CALCOLATORI
Il processore

Giovanni Iacca
giovanni.iacca@unitn.it

Lezione basata su materiale preparato dai Prof. Luigi Palopoli e Marco Roveri

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

Obiettivi

- In questa serie di lezioni cercheremo di capire come è strutturato un processore
- Le informazioni che vedremo si integrano con quanto visto sulle reti logiche
- Facciamo riferimento ad un insieme di istruzioni RISC-V ridotto
  - ISTRUZIONI DI ACCESSO ALLA MEMORIA
    - ✓ **ld, sd**
  - ISTRUZIONI MATEMATICHE E LOGICHE
    - ✓ **add, sub, and, or**
  - ISTRUZIONI DI SALTO
    - ✓ **beq**
- Le altre istruzioni si implementano con tecniche simili

Panoramica generale

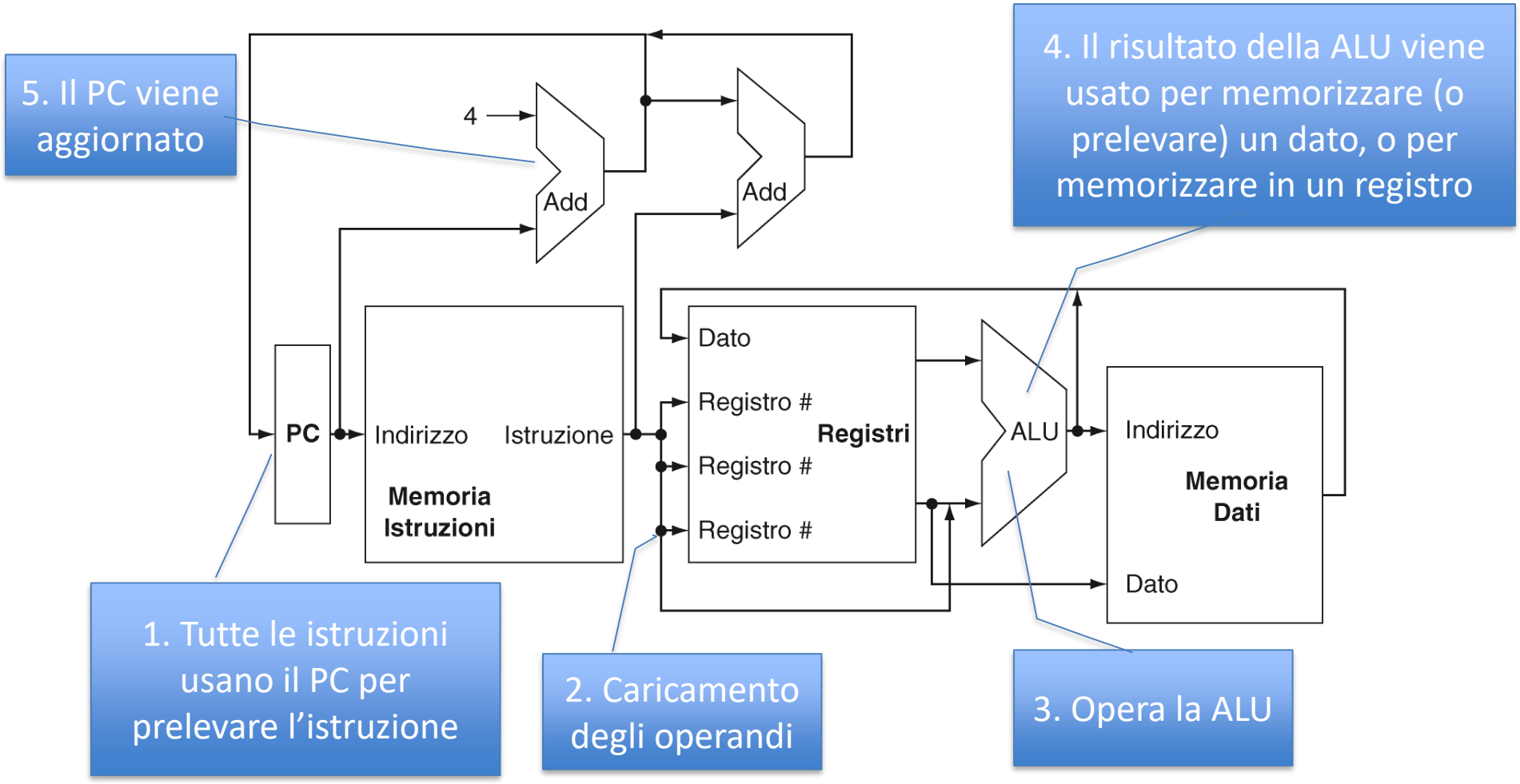
- Nell'esecuzione delle istruzioni, per come è progettata l'ISA, vi sono molti tratti comuni
- Le prime due fasi, *per ogni istruzione,* sono
  - Prelievo dell'istruzione dalla memoria
  - Lettura del valore di uno o più registri operandi che vengono estratti direttamente dai campi dell'istruzione
- I passi successivi dipendono dalla specifica istruzione ma, fortunatamente, sono molto simili per ciascuna delle tre classi individuate

Passi per ciascuna classe

- Tutti i tipi di istruzioni considerate usano la ALU (unità logico aritmetica) dopo aver letto gli operandi
  - Le istruzioni di accesso alla memoria per calcolare l'indirizzo
  - Le istruzioni aritmetico/logiche per eseguire quanto previsto dall'istruzione
  - I salti condizionati per effettuare il confronto
- Dopo l'uso della ALU il comportamento differisce per le tre classi
  - Le istruzioni di accesso alla memoria richiedono o salvano il dato in memoria
  - Le istruzioni aritmetiche/logiche memorizzano il risultato nel registro target
  - Le istruzioni di salto condizionato cambiano il valore del registro PC secondo l'esito del confronto

Schema di base

- Di seguito illustriamo la struttura di base della parte operativa (o datapath) per le varie istruzioni



Pezzi mancanti

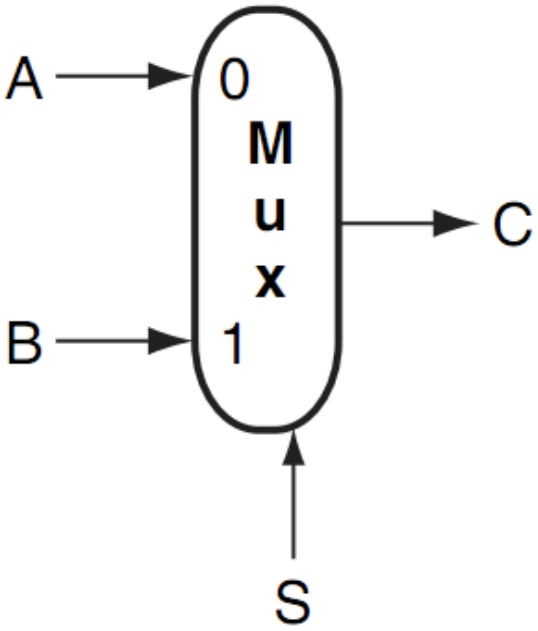
- La figura precedente è incompleta e crea l'impressione che ci sia un flusso continuo di dati
- In realtà ci sono punti in cui i dati arrivano da diverse sorgenti e bisogna sceglierne una (punto di decisione)
- E' il caso per esempio dell'incremento del PC
  - Nel caso "normale" il suo valore proviene da un circuito addizionatore (che lo fa puntare alla word successiva a quella appena letta)
  - Nel caso di salto il nuovo indirizzo viene calcolato a partire dall'offset contenuto nel campo dell'istruzione

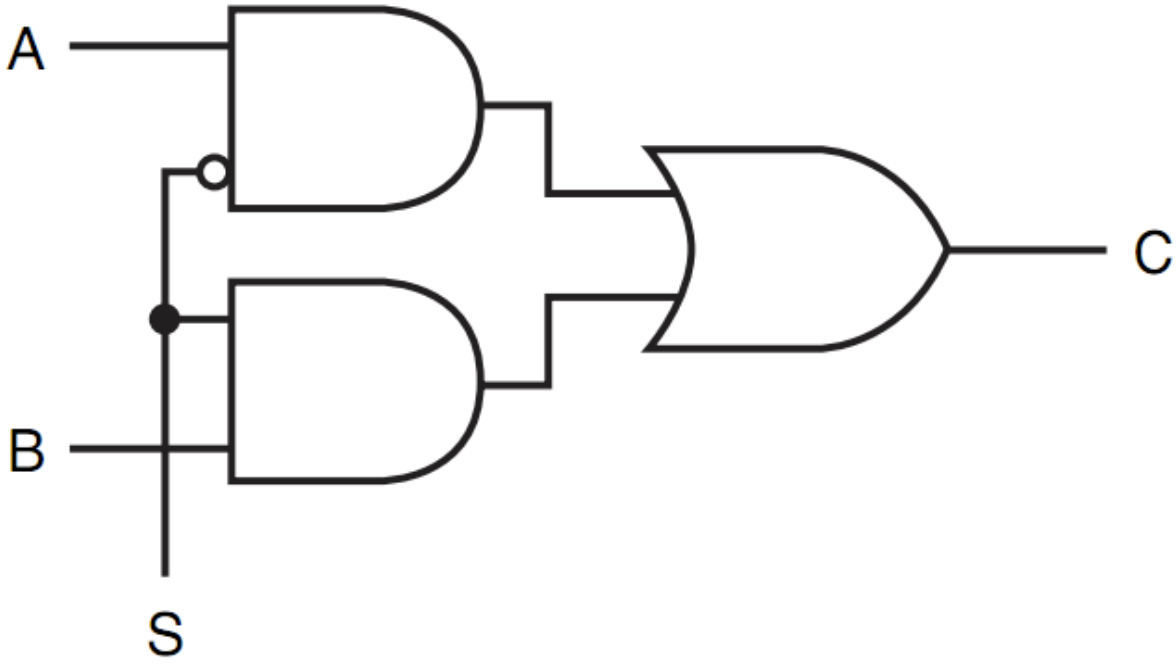
Pezzi mancanti

- Altro esempio
  - Il secondo operando della ALU può provenire dal banco registri (per istruzioni di tipo **R**) o dal codice dell'istruzione stessa (per istruzioni di tipo **I**)
- Per selezionare quale delle due opzioni scegliere viene impiegato una particolare rete combinatoria (multiplexer) che funge da selettore dei dati

Il multiplexer

- Come abbiamo già visto, il multiplexer ha due (o più) ingressi dati, e un ingresso di controllo
- Sulla base dell'ingresso di controllo si decide quale degli input debba finire in output





