# Architettura degli Elaboratori

## Il Processore:
## Implementazione di Istruzioni Aggiuntive

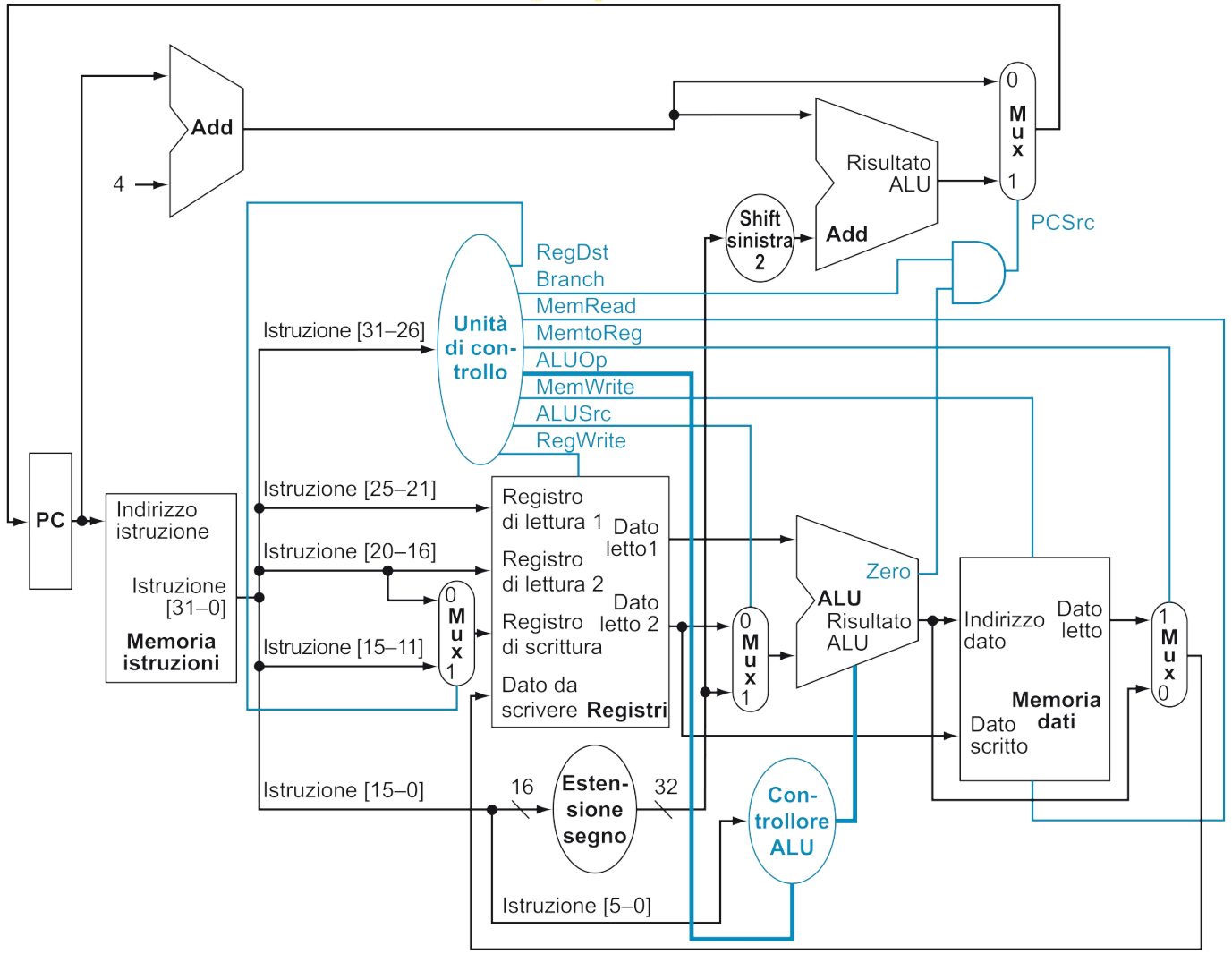
Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

# Punto della situazione

- Abbiamo studiato l'implementazione a ciclo singolo per un set ridotto di istruzioni
  - Istruzioni di accesso alla memoria (lw, sw)
  - Istruzioni aritmetico-logiche (add, sub, and, or, slt)
  - Istruzioni di salto condizionato (beq)

