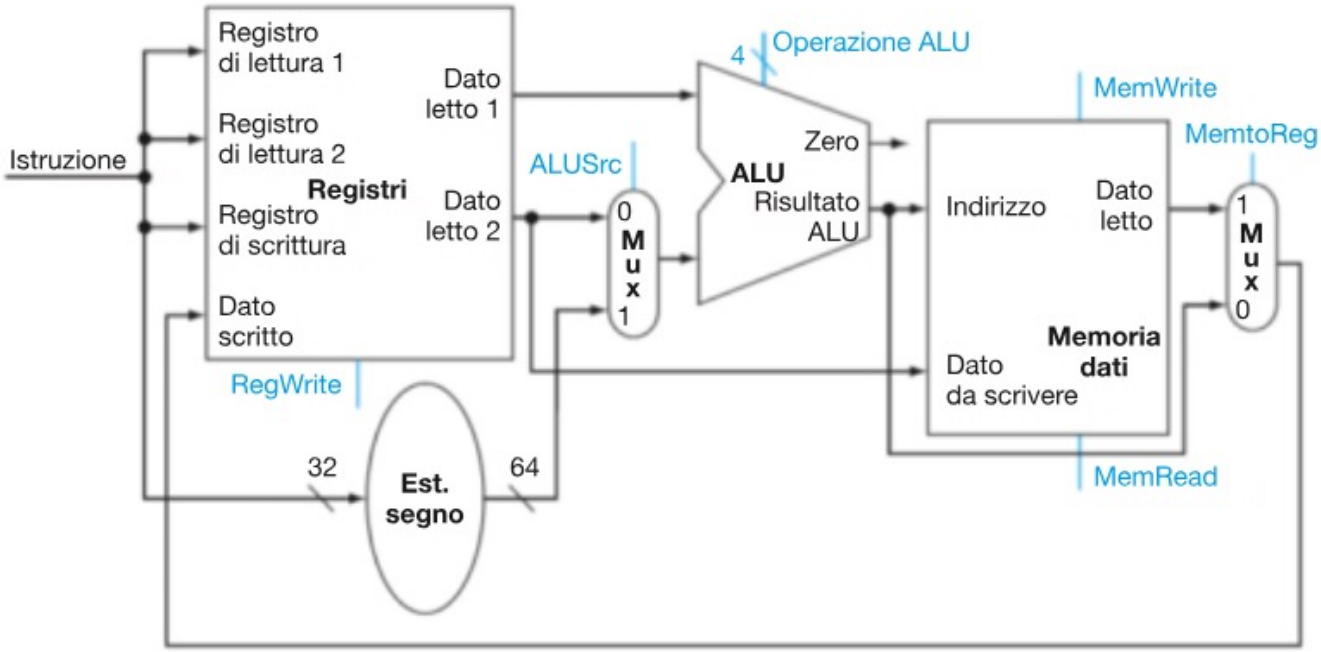
Progetto di un'unità di elaborazione

- Siccome abbiamo il requisito di eseguire ogni istruzione in un ciclo di clock, non possiamo usare un'unità funzionale più di una volta in ogni ciclo
  - Perciò dobbiamo distinguere memoria dati e memoria istruzioni
- Inoltre occorre condividere il più possibile le varie unità
  - Perciò occorreranno opportuni multiplexer per poter selezionare l'input corretto all'unità funzionale tra quelli possibili

Esempio

- Con questo circuito riusciamo a eseguire istruzioni di Tipo R e istruzioni di trasferimento da (Tipo I) e alla memoria (Tipo S)

