

# II Processore RISC-V

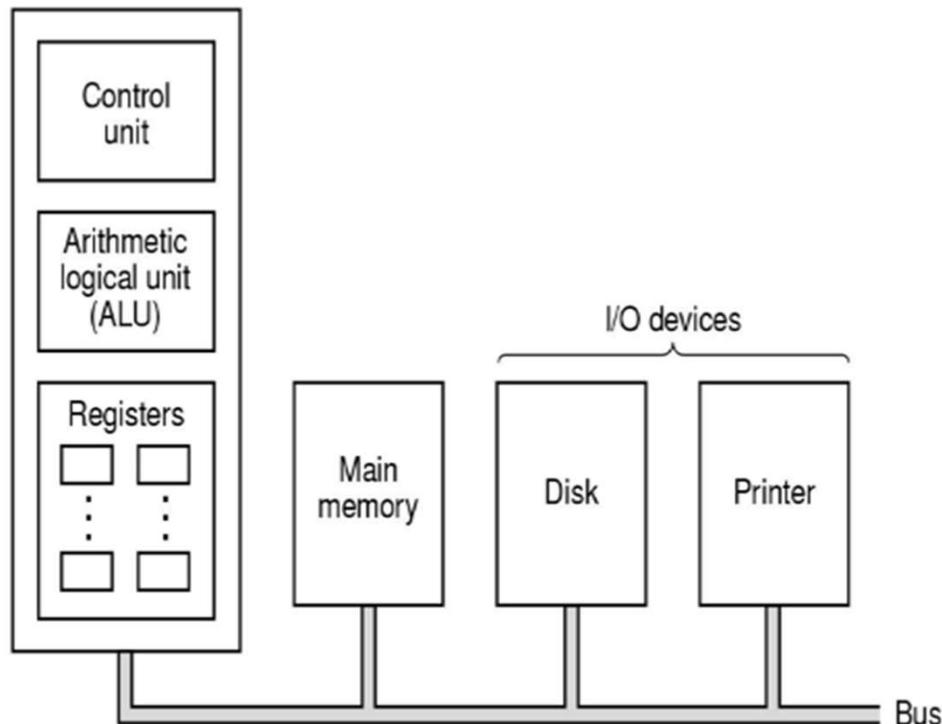
# Obiettivo

Costruzione di una CPU RISC-V che sia in grado di eseguire:

- le istruzioni aritmetico-logiche di tipo R: add, sub, and, or
  - add t0, t1, t2
  - sub t0, t1, t2
- le istruzioni di riferimento alla memoria load doubleword (ld) e store doubleword (sd);
  - ld t0, 24(t1)
  - sd t0, 24(t1)
- le istruzioni di salto condizionato dal risultato di un test di uguaglianza
  - beq t0, t1, etichetta

# Richiamo: Macchina di Von Neumann

Central processing unit (CPU)



- La CPU si compone di diverse parti distinte: **unità di controllo, unità aritmetico-logica, registri**
- I registri, l'unità aritmetico-logica e alcuni bus che li collegano compongono il **data path**
- Due registri importanti: Program Counter (PC) e Instruction Register (IR)
- La main memory contiene sia **istruzioni** sia dati usando **sequenze di bit**.

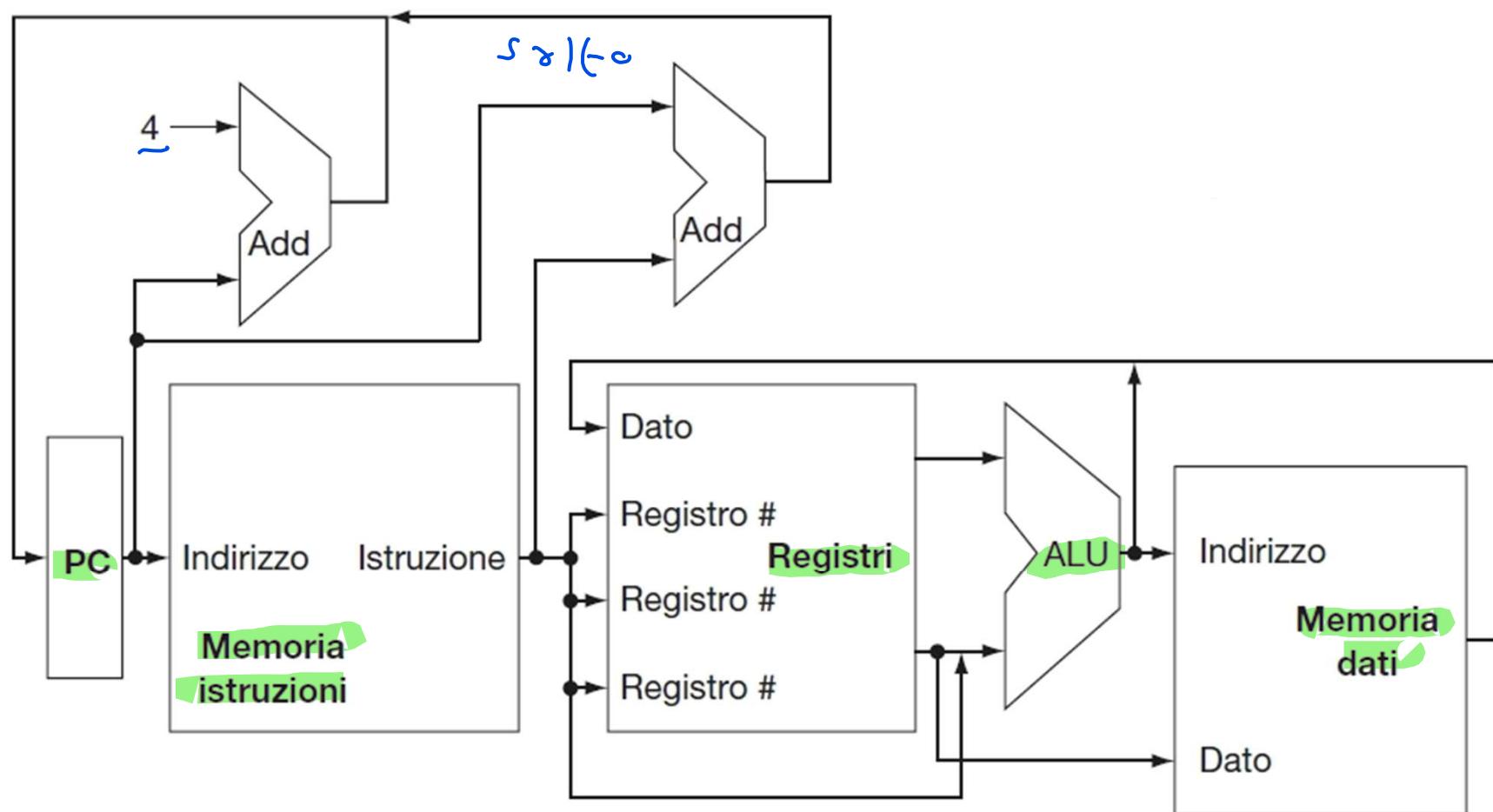
**Ciclo del data path: processo di far passare due operandi attraverso la ALU e memorizzarne il risultato**

## Richiamo: Ciclo di Fetch-Decode-Execute

La CPU esegue ogni istruzione per mezzo di una serie di passi elementari:

1. Prendi l'istruzione seguente dalla memoria
2. Cambia il PC (program counter) per indicare l'istruzione seguente
3. Determina il tipo dell'istruzione appena letta
4. Se l'istruzione usa una parola in memoria, determina dove si trova
5. Metti la parola, se necessario, in un registro della CPU
6. Esegui l'istruzione
7. Torna al punto 1 e inizia a eseguire l'istruzione successiva

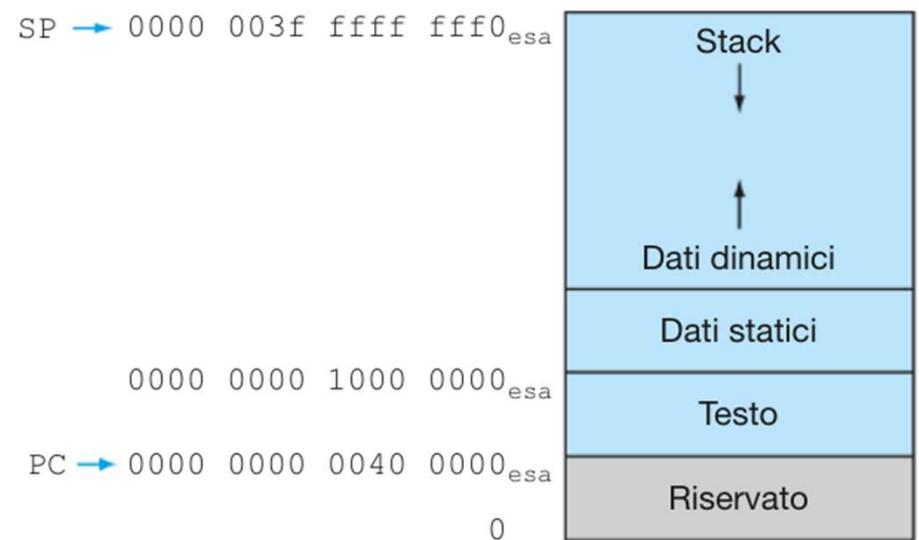
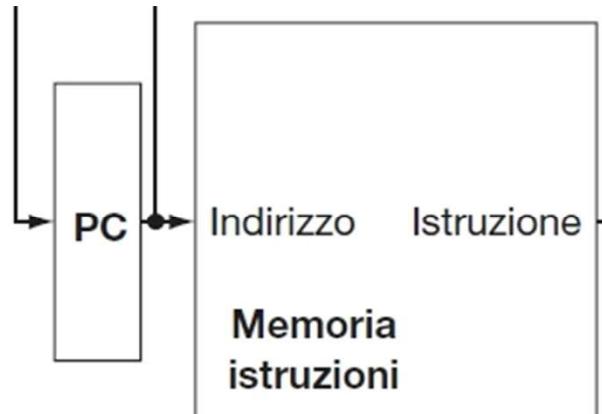
# Data Path RISC-V (Semplificato)



# Data Path RISC-V (Semplificato)

(fetch)

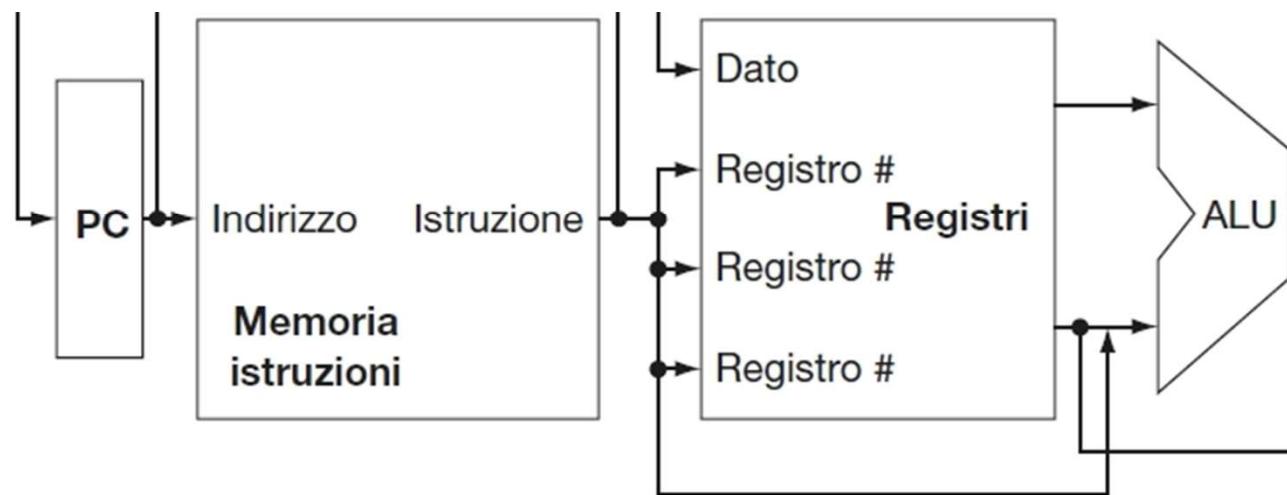
- Tutte le istruzioni iniziano utilizzando il PC per fornire il loro indirizzo alla memoria delle istruzioni
- Per semplificazione, assumiamo che la "memoria delle istruzioni" sia "read-only"
- Assumiamo anche che ci siano 2 memorie diverse:  
Una per le istruzioni ed un'altra per i dati.



# Data Path RISC-V (Semplificato)

(decode)

- La ALU deve "capire" il tipo di istruzione
- Per esempio, il numero d'ordine dei registri contenenti gli operandi può essere letto nei campi opportuni dell'istruzione stessa
- **L'unità di controllo (non mostrata nella figura) è la parte che effettivamente decodifica le istruzioni**

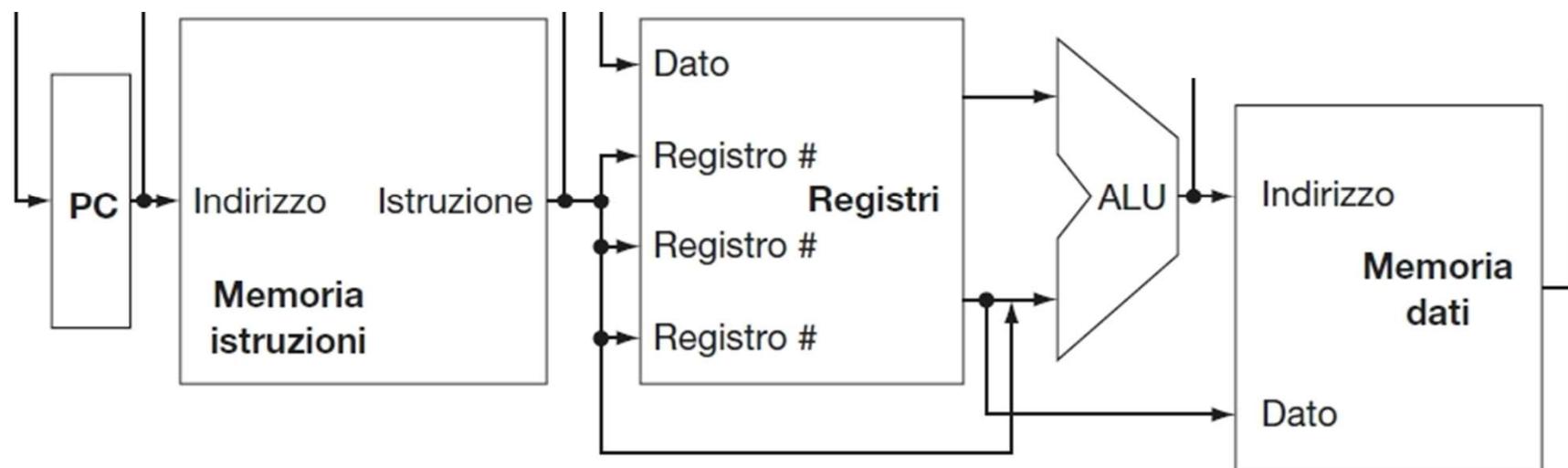


# Data Path RISC-V (Semplificato)

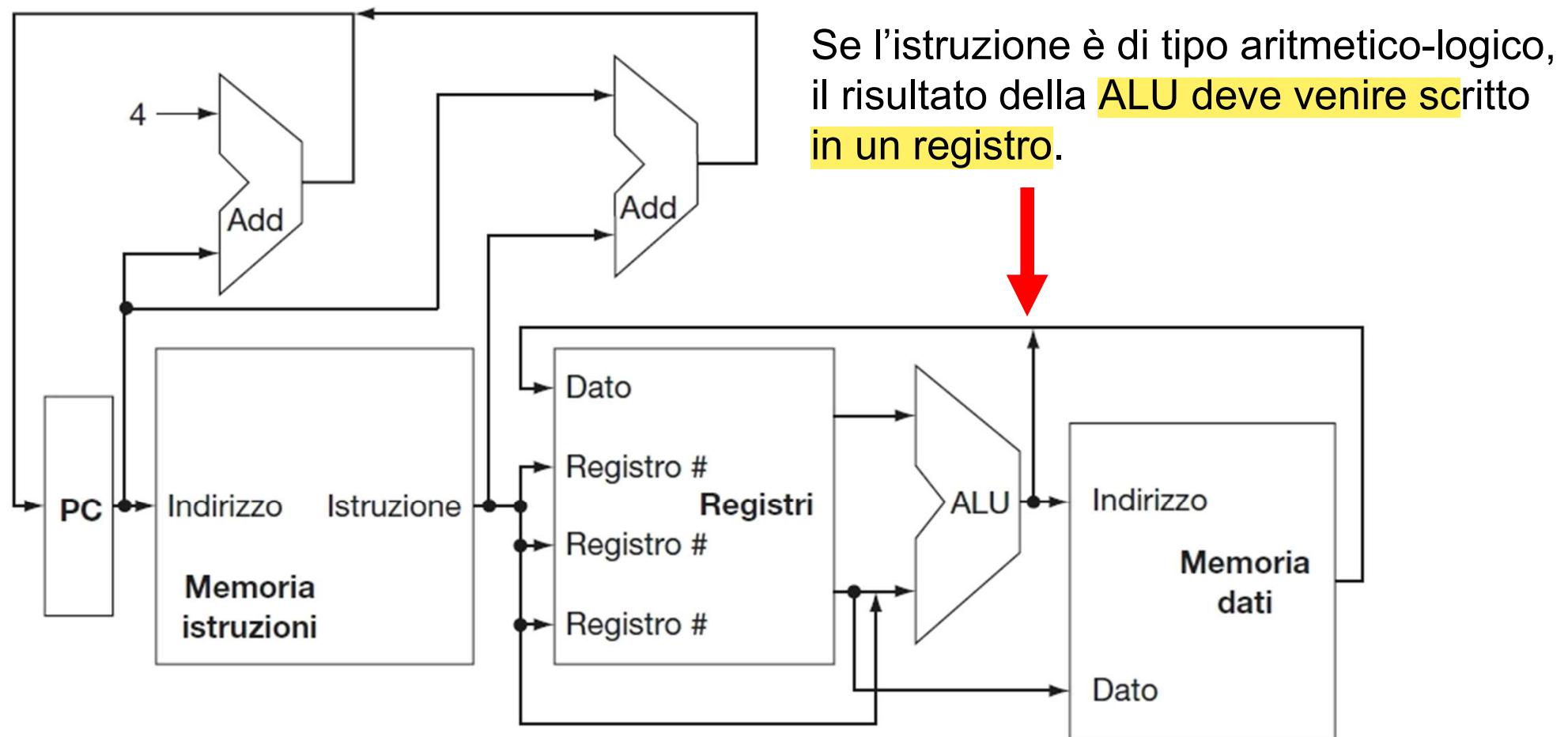
(execute)

Una volta caricati gli operandi, si può:

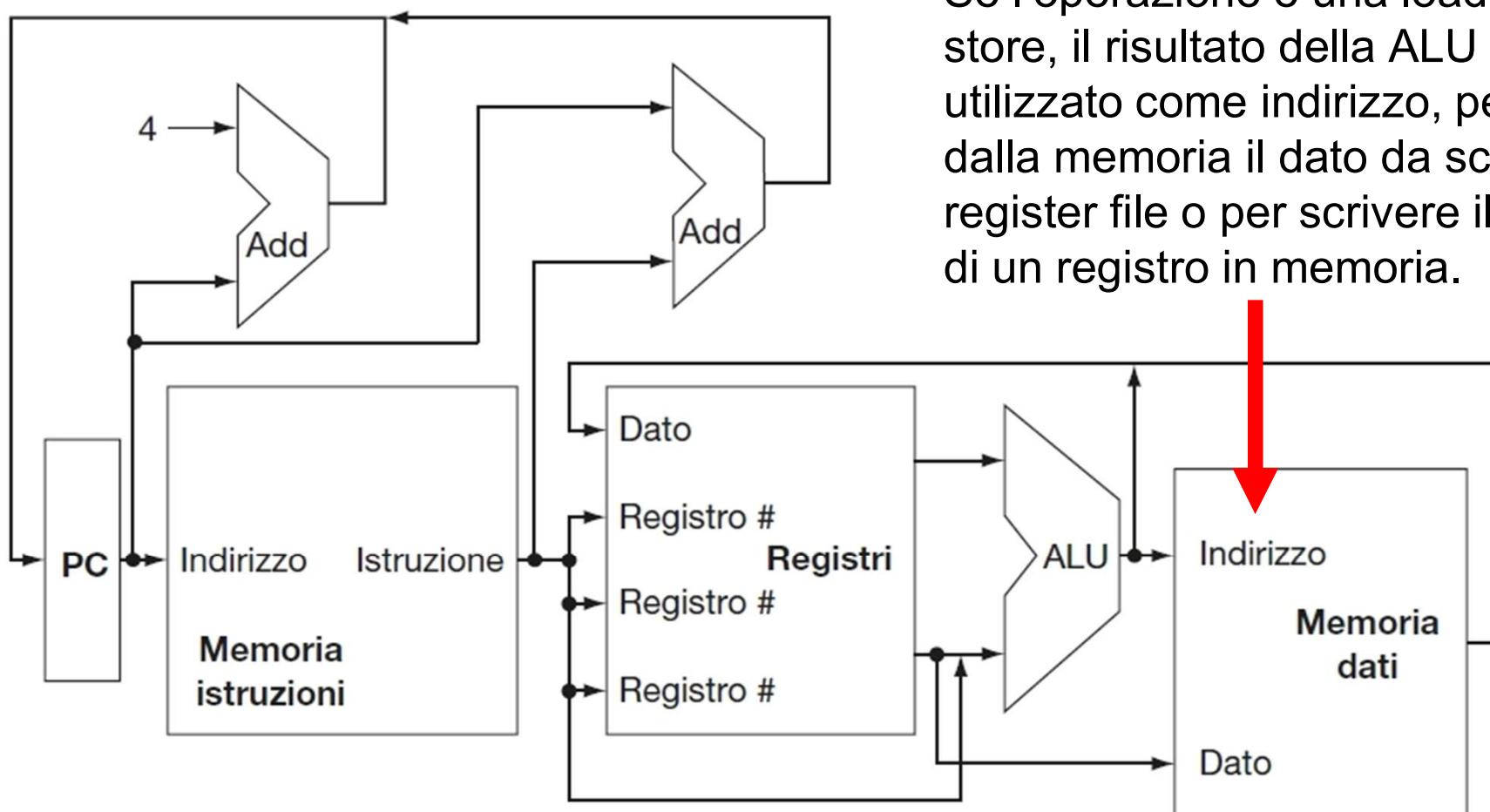
- eseguire un calcolo effettivo (operazioni aritmetico-logiche su interi)
- elaborare il loro contenuto per determinare un indirizzo di memoria (nel caso di load o store)
- eseguire un confronto (salto condizionato)



# Data Path RISC-V (Semplificato)

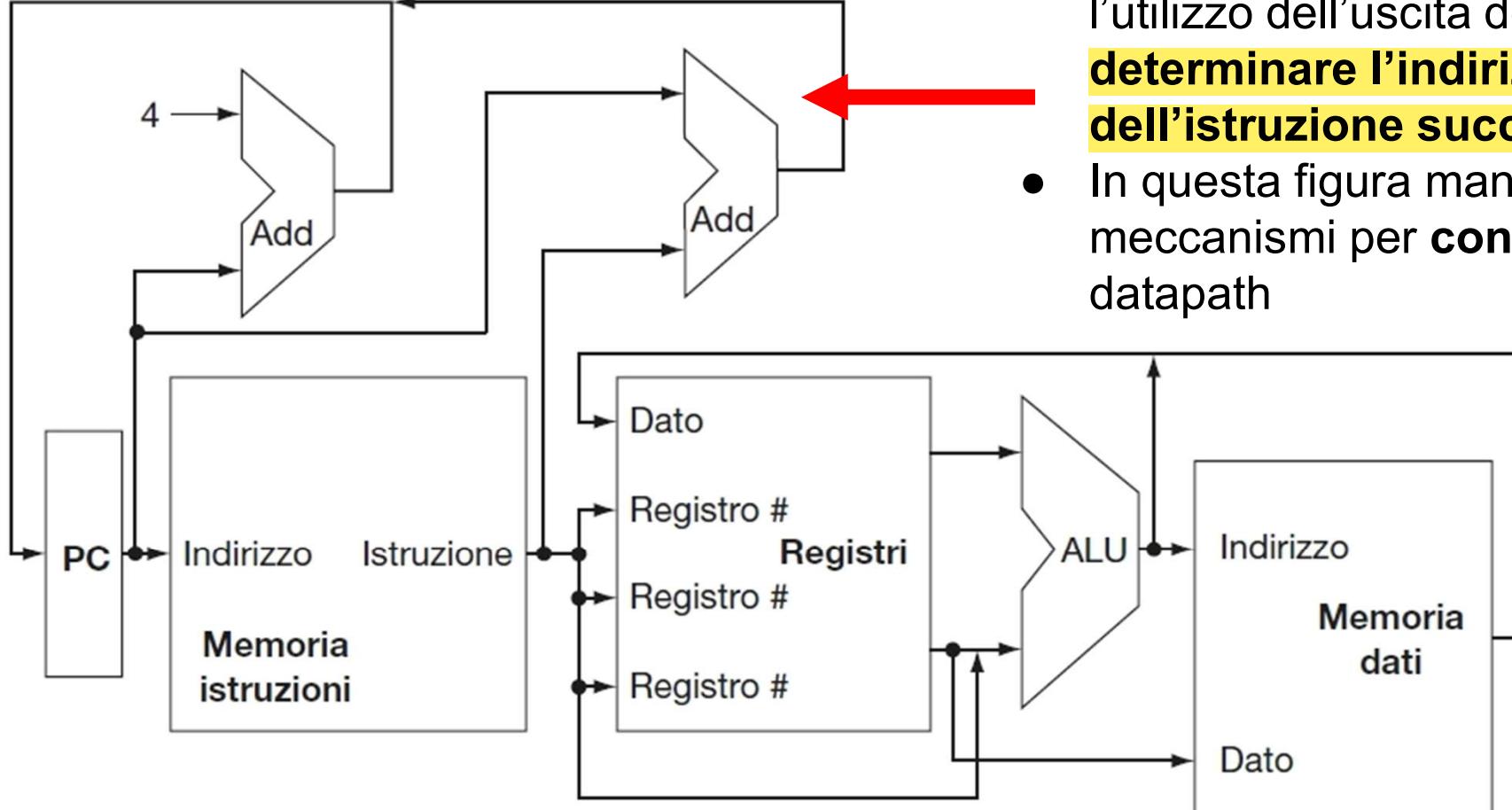


# Data Path RISC-V (Semplificato)



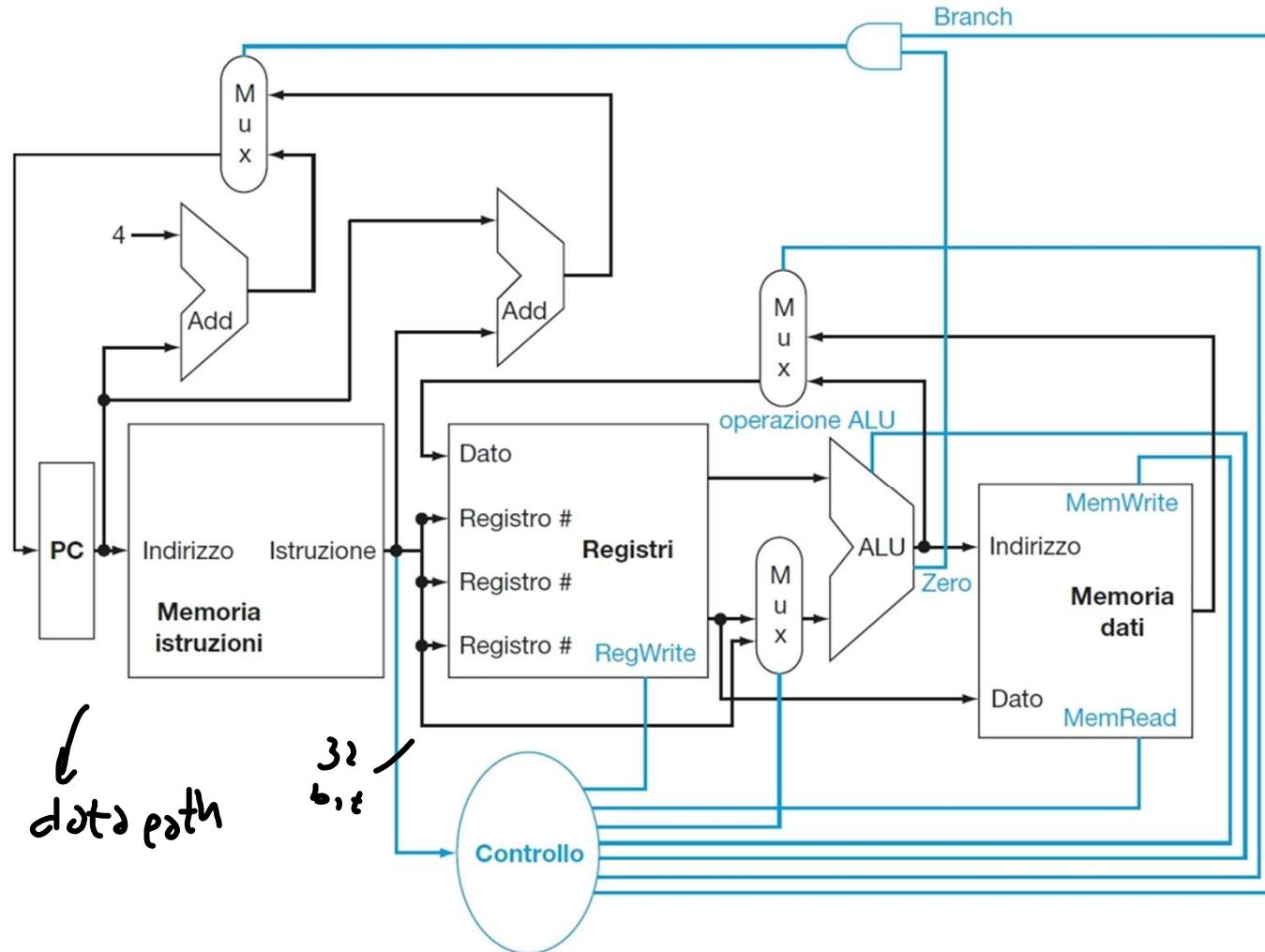
Se l'operazione è una load o una store, il risultato della ALU viene utilizzato come indirizzo, per leggere dalla memoria il dato da scrivere nel register file o per scrivere il contenuto di un registro in memoria.

# Data Path RISC-V (Semplificato)

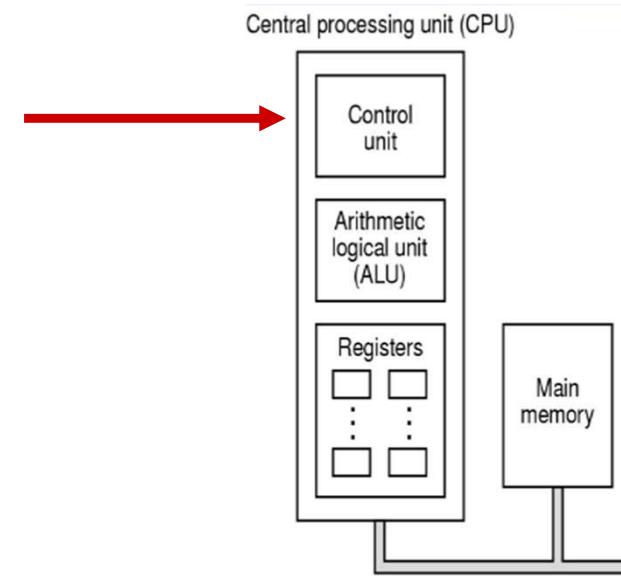


- I salti condizionati richiedono l'utilizzo dell'uscita della ALU per **determinare l'indirizzo dell'istruzione successiva**
- In questa figura mancano però i meccanismi per **controllare** il datapath

# L'unità di Controllo



Controllo  
Unità di controllo



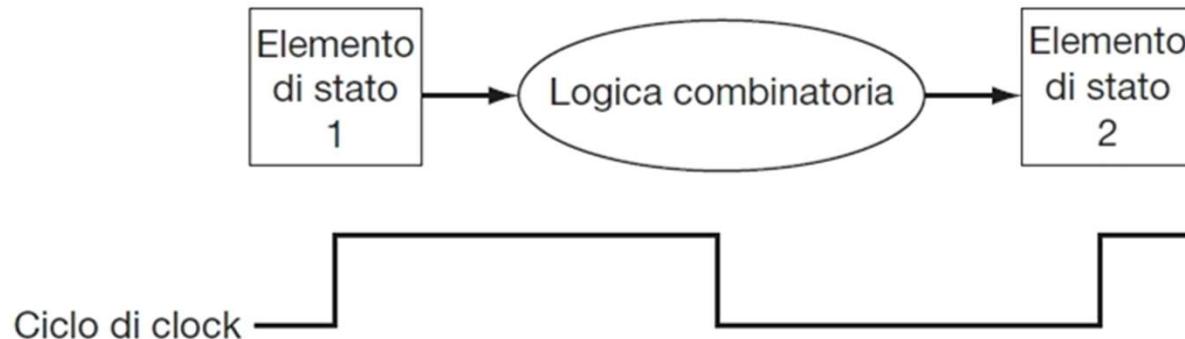
L'unità di controllo contiene i circuiti che effettivamente decodificano le istruzioni

Vedremo nel dettaglio tutti gli elementi della figura per capire l'architettura

# Metodologia di Temporizzazione

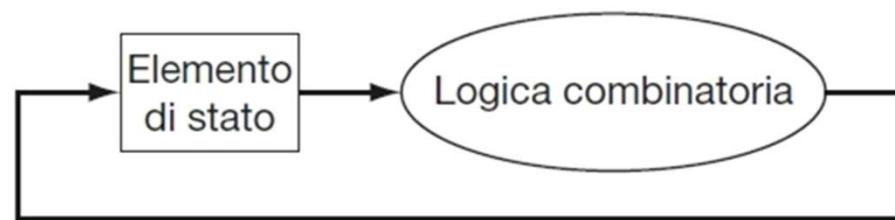
e' solo trucco

- l'approccio utilizzato per determinare quando un dato è valido e stabile in relazione al segnale di clock.
- **Temporizzazione sensibile ai fronti (edge-triggered)**: una tecnica di temporizzazione nella quale tutti i cambiamenti di stato avvengono su un fronte del segnale di clock  
*clip flop*
- In un sistema digitale **sincrono**, il segnale di clock determina quando gli elementi di stato scrivono la loro memoria interna



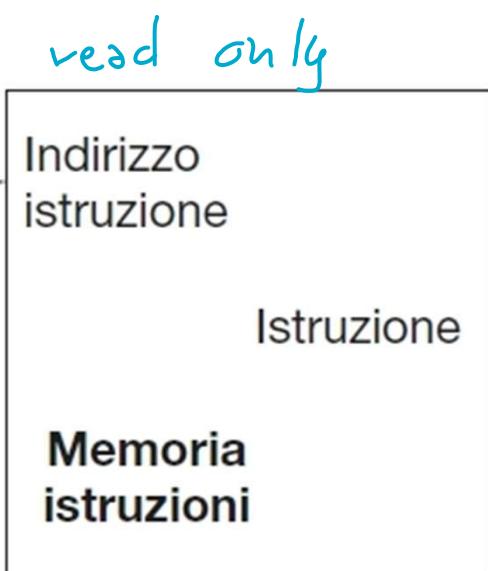
# Metodologia di Temporizzazione

- Il segnale di clock determina **quando gli elementi di stato scrivono la loro memoria interna**
- Gli ingressi a un elemento di stato **devono raggiungere un valore stabile** prima che il fronte attivo del clock provochi l'aggiornamento dello stato
- Supporremo che tutti gli elementi di stato, comprese le memorie, siano sensibili al **fronte di salita**
- La metodologia sensibile ai fronti permette di **leggere e scrivere un elemento di stato nello stesso ciclo di clock**
- Il ciclo di clock deve avere una **durata sufficiente a garantire che gli ingressi siano stabili** quando arriva il fronte attivo del clock

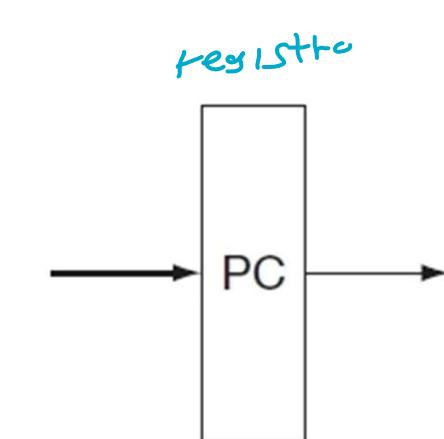


# Realizzazione di un'Unità di Elaborazione

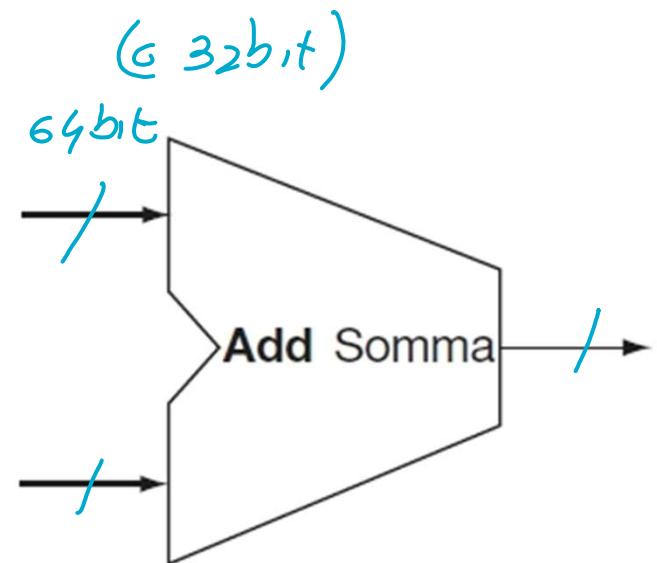
- Prima parte: **Il Program Counter**
- Ci servono alcuni circuiti digitali tra quelli che abbiamo già visto



a. Memoria istruzioni



b. Program counter (PC)



c. Adder

# Memoria delle Istruzioni

1



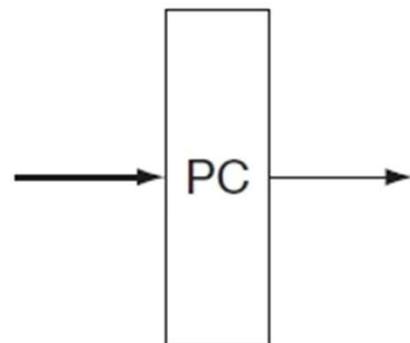
Un'unità di memoria in cui salvare le istruzioni del programma che sia in grado di fornire in uscita l'istruzione associata all'indirizzo dato in ingresso

a. Memoria istruzioni

# Il Progam Counter

2

- Registro utilizzato per memorizzare l'indirizzo dell'istruzione corrente
- La memoria è "**byte addressed**"
- **Registro da 64 bit in questa architettura**



b. Program counter (PC)