- Obiettivo di oggi:
  Aggiungere altre istruzioni al set considerato
  - Istruzioni con operandi immediati (addi, andi, etc…)
  - Istruzione di salto incondizionato (j)
  - Istruzione jump and link (jal)
  - Istruzione jump register (jr)

# Implementazione Precedente



# Istruzione addi

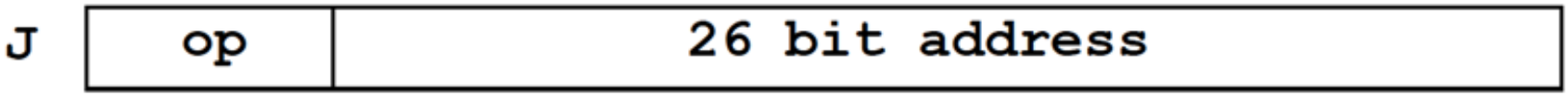
- Aggiungiamo l'istruzione addi
  - È una istruzione di tipo I (come load e store)

| I | op | rs | rt | 16 bit address |
|---|---|---|---|---|

- Comportamento molto simile alla load:
  - L'ALU effettua la somma del contenuto di un registro con una costante
  - Il risultato della ALU però non deve essere interpretato come un indirizzo in memoria, quindi
    - MemRead → 0 (nella load è 1)
    - MemtoReg → 0 (nella load è 1)
- Non sono necessari componenti aggiuntivi
  - Basta modificare il comportamento dell'UC per farle "riconoscere" anche il codice operativo di addi e impostare i segnali di controllo

# Istruzione j

- Aggiungiamo l'istruzione di salto incondizionato: j
  - È una istruzione di tipo J

