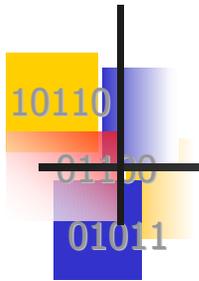


10110

01100

01011

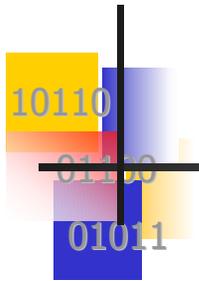
# PROBLEMI E ALGORITMI



## Specifica di un algoritmo

---

- Primo approccio, **scrittura diretta del programma**: la soluzione coincide con la codifica
  - Causa errori difficilmente individuabili,
  - Non garantisce che non esistano soluzioni chiaramente migliori,
  - Non funziona con programmi grandi,
  - Rende difficile cambiare linguaggio di programmazione,
- Perciò, è conveniente **concentrarsi sulla specifica del problema e dell'algoritmo**; la codifica dovrebbe essere solo un passo implementativo.



## ESEMPIO: Equazione di secondo grado

---

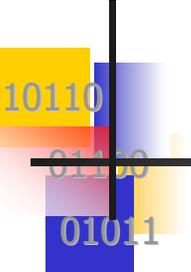
- **Problema:** Calcolare le radici dell'equazione

$$ax^2 + bx + c = 0$$

- **Specifica del problema**

- **Dati di ingresso:** tre numeri reali  $a$ ,  $b$ ,  $c$ .
- **Pre-condizione:** nessuna
- **Dati di uscita:** le radici  $x_1$  e  $x_2$ , se esistono.
- **Post-condizione:**

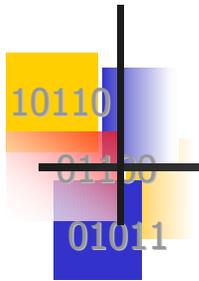
$$ax_1^2 + bx_1 + c = 0 \text{ e } ax_2^2 + bx_2 + c = 0$$



# ESEMPIO: Equazione di secondo grado

---

- **Specifica dell'algoritmo:**
- $\delta = b^2 - 4ac$
- se ( $\delta \geq 0$ )
  - $\text{sqrtdelta} = \text{radice quadrata}(\delta)$
  - $x_1 = (-b + \text{sqrtdelta}) / 2a$
  - $x_2 = (-b - \text{sqrtdelta}) / 2a$
- altrimenti
  - non esistono radici reali  $\delta$



## ESEMPIO: Prodotto di N numeri

---

- **Problema:** Calcolare il prodotto di N numeri.
- **Specifica del problema:**
  - **Dati di ingresso:** N numeri  $\{x_1, x_2, \dots, x_n\}$
  - **Pre-condizione:** nessuna
  - **Dati di uscita:** un numero  $P$
  - **Post-condizione:**  $P = \prod_{i,n} x_i$

10110

01100

01011

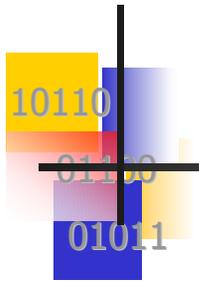
## ESEMPIO: Prodotto di N numeri

---

- **Specifica dell'algoritmo:**

- $P = 1$
- $i = 1$
- Finchè ( $1 \leq i \leq N$ )
  - $P = P * x_i$
  - $i = i + 1$

# Dal Linguaggio Macchina ai Linguaggi di Alto Livello



# Fondamenti di Informatica - **Linguaggio Macchina**

- Le istruzioni elementari eseguite dalla CPU di un computer si chiamano *istruzioni macchina*.
- L'insieme delle istruzioni macchina (*instruction set*) costituiscono il **linguaggio macchina**.
- Un linguaggio macchina
  - consente la programmazione della *Macchina di von Neumann*,
  - è **direttamente eseguibile** da un calcolatore **senza nessuna traduzione**,
  - naturalmente **cambia da macchina a macchina** (ad es., quello del Pentium è diverso da quello dello AMD).

