# Informatica

Dal bit al sistema operativo

## 1 Il bit e la rappresentazione binaria

Dentro un processore ci sono miliardi di **transistor**: piccoli interruttori controllati elettricamente che possono trovarsi in due soli stati — acceso o spento. A questi due stati fisici associamo per convenzione i simboli **1** (tensione presente) e **0** (tensione assente).

Tutta l'informatica nasce da questa semplicità: qualunque informazione — numeri, lettere, immagini, suoni — viene rappresentata come sequenza di 0 e 1.

**Il bit**

Il **bit** (*Binary digit*) è l'unità minima di informazione: rappresenta la scelta tra due sole alternative. Può valere 0 oppure 1.

Con $n$ bit si possono rappresentare esattamente $2^n$ valori distinti (es: $n = 3$, quindi ho $2^3 = 8$ valori distinti).

### Sistema decimale e sistema binario

Il sistema che usiamo ogni giorno è il **sistema decimale**: dieci cifre $\{0, 1, \ldots, 9\}$ e valore posizionale in base 10. La cifra 3 dentro il numero 347 non vale 3 ma 300, perché si trova nella posizione delle centinaia:

$$347 = 3 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0$$

Il computer non può permettersi dieci simboli diversi: ne ha solo due, 0 e 1. Usa allora il **sistema binario**, che funziona esattamente come quello decimale ma con base 2 al posto di base 10.

**Sistema binario**

Il sistema binario usa solo due cifre $\{0, 1\}$. Il valore di un numero binario si ottiene moltiplicando ogni cifra per la corrispondente potenza di 2 e sommando tutto:

$$N_{10} = c_{n-1} \cdot 2^{n-1} + \cdots + c_1 \cdot 2^1 + c_0 \cdot 2^0$$

dove $c_0$ è la cifra più a destra (meno significativa). I coefficienti $c_n, c_1, c_0$ che vedete, nel sistema binario possono solo avere valore 0 o 1.

### Conversione da binario a decimale

La regola è meccanica: si scrivono sopra ogni bit le potenze di 2 decrescenti partendo da destra, e si sommano solo le potenze dei bit che valgono 1.

**Convertire $11011_2$ in decimale**

| **1** | **1** | **0** | **1** | **1** |
|---|---|---|---|---|
| $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 16 | 8 | 4 | 2 | 1 |

Sommo solo dove il bit vale 1:

$$16 + 8 + 0 + 2 + 1 = 27_{10}$$

Quindi $11011_2 = 27_{10}$.

**Altro esempio: $10110_2$**

$$1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 22_{10}$$

### Tabella dei primi valori

Vale la pena memorizzare almeno i primi numeri binari. Notate il meccanismo: la colonna più a destra alterna $0, 1, 0, 1, \ldots$; la successiva alterna a coppie $0, 0, 1, 1, \ldots$; e così via.

| Decimale | Binario | Decimale | Binario |
|---|---|---|---|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | 10 | 1010 |
| 3 | 0011 | 11 | 1011 |
| 4 | 0100 | 12 | 1100 |
| 5 | 0101 | 13 | 1101 |
| 6 | 0110 | 14 | 1110 |
| 7 | 0111 | 15 | 1111 |

### Conversione da decimale a binario

Per fare il percorso inverso — partire da un numero decimale e scriverlo in binario — si usa il **metodo delle divisioni successive per 2**. L'idea: si divide ripetutamente il numero per 2, annotando ogni volta il *resto* (che può valere solo 0 o 1). Ci si ferma quando il quoziente arriva a 0. Il numero binario si ottiene leggendo i resti **dal basso verso l'alto**.

**Metodo delle divisioni successive**

1. Divido il numero decimale per 2 e annoto il resto.
2. Divido il quoziente ottenuto per 2 e annoto il resto.
3. Ripeto finché il quoziente non diventa 0.
4. Leggo i resti *dall'ultimo al primo* (dal basso verso l'alto): ottengo il numero binario.

