# Programmazione I

*Corso di Laurea in Informatica*
*a.a. 2017-2018*

**Dr. Gabriele Tolomei**
Dipartimento di Matematica
Università degli Studi di Padova
gtolomei@math.unipd.it


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

4 Dicembre 2017

# Informazioni Utili



## Ricevimento Studenti

- **Giovedì** dalle **14:30** alle **16:00**
- Stanza **519** (Torre Archimede - **V Piano**, Corridoio **BC**)

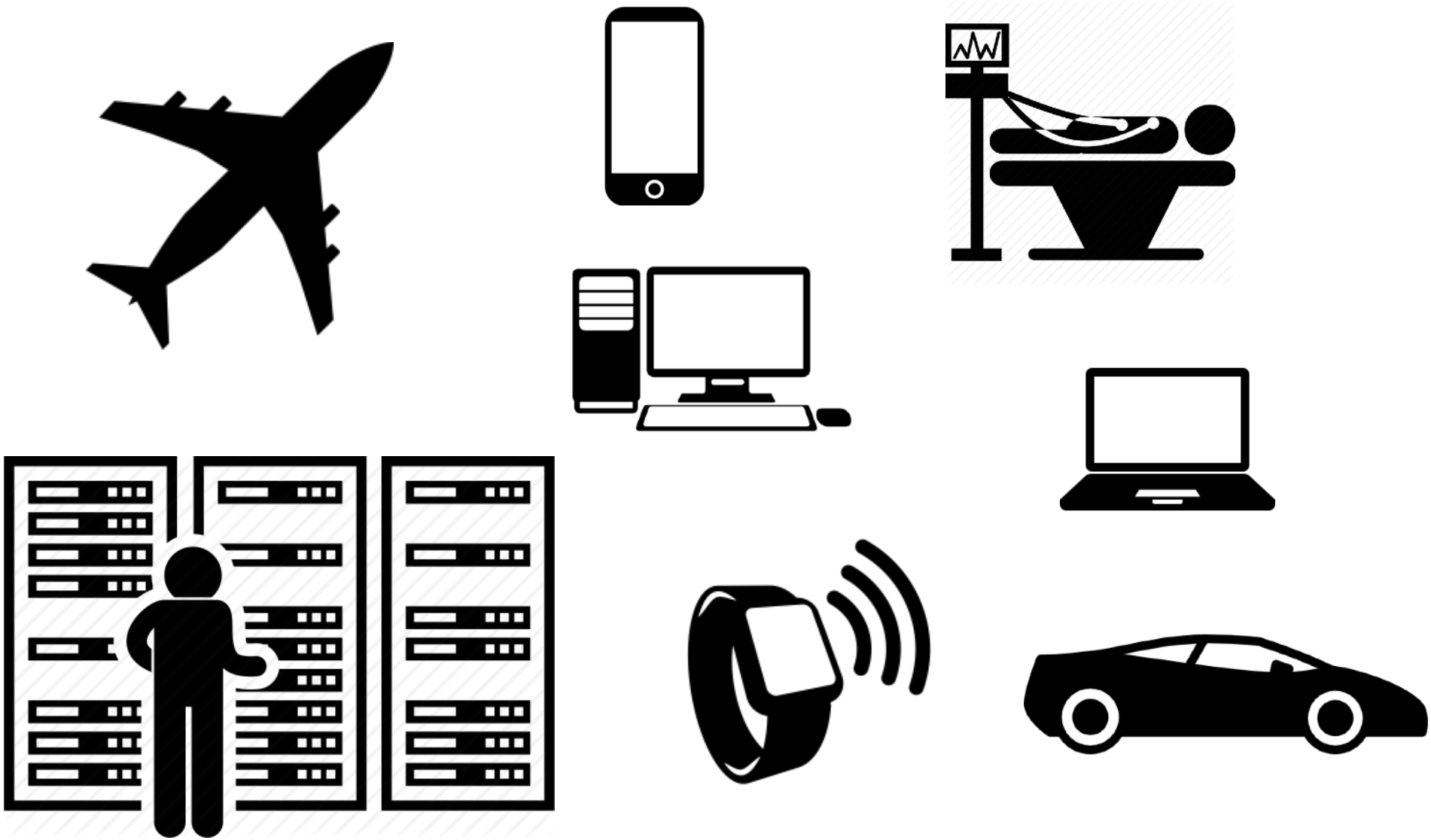
# Oggi Parleremo Di…



- Come sono fatti i calcolatori (architettura)
  - Memoria Principale (RAM)
  - CPU
- "Livelli di Astrazione" (della macchina fisica)
  - Dal Linguaggio Macchina ai Linguaggi per le Applicazioni (**C/C++**, **Java**, etc.)
- Cosa significa *programmare* un calcolatore

