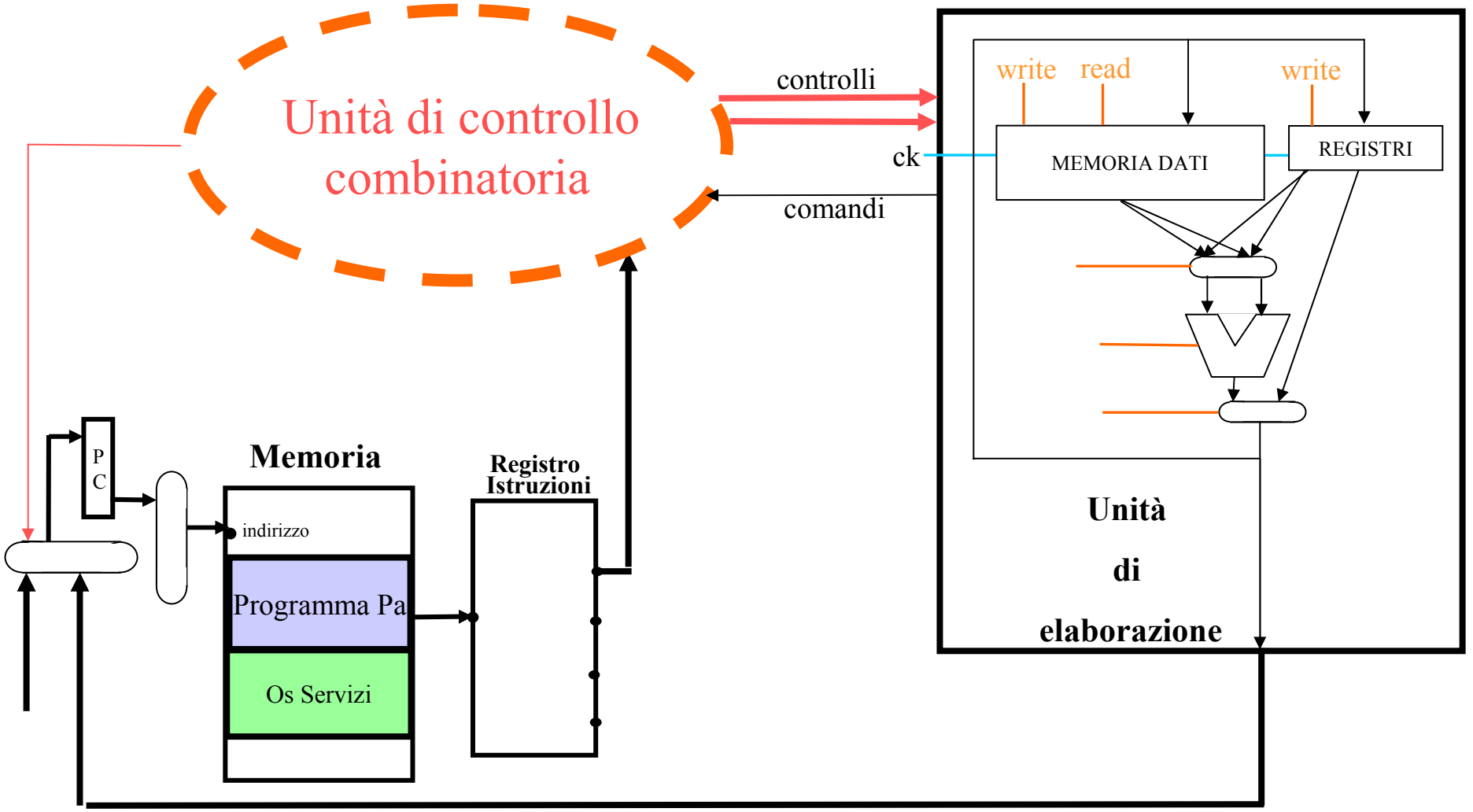
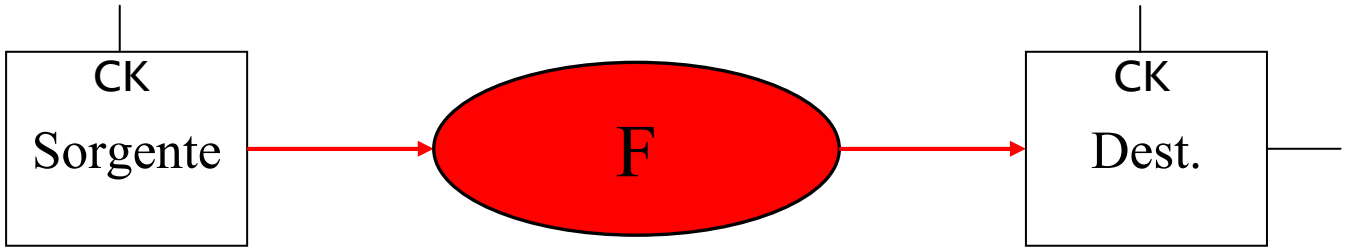
Controllo di un processore a singolo ciclo

NB: schema stilizzato (in particolare, non corrisponde al MIPS)



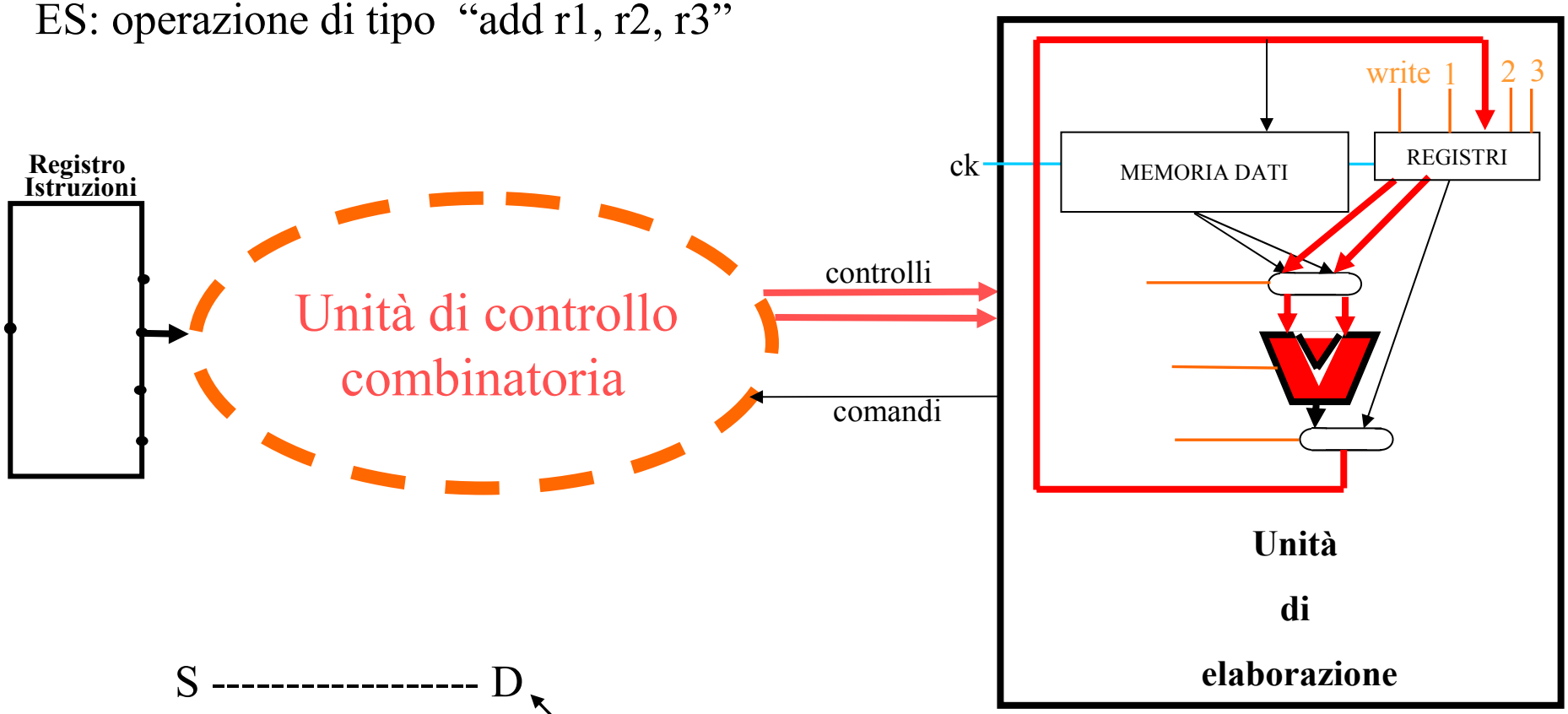
Idea di base

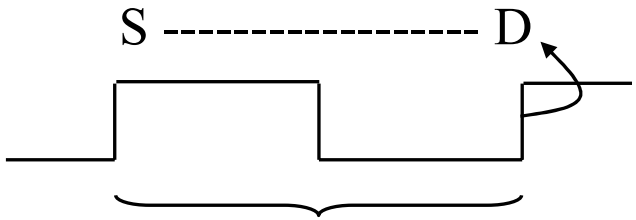
- Ad ogni ciclo di clock, il registro istruzioni contiene l'istruzione corrente
- L'unità di controllo è una rete combinatoria che:
  - riceve in input l'istruzione corrente
  - produce in output segnali di controllo all'unità di elaborazione:
    controllo multiplexer, read e write ad elementi di memoria, controllo ALU
- I segnali di controllo determinano, a seconda del tipo di istruzione:
  - il percorso sorgente-destinazione dei dati mediante:
    indirizzi e numeri registri + segnali di controllo ai multiplexer
  - le operazioni aritmetiche e logiche effettivamente svolte mediante:
    segnali di controllo alle ALU
  - se un elemento di memoria deve scrivere e/o leggere un dato mediante:
    segnali di tipo read/write
- Avremo quindi la determinazione di un "percorso" del tipo:



dove: - sorgente e destinazione possono coincidere
- valore sorgente disponibile "nel corso" del ciclo, destinazione scritta alla fine

ES: operazione di tipo “add r1, r2, r3”





- valori dei registri disponibili (poco dopo – T_{prop} – l’inizio del periodo)
  + si “crea” percorso sorgente-destinazione (con i comandi relativi)
- valori all’ingresso della destinazione (registri) stabili entro la fine del ciclo
- questi valori sono scritti nel fronte successivo – disponibili prossimo ciclo

Conseguenze:

• E' possibile condividere elementi tra istruzioni diverse
(eseguite in cicli diversi del clock!), però…

• Ogni elemento che venga usato più di una volta nell'esecuzione di una istruzione deve essere duplicato:

- Memoria dati (lw, sw)

≠

Memoria programmi (fase di fetch di tutte le istruzioni)

- ALU (lw, sw, beq, aritmetiche)

≠

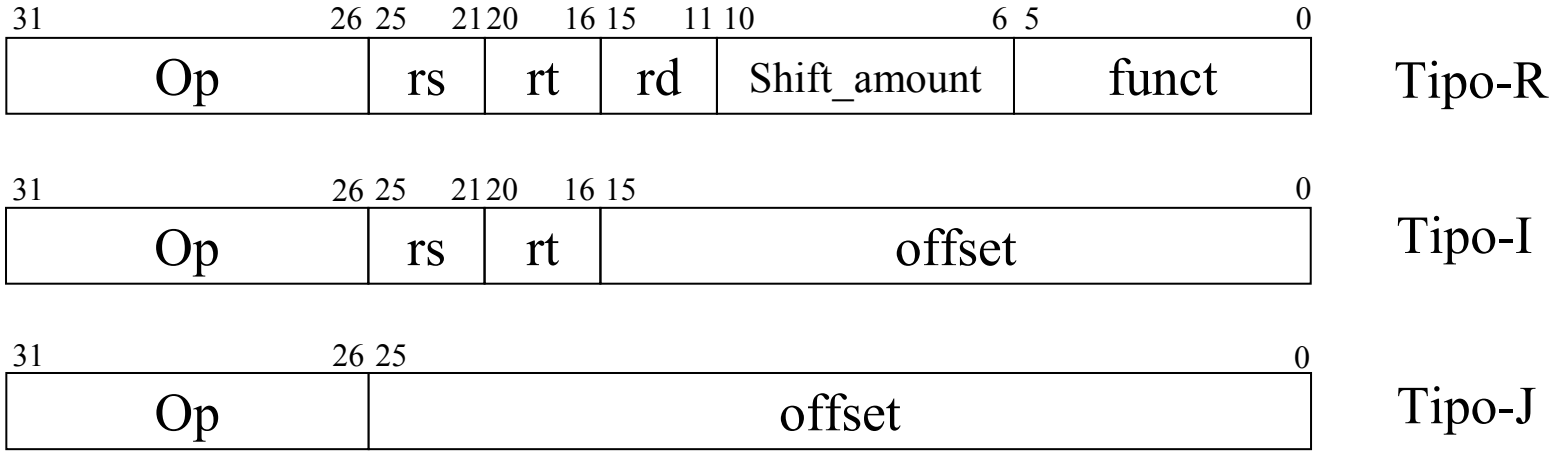
Sommatore PC + 4 (tutte le istruzioni)

