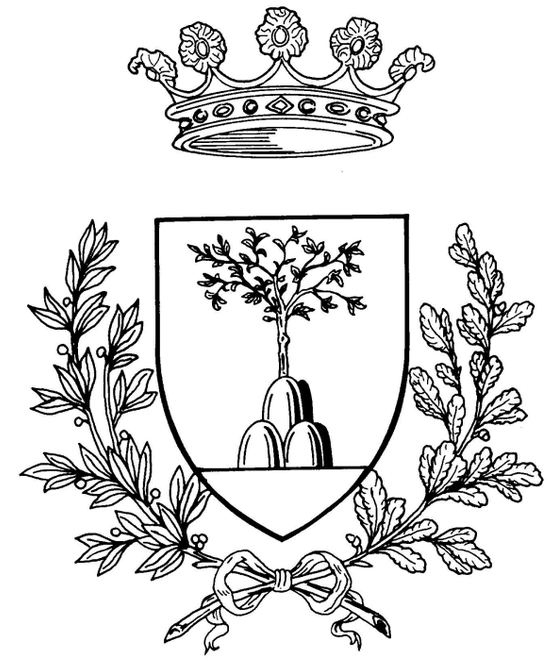


Architettura degli Elaboratori e Laboratorio

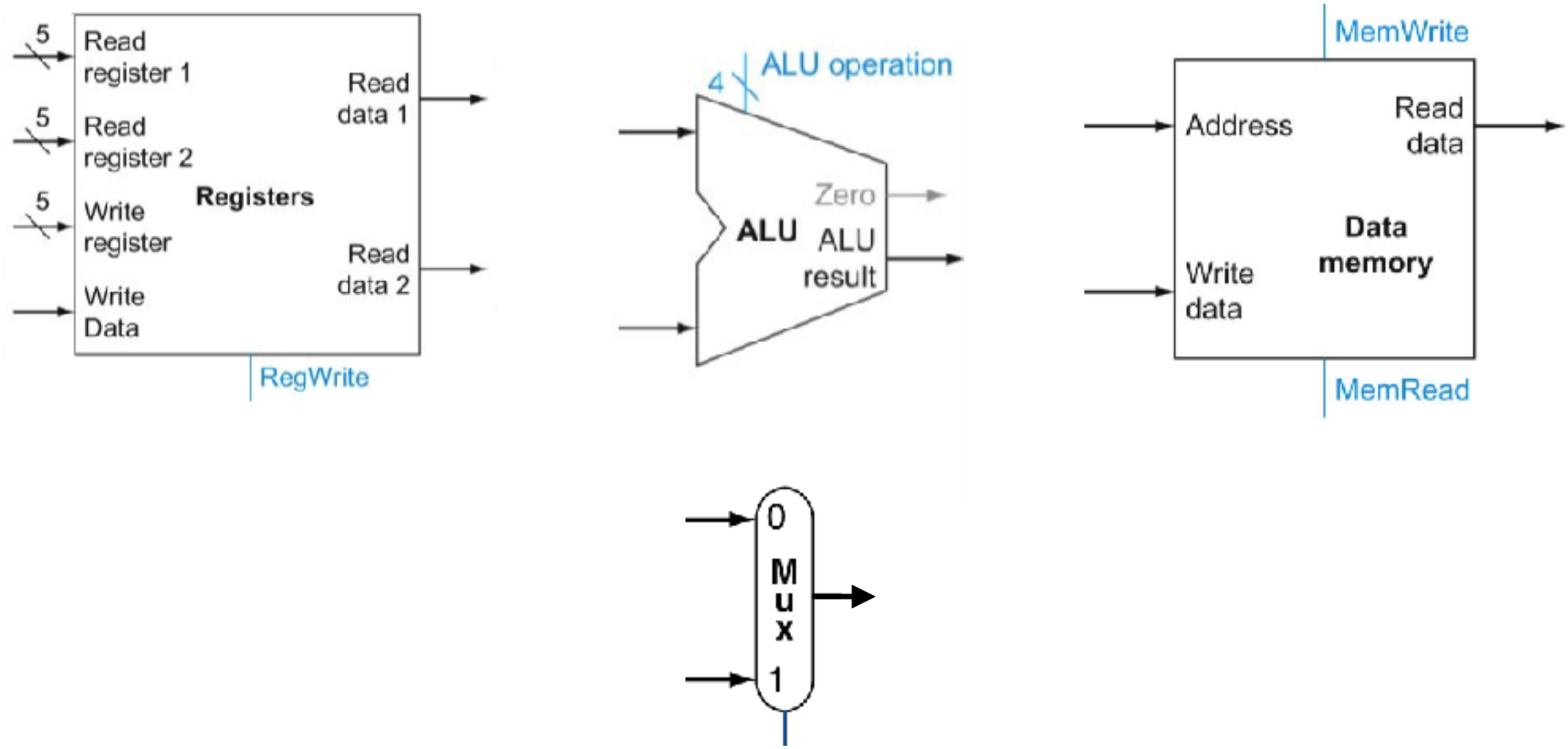
Matteo Manzali

Università degli Studi di Ferrara

Anno Accademico 2016 - 2017

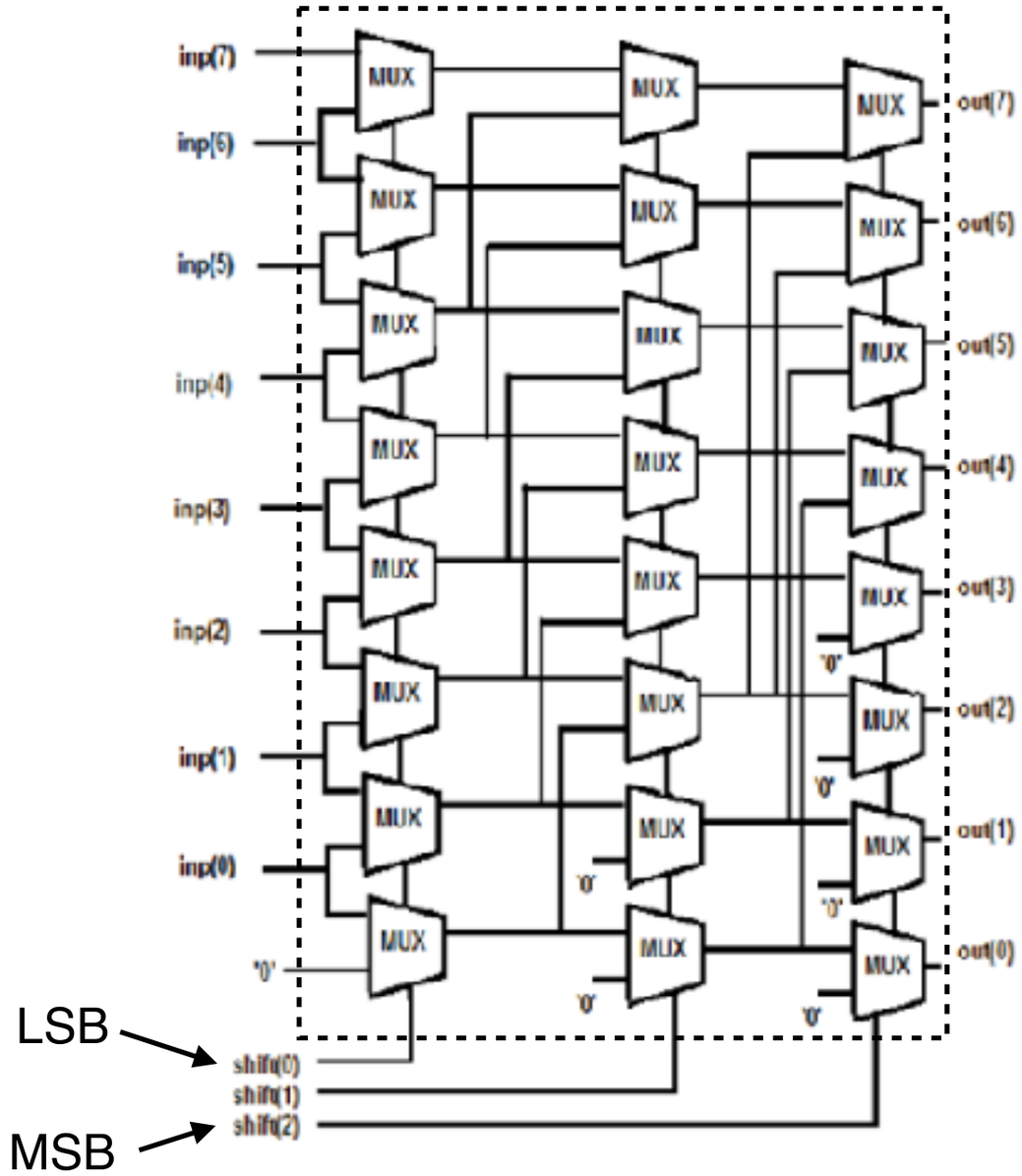
Componenti visti

- Nelle scorse lezioni abbiamo visto come sono realizzati i seguenti componenti:



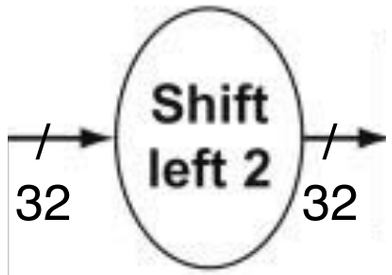
Shifter

- Componente che implementa l'operazione di "shift" verso destra (o sinistra).
- E' composto da una catena di Mux.
- In figura è mostrato un circuito che implementa lo shift a sinistra su 8 bit:
  - 3 livelli (3 bit di controllo)
- Esempio:
  - input → $(00110001)_{due}$
  - control → $(010)_{due}$
  - output → $(11000100)_{due}$



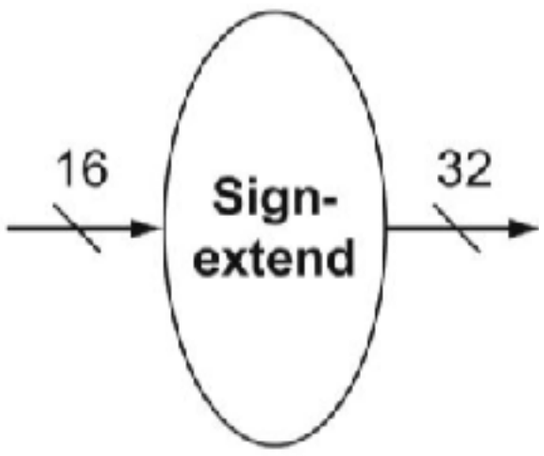
Shifter

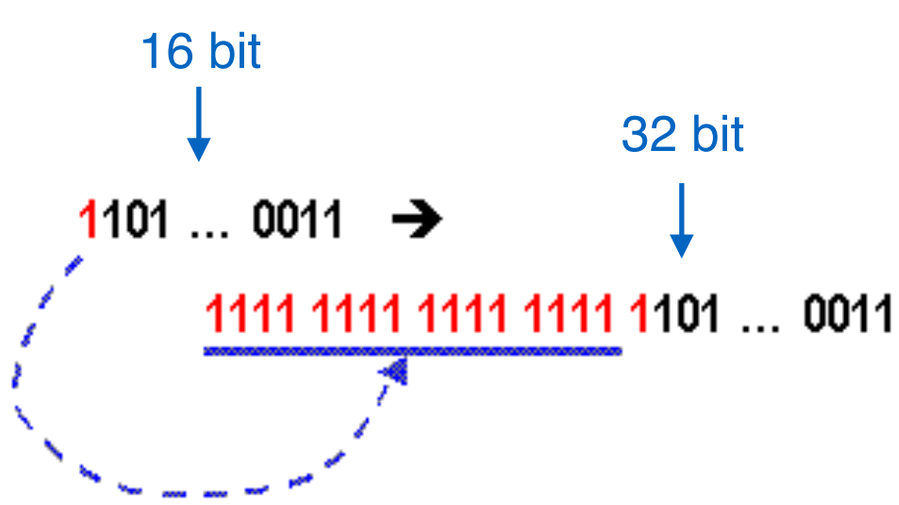
- Nel data path ci servirà un circuito per moltiplicare per 4 gli indirizzi (es. calcolo dell'istruzione successiva).
- Possiamo utilizzare uno shifter a sinistra di 2:
  - 32 bit di input e output → 5 bit di controllo ($2^5 = 32$).
  - bit di controllo hardcoded: $(00010)_{due}$



Bit extender

- Un circuito con N bit in ingresso e M bit in uscita (dove N < M).
- Il circuito interpreta i bit in ingresso come rappresentazione binaria di un numero con segno e ne estende i bit:
  - se il numero è positivo i bit aggiunti saranno 0
  - se il numero è negativo i bit aggiunti saranno 1





Il processore MIPS

- Vedremo un'implementazione semplificata del processore MIPS.
- Copre un sottoinsieme limitato di istruzioni rappresentative dell'ISA:
  - aritmetiche/logiche: add, sub, and, or, slt
  - accesso alla memoria: lw, sw
  - trasferimento di controllo: beq, j
- Ad ogni ciclo di clock viene eseguita una istruzione completa.
- Assumiamo che la memoria che tiene le istruzioni (read-only) e la memoria che tiene i dati (read/write) siano **separate**.
- Possiamo leggere e scrivere nello stesso ciclo di clock.

