
# Architettura degli Elaboratori I

Corso di Laurea Triennale in Informatica

Università degli Studi di Milano

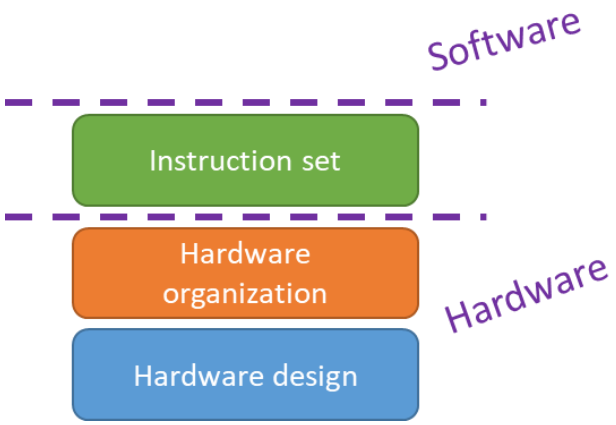
Dipartimento di Informatica “Giovanni Degli Antoni”

Edizione 2 (Cognomi H-Z), A.A. 2022-2023, Nicola.Basilico@unimi.it

# CPU a singolo ciclo

# CPU a singolo ciclo

- **Obiettivo**: costruire una CPU in grado di eseguire le istruzioni dell'ISA MIPS
- Costruiremo una CPU a **singolo ciclo**: esegue **1 istruzione per ciclo di clock**

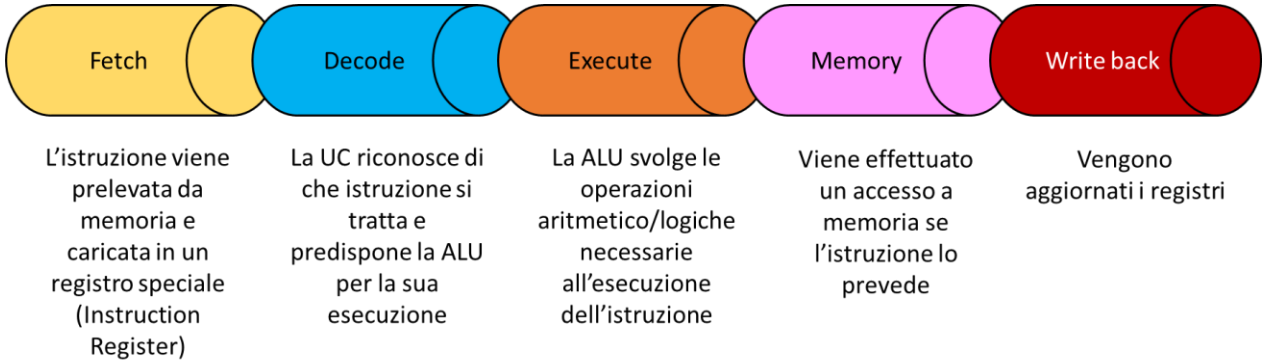



**Che cosa è un'istruzione?** Siamo direttamente sull'hardware!

- Un'istruzione è una **stringa binaria** che codifica le informazioni necessarie all'esecuzione, da parte della CPU, di una particolare operazione
- La CPU è un circuito in grado di **decodificare** tale stringa binaria ed **eseguire** l'elaborazione che essa rappresenta
- Il linguaggio binario con cui sono espresse le istruzioni si chiama **linguaggio macchina**

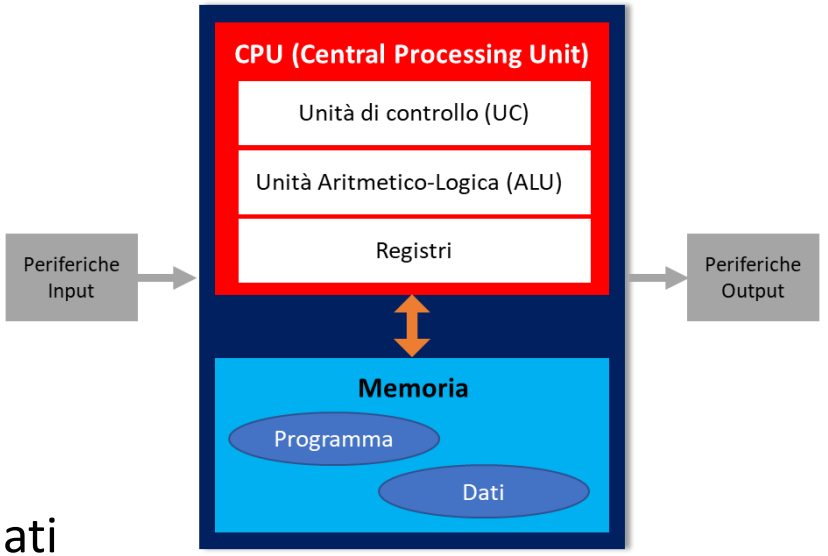
# Macro-elementi della CPU

- Abbiamo già visto che l'esecuzione di una istruzione passa per 5 fasi che la CPU svolte in modo iterativo




- Come è fatto il circuito della CPU in grado di svolgere queste operazioni? Esistono **due macro-elementi fondamentali**
- **Data Path**: è il percorso che le informazioni seguono all'interno della CPU attraversando i suoi sotto-componenti, può variare a seconda dell'istruzione
- **Logica di Controllo** (Unità di controllo)
  - Manovra gli «scambi» del data path in modo che le informazioni seguano un percorso diverso a seconda dell'istruzione
  - Controlla i sotto-componenti della CPU a seconda dell'istruzione (ad es. decidendo i segnali di controllo della ALU)

# Istruzioni supportate

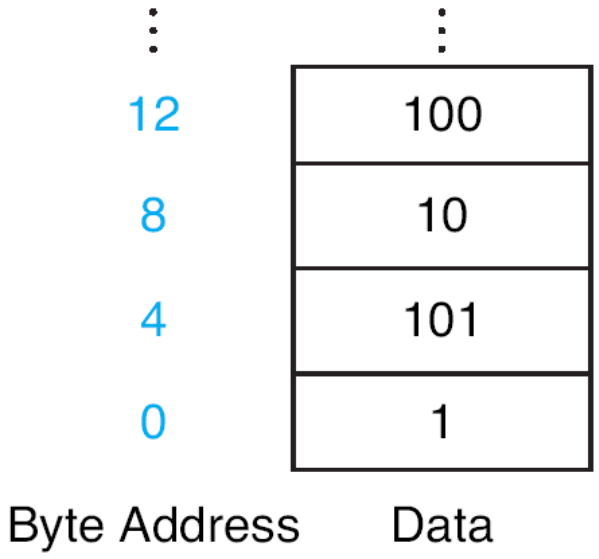



- Tipi di istruzioni che la nostra CPU sarà in grado di svolgere:
  - Istruzioni Aritmetico-Logiche: come ad esempio add o AND
  - Istruzioni di accesso alla memoria dati: load e store
  - Istruzioni di controllo di flusso: salti condizionati e non condizionati

- Ogni istruzione è codificata in linguaggio macchina su 32 bit, organizzati secondo un particolare formato basato su una suddivisione in gruppi detti **campi**
- In MIPS esistono tre formati
  - **Formato R** (Register): per le istruzioni aritmetico-logiche che operano su valori in registri
  - **Formato I** (Immediate): per le istruzioni aritmetico-logiche che operano su valori immediati (costanti) e per le istruzioni di salto condizionato
  - **Formato J** (Jump): per le istruzioni di salto **non** condizionato

# Ingredienti di una CPU: la Memoria

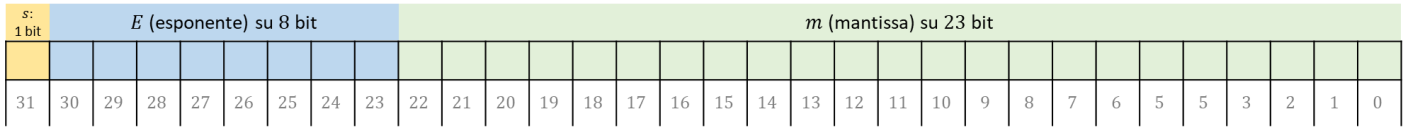
- La memoria è intrepretata come un array unidimensionale dove ogni elemento si chiama **parola di memoria**
- Se ogni parola è di $n$ bit (ampiezza) la memoria può contenere $2^n$ parole (altezza)
- Ad esempio con $n = 32$ ($4$ byte) si hanno $2^{32}$ parole, circa $4\text{GB}$ di spazio
- La parola è l'unità base di trasferimento da e verso la memoria, si definisce in byte (gruppi di 8 bit)
- Ogni parola è associata ad un **indirizzo** su $n$ bit che la «localizza» nella memoria



