Corso di Calcolatori Elettronici I
A.A. 2012-2013

Architettura del calcolatore

Esempi dettagliati di funzionamento interno di memoria e processore

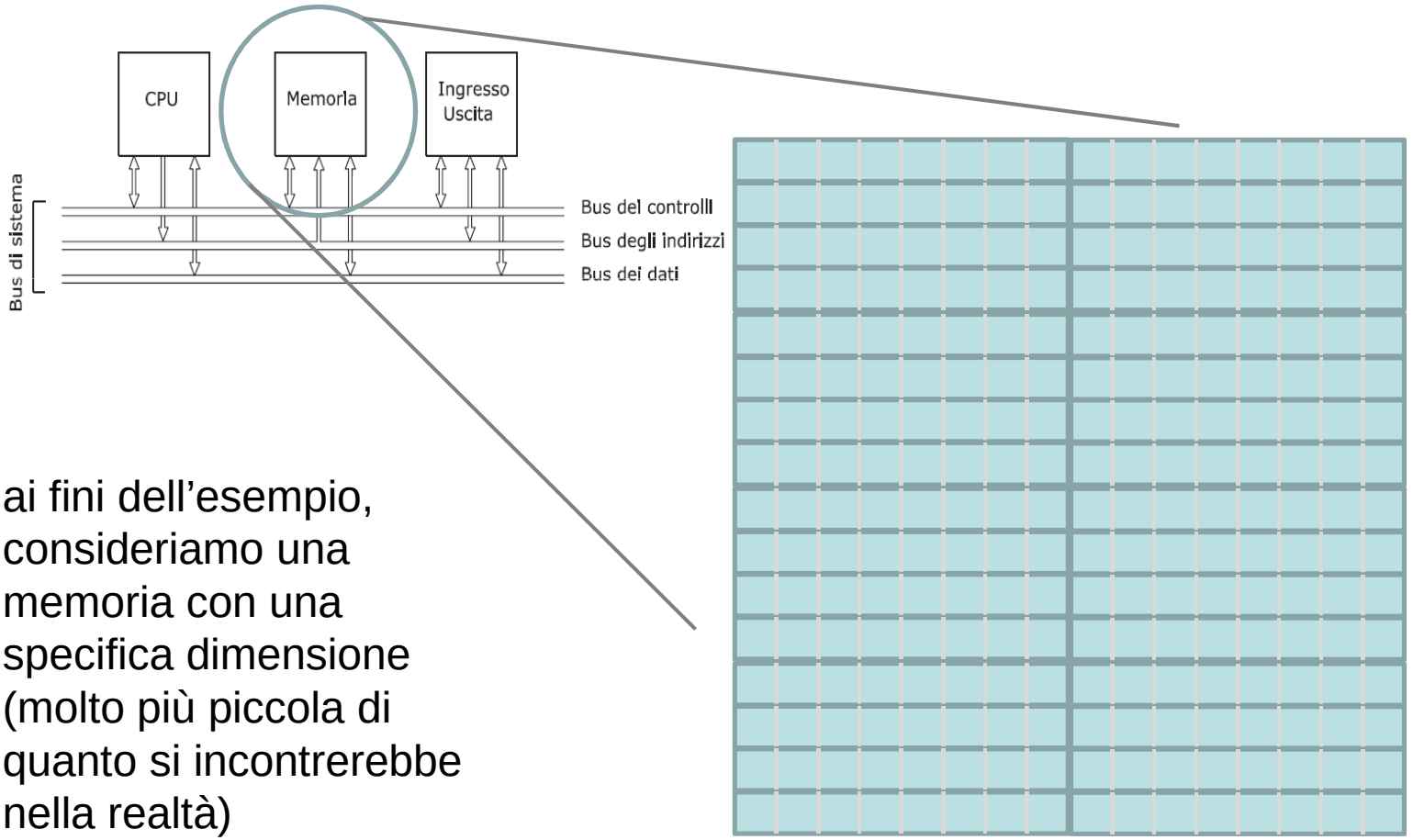
ing. Alessandro Cilardo

Accademia Aeronautica di Pozzuoli
Corso Pegaso V “GArn Elettronici”

Sommario

- In questa presentazione verranno mostrati esempi dettagliati riguardanti:
  - interazione processore-memoria: operazioni di scrittura in memoria
  - interazione processore-memoria: operazioni di lettura da memoria
  - caricamento ed esecuzione di sequenze di istruzioni

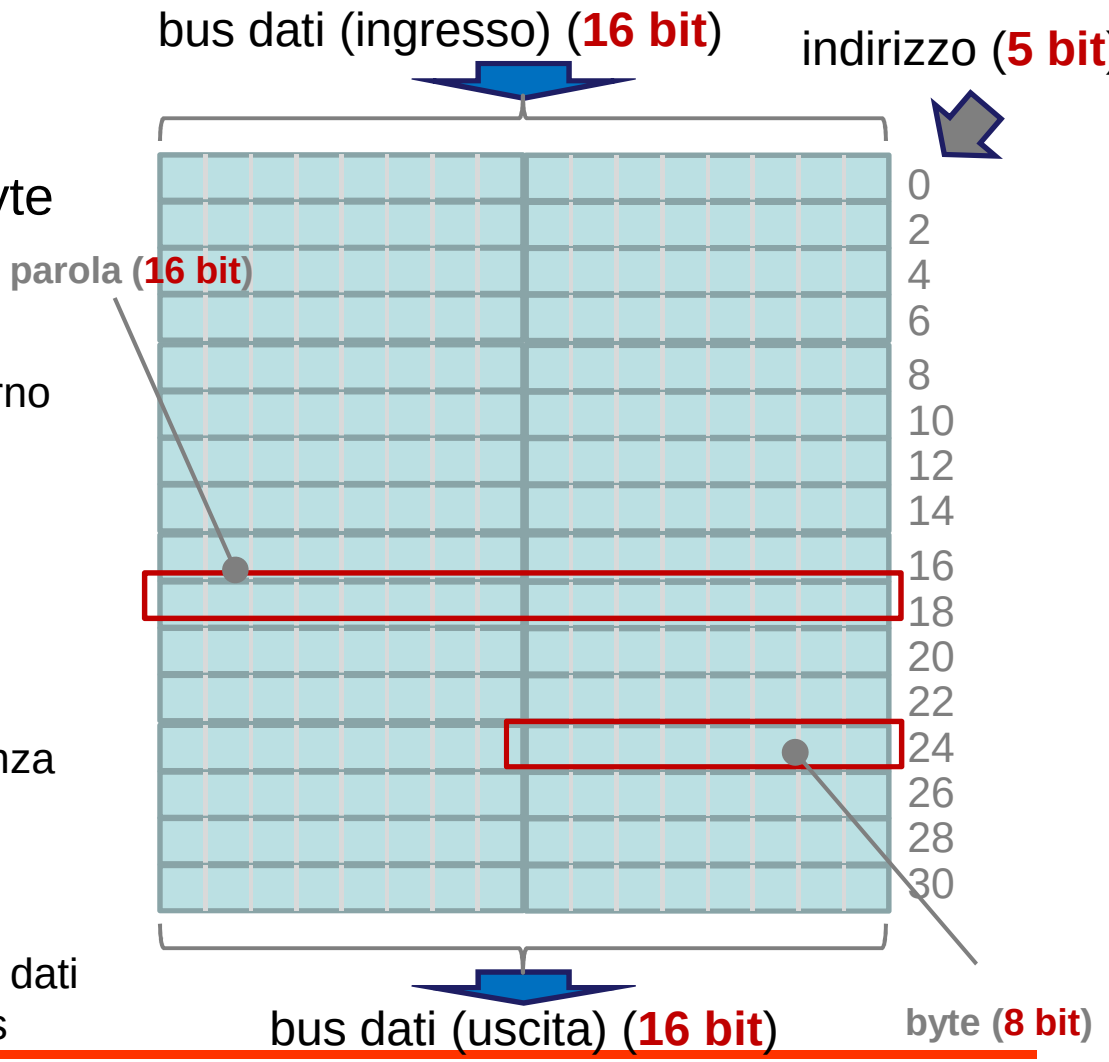
Esempio di memoria



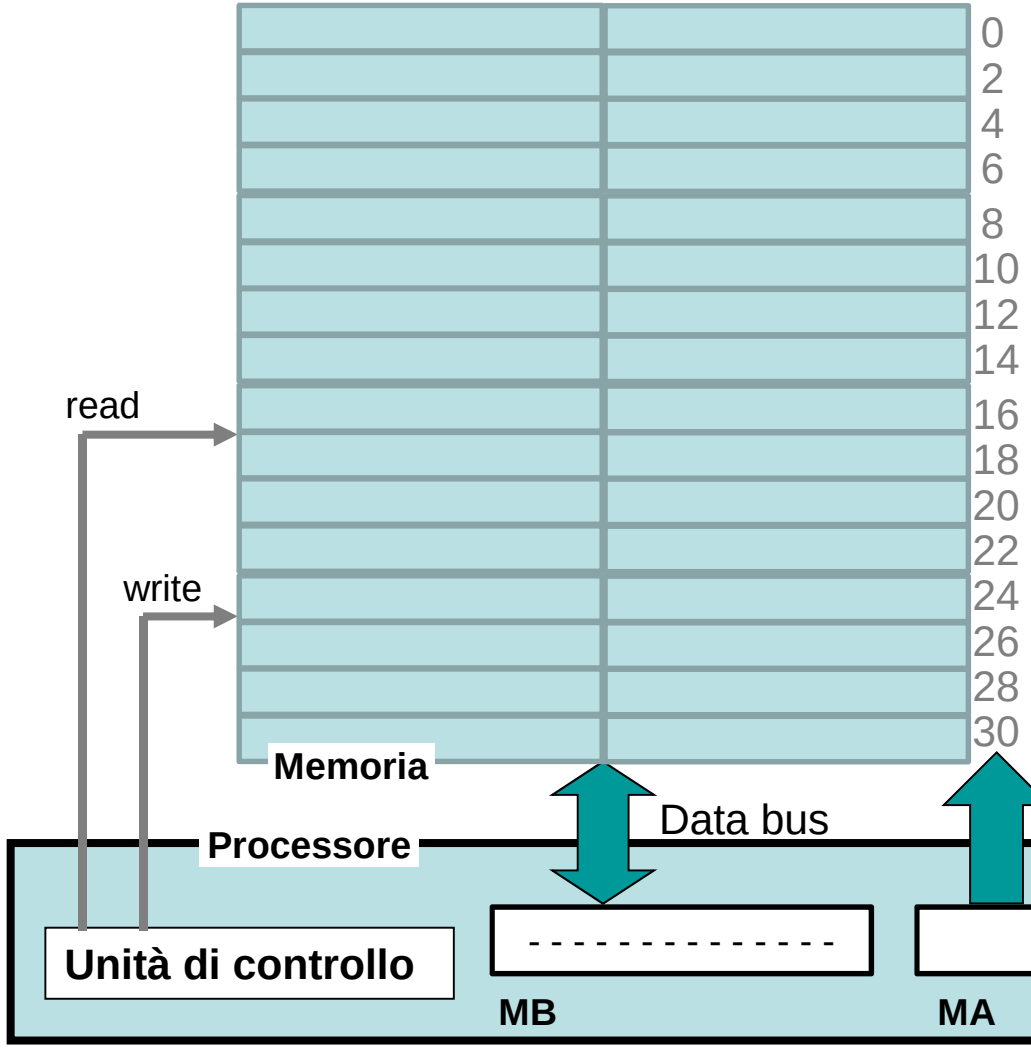
ai fini dell'esempio, consideriamo una memoria con una specifica dimensione (molto più piccola di quanto si incontrerebbe nella realtà)

Esempio di memoria

- Capacità totale: **32 byte**
- Si può vedere come una sequenza lineare di 32 byte
- E' *byte-addressable*:
  - l'indirizzo si riferisce alla **posizione del byte** all'interno di questa sequenza
  - l'indirizzo è a **5 bit** (per indirizzare $32=2^5$ locazioni)
- E' organizzata in parole (*word*) da due byte:
  - si può vedere come sequenza di 16 word da 16 bit
  - la parola di **16 bit** è l'unità minima con la quale si possono leggere o scrivere dati dall'esterno attraverso i bus



Esempio di operazione di scrittura (1/4)



- Supponiamo di voler scrivere la seguente parola di 16 bit:

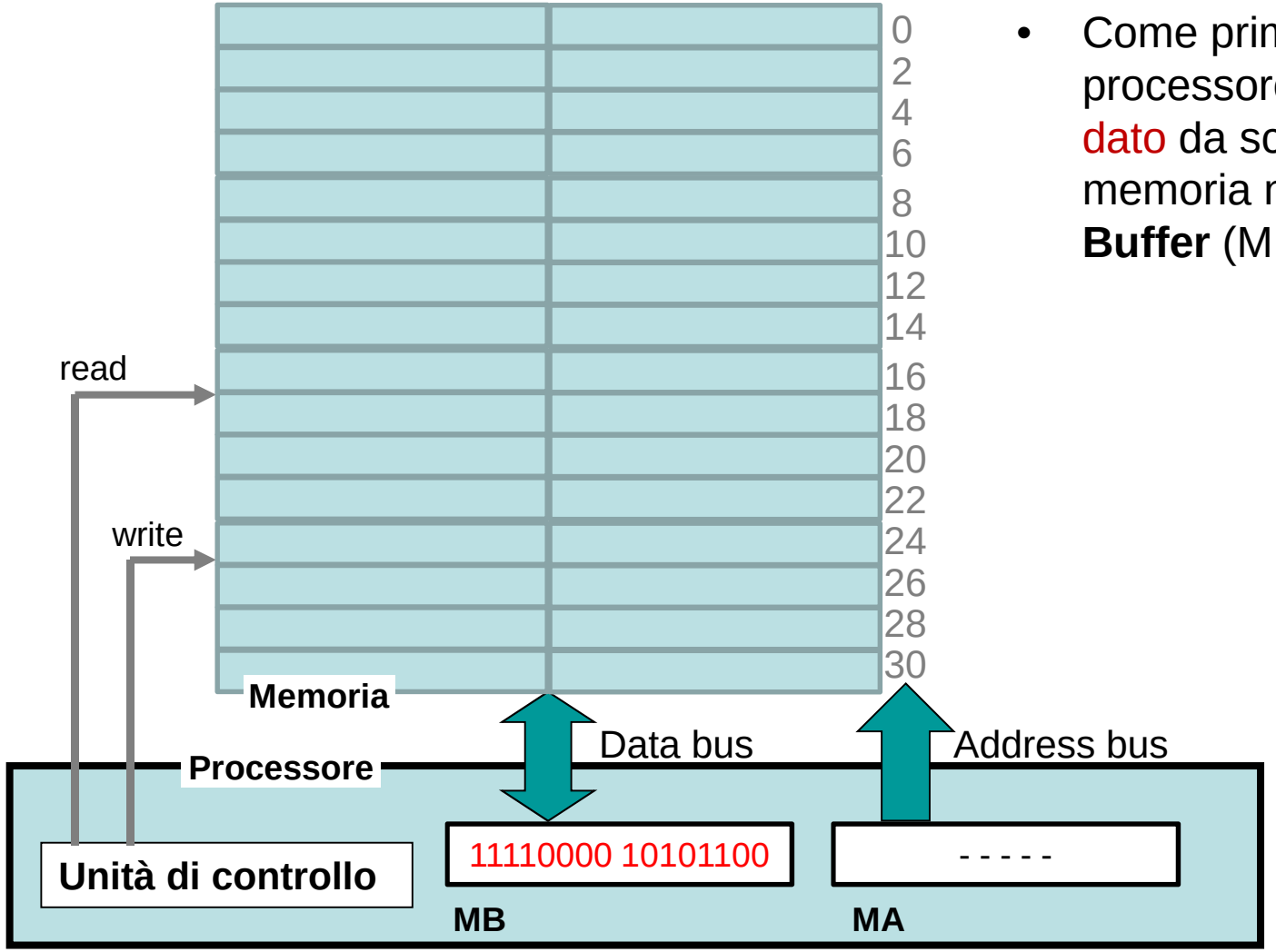
  11110000 10101100

- all'indirizzo **14** in memoria

immaginiamo che il processore usi una convenzione **big-endian**:

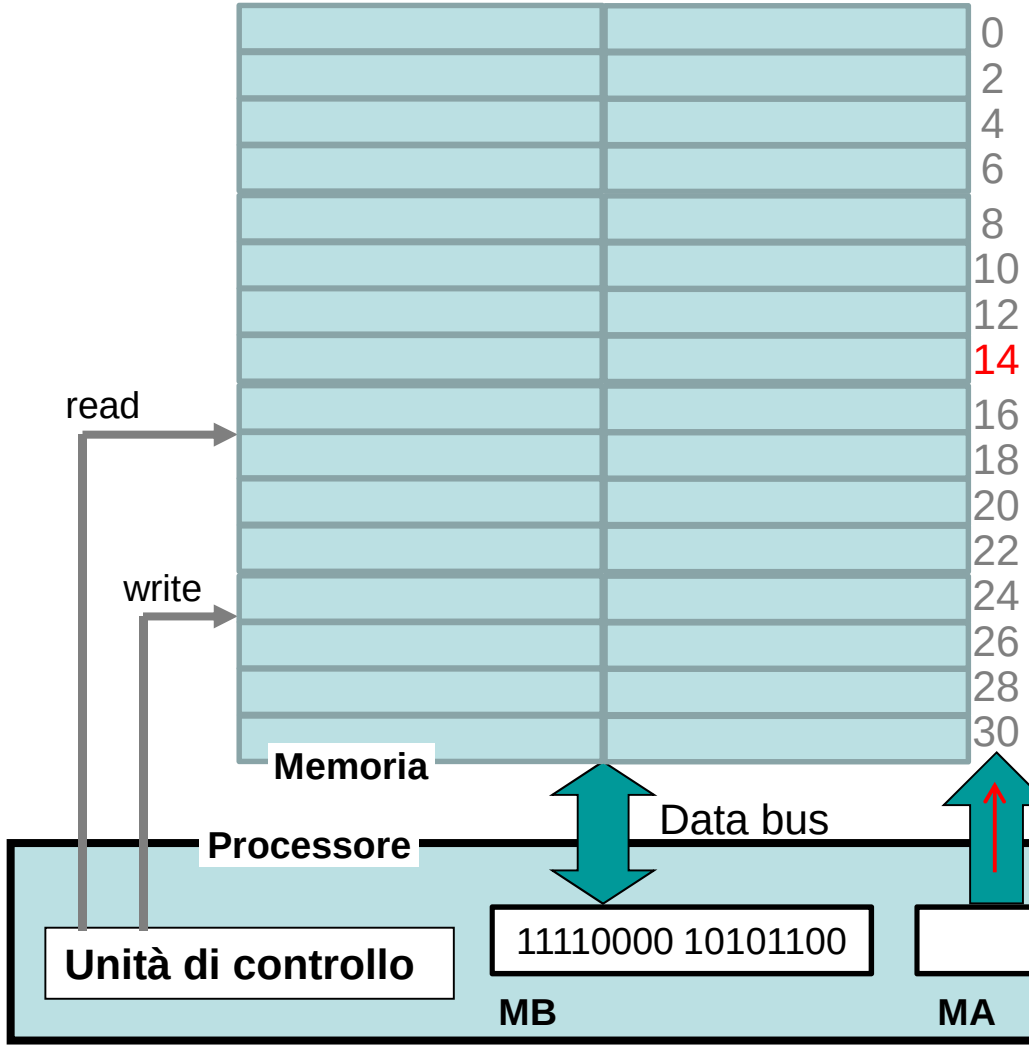
byte di peso maggiore sono memorizzati in locazioni di indirizzo minore

Esempio di operazione di scrittura (2/4)



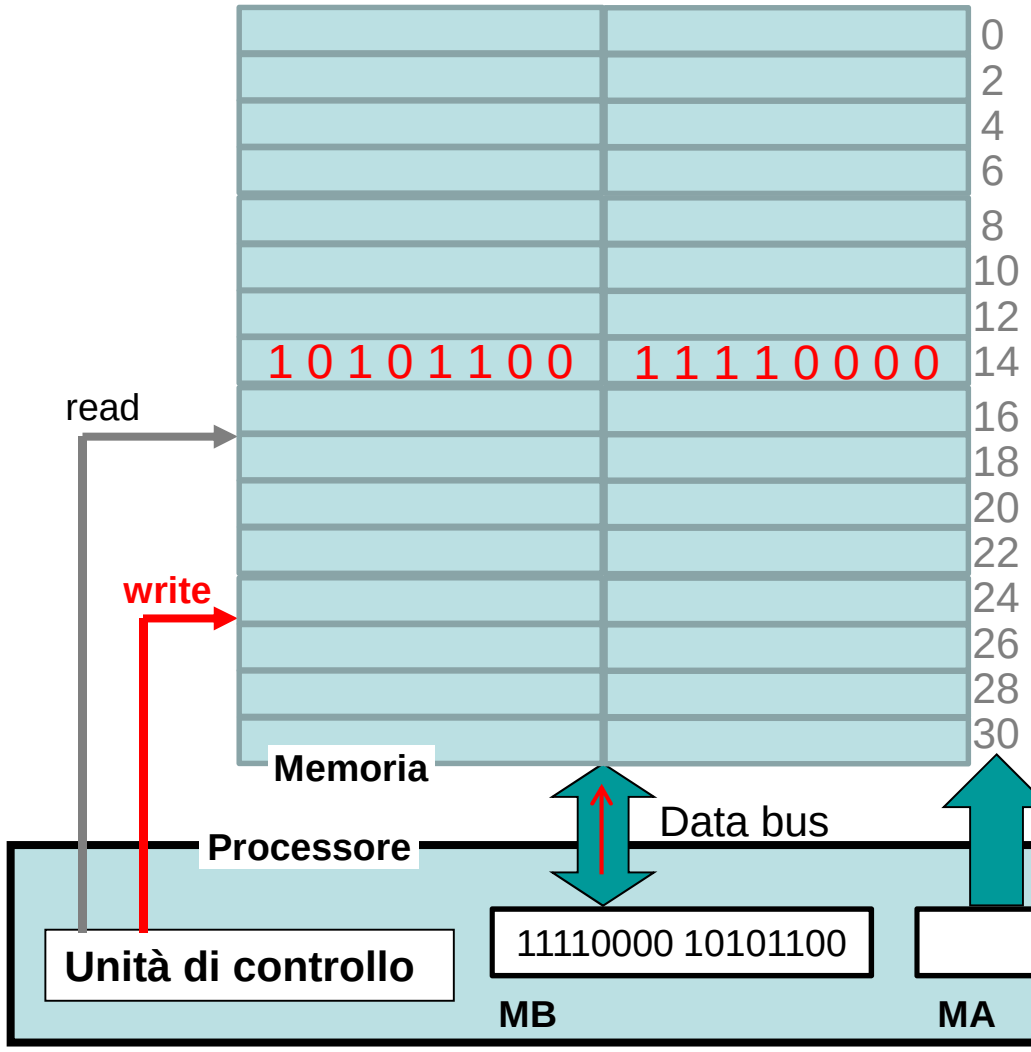
- Come prima cosa, il processore inserisce il dato da scrivere in memoria nel **Memory Buffer** (MB)

Esempio di operazione di scrittura (3/4)



- Successivamente, il processore inserisce l'indirizzo in cui si vuole scrivere il dato in memoria nel **Memory Address** (MA)
- Nell'esempio, l'indirizzo è $(14)_{10} = (01110)_2$

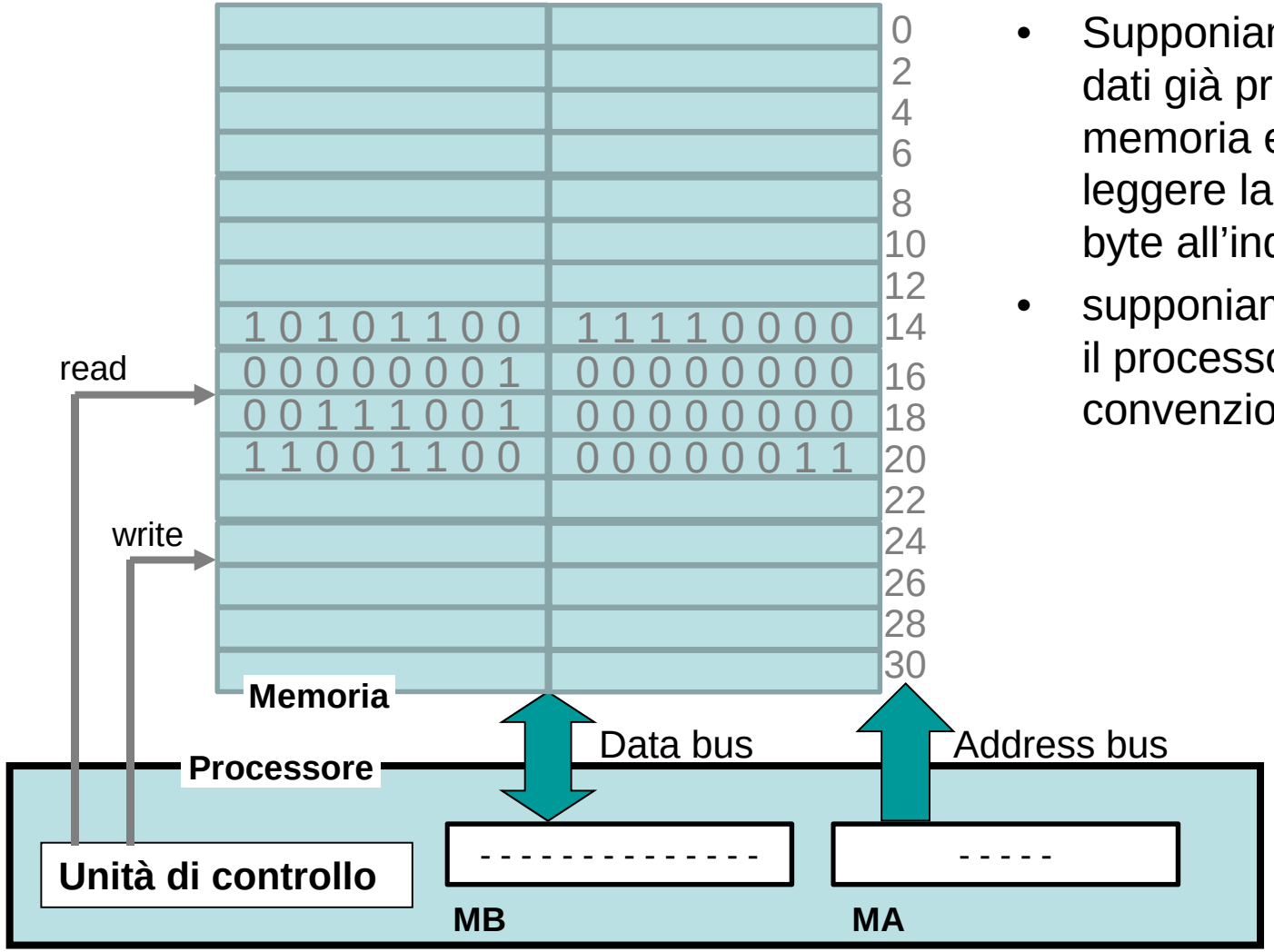
Esempio di operazione di scrittura (4/4)



- Infine, il processore attiva il segnale write per comandare un'operazione di scrittura alla memoria
- Appena il segnale di write si alza, il dato è scritto in memoria all'indirizzo indicato
- Si noti l'ordine **big-endian** con cui vengono memorizzati i due byte:

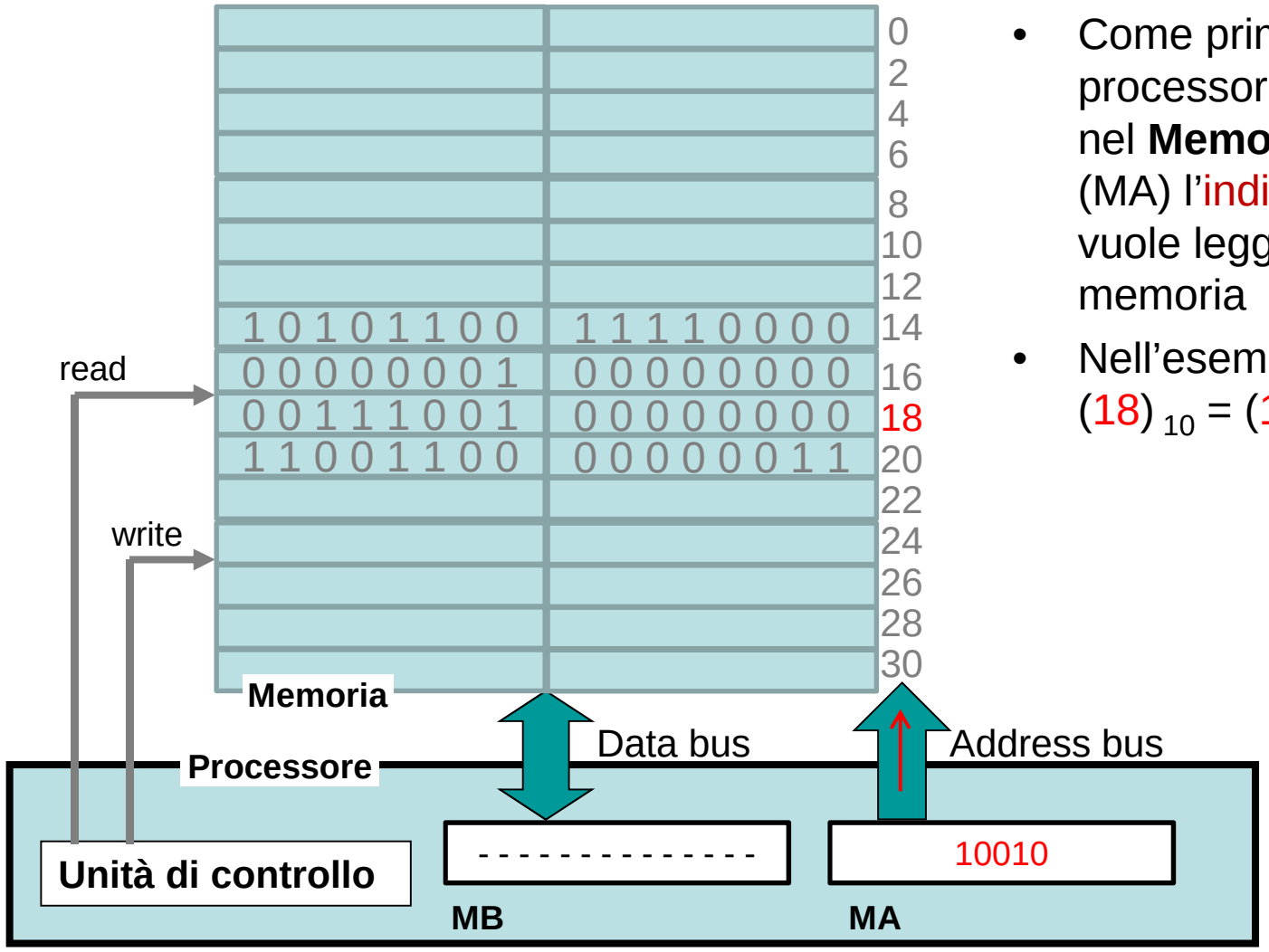
10101100 è all'indirizzo 15, mentre **11110000** all'indirizzo 14.