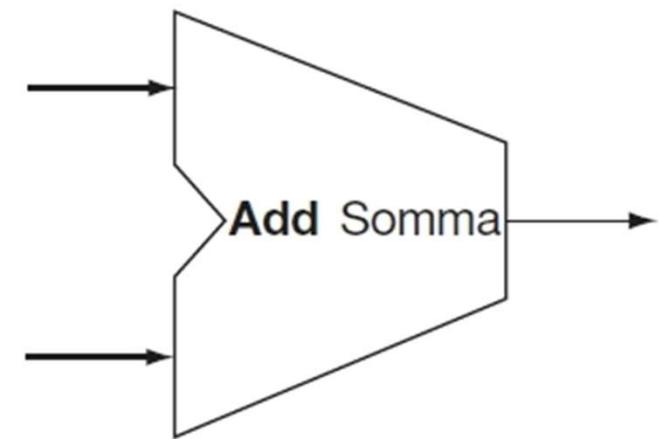
3

# Incremento del Program Counter

- Un sommatore per incrementare il PC e ottenere l'indirizzo dell'istruzione successiva
  - Tutte le istruzioni RISC-V sono rappresentate con 32 bit
  - Quindi, nel caso più comune dobbiamo sommare 4 byte al Program Counter

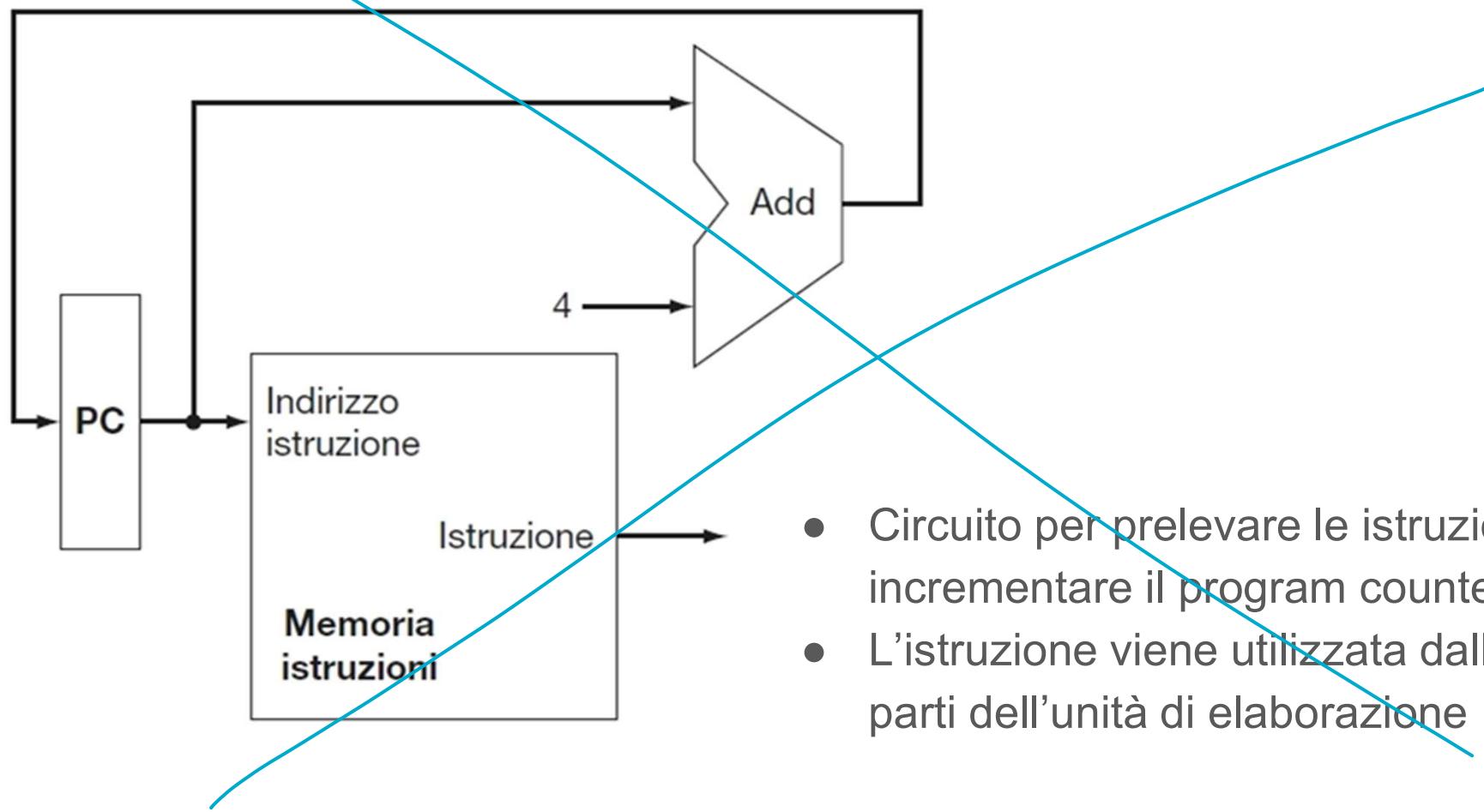
## **FORMATO DELLE ISTRUZIONI CORE**

	31	27	26	25	24	20	19	15	14	12	11	7	6	0				
R	funz7		rs2		rs1		funz3		rd		codop							
I	cost[11:0]				rs1		funz3		rd		codop							
S	cost[11:5]		rs2		rs1		funz3		cost[4:0]		codop							
SB	cost[12 10:5]		rs2		rs1		funz3		cost[4:1 11]		codop							
U	cost[31:12]								rd		codop							
UJ	cost[20 10:1 11 19:12]								rd		codop							

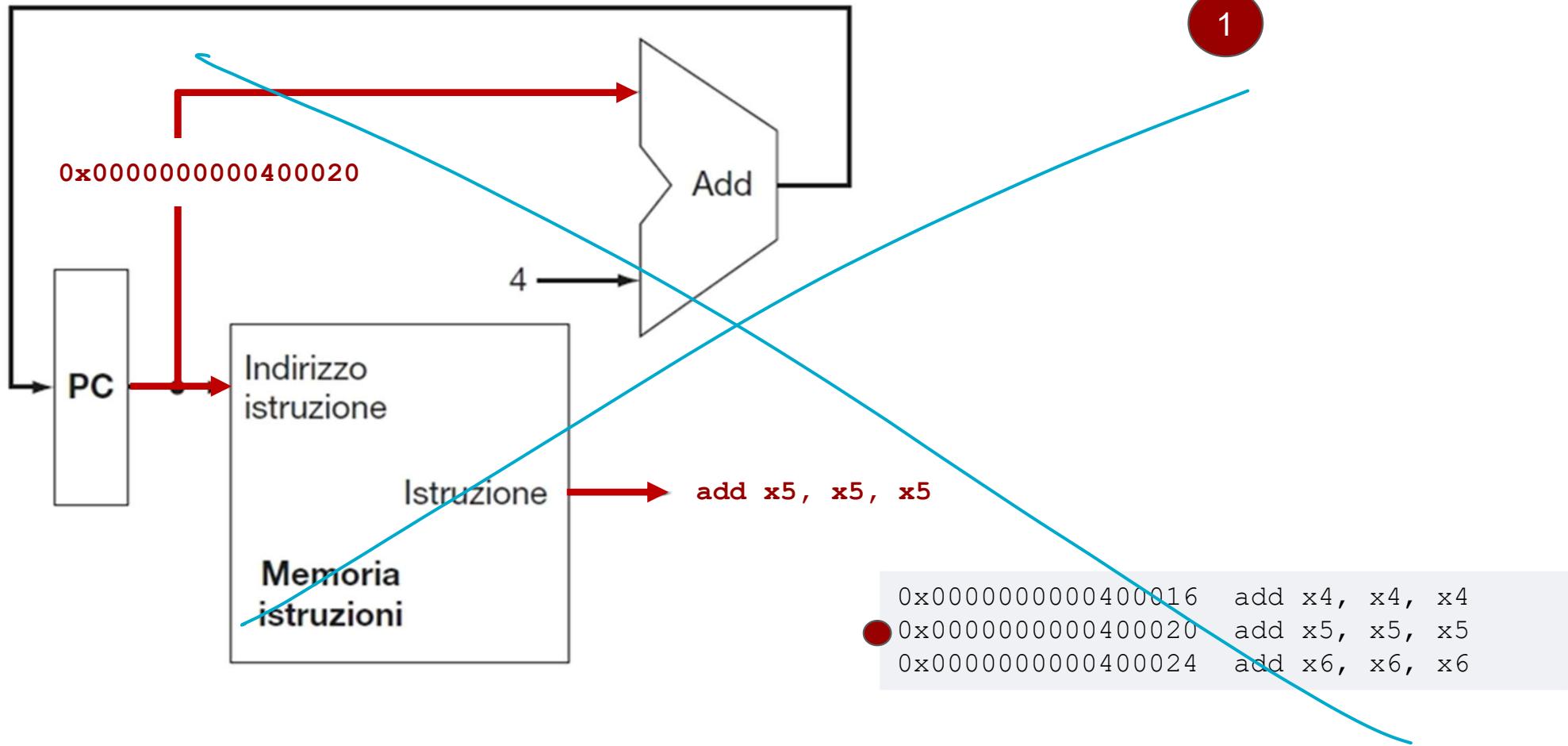


c. Adder

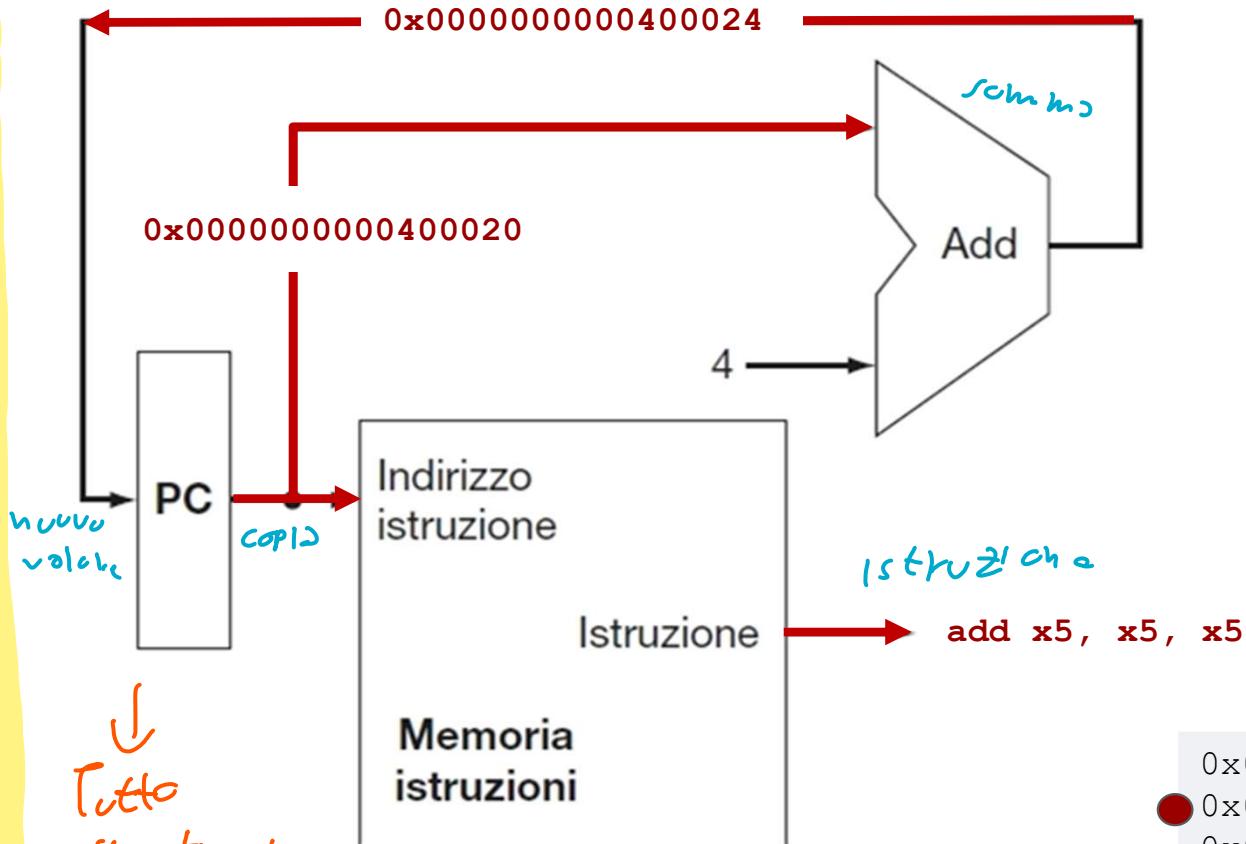
# Circuito per Incrementare il Program Counter



# Incremento del Program Counter



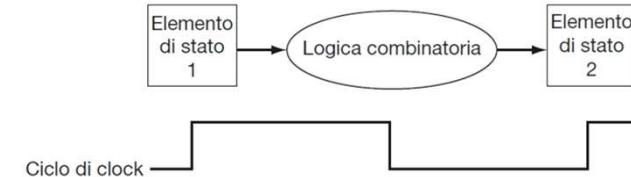
# Incremento del Program Counter



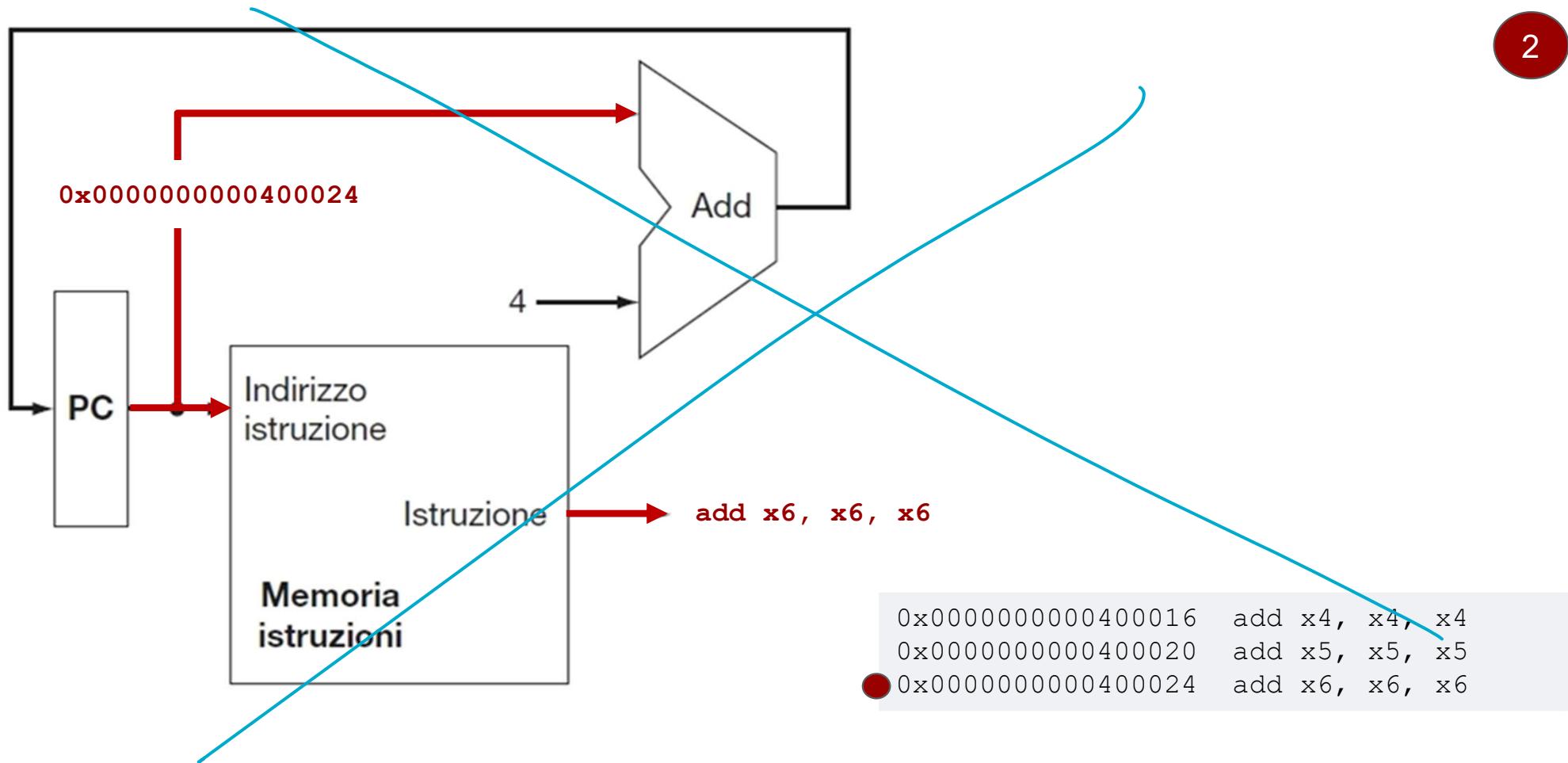
Tutto su fronte  
salta



0x0000000000400016	add x4, x4, x4
0x0000000000400020	add x5, x5, x5
0x0000000000400024	add x6, x6, x6