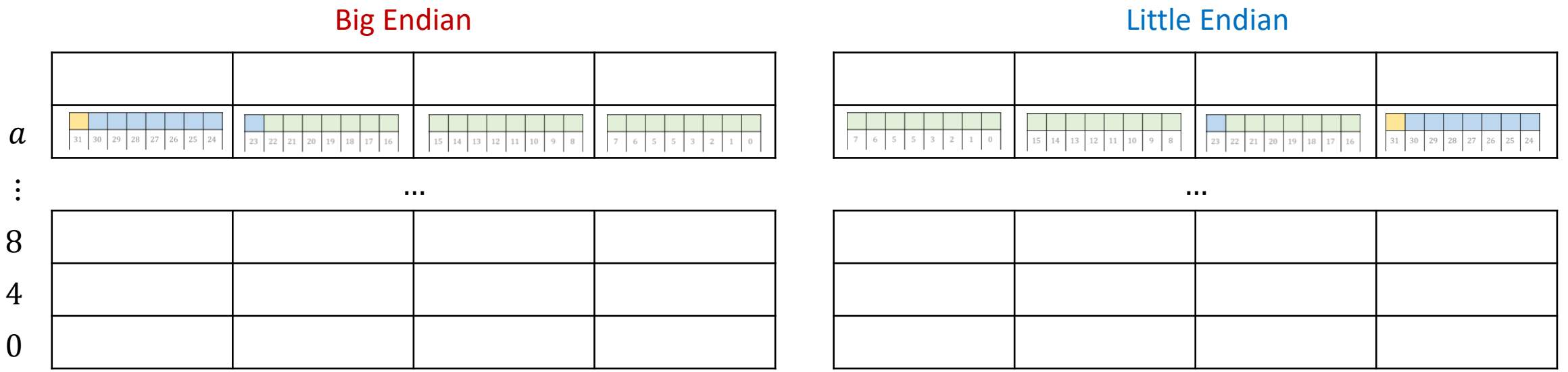

- In MIPS ogni parola è di $4$ byte
- Lo spazio degli indirizzi è definito con risoluzione **al byte**: ogni singolo byte ha un suo indirizzo
- **Allineamento**: l'indirizzo di una parola deve essere multiplo della sua dimensione in byte (e cioè $4$)
- L'indirizzo di una parola è l'indirizzo di uno dei suoi $4$ byte, per questo gli indirizzi di due parole di memoria successive distano $4$
- Quale dei $4$ byte è associato all'indirizzo della parola?
  - Big Endian: il byte più significativo (gli $8$ bit più alti)
  - Little Endian: il byte meno significativo (gli $8$ bit più bassi)

# Big Endian vs Little Endian

- Immaginiamo di dover memorizzare un numero IEEE 754 nella parola di memoria che ha indirizzo pari ad $a$



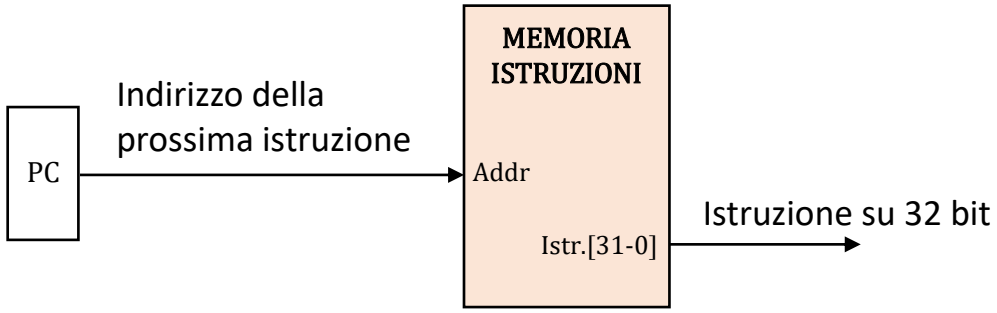

- Il numero viene trasferito in memoria byte per byte da sinistra a destra
  - Il primo byte del numero viene scritto nel byte di indirizzo $a$
  - Il secondo byte del numero viene scritto nel byte di indirizzo $a + 1$ e così via…




- MIPS è Big Endian, la famiglia Intel (ad esempio i386) è Little Endian

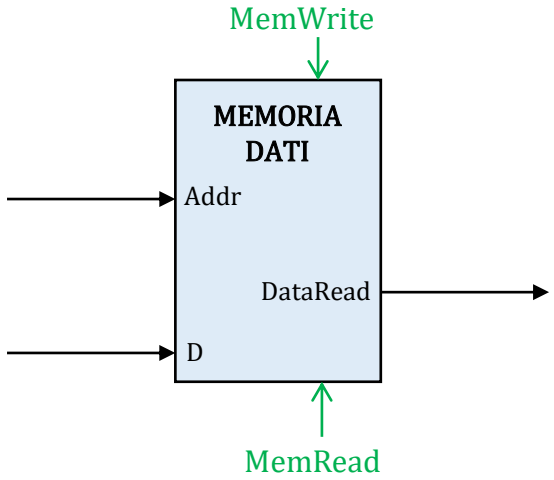
# Ingredienti di una CPU: la Memoria

- Per noi la memoria centrale sarà un modulo a cui possiamo accedere tramite un indirizzo per leggere o scrivere una parola
- Due sezioni di memoria:
  - Memoria istruzioni: ogni parola è un'istruzione per la CPU, la sequenza di parole corrisponde alla sequenza di istruzioni
  - Memoria dati: ogni parola è un dato (ad esempio un valore numerico da usare in un'istruzione aritmetica)
- La memoria istruzioni viene usata solo in lettura, per leggere la prossima istruzione che la CPU deve svolgere
- L'indirizzo della prossima istruzione sta all'interno di un registro speciale chiamato **Program Counter** ($PC$)



# Ingredienti di una CPU: la Memoria

- La memoria dati viene usata sia in lettura (per trasferire una parola dalla memoria alla CPU) che in scrittura (per trasferire una parola dalla CPU alla memoria), ma non nello stesso ciclo di clock
- Input:
  - Addr: indirizzo della parola di memoria (32 bit)
  - MemRead: segnale di controllo (1 bit), 1 per lettura, 0 a riposo
  - MemWrite: segnale di controllo (1 bit), 1 per scrittura, 0 a riposo
  - D: il dato (parola) da trasferire in memoria se siamo in modalità scrittura (32 bit)
- Output
  - DataRead: il dato (parola) recuperato dalla memoria se siamo in modalità lettura (32 bit)



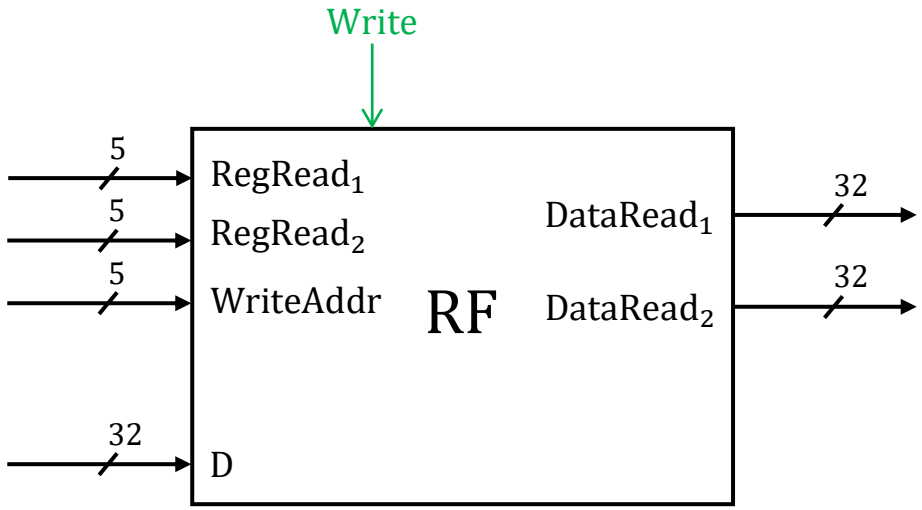

- **Random Access Memory** (RAM): tempo di accesso ad una parola di memoria è fisso, indipendentemente dall'indirizzo a cui si trova una parola
- MemRead e MemWrite: in un ciclo di clock solo uno dei due verrà posto ad 1

