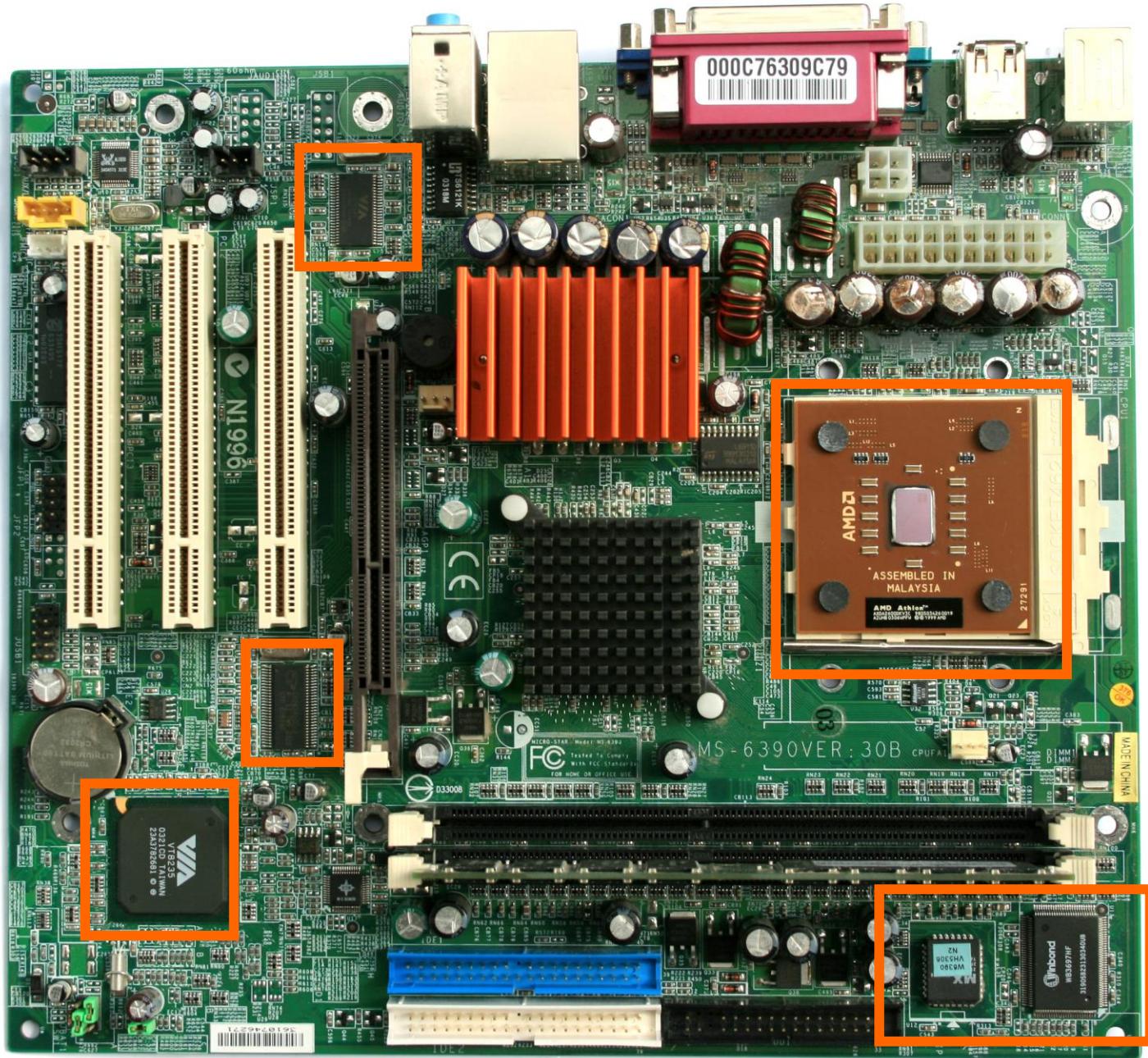


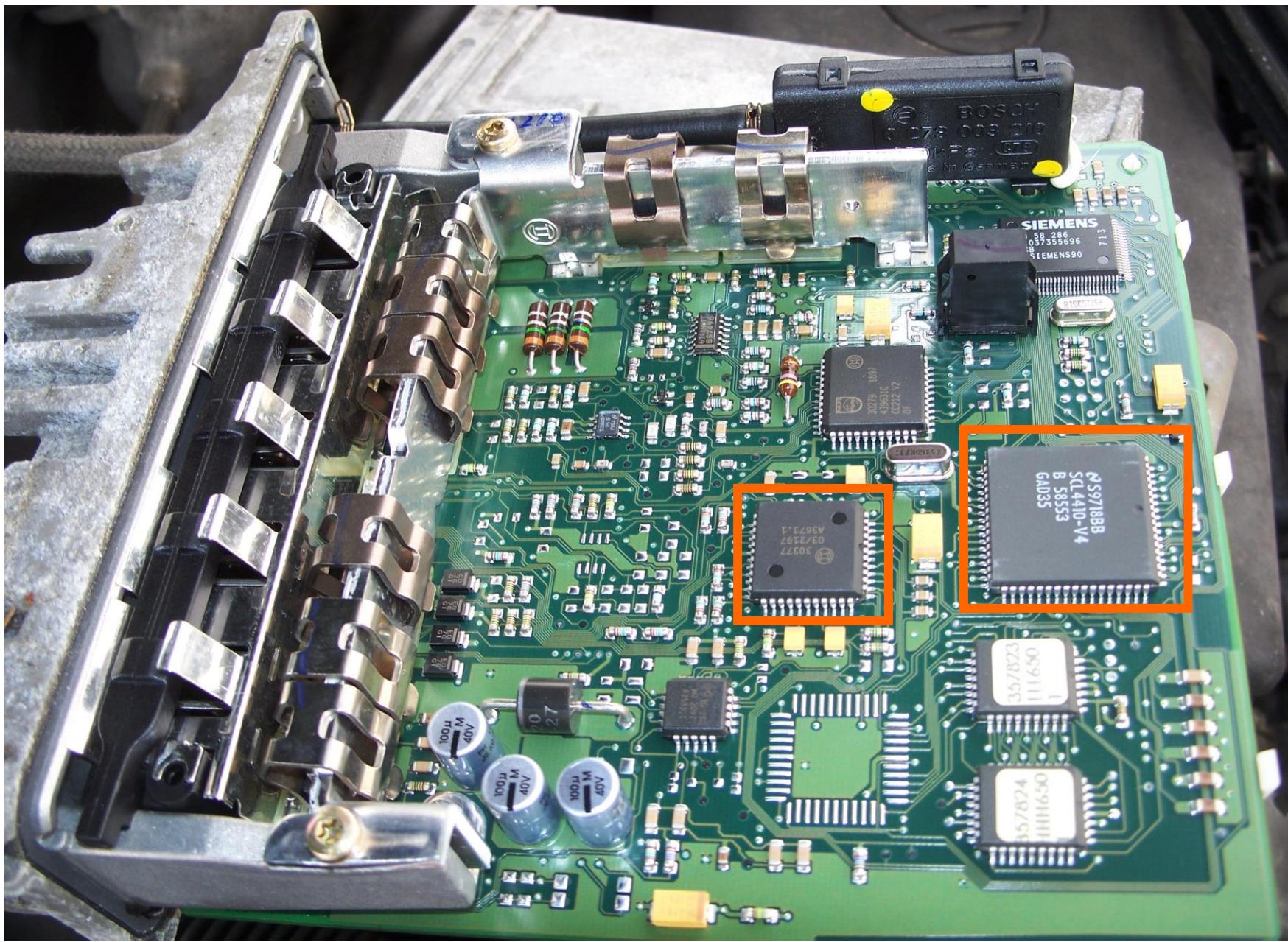
# Macchine digitali e codici

Reti Logiche T  
Ingegneria Informatica

# Cosa hanno in comune questi oggetti?

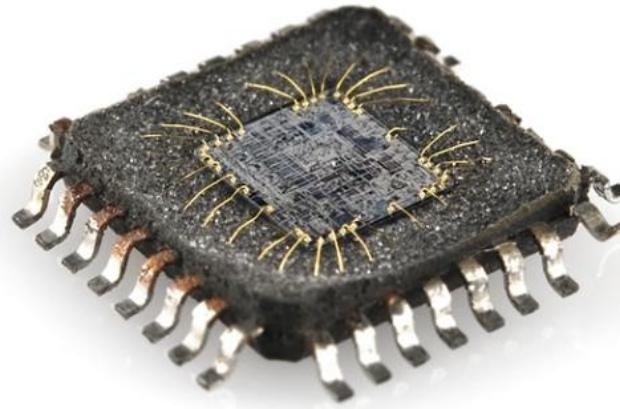
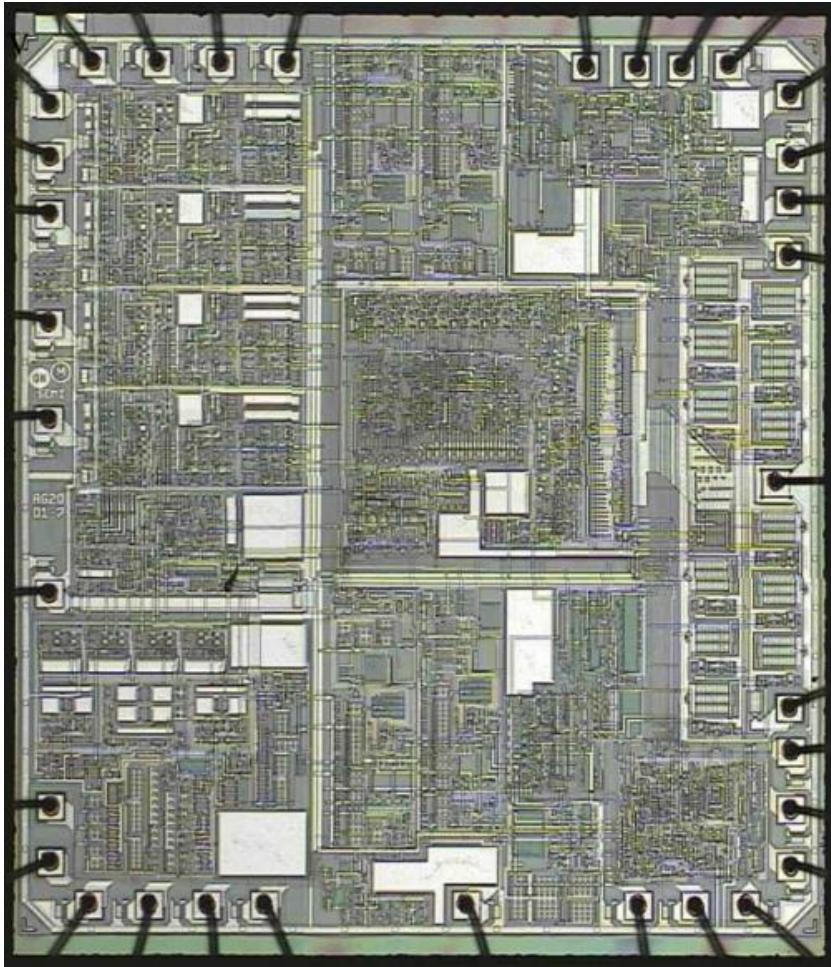




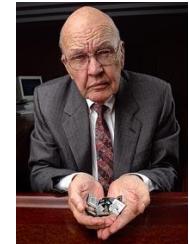




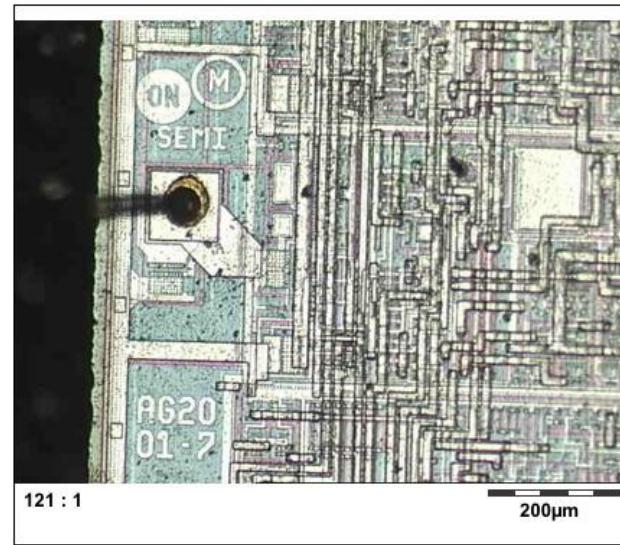
# Circuiti integrati «microchip» o «chip»



Robert Noyce  
(cofounder of  
Intel)



Jack Kilby  
(2000 Nobel  
Prize in physics)

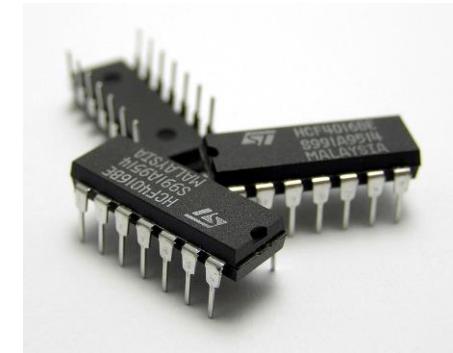


Alcune immagini da <https://www.sparkfun.com/news/384>

Immagine di Noyce da By Intel Free Press - <https://www.flickr.com/photos/intelfreepress/8267615769/sizes/o/in/photostream/>, CC BY-SA 2.0,  
<https://commons.wikimedia.org/w/index.php?curid=27929318>



# Macchine digitali



I circuiti integrati sono il modo più comune con cui oggi realizziamo macchine digitali.

## Macchina digitale: sistema **artificiale**

- progettato per immagazzinare, elaborare e comunicare **informazioni**
- impiegando **segnali digitali**, ovvero grandezze fisiche contraddistinte da un insieme **discreto** (finito, non continuo) di **valori significativi**

**Digitale**: da *digit*, cifra, usato come sinonimo di discreto in quanto nelle prime macchine le entità discrete processate erano cifre

# Scopo del corso

## Imparare a progettare e analizzare macchine digitali

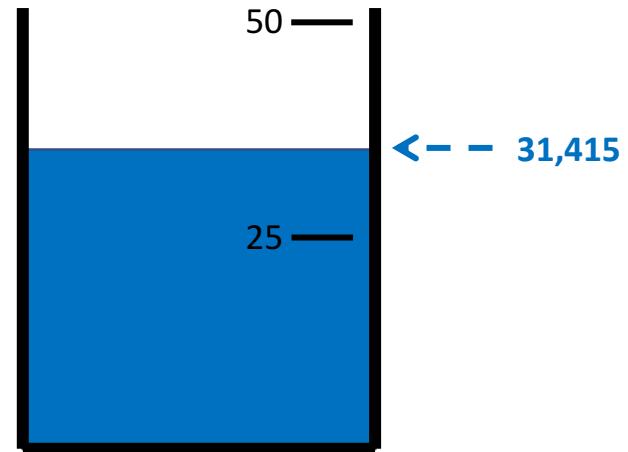
Esempio tipico di macchine digitali sono i **calcolatori**, ma in realtà la stragrande maggioranza si trova nei cosiddetti **sistemi embedded**



# Segnali analogici e digitali

Le grandezze fisiche che noi percepiamo sono **segnali analogici**, ovvero variano in modo continuo all'interno di un intervallo di valori ammissibili

Esempio: se misuro il livello di riempimento del serbatoio di un auto da 50 litri osserverò un valore continuo tra 0 e 50, per esempio 31,415 litri

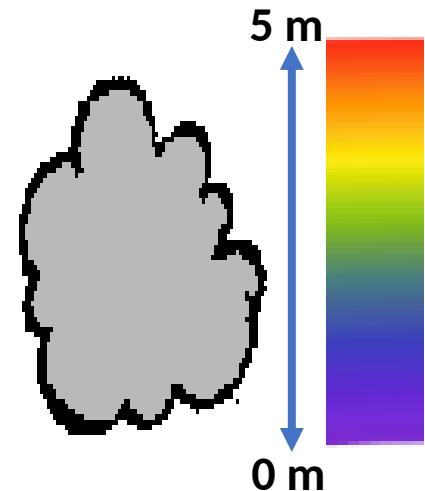
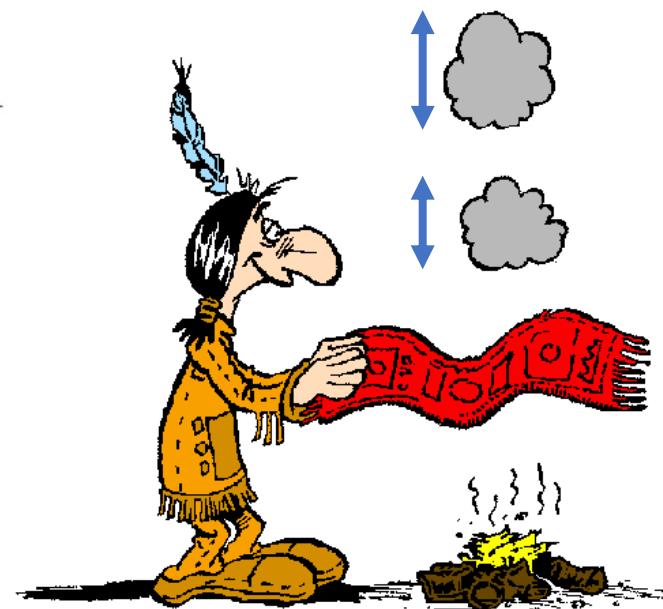


Altri esempi: pressione di uno pneumatico, peso di un oggetto, tensione ai capi di una lampadina, etc..

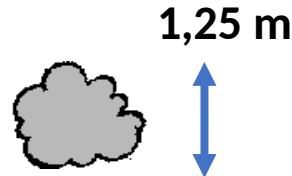
Perché usare segnali digitali per scambiare informazione se la realtà fisica è analogica?