Cosa fa un processore

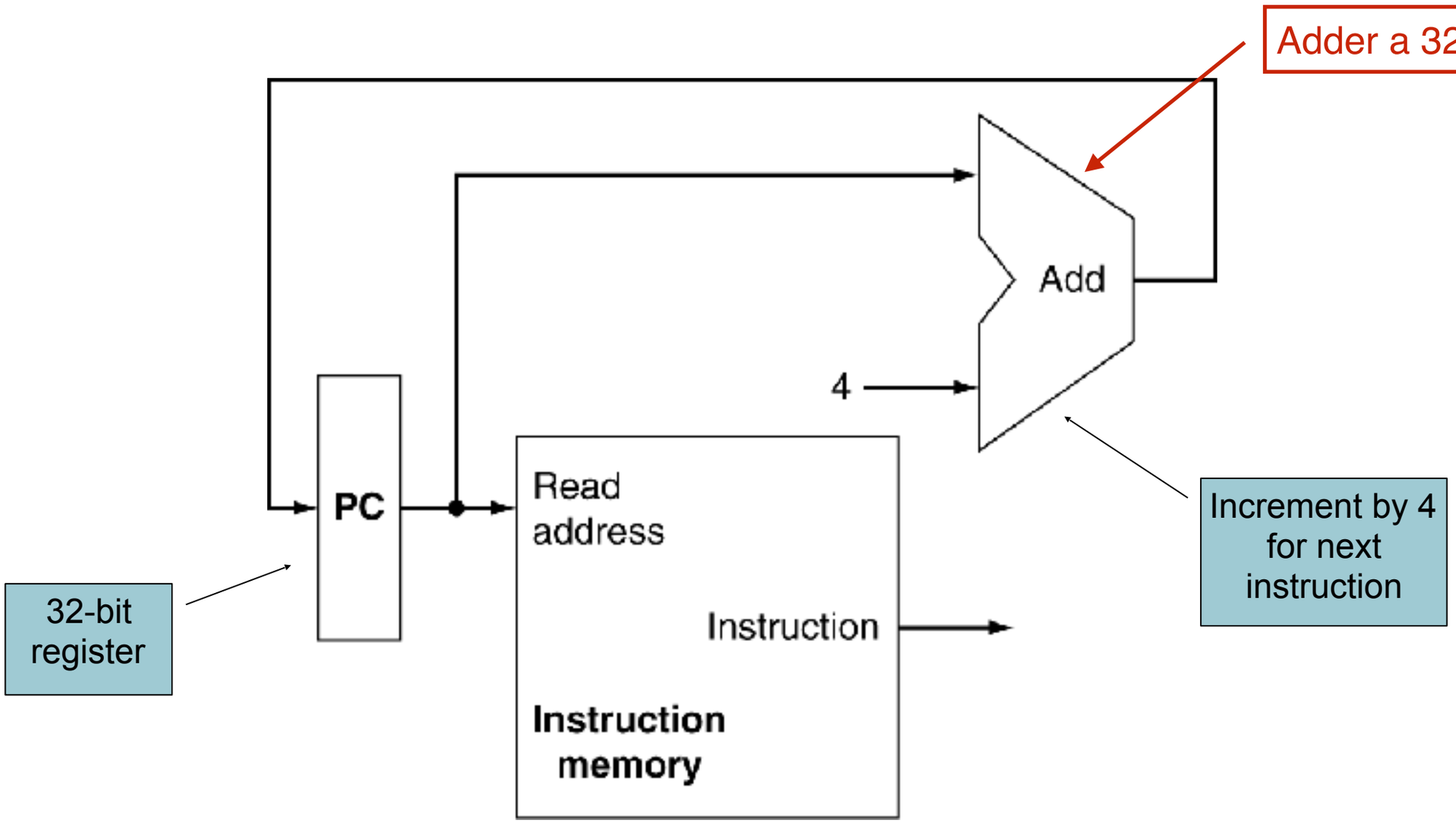
- Per ogni istruzione:
  - legge dalla memoria l'istruzione il cui indirizzo è nel PC
  - legge dei registri (in base a quanto specificato dall'istruzione)
- A seconda dell'istruzione:
  - usa la ALU per calcolare uno tra i seguenti valori:
    - un risultato aritmetico/logico
    - un indirizzo di memoria (registro + offset)
    - l'indirizzo di un branch target
  - accede alla memoria per load/store
  - aggiorna il PC con PC + 4 o con un indirizzo target

Lettura istruzione

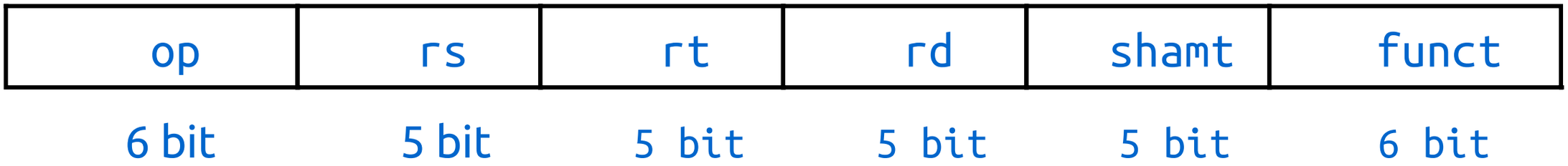
- Per eseguire un'istruzione è necessario prima leggerla dalla memoria.
- PC è un registro dedicato che contiene l'indirizzo dell'istruzione attualmente in esecuzione.
- Escludendo casi di branch o jump, tipicamente la prossima istruzione da eseguire è quella successiva alla corrente.
- Ogni istruzione è rappresentata da un blocco di 32 bit (4 bye).
- L'istruzione successiva a quella corrente si troverà quindi all'indirizzo PC + 4.

Lettura istruzione

- Circuito per l'incremento del PC (ad ogni ciclo di clock viene aggiunto 4 all'indirizzo corrente):

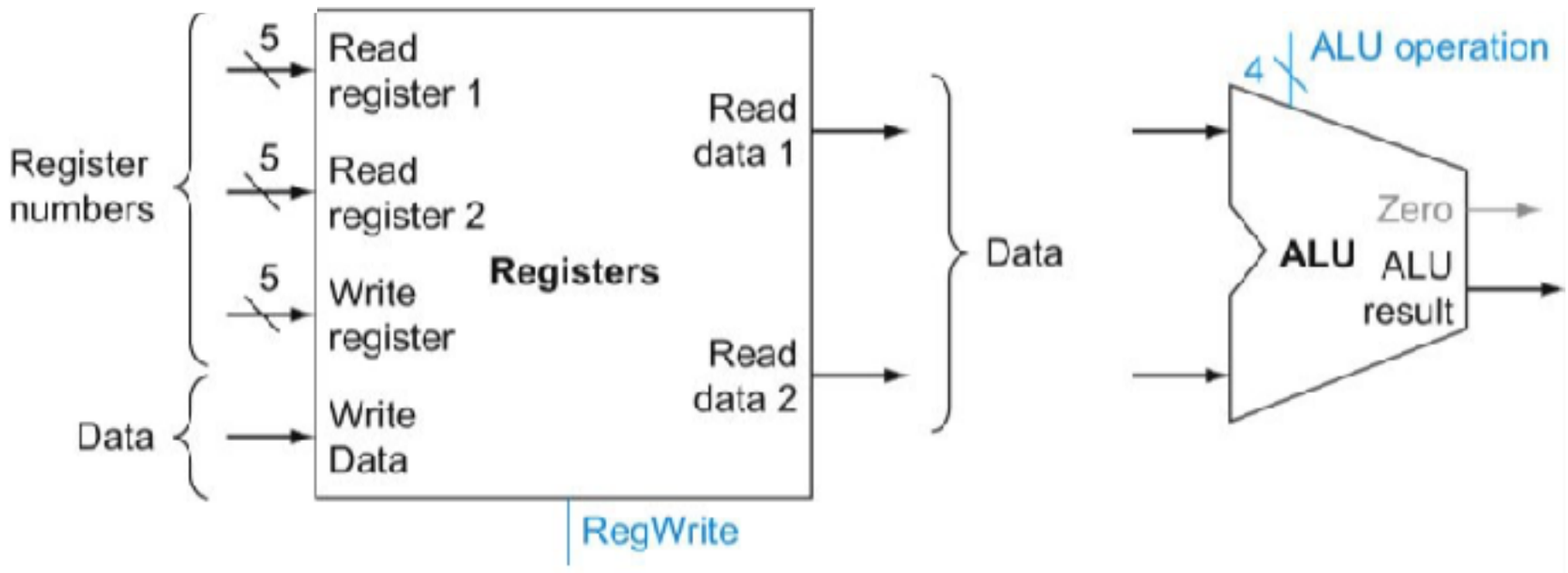


Istruzioni di tipo R



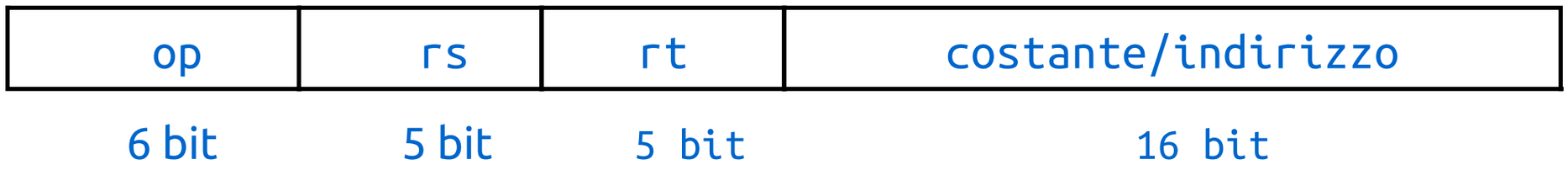
- L'istruzione viene letta dalla memoria (es. add $s0, $t0, $t1).
- Vengono selezionati i due registri in lettura (rs e rt) e il registro in scrittura (rd).
- Vengono generati segnali di controllo dell'ALU.
- Vengono letti i valori dei registri sorgenti e dati in input alla ALU.
- L'ALU esegue l'operazione.
- Il valore in output dalla ALU viene scritto nel registro destinazione.

Istruzioni di tipo R



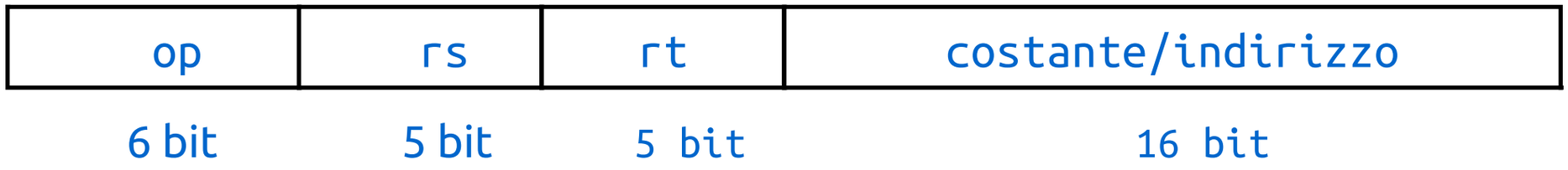
- I segnali di controllo (in blu) verranno affrontati successivamente.
- Sfruttiamo il fatto che il Register File permette di leggere e scrivere nello stesso ciclo di clock (write after read):
  - possiamo collegare l'output dell'ALU come input del "Write Data"

Istruzioni load



- L'istruzione viene letta dalla memoria (es. lw $t0, 0($a0)).
- Il valore contenuto nel registro rs viene mandato come primo operando all'ALU.
- L'offset viene esteso a 32 bit con segno ed inviato all'ALU come secondo operando.
- Vengono generati segnali di controllo dell'ALU.
- L'ALU esegue l'operazione, calcolando l'indirizzo a cui accedere.
- Il dato è letto dalla memoria e scritto nel registro rt.

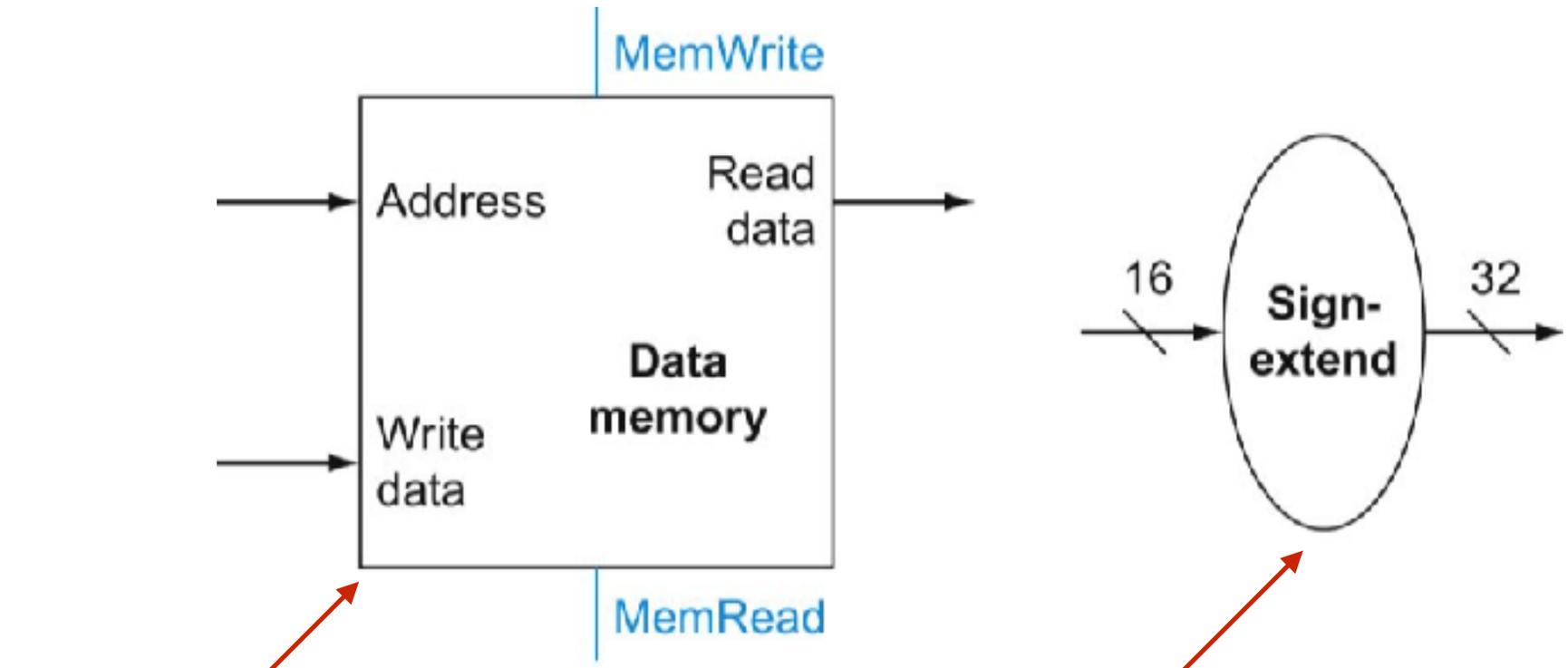
Istruzioni store



- L'istruzione viene letta dalla memoria (es. sw $t0, 0($a0)).
- Il valore contenuto nel registro rs viene mandato come primo operando all'ALU.
- L'offset viene esteso a 32 bit con segno ed inviato all'ALU come secondo operando.
- Vengono generati segnali di controllo dell'ALU.
- L'ALU esegue l'operazione, calcolando l'indirizzo a cui accedere.
- Il dato contenuto nel registro rt è scritto nella memoria principale all'indirizzo selezionato.

