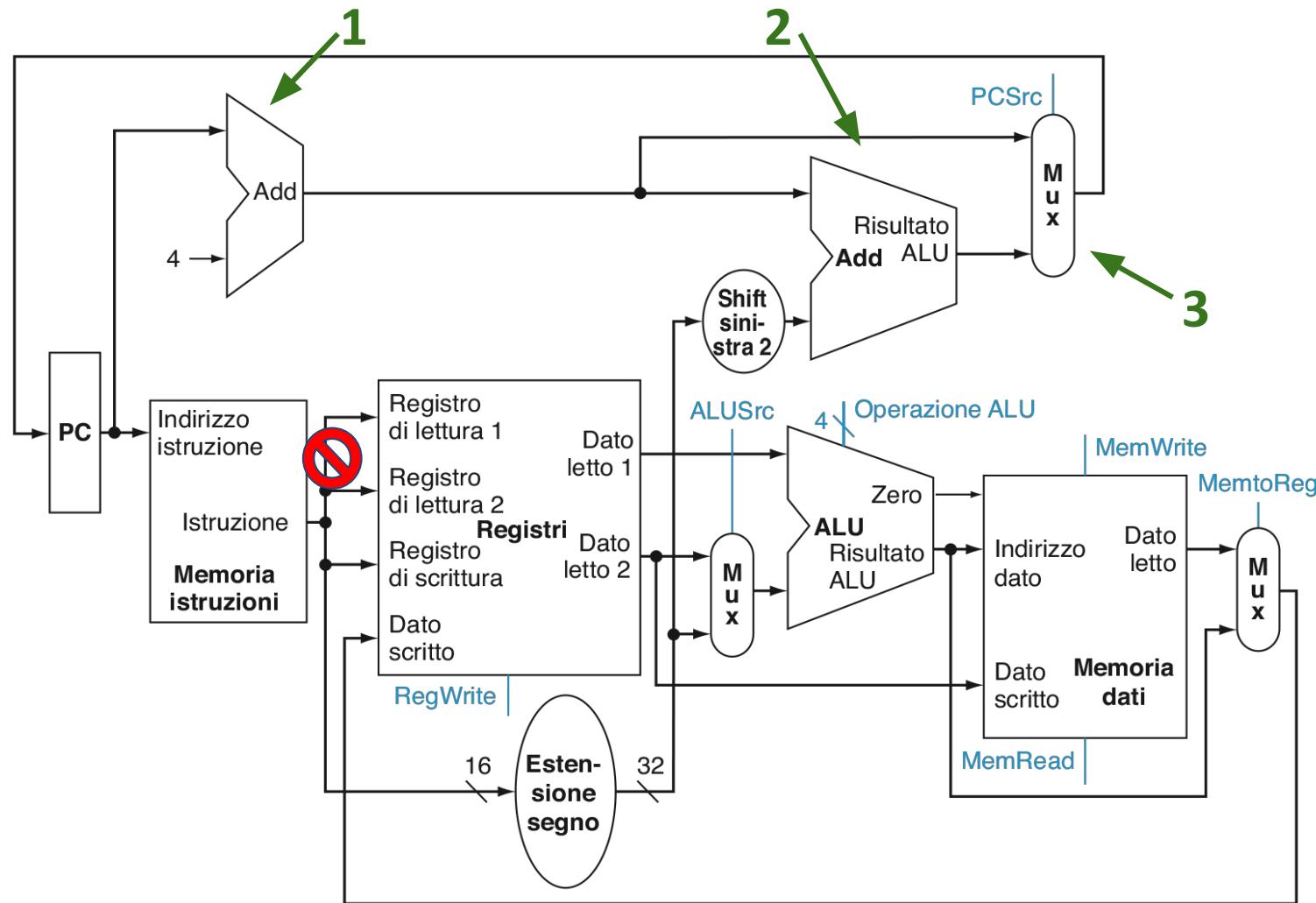
# Esercizio 1 - Punto 2

## Path 1



# Esercizio 1 - Punto 2

Path 2

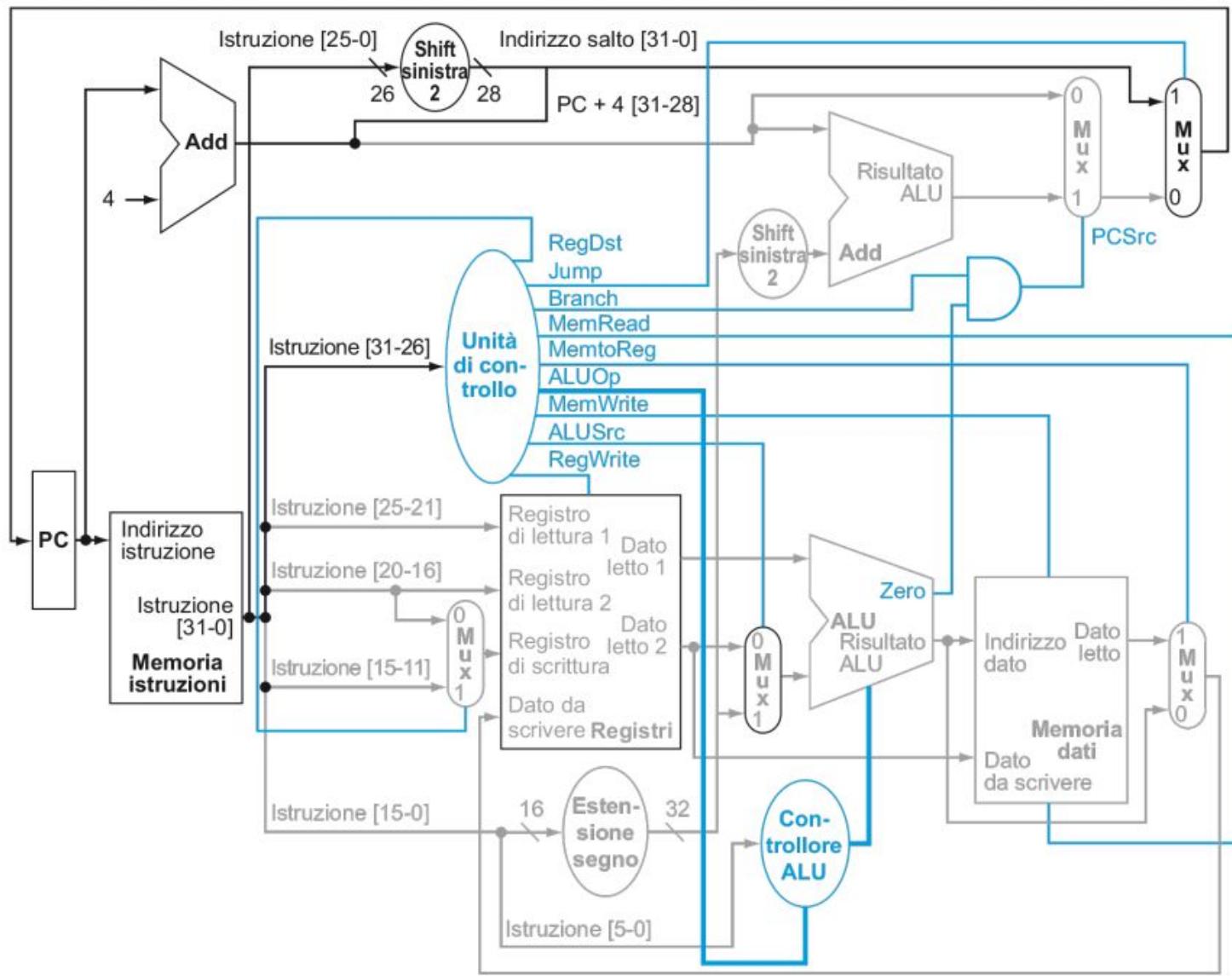


# Esercizio 2

---

- I tre problemi di questo esercizio riguardano l'elemento «Shift left 2».
1. Quali tipi di istruzioni richiedono questa risorsa?
  2. Per quali tipi di istruzioni (se ce ne sono) questa risorsa si trova sul loro cammino critico?
  3. Supponendo che la nostra architettura supporti solo le istruzioni *beq* e *add*, discutere in che modo le variazioni di latenza di questo elemento influiscono sul periodo del clock del processore (le latenze degli altri elementi non cambiano)

