

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

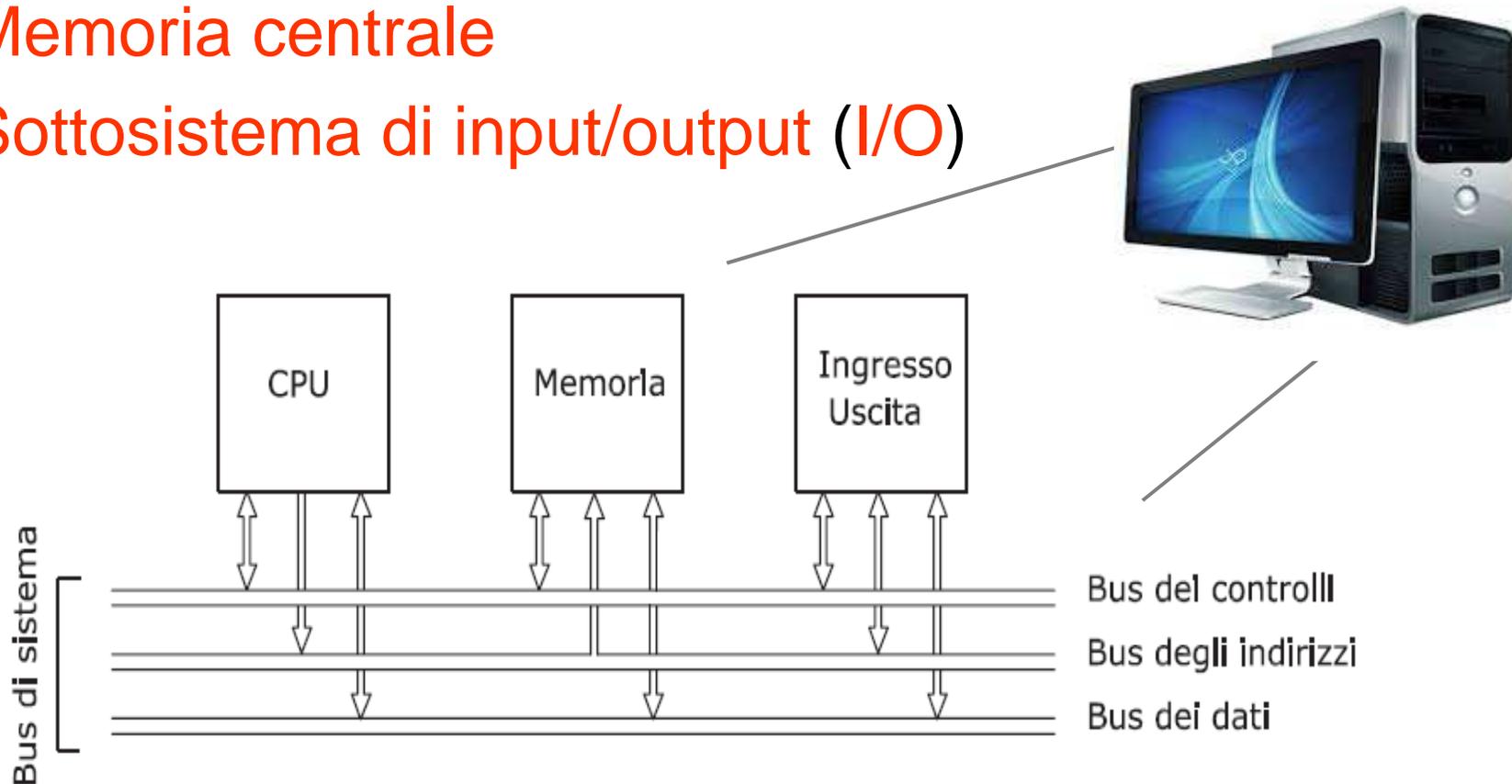
Architettura del calcolatore

ing. Alessandro Cilardo

Accademia Aeronautica di Pozzuoli
Corso Pegaso V "GArn Elettronici"

Architettura del calcolatore

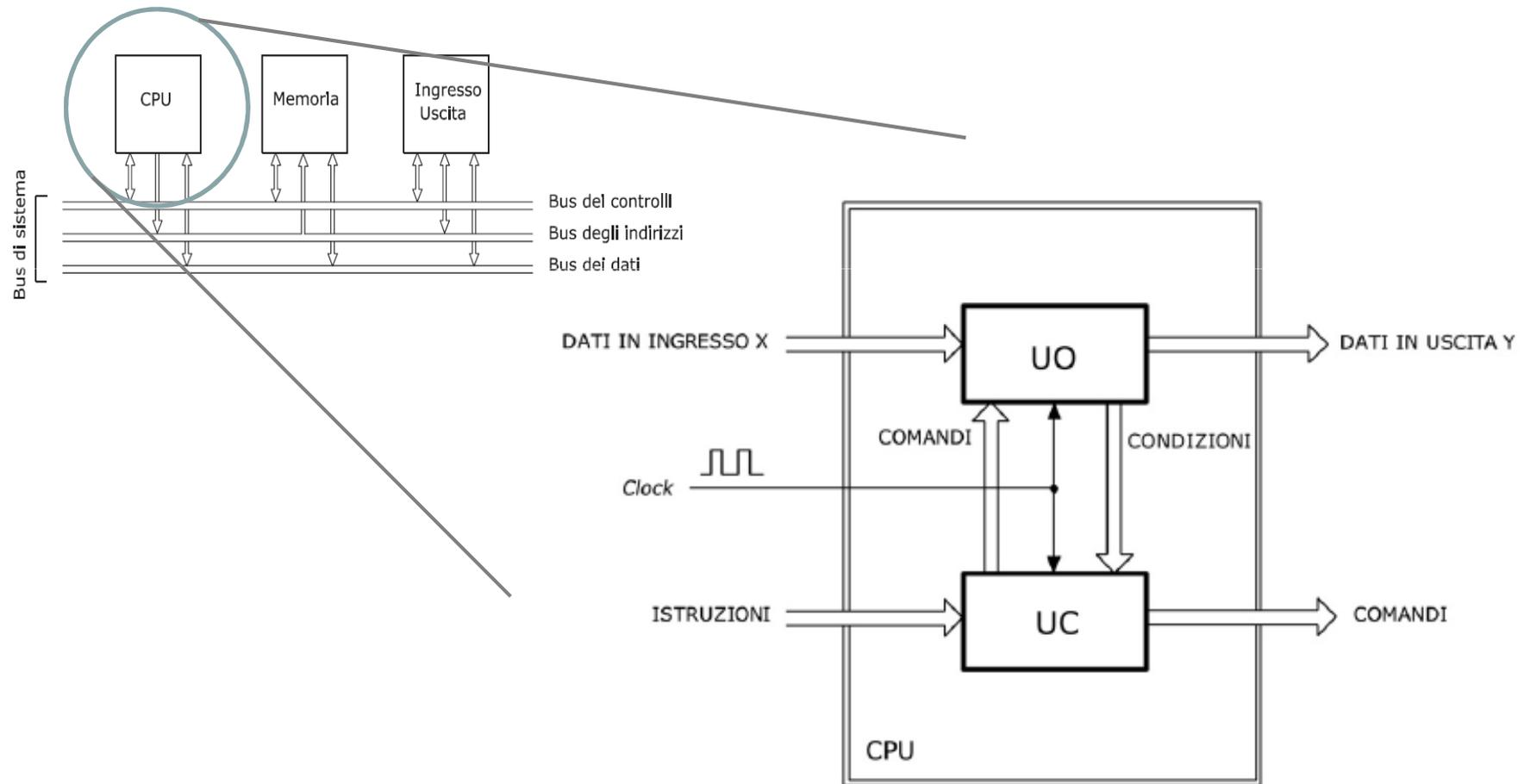
- Processore o CPU (*Central Processing Unit*)
- Memoria centrale
- Sottosistema di input/output (I/O)