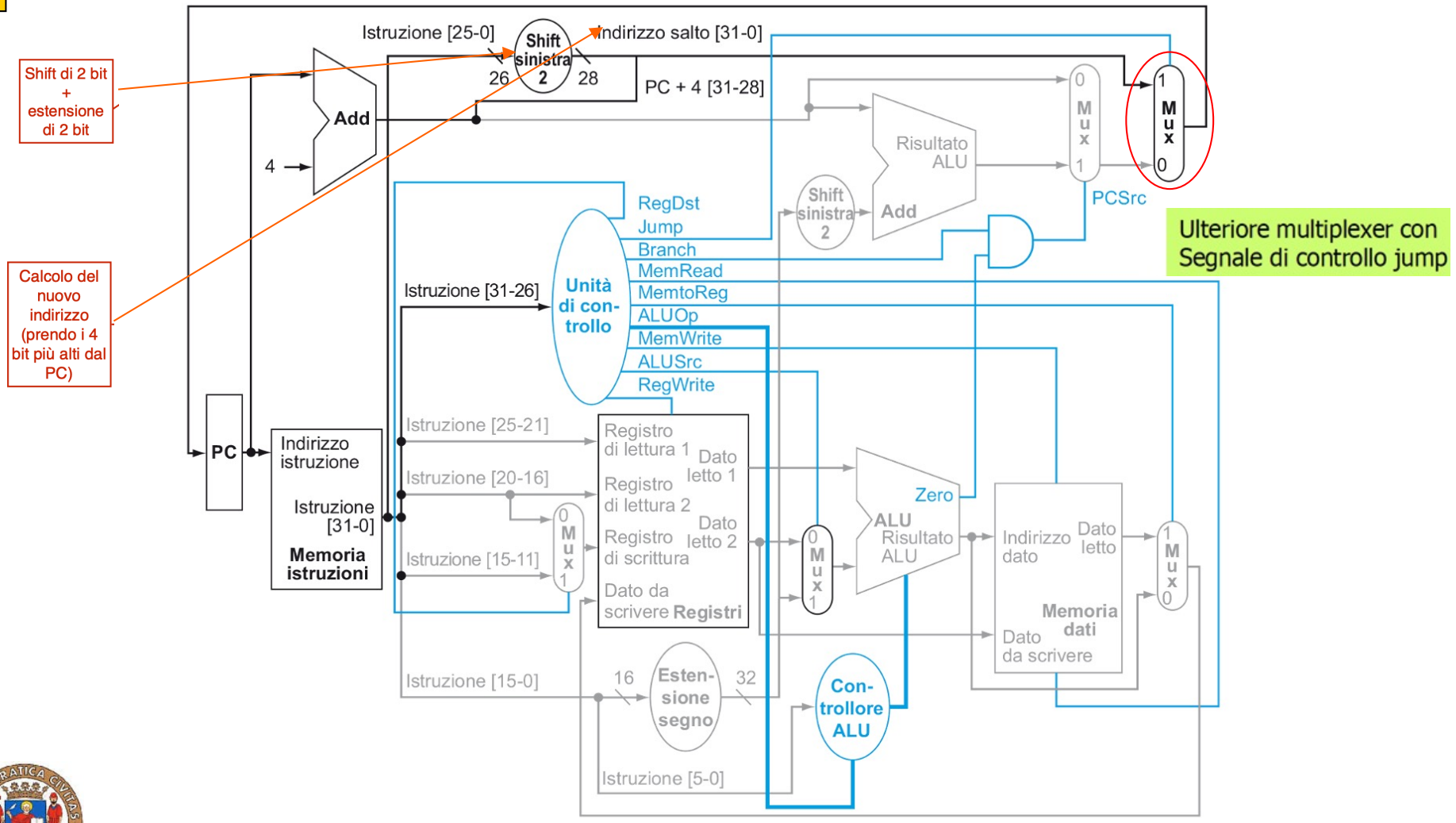



- Comportamento simile all'istruzione beq, ma nel registro PC viene scritta una stringa di 32 bit dove
  - I 4 bit più significativi sono quelli di PC+4 (output dell'Adder)
  - I 2 bit meno significativi sono posti uguali a 0
  - I 26 bit restanti sono sostituiti con il campo indirizzo dell'istruzione j