# Esercizio 2



# Esercizio 3

---

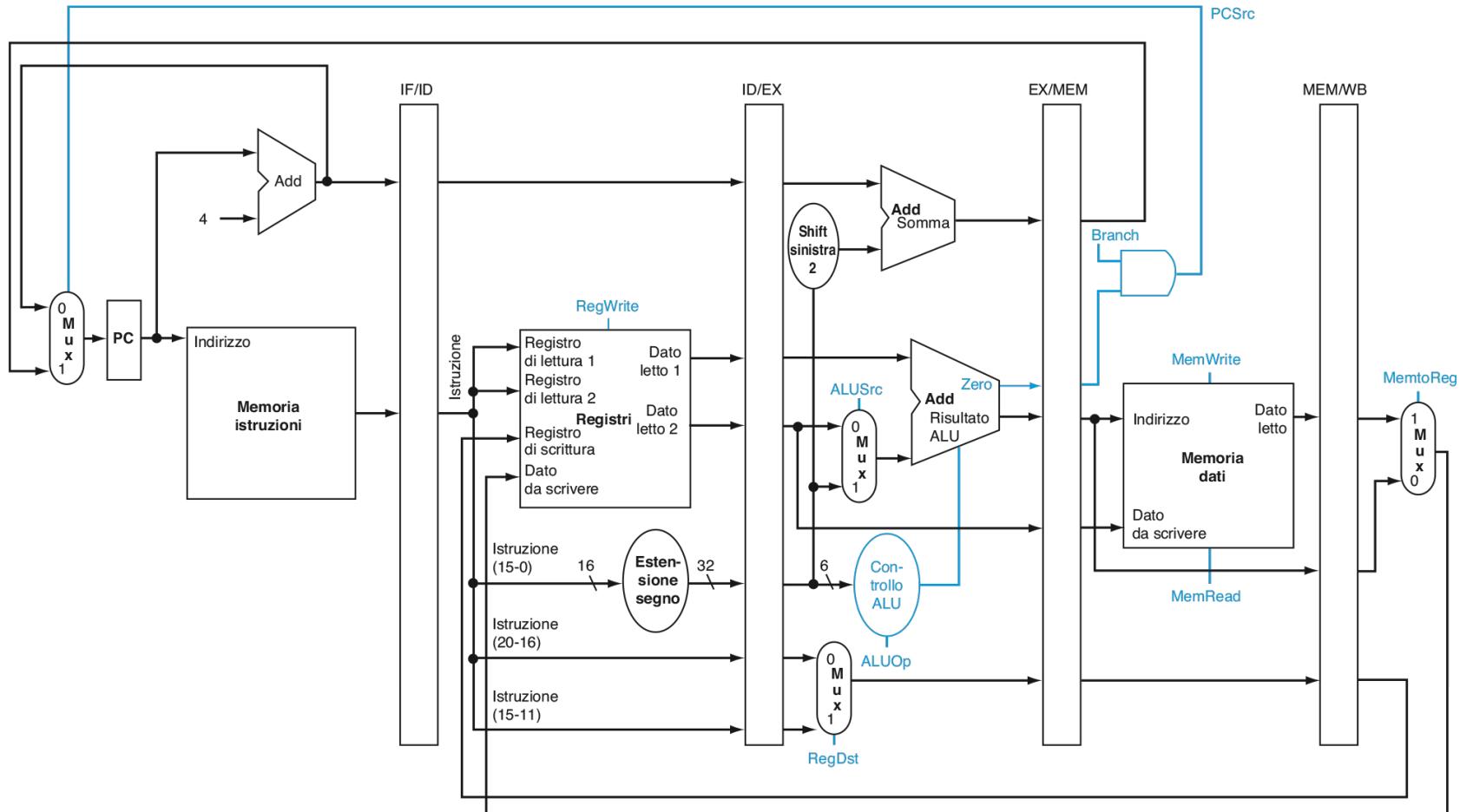
- Per i problemi di questo esercizio si supponga che la percentuale delle singole istruzioni sia la seguente:

add	addi	not	beq	lw	sw
20 %	20 %	0 %	25 %	25 %	10 %

1. In quale percentuale del numero totale di cicli di clock viene utilizzata la memoria dati?
2. In quale percentuale del numero totale di cicli di clock viene richiesto il circuito di estensione del segno?

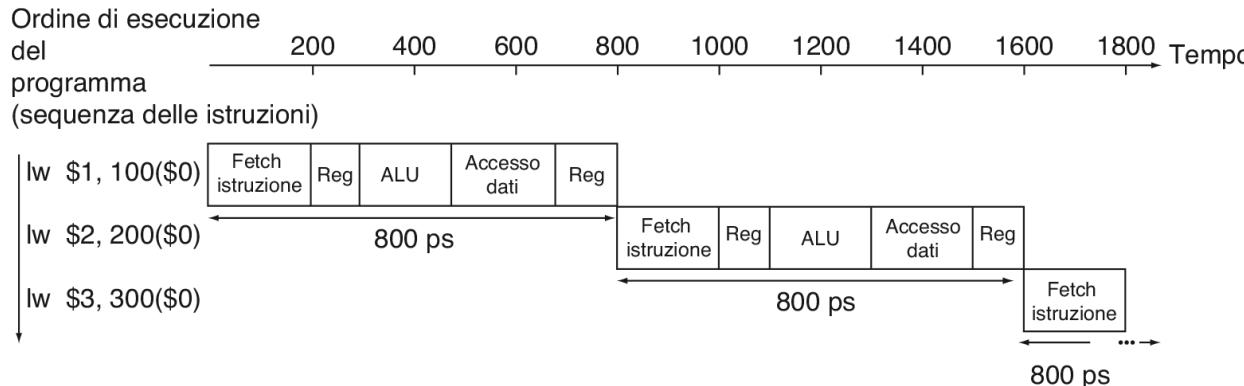
# Architettura Pipelined

- Aggiungendo dei registri tra le varie fasi si ottiene una architettura pipelined

