FONDAMENTI DI INFORMATICA

02 - Che cos'è una codifica?

TABLE OF CONTENTS

RIPASSO

Nelle lezioni abbiamo introdotto il concetto di **bit**, come uno spazio da riempire con due simboli, 0 e 1

$$\_ \rightarrow 0, 1$$

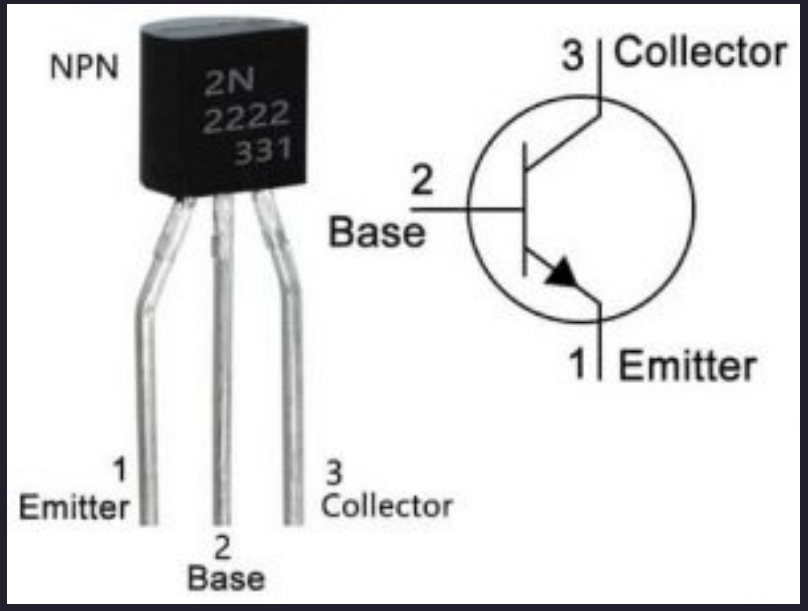
Abbiamo poi discusso il fatto che

possiamo utilizzare le sequenze di bit per rappresentare qualsiasi concetto discreto

---

$$0101011011001110 \longrightarrow \text{un concetto}$$
$$0110101011001111 \longrightarrow \text{un altro concetto}$$

Infine, abbiamo discusso di come poter rappresentare fisicamente questi bit tramite i **transistors**, che ci permettono di creare switch controllabili in modo elettronico.



A partire dai transistors possiamo costruire le architetture hardware moderne, che a loro volta permette lo sviluppo del software moderno.

$$\begin{aligned} \text{transistors} &\longrightarrow \text{logic gates} \\ &\longrightarrow \text{architetture hardware} \\ &\longrightarrow \text{sviluppo software} \end{aligned}$$

In questo video andiamo ad affrontare uno dei concetti più importanti dell'informatica.

Che cos'è una codifica?

RAPPRESENTAZIONI E CODIFICHE

Le architetture hardware ci permettono di lavorare con sequenze di bit

$$0101011011001110$$

Inizialmente però queste sequenze non hanno nessun significato.

Per riuscire a risolvere problemi interessanti il primo passo è quello di

assegnare un significato alle sequenze di bit

È proprio in questa assegnazione di significato che entra in gioco il concetto di **codifica**

una codifica è una assegnazione di significato a sequenze di bit

Per capire meglio questo concetto è importante vedere tanti esempi pratici.

Più si studia l'informatica e più si capisce quanto il concetto di **codifica** è alla base di ogni singola attività che svolgiamo in relazione ad un computer digitale.

ESEMPIO 1: ASCII

Se vogliamo scrivere e visualizzare un testo tramite un computer dobbiamo introdurre nuovi simboli oltre ai simboli 0 e 1 che già conosciamo.

Se il testo è in inglese, ad esempio, dobbiamo introdurre i simboli dell'alfabeto inglese

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Dobbiamo anche introdurre come simboli i segni di punteggiatura

, : .

e le famose cifre arabe

$$0123456789$$

L'idea è quindi quella di passare da sequenze di bit, formate dai soli simboli 0 e 1, a sequenze di simboli molte più variegate

$$101011101000\ldots \longrightarrow \text{Hello World!}$$

A tale fine è stato introdotto, nel 1963, la codifica

ASCII

$$
\begin{aligned}
\text{ASCII} &\longrightarrow \text{American} \\
&\longrightarrow \text{Standard} \\
&\longrightarrow \text{Code for} \\
&\longrightarrow \text{Information} \\
&\longrightarrow \text{Interchange}
\end{aligned}
$$