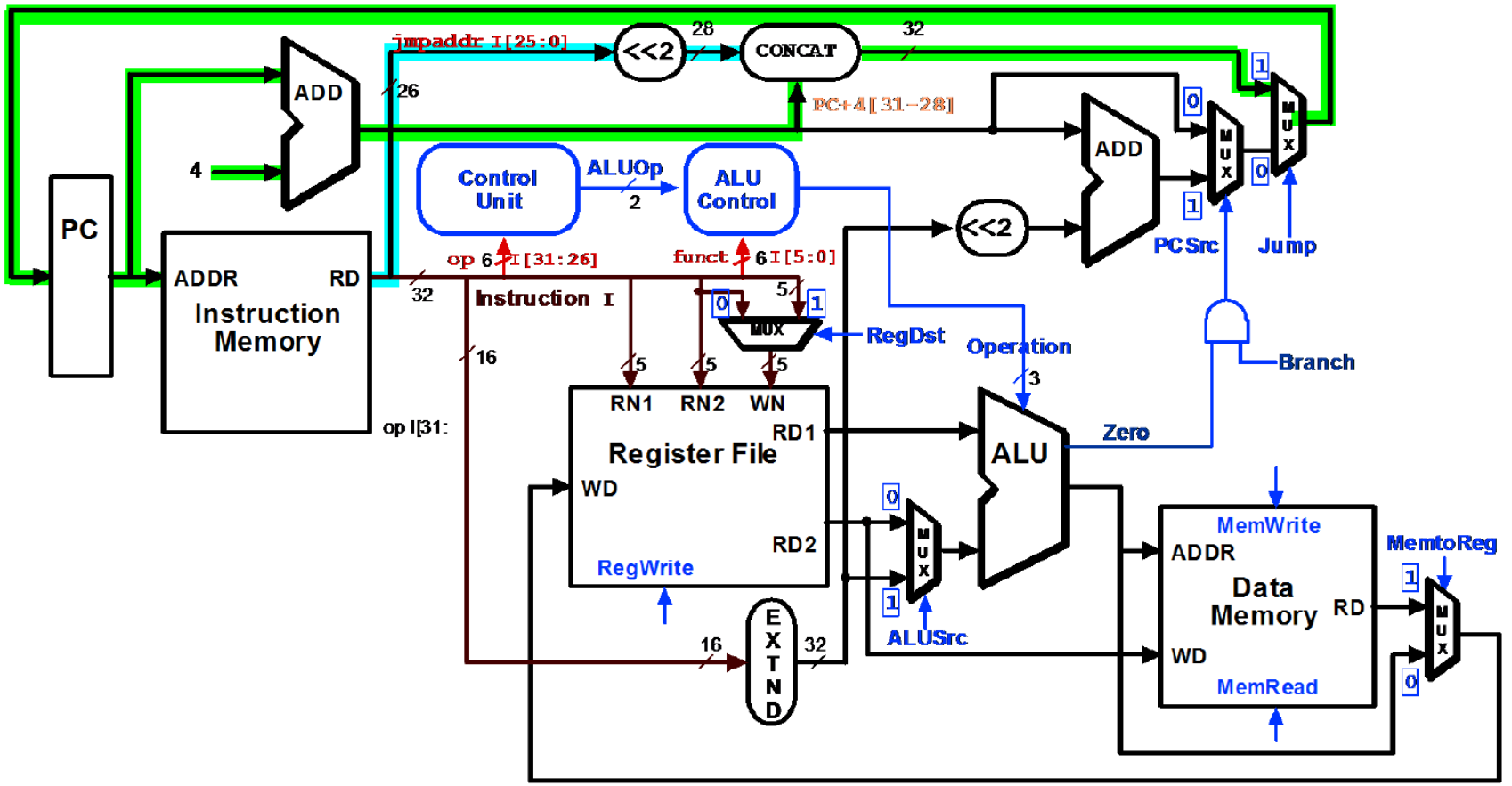
# Istruzione j

- Quindi per implementare j è necessario aggiungere i seguenti componenti
  - Uno shifter a sx di due posizioni che, presa in input una stringa di 26 bit ne restituisca una di 28 bit
  - Un mux 2:1 con un segnale di controllo (Jump)
- Inoltre, va modificata anche l'UC in modo che "riconosca" il codice operativo dell'istruzione j e imposti il segnale di controllo Jump = 1

# Istruzione j

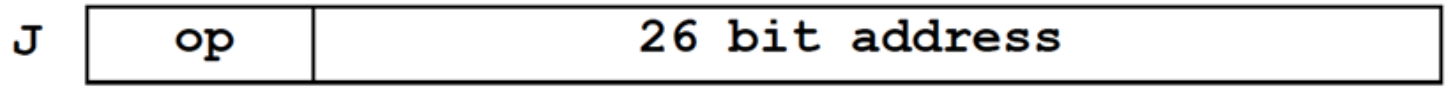


# Datapath Istruzione j




# Istruzione jal

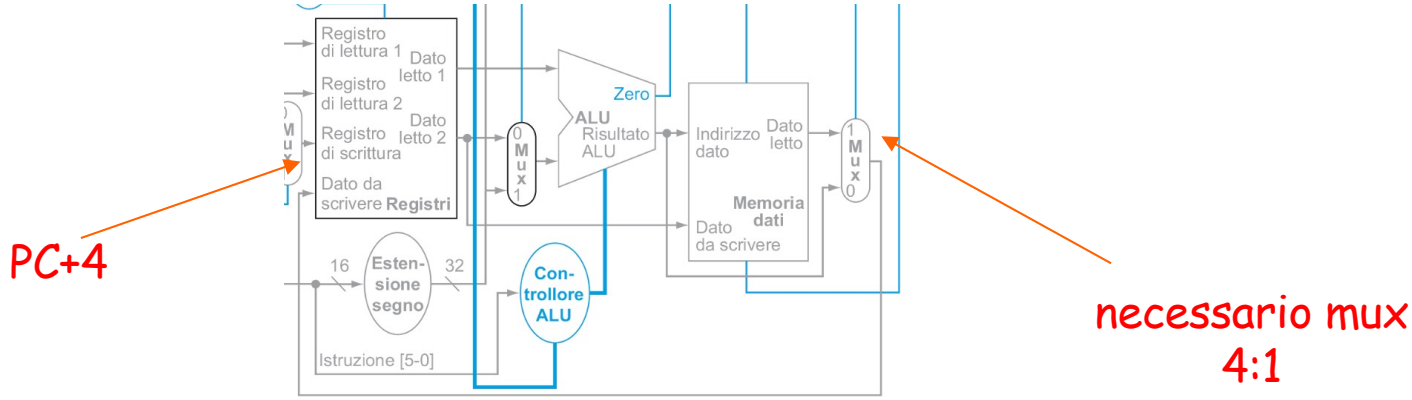
- Aggiungiamo l'istruzione jal
  - E' una istruzione di tipo J (come j)