≠

Sommatore PC+offset (beq)

Consideriamo il caso del MIPS, di cui si vogliono implementare le istruzioni:

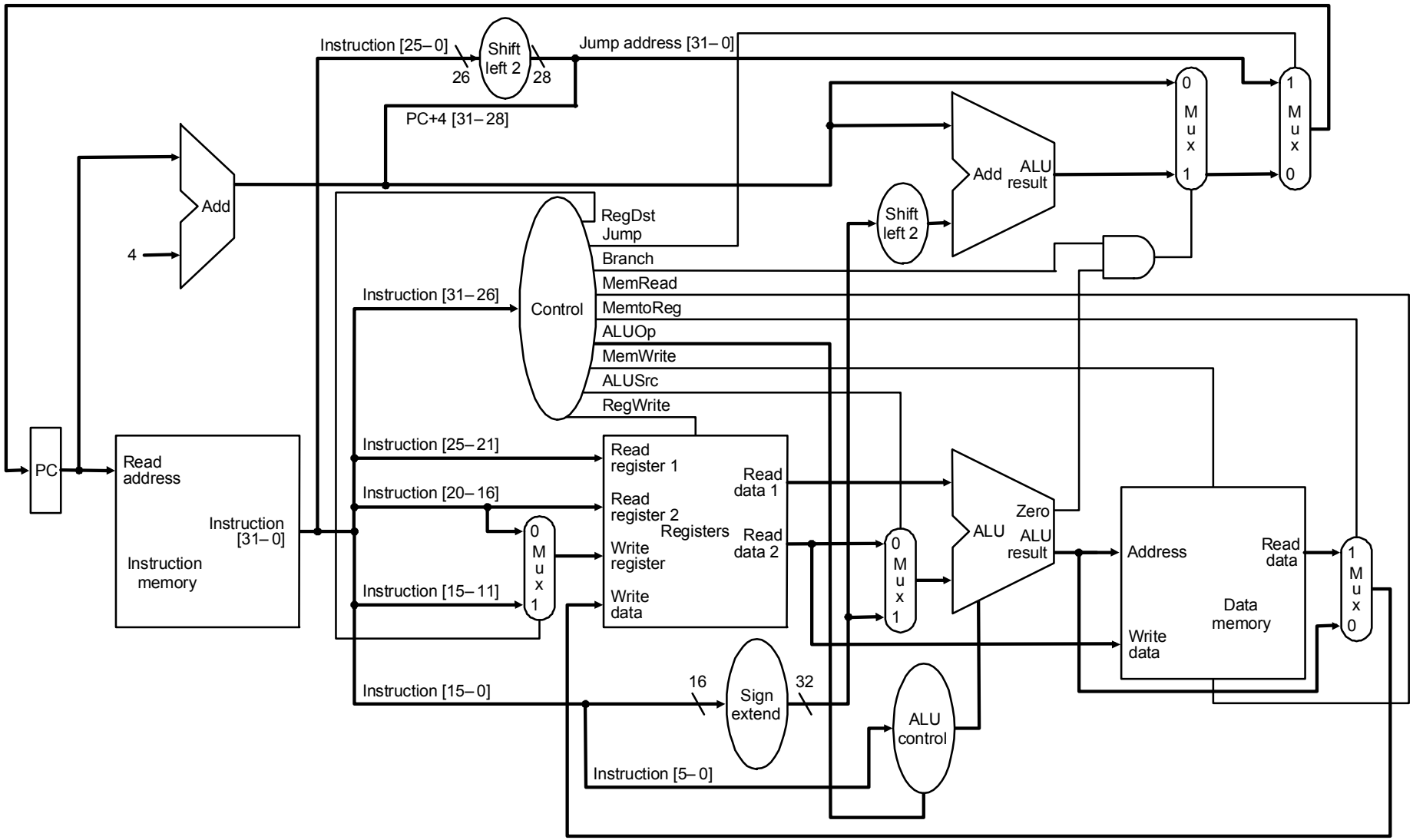
- Aritmetiche e logiche } Tipo-R
- lw, sw } Tipo-I
- beq
- j } Tipo-J



La progettazione è facilitata dal fatto che:

- Campo Op sempre in [31-26]
- i registri da leggere sono sempre in rs e rt
- il registro base [da sommare] per lw e sw è rs [primo ingresso della ALU]
- l'offset a 16 bit da sommare per beq, lw, sw è in [15-0]

Il registro su cui scrivere può essere rd (per operaz. di TIPO-R) o rt (per lw)

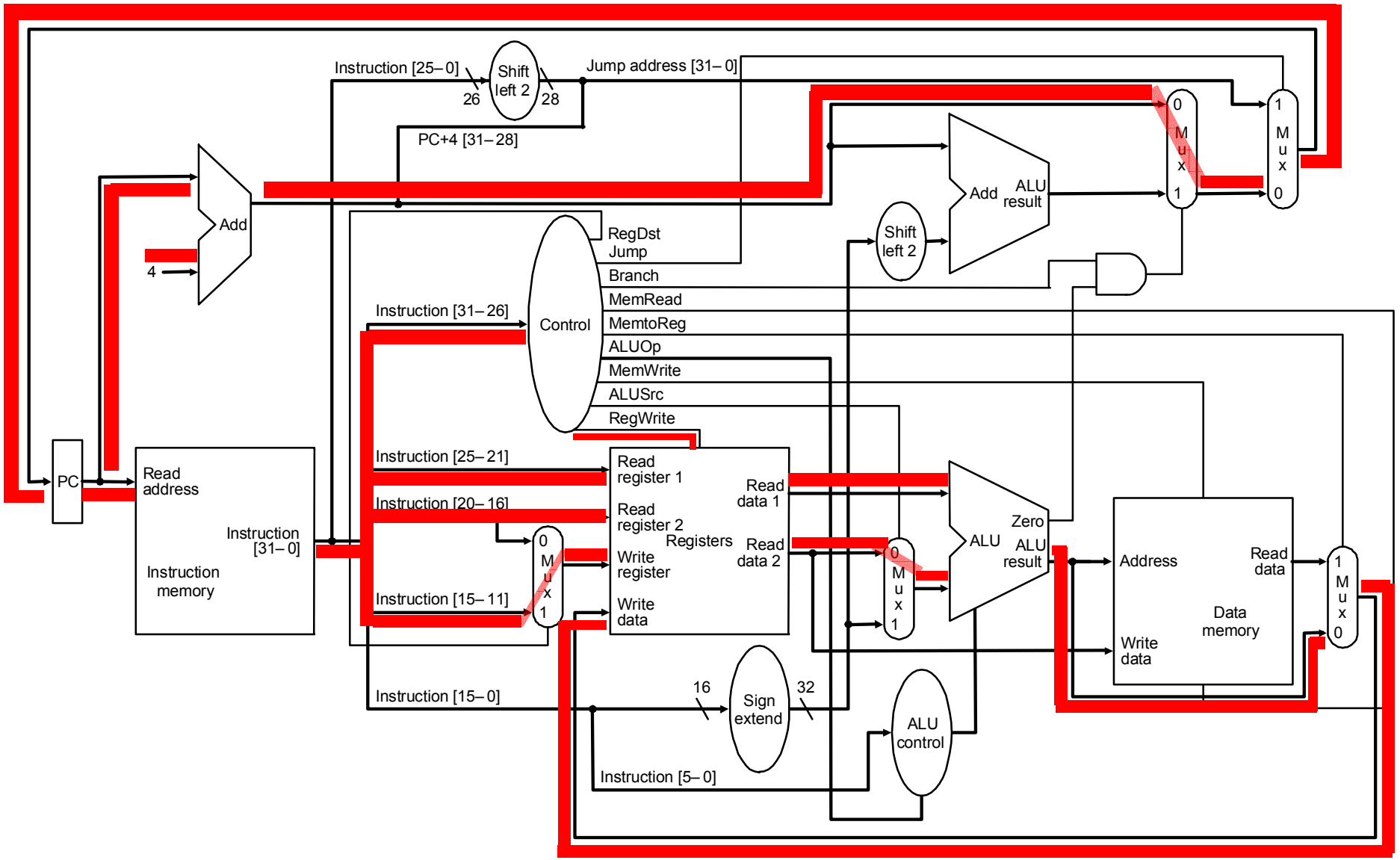

Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
Add
4
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Shift left 2
Add
ALU result
0
M
u
x
1
1
M
u
x
0
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU
Zero
ALU result
ALU control
Address
Read data
Data memory
Write data

Si vede che:

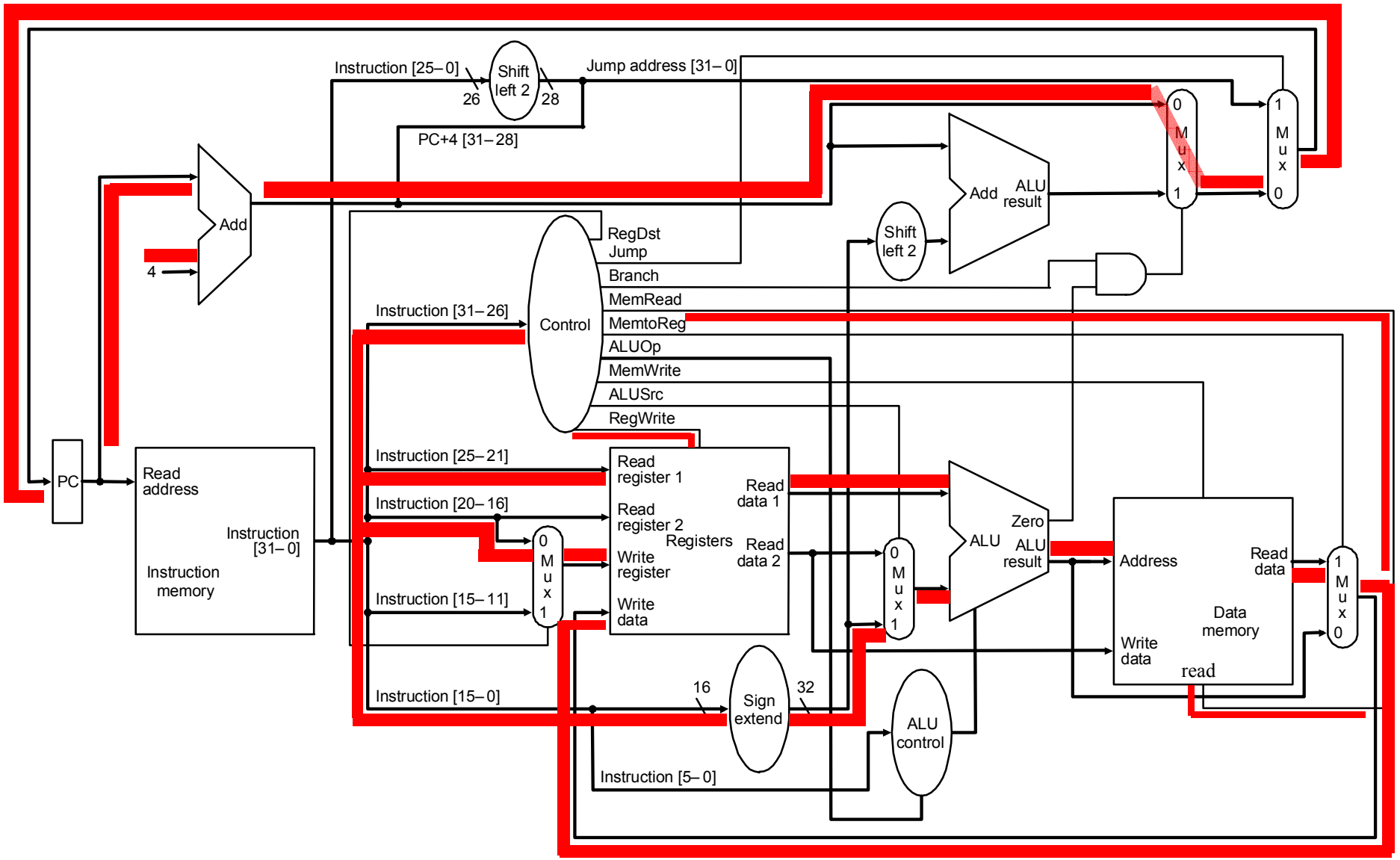
- i segnali di controllo sono determinati in modo “combinatorio” soltanto sulla base del campo Opcode
- non è in generale possibile prevedere l’ordine di arrivo dei segnali di controllo: le operazioni non sono eseguite “in sequenza”, controllo combinatorio (è necessario che T_{clock} sia sufficientemente lungo)