# Esempio: segnali di fumo

- Immaginiamo di essere dei nativi americani e di dover comunicare con un'altra tribù attraverso **segnali di fumo**
- La nostra **macchina** è rappresentata dal fuoco e dal tappeto per coprirlo.
- L'**informazione** che vogliamo scambiarci sono i colori in cui desideriamo le nuove tende che l'altra tribù ci sta preparando.
- Il **segnale** che usiamo per codificare l'informazione è la quantità di fumo in una nuvola, per esempio la sua «altezza». Il mio tappeto trattiene abbastanza fumo da generare nuvole alte fino a 5 metri, ma non di più.
- Come è consigliabile codificare l'informazione nel segnale?



# Segnali analogici



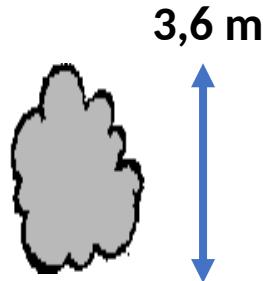
Voglio le tende blu  
oltremare, farò una  
nuvola alta 1,25 metri



Nuvola alta 1,25 metri!  
Tende blu oltremare



# Segnali analogici



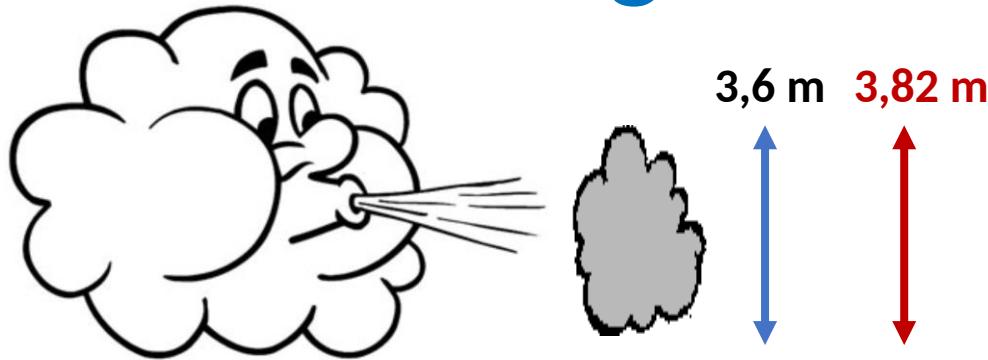
Mio cugino vuole le  
tende giallo canarino,  
farò una nuvola alta  
3,6 metri



Nuvola alta 3,6 metri!  
Tende giallo canarino



# Segnali analogici

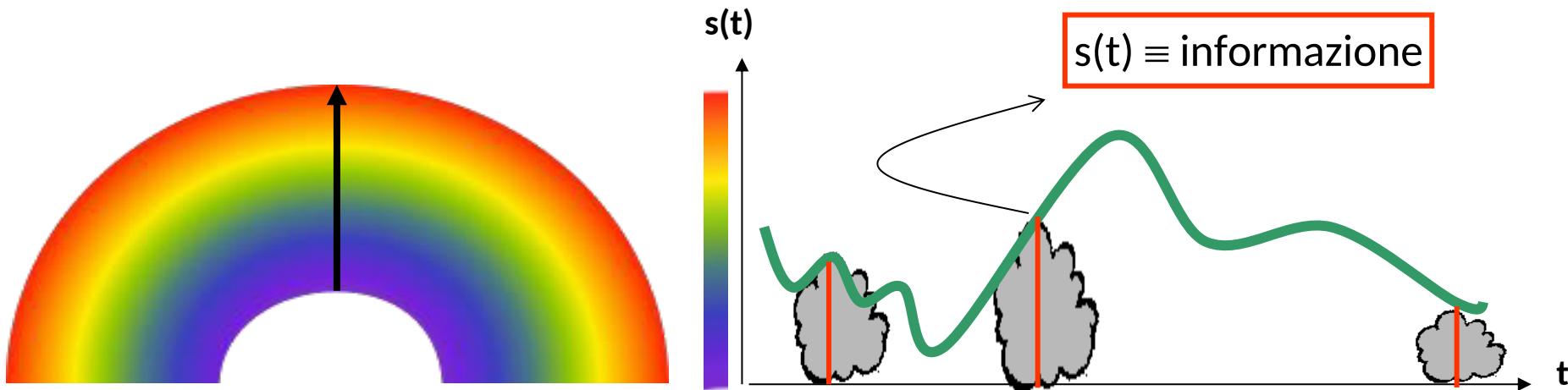


Mio cugino vuole le  
tende giallo canarino,  
farò una nuvola alta  
3,6 metri

Nuvola alta 3,82 metri!  
Tende ocra

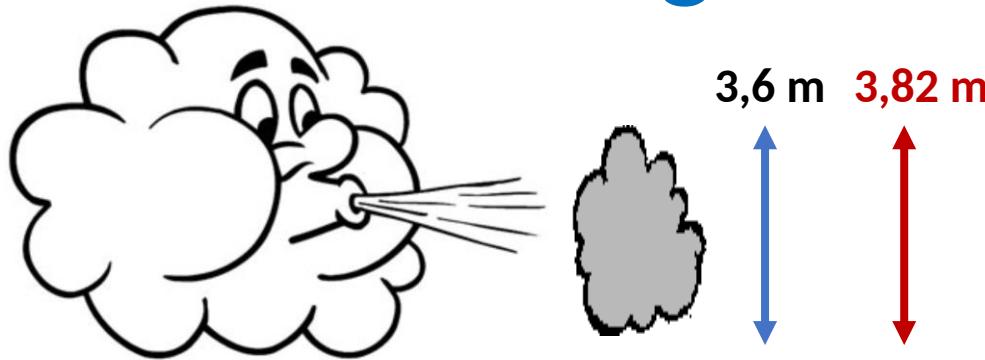


# Segnali analogici



- In un **segnale analogico** l'informazione è rappresentata da ogni possibile **valore** della grandezza fisica (ovvero l'altezza di ogni nuvola di fumo).
- In teoria, posso rappresentare **infinite informazioni** con un unico segnale (1 metro = rosso rubino, 1.1 = bordeaux, 1.2 = viola, 1.3 = prugna, etc...). In pratica, **un piccolo disturbo sul segnale distrugge l'informazione** (se sbaglio di poco a calcolare quanto fumo accumulare prima di far partire una nuvola o se non sono preciso nello stimarne l'altezza da lontano, potrei ricevere o produrre le tende del colore sbagliato). Inoltre è necessario avere **dispositivi estremamente sofisticati** (e quindi costosi e più soggetti a rotture) per produrre e riconoscere il segnale con la precisione richiesta.

# Segnali digitali

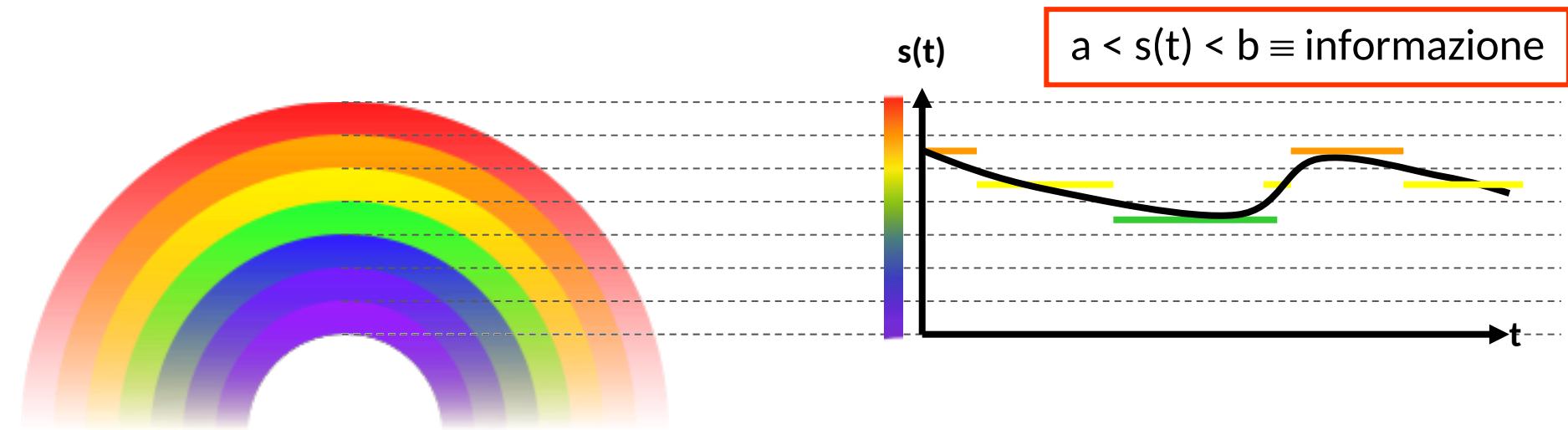


Mio cugino vuole le  
tende gialle, farò una  
nuvola alta  
tra 3 e 4 metri

Nuvola alta 3,82 metri!  
**Tende gialle**



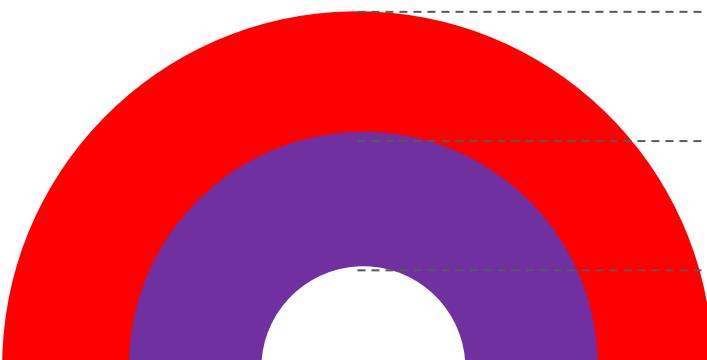
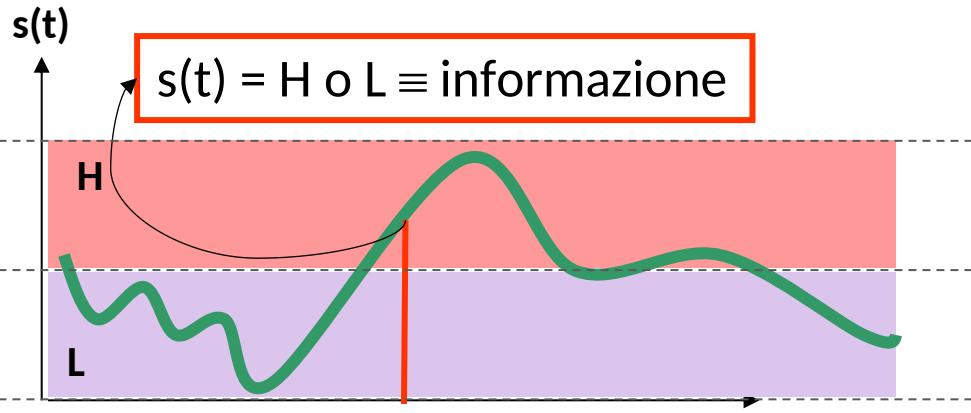
# Segnali digitali



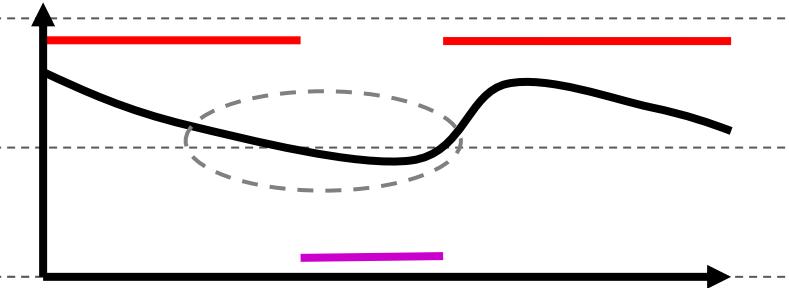
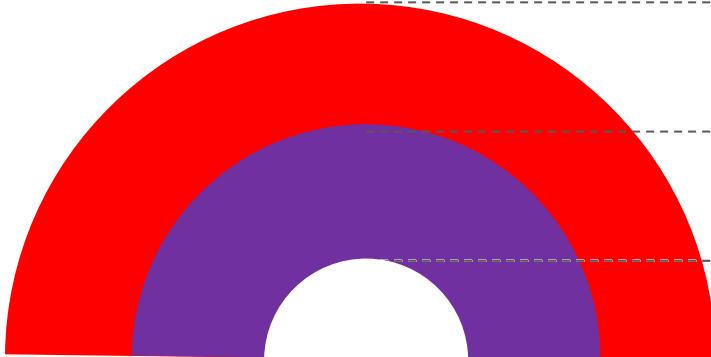
- In un **segnale digitale**, invece, è l'intervallo in cui si trova la grandezza a rappresentare l'informazione. Per esempio, se la nuvola è alta tra 0 e 1 metri = viola, tra 1 e 2 metri = blu, etc..)
- Accetto di rappresentare **meno informazioni** con una stessa grandezza fisica per ottenere maggior robustezza nel trasferire l'informazione e minor complessità (e costo) dei dispositivi necessari
- Possiamo aumentare gli intervalli/colori se necessario, a scapito della robustezza

# Segnali binari

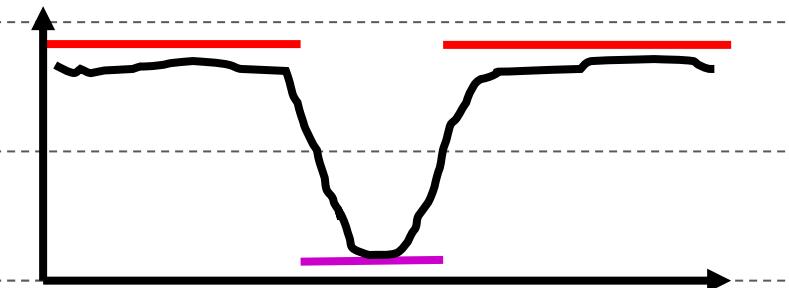
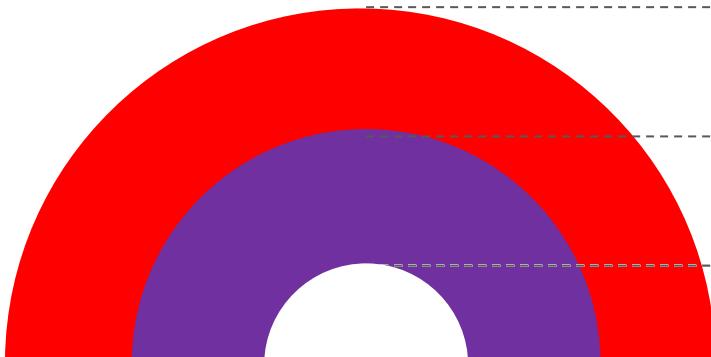
- Per avere la **massima robustezza** al rumore e utilizzare dispositivi più semplici (e quindi più affidabili e più economici) possibili, posso **ridurre al minimo il numero di intervalli** considerati
- I **segnali binari** sono segnali digitali con due soli intervalli
- L'informazione è data dal sapere se il segnale è sopra o sotto una determinata soglia. I due livelli del segnale digitale sono convenzionalmente indicati con **H(igh)** e **L(ow)**



# Segnali binari

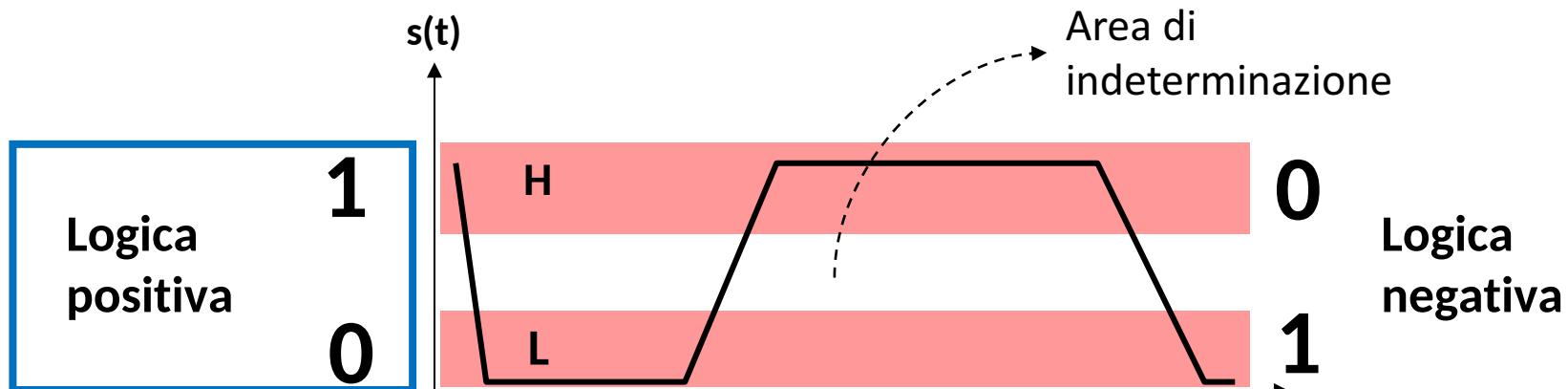


- Per avere una alta robustezza al rumore non bastano due intervalli, devo anche gestire il segnale analogico sottostante con attenzione.
- In generale, il segnale deve mantenersi prossimo al suo valore massimo o minimo e le transizioni tra questi due valori devono essere le più rapide possibili.



# Variabili binarie o bit

- Per descrivere a livello logico un segnale binario si utilizzano variabili, dette variabili binarie o **bit (binary digit)**, che possono assumere i due soli valori «0» e «1»
- E' importante ricordare che, nonostante il nome inglese, i bit nel caso generale **non sono numeri**, ma solo **valori logici** (simboli) che indicano se il segnale è sopra o sotto soglia. Invece di «0» e «1» potrei usare «falso» e «vero», o «Boris» e «Stanis».
- L'associazione dei due simboli logici binari **0** e **1** ai due livelli del segnale è arbitraria: se uso il livello **H** per rappresentare **1** si parla di **logica positiva**, se uso il livello **L** per rappresentare **0** si parla di **logica negativa**. Nel seguito assumeremo logica positiva.
- Per aumentare ulteriormente la robustezza, nei segnali binari vi è tipicamente anche una **zona di indeterminazione** in cui non è definito come verrà interpretato il segnale.



