

# Corso di Calcolatori Elettronici I

---

## Architettura di un calcolatore: Introduzione parte 2

**Prof. Roberto Canonico**



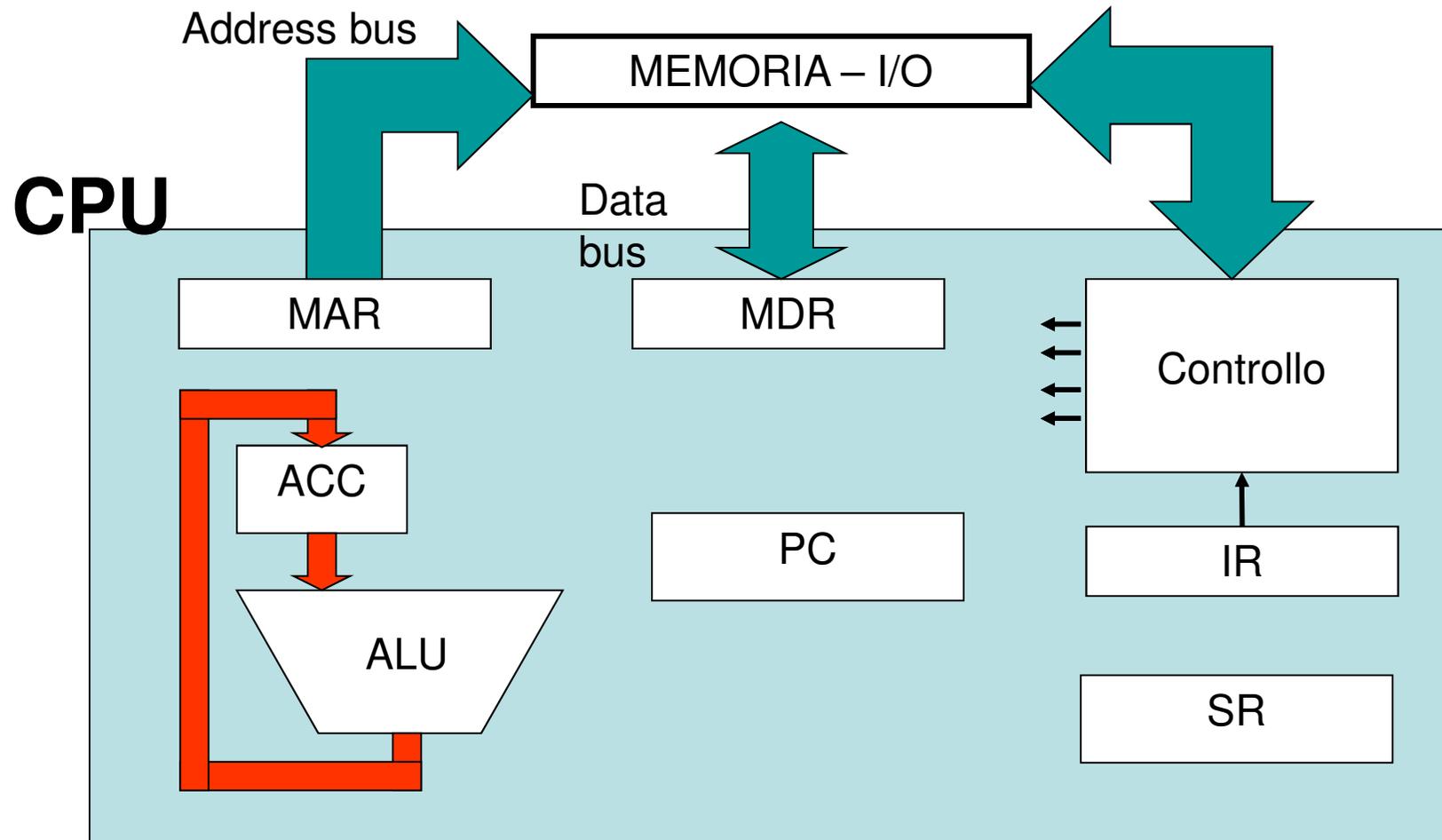
Università degli Studi di Napoli Federico II  
Dipartimento di Ingegneria Elettrica  
e delle Tecnologie dell'Informazione  
Corso di Laurea in Ingegneria Informatica  
Corso di Laurea in Ingegneria dell'Automazione

---

# Modello architetturale di un processore: modello ad accumulatore

---

---



# Processori ad accumulatore

---

- In un processore ad accumulatore tutte le istruzioni aritmetiche, logiche e di confronto hanno un operando in memoria ed un altro (riferito implicitamente) contenuto in un registro interno del processore detto *accumulatore*
  - Esempio: per realizzare  $[x]+[y] \rightarrow z$  con una macchina ad accumulatore (es. Motorola 6809) occorre eseguire una sequenza di istruzioni del tipo

|        |                                       |
|--------|---------------------------------------|
| LDA x  | [x] → accumulatore (Istruzione LOAD)  |
| ADDA y | [y]+[accumulatore] → accumulatore     |
| STA z  | [accumulatore] → z (Istruzione STORE) |
  - Dimensione e velocità di esecuzione dei programmi penalizzate dal fatto che tutte le istruzioni devono indirizzare un dato in memoria
-