Vediamo allora alcuni esempi di esecuzione delle istruzioni…

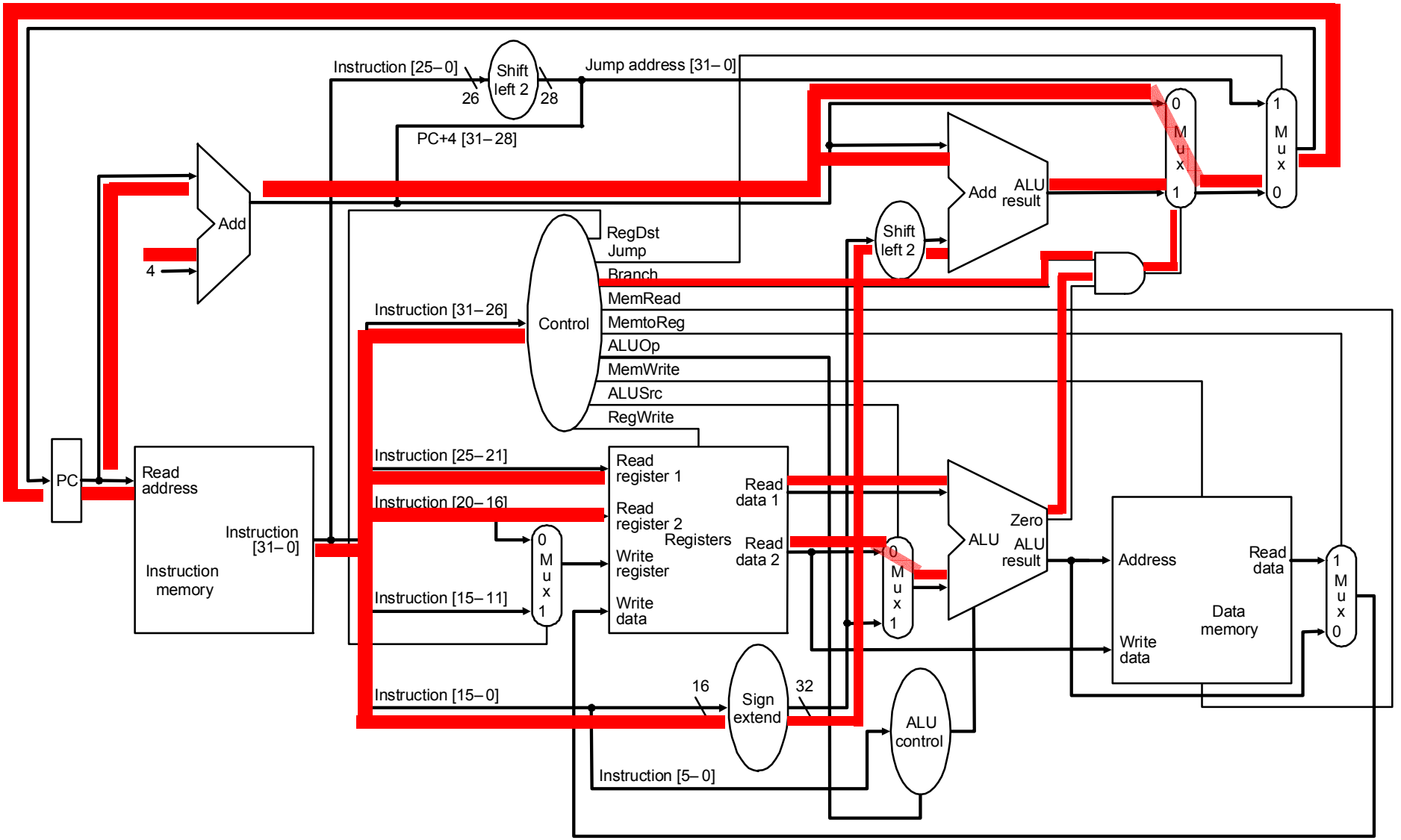
Esempio: istruzione di tipo-R



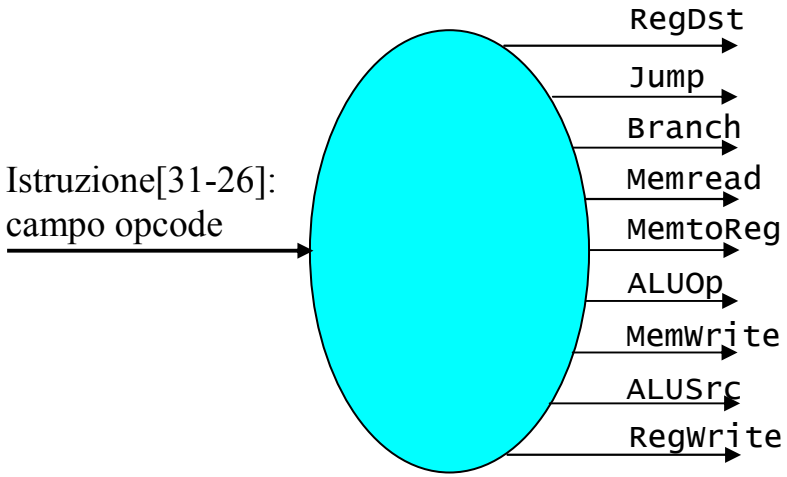
Esempio: istruzione di load



Esempio: istruzione beq



Progetto e realizzazione dell'unità di controllo principale



Progetto e realizzazione corrispondono a quelli di una rete combinatoria:

ISTRUZ	RegDst	Jump	Branch	Mem Read	Memto Reg	ALUOp	Mem Write	ALUSrc	Reg Write
Tipo-R	1	0	0	0	0	10	0	0	1
lw	0	0	0	1	1	00	0	1	1
sw	X	0	0	0	X	00	1	1	0
beq	X	0	1	0	X	01	0	0	0
j	X	1	X	0	X	XX	0	X	0

Opcode [6 bit]

Richiamo su prestazioni

Si ricorda che le prestazioni sono valutate sul tempo di esecuzione di un programma!
Per valutare soluzioni progettuali diverse, bisogna far riferimento
allo stesso programma (o classe di programmi!)

$$T_{esecuzione} = numero\_istruzioni * \underbrace{CPI * T_{clock}}$$

Tempo medio di esecuzione per istruzione

Fisse a parità di
Instruction Set Architecture

T_{clock} non varia al variare delle istruzioni

[progettazione di sistemi "a clock variabile" non adottata in pratica]

E' necessario conoscere la distribuzione delle singole istruzioni per valutare CPI

[il numero di cicli di clock impiegati varia da istruzione a istruzione]

Esempi di processori che usano controllo a singolo ciclo:

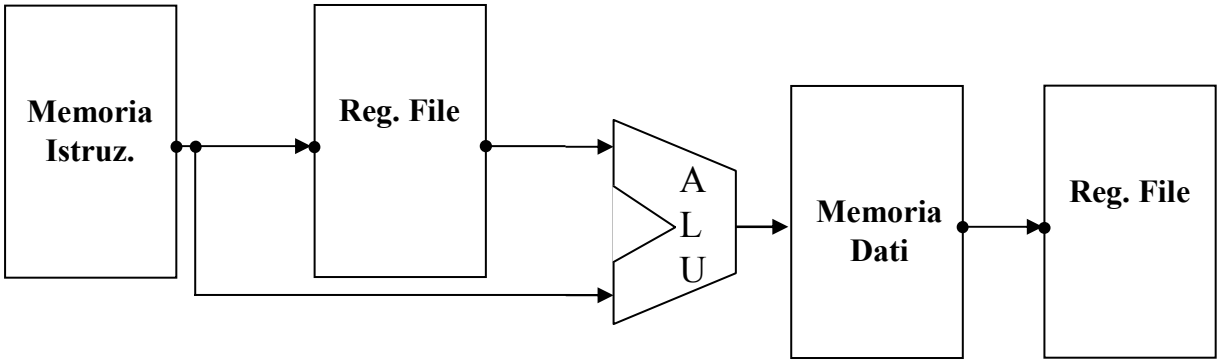
NESSUNO!

Perché?

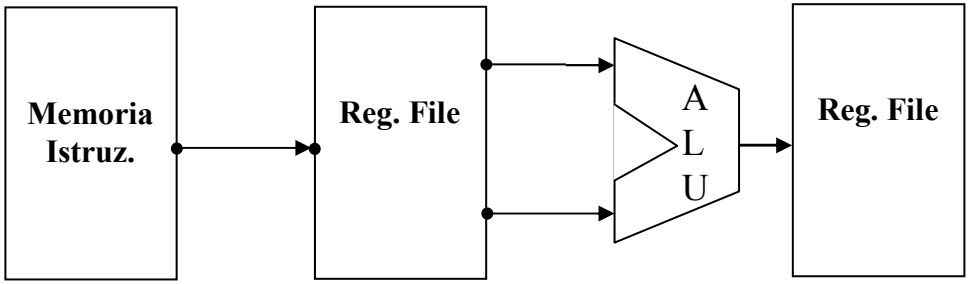
- Periodo di clock: abbastanza lungo per garantire la stabilità dei segnali attraverso il percorso più lungo ⇒ tempo di esecuz. costante per diverse istruz.

  ➡ Istruzioni "più lente" limitano le istruzioni "più veloci"!

Es. lw



Es. Tipo-R



Limitazione aggravata da

- istruzioni che utilizzano decine di unità funzionali in serie, ad esempio comprendenti calcoli in virgola mobile!

⇒ CPI = 1, ma T_{clock} alto, ovvero frequenza di clock molto bassa!

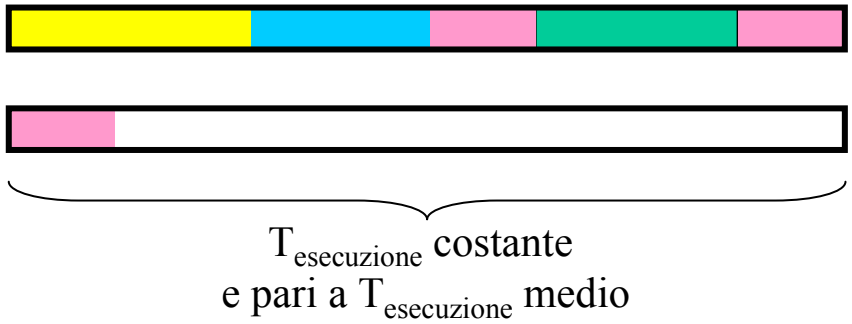
Nel complesso, $T_{esecuzione}$ è sempre pari al caso peggiore, ovvero a quello dell'istruzione più complessa e lunga

• Altro svantaggio: duplicazione dell'HW ⇒ costi elevati!