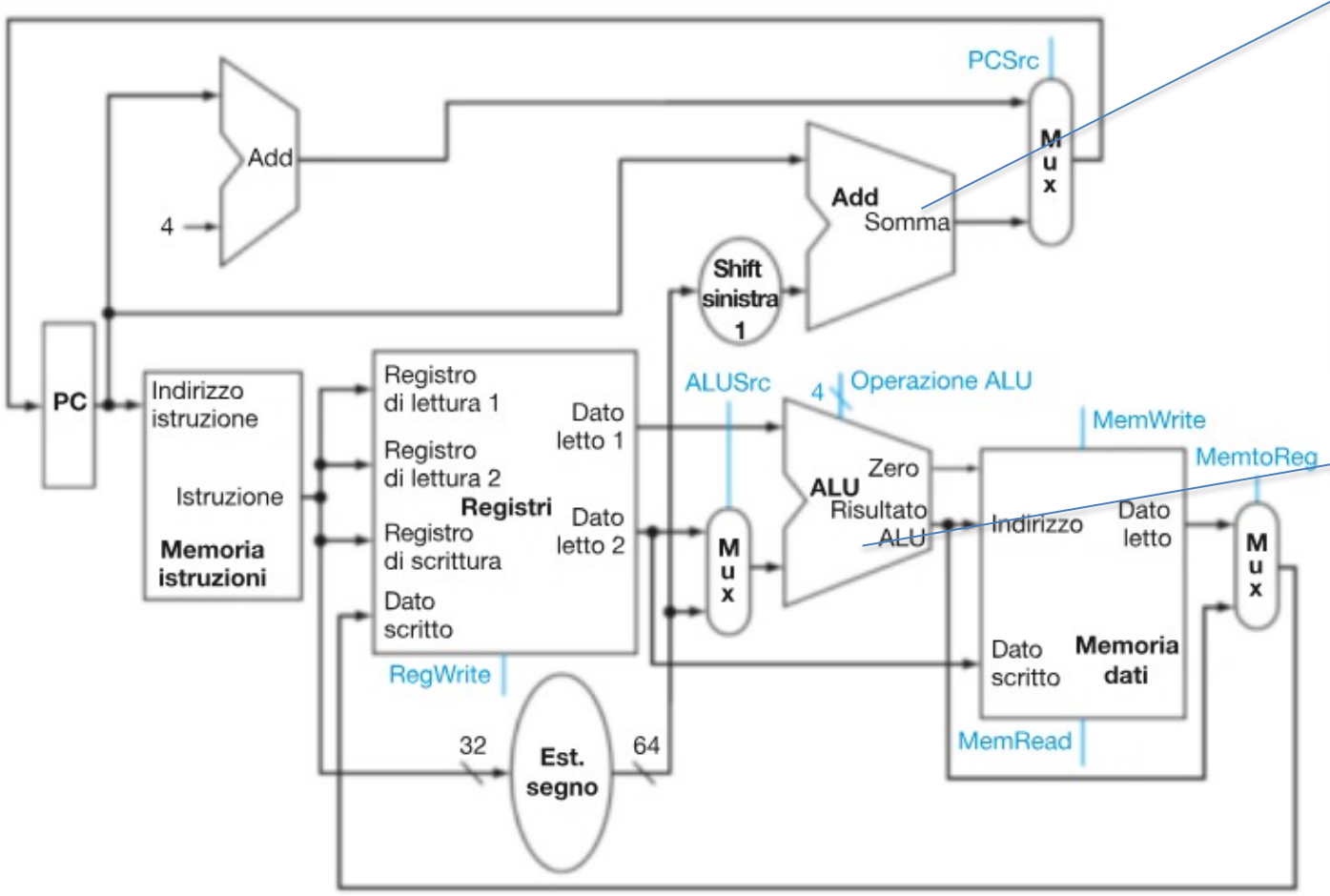


Per istruzione di tipo R:
- ALUSrc = 0
- MemtoReg = 0
- REGwrite = 1
- MemRead = 0
- MemWrite = 0

Per istruzione ld
- ALUSrc = 1
- MemtoReg = 1
- REGwrite = 1
- MemRead = 1
- MemWrite = 0

Per istruzione sd
- ALUSrc = 1
- MemtoReg = X
- REGwrite = 0
- MemRead = 0
- MemWrite = 1

Un esempio più completo



Circuiteria per i salti condizionati

Con questo DATAPATH possiamo fare istruzioni di tipo R, memorizzazioni, caricamento e salti condizionati

Prima implementazione completa

- Per arrivare a una prima implementazione completa partiamo dal datapath mostrato e aggiungiamo la parte di controllo
- Implementeremo le istruzioni
  - ✓add, sub, and, or
  - ✓ld, sd
  - ✓beq

Cominciamo dalla ALU

- La ALU viene impiegata per:
  - Effettuare operazioni logico-aritmetiche (tipo R), compreso slt
  - Calcolare indirizzi di memoria (per sd e ld)
  - Sottrazione per beq
- Per queste diverse operazioni abbiamo una diversa configurazione degli input di controllo (Linea controllo ALU)

Linea controllo ALU	Operazione
0000	AND
0001	OR
0010	Somma
0110	Sottrazione

Ancora sul controllo della ALU

- Per generare i bit di controllo della ALU useremo una piccola unità di controllo che riceve in ingresso
  - i campi funz7 e funz3 prelevati dall'istruzione
  - due bit detti «ALUop»
    - ✓ALUop = 00 -> somma (per istruzioni di sd e ld)
    - ✓ALUOp = 01 -> sottrazione (per beq)
    - ✓ALUOp = 10 -> operazione di tipo R (specificata da funz7 e funz3)

Ancora sul controllo della ALU

Tabella riassuntiva

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

Decodifica a livelli multipli

- Quello che abbiamo visto è un sistema di decodifica e generazione dei comandi a due livelli
  - **Livello 1**: (unità di controllo) genera i segnali di controllo ALUOp per l'unità di controllo della ALU
  - **Livello 2:** (unità di controllo della ALU) genera i segnali di controllo per la ALU

Unità di controllo dell'ALU

- I segnali di controllo della ALU sono generati da una rete logica combinatoria (unità di controllo della ALU)
- Bisognerebbe elencare tutte le combinazioni di ingresso di ALUop e dei campi funz7 e funz3 (12 bit in tutto)
- Per evitare di elencare tutte le combinazioni ($2^{12} = 4096$) useremo X come wildcard (un po' come * nel filesystem).

