

# Architettura degli Elaboratori

## Architettura della CPU MIPS



SAPIENZA  
UNIVERSITÀ DI ROMA

Alessandro Checco  
[checco@di.uniroma1.it](mailto:checco@di.uniroma1.it)

Special thanks and credits:

Andrea Sterbini, Iacopo Masi,  
Claudio di Ciccio

[S&PdC]  
4.1-4.4



# Argomenti

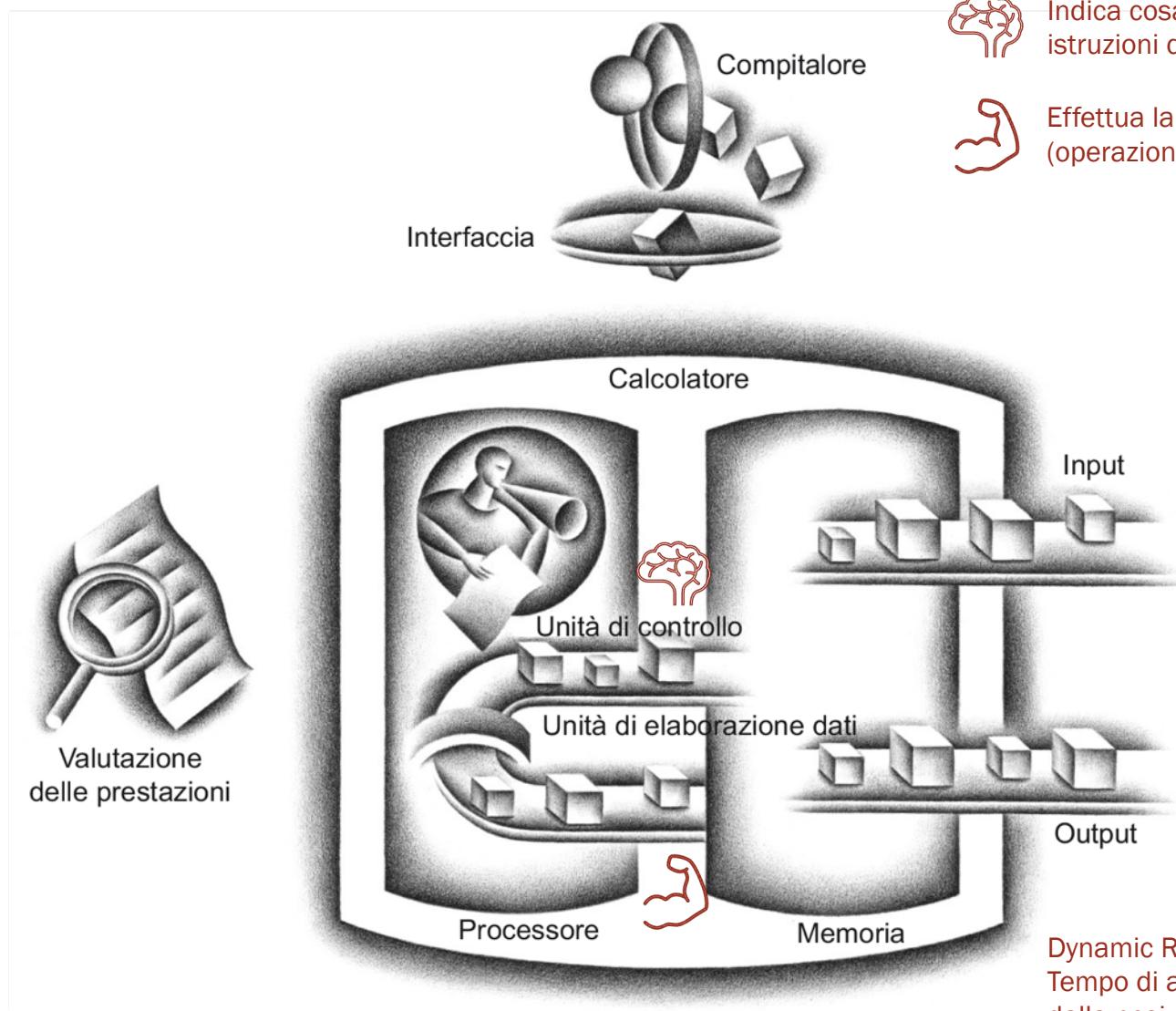
---

## Argomenti della lezione

- Progetto della CPU MIPS (con set di istruzioni semplificato) a singolo colpo di clock
  - Istruzioni da implementare
  - Unità funzionali necessarie
  - Datapath e unità di controllo
  - Tempo di esecuzione delle istruzioni



# L'architettura del calcolatore



Indica cosa fare in base alle istruzioni del programma



Effettua la computazione (operazioni aritmetico-logiche)

Input

Output

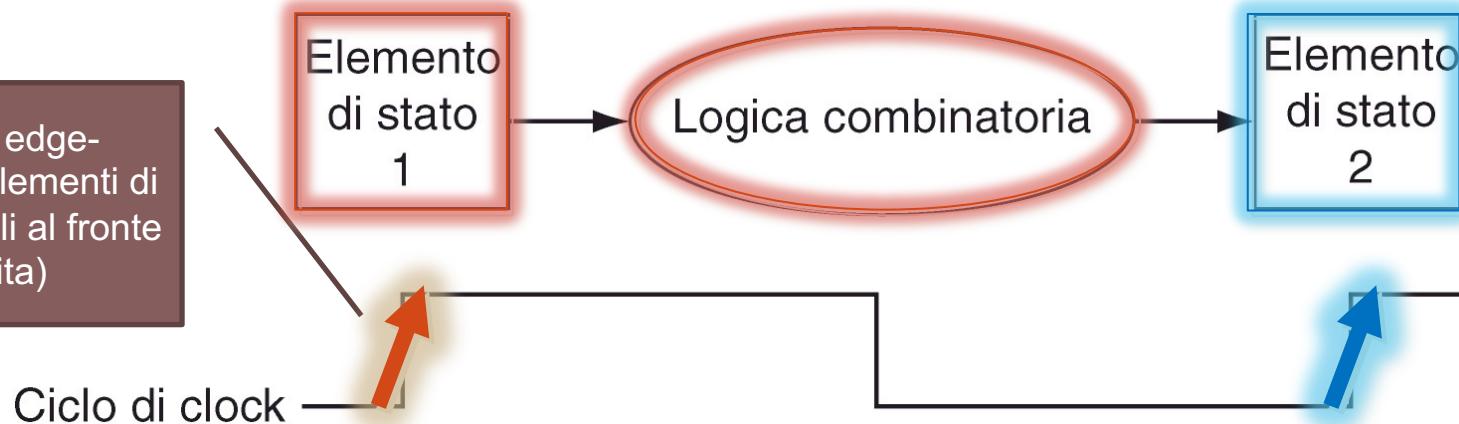
Memoria

Dynamic Random Access Memory  
Tempo di accesso indipendente dalla posizione

# La CPU

CPU = macchina sequenziale    ovvero    Stato + circuito combinatorio

Rappresentazione numerica basata su valore di tensione (alto o basso → 1 o 0, tipicamente)