# Codici binari

- Per mantenere la robustezza del segnale binario, ma essere in grado di rappresentare più di due informazioni, usiamo **una stringa** (una sequenza) di segnali binari
- Le stringhe su cui chi genera e chi riceve l'informazione si sono **accordati** formano un **codice binario**
- Nell'esempio dei segnali di fumo, posso accendere  $n$  fuochi e **in parallelo** generare una nuvola alta o bassa da ognuno di essi



- Quanti fuochi mi servono per rappresentare 7 colori?

# Codifica binaria dell'informazione

- Dal calcolo combinatorio sappiamo che tutte le possibili **permutazioni** di 2 elementi in  $n$  posizioni sono  $2^n$
- Per esempio tutte le stringhe di 3 bit sono  $8 = 2^3$ , ovvero ‘111’, ‘110’, ‘101’, ‘100’, ‘011’, ‘010’, ‘001’, ‘000’
- Quindi per rappresentare 7 colori, devo avere
$$2^n \geq 7 \rightarrow n \geq \lceil \log_2 7 \rceil = \lceil 2,807 \rceil = 3$$
- Per calcolare  $n$  senza bisogno di risolvere il logaritmo, basta considerare la prima **potenza di 2** maggiore del numero richiesto, il suo esponente è il numero di bit necessari

$$2^n \geq 7 \rightarrow 2^n \geq 8 = 2^3 \rightarrow n \geq 3$$

# Potenze di 2

- Le potenze di 2 svolgono un ruolo importante nelle reti logiche e nell'informatica in generale. È utile conoscerle a memoria almeno fino a 16.

$n$	0	1	2	3	4	5	6	7	8
$2^n$	1	2	4	8	16	32	64	128	256

$n$	9	10	11	12	13	14	15	16
$2^n$	512	1024	2048	4096	8192	16384	32768	65536

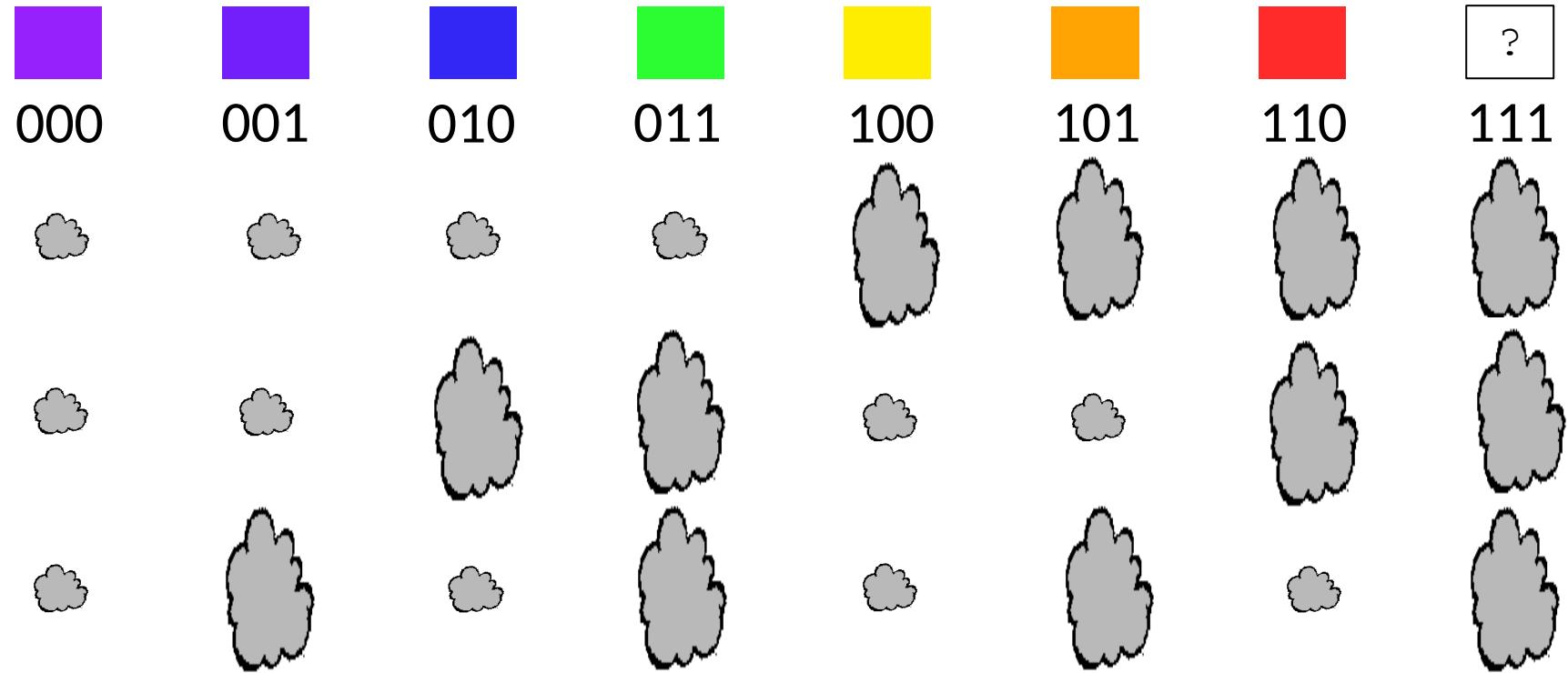
# Configurazioni di $n$ bit

1 bit	2 bit	3 bit	4 bit	5 bit	...
2 conf.	4 conf.	8 conf.	16 conf.	32 conf.	
$b_0$	$b_1 \mid b_0$	$b_2 \mid b_1 \mid b_0$	$b_3 \mid b_2 \mid b_1 \mid b_0$	$b_4 \mid b_3 \mid b_2 \mid b_1 \mid b_0$	
0	0   0	0   0   0	0   0   0   0	0   0   0   0   0	
1	0   1	0   0   1	0   0   0   1	0   0   0   0   1	
	1   0	0   1   0	0   0   1   0	0   0   0   1   0	
	1   1	0   1   1	0   0   1   1	0   0   0   1   1	
		1   0   0	0   1   0   0	0   0   1   0   0	
		1   0   1	0   1   0   1	0   0   1   0   1	
		1   1   0	0   1   1   0	0   0   1   1   0	
		1   1   1	0   1   1   1	0   0   1   1   1	
			1   0   0   0	0   1   0   0   0	
			1   0   0   1	0   1   0   0   1	
			1   0   1   0	0   1   0   1   0	
			1   0   1   1	0   1   0   1   1	
			1   1   0   0	0   1   1   0   0	
			1   1   0   1	0   1   1   0   1	
			1   1   1   0	0   1   1   1   0	
			1   1   1   1	0   1   1   1   1	
				1   0   0   0   0	
				1   0   0   0   1	
				1   0   1   0   0	
				1   0   1   0   1	
				1   0   1   1   0	
				1   0   1   1   1	
				1   1   0   0   0	
				1   1   0   0   1	
				1   1   0   1   0	
				1   1   0   1   1	
				1   1   1   0   0	
				1   1   1   0   1	
				1   1   1   1   0	
				1   1   1   1   1	
				...	

Un modo per elencarle tutte è replicarne la **regolarità**, di nuovo basata su **potenze di 2**:

- Il primo bit alterna «0» e «1»
- Il secondo alterna 2 volte «0» e 2 volte «1»
- Il terzo alterna 4 volte «0» e 4 volte «1»
- ...

# Esempio di codice binario

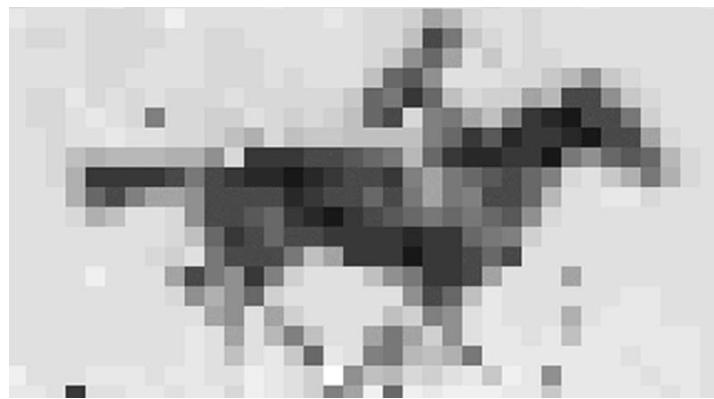


La corrispondenza tra informazione (in questo caso colori) e configurazioni (stringhe) di bit è arbitraria, purché condivisa tra sorgente e destinazione.

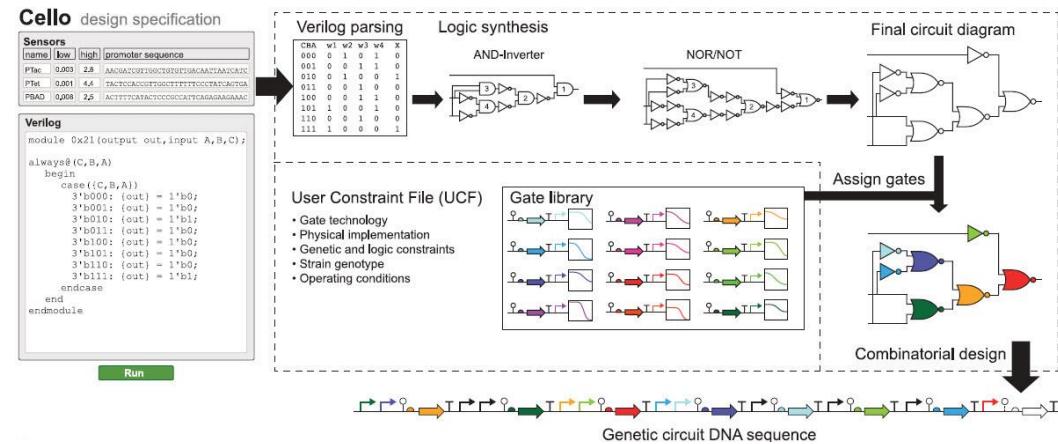
Possono esistere configurazioni non usate (111 in questo esempio)

# Codici digitali oltre l'informatica

- La robustezza della codifica digitale dell'informazione non è sfruttata solo nelle macchine digitali create dall'uomo
- Il **DNA**, per esempio, è una sequenza di 4 possibili basi, Adenina, Timina, Guanina e Citosina. A tutti gli effetti un codice digitale per memorizzare e comunicare informazioni.



Seth L. Shipman et al., **CRISPR–Cas encoding of a digital movie into the genomes of a population of living bacteria**  
*Nature* volume 547, pages 345–349 (20 July 2017)

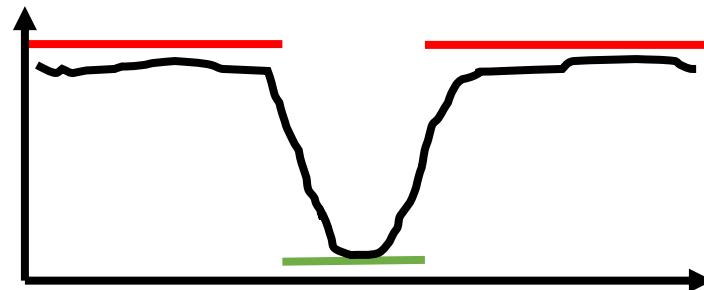


Alec A. K. Nielsen et al., **Genetic circuit design automation**  
*Science* Vol. 352, Issue 6281, (01 Apr 2016)



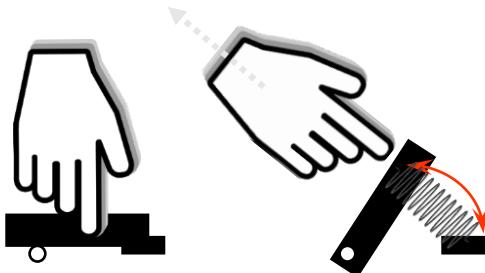
# Interruttori

- Un bit può essere efficacemente generato e rappresentato (oltre che dall'altezza di una nuvola...) dalla posizione di un **interruttore**.



# Interruttore meccanico

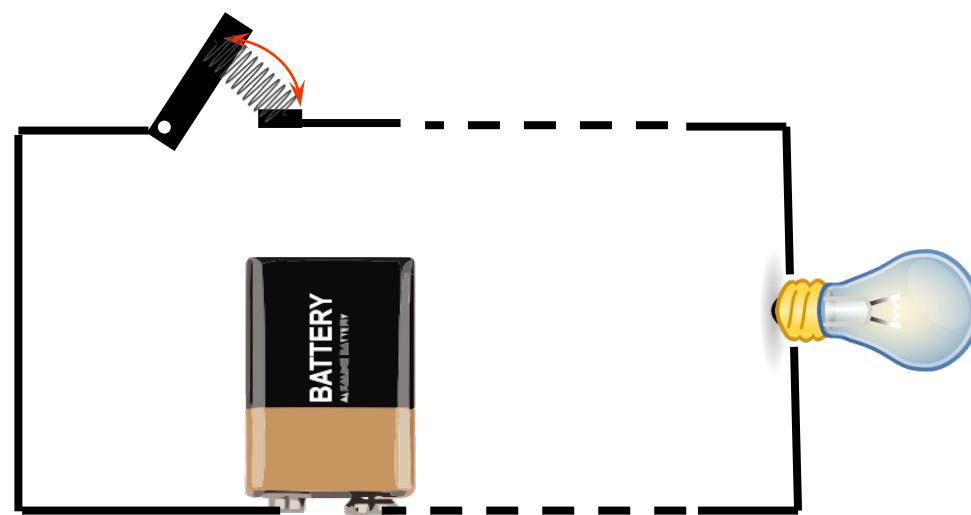
- Facciamo fare un salto tecnologico alle nostre tribù di nativi americani, posando un cavo elettrico tra i due campi
- **Tutte le considerazioni** fatte fino a qui su informazione, robustezza, e economicità del segnale binario **rimangono valide, è cambiata solo la tecnologia** sottostante: per scegliere tra 7 colori avrò ancora bisogno di almeno 3 interruttori e lampadine, etc...



Ingresso: {Chiuso,Aperto}



Uscita: {ON,OFF}

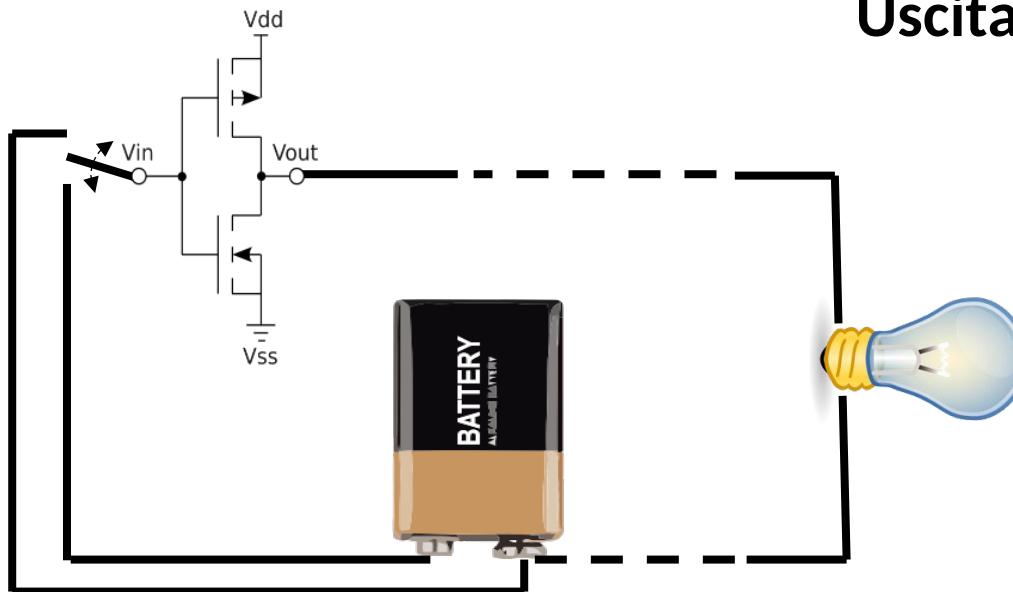


# Interruttore elettronico

- Un altro salto tecnologico si ottiene realizzando gli interruttori non in modo meccanico ma come **interruttori elettronici (transistor)**
- Sono ordini di grandezza più veloci ed affidabili, ma **le considerazioni sui segnali che abbiamo fatto valgono ancora, sono indipendenti dalla tecnologia.**

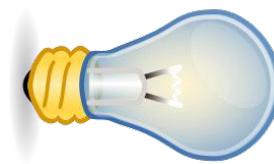
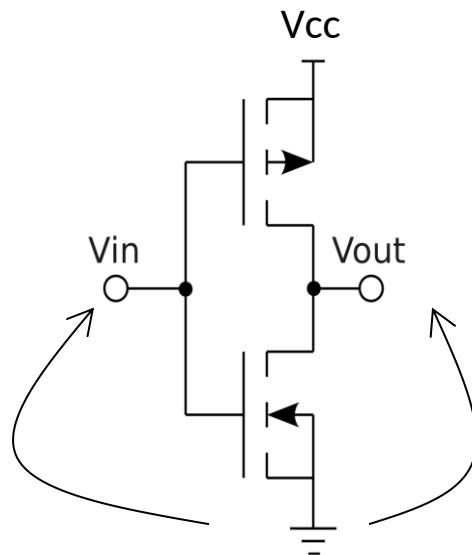
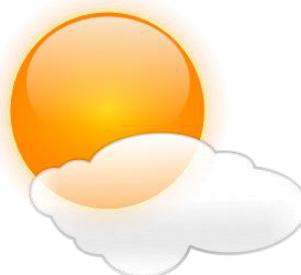
Ingresso: {0 Volt, 5 Volt}