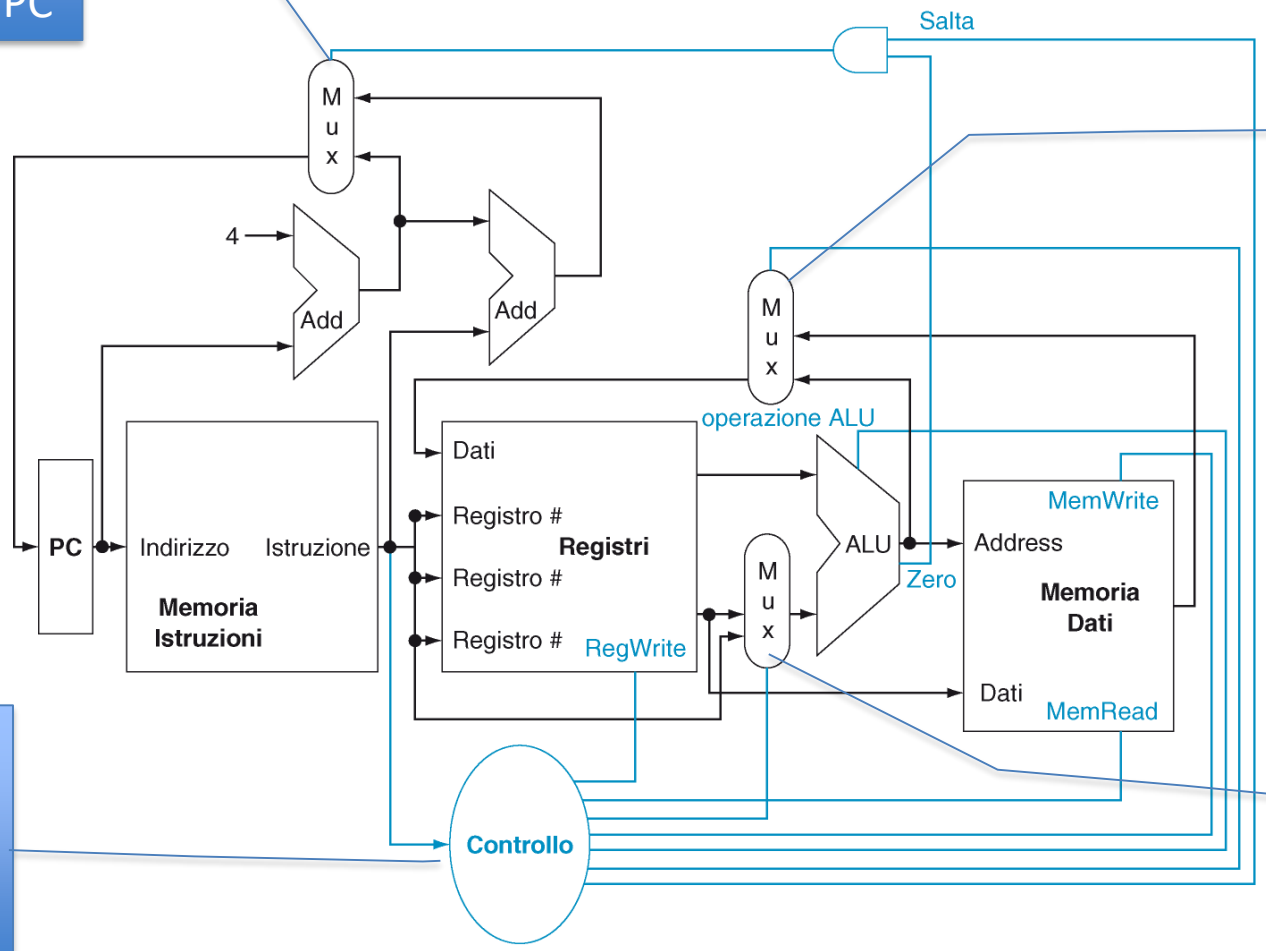
Ulteriori pezzi mancanti

- Le linee di controllo dei multiplexer vengono impostate sulla base del tipo di istruzione
- I vari blocchi funzionali hanno ulteriori ingressi di controllo
  - La ALU ha diversi ingressi per decidere quale operazione effettuare
  - Il banco registri ha degli ingressi per decidere se scrivere o meno in un registro
  - La memoria dati ha degli ingressi per decidere se vogliamo effettuare letture o scritture
- Per decidere come impiegare i vari ingressi di controllo abbiamo bisogno di un'unità che funga da "direttore d'orchestra"

Una figura più completa

Mux per decidere come aggiornare il PC

Nel registro target memorizziamo il risultato della ALU o quello che preleviamo dalla memoria?



Unità che genera i vari segnali di controllo

Secondo operando registro o immediato?

Informazioni di base

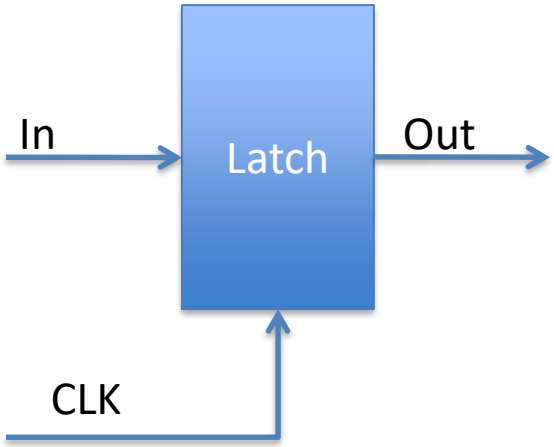
- ASSUNZIONE SEMPLIFICATIVA:
  - Il processore lavora sincronizzandosi con i cicli di clock
  - Per il momento facciamo l'assunzione semplificativa che tutte le istruzioni si svolgano in un singolo ciclo di clock (lungo abbastanza)
- Prima di entrare nella descrizione dei vari componenti ricordiamo velocemente alcuni concetti di reti logiche

Reti logiche

- Definiamo *rete logica combinatoria* un circuito composto di porte logiche che produce un output che è una funzione (statica) dell'input
  - Esempio: il multiplexer visto prima
- Vi sono inoltre elementi che chiamiamo «di stato»:
  - In sostanza se a un certo punto salviamo il valore degli elementi di stato e poi lo ricarichiamo, il computer riparte esattamente da dove si era interrotto
  - Nel nostro caso elementi di stato sono: registri, memoria dati, memoria istruzioni
- Gli elementi di stato sono detti *sequenziali* perché l'uscita a un ingresso dipende dalla storia (sequenza) degli ingressi precedenti
- Gli elementi di stato hanno (almeno) due ingressi:
  - Il valore da immettere nello stato
  - Un clock con cui sincronizzare le transizioni di stato

Flip-Flop

- Come abbiamo già visto, l'elemento base per memorizzare un bit è un circuito sequenziale chiamato *flip-flop D-latch*

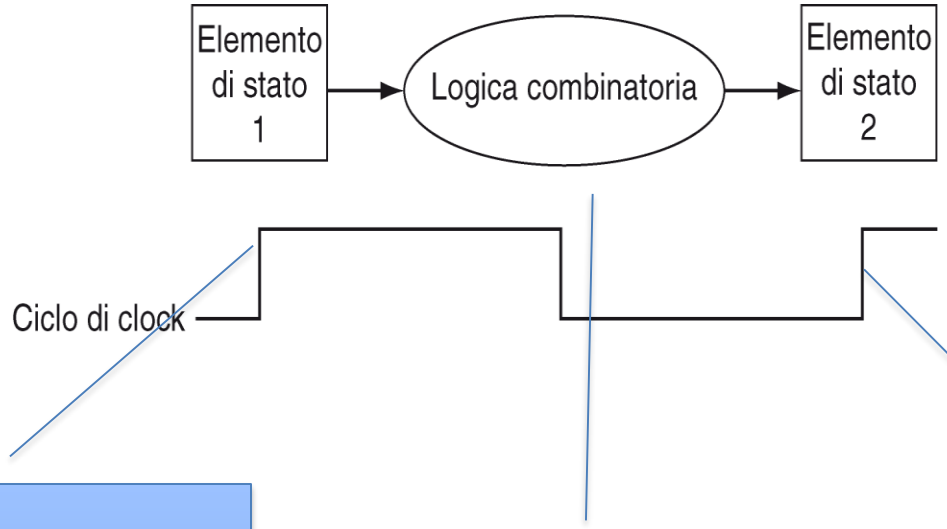


- I registri possono essere ottenuti come array di latch (o in altri modi simili)
- Terminologia
  - Asserito: segnale logico a livello alto
  - Non Asserito: segnale logico a livello basso

Temporizzazione

- La metodologia di temporizzazione ci dice quando i segnali possono essere letti o scritti in relazione al clock
- E' importante stabilire una temporizzazione
  - Se leggo e scrivo su registro, devo sapere se il dato che leggo è quello precedente o successivo alla scrittura
- La tecnica di temporizzazione più usata è quella sensibile ai fronti
  - Il dato viene memorizzato in corrispondenza della salita o della discesa del fronte di clock
- I dati presi dagli elementi di stato sono relativi al ciclo precedente

Esempio



Al tempo t l'elemento di stato 1 viene aggiornato

Il valore viene propagato attraverso una rete combinatoria

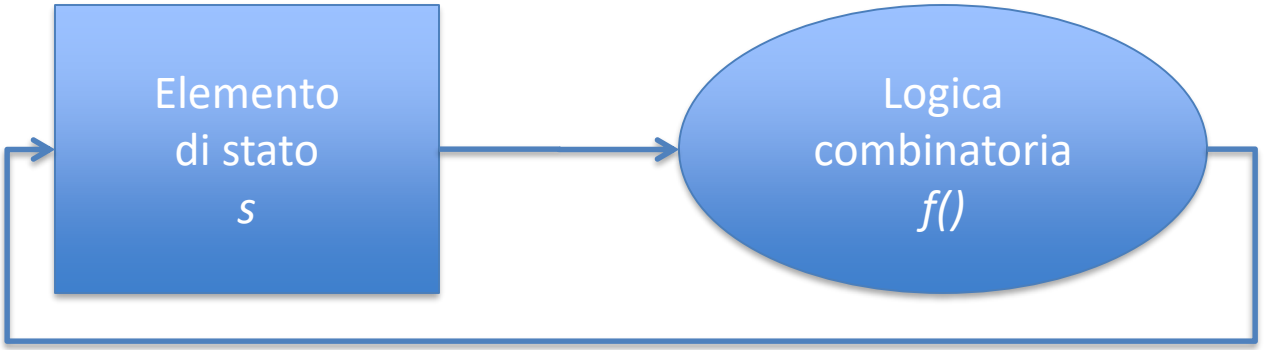
Al tempo t+T il valore arriva all'elemento di stato 2