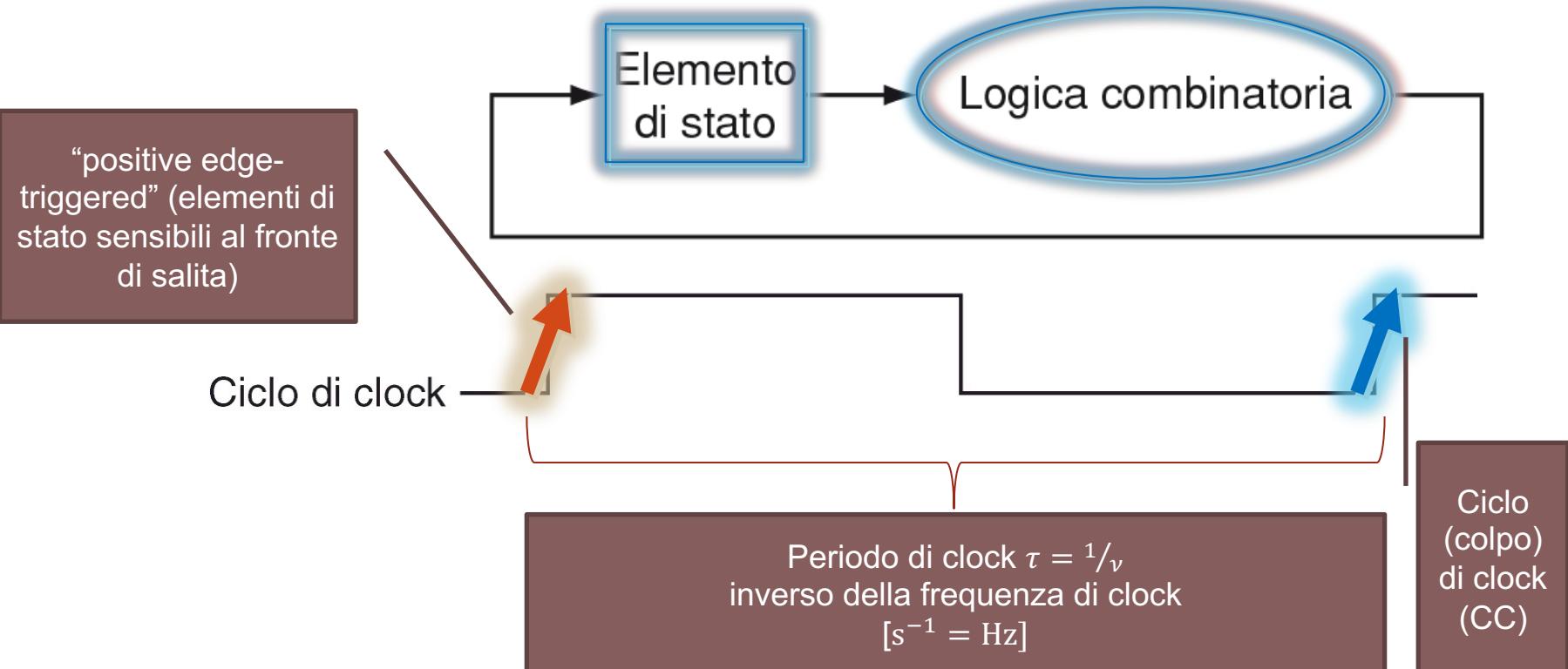


Tempo di esecuzione delle istruzioni ← durata dell’istruzione più lenta

# La CPU

CPU = macchina sequenziale ovvero Stato + circuito combinatorio

Rappresentazione numerica basata su valore di tensione (alto o basso → 1 o 0, tipicamente)



Periodo di clock ← durata dell'istruzione più lenta

# Progettare la CPU MIPS

---

## Prima fase: CPU MIPS semplice non ottimizzata (senza pipeline)

- Definire come viene elaborata una istruzione (**fasi di esecuzione**)
- Scegliere le **istruzioni da realizzare**
- Scegliere le **unità funzionali necessarie**
- **Collegare** le unità funzionali
- **Costruire** la CU (Control Unit) che controlla il funzionamento della CPU
- Calcolare il massimo tempo di esecuzione delle istruzioni (che ci dà il **periodo di clock**)

### Fasi di esecuzione di una istruzione:

<b>Fetch</b>	caricamento di una istruzione dalla memoria alla CU
<b>Decodifica</b>	decodifica della istruzione e <b>lettura argomenti</b> dai registri
<b>Esecuzione</b>	<b>esecuzione</b> (attivazione delle unità funzionali necessarie)
<b>Memoria</b>	accesso alla <b>memoria</b>
<b>Write Back</b>	scrittura dei <b>risultati nei registri</b>

### Altre operazioni necessarie

aggiornamento del PC      (normale / salti condizionati / salti non condizionati)

# Progettare la CPU (segue)

## Istruzioni da realizzare

accesso alla memoria:	lw, sw	(di tipo I)
salti condizionati:	beq	(di tipo I)
operazioni aritmetico-logiche:	add, sub, sll, slt, ...	(di tipo R)
salti non condizionati	j, jal	(di tipo J)
operazioni con costanti	li, addi, subi, ...	(di tipo I)

Formato delle istruzioni MIPS (ovvero la loro codifica in memoria)

Nome	Campi						Commenti
Dimensione del campo	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Tutte le istruzioni MIPS sono a 32 bit
Formato R	op	rs	rt	rd	shamt	funct	Formato delle istruzioni aritmetiche
Formato I	op	rs	rt	indirizzo / costante			Formato delle istruzioni di trasferimento dati di salto condizionato e immediate
Formato J	op	indirizzo di destinazione					Formato delle istruzioni di salto incondizionato

# Unità funzionali necessarie

---

PC	registro che contiene l'indirizzo della istruzione
memoria istruzioni	contiene le istruzioni
adder	per calcolare il PC (successivo o salto)
registri	contengono gli argomenti delle istruzioni
ALU	fa le operazioni aritmetico-logiche, confronti, indirizzi in mem
memoria dati	da cui leggere/in cui scrivere i dati (load/store)

Unità collegate da diversi **datapath** (interconnessioni che definiscono il flusso delle informazioni nella CPU)

Se un'unità funzionale può ricevere dati da **più sorgenti** è necessario inserire un multiplexer (**MUX**) per selezionare la sorgente necessaria

Le unità funzionali sono attivate e coordinate dai segnali prodotti dalla **Control Unit**

# Ingredienti: memoria delle istruzioni, PC, adder

## Memoria istruzioni:

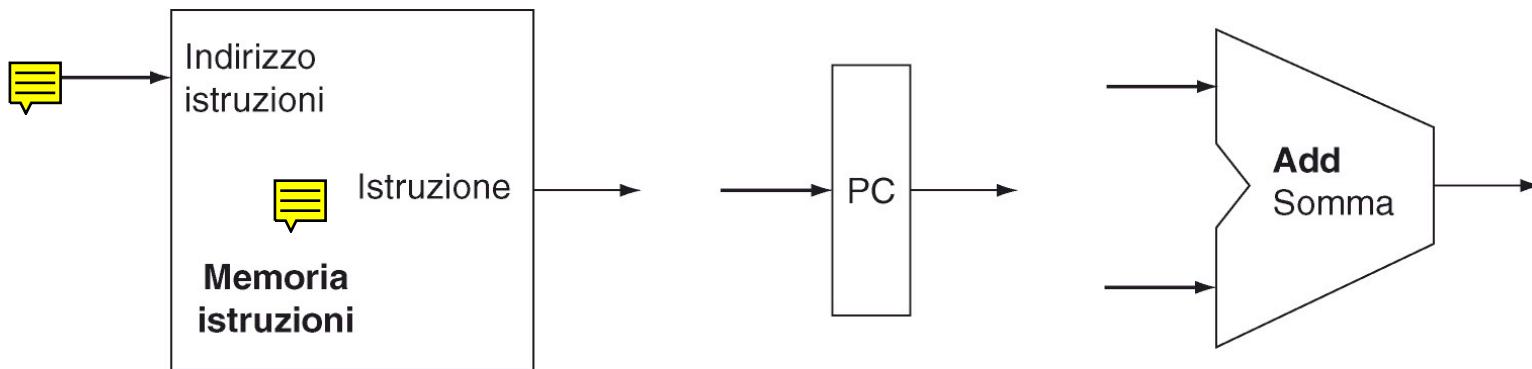
- Input: indirizzo a 32 bit
- Output: istruzione (da 32 bit) situata nell'indirizzo di input

## Program Counter:

- Registro che contiene l'**indirizzo** della istruzione corrente

## Sommatore:

- Necessario per calcolare il nuovo PC e le destinazioni dei salti relativi
- Riceve due valori a 32 bit e ne fornisce in uscita la somma



a. Memoria istruzioni

b. Program counter (PC)

c. Sommatore

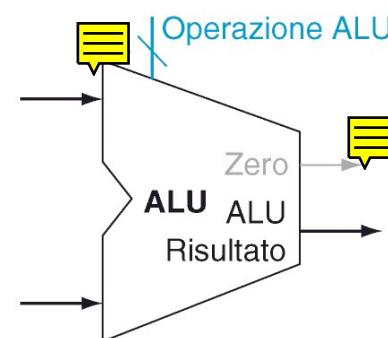
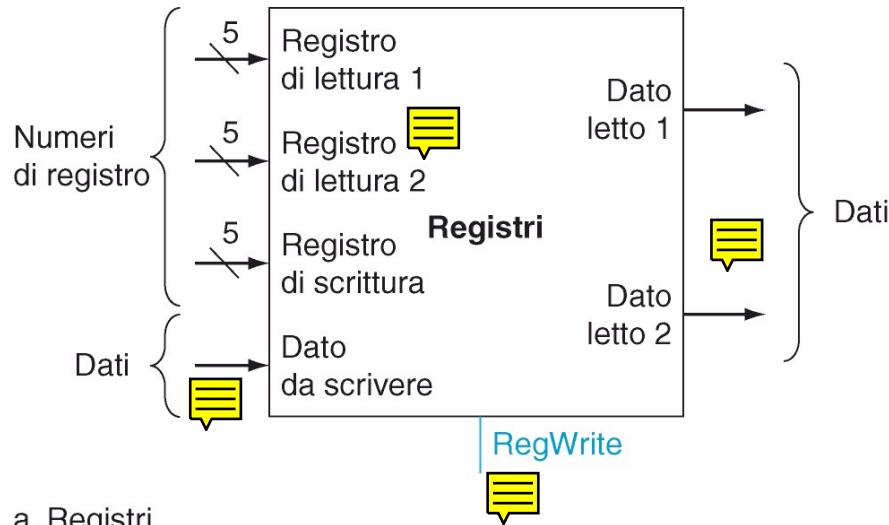
# Ingredienti: registri e ALU