Nell'esempio del MIPS, come visto:

- occorrono due memorie diverse [dati e istruzioni]
  [NB: questo va comunque bene perché negli attuali calcolatori si usa memoria cache + pipeline]
- occorrono tre ALU, una per istruzioni aritmetiche e confronto operandi in beq, una per calcolare PC+4 ed una per calcolare indirizzo di salto beq
  [la cosa si complica considerando modalità di indirizzamento complesse, ad esempio quelle con autoincremento]

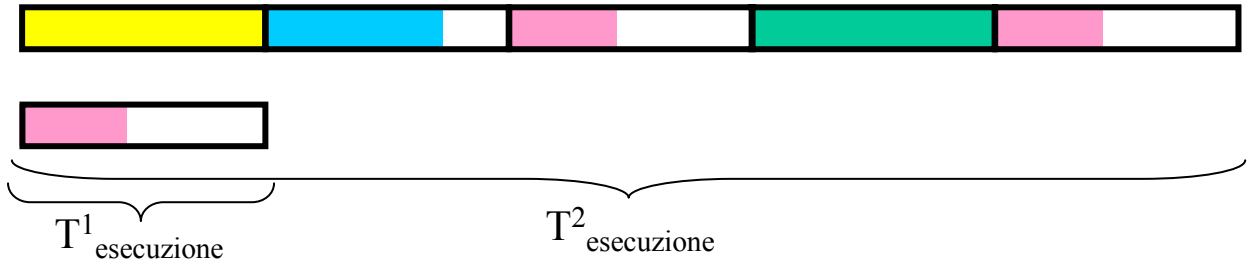
Una strategia alternativa



Istruzione lunga

Istruzione potenzialmente breve

Suddividere le istruzioni in “fasi”: un ciclo di clock per una fase!



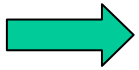
NB: $T^2_{esecuzione}$ (il caso peggiore) può in generale essere maggiore del precedente!

Tuttavia, le istruzioni più lunghe sono anche meno frequenti:

Principio “rendere più veloce l’evento più frequente” comporta un guadagno!

Controllo di un processore-multiciclo

- Ogni istruzione divisa in fasi
- Ad ogni ciclo di clock una fase: controllo sequenziale



- Necessità di memorizzare risultati intermedi in registri temporanei (non visibili al programmatore):
  - registri temporanei: salvano dati prodotti in un ciclo di clock e utilizzati dalla stessa istruzione in un ciclo di clock successivo
  - registri visibili al prog: dati utilizzati da istruzioni successive
- E' possibile utilizzare una stessa unità funzionale più volte nel corso di una istruzione:
  - unica memoria per istruzioni e dati
  - una sola ALU, che svolge le funzioni logico aritmetiche delle istruzioni di Tipo-R, somma PC+4, somma per calcolare indirizzi nei salti condizionati (beq)

Come visto, ciascuna fase deve essere sufficientemente breve (T_{clock} breve)

➡ Bilanciare la suddivisione in fasi,
evitare di mettere in serie più unità funzionali lente.

➡ Faremo in modo di non mettere in serie più di una delle operazioni:
- accesso in memoria (dati o indirizzi)
- accesso al register file (due letture e una scrittura)
- operazioni della ALU

(NB: accesso/scrittura in un registro singolo considerato non oneroso)

➡ Ciascuna di queste unità [memoria, register file, ALU] necessita di registri temporanei per memorizzarne il risultato.

Sono inoltre necessari multiplexer aggiuntivi per instradare
i dati verso le unità funzionali riutilizzate nella stessa istruzione
[p.es. MUX per selezionare indirizzo memoria tra PC per fetch
e uscita ALU per lw o sw]

Esistono diverse modalità di specifica e realizzazione del controllo…

Specifica progressiva del sistema di calcolo di un sistema multiciclo

Di seguito sono schematizzate le attività che una macchina multiciclo svolge per eseguire le istruzioni del set del MIPS ridotto: le fasi rispettano i vincoli posti in precedenza sulle operazioni che si possono eseguire in serie.

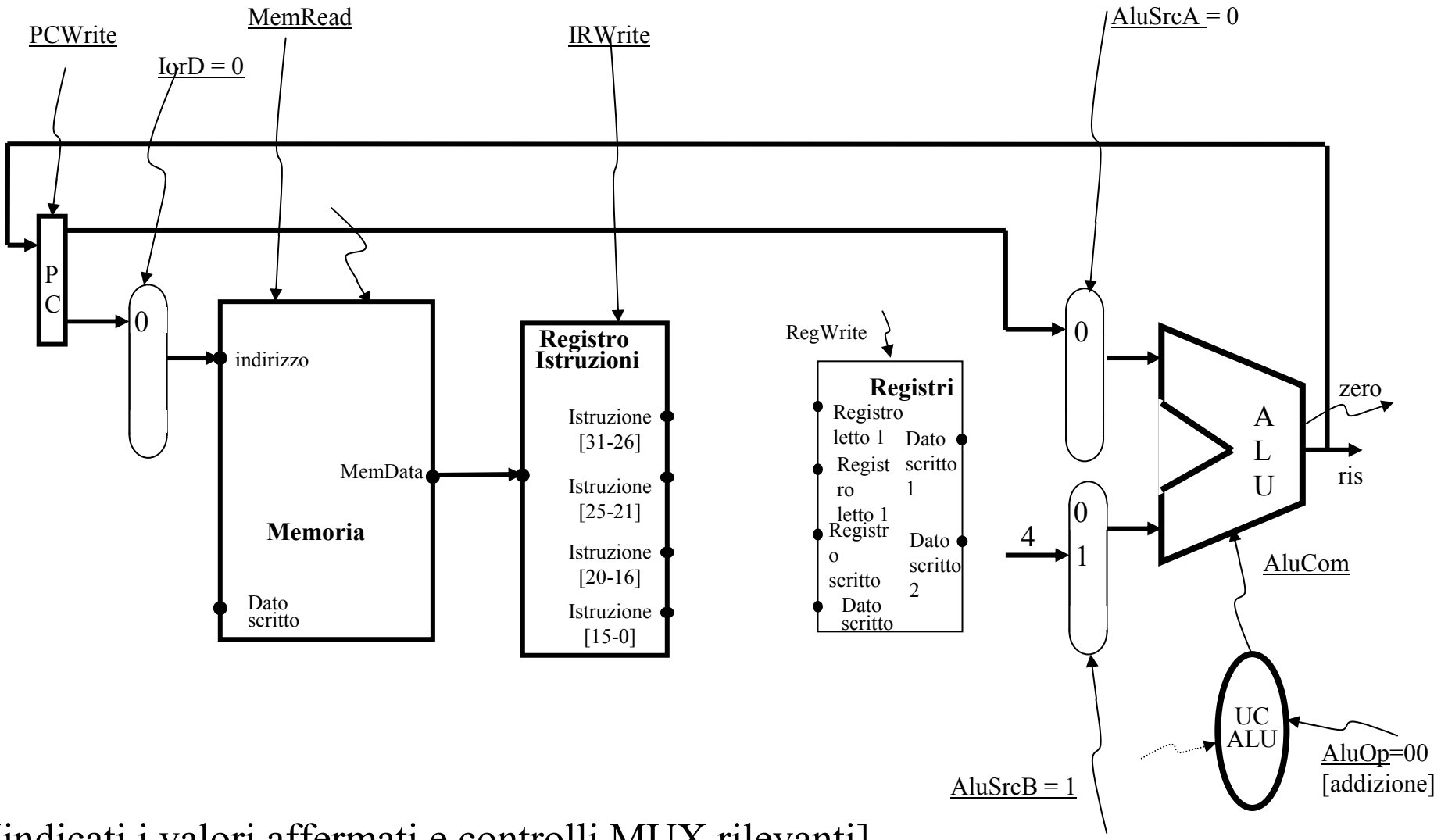
Nello specificare le diverse attività di calcolo si sono seguite le convenzioni adottate nel testo. In particolare, sono rese visibili le risorse impegnate nel ciclo in esame.

La combinazione delle immagini risulta nello schema 5-32 del testo.

I ciclo:Prelievo dell'istruzione

I ciclo: IR ← memoria [PC]

ottimistiche: PC ← PC + 4



[indicati i valori affermati e controlli MUX rilevanti]

NB:

- Durante il primo ciclo di clock vengono creati i due percorsi paralleli:
  - PC_out → Memoria → IR
  - PC_out → ALU → PC_in

  + vengono attivati i segnali di scrittura in IR e PC

- Alla fine del periodo di clock (fronte del clock):
  - PC e IR vengono scritti;

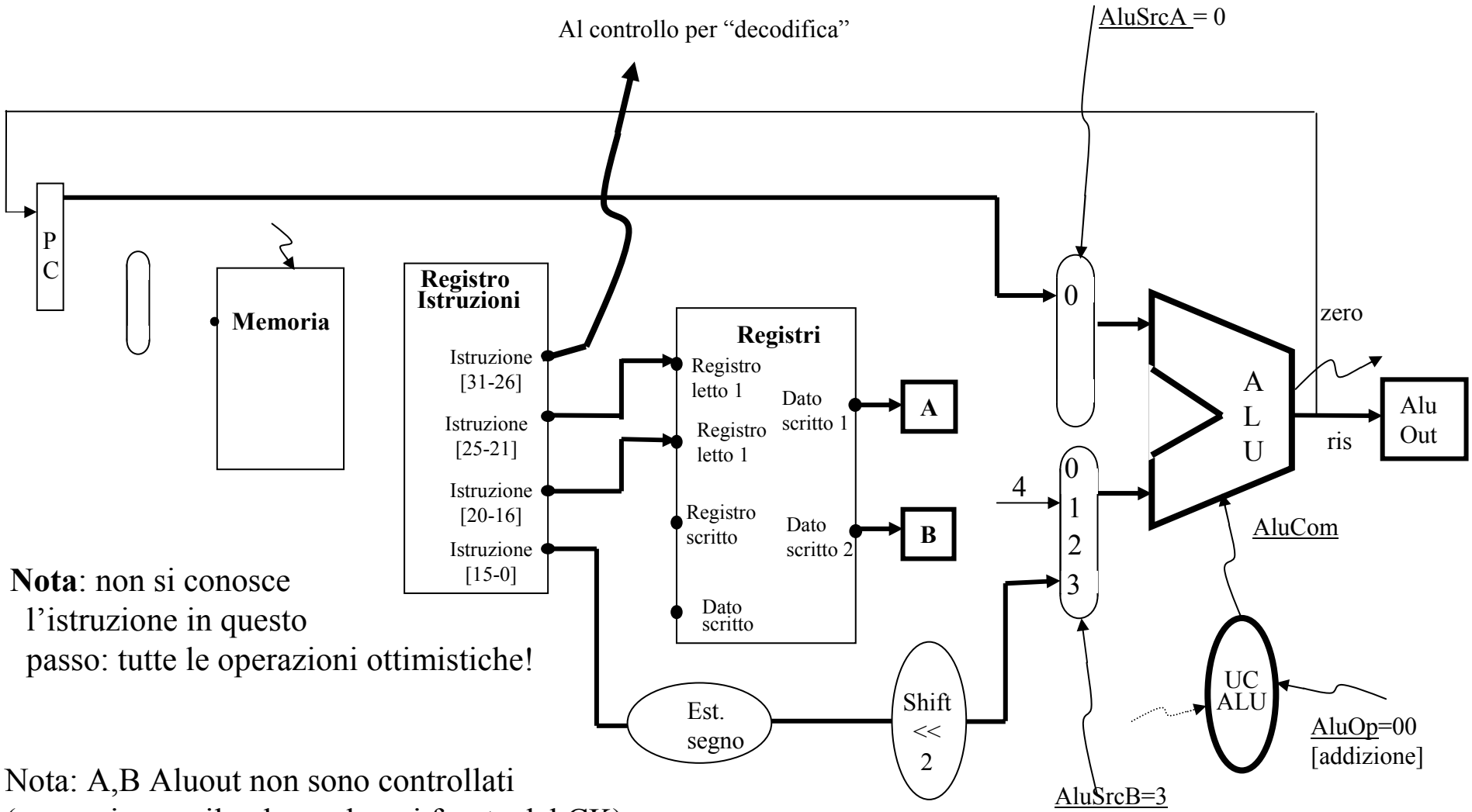
  i valori sono disponibili nel ciclo di clock successivo