# I Calcolatori Sono Ovunque!





# Architettura di un Calcolatore

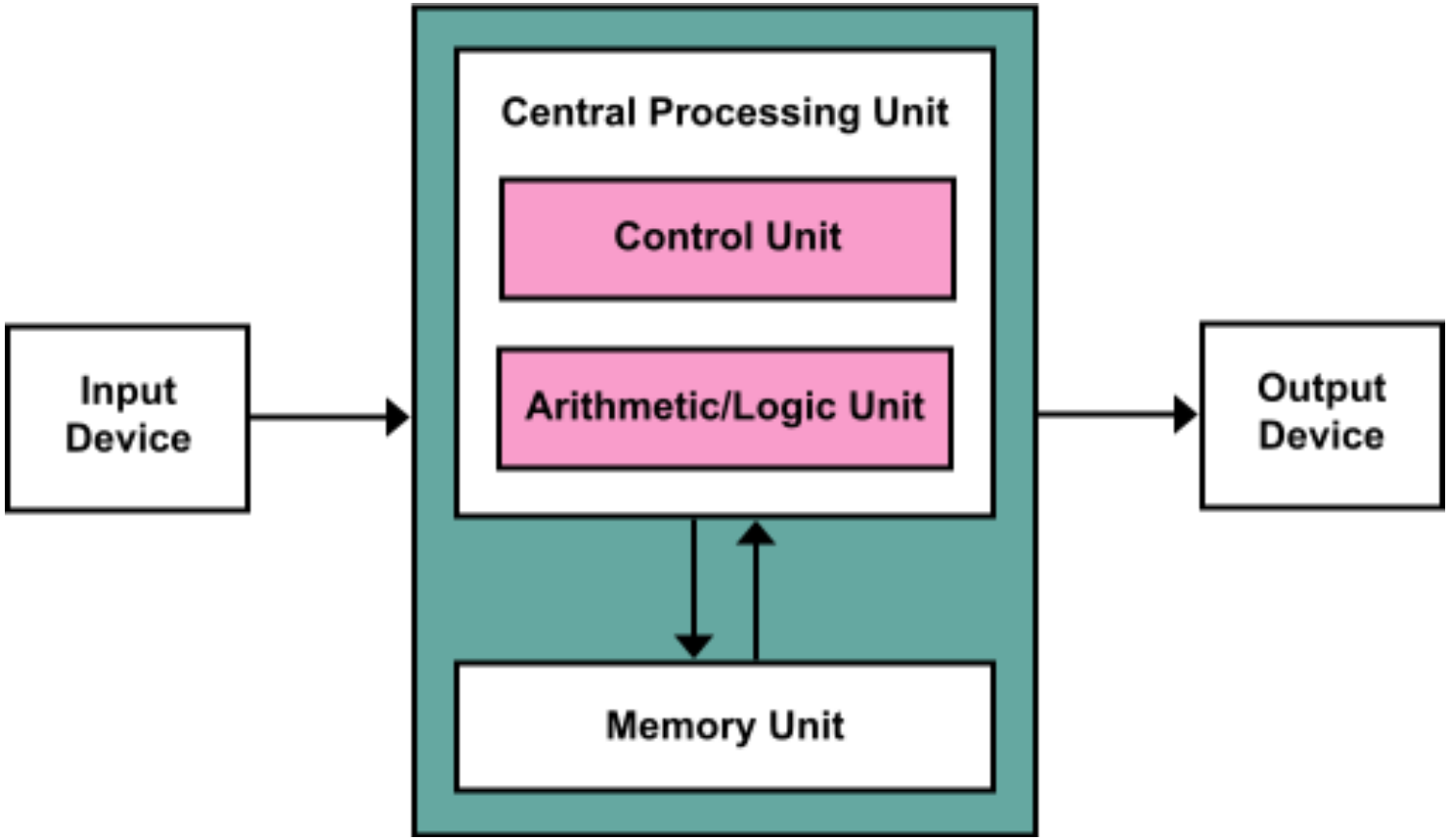


Concettualmente identica per tutti: **von Neumann Architecture**

John von Neumann

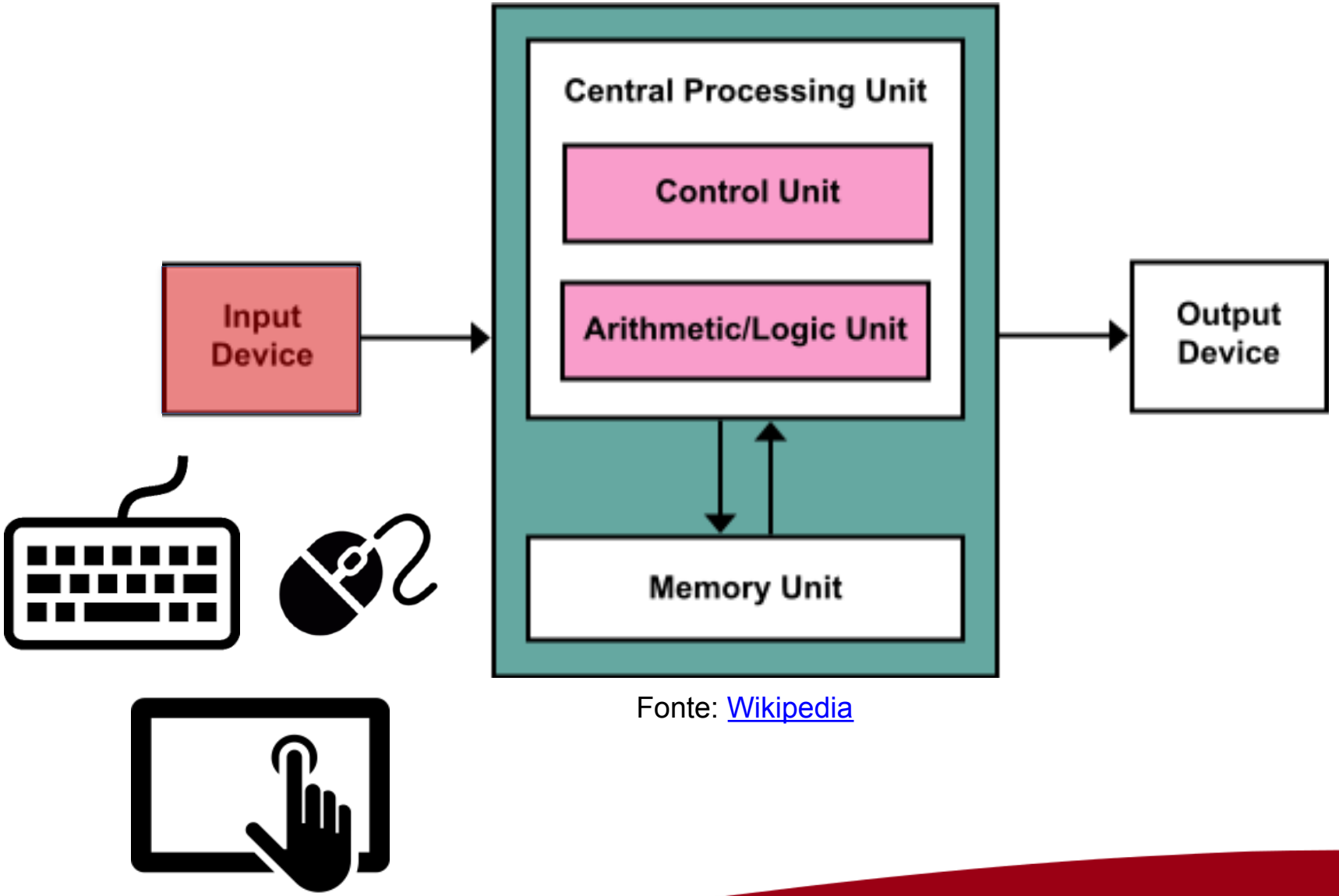
# Architettura di von Neumann






Fonte: Wikipedia

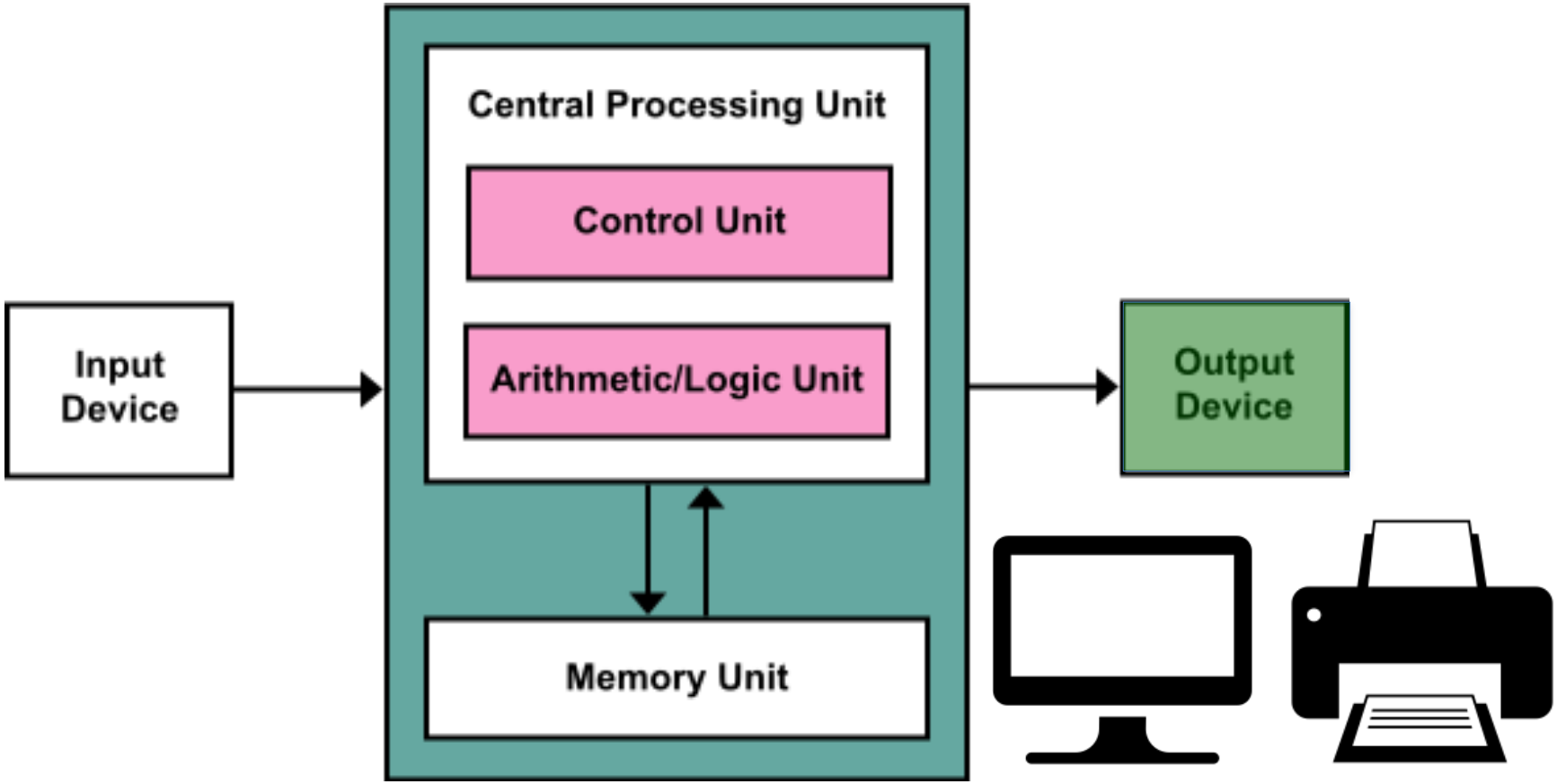
# Architettura di von Neumann




Fonte: Wikipedia

# Architettura di von Neumann

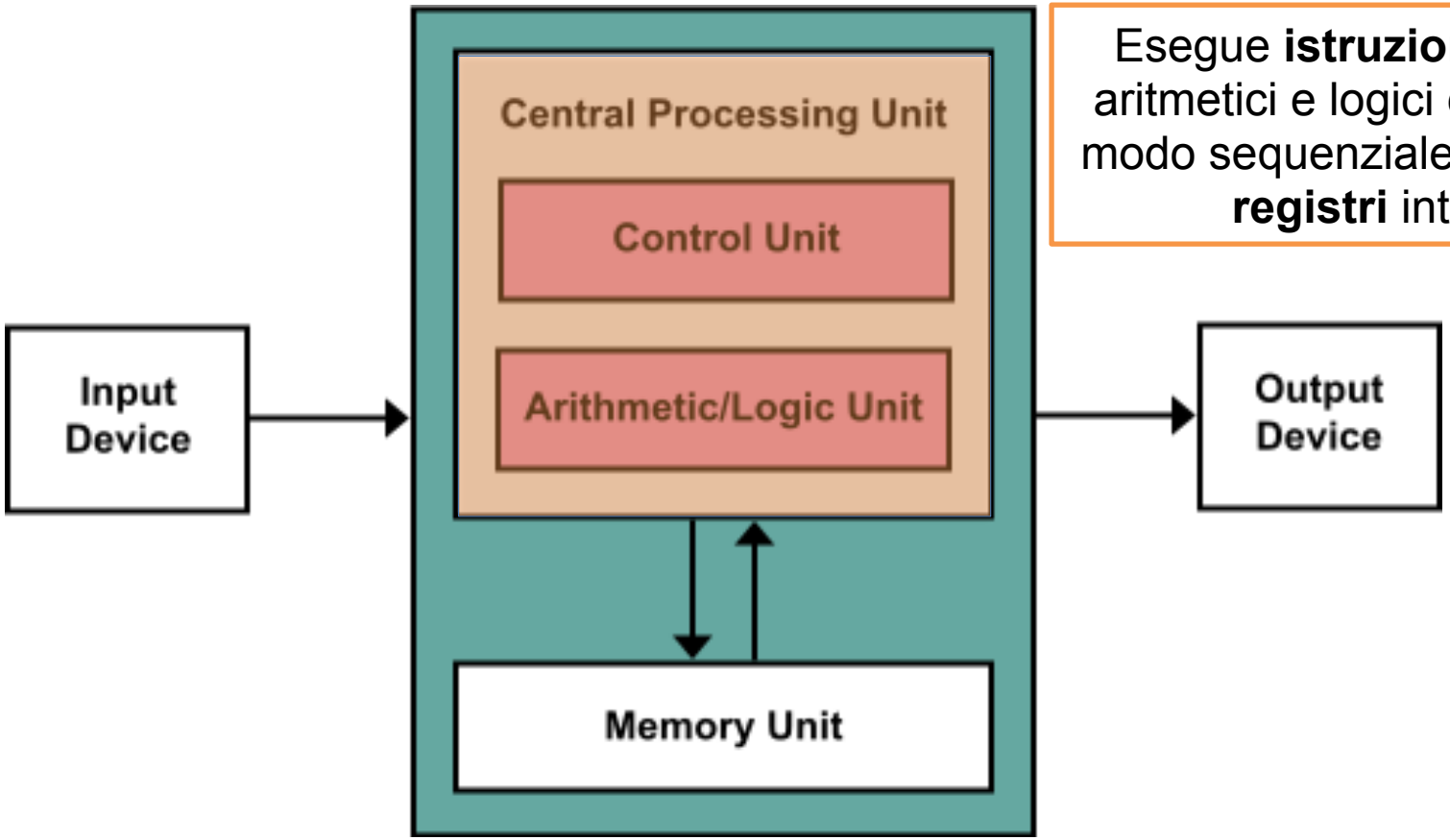