## Blocco dei registri (register file):

- contiene **32 registri** a 32 bit, indirizzabili con 5 bit ( $2^5 = 32$ )
- può memorizzare un dato in un registro e contemporaneamente fornirlo in uscita
- 3 porte a 5 bit per indicare quali 2 registri leggere e quale registro scrivere
- 3 porte dati (a 32 bit)
  - una in ingresso per il valore da memorizzare e
  - 2 di uscita per i valori letti
- il segnale **RegWrite** abilita (se 1) la scrittura nel registro di scrittura

**ALU:** riceve due valori interi a 32 bit e svolge una operazione indicata dai segnali **Op. ALU**

- Oltre al risultato da 32 bit produce un segnale «Zero» asserito se il risultato è zero



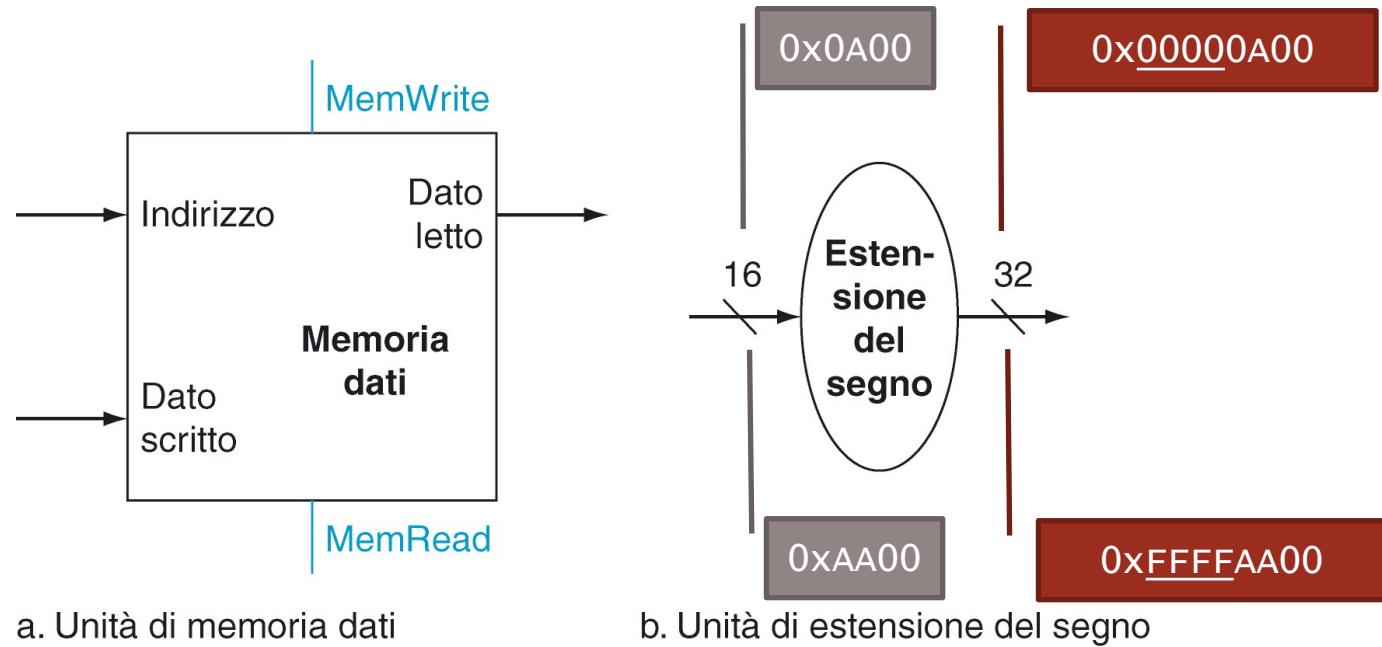
# Ingredienti: memoria dati ed unità di estensione del segno

## Unità di memoria

- riceve un **indirizzo** (da 32 bit) che indica quale word della memoria va letta/scritta
- riceve il segnale **MemRead** che abilita la lettura dall'indirizzo (se **Iw**)
- riceve un dato da 32 bit da scrivere in memoria a quell'indirizzo (se **sw**)
- riceve il segnale di controllo **MemWrite** che abilita (1) la scrittura del dato all'indirizzo
- fornisce su una porta di uscita da 32 bit il lato letto (se **MemRead = 1**)

L'unità di estensione del segno serve a trasformare un intero relativo (in CA2) da 16 a 32 bit

- Ovvero copia il bit del segno nei 16 bit più significativi della parola

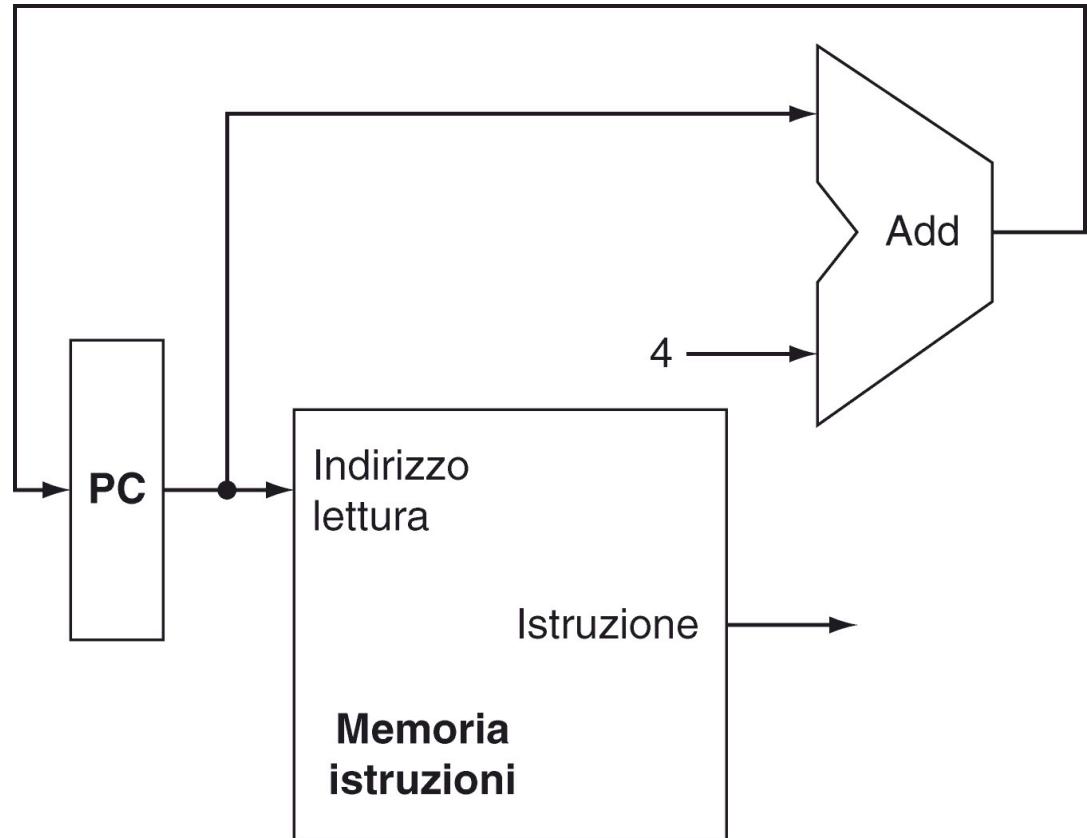


# Fetch della istruzione/aggiornamento PC

- PC = indirizzo dell'istruzione
- Lettura dell'istruzione
- PC incrementato di 4 (1 word)
- Valore aggiornato reimmesso nel PC

## NOTE

- Connessioni da 32 bit
- Mentre viene letta l'istruzione viene già calcolato il nuovo PC