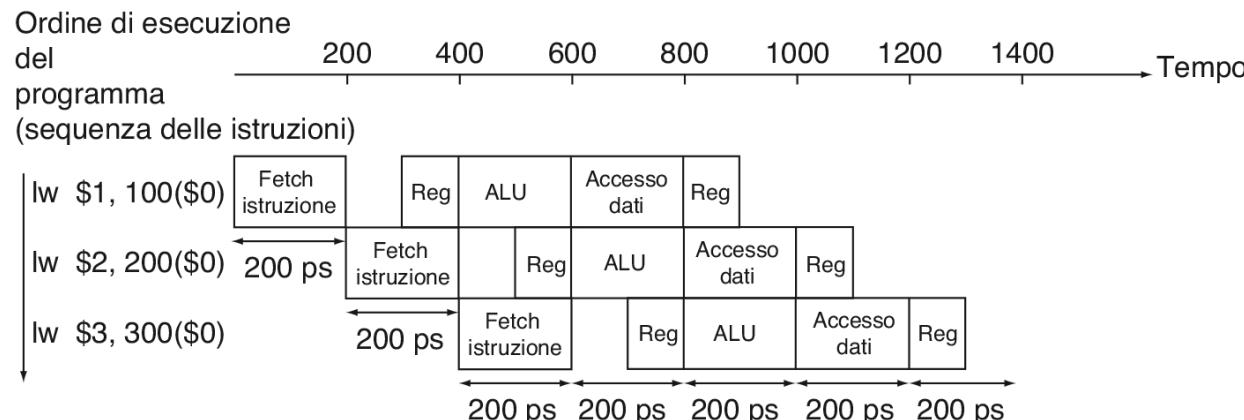


# Esecuzione sequenziale vs pipelined

- Nell'esecuzione sequenziale ogni istruzione inizia la propria esecuzione solo al termine della precedente



- Nell'esecuzione pipelined si migliorano le prestazioni basata sulla sovrapposizione dell'esecuzione di più istruzioni appartenenti ad un flusso di esecuzione sequenziale.



# CPI

---

- Per misurare l'efficienza di una certa architettura si utilizza un indicatore chiamato CPI ( Cycles Per Instruction )
- $$\text{CPI} = \frac{\# \text{ TOTALE CICLI}}{\# \text{ TOTALE ISTRUZIONI}}$$
- Intuitivamente il CPI rappresenta il numero medio di cicli necessari per completare un istruzione
- Attenzione!
  - $\text{CPI} \geq 1$  al massimo termina un istruzione per ciclo se non ci sono stalli