# Esempio: $y = a * b + c * d$

---

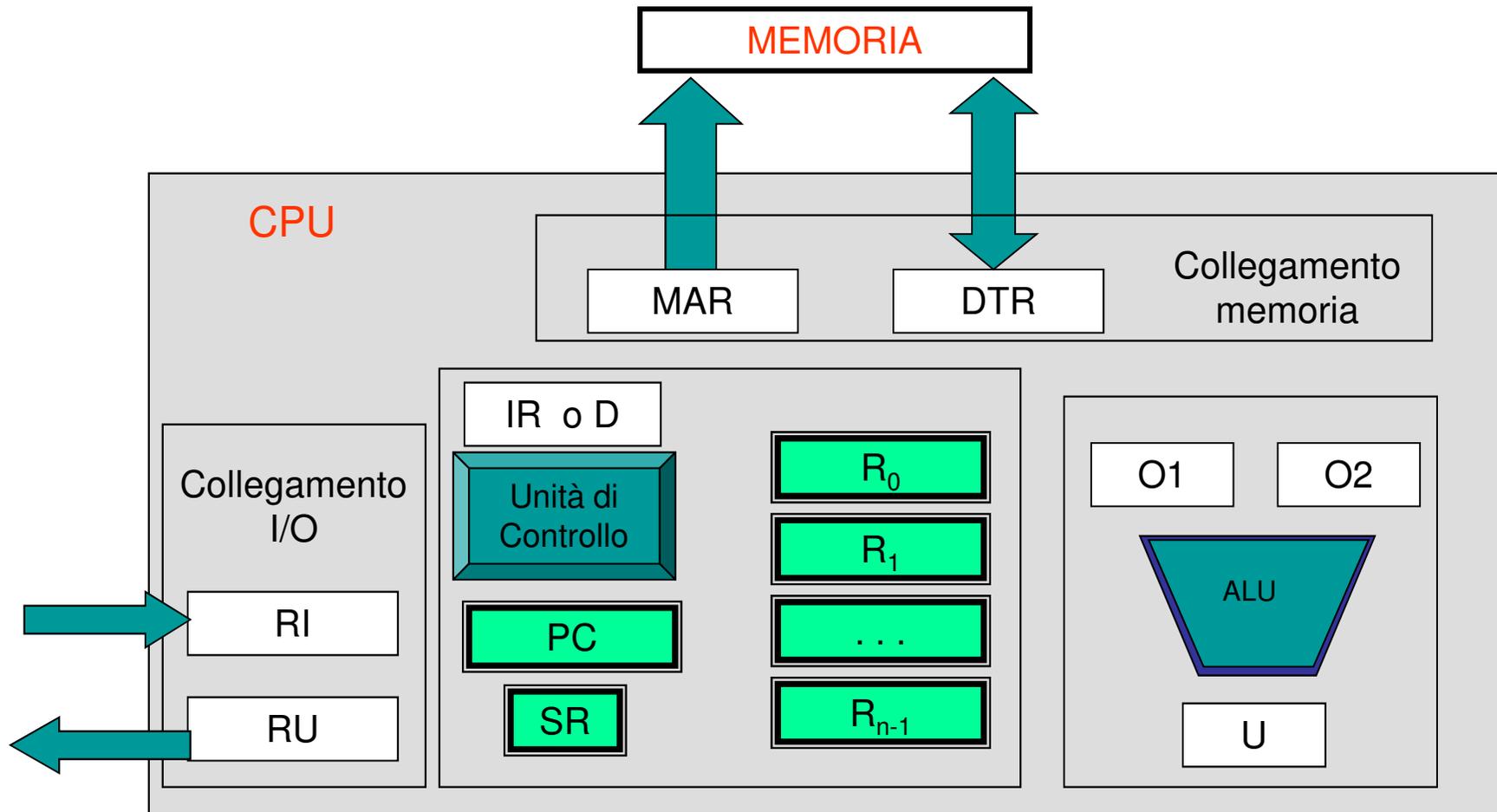
|      |   |                                       |
|------|---|---------------------------------------|
| LDA  | a | [a] → accumulatore (Istruzione LOAD)  |
| MULU | b | [b]*[accumulatore] → accumulatore     |
| STA  | t | [accumulatore] → t (Istruzione STORE) |
| LDA  | c | [c] → accumulatore (Istruzione LOAD)  |
| MULU | d | [d]*[accumulatore] → accumulatore     |
| ADDA | t | [t]+[accumulatore] → accumulatore     |
| STA  | y | [accumulatore] → y (Istruzione STORE) |