Fonte: Wikipedia

# Architettura di von Neumann



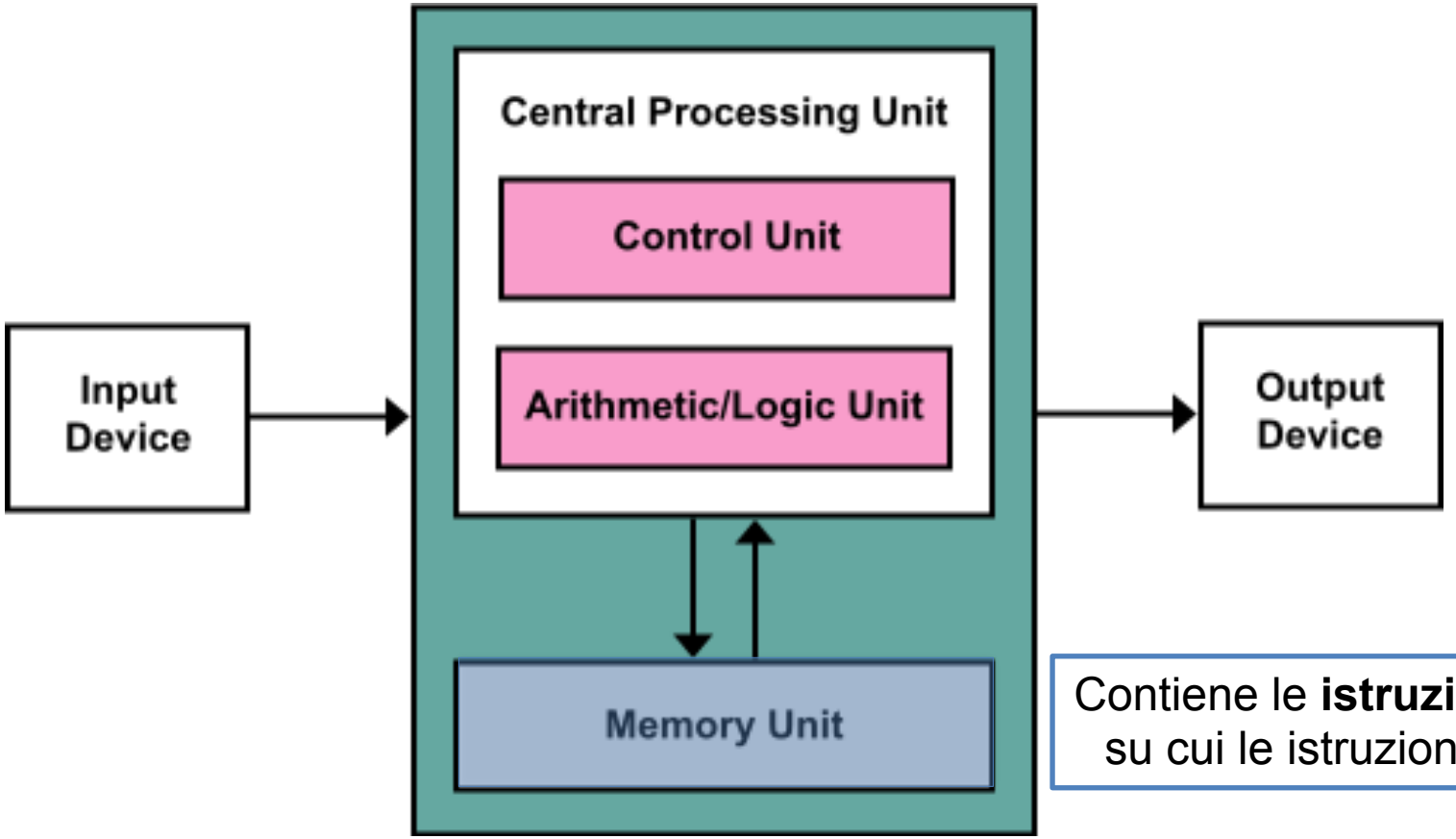



Esegue **istruzioni** (calcoli aritmetici e logici di base) in modo sequenziale sfruttando **registri** interni

Fonte: Wikipedia

# Architettura di von Neumann






Contiene le **istruzioni** e i **dati** su cui le istruzioni operano

Fonte: Wikipedia

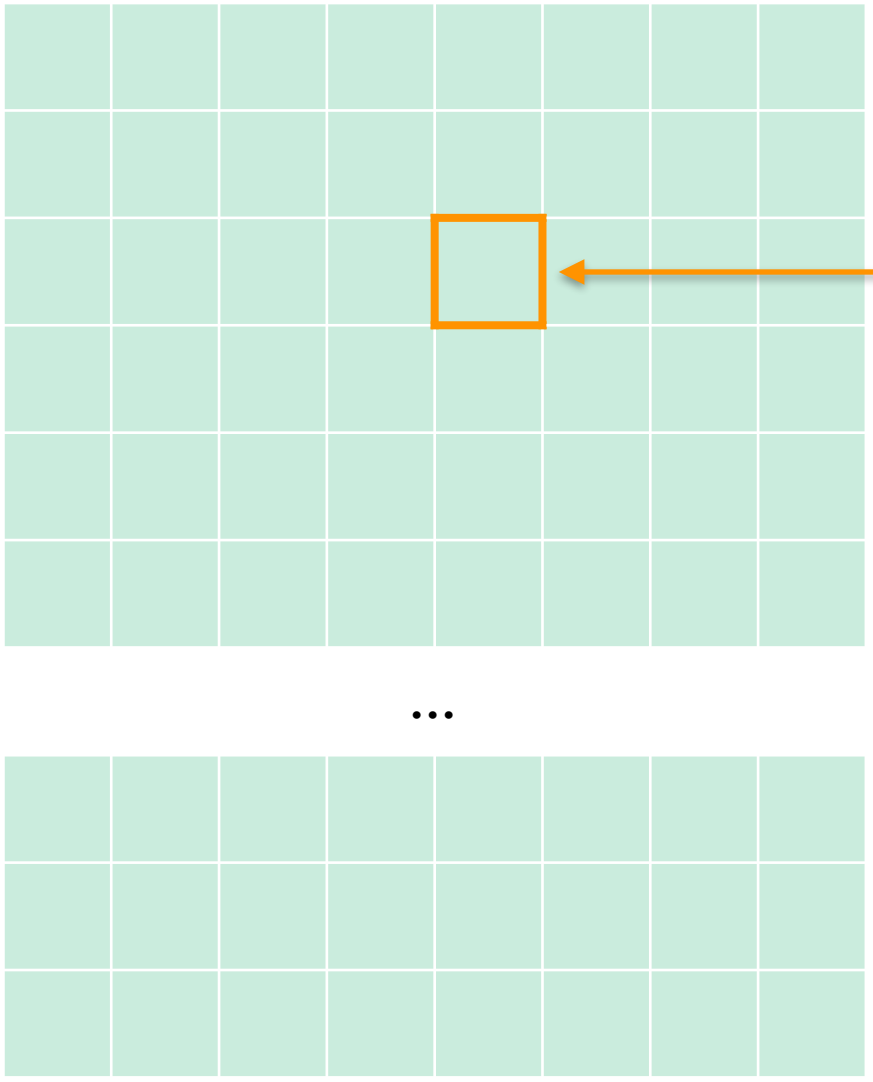
# La Memoria Principale (RAM)



- La memoria principale (RAM) è rappresentata come una sequenza contigua di **celle** (**locazioni**)
- Ciascuna cella è organizzata logicamente in gruppi di **8 bit** (**1 byte**), o multipli del byte (ad es., 32 bit = 4 byte)
- Inoltre, ogni cella ha un proprio **indirizzo**
- Le altre unità (CPU e I/O) possono richiedere operazioni di lettura/scrittura dalla/in memoria principale specificando l'indirizzo della locazione
- L'unità di indirizzamento è, solitamente, il singolo byte