Al secondo ciclo di clock IR ha il valore dell'istruzione da eseguire,
PC sarà già incrementato…

NB: si noti che ciascun percorso contiene soltanto un elemento "critico"
(memoria e ALU);
sorgente e destinazione infatti sono registri singoli (PC e ALU)

II ciclo:Decodifica e caricamento registri

ottimistiche: alimentazione registri: A← registro[IR[25-21]]; B ← registro[IR[20-16]]
calcolo indirizzo branch: AluOut ← PC + (sign_ext(IR[15-0]) << 2)

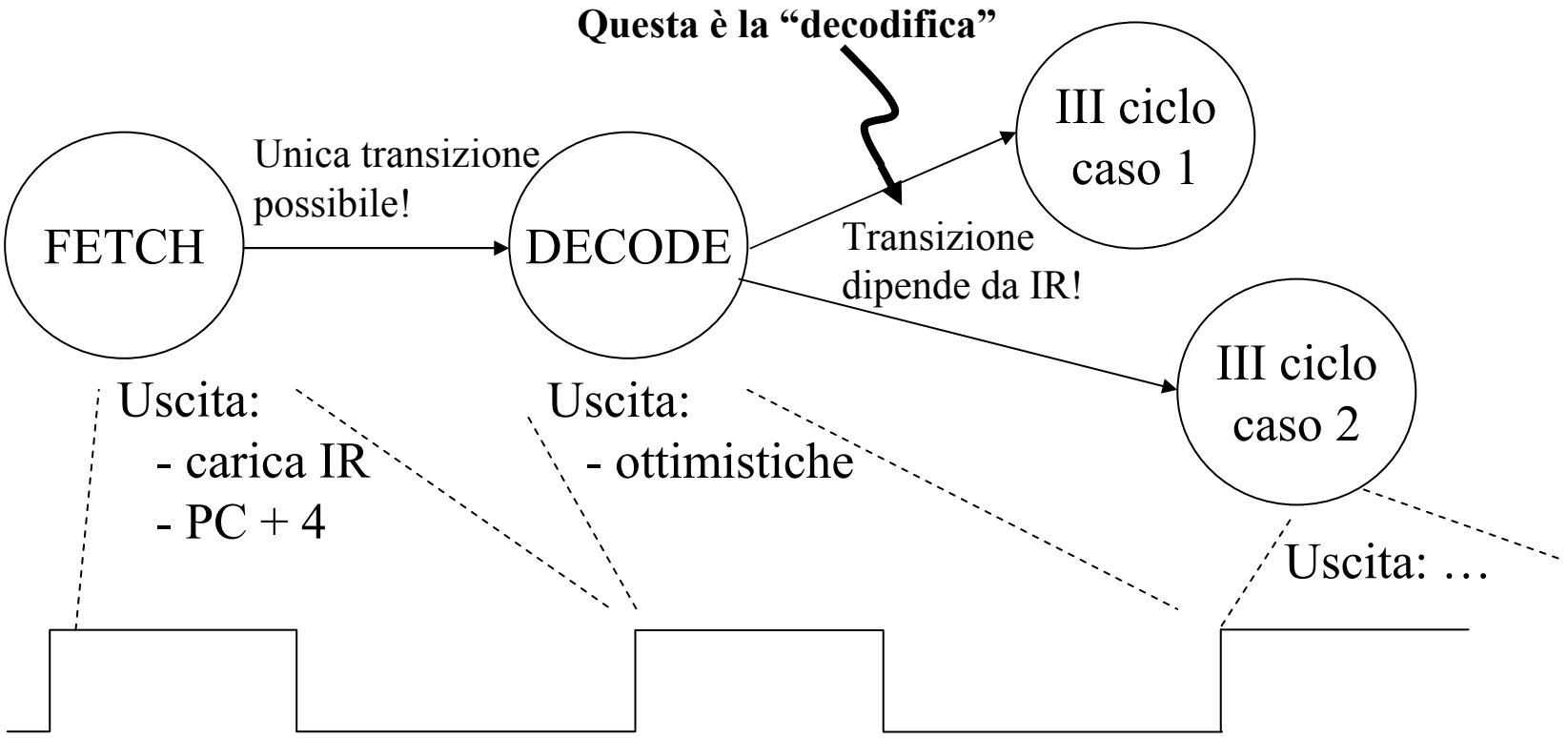


Nota: non si conosce l'istruzione in questo passo: tutte le operazioni ottimistiche!

Nota: A,B Aluout non sono controllati (memorizzano il valore ad ogni fronte del CK)

NB: si è detto che nel secondo ciclo è già disponibile l'istruzione in IR.
D'altra parte si dice anche che "non si conosce quale sia l'istruzione"
e quindi si effettua la decodifica + le uniche operazioni svolte sono "ottimistiche"

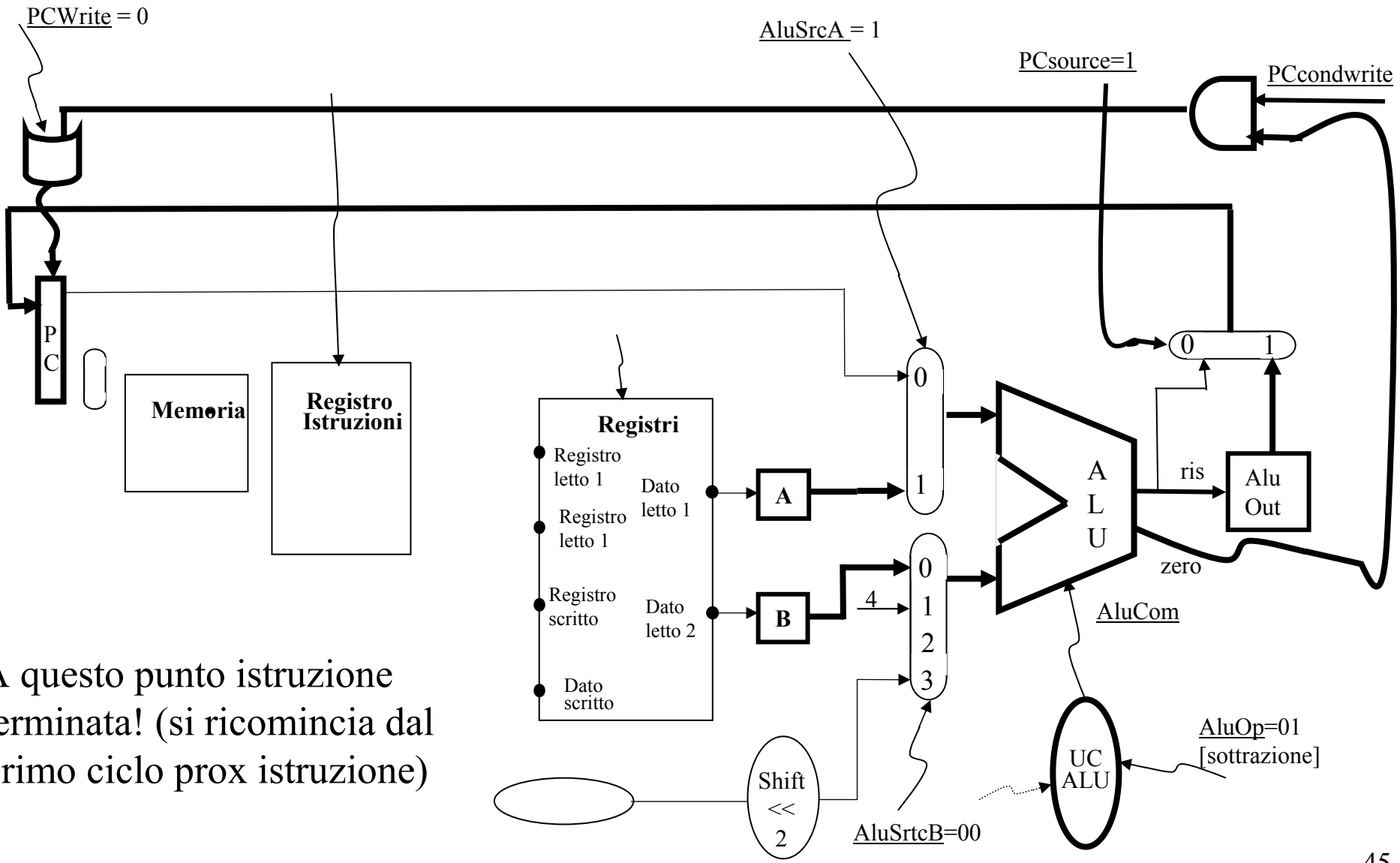
Ciò perché si usa il modello di Moore: uscite del sistema di controllo dipendono solo dallo stato corrente, non dagli ingressi [IR].



III ciclo:se è branch (beq)

IF ([A]= = [B]): PC← [AluOut]

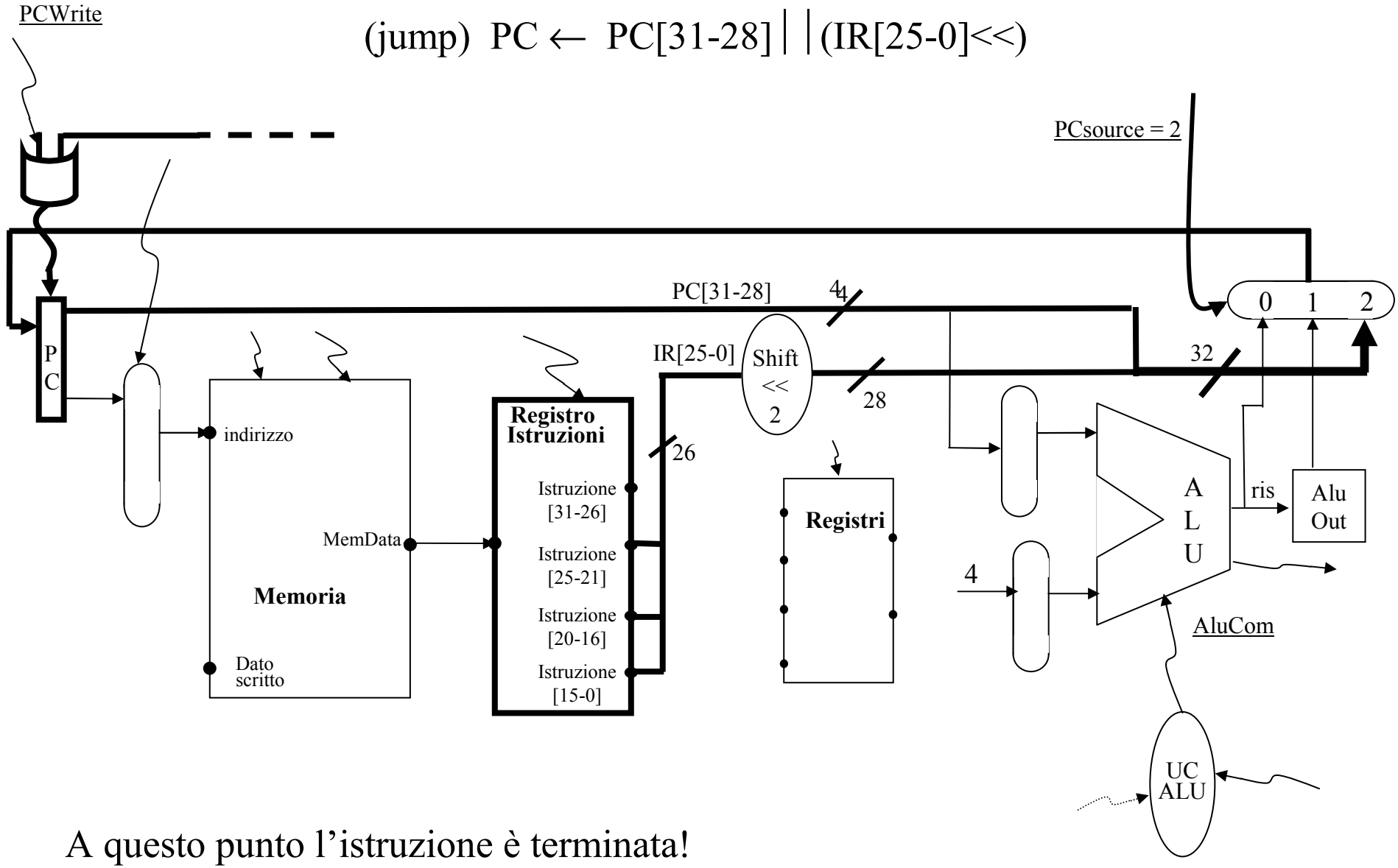
Attenzione: si ritocca il data path e si introduce PCsource