Istruzioni load/store

- Componenti necessari (il Register File e la ALU li abbiamo già):

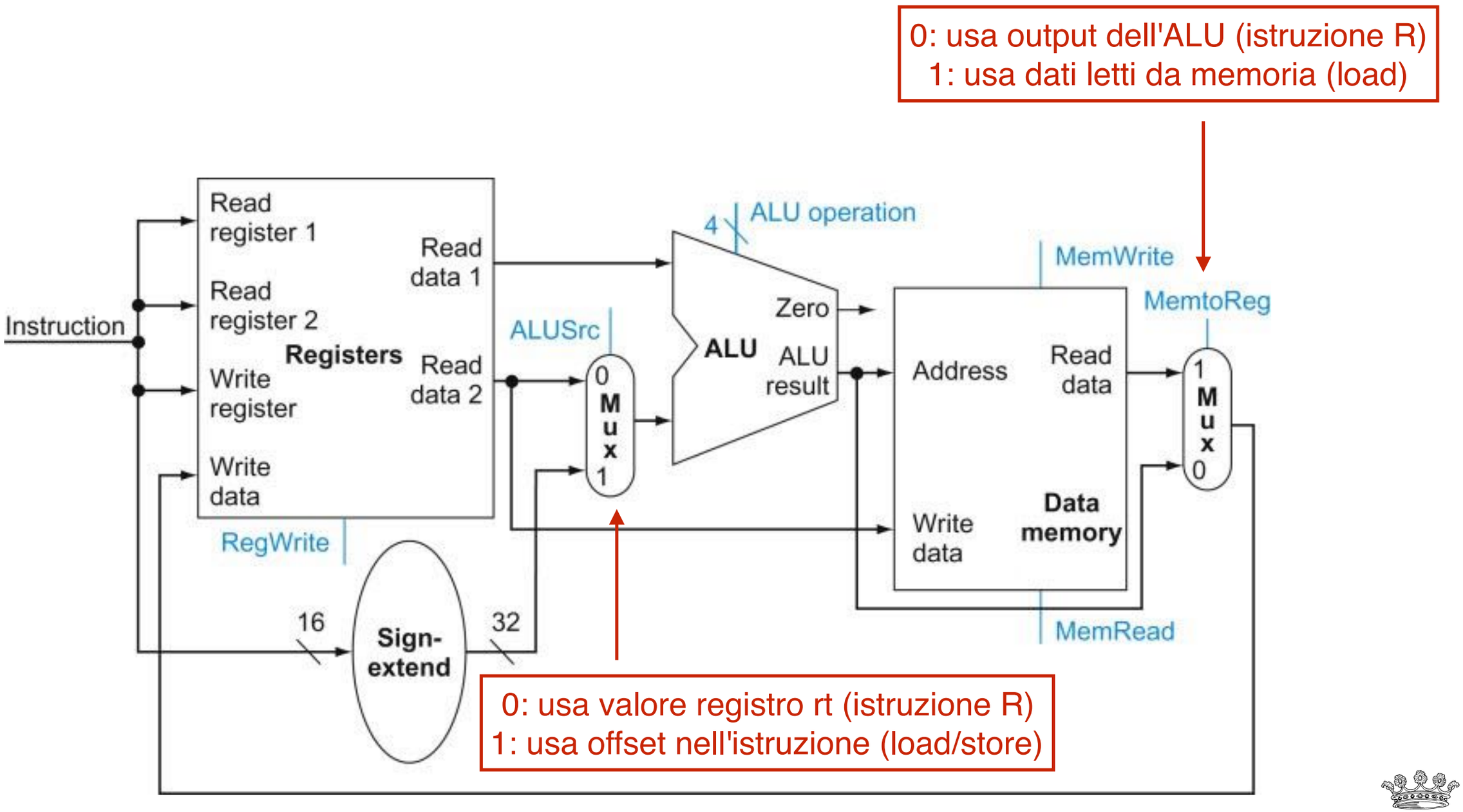


Memoria principale

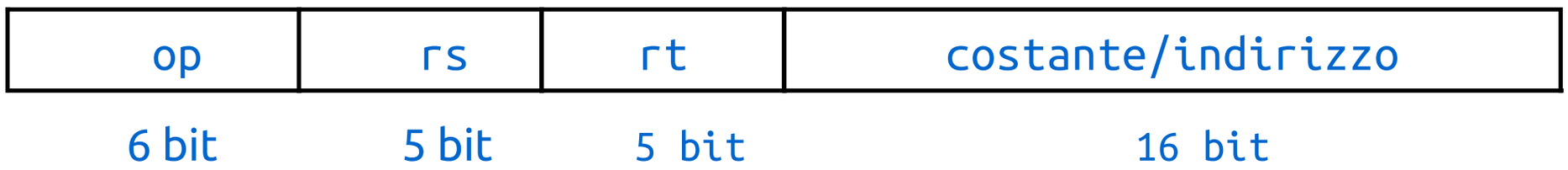
Estensione da 16 a 32 bit (con segno)

Istruzioni load/store

- Mettiamo tutto assieme (escludendo il PC):



Istruzioni di branch (beq)



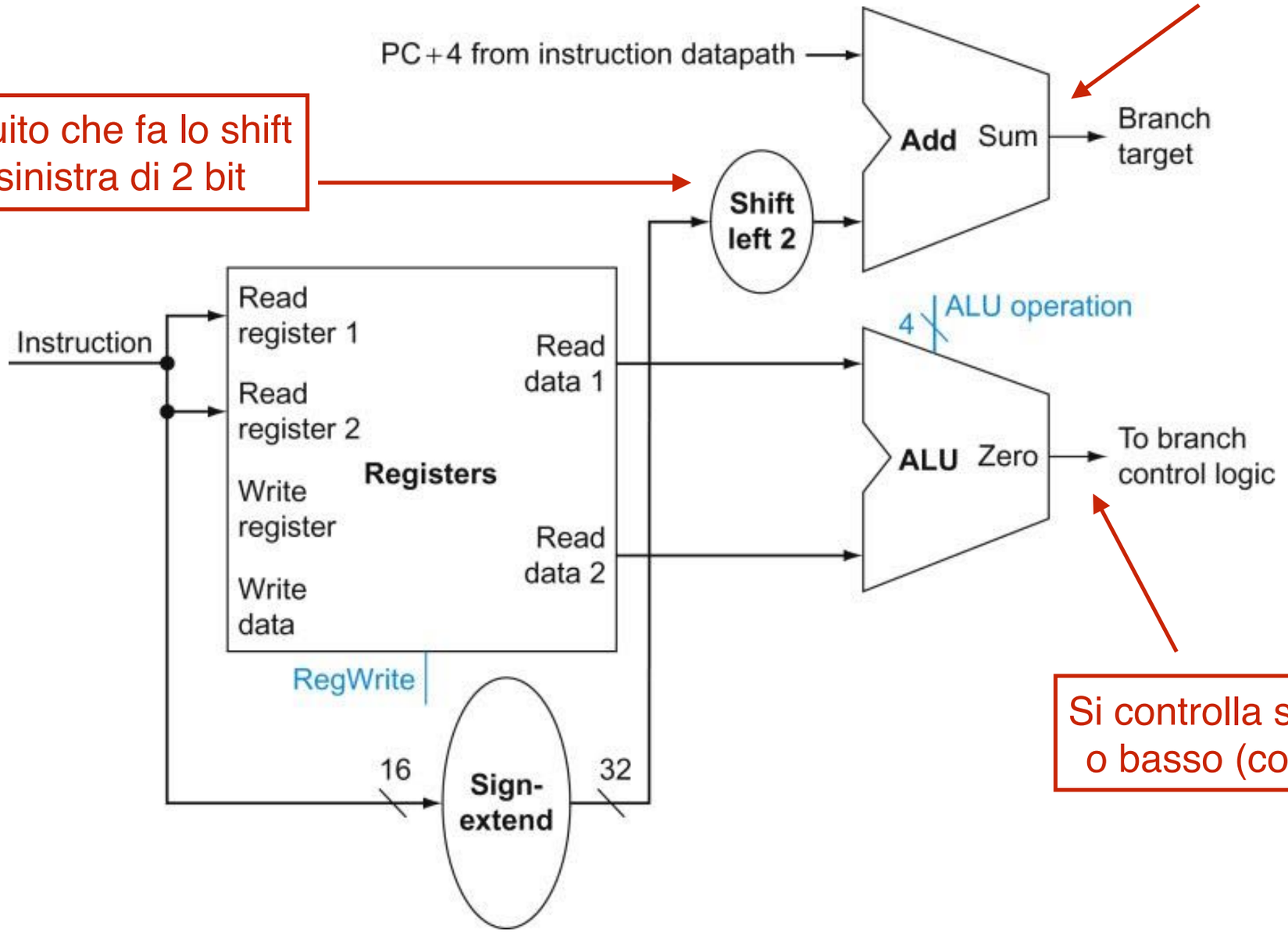
- L'istruzione viene letta dalla memoria.
- L'offset viene shiftato a sinistra di 2 bit e esteso a 32 bit con segno.
- L'indirizzo risultante viene sommato con PC + 4 (adder aggiuntivo).
- I due registri in lettura (rs e rt) vengono passati alla ALU come operandi.
- La ALU calcola la differenza tra i due operandi.
- Viene interpretato il segnale "zero" della ALU per capire se il salto deve o meno avvenire.

Istruzioni di branch (beq)

- Componenti necessari:

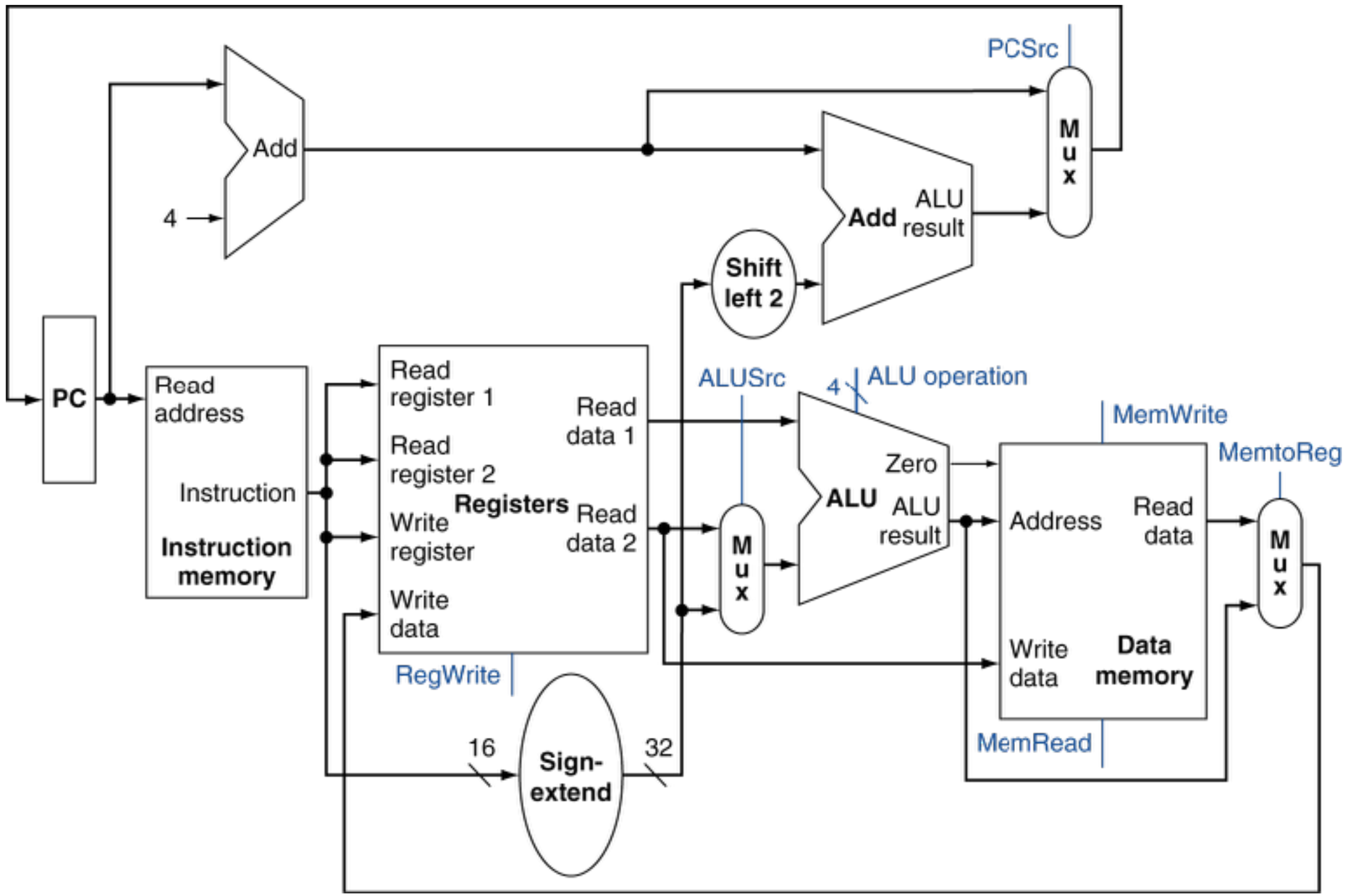
L'indirizzo a cui saltare viene sempre calcolato

Circuito che fa lo shift a sinistra di 2 bit



Si controlla se il segnale "zero" è alto o basso (con beq se è alto si salta)

Datapath completo



Unità di controllo

- Il datapath che abbiamo costruito ha 7 segnali di controllo:
  - RegWrite, ALUSrc, PCSrc, MemWrite, MemtoReg, MemRead → 1 bit
  - ALU Operation → 4 bit
- L'unità di controllo del processore ha il compito di generare (direttamente o indirettamente) quei segnali.
- Segue una logica combinatoria:
  - l'input è dato dall'istruzione (i 6 bit del campo op)
  - in output genera i segnali di controllo
- E' composta da due parti: una di controllo generico ed una specifica per l'ALU.

Controllo della ALU

- L'unità di controllo della ALU deve generare in output i segnali di controllo per la ALU (ALU operation):

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

- Per le istruzioni di tipo R la funzione è data dal codice funct nell'istruzione.
- Per le istruzioni load/store la funzione è sempre add.
- Per la funzione beq la funzione è sempre subtract.

