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

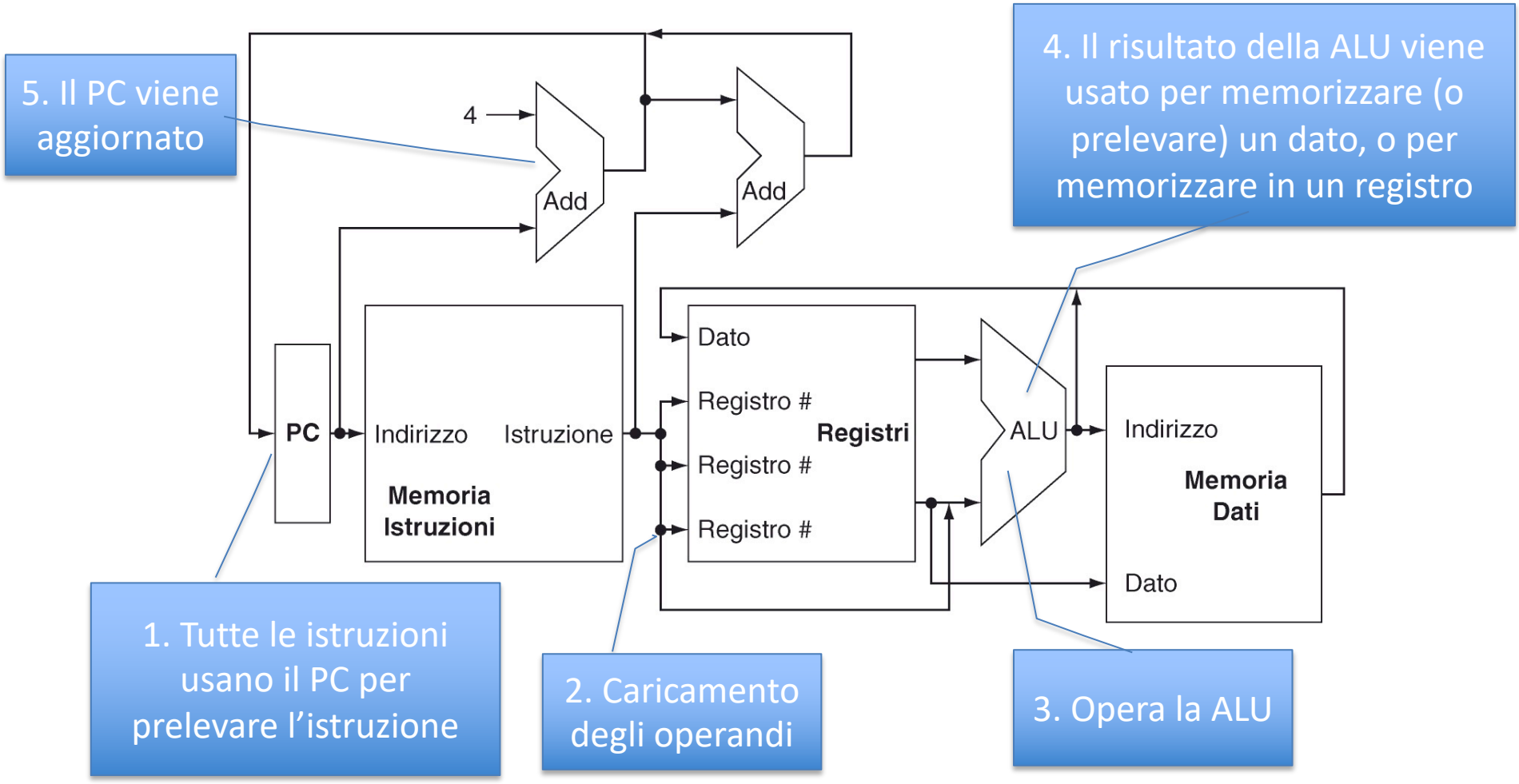
- Nell'esecuzione delle istruzioni, per come è progettata l'ISA, vi sono molti tratti comuni
- Le prime due fasi, *per ogni istruzione,* sono
  - Prelievo dell'istruzione dalla memoria
  - Lettura del valore di uno o più registri operandi che vengono estratti direttamente dai campi dell'istruzione
- 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