# Ingredienti di una CPU: il Register File

- È il «banco di lavoro» della CPU: una memoria interna molto piccola e ad accesso molto rapido, costituita da unità dette **registri**
- La CPU mantiene in questa memoria i dati su cui più frequentemente svolge delle operazioni
- Molte istruzioni operano su valori contenuti dentro registri: anziché indicare il valore di un operando, indicano **il numero** (anche interpretabile come un indirizzo) del registro in cui si trova tale valore
- Le architetture che usano un banco registri sono anche dette **load/store**, perché lavorare sul banco registri implica, tipicamente, di trasferire da memoria (verso il banco, load) degli operandi e, successivamente, di scrivere in memoria (dal banco, store) il risultato di una operazione
- In MIPS un registro ha la stessa dimensione di una parola di memoria (32 bit) seppur questo non sia vero in tutte le architetture
- Il Register File del MIPS contiene 32 registri da 32 bit: per indirizzare un registro, cioè per indicare il suo numero, servono 5 bit

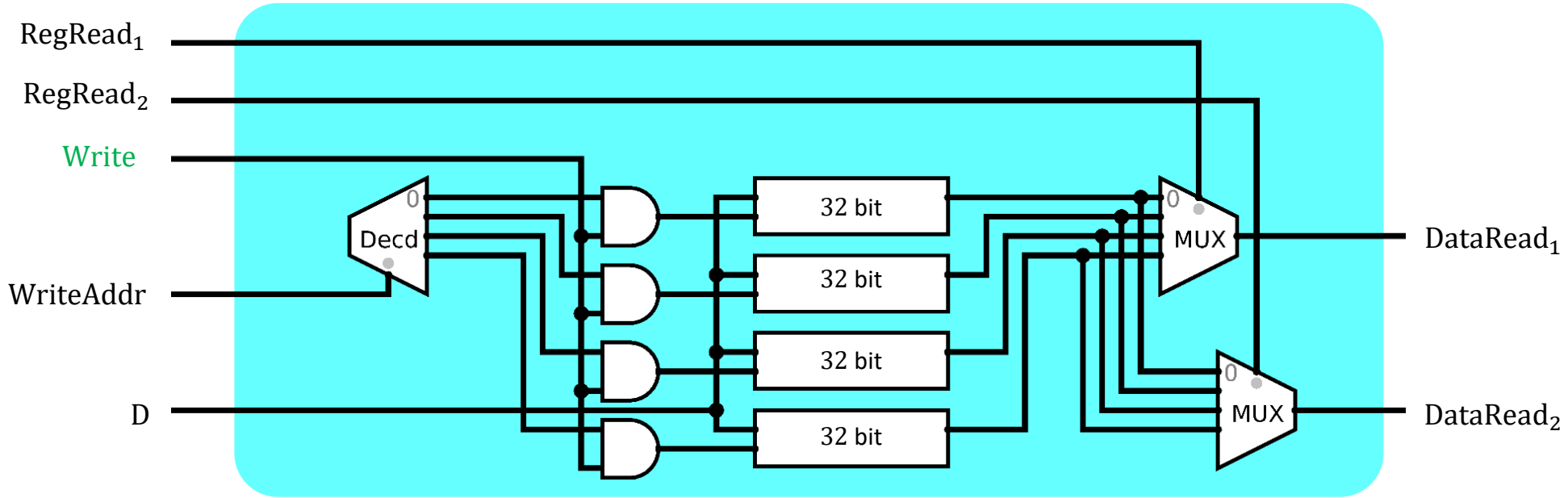
# Ingredienti di una CPU: il Register File

- A differenza della memoria dati, il Register File può essere usato sia in lettura che in scrittura nello stesso ciclo di clock
- Proprietà interessante: si possono leggere due registri contemporaneamente (due uscite di lettura)
- Input
  - Write: segnale di controllo (1 bit) , 1 per scrittura
  - $\mathrm{RegRead}_1$ e $\mathrm{RegRead}_2$: indirizzi (5 bit) dei due registri da leggere
  - WriteAddr: indirizzo (5 bit) del registro in cui scrivere (se in modalità scrittura)
  - D: dato (32 bit) da scrivere nel registro indirizzato daWriteAddr (se in modalità scrittura)
- Output
  - $\mathrm{DataRead}_1$ e $\mathrm{DataRead}_2$: i valori (32 bit) contenuti nei registri indirizzati da $\mathrm{RegRead}_1$ e $\mathrm{RegRead}_2$ (se in modalità lettura)



# Ingredienti di una CPU: Register File

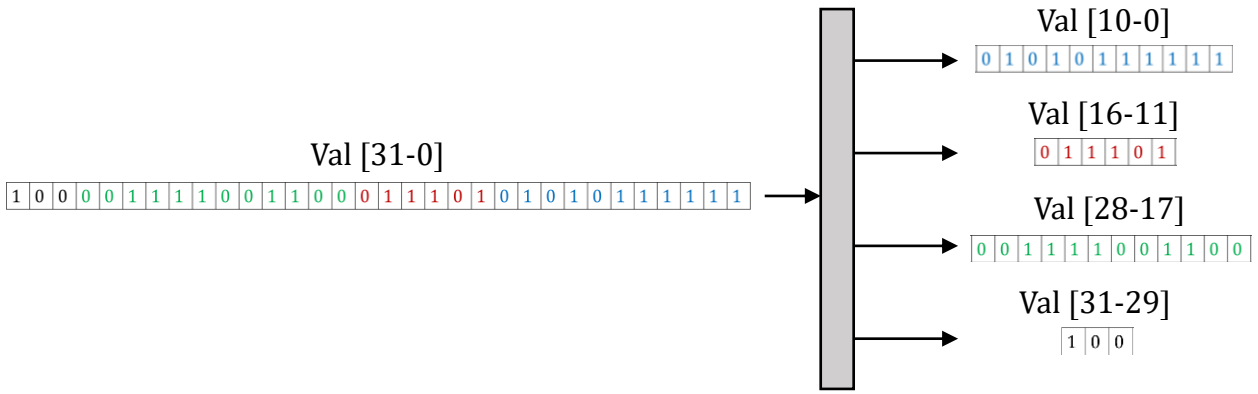
- Esempio di register file con 4 registri a 32 bit
- In questo esempio, avendo solo 4 registri, bastano 2 bit per indirizzare un registro




- **Domanda**: che cosa succede se nello stesso ciclo di clock chiedo di leggere due registri e, allo stesso tempo, di scrivere dentro uno di essi? (Suggerimento: si assuma che i registri siano implementati con dei Flip-Flop)

# Un componente utile: lo Splitter

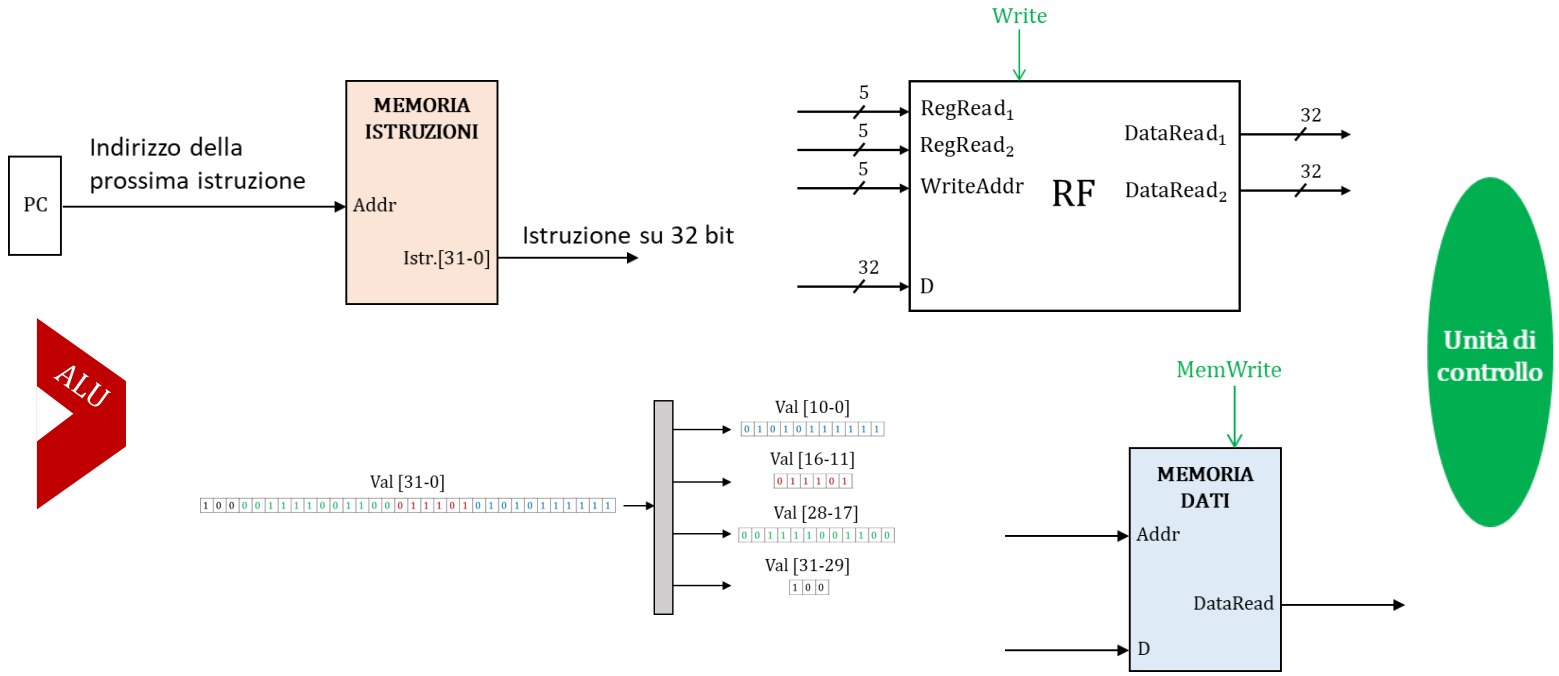
- Splitter: ci permette di separare un segnale su più bit in sottogruppi di bit
- Lo useremo per estrarre i «campi» dai 32 bit di una istruzione
- **Esempio:**