**Convertire $27_{10}$ in binario**

$$\begin{aligned} 27 \div 2 &= 13 \quad \text{resto} \quad 1 \\ 13 \div 2 &= 6 \quad \text{resto} \quad 1 \\ 6 \div 2 &= 3 \quad \text{resto} \quad 0 \\ 3 \div 2 &= 1 \quad \text{resto} \quad 1 \\ 1 \div 2 &= 0 \quad \text{resto } 1 \leftarrow \text{inizio lettura} \end{aligned}$$

Leggendo i resti dal basso verso l'alto: $27_{10} = 11011_2$.

*Verifica:* $16 + 8 + 0 + 2 + 1 = 27$. ✓

**Convertire $22_{10}$ in binario**

$$\begin{aligned} 22 \div 2 &= 11 \quad \text{resto} \quad 0 \\ 11 \div 2 &= 5 \quad \text{resto} \quad 1 \\ 5 \div 2 &= 2 \quad \text{resto} \quad 1 \\ 2 \div 2 &= 1 \quad \text{resto} \quad 0 \\ 1 \div 2 &= 0 \quad \text{resto } 1 \leftarrow \text{inizio lettura} \end{aligned}$$

Quindi $22_{10} = 10110_2$.

## 2 Il byte

Un singolo bit è troppo piccolo per essere utile. I computer lavorano quindi su gruppi di bit. Il gruppo fondamentale si chiama **byte**.

**Byte**

Un **byte** è un gruppo di **8** bit. Con un byte si possono rappresentare

$$2^8 = 256 \text{ valori distinti (da 0 a 255).}$$

Il byte è l'unità di indirizzamento della memoria del computer: ogni cella della RAM (che vedremo tra poco) ha la dimensione di un byte.

## 3 La codifica del testo: ASCII e Unicode

Sappiamo rappresentare numeri in binario. Ma come si rappresenta una lettera? La risposta è semplice: si sceglie una **convenzione** — una tabella che associa a ogni carattere una sequenza di bit.

### Il codice ASCII

Il primo standard di codifica del testo si chiama **ASCII** (*American Standard Code for Information Interchange*, 1963).

**ASCII**

L'ASCII associa a ogni carattere un numero intero a **7 bit**, quindi copre $2^7 = 128$ simboli:

- 33 caratteri di controllo non stampabili (a capo, tabulazione, ...);
- 95 caratteri stampabili: lettere maiuscole e minuscole dell'alfabeto inglese, cifre, segni di punteggiatura.

**Alcuni codici ASCII**

| Carattere | Decimale | Binario |
|---|---|---|
| `A` | 65 | 1000001 |
| `B` | 66 | 1000010 |
| `a` | 97 | 1100001 |
| `0` | 48 | 0110000 |
| spazio | 32 | 0100000 |

### Il problema dell'ASCII: le altre lingue

L'ASCII copre solo l'alfabeto inglese. Non ha la "à", non ha la "ñ", non ha i caratteri greci, cinesi, arabi. Per decenni ogni paese ha usato estensioni incompatibili tra loro — fonte di infinita confusione.

### Unicode

La soluzione definitiva si chiama **Unicode**: uno standard internazionale che assegna un codice univoco a *ogni* carattere di *ogni* sistema di scrittura umano. Oggi Unicode contiene oltre 140 000 caratteri, inclusi gli emoji.

La codifica Unicode più usata si chiama **UTF-8** e ha una proprietà fondamentale: è **retrocompatibile con l'ASCII**. I primi 128 caratteri hanno lo stesso codice in entrambi gli standard. I caratteri "esotici" vengono rappresentati usando più byte (fino a 4).

**Osservazione**

Un file scritto usando solo caratteri ASCII è automaticamente un file UTF-8 valido. È per questo che Unicode ha potuto sostituire gradualmente l'ASCII senza rompere la compatibilità con decenni di software esistente.

## 4 Dalla macchina di Turing al computer moderno

Prima di guardare dentro un computer dobbiamo rispondere a una domanda teorica: *cosa vuol dire, esattamente, "una macchina che calcola"?* La risposta arriva da un matematico inglese, Alan Turing.