# Dettaglio sulle istruzioni

Indice del registro destinazione in campi diversi dell'istruzione

Campo	0	rs	rt	rd	shamt	funct
Posizione dei bit	31-26	25-21	20-16	15-11	10-6	5-0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

b. Istruzioni di load e store

Campo	4	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

c. Istruzioni di salto condizionato

Necessario un MUX per selezionare il campo corretto

Tutte le istruzioni R hanno lo stesso codice (0) e sono distinte dal campo funct

# Operazioni ALU e accesso a MEM

opc|rs|rt|rd|shamt|funct

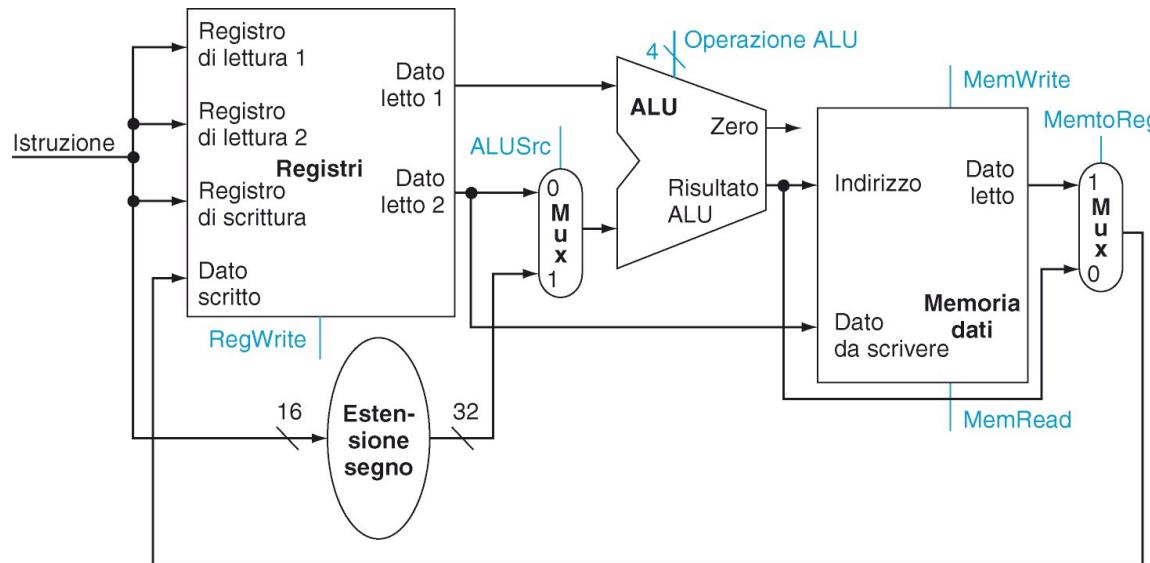
Decodifica facilitata: i formati I ed R sono quasi uguali

Secondo argomento dell'istruzione:

1. registro o
2. campo immediato (in questo caso: valore esteso nel segno)
  - a seconda del segnale di controllo **ALUSrc** che seleziona la porta corrispondente del MUX

Per calcolare l'indirizzo di accesso alla memoria si usa la stessa ALU (reg. base + campo i.)

Il risultato dell'ALU o della lw viene selezionato da **MemtoReg** che comanda il MUX a destra



# Operazioni ALU e accesso a MEM

opc|rs|rt|addr/const

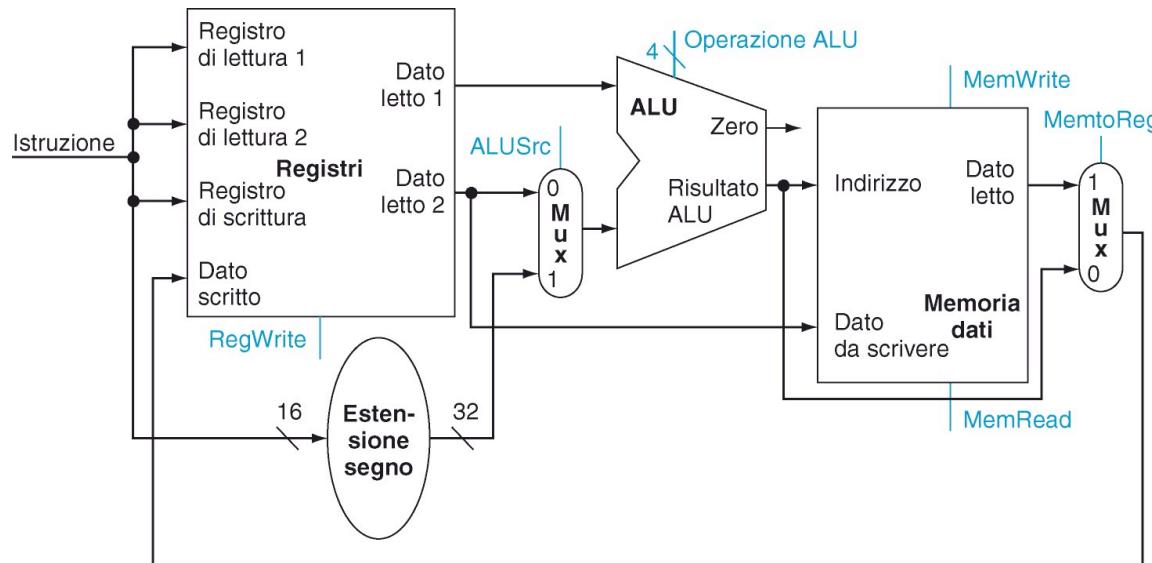
Decodifica facilitata: i formati I ed R sono quasi uguali

Secondo argomento dell'istruzione:

1. registro o
2. campo immediato (in questo caso: valore esteso nel segno)
- a seconda del segnale di controllo **ALUSrc** che seleziona la porta corrispondente del MUX

Per calcolare l'indirizzo di accesso alla memoria si usa la stessa ALU (reg. base + campo i.)

Il risultato dell'ALU o della lw viene selezionato da **MemtoReg** che comanda il MUX a destra



# Operazioni ALU e accesso a MEM

opc|rs|rt|addr/const

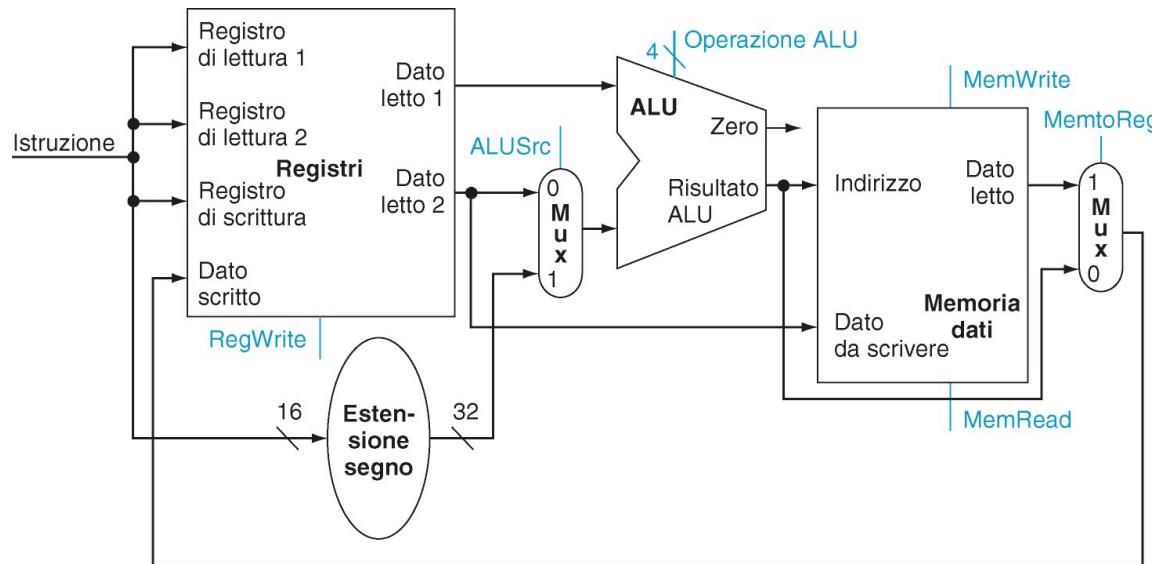
Decodifica facilitata: i formati I ed R sono quasi uguali

Secondo argomento dell'istruzione:

1. registro o
2. campo immediato (in questo caso: valore esteso nel segno)
  - a seconda del segnale di controllo **ALUSrc** che seleziona la porta corrispondente del MUX

Per calcolare l'indirizzo di accesso alla memoria si usa la stessa ALU (reg. base + campo i.)