- Lo useremo anche nel verso opposto per raggruppare bit

# Come si costruisce una CPU?

- Dobbiamo combinare insieme gli elementi che abbiamo appena introdotto per costruire un **data path** che, diretto dall'**unità di controllo**, sia in grado di svolgere le istruzioni dell'ISA MIPS



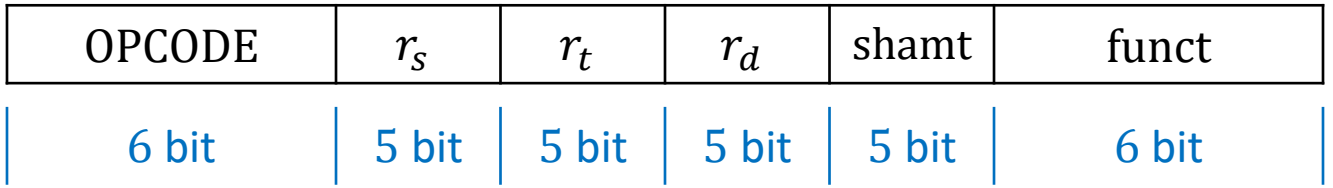

- Approccio incrementale: costruiamo la CPU per passi aggiungendo il supporto di un'istruzione (o tipo di istruzione) per volta
- **Ma quali sono queste istruzioni?**

# Istruzioni con formato R

- Il formato R è usato per le istruzioni che utilizzano esclusivamente valori che sono nei registri e per le operazioni di shift
- Le istruzioni aritmetico-logiche:
  - add $r_d, r_s, r_t$: carica nel registro $r_d$ la somma dei valori contenuti nei registri $r_s$ e $r_t$
  - sub $r_d, r_s, r_t$: carica nel registro $r_d$ la differenza tra valori contenuti nei registri $r_s$ e $r_t$
  - and $r_d, r_s, r_t$: carica nel registro $r_d$ l'AND bit a bit tra valori contenuti nei registri $r_s$ e $r_t$
  - or $r_d, r_s, r_t$: carica nel registro $r_d$ l'OR bit a bit tra valori contenuti nei registri $r_s$ e $r_t$
  - slt $r_d, r_s, r_t$: carica nel registro $r_d$ il valore $1$ se il valore contenuto nel registro $r_s$ è strettamente minore di quello contenuto nel registro $r_t$
- La ALU che abbiamo costruito è già in grado di svolgere le elaborazioni richieste da queste istruzioni! Dobbiamo costruire il data path che gliele faccia eseguire «a comando», una volta letta l'istruzione
- Ciascuna delle istruzioni sopra viene rappresentate su $32$ bit con il formato R, è fatto così:

| OPCODE | $r_s$ | $r_t$ | $r_d$ | shamt | funct |
|---|---|---|---|---|---|
| 6 bit | 5 bit | 5 bit | 5 bit | 5 bit | 6 bit |

# Istruzioni con formato R




- OPCODE: codice identificativo dell'operazione, per le istruzioni di tipo R è sempre pari a 000000
- $r_s$: *source register*, è il numero del registro in cui trovare il primo operando
- $r_t$: *next source register* (la «t» viene dopo la «s» nell'alfabeto), è il numero del registro in cui trovare il secondo operando
- $r_d$: *destination register*, è il numero del registro in cui scrivere il risultato dell'operazione
- shamt: *shift amount*, serve solo per le istruzioni di shift che non considereremo
- funct : *function*, indica quale funzione aritmetico-logica deve essere svolta (add, sub, …)

| Funzione | funct |
|---|---|
| add | 100000 |
| sub | 100010 |
| and | 100100 |
| or | 100101 |
| slt | 101010 |

- (Ci occuperemo solo delle 6 istruzioni indicate in tabella, nel formato R sono espresse anche, shift, jump register, syscall e diverse funzioni aritmetico-logiche con loro varianti)

# Esempio di istruzione con formato R

Istr: $0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0$

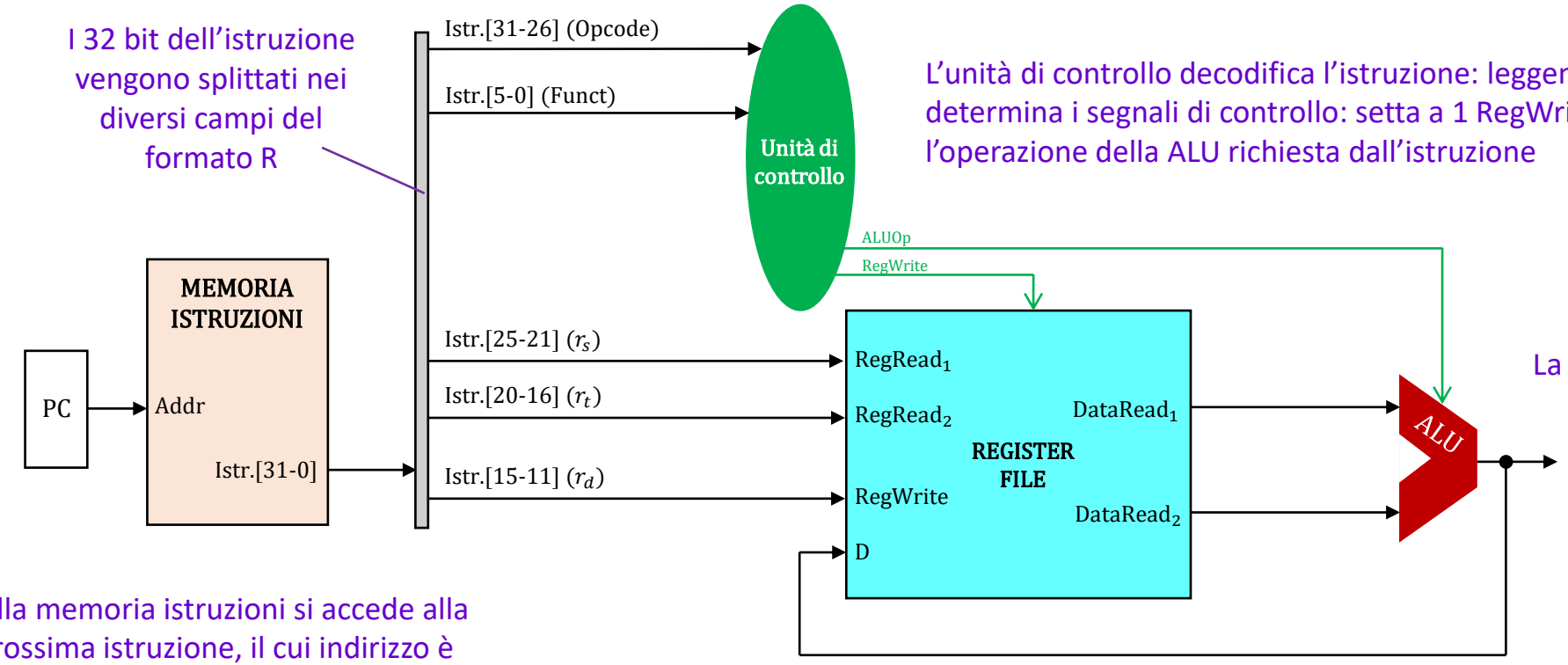
(in formato esadecimale $0x01902824$)

| OPCODE | $r_s$ | $r_t$ | $r_d$ | shamt | funct |
|---|---|---|---|---|---|
| 000000 | 01100 | 10000 | 00101 | 00000 | 100100 |
| 000000 | 12 | 16 | 5 | 00000 | and |

and $r_5, r_{12}, r_{16}$: carica nel registro $r_5$ l'AND bit a bit tra i valori nei registri $r_{12}$ e $r_{16}$