# La Memoria Principale (RAM)



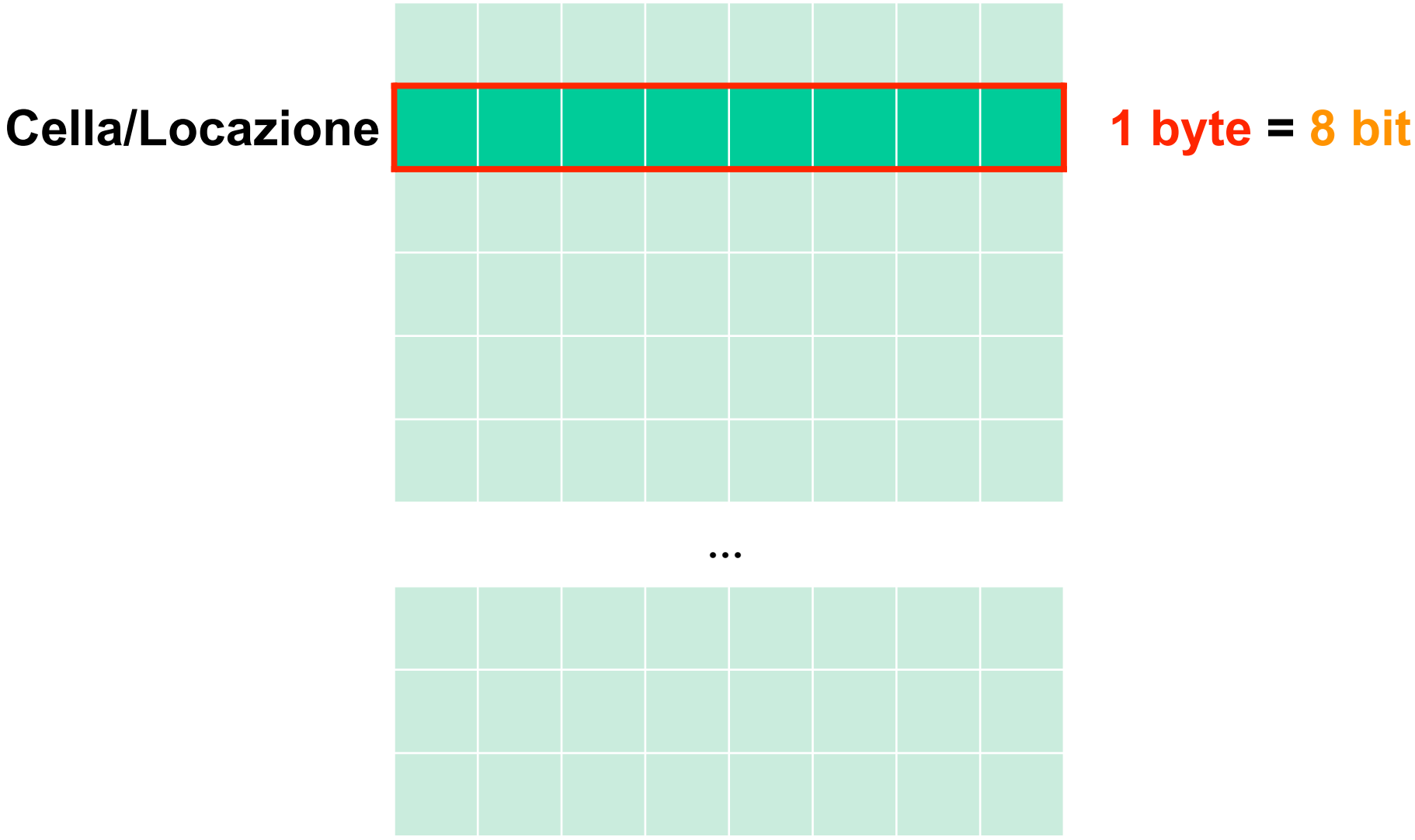



Un **bit** (**bi**nary digi**t**) può esclusivamente assumere uno tra **2** possibili valori: **0** e **1**

Codifica vantaggiosa rappresentata da 2 diversi livelli di tensione elettrica nei circuiti digitali

