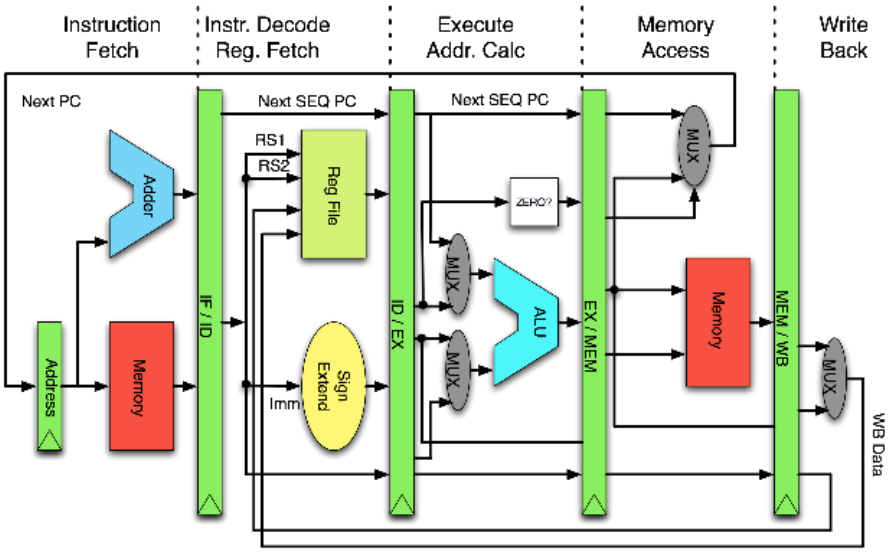
La Microarchitettura MIPS



Michele Favalli

Facciamo il punto

- Abbiamo visto con cosa è fatta una CPU (gate e FF)
- Abbiamo visto cosa fa una CPU (Von Neumann e ISA)
- Adesso possiamo andare a vedere come è fatta una CPU

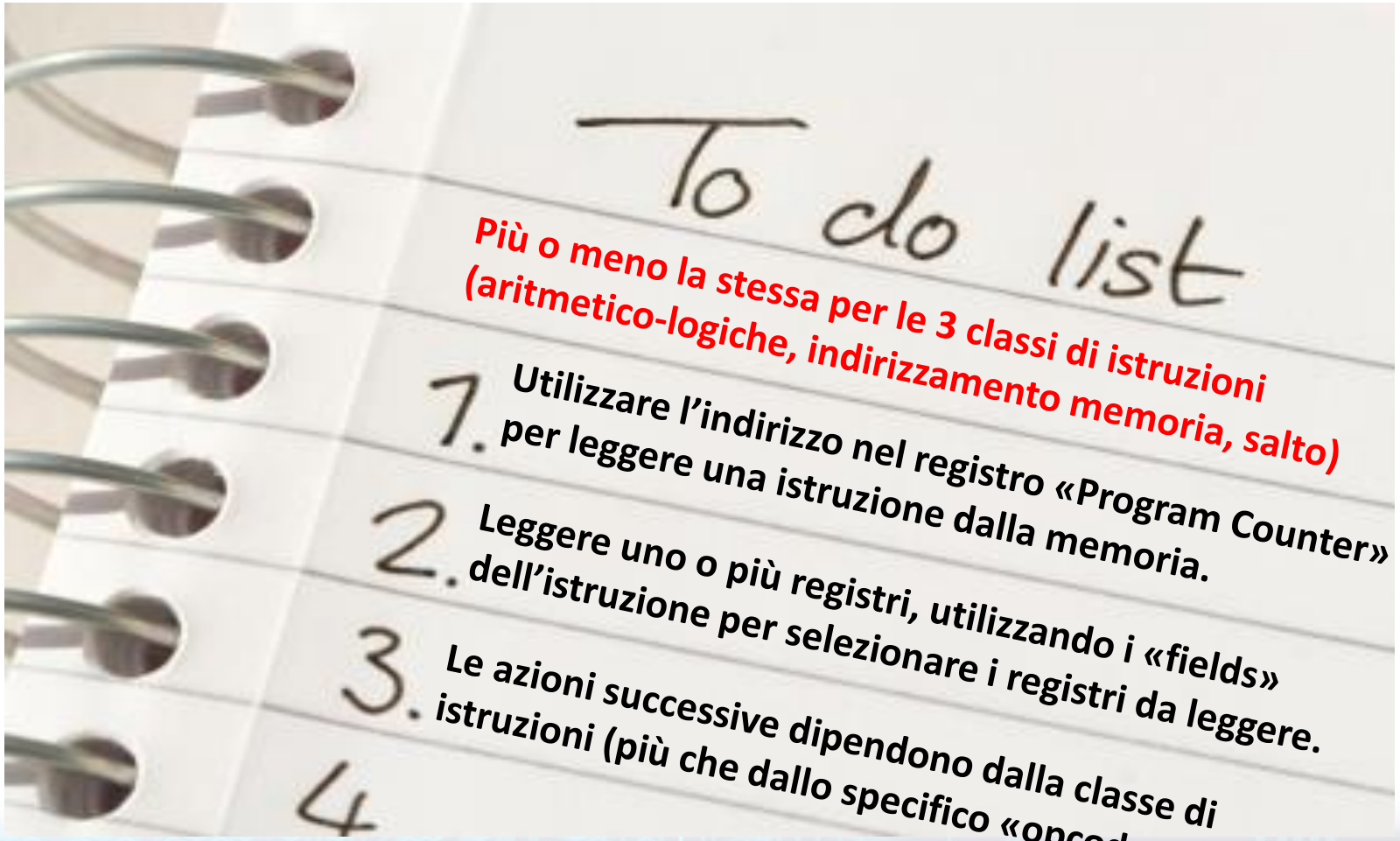
Obiettivo

- **Analizzare l'implementazione della microarchitettura di un microprocessore a singolo core.**
- Per semplicità, verrà illustrata l'implementazione di un sottoinsieme del set di istruzioni:
  - *Istruzioni per il trasferimento di dati (**load**, **store**)*
  - *Istruzioni artimetico-logiche (**add**, **sub**, **and**, **or**, **slt**)*
  - *Istruzioni per il salto condizionale e non (**beq**, **j**)*

Obiettivo

- Capire come l’ISA determina gli aspetti implementativi
- Capire come le strategie implementate determinano le metriche che misurano le prestazioni

Come eseguire una istruzione?

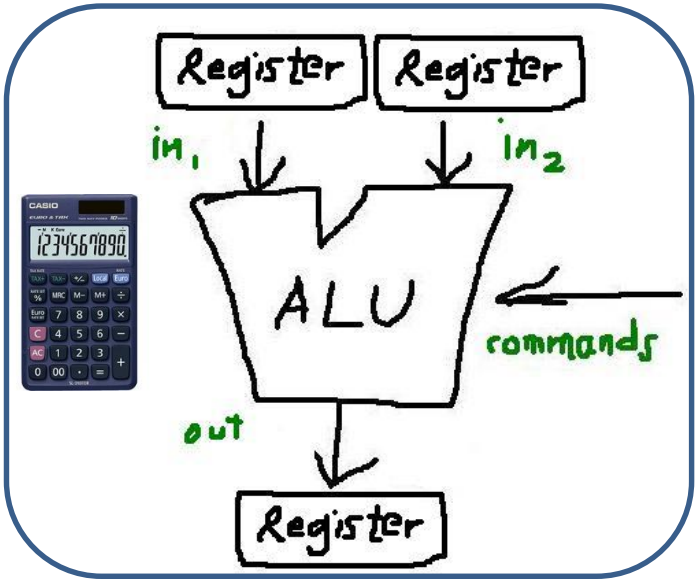


Affinità fra le classi di istruzioni

Tutte le classi di istruzioni (con l'eccezione del salto incondizionato) dopo la lettura dei registri potrebbero usare la stessa ALU => conseguenza della semplicità dell'ISA dell'architettura MIPS

Istruzioni per l'indirizzamento della memoria

Calcolare un indirizzo



Istruzioni aritmetico-logiche

Eseguire le operazioni aritmetiche o logiche

Istruzioni per il salto condizionato

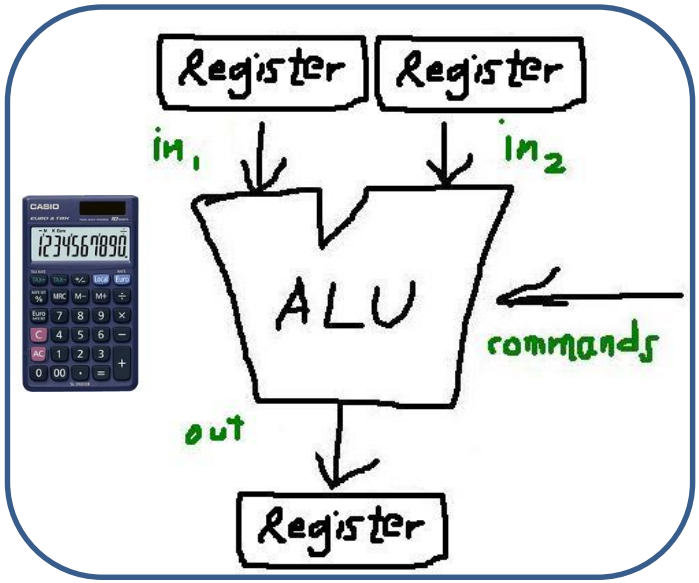
Verificare la condizione per il salto.