Il risultato dell'ALU o della lw viene selezionato da **MemtoReg** che comanda il MUX a destra



# Salti condizionati (beq)

ALU come comparatore

(sottrazione)

- Il segnale **Zero** indica se operare il salto

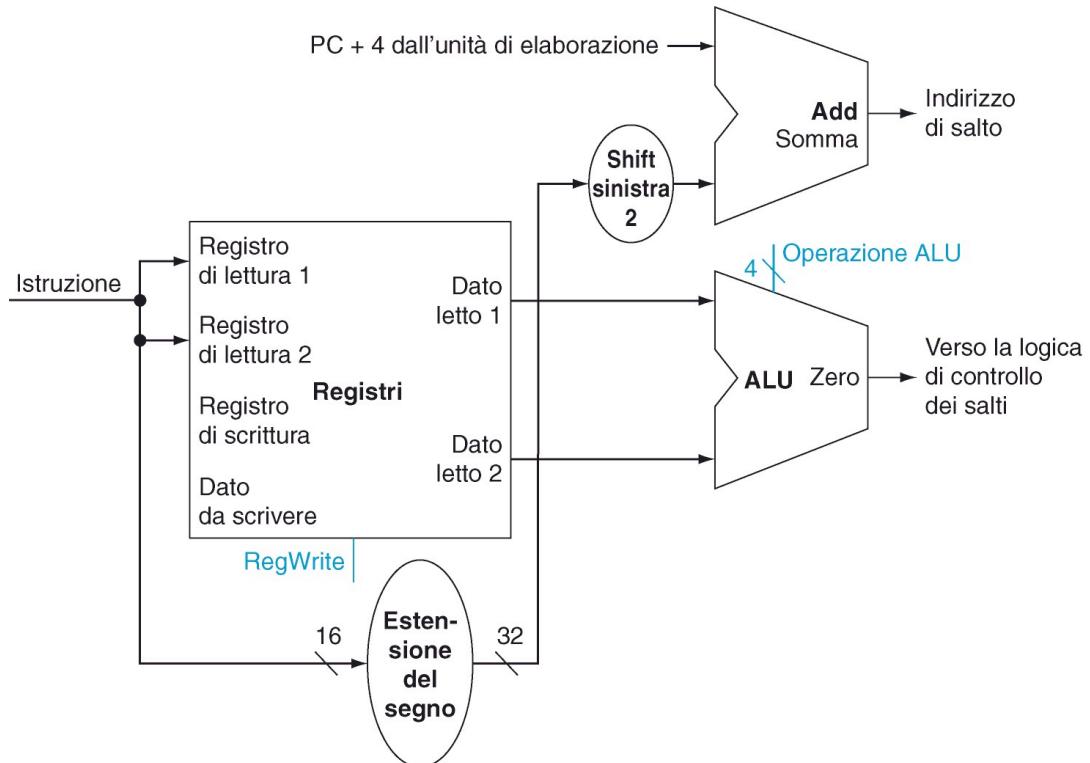
La destinazione dei salti è un **numero relativo di istruzioni** rispetto alla istruzione seguente

- estesa nel segno
- moltiplicata per 4
- sommata a PC + 4

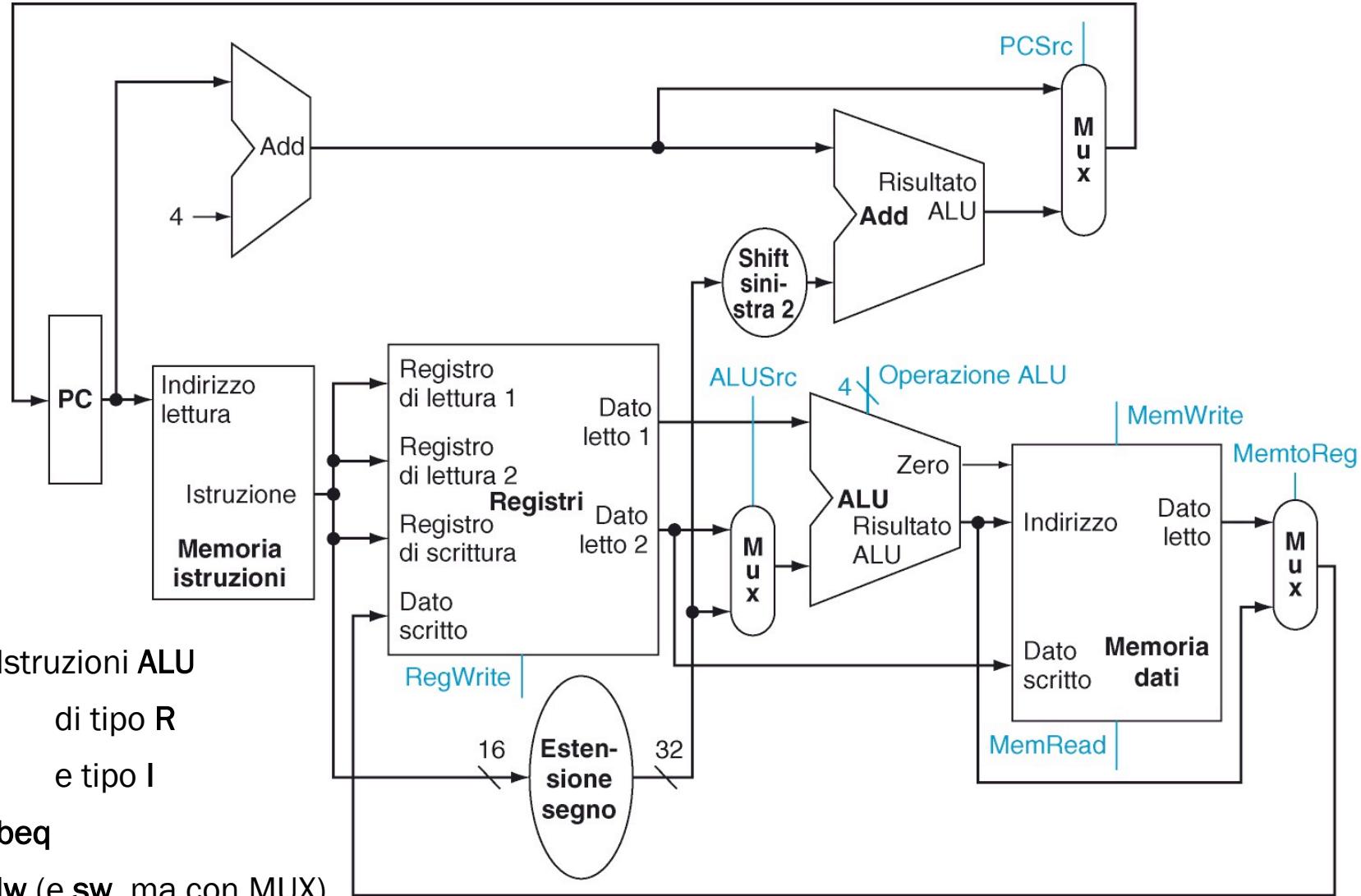
Il nuovo valore del PC può provenire da:

- PC+4 (istruzione seguente) o
- uscita dell'adder (salto)