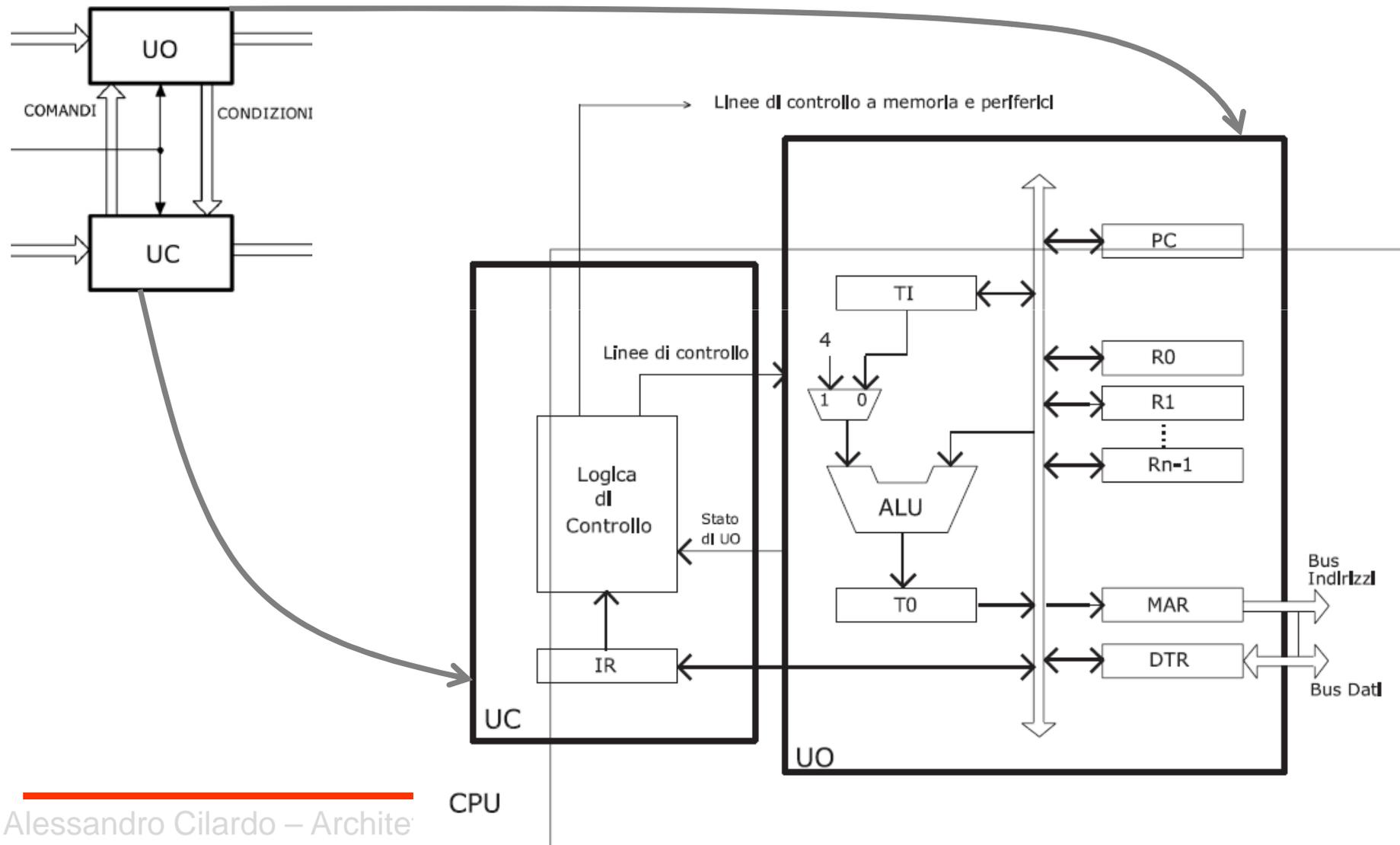
Il processore (CPU)

- È in grado di eseguire un insieme di azioni elaborative elementari più o meno complesse
- Le *istruzioni* sono comandi che
 - » governano il trasferimento di informazioni sia all'interno del calcolatore sia tra il calcolatore e la memoria ed i dispositivi di I/O
 - » specificano le operazioni aritmetiche e logiche che devono essere effettuate sui dati
- I *dati* di ingresso e di uscita dell'elaborazione e la sequenza di istruzioni da eseguire (programma) sono immagazzinati nella memoria centrale
- Il processore preleva ed esegue le istruzioni dalla memoria una ad una, per poi eseguirle al proprio interno

CPU: struttura interna



CPU: struttura interna

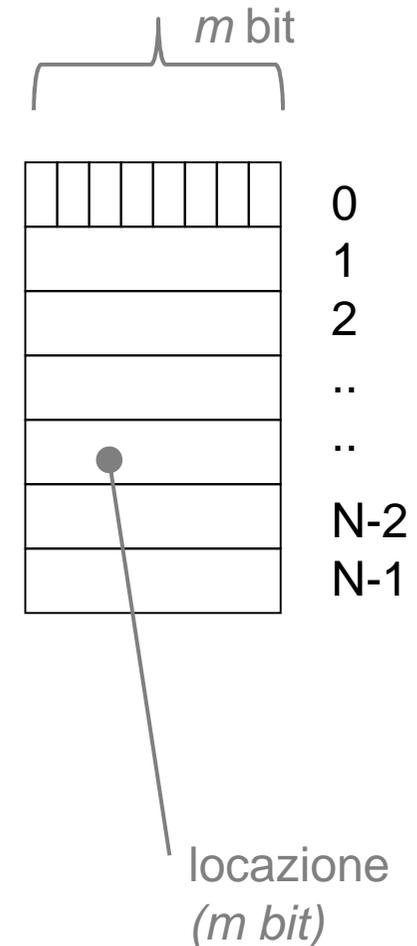


CPU: struttura interna

- Componenti fondamentali del processore:
 - Unità di controllo
 - determina cosa il processore deve fare, ovvero quali sono i passi elementari da compiere per eseguire le istruzioni
 - Unità logico-aritmetica (ALU)
 - contiene i circuiti necessari per effettuare le operazioni logico/matematiche direttamente eseguibili dal processore
 - il suo funzionamento è guidato dall'unità di controllo
 - Sezione di collegamento con la memoria
 - contiene i circuiti necessari per far transitare dati da e verso la memoria (che è un componente esterno alla CPU)
 - Sezione di collegamento con Ingresso-Uscita
- Il *linguaggio macchina* di un processore è costituito dalla codifica in binario delle istruzioni eseguibili dal processore

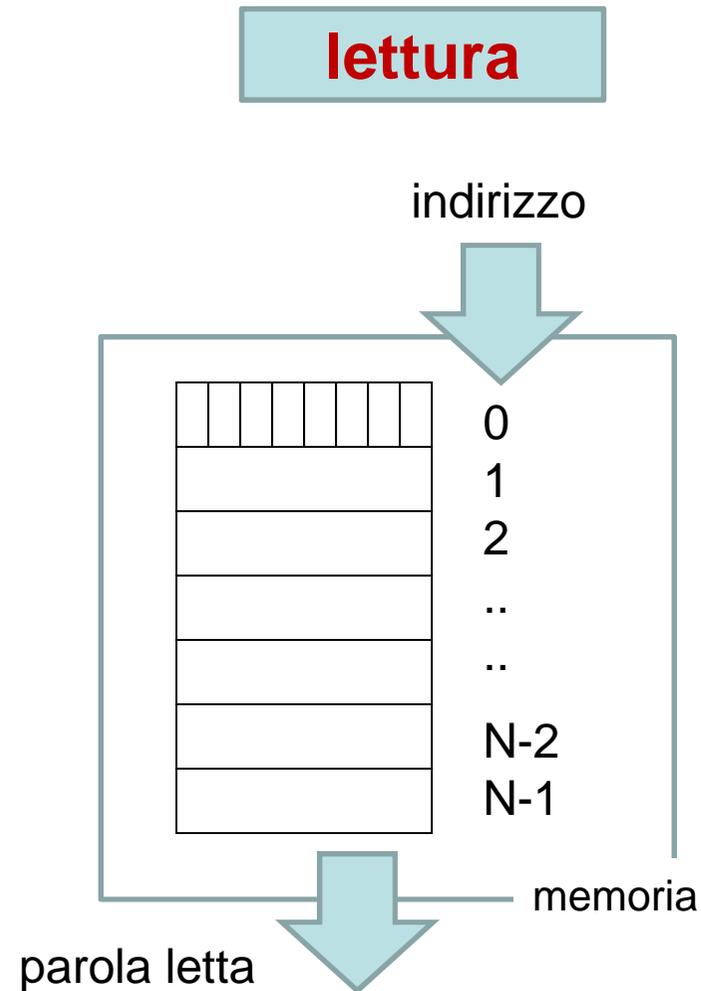
La memoria centrale

- La memoria centrale è un “contenitore” di dati
- E' organizzata come una sequenza di gruppi di bit, ciascuno di lunghezza m , detti *locazioni*
- Gli m bit di una locazione sono accessibili in blocco dal processore (in lettura/scrittura) mediante un'unica operazione
 - gli m bit costituiscono una *parola* in memoria
 - tipiche dimensioni della parola sono 8, 16, 32, 64 bit
- Ogni locazione è individuata da un *indirizzo*, cioè un intero compreso tra 0 e $N-1$, con $N = 2^c$
 - » $[0, N-1] =$ SPAZIO DI INDIRIZZAMENTO
- La memoria centrale è *ad accesso casuale* (RAM) in quanto il tempo di accesso non dipende dalla posizione del dato