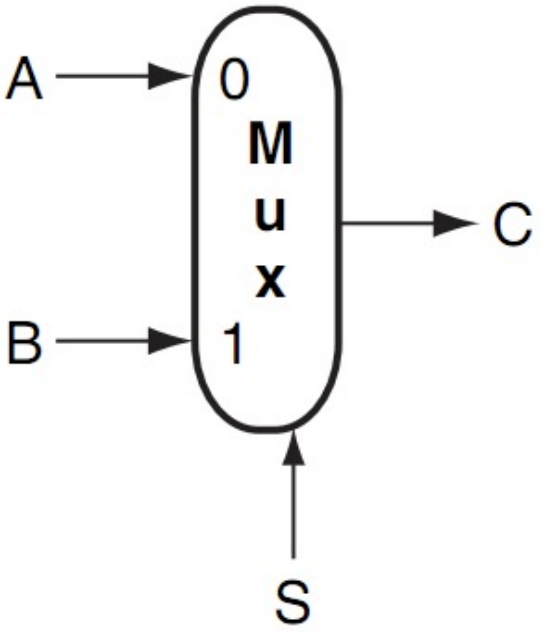
- La figura precedente è incompleta e crea l'impressione che ci sia un flusso continuo di dati
- In realtà ci sono punti in cui i dati arrivano da diverse sorgenti e bisogna sceglierne una (punto di decisione)
- E' il caso per esempio dell'incremento del PC
  - Nel caso "normale" il suo valore proviene 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

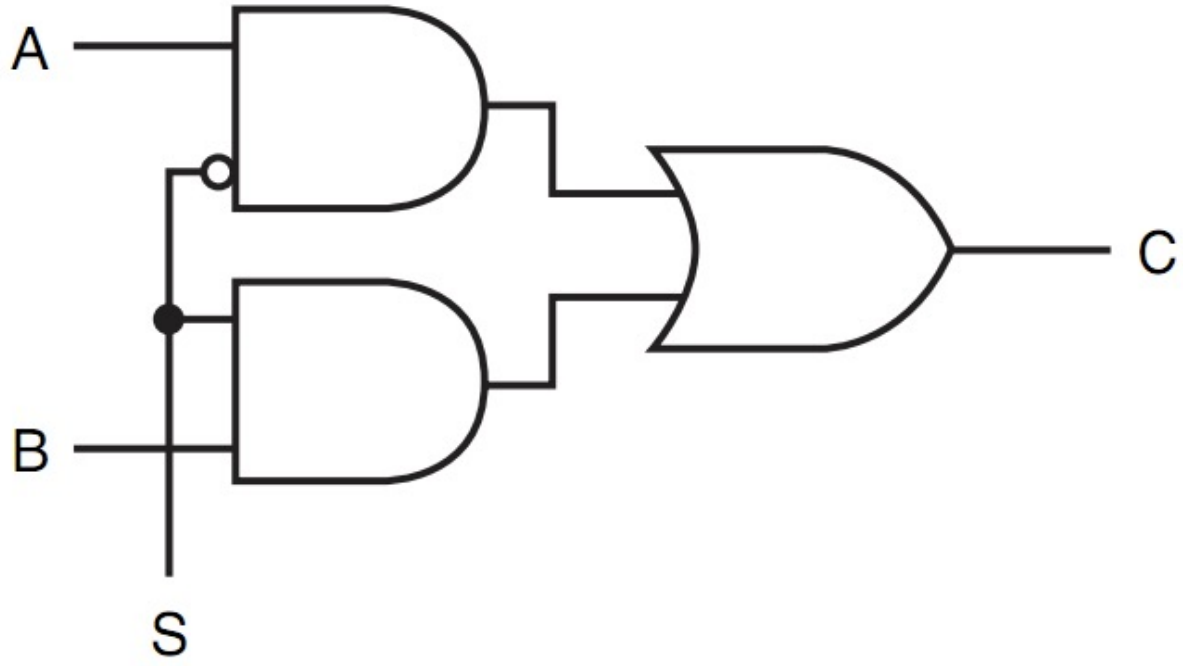
Pezzi mancanti

- 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 un 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