- L'indirizzo di salto viene calcolato allo stesso modo dell'istruzione j
  - Aggiunta dello shifter a sx di due posizioni e del multiplexer 2:1 con segnale di controllo Jump
- A differenza dell'istruzione j però, jal deve anche scrivere l'indirizzo di ritorno (PC + 4) nel registro $ra (31)
  - Come implementare questa scrittura?

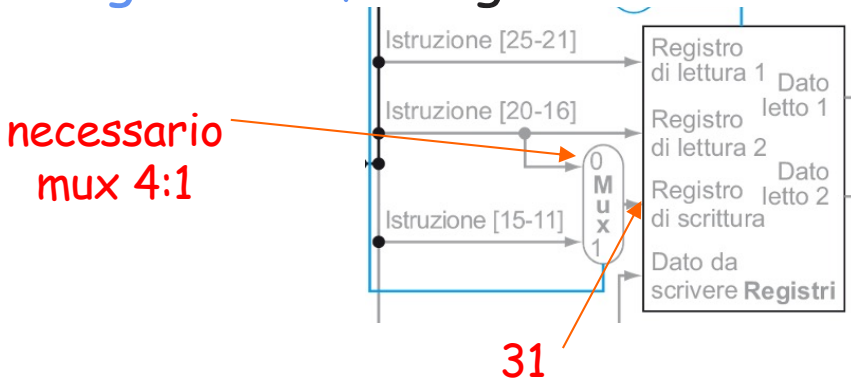
# Istruzione jal

- E' innanzitutto necessario fornire il valore PC+4, ottenuto dall'adder, come input "Dato da scrivere" del Register File
  - L'input "Dato da scrivere" è l'output del mux 2:1 posto in uscita alla memoria (con segnale di controllo "MemtoReg" a 1 bit), dove
    - Se MemtoReg = 0, WriteData viene dall'output dell'ALU
    - Se MemtoReg = 1, WriteData è il dato letto in memoria
  - Quindi è necessario aggiungere una terza linea dati al mux 2:1 (che diventa un mux 4:1, con segnale di controllo a 2 bit)
    - Se MemtoReg = 2, WriteData è PC+4



# Istruzione jal

- Inoltre è necessario scrivere nel registro $ra (31)
  - L'input "Registro di scrittura" è l'output del mux 2:1 posto in uscita al banco dei registri (con segnale di controllo "RegDest" a 1 bit), dove
    - Se RegDest = 0, il registro di scrittura viene dai campi [20-16] dell'istruzione (tipo I)
    - Se RegDest = 1, il registro di scrittura viene dai campi [15-11] dell'istruzione (tipo R)
  - Quindi è necessario aggiungere una terza linea dati al mux 2:1 (che diventa un mux 4:1, con segnale di controllo a 2 bit)
    - Se RegDest = 2, il registro di scrittura è 31




# Istruzione jal

- Quindi per implementare jal, oltre alle aggiunte necessarie per implementare j, è necessario sostituire due mux 2:1 con due mux 4:1

- Inoltre, va modificata anche l'UC in modo che "riconosca" il codice operativo dell'istruzione jal e imposti i segnali di controllo necessari