Considerazioni

- Il tempo di clock T deve essere scelto in modo da dare tempo ai dati di attraversare la rete combinatoria
- Nel caso del RISC-V a 64 bit quasi tutti gli elementi di stato e combinatori hanno ingressi ed uscite a 64 bit
- La metodologia di memorizzazione sensibile ai clock permette di realizzare interconnessioni che, a prima vista, creerebbero dei cicli di retroazione che renderebbero impredicibile l'evoluzione del sistema

Esempio

- Consideriamo un caso come questo:



- Se non avessimo temporizzazioni precise dovremmo scrivere (qualcosa di indecidibile):

$$s = f(s)$$

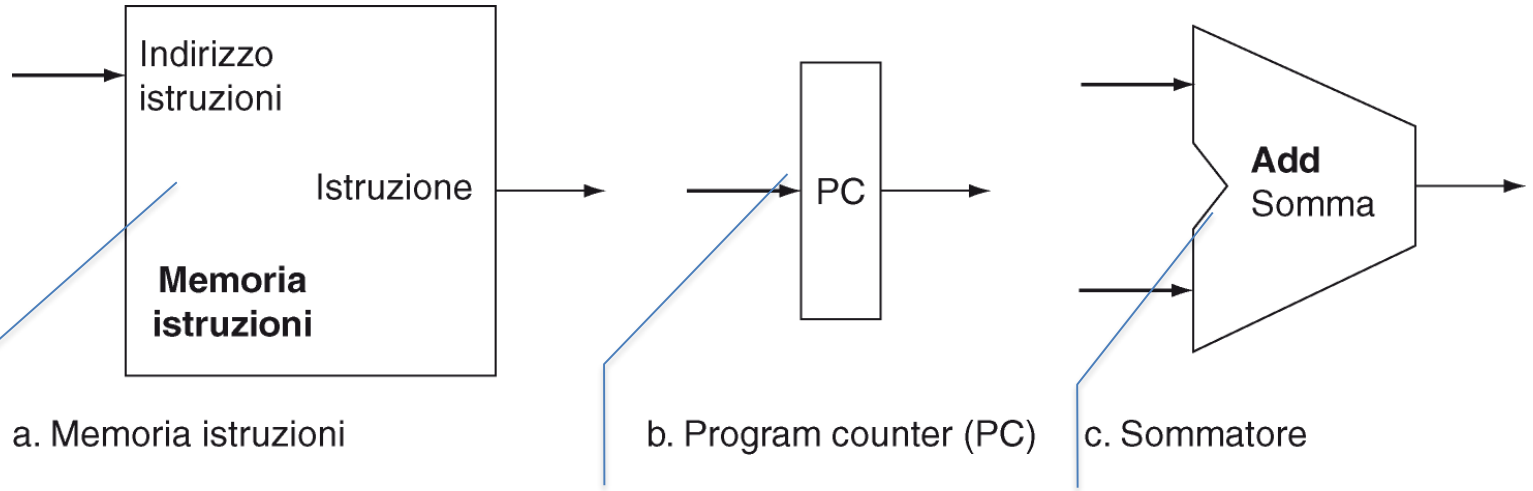
- Grazie alla temporizzazione sensibile al clock abbiamo:

$$s(t+T) = f(s(t))$$

che invece produce un'evoluzione ben determinata

Realizzazione del datapath

- Passiamo ora in rassegna i vari componenti che ci servono per la realizzazione del datapath



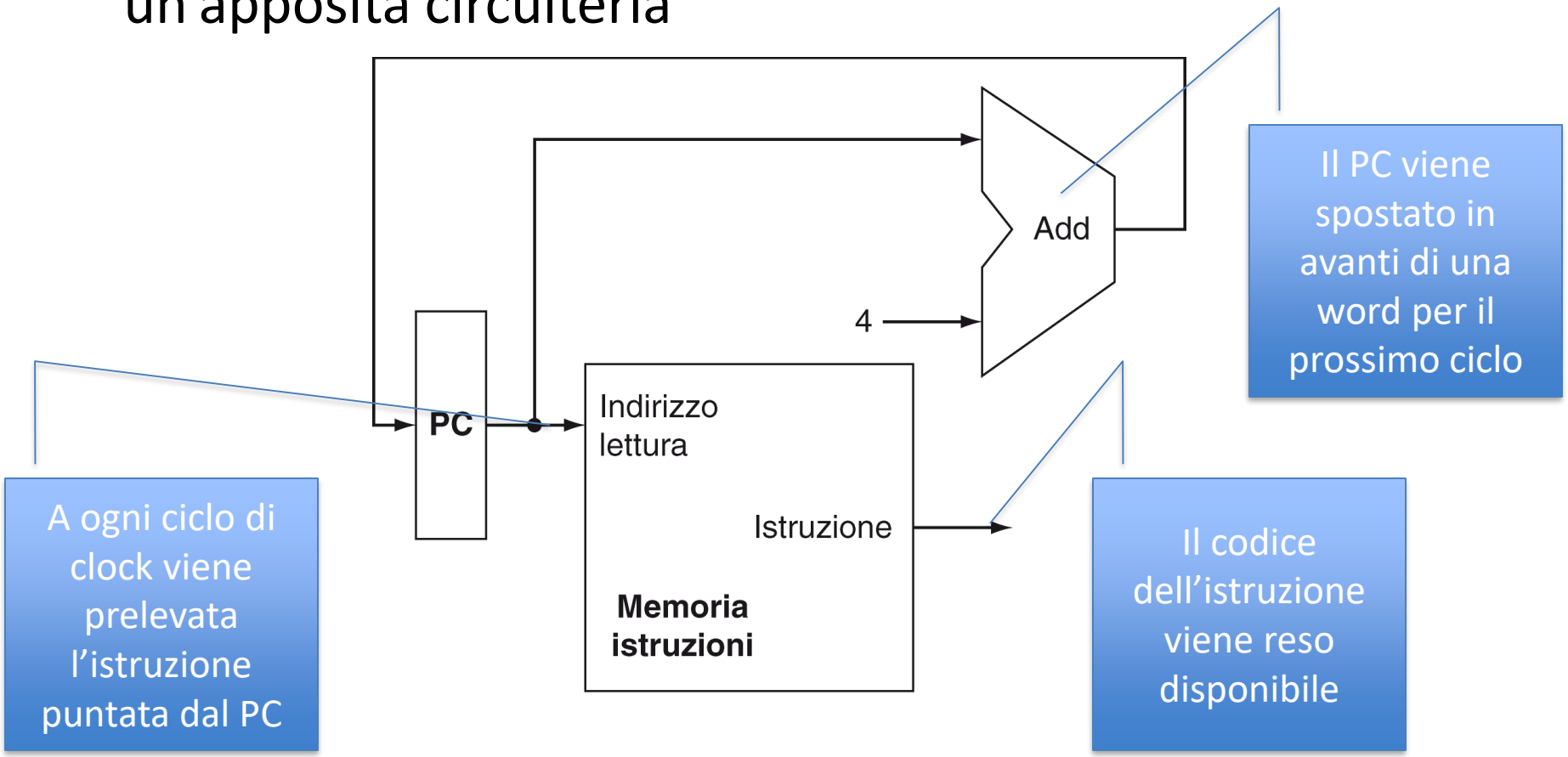
Memoria dove sono memorizzate le istruzioni

Registro che contiene l'indirizzo dell'istruzione da caricare

Addizionatore (ALU specializzata) per incrementare il PC

Prelievo dell'istruzione

- Usando gli elementi che abbiamo visto possiamo mostrare come effettuare il prelievo dell'istruzione con un'apposita circuiteria

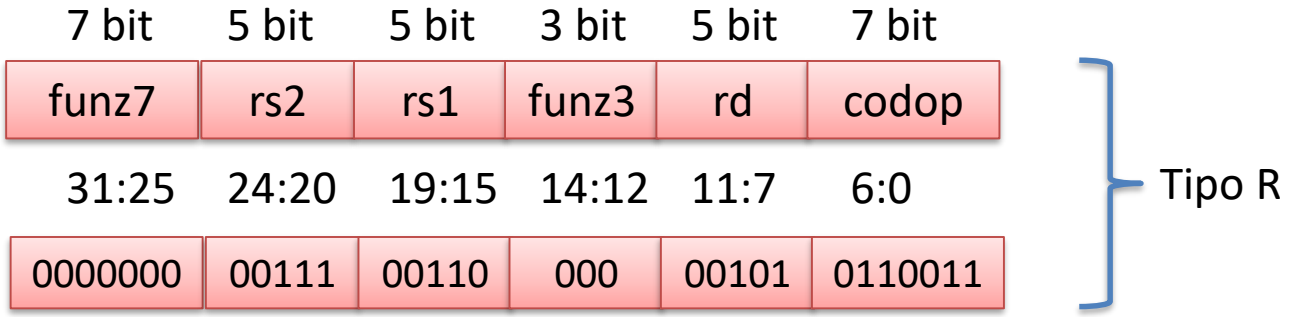


Istruzioni di tipo R

- Cominciamo dal vedere come vengono eseguite le istruzioni di tipo R
- Abbiamo visto che si tratta di istruzioni aritmetiche/logiche che operano tra registri e producono un risultato che viene memorizzato in un registro
- Esempio:

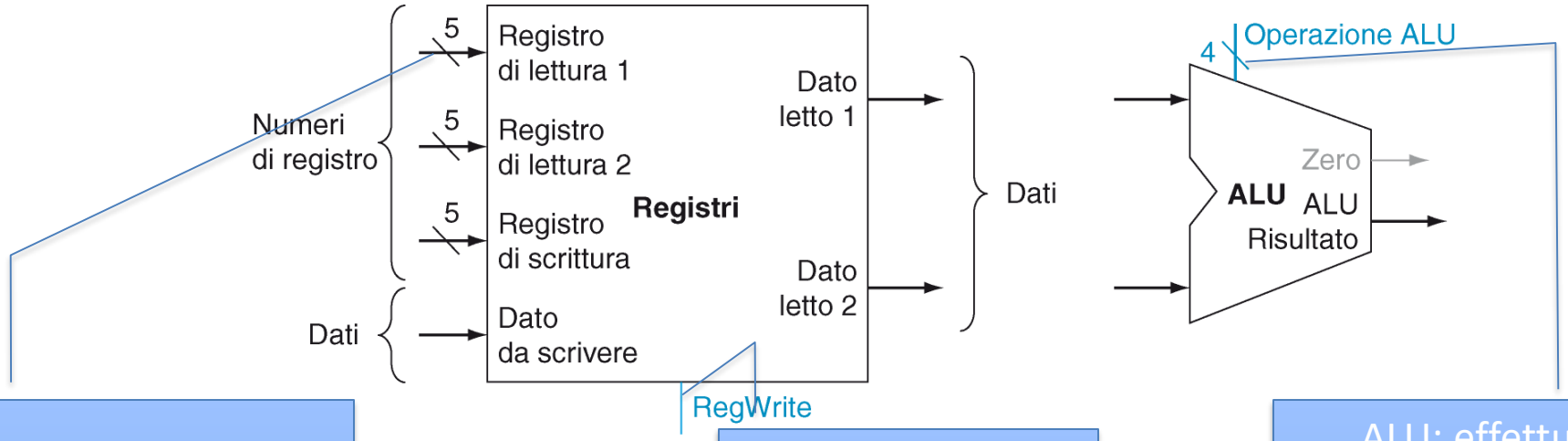
```
add x5, x6, x7
```

- Come noto, il codice binario corrispondente ha la forma:



Blocchi funzionali richiesti

- Per effettuare questi calcoli ho bisogno di due ulteriori blocchi funzionali.



tri

b. ALU

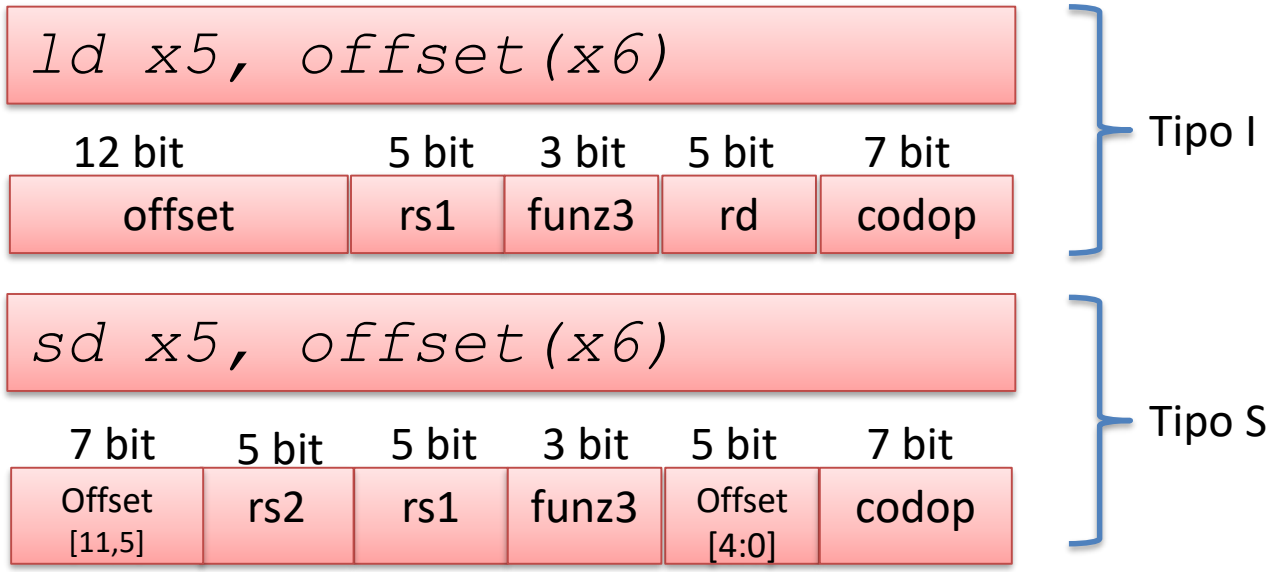
Banco registri: dà in output i registri specificati nel registro di lettura 1 e 2

Se abilitato in scrittura scrive nel registro specificato il dato in ingresso

ALU: effettua l'operazione aritmetica codificata in 4 bit. Setta un bit in uscita se il risultato è zero

Istruzioni load/store

- Consideriamo ora anche le istruzioni load (ld) e store (sd)
- Forma generale

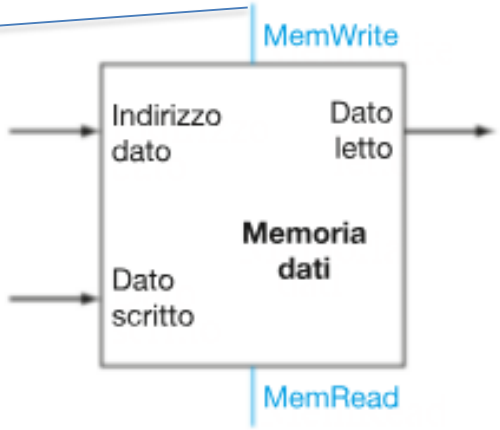


- Per entrambe si deve calcolare un indirizzo di memoria dato dalla somma di x6 con l'offset
- Per entrambe occorre leggere dal register file
- Quindi per eseguire queste istruzioni ci servono ancora la ALU e il register file

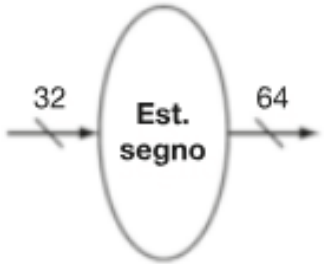
Istruzioni load/store

- Notare che l'offset viene memorizzato in un campo a 12 bit che occorrerà estendere a 64 bit (replicando per 42 volte il bit di segno)
- In aggiunta alle componenti che abbiamo visto prima occorre un'unità di memoria dati dove memorizzare eventualmente con *sd* (o da cui leggere con *ld*)

A differenza della memoria istruzioni questa può essere usata in lettura e scrittura. Quindi ho bisogno di comandi appositi.



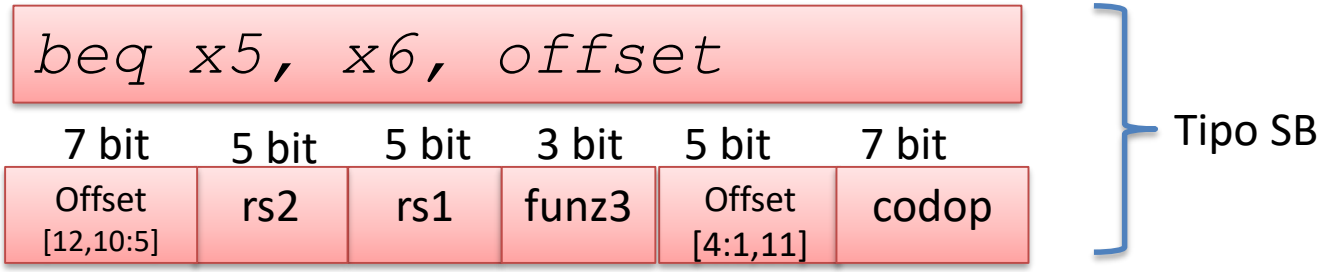
a. Unità di memoria dati



b. Unità di estensione del segno

Salto condizionato

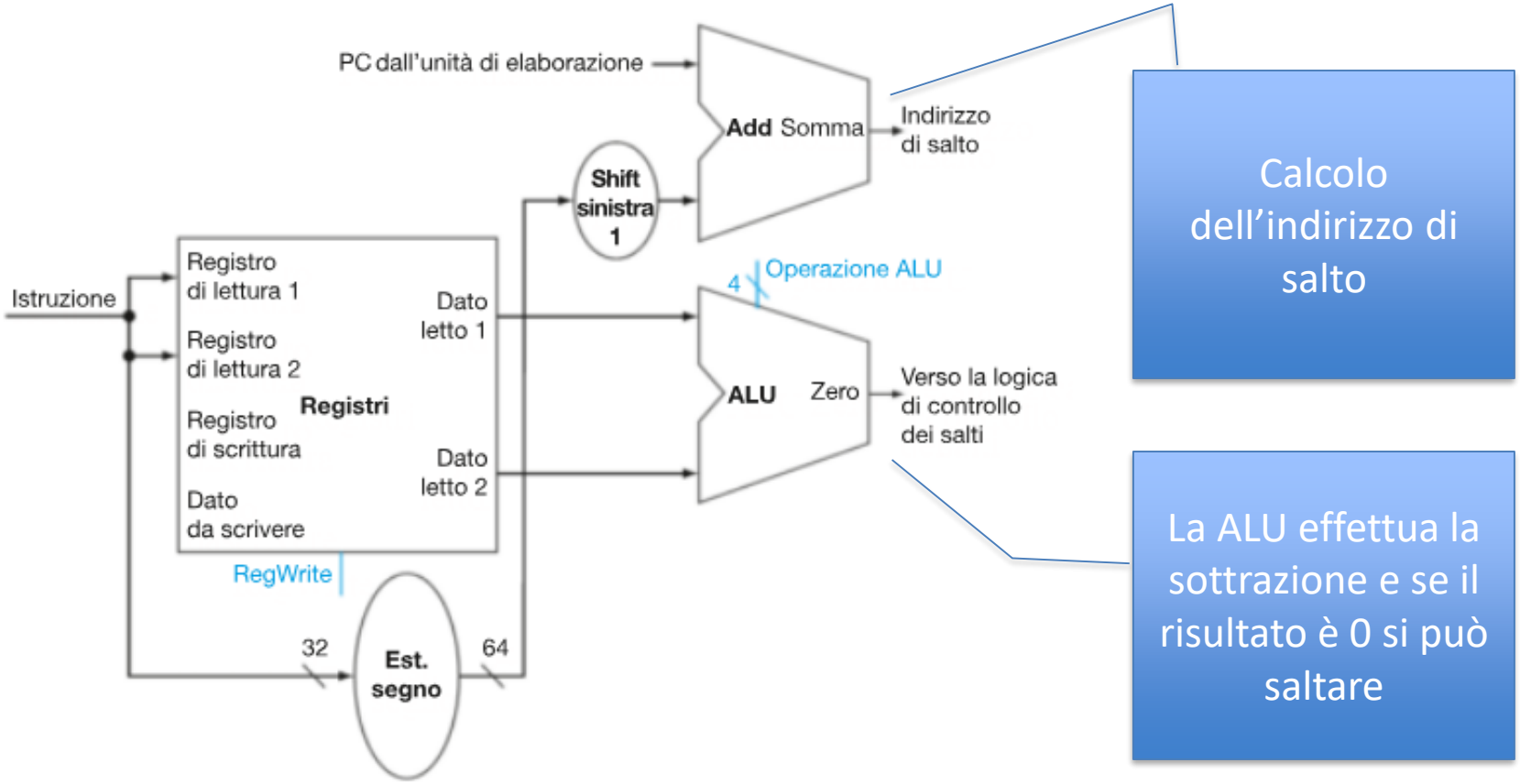
- L'istruzione di salto condizionato ha la forma



- Anche in questo caso bisogna sommare all'attuale PC l'offset a 12 bit (dopo averlo esteso a 64 bit con segno) che consente di fare salti da -2^{12} a 2^{12}.
- Due note:
  - L'architettura dell'insieme delle istruzioni specifica che l'indirizzo di base per il calcolo dell'indirizzo di salto è quello dell'istruzione di salto stessa.
  - L'architettura stabilisce che il campo offset sia spostato di 1 bit a sinistra per fare sì che l'offset codifichi lo spiazzamento in numero di mezze parole (aumentando lo spazio di indirizzamento dell'offset di un fattore 2 rispetto a codifica dello spiazzamento in byte).
  - La ragione per lo shift di uno (anziché di due) è dovuta alla presenza non documentata sul libro di istruzioni compresse a 16 bit per alcuni processori RISC-V.