Esempio di operazione di lettura (1/3)



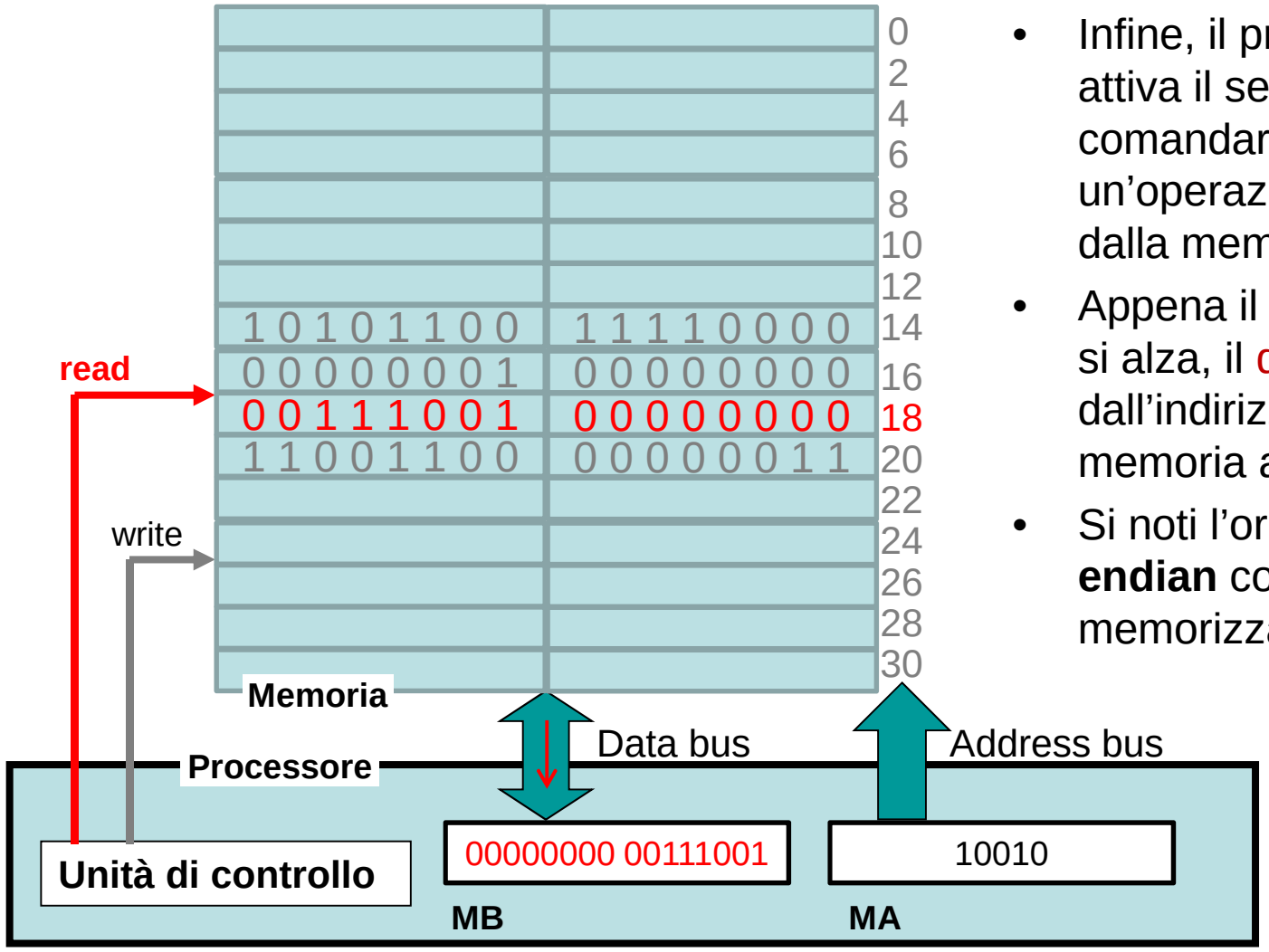
- Supponiamo di avere dei dati già presenti in memoria e di voler leggere la parola di due byte all'indirizzo **18**
- supponiamo sempre che il processore usi una convenzione **big-endian**

Esempio di operazione di lettura (2/3)



- Come prima cosa, il processore inserisce nel **Memory Address** (MA) l'indirizzo da cui si vuole leggere il dato in memoria
- Nell'esempio, l'indirizzo è $(18)_{10} = (10010)_2$

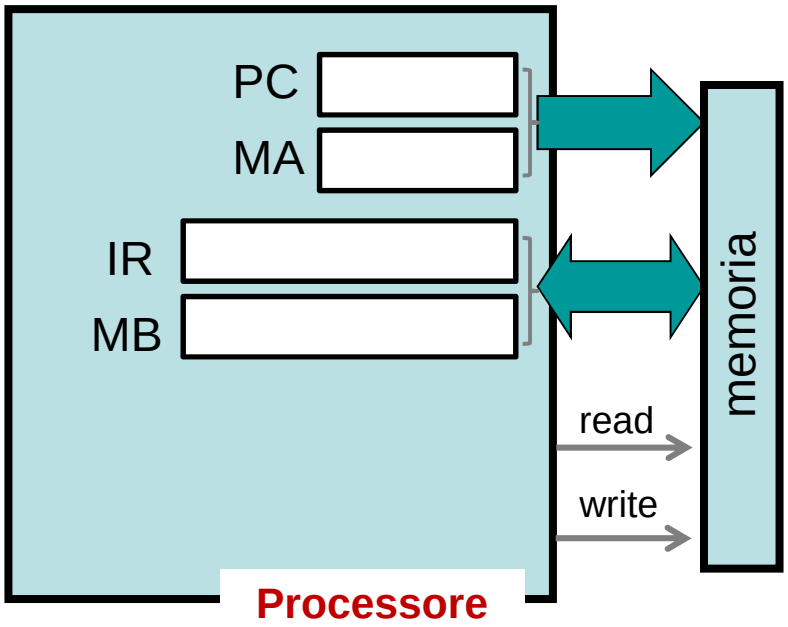
Esempio di operazione di lettura (3/3)



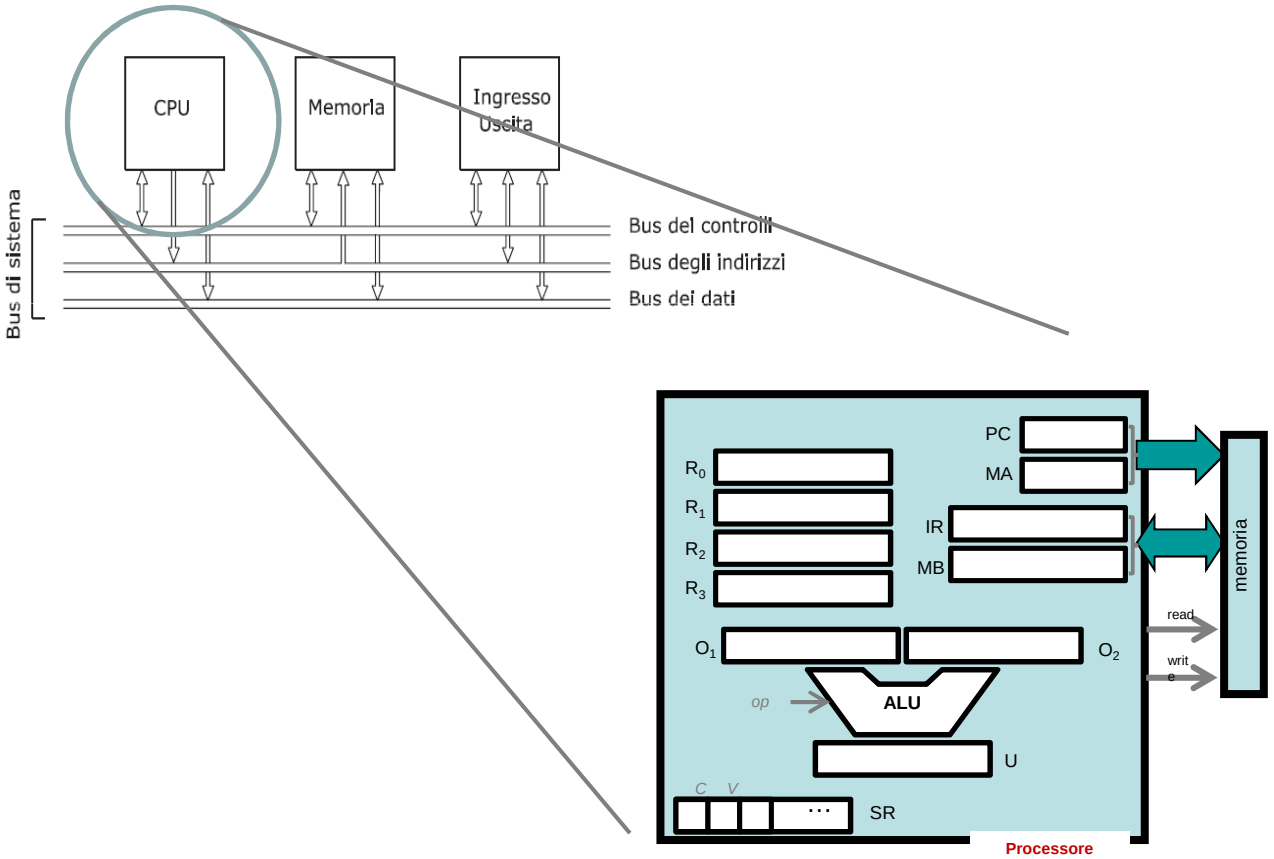
- Infine, il processore attiva il segnale read per comandare un'operazione di lettura dalla memoria
- Appena il segnale di read si alza, il dato è trasferito dall'indirizzo indicato in memoria al registro MB
- Si noti l'ordine **big-endian** con cui sono memorizzati i due byte

Interfaccia processore-memoria

- Oltre alla coppia di registri **MA/MB**, il processore dispone della coppia **PC/IR**, che hanno esattamente lo stesso ruolo, ma sono usati per *leggere istruzioni* (e non dati) dalla memoria.
  - lettura di istruzioni → ***fetch***
- Tipicamente non sono usati per scritture, ma solo per letture
  - normalmente, le istruzioni non vengono mai modificate

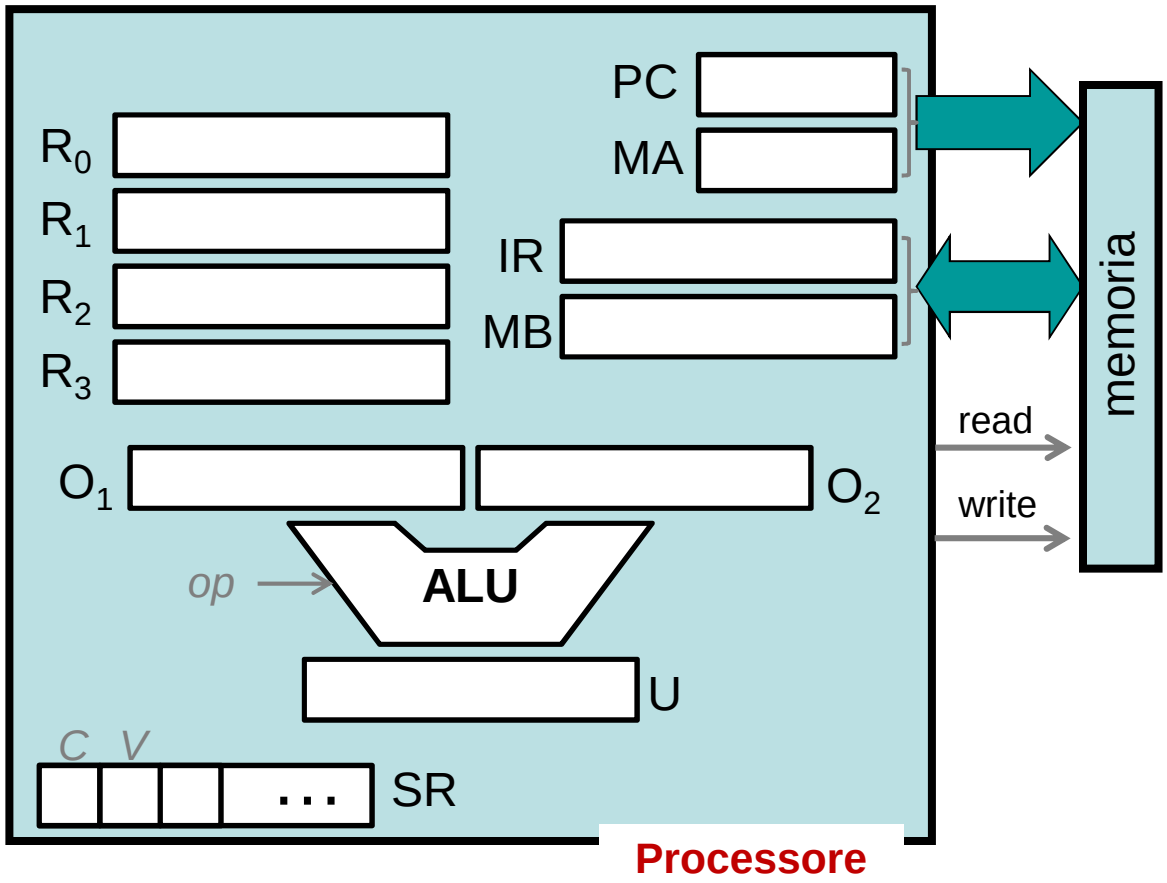


Passiamo al processore…

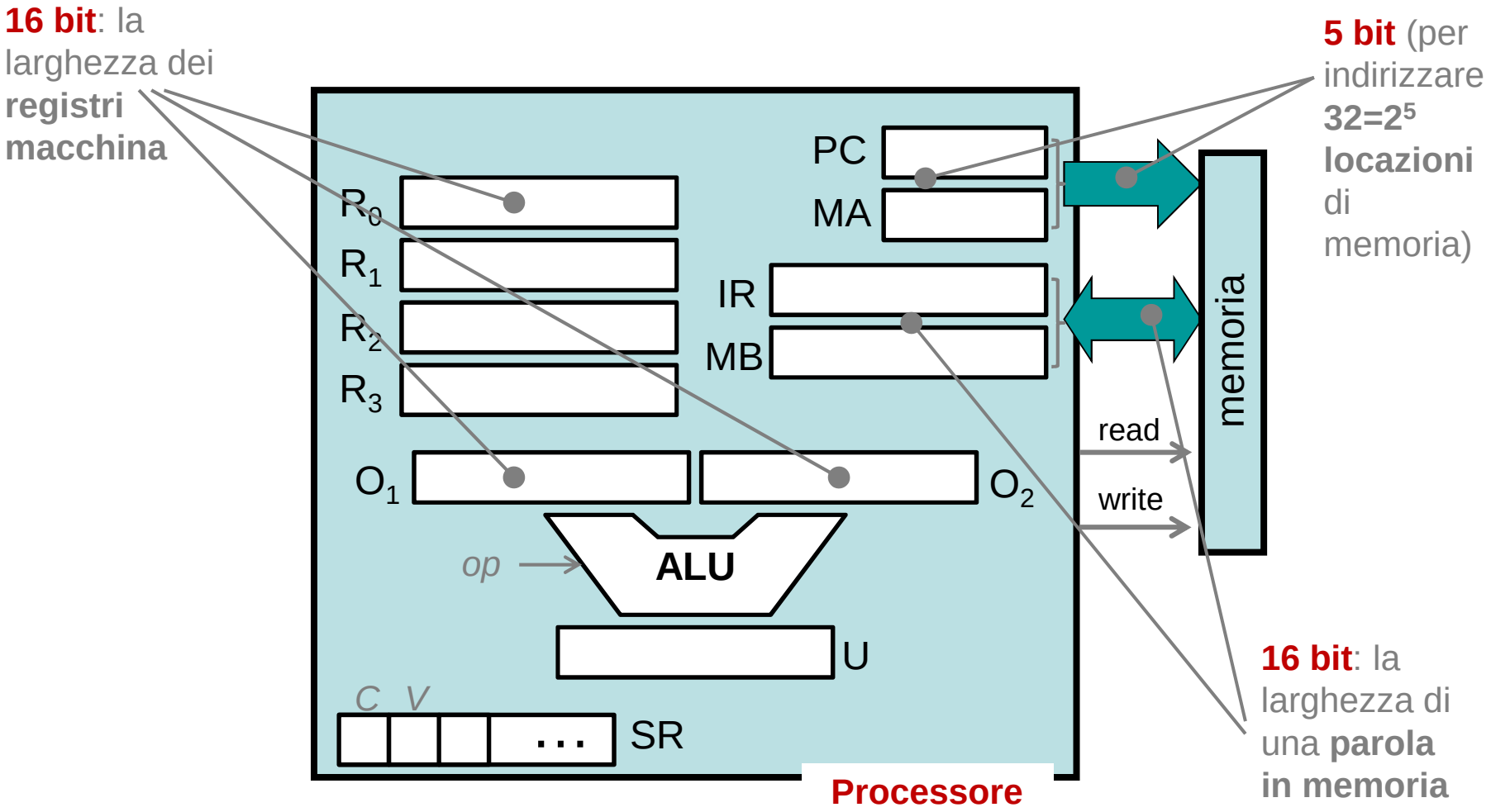


Struttura del processore di esempio

- Quattro registri macchina ad uso generale di **16 bit** (**R_0**…**R_3**)
- **ALU** programmabile tramite il segnale ***op*** per effettuare somme, sottrazioni e shift
- **PC** e **MA** di **5 bit**: si possono generare 2^5=32 indirizzi verso la memoria
- I dati letti dalla memoria (e quindi **MB** e **IR**) sono a **16 bit**
- Registro di stato (SR) con flag di riporto (C) e overflow (V)



Struttura del processore di esempio



“programma” di esempio

- Immaginiamo di voler realizzare il seguente calcolo:

  d = 4*(a – b) + c

- dove in particolare:

  $a = (2131)_{10} = (1000\ 0101\ 0011)_2$

  $b = (1930)_{10} = (\ 111\ 1000\ 1010)_2$

  $c = (\ 331)_{10} = (\ \ \ 1\ 0100\ 1011\ )_2$

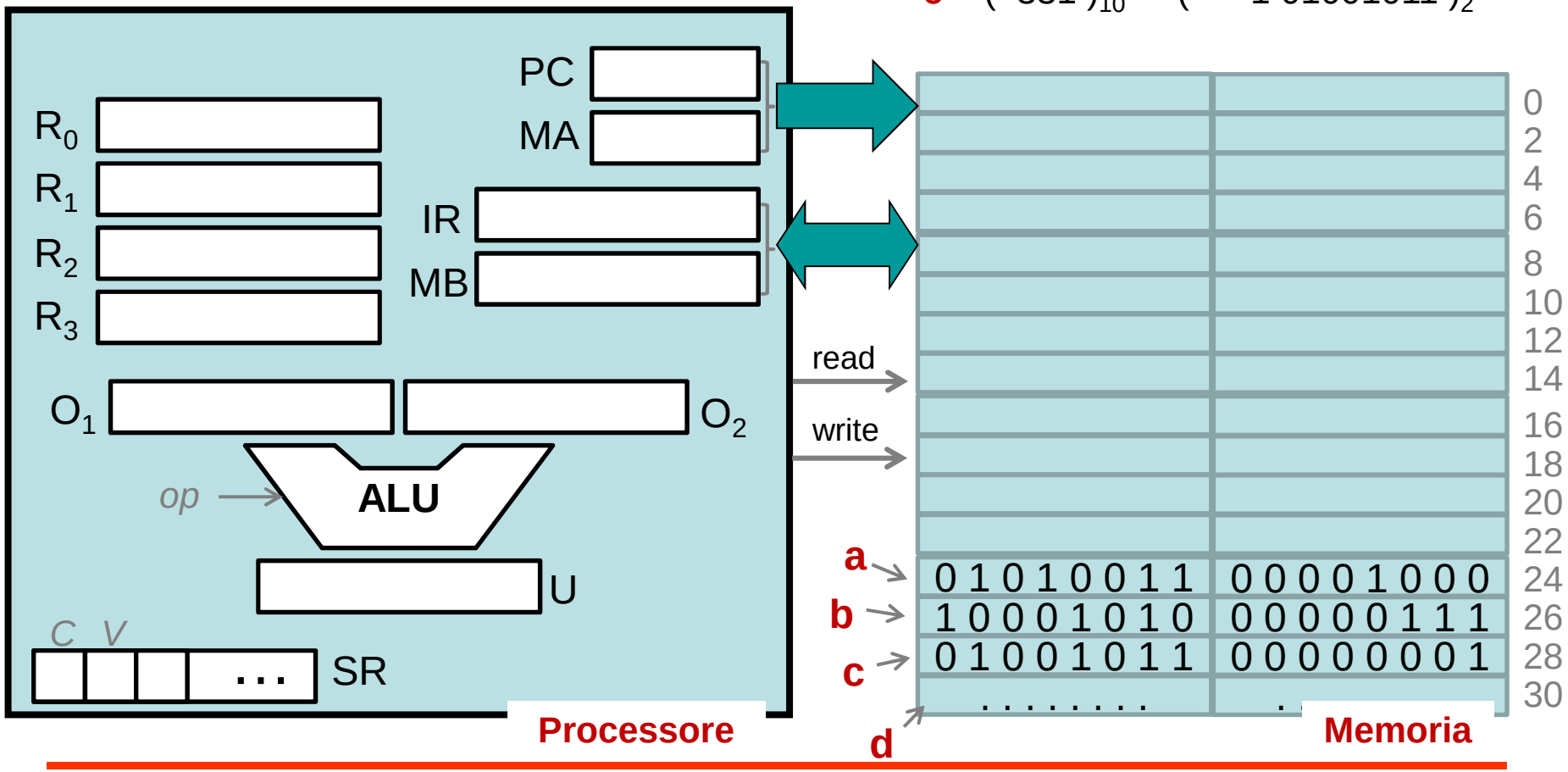
- La moltiplicazione per 4 ($=2^2$) si può realizzare tramite uno shift a sinistra di *due* posizioni
- Le variabili **a**, **b**, **c**, contenenti i valori precedenti, sono memoriazzate in tre locazioni distinte di memoria. Il risultato **d** sarà memorizzato in un’ulteriore locazione

Implementazione del programma

- Si vuole ottenere: **d = 4*(a – b) + c**
- Si supponga che:
  - la variabile **a** sia contenuta in memoria all'indirizzo **24**,
  - la variabile **b** sia contenuta in memoria all'indirizzo **26**,
  - la variabile **c** sia contenuta in memoria all'indirizzo **28**,
  - e che si voglia memorizzare la variabile risultato **d** all'indirizzo **30**

Variabili in memoria (*big-endian*)

a = (2131)$_{10}$ = (1000 01010011)$_2$
b = (1930)$_{10}$ = (111 10001010)$_2$
c = (331)$_{10}$ = (1 01001011)$_2$



Implementazione del programma

- Dobbiamo realizzare il precedente calcolo sfruttando le operazioni elementari messe a disposizione dal processore, che sono le seguenti:
  - trasferisci un dato da una locazione di memoria (**M**) ad uno dei registri di macchina $\mathbf{R_0}$…$\mathbf{R_3}$, e viceversa:
    - es.: **M[18]** → $\mathbf{R_2}$, $\mathbf{R_1}$ → **M[14]**, ….
  - addiziona i contenuti di due dei registri $\mathbf{R_0}$…$\mathbf{R_3}$,
    - es.: $\mathbf{R_2 + R_1 \rightarrow R_2}$
  - sottrai i contenuti di due dei registri $\mathbf{R_0}$…$\mathbf{R_3}$,
    - es.: $\mathbf{R_3 - R_0 \rightarrow R_0}$
  - transla (shift) il contenuto di uno dei registri $\mathbf{R_0}$…$\mathbf{R_3}$,
    - es.: **shiftL ($\mathbf{R_2}$)** → $\mathbf{R_2}$ (Left-shift, ovvero transla R_2 a sinistra di un bit)