Affinità

Dopo l'ALU, le azioni si diversificano

Istruzioni per l'indirizzamento della memoria

Accedere alla memoria per una load o store



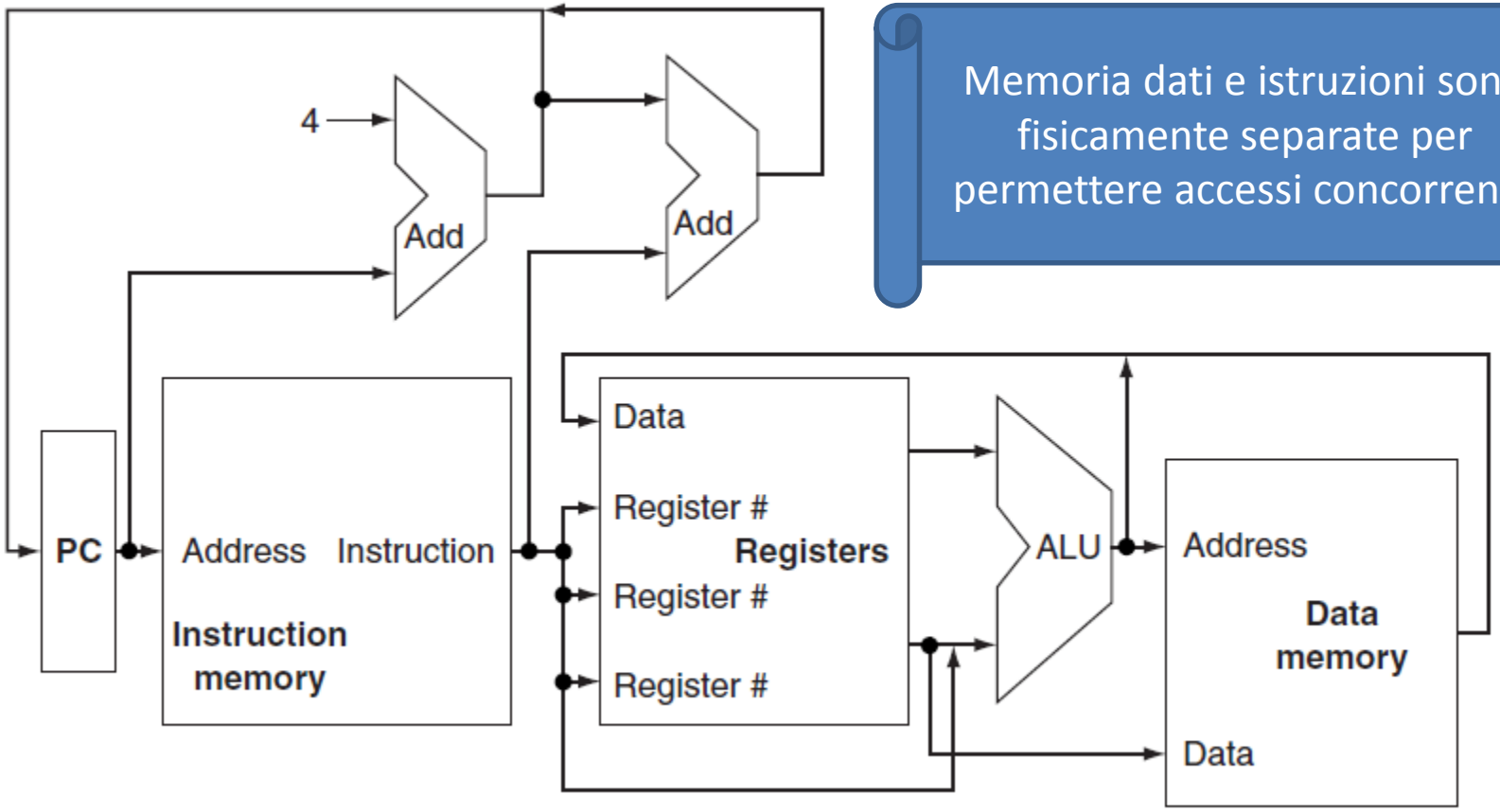
Istruzioni aritmetico-logiche

Scrivere il risultato in un registro destinazione

Istruzioni per il salto condizionato

Aggiornare il valore del PC sulla base della condizione

Vista Astratta di un Microprocessore MIPS a Singolo Core

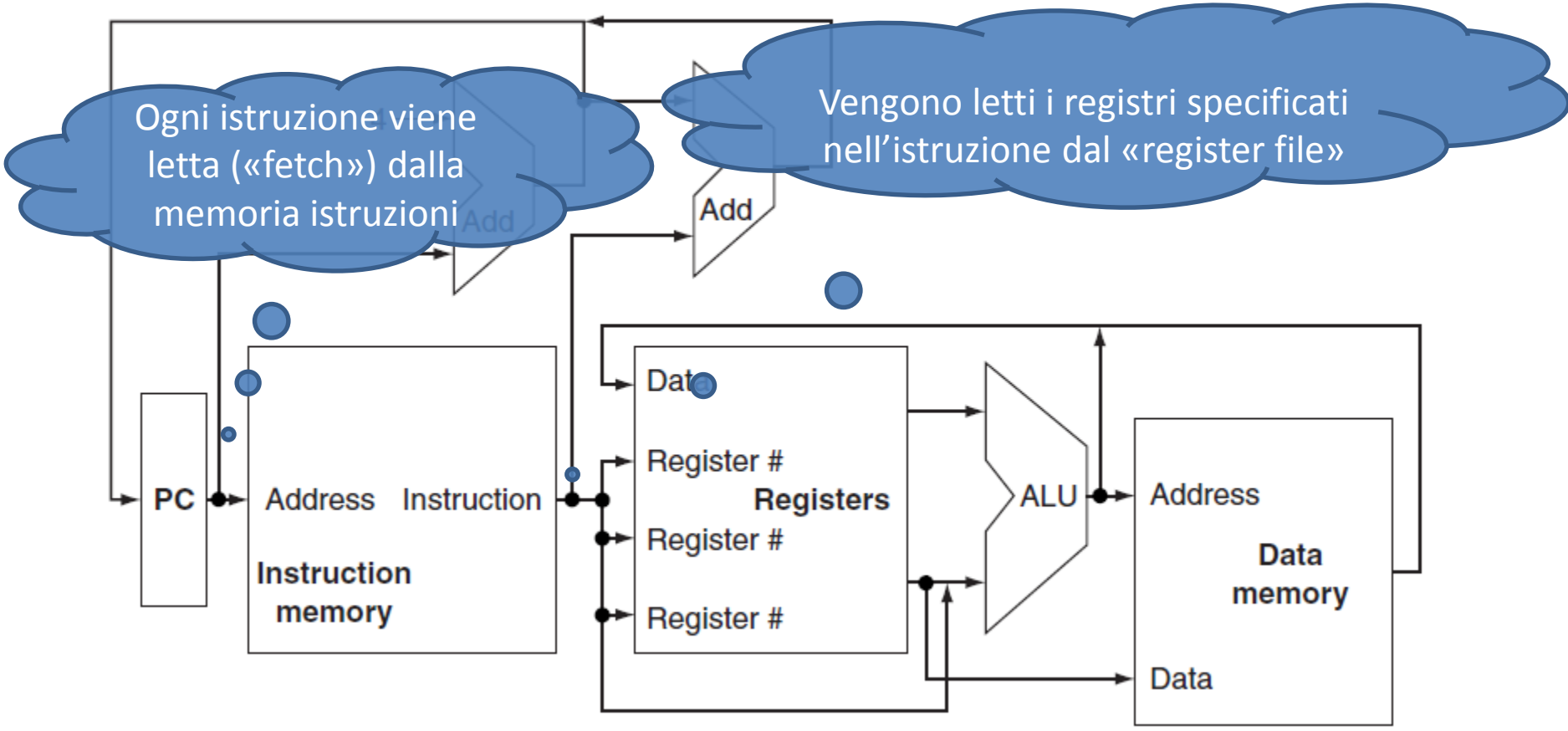


Memoria dati e istruzioni sono fisicamente separate per permettere accessi concorrenti.

Si notino le principali unità funzionali e le connessioni tra loro

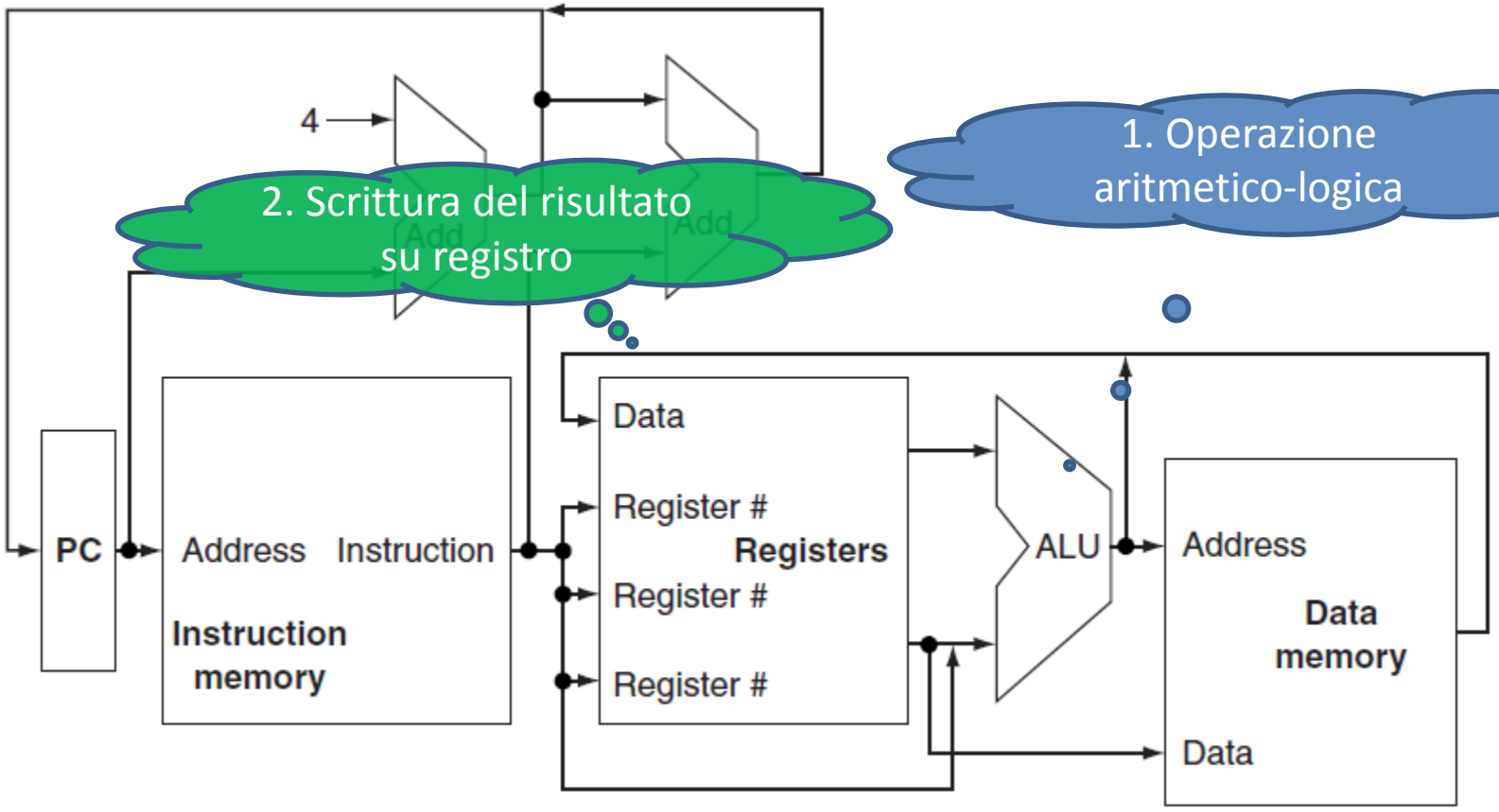
Vista Astratta di un Microprocessore MIPS a Singolo Core