Uscita: {ON,OFF}



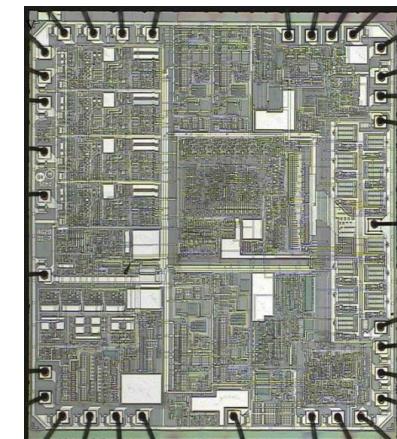
# Interruttore elettronico

- Il grande vantaggio degli interruttori elettronici è di avere ingresso ed uscita della stessa natura, entrambe elettriche: è possibile usare l'uscita di un transistor per pilotarne altri, creando delle reti di interruttori che si influenzano e commutano senza intervento umano, e che possono essere realizzate in un circuito integrato
- Esempio: sensore crepuscolare che accende lampione



Ingresso: {0, 5 Volt}

Uscita: {ON,OFF}



# Moore's Law

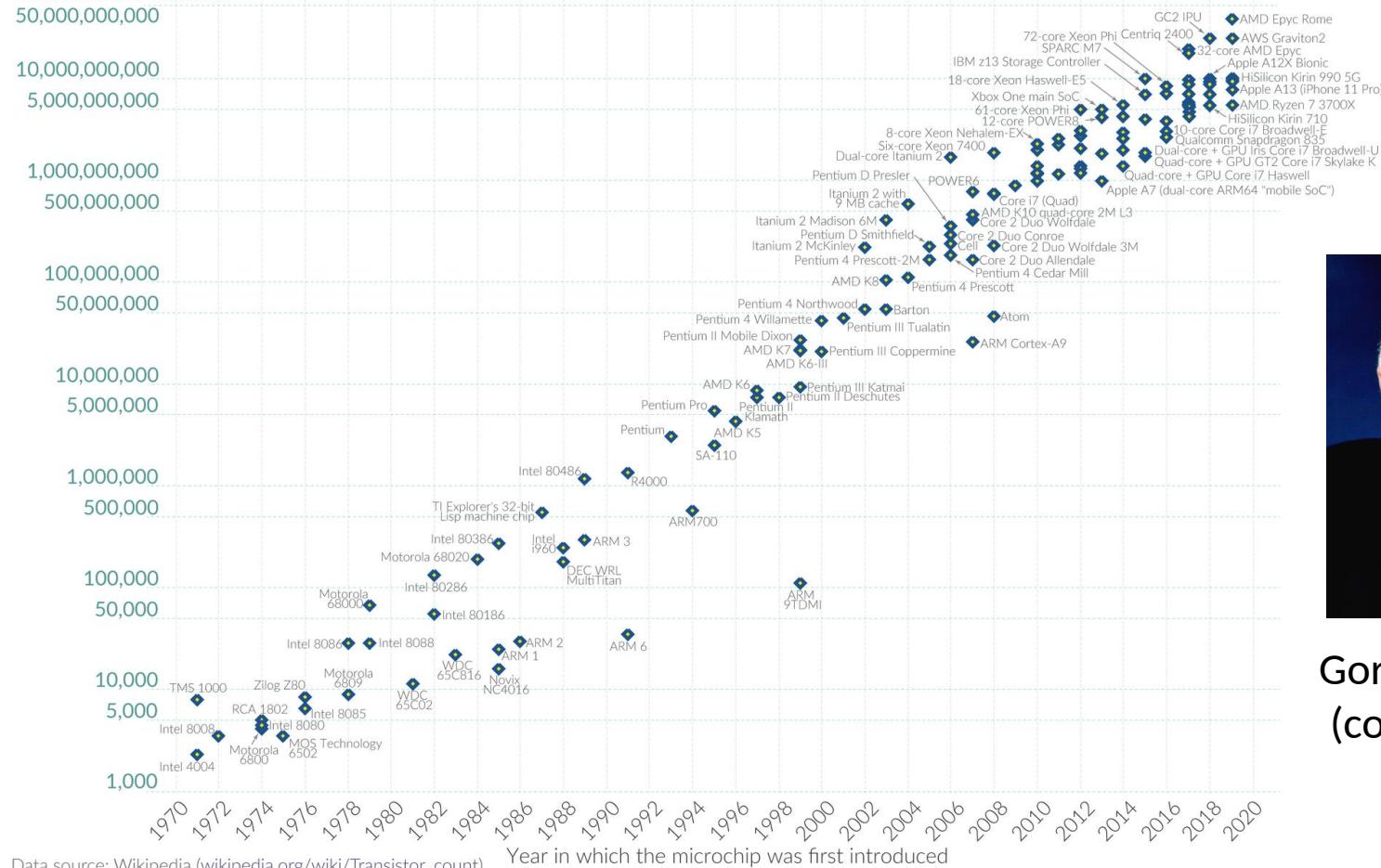
Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.

This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World  
in Data

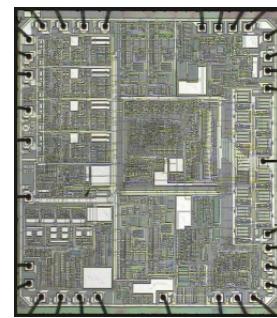
## Transistor count



Gordon Moore  
(cofounder of  
Intel)

By Max Roser, Hannah Ritchie - <https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png>, CC BY 4.0, <https://commons.wikimedia.org/w/index.php?curid=98219918>

# Evoluzione della tecnologia



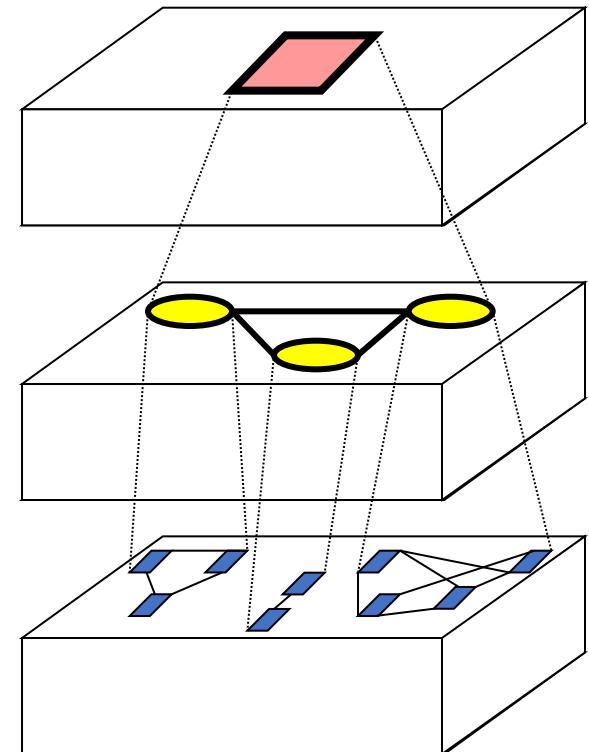
- La prima **CPU Intel (4004)** del **1971** conteneva circa **2300 transistor** in grado di **commutare** (*cambiare stato «aperto/chiuso»*) circa **700 mila volte ogni secondo**.
- Il **System-on-a-Chip (SoC) Apple M2 Ultra (2023)** contiene quasi **134 miliardi** di **transistor**. Ciascuno può commutare **miliardi** di volte al secondo
- La **GPU Nvidia B100 Blackwell (2024)** contiene **208 miliardi** di **transistor**. Anche in questo caso in grado di commutare **miliardi** di volte al secondo
- Utilizzando (tanti) semplici interruttori è possibile eseguire operazioni molto complesse
- **Ma come possiamo progettare sistemi così complessi?**

[https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count)

[https://www.reddit.com/r/dataisbeautiful/comments/cynql1/moores\\_law\\_graphed\\_vs\\_real\\_cpus\\_gpus\\_1965\\_2019\\_oc/?utm\\_source=share&utm\\_medium=ios\\_app](https://www.reddit.com/r/dataisbeautiful/comments/cynql1/moores_law_graphed_vs_real_cpus_gpus_1965_2019_oc/?utm_source=share&utm_medium=ios_app)

# Livelli di astrazione

- Lavorare ad uno specifico livello di astrazione è una pratica standard di ogni ambito ingegneristico, in cui la descrizione di un sistema complesso è sempre articolata su **più livelli**
- Ogni livello individua **componenti “primitivi”** la cui struttura è definita nel livello sottostante e di cui interessa solo il **comportamento («cosa può fare»), definito dalla sua interfaccia**
- Scopo di un livello è definire la **struttura («come è fatto»)** dei componenti primitivi usati dal livello soprastante
- Esplorando i livelli della gerarchia **dall’alto verso il basso, aumenta il numero di entità, ma diminuisce la complessità** di ciascuna di esse



# Livelli di astrazione - esempio

main.c

```
#include <stdio.h>

int main(void) {
    int c = 0;
    scanf(<<%d>>, &c);
    printf(<<Hello world number %d>>, c);
    return 0;
}
```

```
int scanf(char* format, ...){
```

```
...
```

```
c = getchar()
```

```
...
```

```
}
```

```
    int printf(char* format, ...){
```

```
    ...
```

```
        int getchar(void){
```

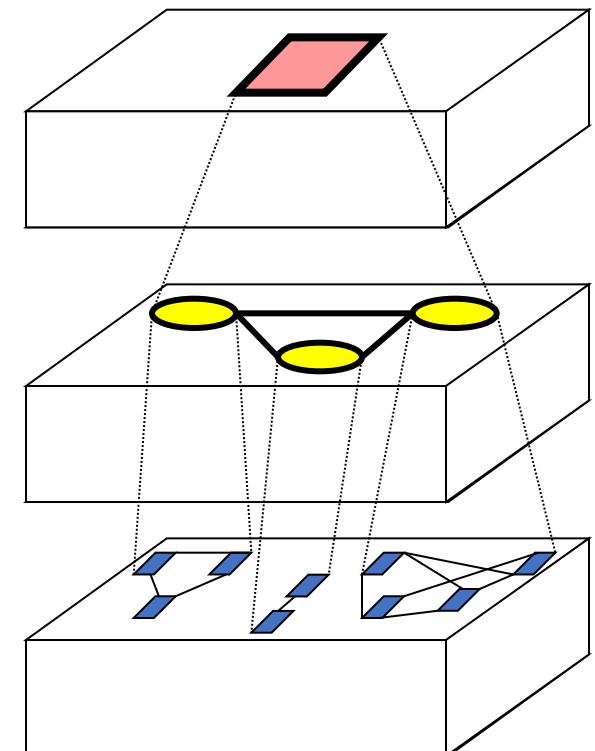
```
            <OS dependent instructions>
```

```
        }
```

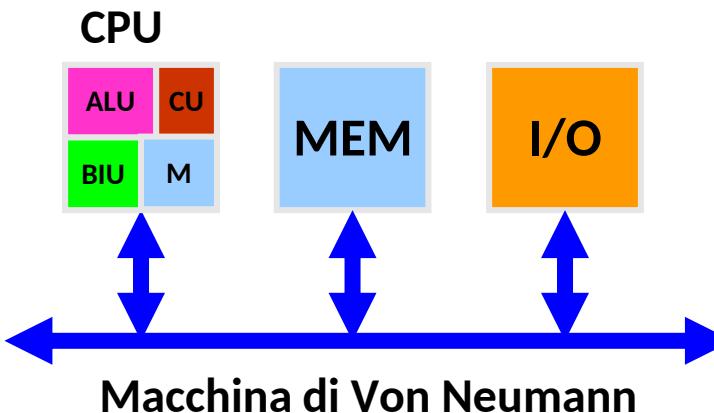
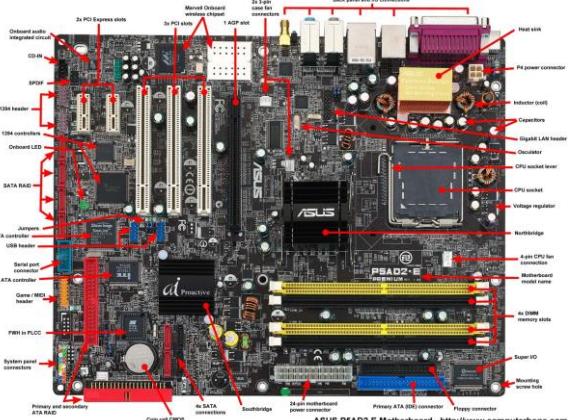
System call #1

System call #2

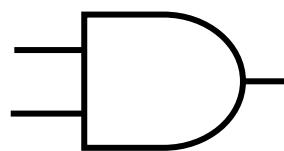
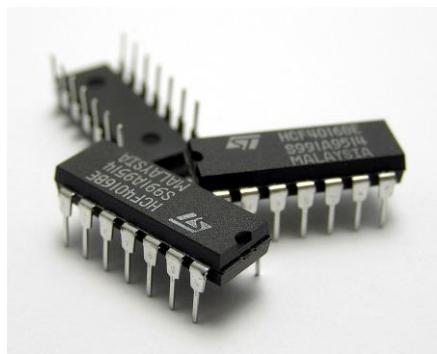
System call #3



# Livelli di astrazione - esempio

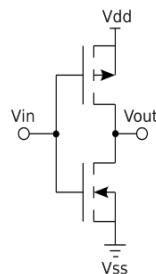
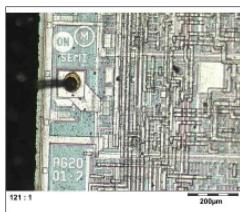


**Livello Architettonico**  
**[Calcolatori Elettronici T]**



Gate

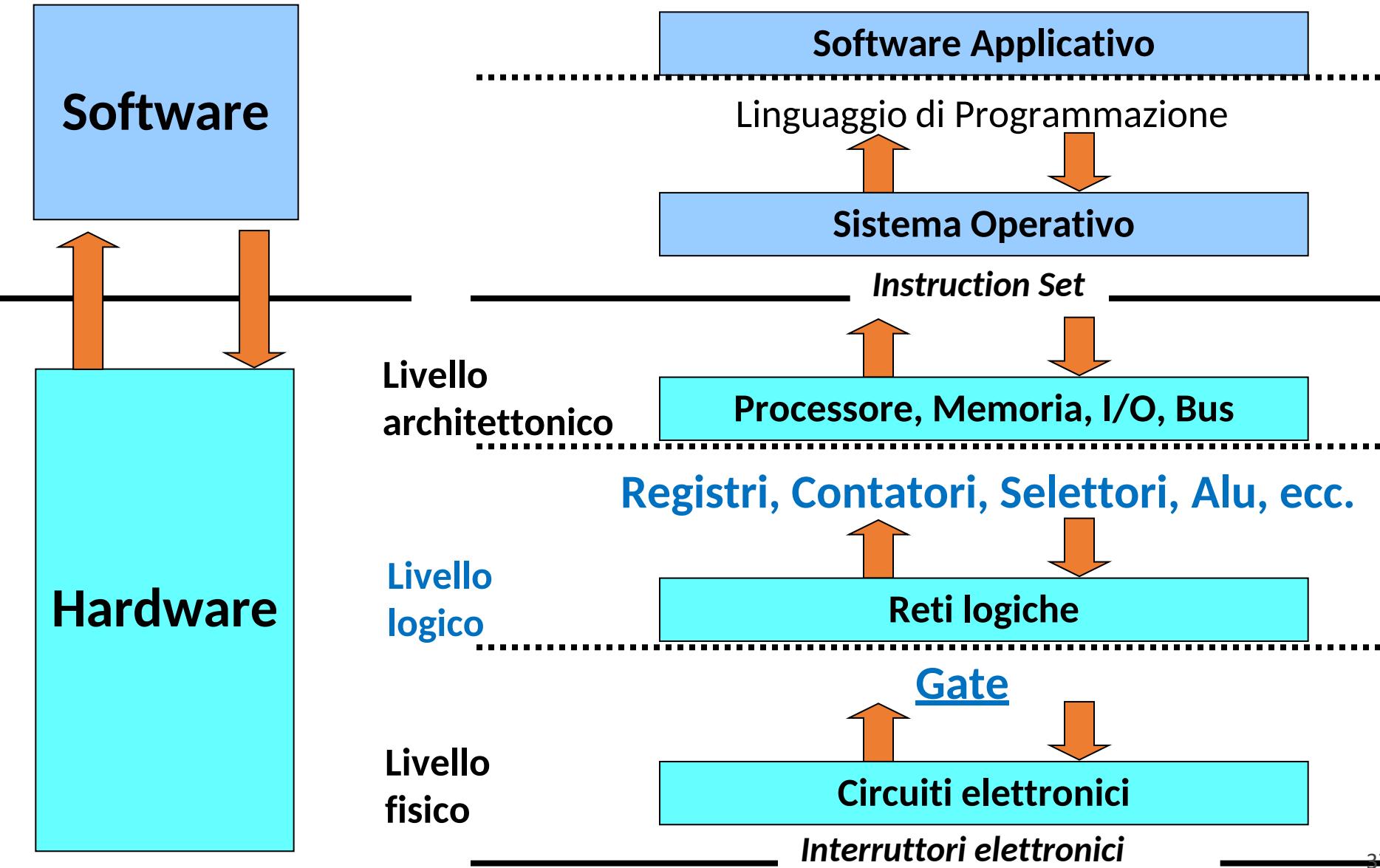
**Livello logico**  
**[Reti Logiche T]**