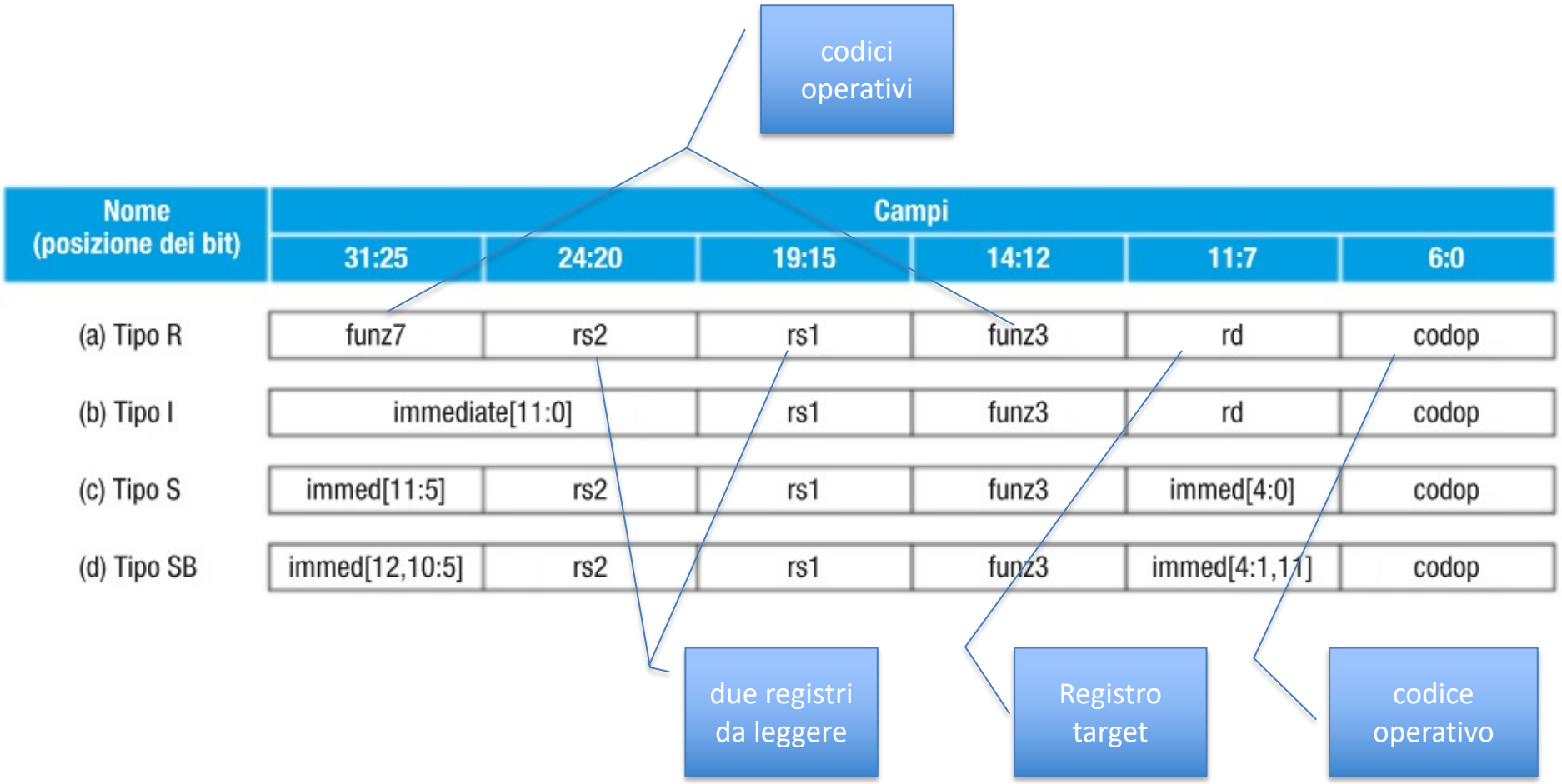
Unità di controllo dell'ALU

Tabella di verità (compressa):

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]	
0	0	X	X	X	X	X	X	X	X	X	X	0010 ld/sd
X	1	X	X	X	X	X	X	X	X	X	X	0110 beq
1	X	0	0	0	0	0	0	0	0	0	0	0010 add
1	X	0	1	0	0	0	0	0	0	0	0	0110 sub
1	X	0	0	0	0	0	0	0	1	1	1	0000 AND
1	X	0	0	0	0	0	0	0	1	1	0	0001 OR

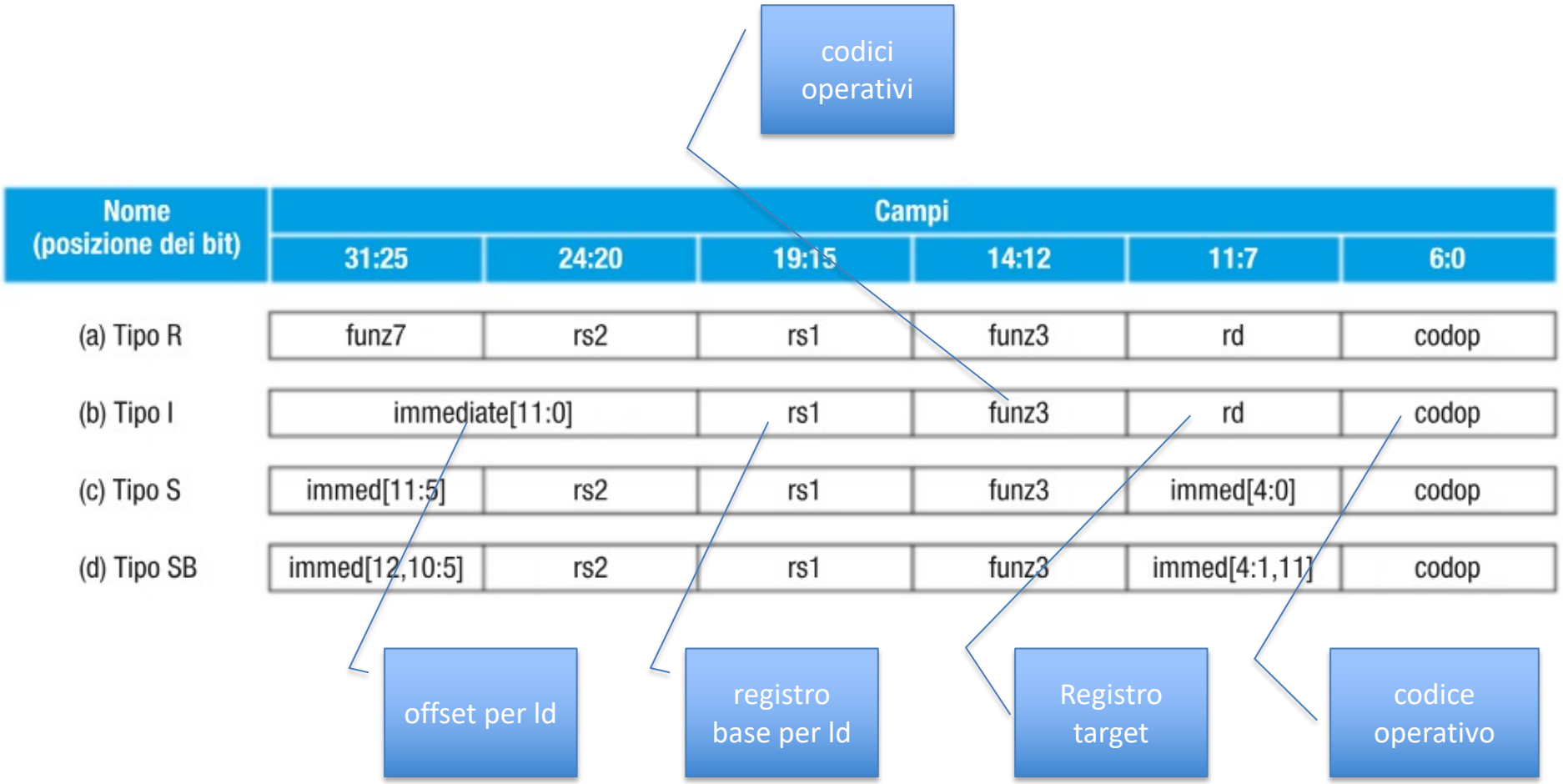
Unità di controllo principale

- Riguardiamo i campi.