# La Memoria Principale (RAM)

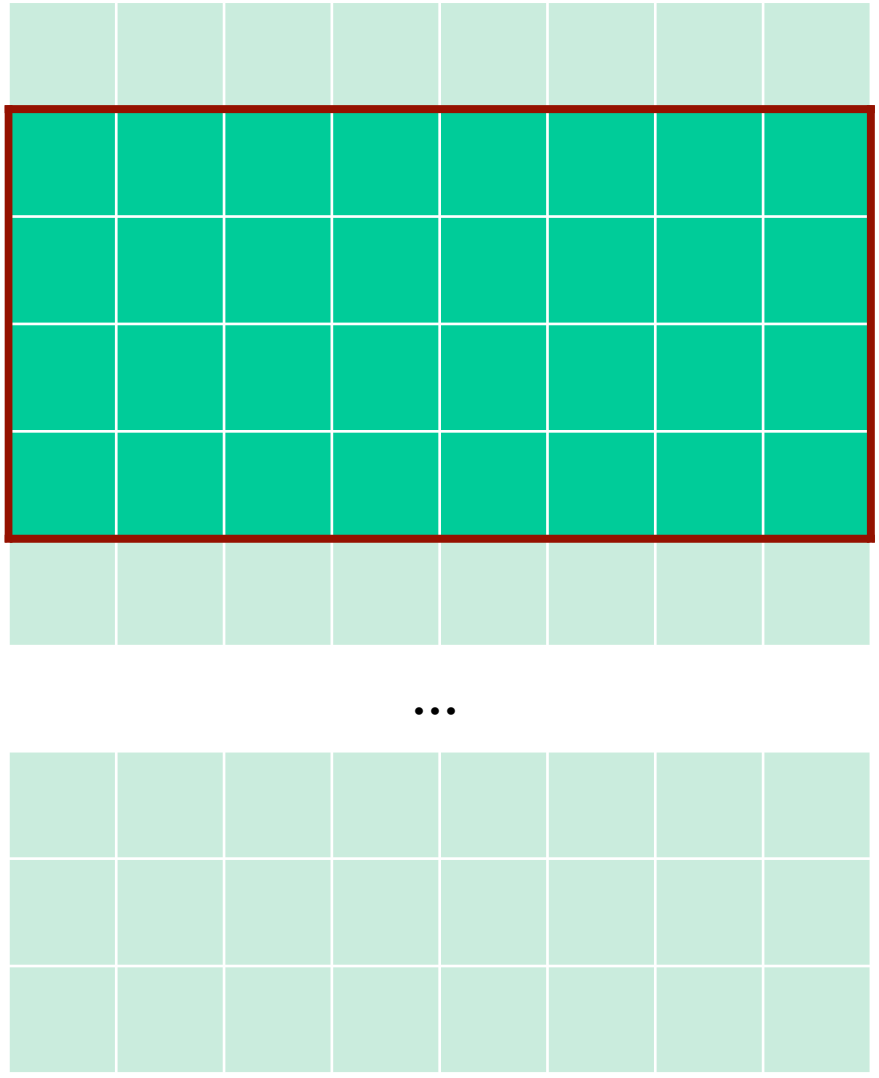




# La Memoria Principale (RAM)

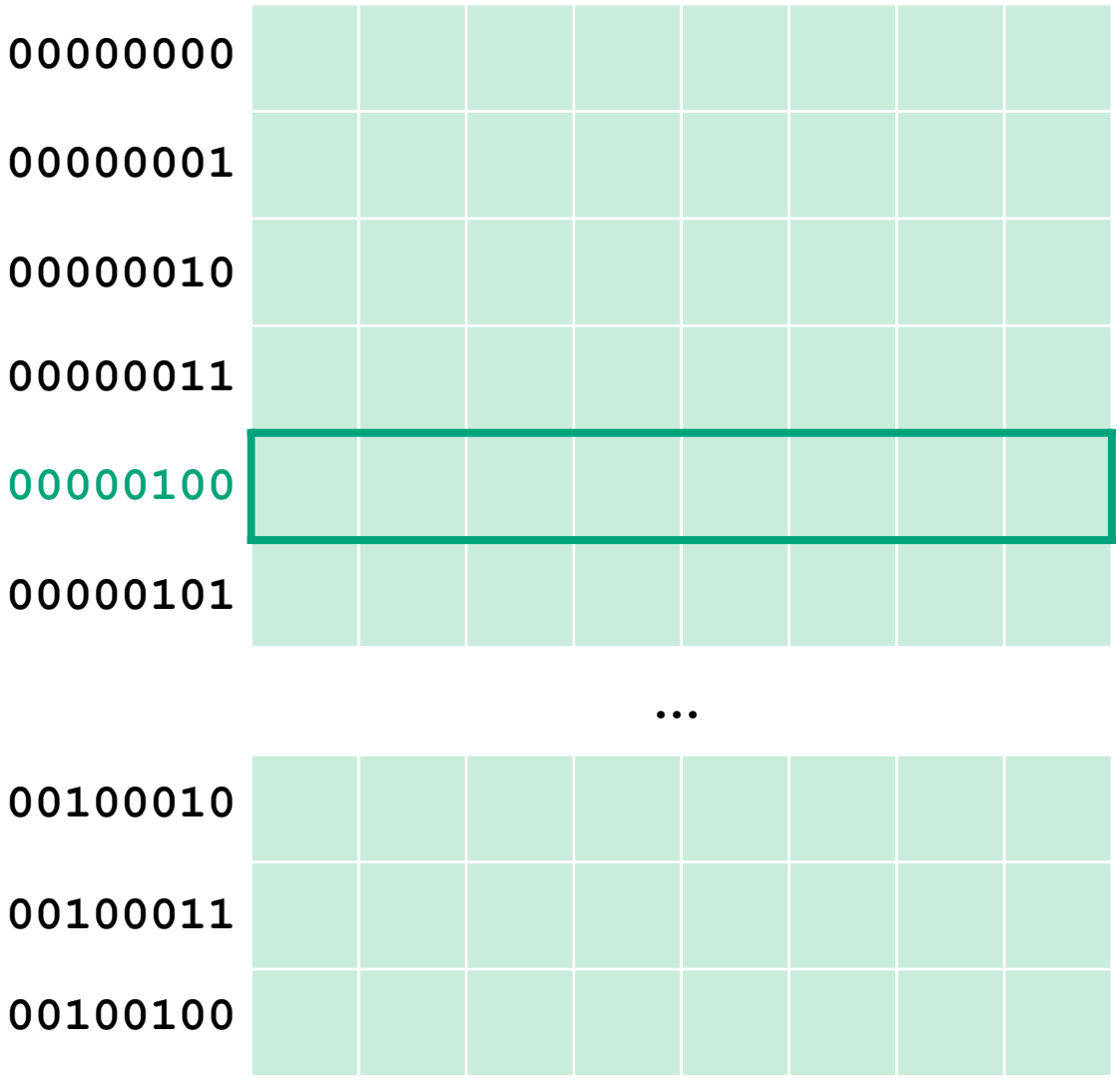


Cella/Locazione



4 byte = 32 bit

# La Memoria Principale (RAM)




indirizzo
di una locazione
(al byte)

# CPU: Fetch-Decode-Execute



In generale, la CPU esegue ciclicamente le seguenti operazioni:

1. **Fetch:** preleva un'istruzione ad un certo indirizzo della memoria RAM (il cui valore è contenuto in uno speciale registro della CPU, chiamato *Program Counter* o *PC*)
2. **Decode:** decodifica l'istruzione prelevata
3. **Execute:** esegue l'istruzione decodificata

# Il Linguaggio Macchina



- Definisce un insieme di **istruzioni** (elementari) che la CPU è in grado di eseguire *direttamente*
- Il linguaggio è codificato attraverso il **sistema numerico binario**
- Ossia, ciascuna istruzione del linguaggio macchina è codificata come una **sequenza di bit**

# Il Sistema Decimale vs. Binario



- Nel **linguaggio naturale**, solitamente ci riferiamo al **sistema numerico decimale** (base 10)
- Ogni cifra può assumere solo uno tra **10** possibili valori: `0, 1, …, 9`

| 1 | 0 | 1 |
|---|---|---|
| $10^2$ | $10^1$ | $10^0$ |

$1*10^0 + 0*10^1 + 1*10^2 = 101$

- Nel **sistema numerico binario** (base 2), ciascuna cifra è un **bit**

| 1 | 0 | 1 |
|---|---|---|
| $2^2$ | $2^1$ | $2^0$ |

$1*2^0 + 0*2^1 + 1*2^2 = 5$

# Instruction Set

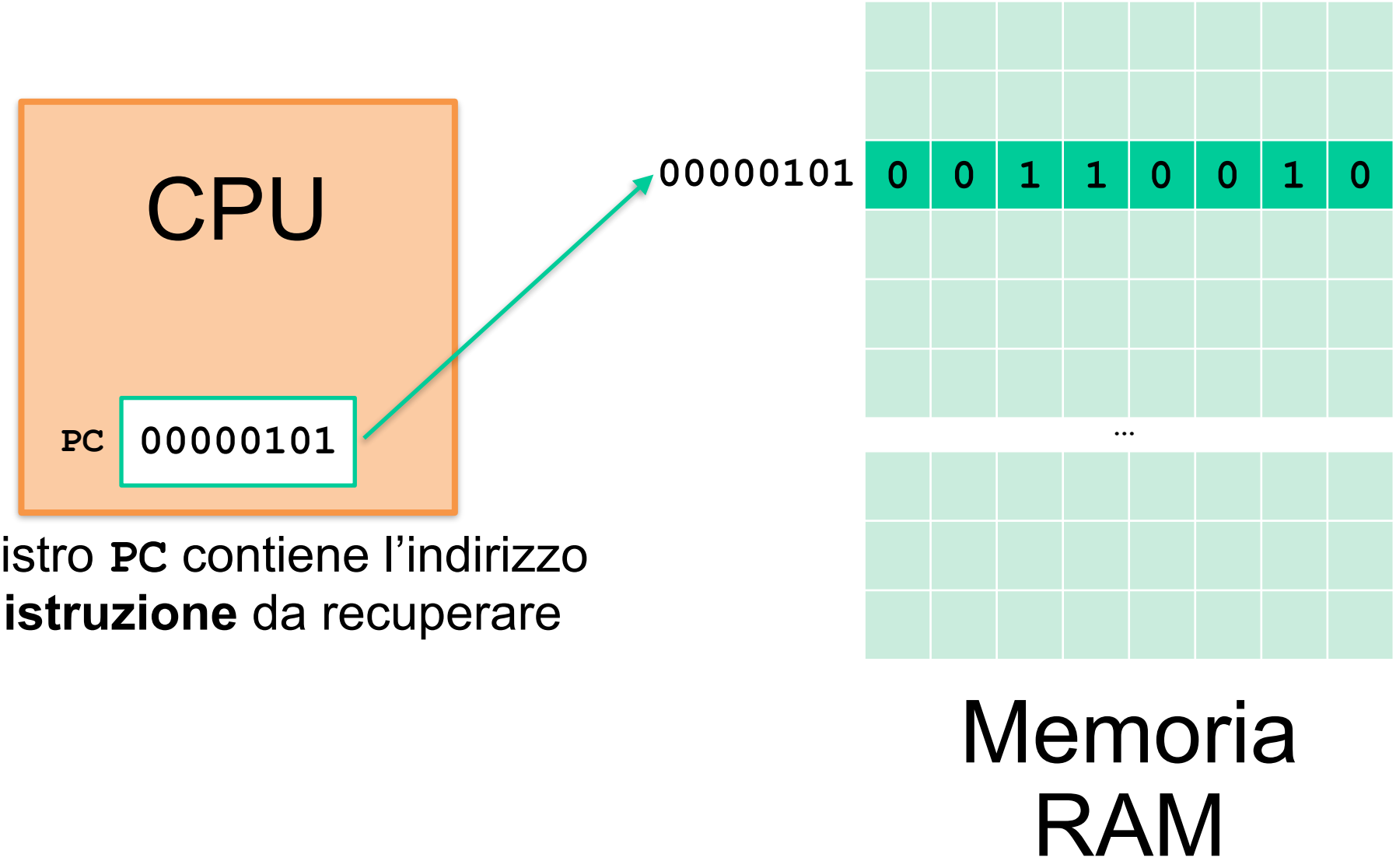


- Le istruzioni definite dal linguaggio macchina sono composte da **2 parti**:
  - Un **operatore** (`op code`)
  - Uno (o più) **operandi** che rappresentano registri interni della CPU o indirizzi di locazioni di memoria
- L'insieme delle istruzioni definite dal linguaggio macchina (**instruction set**) è specifico per un certo tipo di *realizzazione* di architetture **hardware**:

  *Intel x86, ARM, SPARC, MIPS, …*
- L'instruction set definisce il **numero di bit** da cui ciascuna istruzione è composta e quanti di questi bit sono dedicati all'operatore e agli operandi.