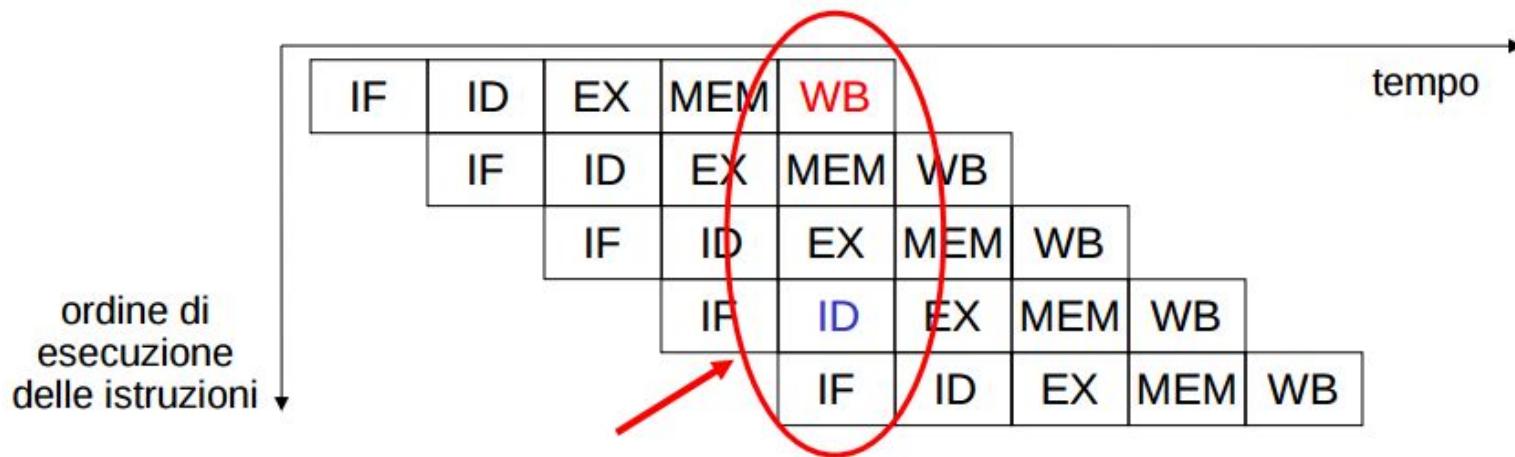
# Conflitti all'interno della pipeline

---

- I conflitti sorgono nelle architetture con pipelining quando non è possibile eseguire un'istruzione nel ciclo immediatamente successivo
  - Conflitti strutturali
    - Tentativo di usare la stessa risorsa hardware da parte di diverse istruzioni in modi diversi nello stesso ciclo di clock
  - Conflitti sui dati
    - Tentativo di usare un risultato prima che sia disponibile
  - Conflitti di controllo
    - Nel caso di salti, decidere quale prossima istruzione da eseguire prima che la condizione sia valutata

# Conflitto strutturali

- Nell'architettura MIPS pipeline non abbiamo conflitti strutturali
  - Memoria dati separata dalla memoria istruzioni
  - Banco dei registri progettato per evitare conflitti tra la lettura e la scrittura nello stesso ciclo
    - **Scrittura** del banco dei registri nella **prima metà** del ciclo di clock
    - **Lettura** del banco dei registri nella **seconda metà** del ciclo di clock

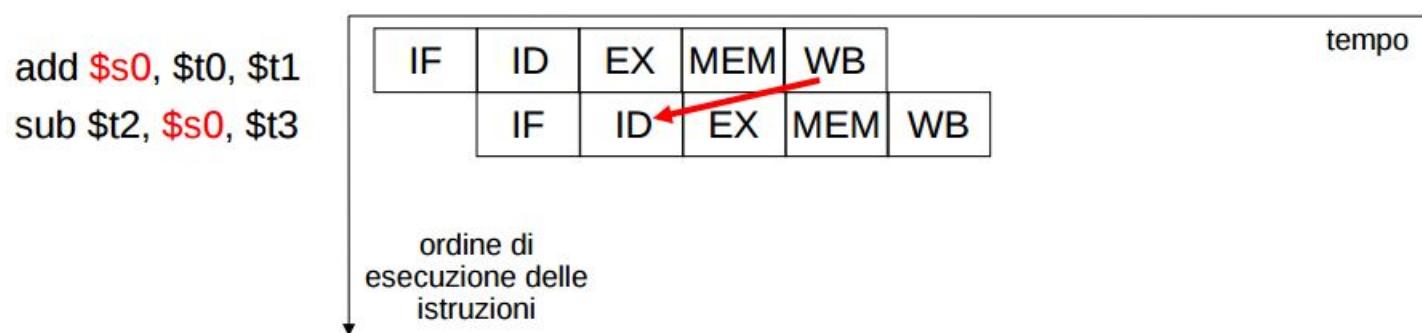


# Conflitti sui dati

- Un'istruzione dipende dal risultato di un'istruzione precedente che è ancora nella pipeline

Istruzione
add \$s0, \$t0, \$t1
sub \$t2, \$s0, \$t3

- Nella pipeline queste istruzioni vengono rappresentate come



- Due possibili soluzioni a questo tipo di conflitti sono
  - NOP Inserimento di istruzioni NOP per evitare il conflitto
  - Scheduling Cambiamento dell'ordine di esecuzione delle istruzioni mantenendo equivalenza funzionale

# Soluzioni ai conflitti sui dati

---

## ■ Soluzioni di tipo hardware

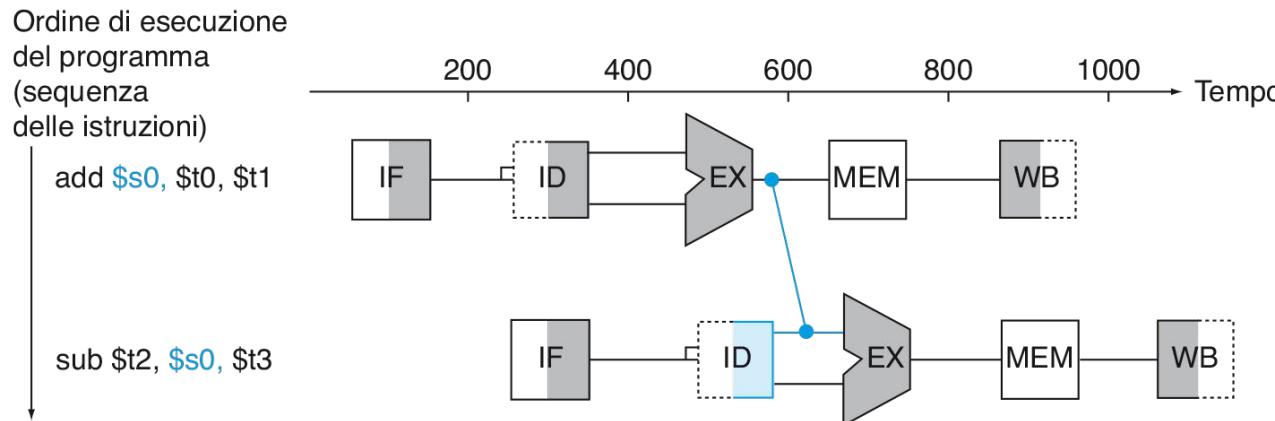
- Inserimento di bolle (**bubble**) o stalli nella pipeline
  - Si inseriscono dei tempi morti
  - Peggiora il throughput
- Propagazione o scavalcamento (**forwarding** o bypassing)
  - Si propagano i dati in avanti appena sono disponibili verso le unità che li richiedono