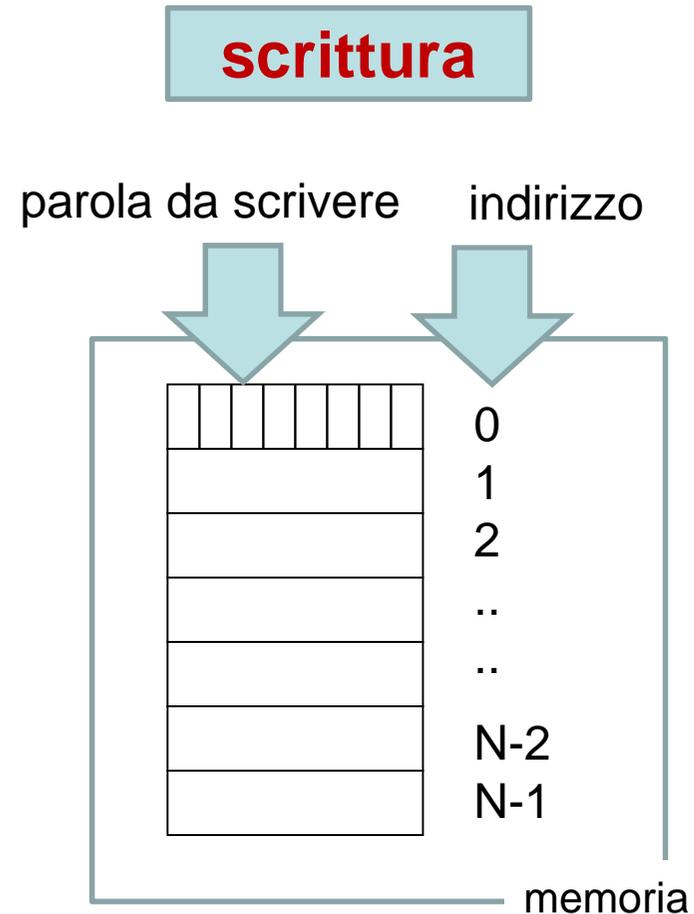
Operazioni di lettura e scrittura

- Le operazioni di **lettura** si effettuano fornendo alla memoria l'informazione che indica *dove* leggere la parola
 - viene ovvero fornito l'**indirizzo**
 - la memoria risponde fornendo il valore della parola contenuta a quell'indirizzo



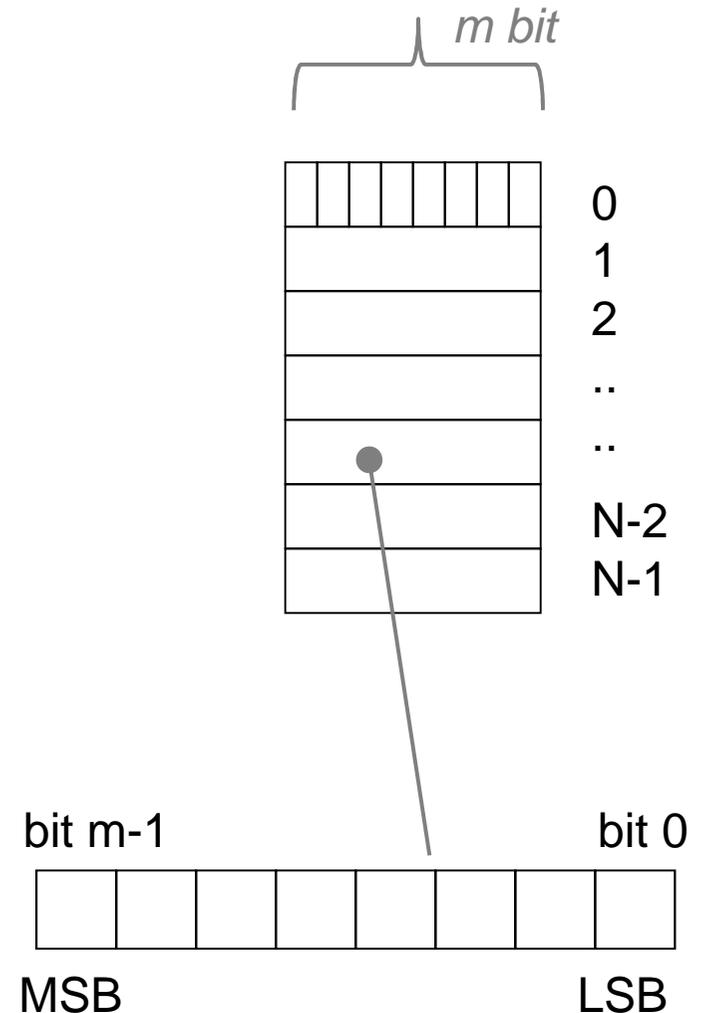
Operazioni di lettura e scrittura

- Le operazioni di **scrittura** si effettuano fornendo alla memoria la parola da scrivere e l'informazione che indica *dove* scrivere la parola stessa
 - vengono ovvero forniti sia il **dato** che l'**indirizzo**
 - in questo caso la memoria non fornisce risposta



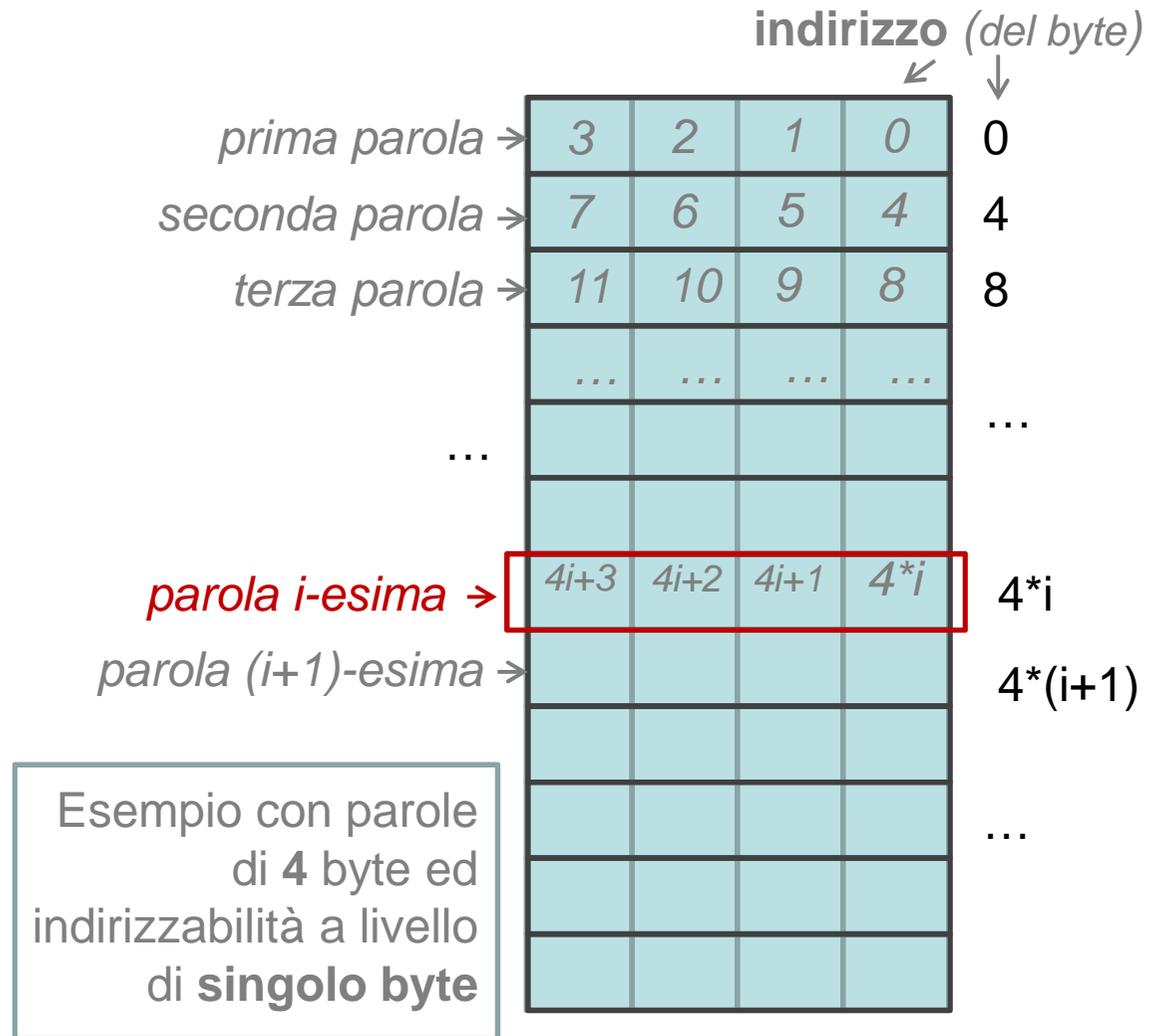
Parole in memoria

- tipiche dimensioni della parola m sono 8, 16, 32, 64 bit
- Le parole ospitano dati elementari:
 - numeri codificati in binario, caratteri, numeri in virgola mobile, etc..
- E' importante conoscere l'ordine con cui sono memorizzati i bit in una parola
 - ovvero dove, tra gli m bit, è collocato il MSB (*most significant bit*) e dove il LSB (*least significant bit*)