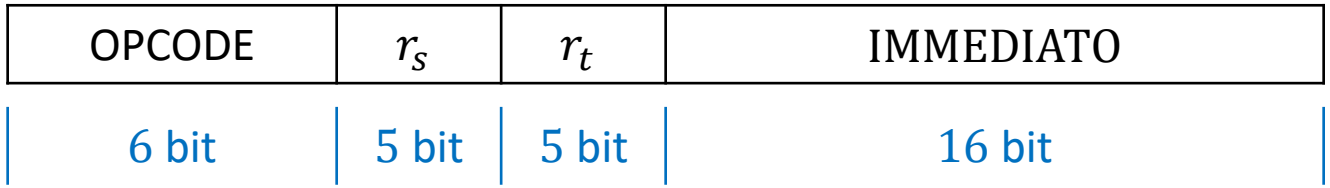
# Data path

- Costruzione del data path per l'esecuzione delle istruzioni di tipo R



# Istruzioni con formato I

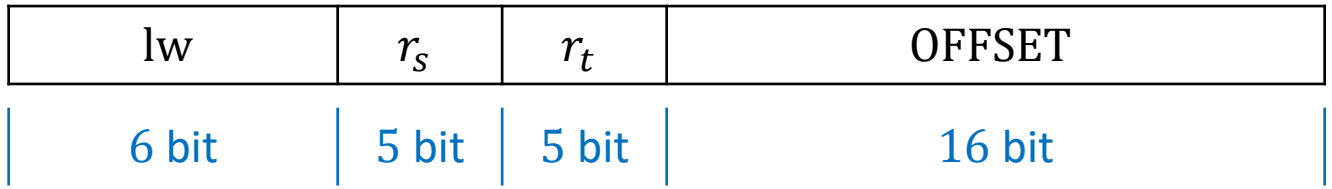
- Il formato I è usato per le istruzioni che utilizzano sia valori che sono in qualche registro sia **costanti**: valori immediati su 16 bit specificati direttamente dentro l'istruzione stessa!
- Le istruzioni di questo formato che vedremo sono:
  - lw $r_t, \mathrm{OFFSET}(r_s)$: carica nel registro $r_t$ la parola di memoria il cui indirizzo è pari a $r_s + \mathrm{OFFSET}$
  - sw $r_t, \mathrm{OFFSET}(r_s)$: trasferisci il contenuto del registro $r_t$ nella parola di memoria all'indirizzo $r_s + \mathrm{OFFSET}$
  - beq $r_s, r_t, \mathrm{OFFSET}$: se il contenuto del registro $r_s$ è uguale a quello del registro $r_t$ salta all'istruzione che sta all'indirizzo $\mathrm{PC} + \mathrm{OFFSET}$, altrimenti prosegui normalmente con la prossima istruzione $(\mathrm{PC} + 4)$




- Questo formato include anche istruzioni aritmetico-logiche come addi, andi, etc.. Queste sono varianti delle istruzioni originali dove uno degli operandi è un immediato (quindi caricato direttamente dall'istruzione e non da un registro)
- Ci sono anche varianti delle istruzioni per accesso a memoria (lb, sb) e dei branch (bne, beqz, …)

# Accesso a memoria: load word

$\text{lw } r_t, \text{OFFSET}(r_s)$




- Opcode: 100011
- Il registro destinazione ora è $r_t$, quindi dobbiamo estendere il data path per poter scegliere, a seconda dell'istruzione, quale registro indirizzare per la scrittura nel Register File
- **Modalità di indirizzamento con base address e offset**: l'indirizzo a cui accedere si ottiene sommando un indirizzo di partenza (base address) e un offset in byte codificato come intero binario in C2 su 16 bit
- Nell'istruzione di load word il base address si trova in un registro il cui indirizzo è specificato nel campo $r_s$
- L'offset da sommare è il valore immediato specificato direttamente nell'istruzione
- Per calcolare l'indirizzo dobbiamo fare una somma: **useremo la ALU**

# Esempio di istruzione lw con formato I

Istr: 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0

(in formato esadecimale $0x8DC500F0$)

| OPCODE | $r_s$ | $r_t$ | OFFSET |
|---|---|---|---|
| 100011 | 01110 | 00101 | 0000000011110000 |
| lw | 14 | 5 | 240 |

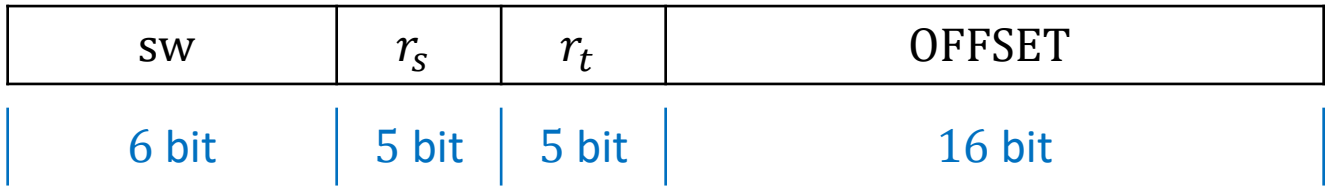
lw $r_5, 240(r_{14})$: carica nel registro $r_5$ la parola di memoria il cui indirizzo si ottiene sommando $240$ al valore contenuto nel registro $r_{14}$

**Attenzione!**

- Il contenuto del registro $r_{14}$ deve essere un indirizzo valido, altrimenti si genera un errore
- Il valore dell'offset deve essere un multiplo di $4$ altrimenti l'indirizzo risultante non sarà allineato e si genera un errore

# Accesso a memoria: store word

$\text{sw } r_t, \text{OFFSET}(r_s)$




- Opcode: $101011$
- Questa istruzione non scrive nel Register File ma nella memoria
- Si usa la stessa modalità di indirizzamento di load word: somma del base address contenuto in $r_s$ e offset immediato (anche in questo caso sfrutteremo la ALU)
- Il registro $r_t$ contiene il valore da trasferire in memoria

# Esempio di istruzione **sw** con formato I

Istr: 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0

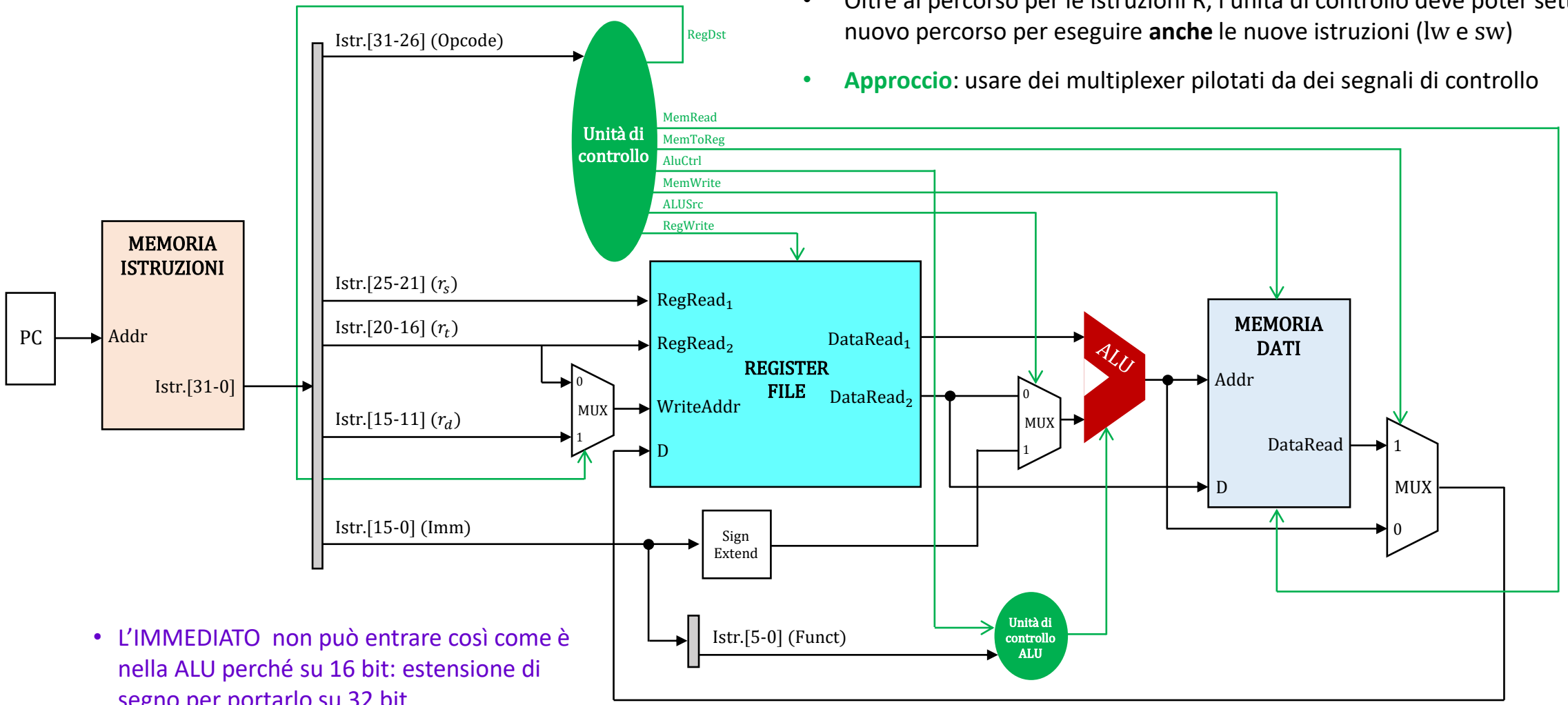
(in formato esadecimale $0xAFCD0C04$)