### La macchina universale di Turing (1936)

Nel 1936 Alan Turing propose un modello astratto di macchina calcolatrice, non una macchina reale da costruire ma un'idea matematica. Nella versione più famosa, la **macchina universale di Turing** è costituita da:

- un **nastro** infinito diviso in celle, su cui si possono leggere e scrivere simboli;
- una **testina** che legge/scrive una cella alla volta e si sposta a destra o a sinistra;
- un insieme di **regole** che dicono, per ogni stato e simbolo letto, cosa scrivere, dove muoversi e a quale stato passare.

**L'intuizione di Turing**

Una macchina di Turing **universale** è una macchina che può risolvere qualunque problema calcolabile, purché le si diano le istruzioni giuste. Non serve una macchina diversa per ogni problema.

Questa è la base teorica di tutta l'informatica. Un computer moderno non è altro che una realizzazione fisica di una macchina universale di Turing.

### Dalla teoria alla pratica: cosa mancava ai primi computer

I primi calcolatori elettronici (come l'ENIAC, 1945) funzionavano, ma avevano un problema pratico enorme: per cambiare programma bisognava **ricollegare fisicamente** i circuiti. Il programma non era memorizzato dentro la macchina — era la macchina stessa.

Questo li rendeva molto diversi dalla macchina universale di Turing, dove il programma era un dato sul nastro, modificabile senza toccare il meccanismo.

### La macchina di Von Neumann (1945)

Il matematico ungherese **John von Neumann** risolse il problema con un'idea semplice e rivoluzionaria.

**Il principio del programma memorizzato**

Il programma deve essere memorizzato *nella stessa memoria dei dati*.

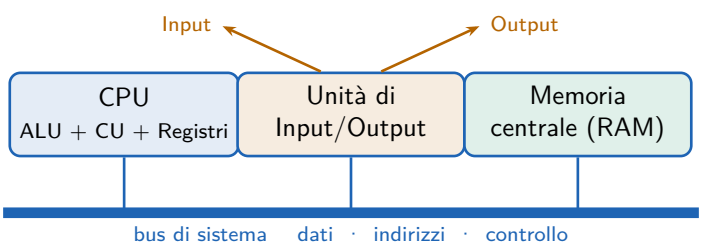
Un programma è trattato come qualunque altro dato: può essere caricato, copiato, modificato. Cambiare il comportamento della macchina diventa un'operazione software, non hardware.

È qui che la macchina universale di Turing diventa finalmente reale: il "nastro con la descrizione del programma" che Turing immaginava è ora la memoria RAM di un computer.

**Turing vs. Von Neumann**

La macchina di Turing è un **modello teorico** che stabilisce *cosa* è calcolabile. La macchina di Von Neumann è un **progetto di architettura** che stabilisce *come* costruire fisicamente un calcolatore universale. Turing dice cosa si può fare, Von Neumann dice come farlo davvero.

### I quattro componenti dell'architettura di Von Neumann



**CPU** (*Central Processing Unit*): il cervello del computer. Esegue le istruzioni del programma una dopo l'altra. Contiene:

- l'**ALU** (*Arithmetic Logic Unit*): esegue operazioni aritmetiche ($+, -, \times, \div$) e logiche (AND, OR, NOT, confronti);
- l'**unità di controllo** (CU): legge le istruzioni, le interpreta e coordina gli altri componenti;
- i **registri**: piccole memorie ultraveloci *dentro* la CPU, dove stanno i dati su cui si sta lavorando in questo istante.

**Memoria centrale (RAM)**: contiene *sia* le istruzioni del programma in esecuzione, *sia* i dati su cui lavora. È l'elemento chiave dell'architettura di Von Neumann. La RAM è **volatile**: il suo contenuto svanisce quando si spegne il computer.

**Unità di Input/Output**: tutto ciò che permette al computer di comunicare con il mondo esterno — tastiera, mouse, schermo, stampante, rete.