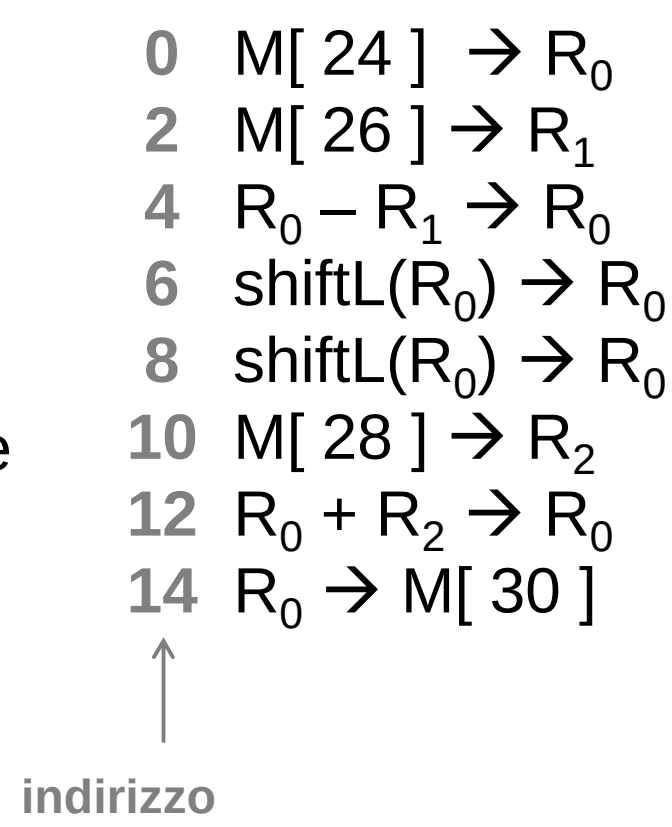
Implementazione del programma

- Si vuole ottenere: **d = 4*(a – b) + c**

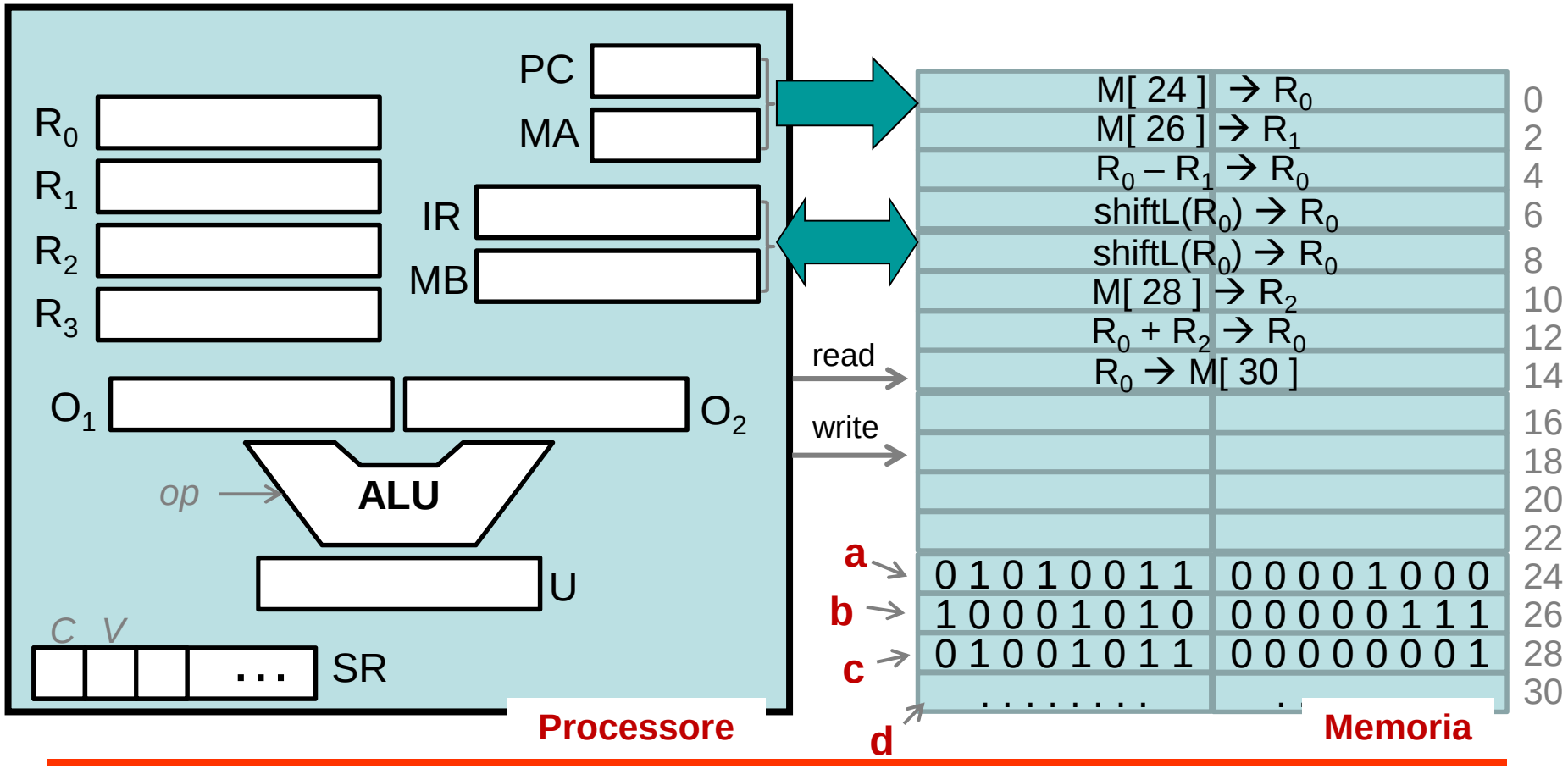
- Un possibile programma che esegue il precedente calcolo è fatto dalle seguenti otto ***istruzioni***:
  - M[24] → R_0 (*sposta **a**, contenuta in mem. all'indirizzo 24, nel reg. R_0*)
  - M[26] → R_1 (*sposta **b**, contenuta in mem. all'indirizzo 26, nel reg. R_1*)
  - $R_0 – R_1 \rightarrow R_0$ (*sottrai il valore di **a** da quello **b** e poni il risultato in R_0*)
  - shiftL(R_0) → R_0 (*trasla a sinistra, ovvero moltiplica per 2, il contenuto di R_0*)
  - shiftL(R_0) → R_0 (*trasla a sinistra, ovvero moltiplica per 2, il contenuto di R_0*)
  - M[28] → R_2 (*sposta **c**, contenuta in mem. all'indirizzo 28, nel reg. R_2*)
  - $R_0 + R_2 \rightarrow R_0$ (*somma **c** con R_0, contenente **4(a-b)** e poni il risultato in R_0*)
  - R_0 → M[30] (*sposta il risultato in **d**, corrispondente all'indirizzo 30 in mem.)*

Programma in memoria

- Il **programma** (le otto istruzioni viste prima) **è esso stesso immagazzinato in memoria**
- Ogni istruzione occupa una parola (due byte)
- Scegliamo di collocare le otto istruzioni nelle prime otto parole in memoria (indirizzi **0**, **2**, **4**, **6**, **8**, **10**, **12**, **14**)



Programma in memoria



Esecuzione passo-passo

- Seguiamo l'esecuzione del precedente programma, soffermandoci su ogni passo necessario per eseguire le sue otto istruzioni

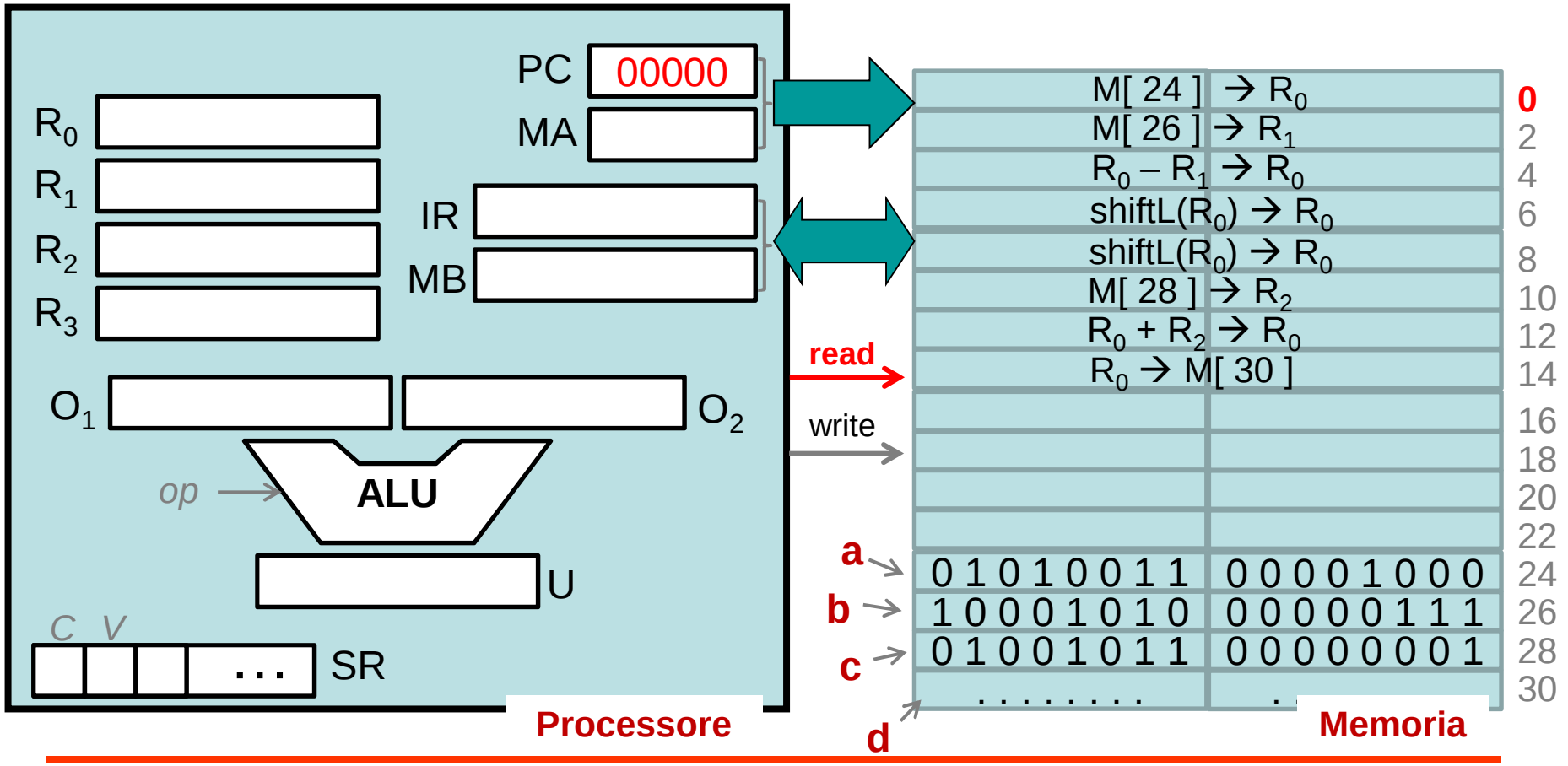
Per cominciare:

- ricordiamo che il ciclo del processore prevede di scrivere nel **PC** l'indirizzo della prossima istruzione da caricare ed eseguire.
- **PC** agisce esattamente come il registro **MA**, facendo riferimento però all'*area istruzioni* → determina una lettura dalla memoria verso il registro interno **IR**
- Il programma, pertanto, può essere avviato **scrivendo l'indirizzo 0** (quello della prima istruzione) nel registro **PC**

Esecuzione passo-passo

Prima istruzione

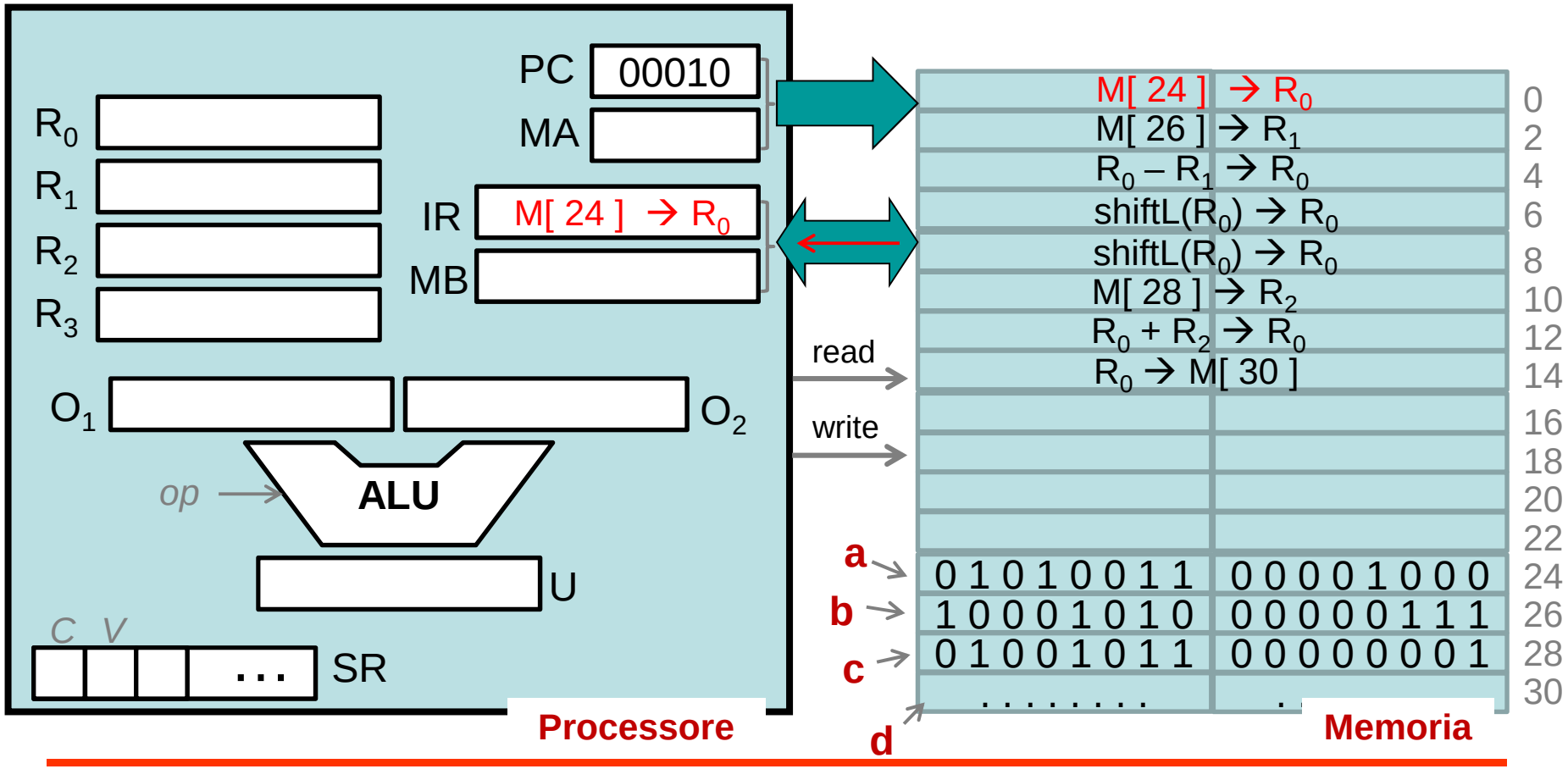
Scriviamo l'indirizzo della prima istruzione (**0**) in **PC**. Avviamo poi una lettura attivando il segnale di **read**. Questa lettura rappresenta il **fetch** dell'istruzione



Esecuzione passo-passo

Prima istruzione

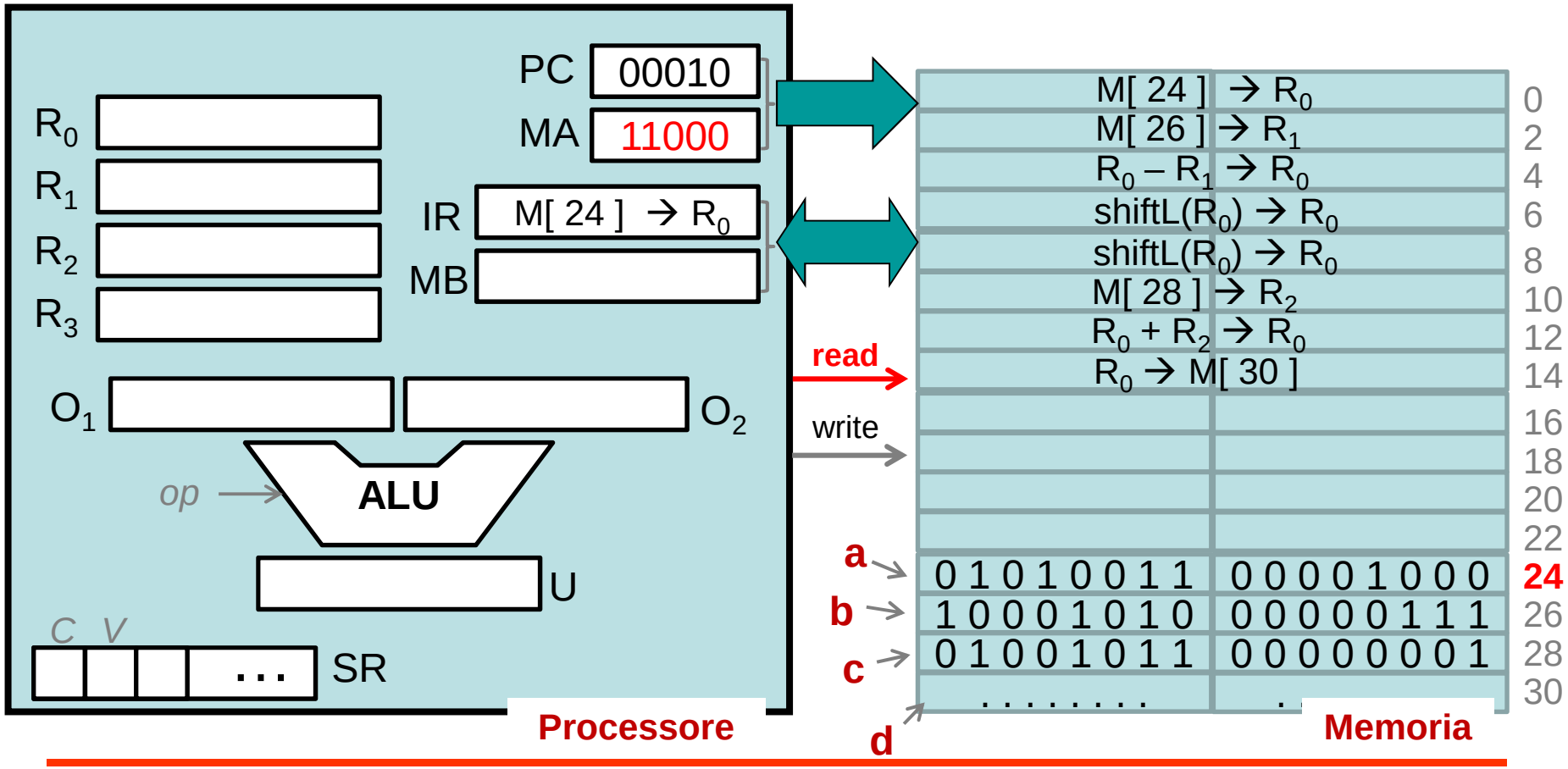
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui serve un accesso in memoria all'indirizzo **24**.



Esecuzione passo-passo

Prima istruzione

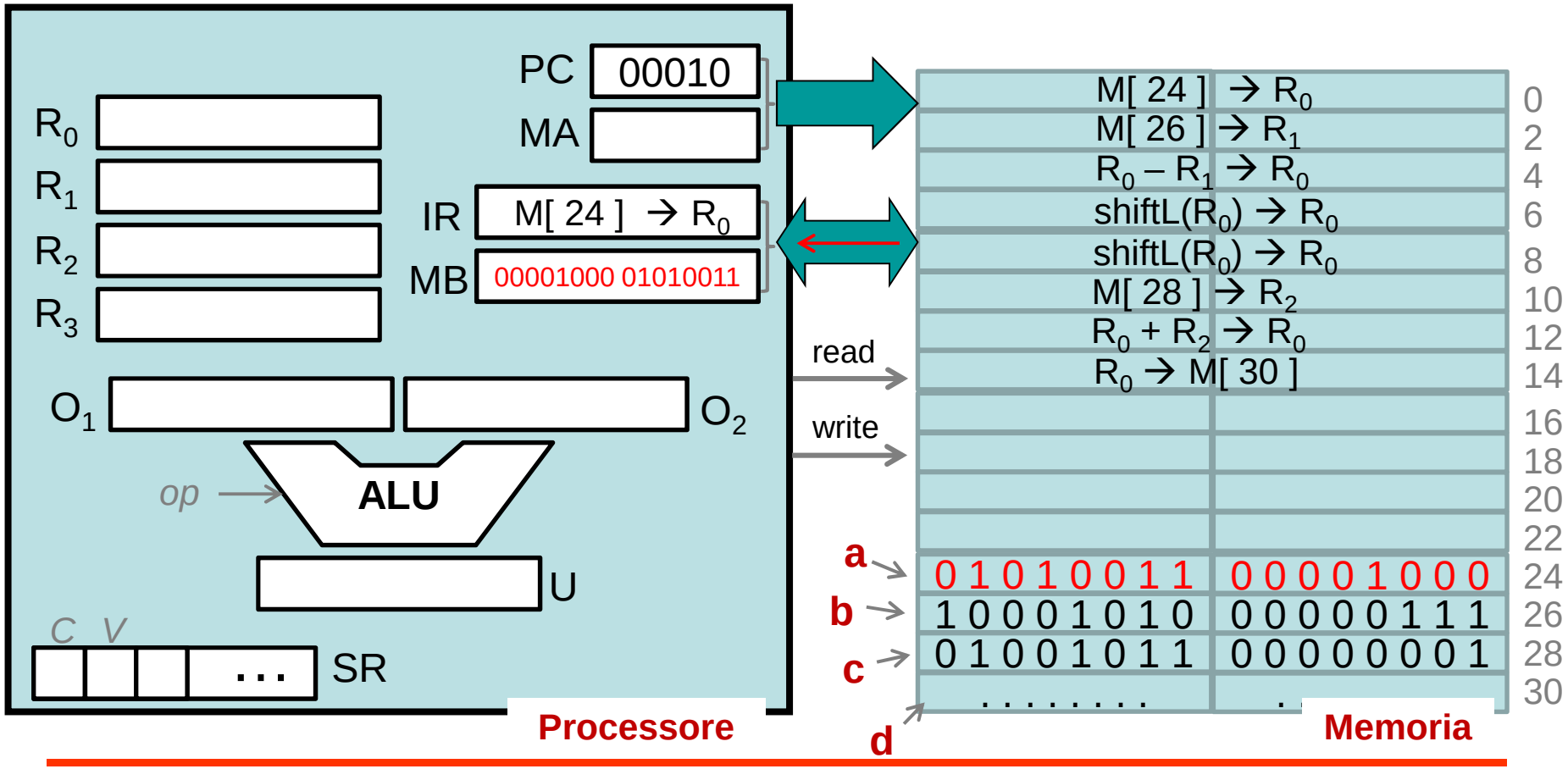
Il processore pone quindi il valore dell'indirizzo **24**=(**11000**)$_2$ all'interno del registro **MA** e comanda un'operazione di lettura (read)



Esecuzione passo-passo

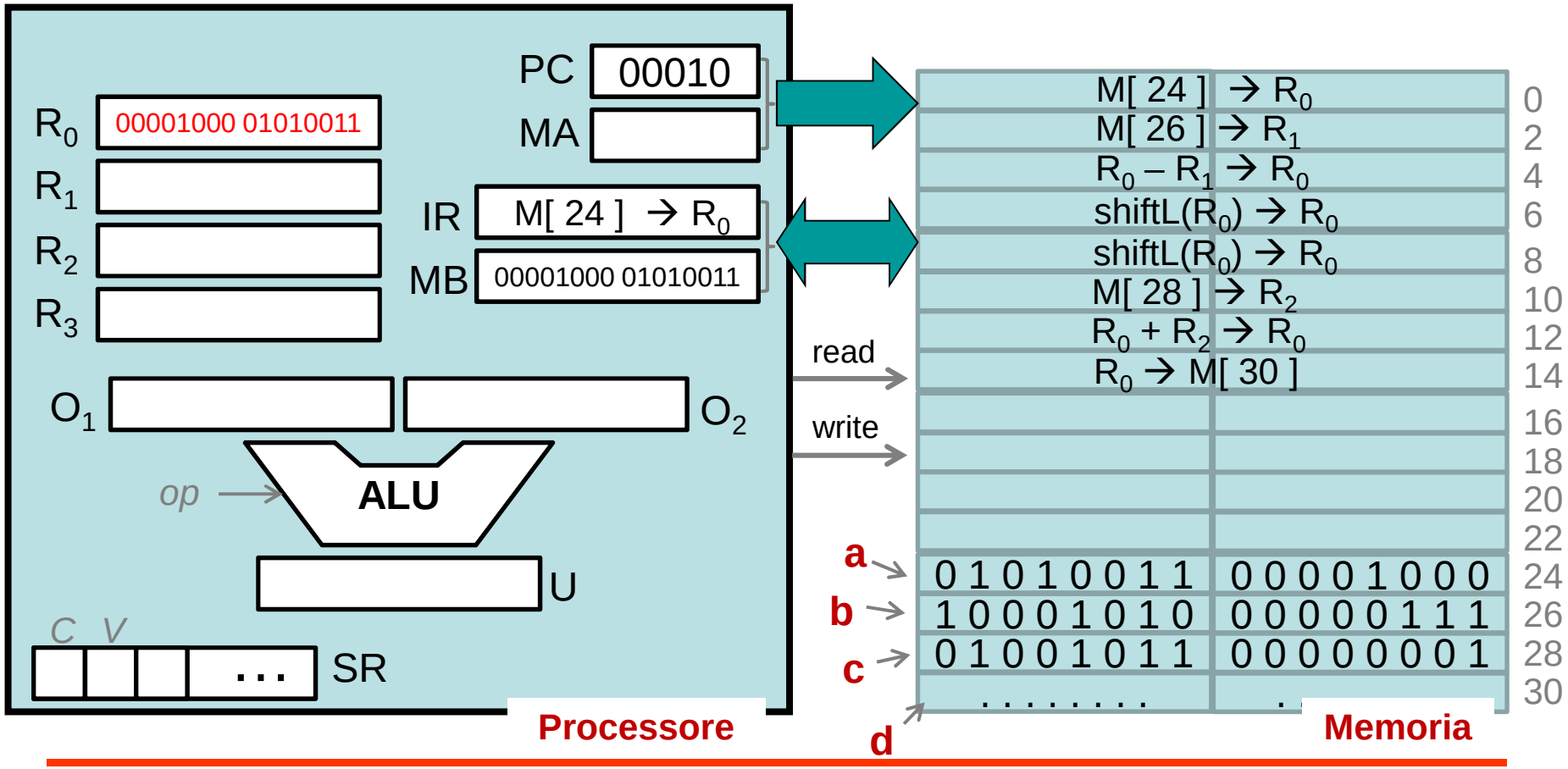
Prima istruzione

La memoria risponde fornendo all'interno di **MB** il contenuto della locazione **24** (la variabile **a**)



Esecuzione passo-passo

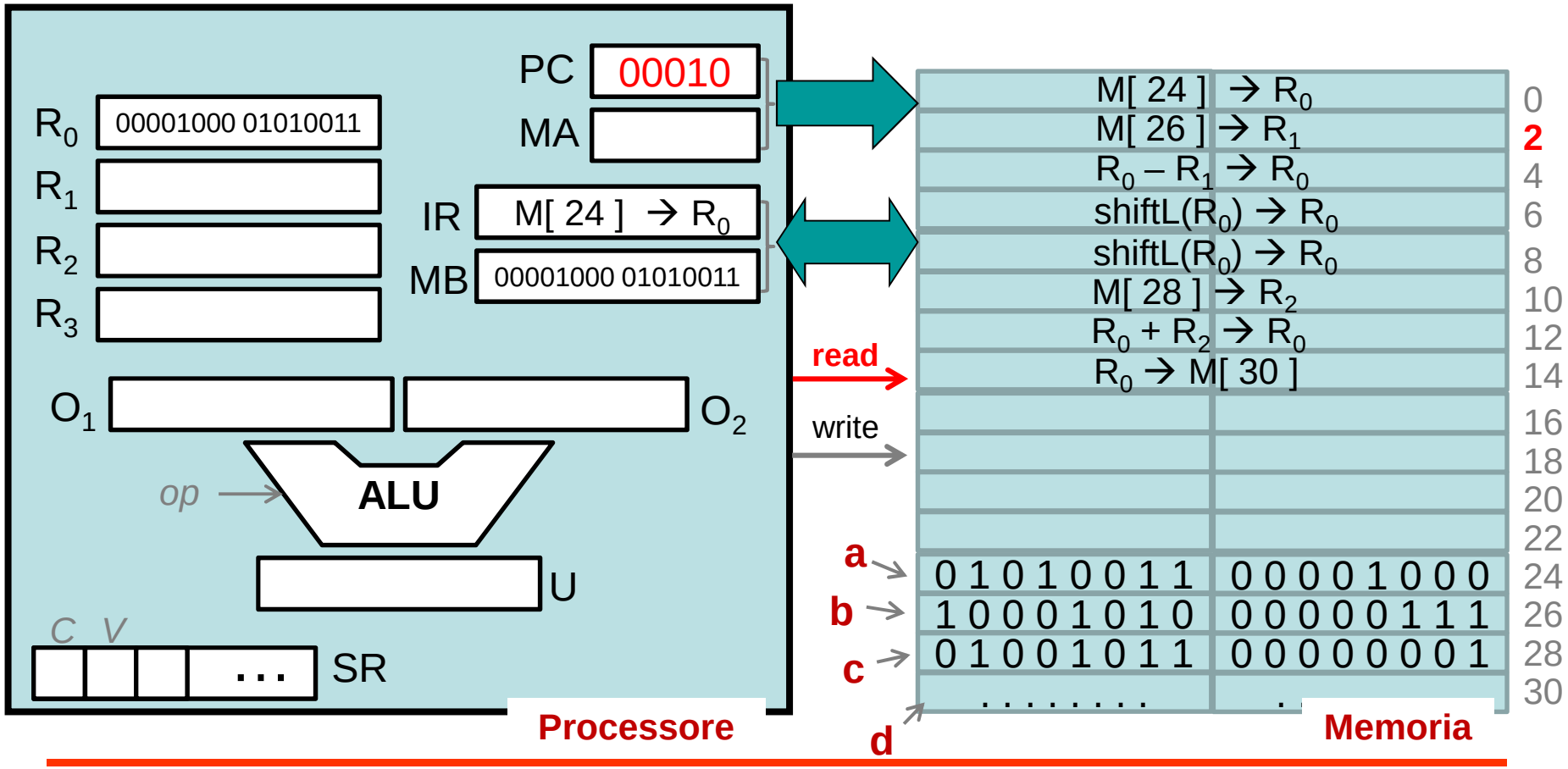
Infine, il valore è spostato da **MB** al registro destinazione indicato dall'istruzione, ovvero **R_0**. L'esecuzione della prima istruzione è completata!