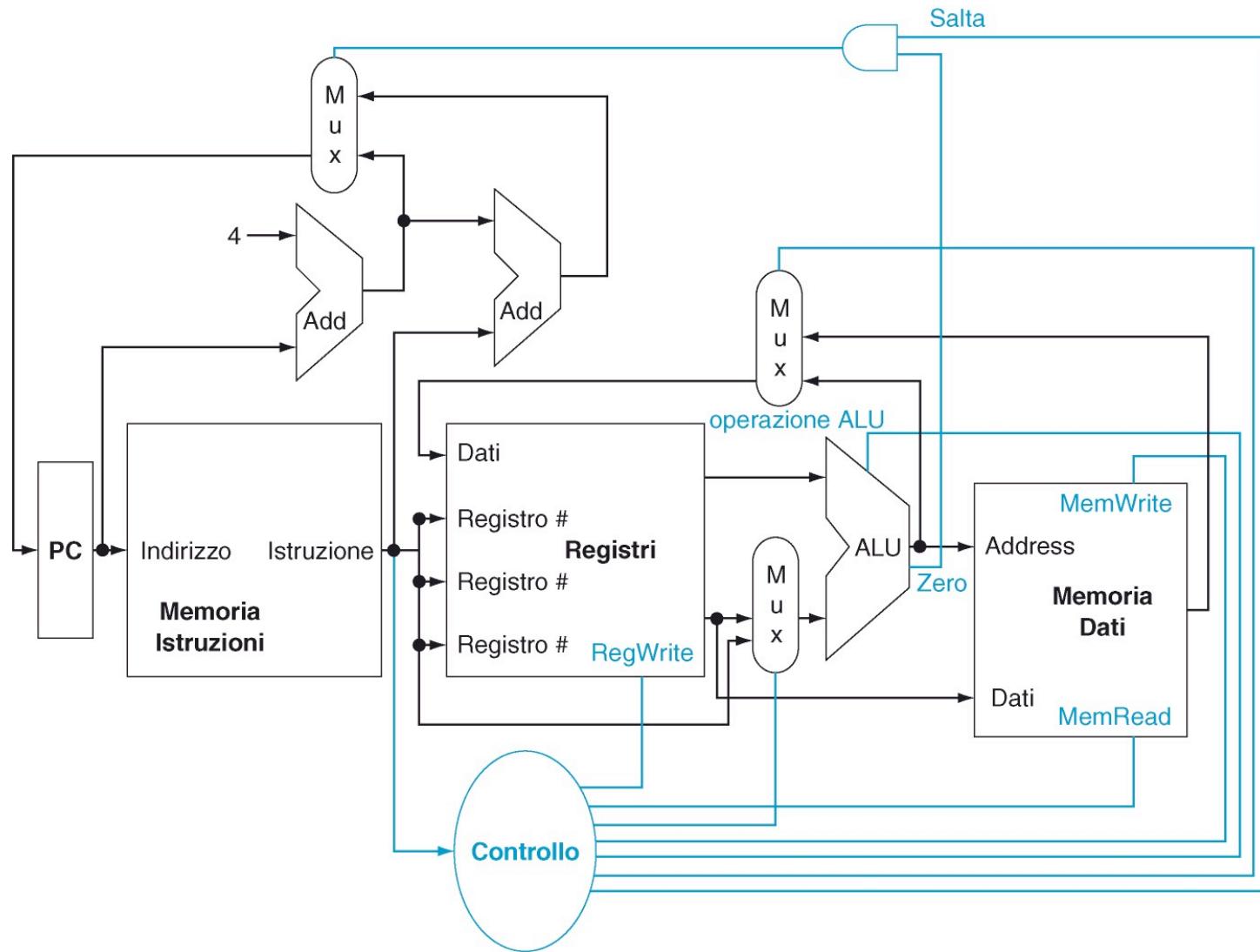
Necessario un **MUX** di selezione



# Tutto insieme



## ... con Control Unit e logica di salto



# Unità di controllo della ALU

---

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

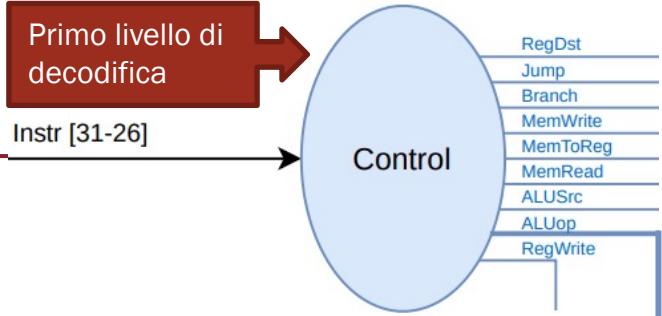
# Formato istruzioni e bit di controllo ALU

La logica di controllo per la ALU è implementata a «cascata».

- Livelli multipli di decodifica.  
→ La circuiteria è più semplice.

Prima vi sono i 3 codici di selezione per ALUOp.

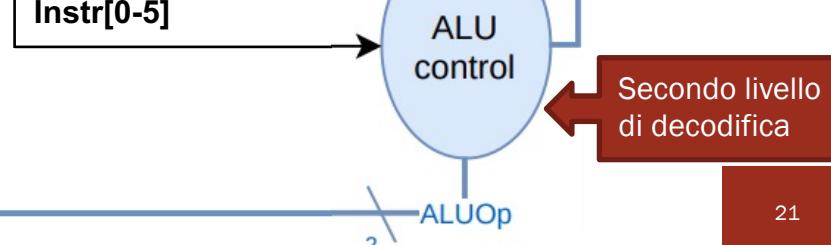
Poi, a seconda dei casi, viene considerato il campo funct nella istruzione.



Codice operativo istruzione	ALUOp	Operazione eseguita dall'istruzione	Campo funzione	Operazione dell'ALU	Ingresso di controllo alla ALU
Lw	00	load di 1 parola	XXXXXX	somma	0010
Sw	00	store di 1 parola	XXXXXX	somma	0010
Branch equal	01	salto condizionato all'uguaglianza	XXXXXX	sottrazione	0110
Tipo R	10	somma	100000	somma	0010
Tipo R	10	sottrazione	100010	sottrazione	0110
Tipo R	10	AND	100100	AND	0000
Tipo R	10	OR	100101	OR	0001
Tipo R	10	set less than	101010	set less than	0111

Instr[0-5]

