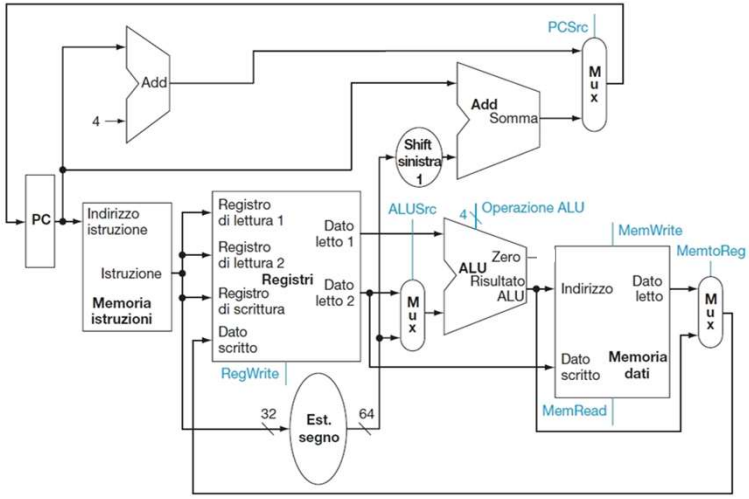
Il Processore RISC-V

Unità di Controllo

L'Unità di Controllo



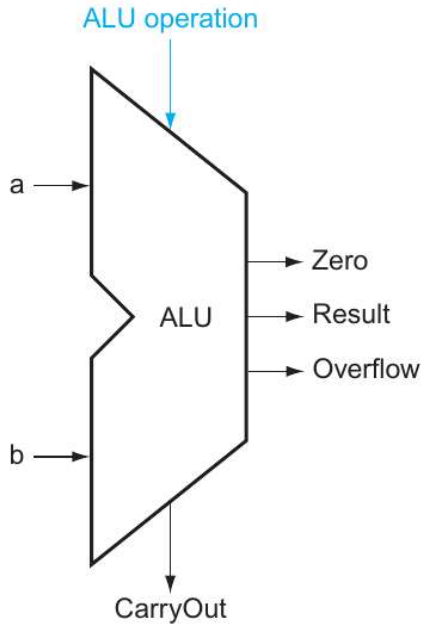
- Abbiamo completato un'unità di elaborazione dati elementare
- **Manca l'unità di controllo**
- Essa dovrà accettare dei valori di ingresso (una istruzioni da eseguire) e generare:
  - un segnale di scrittura per ciascun elemento di stato
  - un segnale di selezione per ciascun multiplexer
  - i segnali di controllo per la ALU
- Il controllo della ALU è particolare e conviene progettarlo prima delle altre parti dell'unità di controllo

Vediamo:

- load doubleword (`ld`)
- store doubleword (`sd`)
- branch if equal (`beq`)
- istruzioni aritmetico-logiche `add`, `sub`, `and`, `or`

Controllo Operazioni della ALU

- Per il controllo della ALU:
  - 1 bit per l'ingresso Ainvert
  - 1 bit per l'ingresso Bnegate
  - 2 bit per gli ingressi Operation
- Controllo a 4 bit per fare in modo che la ALU esegua la somma, la sottrazione, l'AND, l'OR, la NOR o la `slt`
- Facciamo l'unità di controllo per **AND, OR, ADD e «subtract»**



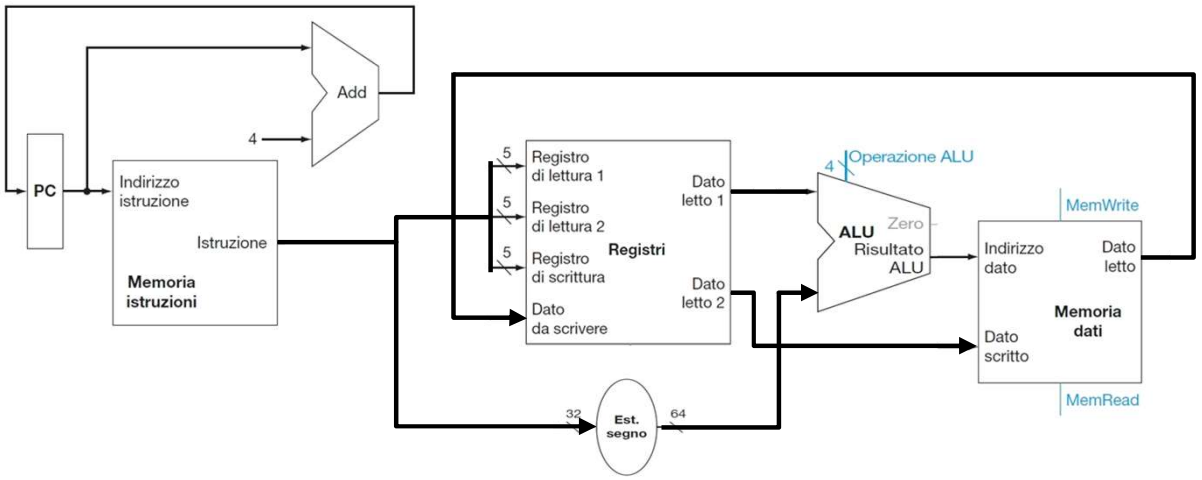
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR

Controllo Operazioni della ALU

- **load doubleword (`ld`)**
- **store doubleword (`sd`)**
- branch if equal (`beq`)
- istruzioni aritmetico-logiche `add`, `sub`, `and`, `or`

Per le istruzioni **load** e **store** la ALU deve eseguire una somma per calcolare l'indirizzo di memoria

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



Controllo Operazioni della ALU

- load doubleword (`ld`)
- store doubleword (`sd`)
- branch if equal (`beq`)
- **istruzioni aritmetico-logiche `add, sub, and, or`**

Per le istruzioni di **Tipo R** la ALU deve eseguire una delle operazioni (AND, OR, somma o sottrazione) in funzione del valore dei 7 bit del campo funz7 (bit 31:25) e dei 3 bit del campo funz3 (bit 14:12) dell'istruzione

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

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

Controllo Operazioni della ALU

- load doubleword (`ld`)
- store doubleword (`sd`)
- branch if equal (`beq`)
- **istruzioni aritmetico-logiche `add, sub, and, or`**

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

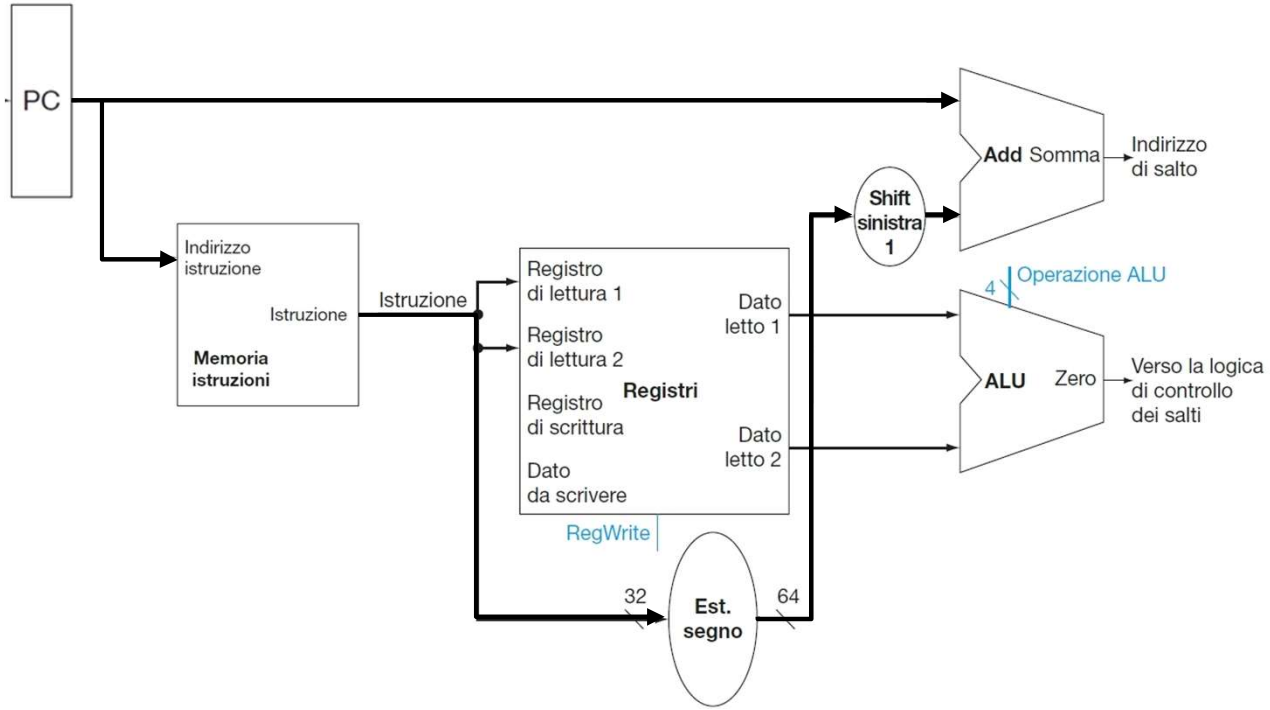
Tipo R

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Controllo Operazioni della ALU

- load doubleword (`ld`)
- store doubleword (`sd`)
- **branch if equal (`beq`)**
- istruzioni aritmetico-logiche `add`, `sub`, `and` e `or`

Per l'istruzione di salto condizionato, la ALU deve eseguire una sottrazione tra i due operandi e controllare se il risultato è 0.



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

Controllo Operazioni della ALU

Termine indifferente (don't care): una variabile di ingresso di una funzione logica che non ha effetto sull'uscita

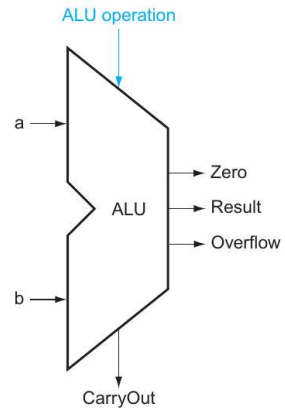
Codice operativo istruzione	ALUOp	Operazione eseguita dall'istruzione	Campo funz7	Campo funz3	Operazione dell'ALU	Ingresso di controllo alla ALU
ld	00	load di 1 parola doppia	XXXXXXX	XXX	somma	0010
sd	00	store di 1 parola doppia	XXXXXXX	XXX	somma	0010
beq	01	salto condizionato all'uguaglianza	XXXXXXX	XXX	sottrazione	0110
Tipo R	10	add	0000000	000	somma	0010
Tipo R	10	sub	0100000	000	sottrazione	0110
Tipo R	10	and	0000000	111	AND	0000
Tipo R	10	or	0000000	110	OR	0001

in uscita di UC per ALU (è classificazione "intermedia")

Controllo Operazioni della ALU

Termine indifferente (don't care): una variabile di ingresso di una funzione logica che non ha effetto sull'uscita