Esecuzione passo-passo

Seconda istruzione

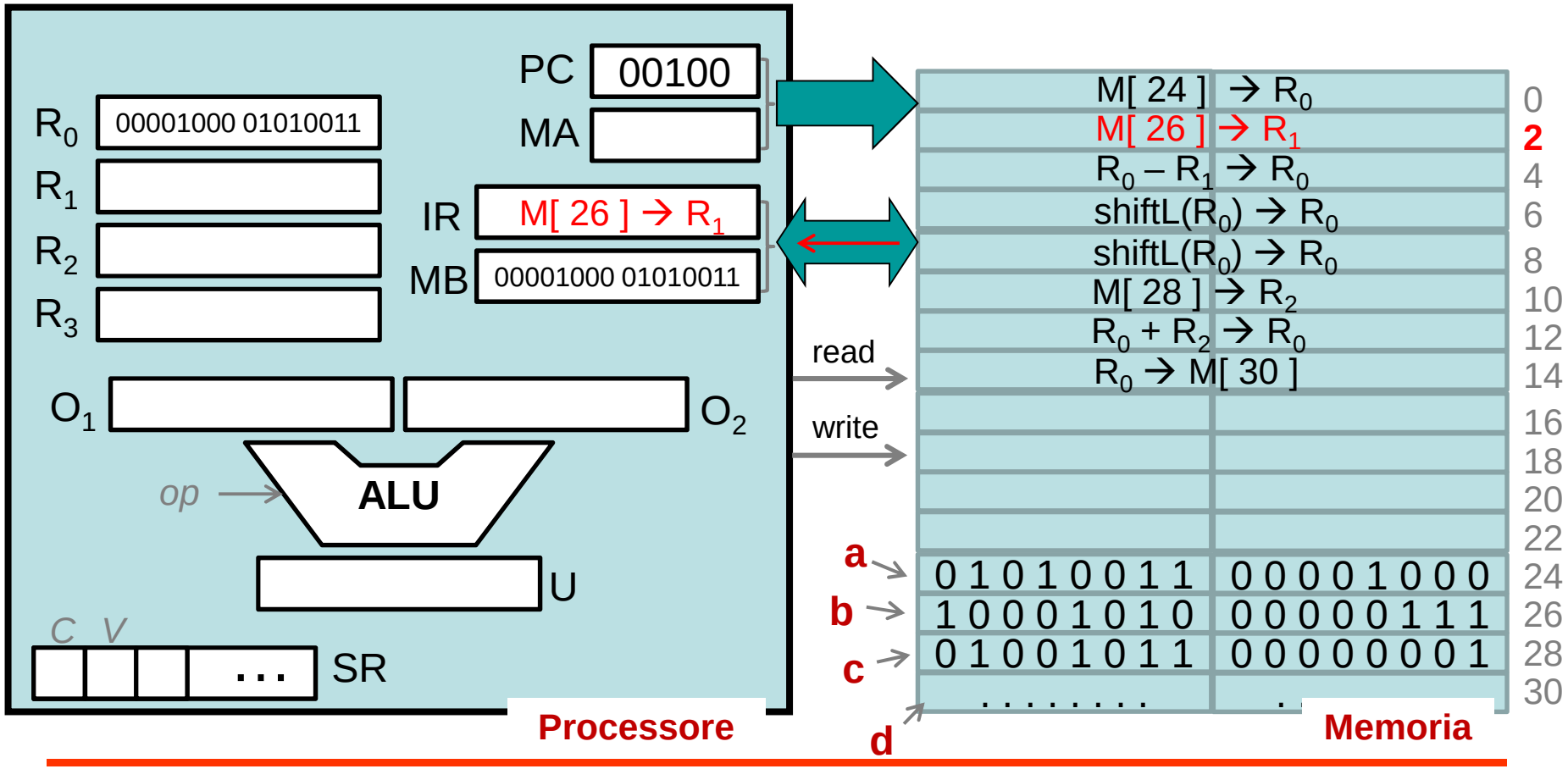
Si noti che nel frattempo **PC** è stato incrementato di 2: ora contiene $(\mathbf{00010})_2=\mathbf{2}$.
In questo modo, il processo di **fetch** si può ripetere con la seconda istruzione!



Esecuzione passo-passo

Seconda istruzione

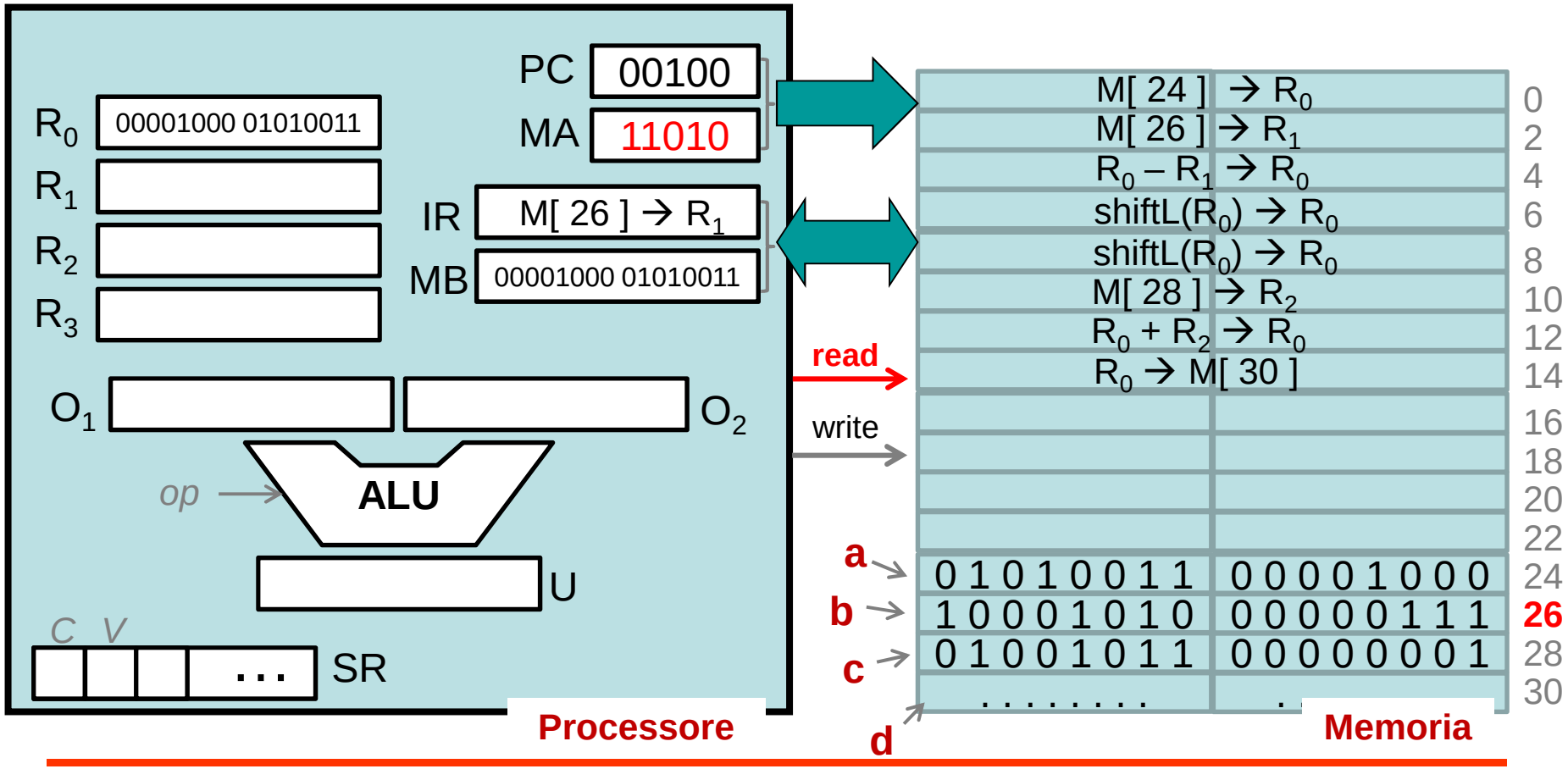
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui serve un accesso in memoria all'indirizzo **26**.



Esecuzione passo-passo

Seconda istruzione

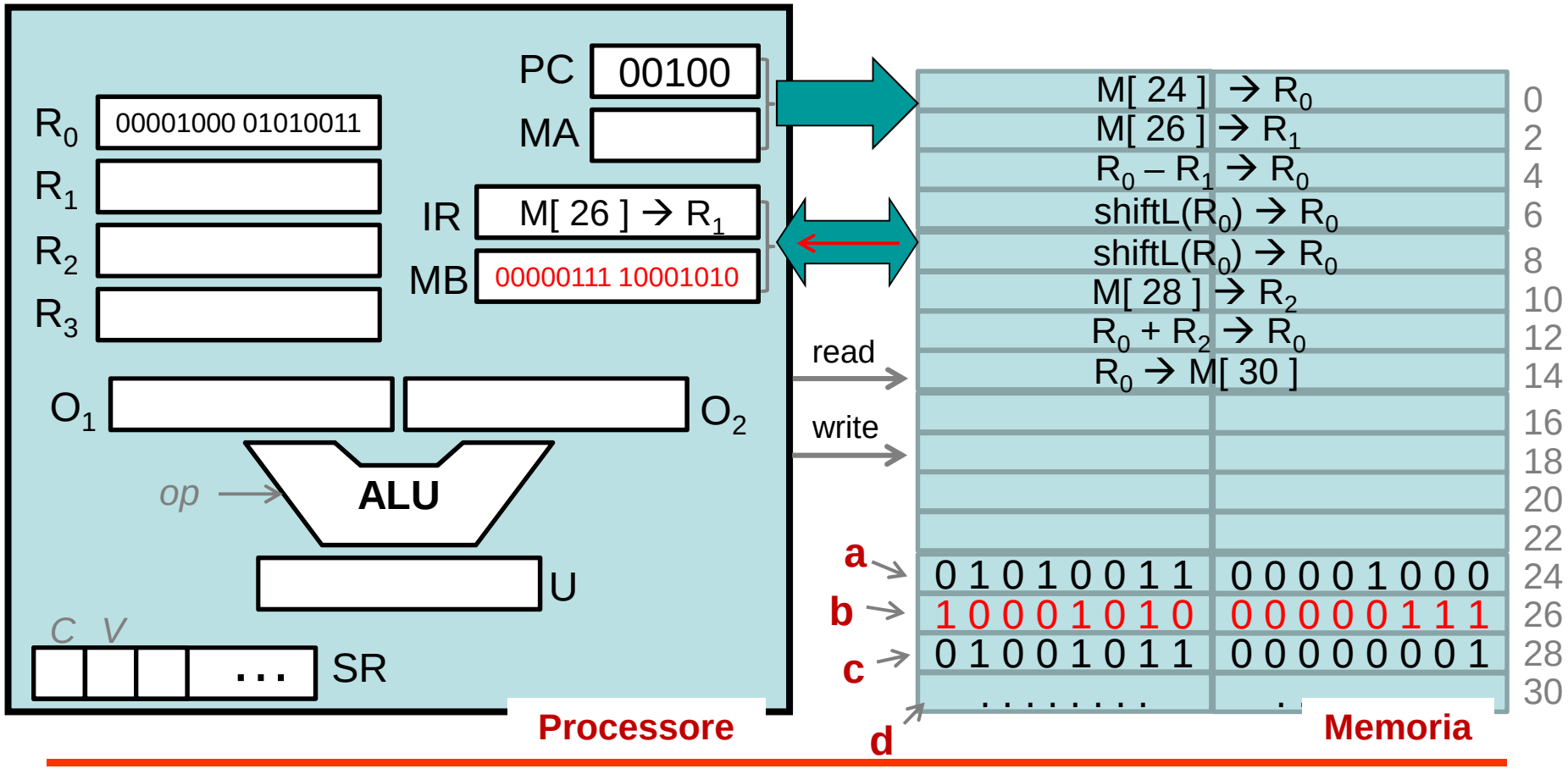
Il processore pone quindi il valore dell'indirizzo **26**=(**11010**)$_2$ all'interno del registro **MA** e comanda un'operazione di lettura (read)



Esecuzione passo-passo

Seconda istruzione

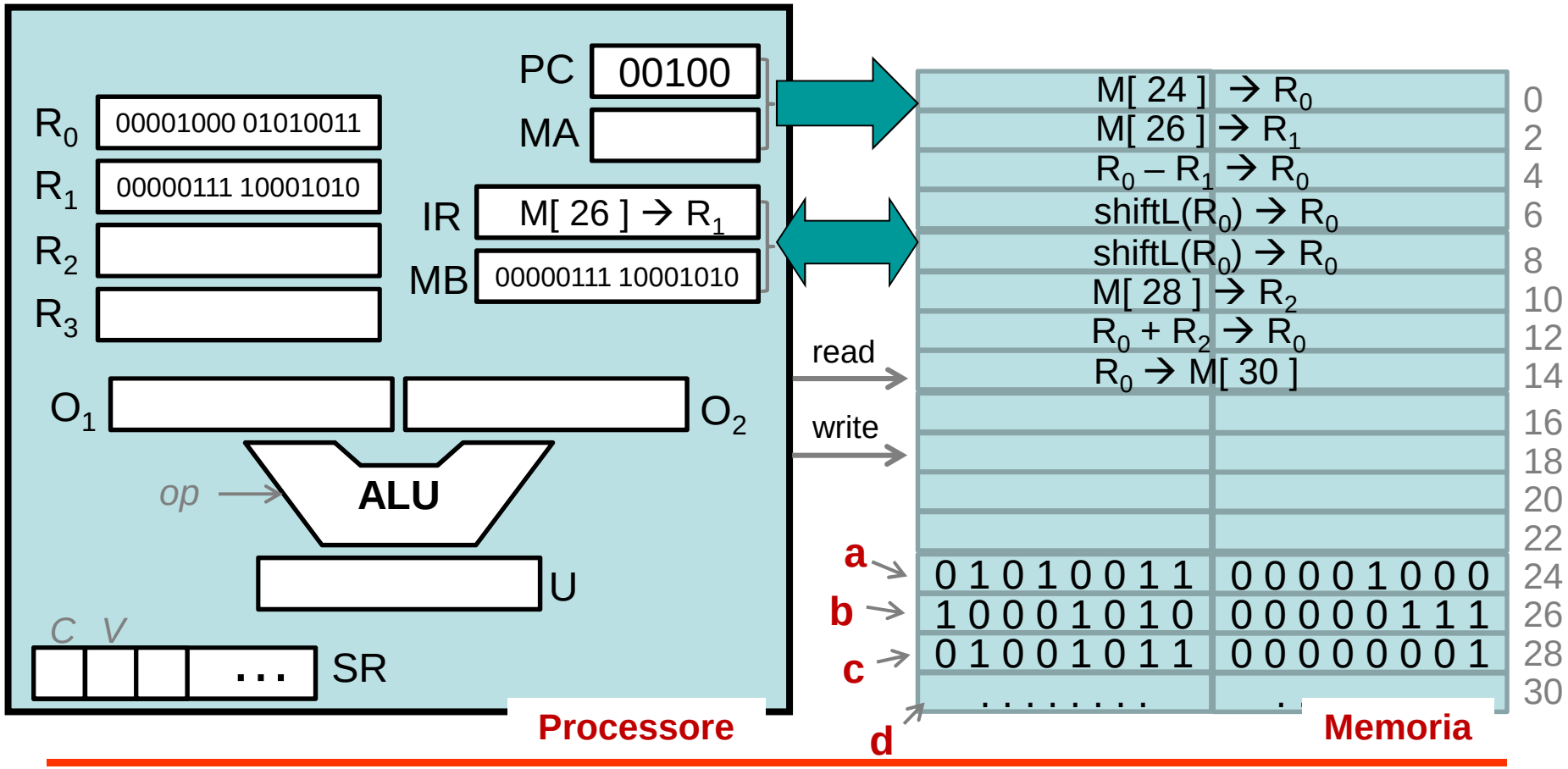
La memoria risponde fornendo all'interno di **MB** il contenuto della locazione **26** (la variabile **b**)



Esecuzione passo-passo

Seconda istruzione

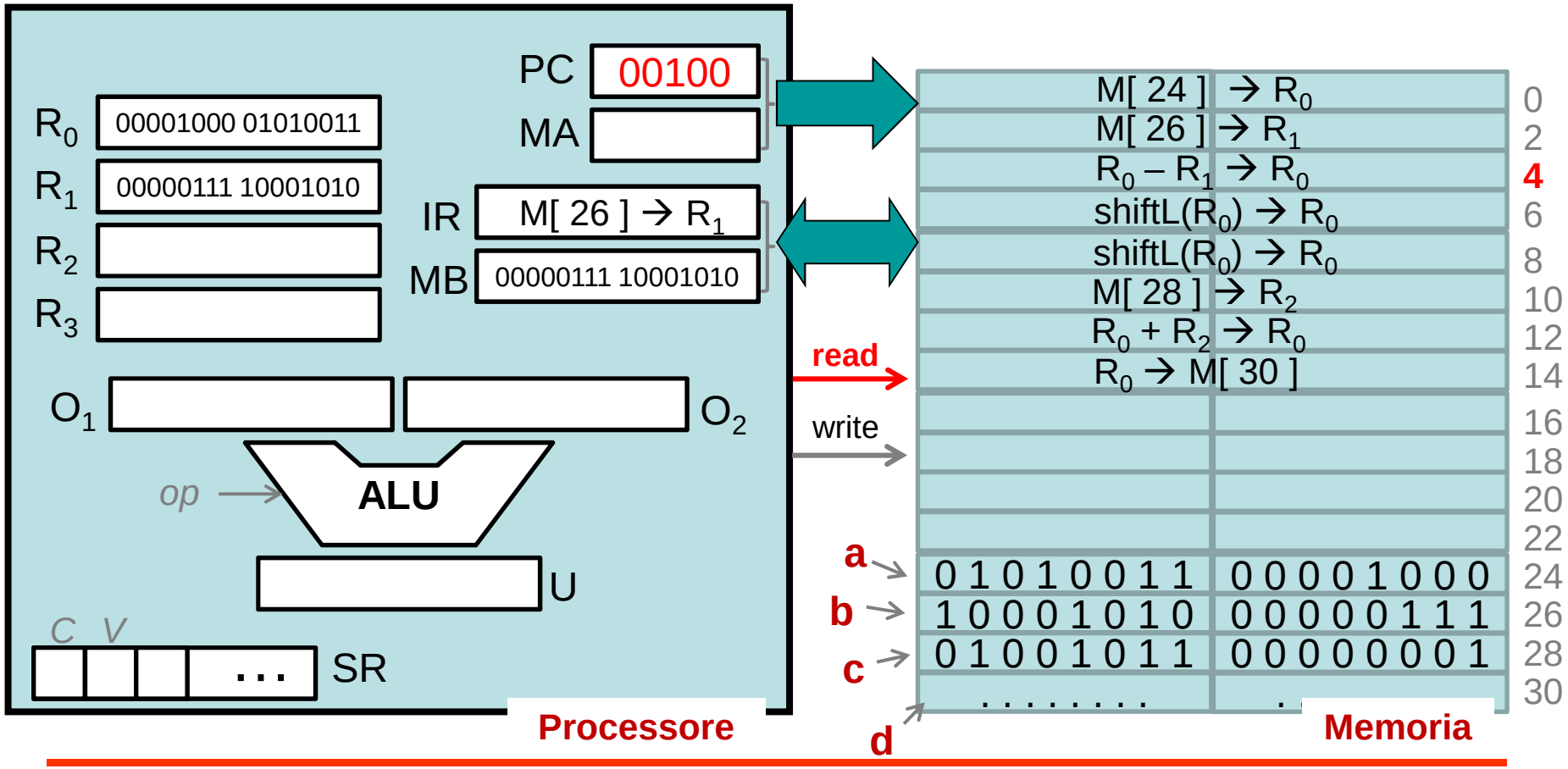
Infine, il valore è spostato da **MB** al registro destinazione indicato dall'istruzione, ovvero **R_1**. L'esecuzione della seconda istruzione è completata!



Esecuzione passo-passo

Terza istruzione

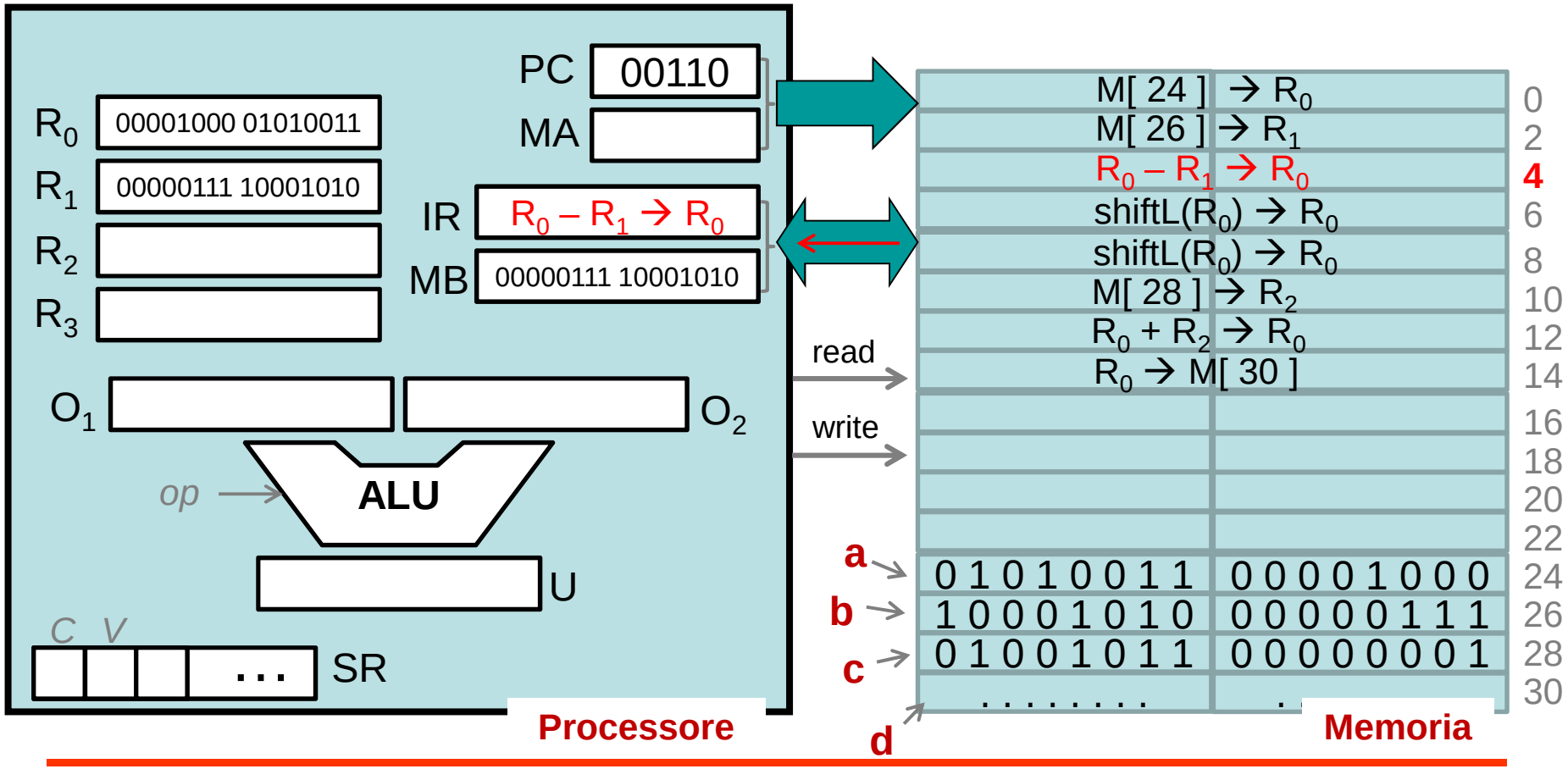
Nel frattempo **PC** è stato incrementato di 2: ora contiene $(\mathbf{00100})_2$=**4**. In questo modo, il processo di **fetch** si può ripetere identico con la terza istruzione!



Esecuzione passo-passo

Terza istruzione

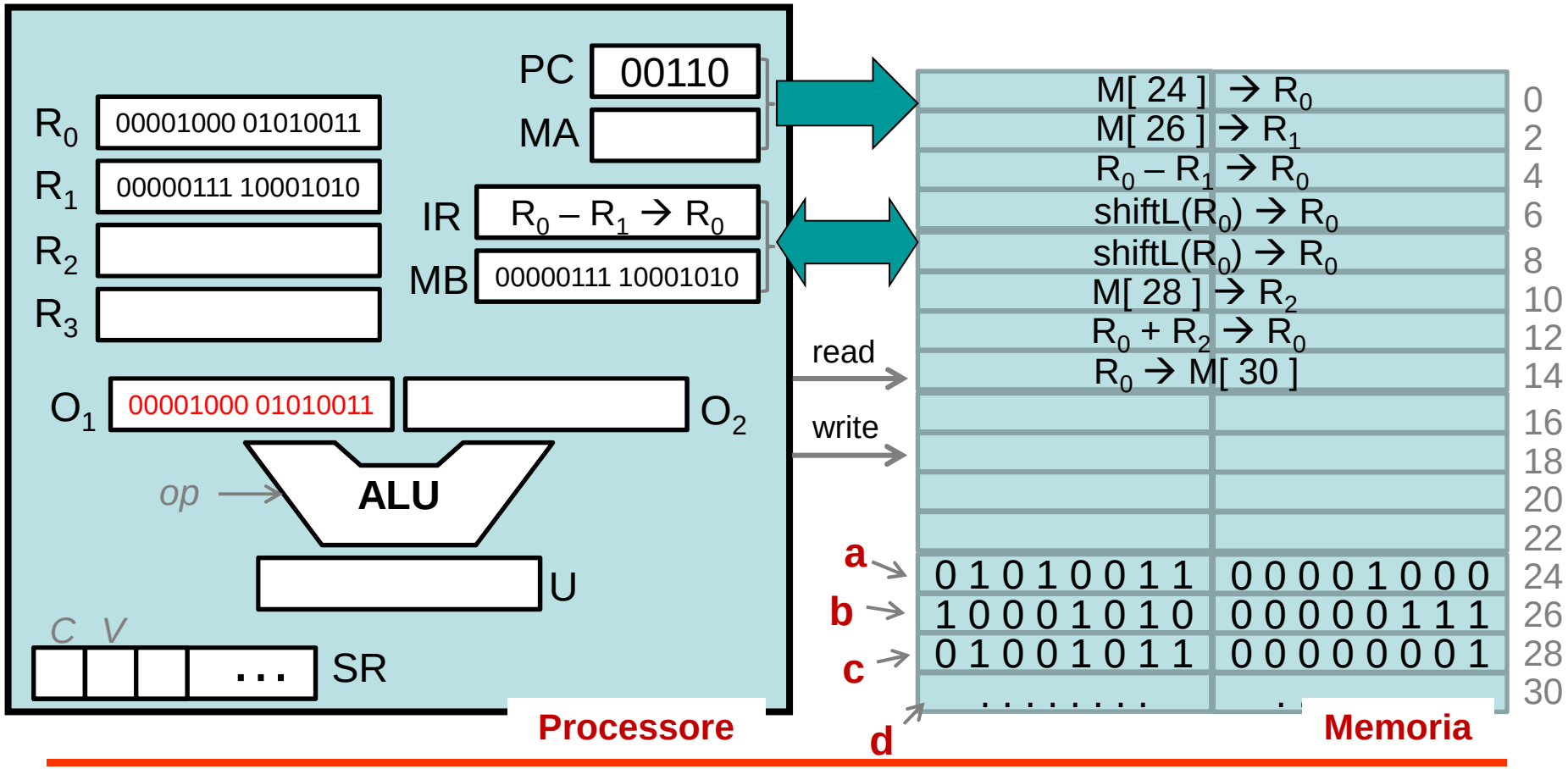
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui viene chiesta una sottrazione tra due registri