# Istruzione jr

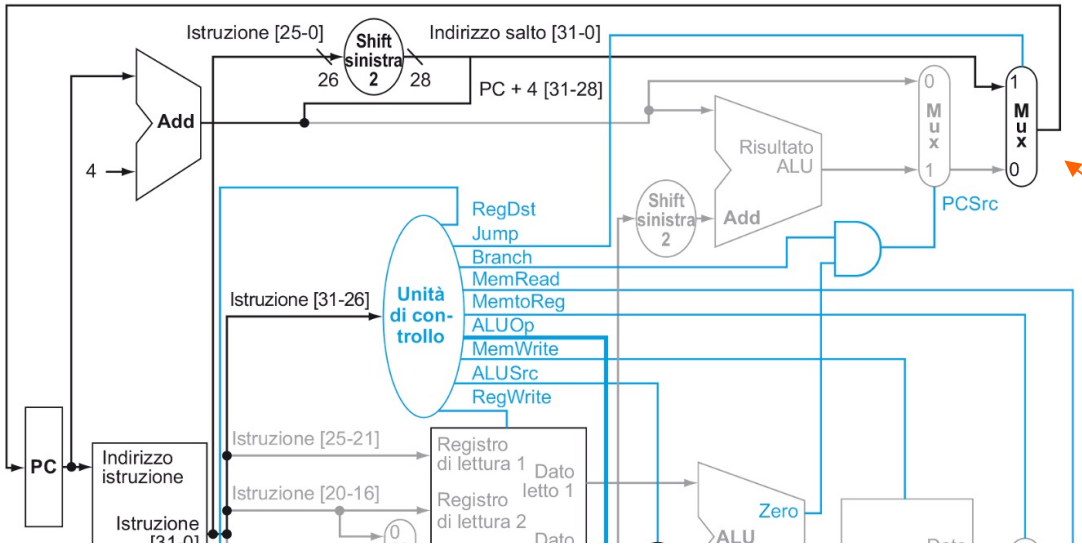
- Aggiungiamo l'istruzione jr
  - È una istruzione di tipo R

| 6 bit | 5 bit | 5 bit | 5 bit | 5 bit | 6 bit |
|---|---|---|---|---|---|
| OP | rs | rt | rd | shamt | funct |
| 000000 | indirizzo | 00000 | 00000 | 00000 | 001000 |

- Comportamento:
  - Scrive nel registro PC il contenuto del registro rs, corrispondente all'indirizzo di memoria istruzioni a cui saltare
  - Tipicamente si usa jr $ra e quindi rs=31

# Istruzione jr

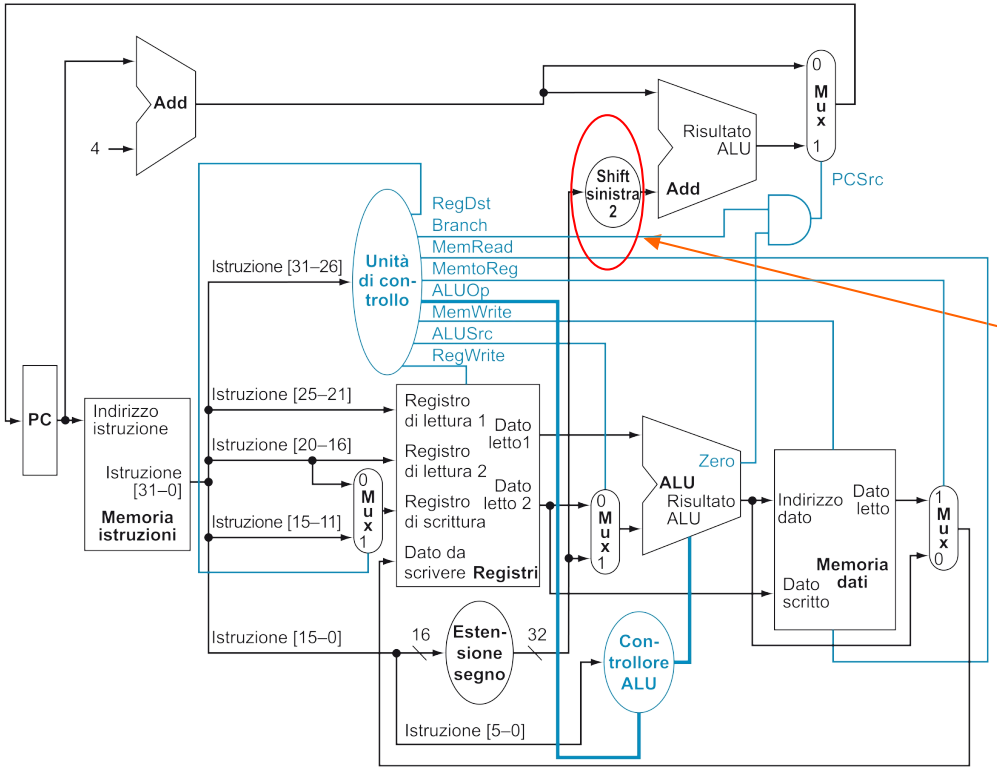
- Nell'implementazione a cui è stata aggiunta l'istruzione j, l'input del registro PC viene da un mux 2:1 con segnale di controllo Jump
  - Quindi è necessario aggiungere una terza linea dati al mux 2:1 (che diventa un mux 4:1, con segnale di controllo a 2 bit)
  - La nuova linea dati deve provenire dall'output Dato letto 1 del Register file