A questo punto istruzione terminata! (si ricomincia dal primo ciclo prox istruzione)

III ciclo: se è salto incondizionato

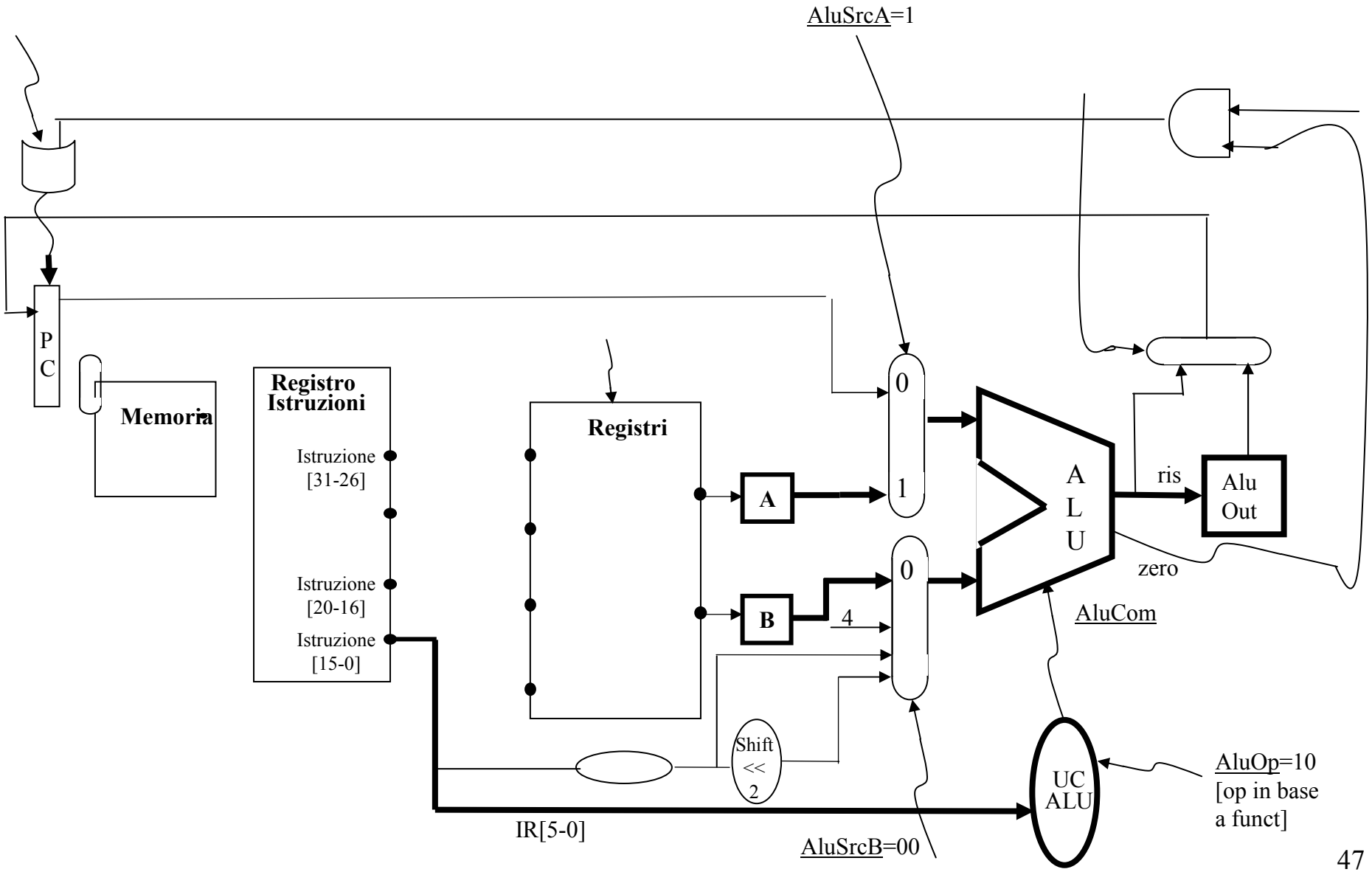
(jump) PC ← PC[31-28]||(IR[25-0]<<)



A questo punto l'istruzione è terminata!
(si ricomincia dal primo ciclo prossima istruzione)

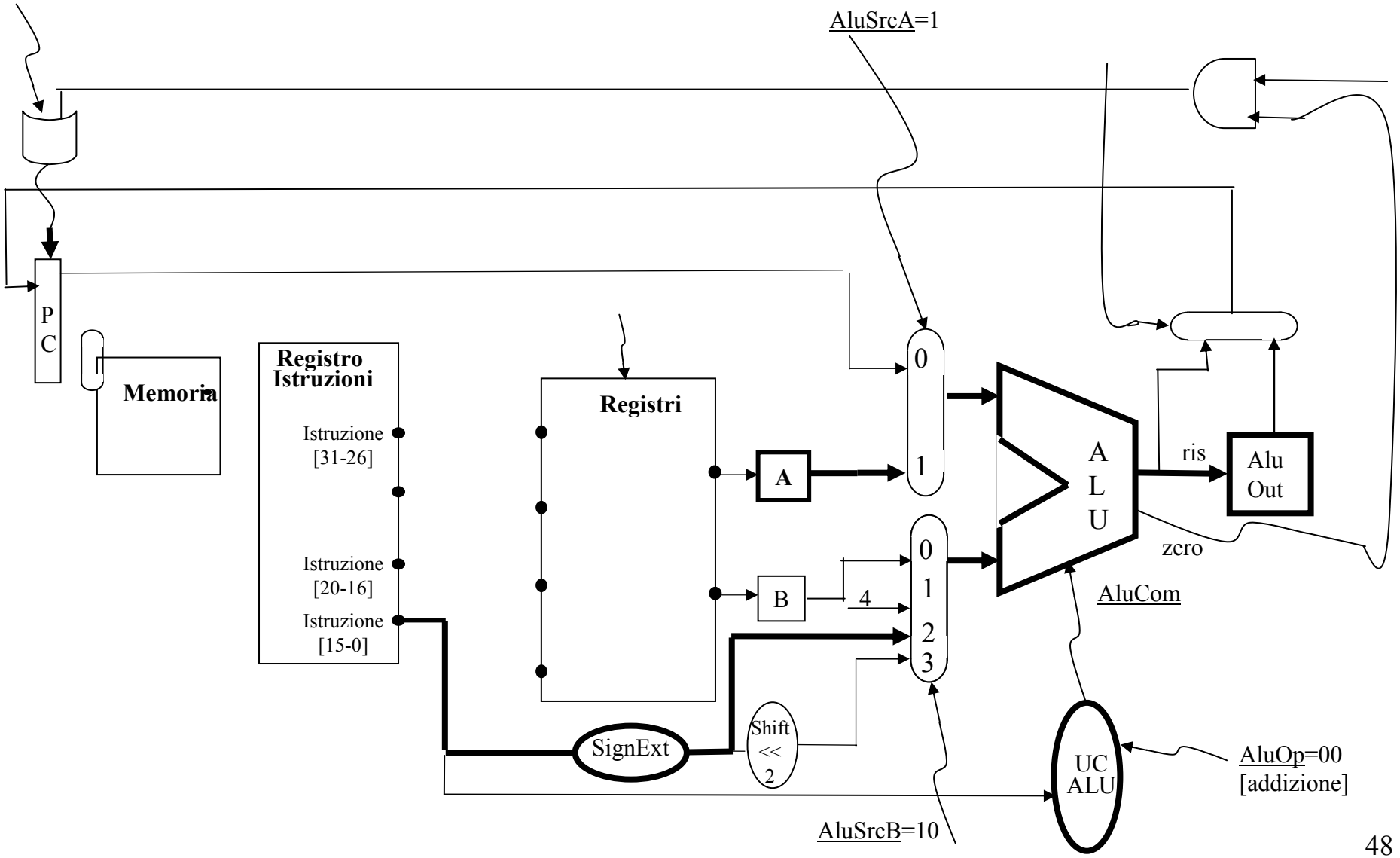
III ciclo:per tipo R

AluOut← A op B



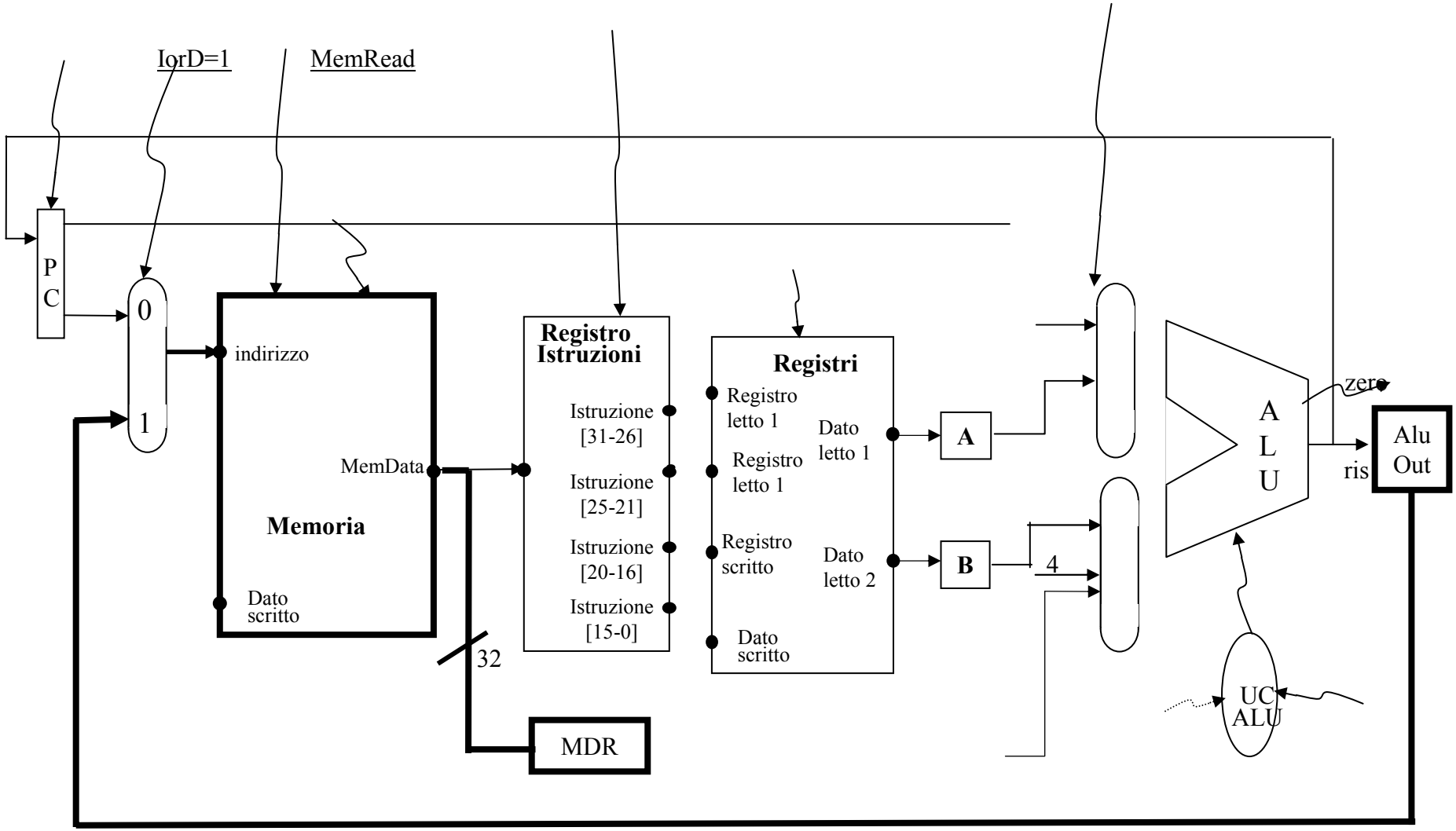
III ciclo:per tipo accesso a memoria

AluOut← [A]+ (signExt([IR[15-0]])