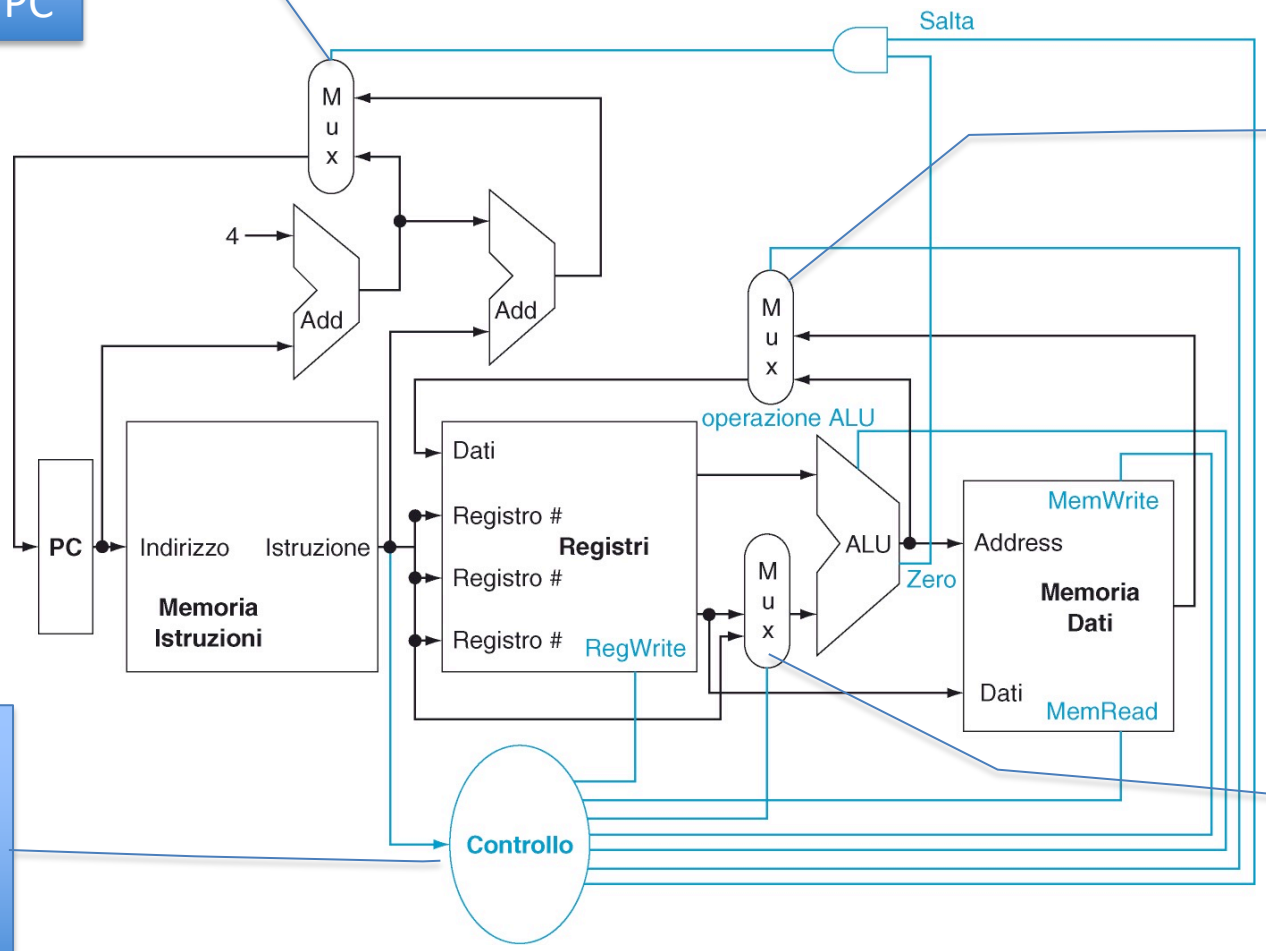


Ulteriori pezzi mancanti

- Le linee di controllo dei multiplexer vengono impostate sulla base del tipo di istruzioni
- 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

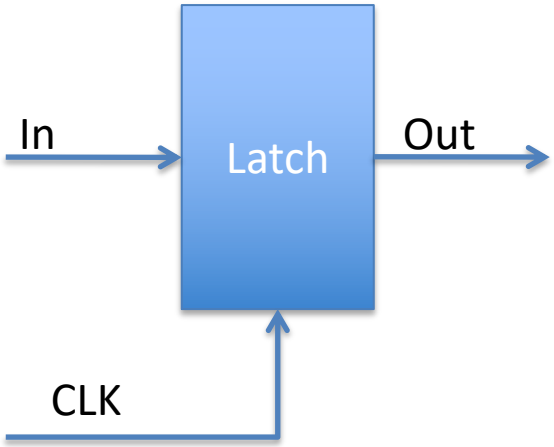
- ASSUNZIONE SEMPLIFICATIVA:
  - Il processore lavora sincronizzandosi con i cicli di clock
  - Per il momento facciamo l'assunzione semplificativa che tutte le istruzioni si svolgano in un singolo ciclo di clock (lungo abbastanza)
- Prima di entrare nella descrizione dei vari componenti ricordiamo velocemente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