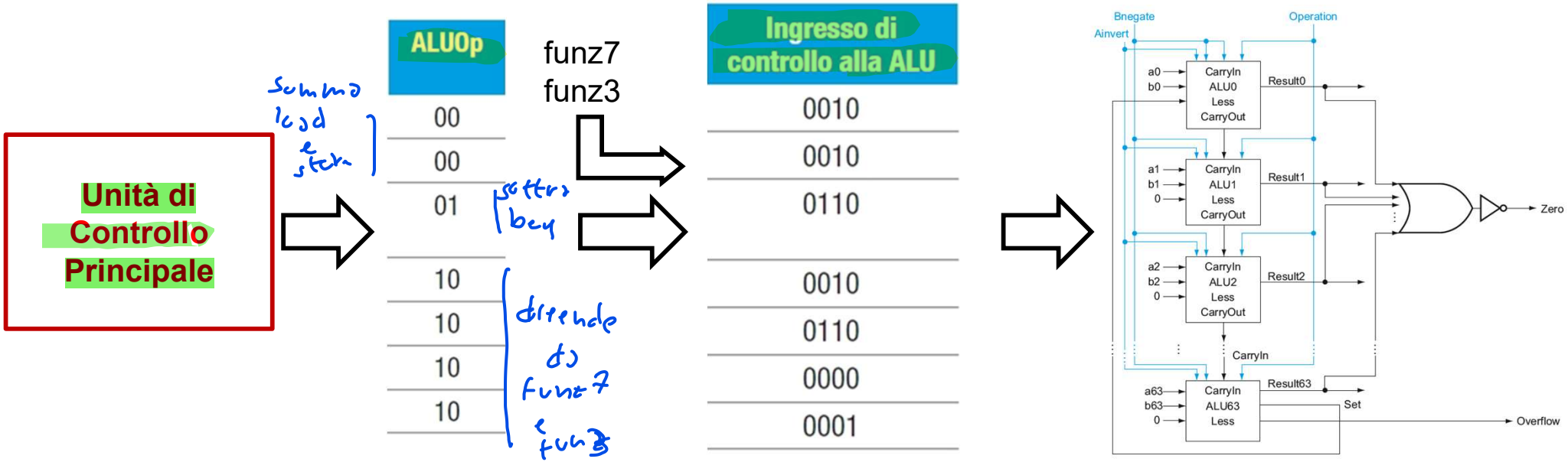
Codice operativo istruzione	ALUOp	Operazione eseguita dall'istruzione	Campo funz7	Campo funz3	Operazione dell'ALU	Ingresso di controllo alla ALU
ld	00	load di 1 parola doppia	XXXXXXX	XXX	somma	0010
sd	00	store di 1 parola doppia	XXXXXXX	XXX	somma	0010
beq	01	salto condizionato all'uguaglianza	XXXXXXX	XXX	sottrazione	0110
Tipo R	10	add	0000000	000	somma	0010
Tipo R	10	sub	0100000	000	sottrazione	0110
Tipo R	10	and	0000000	111	AND	0000
Tipo R	10	or	0000000	110	OR	0001



- I 4 bit di controllo della ALU possono essere generati utilizzando una piccola **unità di controllo** che riceve in ingresso i campi **funz7** e **funz3** dell'istruzione e un campo di controllo su 2 bit, chiamato **ALUOp**
- **ALUOp** = 00 ➔ somma per le istruzioni di load e store
- **ALUOp** = 01 ➔ sottrazione per le beq
- **ALUOp** = 10 ➔ l'operazione viene determinata dal contenuto dei campi **funz7** e **funz3**

Controllo Operazioni della ALU

- **Livelli multipli di decodifica**: **l'unità di controllo principale** imposta i bit **ALUOp**, poi utilizzati come **ingressi** **dell'unità di controllo della ALU**, che genera i **segnali** effettivi della ALU



- Più livelli di controllo possono ridurre le dimensioni dell'unità di controllo principale, riducendo la **latenza** dell'unità di controllo
- Spesso l'unità di controllo è un elemento critico per la definizione della durata del **ciclo di clock**

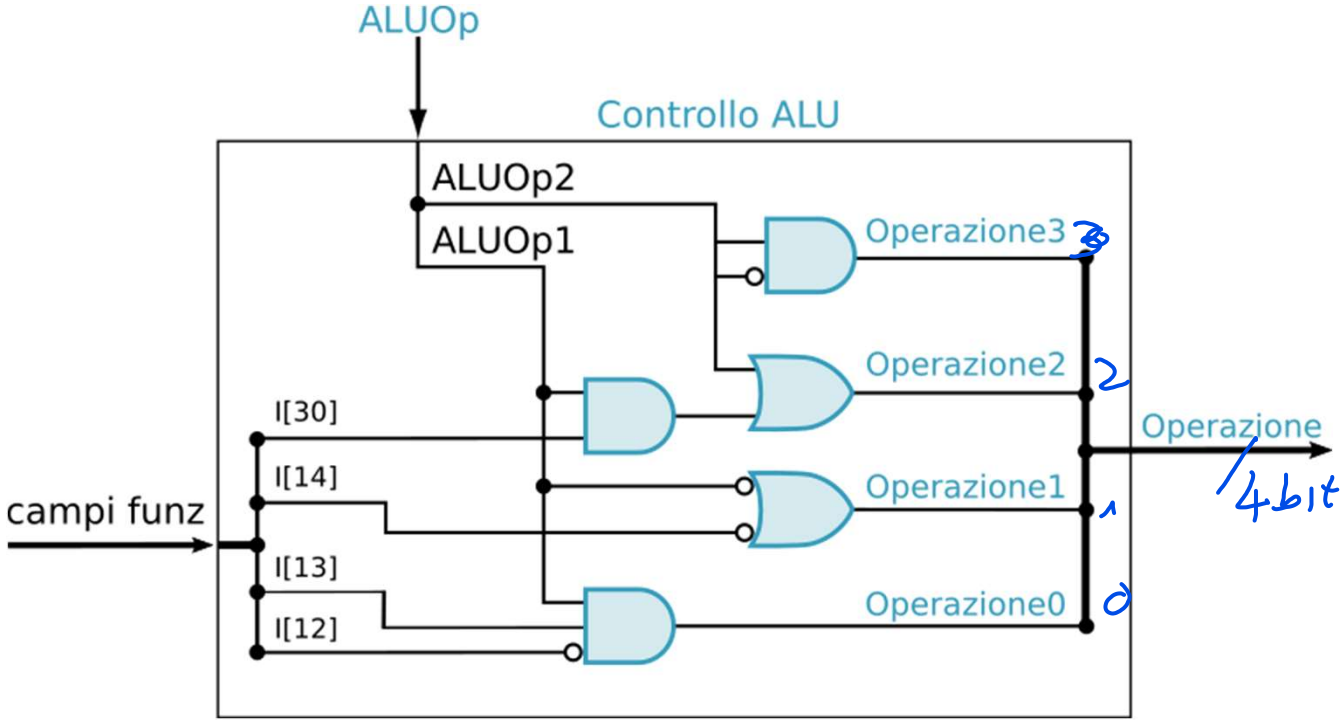
ALUOp + Funz ➔ Controllo della ALU

Termini indifferenti (don't care)

	ALUOp		Campo funz7							Campo funz3			Operazione
	ALUOp1	ALUop2	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
ld / sd	0	0	X	X	X	X	X	X	X	X	X	X	0010
beq	X	1	X	X	X	X	X	X	X	X	X	X	0110
add	1	X	0	0	0	0	0	0	0	0	0	0	0010
sub	1	X	0	1	0	0	0	0	0	0	0	0	0110
and	1	X	0	0	0	0	0	0	0	1	1	1	0000
or	1	X	0	0	0	0	0	0	0	1	1	0	0001

Tabella di verità: Corrispondenza tra i 2 bit del campo ALUOp e i bit dei campi funz con i 4 bit di controllo della ALU che selezionano l'operazione

	ALUOp		Campo funz7							Campo funz3			Operazione
	ALUOp1	ALUop2	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
ld / sd	0	0	X	X	X	X	X	X	X	X	X	X	0010
beq	X	1	X	X	X	X	X	X	X	X	X	X	0110
add	1	X	0	0	0	0	0	0	0	0	0	0	0010
sub	1	X	0	1	0	0	0	0	0	0	0	0	0110
and	1	X	0	0	0	0	0	0	0	1	1	1	0000
or	1	X	0	0	0	0	0	0	0	1	1	0	0001



Formato delle Istruzioni (Recap)

Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	codop
(b) Tipo I	immediate[11:0]		rs1	funz3	rd	codop
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	codop

- **Tipo R**: i campi rs1 e rs2 contengono il numero dei registri sorgenti e rd contiene il numero del registro destinazione. L'operazione da eseguire è codificata nei campi funz3 e funz7
- **Tipo I, load**: rs1 è il registro base il cui contenuto viene sommato al campo immediato di 12 bit per ottenere l'indirizzo del dato in memoria. Il campo rd è il registro destinazione per il valore letto
- **Tipo S, store**: rs1 è il registro base il cui contenuto viene sommato al campo immediato di 12 bit (suddiviso in 2 gruppi) per ottenere l'indirizzo del dato in memoria. Il campo rs2 è il registro sorgente il cui valore viene copiato nella memoria
- **Tipo SB:** I registri rs1 e rs2 vengono confrontati. Il campo indirizzo immediato di 12 bit viene preso, il suo bit di segno esteso, fatto scorrere a sinistra di una posizione e sommato al PC per calcolare l'indirizzo di destinazione del salto

VIP

Unità di Controllo Principale

Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	codop
(b) Tipo I	immediate[11:0]		rs1	funz3	rd	codop
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	codop

- **Codice operativo (codop):** il campo che denota il tipo di operazione e il formato di un'istruzione
- Il **codop** (**opcode** in inglese) è sempre contenuto nei bit 6:0
- A seconda del codop, il campo funz3 (bit 14:12) e funz7 (31:25) servono come campi di **estensione del codice operativo**

t vecchio

Unità di Controllo Principale

Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	51 (dec)
(b) Tipo I	immediate[11:0]		rs1	funz3	rd	load = 3
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	store = 35
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	beq = 99

- **Codice operativo (codop):** il campo che denota il tipo di operazione e il formato di un'istruzione
- Il **codop** (**opcode** in inglese) è sempre contenuto nei bit 6:0
- A seconda del codop, il campo funz3 (bit 14:12) e funz7 (31:25) servono come campi di **estensione del codice operativo**

Unità di Controllo Principale

Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	codop
(b) Tipo I	immediate[11:0]		rs1	funz3	rd	codop
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	codop

- Il primo registro operando si trova sempre nei bit in posizione 19:15 (rs1) per le istruzioni di tipo R e le istruzioni di salto condizionato
- Questo campo specifica anche il registro base per le istruzioni di load e di store

Unità di Controllo Principale

Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	codop
(b) Tipo I	immed[11:0]		rs1	funz3	rd	codop
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	codop

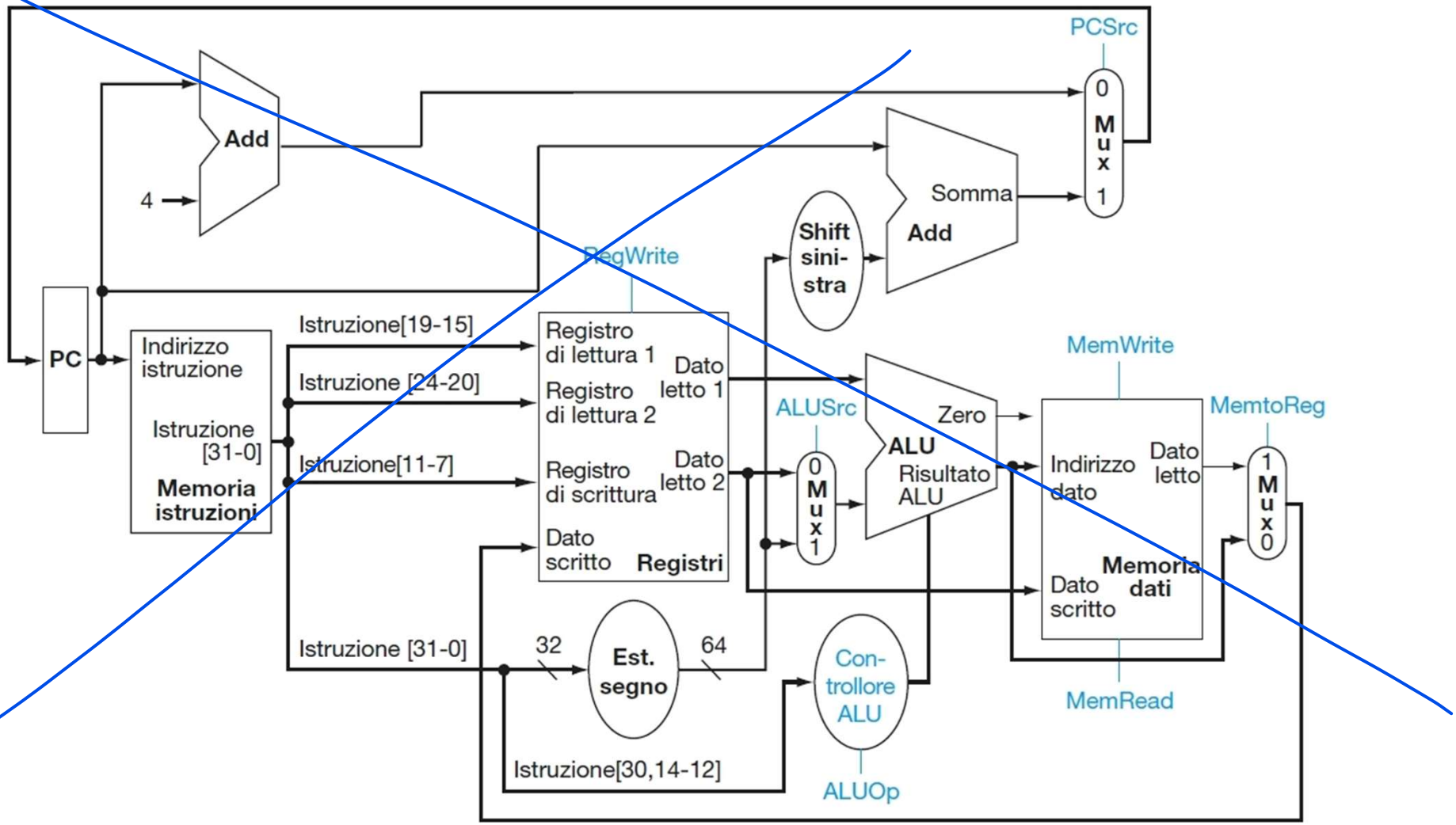
- Il secondo registro operando si trova sempre nei bit in posizione 24:20 (rs2) per le istruzioni di tipo R e le istruzioni di salto condizionato.
- Questo campo specifica anche il registro il cui contenuto viene copiato in memoria nelle istruzioni di store.