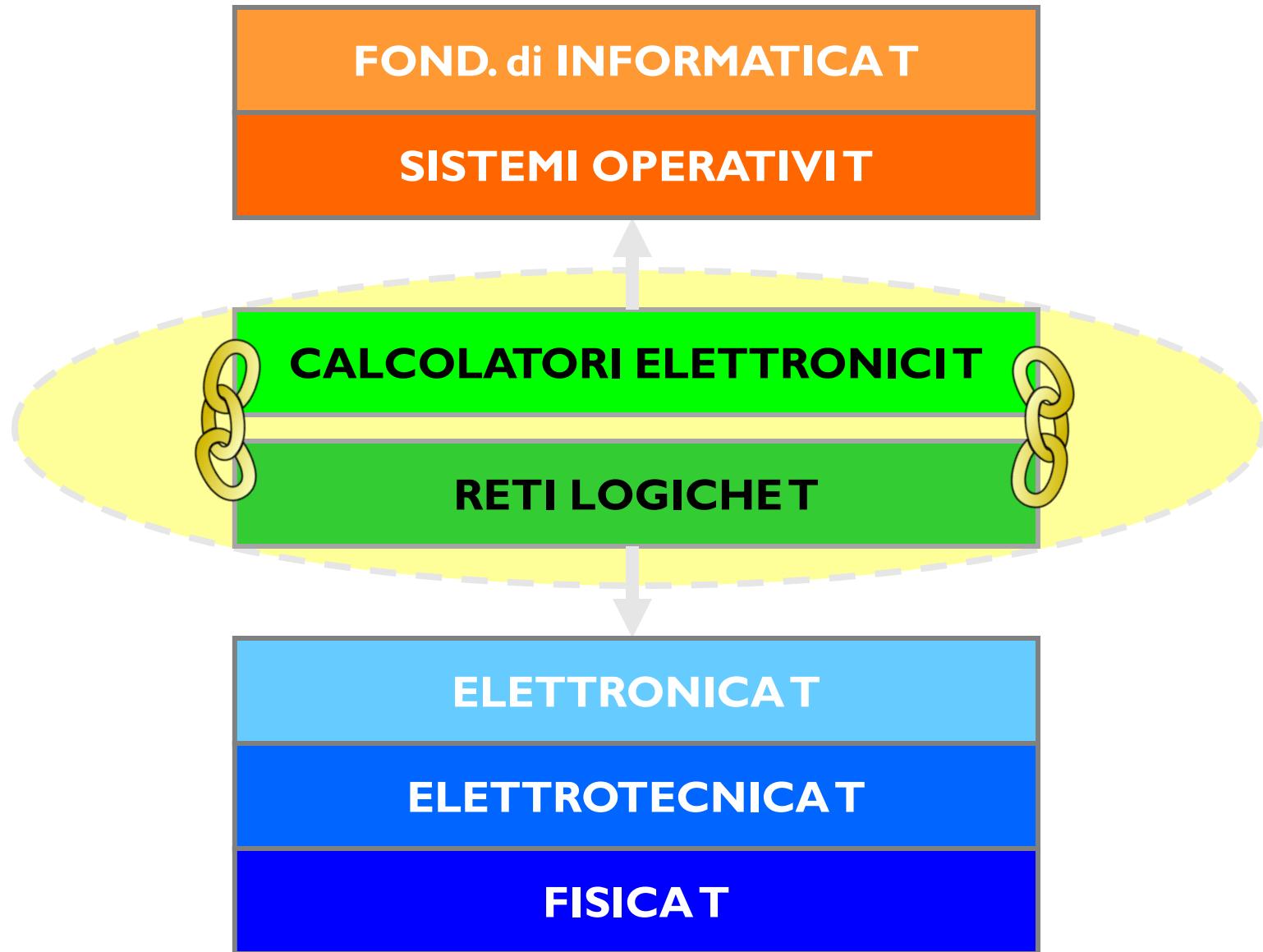
Transistor

**Livello fisico**  
**[Elettronica T]**

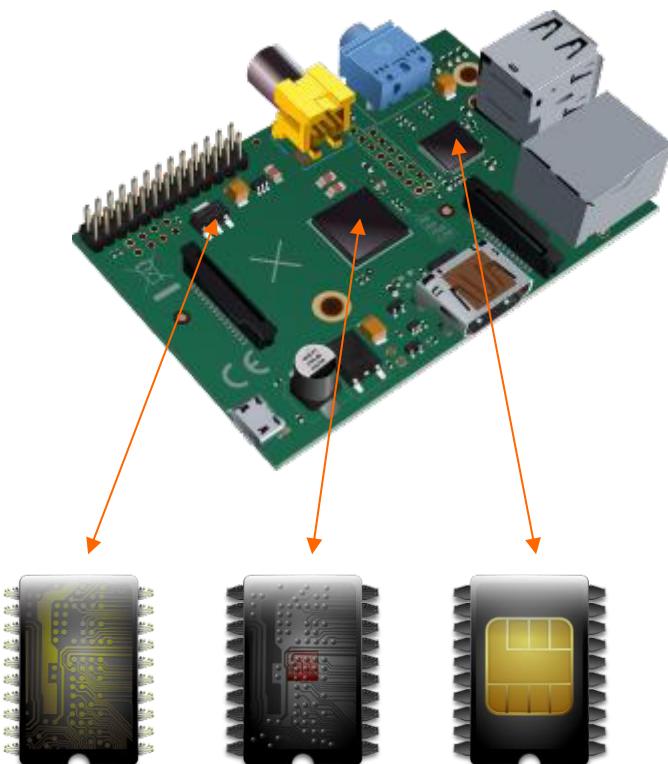
# Livelli di astrazione in Informatica



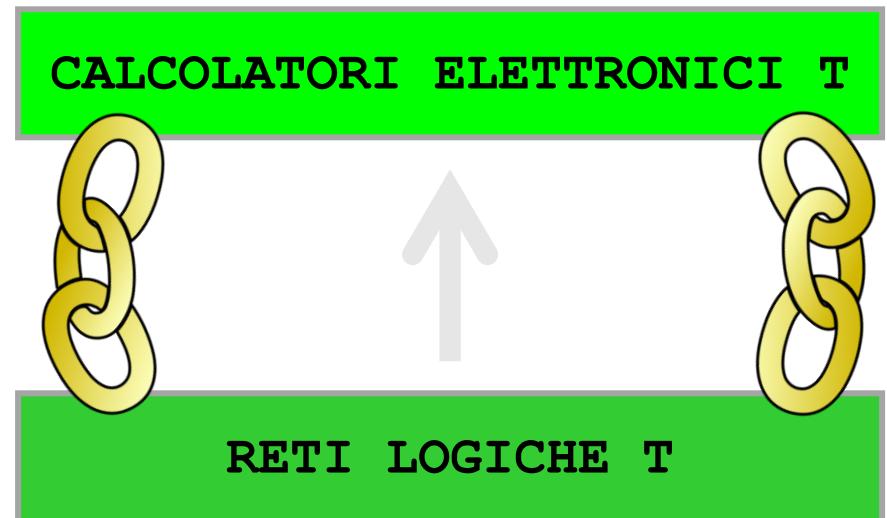
# Relazioni con altri corsi



# Reti Logiche e Calcolatori



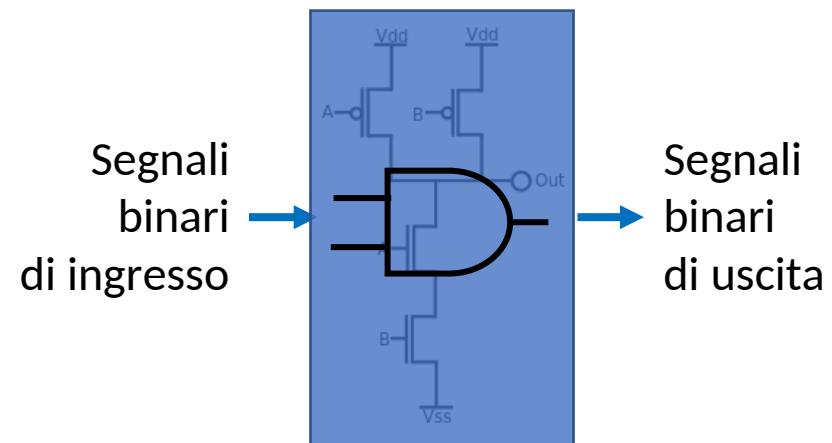
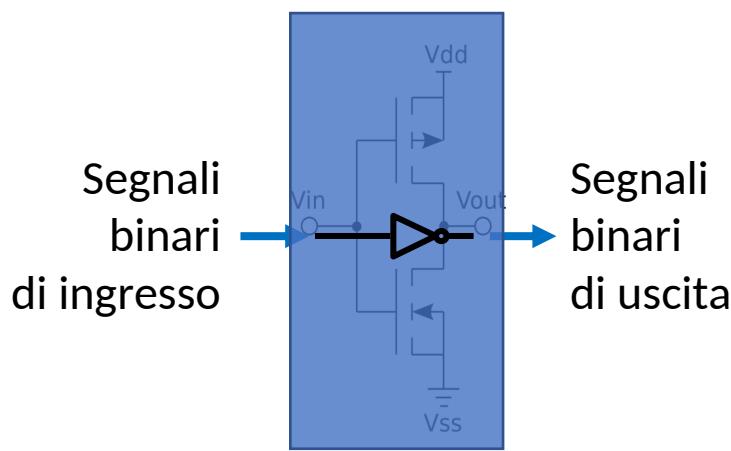
Progettazione di sistemi digitali complessi composti da numerose reti logiche (CPU, I/O, etc...)



Progettazione di semplici sistemi digitali, componenti primitivi per il livello architetturale

# Reti logiche

- Una rete logica è quindi **un'astrazione** per una **combinazione di interruttori** che elaborano **segnali binari**
- Per **astrarre dalla complessità** della tecnologia sottostante si definiscono dei **componenti elementari** (gate, realizzati solitamente da interruttori elettronici) di cui ci interessa solo **la relazione tra i segnali binari di ingresso e di uscita (comportamento)**, non come sono realizzati (struttura).
- Quanti e quali componenti elementari esistono?



# Gate con 1 ingresso

- In generale **il numero di funzioni diverse di  $n$  ingressi binari** con un'uscita binaria è

$$2^{2^n}$$

- I componenti elementari (=funzioni) possibili limitandosi a quelli con un unico segnale binario di ingresso  $x$  sono quindi 4 ( $n = 1$ )

$x$	$f_0$	$f_1$	$f_2$	$f_3$
0	0	0	1	1
1	0	1	0	1

- Tolte l'identità ( $f_1$ ) e le costanti ( $f_0$  e  $f_3$ ), rimane un'unica funzione interessante, l'operatore **NOT** o **invertitore** ( $f_2$ ).

# Il gate NOT

Ogni gate elementare è descritto da

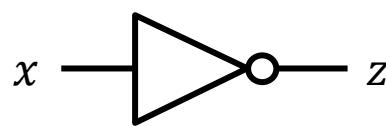
- una **tabella della verità**, ovvero una tabella in cui ogni riga riporta una possibile configurazione degli ingressi e il valore dell'uscita ad essi corrispondente
- un **simbolo circuitale**, che astrae gli interruttori necessari ad implementare quel comportamento
- un'**espressione**, ovvero un modo di rappresentare la relazione tra ingressi ed uscite in forma compatta (ci torneremo)

Nel caso del gate NOT abbiamo

Tabella della Verità

$x$	$z$
0	1
1	0

Simbolo



Espressione

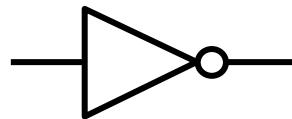
$$z = \bar{x}$$

oppure

$$z = x'$$

# II gate NOT

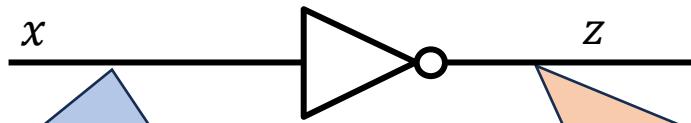
$x$



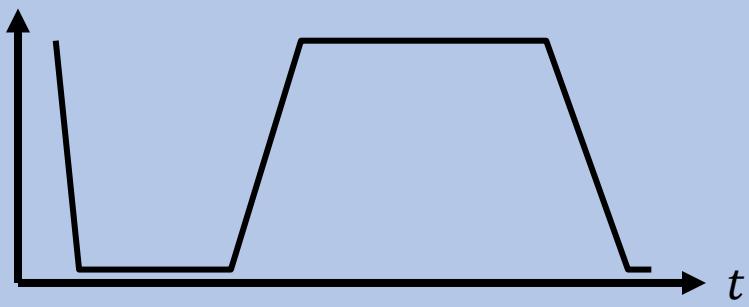
$z$



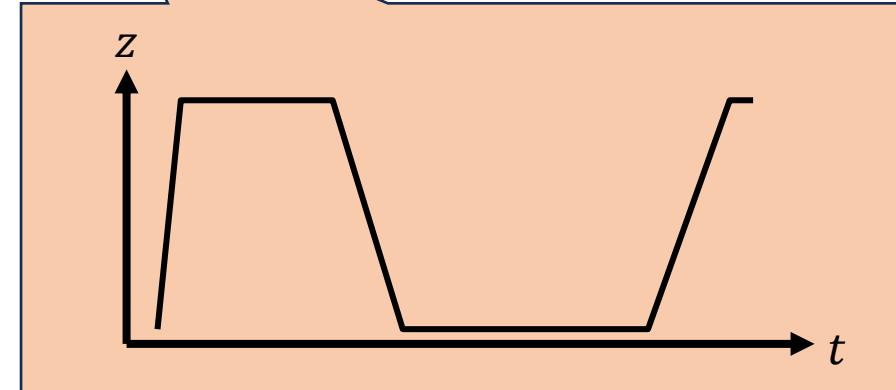
$x$



$x$



$z$



# Gate con 2 ingressi

- Usando la formula  $2^{2^n}$  con  $n = 2$  otteniamo che le possibili funzioni elementari di 2 ingressi sono 16

$x$	$y$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$	$f_{10}$	$f_{11}$	$f_{12}$	$f_{13}$	$f_{14}$	$f_{15}$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- Tolte le costanti e le funzioni identità e NOT di ognuno dei due ingressi, rimangono 10 funzioni realmente di 2 ingressi, evidenziate in blu
- Solo 6 sono disponibili come componenti elementari di reti logiche, quelle cerchiate in arancio, che sono descritte nelle slide seguenti.
- Le altre non sono disponibili come componenti elementari: possono comunque essere realizzate a partire da quelle disponibili, come vedremo.

# Il gate AND

- Il componente AND è l'astrazione di due interruttori **in serie**
- L'uscita assume valore logico «1» (la lampadina è accesa) se entrambi gli ingressi hanno valore «1», ovvero entrambi gli interruttori sono chiusi

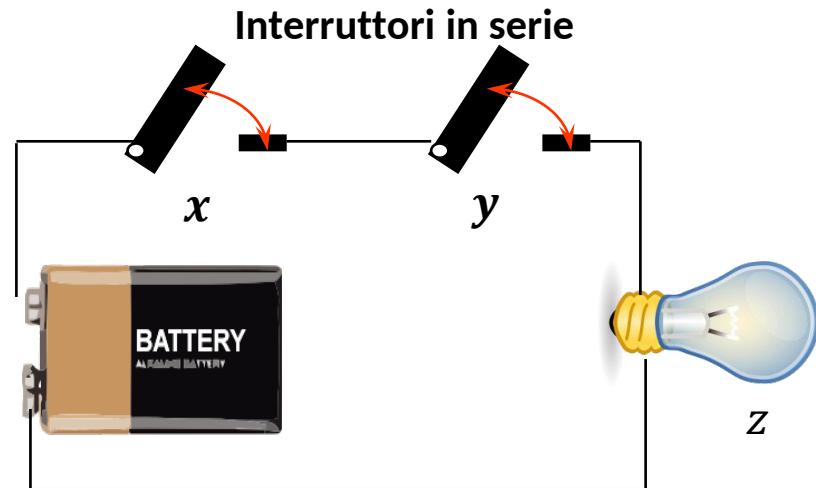
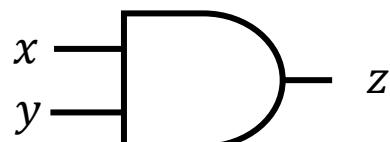


Tabella della Verità

$x$	$y$	$z$
0	0	0
0	1	0
1	0	0
1	1	1

Simbolo



Espressione

$$z = x \cdot y$$

oppure

$$z = xy$$

# Il gate OR

- Il componente OR è l'astrazione di due interruttori **in parallelo**
- L'uscita assume valore logico «1» (la lampadina è accesa) se almeno un ingresso ha valore «1», ovvero almeno uno degli interruttori è chiuso

Interruttori in parallelo

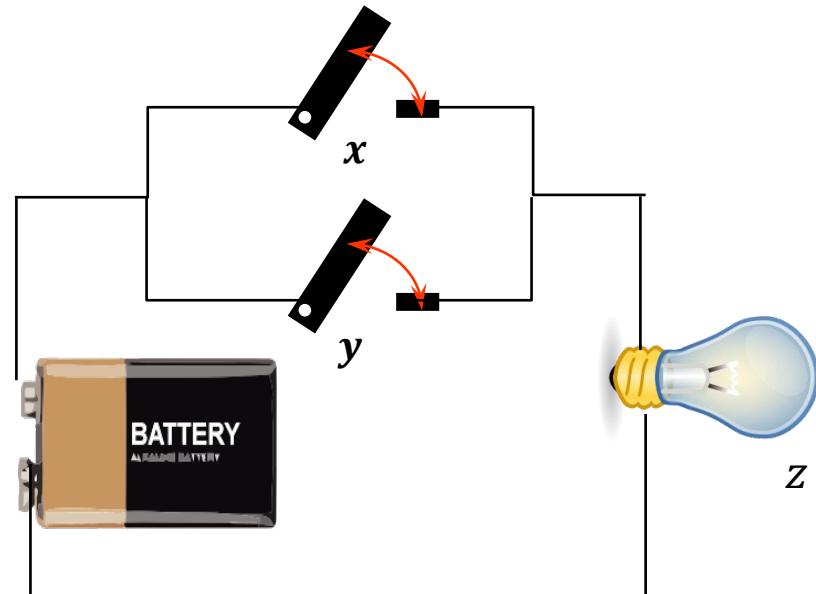
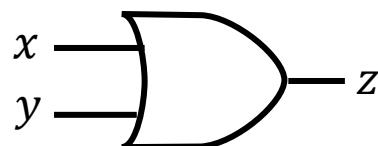