Esecuzione passo-passo

Terza istruzione

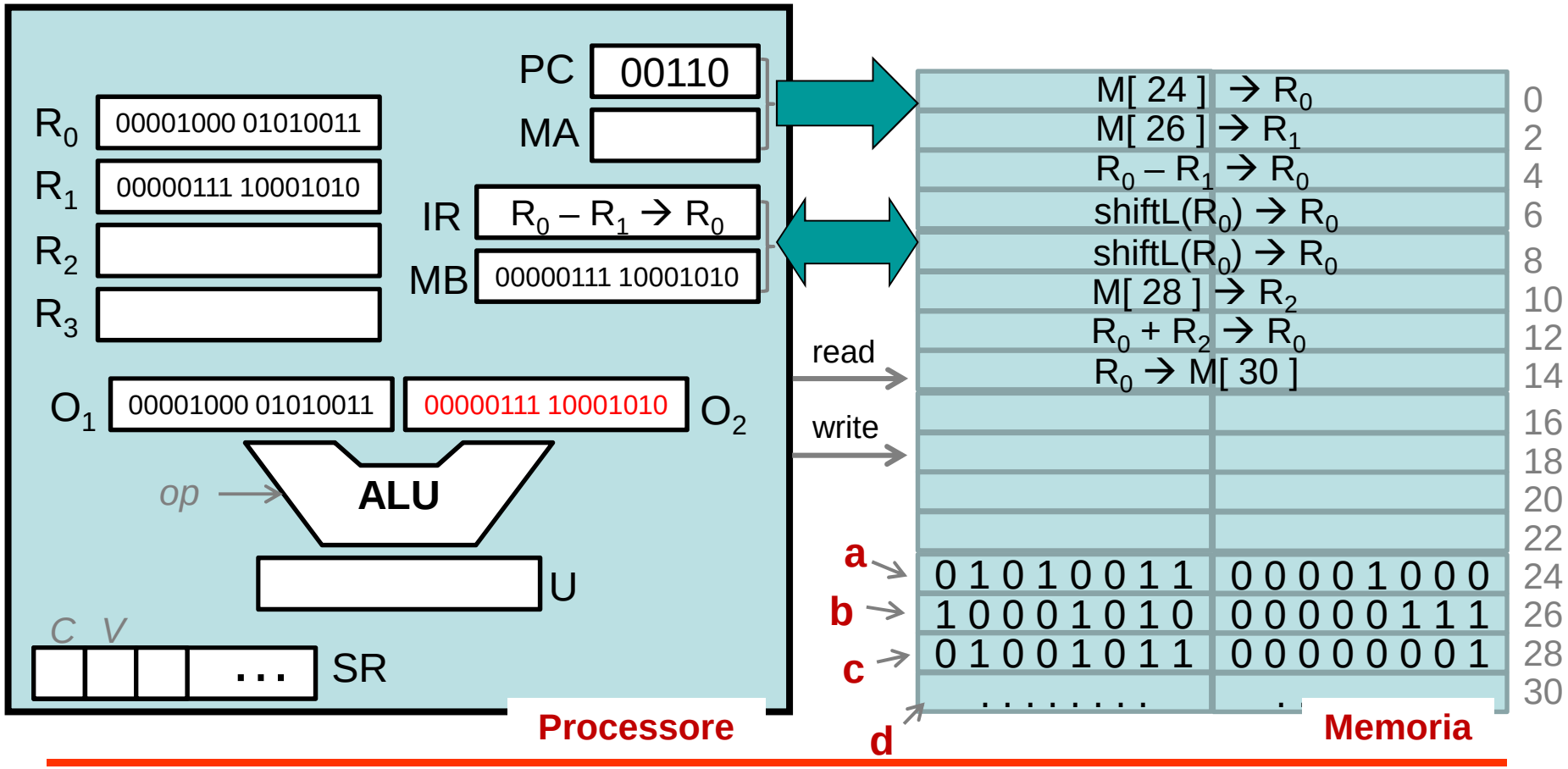
Per eseguire la sottrazione usando l'**ALU**, il primo passo è spostare gli operandi nei suoi registri di ingresso. Viene quindi copiato il valore di $\mathbf{R_0}$ in $\mathbf{O_1}$



Esecuzione passo-passo

Terza istruzione

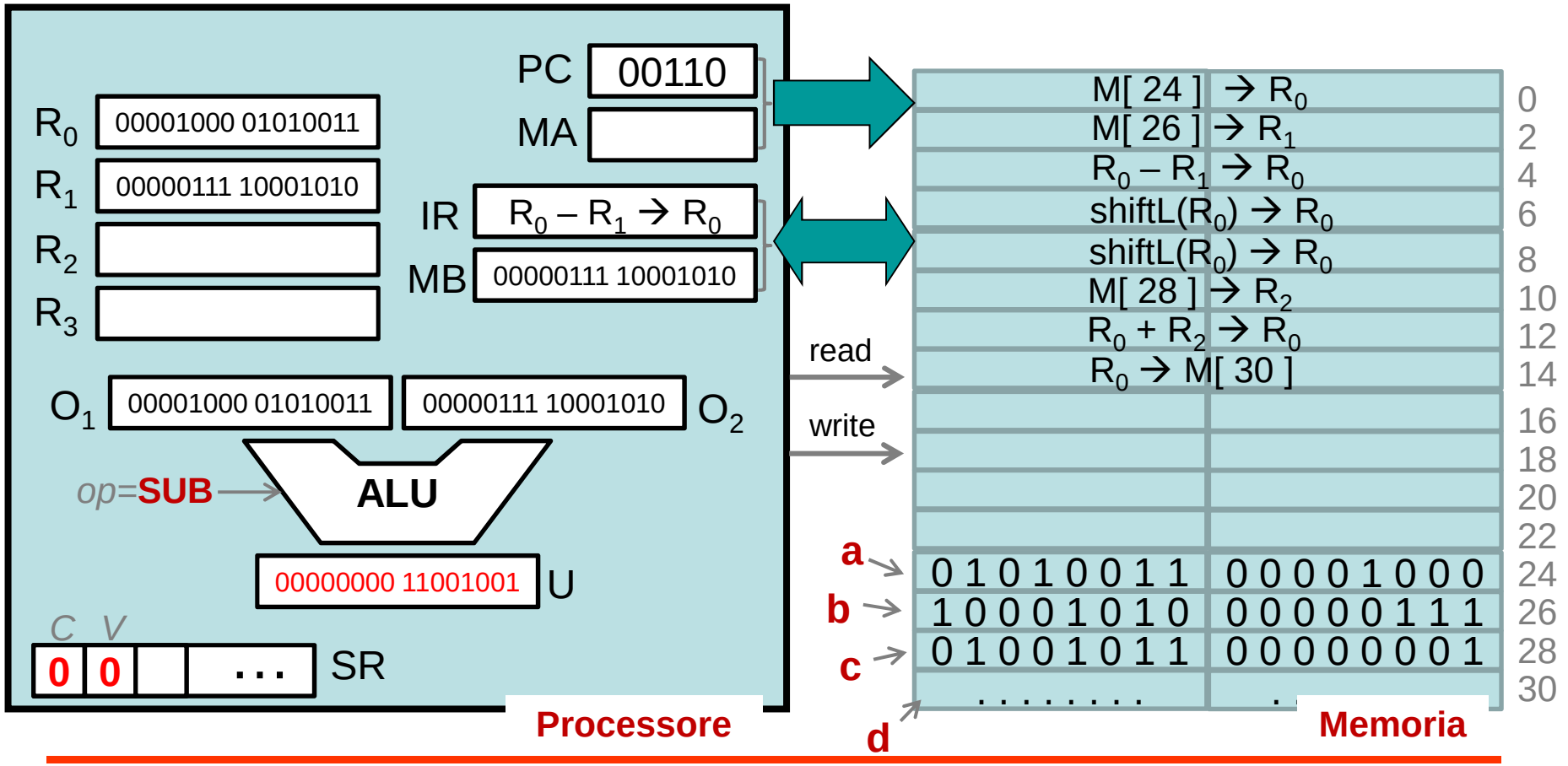
Quindi viene copiato il valore di $\mathbf{R_1}$ in $\mathbf{O_2}$



Esecuzione passo-passo

Terza istruzione

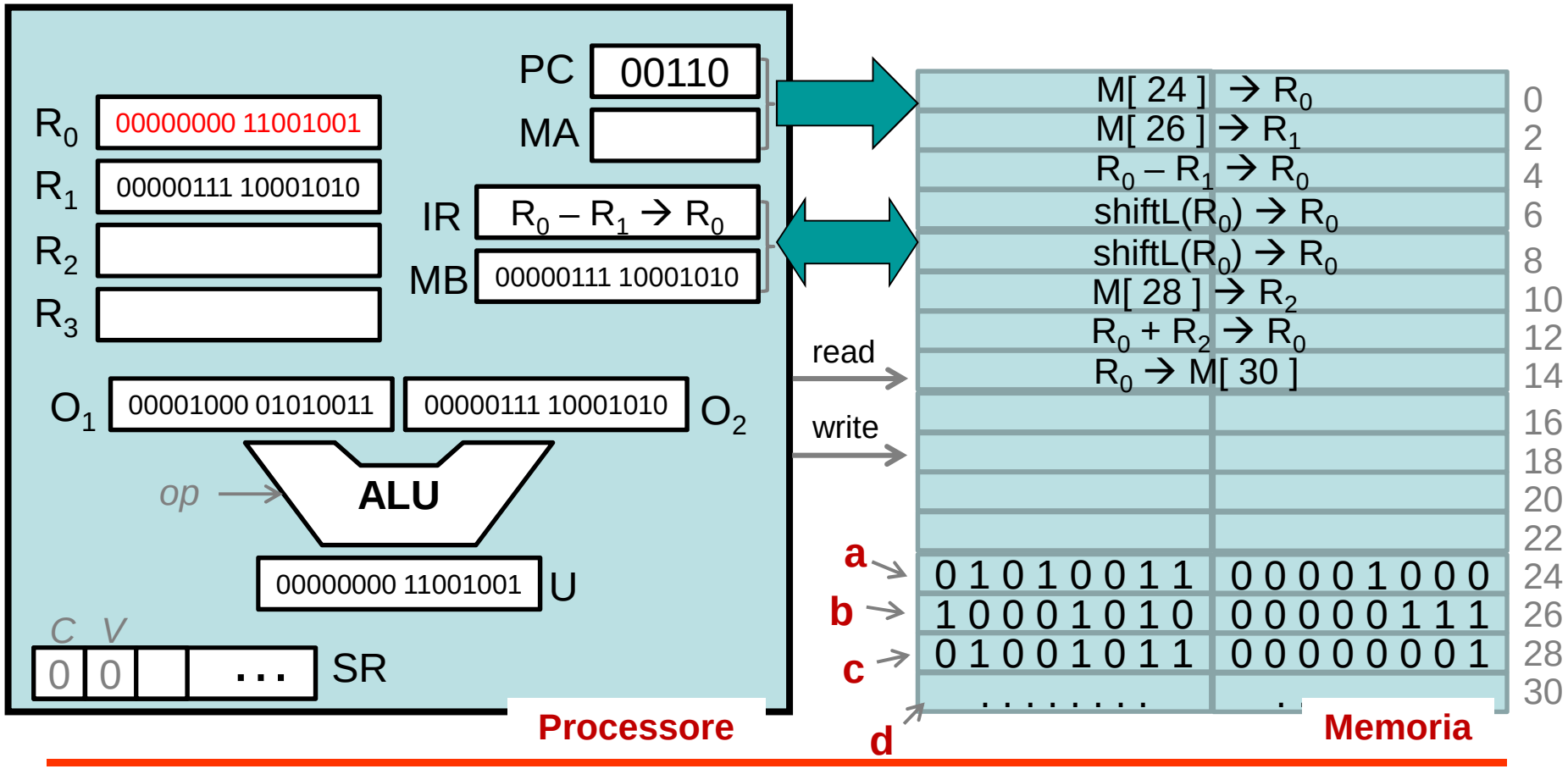
Viene poi comandata all'**ALU** l'operazione di sottrazione attraverso il segnale *op*.
Il risultato è scritto nel registro di uscita **U**. Non si produce né riporto né overflow.



Esecuzione passo-passo

Terza istruzione

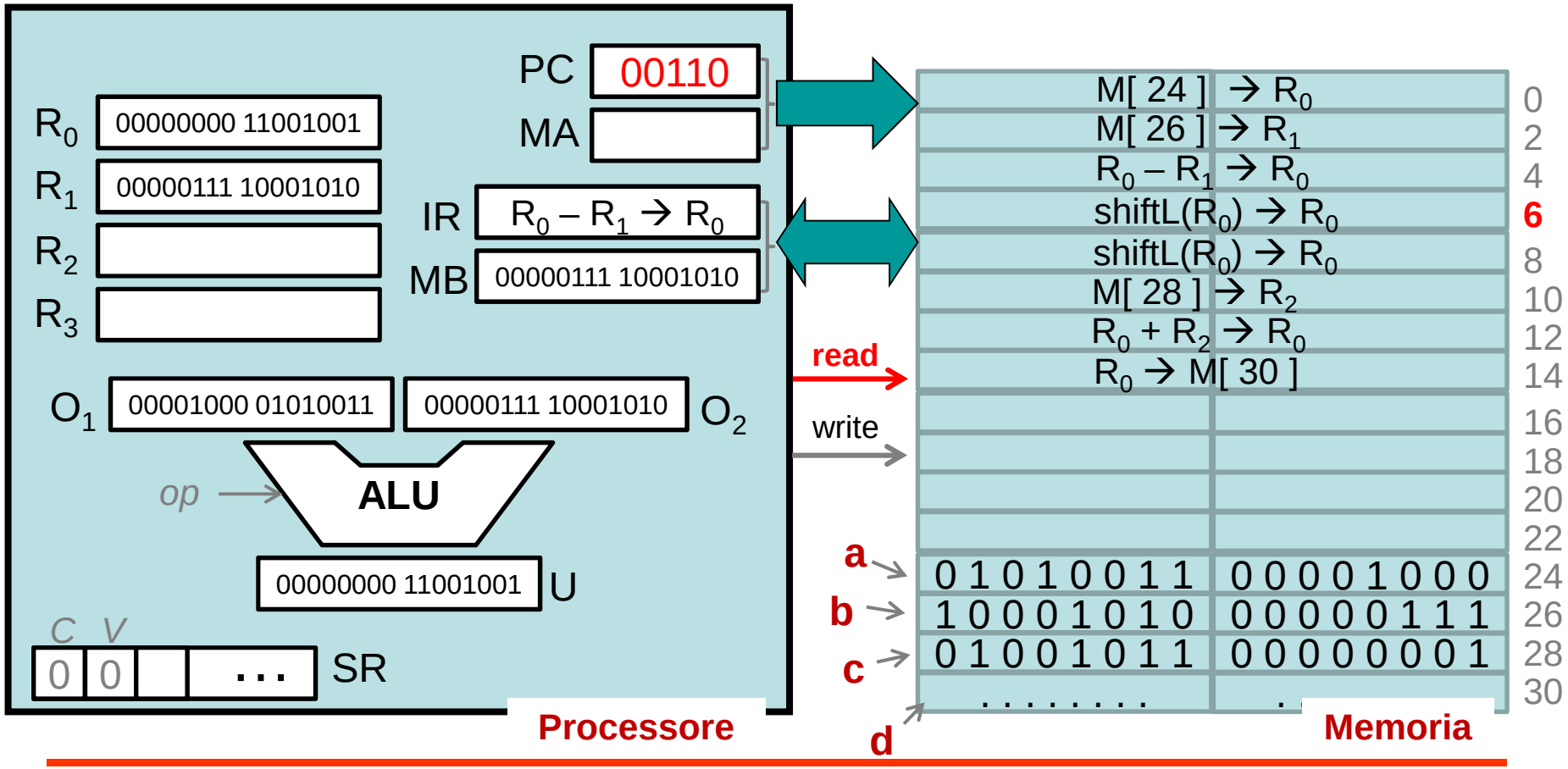
Il risultato è infine copiato dall'uscita dell'**ALU** al registro $\mathbf{R_0}$ (il contenuto precedente viene perso). Questo completa la terza istruzione.



Esecuzione passo-passo

Quarta istruzione

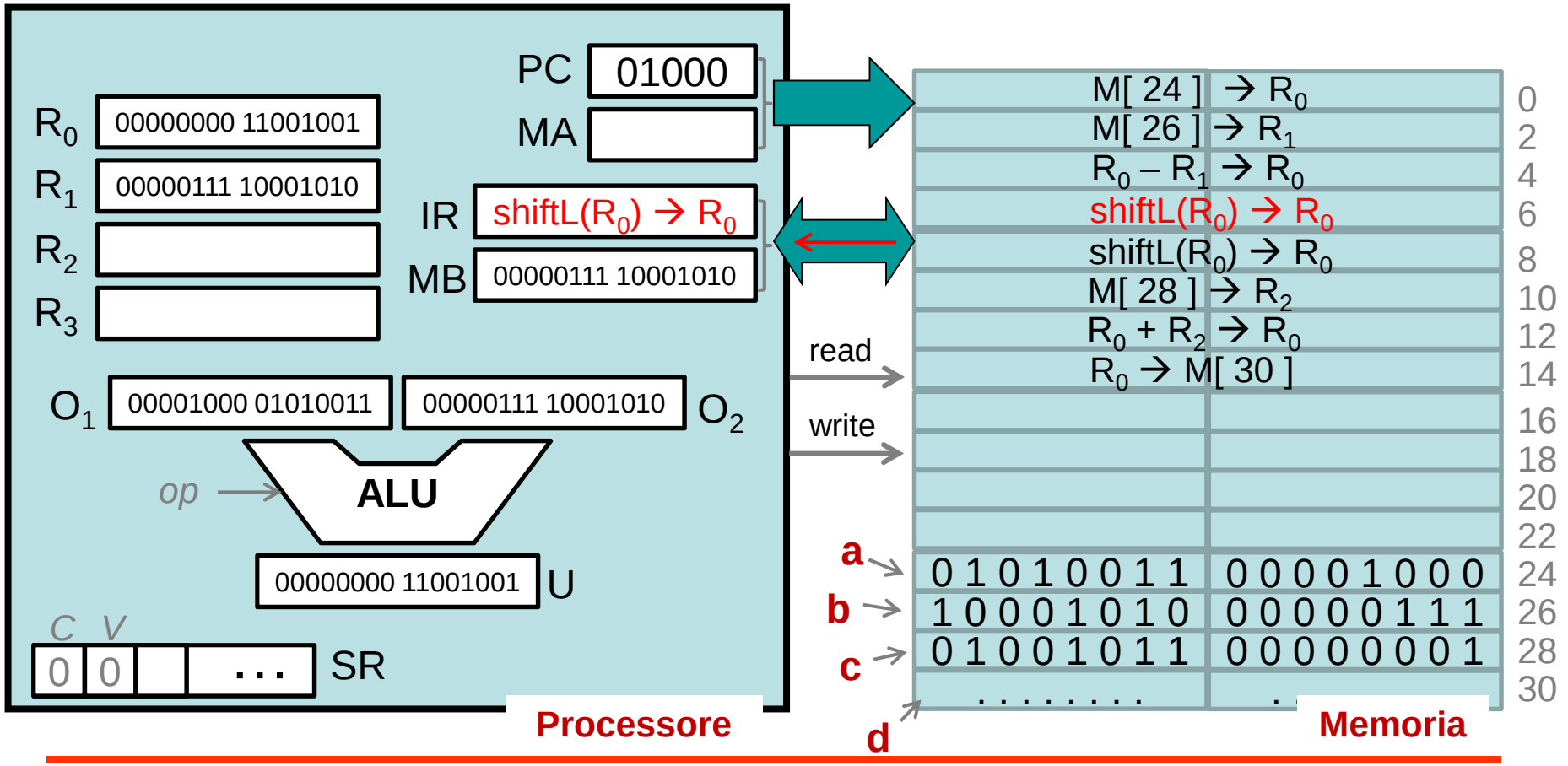
PC nel frattempo è stato incrementato di 2 ed ora vale $(\mathbf{00110})_2$=**6**. Questo permette di riavviare il *fetch*, questa volta con la quarta istruzione.



Esecuzione passo-passo

Quarta istruzione

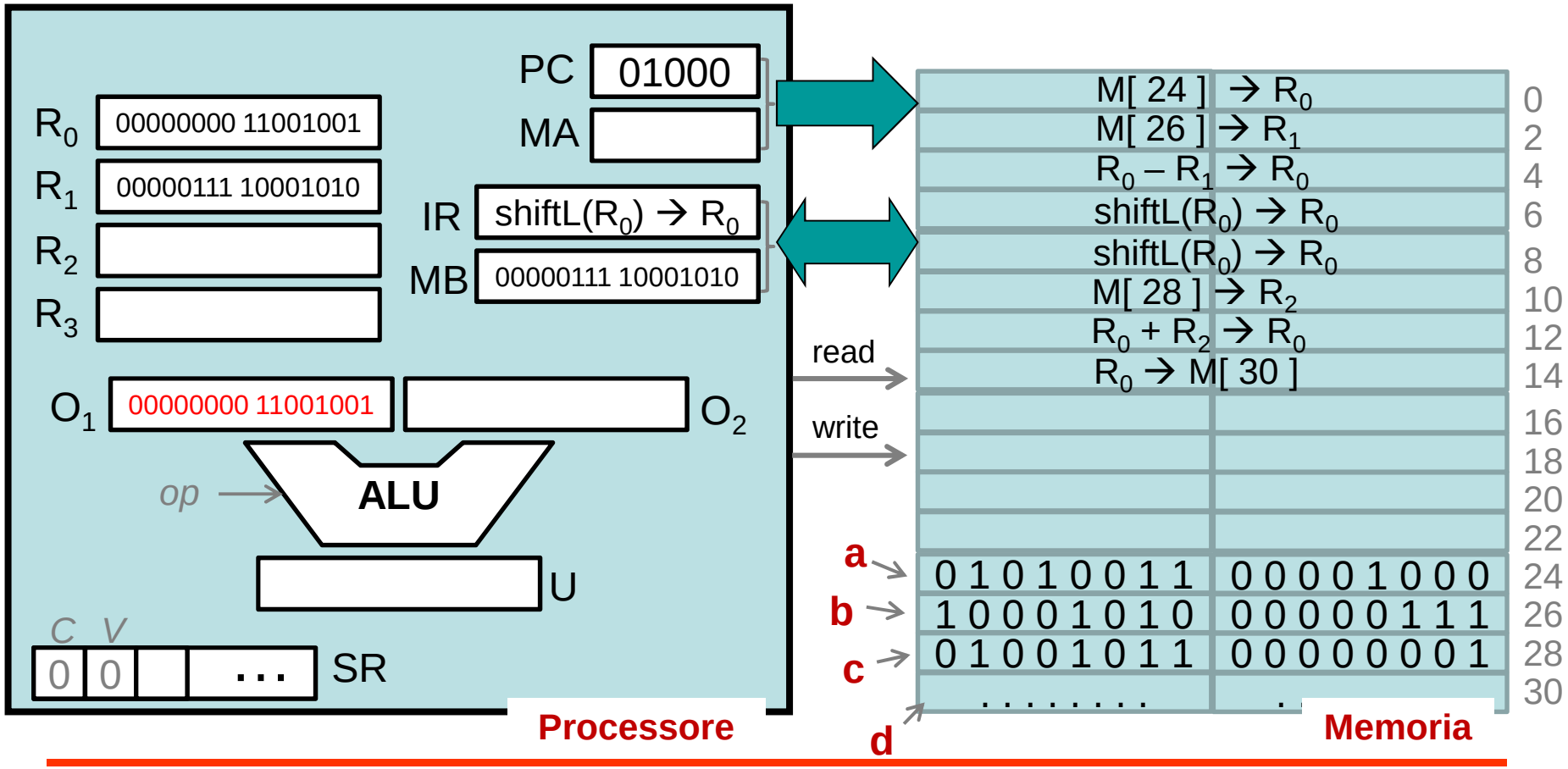
All'interno di **IR** viene caricata la prossima istruzione. Il processore la analizza e si prepara ad eseguirla. In questo caso si tratta di uno shift.



Esecuzione passo-passo

Quarta istruzione

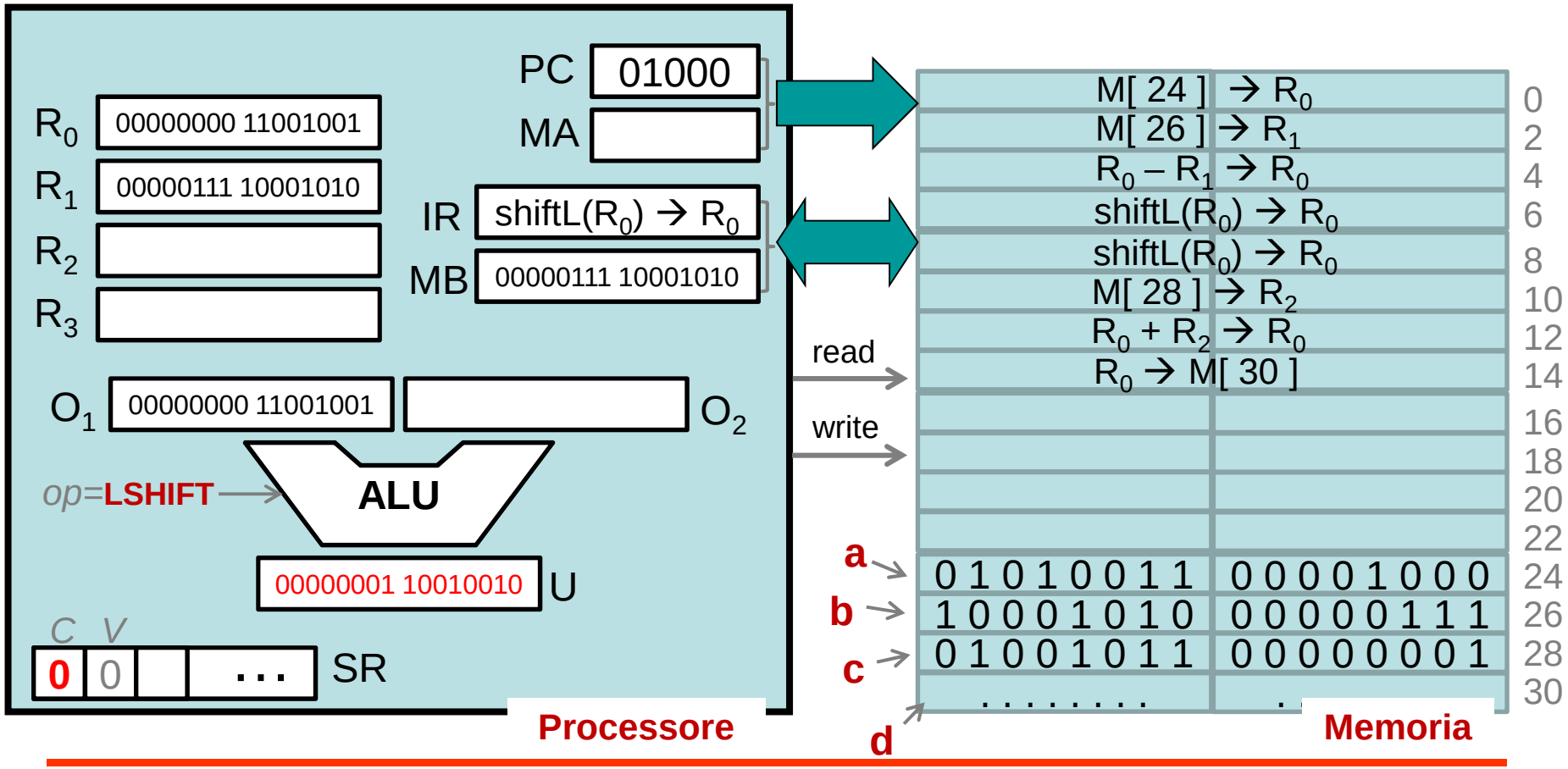
Il registro interessato dallo shift, ovvero **R_0**, viene copiato in uno dei registri di ingresso dell'**ALU**



Esecuzione passo-passo

Quarta istruzione

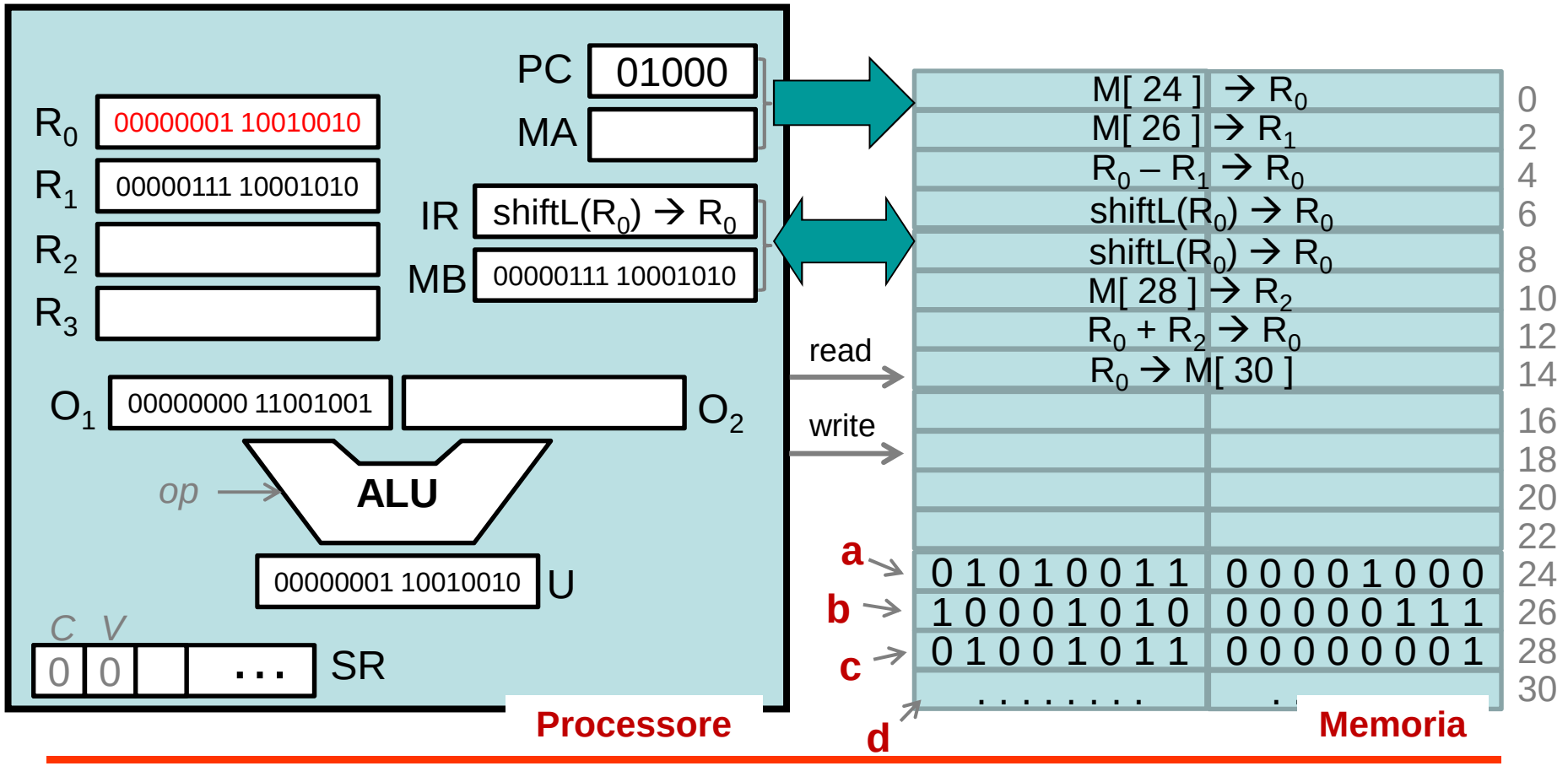
Viene comandata all'ALU un'operazione di shift a sinistra sul suo operando sinistro **O_1**. Il risultato è scritto nel registro di uscita **U**. Non si è generato riporto.