Unità di Controllo Principale

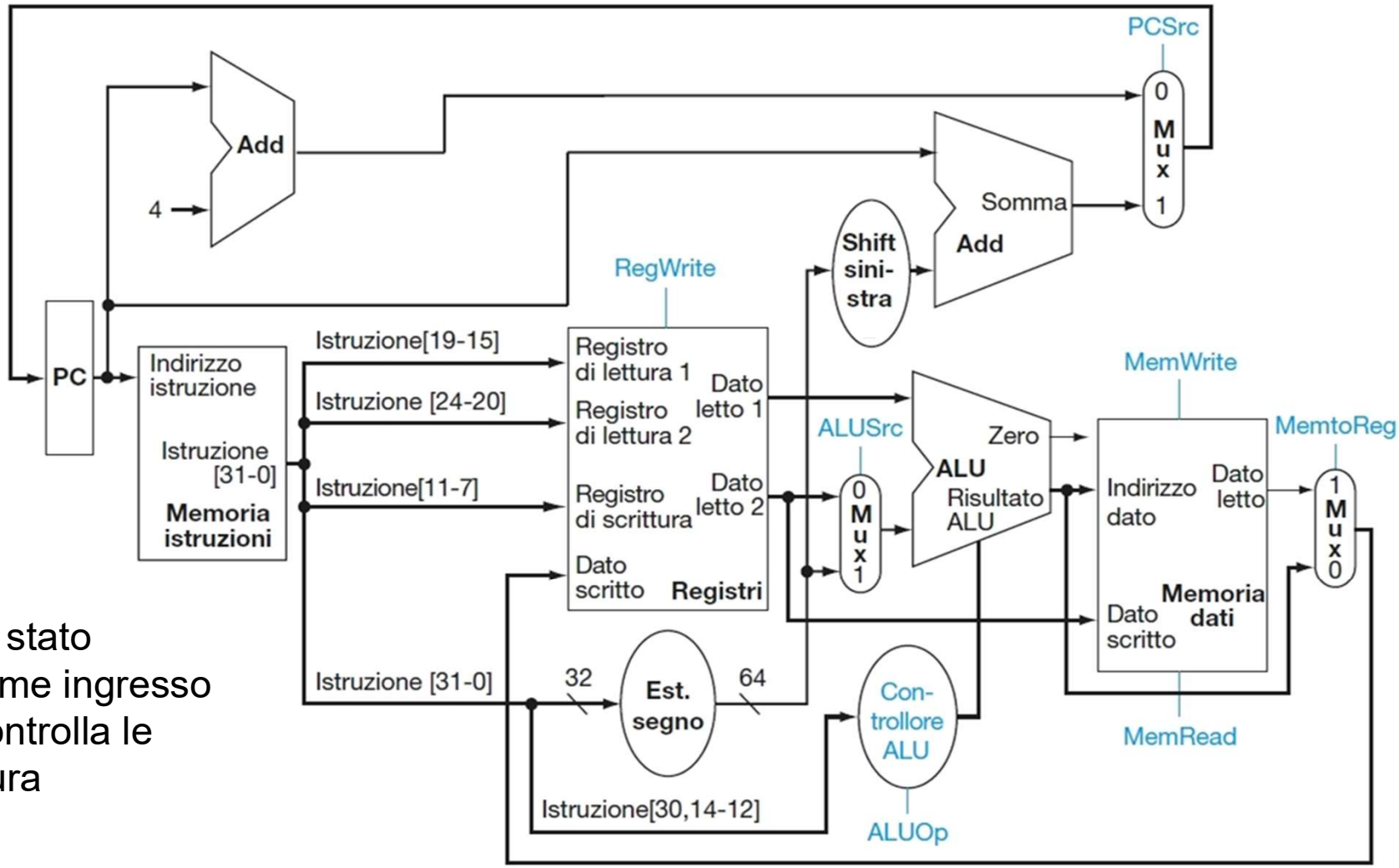
Nome (posizione dei bit)	Campi					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) Tipo R	funz7	rs2	rs1	funz3	rd	codop
(b) Tipo I	immediate[11:0]		rs1	funz3	rd	codop
(c) Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
(d) Tipo SB	immed[12,10:5]	rs2	rs1	funz3	immed[4:1,11]	codop

- Il secondo operando può anche essere costituito dai 12 bit di offset delle istruzioni di branch o di load/store.
- Il registro destinazione si trova sempre nei bit in posizione 11:7 (rd) per le istruzioni di tipo R e di load.

L'Unità di Elaborazione con i Segnali di Controllo



L'Unità di Elaborazione con i Segnali di Controllo



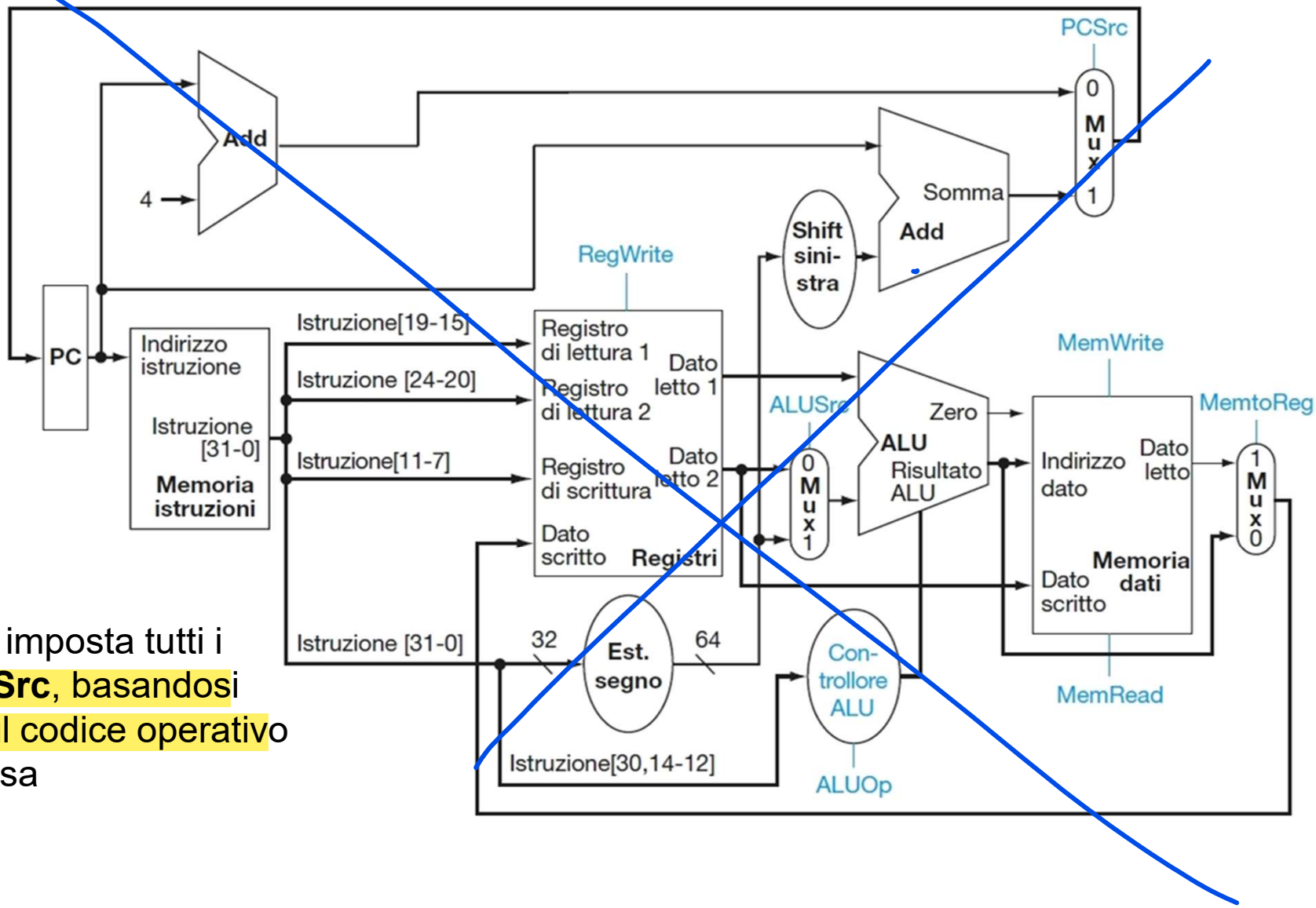
Tutti gli elementi di stato ricevono il clock come ingresso implicito. Il clock controlla le operazioni di scrittura

Effetto dei Sei Segnali di Controllo (senza ALUOp)

Nome del segnale	Effetto quando non asserito	Effetto quando asserito
RegWrite	Nullo	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 12 bit del campo immediato 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	Nullo	Il dato della memoria nella posizione puntata dall'indirizzo viene inviato in uscita sulla linea "Dato letto"
MemWrite	Nullo	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

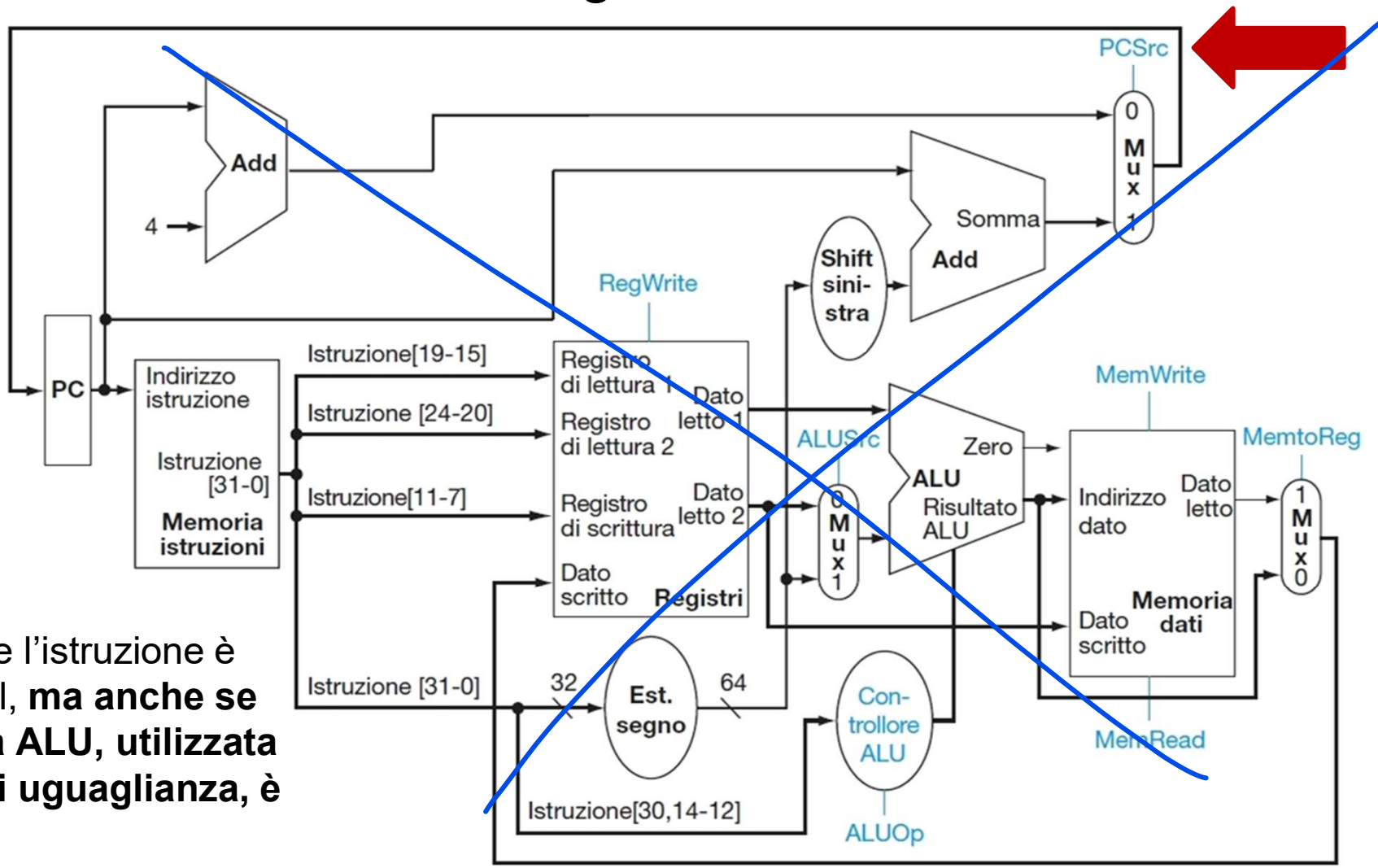
Quando il segnale di controllo a 1 bit di un multiplexer a due vie è asserito, il multiplexer seleziona l'ingresso etichettato con 1; in caso contrario il multiplexer seleziona l'ingresso etichettato con 0