Comune a tutte le istruzioni



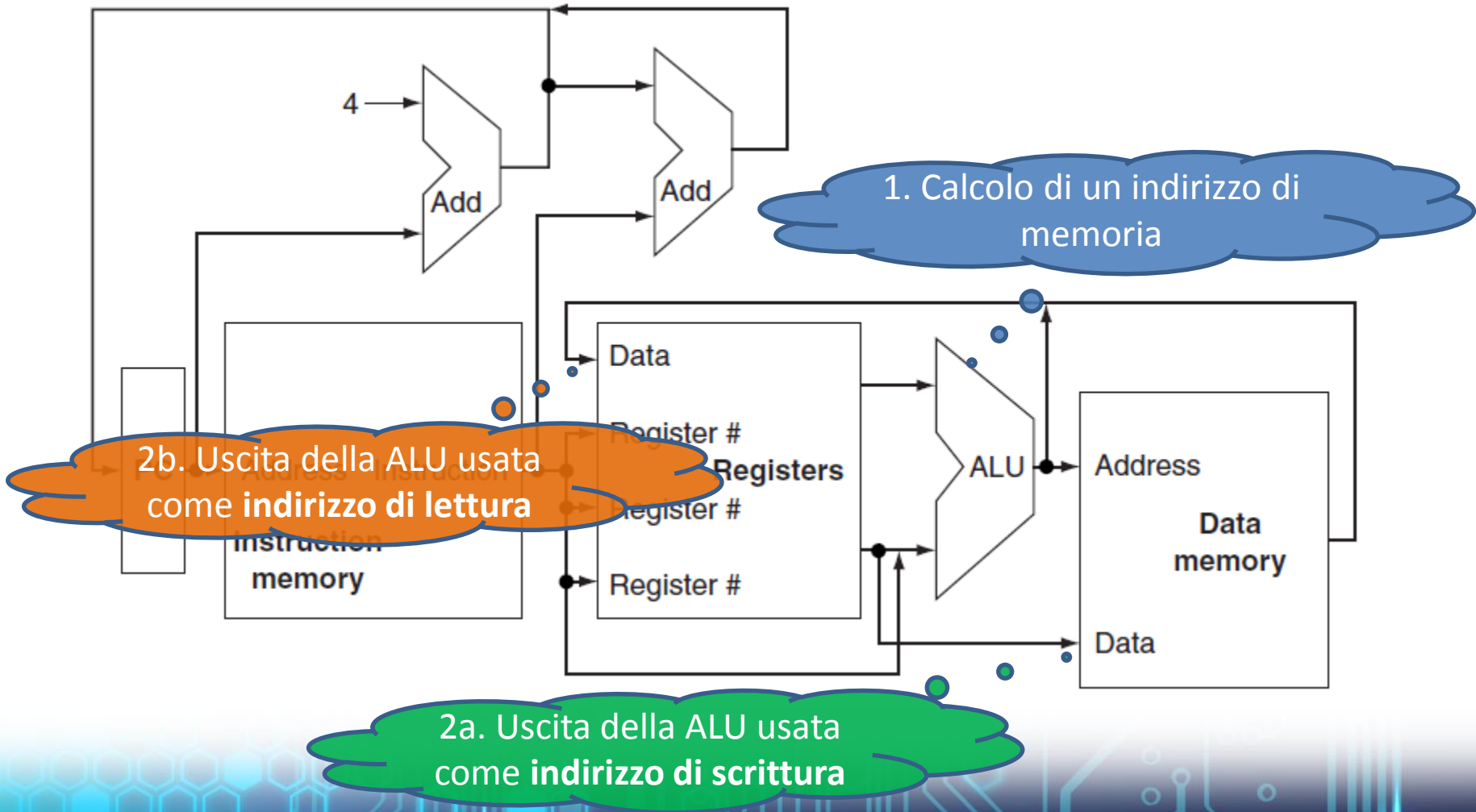
Vista Astratta di un Microprocessore MIPS a Singolo Core

Istruzioni aritmetico-logiche



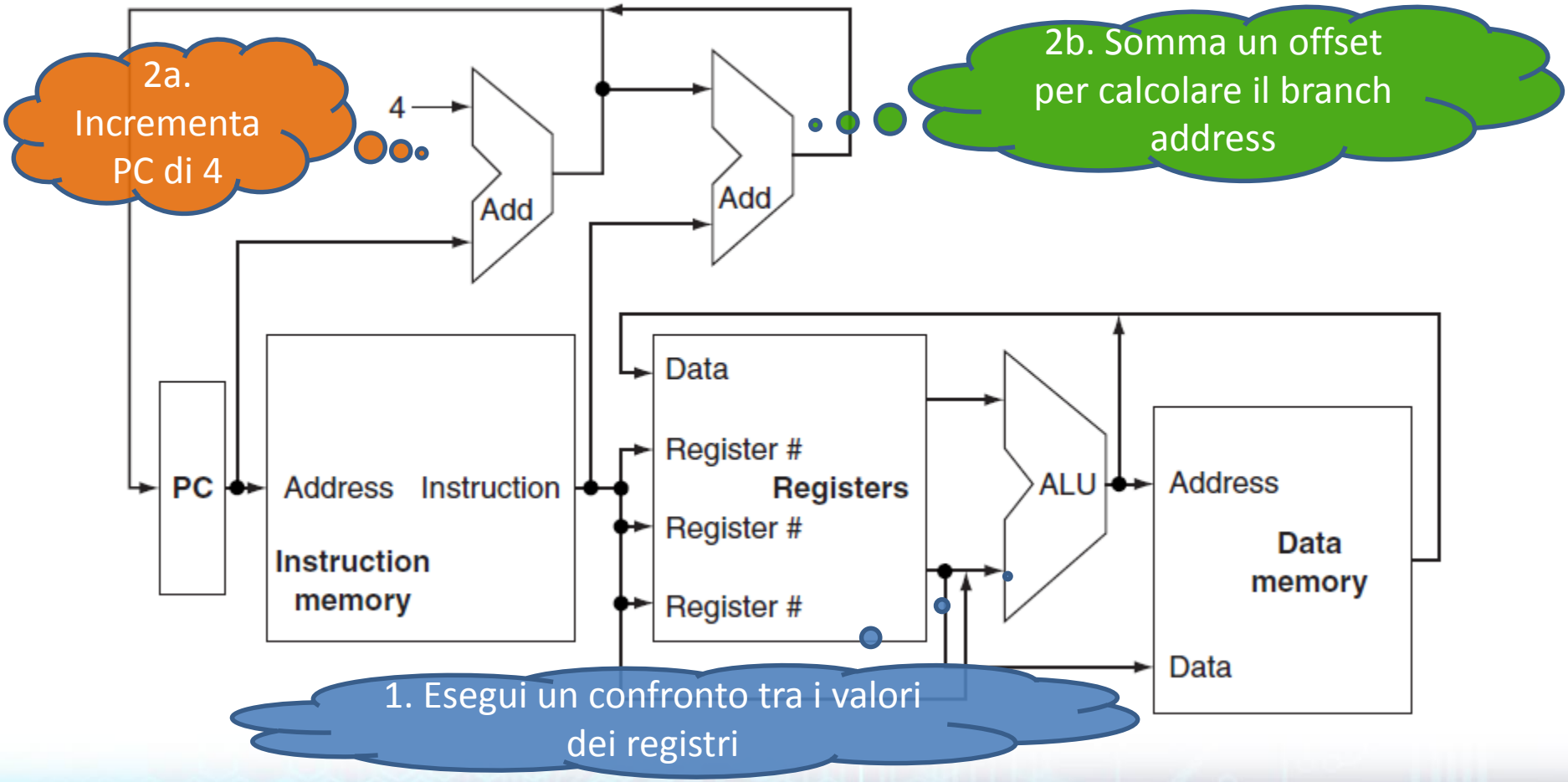
Vista Astratta di un Microprocessore MIPS a Singolo Core

Istruzioni di load/store



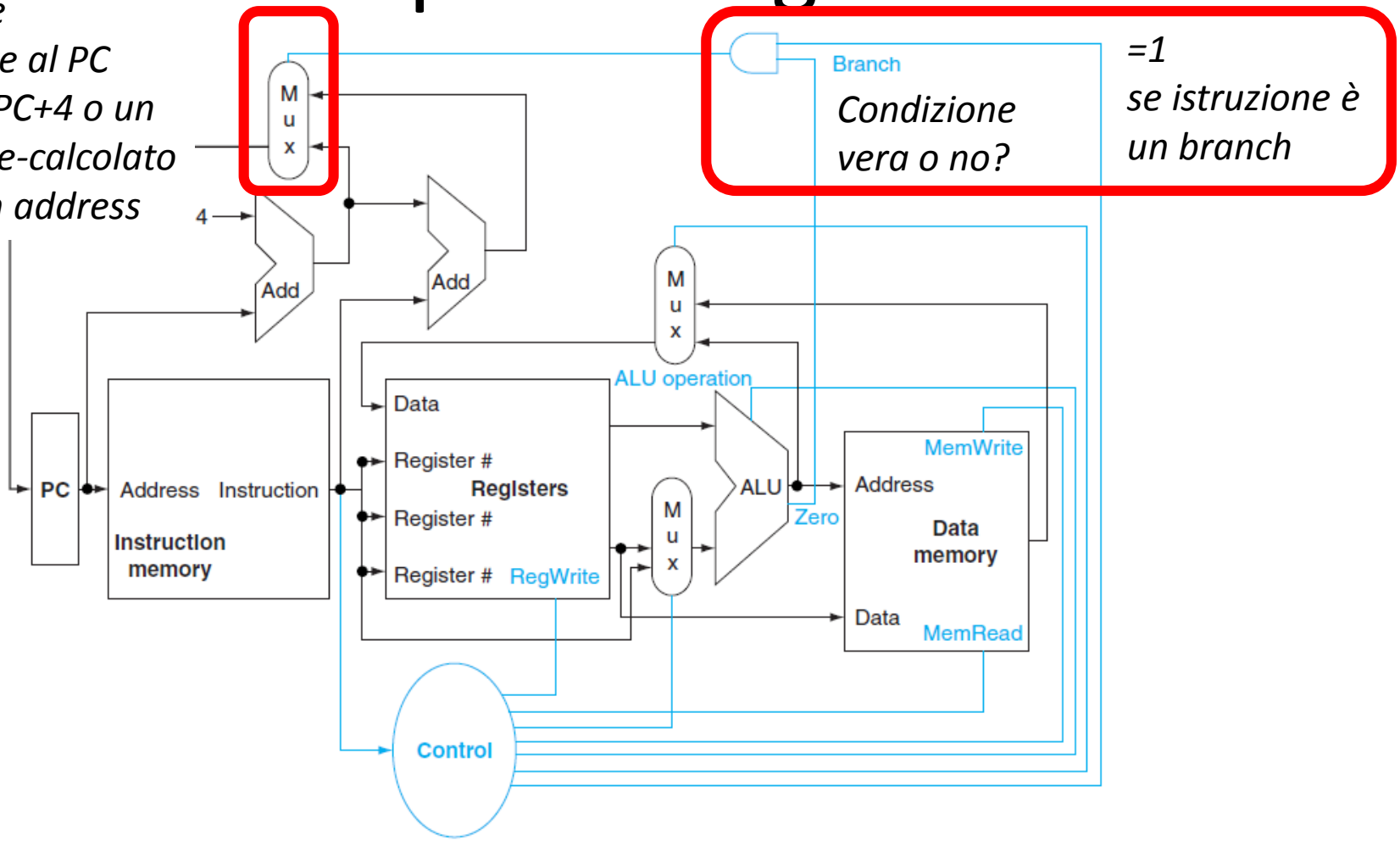
Vista Astratta di un Microprocessore MIPS a Singolo Core

Istruzioni di «branch»