Salto condizionato

- Nell'esecuzione della *beq* occorre anche un meccanismo in base al quale decidere se aggiornare il PC a PC + 4 o a PC + offset

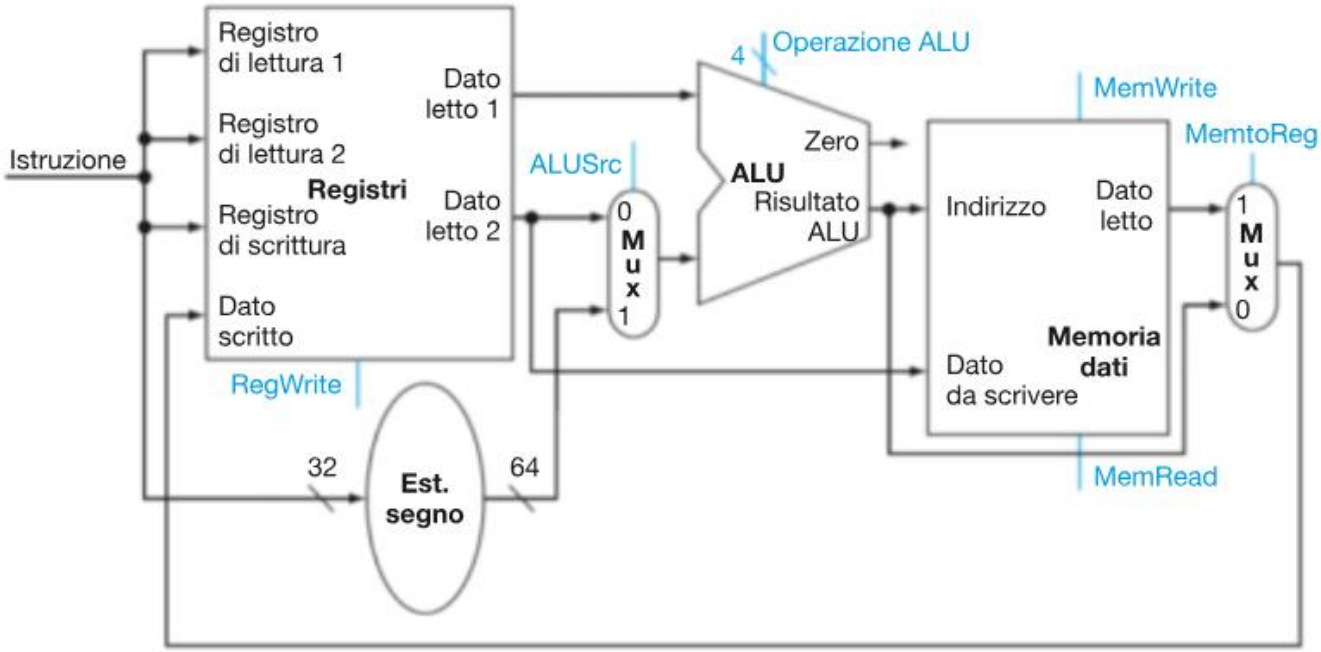


Progetto di un'unità di elaborazione

- 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)

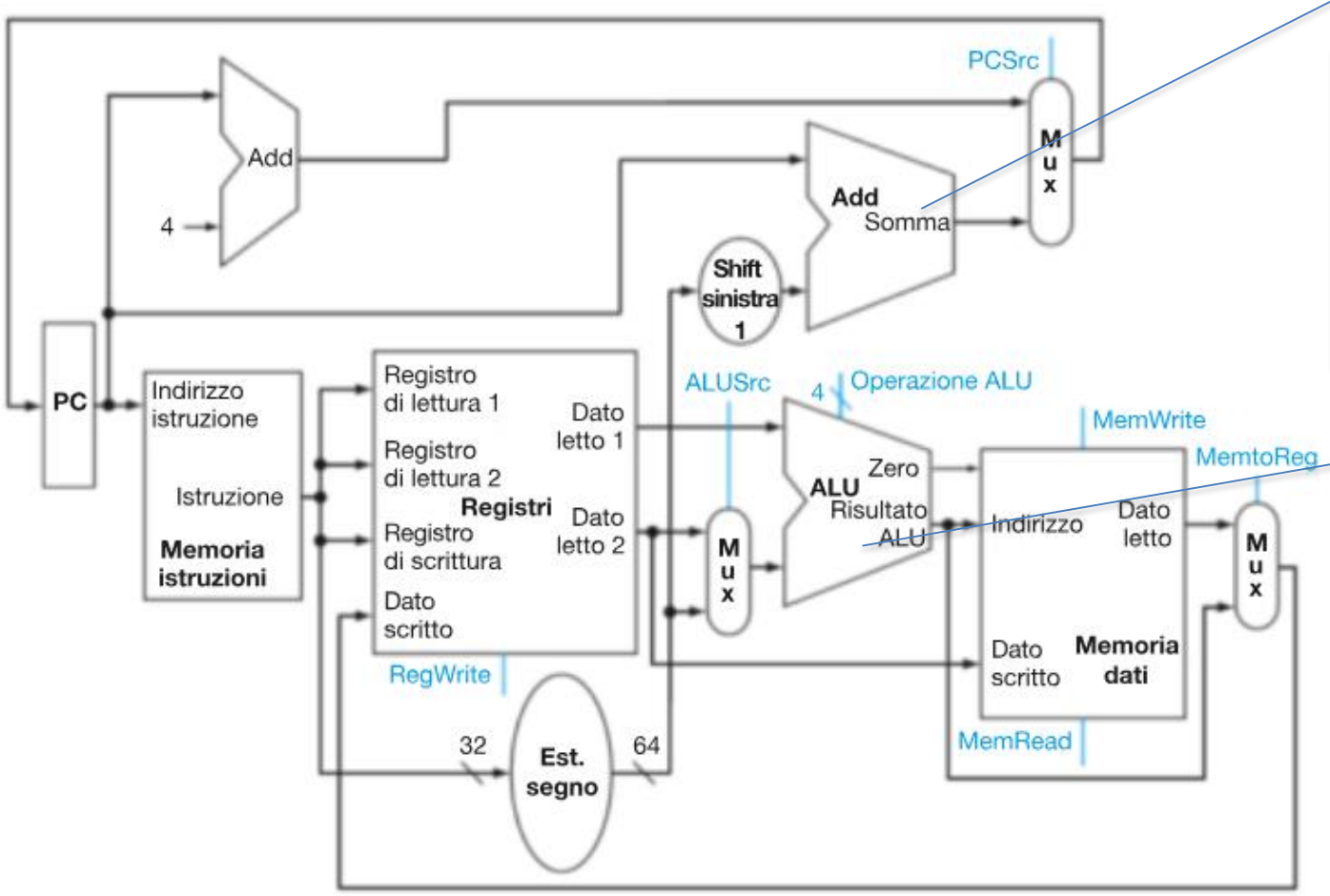


Per istruzione di tipo R:
- ALUSrc = 0
- MemtoReg = 0
- REGwrite = 1
- MemRead = 0
- MemWrite = 0

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).