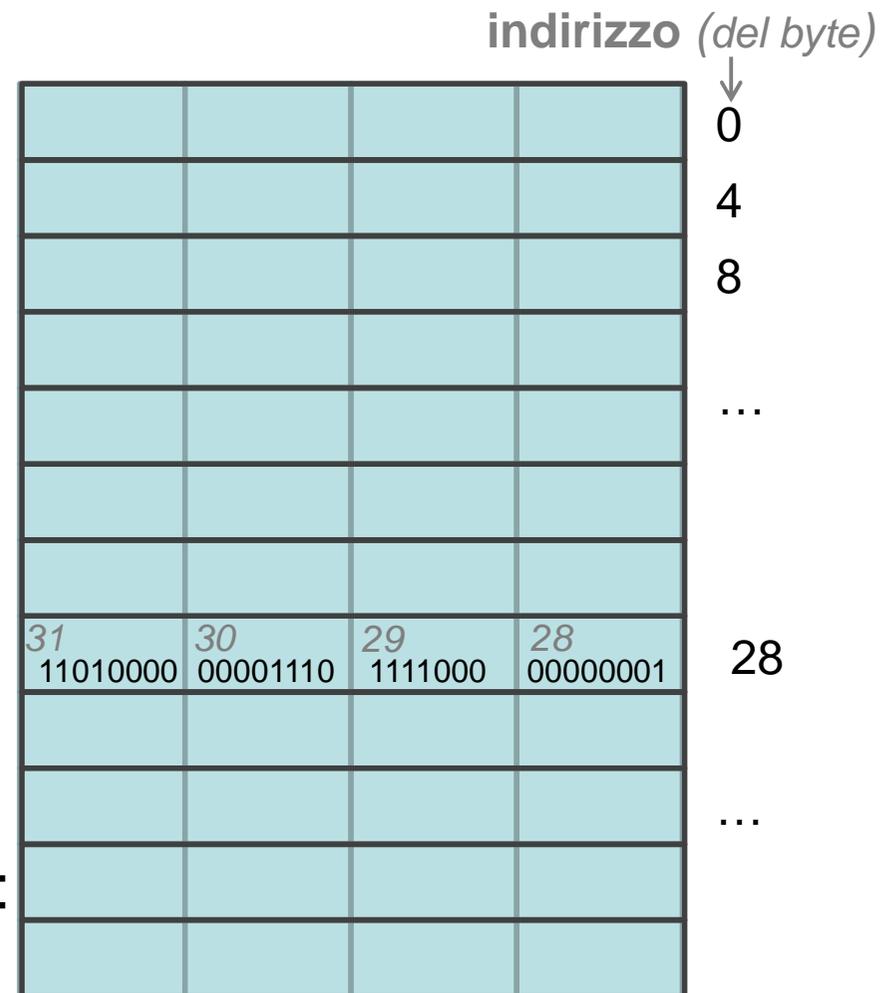
Parole in memoria

- Quando le parole sono composte da *più* byte (tipicamente 2, 4 o 8), molti calcolatori prevedono la possibilità di localizzare tramite l'indirizzo lo specifico byte all'interno della parola
- L'indirizzo è in tal caso **associato al byte** all'interno di ciascuna parola, e non alla parola stessa



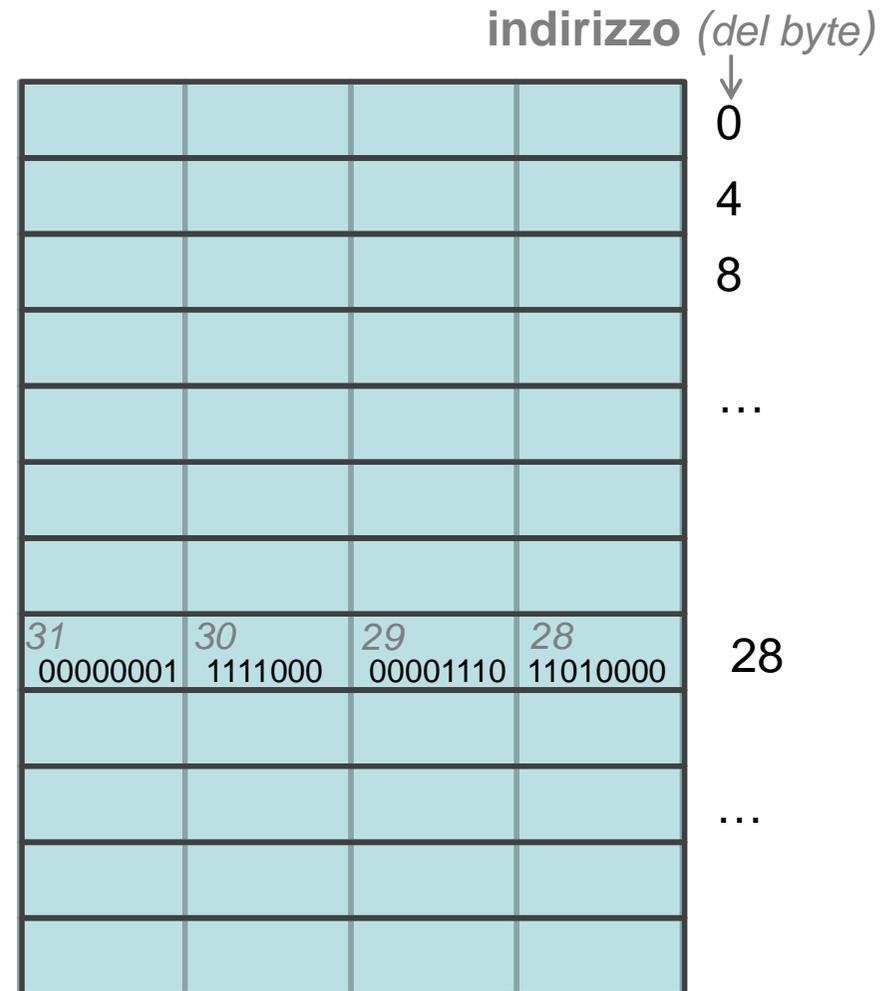
Parole in memoria: little-endian

- Memorizziamo il seguente numero binario:
`11010000 00001110 1111000 00000001`
all'indirizzo **28**
- Si noti l'ordine:
`11010000` è all'indirizzo 31
`00001110` è all'indirizzo 30
`1111000` è all'indirizzo 29
`00000001` è all'indirizzo 28
- *Byte meno significativi finiscono in indirizzi più bassi:*
- organizzazione **little-endian**



Parole in memoria: big-endian

- Memorizziamo il numero binario
11010000 00001110 1111000 00000001
- all'indirizzo 28
- Si noti l'ordine:
11010000 è all'indirizzo 28
00001110 è all'indirizzo 29
1111000 è all'indirizzo 30
00000001 è all'indirizzo 31
- *Byte meno significativi finiscono in indirizzi più alti:*
- organizzazione **big-endian**



little-endian e big-endian

- Ai fini dell'organizzazione del calcolatore, la scelta tra *little-endian* e *big-endian* è indifferente.
- Tuttavia, una volta fatta, questa scelta condiziona il modo nel quale sono organizzate le strutture dati dei programmi e gli accessi in memoria da parte del processore
- programmi scritti per architetture big-endian non sono compatibili con programmi scritti per architetture little-endian
 - Processore **Mototola 68000**: esempio di processore big-endian
 - Processore **Intel 8086**: esempio di processore little-endian