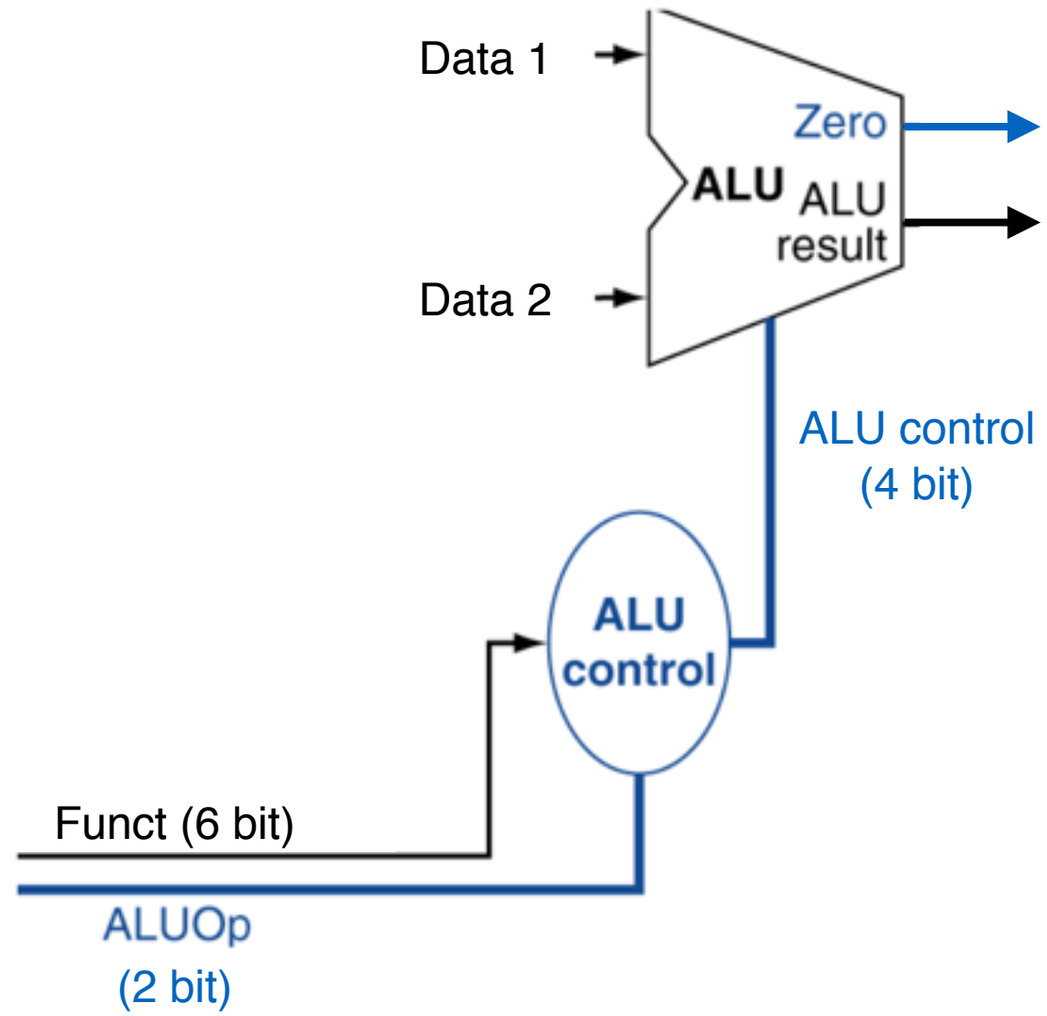
Controllo della ALU

- L'input all'unità di controllo della ALU è dato da:
  - due bit che indicano a che categoria appartiene l'istruzione (ALUOp)
  - il codice funct dell'istruzione per le istruzioni R

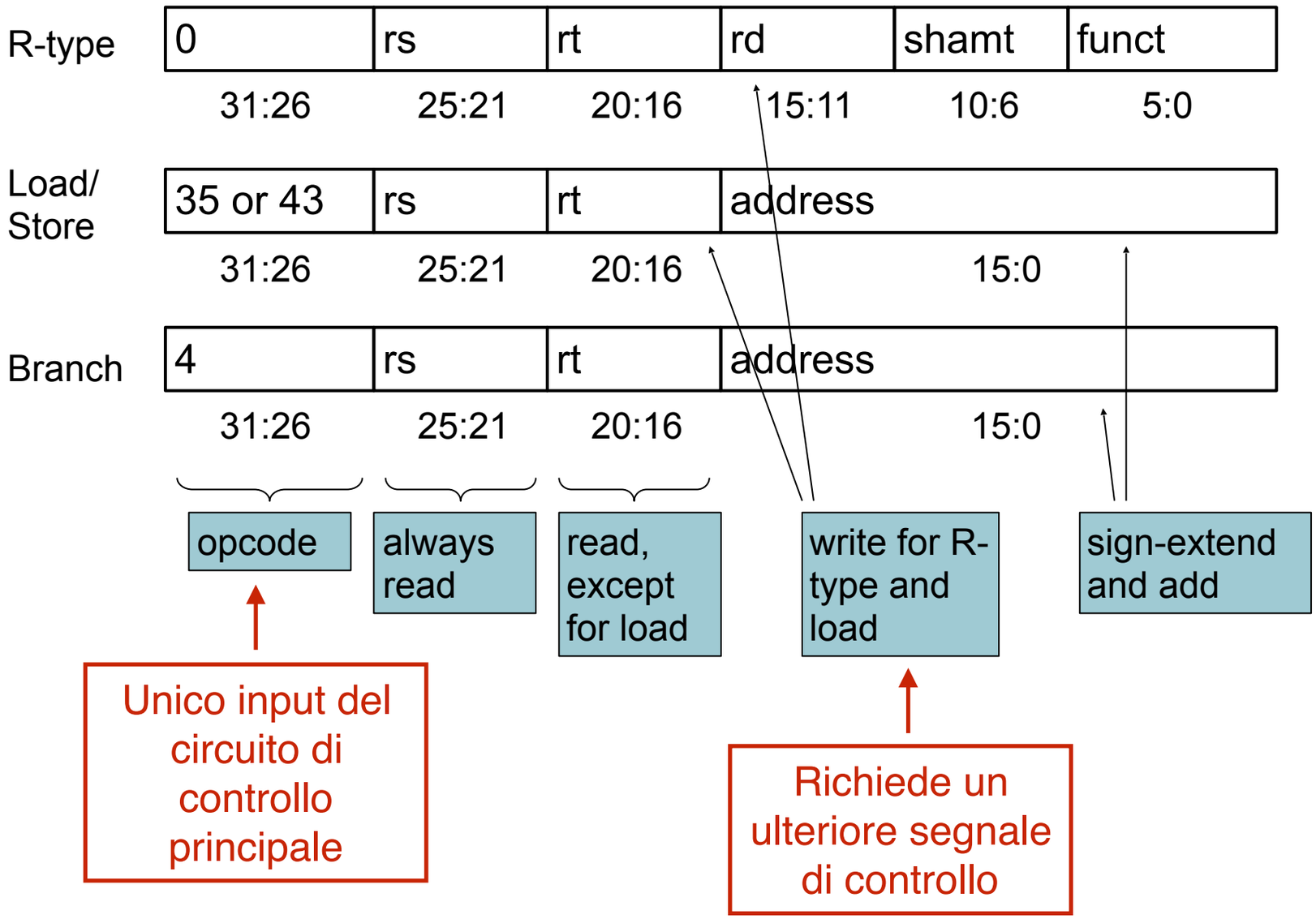
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

- I bit ALUOp sono settati dall'unità di controllo principale.

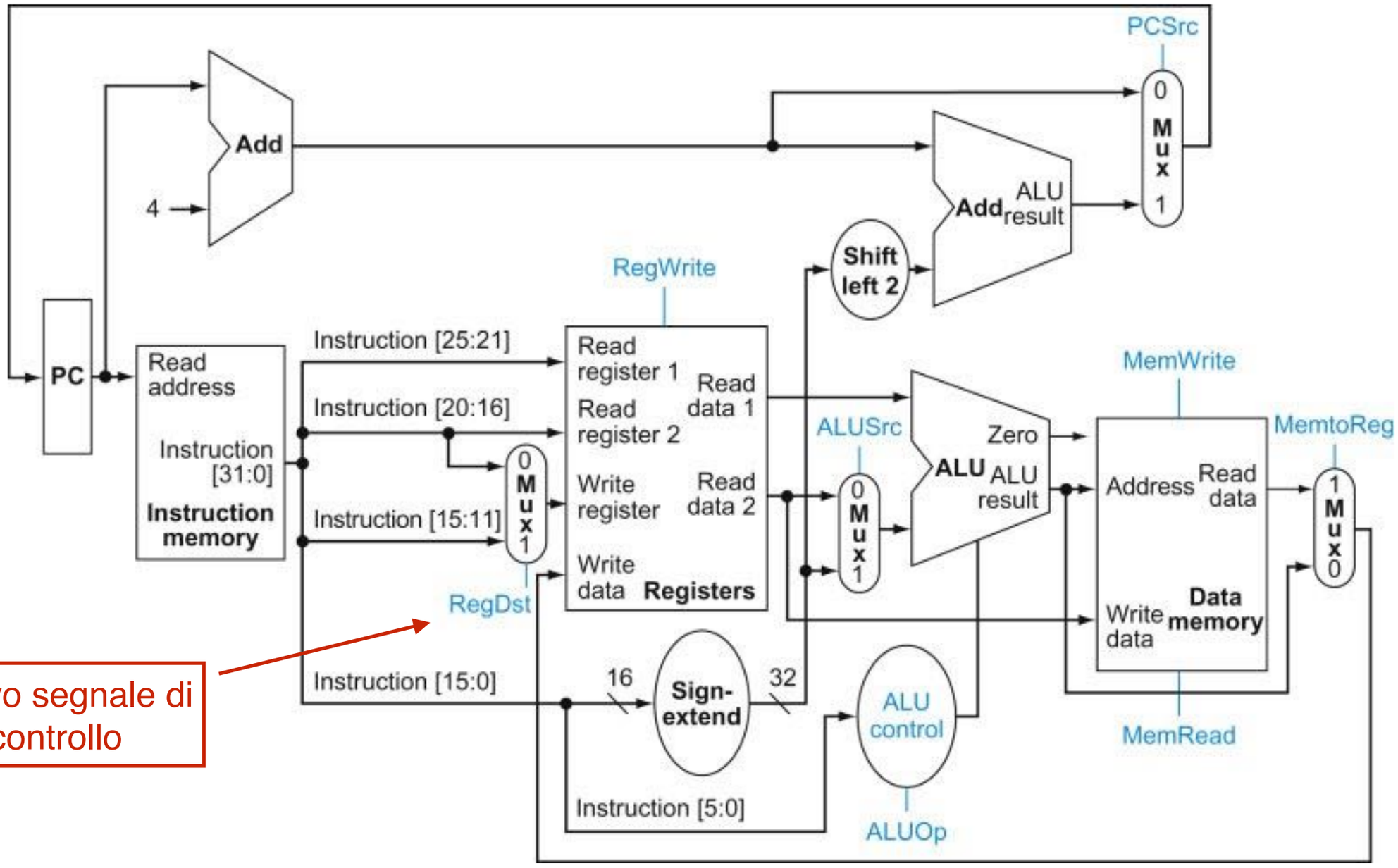
Controllo della ALU



Controllo e istruzioni

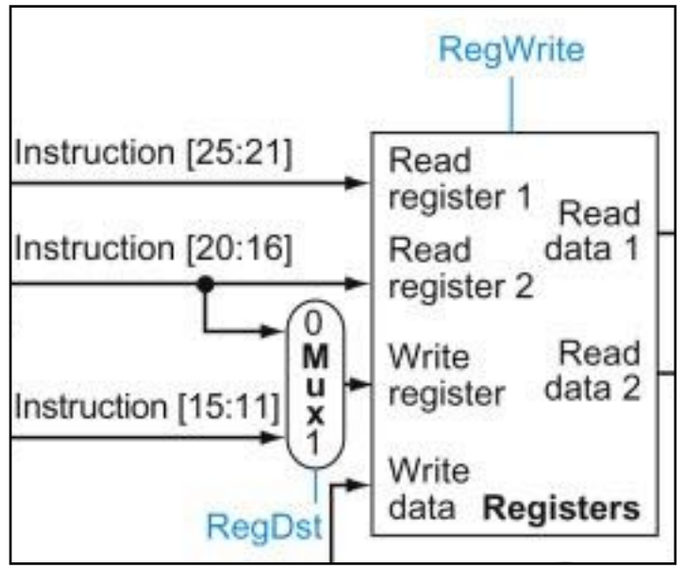


Controllo principale



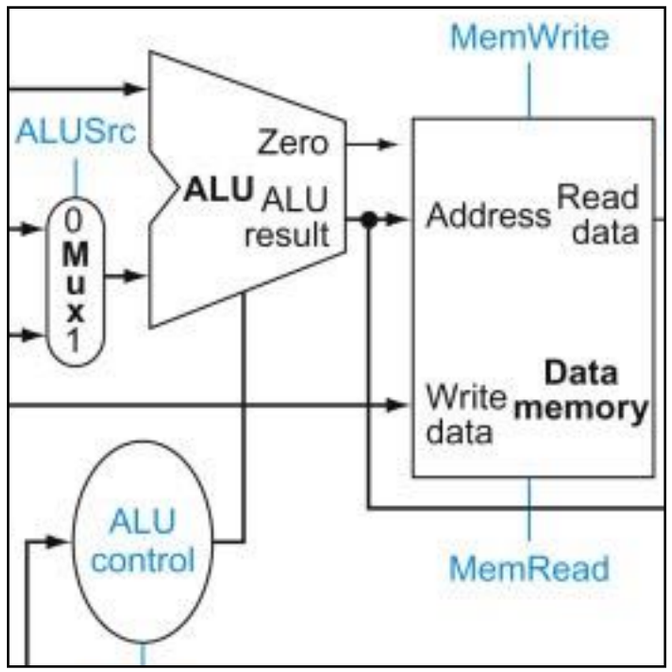
Segnali di controllo

- **RegDst**:
  - 0 → il numero del registro su cui scrivere deriva dal campo rt (bits 20:16)
  - 1 → il numero del registro su cui scrivere deriva dal campo rd (bits 15:11)
- **RegWrite**:
  - 0 → nessun effetto
  - 1 → il registro specificato in “Write register” viene scritto con il valore in “Write data”



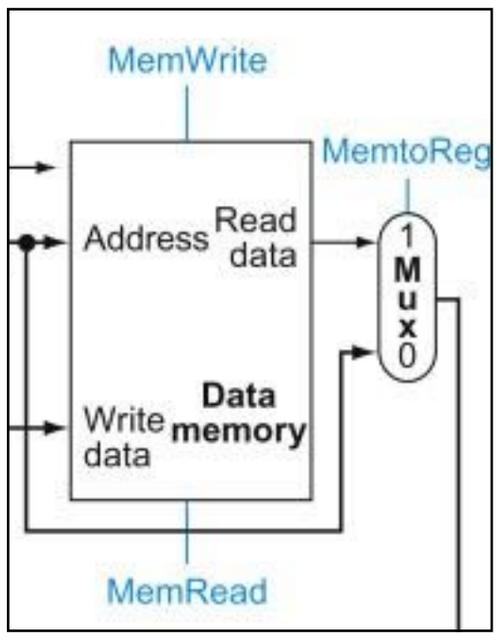
Segnali di controllo

- **ALUSrc**:
  - 0 → il secondo operando dell'ALU deriva dal secondo registro in lettura del Register File
  - 1 → il secondo operando dell'ALU è l'offset con segno ed esteso a 32 bit (bits 15:0)
- **MemRead**:
  - 0 → nessun effetto
  - 1 → il dato presente in memoria principale all'indirizzo letto nell'input "Address" viene fornito in output su "Read data"



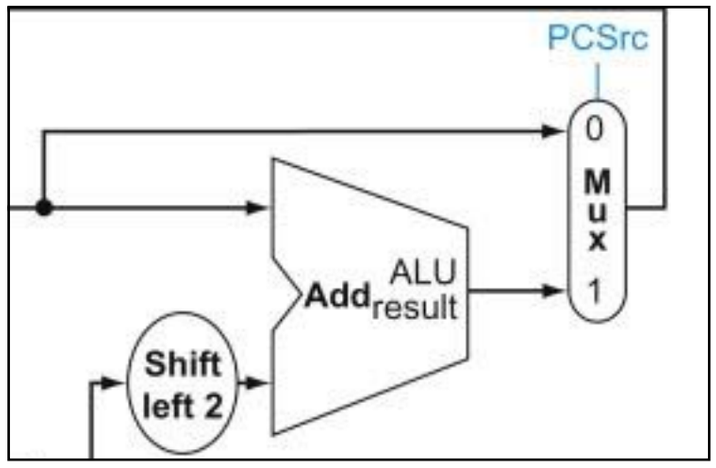
Segnali di controllo

- **MemWrite**:
  - 0 → nessun effetto
  - 1 → il dato presente nell'input "Write data" viene scritto in memoria principale all'indirizzo letto nell'input "Address"
- **MemtoReg**:
  - 0 → il valore passato all'input "Write data" è il risultato dell'ALU
  - 1 → il valore passato all'input "Write data" è il valore letto dalla memoria principale