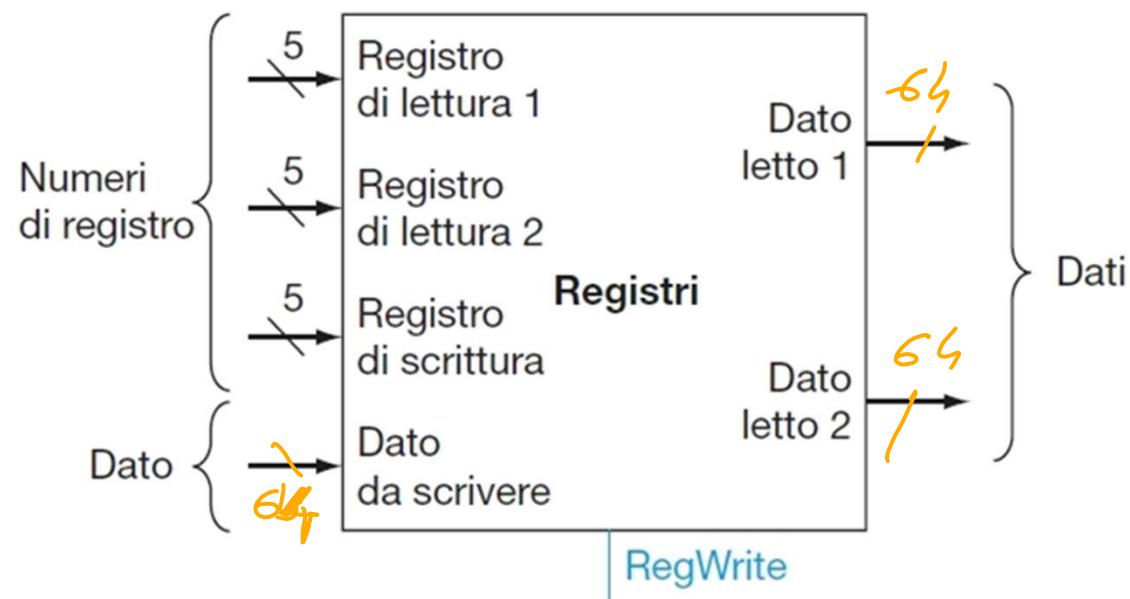


# Incremento del Program Counter

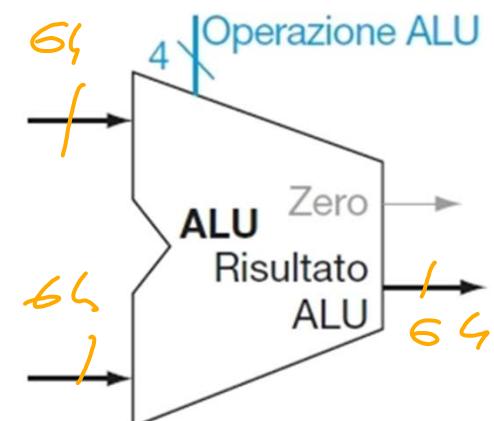


# Datapath per le Istruzioni di Tipo R

- Ci servono altri due circuiti digitali (già visti)
- Register File e la ALU

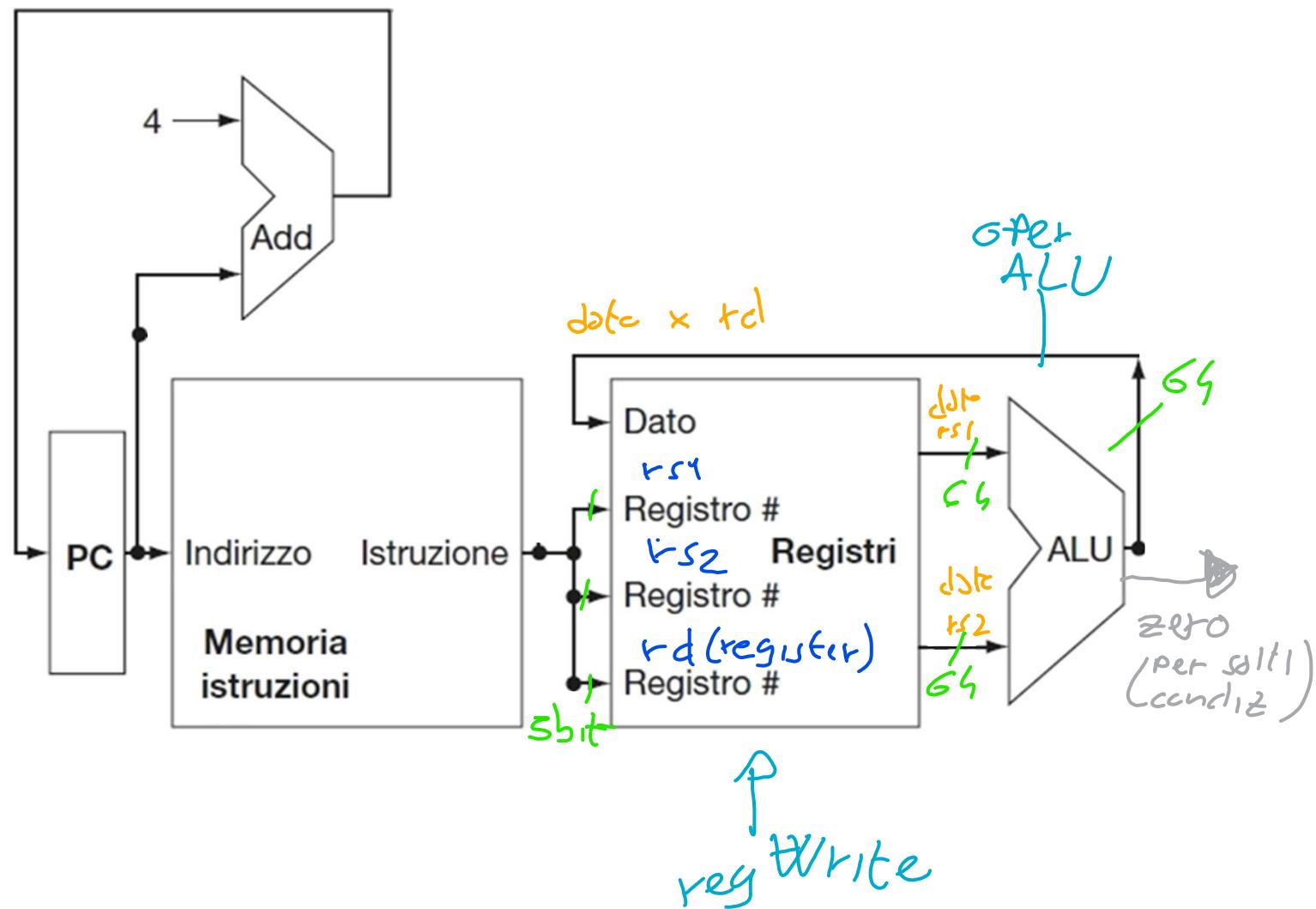


a. Registri

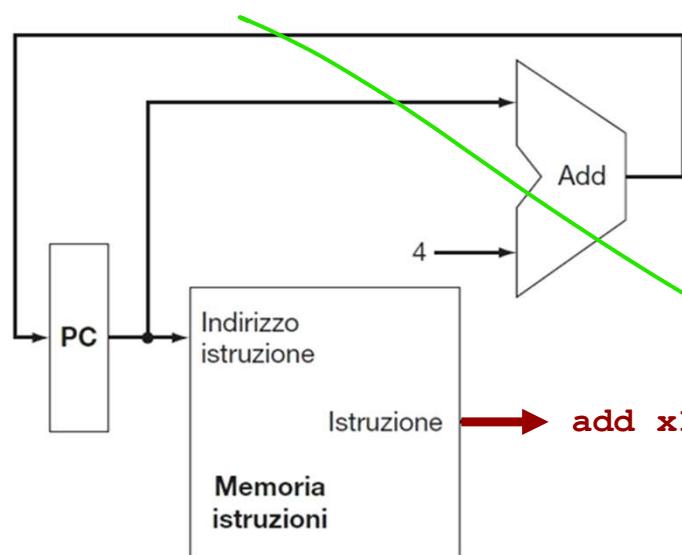


b. ALU

# Datapath per le Istruzioni di Tipo R



# Datapath per le Istruzioni di Tipo R



Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
add	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

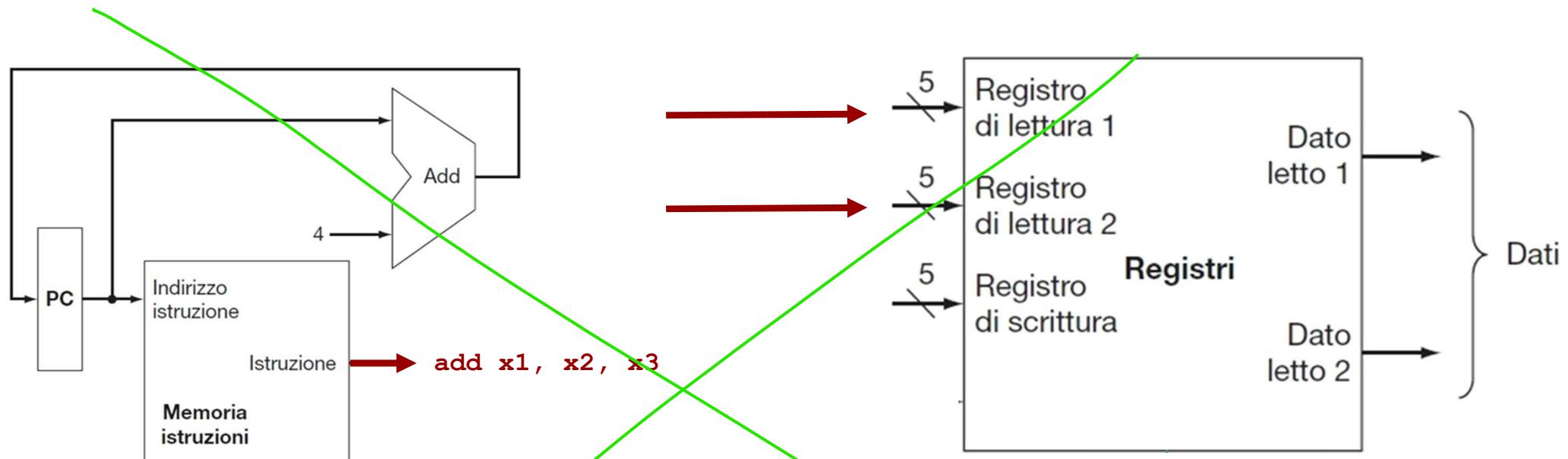
↳dif

> 3

X<sub>2</sub>

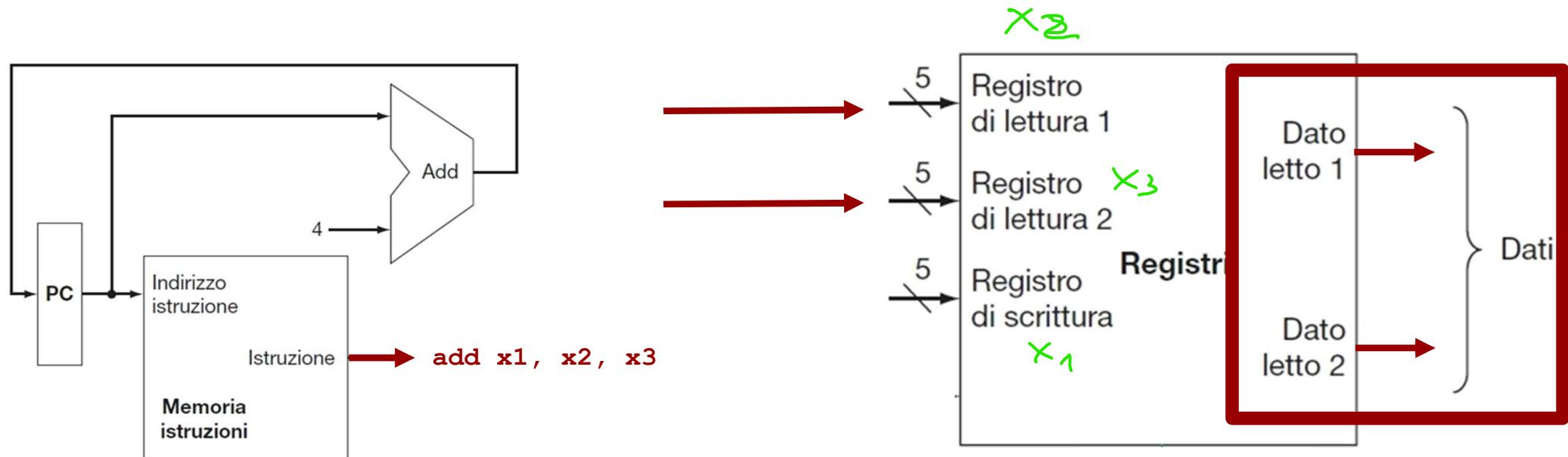
X~

# Datapath per le Istruzioni di Tipo R



Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
add	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

# Datapath per le Istruzioni di Tipo R

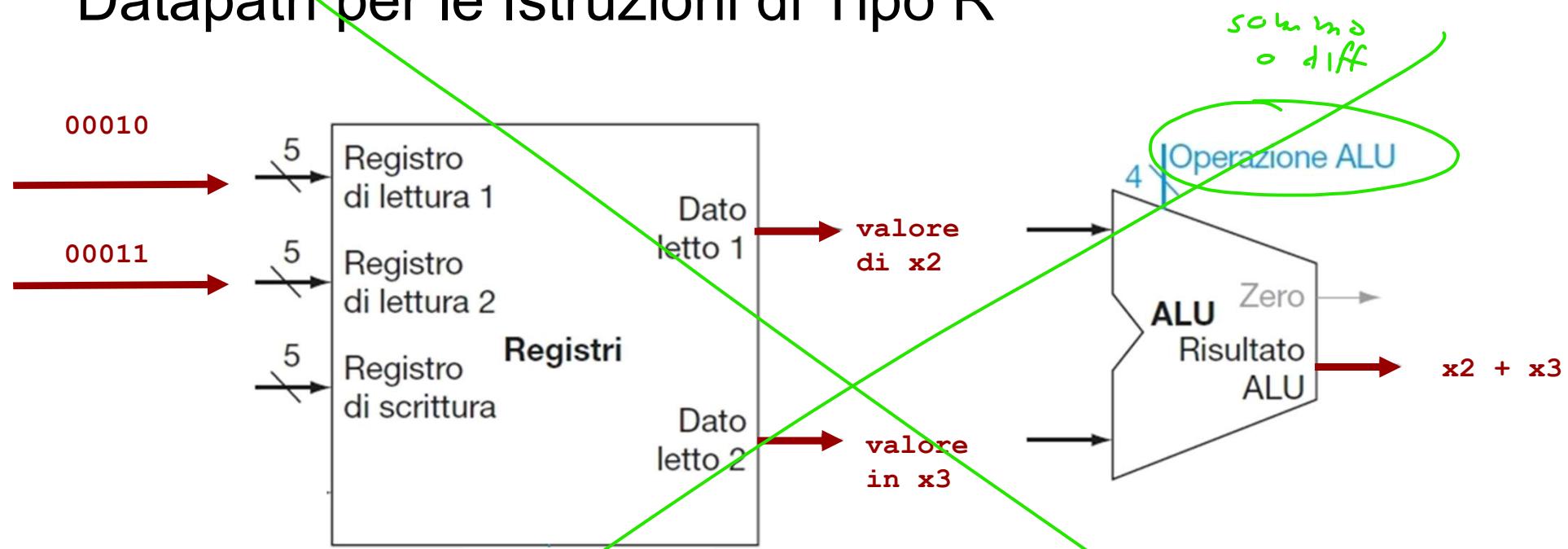


Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
add	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

Annotations in green:

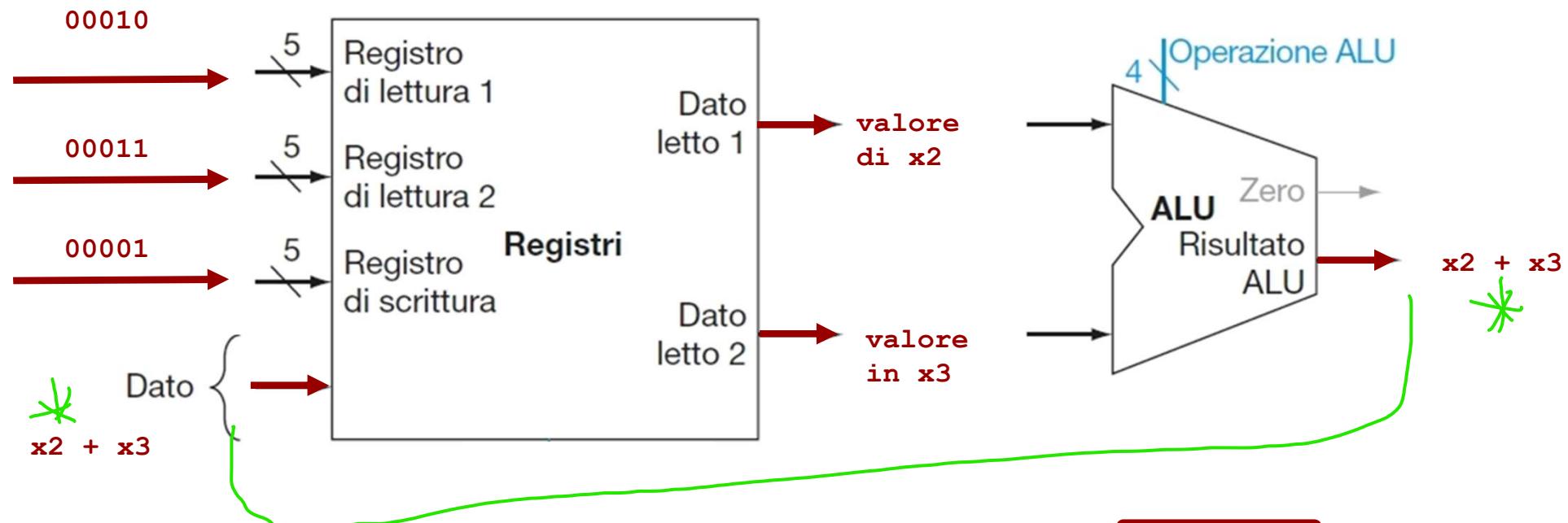
- A green arrow points from the 'dif' label to the 'funz7' column.
- The 'rs2' and 'rs1' columns are highlighted with a red box.
- The 'x3' label is written above the 'rs1' column.
- The 'x2' label is written above the 'rd' column.
- The 'x1' label is written above the 'codop' column.