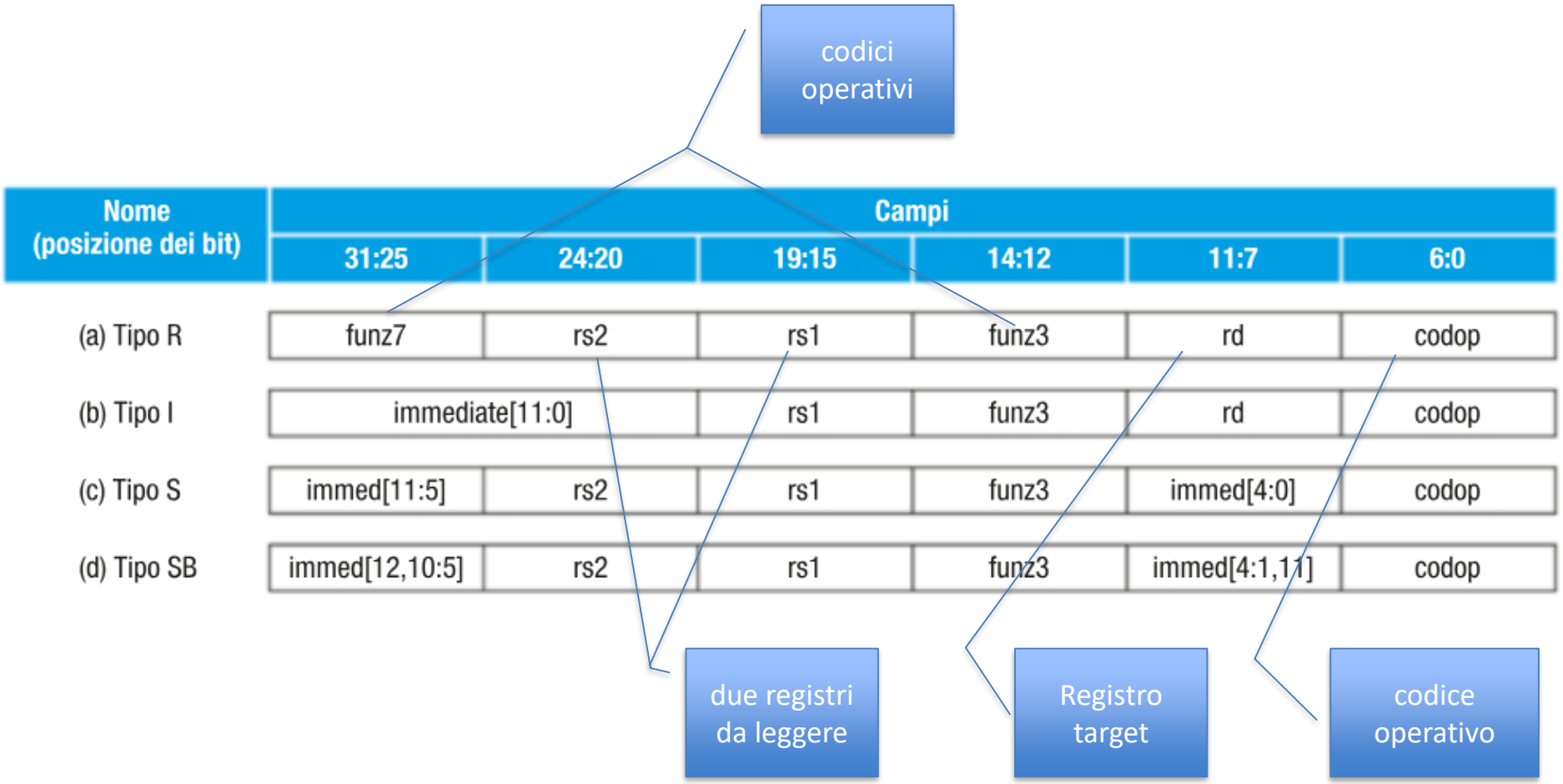
Unità di controllo dell'ALU

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

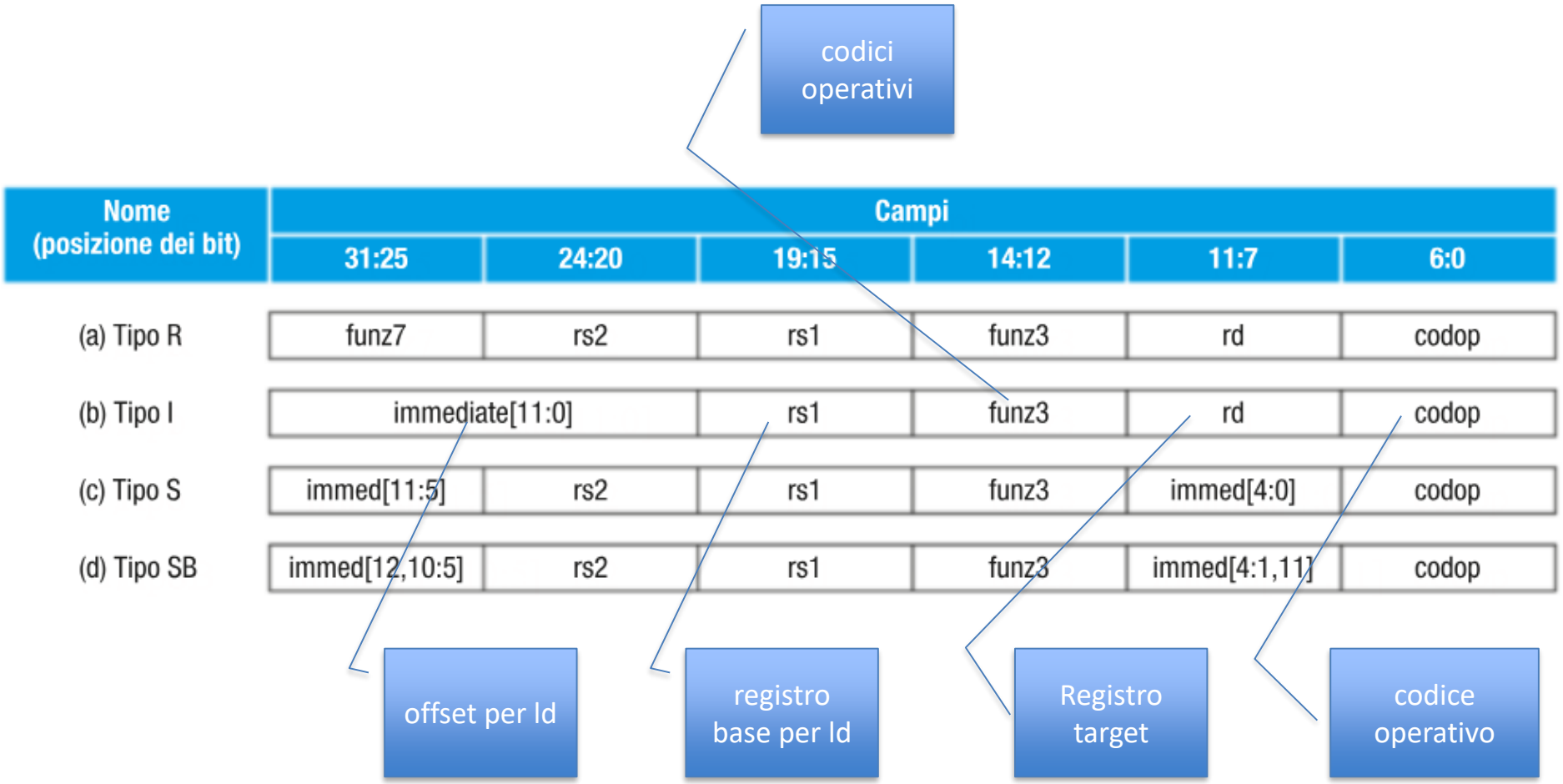
Unità di controllo principale

- Riguardiamo i campi.



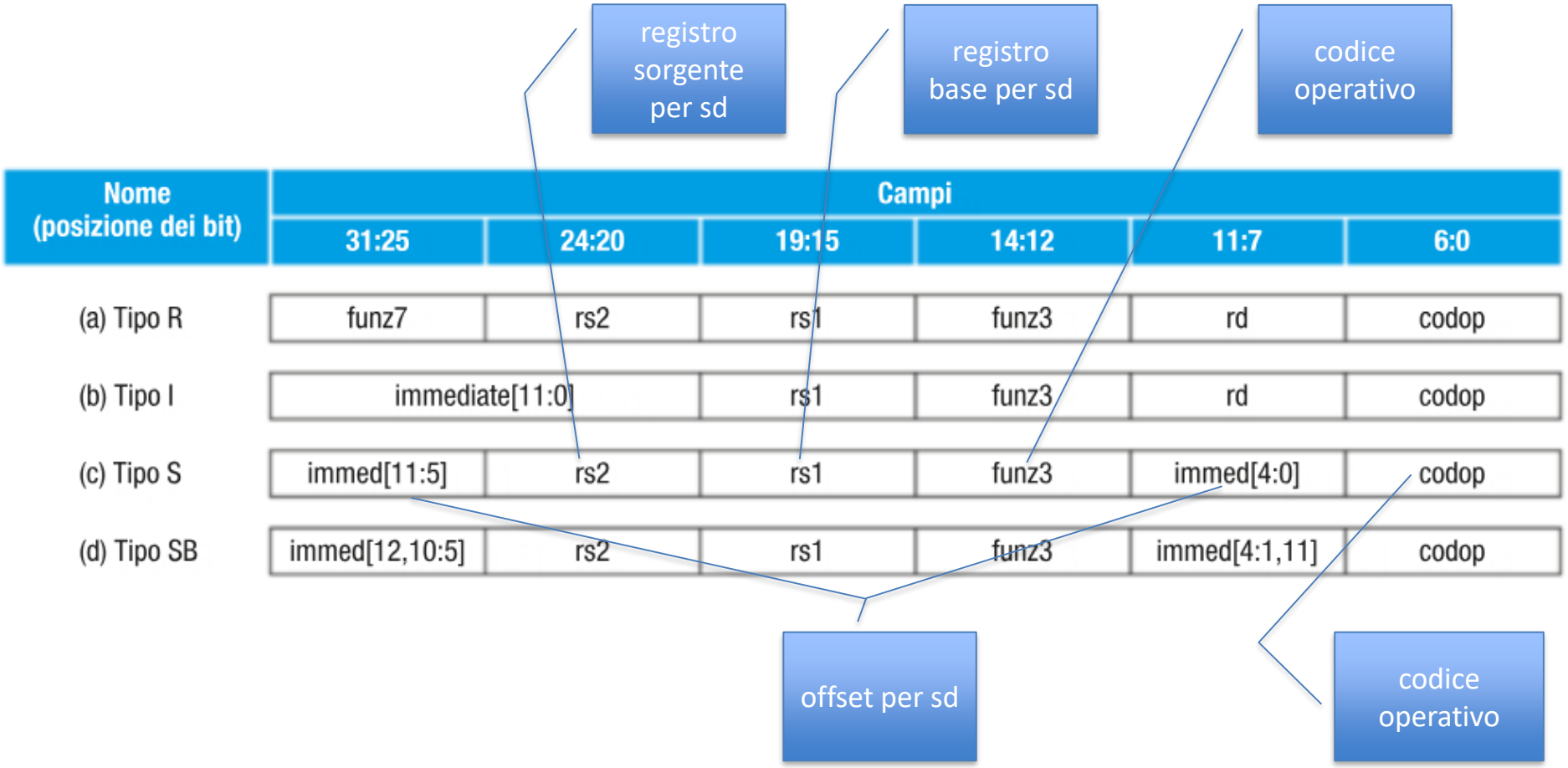
Unità di controllo principale

- Riguardiamo i campi.