---

# Modello architetturale di un processore: modello a registri generali

---

---



# Processore a registri generali

---

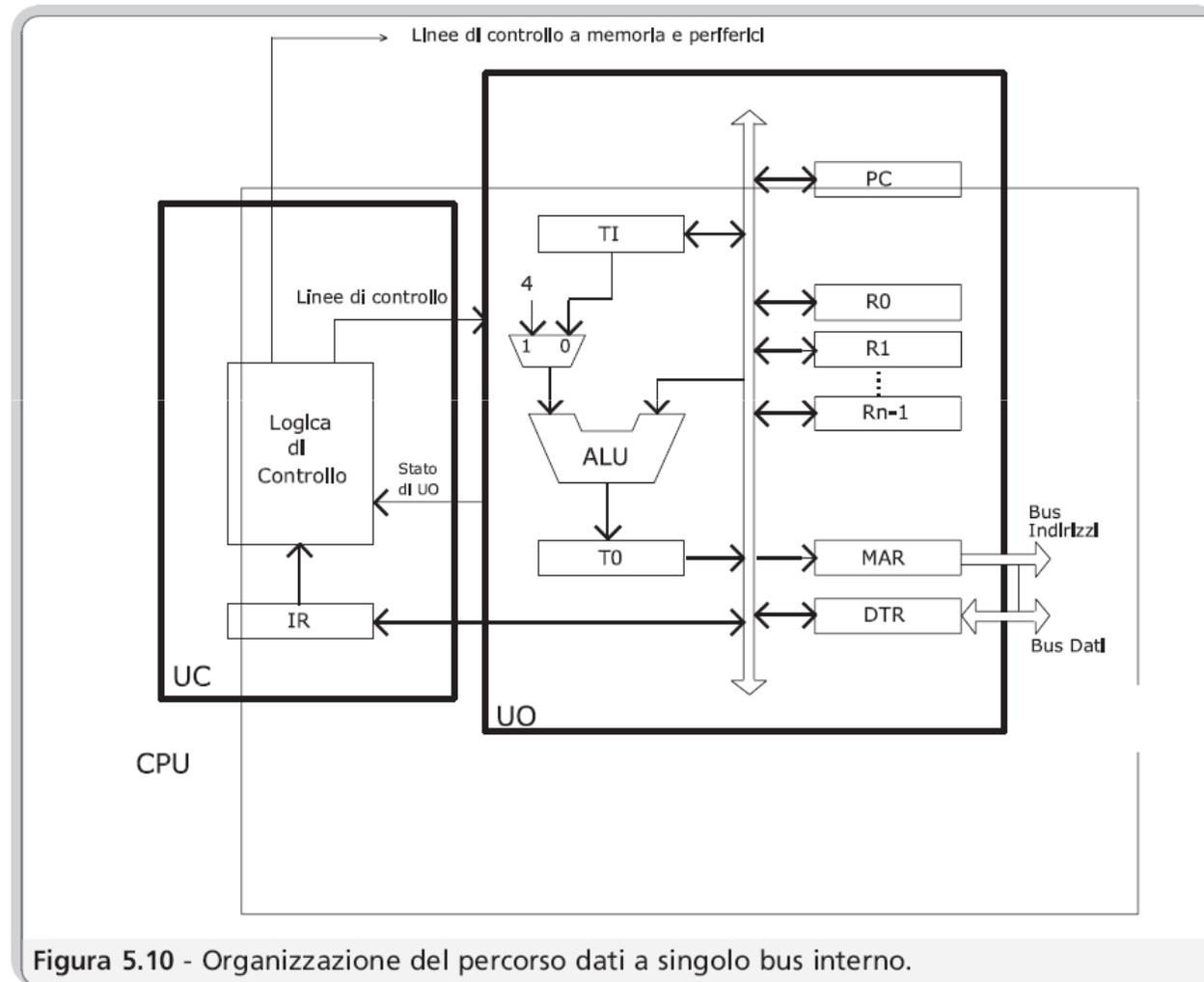
- Il processore dispone di un set di registri  $R_0, R_1, \dots, R_{N-1}$  utilizzabili indifferentemente dal programmatore
  - Le istruzioni che operano su registri sono più veloci di quelle che operano su locazioni di memoria
  - Il programmatore può utilizzare i registri del processore per memorizzare 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*
-