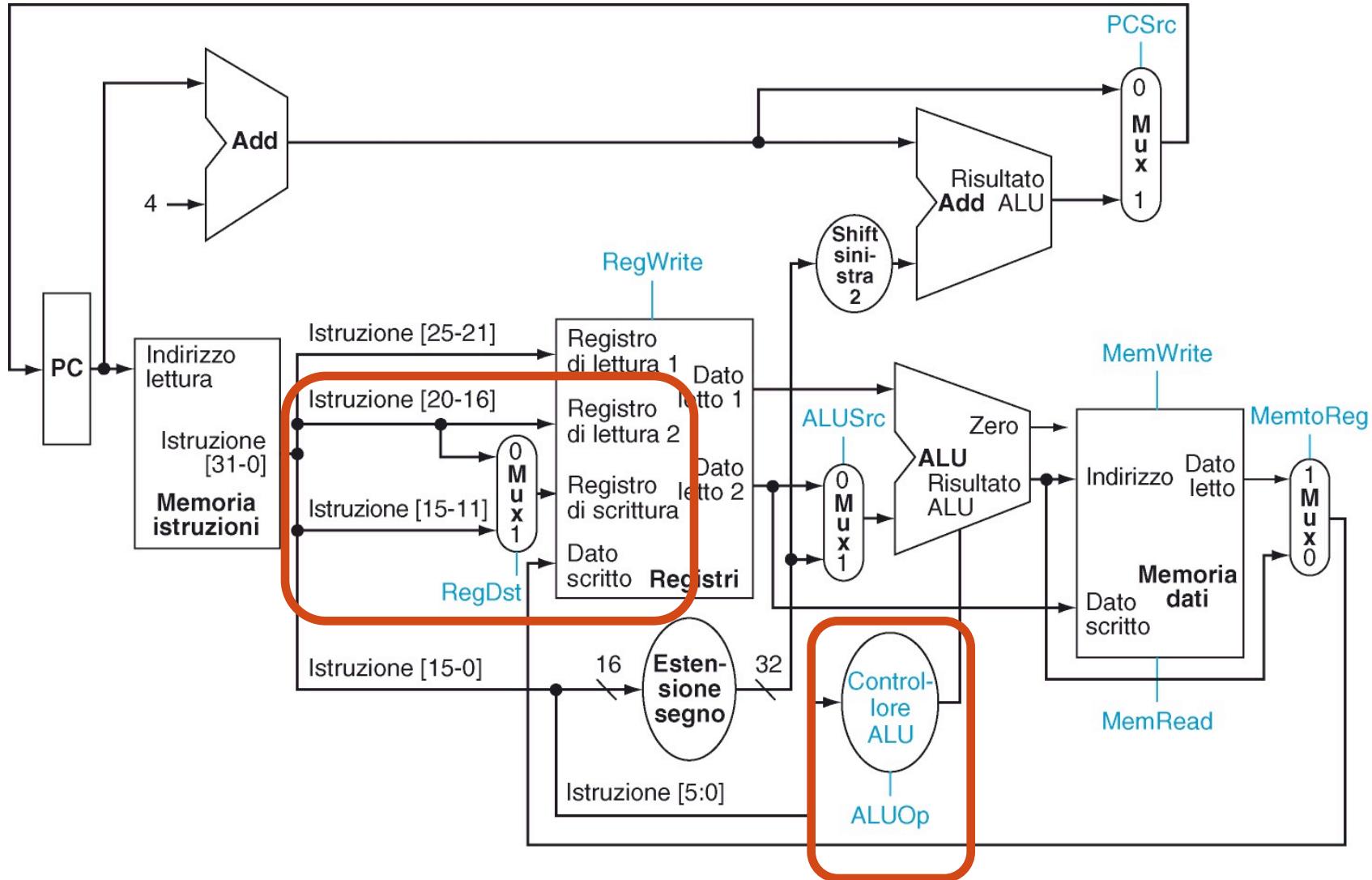
2



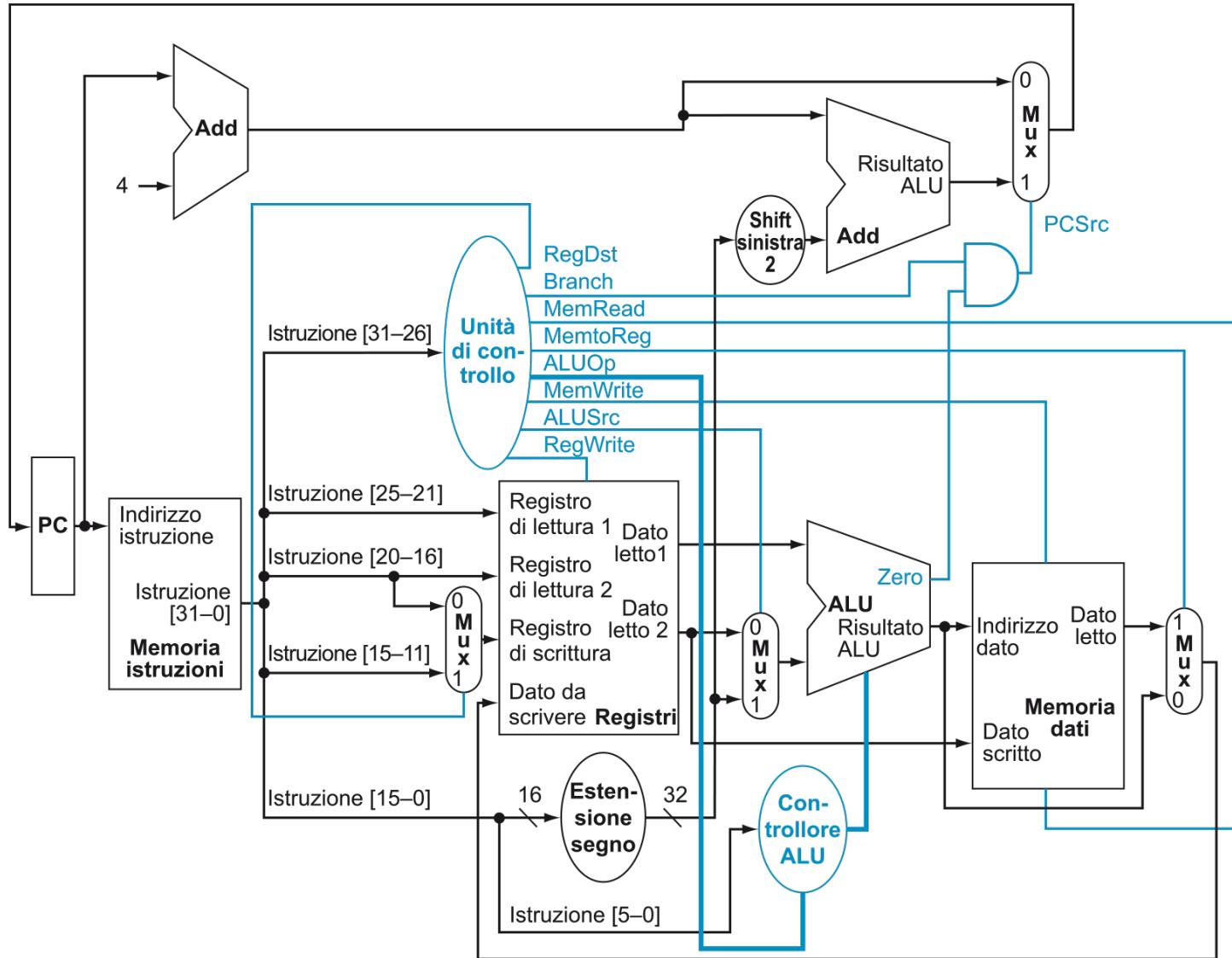
# Input di controllo e tabella di verità

ALUOp	Campo funz								Operazione
	ALUOp1	ALUOp2	F5	F4	F3	F2	F1	F0	
lw, sw	0	0	X	X	X	X	X	X	0010 +
beq	X	1	X	X	X	X	X	X	0110 -
add	1	X	X	X	0	0	0	0	0010 +
sub	1	X	X	X	0	0	1	0	0110 -
and	1	X	X	X	0	1	0	0	0000
or	1	X	X	X	0	1	0	1	0001
slt	1	X	X	X	1	0	1	0	0111

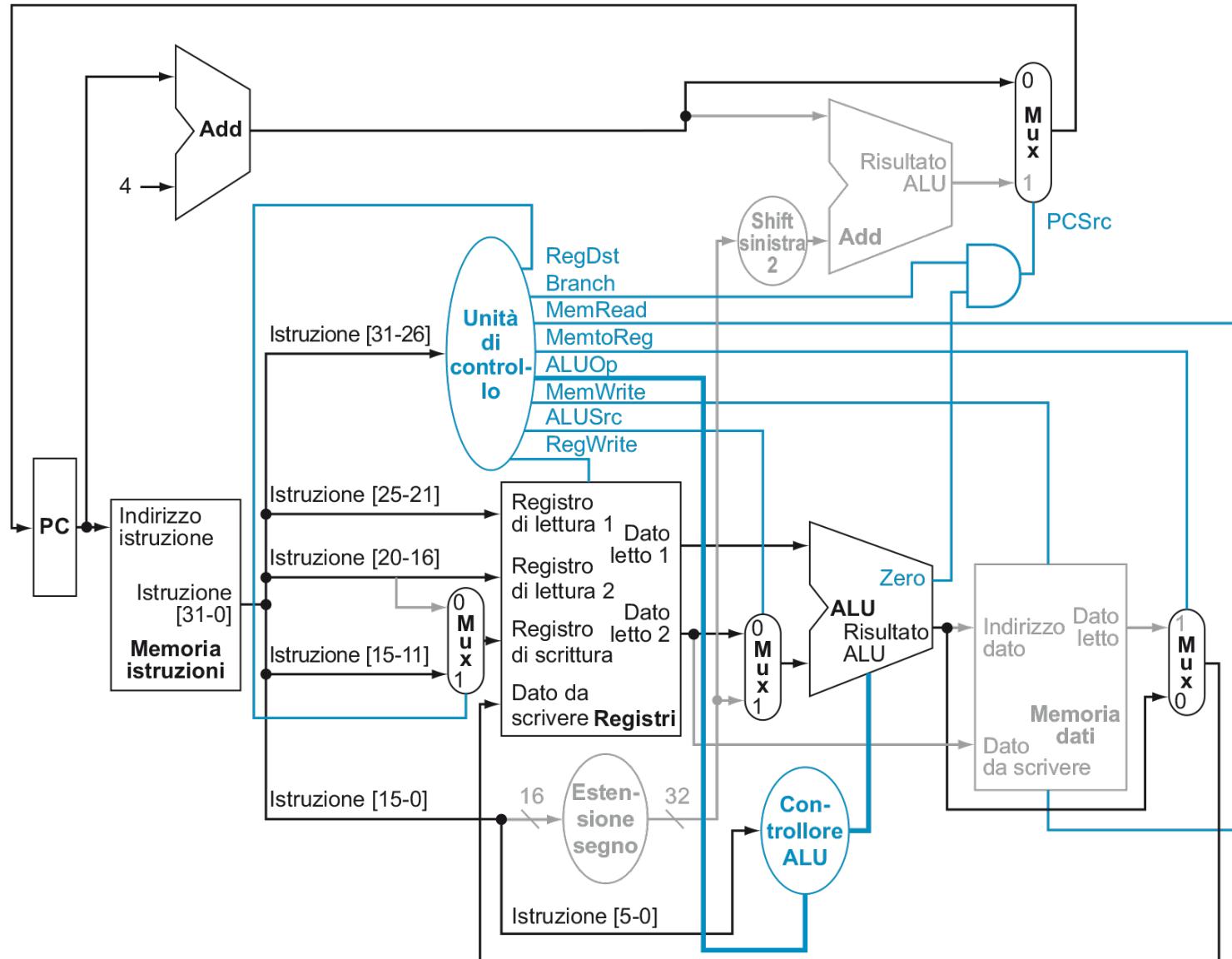
# Datapath completo (senza CU)