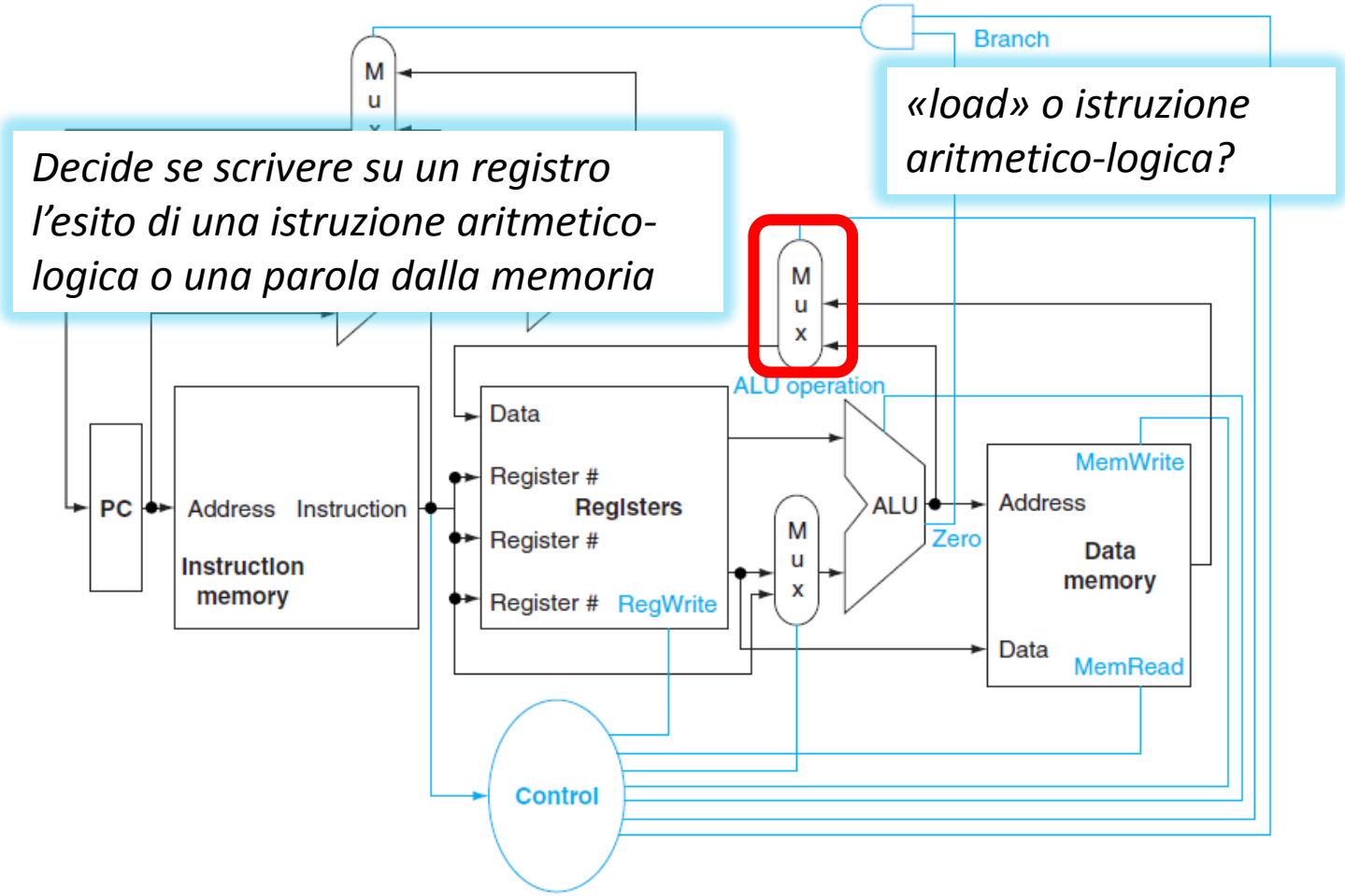


Vista più dettagliata

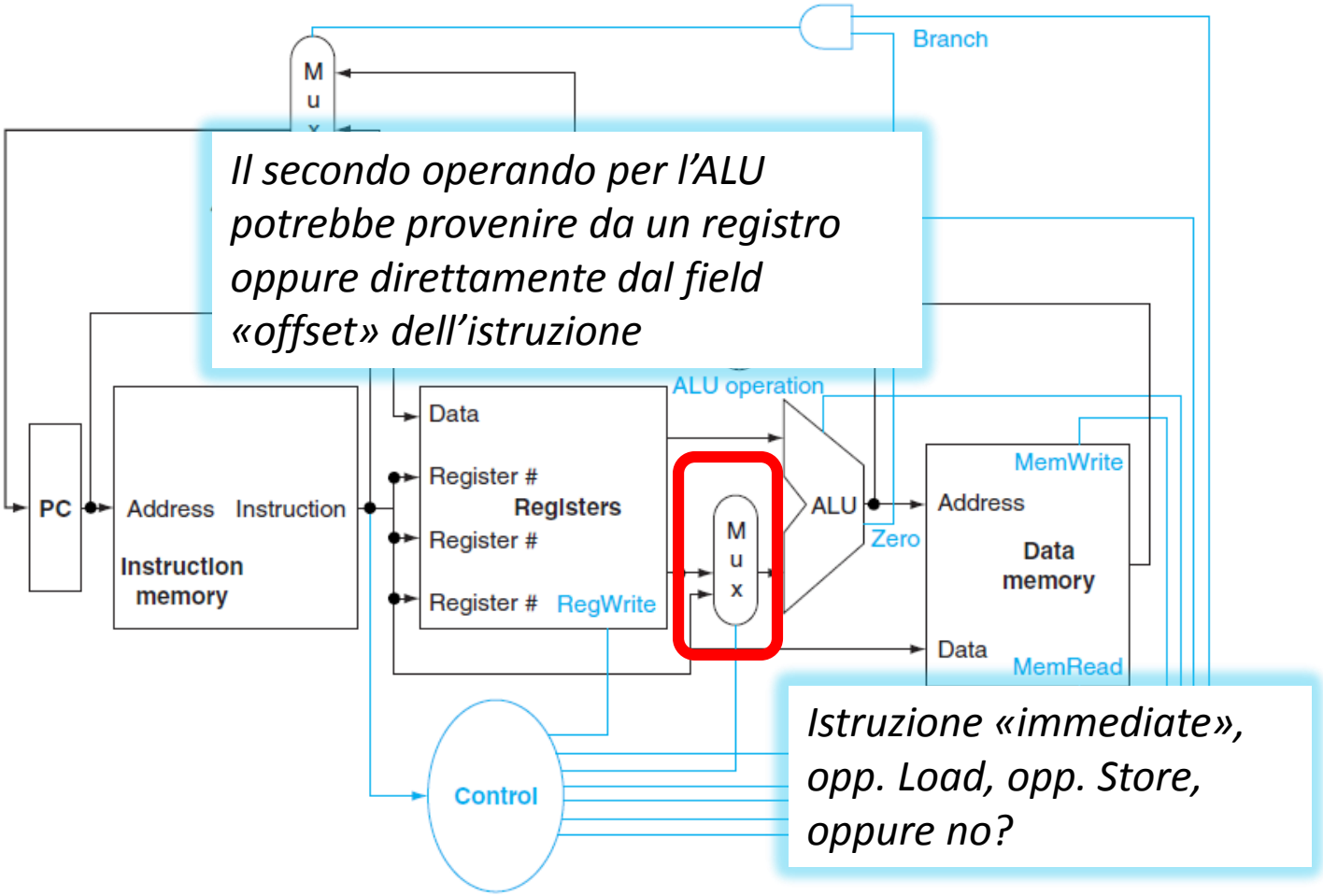
Decide se assegnare al PC Il valore PC+4 o un valore pre-calcolato di branch address



Vista più dettagliata



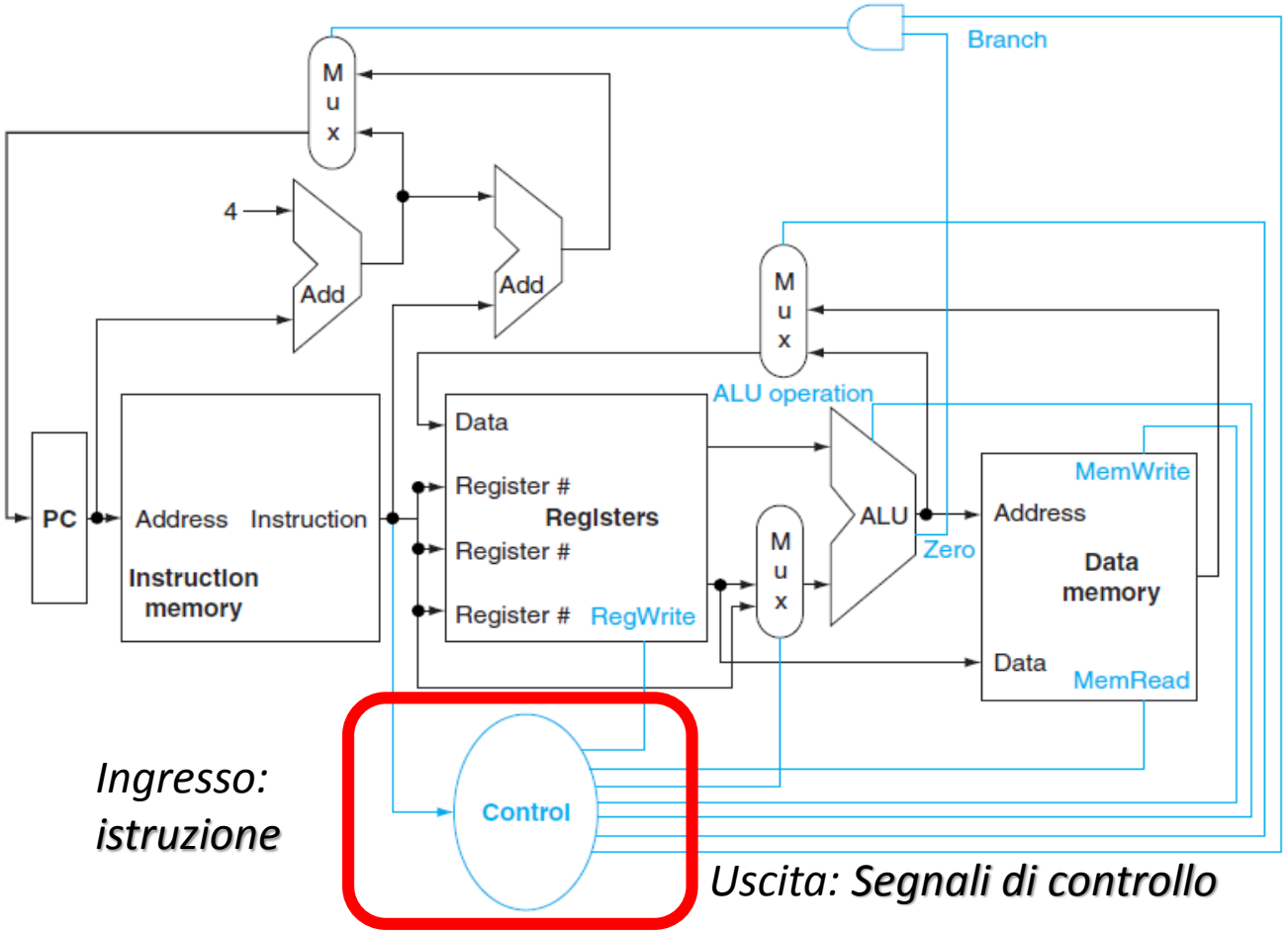
Vista più dettagliata



Il secondo operando per l'ALU potrebbe provenire da un registro oppure direttamente dal field «offset» dell'istruzione

Istruzione «immediate», opp. Load, opp. Store, oppure no?