Tabella della Verità

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Simbolo

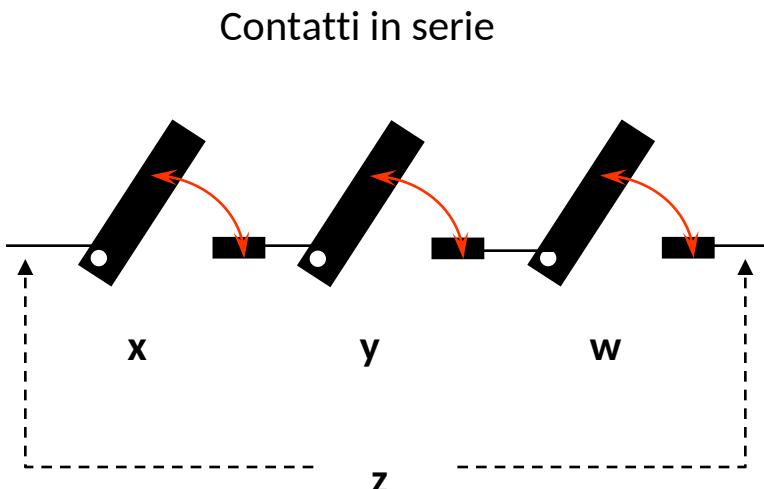


Espressione

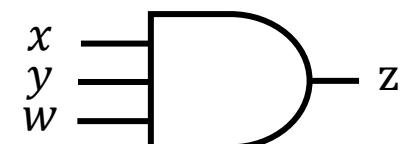
$$z = x + y$$

# AND e OR con più ingressi

- Dato il tipo di circuito di cui sono astrazione, risulta chiaro il comportamento di gate AND e OR a più di due ingressi
- Per esempio, un AND a 3 ingressi corrisponde a 3 interruttori in serie: l'uscita vale «1» se e solo se tutti e 3 gli ingressi hanno valore «1»
- Il numero di ingressi di un gate viene detto il suo ***fan-in***



$x$	$y$	$w$	$z$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



$$z = xyw$$

# Il gate EXOR/XOR

- Il componente EXOR o XOR è l'astrazione di due **deviatori**, ovvero interruttori che chiudono il circuito verso uno di due possibili percorsi
- L'uscita assume valore logico «1» (la lampadina è accesa) se un ingresso ha valore «1» ma non entrambi, ovvero i due deviatori sono su **posizioni diverse**

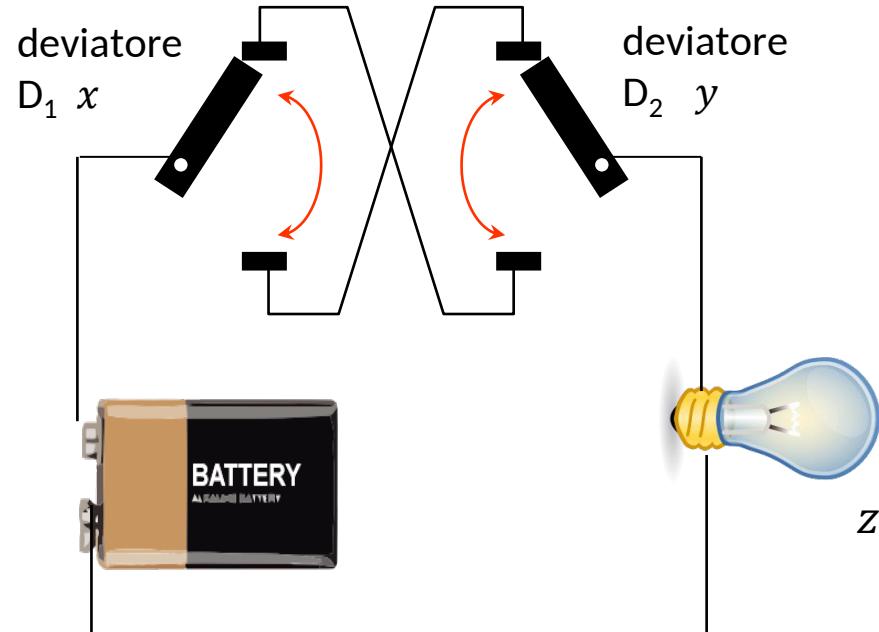
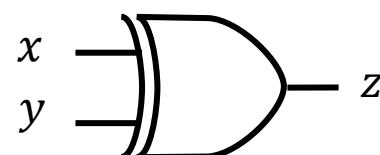


Tabella della Verità

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Simbolo



Espressione

$$z = x \oplus y$$

# Comportamento del gate EXOR/XOR

- Il nome EXOR o XOR è la contrazione di «exclusive OR», ovvero un OR in cui solo uno dei due ingressi può essere 1 affinché l'uscita sia a sua volta 1
- L'operatore EXOR viene anche detto **somma modulo 2**, in quanto il suo output può essere interpretato come il risultato della somma (escluso il riporto) di due bit (vedi lezioni sull'aritmetica binaria in seguito)
- Infine si può interpretare il comportamento del gate EXOR come controllo sul numero di «1» in input: **l'uscita vale «1» se e solo se il numero di «1» in ingresso è dispari**
- Con quest'ultima interpretazione si può facilmente estendere il gate EXOR a più ingressi, anche se è meno comune farne uso rispetto a AND e OR.

# I gate NAND, NOR, EXNOR

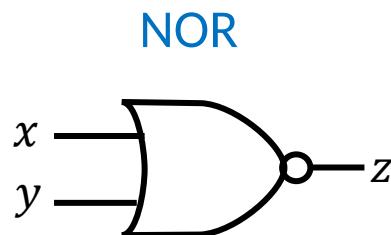
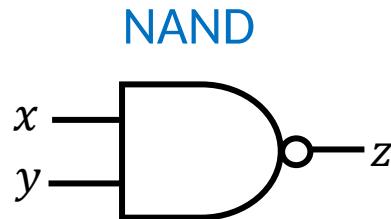
Tabella della Verità

x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

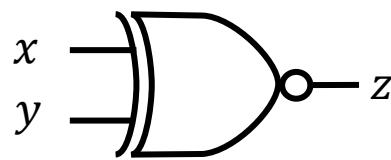
x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

x	y	z
0	0	1
0	1	0
1	0	0
1	1	1

Simbolo



EXNOR (o EQUIVALENCE)



Espressione

$$z = x \uparrow y$$

oppure

$$z = \overline{xy}$$

$$z = x \downarrow y$$

oppure

$$z = \overline{x + y}$$

$$z = x \equiv y$$

oppure

$$z = \overline{x \oplus y}$$

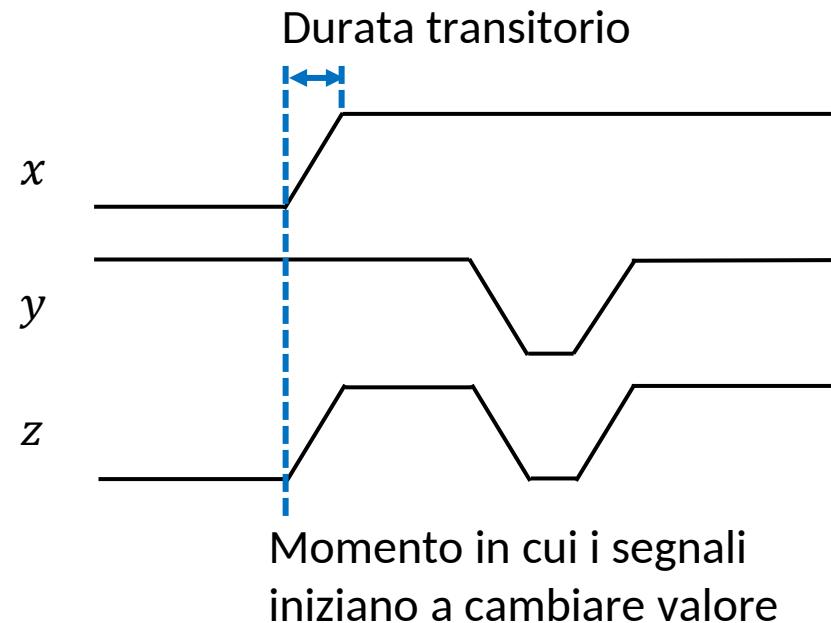
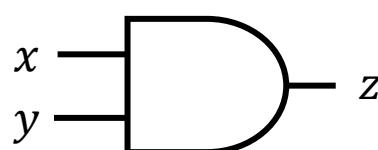
# I gate NAND, NOR, EXNOR

- Il gate EXNOR viene detto anche EQUIVALENCE, in quanto (per due ingressi) ha uscita «1» solo se i due ingressi sono uguali
- I tre gate sono le versioni negate dei gate definiti in precedenza, quindi l'estensione a più ingressi si può ricavare negando quella di AND, OR e EXOR: per esempio un NAND a 4 ingressi ha uscita «0» solo quando tutti e 4 gli ingressi valgono «1», uscita «1» altrimenti
- Essendo ottenibili dai gate precedenti con un NOT in cascata non sarebbero strettamente necessari, ma hanno proprietà che li rendono utili e che vedremo in seguito



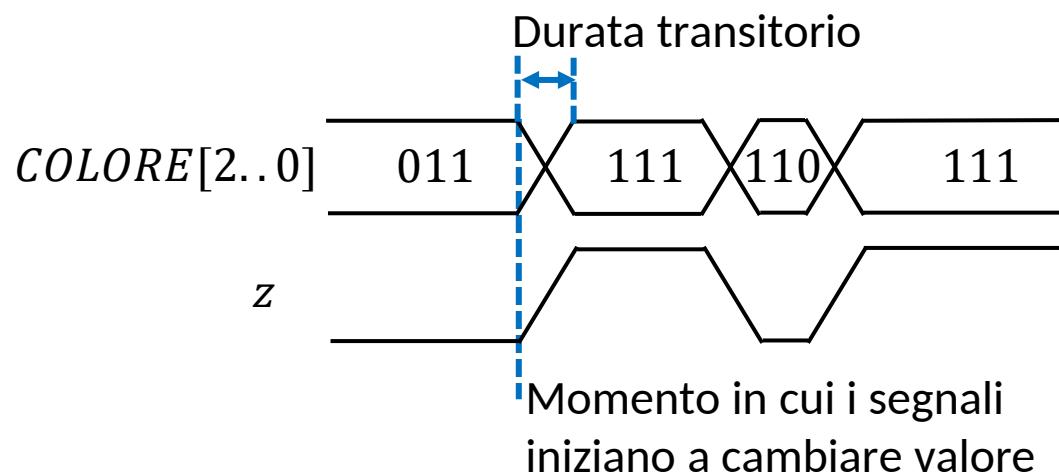
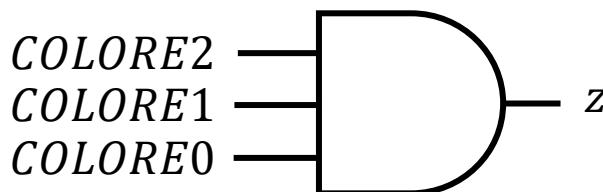
# Diagrammi ad occhio

- Per rappresentare l'evoluzione di gruppi di segnali binari in forma compatta, per esempio gli ingressi di un gate (o di una rete più complessa), si usano i cosiddetti **diagrammi ad occhio**
- Poiché i segnali analogici sottostanti **non cambiano valore istantaneamente**, il diagramma riporta anche i transitori tra una configurazione stabile e la successiva



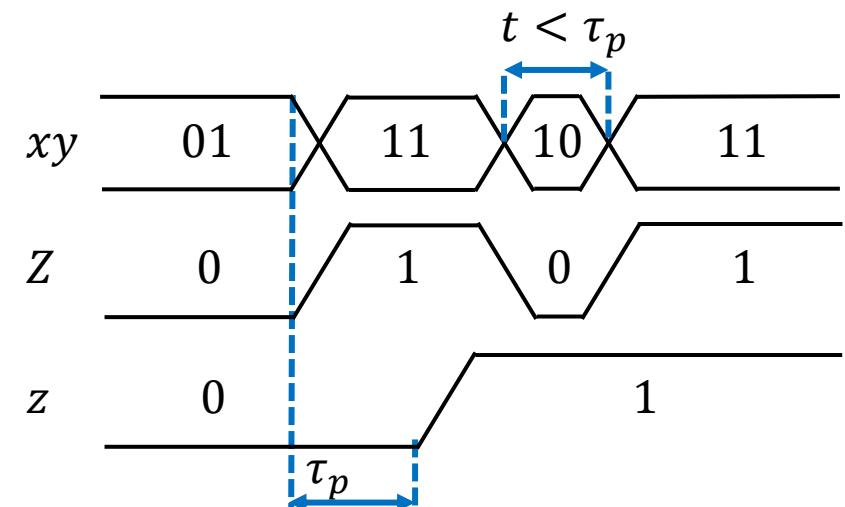
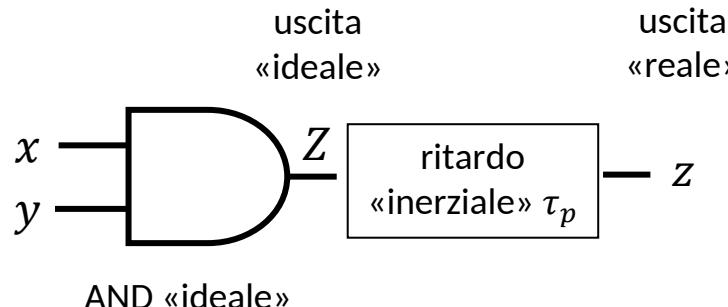
# Bus di segnali

- L'evoluzione di **un gruppo di bit** può anche essere descritta scrivendone la configurazione binaria nel diagramma.
- Un gruppo di segnali viene anche detto un **bus**.
- Per indicare un bus di  $n$  segnali che codificano un'informazione, useremo anche la notazione con parentesi quadre  $[n - 1..0]$ . Per esempio, COLORE[2..0] potrebbe essere un modo di indicare il bus a 3 bit usato nell'esempio dei segnali di fumo.
- Per riferirci ad uno dei segnali del bus useremo la notazione COLORE0, COLORE1, COLORE2.



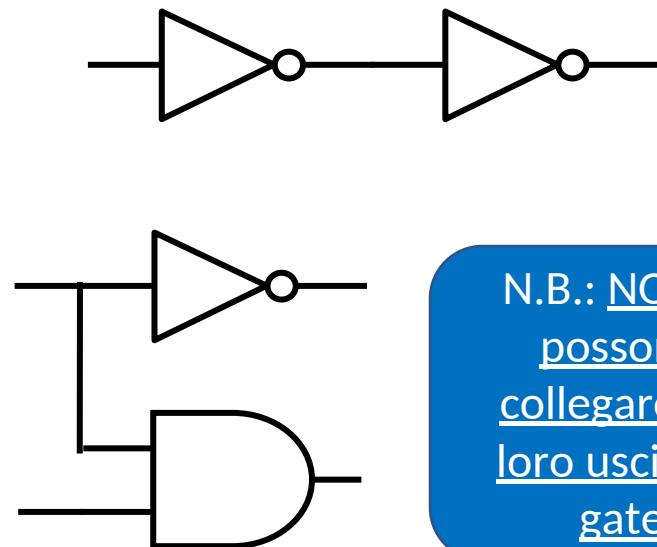
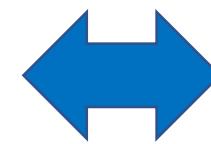
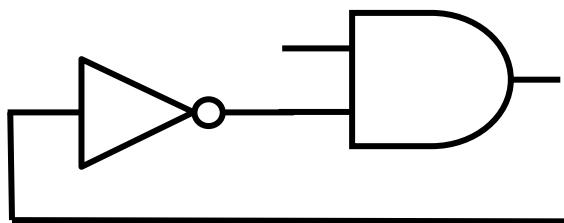
# Ritardi di propagazione

- Anche se stiamo astraendo dagli interruttori sottostanti per rendere gestibile il progetto e l'analisi, dobbiamo sempre ricordare che i gate sono anche **componenti reali**
- La differenza principale tra l'astrazione fornita dalla tabella della verità e un gate reale è il **ritardo di propagazione**
- Quando cambia un ingresso di un gate, **l'uscita non cambia istantaneamente**, ma dopo un tempo  $\tau_p$  che dipende dalla tecnologia utilizzata
- Il ritardo di propagazione non è solo un semplice ritardo puro, ma un **ritardo «inerziale»**: un impulso di durata inferiore a  $\tau_p$  su uno degli ingressi non appare in uscita.
- Esiste quindi un **limite superiore per la velocità di funzionamento** di ogni gate.