| OPCODE | $r_s$ | $r_t$ | OFFSET |
|---|---|---|---|
| 101011 | 11110 | 01101 | 0000110000000100 |
| sw | 30 | 13 | 3076 |

sw $r_{13}, 3076(r_{30})$: trasferisci il valore contenuto nel registro $r_{13}$ nella parola di memoria il cui indirizzo si ottiene sommando $3076$ al valore contenuto nel registro $r_{30}$

# Costruzione del data path

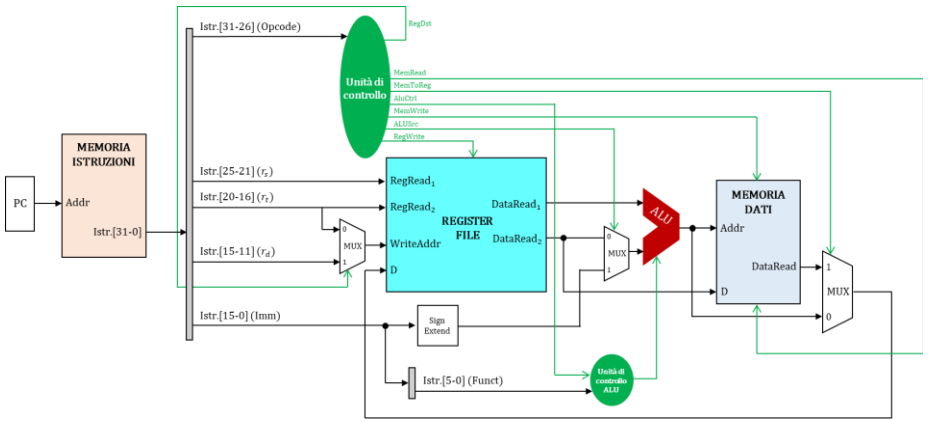
- Dobbiamo consentire all'unità di controllo di poter impostare diversi percorsi sul data path in modo che cambieranno a seconda dell'istruzione da eseguire
- Oltre al percorso per le istruzioni R, l'unità di controllo deve poter settare un nuovo percorso per eseguire **anche** le nuove istruzioni (lw e sw)
- **Approccio**: usare dei multiplexer pilotati da dei segnali di controllo

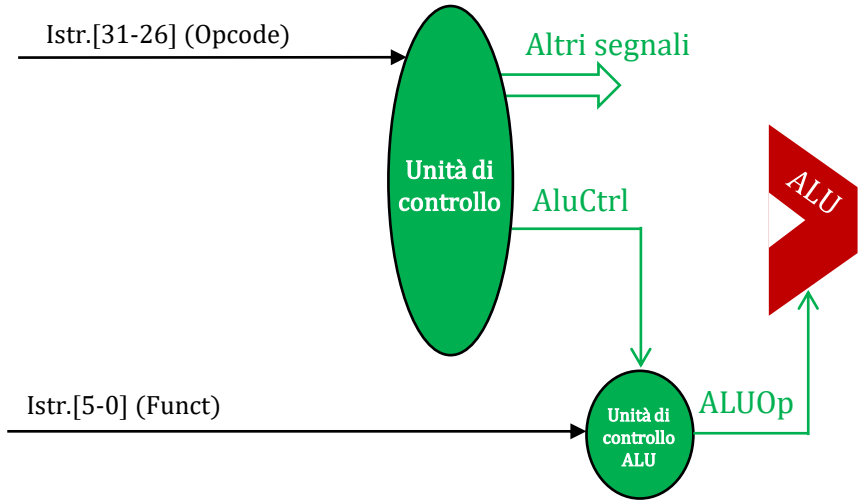



- L'IMMEDIATO non può entrare così come è nella ALU perché su 16 bit: estensione di segno per portarlo su 32 bit

# Costruzione del data path

- Nell'estendere il data path per poter eseguire lw e sw abbiamo separato dall'unità di controllo la parte dedicata alla ALU
- L'unità di controllo della ALU ha il compito di configurare la ALU in modo che svolga una precisa operazione
- L'unità di controllo centrale ha il compito segnalare quale è lo scenario in cui si sta operando (quale istruzione)
- Vedremo i dettagli implementativi più avanti quando sintetizzeremo il circuito per l'unità di controllo



