

# Corso di Calcolatori Elettronici I

## A.A. 2011-2012

---

# Architettura di un calcolatore: introduzione

Lezione 18

Prof. Antonio Pescapè

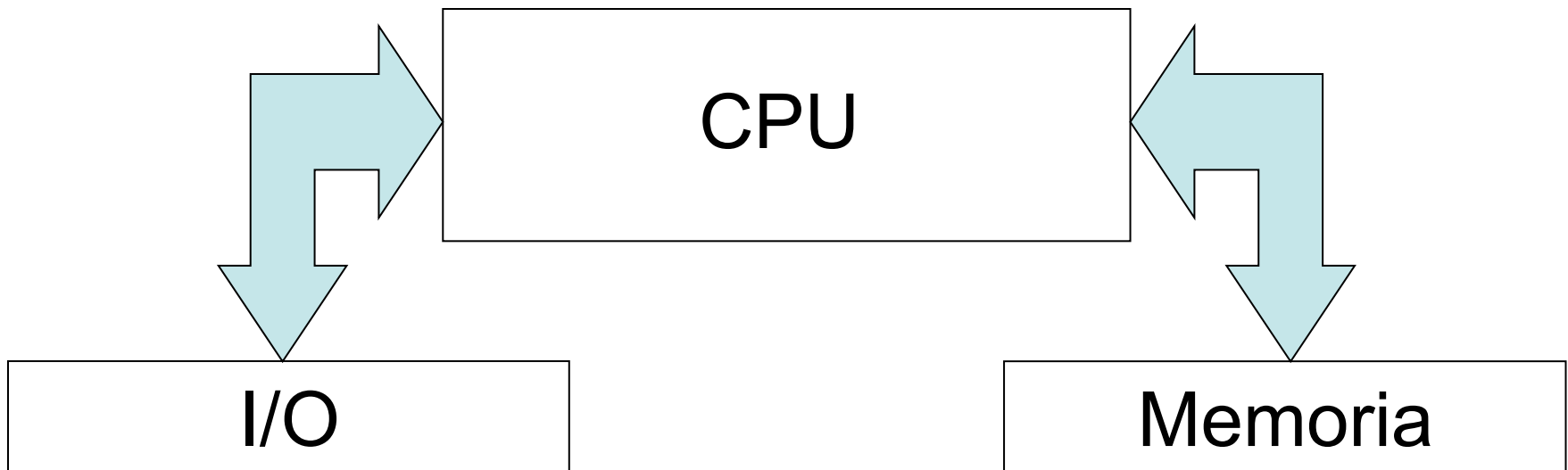
Università degli Studi di Napoli Federico II  
Facoltà di Ingegneria  
Corso di Laurea in Ingegneria Informatica



# Calcolatore: sottosistemi

---

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



# Calcolatore: organizzazione a bus

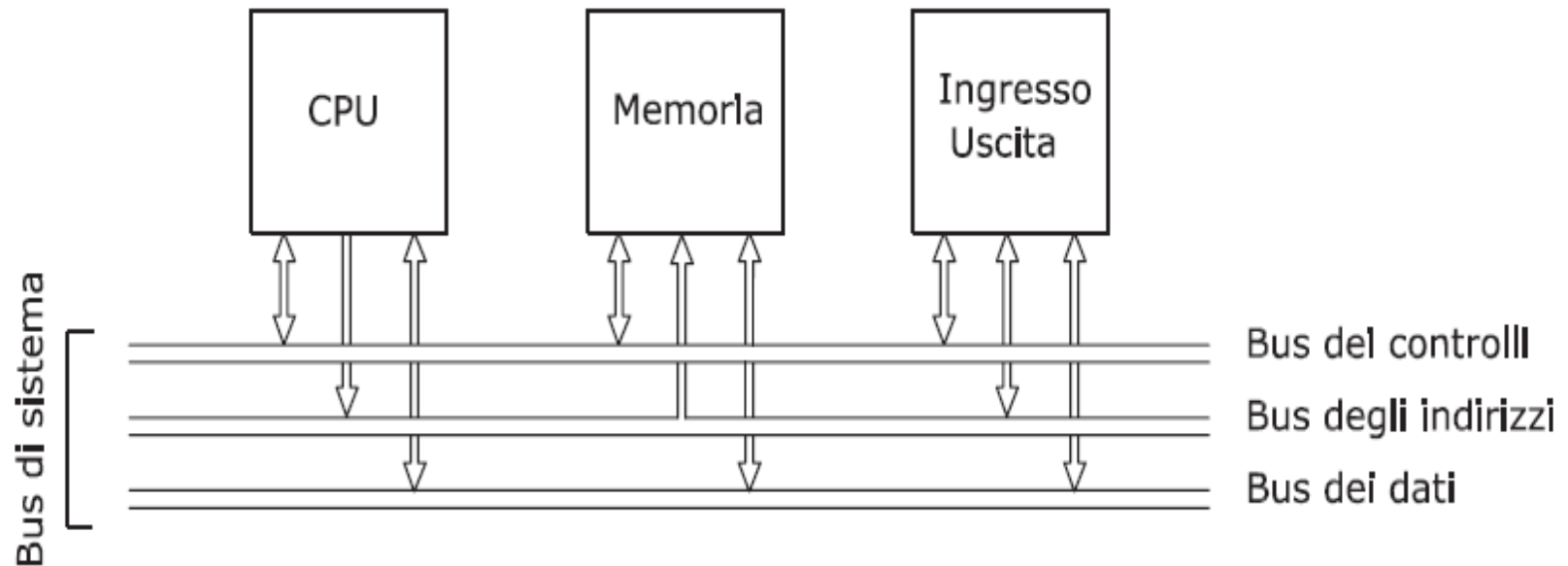


Figura 5.2 - Organizzazione a bus. Un bus è semplicemente un percorso che collega tutte le parti che si affacciano su di esso. La figura mostra che il bus di sistema si compone di tre sotto-bus.

# Il processore o CPU

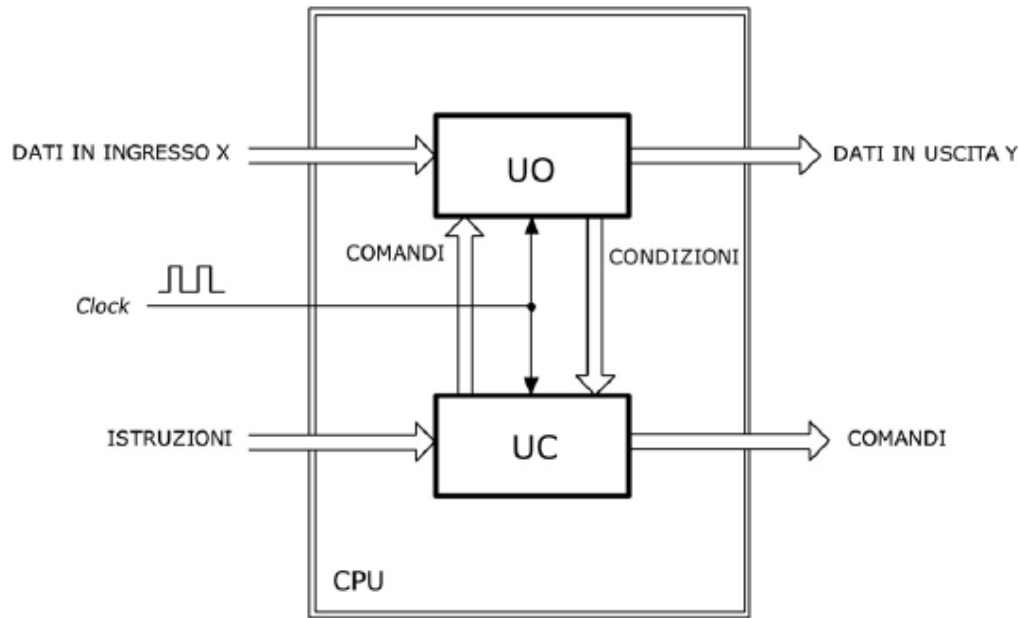
---



Processore o CPU

---

# CPU: struttura interna



**Figura 5.8** - Struttura generale della CPU. Lo schema mette in evidenza due parti: l'unità operativa (UO) e l'unità di controllo (UC). Lo schema mostra anche i flussi informativi tra UC e UO e tra l'intera CPU e l'esterno [LP86].