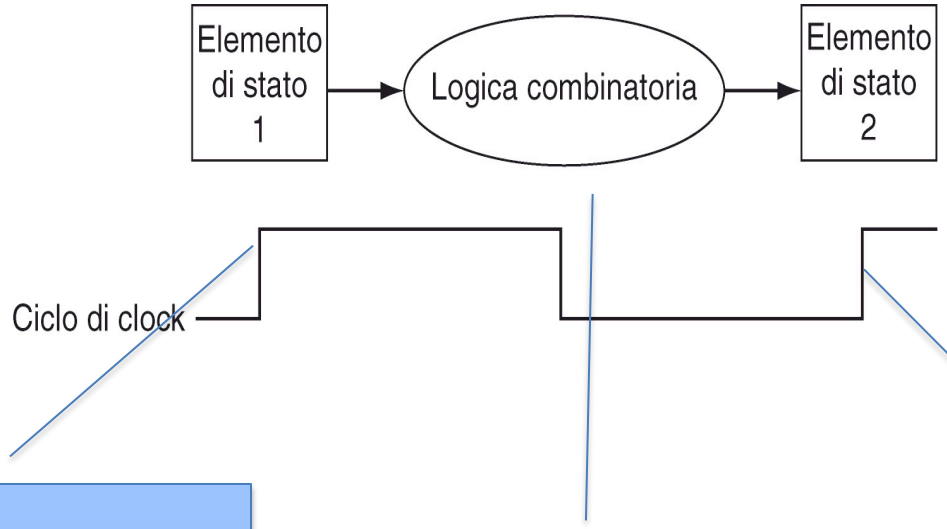


- I registri possono essere ottenuti come array di latch (o in altri modi simili)
- Terminologia
  - Asserito: segnale logico a livello alto
  - Non Asserito: segnale logico a livello basso

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

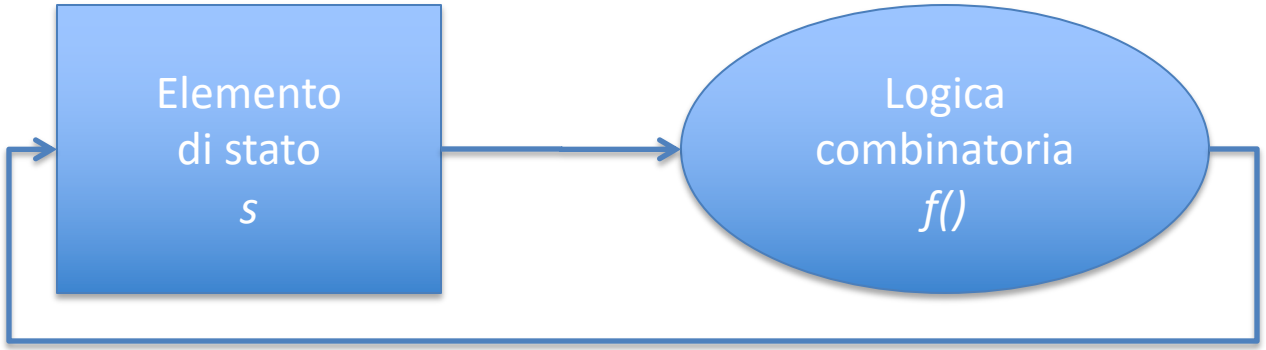
Al tempo t+T il valore arriva all'elemento di stato 2

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):