Vista più dettagliata



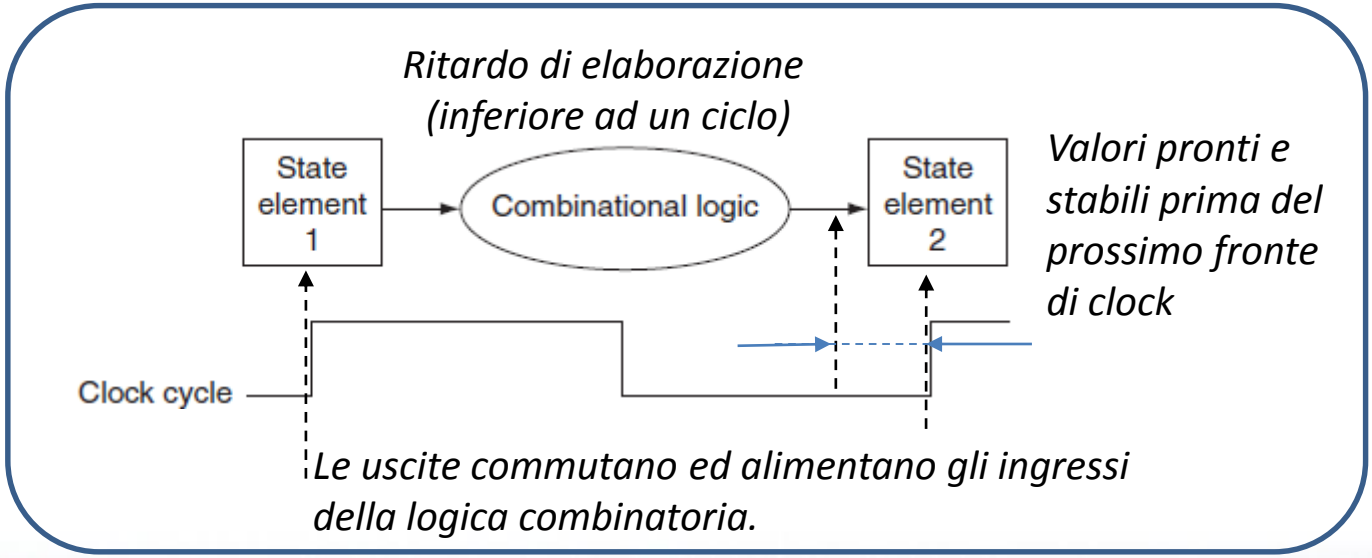
Ingresso: istruzione

Uscita: Segnali di controllo

La «logica di controllo» configura il **tipo di operazione** che i blocchi funzionali devono svolgere (es., read o write, ALU operation), e i **segnali di controllo dei multiplexers**.

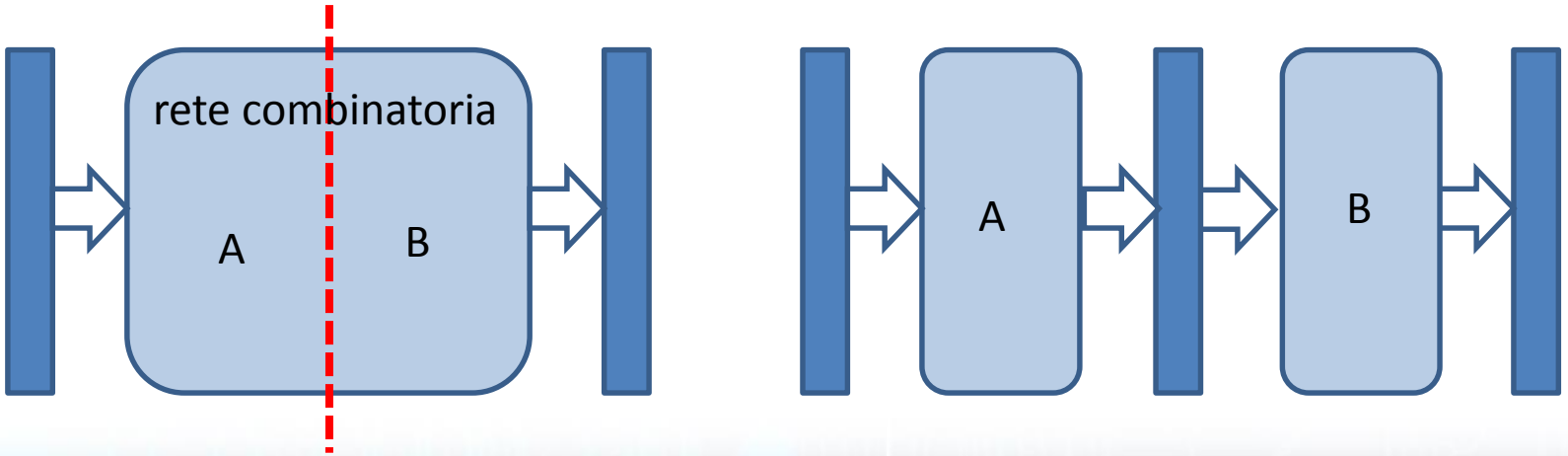
Metodologia di clocking

- Siccome solo i circuiti con memoria possono memorizzare valori, è necessario che elementi di memoria siano piazzati prima (per fornire ingressi stabili) e dopo (per conservare i risultati) la logica combinatoria
- Il massimo ritardo di propagazione del segnale attraverso la logica combinatoria determina il minimo periodo di clock
- Es.: ritardo massimo combinatorio di 0.9ns => Freq. di clock: 1/(0.9+0.1)ns=1 GHz (dove 0.1ns è il tempo di setup più quello di risposta dei FF)



Pipelining

- Spezzando la logica combinatoria in stadi di pipeline è possibile aumentare la frequenza di clock
- Dal punto di vista del tempo impiegato per completare un operazione non ci sono vantaggi
- Il vero vantaggio dipende dal fatto che i due blocchi combinatori sono disaccoppiati e il primo blocco può iniziare a lavorare su un nuovo dato mentre il secondo completa il dato precedente

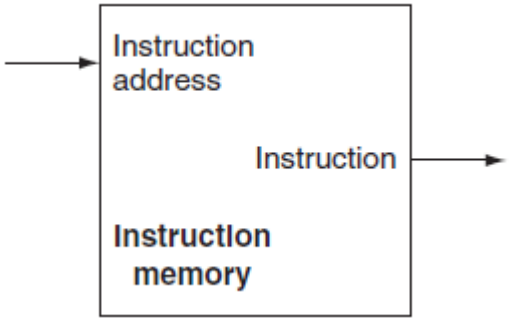


Costruzione del Datapath

- Fetch progressivo delle istruzioni

Ingredienti:

Memoria istruzioni

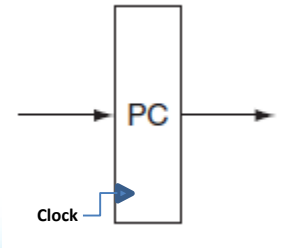


Indirizzo	Contenuto
0	10110..
4	11010..
8	10000..
12	00100..
16	11001..

Dato un indirizzo, la memoria ritorna l'istruzione desiderata

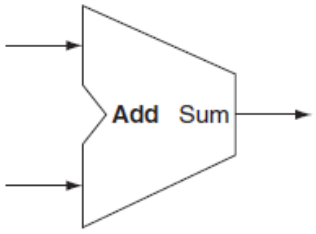
E' read-only, fatta eccezione per il LOADER

Program Counter



E' un registro che contiene **l'indirizzo dell'istruzione attualmente in esecuzione**