Il funzionamento della CPU è scandito dal clock.

# CPU: struttura interna (2)

---

- Componenti fondamentali del processore:
  - Unità di controllo
    - registro Program Counter (PC) o Prossima Istruzione
    - Instruction Register o registro di decodifica (IR o D)
    - registri di Macchina
  - Unità aritmetico-logica (ALU)
  - Sezione di Collegamento con la memoria
    - registro degli indirizzi di memoria o Memory Address Register MAR
    - registro di transito dei dati dalla memoria DTR o Memory Buffer MB
  - Sezione di Collegamento con Ingresso-Uscita
- Il *linguaggio macchina* di un processore è costituito dalla codifica in binario delle istruzioni eseguibili dal processore

# Registri della CPU

---

## ➤ **Registri interni**

- » Necessari al funzionamento del processore
- » Non direttamente visibili al programmatore
  - » *non appartengono al modello di programmazione*
  - » *Es. MAR, MDR/DTR, IR, ...*

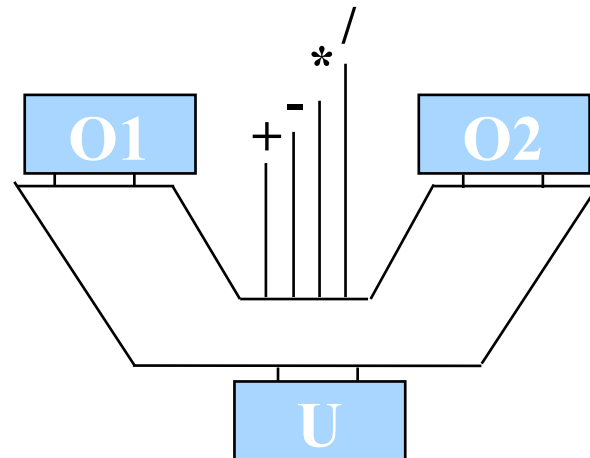
## ➤ **Registri di macchina**

- » Visibili al programmatore
    - » *appartengono al modello di programmazione*
      - Registri generali (R0, R1, Rn-1)
      - Registri speciali (PC, SR, ...)
-

# Unità Aritmetico-Logica (ALU)

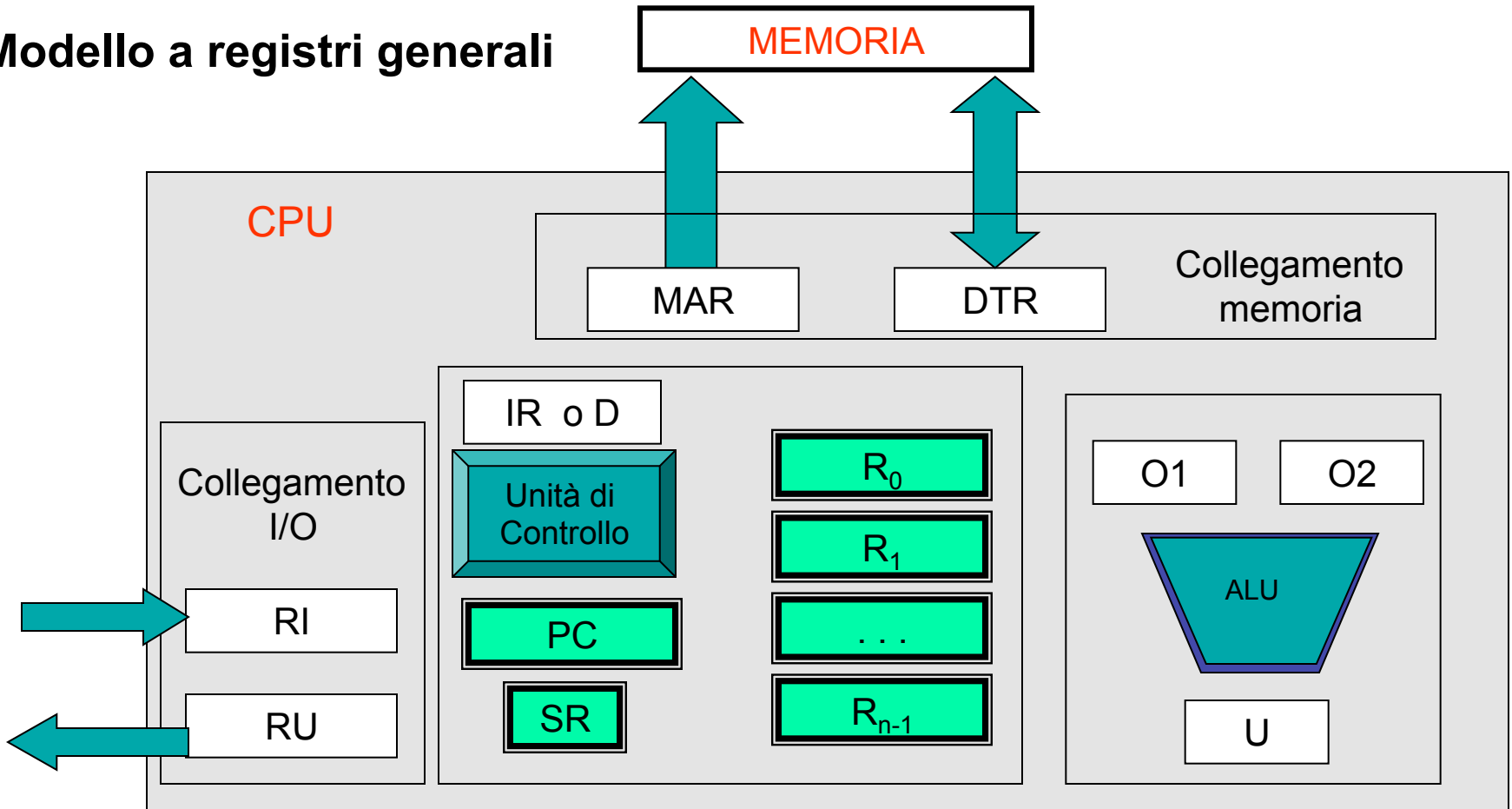
---

- 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 (O1, O2)
- La ALU esegue l' operazione e pone il risultato nel registro risultato (U); inoltre, altera il valore dei flag del registro di stato (SR) in funzione del risultato