## ■ Soluzioni di tipo software

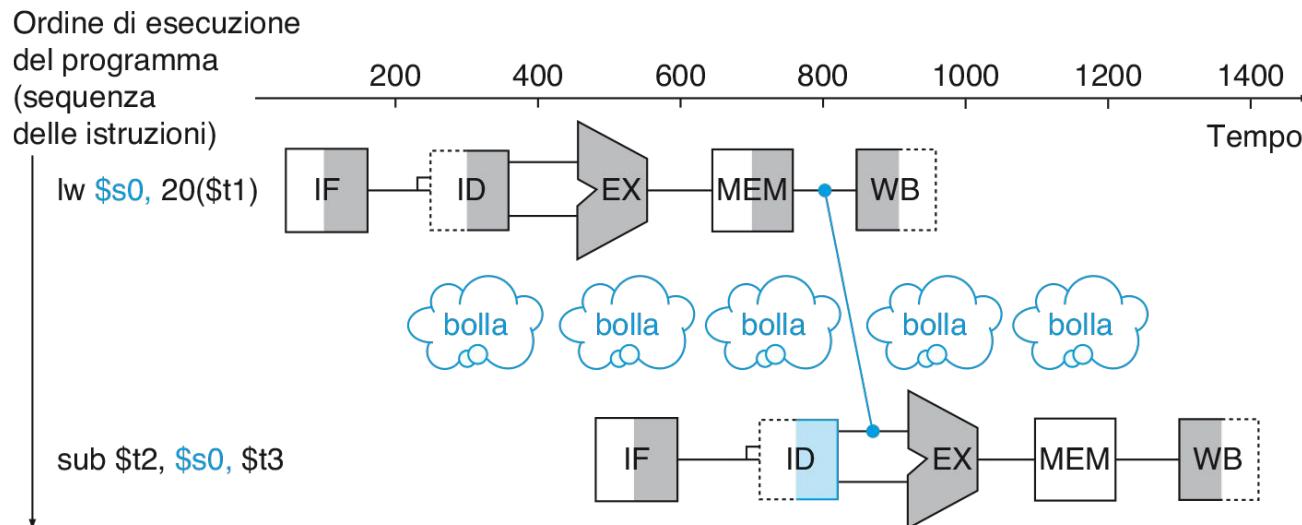
- Inserimento di istruzioni **nop** (no operation)
  - Peggiora il throughput
- Riordino delle istruzioni
  - Spostare istruzioni “innocue” in modo che esse eliminino la criticità