necessario mux 4:1

# Left Logic Shifter di 2 posti (su input di 32 bit)

- Nel datapath del MIPS è presente anche una unità che effettua lo shift logico a sinistra di due posti
  - Come funziona questa unità?



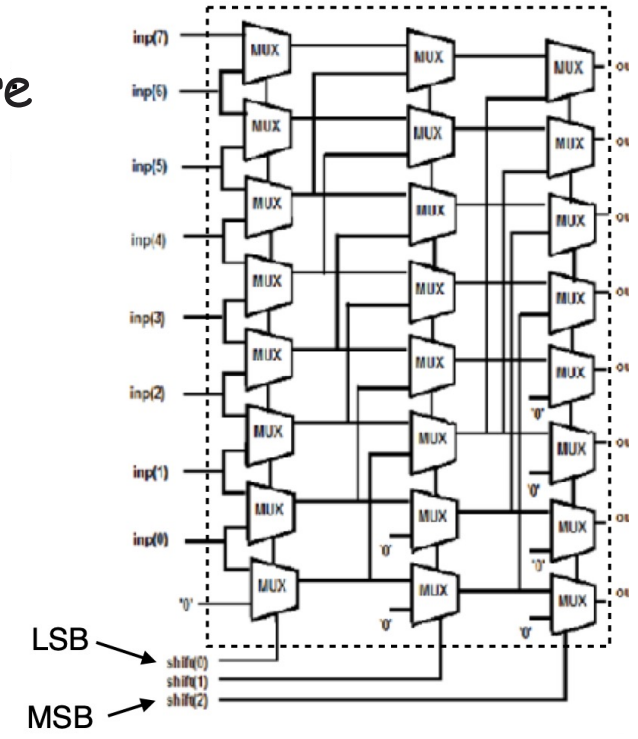

Shift logico a sinistra di 2 posti a partire da una stringa di 32 bit

# Left Logic Shifter di 2 posti (su input di 32 bit)

- L'unità in questione è un caso particolare di shifter a 32 bit
  - Rete combinatoria costituita da 5 colonne, ciascuna con $32=2^5$ MUX 2:1, collegate tra loro
  - Ogni colonna condivide lo stesso segnale di controllo (1 bit) per tutti i MUX
  - In totale quindi ci sono 5 segnali di controllo
    - Corrispondono all'ammontare dello shift, che in questo caso è pari a $2_{10} = 00010_2$

# Left Logic Shifter (su input di 8 bit)

- Rete combinatoria costituita da 3 colonne, ciascuna con 8 =$2^3$ MUX 2:1, collegate tra loro
  - Ogni colonna condivide lo stesso segnale di controllo (1 bit) per tutti i MUX
  - In totale quindi ci sono 3 segnali di controllo, corrispondenti all'ammontare dello shift
    - 000 = shift di 0 posti
    - 111= shift di 7 posti



# Riepilogo e riferimenti

- Abbiamo visto come modificare l'implementazione a ciclo singolo per aggiungere alcune istruzioni al sottoinsieme studiato
  - Istruzioni con operandi immediati (addi, andi, etc…)
  - Istruzione di salto incondizionato (j)
  - Istruzione jump and link (jal)
  - Istruzione jump register (jr)

- Per permettere l'esecuzione di altre istruzioni è necessario aggiungere ulteriori componenti hardware e complicare la logica di controllo
  - Ad esempio la moltiplicazione (mult) necessita anche di shifter