# Architetture a stack

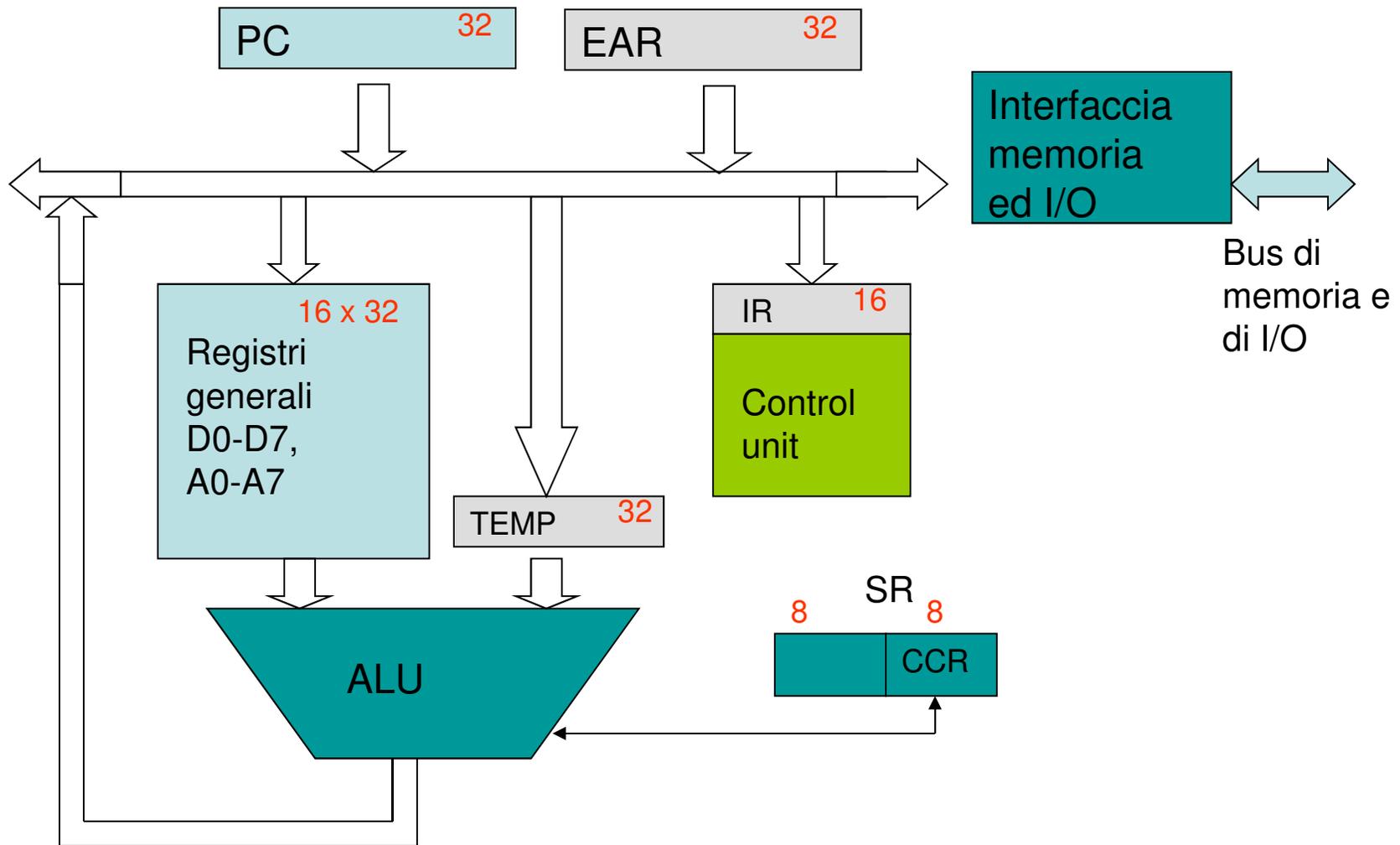
---

- In un'architettura a stack si impiega una struttura di memoria organizzata a stack
  - Lo stack può essere realizzato all'interno della CPU e/o utilizzando memorie esterne
  - Gli operandi devono essere memorizzati sullo stack ed il risultato di una qualsiasi operazione (eseguita sullo stack) sostituisce successivamente gli operandi
  - Di solito il valore in cima allo stack viene «duplicato» all'interno della CPU
-

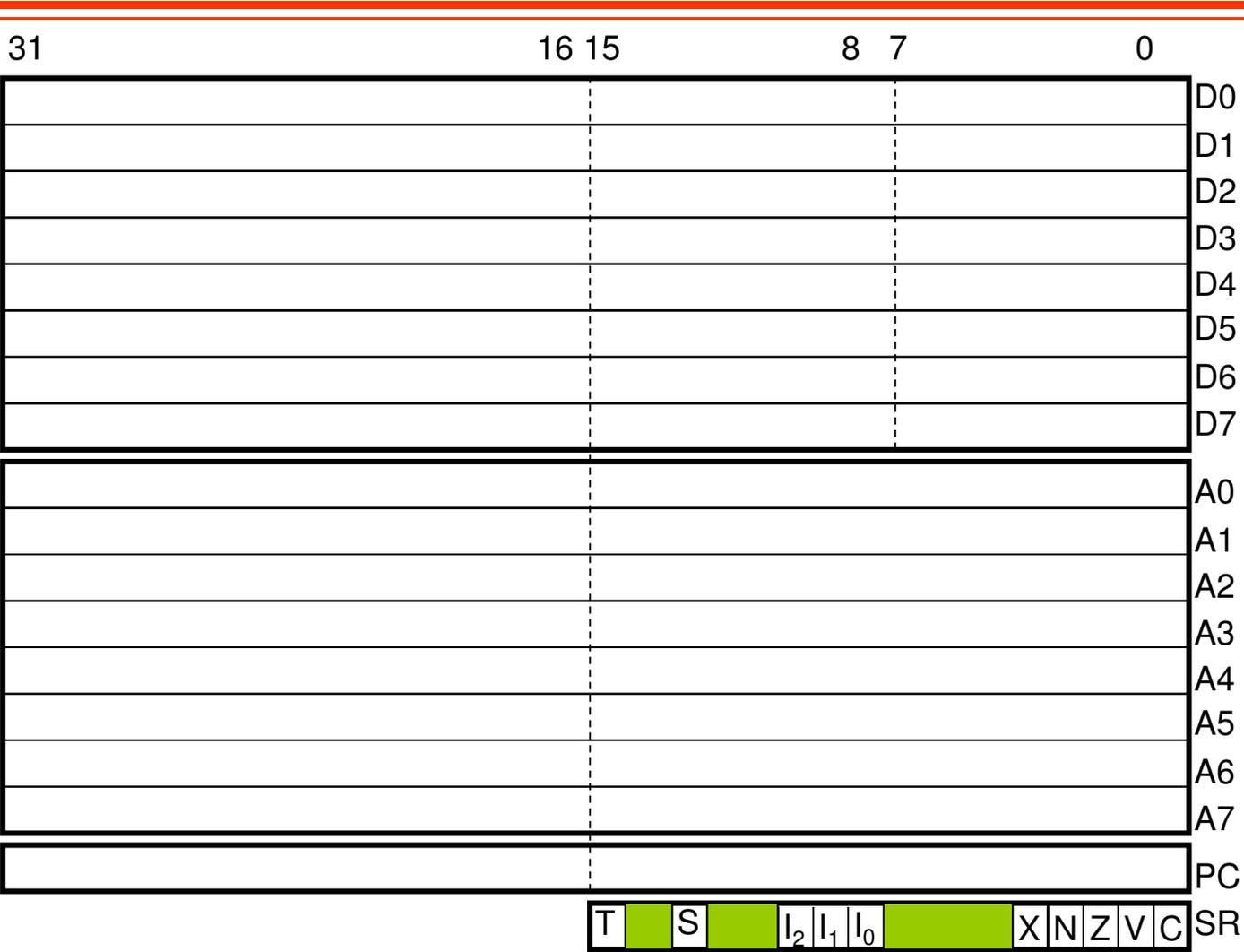
# CPU: struttura interna ad 1 bus



# Architettura del processore MC68000



# Modello di programmazione del 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
-

# Esempio: $y=a*b+c*d$ (MC68000)

---

|              |                |
|--------------|----------------|
| MOVE a, D0   | [a] → D0       |
| MULU b, D0   | [b]*[D0] → D0  |
| MOVE c, D1   | [c] → D1       |
| MULU d, D1   | [d]*[D1] → D1  |
| ADD.L D0, D1 | [D0]+[D1] → D1 |
| MOVE.L D1, y | [D1] → y       |