**Bus**: i "corridoi" attraverso cui i dati viaggiano tra CPU, RAM e I/O. Ne esistono di tre tipi: bus dati (trasporta i valori), bus indirizzi (specifica *dove* in memoria leggere/scrivere), bus di controllo (segnali di coordinamento).

### La memoria di massa (Storage)

La RAM è **volatile**: allo spegnimento del PC i dati svaniscono. Per conservare file e programmi in modo permanente, si usa la **memoria di massa**:

- **Hard Disk (HDD):** dischi magnetici rotanti. Economici e capienti, ma lenti e delicati.
- **SSD (Solid State Drive):** chip elettronici (senza parti mobili). Molto veloci e resistenti, sono lo standard attuale.

**Osservazione**

Immagina la RAM come la tua **scrivania** (spazio di lavoro veloce ma piccolo) e l'Hard Disk/SSD come la tua **libreria** (archivio grande e permanente).

## 5 Il ciclo Fetch–Decode–Execute

La CPU non fa altro che ripetere, miliardi di volte al secondo, lo stesso ciclo di tre passi.

**Ciclo Fetch–Decode–Execute (FDE)**

1. **Fetch** (preleva): la CU legge dalla RAM l'istruzione successiva da eseguire. Un registro speciale chiamato *Program Counter* contiene l'indirizzo della prossima istruzione.
2. **Decode** (decodifica): la CU interpreta i bit dell'istruzione per capire *cosa* fare e *su quali dati*.
3. **Execute** (esegui): l'operazione viene effettivamente eseguita — dall'ALU se è un calcolo, dalla CU se è un trasferimento di dati o un salto. Il risultato viene scritto in un registro o in RAM.

Terminato il ciclo, il Program Counter viene aggiornato e si ricomincia dal punto 1.

**Un ciclo FDE concreto (esempio)**

Programma che calcola $3 + 5$:

| Indirizzo | Istruzione | |
|---|---|---|
| 100 | `LOAD R1, 3` | (carica 3 nel registro R1) |
| 101 | `LOAD R2, 5` | (carica 5 nel registro R2) |
| 102 | `ADD R3, R1, R2` | (R3 = R1 + R2) |
| 103 | `STORE R3, mem` | (salva il risultato in RAM) |

La CPU esegue queste quattro istruzioni in sequenza, un ciclo FDE ciascuna.

# 6 Hardware e software

**Hardware e software**

- **Hardware**: tutto ciò che è *fisico* e tangibile — processore, RAM, disco, schermo, tastiera. La macchina *è* l'hardware.
- **Software**: l'insieme delle *istruzioni* che dicono all'hardware cosa fare. Non si tocca, non ha peso: esiste solo come sequenza di bit in memoria.

I due sono inseparabili: senza software il computer è un blocco di silicio inutile; senza hardware il software non ha dove girare.

## Le tre tipologie di software

**Tipi di software**

- **Software di sistema**: gestisce le risorse hardware e crea l'ambiente in cui girano gli altri programmi. Il protagonista è il **sistema operativo** (Windows, macOS, Linux); rientrano qui anche driver (fa comunicare il sistema operativo con i componenti), firmware (componente software fisso dentro ogni singolo componente) e BIOS (accende il PC e avvia il sistema operativo).
- **Software applicativo**: i programmi che l'utente usa per svolgere compiti concreti — elaboratori di testo (Word), browser (Chrome, Firefox), videogiochi, fogli di calcolo (Excel).
- **Software di utilità**: programmi ausiliari che supportano il sistema — antivirus, strumenti di backup dei dati, compressori di file (WinRAR, 7-Zip), strumenti di diagnostica.

**Classificazione rapida**

| Programma | Tipo |
|---|---|
| Windows 11, macOS, Linux | Software di sistema |
| Microsoft Word, Google Chrome, Minecraft | Software applicativo |
| Windows Defender, 7-Zip | Software di utilità |
| Driver della scheda grafica | Software di sistema |