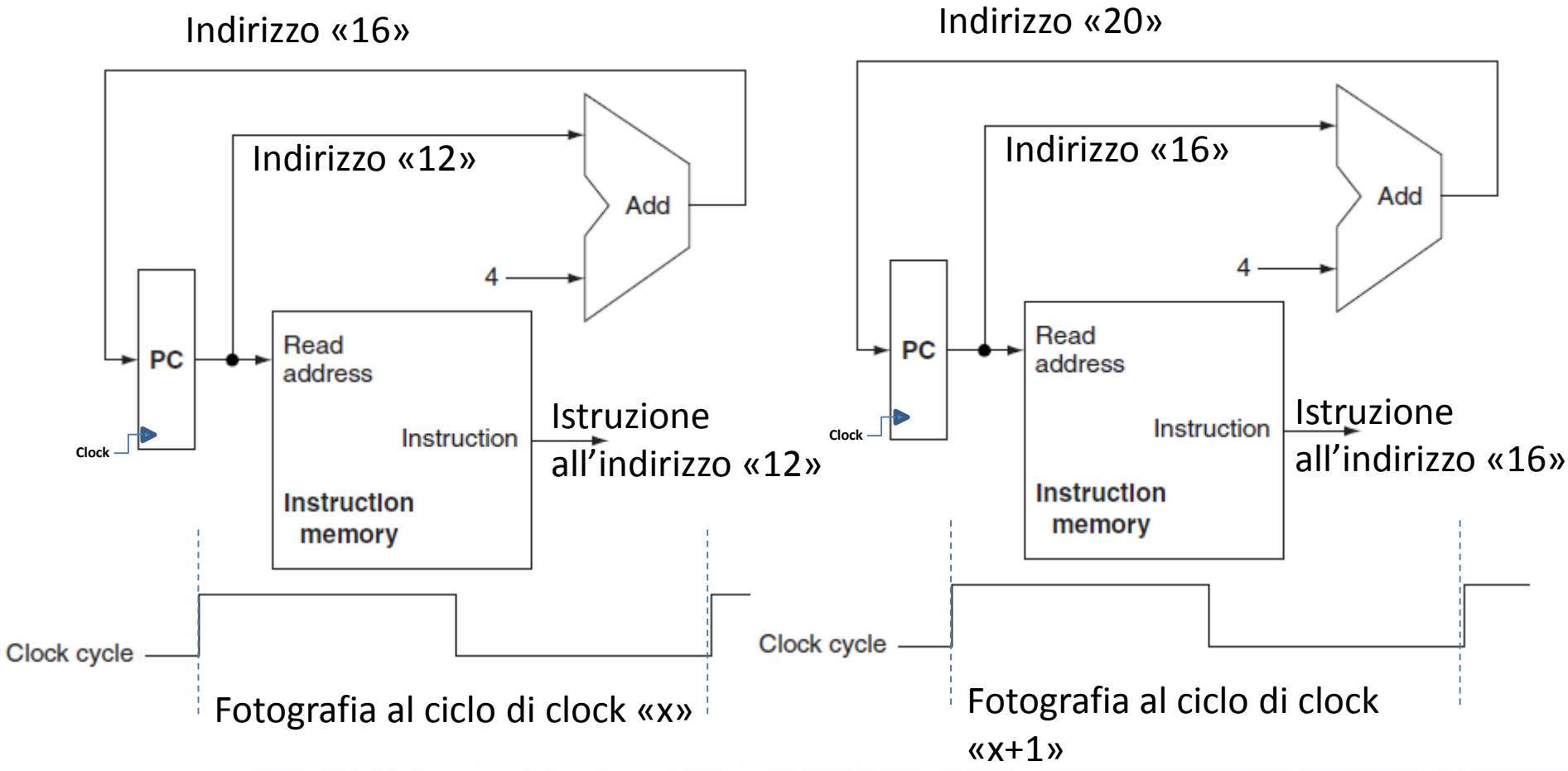
Adder



Deve sommare «4» al PC per ottenere l'indirizzo dell'istruzione successiva

Costruzione del Datapath

- Fetch progressivo delle istruzioni



Le operazioni devono finire in tempo, prima del fronte di clock successivo

Costruzione del Datapath

- Implementazione di istruzioni di tipo R

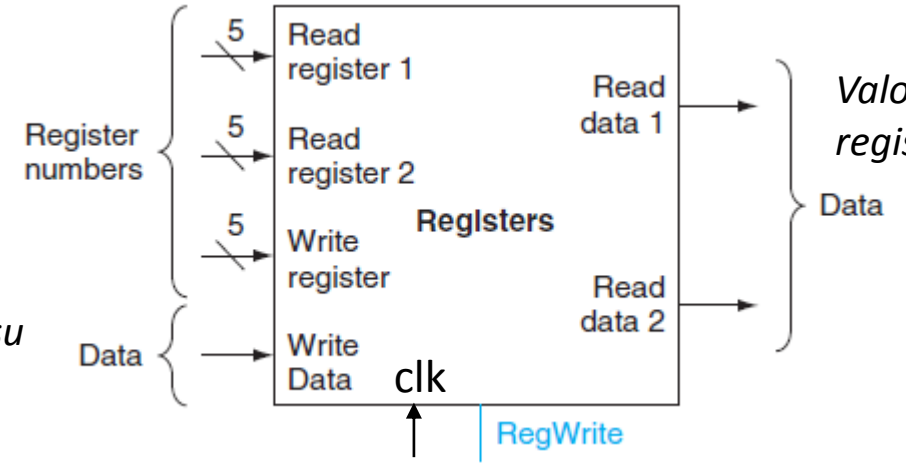
In generale, si legge da due registri, e si scrive in un terzo.

Add $t1, $t2, $t3 # leggi $t2 e $t3, sommali, e scrivi il risultato in $t1.
Anche **sub, or, and, slt,** ….

Ingredienti: **Registri. Essi sono «raggruppati» in una struttura ad-hoc multi-porta chiamata «register file».**

Indirizzi dei registri di cui leggere e/o scrivere il contenuto (5 bit => 32 registri)

Dato da scrivere su un registro



Valori letti dai registri specificati

Segnale di controllo per ordinare la scrittura di un registro sul prossimo fronte di clock. La lettura si suppone essere di «default».

Costruzione del Datapath

- Implementazione di istruzioni di tipo R

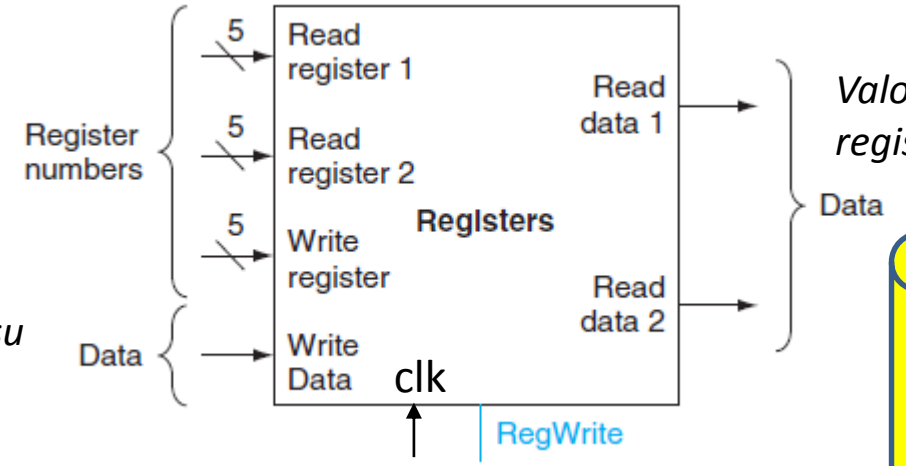
In generale, si legge da due registri, e si scrive in un terzo.

Add $t1, $t2, $t3 # leggi $t2 e $t3, sommali, e scrivi il risultato in $t1.
Anche **sub, or, and, slt,** ….

Ingredienti: **Registri. Essi sono «raggruppati» in una struttura ad-hoc multi-porta chiamata «register file».**

Numero ID dei registri di cui leggere e/o scrivere il contenuto (5 bit => 32 registri)

Dato da scrivere su un registro

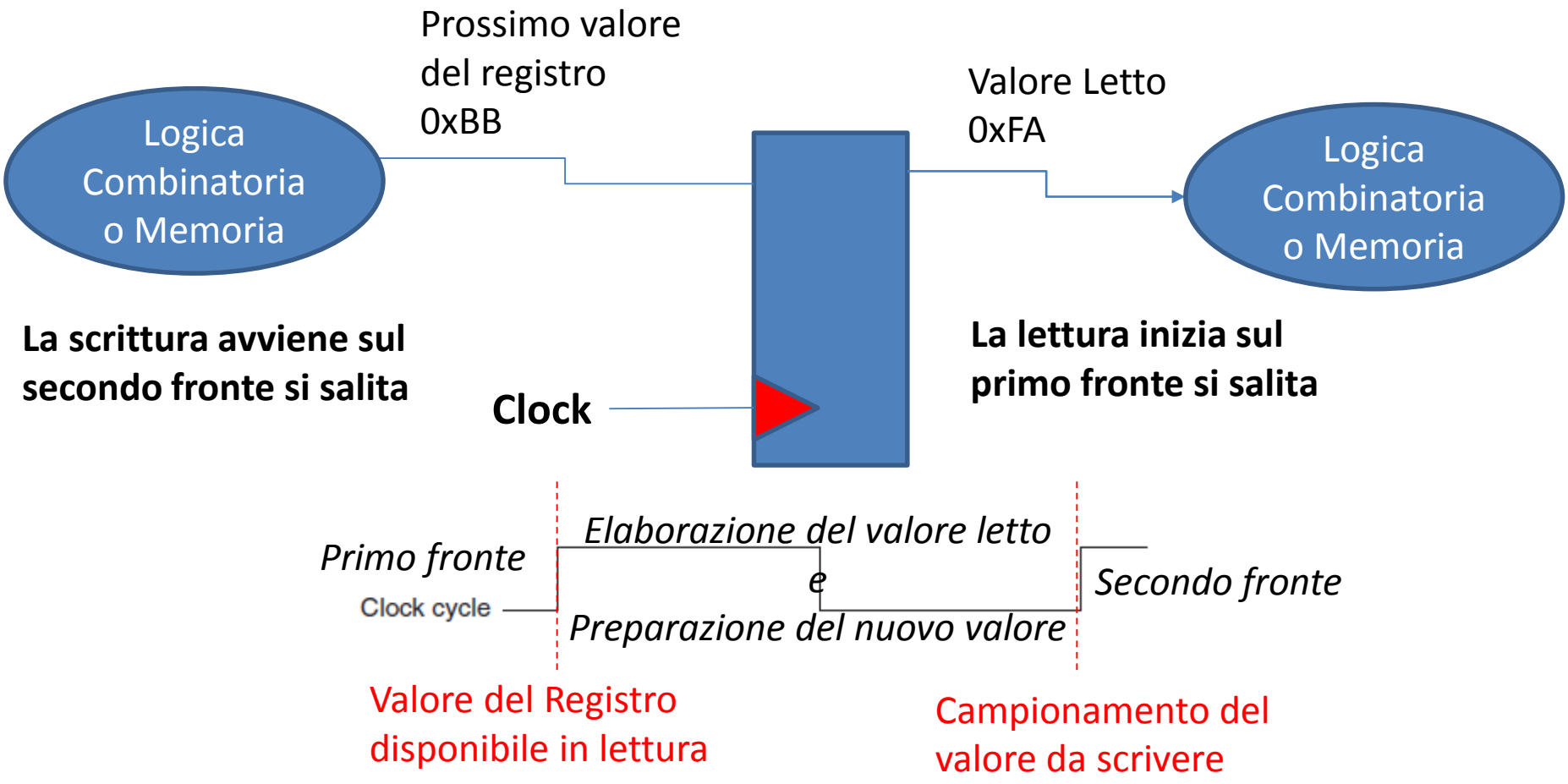


Valori letti dai registri specificati

Segnale di controllo per ordinare la scrittura di un registro sul prossimo fronte di clock. La lettura si suppone essere di «default».

E' possibile leggere e scrivere lo stesso registro entro lo stesso ciclo di clock???

Accesso ad un Registro



Nello stesso periodo di clock è possibile sia scrivere 0xBB sia leggere 0xFA

Costruzione del Datapath

- Implementazione di istruzioni di tipo R

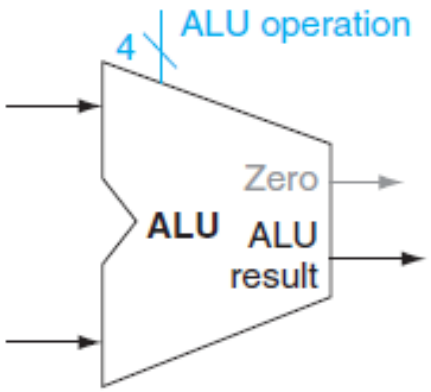
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

In generale, si legge da due registri, e si scrive in un terzo.

Add $t1, $t2, $t3 # leggi $t2 e $t3, sommali, e scrivi il risultato in $t1.
Anche **sub, or, and, slt,** ….

Ingredienti: **ALU**

Selezione del tipo di operazione



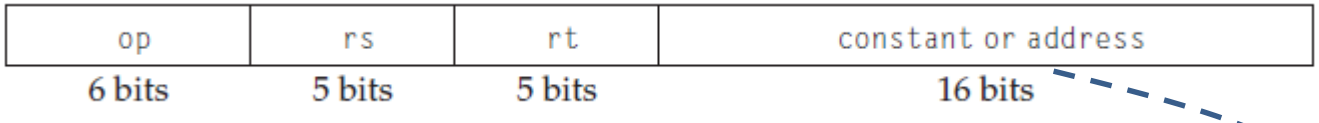
Due registri in ingresso

Singolo bit che indica se il risultato è zero.