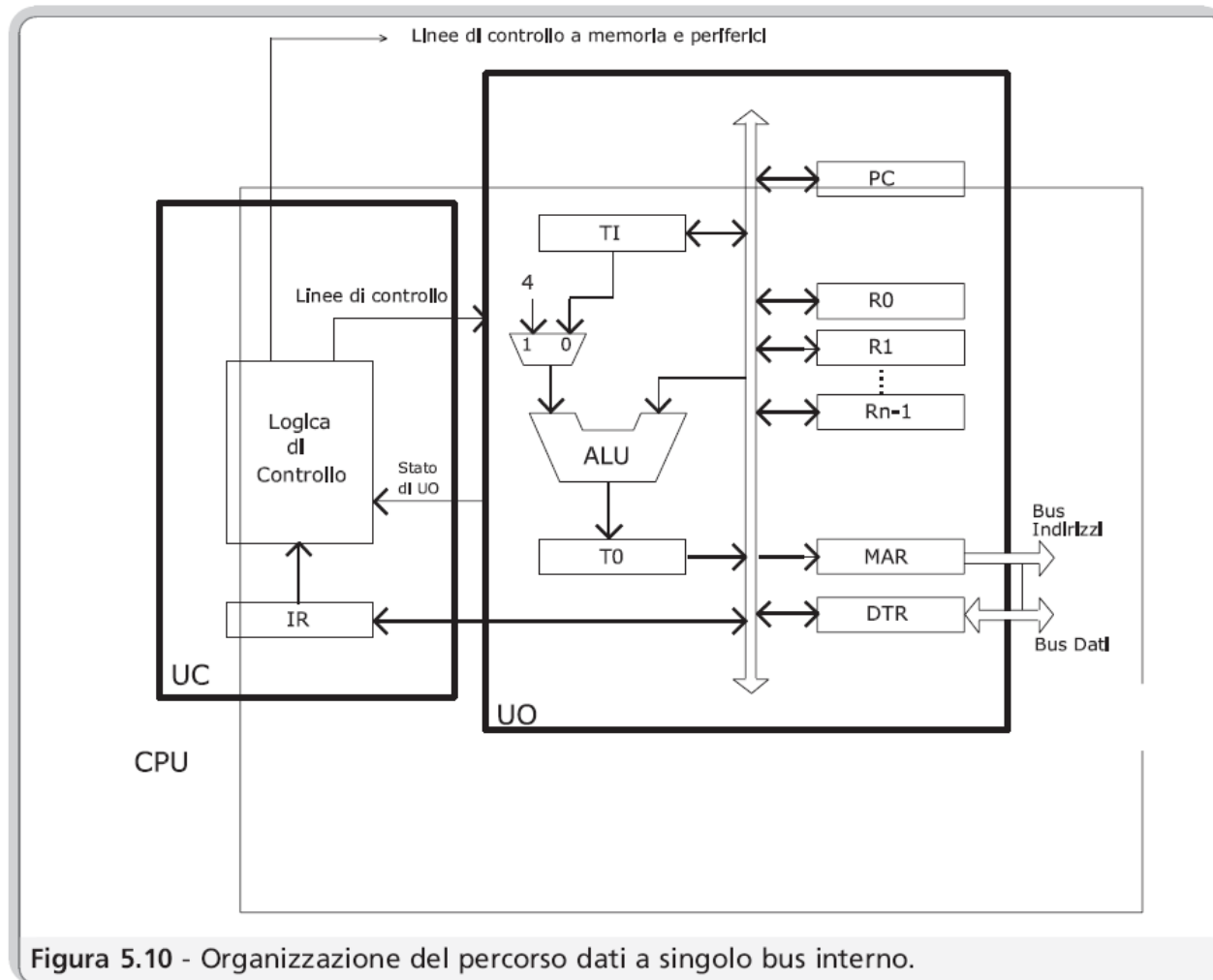


# Modello architetturale di un processore

## Modello a registri generali

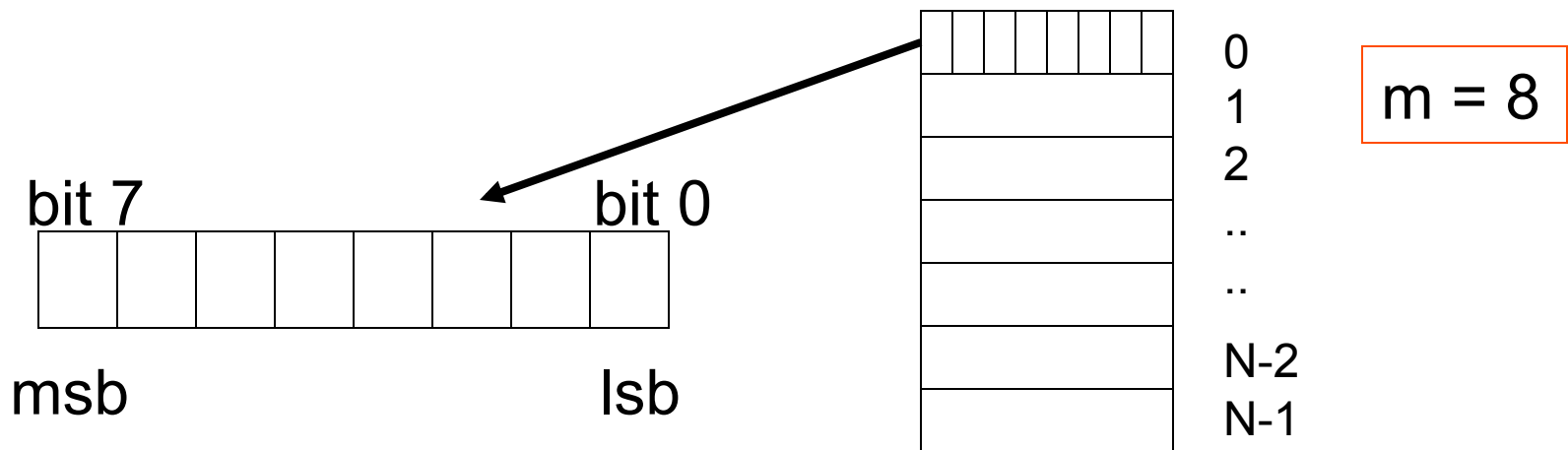


# CPU: struttura interna ad 1 bus



# La memoria centrale

- La memoria centrale di un computer è organizzata come un array di stringhe di bit di lunghezza  $m$ , dette *locazioni*
- Gli  $m$  bit di una locazione sono accessibili dal processore (in lettura/scrittura) mediante un' unica operazione
- Ogni locazione è individuata da un *indirizzo*, cioè un intero compreso tra 0 e  $N-1$ , con  $N = 2^k$ 
  - »  $[0, N-1] =$  SPAZIO DI INDIRIZZAMENTO
- La memoria centrale è *ad accesso casuale* (RAM) cioè il tempo di accesso non dipende dalla posizione del dato



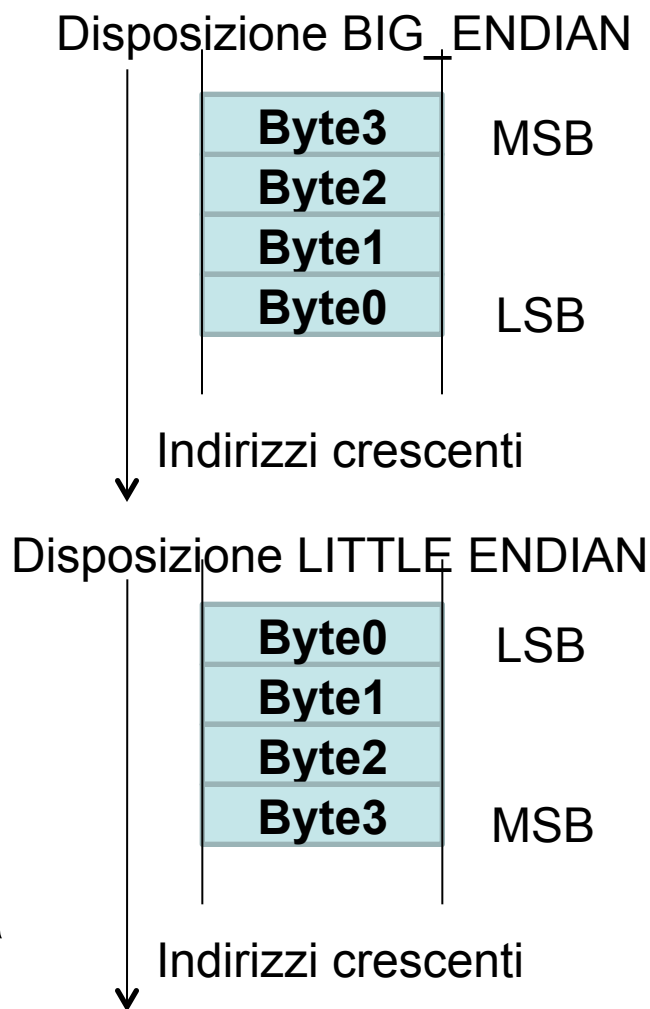
# Processori a carattere e processori a parola