L'Unità di Elaborazione con i Segnali di Controllo



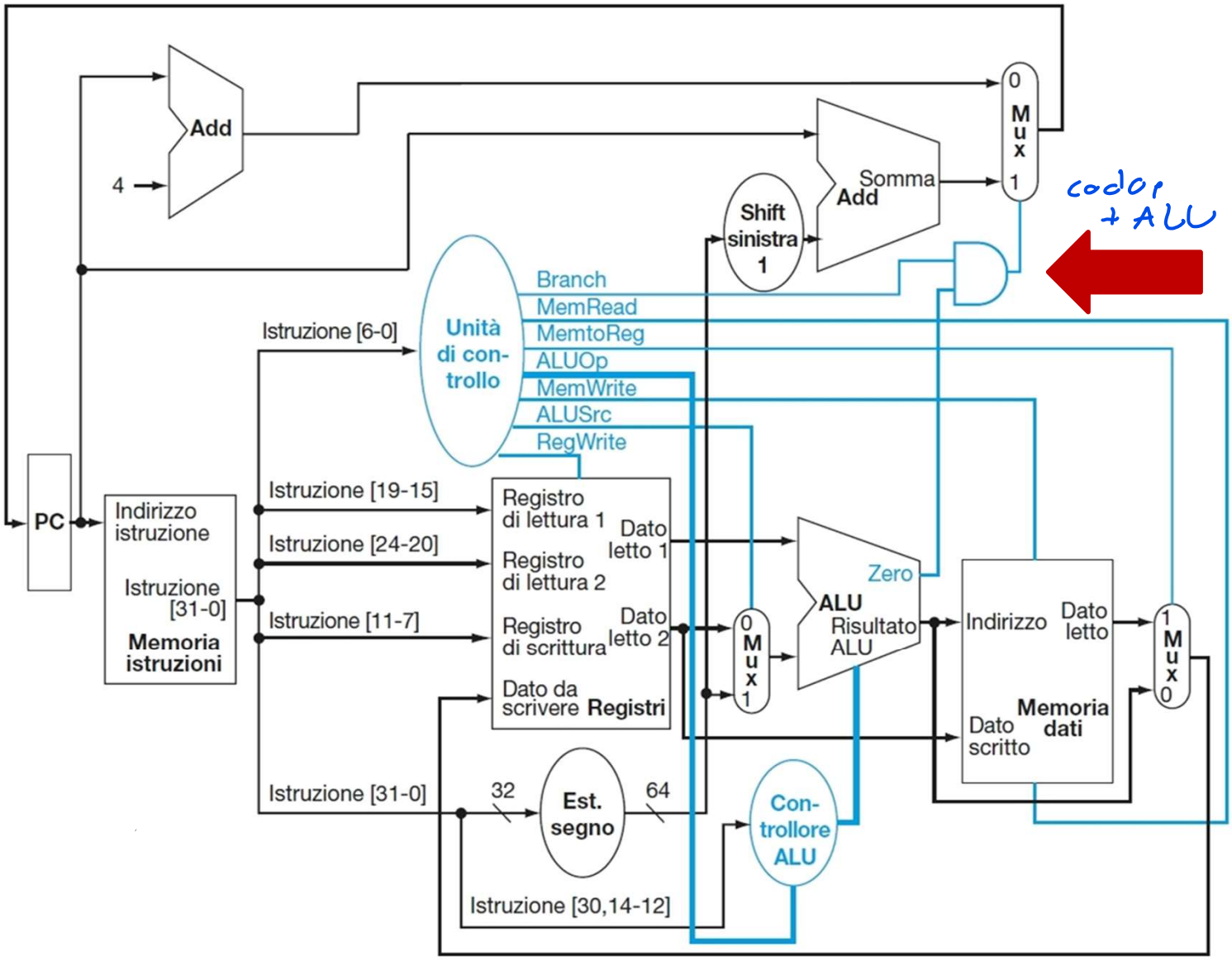
L'unità di controllo imposta tutti i segnali tranne **PCSrc**, basandosi esclusivamente sul codice operativo dell'istruzione stessa

L'Unità di Elaborazione con i Segnali di Controllo



PCSrc: asserito se l'istruzione è una branch if equal, **ma anche se l'uscita Zero della ALU, utilizzata per il confronto di uguaglianza, è vera**.

L'Unità di Elaborazione Completa



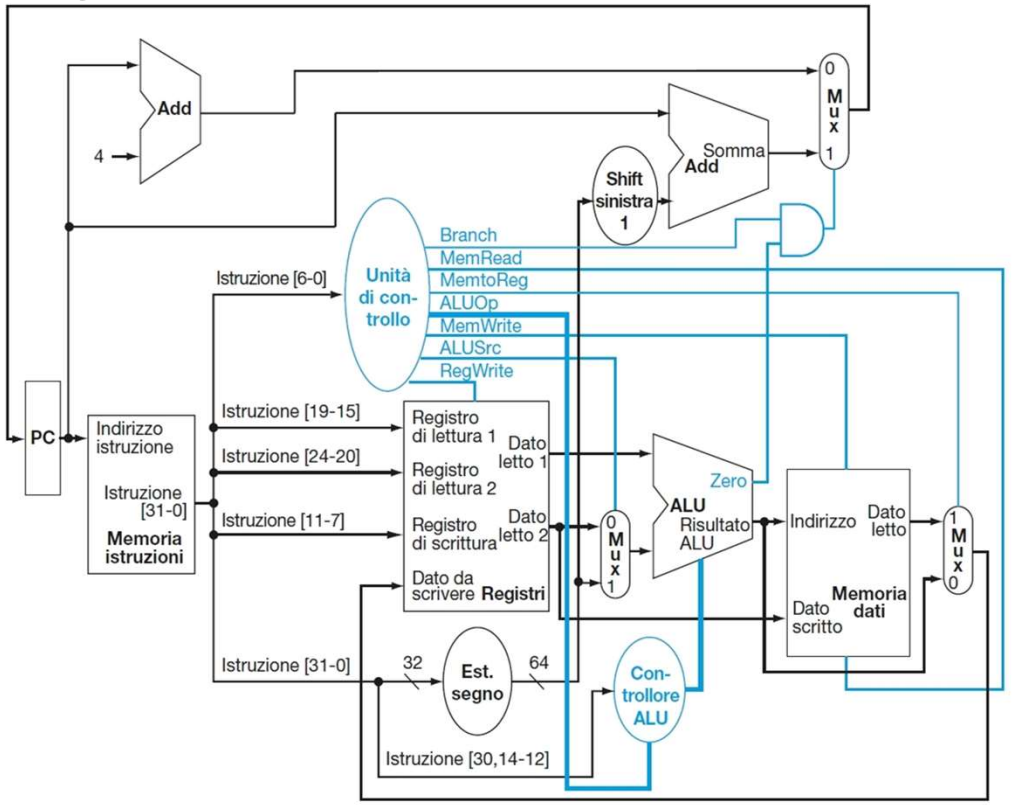
L'Unità di Controllo – Riassunto

Ingresso:

- 7 bit dell'istruzione (**codop**)

Uscita (8 bit)

- due segnali a 1 bit utilizzati per controllare i multiplexer (**ALUSrc** e **MemtoReg**)
- tre segnali a 1 bit per controllare lettura e scrittura del register file e della memoria dati (**RegWrite**, **MemRead** e **MemWrite**)
- un segnale a 1 bit utilizzato come segnale di controllo per i salti condizionati (**Branch**)
- un segnale di controllo a 2 bit per la ALU (**ALUOp**)

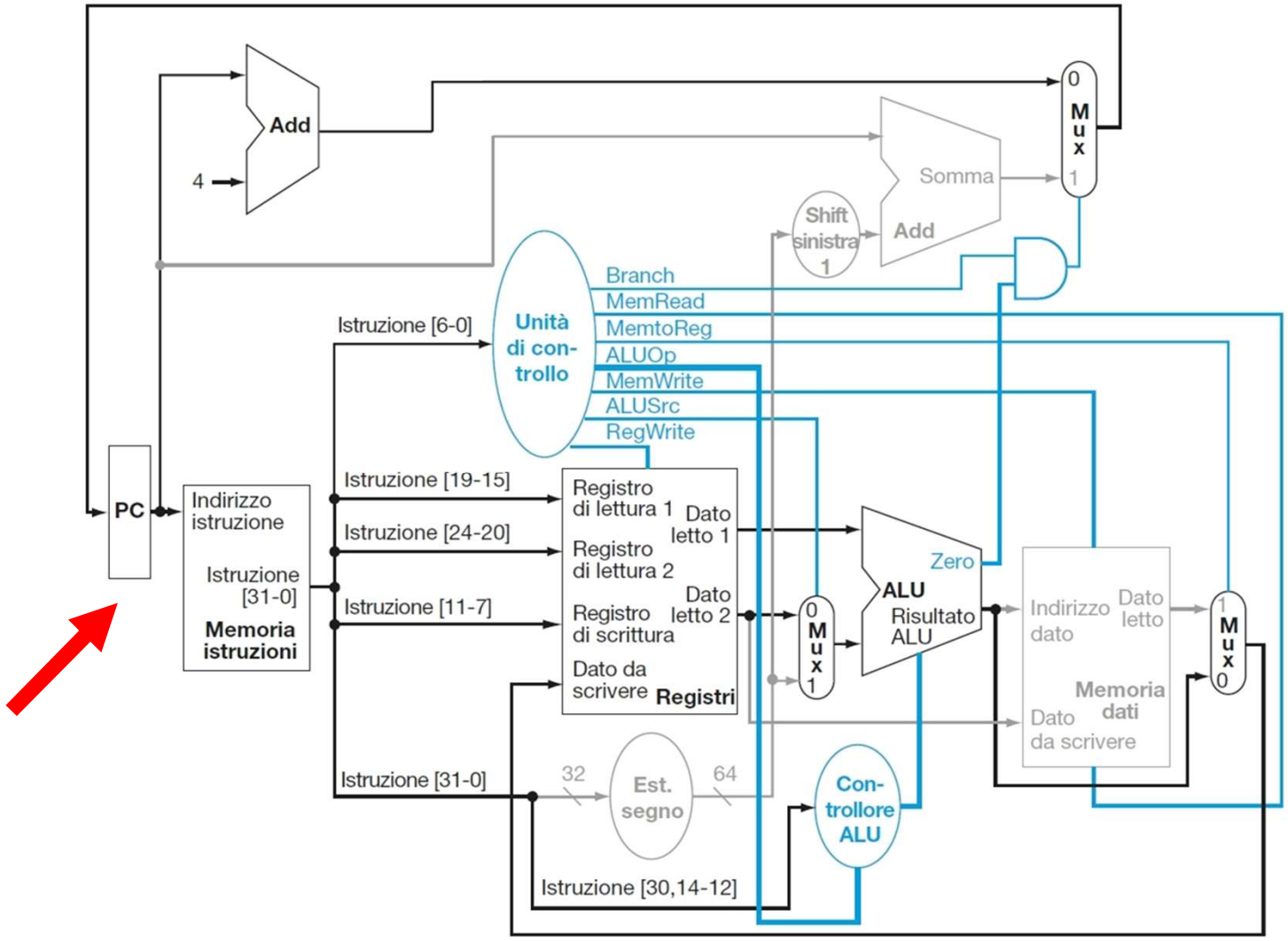


Valore delle Linee di Controllo

Istruzione	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Tipo R	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