# Datapath per le Istruzioni di Tipo R



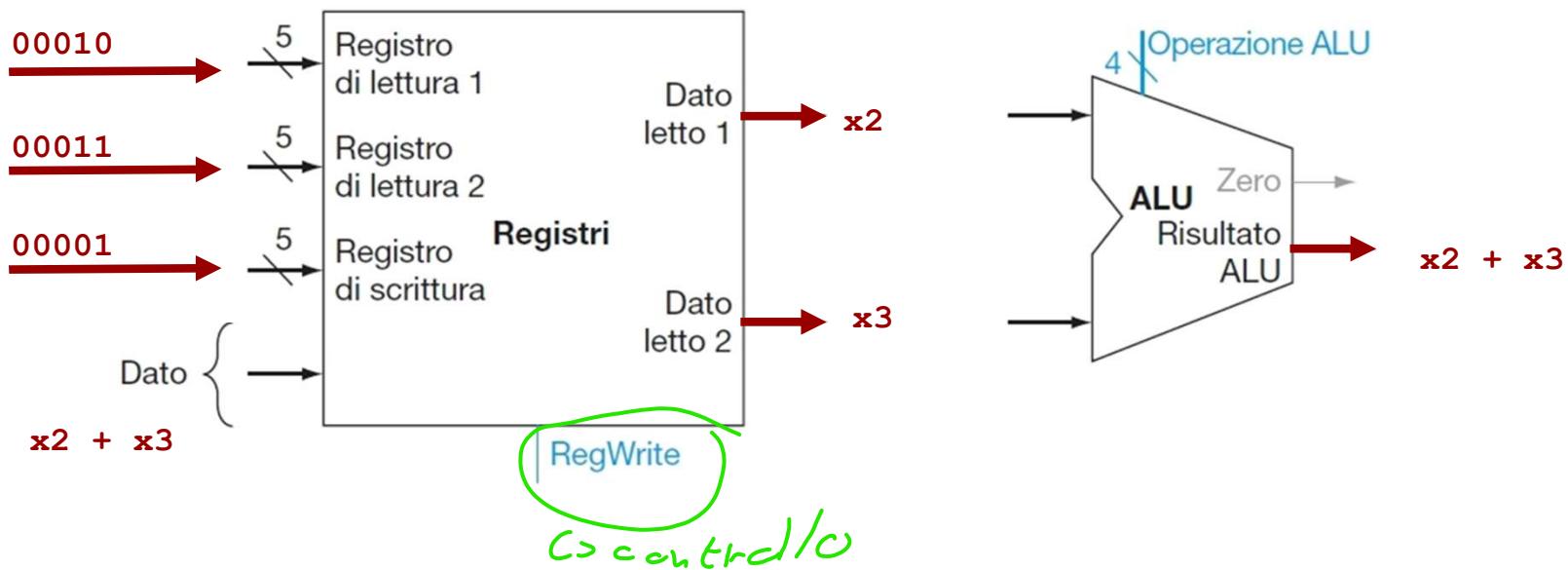
Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
add	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

# Datapath per le Istruzioni di Tipo R



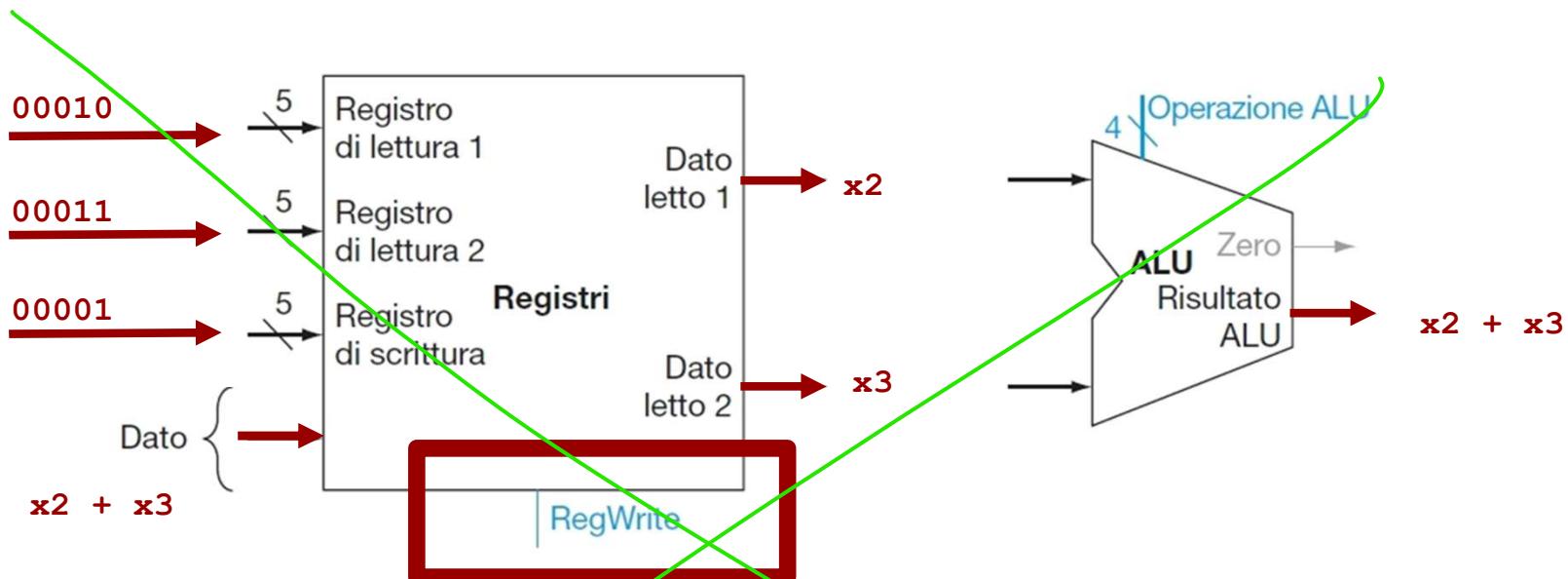
Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
add	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

# Datapath per le Istruzioni di Tipo R



Per implementare le operazioni di **Tipo R** servono il **register file** e la **ALU**. Il register file contiene tutti i registri e dispone di due porte di lettura e una di scrittura.

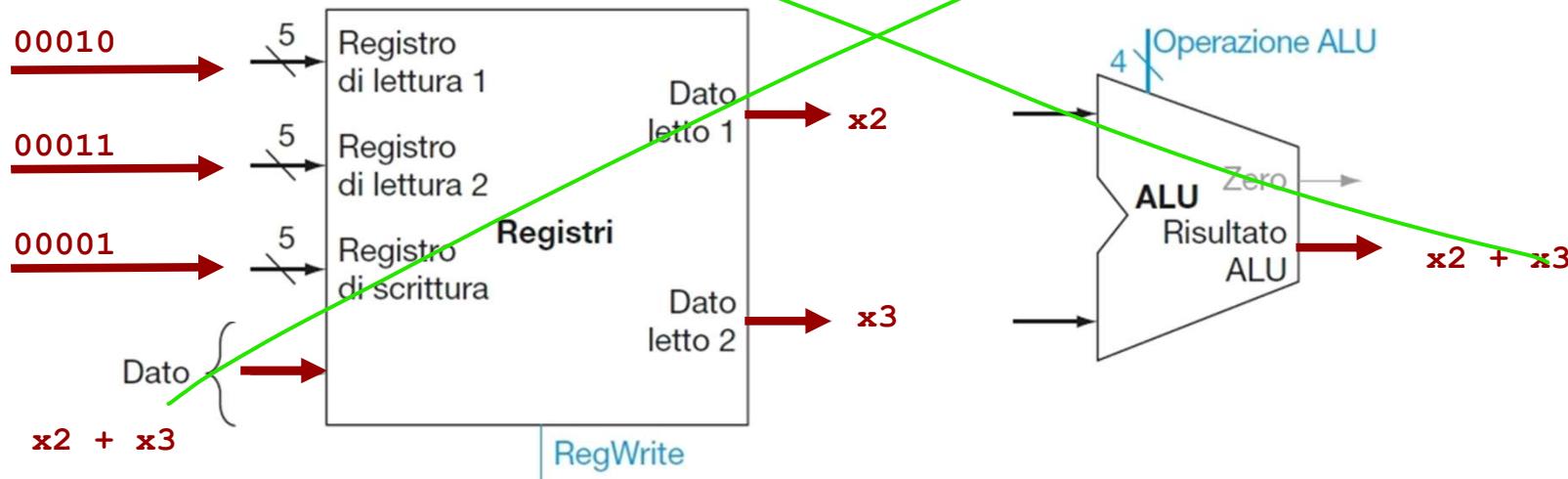
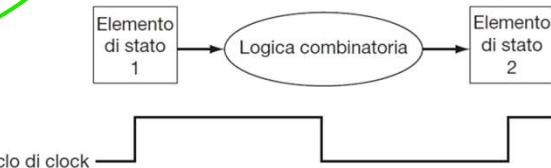
# Datapath per le Istruzioni di Tipo R



In ogni istante il register file fornisce in uscita il contenuto dei registri corrispondenti agli ingressi "Registro di lettura", senza richiedere alcun segnale di controllo.

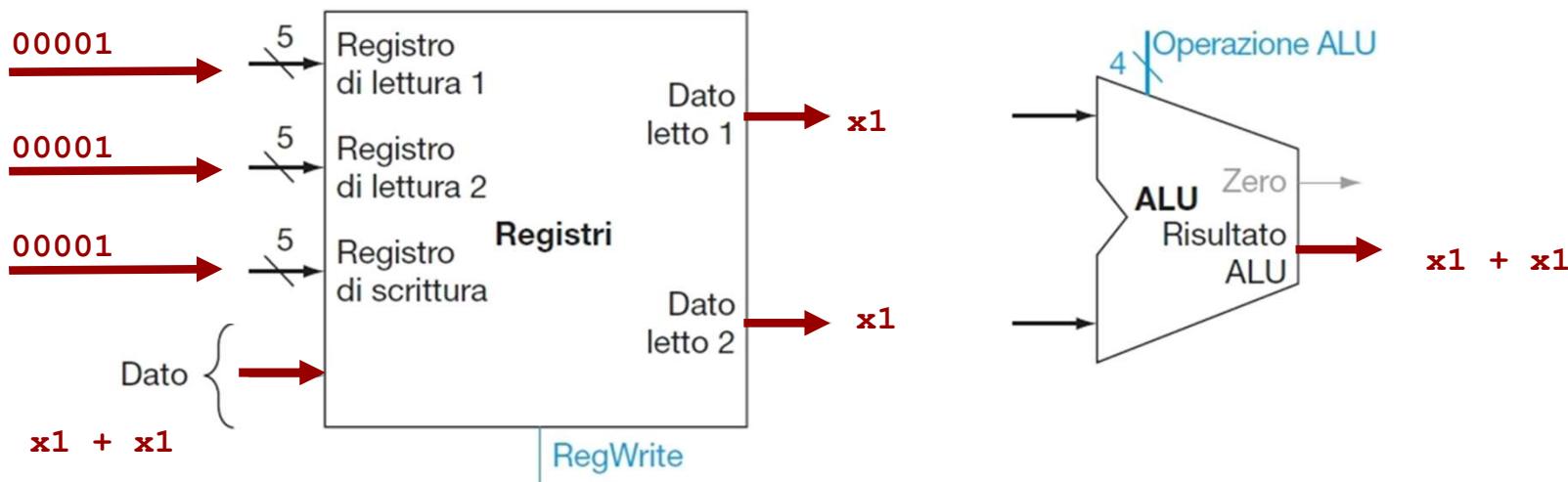
La scrittura di un registro deve essere indicata esplicitamente asserendo il segnale di controllo della scrittura, "**RegWrite**"

# Datapath per le Istruzioni di Tipo R



La scrittura avviene sul **fronte attivo del clock**, per cui tutti gli ingressi interessati (cioè il dato da scrivere, il numero di registro e il segnale di controllo) devono essere validi sul fronte attivo del clock.

# Datapath per le Istruzioni di Tipo R



Essendo la scrittura attiva sui fronti, l'architettura può leggere e scrivere lo stesso registro nello stesso ciclo di clock (e.g., **add  $x_1, x_1, x_1$  è lecita**).

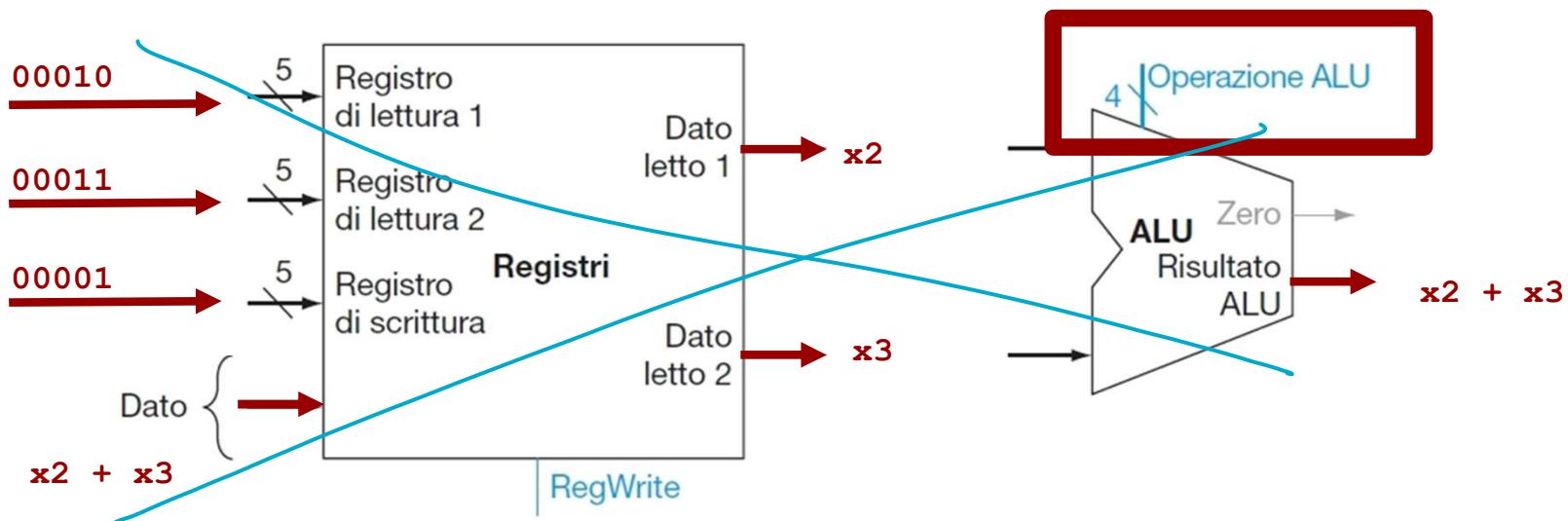
**La lettura fornirà il dato** contenuto nel registro **scritto in un ciclo di clock precedente**, mentre il valore scritto potrà essere letto nel ciclo di clock successivo.

# Datapath per le Istruzioni di Tipo R



Gli ingressi che specificano al register file il numero d'ordine dei registri hanno  
ampiezza di 5 bit, mentre i bus che trasportano i dati sono ampi 64 bit.

# Datapath per le Istruzioni di Tipo R

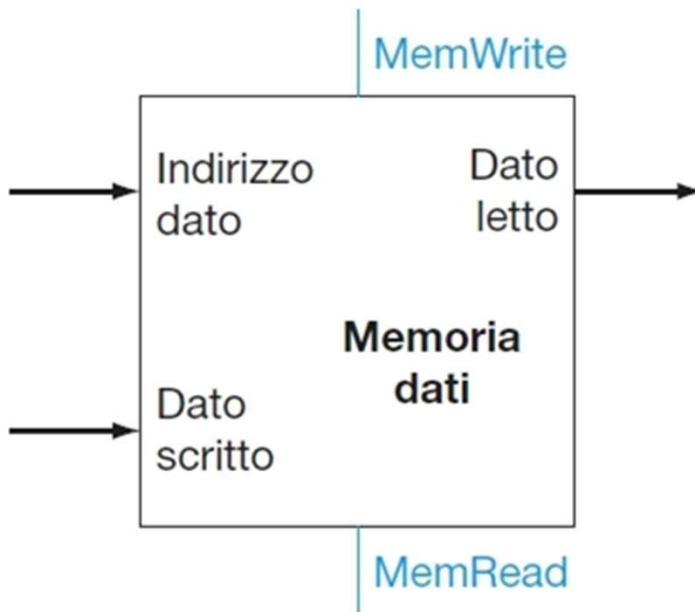


L'operazione che la ALU deve eseguire viene specificata dal segnale "**Operazione ALU**", che è ampio 4 bit. Il circuito di controllo della ALU sarà visto nel dettaglio dopo.

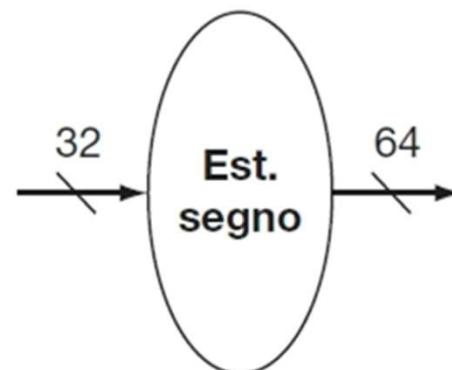
L'uscita "Zero" della ALU, verrà utilizzata per implementare i salti condizionati.

# Istruzioni Load (Tipo I) e Store (Tipo S)

- Ci servono ancora altri due circuiti digitali
- La **Memoria Dati** e un'**Unità di Estensione del Segno**