Allineamento

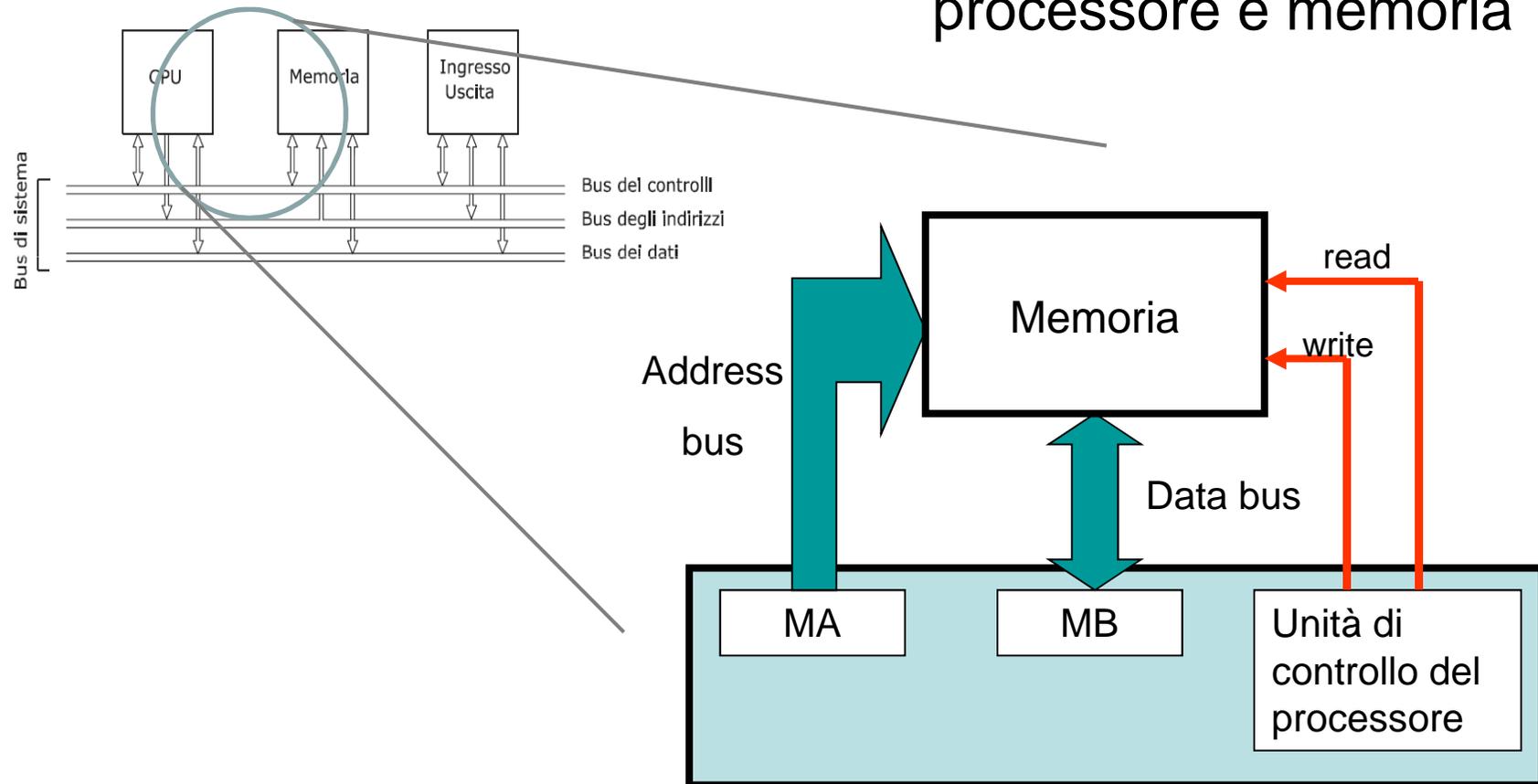
- Per un processore a parola di 16 bit, una *parola* che inizia ad un indirizzo pari si dice “allineata sul limite di parola”
- Tipicamente, un tale processore è in grado di accedere ai due byte che costituiscono una parola allineata mediante una sola operazione di lettura
- Ad esempio, il processore 8086 consente l’uso di parole non allineate, cioè parole che iniziano ad un indirizzo dispari, ma in tal caso sono necessari 2 distinti accessi in memoria
- Il processore 68000 **NON** consente l’accesso a parole non allineate



(X pari) La parola (X+1) non è allineata sul limite di parola

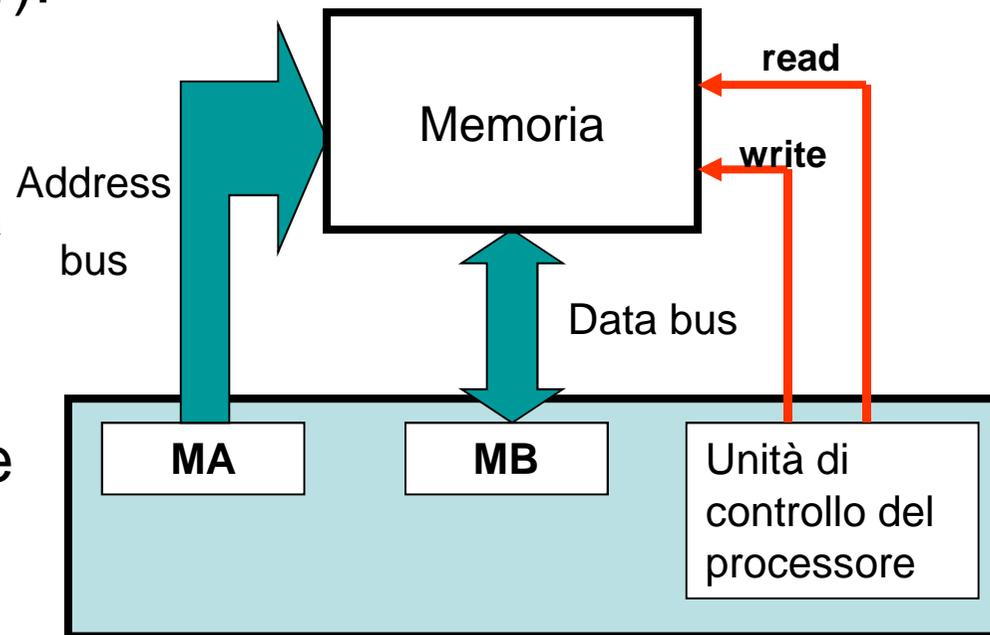
Collegamento CPU-memoria

Interfacciamento fisico tra processore e memoria



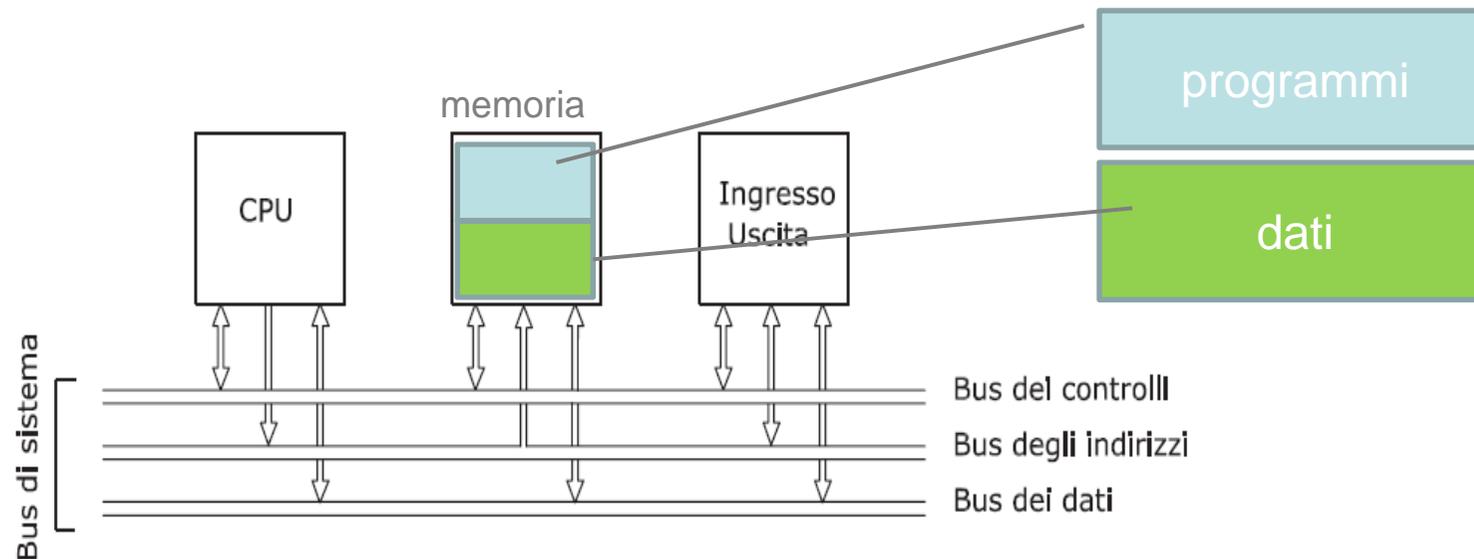
Collegamento CPU-memoria

- **MA (Memory Address):** registro usato dal processore per scrivere l'indirizzo da comunicare alla memoria per accedere ad una determinata locazione
- **MB (Memory Buffer):** registro usato per trasferire il dato da/verso la memoria
- segnali **read / write**: dicono alla memoria se si vuole effettuare un'operazione di lettura o di scrittura



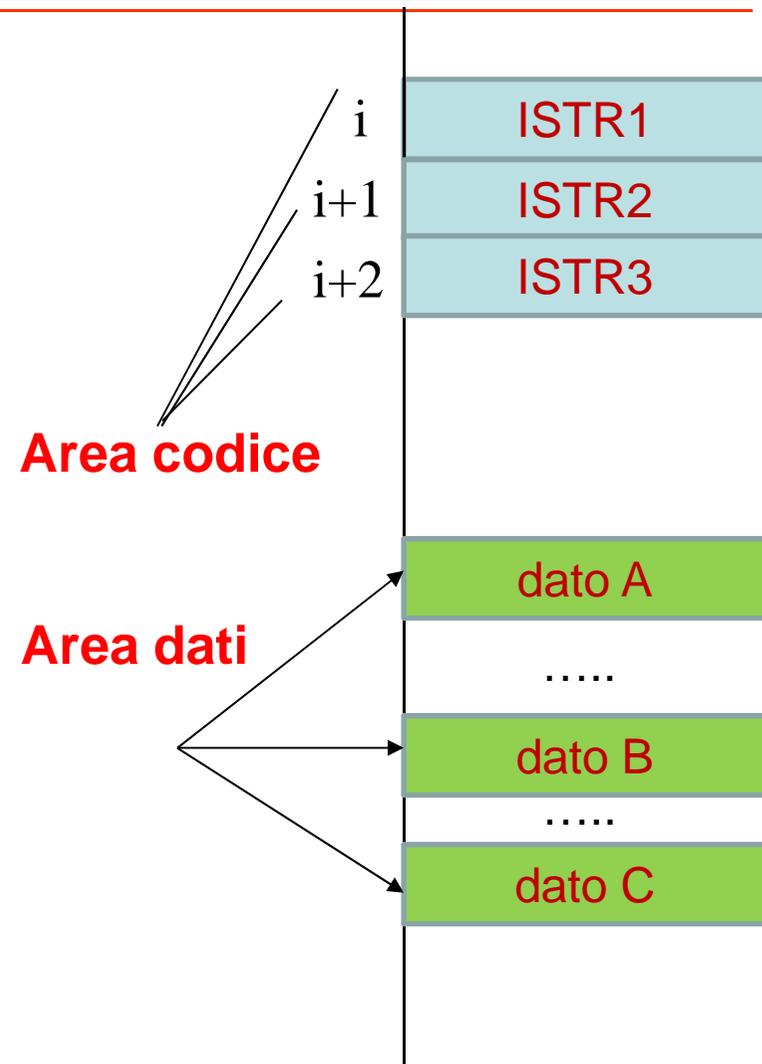
Memoria dati ed istruzioni

- La memoria è usata per ospitare sia le **istruzioni** (programmi) che il processore deve eseguire, sia i **dati** su cui deve lavorare