La codifica ASCII associa ad ogni sequenza di sette bit un particolare carattere.

$$
\begin{aligned}
1000001 &\longrightarrow \mathrm{A} \\
1000010 &\longrightarrow \mathrm{B} \\
1000011 &\longrightarrow \mathrm{C} \\
1000100 &\longrightarrow \mathrm{D} \\
&\vdots
\end{aligned}
$$

I caratteri presenti nella codifica ASCII sono di due tipologie

- **Control characters**
- **Printable characters**

I **control characters** servono per aggiungere meta-informazioni rispetto al file, o per controllare il comportamento dei dispositivi che processano lo stream di caratteri.

Un esempio tipico di control character è il carattere newline, molto spesso indicato con `\n`, che serve ad indicare che la riga su cui si stava scrivendo è finita, e che si vuole quindi iniziare una nuova riga.

I **printable characters** invece sono caratteri che vengono visualizzati a schermo.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
 !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Vediamo qualche esempio nella pratica

`hello world`

```
h -> 1101000
e -> 1100101
l -> 1101100
l -> 1101100
o -> 1101111
  -> 0100000
w -> 1110111
o -> 1101111
r -> 1110010
l -> 1101100
d -> 1100100
```

leonardo

```
l -> 1101100
e -> 1100101
o -> 1101111
n -> 1101110
a -> 1100001
r -> 1110010
d -> 1100100
o -> 1101111
```

CODIFICARE

L'operazione di **codificare** consiste nel prendere un oggetto astratto e nel rappresentare tale oggetto utilizzando i simboli base a nostra disposizione, che nel caso del computer sono 0 e 1.

Ad esempio, se vogliamo rappresentare la sequenza di caratteri

Leonardo

possiamo utilizzare la codifica ASCII per ottenere la rappresentazione di ciascun carattere

```
L -> 1001100
e -> 1100101
o -> 1101111
n -> 1101110
a -> 1100001
r -> 1110010
d -> 1100100
o -> 1101111
```

Mettendo tutto assieme, la sequenza di caratteri

`Leonardo`

diventa la seguente sequenza di bit

100110011001011101111110111011000011110010110010011101

DECODIFICARE

L'operazione di **decodifica** consiste nel prendere una sequenze di simboli base, come ad esempio i simboli 0 e 1, ed interpretarli tramite una codifica per andare ad estrarre l'oggetto astratto che avevamo precedentemente codificato.

Nell'esempio precedente, l'oggetto astratto era una sequenza di caratteri, e per codificarla avevamo utilizzato la codifica ASCII.

Per estrarre il significato procediamo come segue:

1. Dividiamo la sequenza in sotto-sequenze di sette bit
2. Per ciascuna sottosequenza, tramite la tabella ASCII, otteniamo il carattere rappresentato.

In pratica, dalla sequenza

$$1001100110010111011111101110110000111100101100100110$$

Otteniamo otto sotto sequenze, ciascuna di sette bit

$$\begin{array}{ccc} 1001100 & , & 1100101 \\ 1101111 & , & 1101110 \\ 1100001 & , & 1110010 \\ 1100100 & , & 1101111 \end{array}$$

E poi, tramite la tabella ASCII, ad otto caratteri

$$
\begin{aligned}
1001100 &\to \mathrm{L} \quad , \quad 1100101 \to \mathrm{e} \\
1101111 &\to \mathrm{o} \quad , \quad 1101110 \to \mathrm{n} \\
1100001 &\to \mathrm{a} \quad , \quad 1110010 \to \mathrm{r} \\
1100100 &\to \mathrm{d} \quad , \quad 1101111 \to \mathrm{o}
\end{aligned}
$$

Per ottenere la parola iniziale

Leonardo

Come possiamo vedere quindi, **codificare** e **decodificare** sono due operazioni che vanno in direzioni opposte.

Se vogliamo codificare caratteri ripresi da altri alfabeti abbiamo bisogno di una codifica che utilizza più bit.

A tale fine è stato inventato **UNICODE**, che sarà trattato in una lezione futura.

ESEMPIO 2: NUMERI

Oltre a lavorare con i **caratteri**, i computers sono progettati anche per lavorare con i **numeri** e per effettuare **complessi calcoli matematici**.

Abbiamo dunque bisogno di codificare i numeri all'interno della memoria del computer.

A tale fine c'è un modo molto utile e potente di associare ad ogni sequenza di bit uno specifico valore numerico.