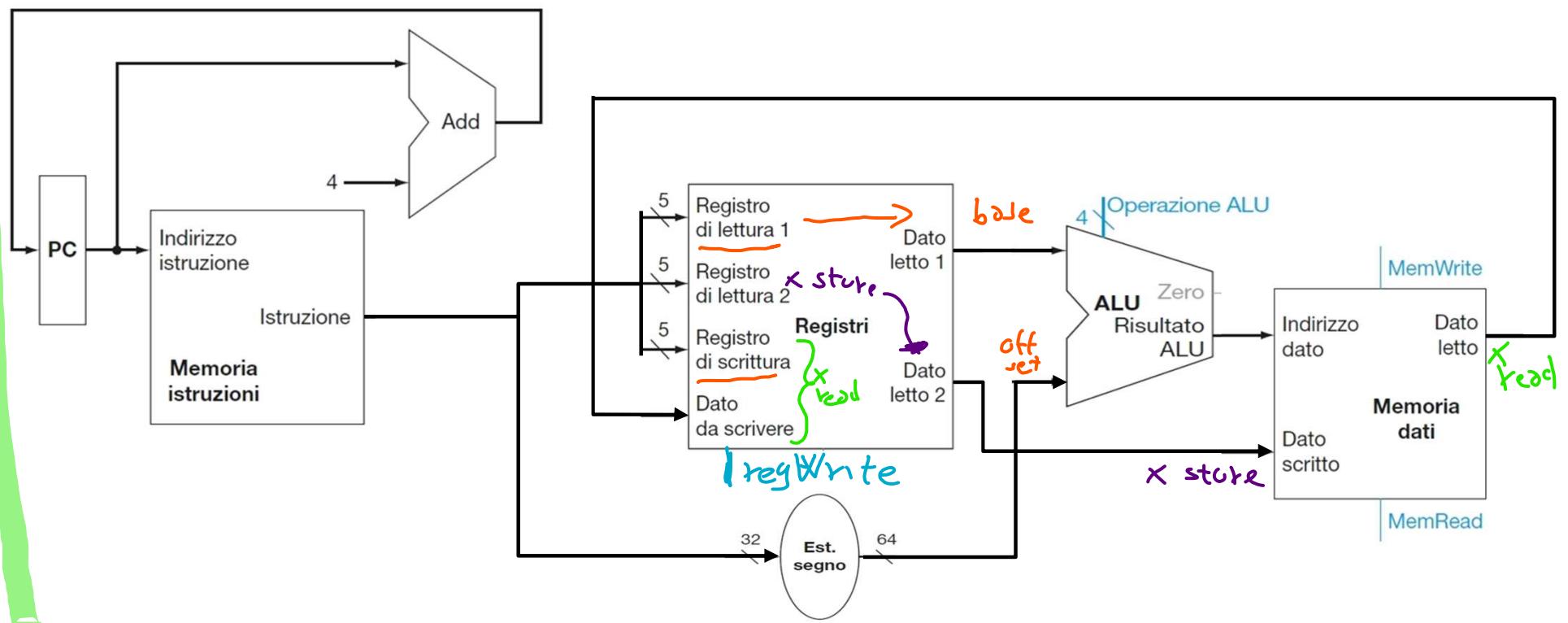
a. Unità di memoria dati



b. Unità di estensione del segno

# Istruzioni Load (Tipo I) e Store (Tipo S)

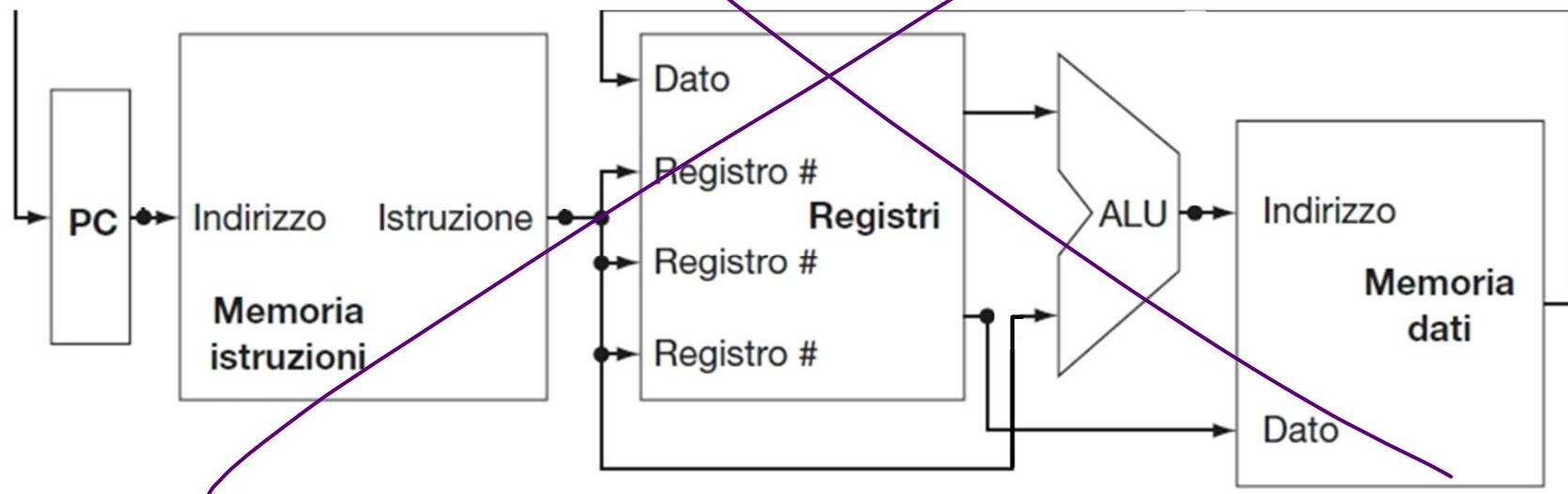
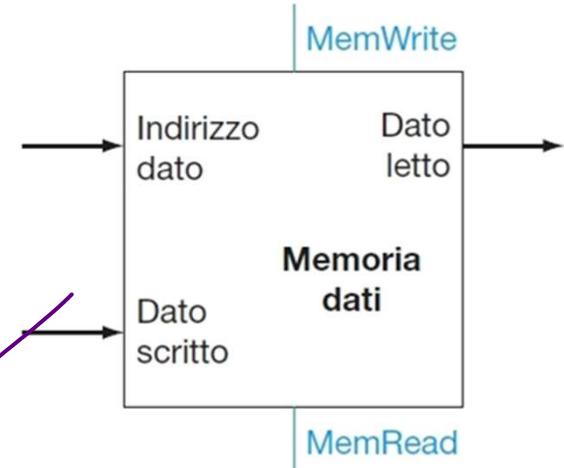
I	cost[11:0]	rs1	funz3	rd	codop
S	cost[11:5]	rs2	rs1	funz3	cost[4:0] codop



# Istruzioni Load (Tipo I) e Store (Tipo S)

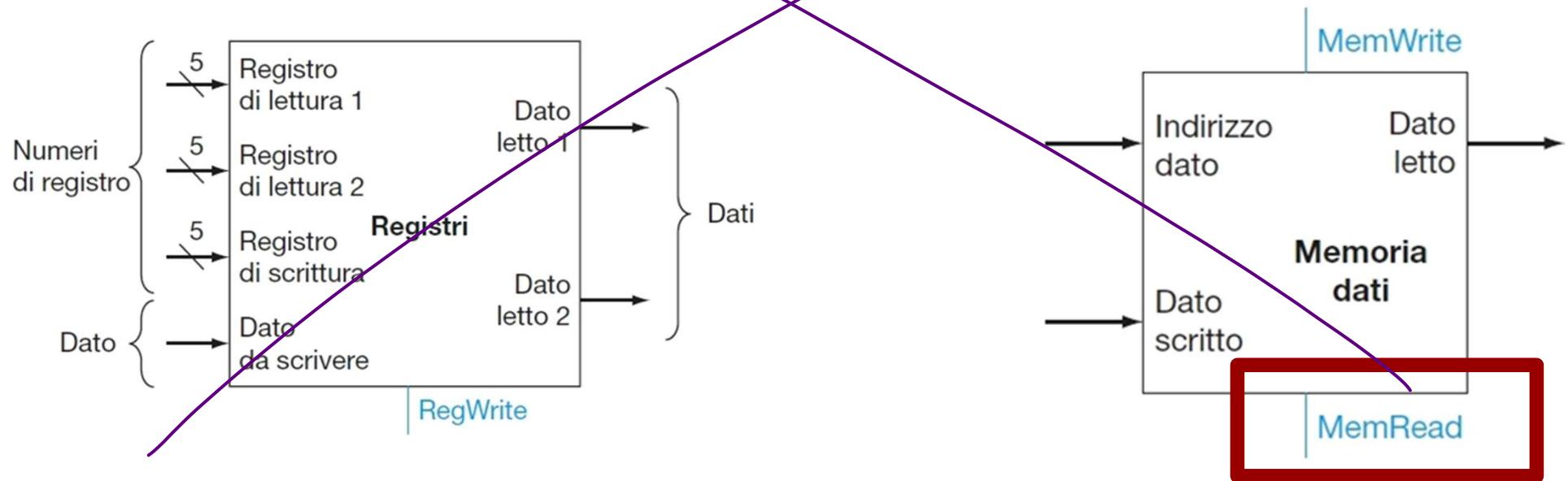
- L'unità di memoria è un elemento di stato
- Ingressi
  - l'indirizzo del dato (della ALU)
  - Il dato da scrivere (dai Registri)
- Uscita:
  - il dato letto (ai Registri)

**ld x1, offset(x2)**  
**sd x1, offset(x2)**



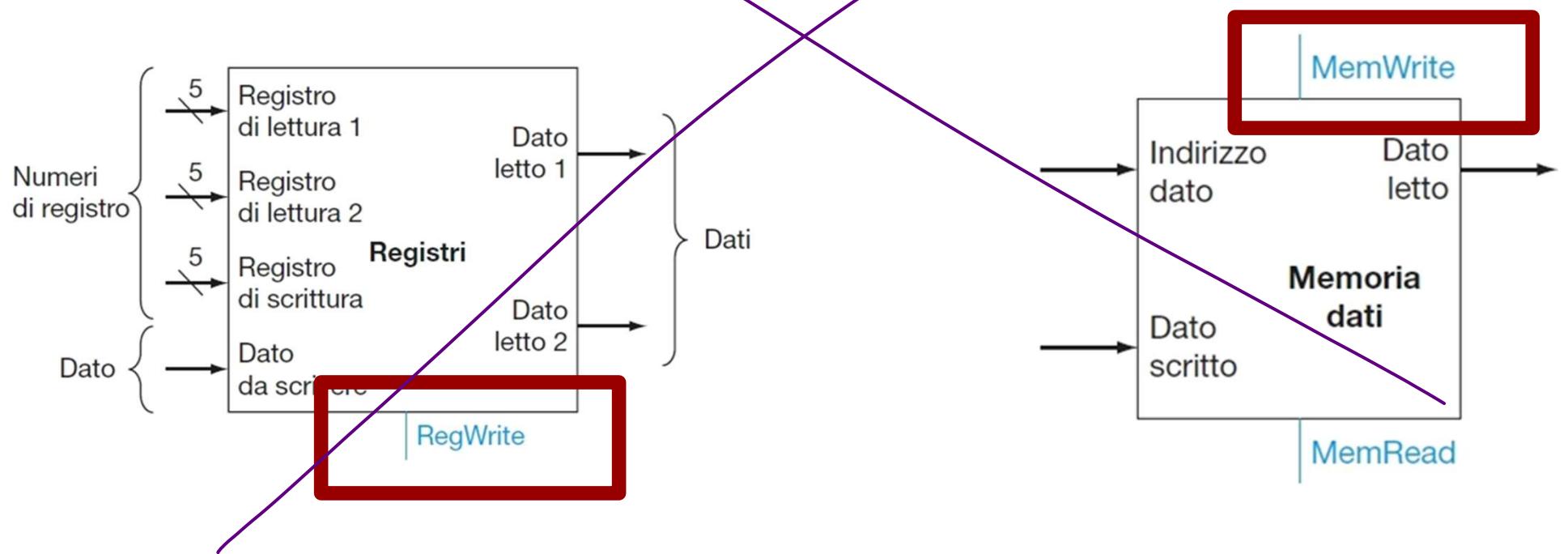
# Istruzioni Load (Tipo I) e Store (Tipo S)

- L'unità di memoria ha bisogno di un segnale di lettura esplicito, diversamente da quanto avviene per il register file.



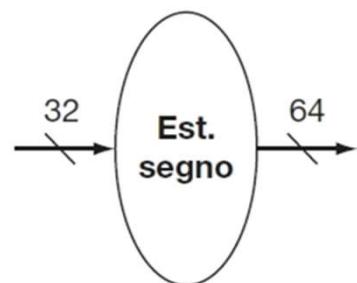
# Istruzioni Load (Tipo I) e Store (Tipo S)

- Come nel caso del funzionamento dei Registri, (si suppone che) la scrittura della memoria è attiva su un fronte del segnale di clock



# Istruzioni Load (Tipo I) e Store (Tipo S)

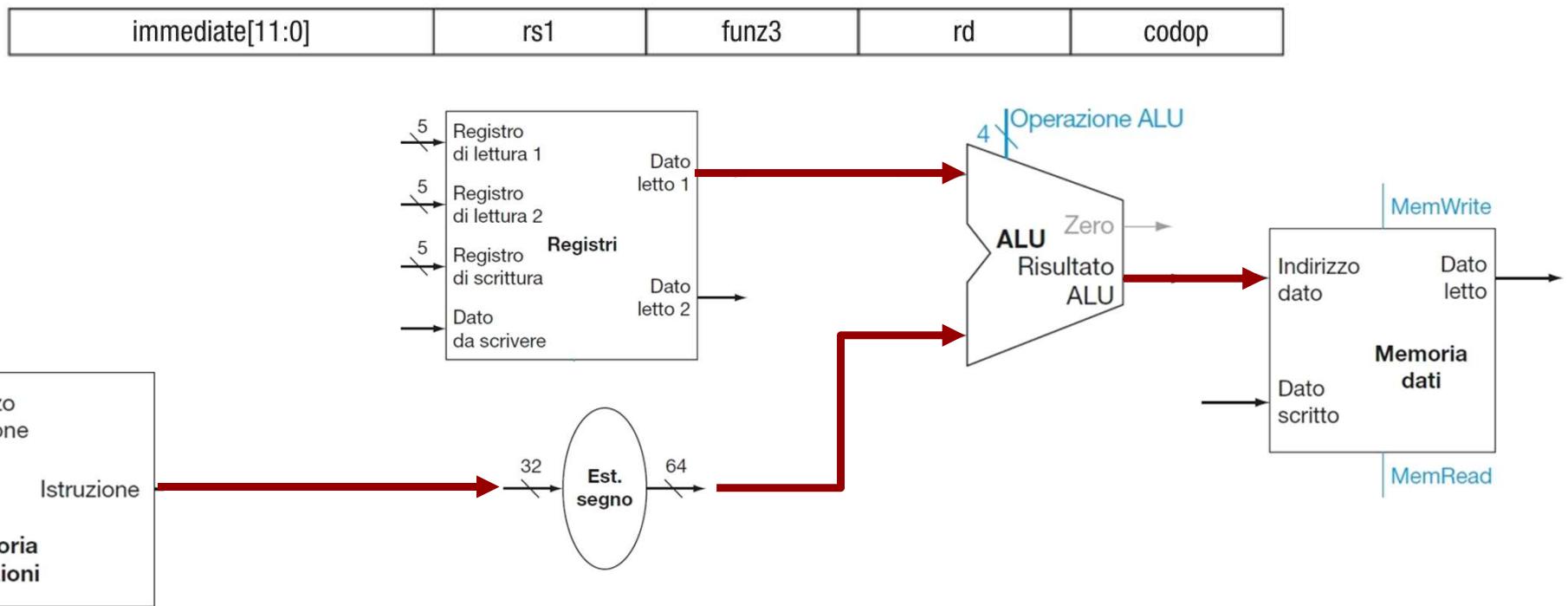
- Estensione del segno: aumento della dimensione di un dato ottenuta replicando il bit più significativo, che rappresenta il segno del dato originale, nei bit più significativi del dato di destinazione.



# Istruzioni Load (Tipo I) e Store (Tipo S)

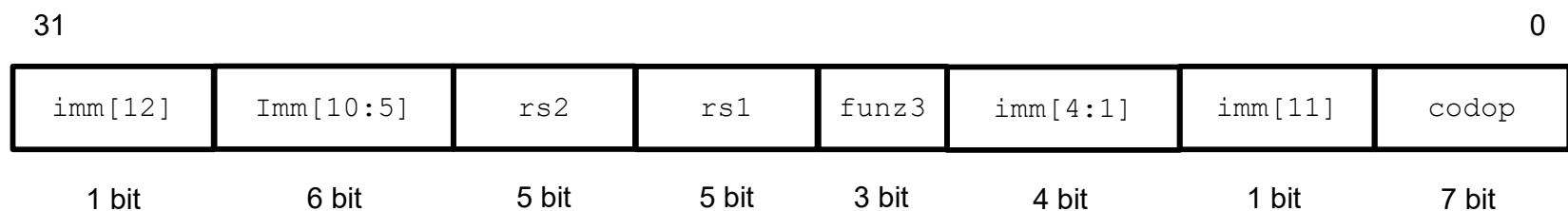
- L'unità di estensione del segno riceve in ingresso un numero a 12 bit per una load, una store e un salto condizionato all'uguaglianza, e lo fornisce in uscita esteso a 64 bit dotato di segno

(b) Tipo I



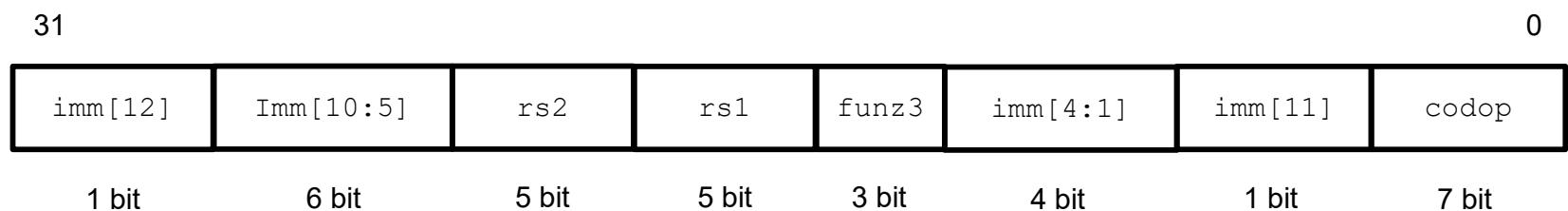
# L'Istruzione di Salto **beq** (Tipo SB)

- Istruzione che richiede il confronto fra due valori e che consente il trasferimento del controllo a un altro indirizzo del programma a seconda del risultato del confronto
- Le istruzioni di salto condizionato utilizzano il **formato di tipo SB**
- Indirizzamento relativo al Program Counter (PC-relative addressing).
- Può rappresentare indirizzi di salto da -4096 a 4094, in multipli di due



# L'Istruzione di Salto **beq** (Tipo SB)

- L'architettura stabilisce che il campo offset sia **spostato di 1 bit a sinistra**, in modo tale che l'offset non codifichi lo spiazzamento in numero di byte, ma in **numero di mezze parole**. Di fatto, tale spostamento aumenta lo spazio di indirizzamento dell'offset di un fattore 2 rispetto alla codifica dello spiazzamento in byte.
- Quindi, i 12 bit possono rappresentare indirizzi di salto da -4096 a 4094, in multipli di due



# L'Istruzione di Salto **beq** (Tipo SB)

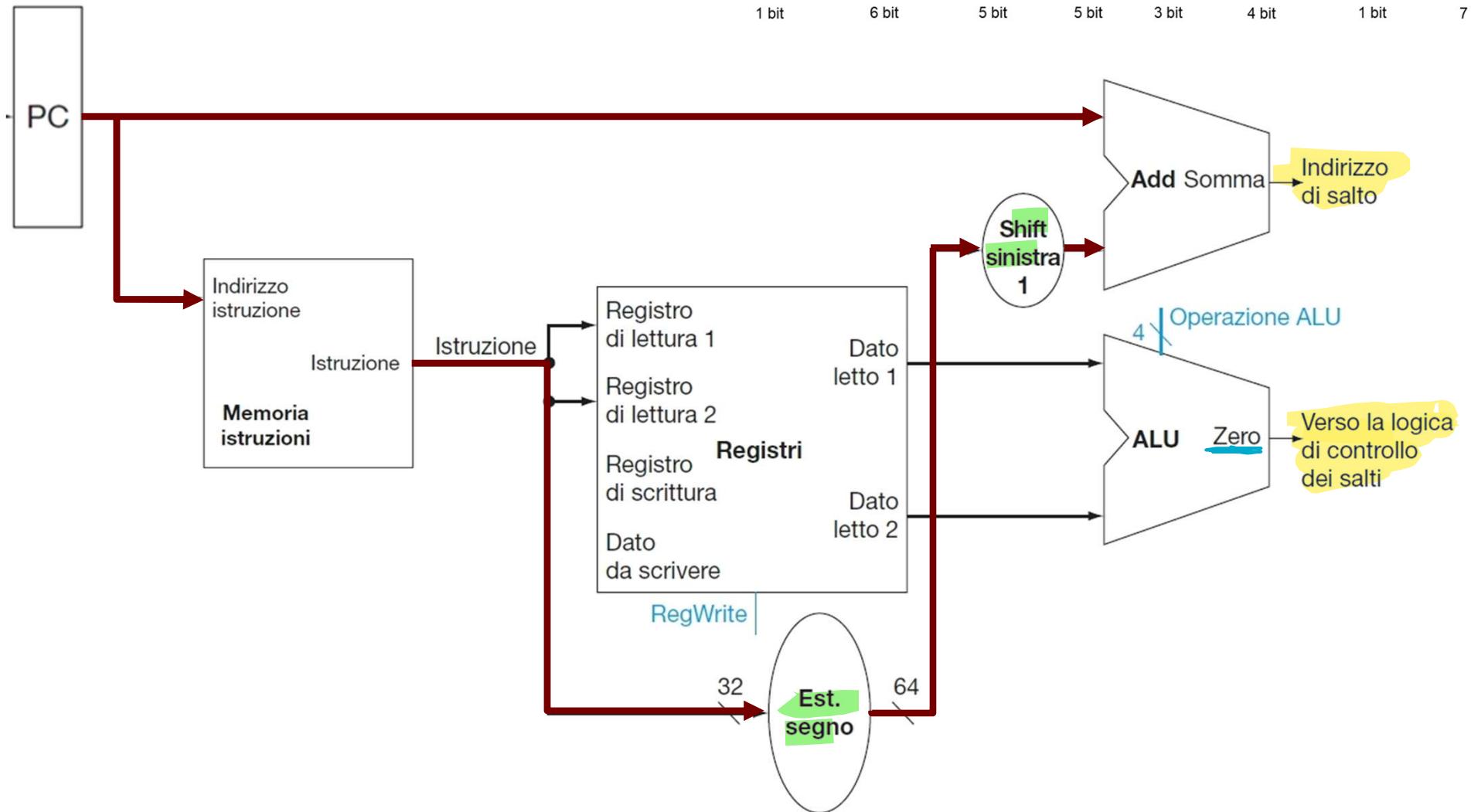
- **Indirizzo di destinazione del salto:** l'indirizzo che viene specificato in un salto condizionato
- Diventerà il nuovo indirizzo contenuto nel Program Counter (PC) se il salto condizionato viene eseguito
- L'indirizzo di destinazione di un salto è ottenuto mediante **la somma del campo offset contenuto nell'istruzione stessa (e.g., L1)** con **l'indirizzo dell'istruzione di salto condizionato (e.g., 0x0000000000400020)**