Segnali di controllo

- **PCSrc**:
  - 0 → il PC diventa PC + 4
  - 1 → il PC diventa il risultato della somma tra PC e l'offset
- Ma PCSrc vale 1 quando:
  - stiamo eseguendo un'operazione di branch
  - il segnale "zero" dell'ALU è alto
- Si può quindi ottenere PCSrc mettendo in AND questi due controlli:
  - l'unità di controllo avrà un segnale "Branch" che indica se si sta eseguendo una operazione di branch



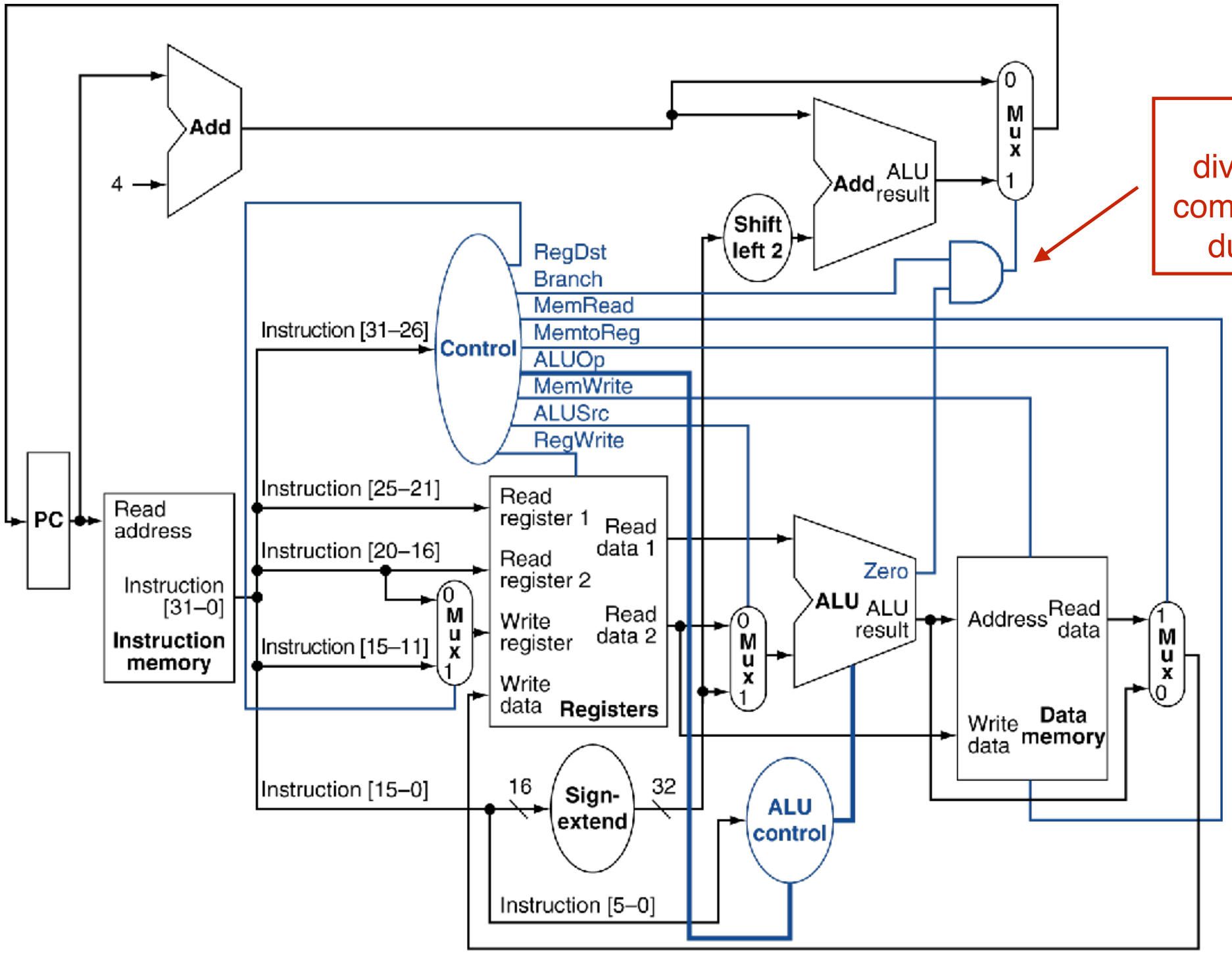
Controllo principale

- L'input dell'unità di controllo principale è costituito esclusivamente dall'opcode dell'istruzione:

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

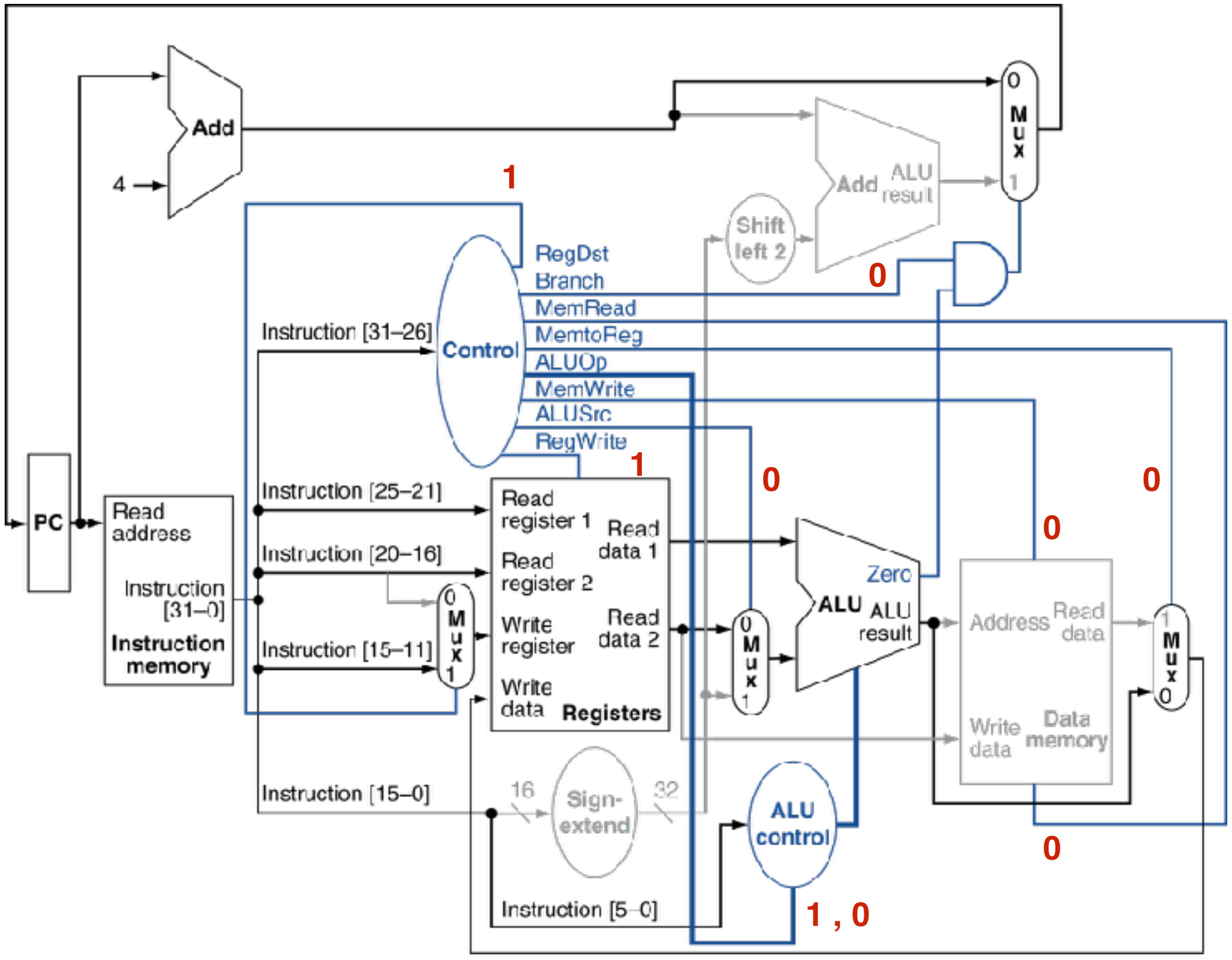
- X significa "don't care" (non importa, non significativo)