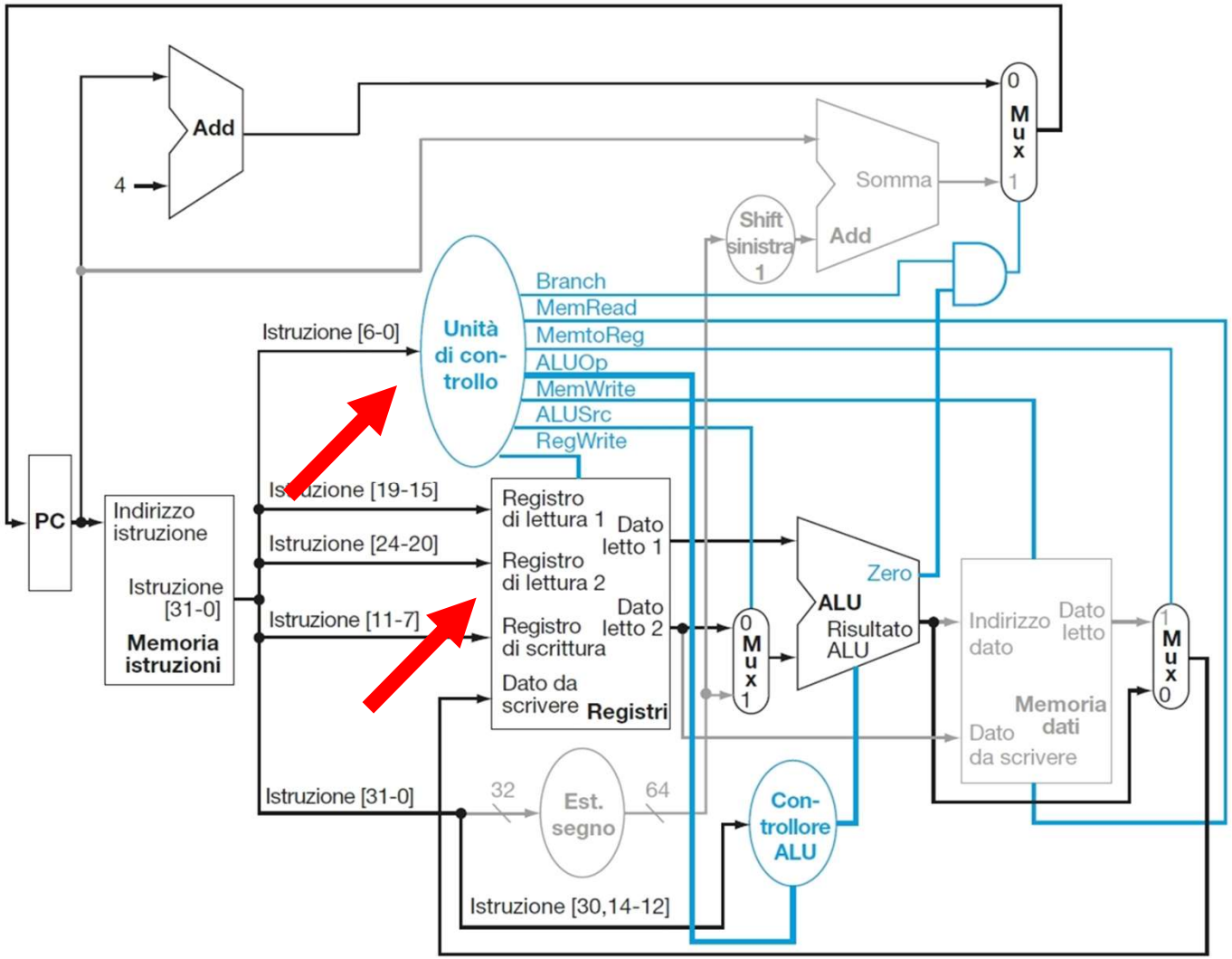
Esecuzione di un'Istruzione di Tipo R – 1 `add x1,x2,x3`

- L'istruzione viene prelevata dalla memoria istruzioni e si incrementa il PC



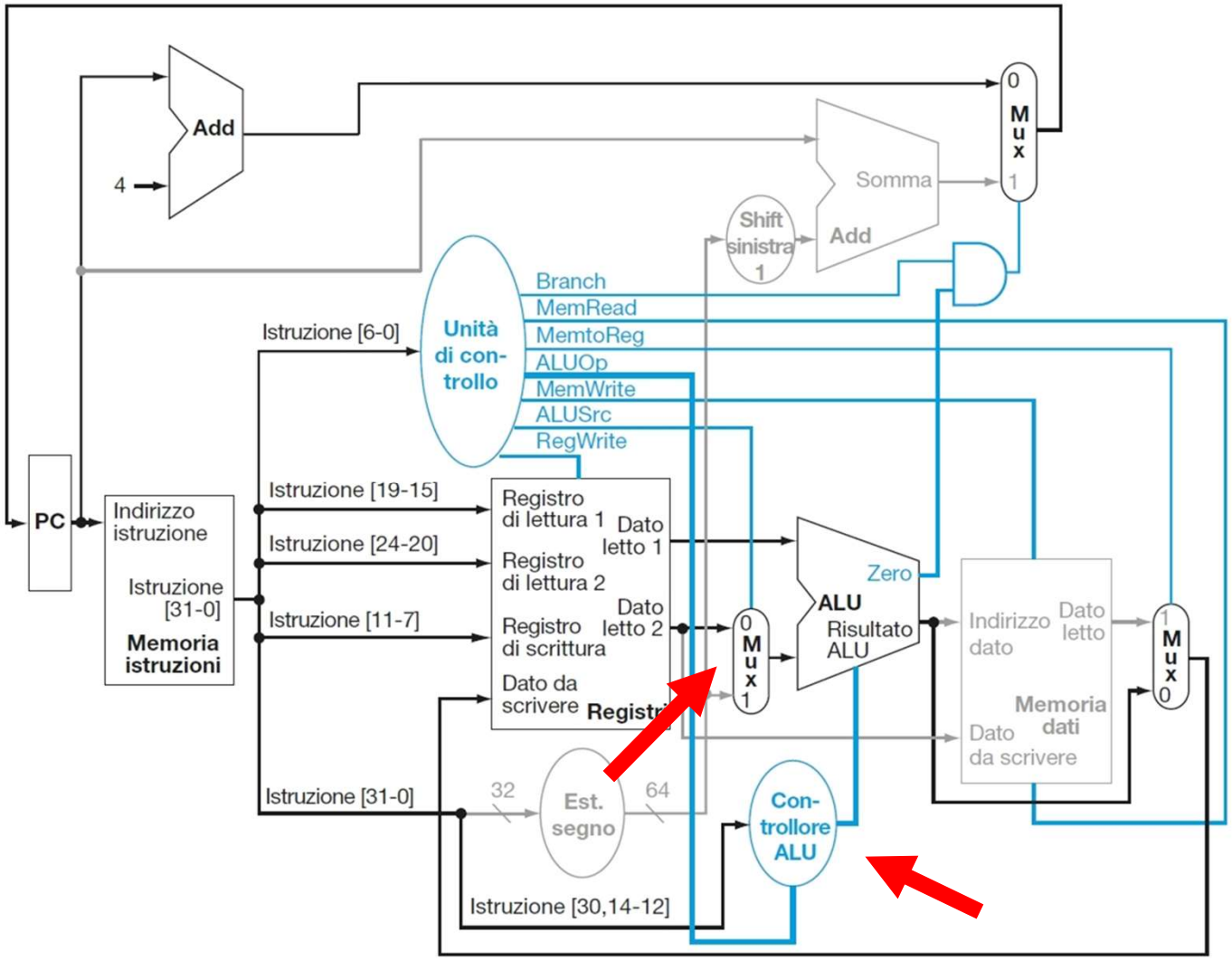
Esecuzione di un'Istruzione di Tipo R – 2 `add x1,x2,x3`

- I due registri x2 e x3 vengono letti dal register file
- L'unità di controllo principale calcola il valore da attribuire alle linee di controllo



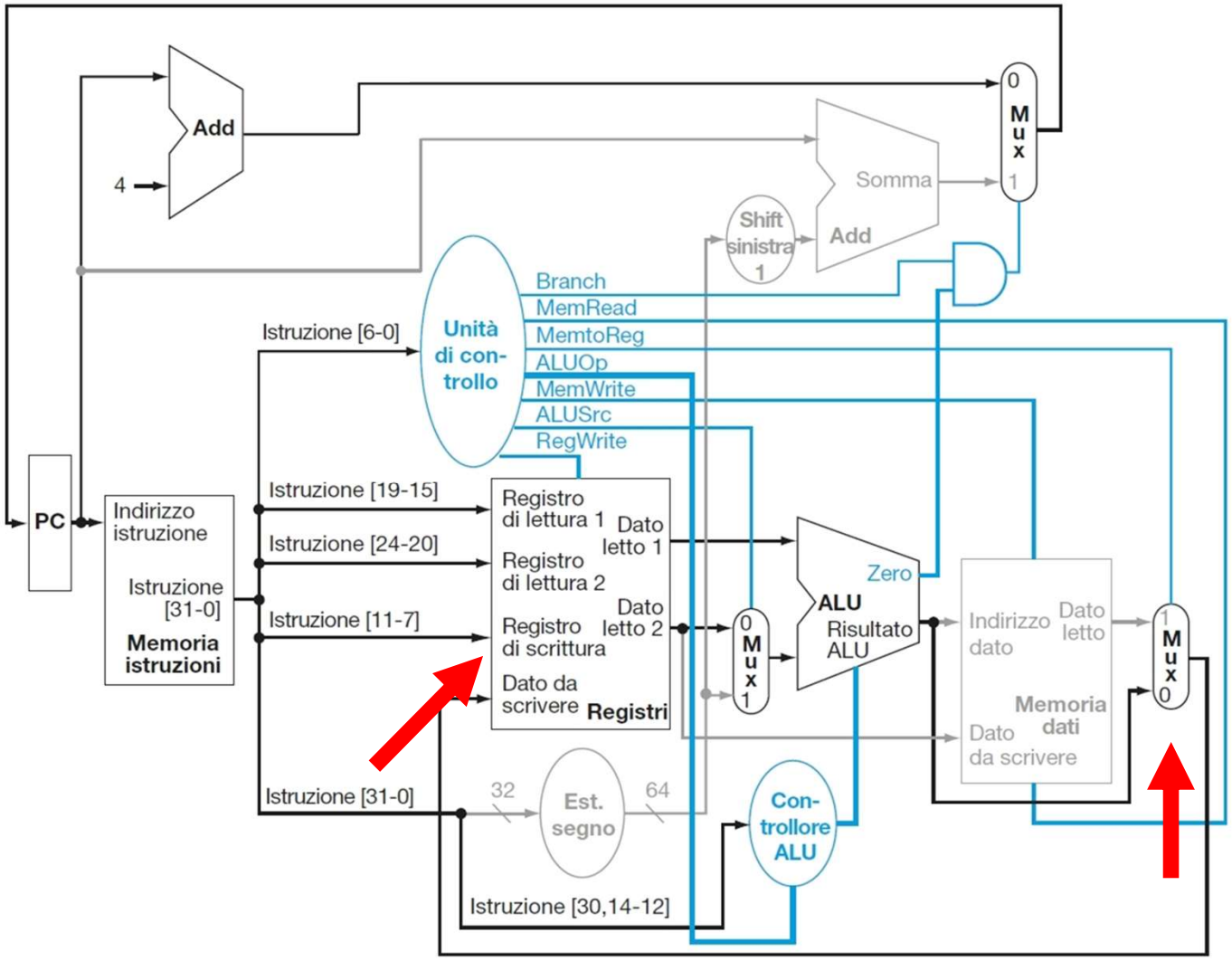
Esecuzione di un'Istruzione di Tipo R – 3 `add x1,x2,x3`