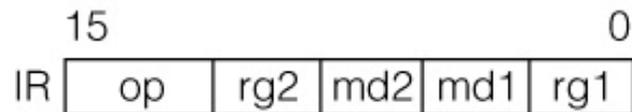
Rispetto all'esempio del processore ad accumulatore, il programma:

- necessita di una istruzione in meno (STA a)
- fa un accesso in meno in scrittura sulla memoria (STA a)
- fa un accesso in meno in lettura sulla memoria (ADDA t)

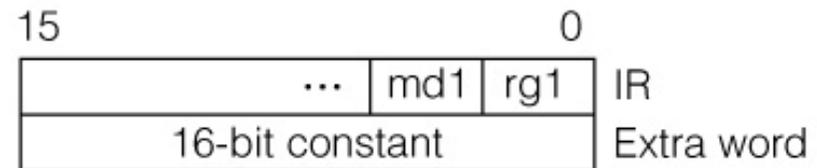
Il programma illustra i vantaggi di un'architettura con più registri generali rispetto ad un processore con singolo accumulatore

---

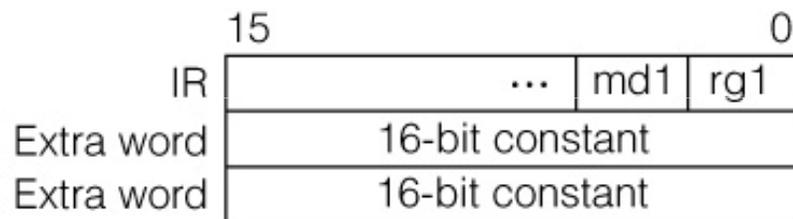
# Codifica istruzioni MC68000



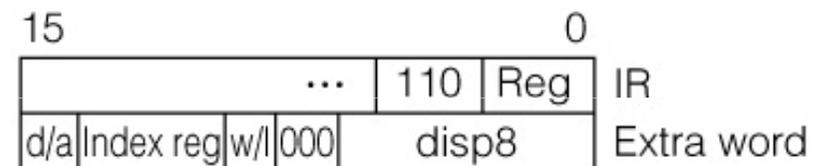
**(a) A 1-word move instruction**



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



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

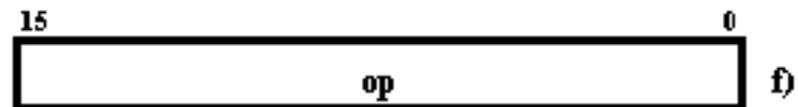
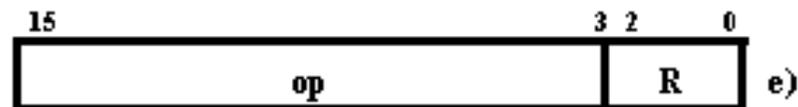
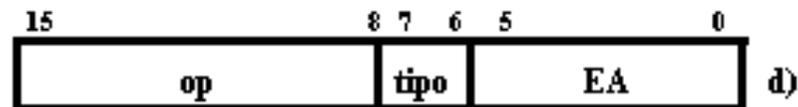
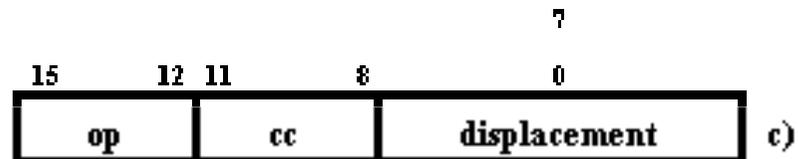


**(d) Instruction with indexed address**

**Codifica a lunghezza variabile multipla di 2 byte:  
opcode word + extra word(s)**

# 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

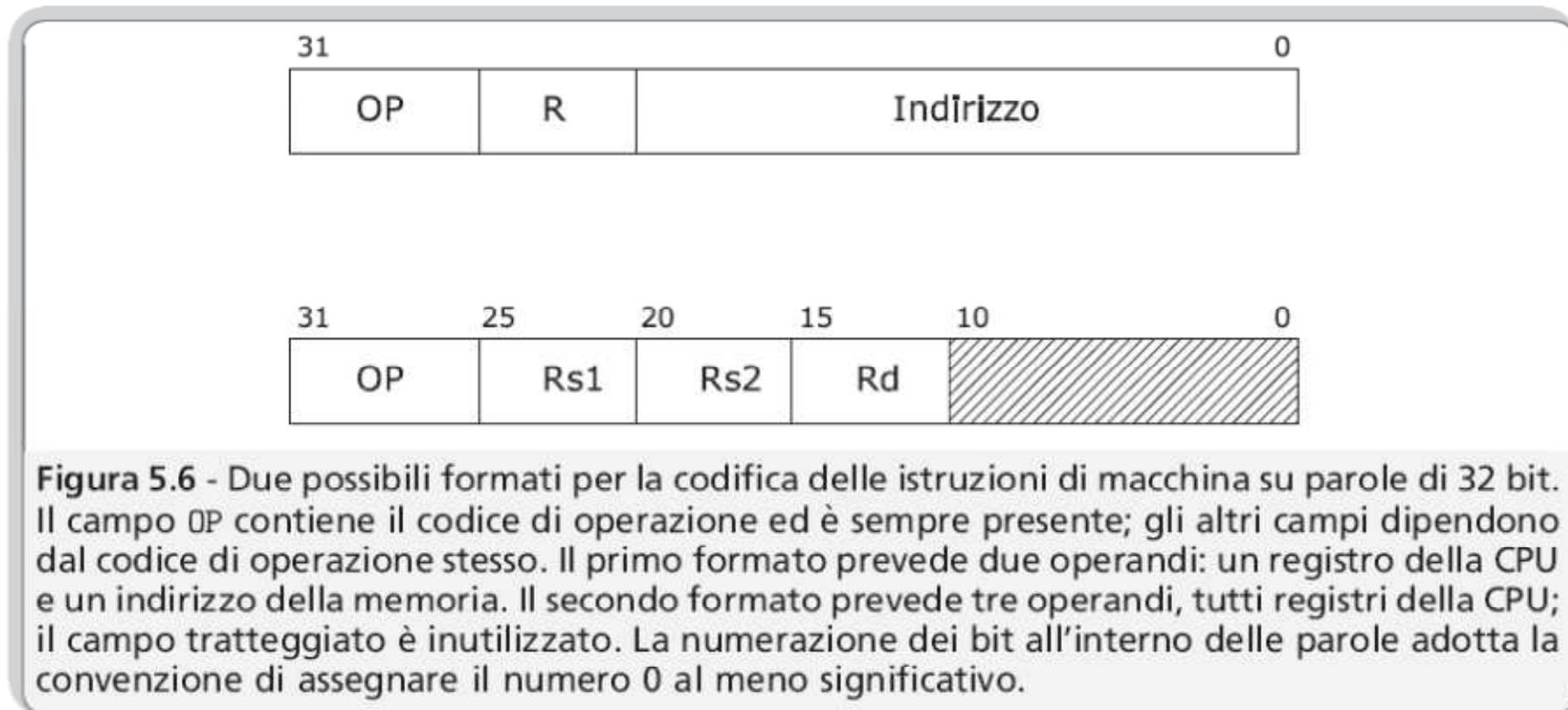
EA = Effective Address (cfr. § 4)