Memoria dati ed istruzioni

- Esempio di organizzazione in memoria
- Alcuni gruppi di locazioni sono utilizzati per ospitare le **istruzioni** in sequenza (*area codice, o istruzioni*)
- altre parti della memoria sono usate per ospitare i **dati** (*area dati*)

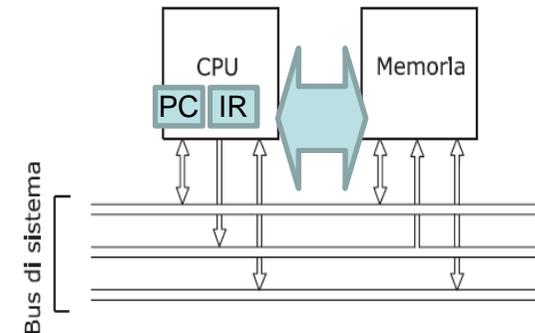


Algoritmo del processore

- Qualsiasi processore contiene due registri speciali
- Registro **PC**: contiene l'indirizzo in memoria della prossima istruzione che deve essere eseguita
- Registro **IR**: contiene una copia dell'istruzione da eseguire prelevata dalla memoria

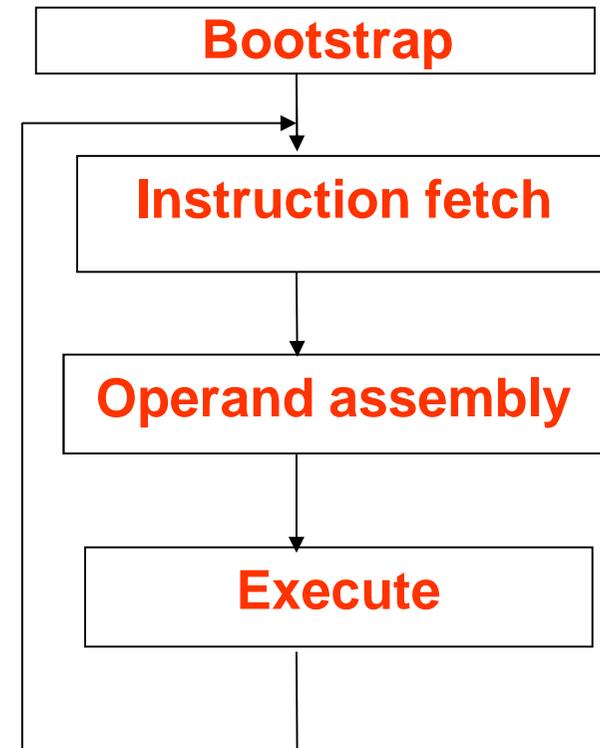
Algoritmo del processore

- **Prelievo dell'istruzione (Fetch)**
 - La **CPU** preleva dalla **memoria** l'istruzione leggendola dall'area codice all'indirizzo indicato dal registro **PC**
 - L'istruzione viene copiata nel registro **IR**
- **Decodifica / prelievo degli operandi (Operand Assembly)**
 - L'unità di controllo esamina il contenuto di **IR** e ricava il tipo di operazione ed i relativi operandi
 - Eventuali operandi contenuti in memoria vengono prelevati con un *ulteriore accesso* in lettura (dall'area dati)
- **Esecuzione dell'istruzione (Execute)**
 - L'unità di controllo richiede all'**ALU** l'operazione specificata nell'istruzione ed invia il risultato ad un registro o alla memoria



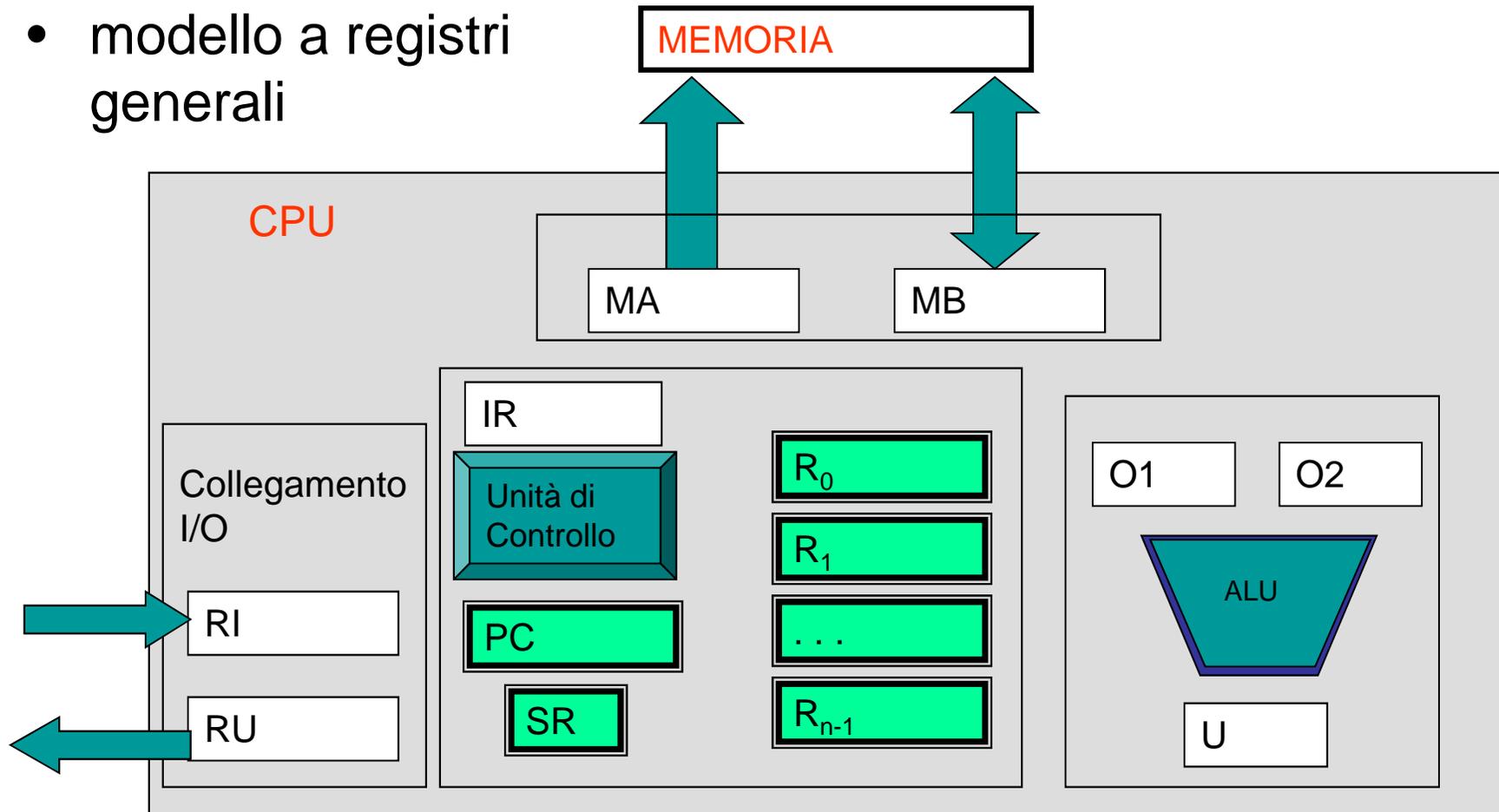
Algoritmo del processore

- L'unità di controllo opera in un ciclo infinito:
 - Prelievo (*fetch*)
 - Preparazione degli operandi (*operand assembly*)
 - Esecuzione
- Nella fase di *bootstrap* il ciclo viene inizializzato
 - avviene **solo all'accensione**
 - viene avviata l'esecuzione di un programma iniziale in una memoria non volatile inserendo un valore iniziale opportuno nel registro PC