Datapath con controllo

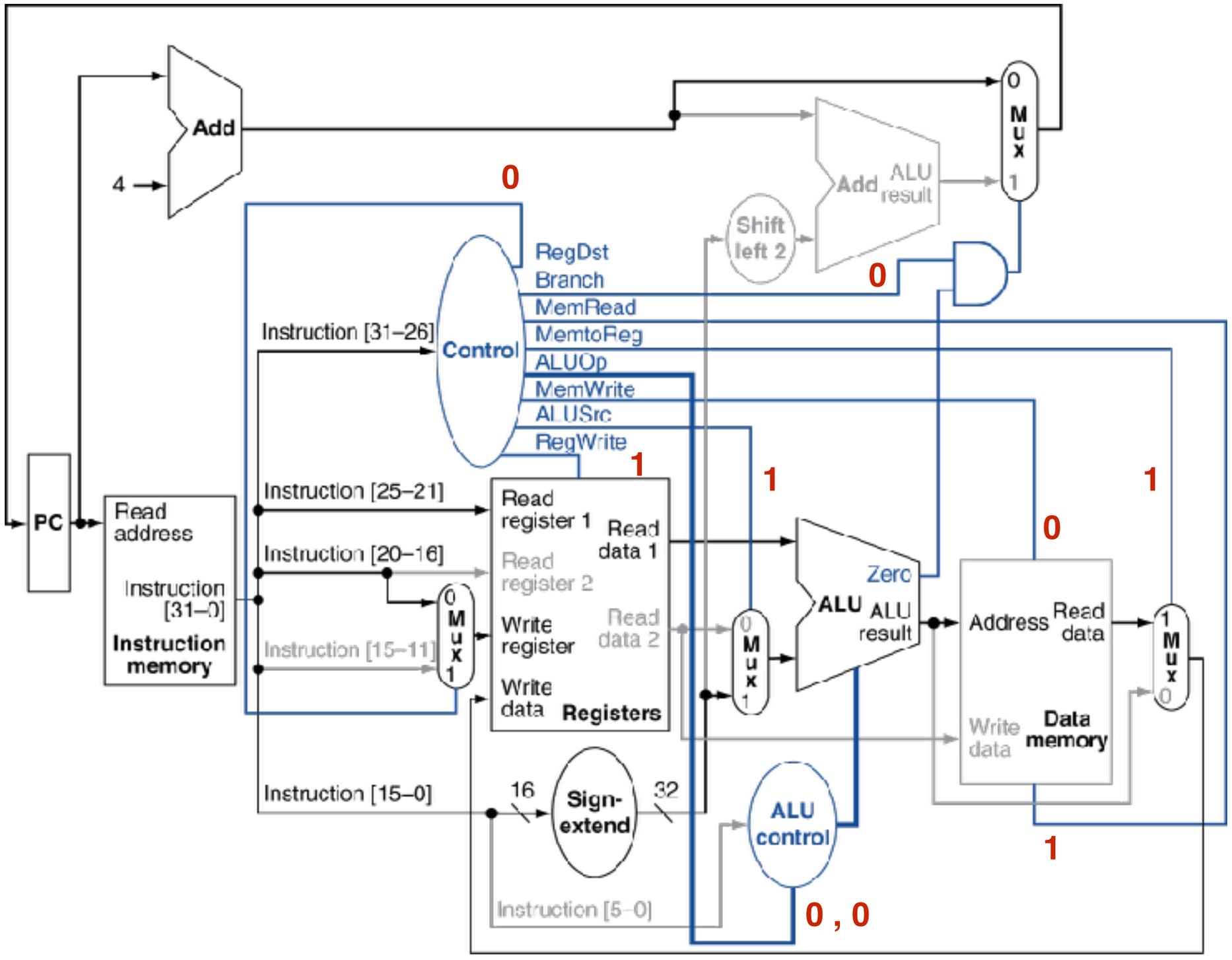


PCSrc è diventato una combinazione di due segnali

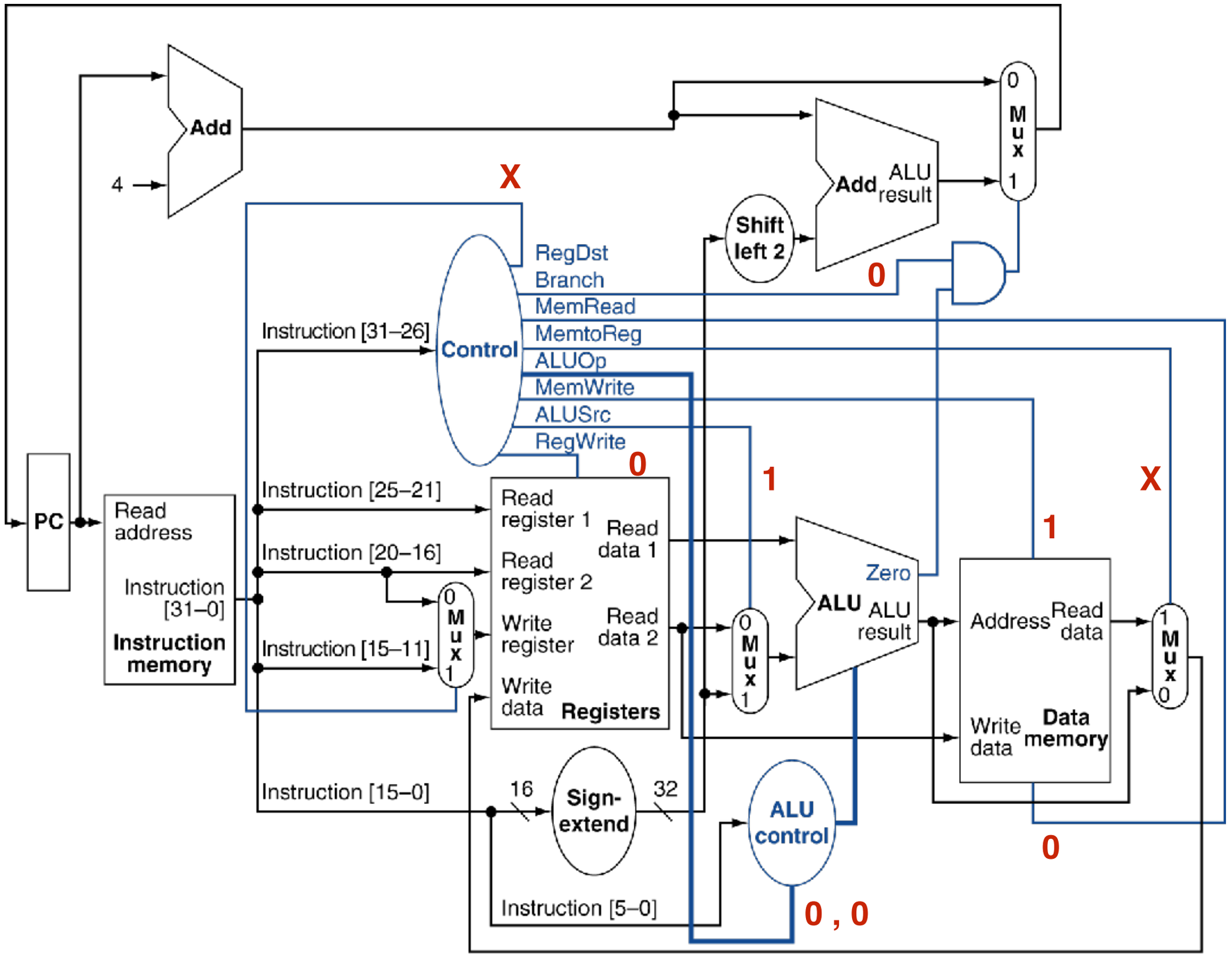
Istruzioni di tipo R



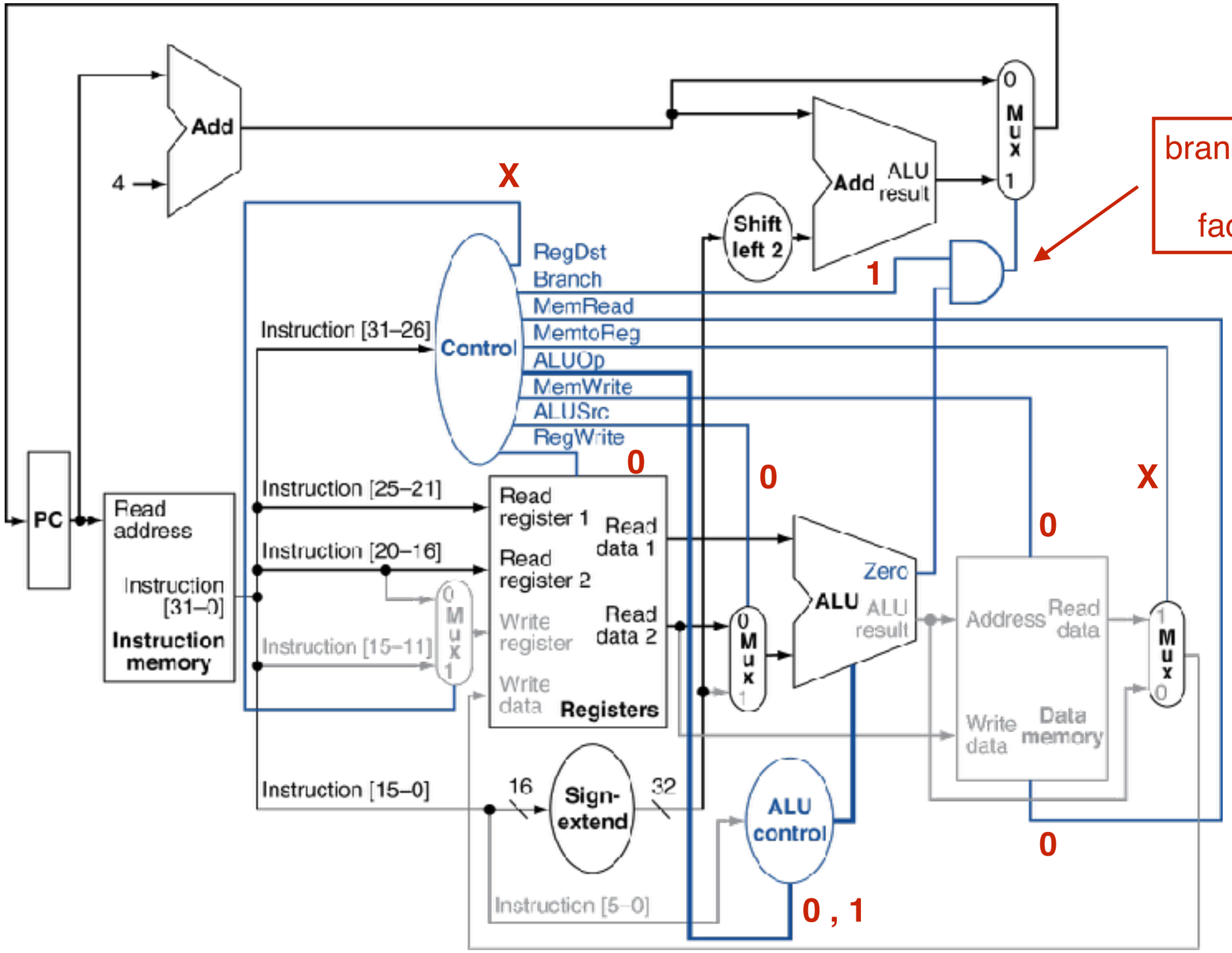
Load


Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
0
1
1
1
0
1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory
0 , 0

Store

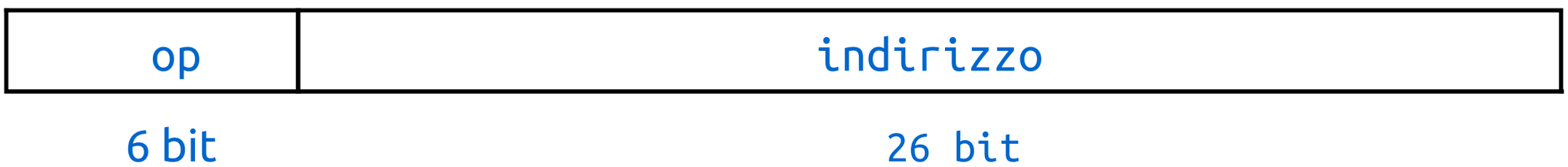

Add
4
X
0
M
u
x
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
0
1
X
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory
1
0
0 , 0

Branch-on-equal



Jump incondizionato

- Formato J:



- L'op del jump è $(000010)_{due}$.
- Il PC viene aggiornato con una concatenazione di:
  - 4 bit più significativi presi dal PC
  - 26 bit di indirizzo
  - 2 bit meno significativi sono sempre 0

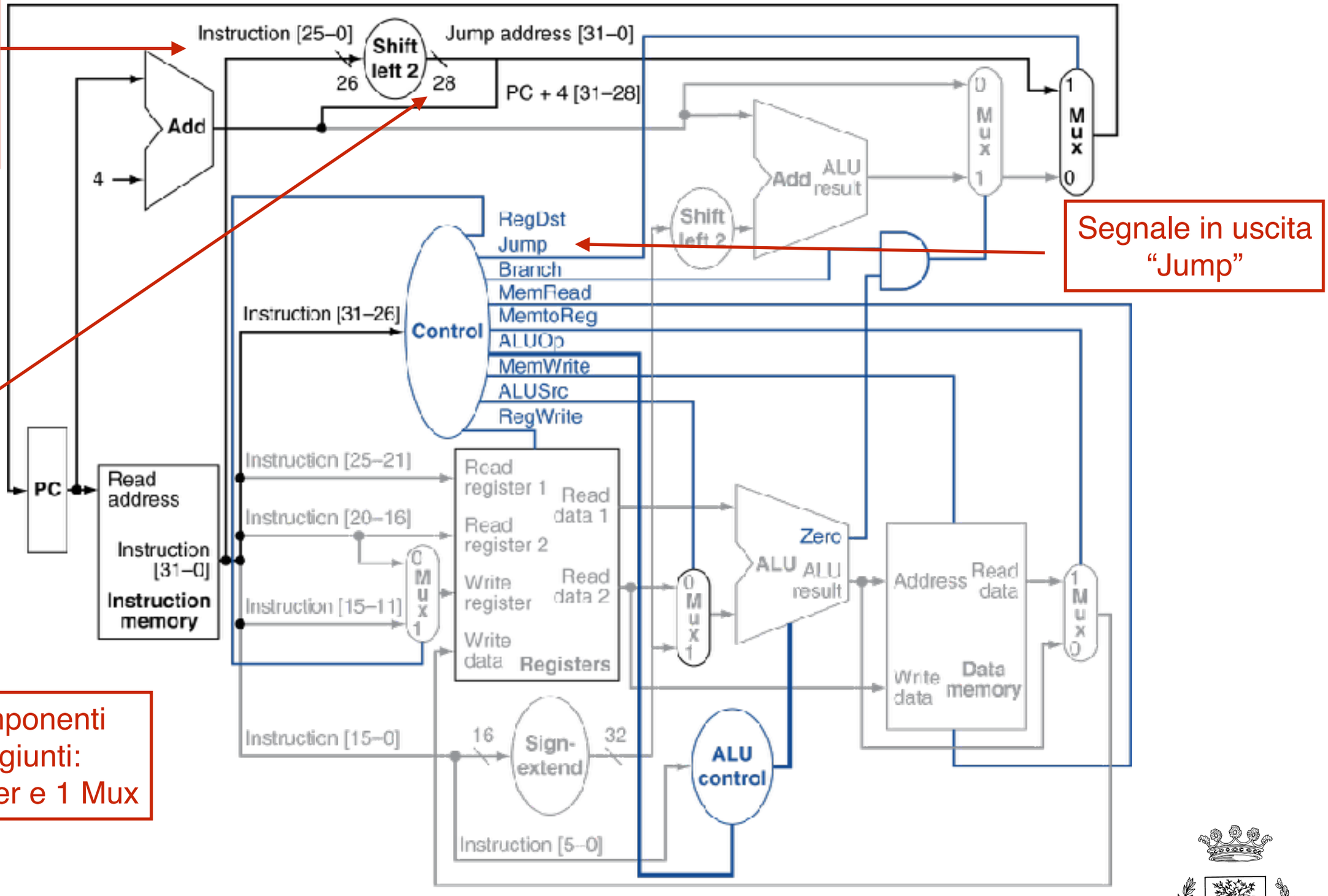
Jump incondizionato

Calcolo del nuovo indirizzo (prendo i 4 bit più alti dal PC)

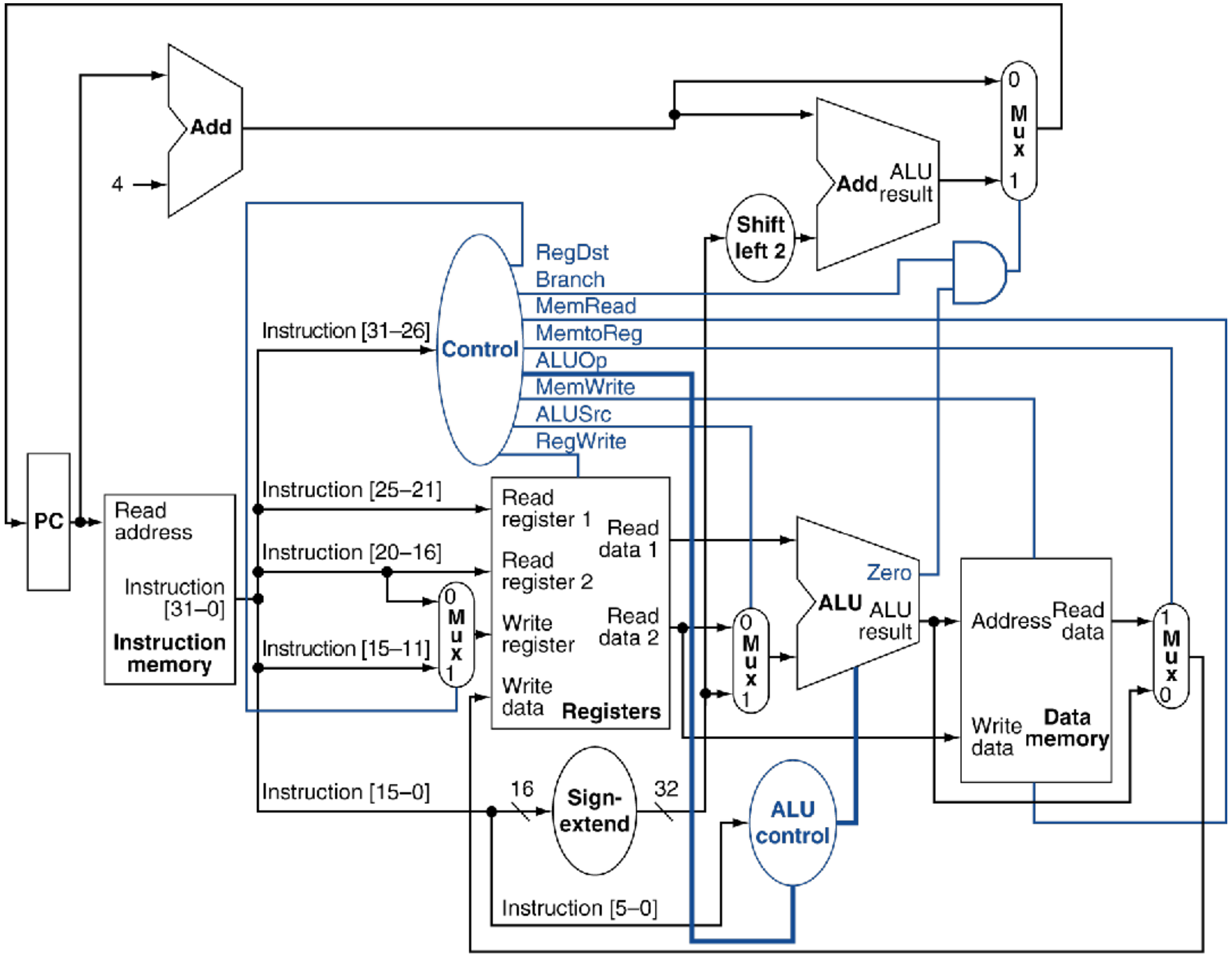
Shift di 2 bit + estensione di 2 bit

Segnale in uscita "Jump"