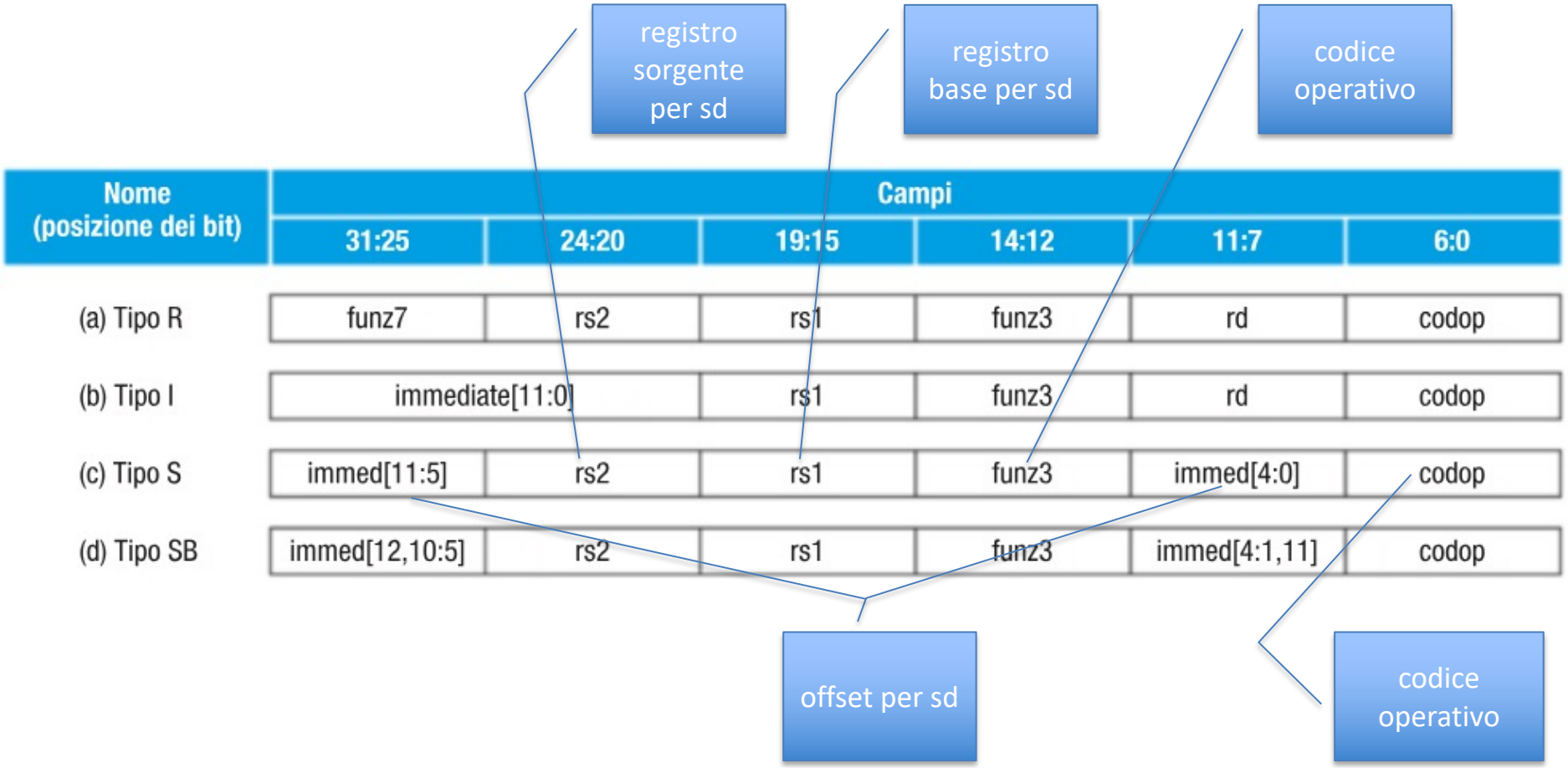
Unità di controllo principale

- Riguardiamo i campi.