- La ALU elabora i dati letti dal register file, utilizzando alcuni bit del codice operativo per selezionare l'operazione della ALU



Esecuzione di un'Istruzione di Tipo R – 4 `add x1,x2,x3`

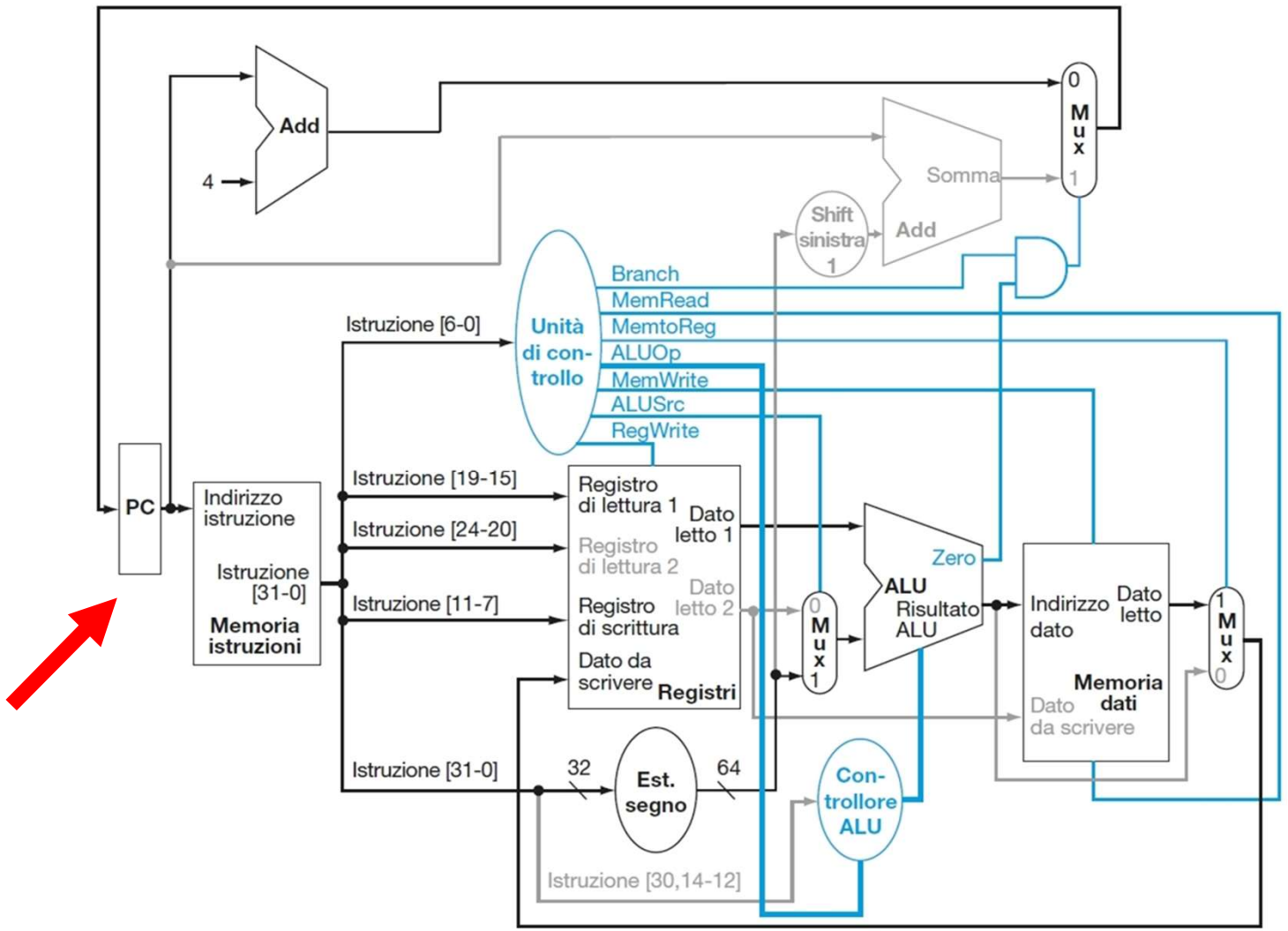
- Il risultato calcolato dalla ALU viene scritto nel registro destinazione (x1) del register file



Esecuzione di Load – 1

`ld x1,offset(x2)`

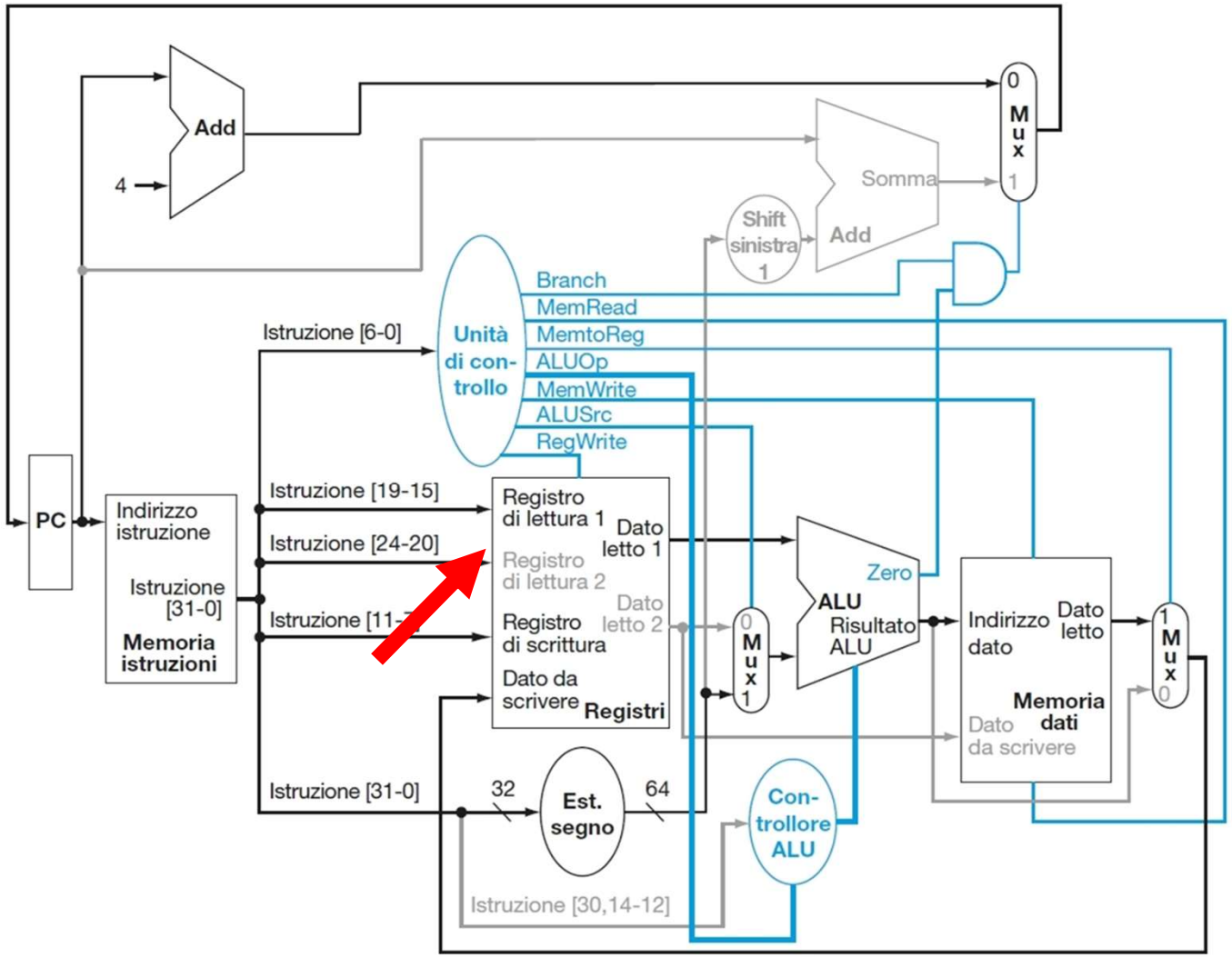
- L'istruzione viene prelevata dalla memoria istruzioni e si incrementa il PC



Esecuzione di Load – 2

`ld x1,offset(x2)`

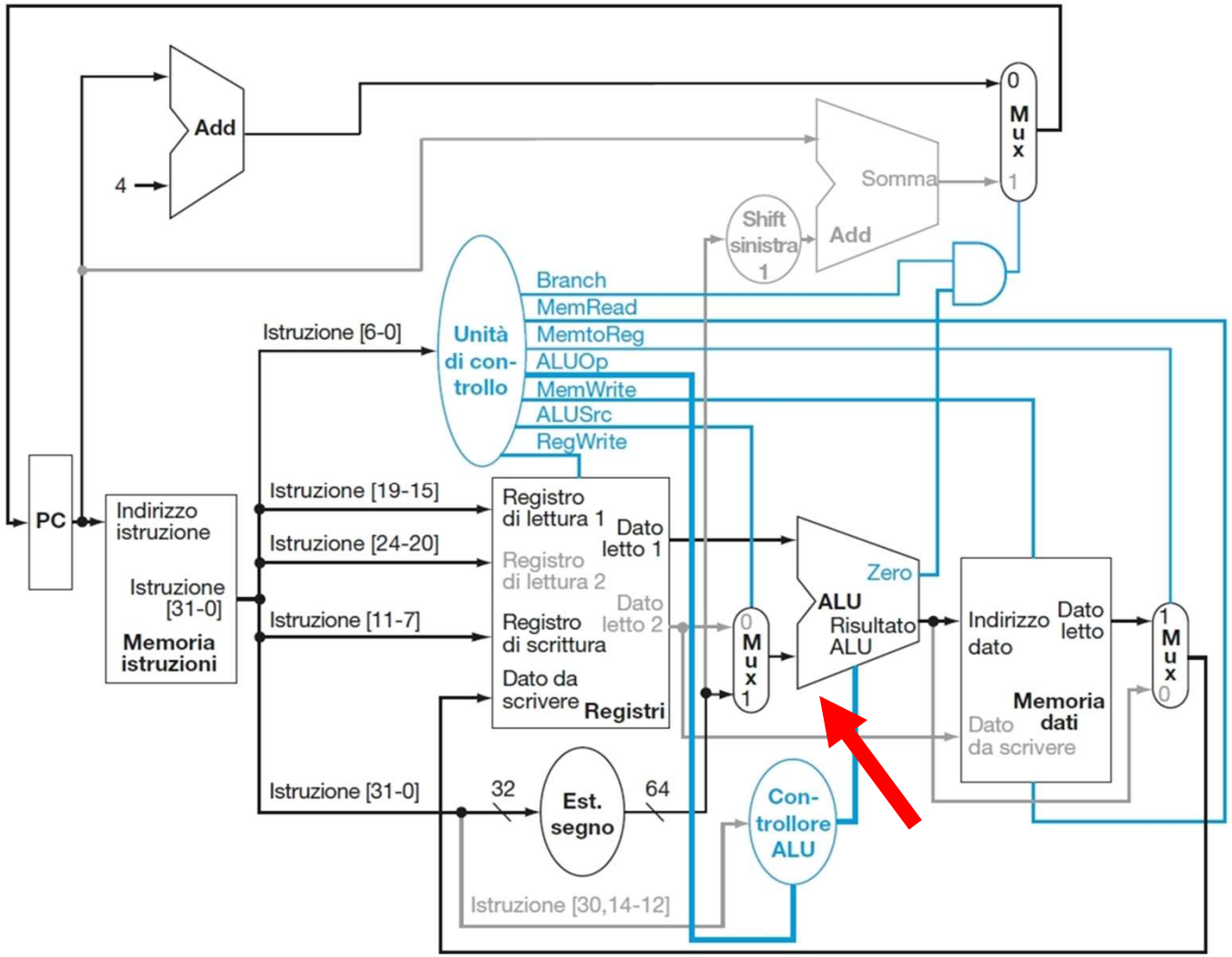
- Viene letto il contenuto di un registro (x2) dal register file