Esecuzione passo-passo

Quarta istruzione

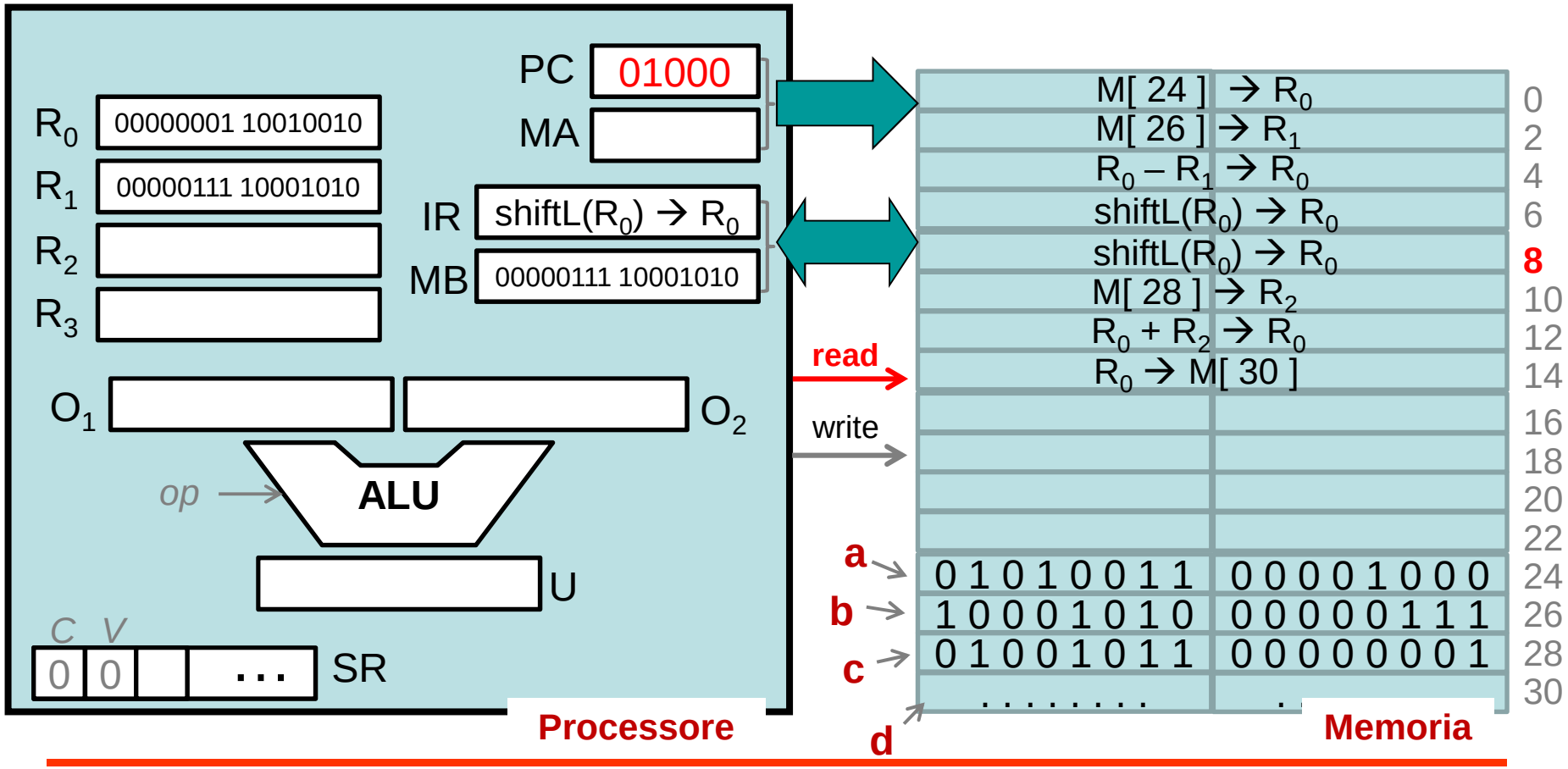
Infine, l'uscita dell'ALU è copiata nel registro **R_0**, come richiesto dall'istruzione (il contenuto precedente viene perso). Questo completa la quarta istruzione.



Esecuzione passo-passo

Quinta istruzione

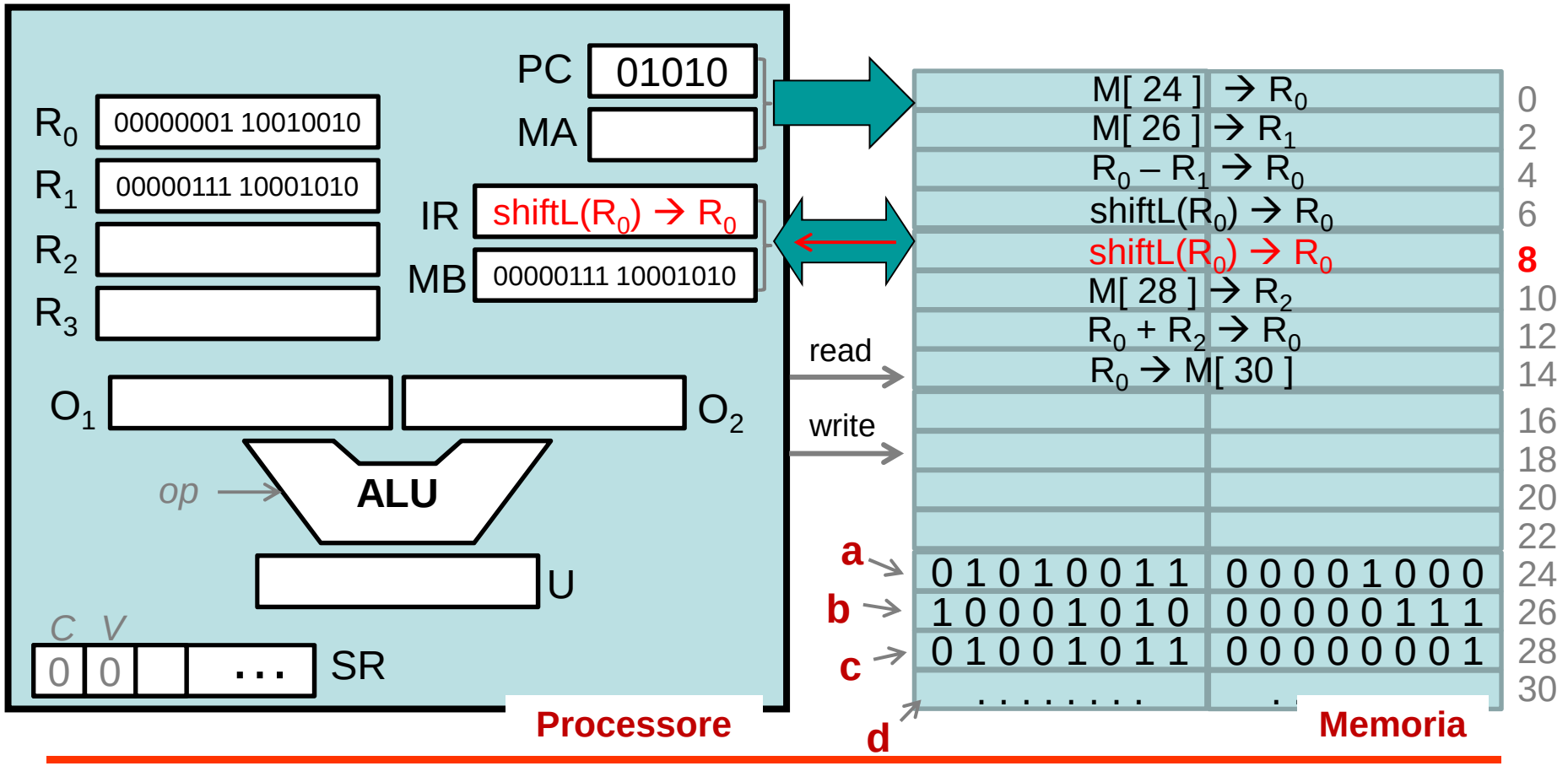
PC nel frattempo è stato incrementato di 2 ed ora vale $(\mathbf{01000})_2$=**8**. Questo permette di riavviare il *fetch*, questa volta con la quinta istruzione.



Esecuzione passo-passo

Quinta istruzione

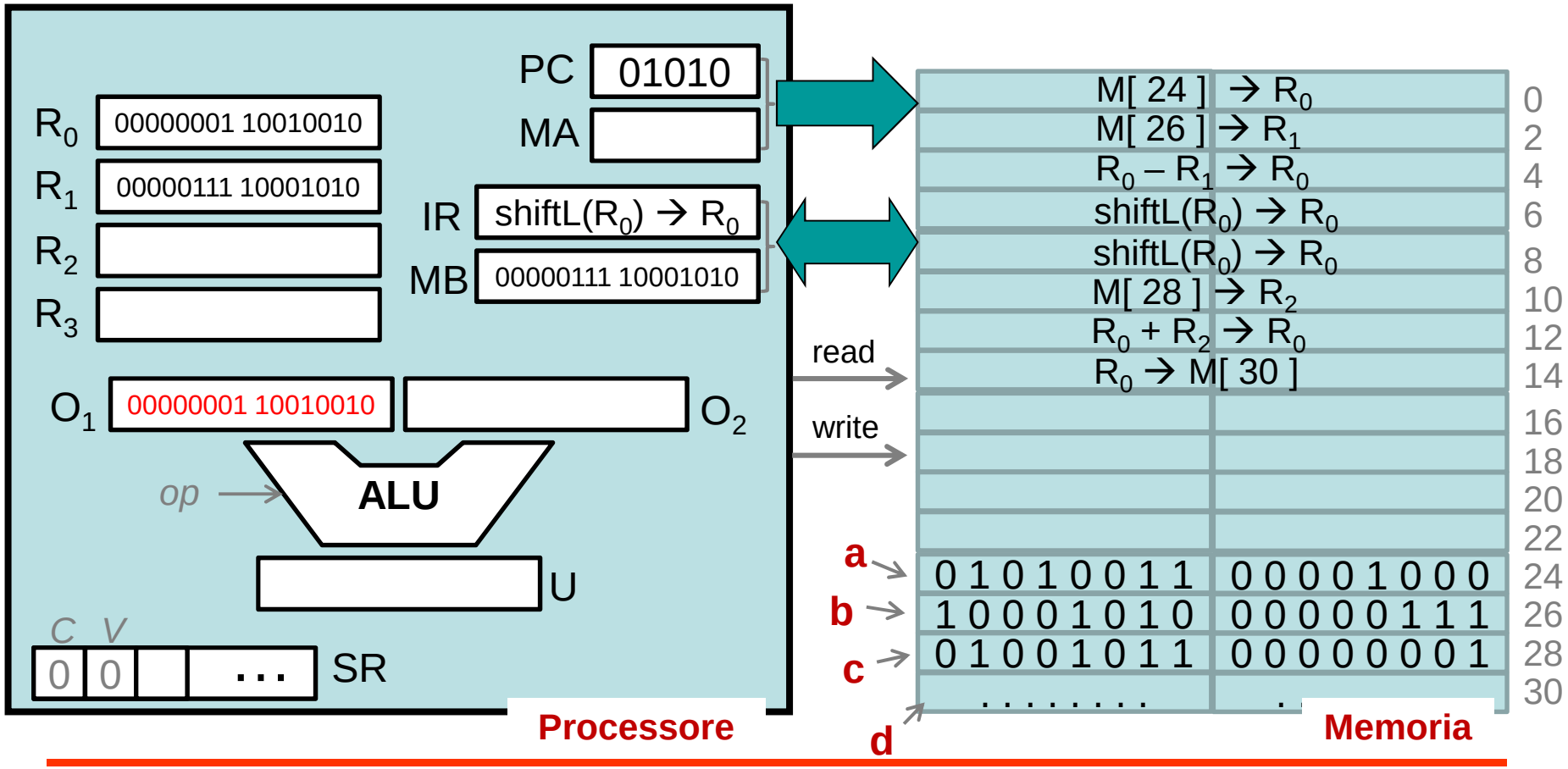
All'interno di **IR** viene caricata la prossima istruzione. Il processore la analizza e si prepara ad eseguirla. In questo caso si tratta di uno shift.



Esecuzione passo-passo

Quinta istruzione

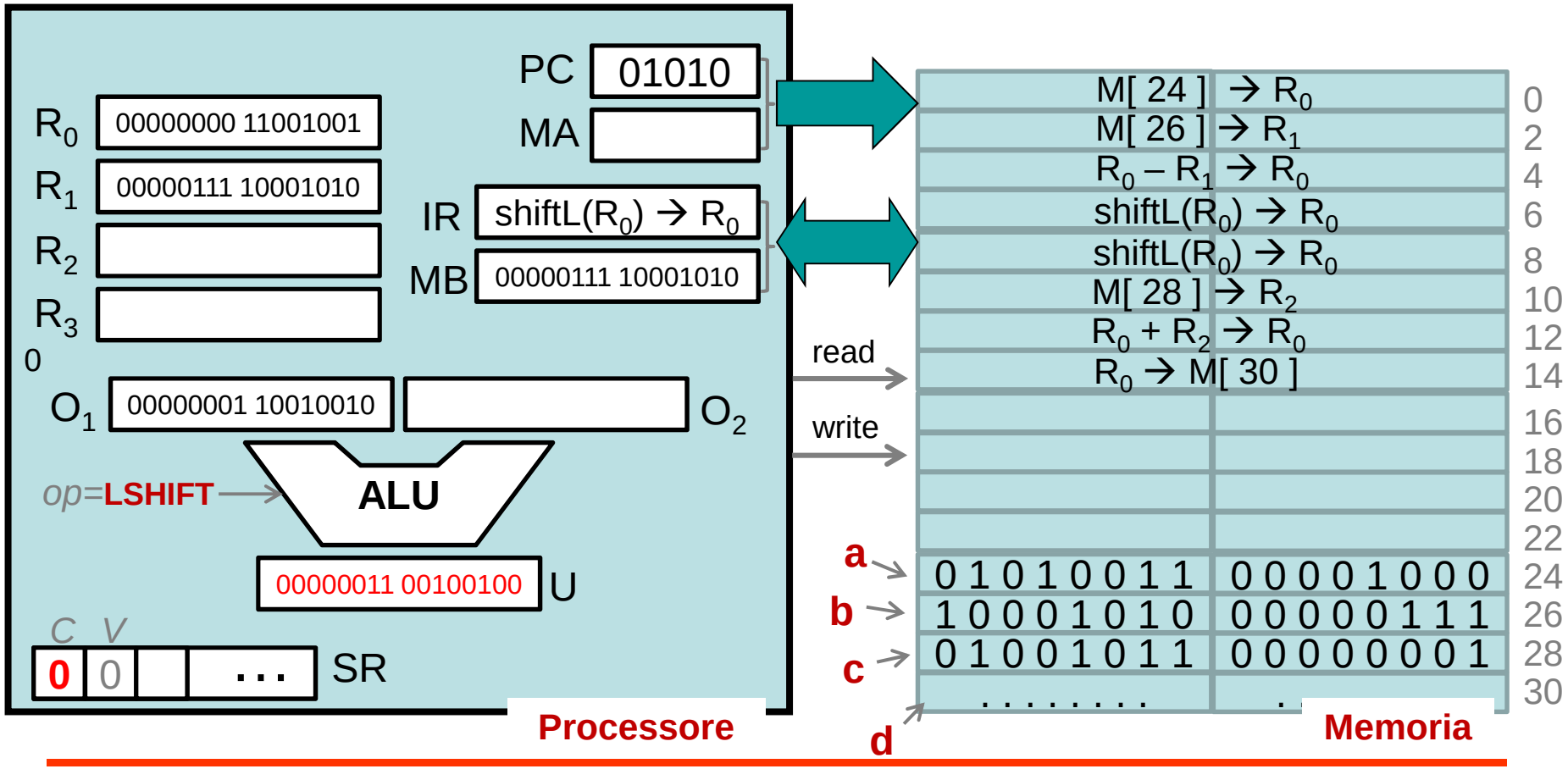
Il registro interessato dallo shift, ovvero **R_0**, viene copiato in uno dei registri di ingresso dell'**ALU**



Esecuzione passo-passo

Quinta istruzione

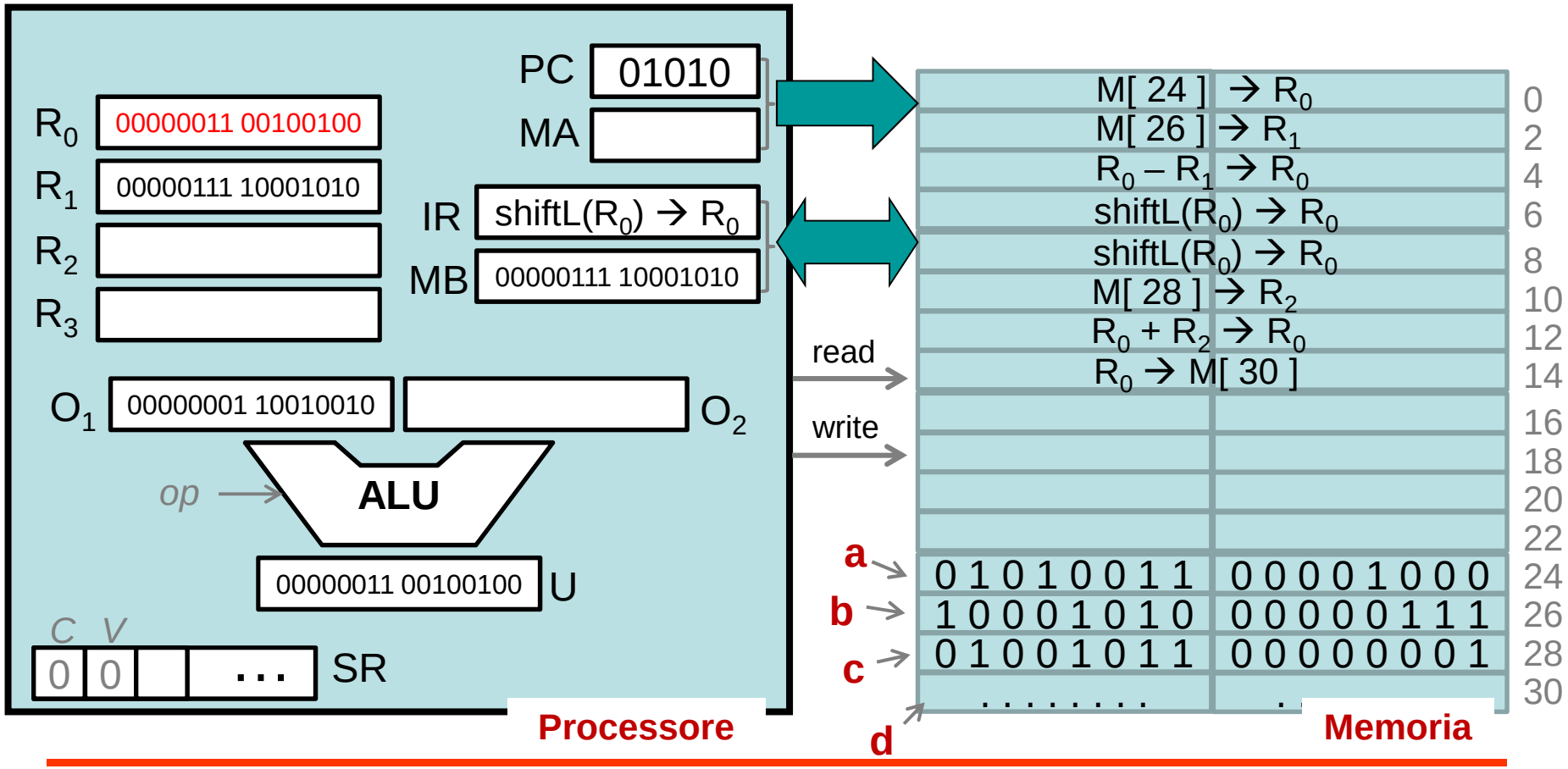
Viene comandata all'ALU un'operazione di shift a sinistra sul suo operando sinistro **O_1**. Il risultato è scritto nel registro di uscita **U**. Non si è generato riporto.



Esecuzione passo-passo

Quinta istruzione

Infine, l'uscita dell'ALU è copiata nel registro **R_0**, come richiesto dall'istruzione (il contenuto precedente viene perso). Questo completa la quinta istruzione.



Esecuzione passo-passo

- Le istruzioni eseguite finora ci hanno permesso di calcolare il valore 4*(**a** - **b**)
- Infatti, ricordando che

  a = (2131)$_{10}$ = (1000 01010011)$_2$

  b = (1930)$_{10}$ = (111 10001010)$_2$
- all'interno di $\mathbf{R_0}$ troviamo adesso il valore

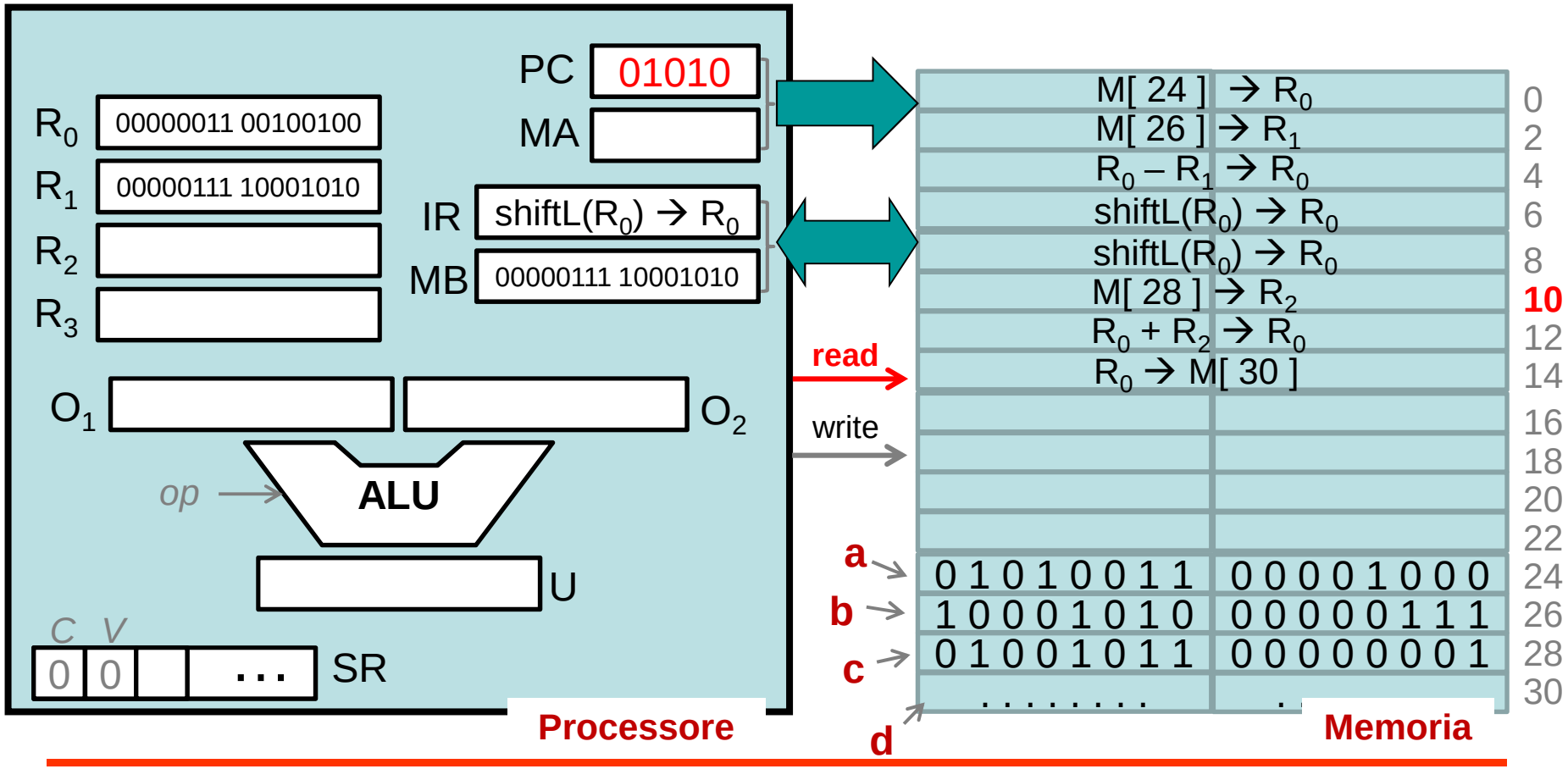
  (1100100100)$_2$ = (804)$_{10}$
- che è pari proprio a 4*(**a** - **b**)

- Andiamo avanti con le altre istruzioni...

Esecuzione passo-passo

Sesta istruzione

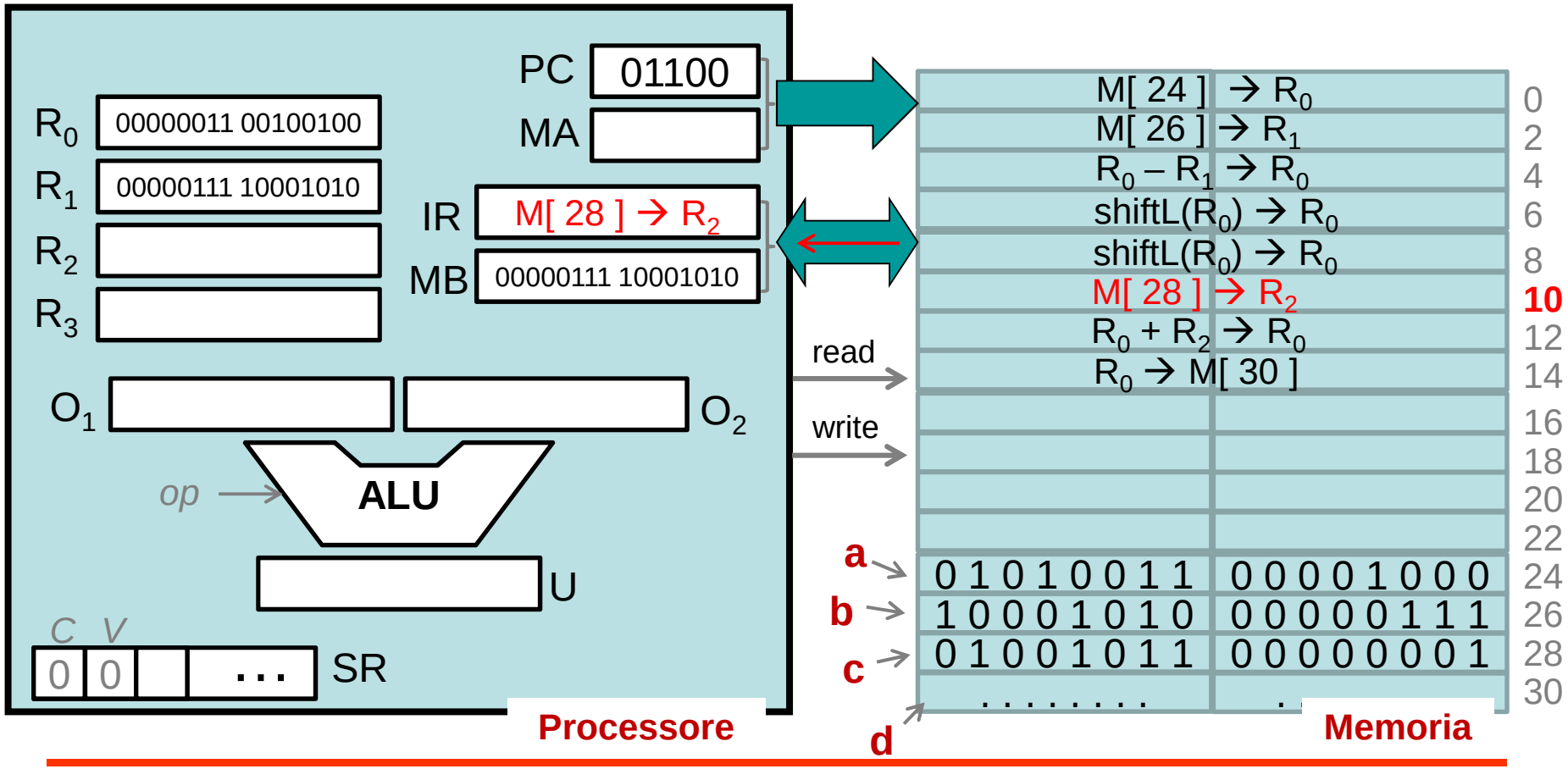
PC nel frattempo è stato incrementato di 2 e vale ora $(\mathbf{01010})_2$=**10**. Questo permette di ripetere il *fetch* con la sesta istruzione.