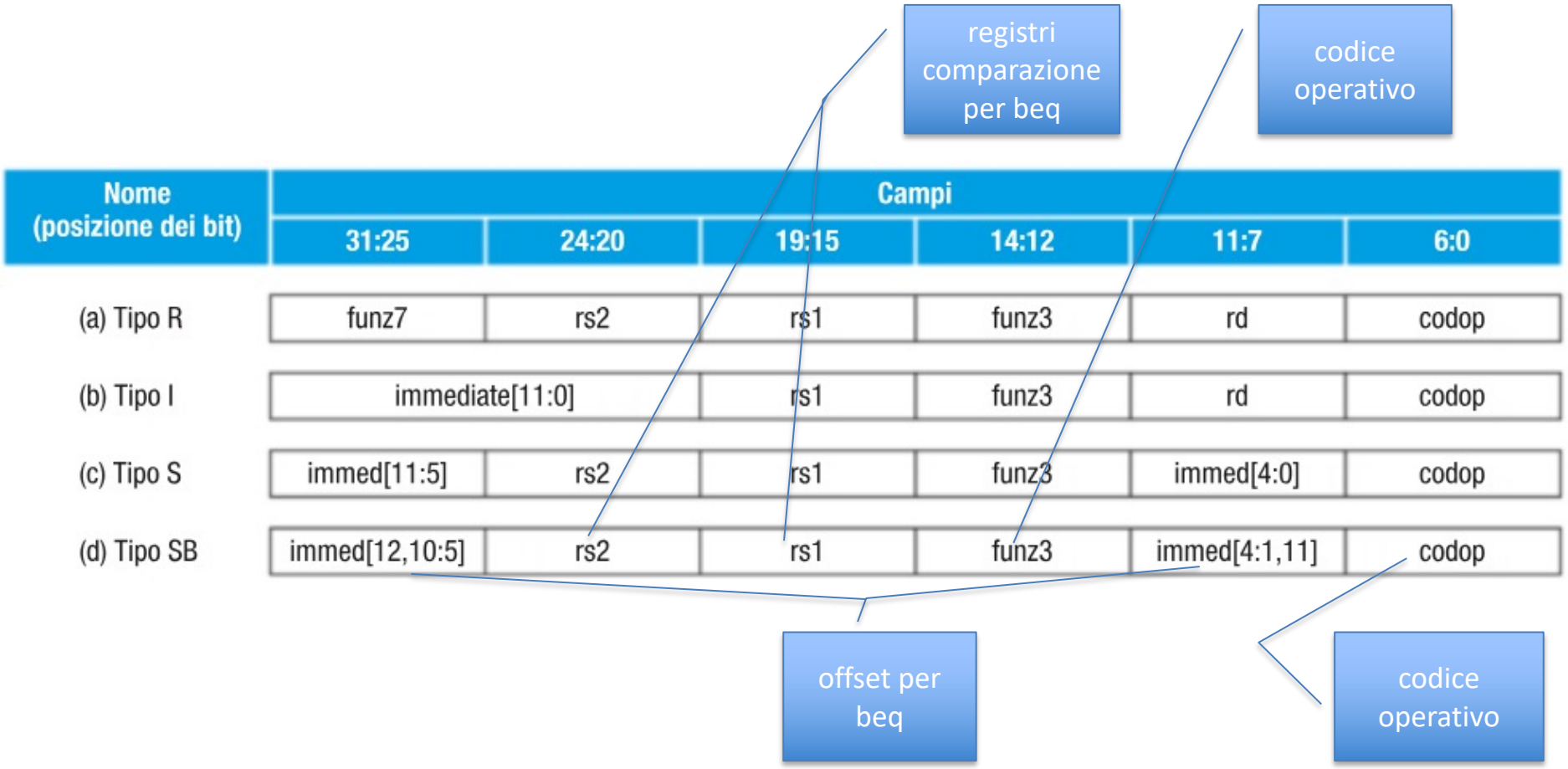
Unità di controllo principale

- Riguardiamo i campi.

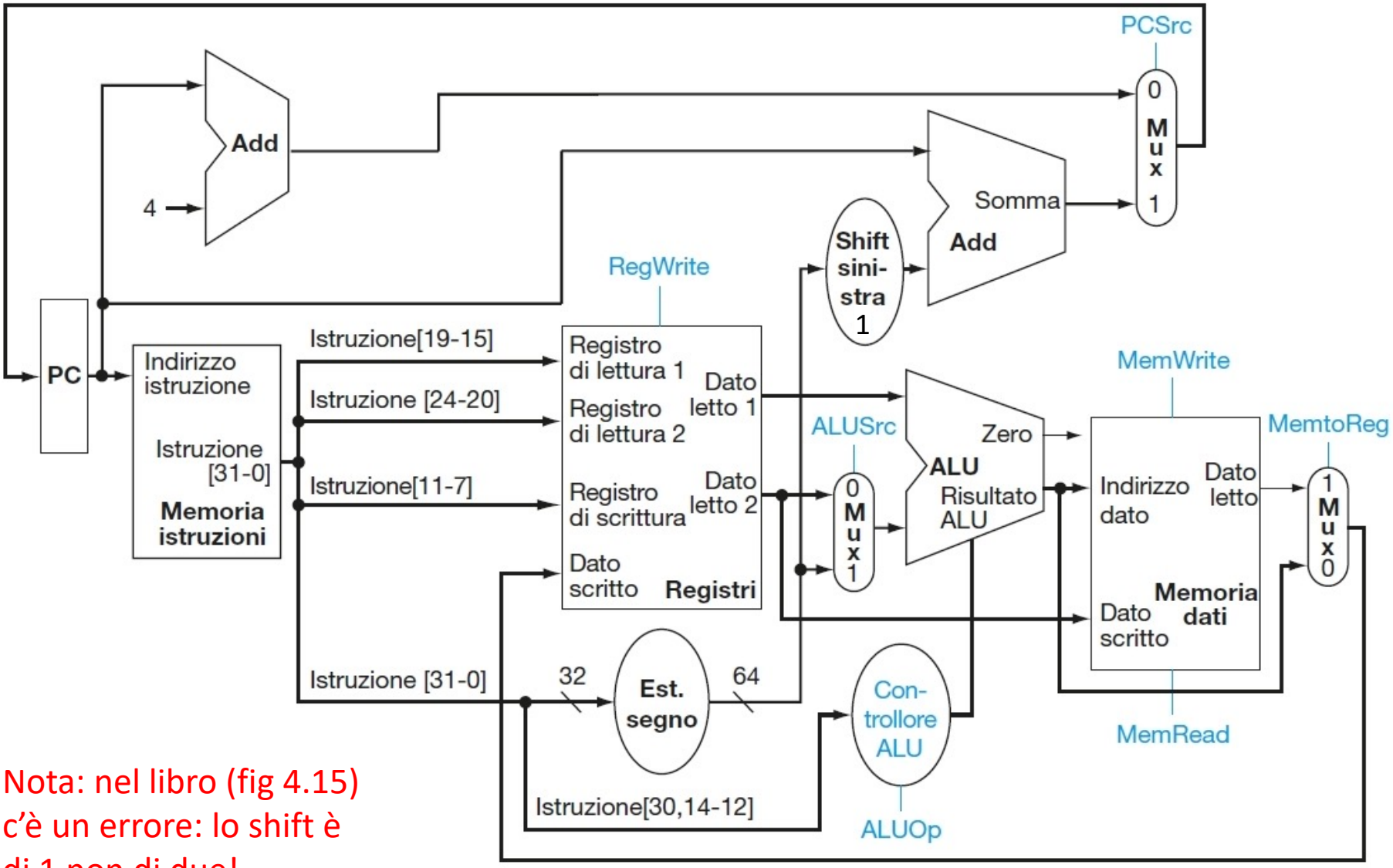


Unità di controllo principale

- Riguardiamo i campi.