r =il registro è origine o destinazione

cc= codice di condizione

# Codifica delle istruzioni di un processore in stile RISC

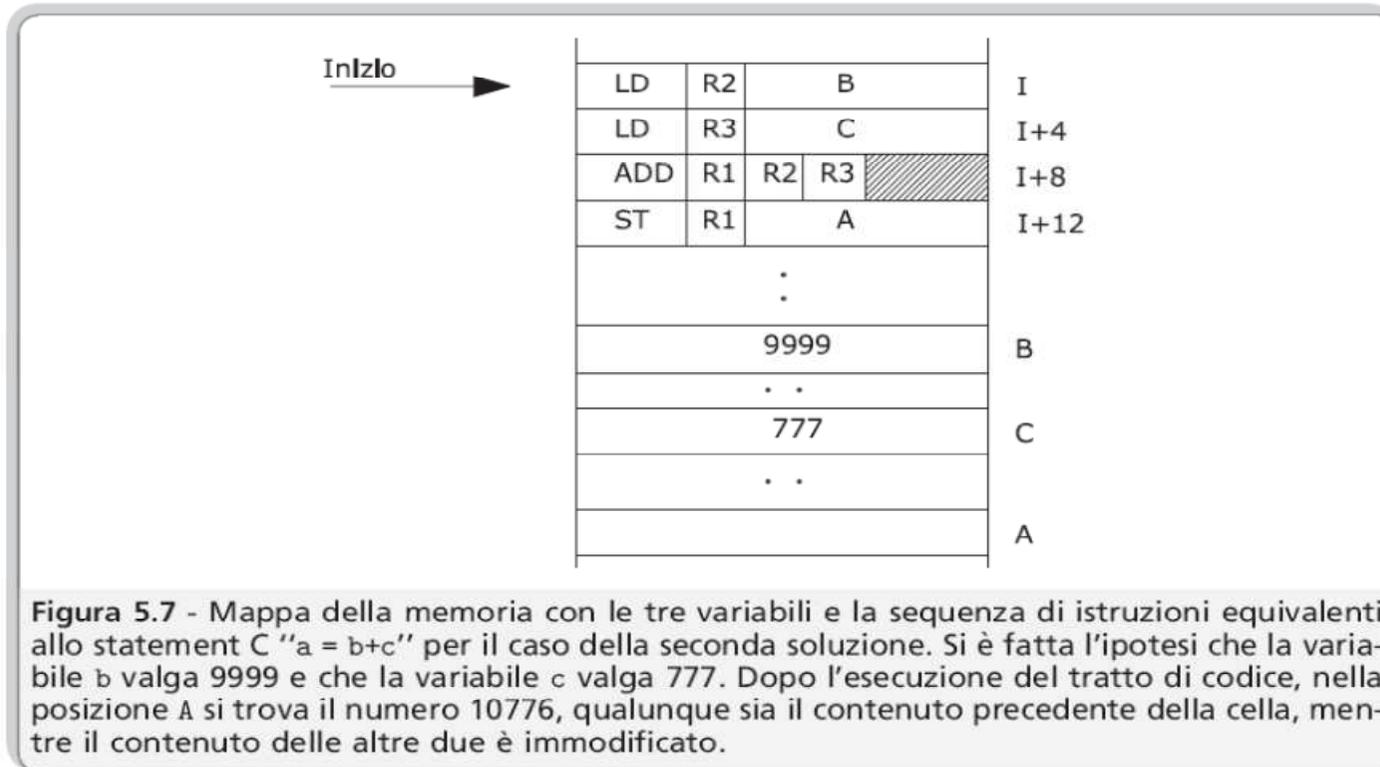
ESEMPIO: una CPU con istruzioni a lunghezza fissa di 32 bit



LD R1, Var ; R1 ← M[Var]