Esecuzione di Load – 3

`ld x1,offset(x2)`

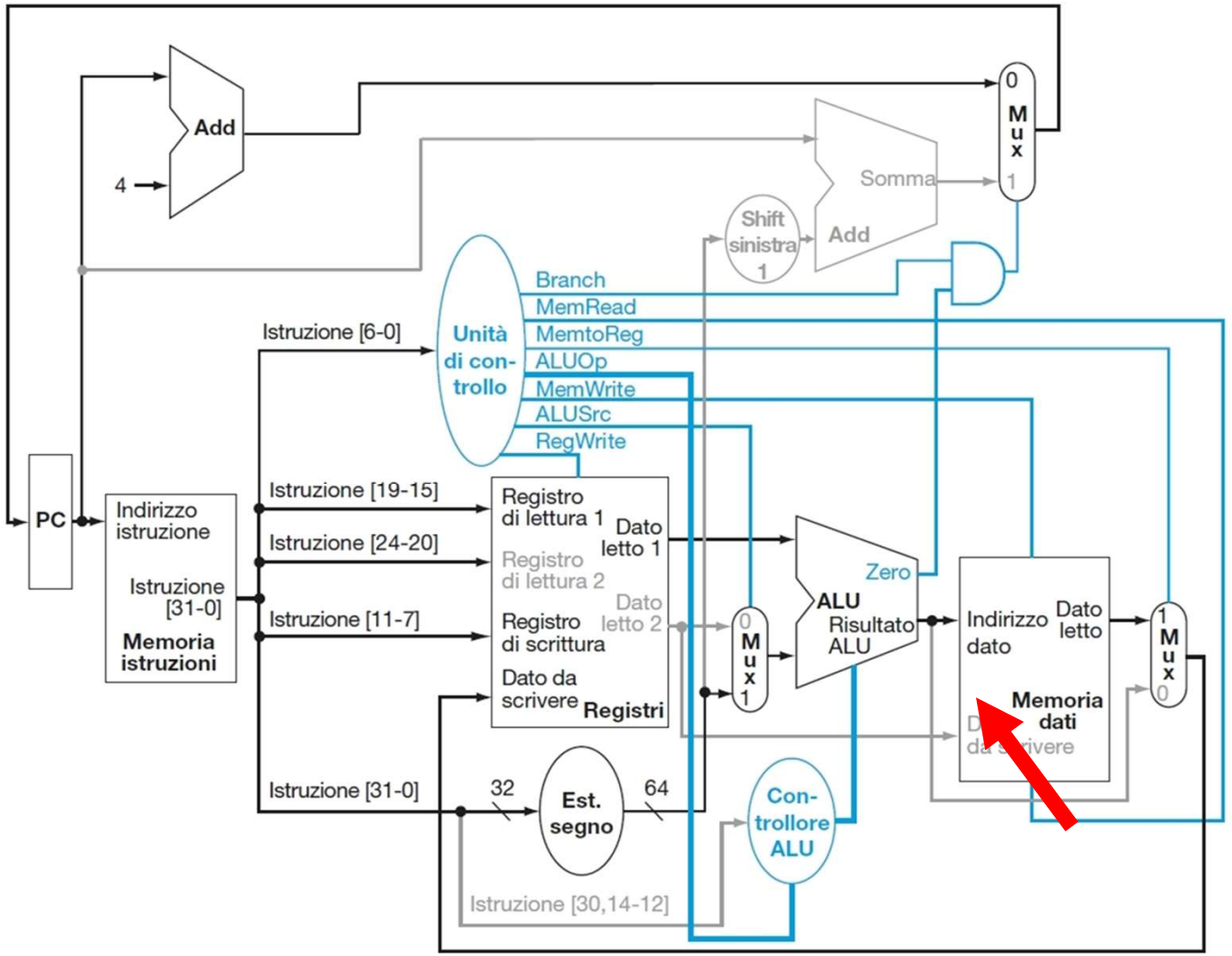
- La ALU somma il valore letto dal register file ai 12 bit del campo offset dell'istruzione, dotati di segno ed estesi a 64 bit



Esecuzione di Load – 4

`ld x1,offset(x2)`

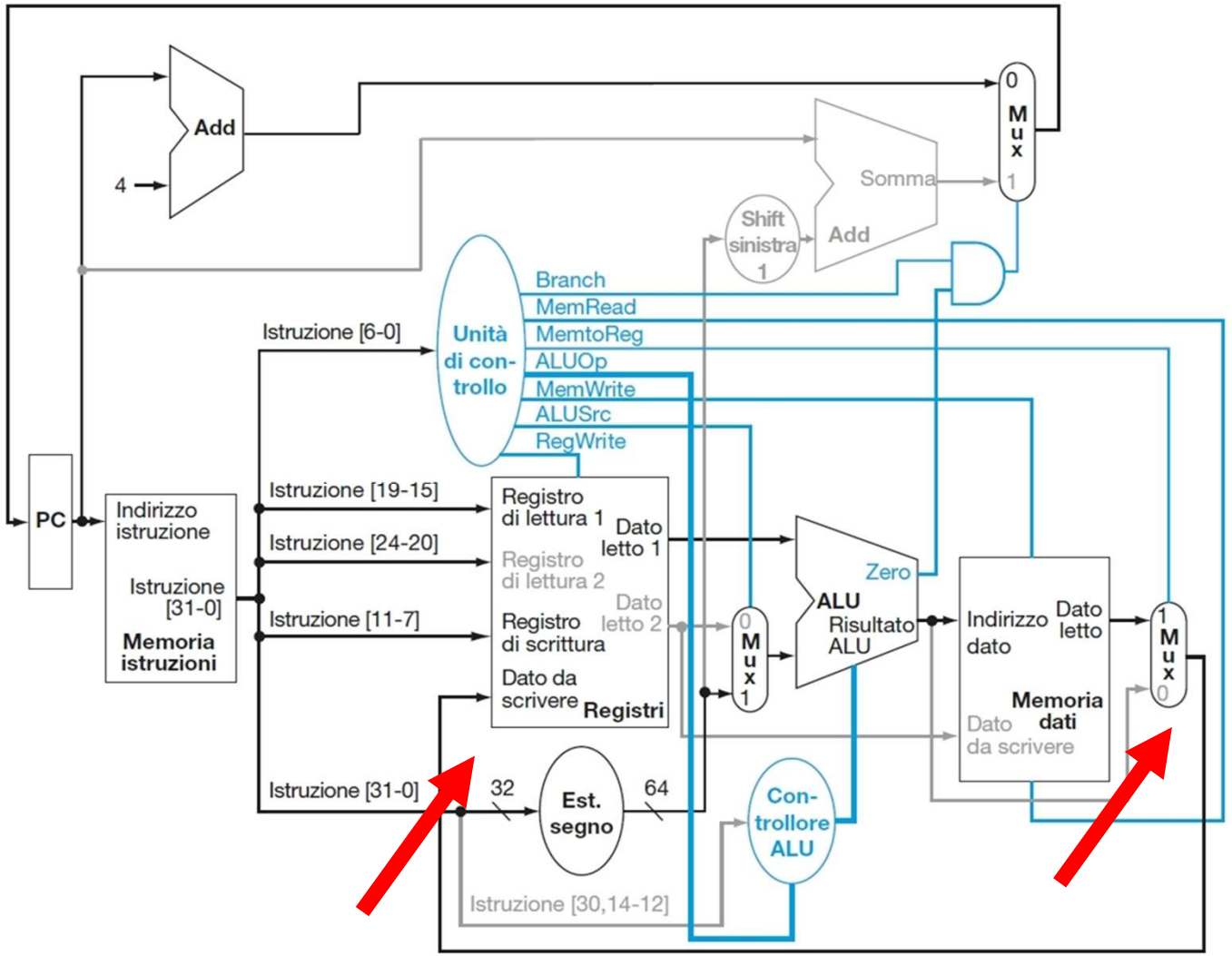
- La somma calcolata dalla ALU viene utilizzata come indirizzo per la memoria dati



Esecuzione di Load – 5

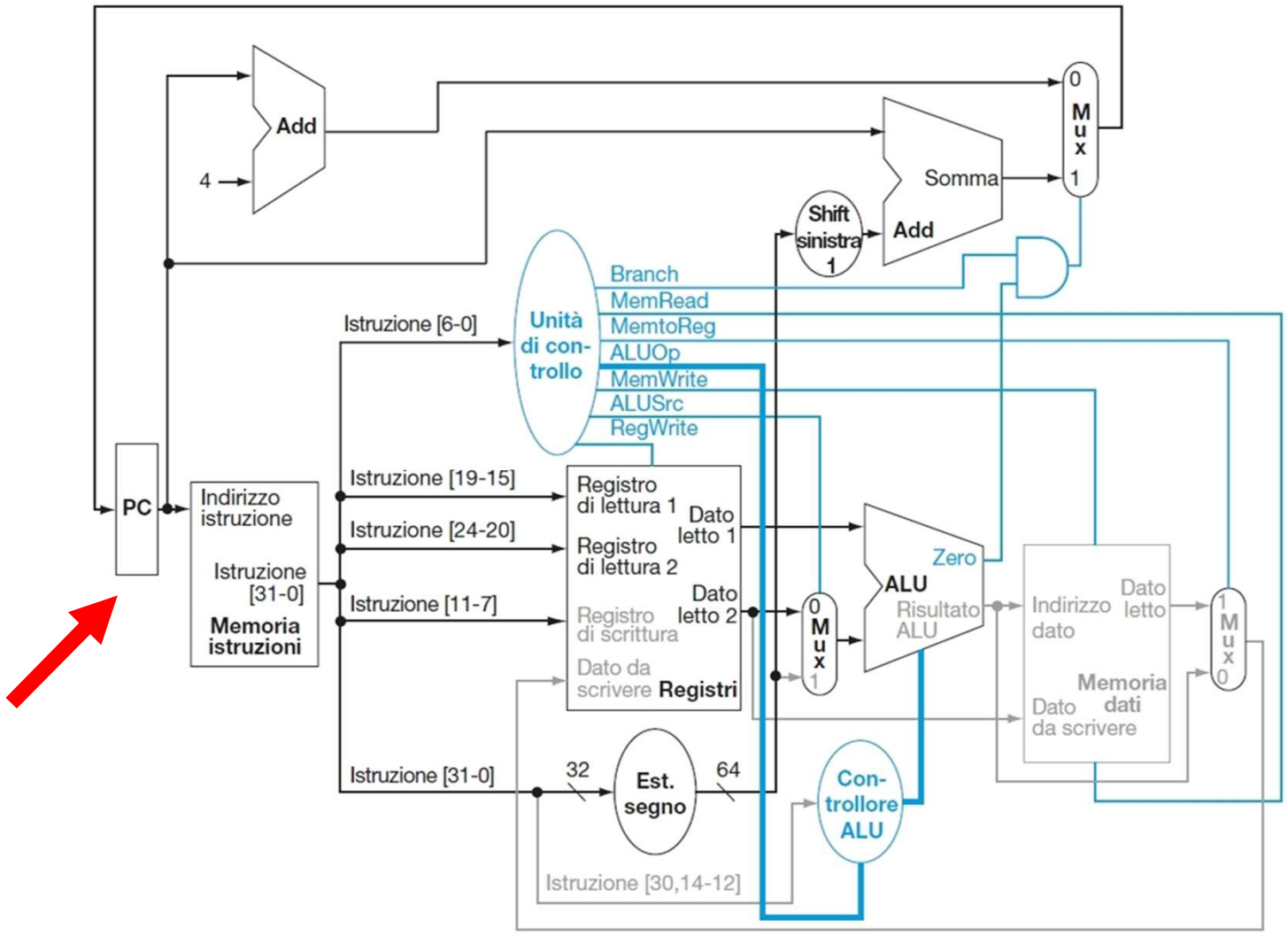
`ld x1,offset(x2)`

- Il dato proveniente dall'unità di memoria dati viene scritto nel register file nel registro x1