---

- I processori “a carattere” accedono alla memoria con parallelismo 1 byte (8 bit)
    - prime CPU ad accumulatore
  - I processori “a parola” hanno la capacità di indirizzare ed accedere la memoria per unità (parole o *word*) di 16 bit, 32 bit o 64 bit
  - In questi sistemi (tranne pochissime eccezioni nel passato) l’unità indirizzabile di memoria (*locazione*) è ancora il byte
    - Si parla di sistemi a memoria *byte-addressable*: ogni byte ha il suo indirizzo
  - Terminologia Motorola 68000:  
word = 2 byte, longword = 4 byte
-

# Big-endian e little-endian

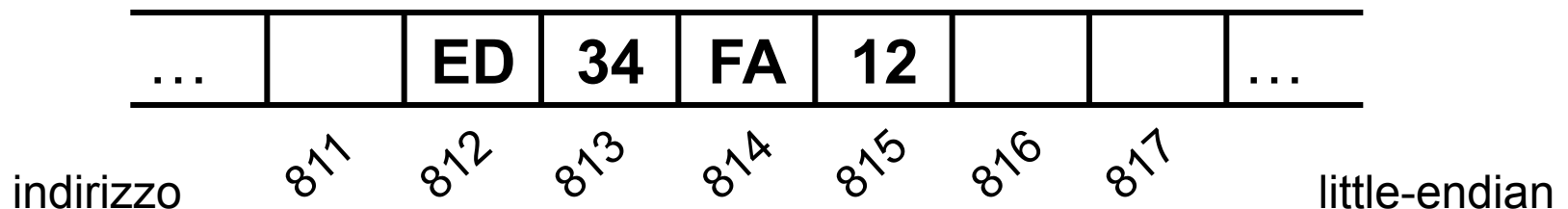
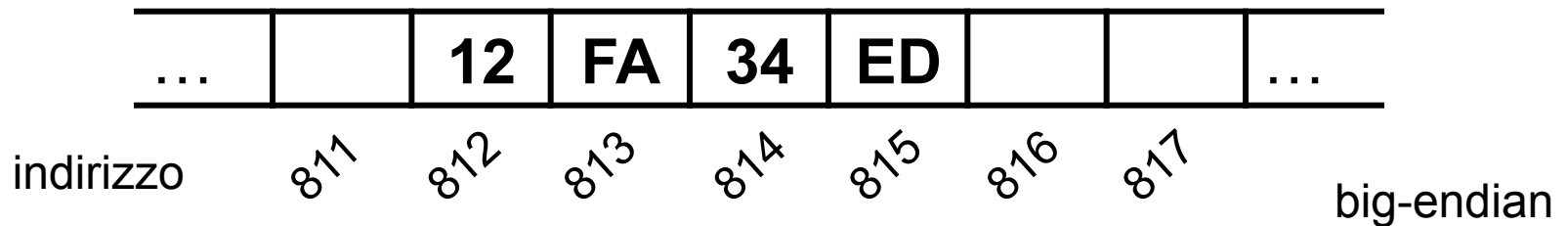
- I processori “a parola” possono disporre in memoria i byte che formano una parola in due modi
  - **Big-endian:**  
i byte sono disposti in memoria in modo che il più significativo MSB occupi la locazione di memoria di indirizzo minore, e poi via via gli altri, fino a quello meno significativo LSB che è collocato nella locazione di indirizzo maggiore
  - **Little-endian:** disposizione opposta
- Il processore Motorola 68000 usa la convenzione Big Endian



# Big-endian e little-endian: un esempio

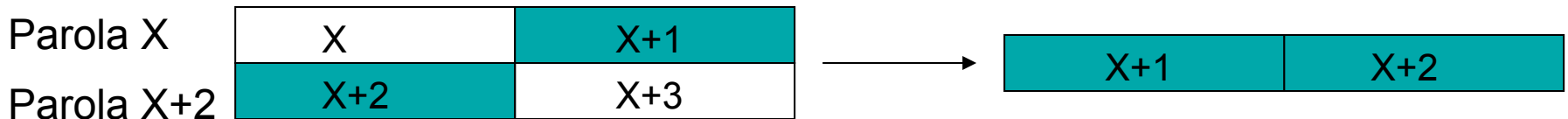
---

- Immaginiamo di avere un processore a parola, con parole di 32 bit (4 byte) e voler scrivere in memoria il valore intero (esadecimale) **\$12FA34ED** all'indirizzo **812**
- Le figure sottostanti illustrano il contenuto della memoria nei due casi big-endian e little-endian