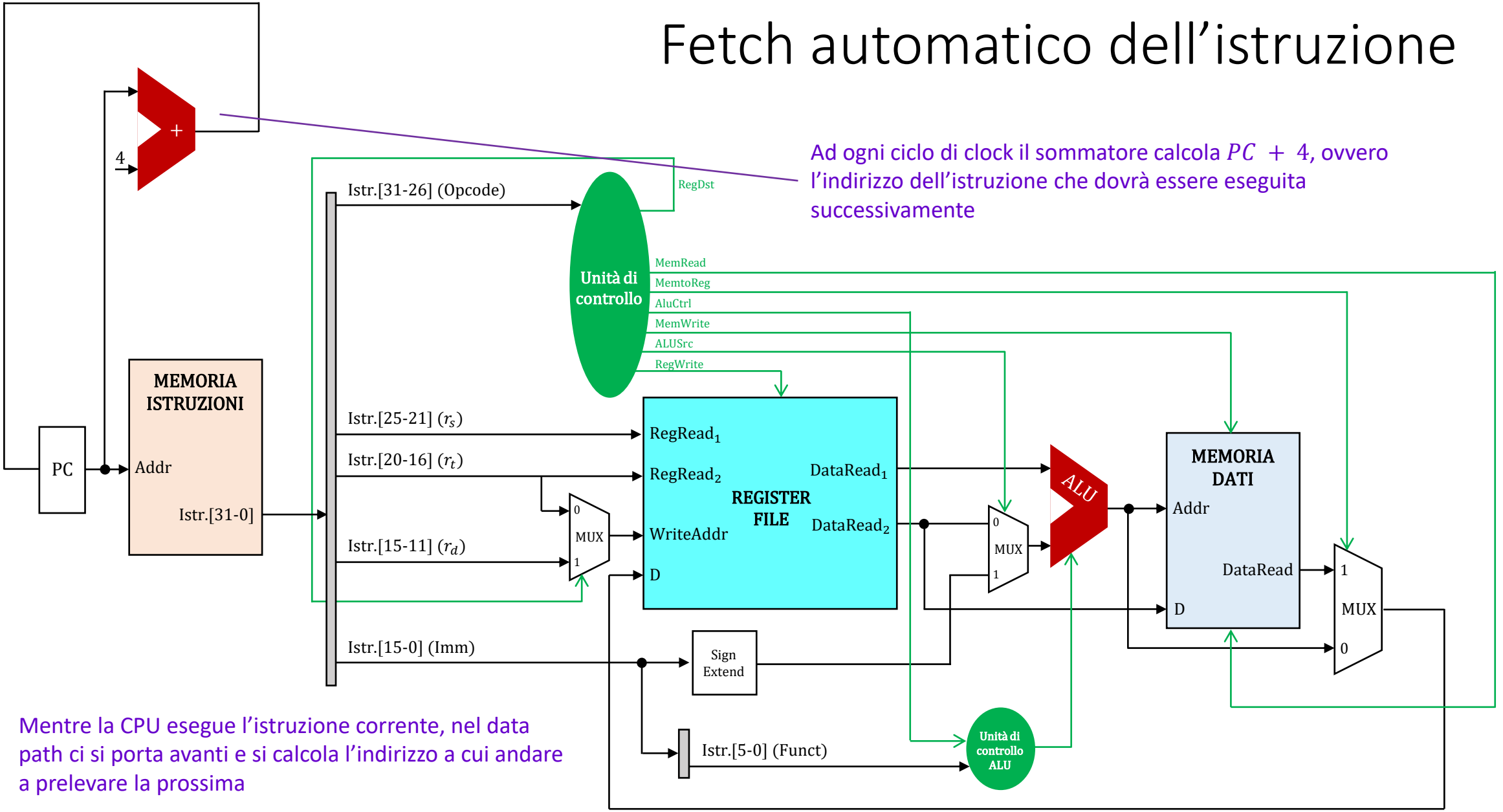



# Fetch automatico dell'istruzione

- Il data path costruito fino ad ora è in grado di eseguire add, sub, and, or, slt, lw e sw
    1. L'istruzione si presenta in uscita dalla memoria istruzioni
    2. L'unità di controllo la decodifica e genera gli opportuni segnali di controllo attivando, sul data path, il percorso appropriato che i segnali devono attraversare perché venga svolta l'elaborazione richiesta
    3. I segnali attraversano il data path e l'istruzione viene eseguita
- Questo procedimento vale per una singola istruzione che assumiamo sia data
- La CPU deve eseguire in modo iterativo un'istruzione dopo l'altra ad ogni ciclo di clock: **eseguita un'istruzione si deve passare alla prossima in modo automatico**
- Dobbiamo estendere il data path in modo che implementi la **fase di fetch** della prossima istruzione: il prelievo dell'istruzione che si trova all'indirizzo $PC + 4$ nella memoria dati
- In questo modo equipaggiamo la nostra CPU con la capacità di eseguire in sequenza le istruzioni che trova nella memoria: tale sequenza rappresenta il **normale flusso di esecuzione** che le istruzioni di salto potranno alterare in modo condizionato o meno

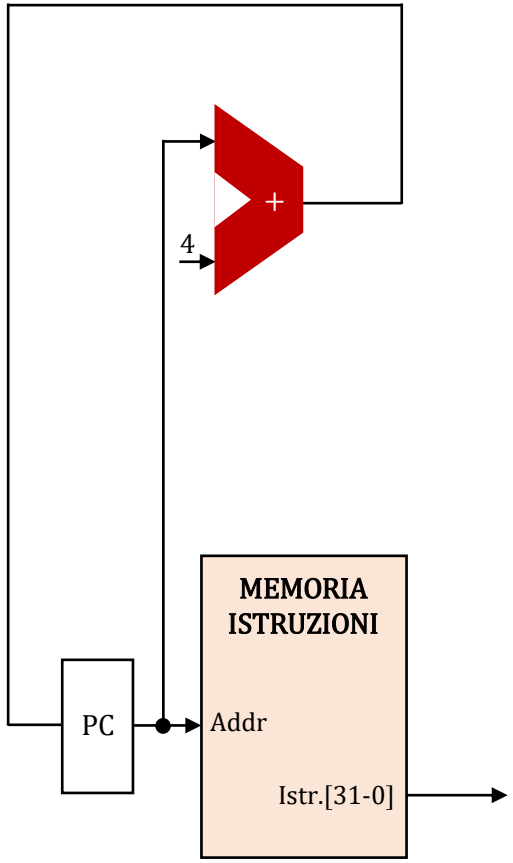
# Fetch automatico dell'istruzione

Ad ogni ciclo di clock il sommatore calcola $PC + 4$, ovvero l'indirizzo dell'istruzione che dovrà essere eseguita successivamente




Mentre la CPU esegue l'istruzione corrente, nel data path ci si porta avanti e si calcola l'indirizzo a cui andare a prelevare la prossima

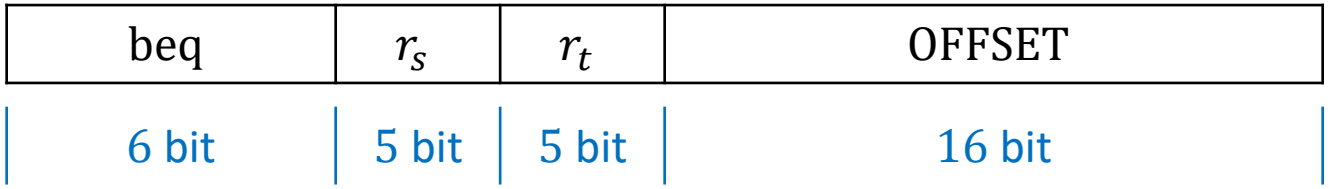
# Alterare il flusso di esecuzione




- Controllo di flusso: decidere chi sarà la prossima istruzione da eseguire
- Se non si specifica nulla, la prossima istruzione è quella che segue nella memoria cioè quella rappresentata dai 32 bit che stanno nella parola successiva ($PC + 4$)
- I salti provocano un'alterazione del normale flusso di controllo
- Che aspetto ha un salto dentro il data path? Fare un salto significa scrivere nel $PC$ un indirizzo diverso da $PC + 4$
- Si può fare in due modi
  - **Salto condizionato**: se una condizione è vera si sovrascrive $PC$, altrimenti $PC + 4$
  - **Salto non condizionato**: si sovrascrive $PC$ **sempre** con un nuovo indirizzo
- In entrambi in casi il data path deve essere esteso per supportare il calcolo dell'indirizzo di salto
- Nel primo caso (salto condizionato) si deve anche supportare il calcolo dell'esito di una condizione

# Salto condizionato: branch on equal

$\text{beq } r_s, r_t, \text{OFFSET}$




- Siamo sempre in formato I con Opcode: $000100$
- L'istruzione prevede la verifica di una condizione di uguaglianza tra i contenuti dei registri $r_s$ e $r_t$
  **Come verificarla?** Usiamo la ALU: facciamo la differenza tra i contenuti dei registri e controlliamo il **bit di zero**
  - Condizione non verificata: non succede niente, si procede con l'istruzione che sta all'indirizzo $PC + 4$
  - Condizione verificata: bisogna fare un salto! **Come si calcola l'indirizzo di salto?**
- **Modalità di indirizzamento PC-Relative**: i 16 bit dell'immediato (intero in C2) si interpretano come un offset in **numero di istruzioni** (**Attenzione! Non è in byte** come invece succedeva con l'indirizzamento base address + offset)
- L'indirizzo di salto si ottiene facendo $(\text{PC} + 4) + \mathbf{4} \times \text{OFFSET}$
- Salti di ampiezza limitata:
  - Salto all'indietro più ampio: $(\text{PC} + 4) + 4 \times (-2^{15})$
  - Salto in avanti più ampio: $(\text{PC} + 4) + 4 \times (2^{15} - 1)$