# Propagazione

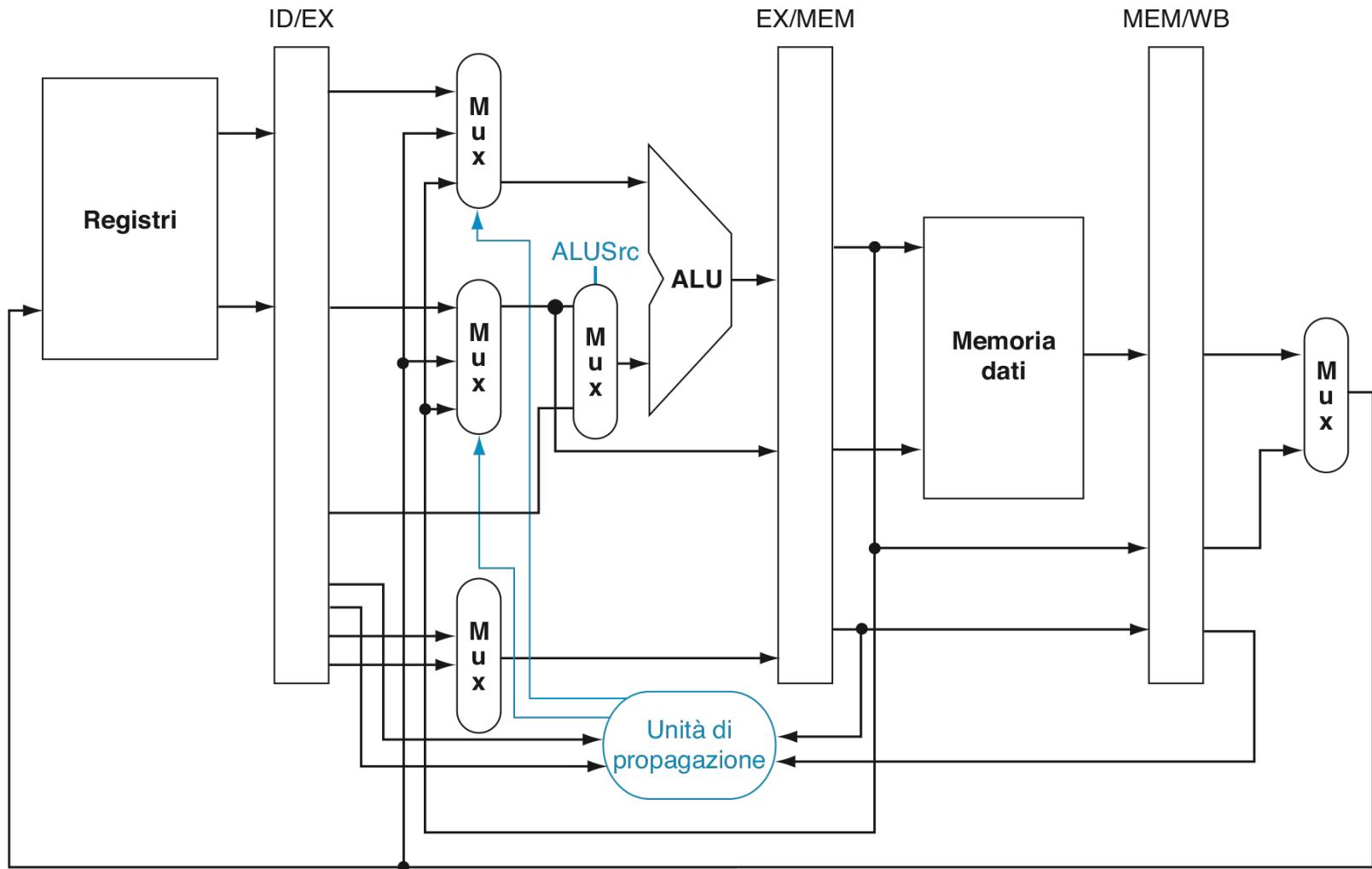
## ■ Propagazione di una istruzione aritmetica