ADD R1, R2, R3 ; R1 ← R2 + R3

# Sequenze di istruzioni in memoria

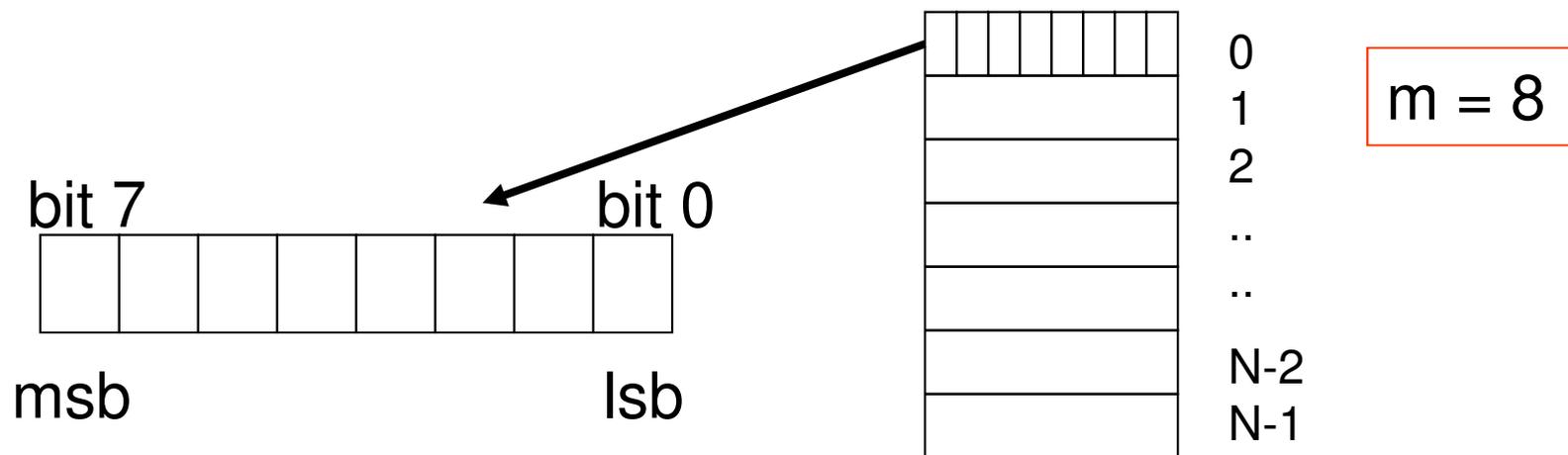


Lo statement  $a = b + c$  si traduce come:

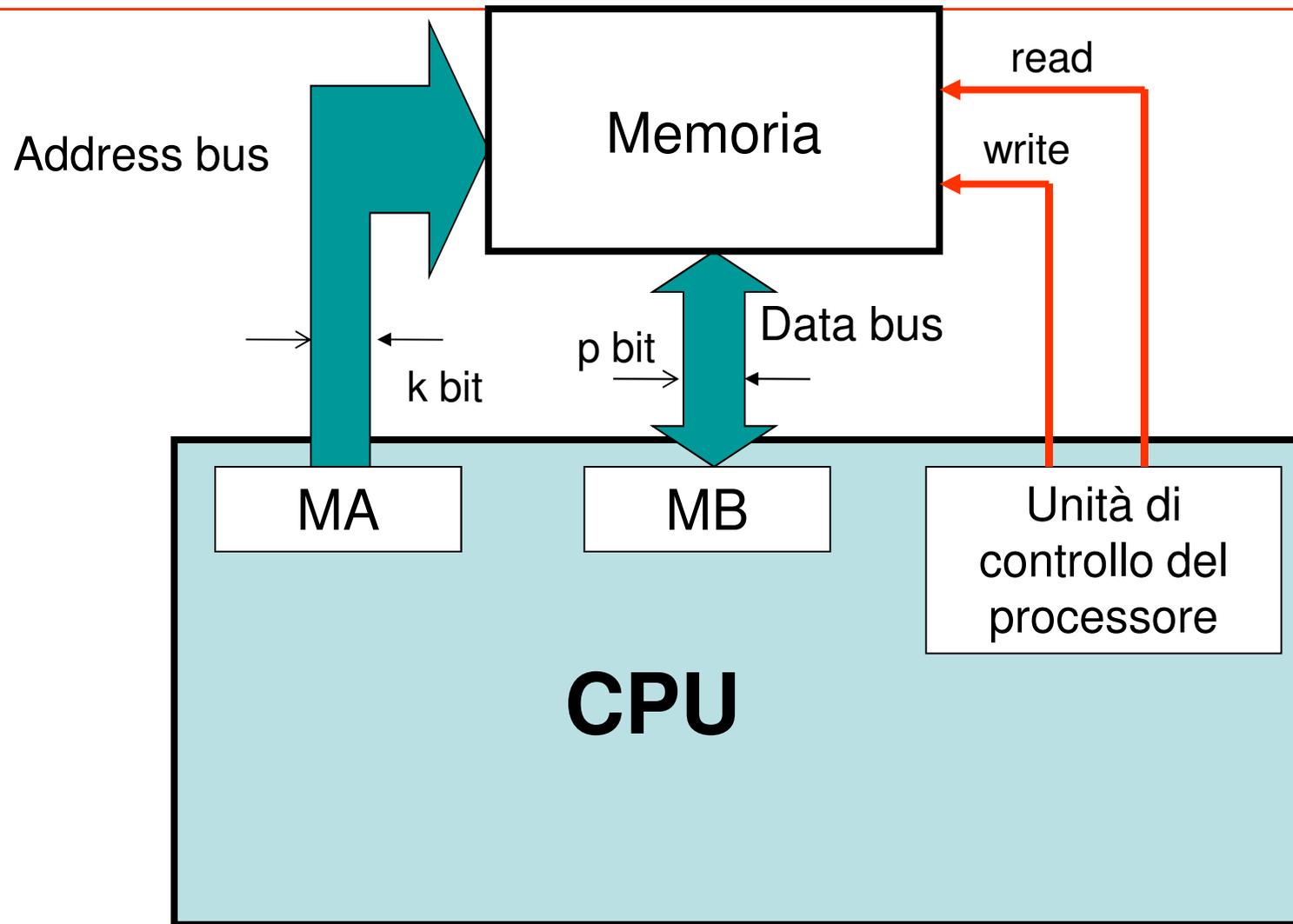
|                |   |
|----------------|---|
| LD R2, B       | ;B indirizzo a cui è allocata la parola b |
| LD R3, C       | ;C indirizzo a cui è allocata la parola c |
| ADD R1, R2, R3 |   |
| ST A, R1       | ;A indirizzo a cui è allocata la parola a |