Panoramica dei segnali di controllo

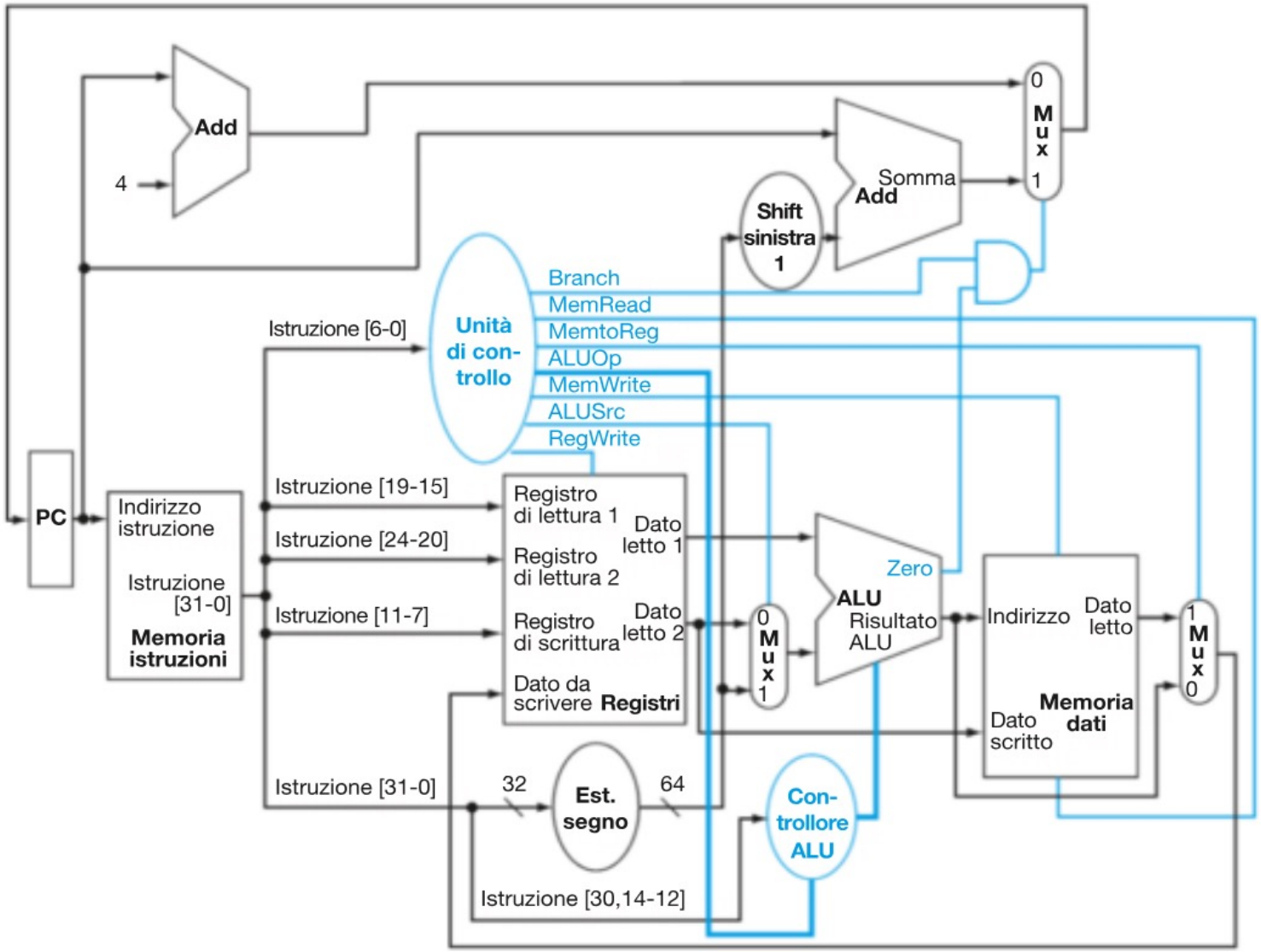


Nota: nel libro (fig 4.15) c'è un errore: lo shift è di 1 non di due!

Tabella riassuntivo dei segnali di controllo

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

Schema complessivo



L'unità di controllo

- L'unità di controllo genera tutti i segnali di controllo (incluso ALUOp)
- Ancora è una rete combinatoria che prende come input il codice operativo dell'istruzione e genera i comandi del caso, secondo la seguente tabella