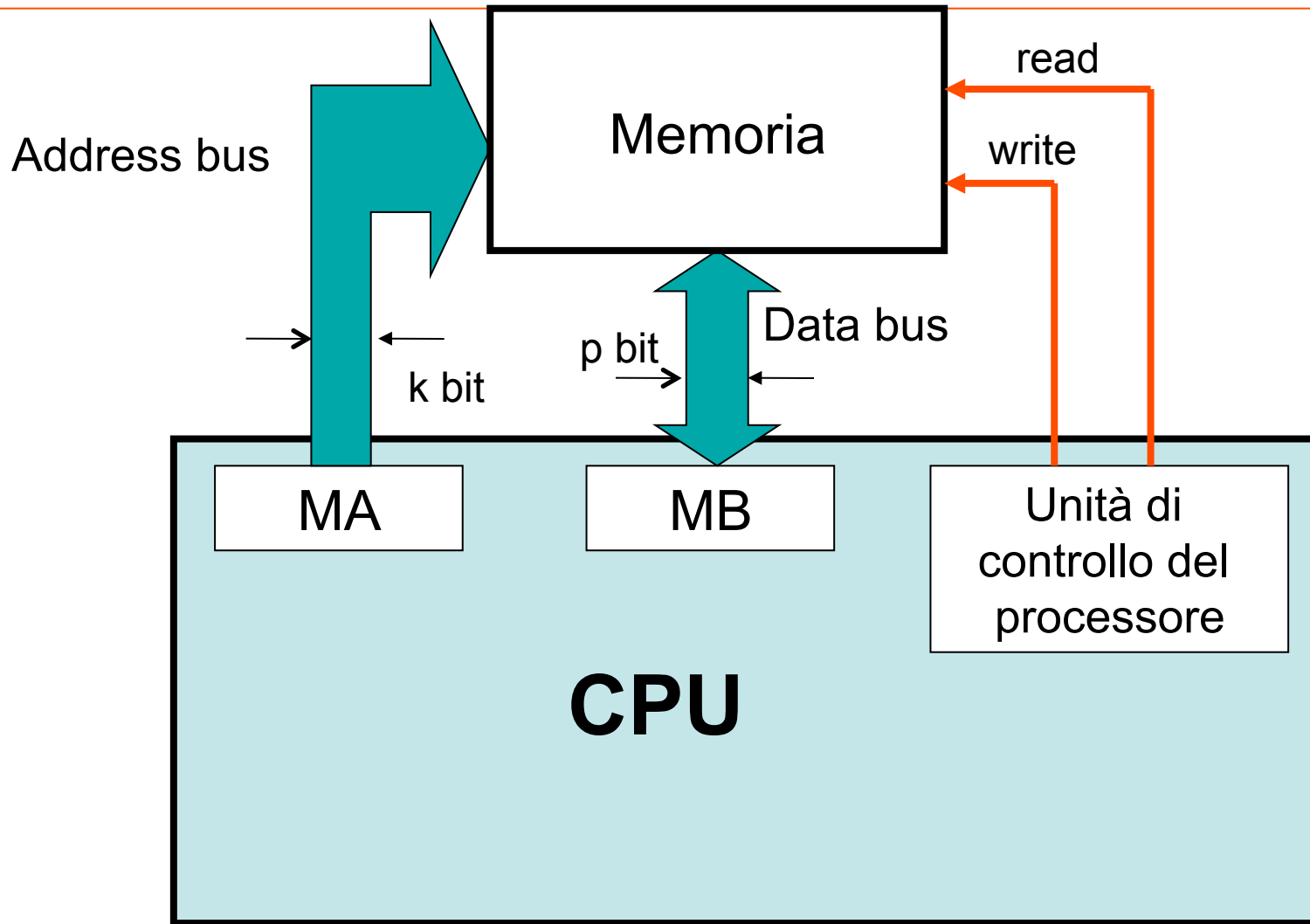
# Memoria: parole allineate e non

- 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
- Il processore 8086 consente l'uso di parole non allineate, cioè parole che iniziano ad un indirizzo dispari, in tal caso sono necessari 2 distinti accessi in memoria (uno per  $X+1$  e uno per  $X+2$ )
- Il processore 68000 NON consente l'accesso a parole non allineate



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

# Interazione processore-memoria



# Processore a registri generali

---

- Il processore dispone di un set di registri  $R_0, R_1, \dots, R_{N-1}$  utilizzabili indifferentemente dal programmatore
  - 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
    - 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*
-

# Algoritmo del Processore

---

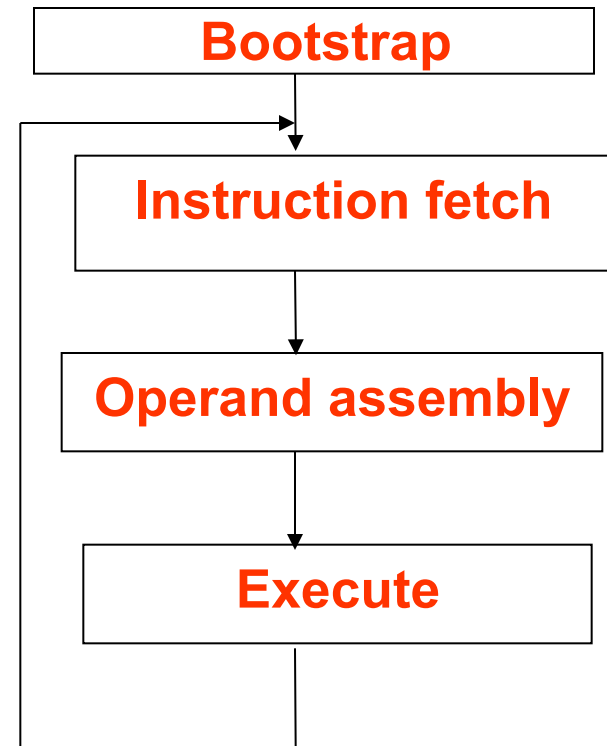
- **Prelievo dell'istruzione (Fetch)**
    - La CPU preleva dalla memoria l'istruzione il cui indirizzo è in PC
    - L'istruzione viene copiata nel registro IR
  - **Decodifica / prelievo degli operandi (Operand Assembly)**
    - L'unità di controllo esamina il contenuto di IR e ricava il tipo di operazione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

# Algoritmo del processore

---

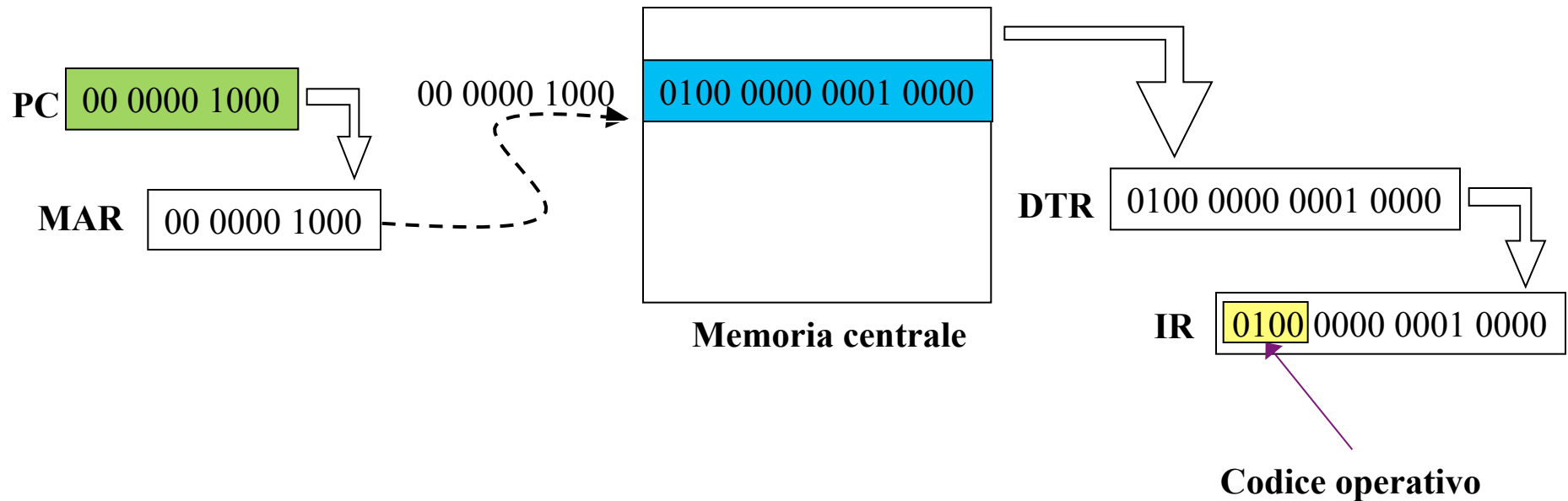
- L'unità di controllo opera in un ciclo infinito:
  1. Prelievo
  2. Preparazione degli operandi
  3. Esecuzione

**Nella fase di bootstrap il ciclo viene inizializzato; viene assegnato un valore iniziale opportuno a PC in modo da avviare l'esecuzione di un programma iniziale in ROM**