Questa codifica prende il nome di

notazione posizionale in base due

Consideriamo le possibili sequenze formate da due bit

00

01

10

11

Associamo a tale sequenze i seguenti numeri

$$
\begin{aligned}
00 &\longrightarrow \text{zero} \\
01 &\longrightarrow \text{uno} \\
10 &\longrightarrow \text{due} \\
11 &\longrightarrow \text{tre}
\end{aligned}
$$

Il numero associato alla sequenza di bit può essere calcolato a partire dalle potenze di due nel seguente modo:

$$b_1 b_0 \longrightarrow b_1 \cdot 2^1 + b_0 \cdot 2^0 = b_1 \cdot 2^1 + b_0$$

Da notare che nel lato sinistro abbiamo solo dei bit, simboli privi di significato, implementati fisicamente come impulsi elettrici.

$$\underbrace{b_1 b_0}_{\text{sequenze di bit}} \longrightarrow \overbrace{b_1 \cdot 2^1 + b_0 \cdot 2^0 = b_1 \cdot 2^1 + b_0}^{\text{quantità numerica}}$$

Nel lato destro invece abbiamo quantità numeriche e matematiche con il quale possiamo effettuare dei calcoli.

Ripasso potenze di due

$$
\begin{aligned}
2^0 = 1 \quad , \quad 2^5 = 32 \\
2^1 = 2 \quad , \quad 2^6 = 64 \\
2^2 = 4 \quad , \quad 2^7 = 128 \\
2^3 = 8 \quad , \quad 2^8 = 256 \\
2^4 = 16 \quad , \quad 2^9 = 512
\end{aligned}
$$

Qualche esempio (1/4)

$$
\begin{aligned}
00 \longrightarrow & b_1 \cdot 2^1 + b_0 \cdot 2^0 \\
&= 0 \cdot 2^1 + 0 \cdot 2^0 \\
&= 0 \cdot 2 + 0 \cdot 1 \\
&= 0
\end{aligned}
$$

Qualche esempio (2/4)

$$
\begin{aligned}
01 \longrightarrow & b_1 \cdot 2^1 + b_0 \cdot 2^0 \\
& = 0 \cdot 2^1 + 1 \cdot 2^0 \\
& = 0 \cdot 2 + 1 \cdot 1 \\
& = 1
\end{aligned}
$$

Qualche esempio (3/4)

$$
\begin{aligned}
10 \longrightarrow & b_1 \cdot 2^1 + b_0 \cdot 2^0 \\
&= 1 \cdot 2^1 + 0 \cdot 2^0 \\
&= 1 \cdot 2 + 0 \cdot 1 \\
&= 2
\end{aligned}
$$

Qualche esempio (4/4)

$$
\begin{aligned}
11 \longrightarrow & b_1 \cdot 2^1 + b_0 \cdot 2^0 \\
&= 1 \cdot 2^1 + 1 \cdot 2^0 \\
&= 1 \cdot 2 + 1 \cdot 1 \\
&= 3
\end{aligned}
$$