**0x0000000000400020      beq rs1, rs2, L1**

- il campo offset dell'istruzione va esteso a 64 bit con segno per la somma

31

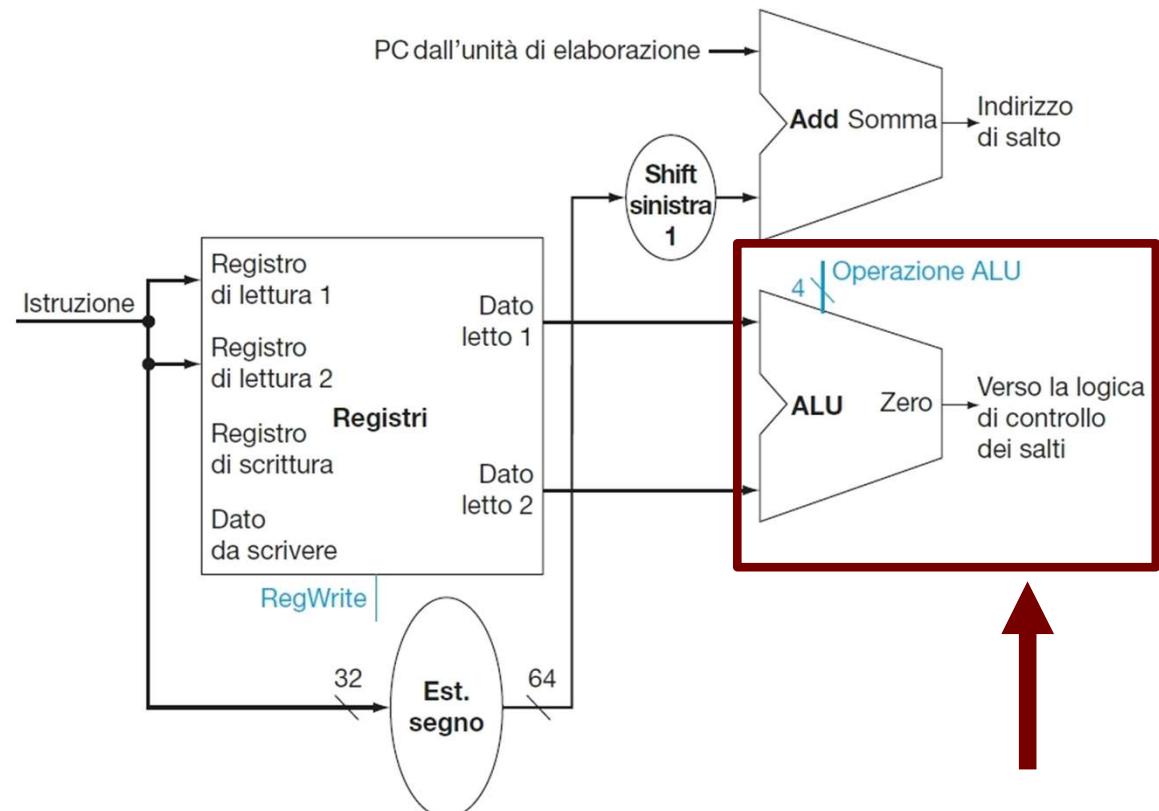
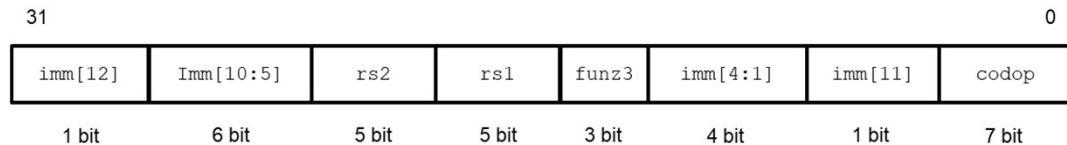
0



# L'Istruzione di Salto beq

Bisogna determinare se l'istruzione da eseguire dopo sarà:

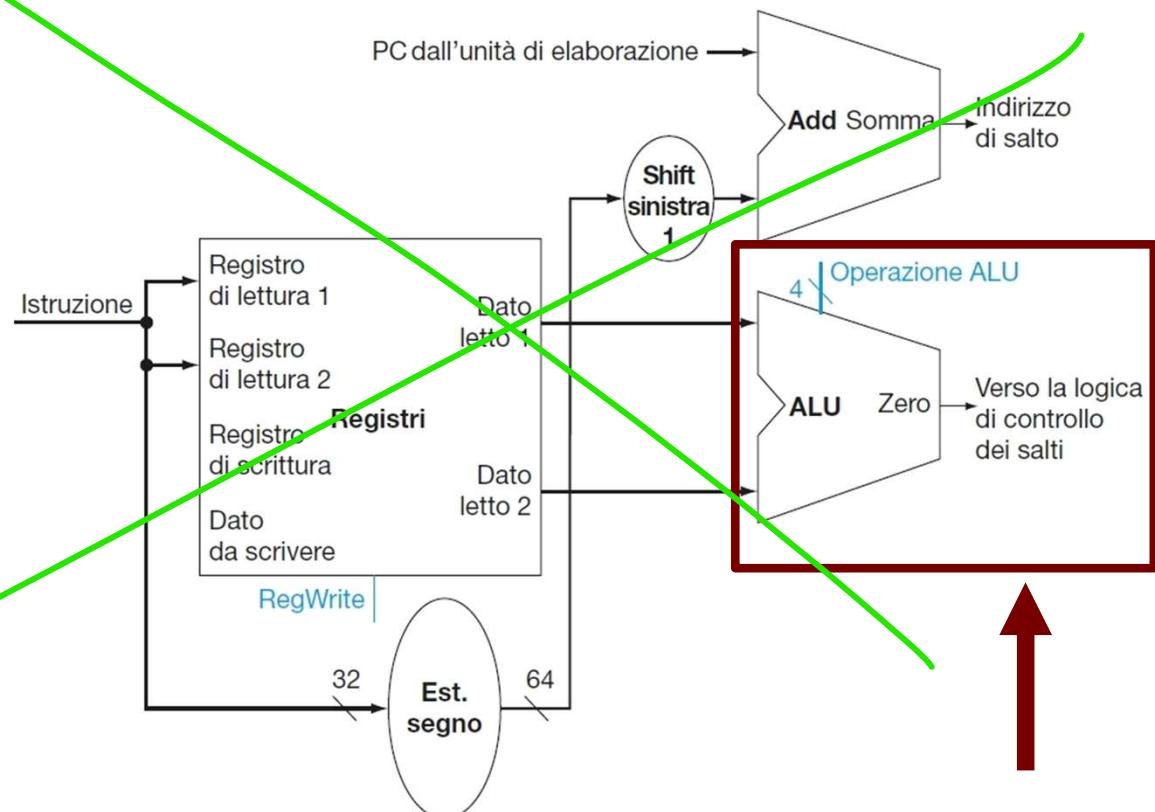
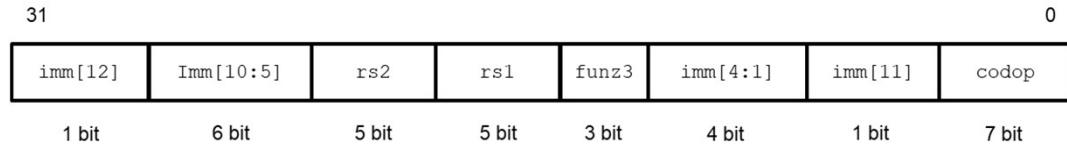
- branch not taken, salto non preso:** presente nella posizione successiva della memoria delle istruzioni
- branch taken, salto preso:** contenuta all'indirizzo di destinazione del salto
- Salto incondizionato:** assimilate a dei salti condizionati che vengono sempre eseguiti



hermole

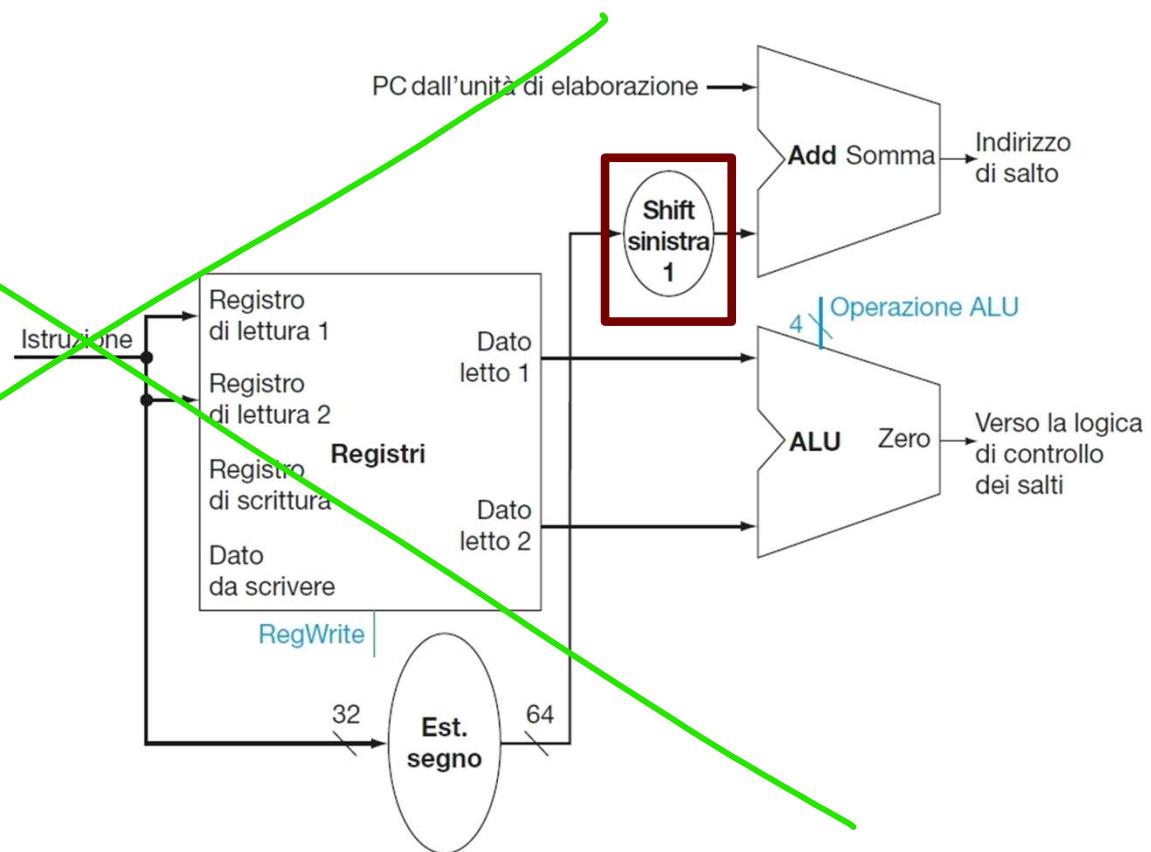
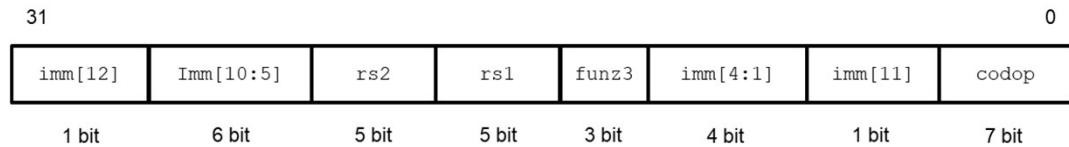
# L'Istruzione di Salto beq

- Si leggi il contenuto dei due registri contenenti gli operandi
- Il confronto tra i due operandi viene fatto dalla ALU
- La ALU fornisce un segnale in uscita che indica se il risultato è pari a 0
- Insieme ai operandi si invia alla ALU i segnali di controllo per effettuare una sottrazione
- Se l'uscita Zero della ALU viene asserita, i due operandi sono uguali



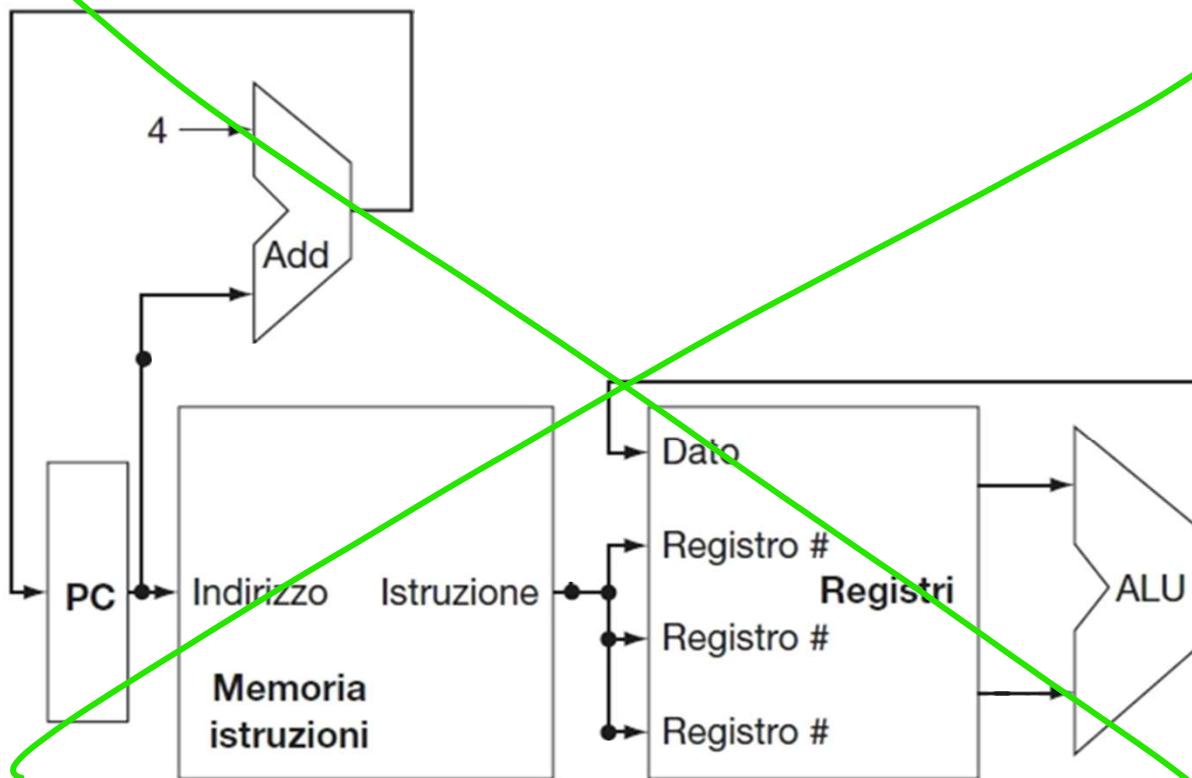
# L'Istruzione di Salto beq

- L'unità "Shift a sinistra 1" corrisponde a un instradamento dei segnali che rappresentano lo spiazzamento da 1 bit
- l'intradamento è effettuato in modo tale che venga aggiunto un segnale uguale a 0 come bit meno significativo del offset
- poiché l'entità è costante, non è richiesto alcun componente dedicato o circuito di controllo



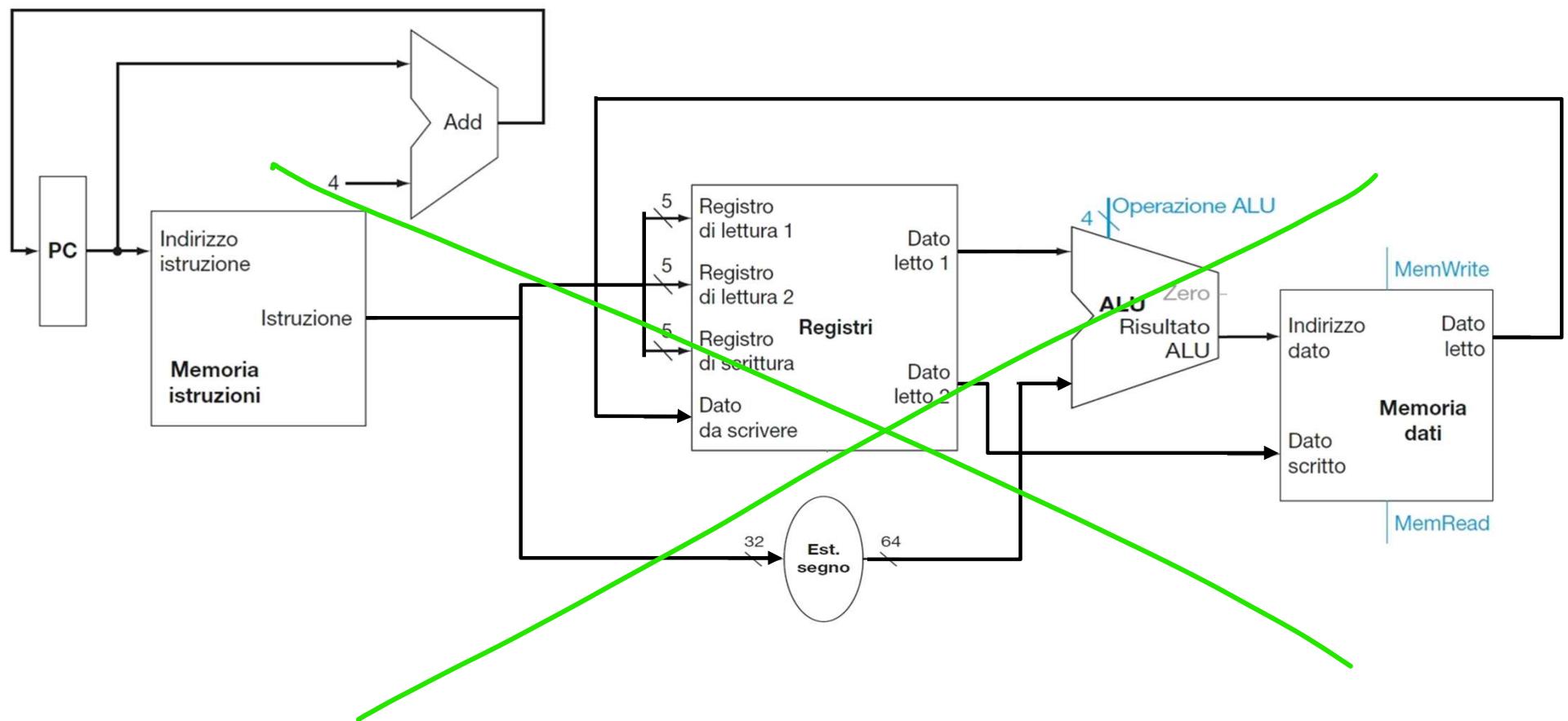
# Unità di Elaborazione Unificata

- Abbiamo prodotto unità diverse per ogni tipo di istruzione (Tipo R)