$$s = f(s)$$

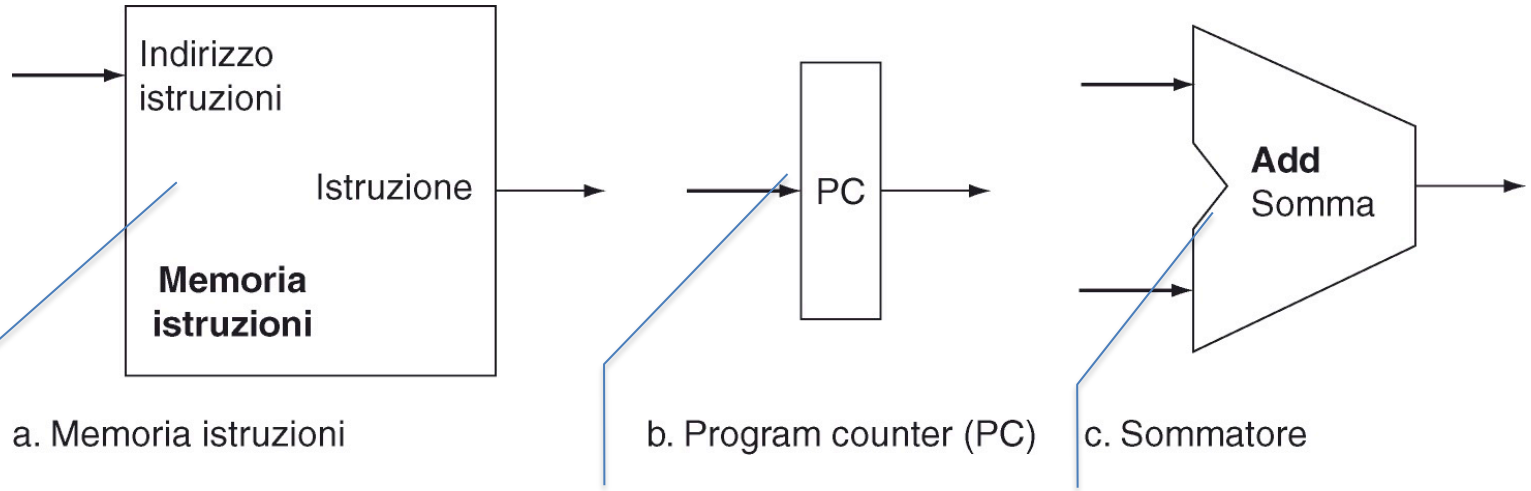
- Grazie alla temporizzazione sensibile al clock abbiamo:

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

che invece produce un'evoluzione ben determinata

Realizzazione del datapath

- Passiamo ora in rassegna le varie componenti che ci servono per la realizzazione del datapath



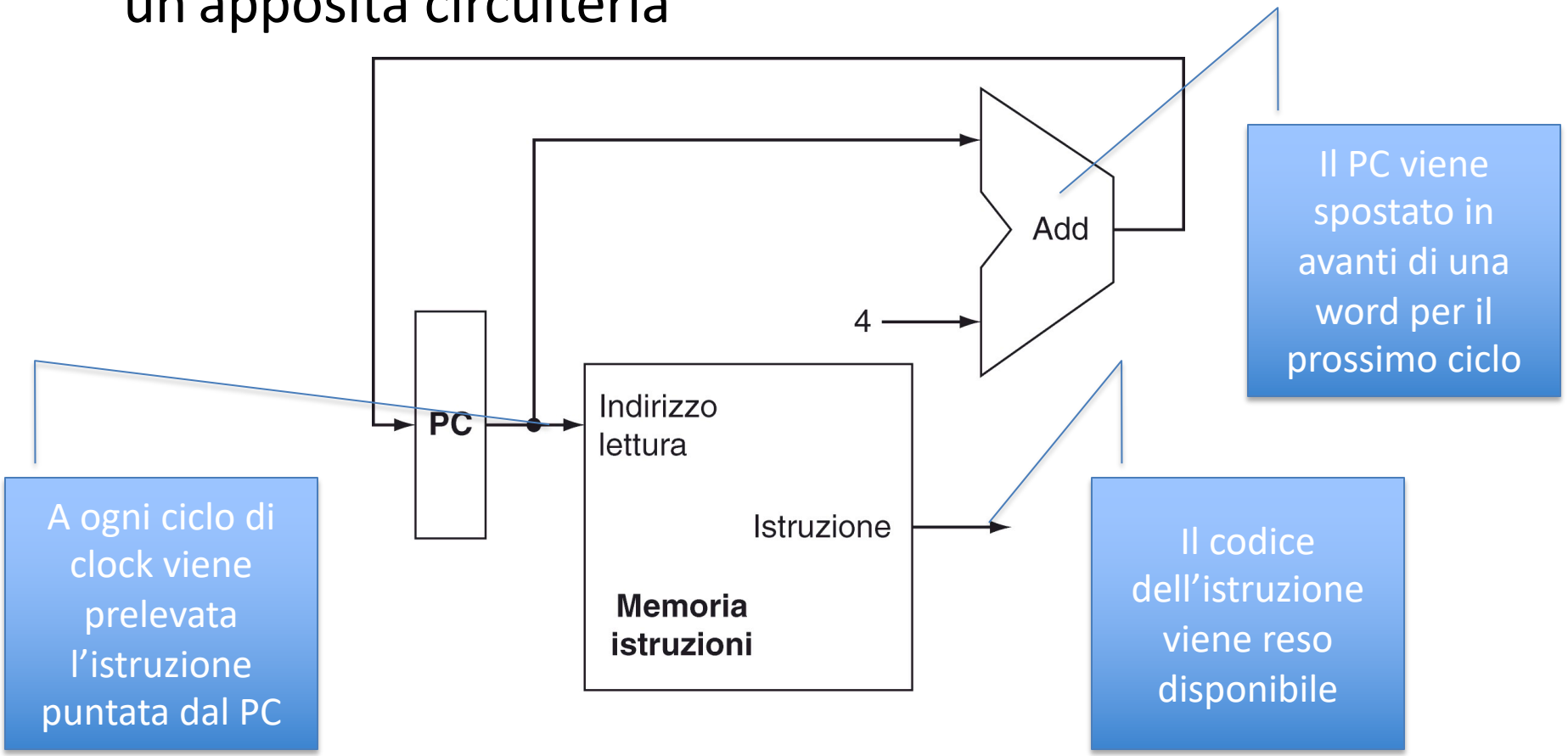
Memoria dove sono memorizzate le istruzioni