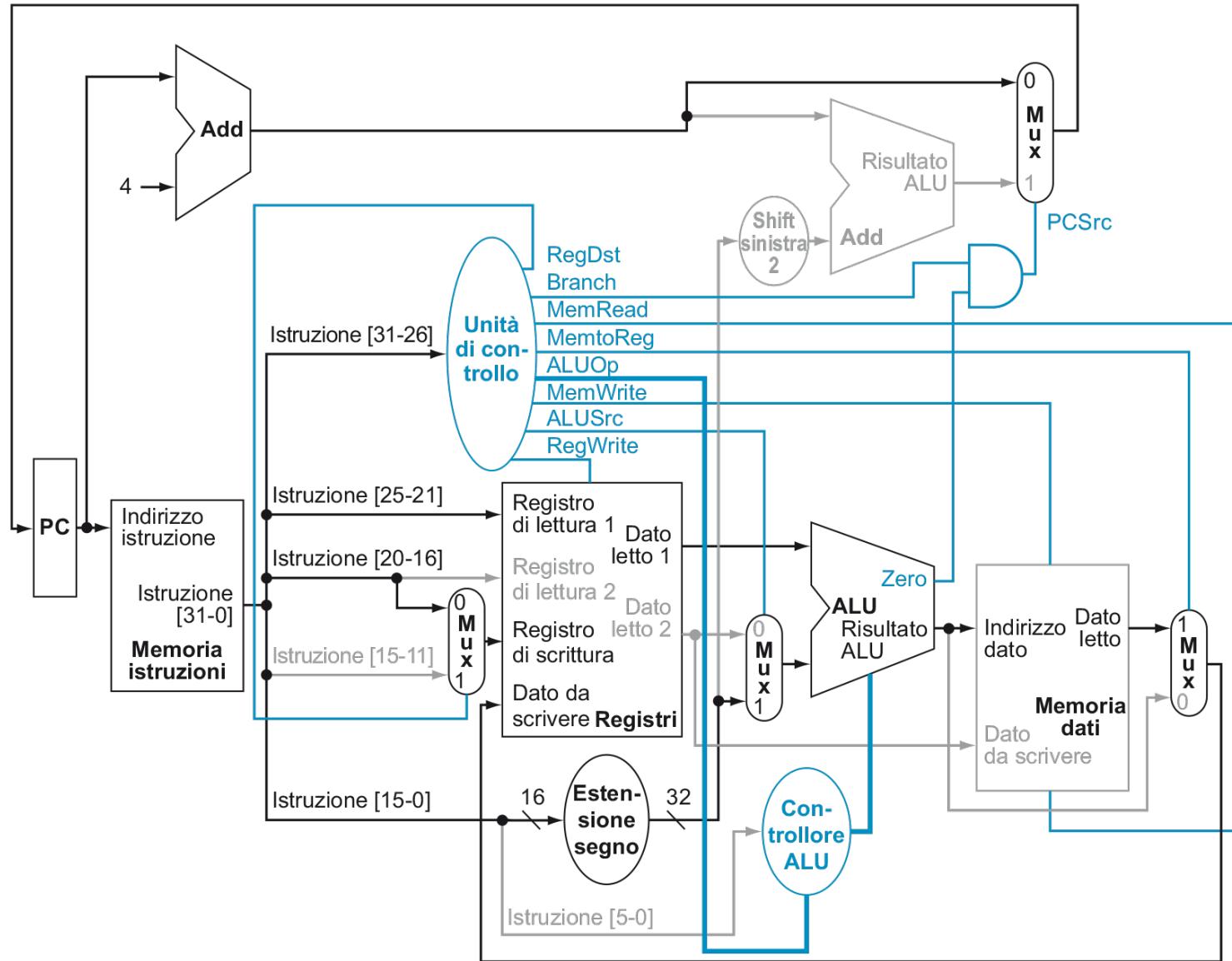
# Datapath completo



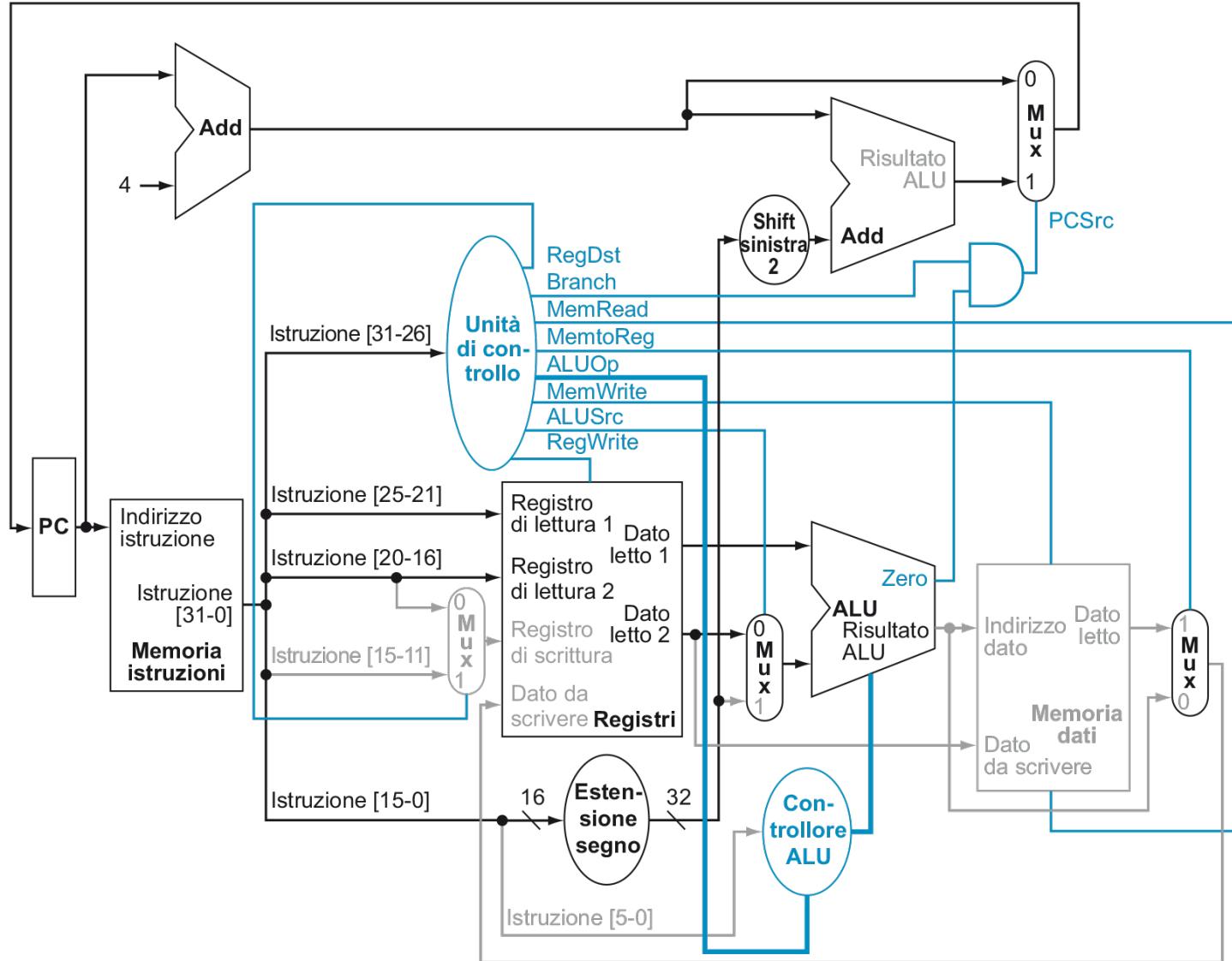
# Esecuzione di un'istruzione di tipo R



# Esecuzione di lw



# Esecuzione di beq



# Segnali 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

# I segnali da generare

L'ALU deve seguire 4 tipi di comportamento:

- Se l'istruzione è di **tipo R** eseguire l'operazione indicata dal campo **funct** dell'istruzione
- Se l'istruzione accede alla memoria (**lw, sw**) svolgere la **somma** che calcola l'indirizzo
- Se l'istruzione è un **beq** deve svolgere una **differenza**

Per codificare 3 comportamenti bastano 2 segnali dalla CU: **ALUOp1** ed **ALUOp0**

Quindi i segnali che la CU deve produrre per i diversi tipi di istruzione devono essere:

Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Tipo R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

X indica ‘don’t care’ (termini indifferenti)

# Tempi di esecuzione

Se conosciamo il tempo necessario a produrre i risultati delle diverse unità funzionali allora possiamo calcolare il tempo totale di ciascuna istruzione. Supponiamo che i tempi siano:

**accesso alla memoria** (dati o istruzione) = 100 ns (Fetch e MEM)

**ALU e sommatori** = 150 ns (EX e PC)

**accesso ai registri** (in lettura o scrittura) = 50 ns (ID e WB)

tutte le altre componenti = 0 ns

Allora i tempi di esecuzione delle istruzioni saranno

Istruzione	Instruction Fetch	Instruction Decode	Execution	MEM	Write Back	Totale
di tipo R	100	50	150		50	350
lw	100	50	150	100	50	450
sw	100	50	150	100		400
beq	100	50	150			300

NOTA: le due operazioni di somma per calcolare PC +4 (150ns) e salti condizionati (altri 150ns) sono svolte in parallelo al Fetch, Decode ed Execution e non allungano i tempi

# Esercizio per casa



Istruzione	Codice decimale	Codice binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

Progettare la tabella di verità della CU (solo per le righe necessarie) avente

- in ingresso: i 6 bit del codice operativo dell'istruzione
- in uscita: i 9 bit dei segnali di controllo da produrre:  
(RegDst, ALUSrc, MemtoReg, RegWrite, MemRead ,MemWrite, Branch, ALUOp1, ALUOp0)