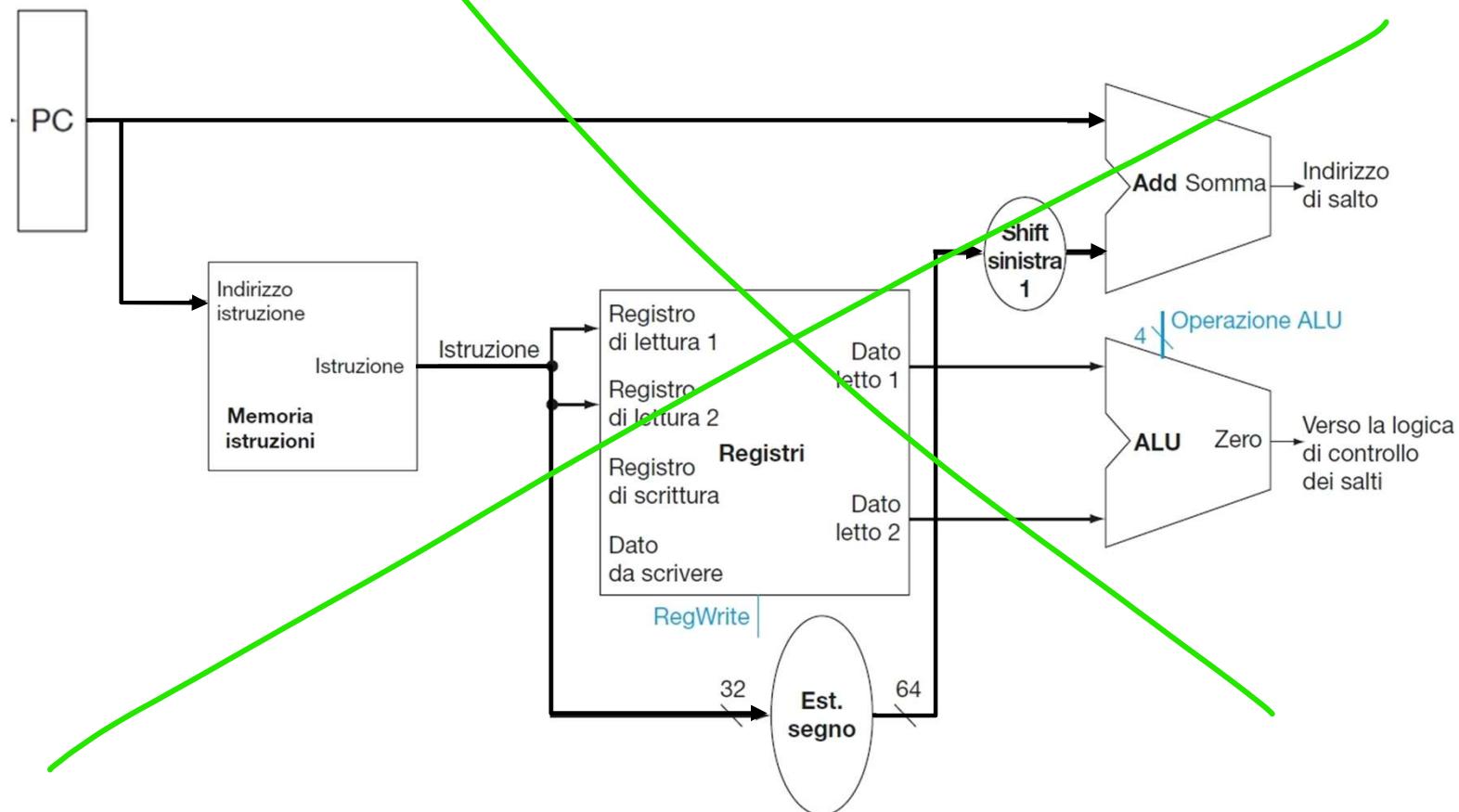
# Unità di Elaborazione Unificata

- Abbiamo prodotto unità diverse per ogni tipo di istruzione (Load & Store)



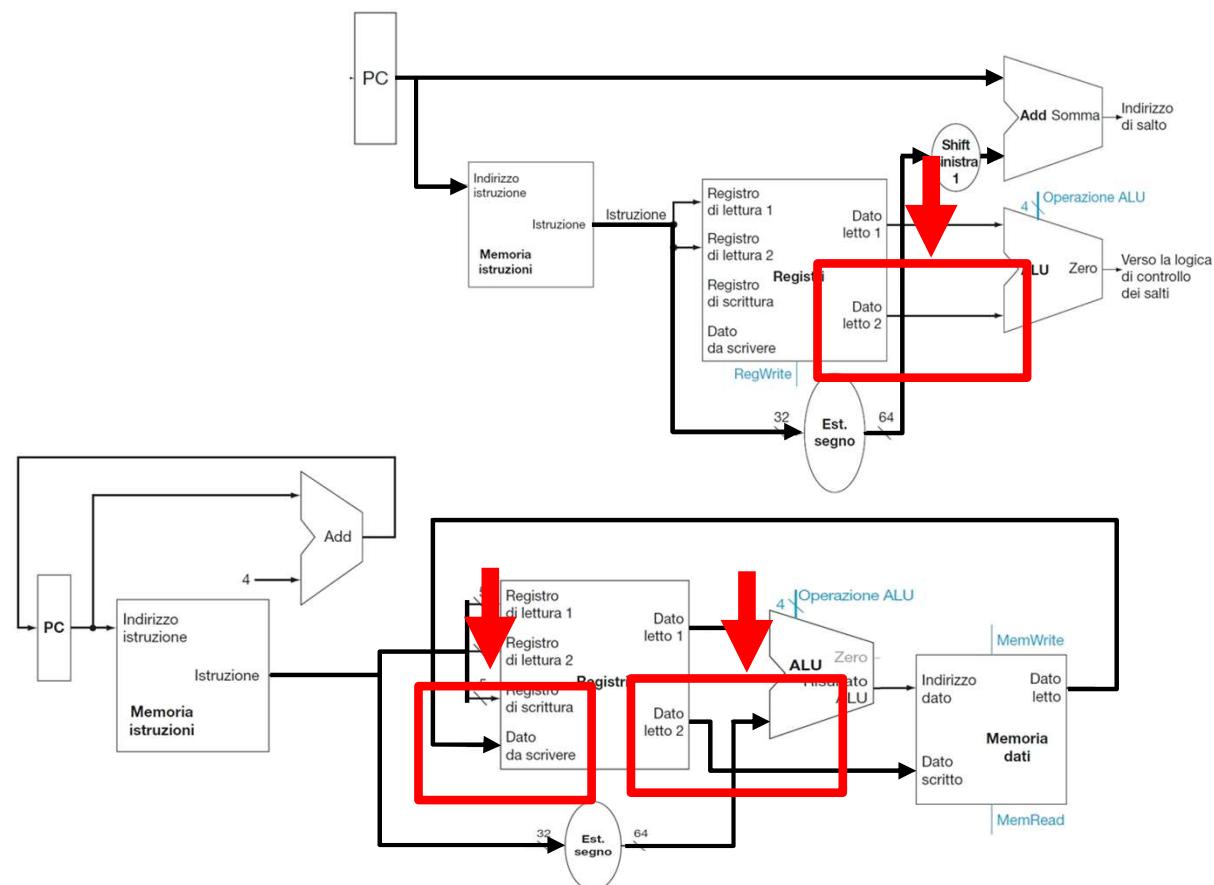
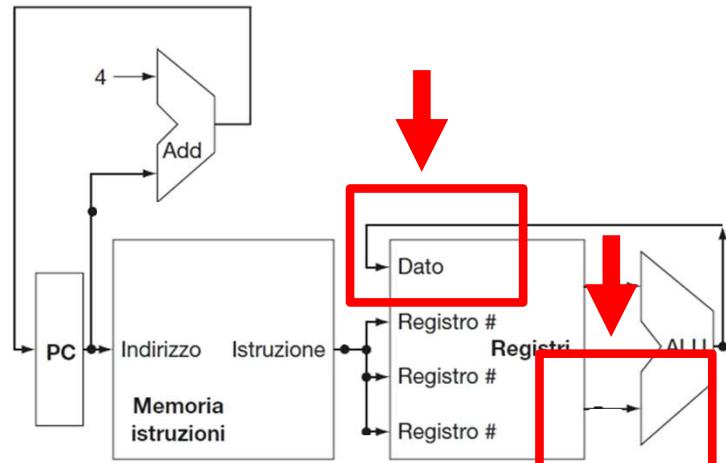
# Unità di Elaborazione Unificata

- Abbiamo prodotto unità diverse per ogni tipo di istruzione (beq – Tipo SB)

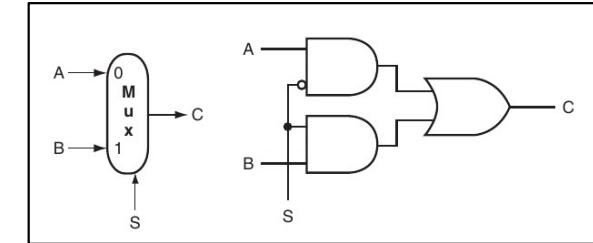
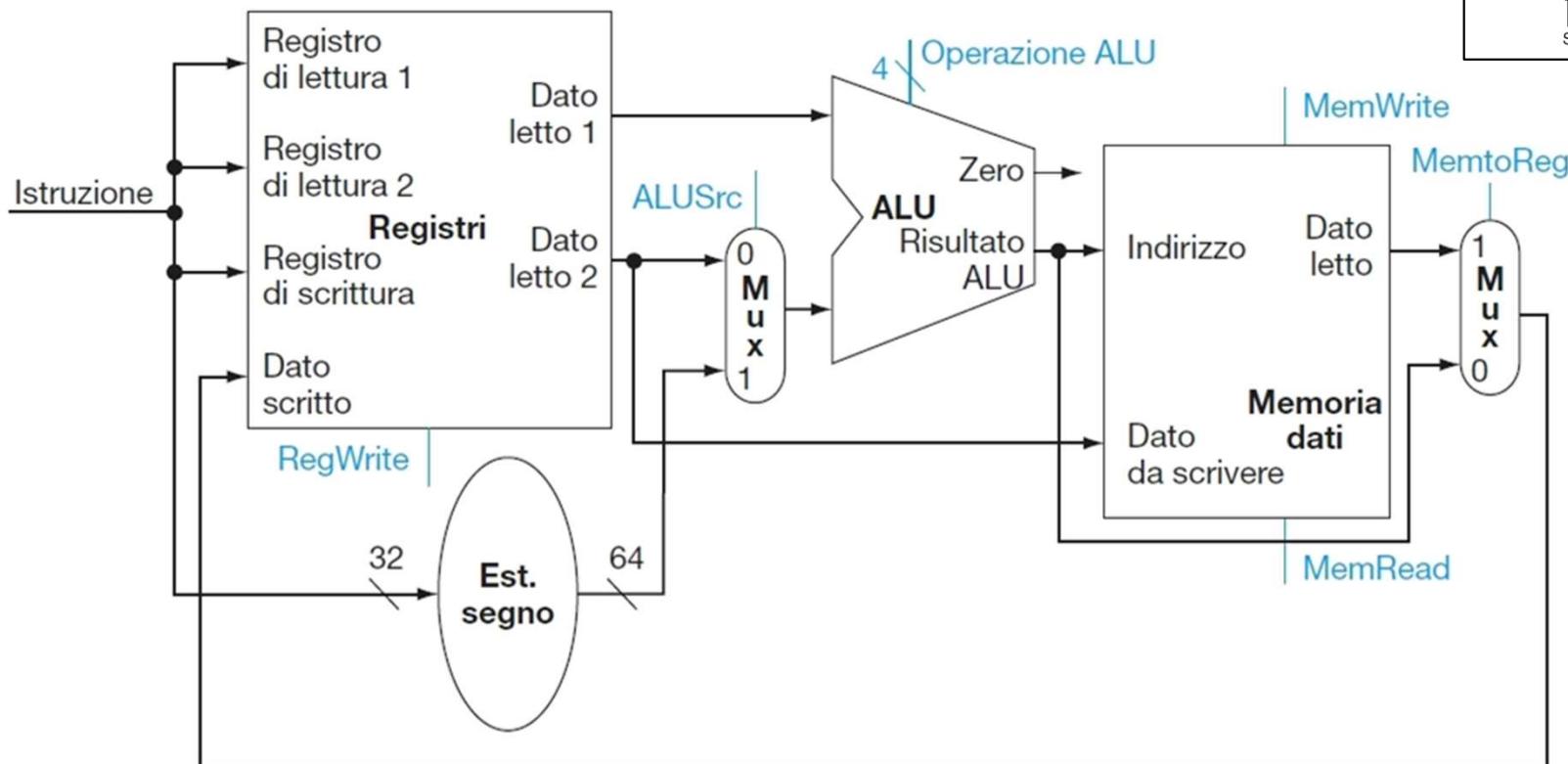


# Unità di Elaborazione Unificata

- Come unificare le tre in un singolo processore RISC-V?
- Problema: Selezionare quale segnale considerare per ogni operazione (ciclo di clock)
- Soluzione: Multiplexers



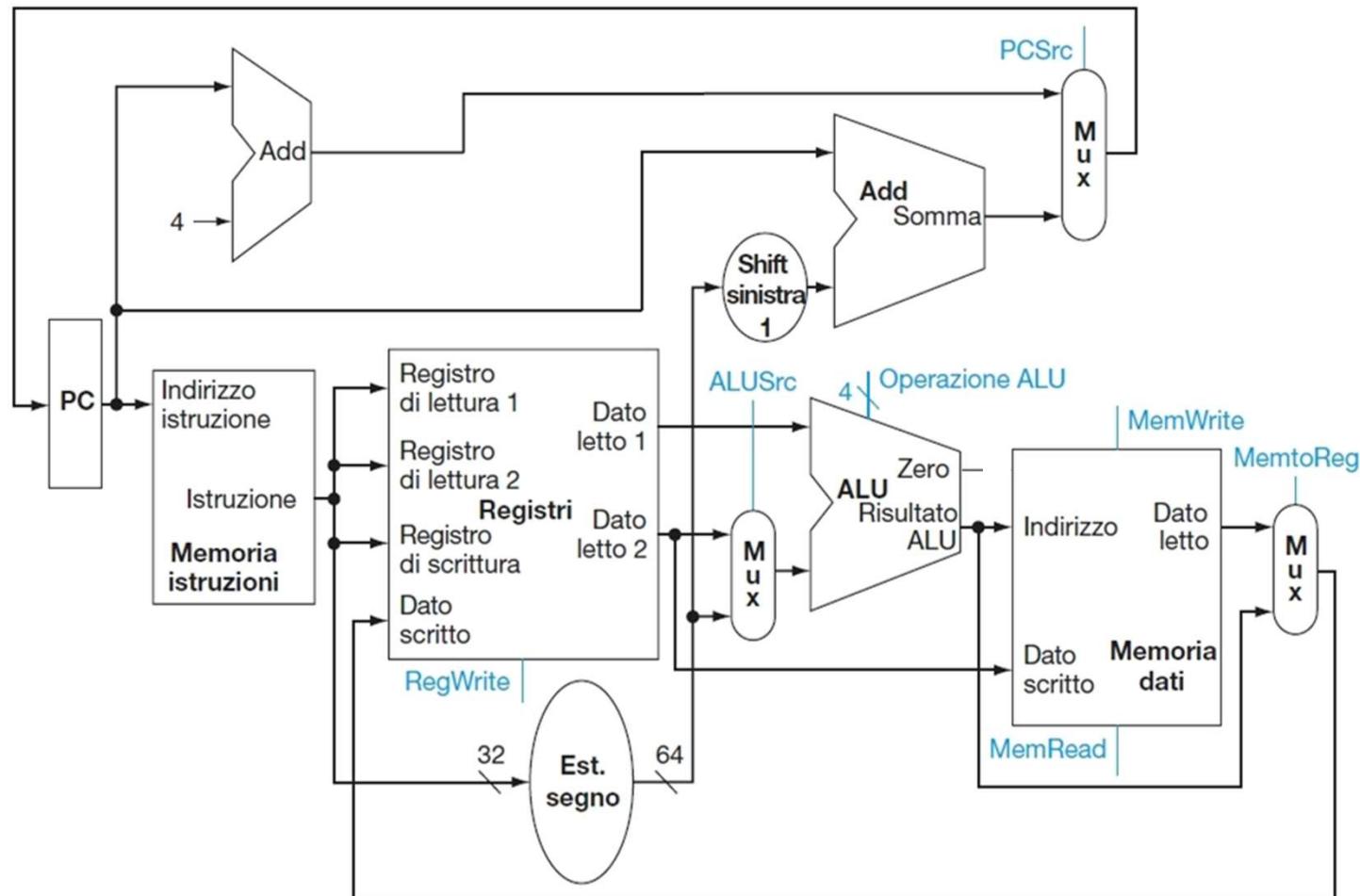
# Unità di Elaborazione Unificata



Unità di elaborazione per le istruzioni di accesso alla memoria e per le istruzioni di tipo R, ottenuta con l'aggiunta di due multiplexer.

# Unità di Elaborazione Unificata

- Unità di elaborazione per eseguire le istruzioni di base in un singolo ciclo di clock
- **Per integrare i salti condizionati si è resa necessaria l'aggiunta di un altro multiplexer**



# Unità di Elaborazione Unificata

- Questa unità di elaborazione **esegue tutte le istruzioni in un singolo ciclo di clock.**
- Nessuna risorsa dell'unità potrà essere utilizzata più di una volta per ogni istruzione
- **Il controllo è fatto da un circuito combinatorio**
- Abbiamo duplicato diversi componenti (e.g., una memoria delle istruzioni separata da quella dei dati)