Risultato

Costruzione del Datapath

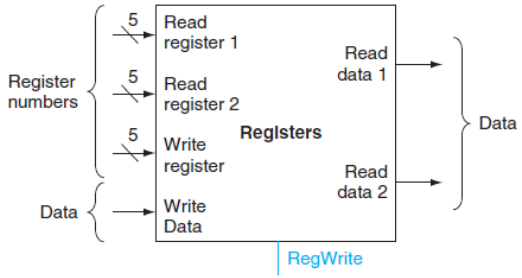
- Implementazione di istruzioni LOAD/STORE



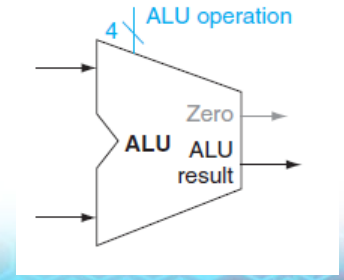
lw $t1, offset_value($t2)
sw $t1, offset_value($t2)

Ingredienti:

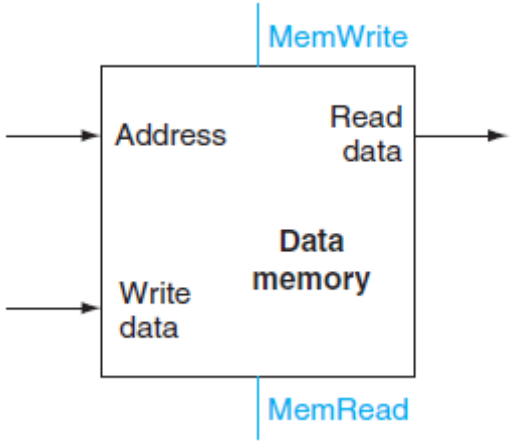
Register File



ALU

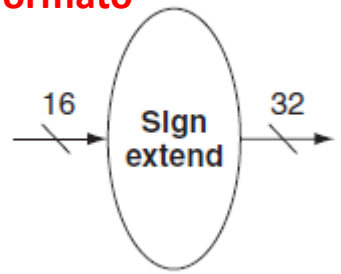


Memoria Dati



*Segnali di controllo per discriminare tra **load** e **store***

Unità per la conversione di formato



Converte un offset a 16 bit in un numero a 32 bit

Costruzione del Datapath

- Implementazione di istruzioni tipo R e di load/store
- Esiste una somiglianza fra queste classi di istruzioni

Istruzioni R *Istruzioni LW/SW*

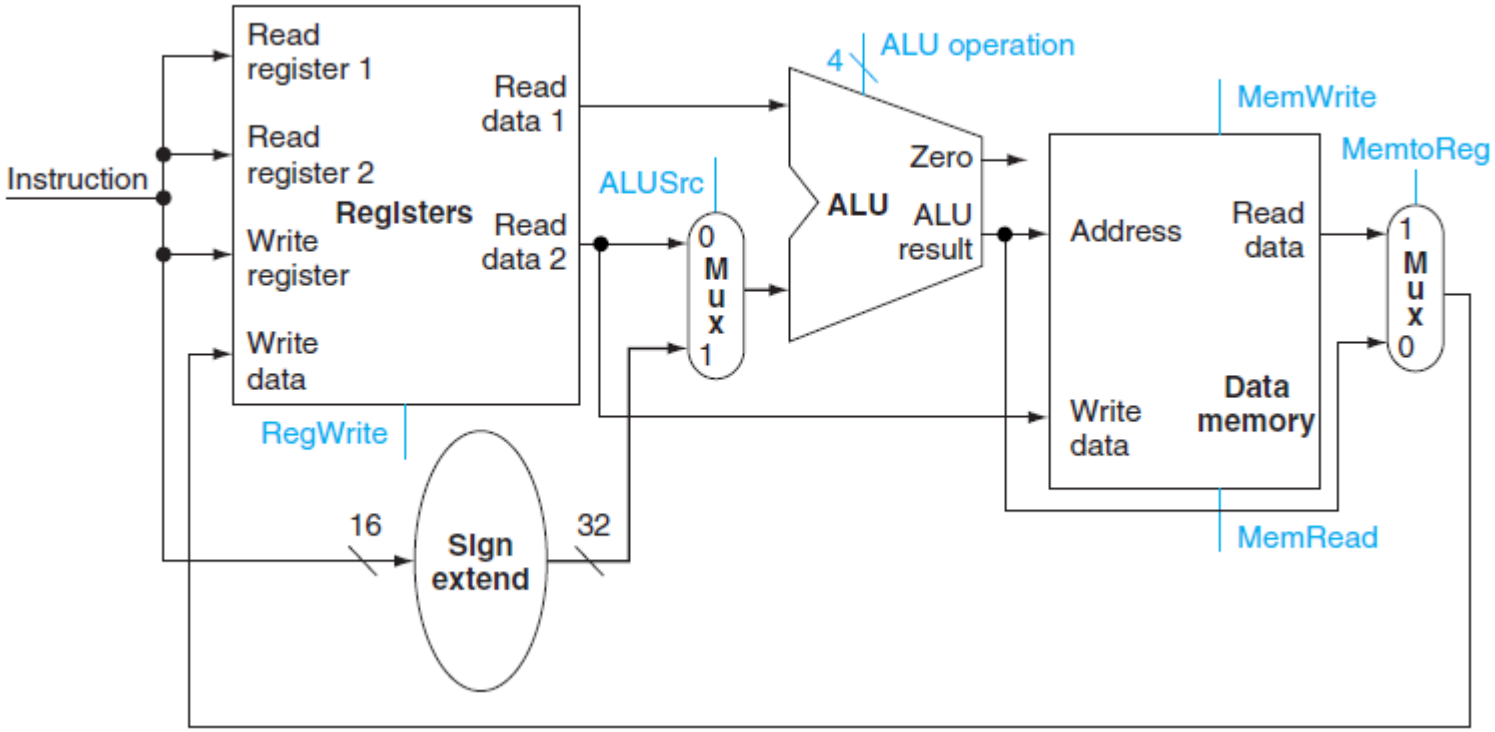
A. Le istruzioni aritmetico-logiche usano l'ALU con ingressi provenienti da 2 registri.
B. Load/Store usano l'ALU con un ingresso da registro ed un ingresso dall'offset nell'istruzione

A. Le istruzioni aritmetico-logiche scrivono il risultato dell'ALU su registro destinazione.
B. Le Load scrivono un dato letto in memoria sul registro destinazione.

Sfida progettuale:
Condividere il register file e l'ALU tra questi due tipi di operazioni!

Costruzione del Datapath

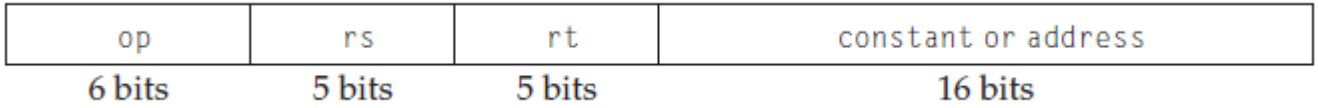
- Implementazione di istruzioni tipo R e di load/store



La condivisione delle risorse introduce la necessità di MUX e rispettivi segnali di controllo

Costruzione del Datapath

- Implementazione di salti condizionali



beq $t1, $t2, offset # se $t1 = $t2, salta all'indirizzo «branch target address»
branch target address = indirizzo base + offset

- **La BASE per calcolare «branch target address» è l'indirizzo dell'istruzione successiva al branch!**
  - Nel «fetch datapath», viene calcolato PC+4 (indirizzo della prossima istruzione), che dunque viene fornito «gratis» al «branch datapath».
- **L'offset dell'istruzione si riferisce alle parole, non ai byte**, per aumentare la «branching distance» potenziale.
- Prossimo valore del PC:

Condizione è vera?

BRANCH TAKEN.
PC_new = Branch target address

BRANCH NOT TAKEN.
PC_new = PC_current + 4

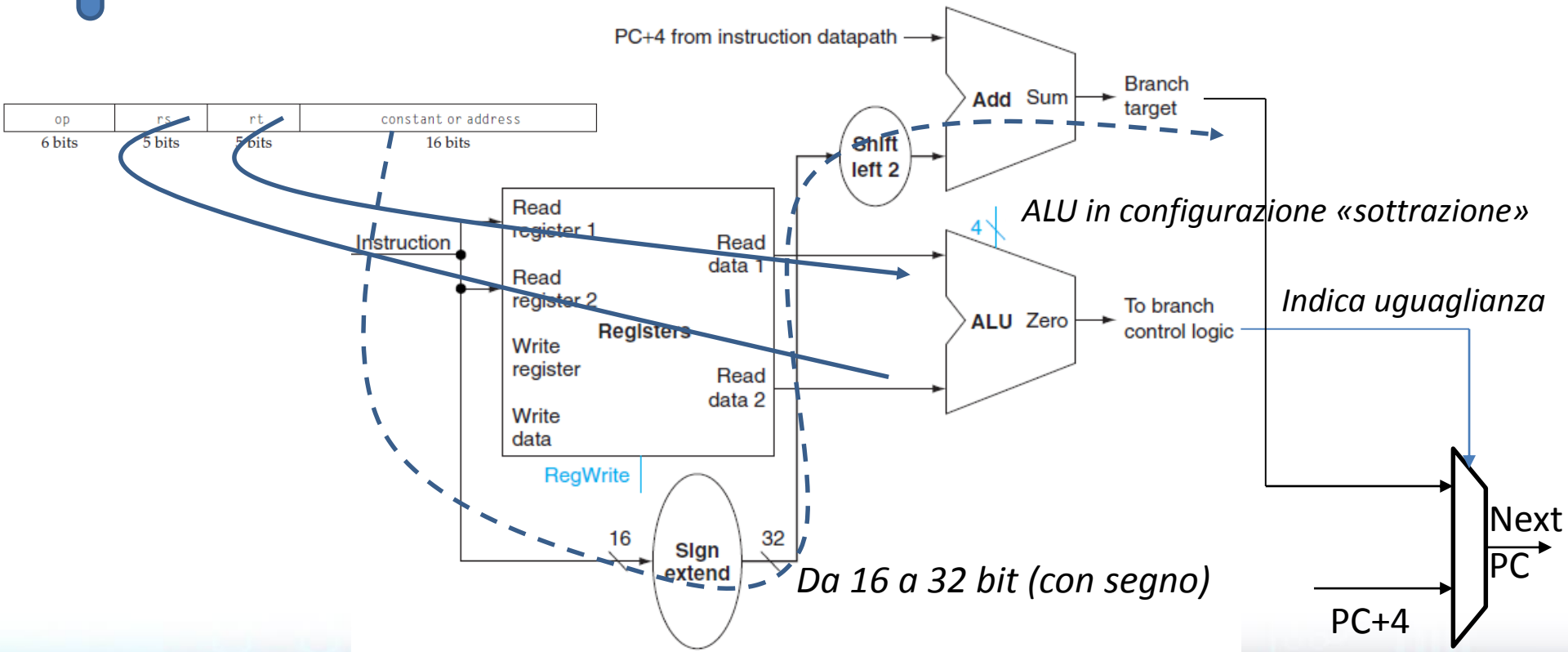
Costruzione del Datapath

beq $t1, $t2, offset # se $t1 = $t2, salta all'indirizzo «branch target address»

Eseguire un branch implica due operazioni fondamentali:

A. **Calcolo del branch target address**
B. **Confronto del contenuto di due registri**

..e le due cose in parallelo!