Esecuzione passo-passo

Sesta istruzione

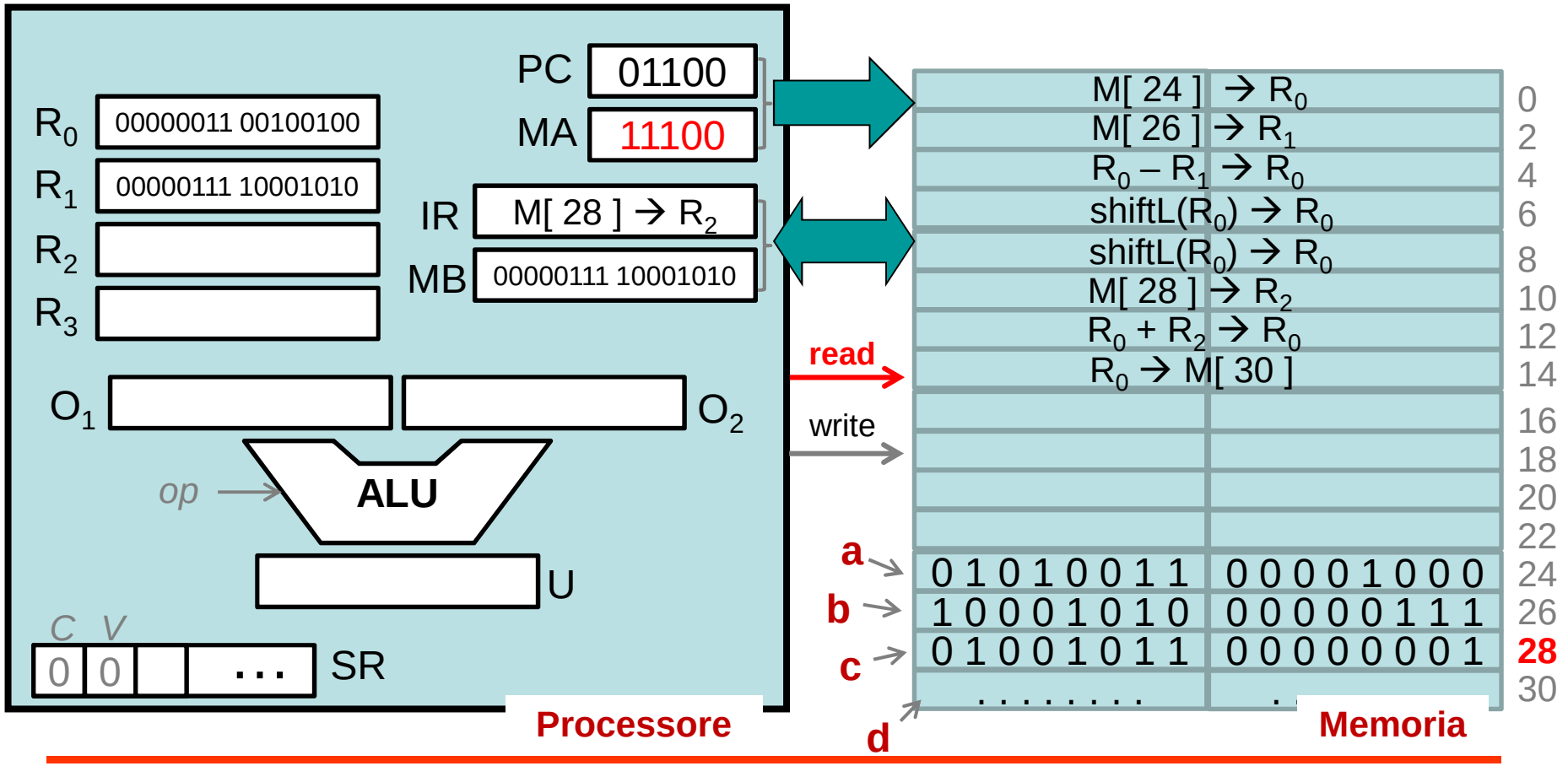
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui serve un accesso in memoria all'indirizzo **28**.



Esecuzione passo-passo

Sesta istruzione

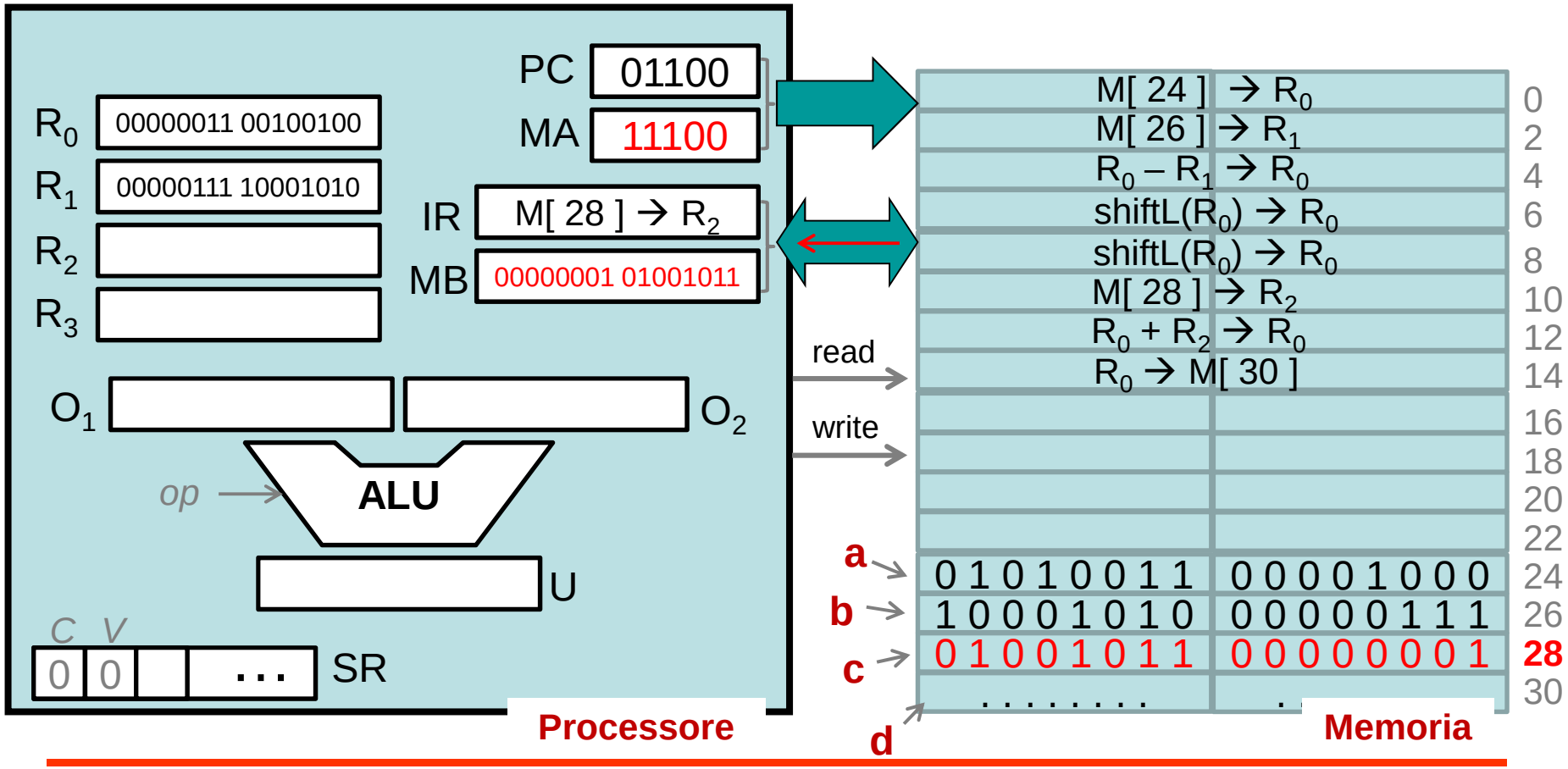
Il processore pone quindi il valore dell'indirizzo **28**=(**11100**)$_2$ all'interno del registro **MA** e comanda un'operazione di lettura (read)



Esecuzione passo-passo

Sesta istruzione

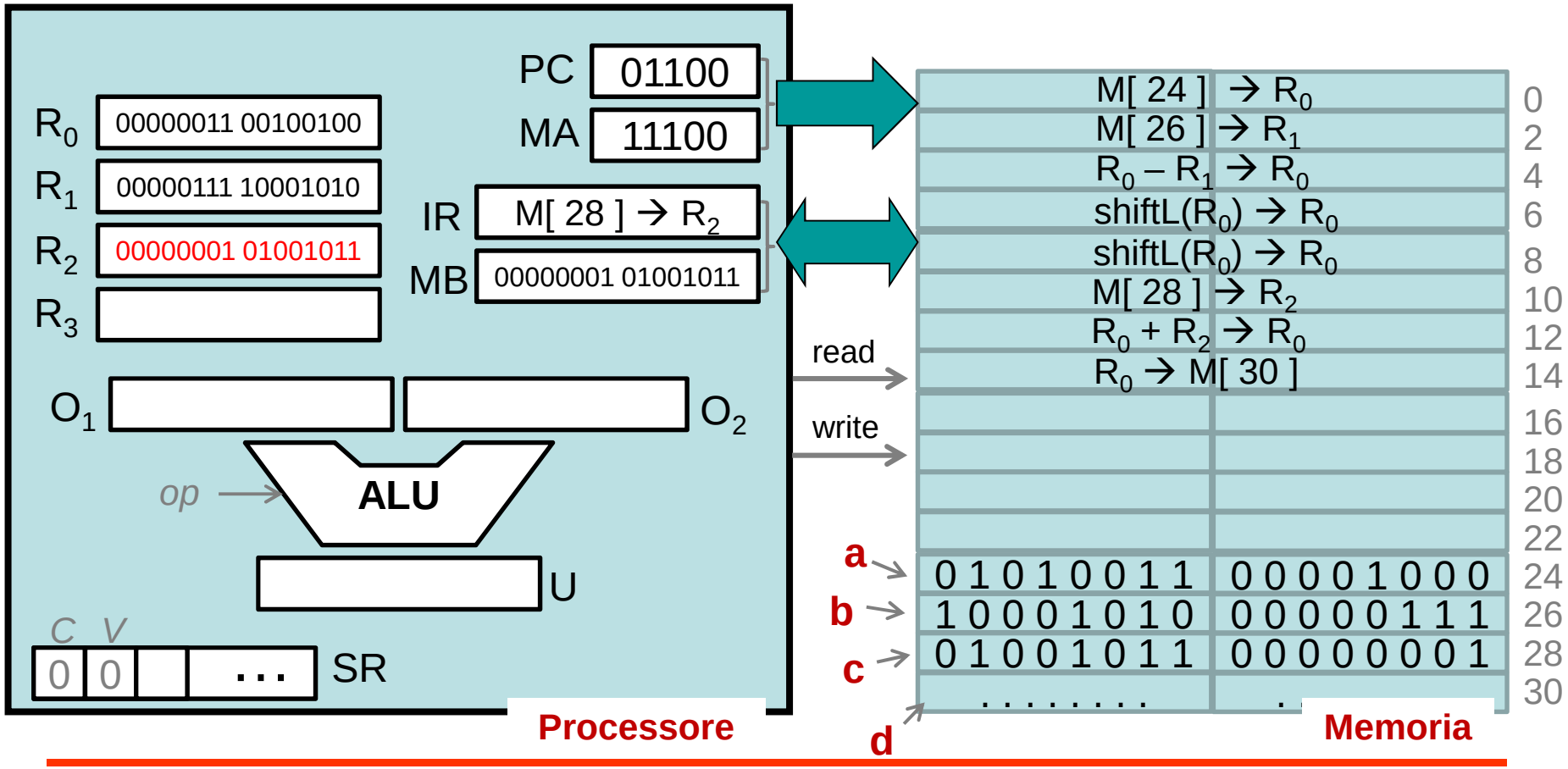
La memoria risponde fornendo all'interno di **MB** il contenuto della locazione **28** (la variabile **c**)



Esecuzione passo-passo

Sesta istruzione

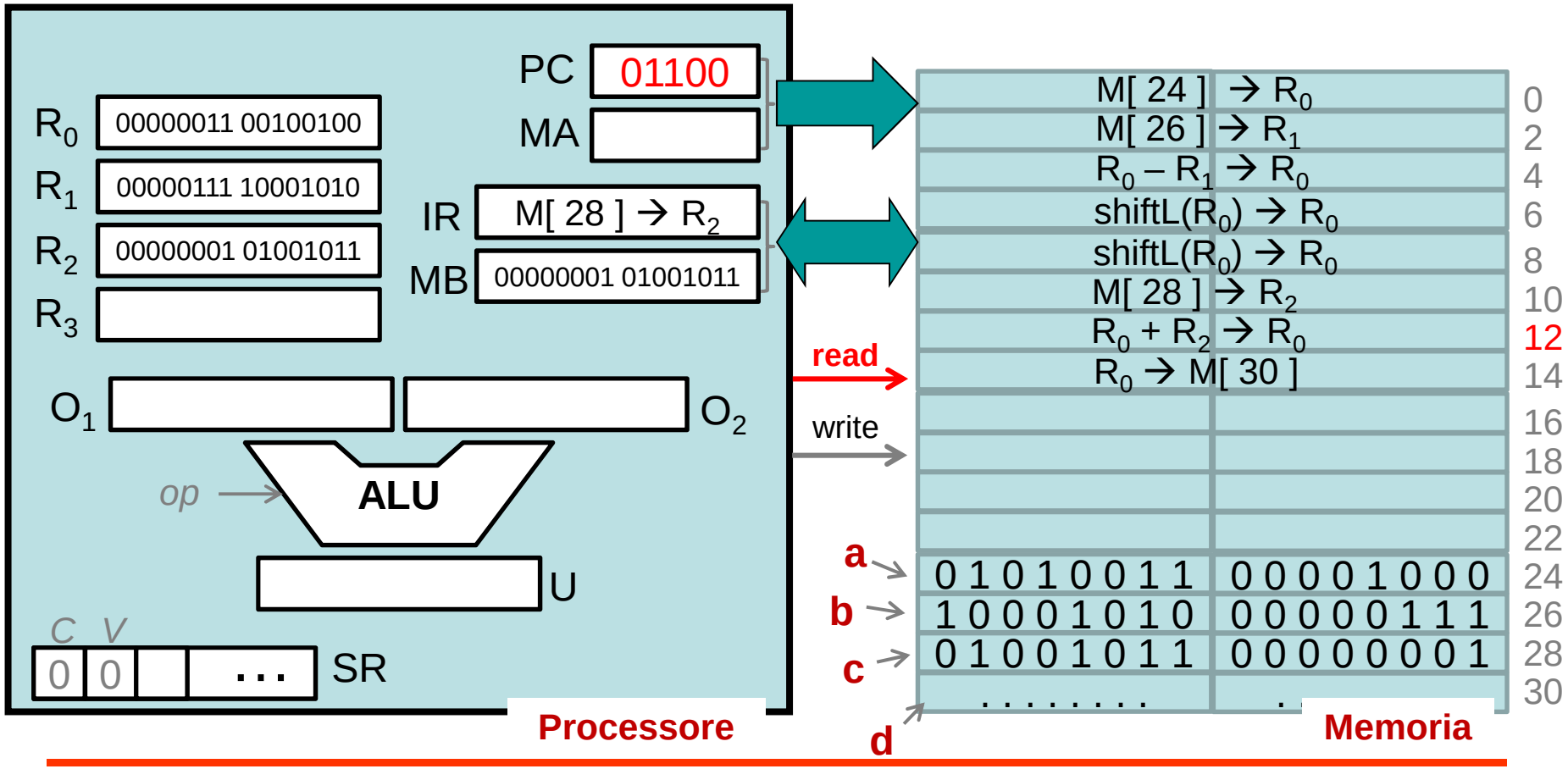
Infine, il valore viene copiato dal registro **MB** al registro indicato come destinazione dall'istruzione, ovvero **R_2**. Anche la sesta istruzione è completa.



Esecuzione passo-passo

Settima istruzione

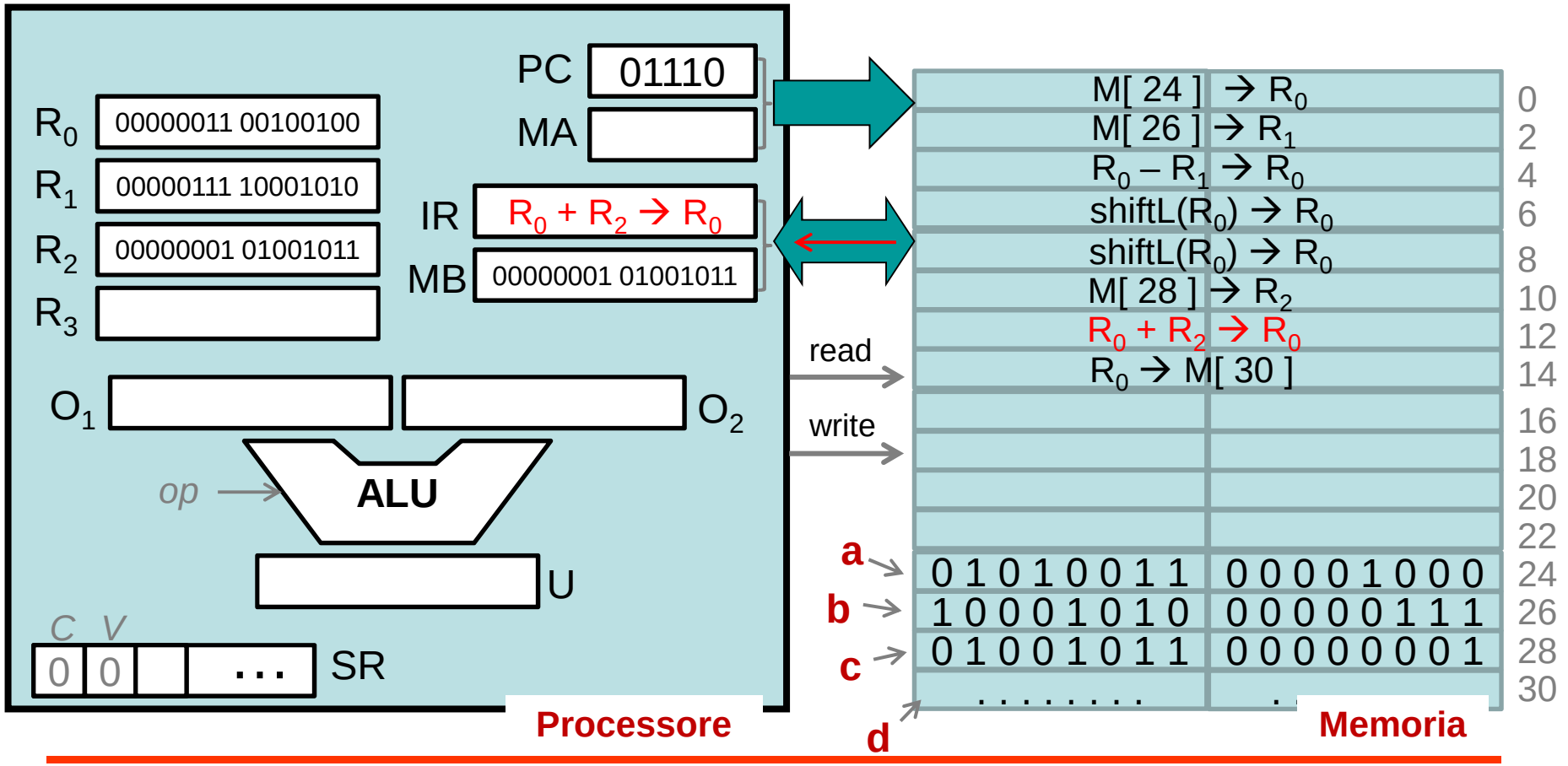
PC nel frattempo è stato incrementato di 2 e vale ora $(\mathbf{01100})_2$=**12**. Questo permette di ripetere il *fetch* con la settima istruzione.



Esecuzione passo-passo

Settima istruzione

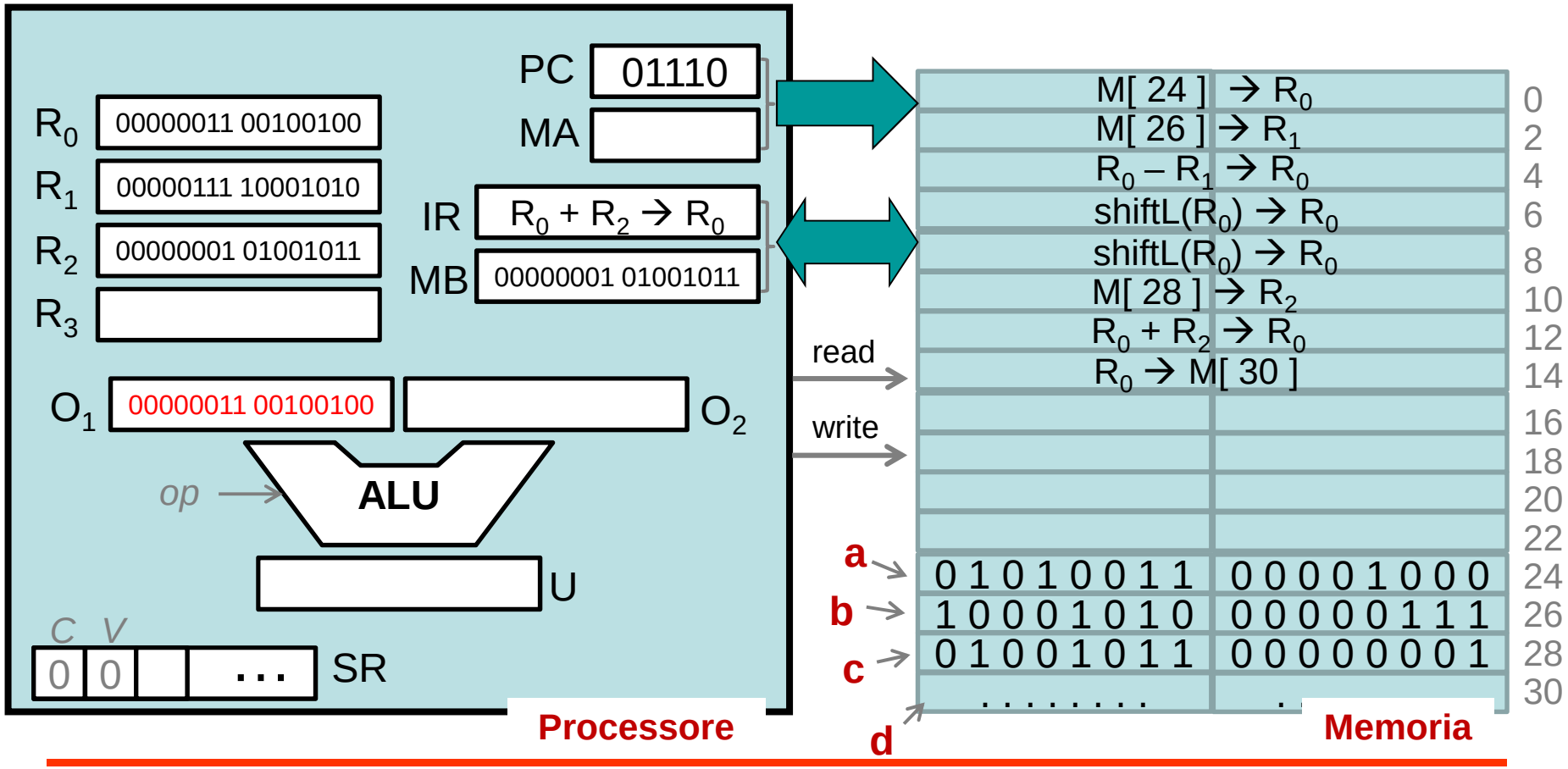
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui occorre fare un'addizione tramite la **ALU**.



Esecuzione passo-passo

Settima istruzione

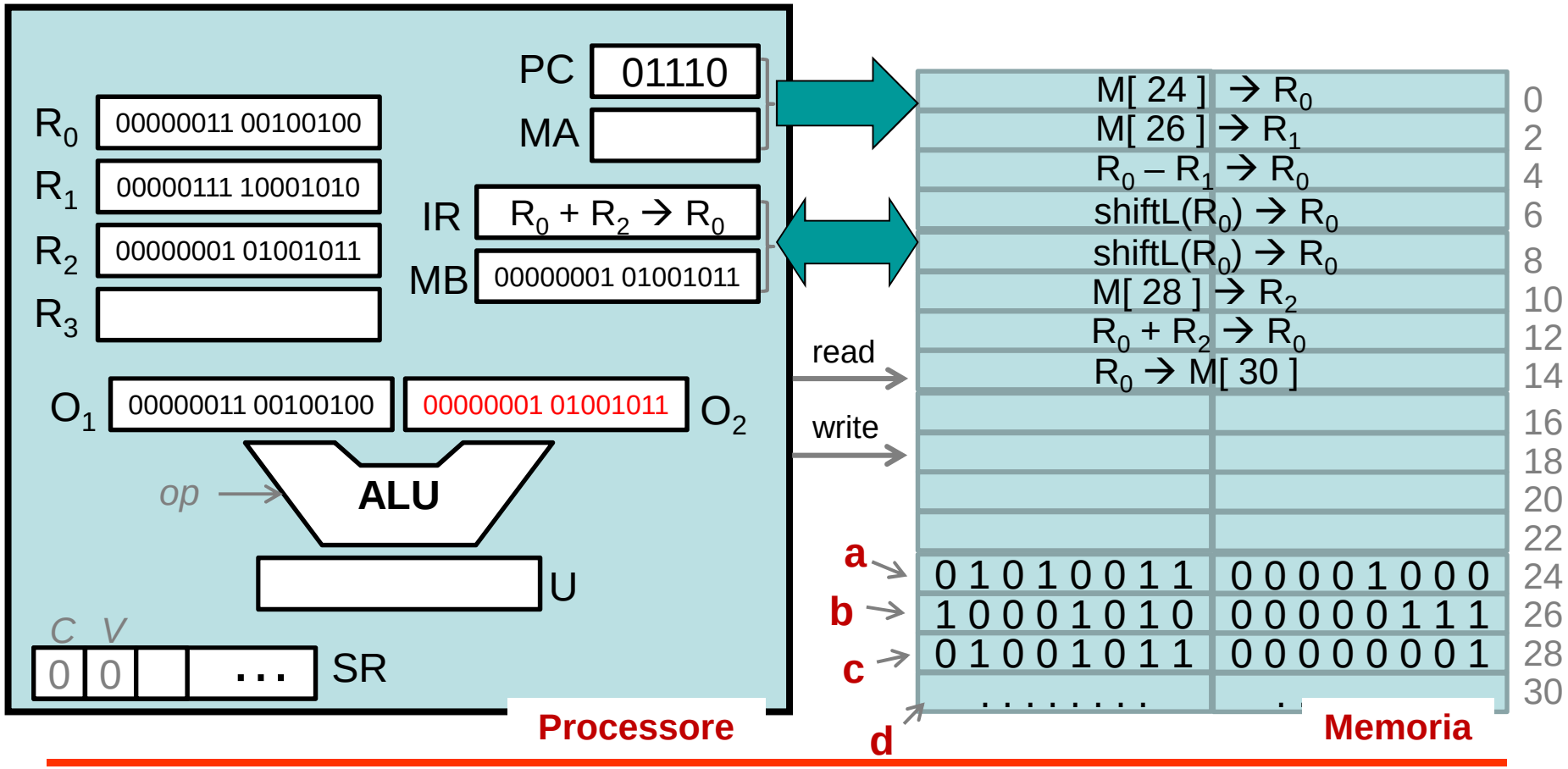
Come prima, occorre portare gli operandi nei registri di ingresso dell'ALU. Viene quindi copiato **R_0** in **O_1**.