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

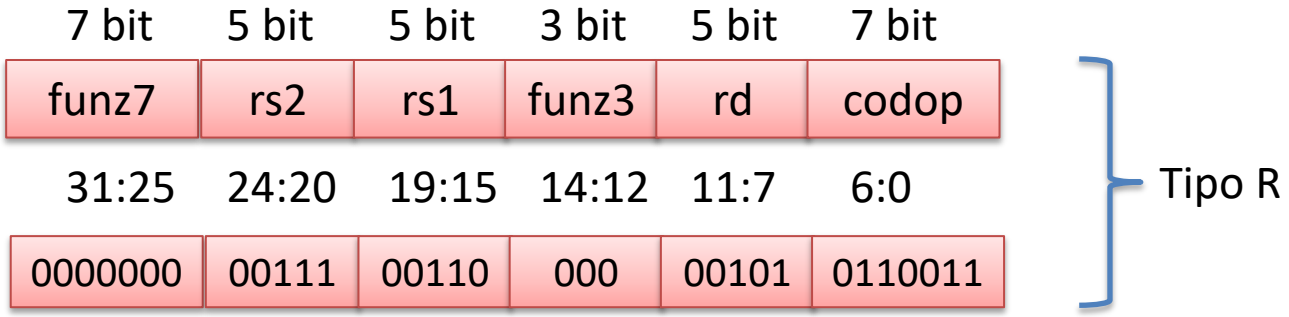


Istruzioni di tipo R

- 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:

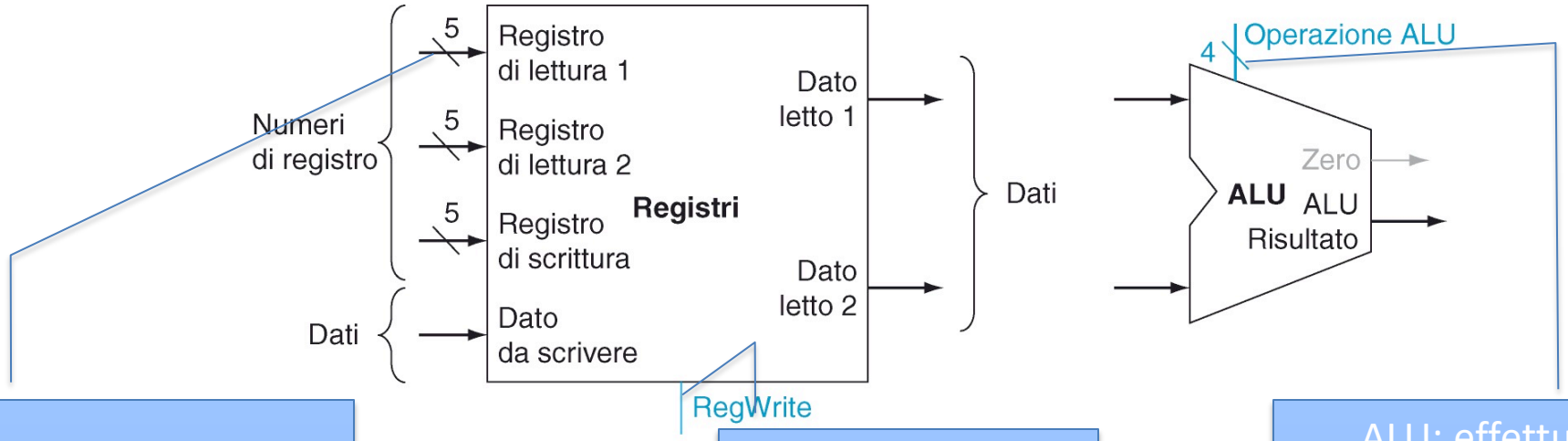
```
add x5, x6, x7
```

- 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

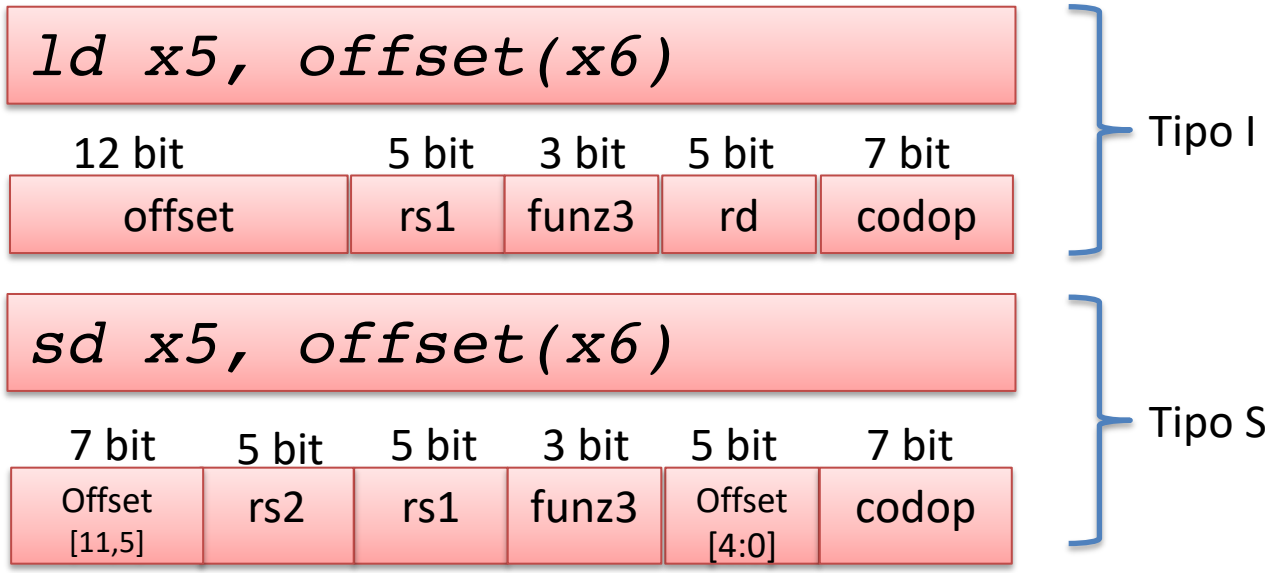
Banco registri: dà in output i registri specificati nel registro di lettura 1 e 2

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

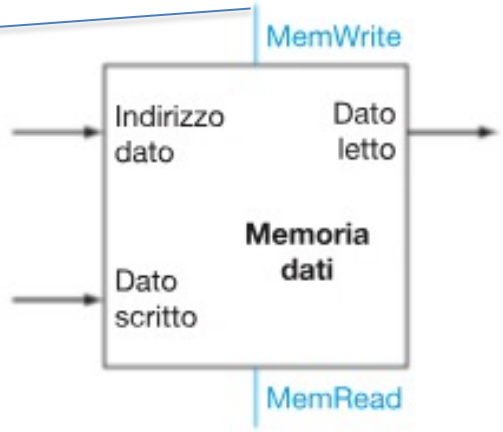
- Consideriamo ora anche le istruzioni load (ld) e store (sd)
- Forma generale



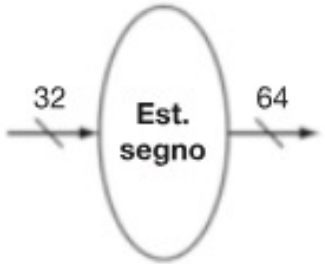
- Per entrambe si deve calcolare un indirizzo di memoria dato dalla somma di x6 con l'offset
- Per entrambe occorre leggere dal register file
- Quindi per eseguire queste istruzioni ci servono ancora la ALU e il register file