10110

01100

01011

# Fondamenti di Informatica - Linguaggio Macchina

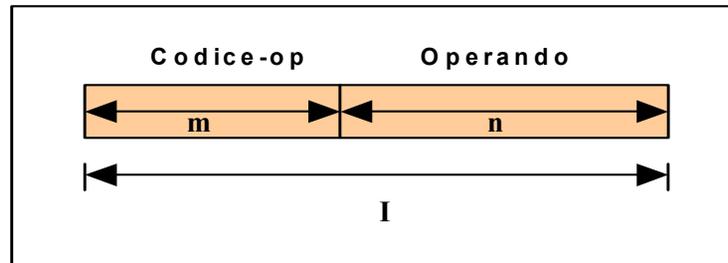
- **Le istruzioni sono codificate in formato binario** e sono composte da
  - **CODICE OPERATIVO** : indica l'istruzione da eseguire
  - **OPERANDI** : indicano gli operandi (indirizzi o valore)

Per semplicità ipotizziamo di avere istruzioni con solo operando.

- Lunghezza delle istruzioni :

$$I = m + n$$

**m**: num bit del codice operativo, **n**: num bit dell'operando.



10110

01100

01011

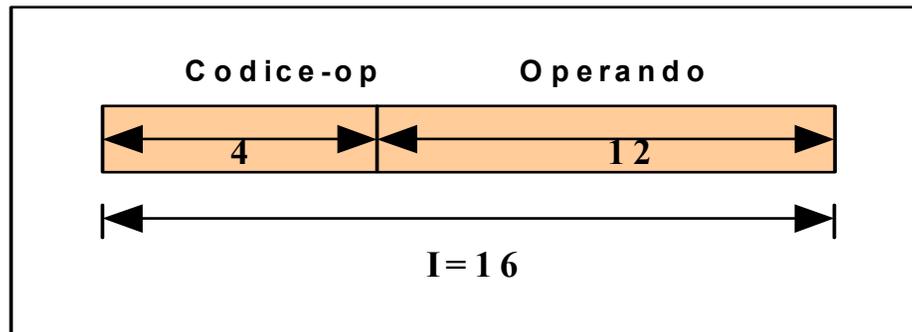
# Fondamenti di Informatica - Linguaggio Macchina

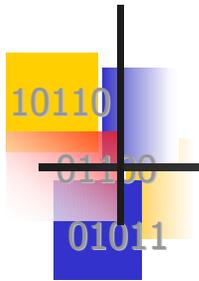
## ■ Set di istruzioni

insieme delle operazioni del linguaggio macchina (  $\leq 2^m$  ).

■ Istruzioni aritmetiche, logiche, di salto, di trasferimento dati.

■ ***Ipotesi:*** istruzione a 16 bit : 4 bit per il codice operativo e 12 bit per l'operando.



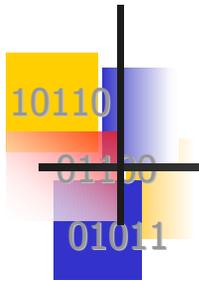


## Esecuzione delle istruzioni

- Un programma è fatto di **DATI + ISTRUZIONI**.
- I dati hanno un formato e vengono scritti in memoria di massa per non perdere il loro valore.

### Ciclo di esecuzione:

1. **Acquisizione dell'istruzione dalla memoria centrale** (fase di FETCH);
2. **Interpretazione** (analisi del codice operativo dell'istruzione);
3. **Esecuzione** (in questa fase se c'è un operando va caricato nella CPU).



## Linguaggi Assemblatori (ASSEMBLER)

- Linguaggi le cui istruzioni corrispondono univocamente a quelle del linguaggio macchina, ma sono espresse tramite nomi simbolici (parole chiave) invece che in binario.