# Fase fetch

- $IR = M[PC]; PC = PC + k$



# Fase fetch: sottopassi

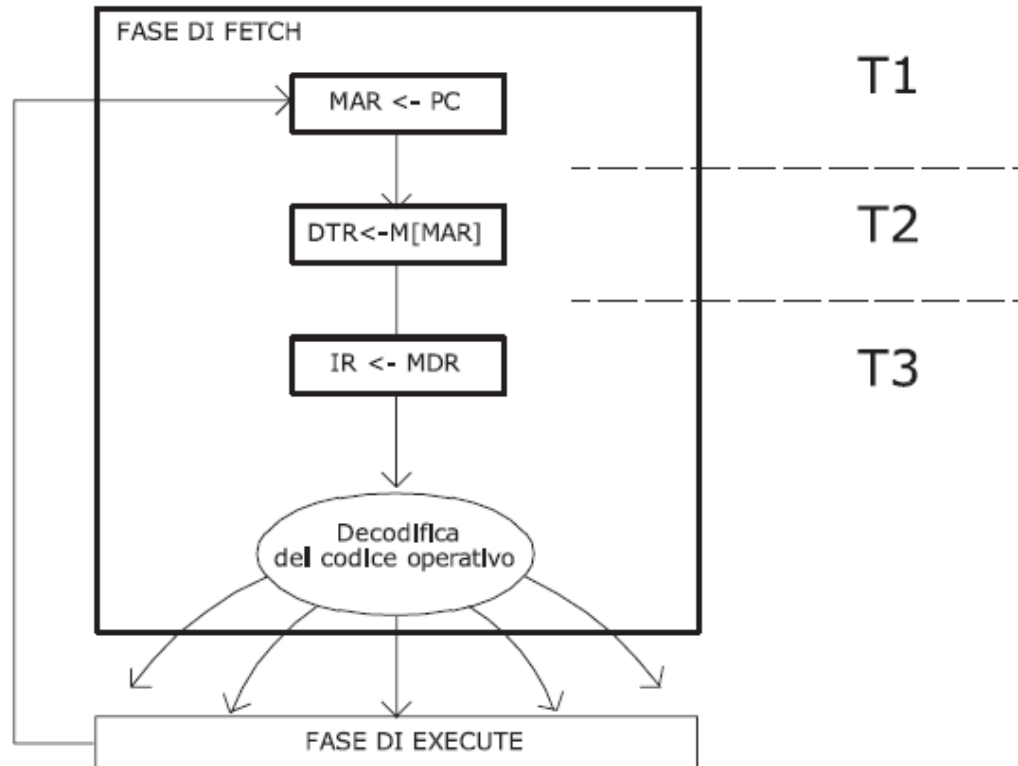
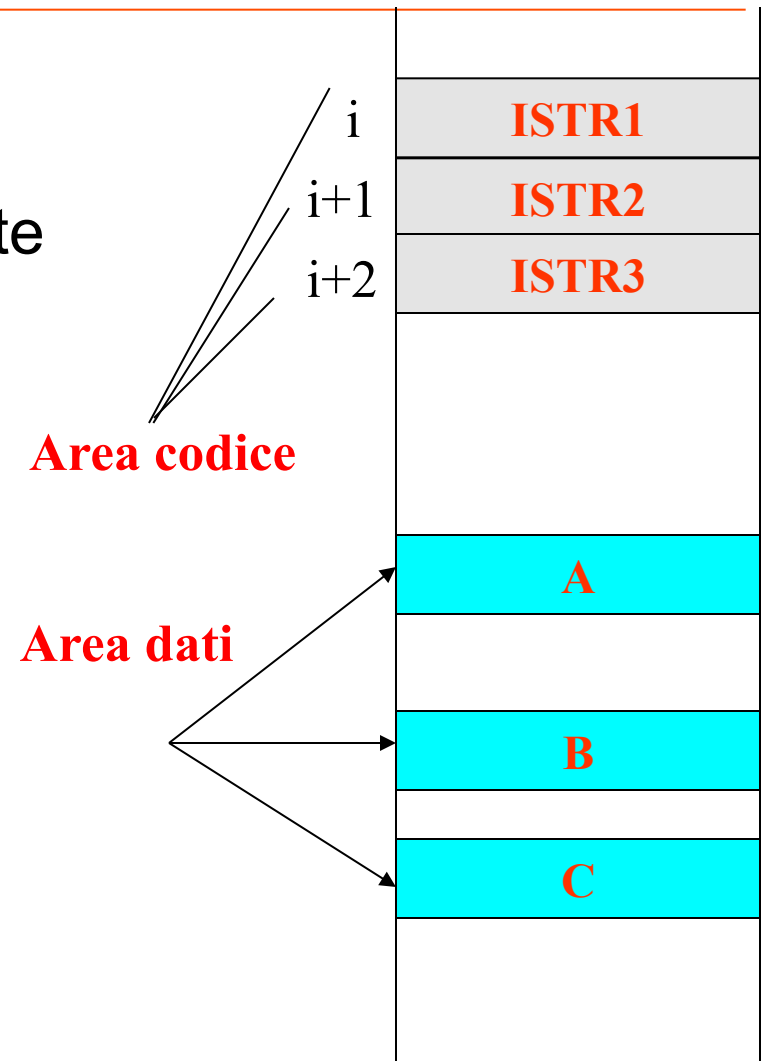


Figura 5.11 - Esplosione della fase di fetch.

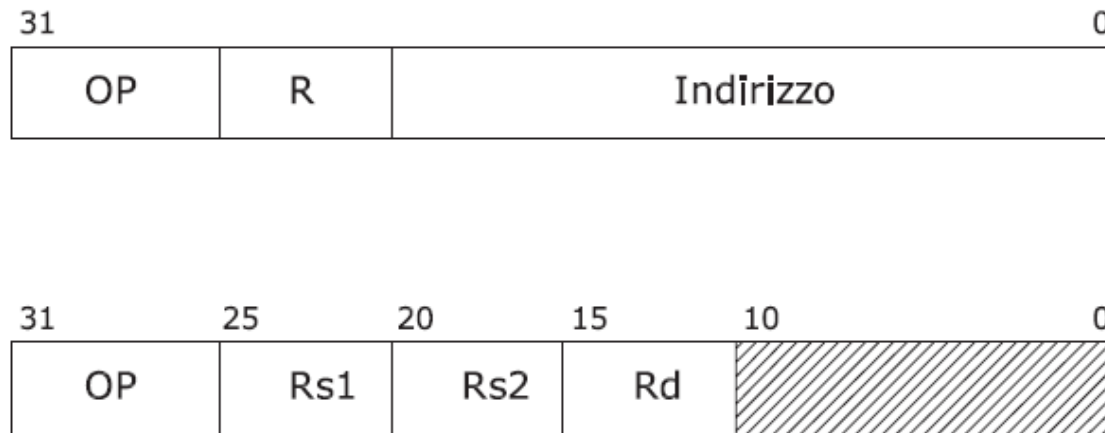
# Esecuzione sequenziale delle istruzioni

- Alla fine della fase fetch:
  - $PC = PC + k$
  - $k$  = lunghezza istruzioni in byte
- serve a far sì che PC punti all'istruzione posta subito dopo
- Esecuzione delle istruzioni in sequenza così come sono memorizzate
- Per cicli e figure di controllo (if-then, if-then-else, switch) occorrono istruzioni di salto