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

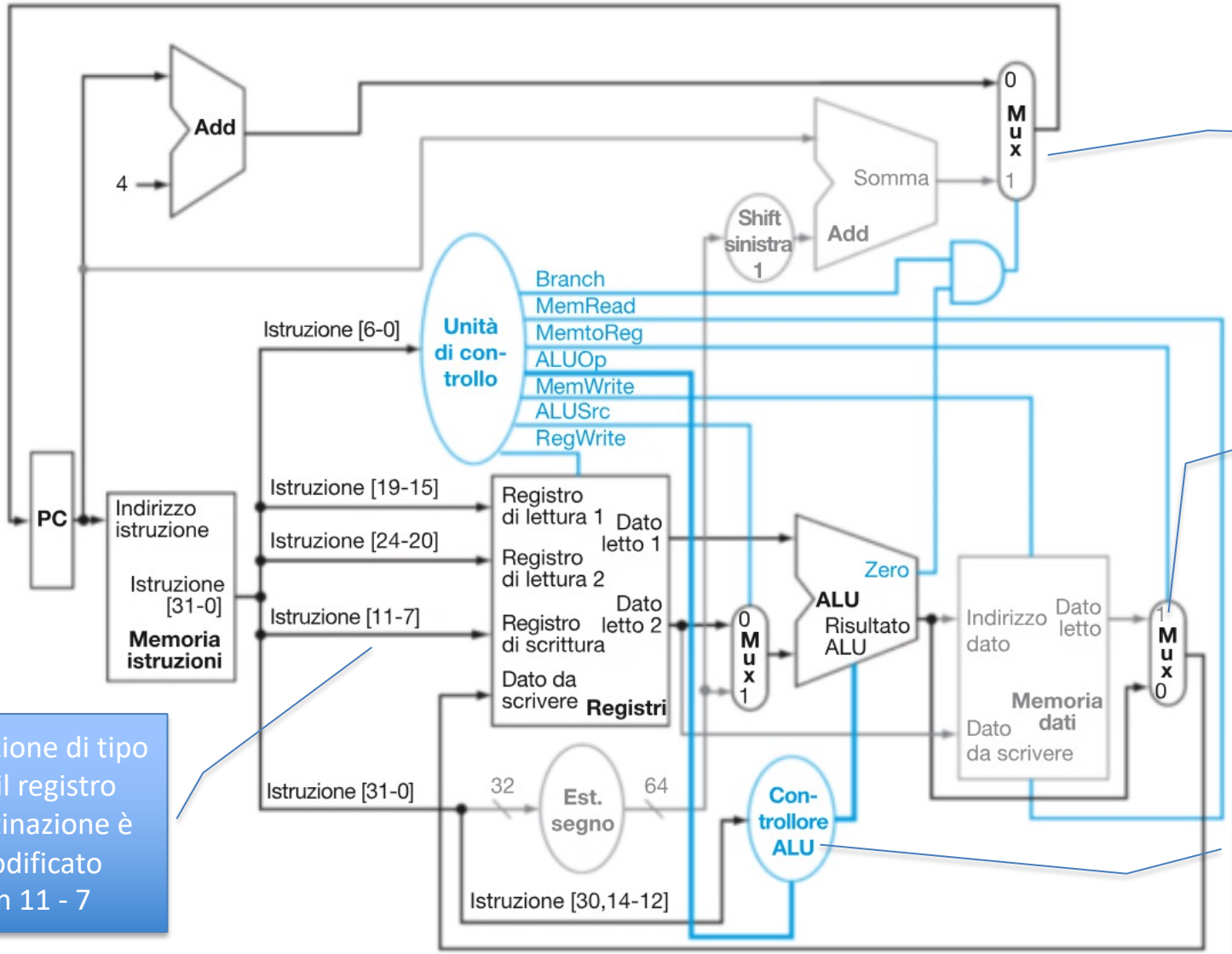
Partiamo dalla ADD

- Per eseguire un'istruzione di tipo R (ad esempio: *add x5, x6, x7*) occorre:

  1. prelevare l'istruzione dalla memoria e incrementare il PC di 4
  2. leggere x6 e x7 dal register file mentre l'unità di controllo principale calcola il valore da attribuire alle linee di controllo
  3. attivare la ALU con in input i dati dal register file usando alcuni bit del codice operativo per selezionare l'operazione della ALU (ad esempio: add)
  4. memorizzare il risultato nel registro destinazione (x5)

- Tutto questo avviene in un solo ciclo

Passi sul processore



Prelievo: nel Mux rimane attivo il ramo alto

In questo Mux è attiva la linea bassa (non uso la memoria per il risultato)

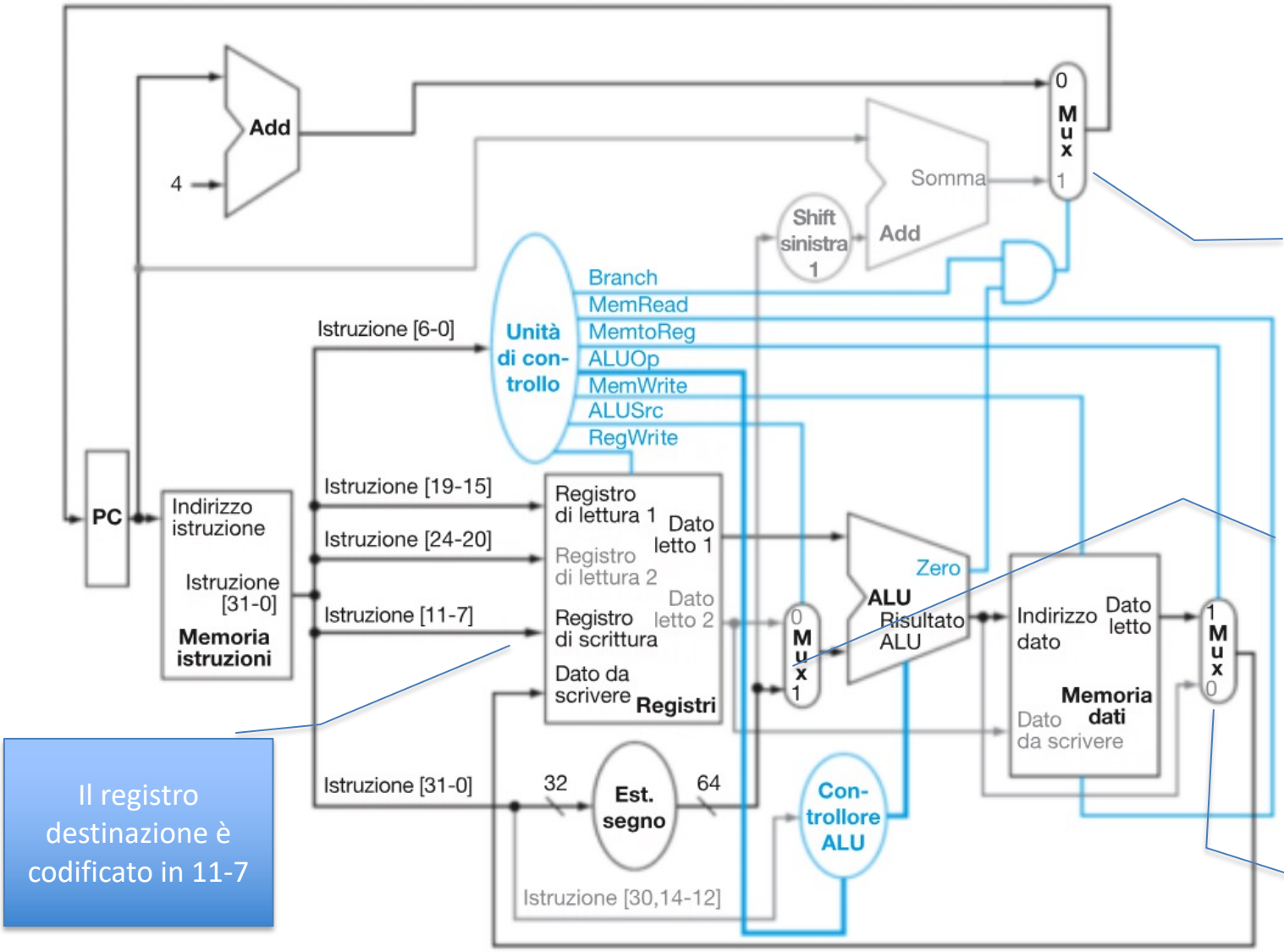
Istruzione di tipo R: il registro destinazione è codificato in 11 - 7