# Esempio di istruzione **beq** con formato I

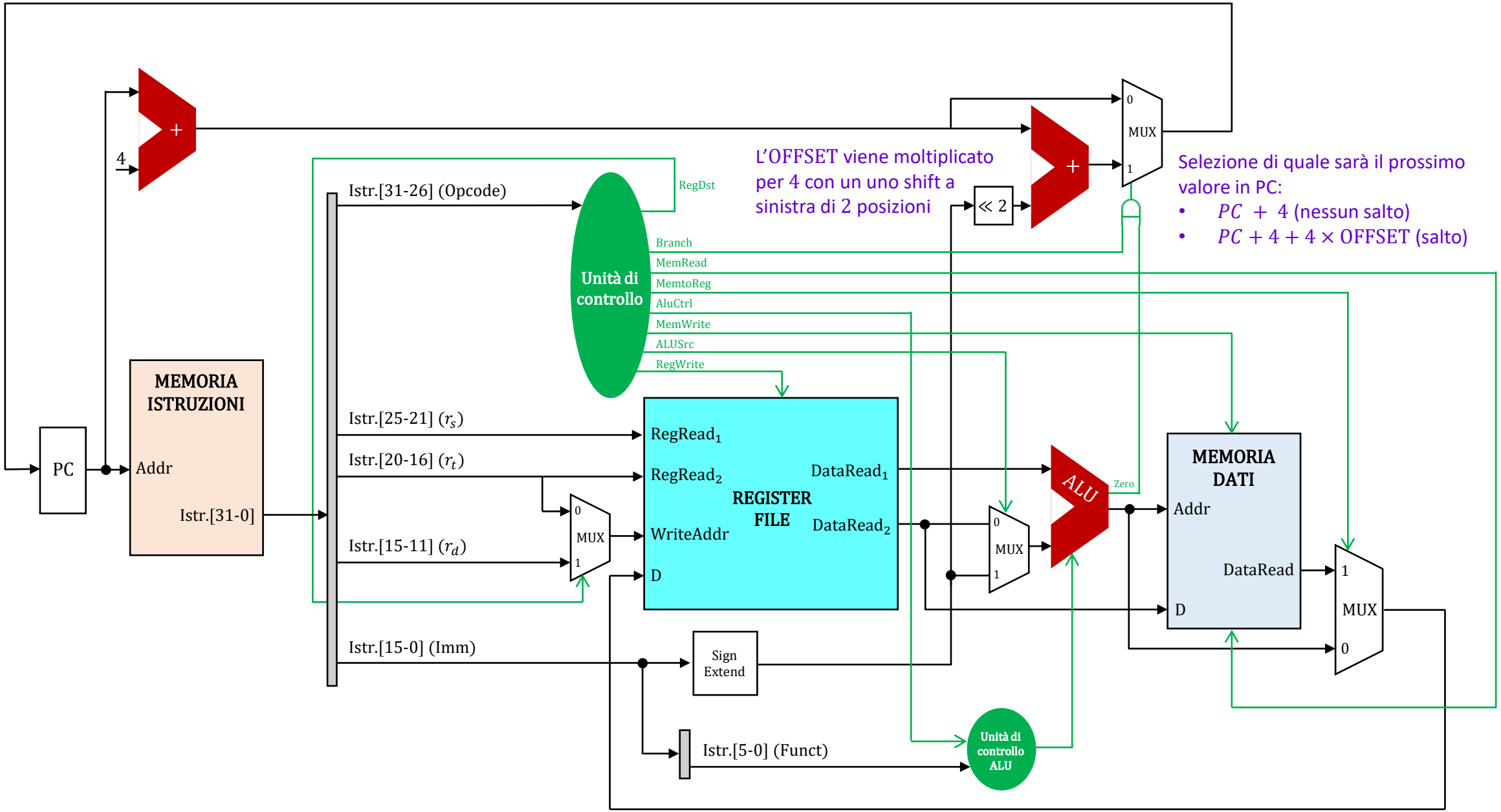
Istr: 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1

(in formato esadecimale $0x124B001B$)

| OPCODE | $r_s$ | $r_t$ | OFFSET |
|---|---|---|---|
| 000100 | 10010 | 01011 | 0000000000011011 |
| beq | 18 | 11 | 27 |

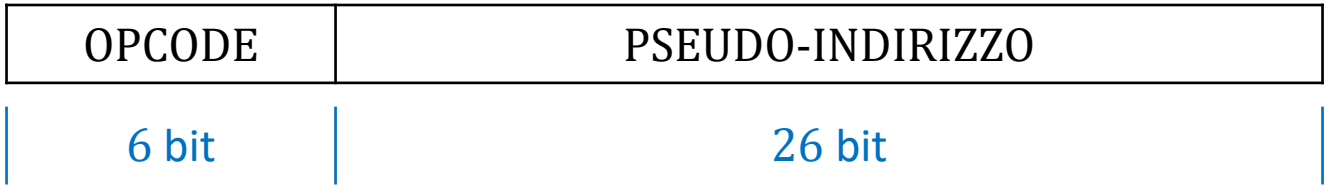
Se il contenuto del registro $r_{18}$ è uguale al contenuto del registro $r_{11}$ la prossima istruzione che la CPU deve eseguire è quella che sta all'indirizzo $\mathrm{PC} + 4 + 4 \times 27$ nella memoria istruzioni

# Costruzione del data path



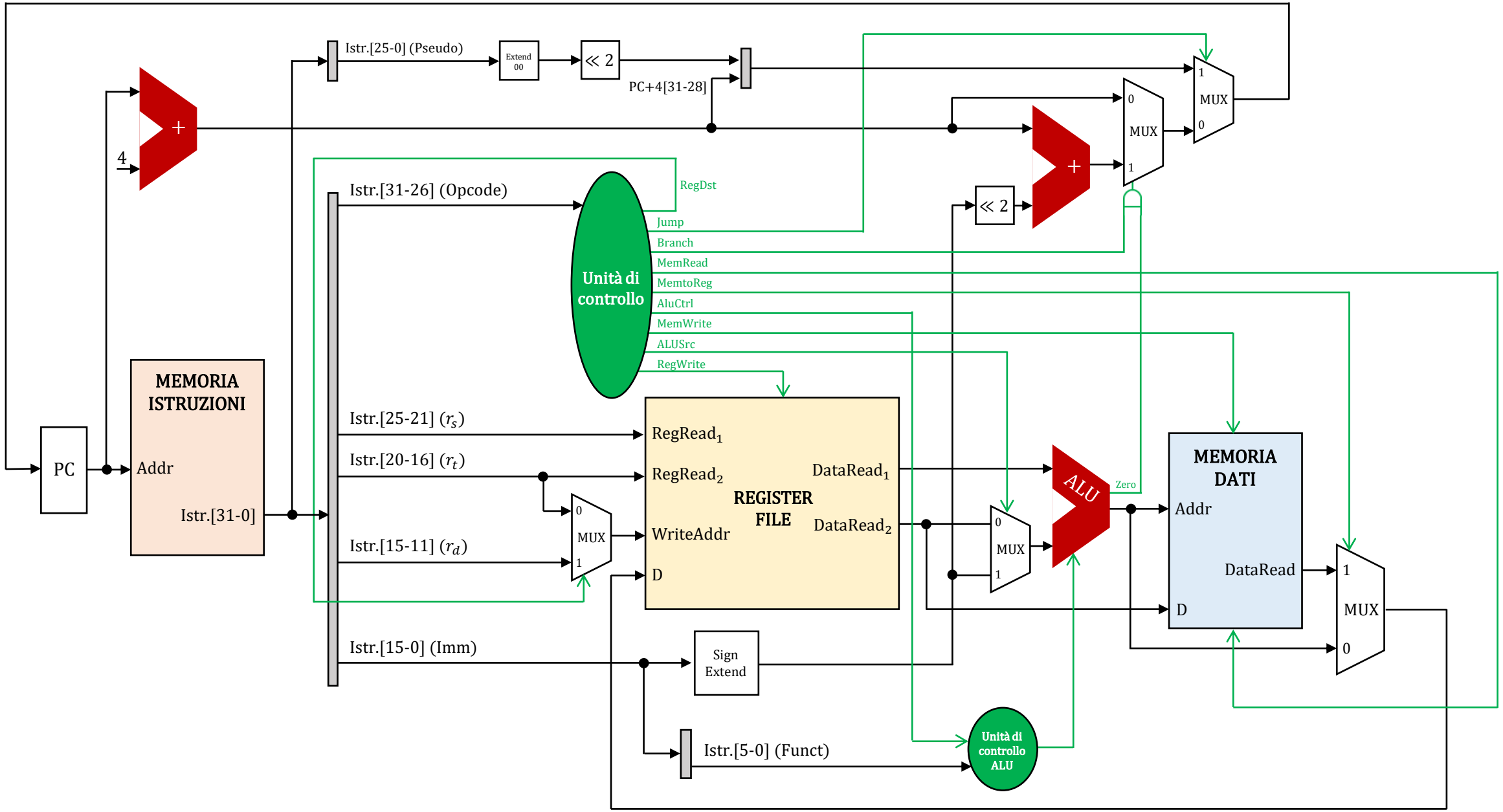
# Istruzioni con formato J

- Il formato J è dedicato interamente alle istruzioni di salto non condizionato




- j PSEUDO-INDIRIZZO: scrivi nel PC un nuovo indirizzo ottenuto applicando la **modalità di indirizzamento pseudo-diretta** sui 26 bit del campo PSEUDO-INDIRIZZO
- Opcode: 000010
- Modalità di indirizzamento pseudo-diretta:
  - Aggiungere due 0 a destra dei 26 bit del campo PSEUDO-INDIRIZZO (come moltiplicare per 4)
  - Aggiungere a sinistra i 4 bit più significativi del Program Counter
- Il risultato è un indirizzo su 32 bit da sovrascrivere nel PC

# Istruzioni con formato J



# Le cinque fasi di esecuzione di una istruzione