# Esempio Istruzione 8 bit: Fetch

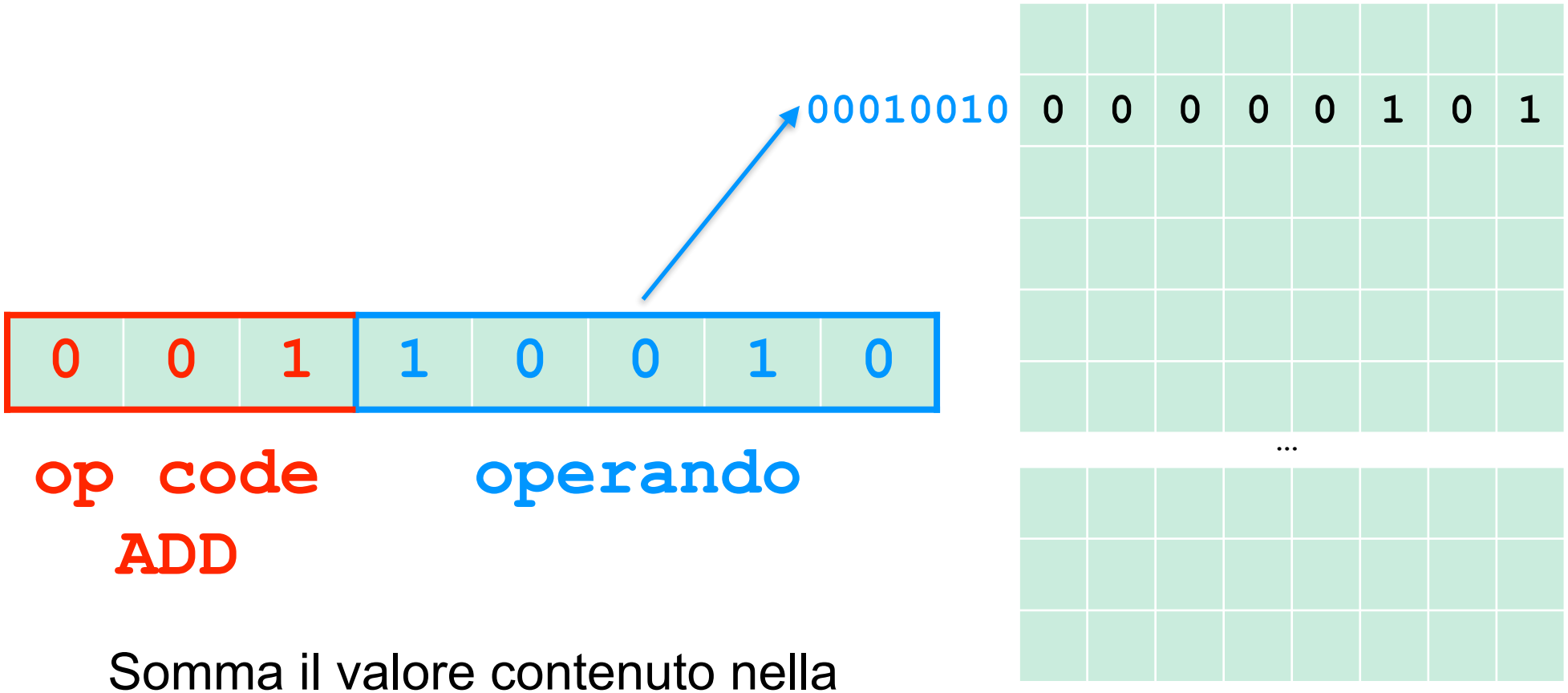





Il registro **PC** contiene l'indirizzo dell'**istruzione** da recuperare

# Esempio Istruzione 8 bit: Decode

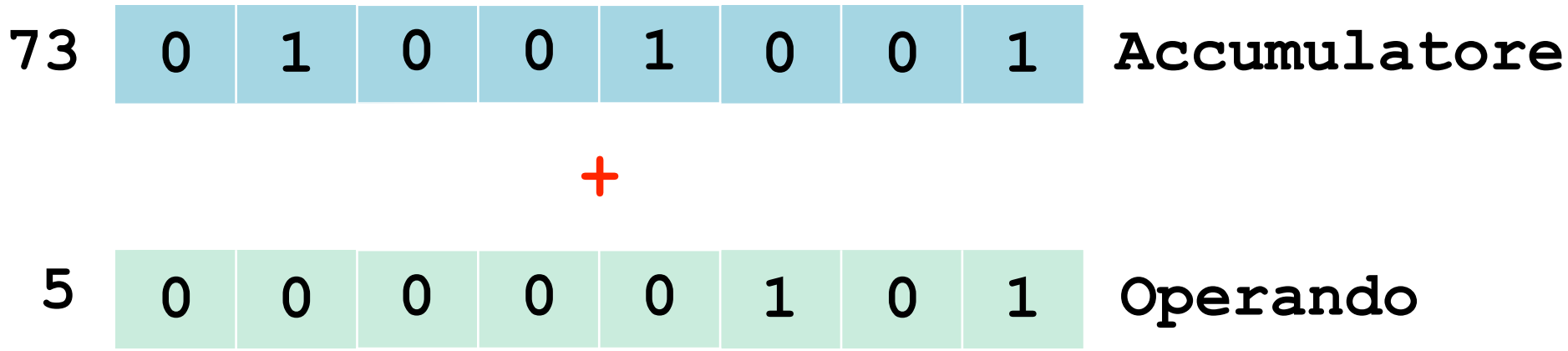





Somma il valore contenuto nella locazione di memoria indicata dall'operando a quello contenuto in un registro interno della CPU (**accumulatore**)

# Esempio Istruzione 8 bit: Execute

# Esempio Istruzione 8 bit: Execute



# Esempio Istruzione 8 bit: Execute



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 73 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | Accumulatore |
| | | | | + | | | | | |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Operando |
| | | | | = | | | | | |
| 78 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | Accumulatore |

# Istruzioni vs. Dati



- Nel modello di von Neumann, le locazioni di memoria contengono sia le istruzioni che i dati
- La CPU non potrebbe distinguere tra istruzioni e dati semplicemente “leggendo” la sequenza di bit memorizzata ad un certo indirizzo
- Il registro PC garantisce la separazione tra istruzioni e dati, poiché deve sempre contenere l’indirizzo in cui è memorizzata un’istruzione!