Ad esempio :

```
READ X;  MULT X, Y;  LOAD Z;
```

## Assemblatore

- Strumento automatico (programma) che traduce le istruzioni da formato simbolico al formato binario.

```
0100  1001  →  READ X.
```

10110

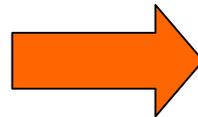
01100

01011

## In un linguaggio Assembler

- Le istruzioni vengono specificate con nomi simbolici (**parole chiave**).
  - I riferimenti alle celle di memoria (dati) sono fatti mediante nomi simbolici (**identificatori**).
  - I modi di indirizzamento vengono indicati tramite simboli.
- 
- Il programma prima di essere eseguito **deve essere tradotto** in linguaggio macchina dall' **Assemblatore**.

```
READ X
READ Y
SUB X, Y
```



```
1010 1000
1010 0101
1110 0111
```

10110

01100

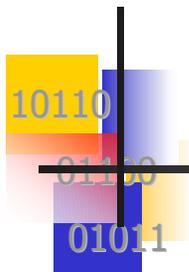
01011

# Prodotto di due numeri – Linguaggio Macchina

## Esempio

- Programma assembler che calcola il prodotto di due numeri con il corrispondente programma in linguaggio macchina

ASSEMBLER		LINGUAGGIO MACCHINA		
READ	X	0	0100	1000
READ	Y	1	0100	1001
LOADA	X	2	0000	1000
LOADB	Y	3	0000	1001
MUL		4	1000	
STOREA	X	5	0010	1000
WRITE	X	6	0101	1000
HALT		7	1101	0000
X	INT	8	0000	0000
Y	INT	9	0000	0000



10110

01100

01011

# VERSO LINGUAGGI DI ALTO LIVELLO

---

- **Linguaggio Macchina**

Conoscenza precisa dei metodi di rappresentazione e manipolazione delle informazioni utilizzate.

- **Linguaggio Macchina ed Assembler**

- Necessità di conoscere dettagliatamente le caratteristiche della macchina (registri, dimensioni dati, set di istruzioni).
- Semplici algoritmi richiedono l'uso di molte istruzioni.

- **Linguaggi di Alto Livello**

- Il programmatore può astrarre dai dettagli legati all'architettura e può esprimere i propri algoritmi in modo simbolico.
- I linguaggi di alto livello sono **indipendenti dalla macchina fisica** (astrazione).

10110

01100

01011

# VERSO LINGUAGGI DI ALTO LIVELLO

- COME ESEGUIRE UN PROGRAMMA SCRITTO IN UN LINGUAGGIO DI ALTO LIVELLO ?
- Occorre **tradurlo** nel linguaggio macchina dello specifico processore che si sta usando.
- **Due possibili modi:**
  - **Compilazione** (es. C, FORTRAN, Pascal, C++, COBOL, ...)
  - **interpretazione** (es. Basic, Perl, JavaScript, ...)

10110

01100

01011

# COMPILATORI

- I **compilatori** traducono un intero programma dal linguaggio  $L_1$  al linguaggio macchina  $L_0$  della macchina prescelta:
- **traduzione e esecuzione procedono separatamente,**
- al termine della compilazione è disponibile la versione tradotta del programma,
- la versione tradotta è però specifica di quella macchina; per eseguire il programma basta avere disponibile la versione tradotta (non serve il programma originale!).

$$L_1 \rightarrow L_0$$

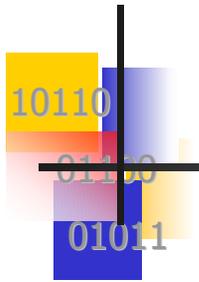
10110

01100

01011

# INTERPRETI

- Gli **interpreti** invece traducono e immediatamente eseguono il programma **istruzione per istruzione**
- traduzione ed esecuzione procedono insieme,
- al termine non vi è alcuna versione tradotta del programma originale,
- se si vuole ri-eseguire il programma occorre anche ri-tradurlo.



# FASI DI SVILUPPO DI UN PROGRAMMA

---

- Qualunque sia il linguaggio di programmazione scelto occorre:
  1. **Scrivere** il **testo del programma** e **memorizzarlo** su supporti di memoria permanenti (editing);
  2. Se il linguaggio è compilato:
    - Tradurre il linguaggio in **linguaggio macchina** (**compilazione**);
    - **Eseguire** il programma tradotto.
  3. Se il linguaggio è interpretato:
    - **Usare l'interprete per eseguire** il programma.

10110

01100

01011

## Prodotto di due numeri – Linguaggio Java

Programma Java che calcola il prodotto di due numeri:

```
class prodotto {  
    public static void main(String args[]) {  
        int x;  
        int y;  
        int p;  
        p = x * y;  
        System.out.println("Prodotto = " + p);  
    }  
}
```