Istruzione di tipo R: il controllore della ALU usa un bit di funz7 e tutto funz3

Load

- Consideriamo ora l'istruzione
  ld x5, offset(x6)
- Fasi
  1. la fase di prelievo istruzione ed incremento del PC è uguale a prima
  2. prelevare x6 dal register file
  3. la ALU somma il valore letto dal register file i 12 bit del campo offset dell'istruzione, dotati di segno ed estesi a 64 bit.
  4. il risultato della somma viene usato come indirizzo per memoria dati
  5. il dato prelevato dall'unità di memoria dati viene scritto nel register file nel registro x5

Passi sul processore



Prelievo:
nel Mux rimane attivo il ramo alto

In questo Mux è attiva la linea 1 (secondo input dato da offset)

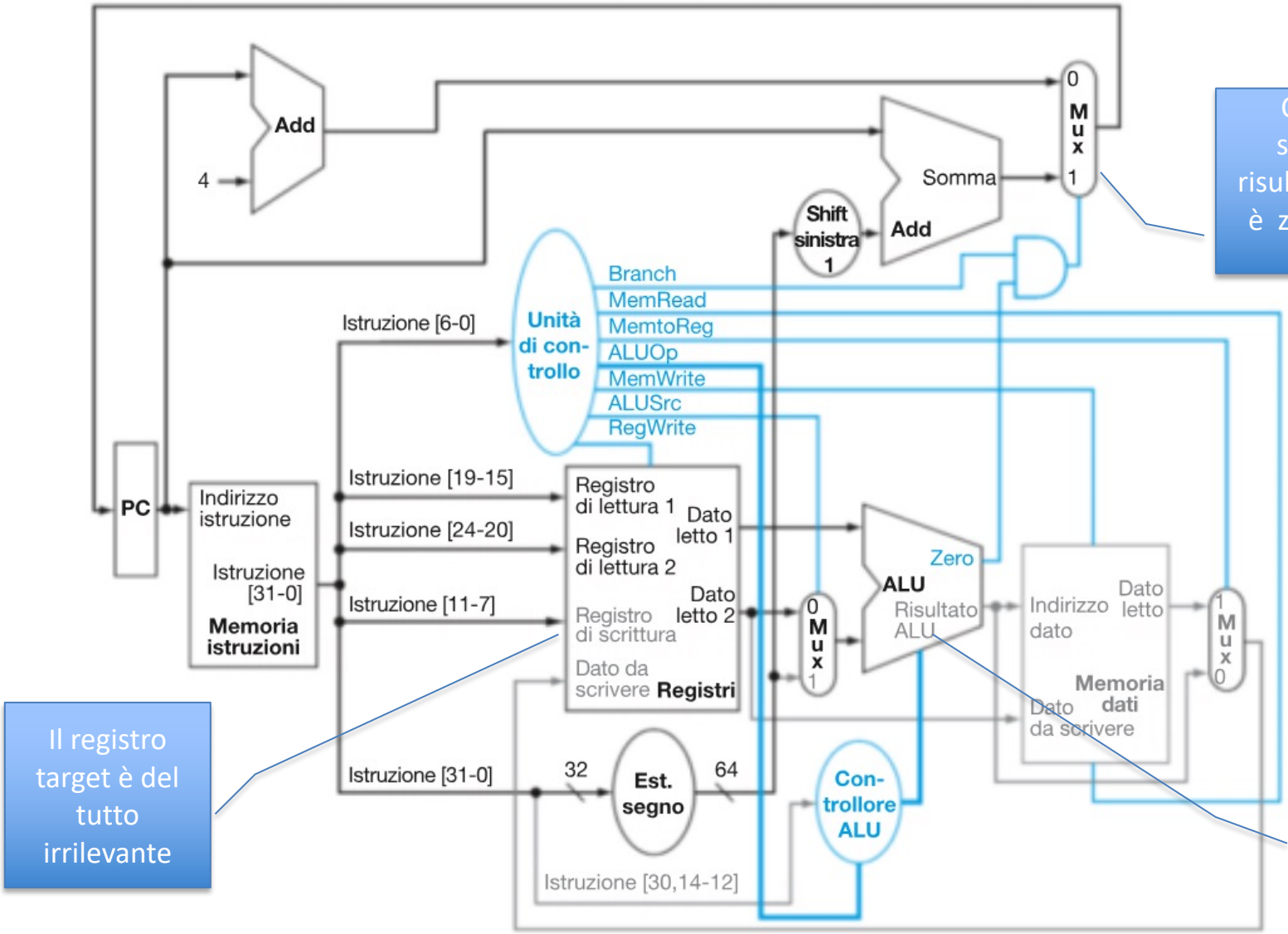
In questo Mux, stavolta, il dato che passa è quello sopra (provenienza memoria)

Il registro destinazione è codificato in 11-7

Salto condizionale

- Consideriamo ora l'istruzione
  beq x5, x6, offset
- Fasi
  1. la fase di prelievo istruzione ed incremento del PC è uguale a prima
  2. prelevare x5 e x6 dal register file
  3. la ALU sottrare x5 da x6. 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
  4. la linea Zero in uscita dalla ALU viene usata per decidere da quale sommatore prendere l'indirizzo successivo da scrivere nel PC.

Sul processore



Con branch settato, se il risultato della ALU è zero si attiva il ramo uno

Il registro target è del tutto irrilevante

La ALU sottrae I due registri

Implementazione

- Dopo aver capito a cosa servono i vari comandi possiamo passare all'implementazione secondo la tabella già vista in precedenza

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

- Le diverse colonne di questa tabella vanno a determinare il codice operativo (bit da 0 a 6 dell'istruzione, insieme agli altri codici operativi addizionali)

Implementazione

- A questo punto gli input e gli output sono quelli specificati nella seguente tabella:

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

Considerazioni conclusive

- Abbiamo visto come realizzare un semplice processore che esegue le istruzioni in un ciclo
- Questo non si fa più perché:
  - A dettare il clock sono le istruzioni più lente (accesso alla memoria)
  - Se si considerano istruzioni più complesse di quelle che abbiamo visto, le cose peggiorano ancora di più (esempio: istruzioni floating point)
  - Non si riesce a fare ottimizzazioni aggressive sulle operazioni fatte più di frequente