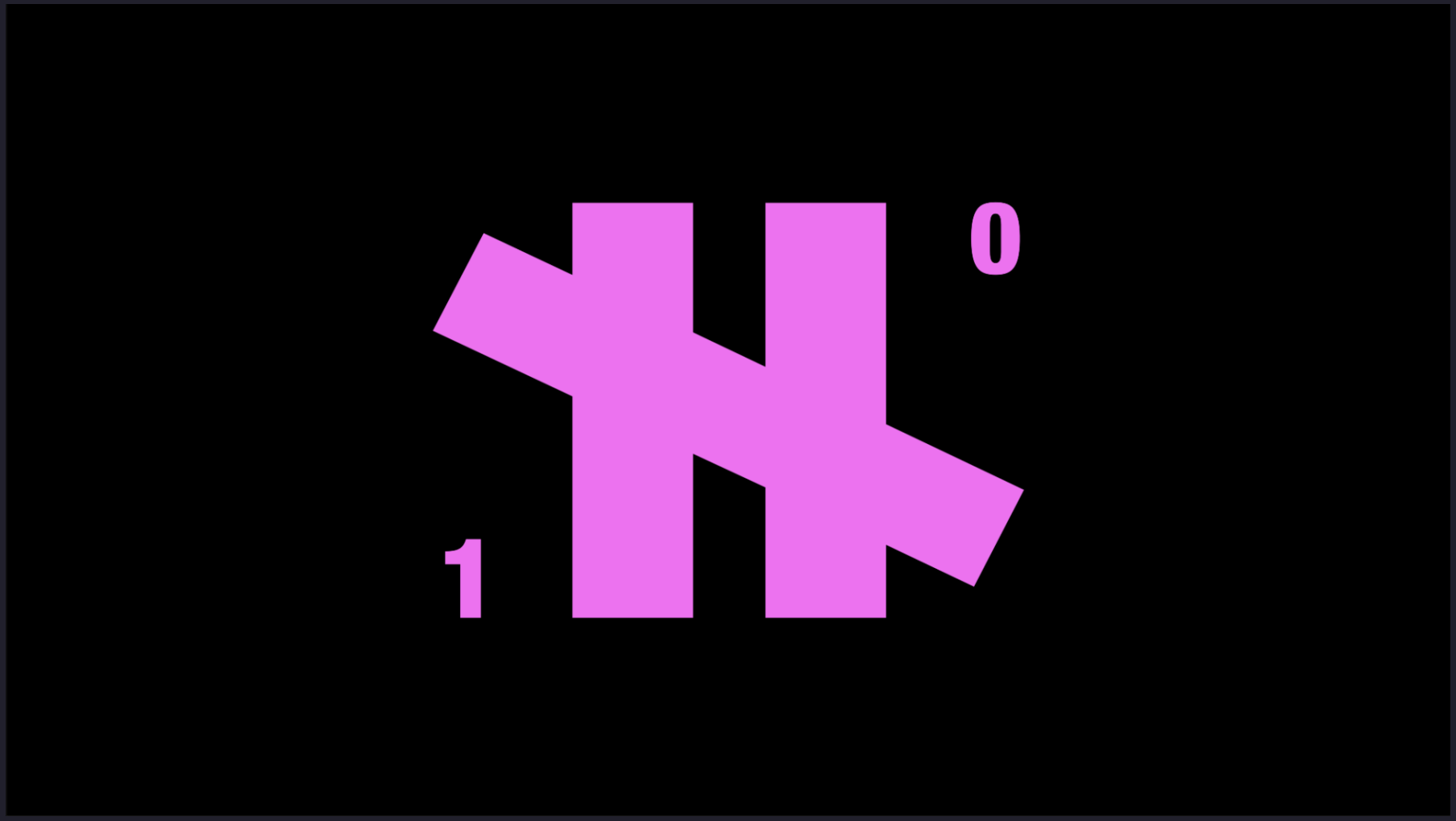
Quanto mostrato nel caso di due bit vale in generale, dobbiamo solo stare attenti ad utilizzare le giuste potenze di due a seconda della posizione del bit.

$$
\begin{aligned}
1011001 \longrightarrow & b_6 \cdot 2^6 + b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + \\
&= 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \\
&= 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\
&= 64 + 16 + 8 + 1 \\
&= 89
\end{aligned}
$$

In futuro analizzeremo con maggior dettaglio questa particolare codifica.

ESEMPIO 3: RGB

Un'altra cosa che siamo in grado di catturare tramite un computer, passando per un monitor, è il **colore**.