# 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



# Interazione processore-memoria



# Processori a carattere e processori a parola

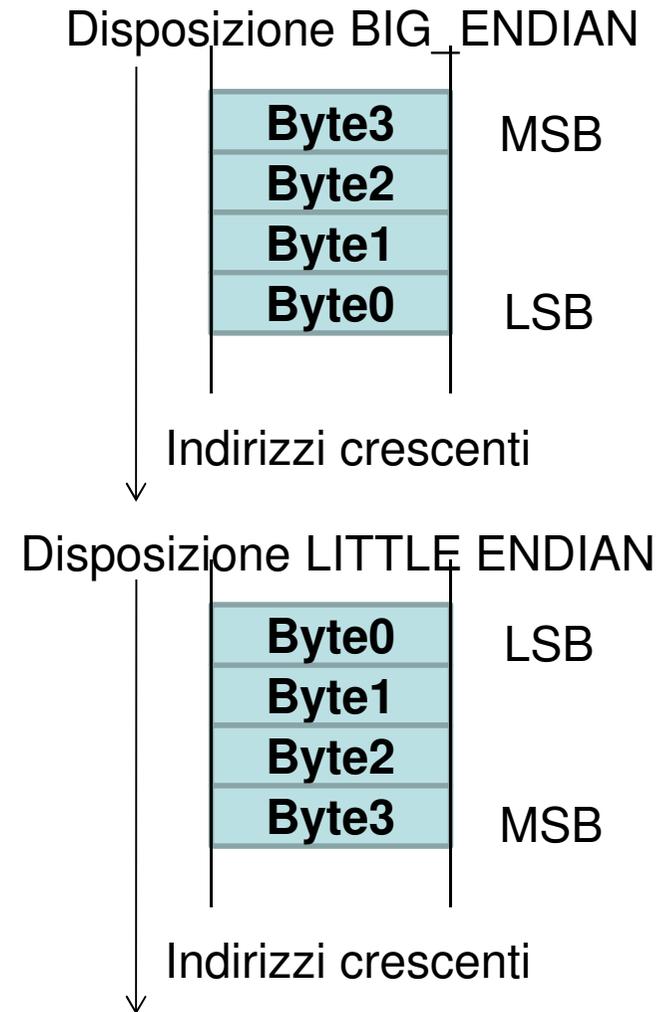
---

- 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

---

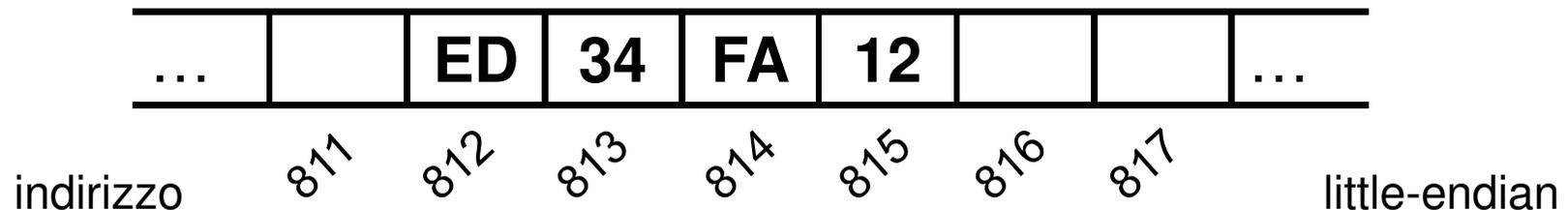
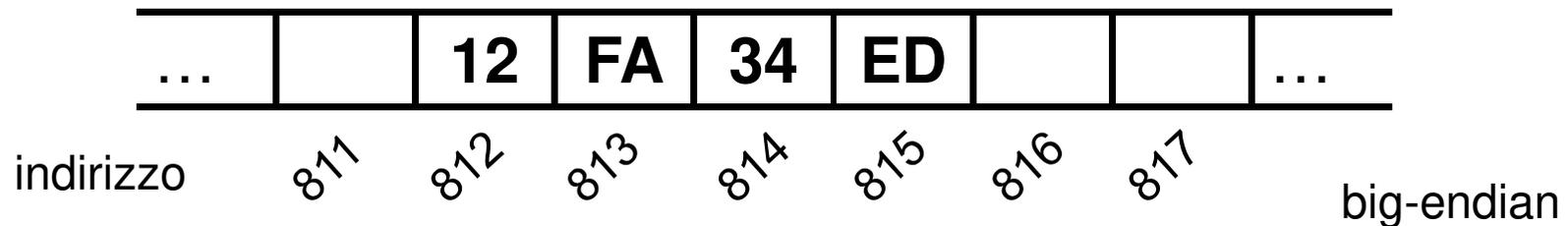
- I processori possono disporre in memoria i byte che formano una parola da 16, 32 o 64 bit in 2 modi
  - *Big-endian*: i byte sono disposti in memoria in modo che il più significativo MSB occupi la locazione di memoria di indirizzo minore, e poi via via gli altri, fino a quello meno significativo LSB 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’utilizzo 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