Componenti del processore

- modello a registri generali

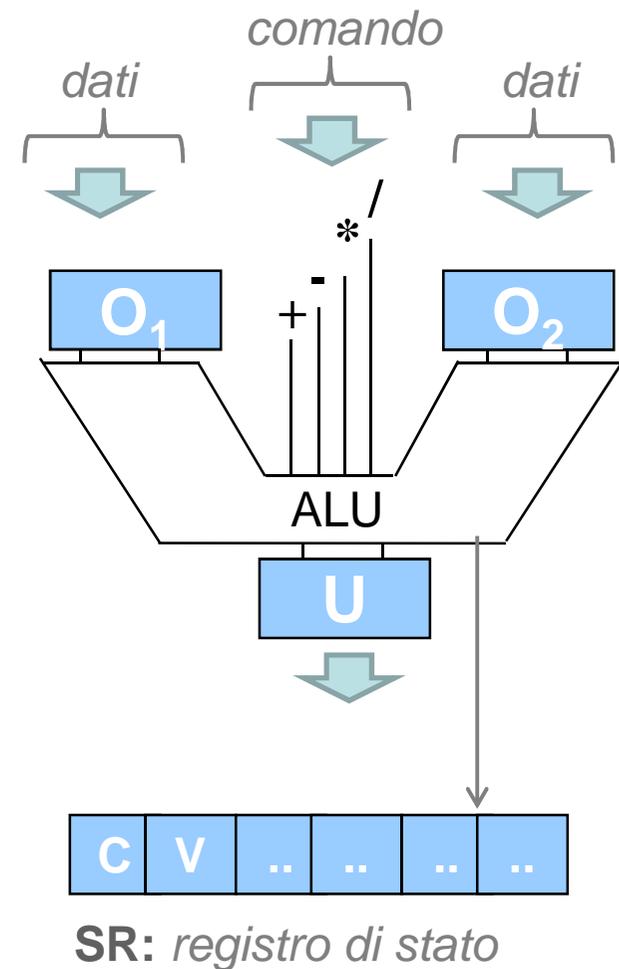


Componenti del processore

- Componenti fondamentali del processore:
 - Unità di controllo
 - registro Program Counter (PC) o Prossima Istruzione
 - Instruction Register o registro di decodifica (IR)
 - registri di Macchina
 - Unità aritmetico-logica (ALU)
 - Sezione di Collegamento con la memoria
 - registro degli indirizzi di memoria o Memory Address (MA)
 - registro di transito dei dati dalla memoria Memory Buffer (MB)
 - Sezione di Collegamento con Ingresso-Uscita

Unità logico-aritmetica

- L'Unità di controllo fornisce alla ALU gli **operandi**, insieme ad un **comando** che indica l'operazione da effettuare
- Gli operandi sono copiati nei registri di ingresso della ALU (O_1 , O_2)
- La ALU esegue l'operazione e pone il risultato nel registro risultato (**U**);
- inoltre, altera il valore dei bit (o *flag*) del registro di stato (**SR**) in funzione del risultato. Esempi di flag:
 - **C**: bit che viene alzato se l'operazione ha determinato un riporto (*Carry*)
 - **V**: bit che viene alzato se l'operazione ha determinato un *overflow*
 - etc..



Registri del processore

➤ Registri *interni*

- » Necessari al funzionamento del processore
- » Non direttamente utilizzabili dal programmatore
 - » non appartengono al *modello di programmazione*
 - » Es. MA, MB, IR, ...

➤ Registri *di macchina*

- » Visibili al programmatore (ovvero, utilizzabili)
 - » appartengono al *modello di programmazione*
 - Registri generali (R_0, R_1, R_{n-1})
 - Registri speciali (PC, SR, ...)

Registri del processore

- Insieme di registri R_0, R_1, \dots, R_{N-1} utilizzabili indifferentemente
- Le istruzioni che operano su registri sono **più veloci** di quelle che operano su locazioni di memoria
- Il programmatore può utilizzare i registri del processore per memorizzare i dati di uso più frequente invece che tenere i dati in memoria (concetto di *gerarchia di memorie*)
- Istruzioni con operandi registri:
 $[R_0] + [R_1] \rightarrow R_1$
- Istruzioni con operandi memoria-registri:
 $[R_0] + M[1000] \rightarrow R_0$ (*memory-to-register*)
 $M[1000] + [R_1] \rightarrow M[1000]$ (*register-to-memory*)

Registri del processore

- Istruzioni con operandi registri:

$$[R_0] + [R_1] \rightarrow R_1$$

preleva il contenuto dei **registri** R_0 e di R_1 , somma i due valori tramite l'ALU, poni il risultato in R_1 (sovrascrivendo il precedente contenuto)

- Istruzioni con operandi memoria-registri:
memory-to-register:

$$[R_0] + M[1000] \rightarrow R_0$$

preleva il contenuto di R_0 e della locazione numero 1000 in **memoria** (M), somma i due valori tramite l'ALU, poni il risultato in R_0

register-to-memory:

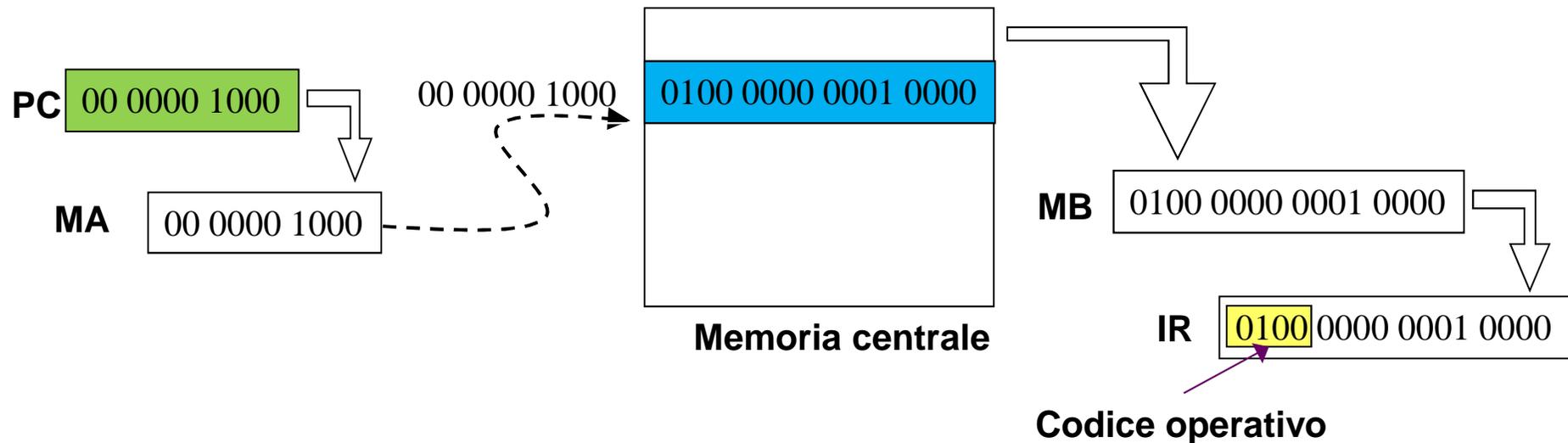
$$M[1000] + [R_1] \rightarrow M[1000]$$

preleva il contenuto della locazione numero 1000 in **memoria** e di R_1 , somma i due valori tramite l'ALU, poni il risultato in **memoria** all'indirizzo 1000 (sovrascrivendo il precedente contenuto)

Fase di fetch

- Descrizione dettagliata del processo di fetch
- $M[PC] \rightarrow IR; PC + k \rightarrow PC$

k è la dimensione in byte di una parola in memoria, ad es. 4 byte (32 bit)



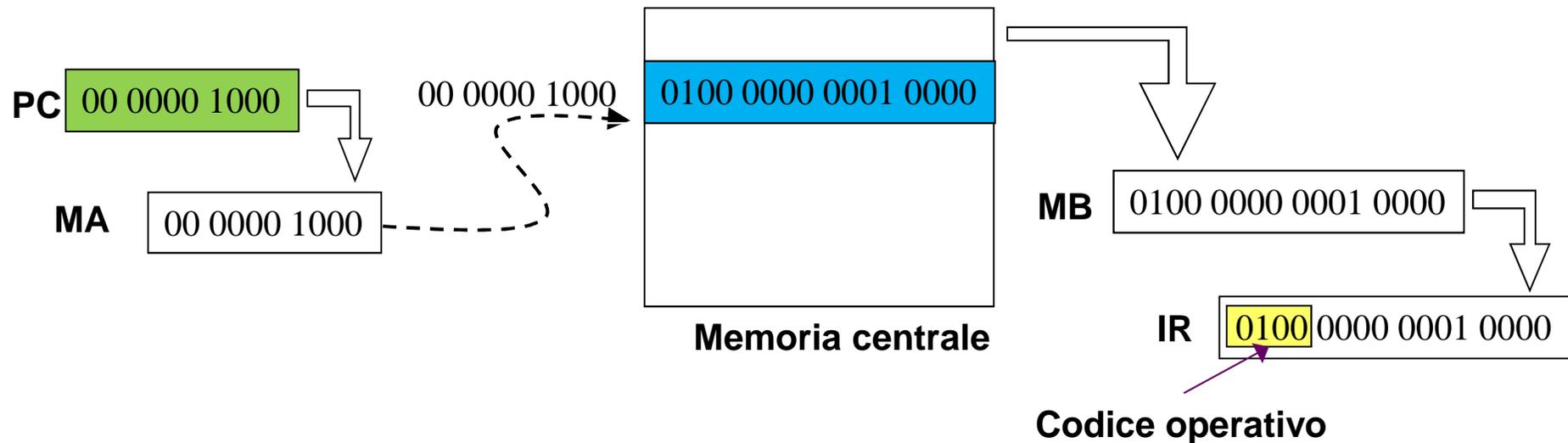
Fase di fetch

- $M[PC] \rightarrow IR;$

preleva il contenuto della locazione in **memoria** il cui indirizzo è **PC** e spostare il contenuto nel registro **IR**: l'istruzione viene così caricata dalla memoria all'interno del processore