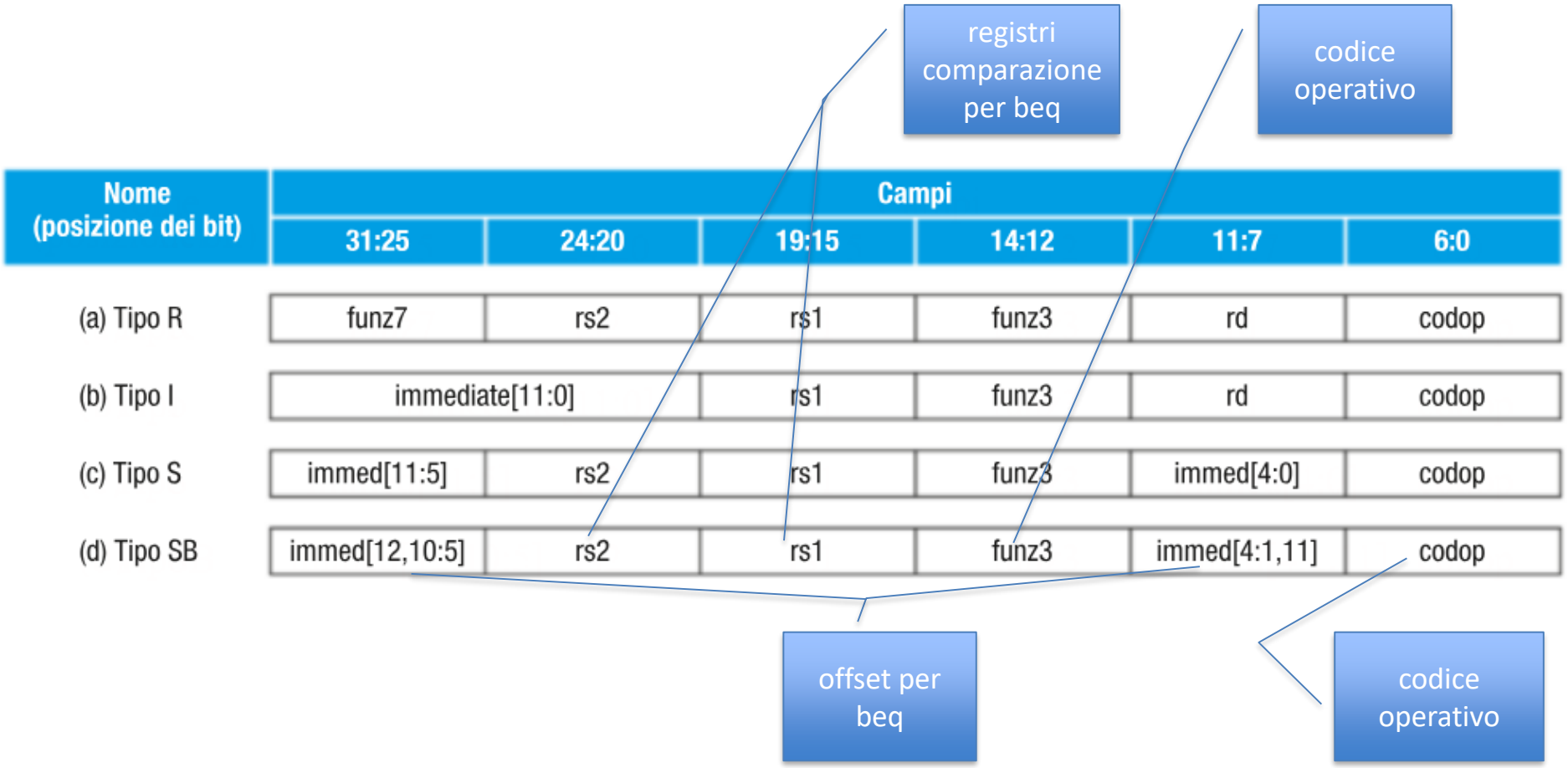
Unità di controllo principale

- Riguardiamo i campi.

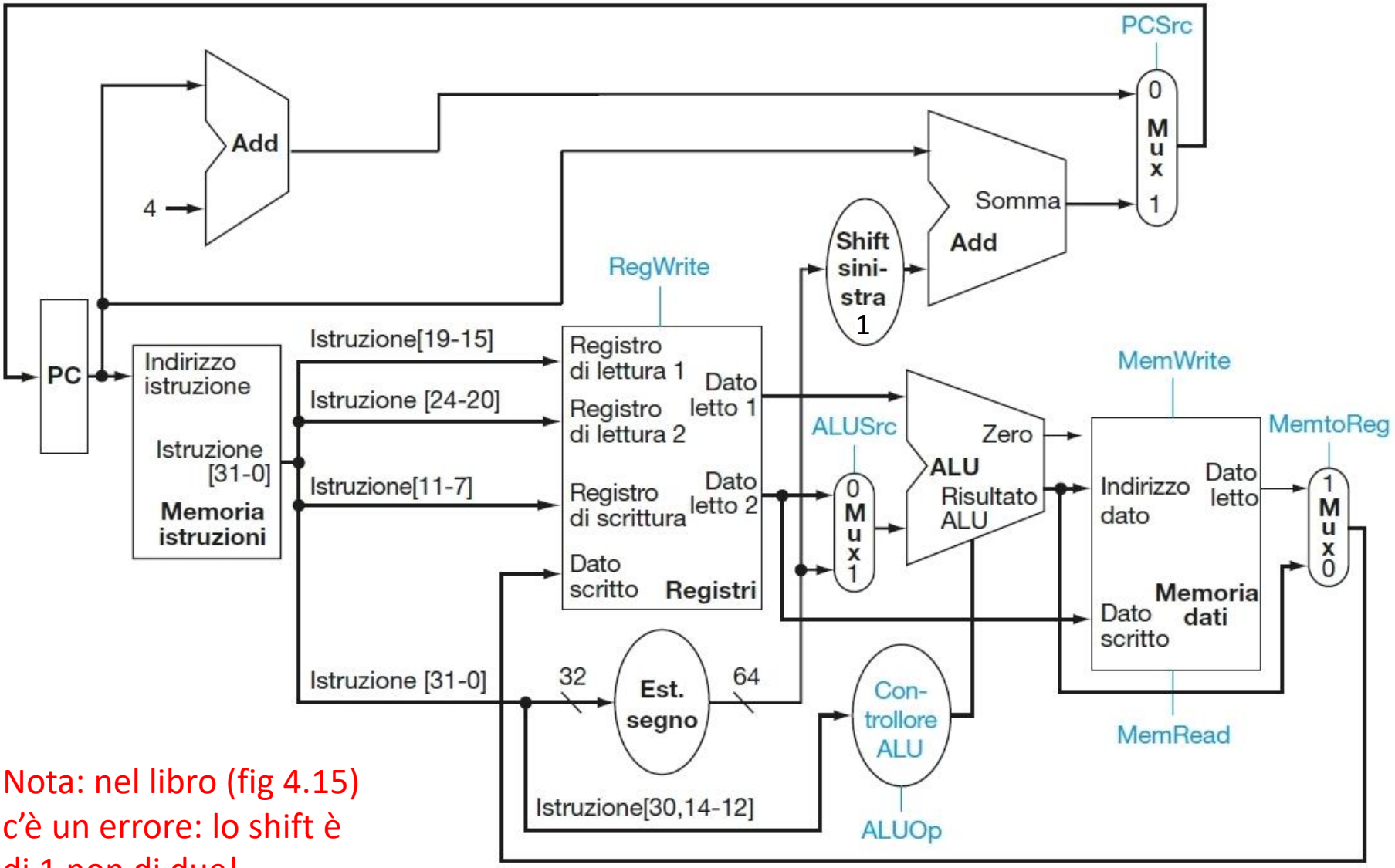


Unità di controllo principale

- Riguardiamo i campi.



Panoramica dei segnali di controllo

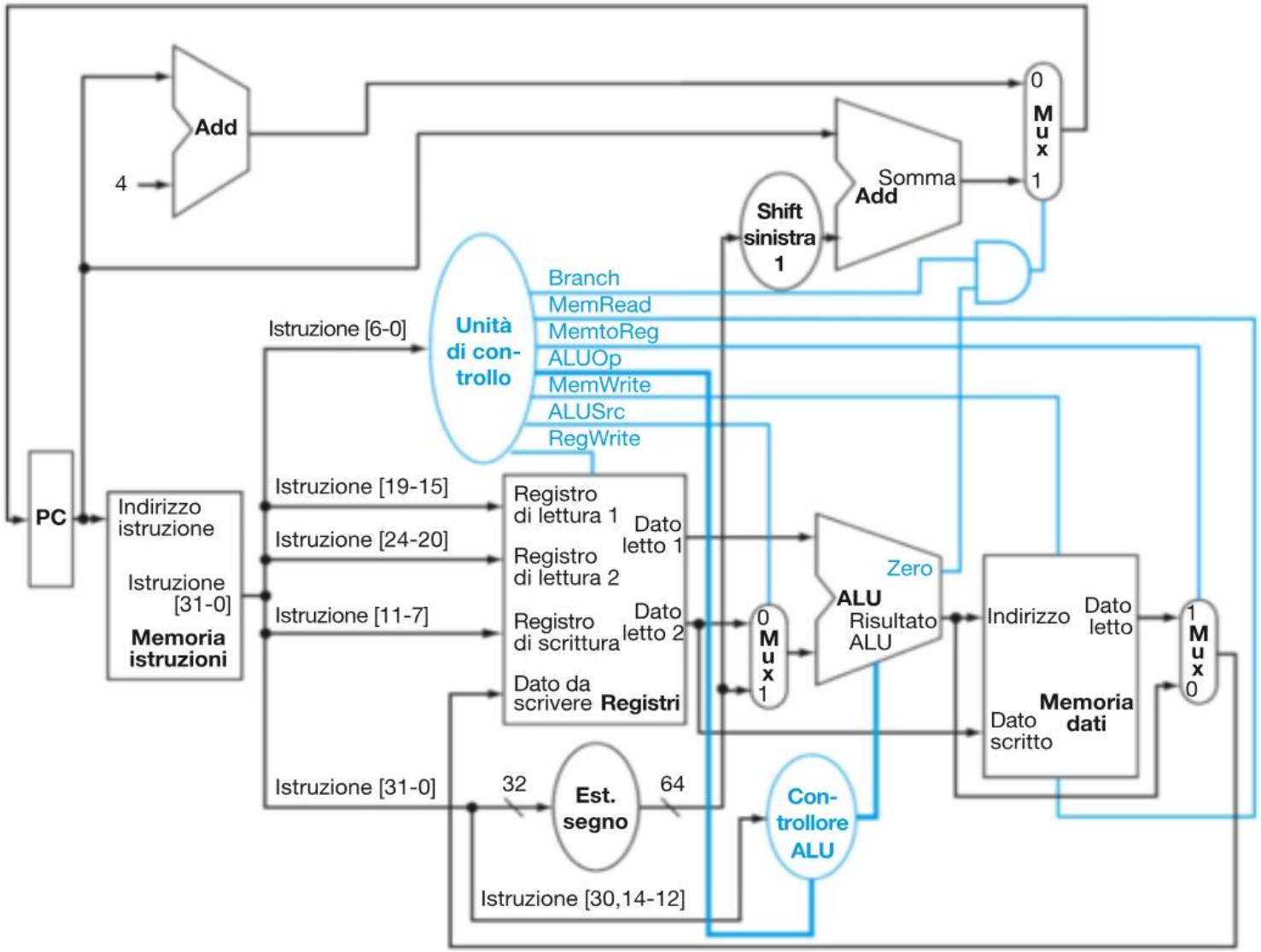


Nota: nel libro (fig 4.15) c'è un errore: lo shift è di 1 non di due!