Esecuzione passo-passo

Settima istruzione

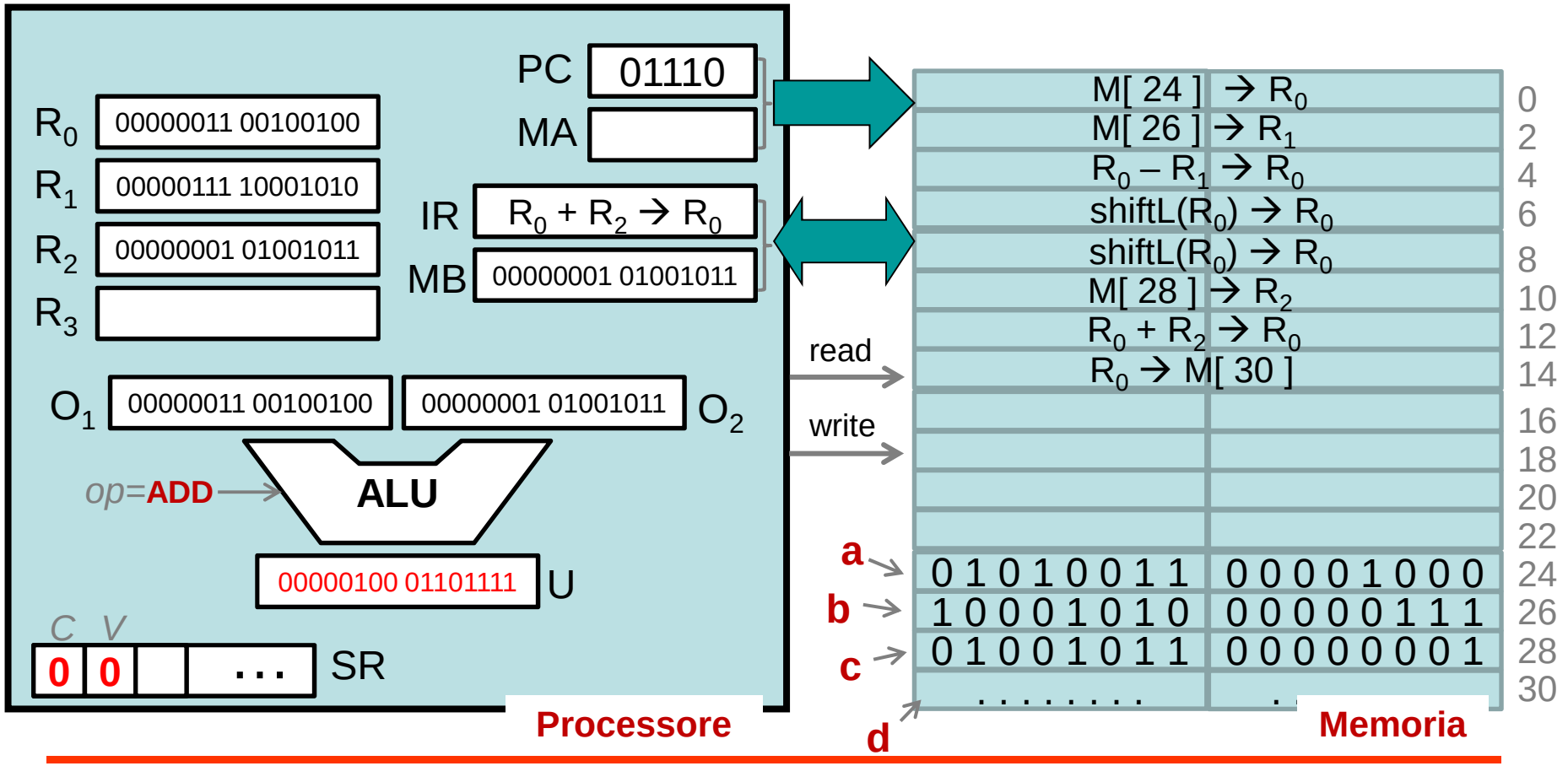
Successivamente viene copiato **R_2** in **O_2**.



Esecuzione passo-passo

Settima istruzione

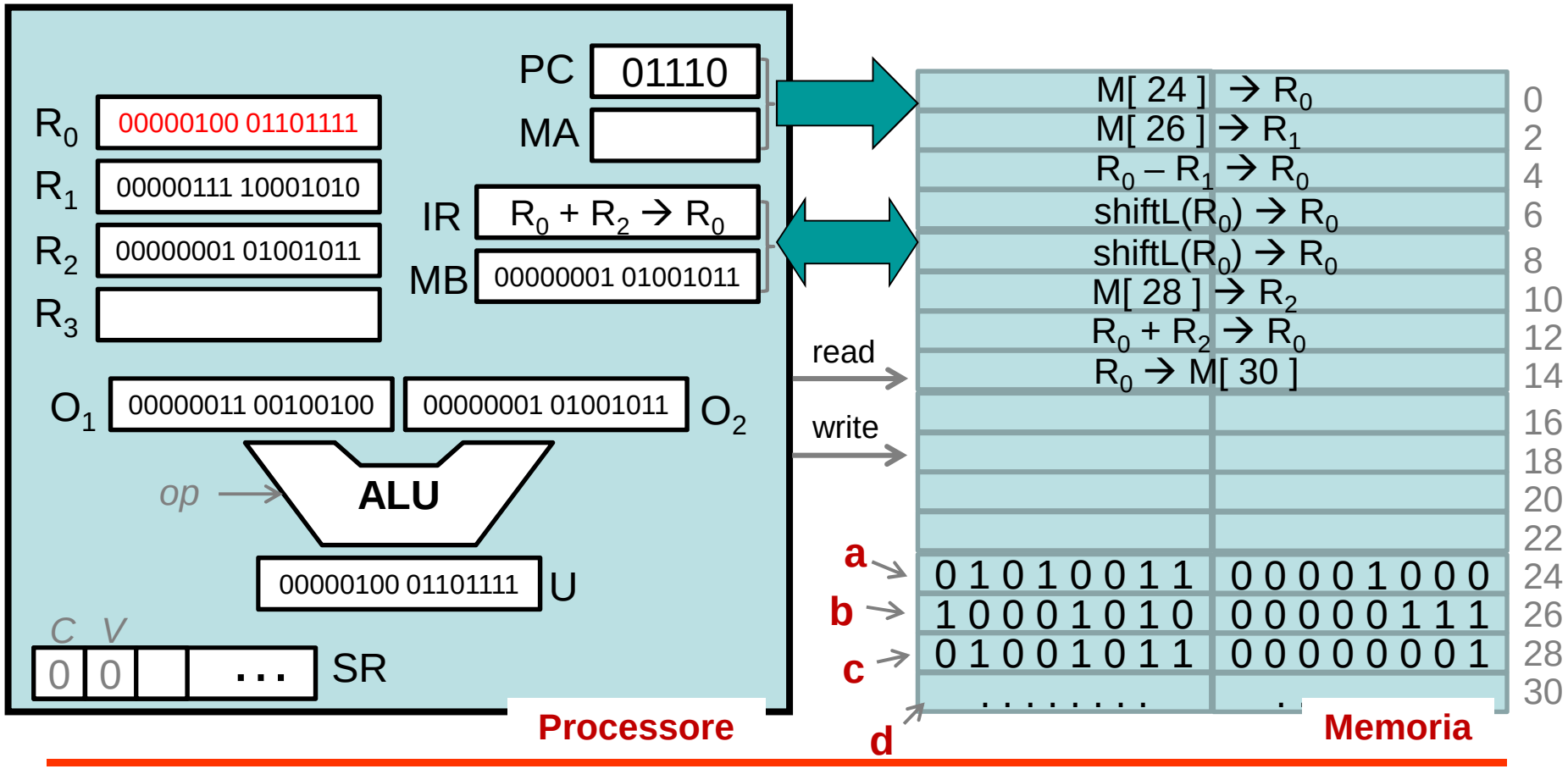
Quindi viene comandata all'**ALU** un'operazione di addizione. Il risultato è scritto nel registro di uscita **U**. L'operazione non genera riporto né overflow.



Esecuzione passo-passo

Settima istruzione

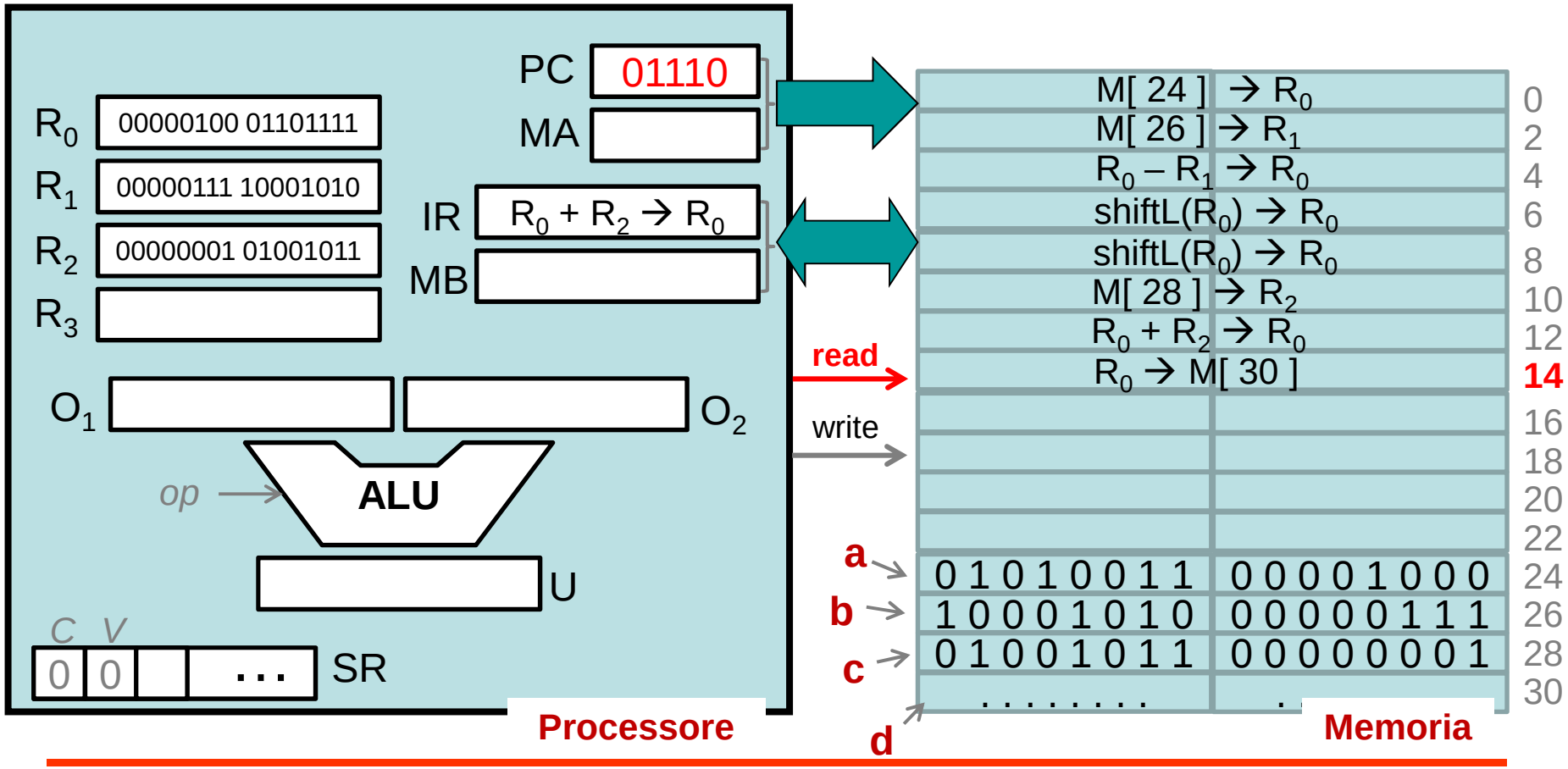
Infine, il risultato viene copiato dall'uscita dell'**ALU** nel registro destinazione indicato dall'istruzione, $\mathbf{R_0}$. La settima istruzione è completa.



Esecuzione passo-passo

Ottava istruzione

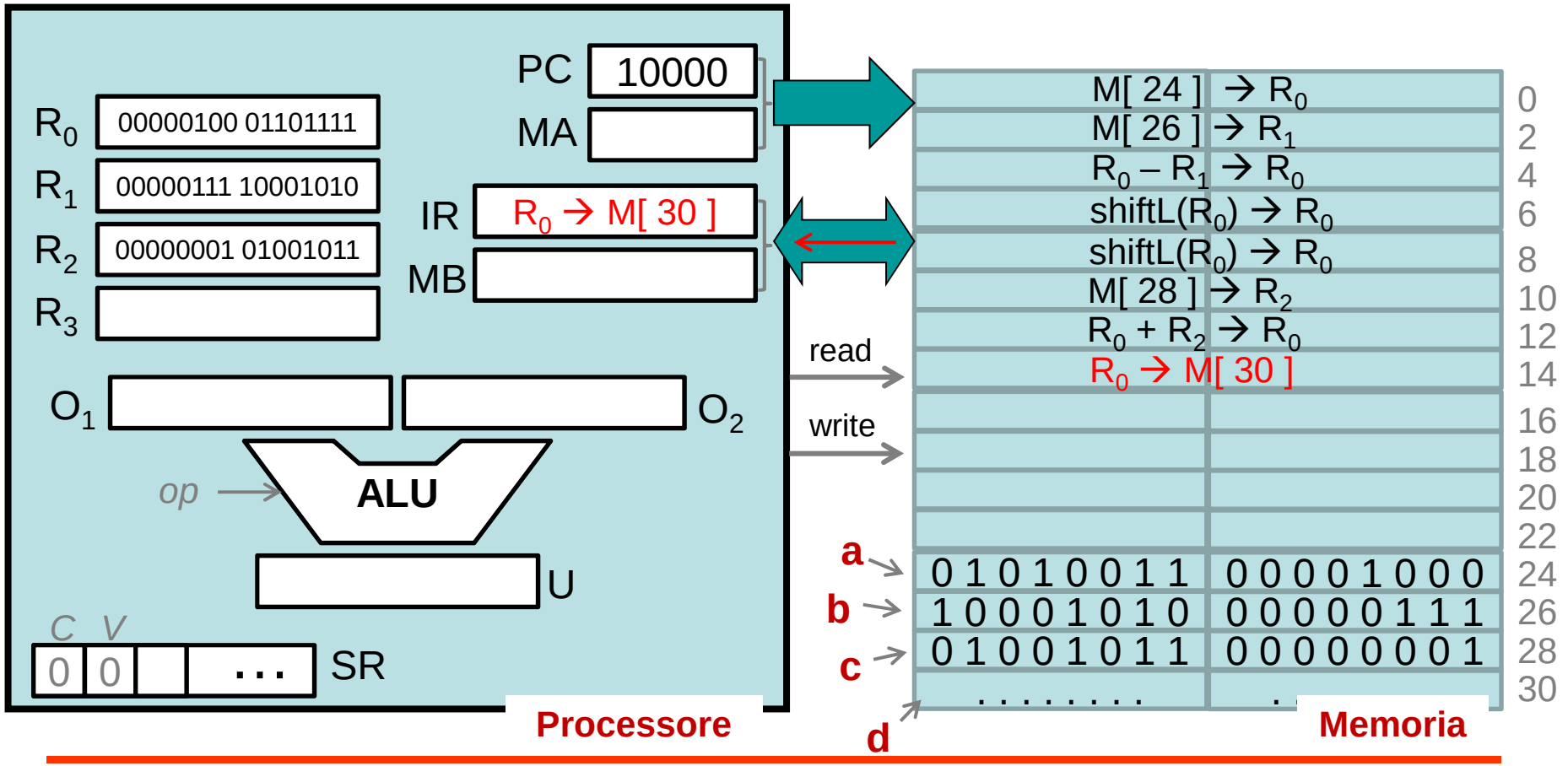
PC nel frattempo è stato incrementato di 2 e vale ora $(\mathbf{01110})_2$=**14**. Questo permette di ripetere il *fetch* con l'ottava istruzione.



Esecuzione passo-passo

Ottava istruzione

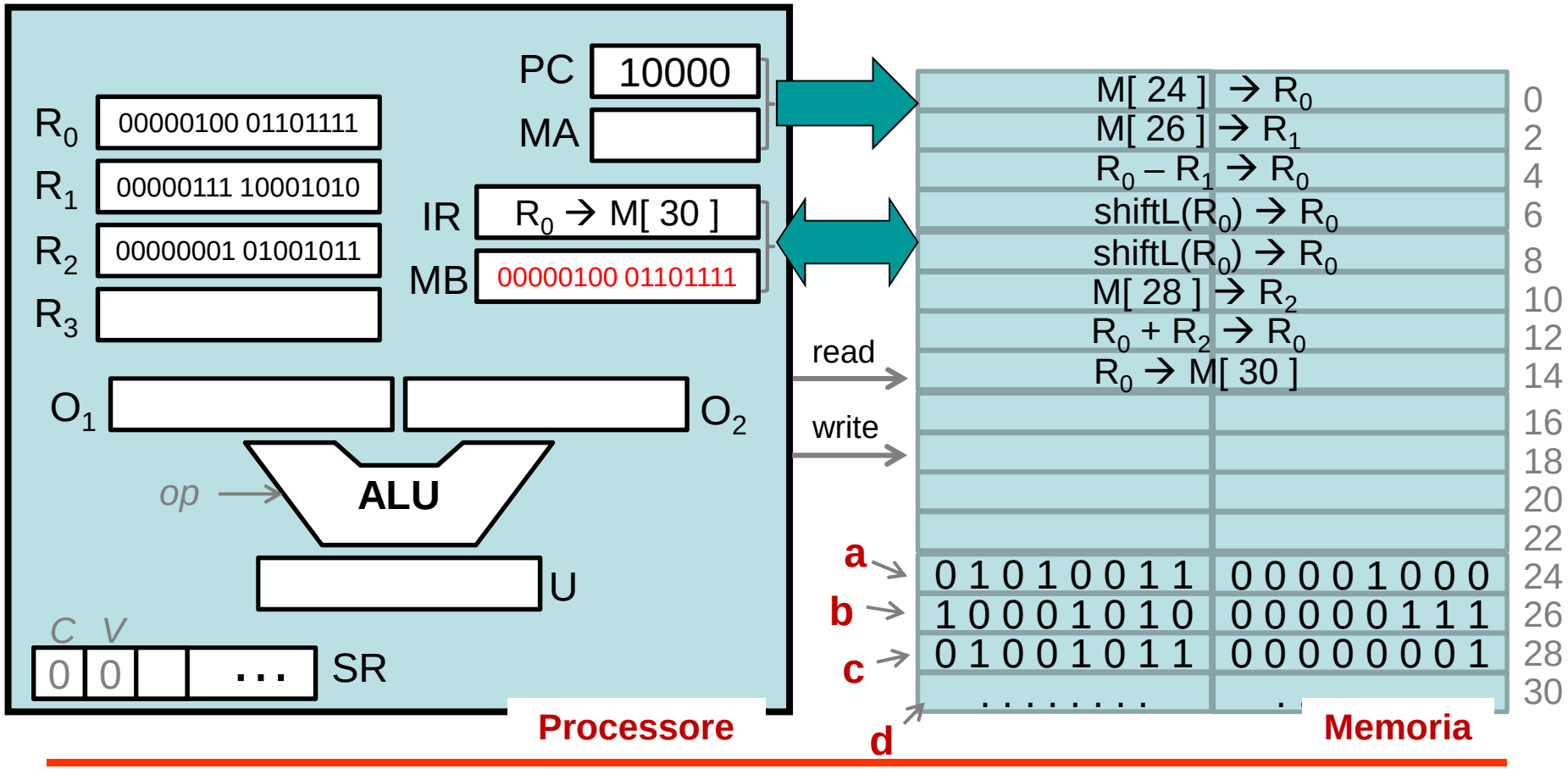
L'istruzione viene trasferita nel registro **IR**. Il processore la legge e determina i passi necessari per eseguirla. Qui è richiesta una **scrittura** registro→ memoria.



Esecuzione passo-passo

Ottava istruzione

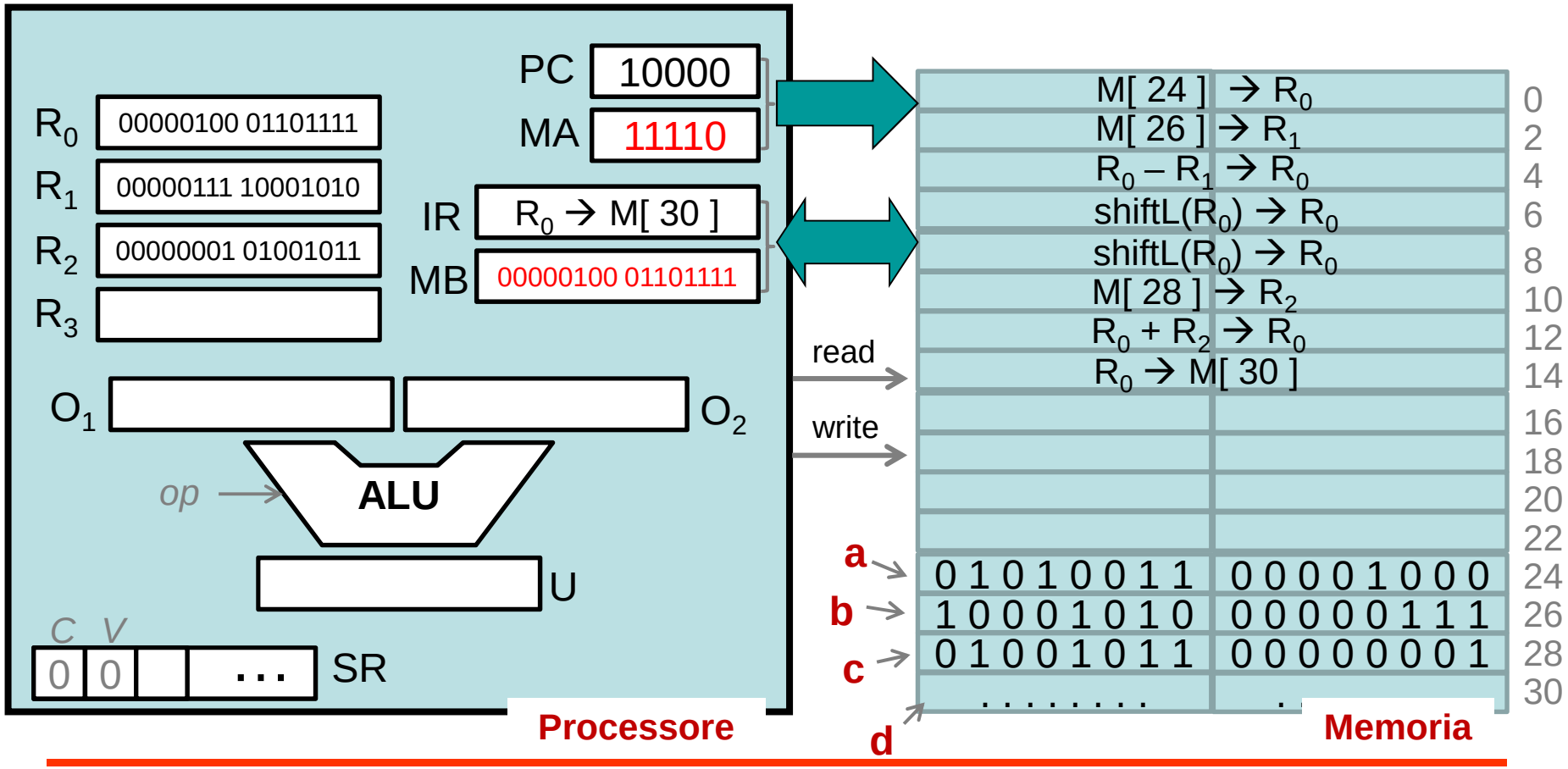
Il processore pone quindi in **MB** il valore contenuto nel registro $\mathbf{R_0}$ in vista del suo trasferimento verso la memoria.



Esecuzione passo-passo

Ottava istruzione

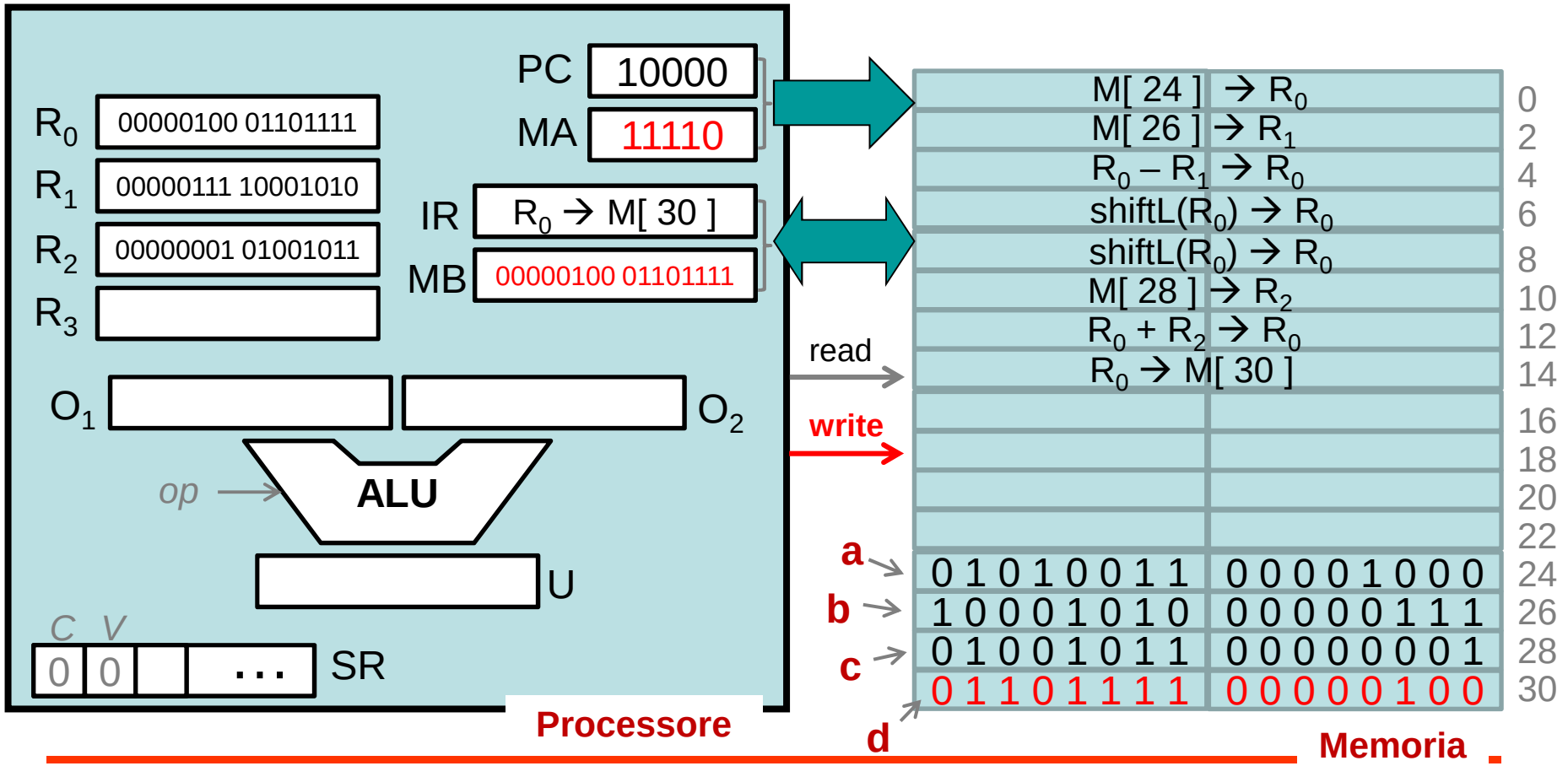
Il processore pone poi il valore dell'indirizzo **30**=(**11110**)$_2$ all'interno del registro **MA**



Esecuzione passo-passo

Ottava istruzione

Infine, comanda un'operazione di scrittura (write). Il risultato è finalmente scritto nella locazione di memoria che corrisponde alla variabile **d**.



Esecuzione passo-passo

- Dopo l'esecuzione delle 8 istruzioni, in memoria all'indirizzo corrispondente alla variabile **d** (indirizzo 30) si trova il risultato dell'operazione:
  - **d = 4*(a – b) + c**
- dove

  a = (2131)$_{10}$ = (1000 01010011)$_2$

  b = (1930)$_{10}$ = (111 10001010)$_2$

  c = (331)$_{10}$ = (1 01001011)$_2$
- Il risultato è

  d = (1135)$_{10}$ = (100 01101111)$_2$
- (scritto in memoria, come le altre variabili, con convenzione *big-endian*)

dopo?

- Si noti che PC continua ad essere incrementato
- sarà effetuato un fetch alla posizione 16
- Il processore presuppone che le istruzioni da eseguire siano tutte presenti in sequenza, una dopo l'altra
- Cosa si può fare se le istruzioni non sono sempre consecutive?
- Semplicemente, basta cambiare il valore del **PC** inserendo "manualmente" l'indirizzo della prossima istruzione che si vuole far eseguire al processore (dovunque sia collocata in memoria):
  - *indirizzo_prossima_istruzione* → **PC**
- E' un'istruzione di **salto**