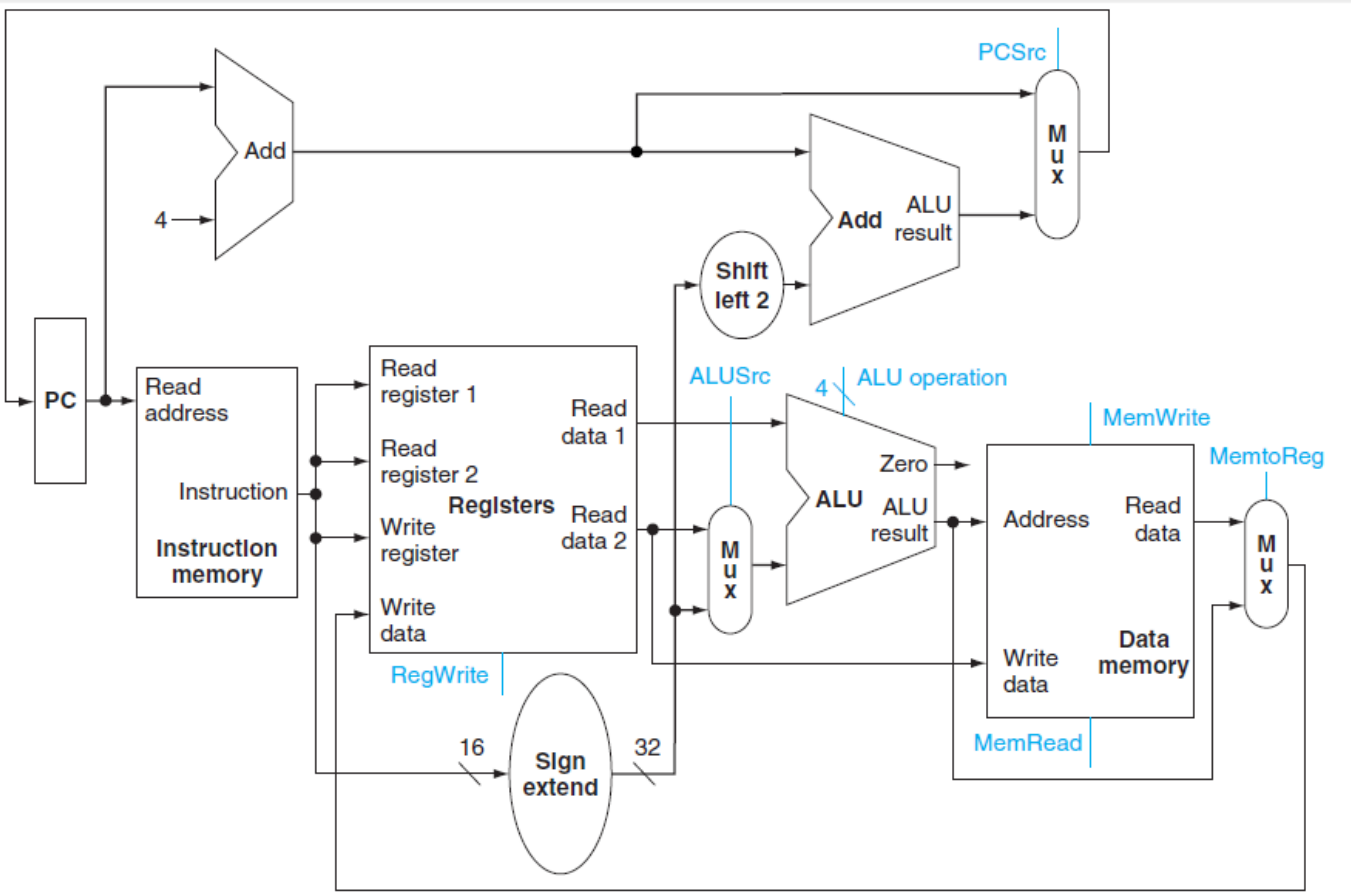


Costruzione del Datapath

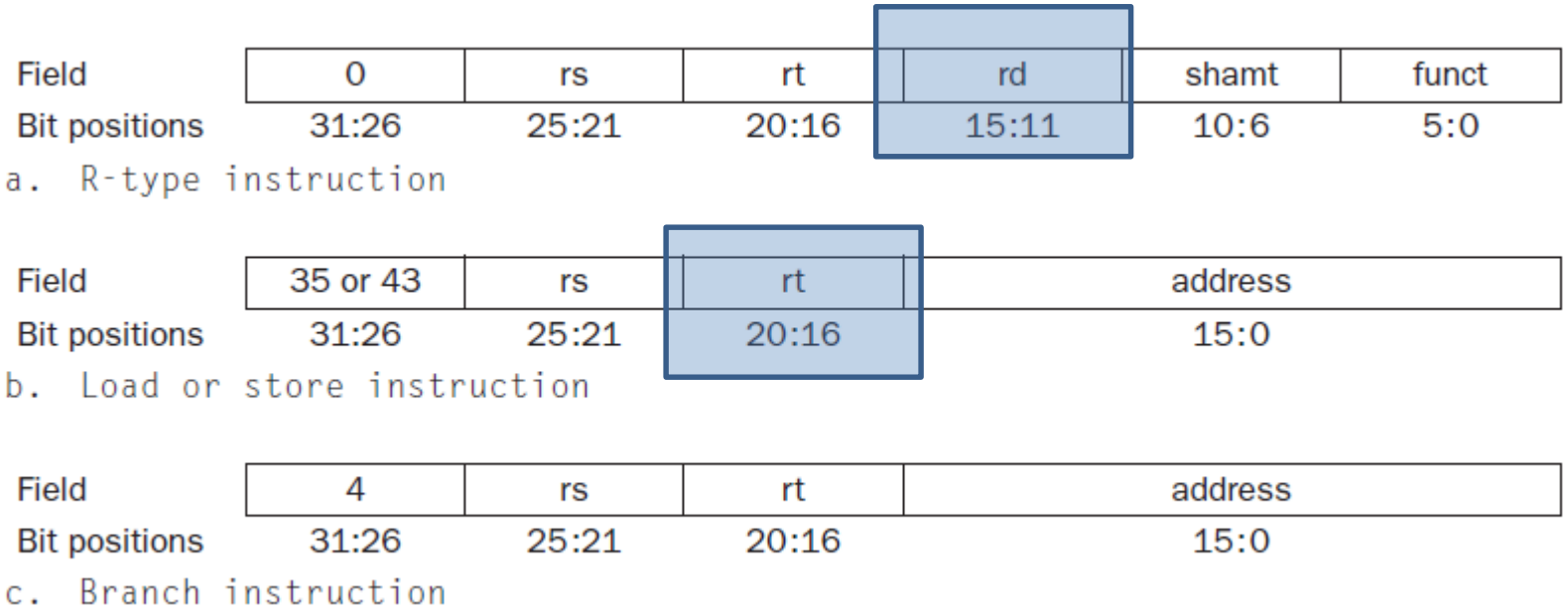
- Mettiamo tutto insieme con queste ipotesi semplificative
  - **Esecuzione di tutte le istruzioni in un solo ciclo di clock!**
    - ✓ **CPI = 1 (cicli di clock per istruzione)**
  - **Non sfrutto il fatto che non tutte le istruzioni hanno lo stesso tempo di esecuzione effettivo**
    - ✓ **Periodo di clock determinato dal caso peggiore**
    - ✓ **Le operazioni non critiche introducono «idleness»**

Costruzione del Datapath

- Mettiamo tutto insieme!

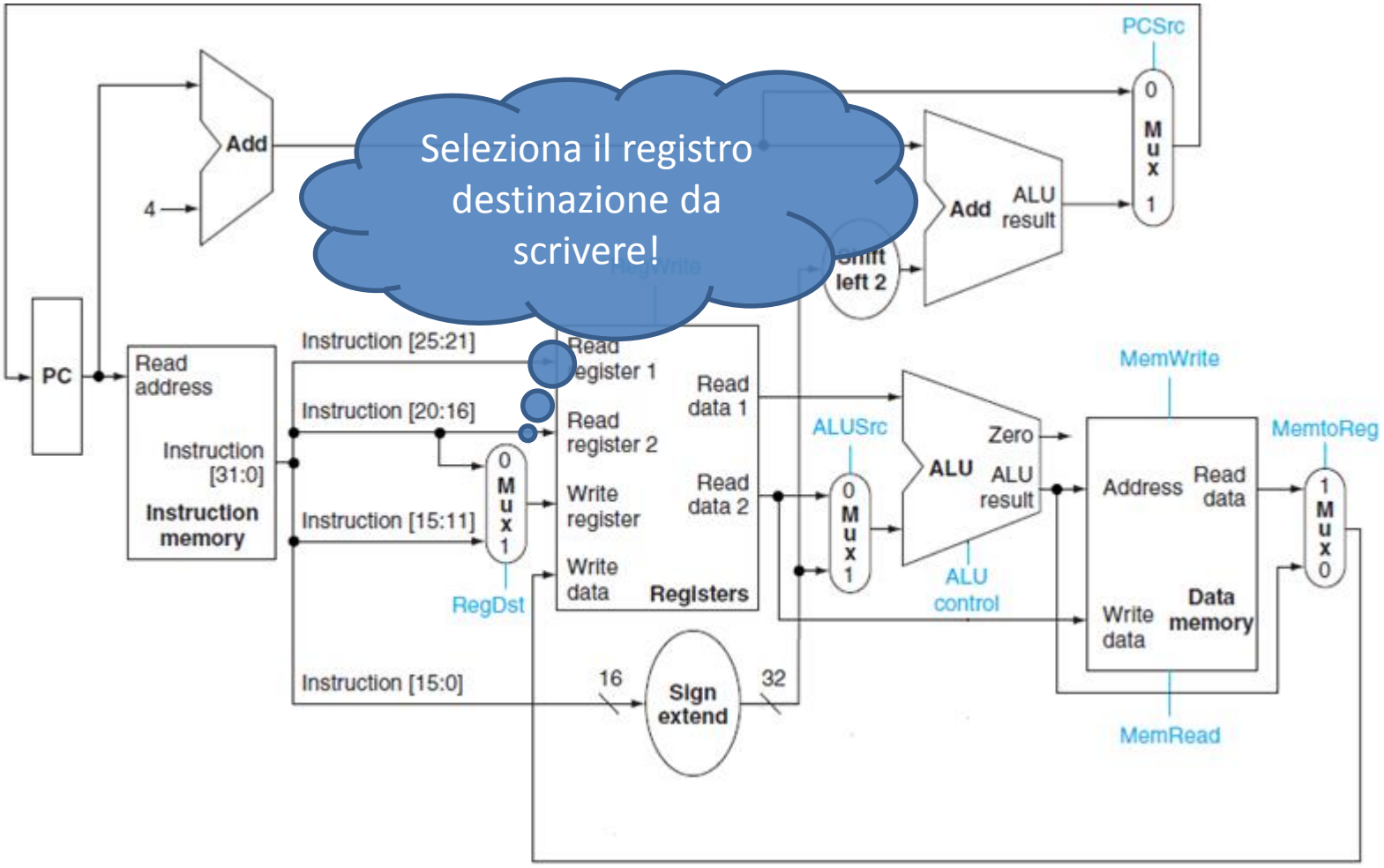


Manca un Multiplexer



Il registro destinazione dipende dalla classe di istruzione. Ciò renderà necessario un multiplexer per selezionare il registro destinazione per la istruzione in esecuzione.

Aggiunta dei Multiplexer



Come controlliamo la ALU?