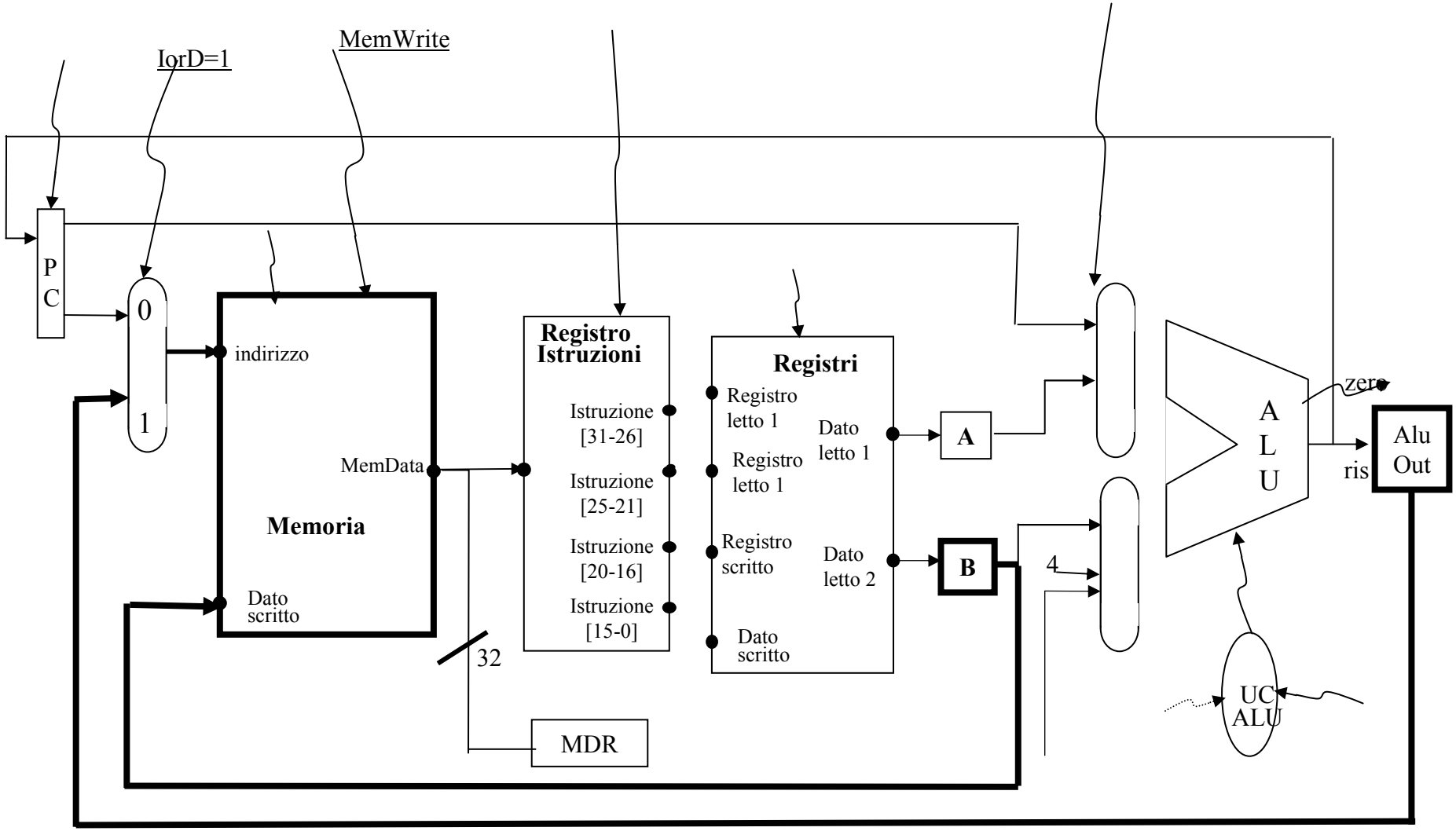
IV ciclo: carica parola (lw)

MDR ← memoria [AluOut]



IV ciclo: memorizza parola (sw)

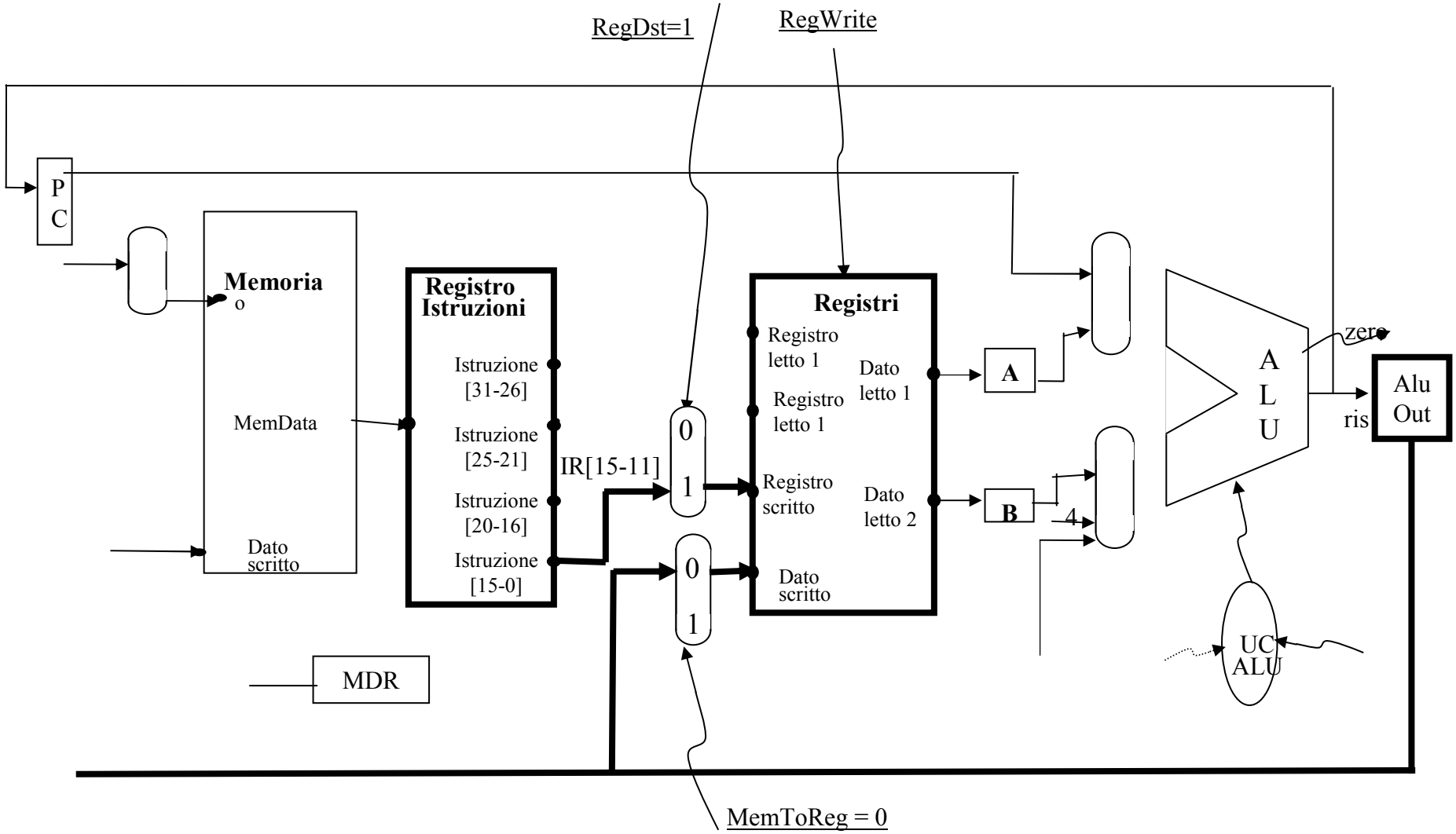
memoria [AluOut] ← B (NB: B è stato letto due volte…)



A questo punto l'istruzione è terminata!

IV ciclo: completamento istruzioni R

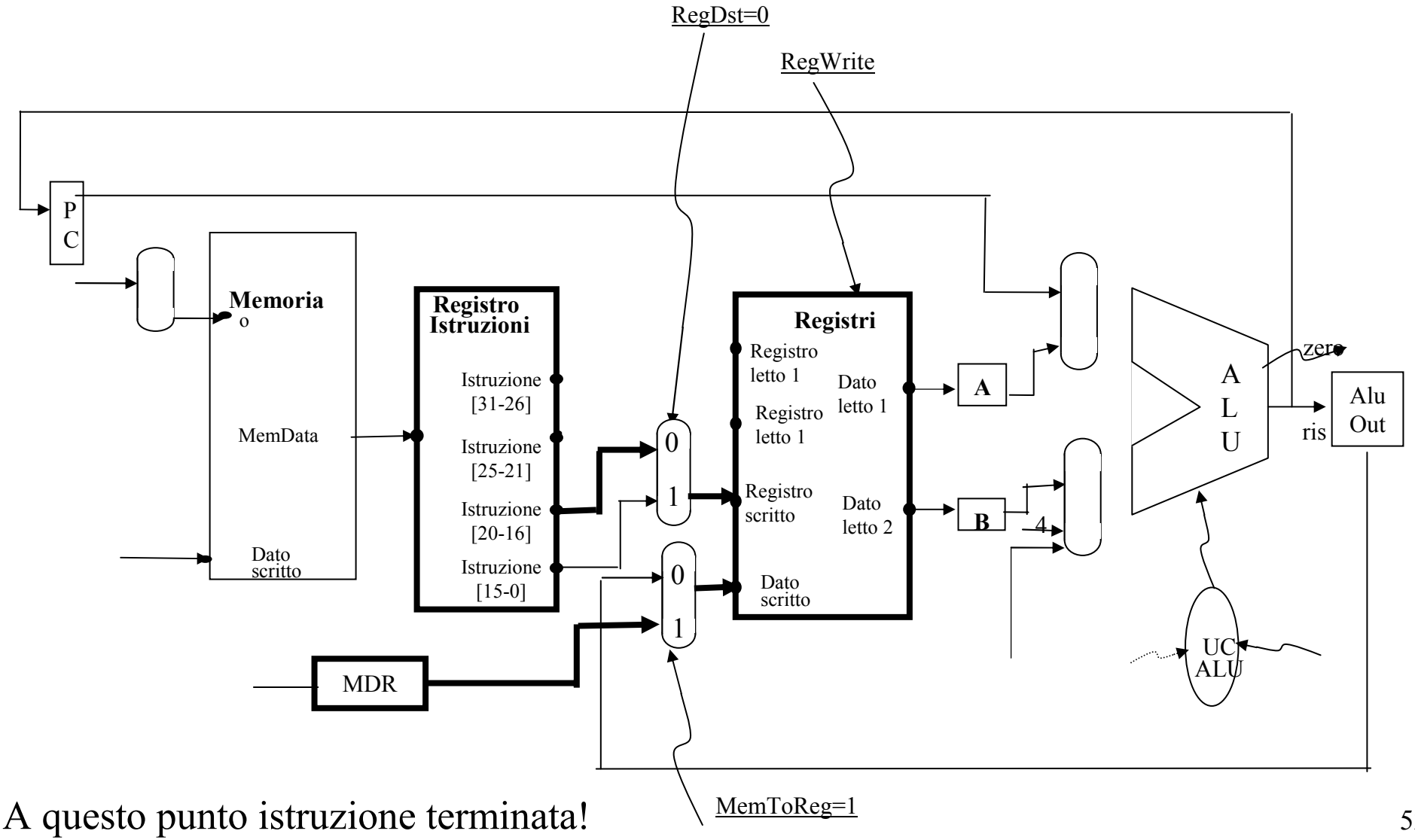
Reg [IR[15-11]] ← Aluout



A questo punto l'istruzione è terminata!