# Codifica delle istruzioni

ESEMPIO: una CPU con istruzioni a lunghezza fissa di 32 bit

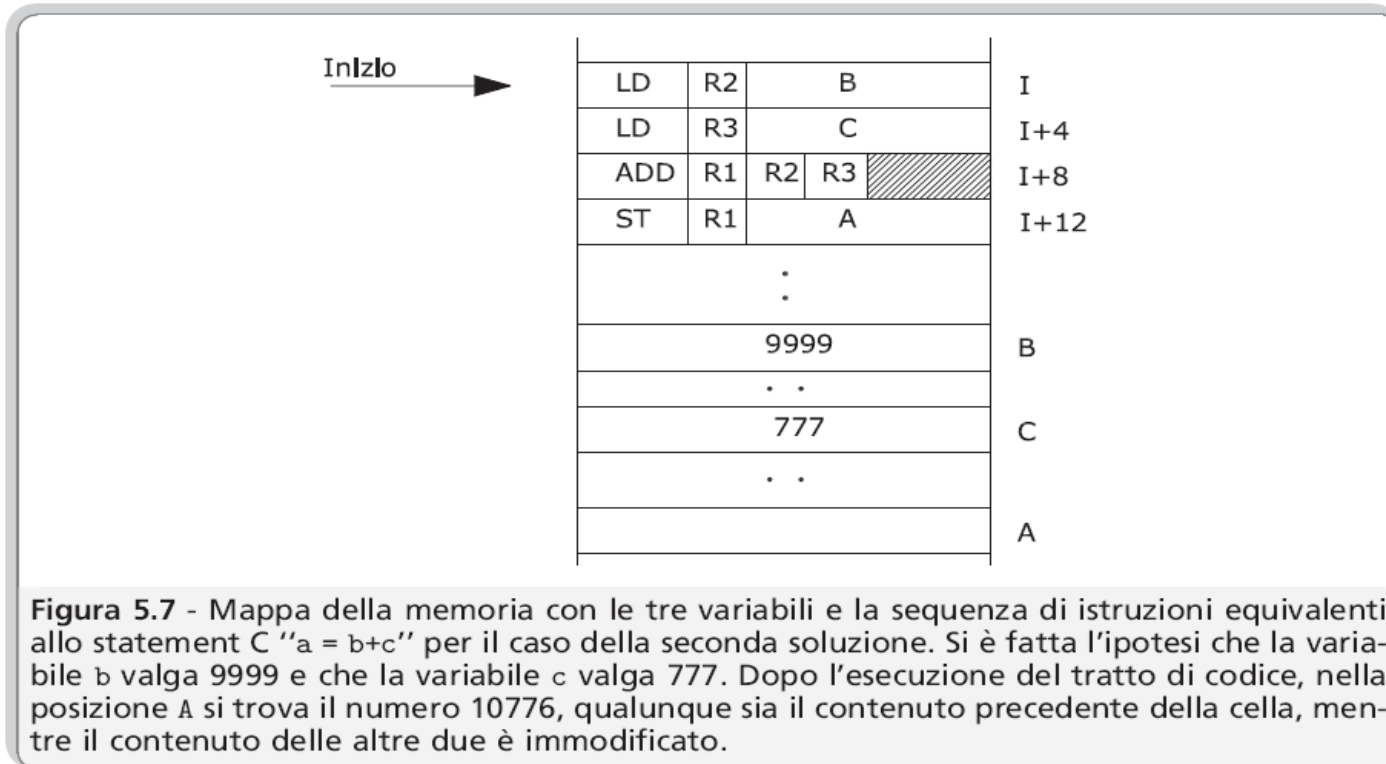


**Figura 5.6** - Due possibili formati per la codifica delle istruzioni di macchina su parole di 32 bit. Il campo OP contiene il codice di operazione ed è sempre presente; gli altri campi dipendono dal codice di operazione stesso. Il primo formato prevede due operandi: un registro della CPU e un indirizzo della memoria. Il secondo formato prevede tre operandi, tutti registri della CPU; il campo tratteggiato è inutilizzato. La numerazione dei bit all'interno delle parole adotta la convenzione di assegnare il numero 0 al meno significativo.

LD R1, Var ;  $R1 \leftarrow M[\text{Var}]$

ADD R1, R2, R3 ;  $R1 \leftarrow R2 + R3$

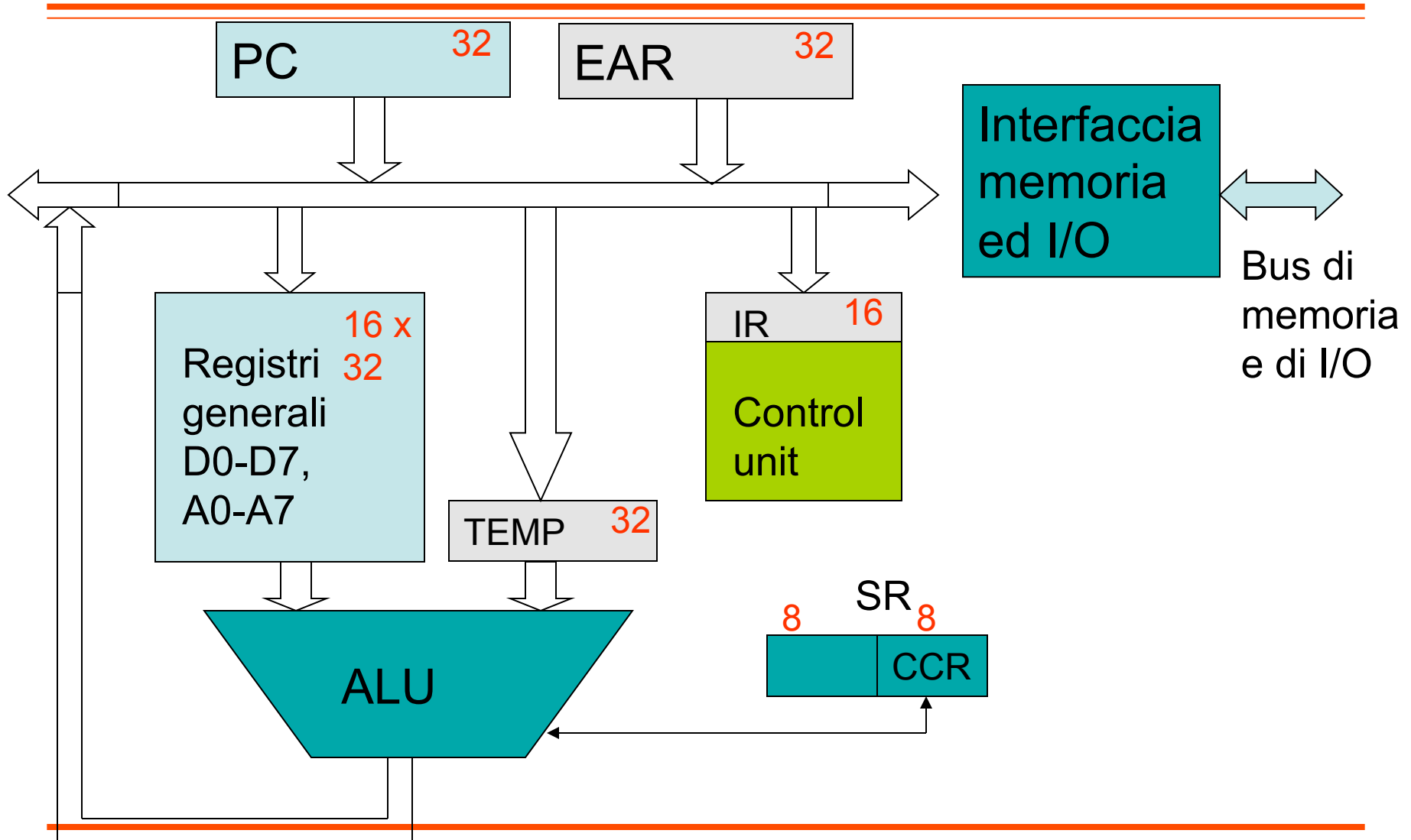
# Sequenze di istruzioni in memoria



Lo statement  $a = b + c$  si traduce come:

LD R2, B	;B indirizzo a cui è allocata la parola b
LD R3, C	;C indirizzo a cui è allocata la parola c
ADD R1, R2, R3	
ST A, R1	;A indirizzo a cui è allocata la parola a