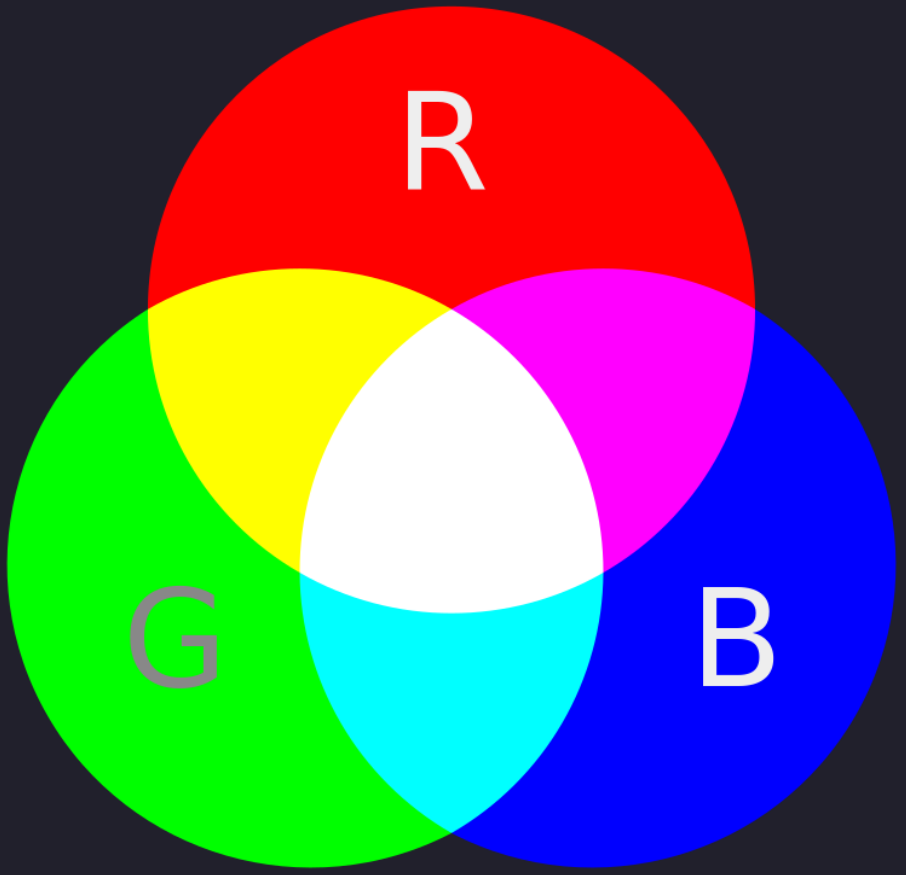


Per rappresentare un colore nella memoria del computer sono state sviluppate varie codifiche.

Tra queste, la codifica **RGB** è una delle più famose.

$$\begin{aligned} \mathrm{RGB} &\longrightarrow \mathrm{Red} \\ &\longrightarrow \mathrm{Green} \\ &\longrightarrow \mathrm{Blue} \end{aligned}$$

L'idea dietro ad RGB è quella di rappresentare un colore come combinazione di tre colori principali, che sono il rosso, il verde e il blue.



In particolare, rappresentiamo i tre componenti di un colore tramite sequenze di otto bit, che rappresentano a loro volta numeri tra 0 e 255

$$R \longrightarrow 00000000, 11111111 \longrightarrow 0, 255$$
$$G \longrightarrow 00000000, 11111111 \longrightarrow 0, 255$$
$$B \longrightarrow 00000000, 11111111 \longrightarrow 0, 255$$

Il colore $ROSSO$ è rappresentato dai seguenti numeri

$$(R, G, B) \longrightarrow (255, 0, 0)$$

o, equivalentemente, dai seguenti bit

$$(R, G, B) \longrightarrow (11111111, 00000000, 00000000)$$

Il colore $VERDE$ è rappresentato dai seguenti numeri

$$(R, G, B) \longrightarrow (0, 255, 0)$$

o, equivalentemente, dai seguenti bit

$$(R, G, B) \longrightarrow (00000000, 11111111, 00000000)$$

Il colore BLU è rappresentato dai seguenti numeri

$$(R, G, B) \longrightarrow (0, 0, 255)$$

o, equivalentemente, dai seguenti bit

$$(R, G, B) \longrightarrow (00000000, 00000000, 11111111)$$

Il colore $BIANCO$ è rappresentato dai seguenti numeri

$$(R, G, B) \longrightarrow (255, 255, 255)$$

o, equivalentemente, dai seguenti bit

$$(R, G, B) \longrightarrow (11111111, 11111111, 11111111)$$

Il colore $NERO$ è rappresentato dai seguenti numeri

$$(R, G, B) \longrightarrow (0, 0, 0)$$

o, equivalentemente, dai seguenti bit

$$(R, G, B) \longrightarrow (00000000, 00000000, 00000000)$$

Possiamo combinare diverse quantità di rosso, verde e blu per ottenere qualsiasi colore desideriamo.

ESSENZA E CONVENZIONI SOCIALI

Molto spesso è importante capire la distinzione tra l'**essenza di un concetto**, e la **convenzione sociale** necessaria per poterlo implementare nella realtà.

Consideriamo ad esempio la codifica ASCII:

L'essenza di ASCII: è una codifica, ovvero un modo per rappresentare simboli complessi in sequenze di simboli più primitivi

La convenzione sociale di ASCII: utilizza sette bit per rappresentare i caratteri dell'alfabeto inglese tramite una specifica associazione.

$$\mathrm{A} \longrightarrow \texttt{1000001}$$

Riuscire a separare la convenzione dall'essenza ci permette di esplorare nuove possibilità e di estendere la conoscenza informatica in luoghi ancora inesplorati.