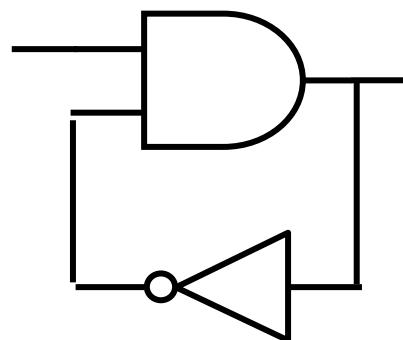


# Possibili montaggi

- Dati due gate (o gruppi di gate), è possibile montarli in tre modi:
  - In **serie**, ovvero l'uscita del gate a monte è uno degli ingressi del gate a valle
  - In **parallelo**, ovvero alcuni degli ingressi sono in comune (**e non le uscite**)
  - In **retroazione**, ovvero l'uscita di un gate è uno degli ingressi dell'altro e **viceversa**

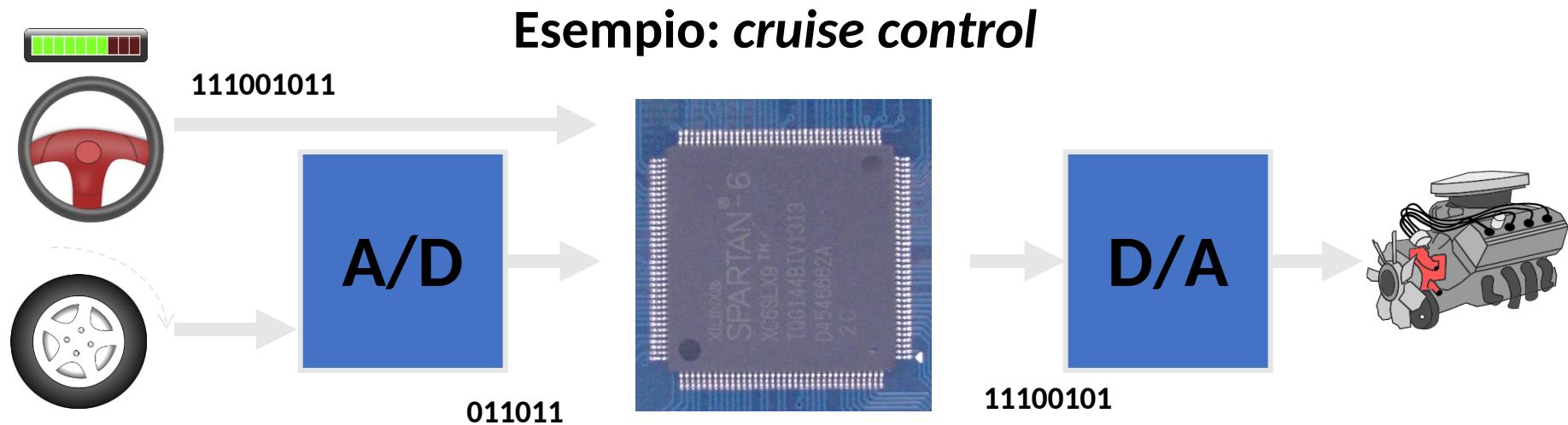


N.B.: NON si  
possono  
collegare tra  
loro uscite di  
gate



# Conversione A/D e D/A

- Abbiamo visto che è conveniente rappresentare l'informazione sotto forma di (stringhe di) segnali binari
- Input e output possono però essere anche segnali analogici: si rende quindi necessario per alcuni segnali operare conversioni da analogico a digitale (A/D) e viceversa (D/A)



# Esempio: acquisizione immagini



Scena reale

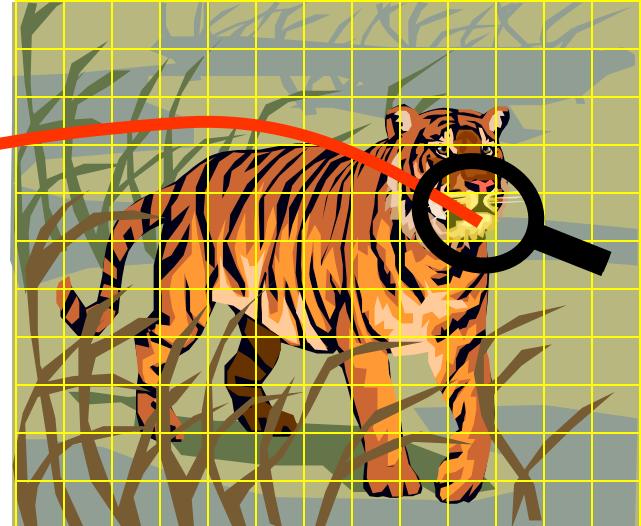
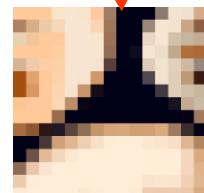


Immagine digitale  
acquisita dal sensore

$$R \in \{0, 1, 2, \dots, 254, 255\}$$

$$G \in \{0, 1, 2, \dots, 254, 255\}$$

$$B \in \{0, 1, 2, \dots, 254, 255\}$$

# Informazione

- Una macchina digitale elabora e comunica **informazione**
- Ma cos'è l'informazione? In questo corso usiamo la definizione che deriva dalla **teoria dell'informazione**
- Informazione: una **stringa** di lunghezza finita di **simboli** appartenenti ad un **alfabeto**
- L'alfabeto definisce le **informazioni «elementari»**

“testo” e caratteri

“numero” e cifre

“immagine” e valore di un pixel

“parlato” e fonemi

“musica” e note

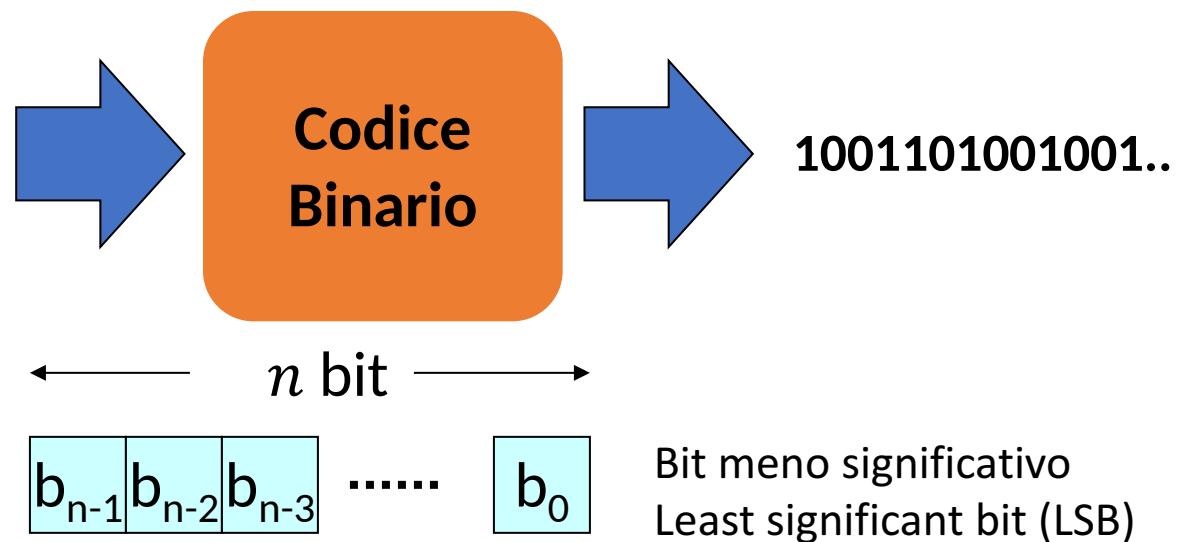
“disegno” e pendenza/lunghezza di tratti



# La codifica binaria dell'informazione

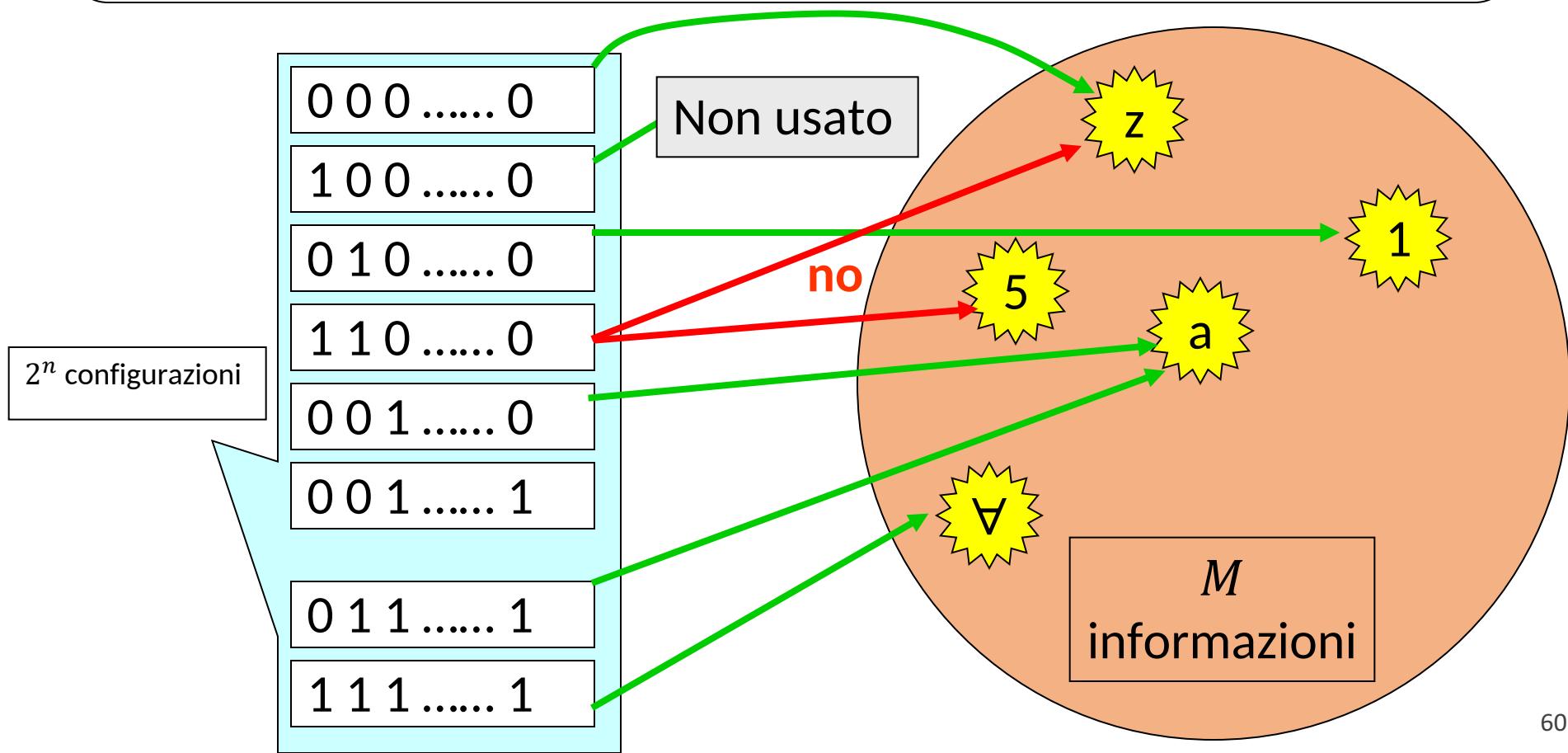
- L'utilizzo di **segnali binari** per l'elaborazione e la comunicazione di una macchina digitale comporta che
  - la macchina impieghi un **alfabeto binario** (ovvero che utilizzi i due soli simboli «0» e «1»)
  - qualsiasi informazione elaborata o comunicata sia rappresentata da una **stringa di bit**
- Attraverso opportuni **codici binari** si trasformano dunque informazioni di varia natura in informazione sotto forma di stringhe di bit (**rappresentazione binaria dell'informazione**)

- Testi, immagini, suoni
- Istruzioni, dati, risultati
- Ingresso, uscita, stato
- ...



# Codice binario

Codice binario - **Funzione** dall'insieme delle  $2^n$  configurazioni di  $n$  bit ad un insieme di  $M$  informazioni (*simboli alfanumerici, colori, numeri, ecc.*). Condizione necessaria per la codifica:  $2^n \geq M$  (se vi sono  $M$  simboli da codificare, occorrono almeno  $2^n \geq M$  differenti configurazioni binarie)



# Proprietà di un codice

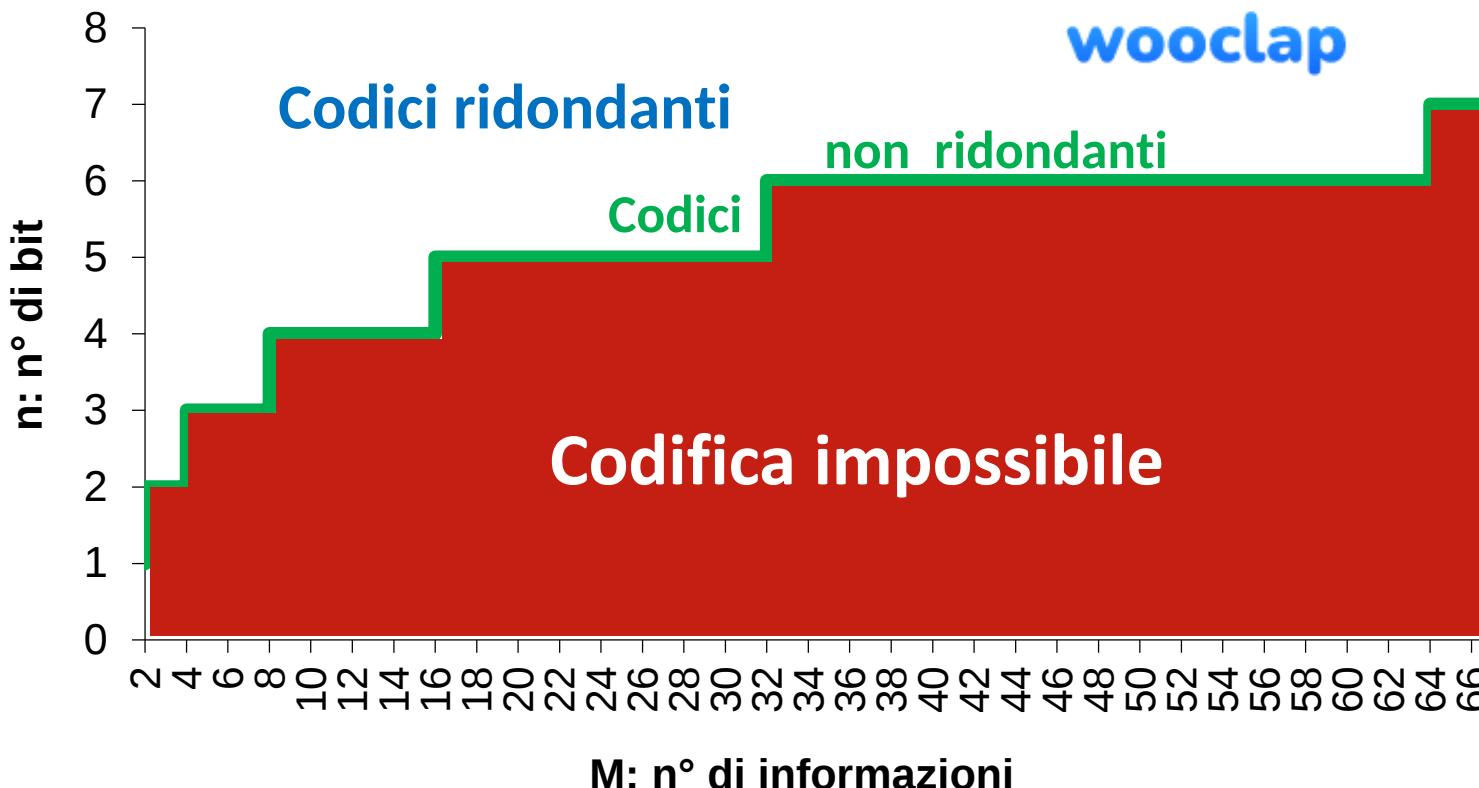
- Un codice è una **rappresentazione convenzionale** dell'informazione
- La **scelta** di un codice deve essere **condivisa** da **sorgente** e **destinazione**, ed ha due gradi di libertà:
  - il **numero  $n$  di bit** (qualsiasi, purché  $2^n \geq M$ )
  - l'**associazione** tra configurazioni e informazioni
- A parità di  $n$  e di  $M$ , le associazioni possibili (disposizioni senza ripetizione) sono:

$$C = 2^n! / (2^n - M)!$$

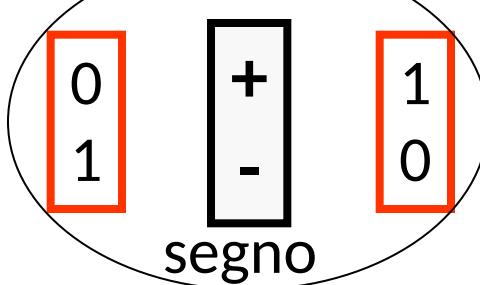
n.bit, n.informazioni	associazioni possibili
$n = 1, M = 2$	$C = 2$
$n = 2, M = 4$	$C = 24$
$n = 3, M = 8$	$C = 40.320$
$n = 4, M = 10$	$C = 29.059.430.400$

# Codici ridondanti e codici non ridondanti

- Poiché la condizione per rendere possibile la codifica è  $2^n \geq M$ , il **numero minimo** di bit necessari per creare un codice che rappresenti  $M$  informazioni è  $n_{min} = \lceil \log_2 M \rceil$  ( $\lceil \cdot \rceil$  indica l'intero superiore al numero)
- Un codice che utilizza esattamente  $n_{min}$  bit è detto **codice non ridondante**
- Un codice che utilizza un numero  $n > n_{min}$  di bit è detto **codice ridondante**



M=2  
n=1  
C=2



*Non ridondanti*

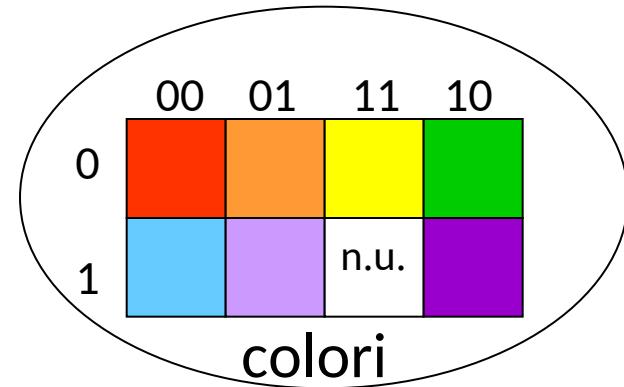
altri  
29.059.430.399  
codici  
a 4 bit

0000  
0001  
0010  
0011  
0100  
0101  
0110  
0111  
1000  
1001

BCD

Binary-Coded Decimal

M=7  
n=3  
C=40.320



colori

*Ridondanti*

*zero*  
*uno*  
*due*  
*tre*  
*quattro*  
*cinque*  
*sei*  
*sette*  
*otto*  
*nove*

1111110  
0110000  
1101101  
1111001  
0110011  
1011011  
0011111  
1110000  
1111111  
1110011

7 segmenti

1000000000  
0100000000  
0010000000  
0001000000  
0000100000  
0000010000  
0000001000  
0000000100  
0000000010  
0000000001

1 su 10

Per codificare un alfabeto di simboli formato dalle 10 cifre decimali (M=10) occorrono almeno n=4 bit. Ma è possibile utilizzarne anche 5, 6, ..