## ■ Propagazione di un istruzione LW

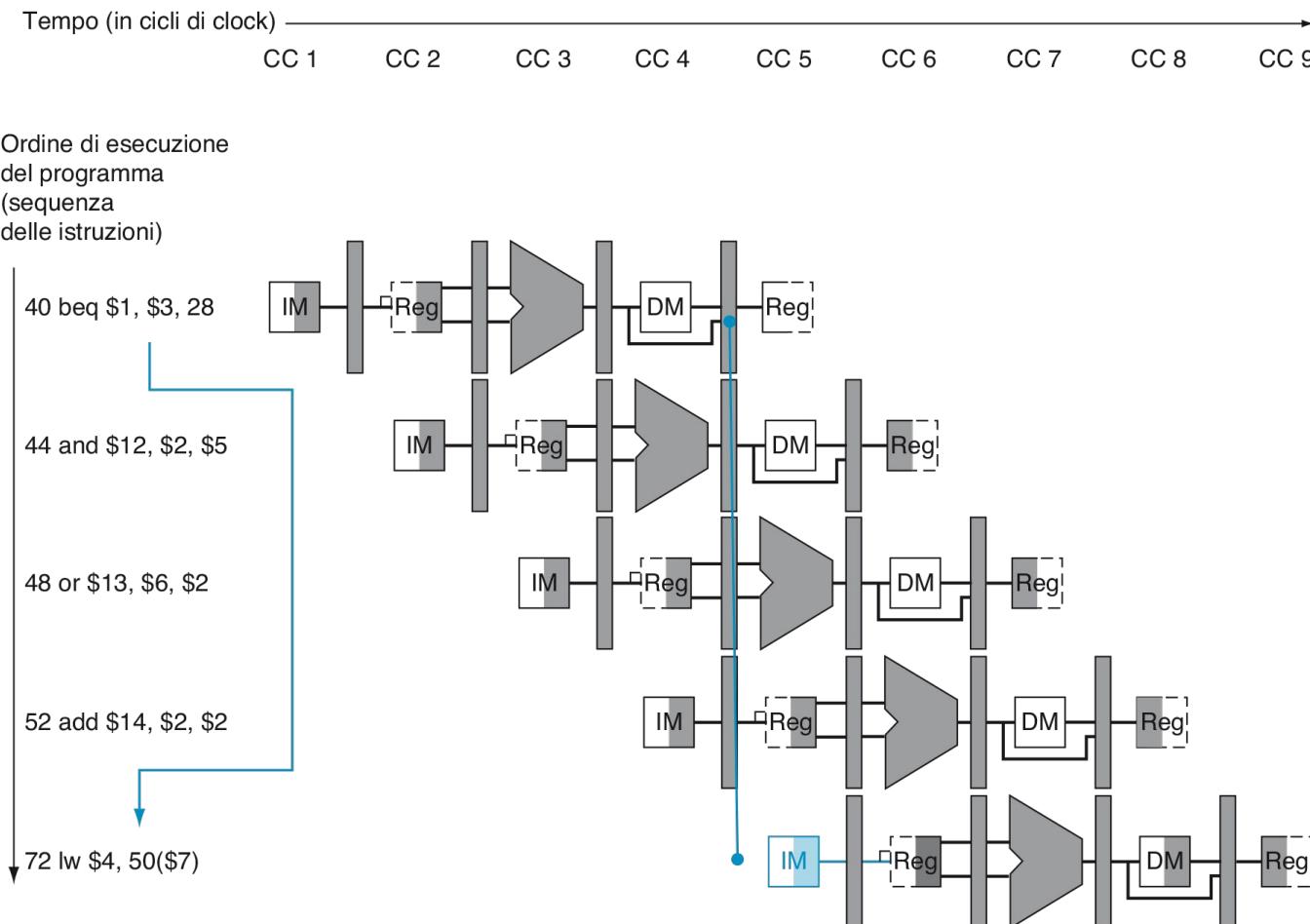


# Architettura con forwarding



# Conflitti condizionati

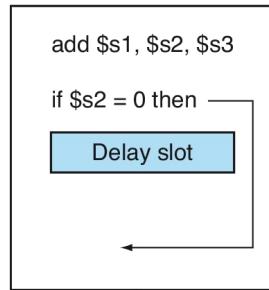
- Nel processore MIPS la decisione sul salto condizionato non viene presa fino al quarto passo (MEM) dell'istruzione beq



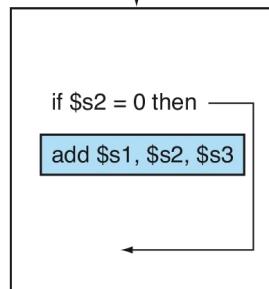
# Conflitti condizionati

- Per avere un esecuzione corretta da parte del processore si possono attuare due soluzioni:
  - NOP Inserimento di istruzioni NOP finché non conosco il risultato della branch
  - Delay Slot Cambio l'ordine di esecuzione delle istruzioni in modo da mantenere equivalenza funzionale e non inserire le NOP

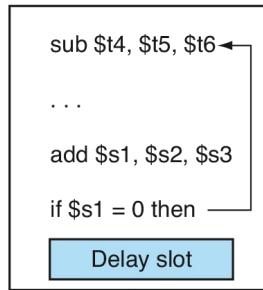
a. Da prima



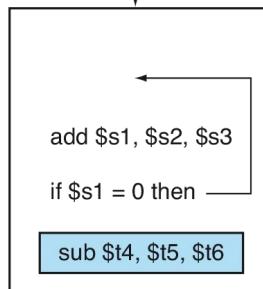
Diviene



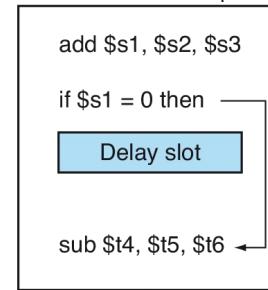
b. Dall'indirizzo di salto



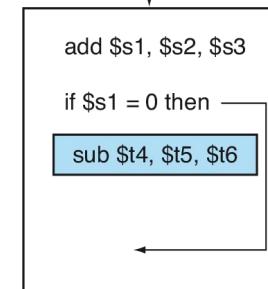
Diviene



c. Dall'indirizzo di salto, nel caso di fallimento della predizione



Diviene



# Soluzioni ai conflitti di controllo

---

## ■ Soluzioni di tipo hardware

- Inserimento di bolle (**bubble**) o stalli nella pipeline (3 cicli)
  - Si inseriscono dei tempi morti
  - Peggiora il throughput
- Ridurre i ritardi associati ai salti condizionati
  - Comparatore in fase di **decode**
  - Calcolo dell'indirizzo di destinazione in fase di **decode**
- Predizione Statica
  - Si assume **branch taken** o **branch not taken**
    - In caso di errore si **invalida** l'instruzione in esecuzione
- Predizione Dinamica
  - Comparatore in fase di **decode**
  - Tabella della storia dei salti ( **branch prediction buffer** )
  - Predizione a 1 o 2 bit