Componenti aggiunti: 1 Shifter e 1 Mux



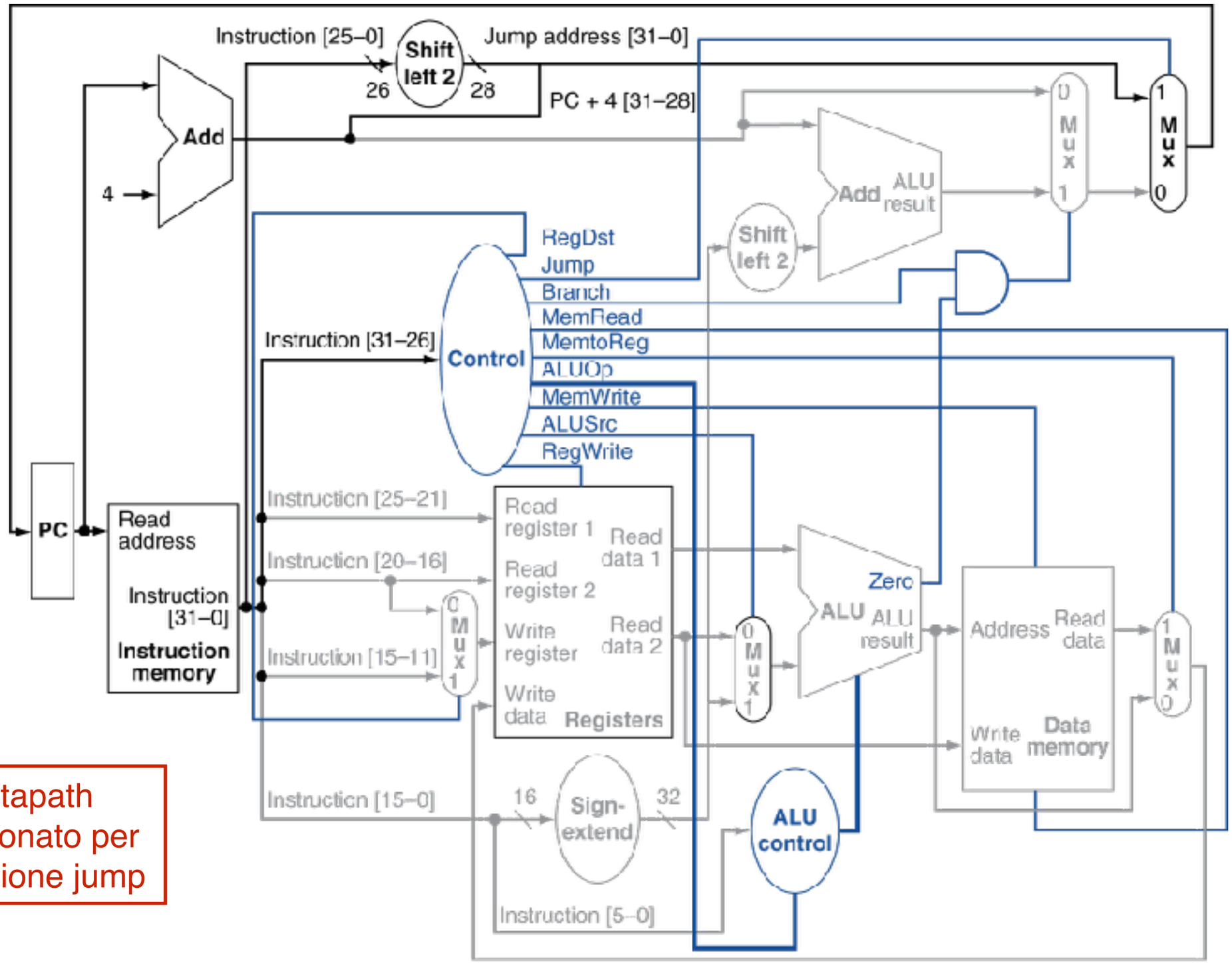
Come implementare addi?



Implementazione di addi

- E’ una istruzione di tipo I (come load e store).
- Il suo comportamento è molto simile alla load:
  - si somma il valore di un registro con una costante
- Il risultato della ALU **non** deve essere interpretato come un indirizzo in memoria:
  - MemtoReg → 0 (nella load è 1)
  - MemRead → 0 (nella load è 1)
- Basta modificare il comportamento della logica di controllo per farle “riconoscere” anche l’opcode di addi $(001000)_{due}$

Come implementare jal?

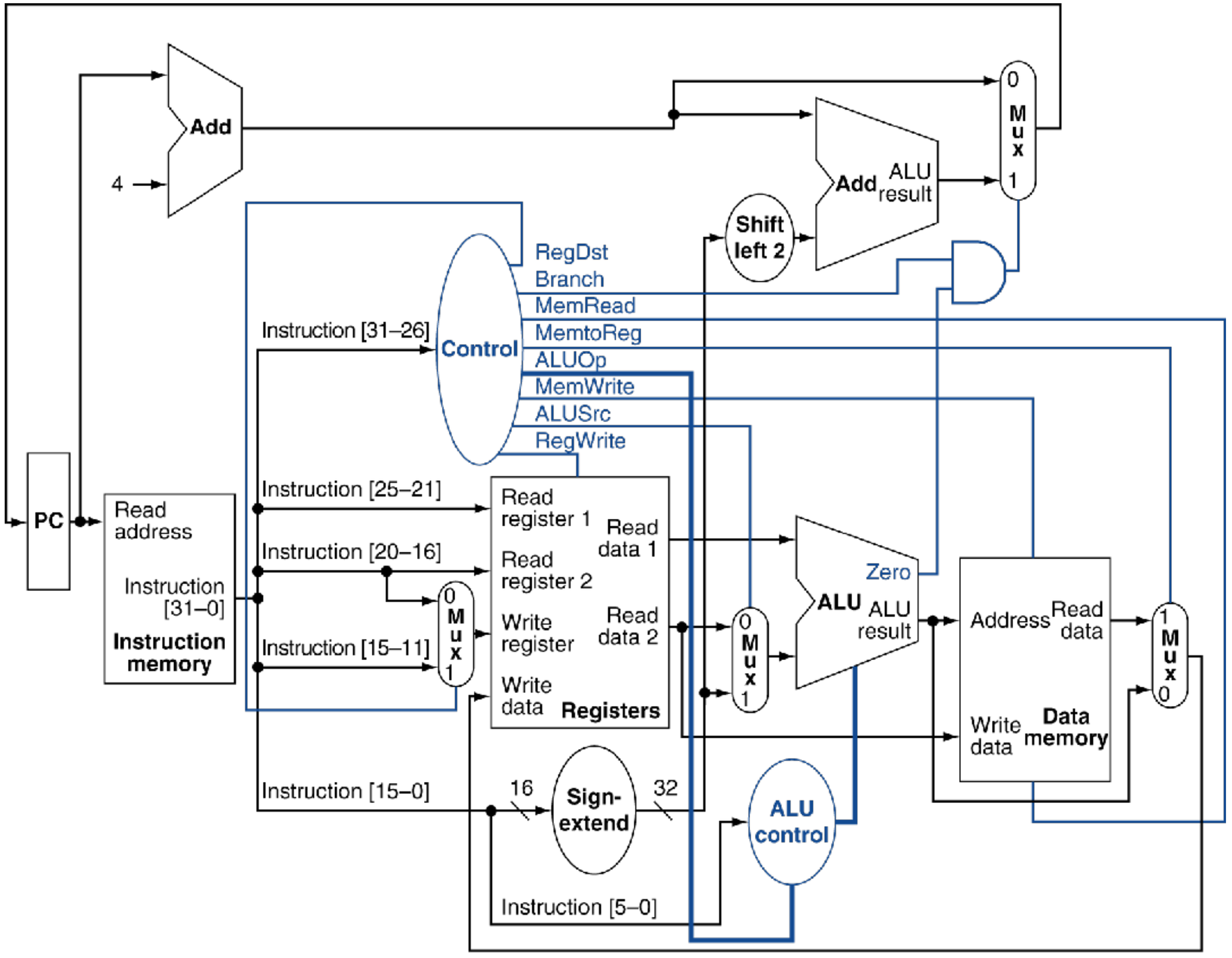


Datapath selezionato per l'istruzione jump

Implementazione di jal

- E’ una istruzione di tipo J (come jump).
- A differenza di jump deve scrive il valore PC + 4 nel registro $ra (31).
- Devo collegare l’output dell’adder di PC + 4 a “Write data” del Register File:
  - aggiungo un input al Mux di “MemtoReg” (il controllo passa da 1 a 2 bit)
- Devo poter scrivere il valore 31 nell’input “Write register”:
  - aggiungo un input al Mux di “RegDst” (il controllo passa da 1 a 2 bit)
- La logica di controllo deve essere in grado di riconoscere l’opcode di jal $(000011)_{due}$

Come implementare jr?



Implementazione di jr

- E’ una istruzione di tipo R.

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	sssss	00000	00000	00000	001000

- Viene codificata passando il numero del registro specificato in rs (tipicamente $ra, ovvero 31) all’input “Read register 1” del Register File.
- rt, rd e shamt non sono utilizzati.
- Il valore letto dal Register File è l’indirizzo a cui saltare, quindi viene scritto nel PC (nuovo collegamento al Mux del PC).
- Necessari collegamenti e controlli aggiuntivi.

Le eccezioni

- Fino ad ora abbiamo visto l'esecuzione di istruzioni nell'area "utente" della memoria (indirizzo < 0x80000000).
- Sappiamo che possono esserci errori durante l'esecuzione del programma (accesso ad un'area di memoria non consentita, divisione per zero, etc.).
- In caso di errore l'elaboratore smette di eseguire il programma e salta ad un'area del kernel dedicata:
  - l'indirizzo 0x80000000 contiene la prima istruzione di un programma del kernel chiamato "exception handler"
  - in caso di errore il PC deve essere aggiornato con quell'indirizzo (nuovo collegamento al Mux del PC)
- Necessari collegamenti e controlli aggiuntivi.

E le altre istruzioni?

- Il datapath visto ora permette solo l'esecuzione di un sottoinsieme di istruzioni MIPS:
  - scopo "didattico" (come nel libro di testo)
  - approccio utilizzato anche per altri argomenti
- Per permettere l'esecuzione di altre istruzioni è necessario aggiungere componenti hardware e complicare la logica di controllo (come visto per addi).
- Nelle slides successive vedremo una possibile implementazione del calcolo di prodotto e divisione:
  - non andremo in dettaglio come negli argomenti precedenti

Prodotto

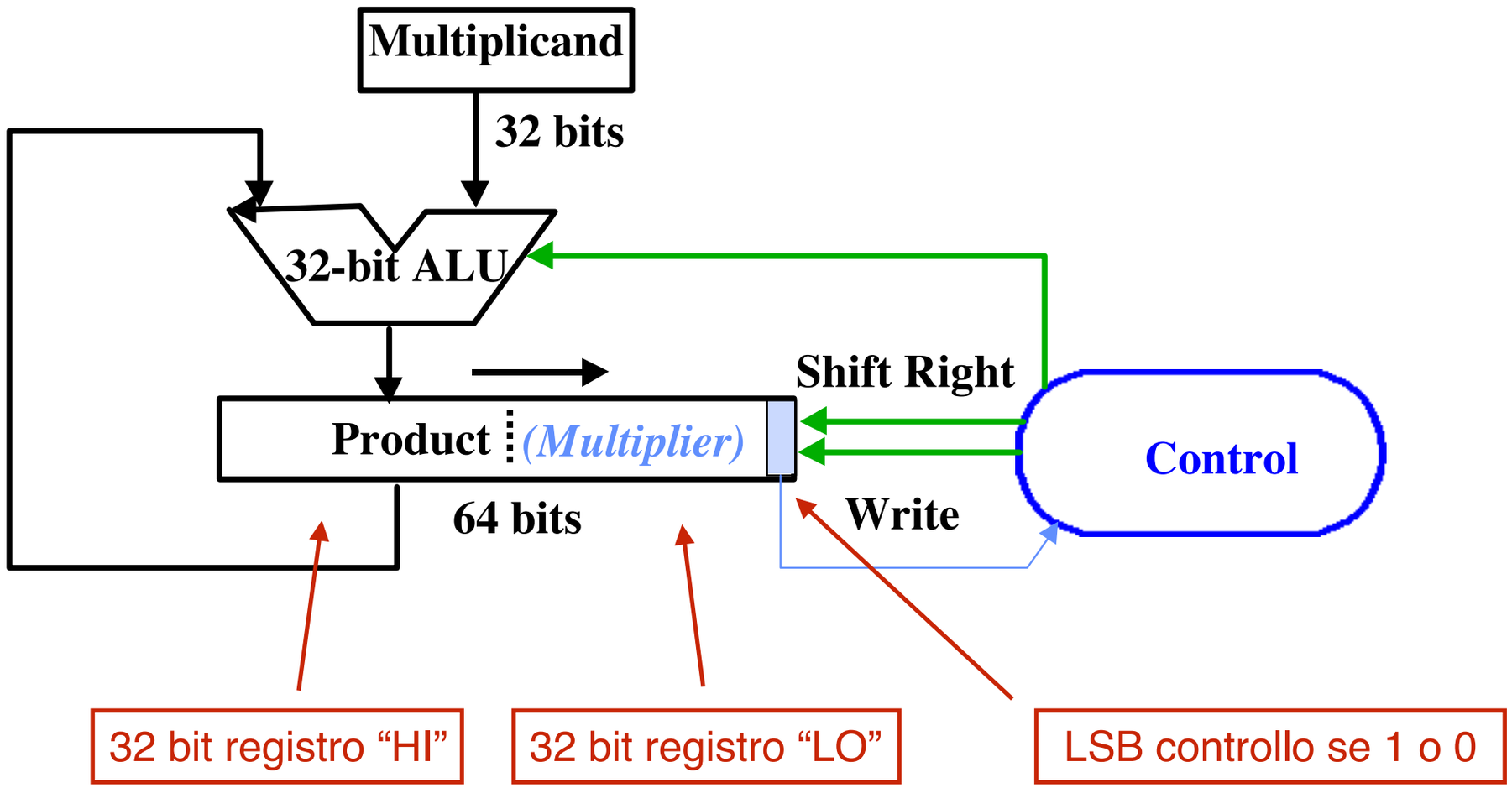
- Il prodotto può essere implementato attraverso una serie di somme e shift (come visto nelle scorse lezioni):

```
 1110 x    // moltiplicando
 0110 =    // moltiplicatore
 ————————
 0000 +
1110  +
 1110   +
0000    =
————————————
1010100      // prodotto
```