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. Unità di memoria dati

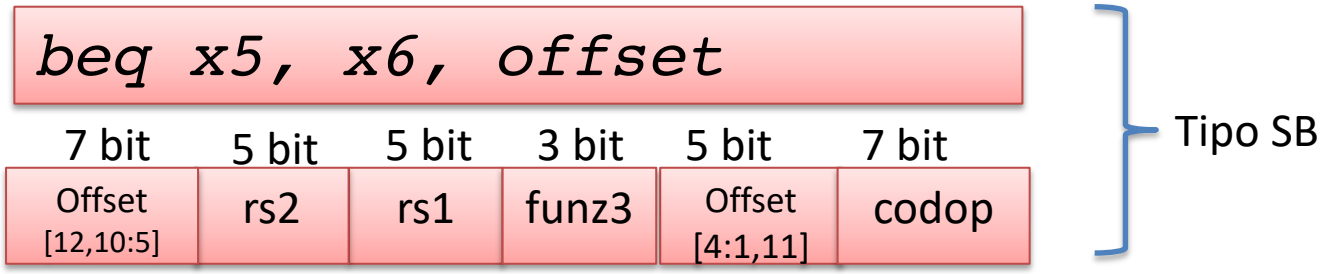


b. Unità di estensione del segno

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

Salto condizionato

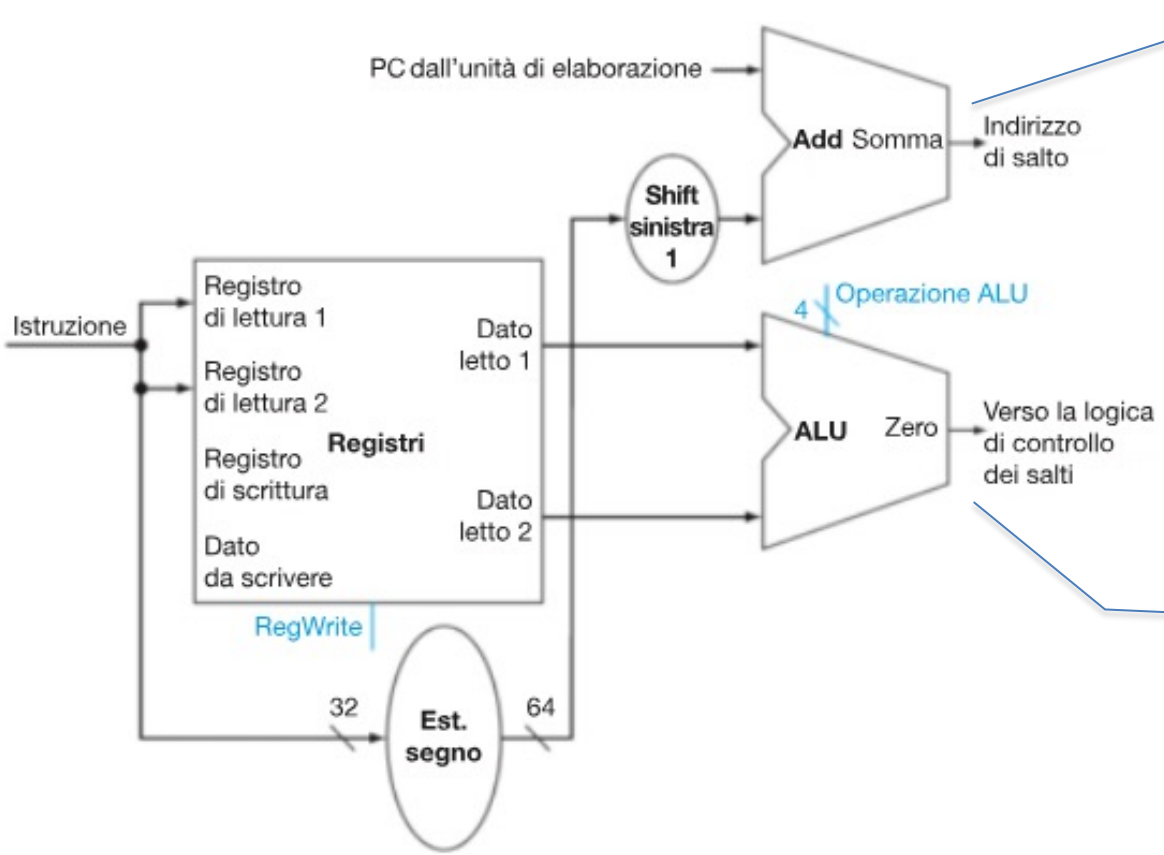
- L'istruzione di salto condizionato ha la forma



- 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



Calcolo dell'indirizzo di salto

La ALU effettua la sottrazione e se il risultato è 0 si può saltare