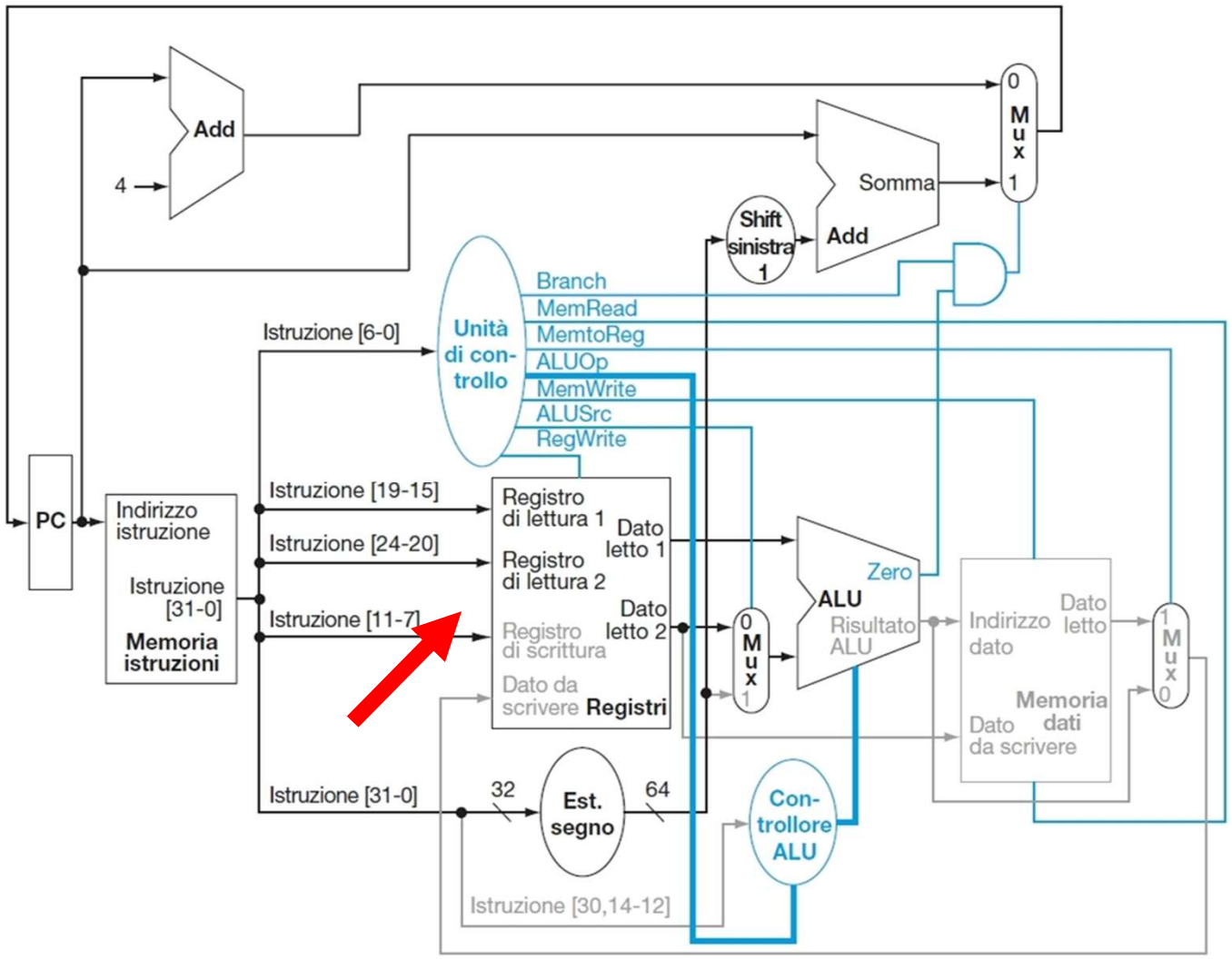
Esecuzione di Branch Equal – 1 `beq x1,x2,offset`

- L'istruzione viene prelevata dalla memoria istruzioni e si incrementa il PC



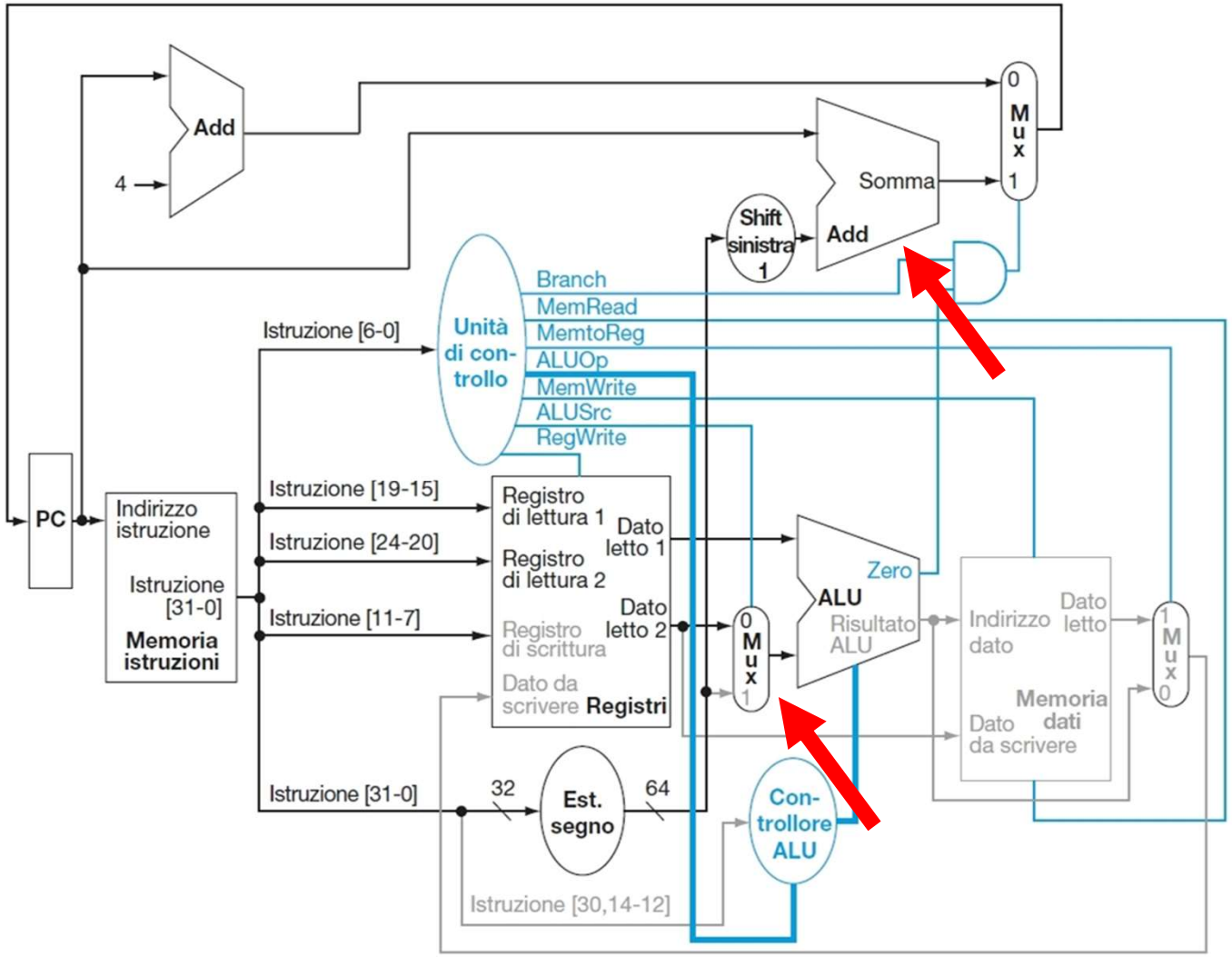
Esecuzione di Branch Equal – 2 `beq x1,x2,offset`

- Vengono letti dal register file i due registri x1 e x2



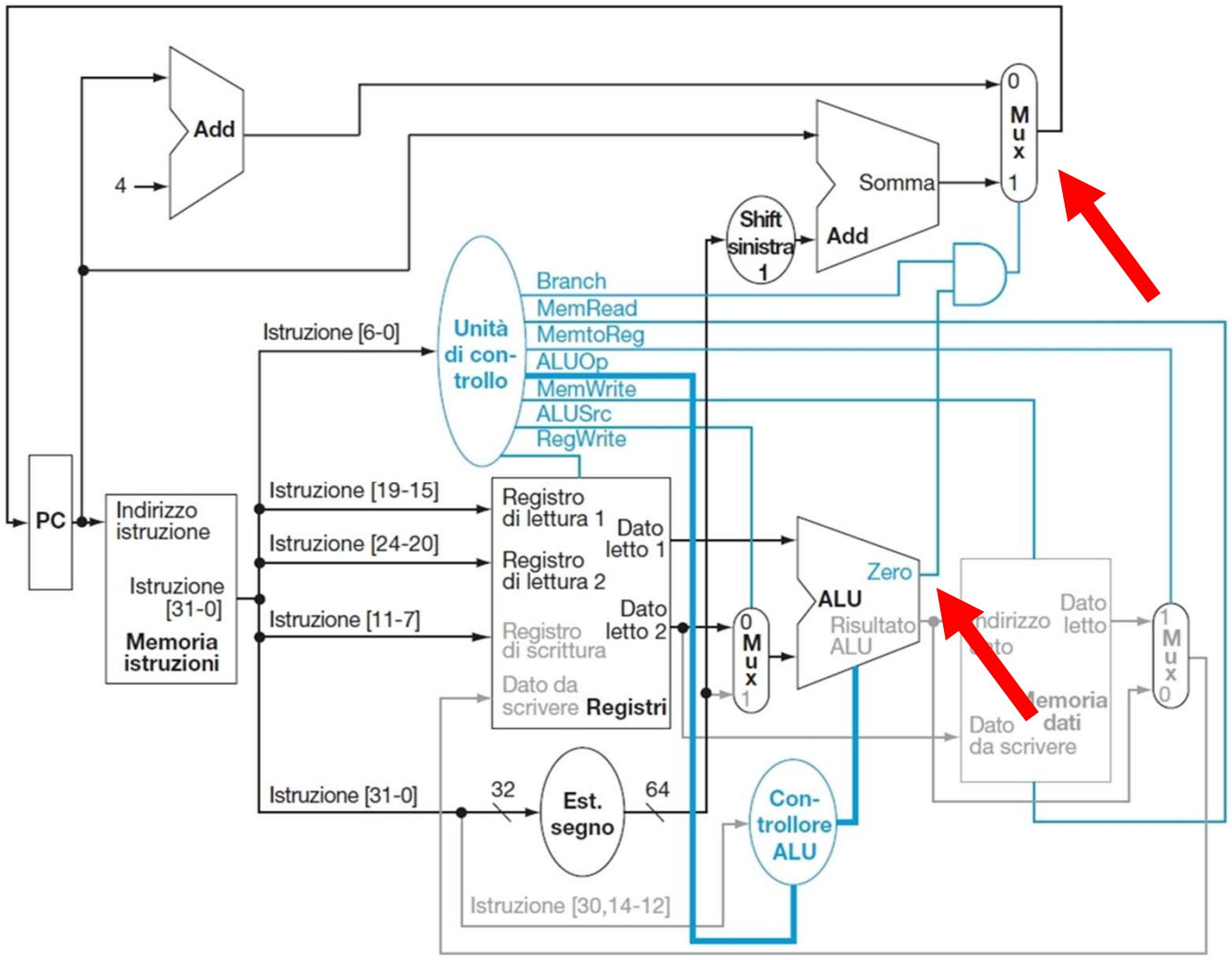
Esecuzione di Branch Equal – 3 `beq x1,x2,offset`

- La ALU esegue la sottrazione del contenuto dei due registri letti dal register file
- il valore del PC viene sommato ai 12 bit del campo offset dell'istruzione, dotati di segno, estesi a 64 bit e fatti scorrere di una posizione a sinistra
- il risultato costituisce l'indirizzo di destinazione del salto



Esecuzione di Branch Equal – 4 `beq x1,x2,offset`

- la linea **Zero** in uscita dalla ALU viene utilizzata per determinare da quale sommatore prendere l'indirizzo successivo da scrivere nel PC



Unità di Controllo

4 formati

Input o output	Nome del segnale	Formato R	ld	sd	beq
Input	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Output	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp01	1	0	0	0
	ALUOp02	0	0	0	1

Esercizio: La tabella specifica completamente la funzione di controllo. Come possiamo implementarla mediante porte logiche?