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

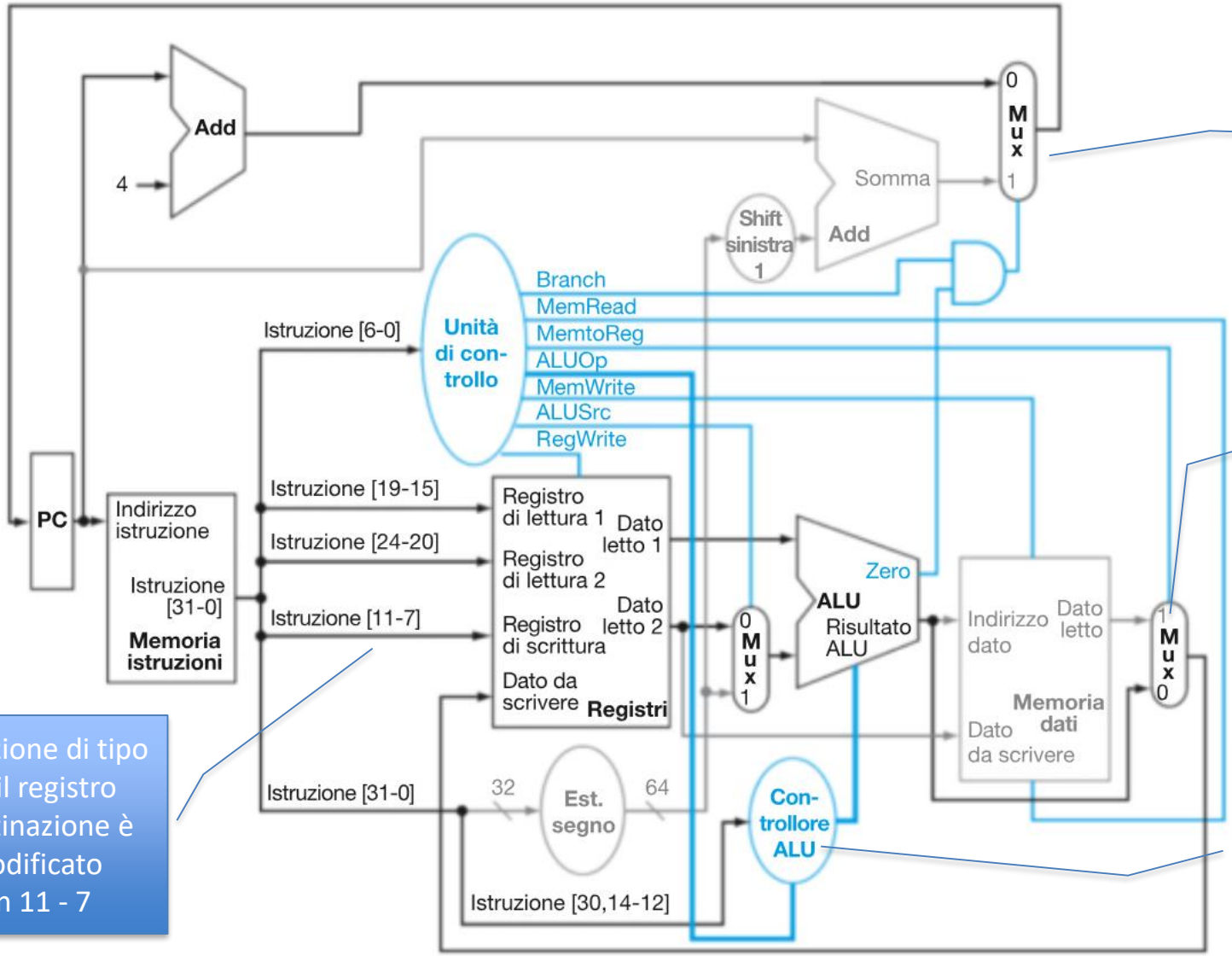
Partiamo dalla ADD

- 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)

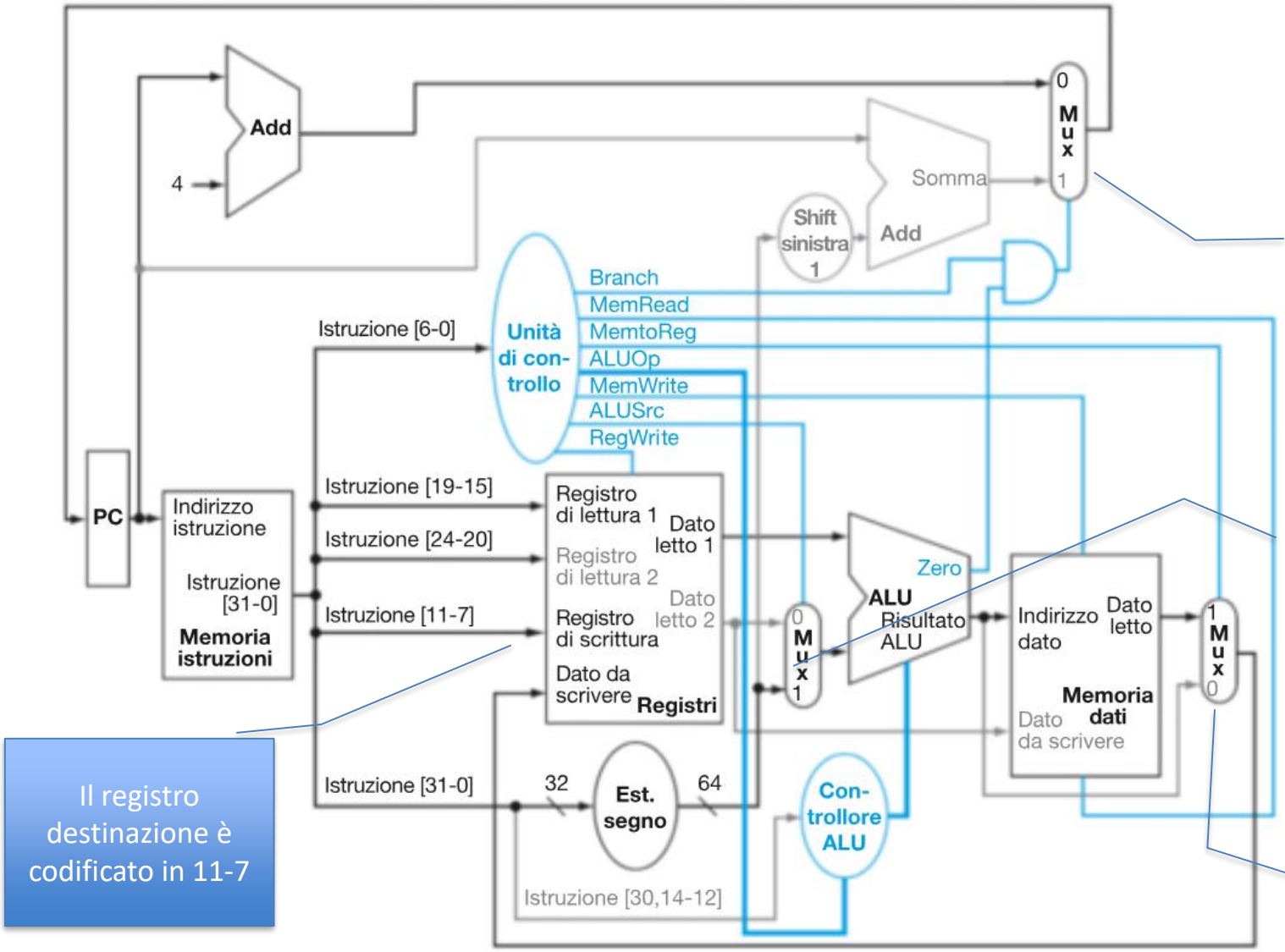
Istruzione di tipo R: il registro destinazione è codificato in 11 - 7

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)

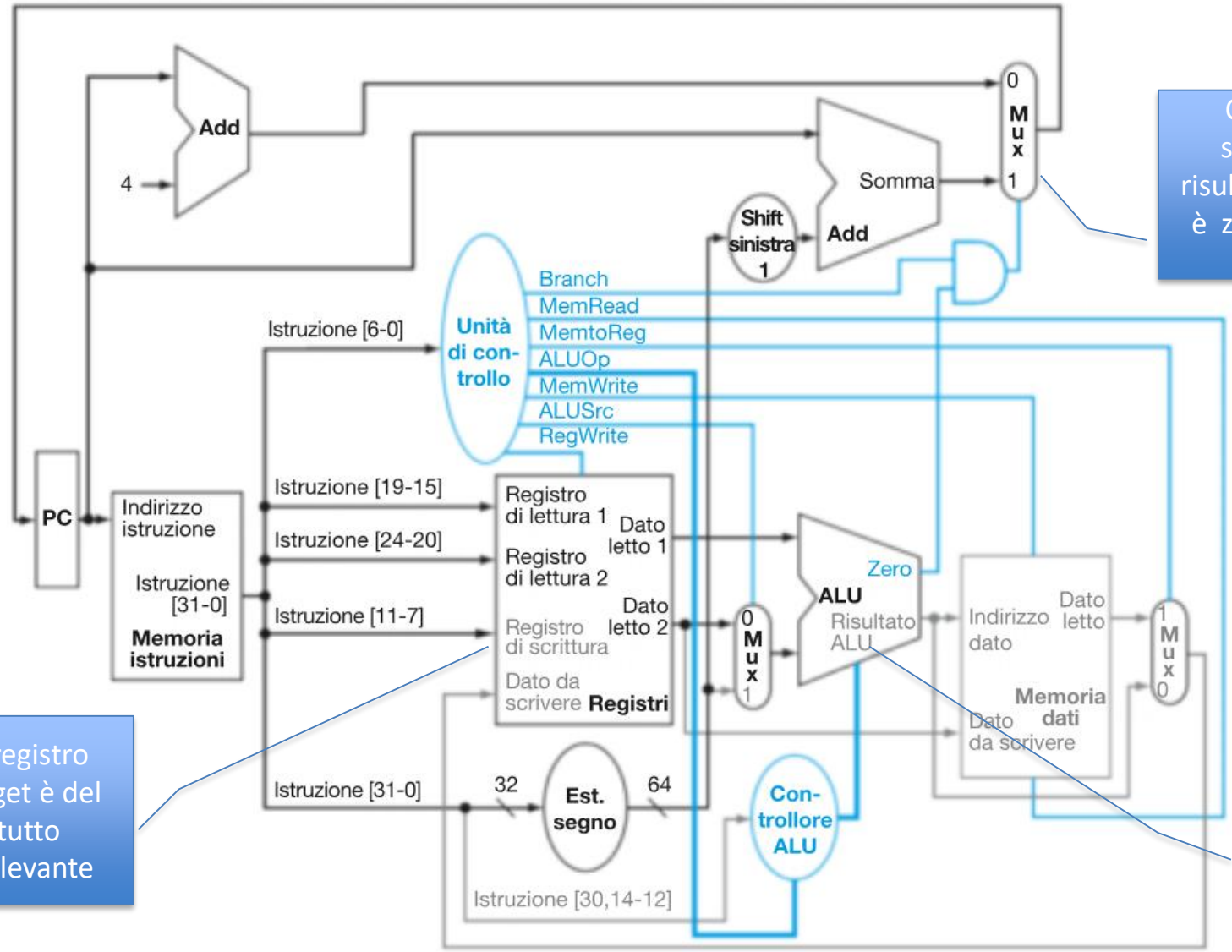
In questo Mux, stavolta, il dato che passa è quello sopra (provenienza memoria)

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