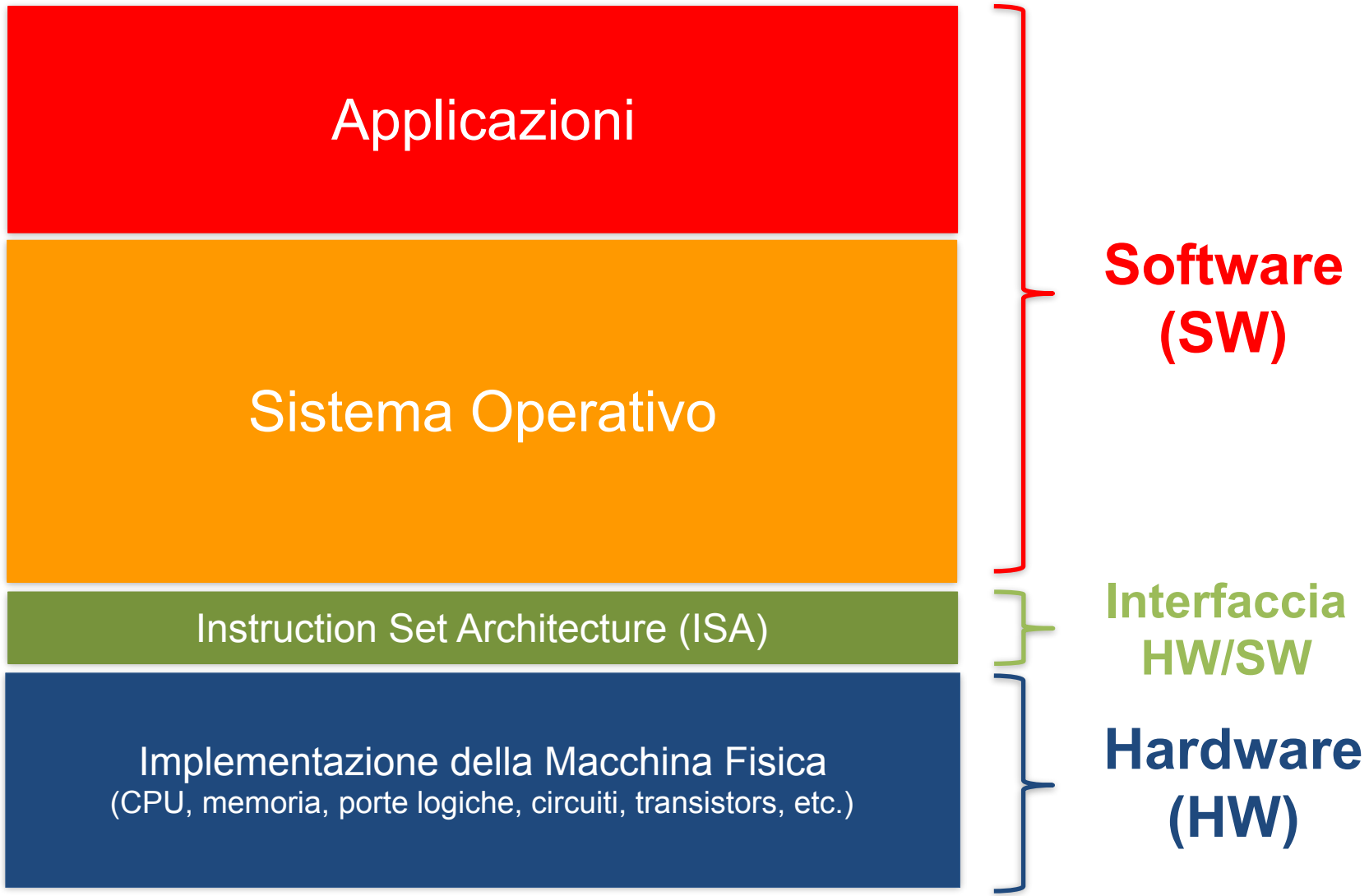
# Il Linguaggio Macchina



- Il linguaggio macchina (binario) fornisce la rappresentazione di un programma più “vicina” alla macchina fisica (hardware)
- Teoricamente sarebbe possibile programmare un calcolatore utilizzando direttamente il linguaggio macchina
- In pratica, sarebbe una follia! (Considerate la complessità dei programmi in esecuzione sui nostri calcolatori…)
- Perciò si è cercato via via di “astrarre” sempre più il linguaggio macchina (basso livello) con linguaggi che si avvicinassero a quello naturale (alto livello)

# Livelli di Astrazione





# Livelli di Astrazione



- Ad ogni livello coincide un **linguaggio** in esso adottato
- Ogni funzionalità (del linguaggio di) ciascun livello è implementata da un programma scritto nel/i linguaggio/i del/i livello/i sottostante/i
- **PRO:**

  L'astrazione favorisce la separazione tra "**cosa**" deve essere realizzato e "**come**" (*implementazione*)
- **CONTRO:**

  Tanto più ci si astrae dalla macchina fisica tanto minore sarà il "controllo" che avremo su di essa (delegato ai livelli inferiori)

# Cosa Significa Programmare?



- In generale, significa scrivere un programma in un linguaggio di alto livello che risolva un certo problema
  - Ad es., trovare il minimo elemento in un insieme di interi
- In questo corso, utilizzeremo **C/C++** come linguaggio di alto livello per scrivere i nostri programmi
- Il codice risultante dalla scrittura del programma è comunemente detto **sorgente** e non può essere direttamente eseguito dal calcolatore
  - Ricordatevi: una CPU è in grado di eseguire solamente le istruzioni definite dalle specifiche di un certo linguaggio macchina (binario)

# Traduzione



- Il processo tramite cui un programma scritto in un linguaggio ad alto livello (**sorgente**) viene "trasformato" nel corrispondente linguaggio macchina (**oggetto**)
- Le fasi che compongono il processo di traduzione dipendono dall'implementazione del linguaggio ad alto livello
- In generale, si possono identificare le seguenti fasi:

  **Compilazione** (*compiling*)

  **Assemblaggio** (*assembling*)

  **Collegamento** (*linking*)

# Compilazione



- Traduzione del programma sorgente in linguaggio macchina “intermedio”, ossia non numerico, detto **assembly**
- Il risultato della compilazione (intesa come fase “logica”) non è pertanto direttamente eseguibile sulla macchina fisica
- Il compilatore (**compiler**) è l’apposito programma responsabile di questa attività
  - Riceve in input un programma scritto in un linguaggio di alto livello e restituisce in output il corrispondente codice assembly

# Compilazione



- Il codice assembly, seppur non direttamente eseguibile dalla CPU, è strettamente legato all'architettura del calcolatore (ISA)
  - Il compilatore deve "conoscere" l'architettura di destinazione
  - Lo stesso programma che debba essere eseguito su architetture diverse, necessita di compilatori che producano codice assembly diverso

# Assemblaggio



- Il linguaggio assembly prodotto dalla compilazione è un'astrazione più "leggibile" del linguaggio macchina binario
- Per esempio, la sequenza di bit `00110010` dell'istruzione dell'esempio precedente può essere scritta in assembly come:

```
ADD $Memory_Address
```

- Sarà compito dell'assemblatore (**assembler**) tradurre il codice assembly nel corrispondente linguaggio macchina binario (**oggetto**)

# Collegamento



- Spesso un programma scritto in linguaggio sorgente è composto da più file
- Ciascuno di questi file deve essere compilato ed assemblato in modo da ottenere il relativo file oggetto (binario)
- Il **linker** si occupa di creare **un singolo file** eseguibile a partire da ciascun file oggetto

# Esempio: gcc/g++