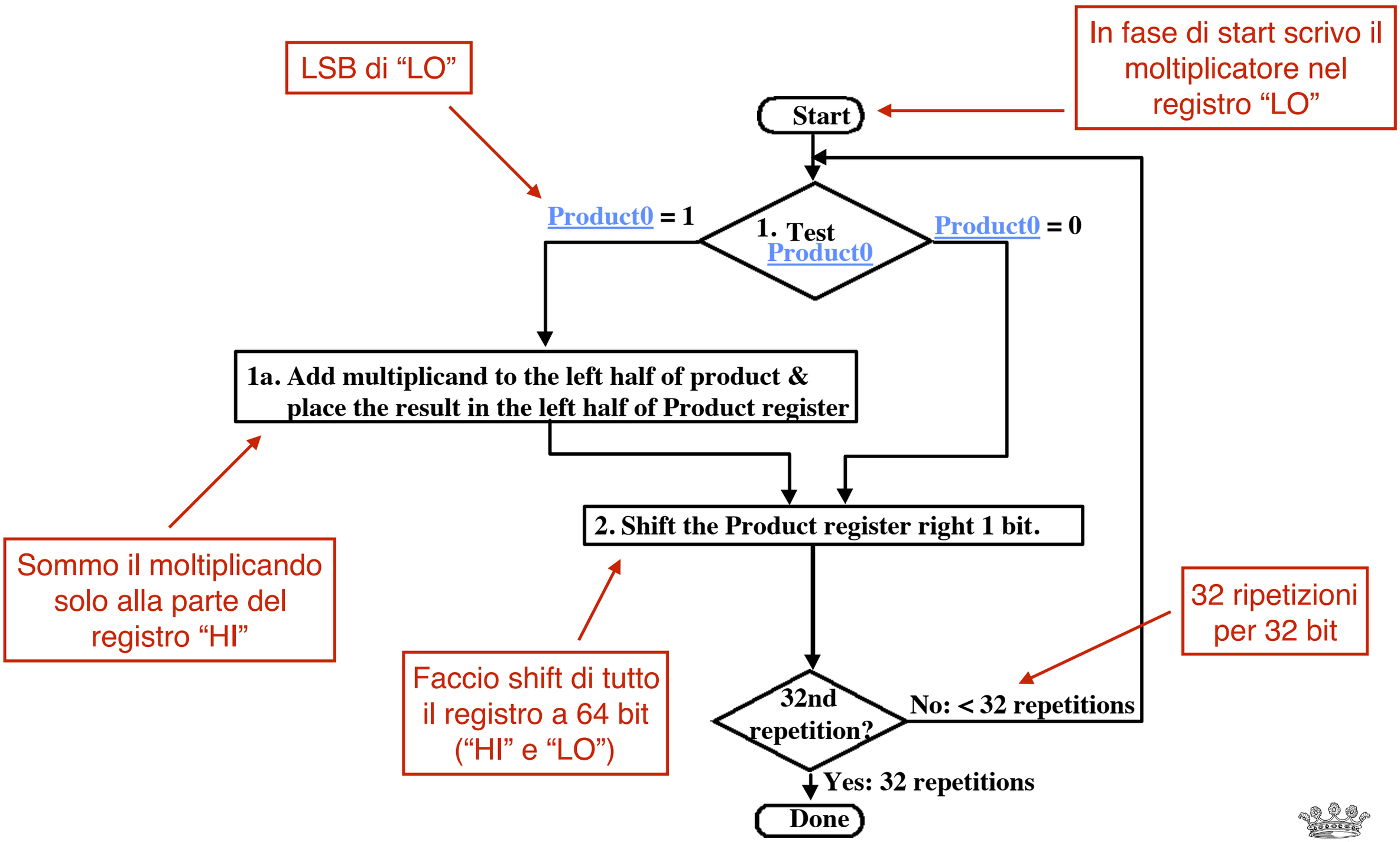
- Ogni risultato parziale o è zero o è il moltiplicando spostato progressivamente a sinistra.
- La moltiplicazione si riduce ad una serie di somme e spostamenti a sinistra.

Prodotto

- Possibile implementazione in hardware del prodotto (istruzione MULT):



Prodotto


LSB di “LO”
In fase di start scrivo il moltiplicatore nel registro “LO”
Start
Product0 = 1
1. Test Product0
Product0 = 0
1a. Add multiplicand to the left half of product & place the result in the left half of Product register
Sommo il moltiplicando solo alla parte del registro “HI”
2. Shift the Product register right 1 bit.
Faccio shift di tutto il registro a 64 bit (“HI” e “LO”)
32 ripetizioni per 32 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Divisione

- Il circuito visto nelle slides precedenti per il prodotto può essere utilizzato anche per implementare la divisione attraverso una serie di differenze:
  - modifiche nella ALU
  - modifiche nella logica di controllo

Ricapitolando

- Un datapath contiene i componenti e i collegamenti necessari ad implementare un ISA (Instruction Set Architecture).
- L'implementazione vista è:
  - a singolo ciclo di clock (1 ciclo → 1 istruzione)
  - a memorie separate (istruzioni e dati)
  - con un ISA ridotto rispetto al MIPS classico
  - a 32 bit
- La logica di controllo dice al datapath che cosa fare:
  - si basa su una logica combinatoria
  - riceve in input l'istruzione binaria (la parte significativa)

MARS

- Aprite MARS:
- Scrivete qualche istruzione, ad esempio:

```
1  ori $t0, $zero, 5
2  ori $t1, $zero, 7
3  add $t2, $t1, $t0
```

- Salvate
- Andate su Tools → MIPS X-Ray
- Assemblate
- Cliccate su “Connect to MIPS”
- Eseguite

Performance

- Come si definisce il concetto di “performance”?
- Tempo di esecuzione di un programma:
  - **wall-clock time** → tiene conto anche di I/O e delle attese dovute alla condivisione delle risorse con altri programmi
  - **CPU time** → tiene conto solo del tempo in cui il programma usa la CPU
- Throughput:
  - numero di task/transazioni eseguiti per unità di tempo
- Altro…

CPU time

$$\text{CPU time} = \#\text{instructions} \cdot \text{CPI} \cdot \text{Clock cycle}$$

- #instructions: numero di istruzioni nel programma
- CPI: numero medio di cicli di clock per istruzione
- Clock cycle: durata di un ciclo di clock

Istruzioni

- Non siamo interessati alla quantità “statica” di istruzioni, ovvero alle linee di codice del programma.
- Quello che dobbiamo considerare è la quantità dinamica di istruzioni, ovvero quante istruzioni vengono realmente eseguite durante l’esecuzione del programma.
- Il codice seguente contiene 3 linee, ma il numero di istruzioni eseguite sarà 2001:

```
          li   $a0, 1000
ostrich:  sub  $a0, $a0, 1
          bne  $a0, $0, ostrich
```

CPI

- Il numero medio di cicli di clock per istruzione (CPI) dipende sia dal processore che dal programma:
  - un programma con molte istruzioni floating point può avere un CPI più alto di un programma che utilizza solo interi
  - un processore può eseguire lo stesso programma molto più velocemente rispetto ad un altro processore meno performante.
- Nel datapath descritto nelle slides precedenti abbiamo assunto che una istruzione venga eseguita in un ciclo di clock, quindi il CPI è 1.
- In generale:
  - CPI > 1 a causa di stalli sulla memoria o istruzioni lente
  - CPI < 1 su macchine che eseguono più istruzioni nello stesso ciclo di clock

Ciclo di clock

- Un ciclo di clock è l'unità di tempo minima richiesta dalla CPU per fare qualcosa:
  - il tempo del ciclo di clock (o periodo) è la lunghezza del ciclo
  - la frequenza di clock è il reciproco del periodo
- Argomento già introdotto nelle precedenti lezioni.
- Generalmente una frequenza più alta porta a maggiori prestazioni.

Performance

CPU time = #instructions • CPI • Clock cycle

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Clock cycles}}{\text{Instructions}} * \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Per migliorare le performances si può migliorare ogni singolo fattore.
- Il miglioramento di un fattore può portare al peggioramento degli altri.

Singolo ciclo di clock

- Possibili operazioni previste dal datapath:
  - una nuova istruzione viene letta dalla memoria
  - l'unità di controllo configura i segnali
  - vengono letti i registri dal Register File
  - viene generato l'output della ALU
  - viene effettuata la scrittura o lettura nella memoria principale
  - viene calcolato l'indirizzo di branch
  - viene scritto un valore nel Register File
  - viene aggiornato il PC
  - …?

Singolo ciclo di clock

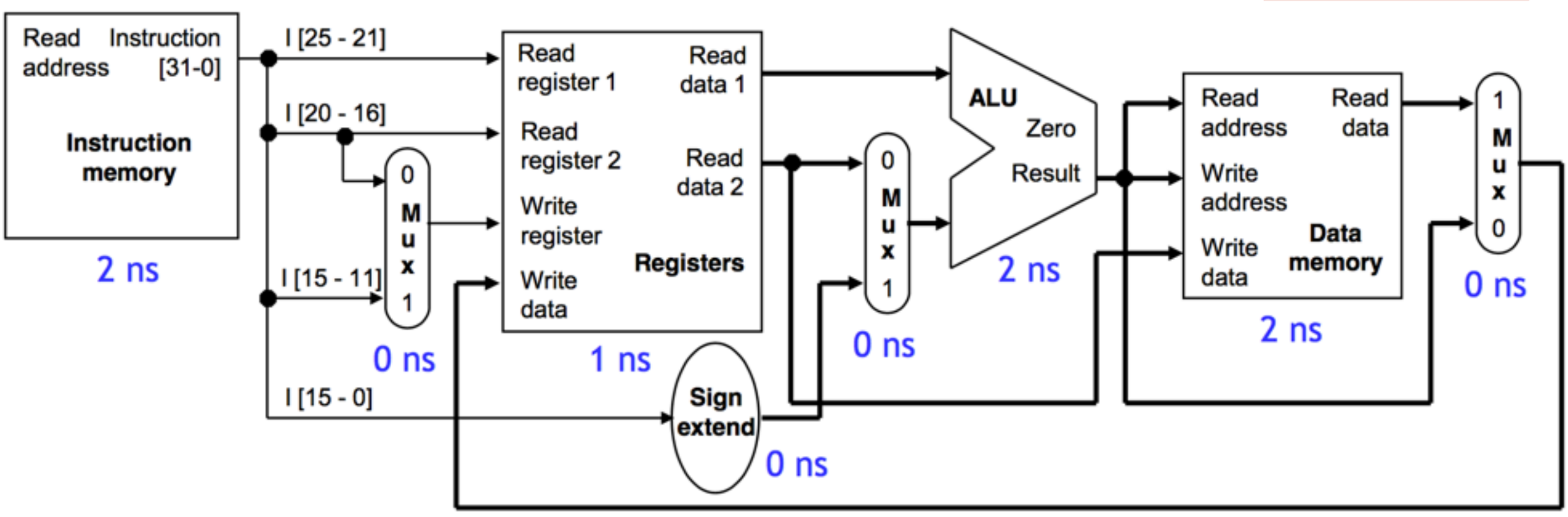
- Ogni istruzione dell'ISA prevede una combinazione di una parte delle operazioni appena descritte.
- In un ciclo di clock deve poter essere eseguita una istruzione:
  - il ciclo di clock deve essere abbastanza lungo da poter eseguire l'istruzione più lenta dell'ISA.
- Ma una volta decisa la lunghezza del ciclo di clock, ogni istruzione durerà esattamente quel tempo (anche se potrebbe essere completata in meno tempo).

Esempio: lw

- Esempio del tempo di esecuzione di una lw:

reading the instruction memory	2ns	8ns
reading the base register $sp	1ns	
computing memory address $sp-4	2ns	
reading the data memory	2ns	
storing data back to $t0	1ns	

I tempi mostrati sono puramente esemplificativi

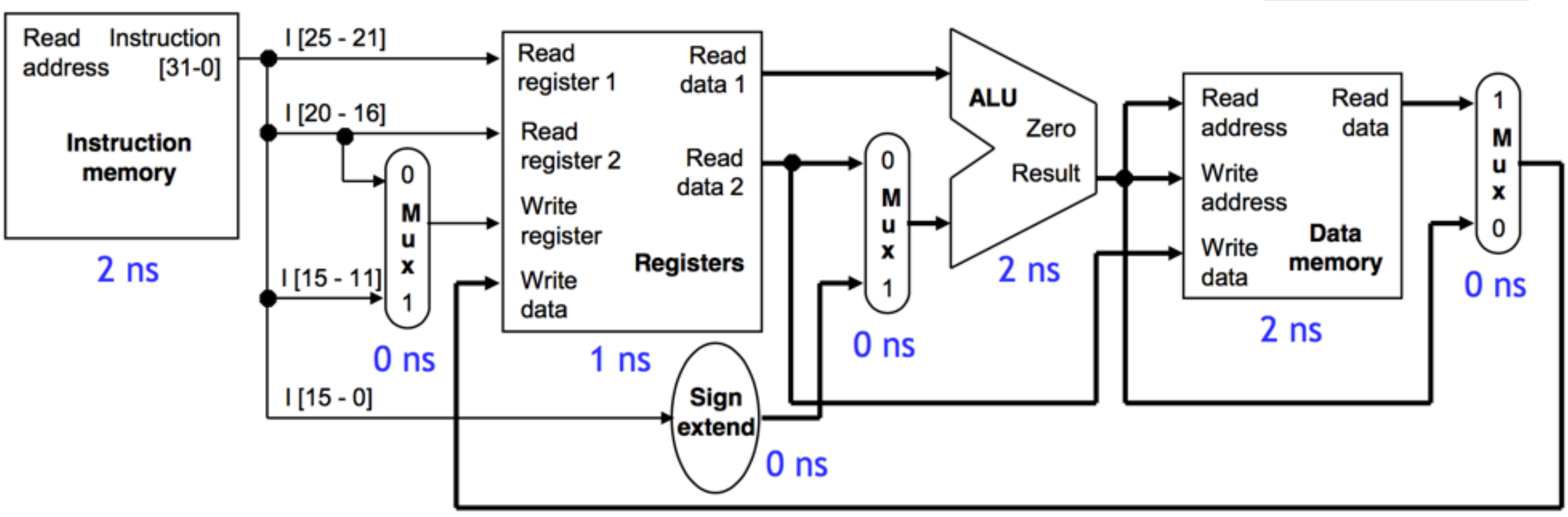


Esempio: add

- Esempio del tempo di esecuzione di una add:

reading the instruction memory	2 ns	6 ns
reading registers $t1 and $t2	1 ns	
computing $t1 + $t2	2 ns	
storing the result into $s0	1 ns	

I tempi mostrati sono puramente esemplificativi



Esempio: performance

- Consideriamo 8 ns come tempo di esecuzione dell'istruzione più lenta (lw), allora il ciclo di clock dovrà durare 8 ns.
- Anche la add impiegherà 8 ns (anche se potenzialmente potrebbe impiegarne sono 6 ns).
- Poniamo sempre come esempio:
  - istruzioni sw → 7 ns
  - istruzioni beq → 5 ns
- E poniamo che in tabella siano mostrate le frequenze di esecuzione delle diverse istruzioni.

Instruction	Frequency
Arithmetic	48%
Loads	22%
Stores	11%
Branches	19%

Esempio: performance

- Con un datapath a singolo ciclo ogni istruzione richiederebbe comunque 8 ns.
- Se potessimo eseguire le istruzioni alla massima velocità possibile, avremmo che il tempo medio di esecuzione di un'istruzione sarebbe:

  (48% • 6 ns) + (22% • 8 ns) + (11% • 7 ns) + (19% • 5 ns) = 6.36 ns

- Il datapath a singolo ciclo di clock è 8 / 6.36 = 1.26 volte più lento!
- Questa è una stima molto ottimistica: l'accesso alla memoria principale (RAM) è dell'ordine di 100 ns.
- L'istruzione più lenta prevede due accessi in memoria, potenzialmente quindi richiederebbe 200 ns (le cache mitigano il problema).

Esempio: performance

- I moderni calcolatori non usano il datapath a singolo ciclo di clock per motivi di inefficienza.
- Nelle prossime lezioni vedremo come questa problematica può essere superata:
  - introduzione del concetto di pipeline