- $PC+k \rightarrow PC$

incrementa **PC** della dimensione della parola (**k**). In questo modo, **PC** assume il valore dell'indirizzo della successiva istruzione presente in memoria, la prossima da caricare ed eseguire



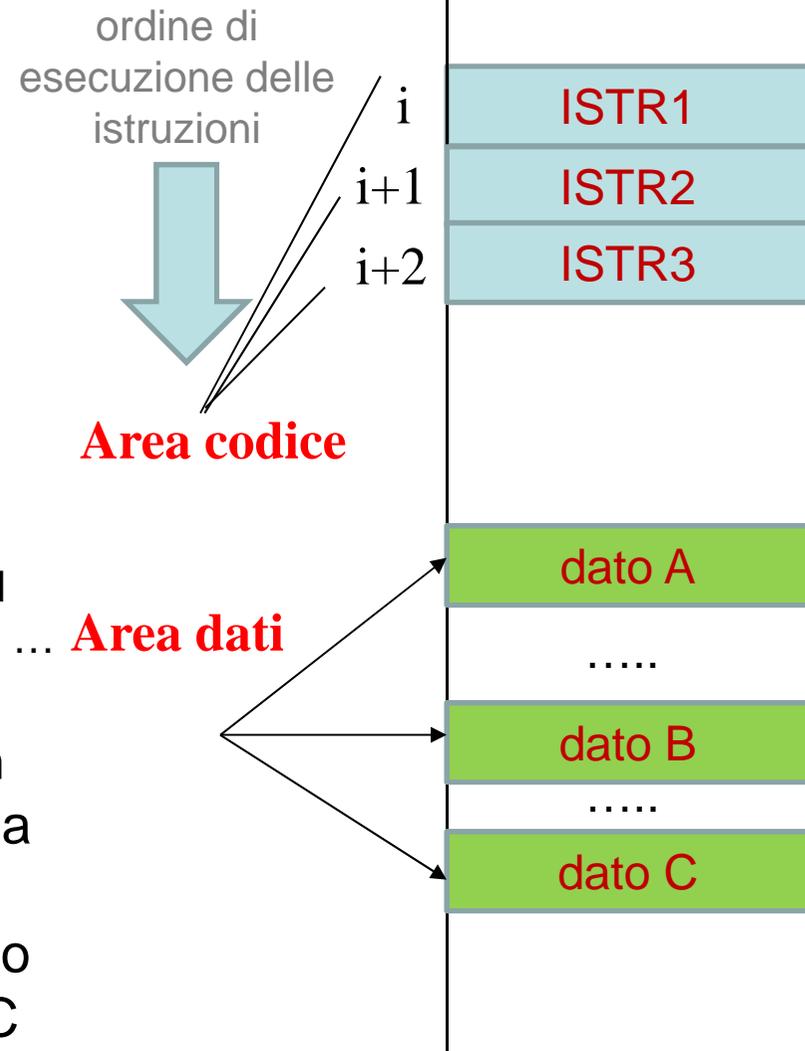
Fase di fetch

- Alla fine della fase fetch:

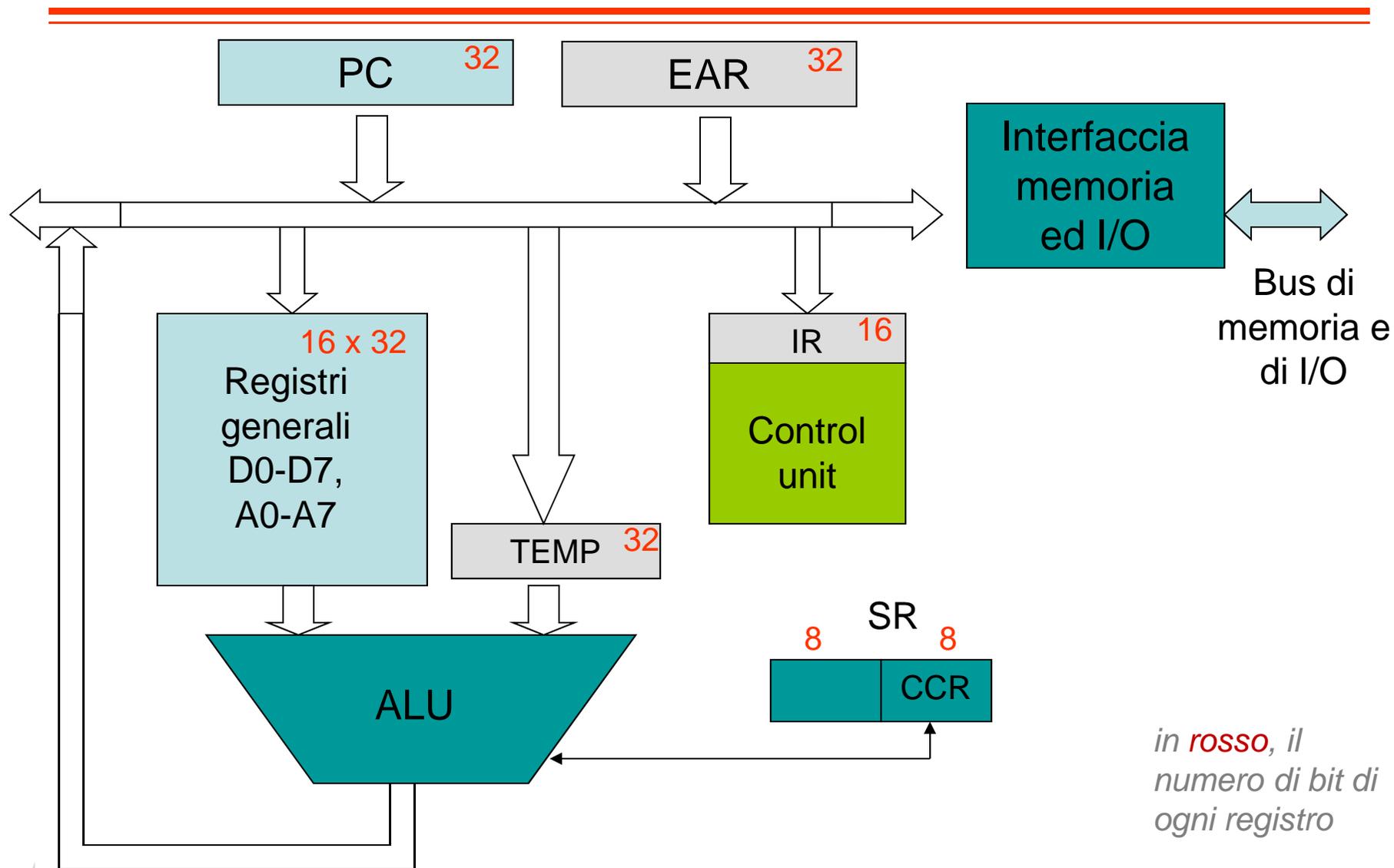
$$PC = PC + k$$

k = lunghezza istruzioni in byte

- serve a far sì che PC punti all'istruzione posta in memoria subito dopo
 - in questo esempio, verranno caricate ed eseguite una dopo l'altra ISTR1, ISTR2, ...
- Esecuzione delle istruzioni *in sequenza* nell'ordine in cui sono memorizzate nella memoria
- Per cicli e strutture di controllo (*if-then, if-then-else, switch*) si usano istruzioni di salto che modificano PC



Architettura del processore 68000



Architettura del processore 68000

- **Dati:**
 - All'esterno:
 - parola di 16 bit (16 segnali, o *pin*, per i dati)
 - All'interno:
 - registri di 32 bit
- **Indirizzi:**
 - All'esterno:
 - 24 bit (spazio di indirizzamento fisico $2^{24} = 16\text{M}$)
 - 512 pagine (2^9) da 32K (2^{15})
 - All'interno:
 - 32 bit

Architettura del processore 68000

- **Parallelismo della memoria:**
 - Parole di 16 bit, ognuna costituita da due byte con indirizzi distinti (memoria byte addressable, ovvero indirizzabile a livello di byte)
- **Convenzioni della memoria:**
 - Una parola deve essere allineata ad un indirizzo pari (*even boundary*)
 - Convenzione big-endian

Architettura del processore 68000

- Modello di programmazione del 68000