V ciclo: completamento lettura da memoria (lw)

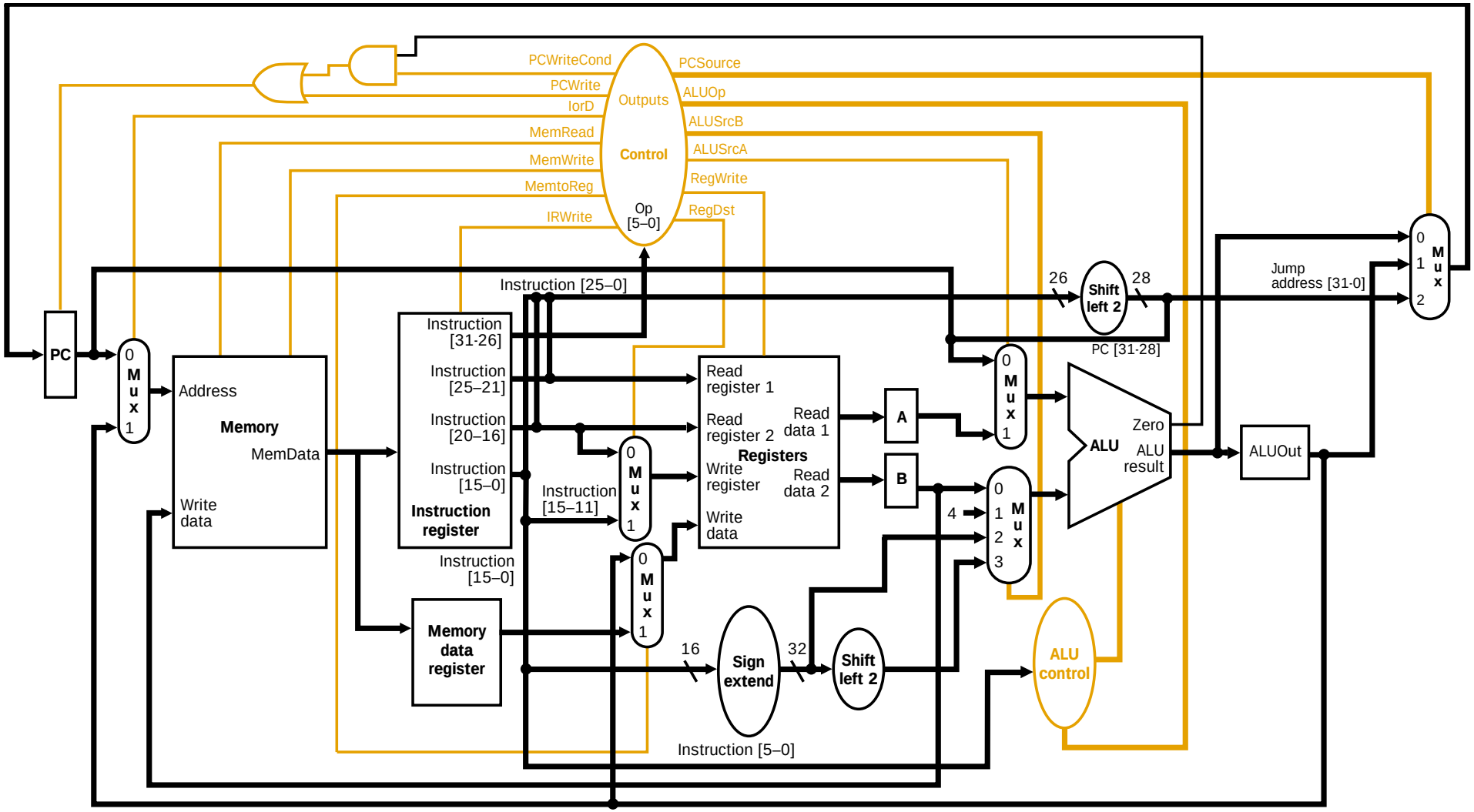
Reg [IR[20-16]] ← MDR



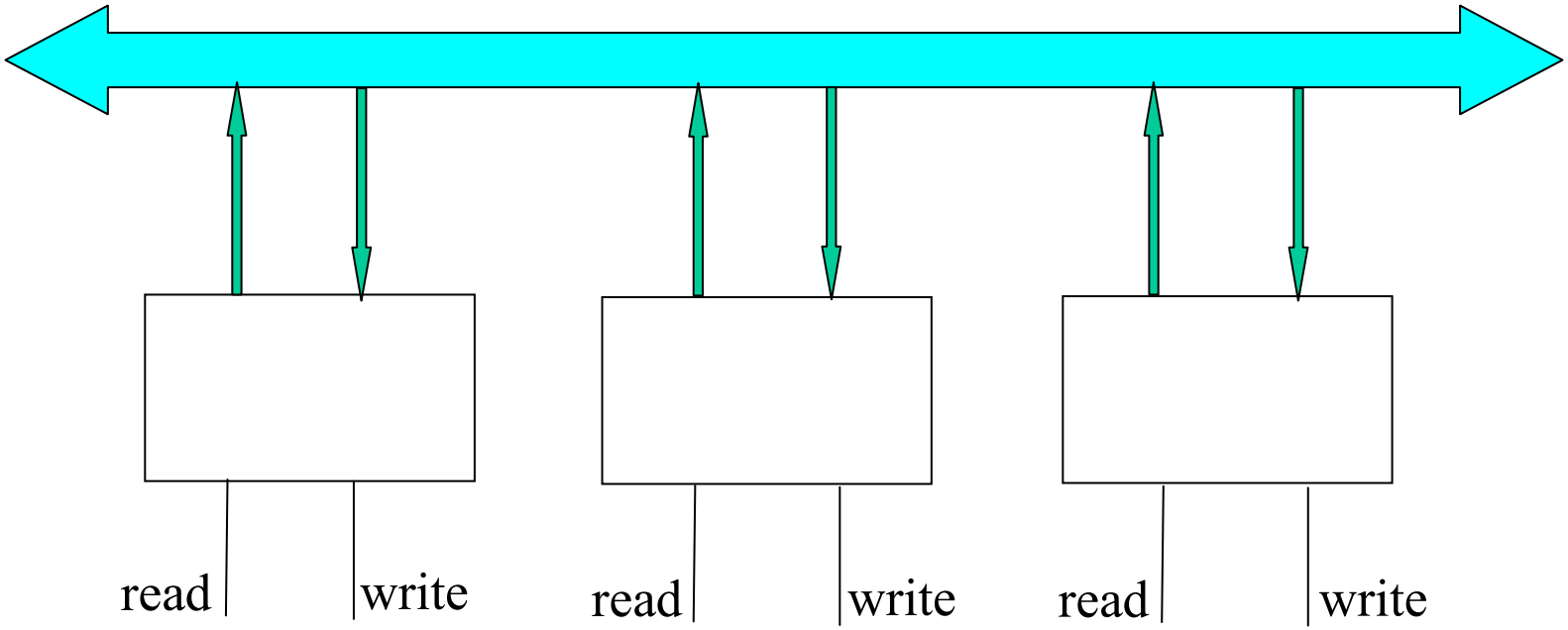
A questo punto istruzione terminata!

Da queste considerazioni risultano immediatamente:

- unità di elaborazione (datapath)
- specifica dell'unità di controllo



Soluzione alternativa (permette di risparmiare connessioni):
bus condiviso interno alla CPU



Permette di risparmiare delle linee di dato, ma comporta una riduzione delle prestazioni, perché un bus è meno veloce di una connessione punto-punto

Nel seguito, non considereremo questa soluzione
(si parla di bus interno alla CPU!)

Specifica dell'unità di controllo:

1) Come macchina a stati finiti (Modello di Moore)

2) Come microprogramma

Partiamo dalla prima modalità, ottenendo direttamente il diagramma a stati…

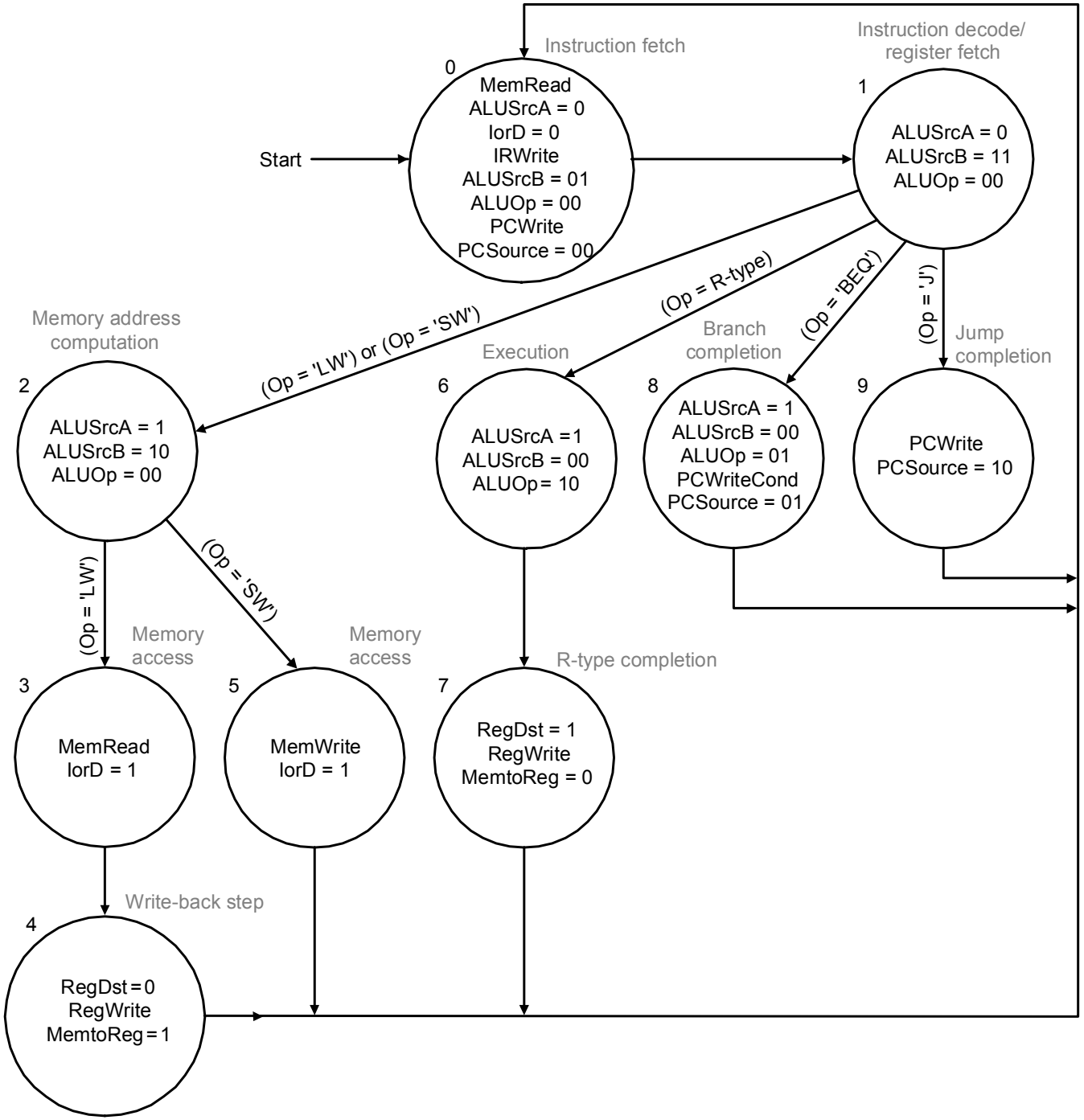
Ad ogni stato sono associate le uscite corrispondenti (cfr. lucidi precedenti):

- segnali di scrittura/lettura:

  non indicati $\Rightarrow$ 0

- segnali di controllo MUX:

  non indicati $\Rightarrow$ valore indifferente


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

Come può essere realizzata la macchina corrispondente a questa specifica?

1) Con una macchina a stati finiti

2) Con tecnica di controllo microprogrammato

Vediamo la prima modalità di realizzazione…