# Architettura del processore MC68000



# Caratteristiche del processore MC68000

---

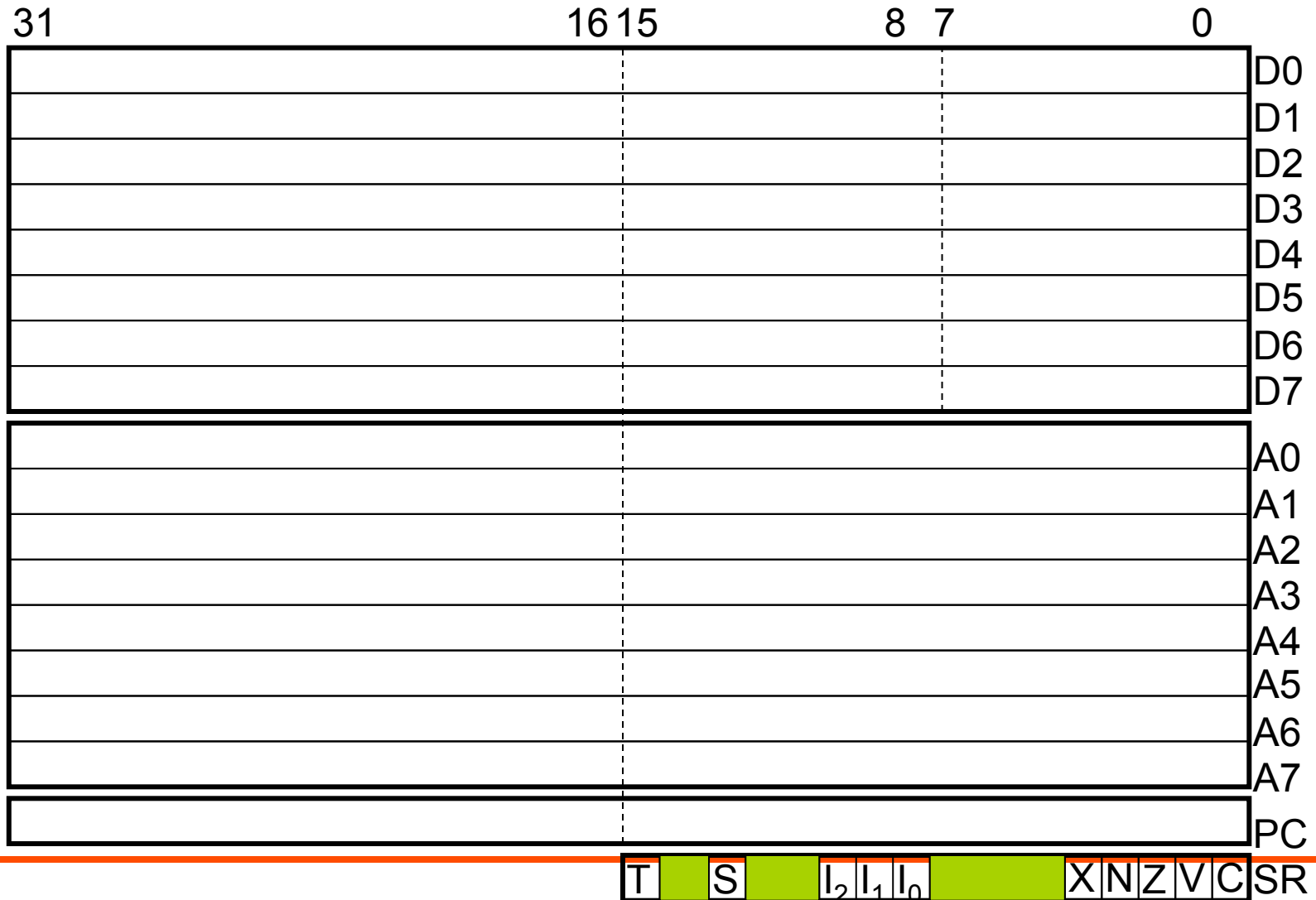
- Dati:
    - All' esterno:  
parola di 16 bit (16 pin per i dati)
    - All' interno:  
registri di 32 bit
  - Indirizzi:
    - All' esterno:  
24 bit (spazio di indirizzamento fisico  $2^{24} = 16\text{M}$ )
      - 512 pagine ( $2^9$ ) da 32K ( $2^{15}$ )
    - All' interno:  
32 bit
-

# Caratteristiche del processore MC68000

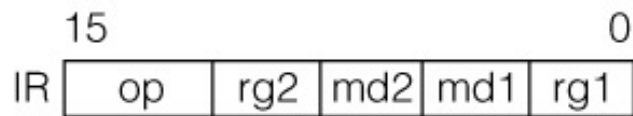
---

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

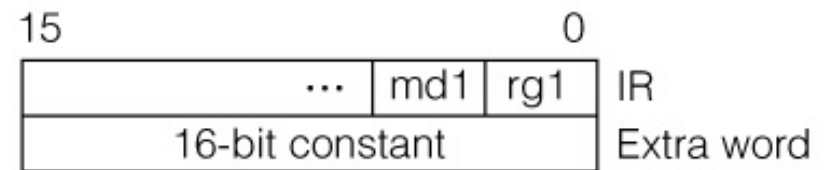
# Modello di programmazione del MC68000



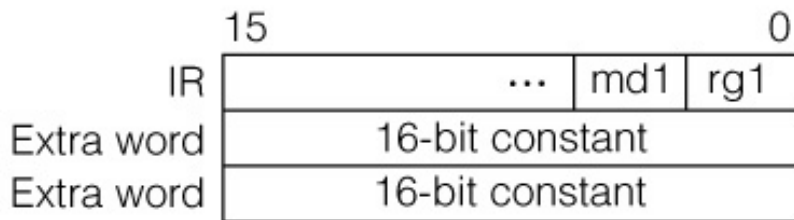
# Codifica istruzioni MC68000



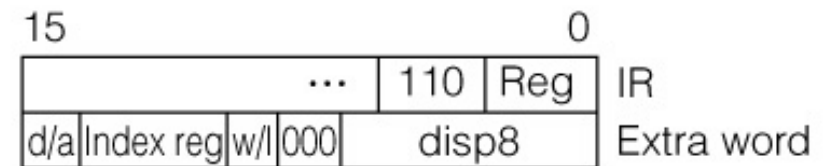
**(a) A 1-word move instruction**



**(b) A 2-word instruction**



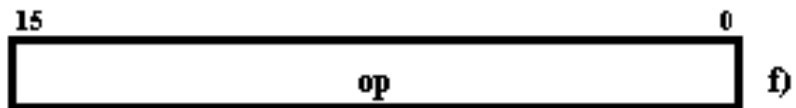
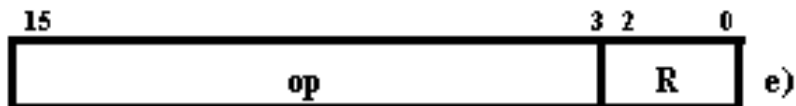
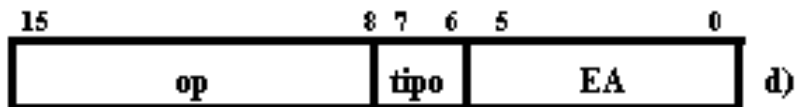
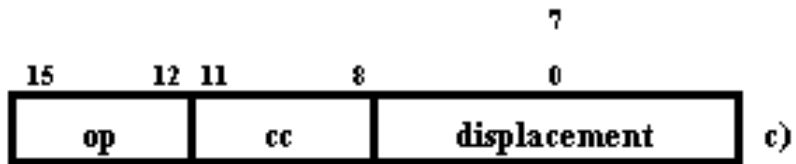
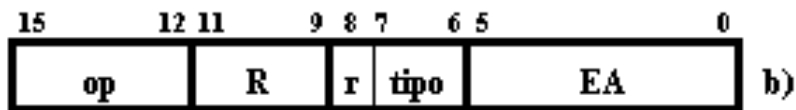
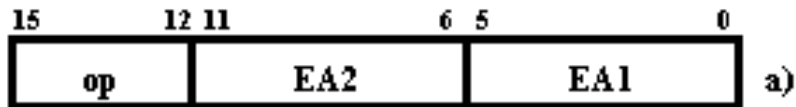
**(c) A 3-word instruction**



**(d) Instruction with indexed address**

# Codifica istruzioni MC68000 (2)

Si analizza qui solo la struttura della prima word (16 bit) del codice di una istruzione, detta **OPCODE WORD**



op= codice operativo

R =indirizzo di registro D oppure A

tipo= tipo di operando (B,W,L)

displacement =indirizzo di salto relativo

EA = Effective Address (cfr. § 4)

r =il registro è origine o destinazione

cc= codice di condizione