# Codice BCD

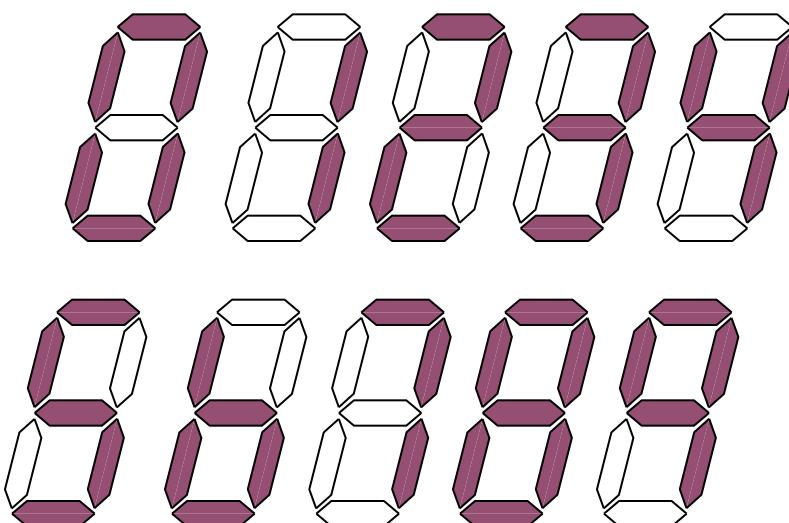
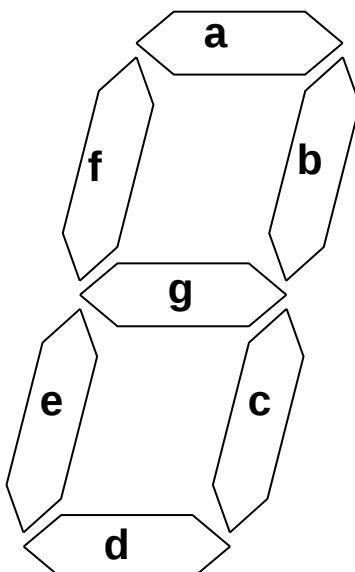
- Codice che rappresenta ciascuna cifra decimale con 4 bit
- k cifre decimali -> 4k bit

<i>zero</i>	0000
<i>uno</i>	0001
<i>due</i>	0010
<i>tre</i>	0011
<i>quattro</i>	0100
<i>cinque</i>	0101
<i>sei</i>	0110
<i>sette</i>	0111
<i>otto</i>	1000
<i>nove</i>	1001

- Es.: nel caso di due cifre decimali: ventuno -> 0010 0001 (inefficiente)

# Codice a 7 segmenti

- Codice ridondante utilizzato per visualizzare a display numeri decimali
- $M=10$ ,  $n=7>4$

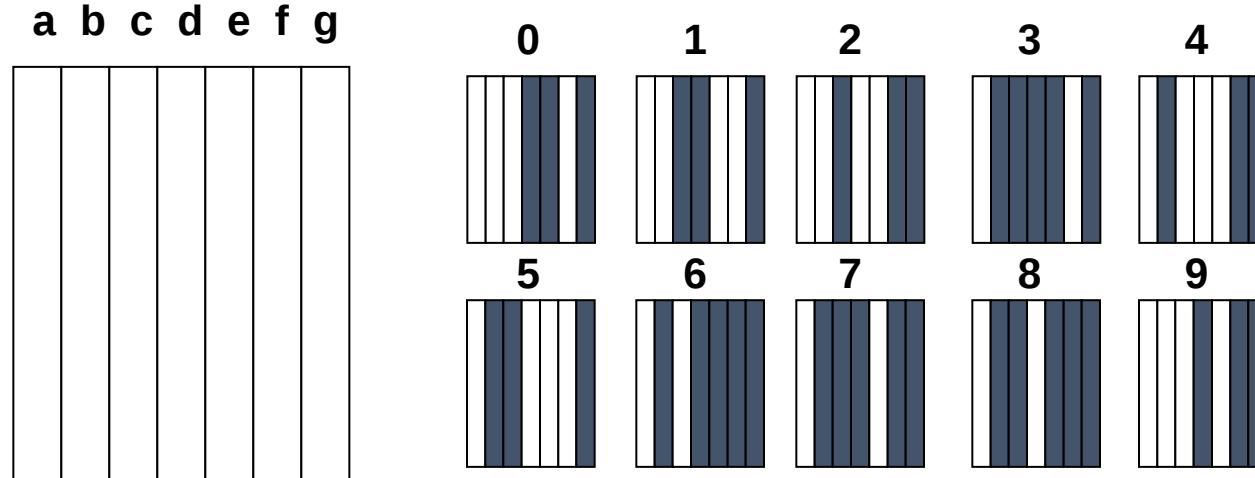


Assumeremo segmento attivo se ingresso «0»  
(ingressi attivi bassi)

	a	b	c	d	e	f	g
zero	0	0	0	0	0	0	1
uno	1	0	0	1	1	1	1
ecc.							

# Universal Product Code

- Codice ridondante utilizzato per associare un valore numerico (facilmente leggibile da un apposito sensore senza contatto) a un prodotto
- Per ogni cifra  $M=10$ ,  $n=7>4$



# Rappresentazione dei numeri

Che numero rappresenta la stringa qui sotto?

10

Qual è il risultato di questa somma?

1+1

Risposta corretta in questo corso: **dipende dalla base del sistema di numerazione**



# Il sistema posizionale decimale

- Noi siamo abituati ad una rappresentazione
  - **in base «dieci»**
  - **posizionale**
- Es. per rappresentare il numero «duemilaquarantotto» scriviamo

$$2048 = (2048)_{10} = 2 \times 10^3 + 0 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$$

- N.B. : esistono anche sistemi di numerazione non posizionali (numeri romani, «duemilaquarantotto» = MMXLVIII)
- Poiché le macchine digitali operano su segnali binari che possono assumere solo due valori, il sistema usato è sempre **posizionale**, ma in **base 2**

# Rappresentazione posizionale

- In generale, un numero  $N$  in **base  $\beta \geq 2$**  è rappresentato da  **$n+m$  cifre**  $c_i$  (parte intera e parte frazionaria)

$$(N)_\beta = (c_{n-1} \dots c_0, c_{-1} \dots c_{-m})_\beta$$

- In base  $\beta$  avrò  **$\beta$  cifre possibili**: per basi  $\leq 10$  si usano le cifre tradizionali, **per basi  $> 10$  si usano i caratteri (A,B,C,...) come cifre aggiuntive**
- Ognuna delle  $\beta$  cifre **corrisponde ad un diverso numero naturale** tra 0 e  $\beta - 1$

$$c_k \rightarrow n_k \in \{0, 1, \dots, \beta - 1\} \quad \forall k$$

- Per **calcolare il valore** del numero rappresentato si sommano i numeri naturali corrispondenti alle cifre, moltiplicati per **una potenza di  $\beta$  che dipende dalla posizione (espansione polinomiale)**

$$N = (n_{n-1}\beta^{n-1} + \dots + n_0\beta^0 + n_{-1}\beta^{-1} + \dots + n_{-m}\beta^{-m})$$

# Esempi

	Sistema Decimale	Sistema Binario	Sistema Esadecimale
Base	«dieci»	«due»	«sedici»
Cifre	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	0, 1	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Numeri naturali corrispondenti	«zero» «uno» ... «nove»	«zero» «uno»	«zero» «uno» ... «quindici»
Rappresentazioni del numero «ventotto»	$(28)_{10}$	$(11100)_2$	$(1C)_{16}$
Verifica del valore	$2*10^1+8*10^0=$ «ventotto»	$1*2^4 + 1*2^3 + 1*2^2 +$ $0*2^1 + 0*2^0=$ «ventotto»	$1*16^1 + 12*16^0=$ «ventotto»

# Cambio di base da 2 a 16

- La base 16 è spesso usata per rappresentare lunghe stringhe di numeri binari in forma compatta (per esempio **indirizzi di memoria** in un calcolatore)
- Il cambio di base tra questi due sistemi è semplice: ad ogni gruppo di 4 cifre binarie corrisponde un simbolo esadecimale e viceversa, per esempio

$$(1010\textcolor{brown}{1111}0001)_2 = (\textcolor{brown}{A}\textcolor{blue}{F}1)_{16}$$
$$(\textcolor{blue}{4C})_{16} = (\textcolor{blue}{0100}\textcolor{purple}{1100})_2$$

N <sub>10</sub>	N <sub>2</sub>	N <sub>16</sub>
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

# Cambio di base

- In generale, cambiare base da una generica  $\beta$  ad un'altra base  $\beta^*$  non è così diretto come tra 2 e 16

$$(N)_\beta = (I, F)_\beta = (? , ? )_{\beta^*} = (c_{n-1} \dots c_0, c_{-1} \dots c_m)_{\beta^*}$$

- Siamo interessati a due casi notevoli:
  - conversione da binario/esadecimale a decimale ( $\beta=2/16$  e  $\beta^*=10$ ): è sufficiente calcolare il valore di un numero tramite **l'espansione polinomiale** che abbiamo visto per averne anche la rappresentazione in base 10
  - conversione da decimale a binario/esadecimale ( $\beta=10$  e  $\beta^*=2/16$ ): **metodo di conversione iterativa** descritto tramite esempi nelle prossime slide
  - E se né la base di partenza né la base di arrivo sono 10?  
 $\beta \rightarrow 10 \rightarrow \beta^*$

# Conversione iterativa: esempio

Supponiamo di voler convertire  $(41,6875)_{10}$  in binario, ovvero vogliamo conoscere le cifre  $(b_{n-1} \dots b_0, b_{-1} \dots b_{-m})_2$  che rappresentano lo stesso numero

Cifre parte intera				Cifre parte frazionaria			
	Quoziente intero	Resto	Cifra		Risultato intero	Risultato frazionario	Cifra
41/2	20	1	$b_0 = 1$	0,6875*2	1	0,375	$b_{-1} = 1$
20/2	10	0	$b_1 = 0$	0,375*2	0	0,75	$b_{-2} = 0$
10/2	5	0	$b_2 = 0$	0,75*2	1	0,5	$b_{-3} = 1$
5/2	2	1	$b_3 = 1$	0,5*2	1	0	$b_{-4} = 1$
2/2	1	0	$b_4 = 0$				
1/2	0	1	$b_5 = 1$				

N.B.: la conversione della parte frazionaria può richiedere un numero infinito di cifre. Devo fissare un numero di cifre significative dopo il quale la interrompo, se non termina prima.

La rappresentazione cercata è quindi  $(101001,1011)_2$

# Conversione iterativa: esempio

Supponiamo di voler convertire  $(41,6875)_{10}$  in esadecimale, ovvero vogliamo conoscere le cifre  $(h_{n-1} \dots h_0, h_{-1} \dots h_{-m})_{16}$  che rappresentano lo stesso numero

Cifre parte intera				Cifre parte frazionaria		
	Quoziente intero	Resto	Cifra	Risultato intero	Risultato frazionario	Cifra
41/16	2	9	$h_0 = 9$	0,6875*16	11	0
2/16	0	2	$h_1 = 2$			$h_{-1} = B$

La rappresentazione cercata è quindi  $(29, B)_{16}$

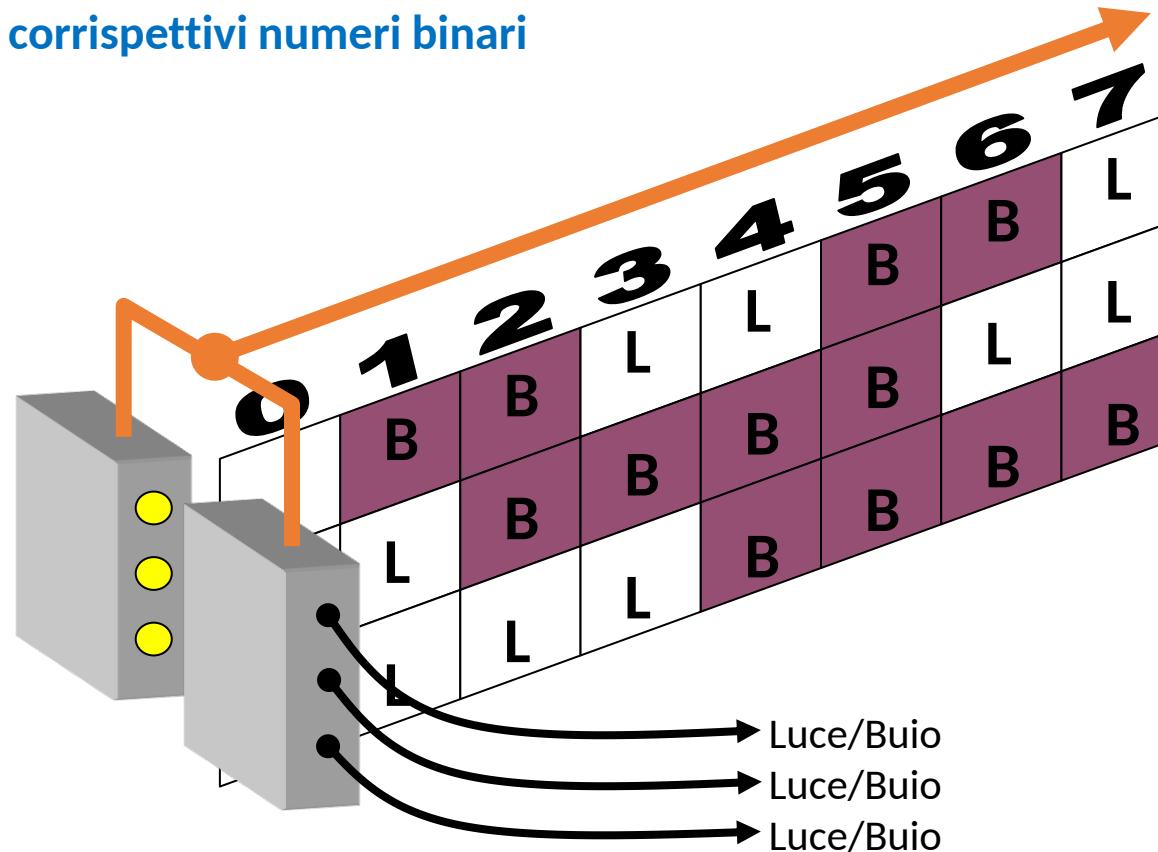
Lo si poteva dedurre anche dalla rappresentazione binaria della slide precedente convertendola in esadecimale

# Numeri binari senza segno

- Per rappresentare un **numero senza segno**  $N$  in una macchina digitale si usa quindi la rappresentazione in base 2
- Altra **differenza fondamentale** con la rappresentazione a cui siamo abituati «su carta» è che il numero di cifre disponibili è **fisso e finito**, in quanto l'hardware che uso stabilisce quanti segnali e quindi cifre binarie (**bit**, *binary digit*) ho a disposizione per rappresentarlo.
- **Dati  $n$  bit, i numeri senza segno rappresentabili sono compresi tra 0 e  $2^n - 1$**
- Esempio: in C, il tipo `unsigned short` ha (tipicamente) dimensione 16 bit, i numeri rappresentabili appartengono quindi all'intervallo  $[0, 65.535]$
- [https://www.onlinegdb.com/online\\_c\\_compiler](https://www.onlinegdb.com/online_c_compiler)

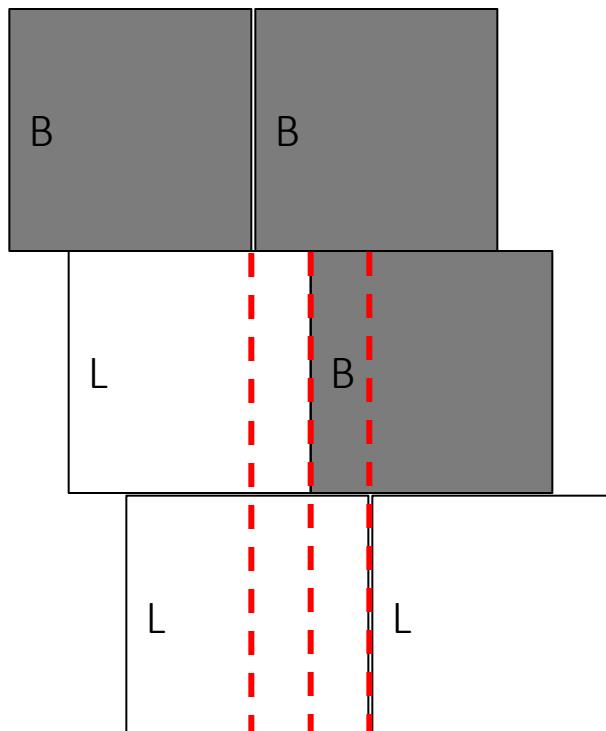
# Codifica della posizione

- Vogliamo conoscere in ogni istante la posizione di un carrello lungo una rotaia.
- Il carrello trasporta 3 fotodiodi e 3 fotocellule
- La fotocellula vede o no la luce emessa dal fotodiodo per via della diversa trasparenza del materiale interposto
- La posizione viene così discretizzata in 8 intervalli, descritti tramite 3 segnali binari
- **Posizioni consecutive differiscono di un solo bit e le posizioni NON sono codificate con i corrispettivi numeri binari**



# Conversione A/D

- L'allineamento meccanico perfetto è in pratica impossibile, ovvero due segnali non possono variare contemporaneamente
- Per evitare letture scorrette nei punti di disallineamento **è opportuno utilizzare configurazioni relative a posizioni consecutive che differiscono di un solo valore**



B B B B B B  
L L L B B B  
L L L L L L



B B B B B B  
L L L B B B  
L L L L B B



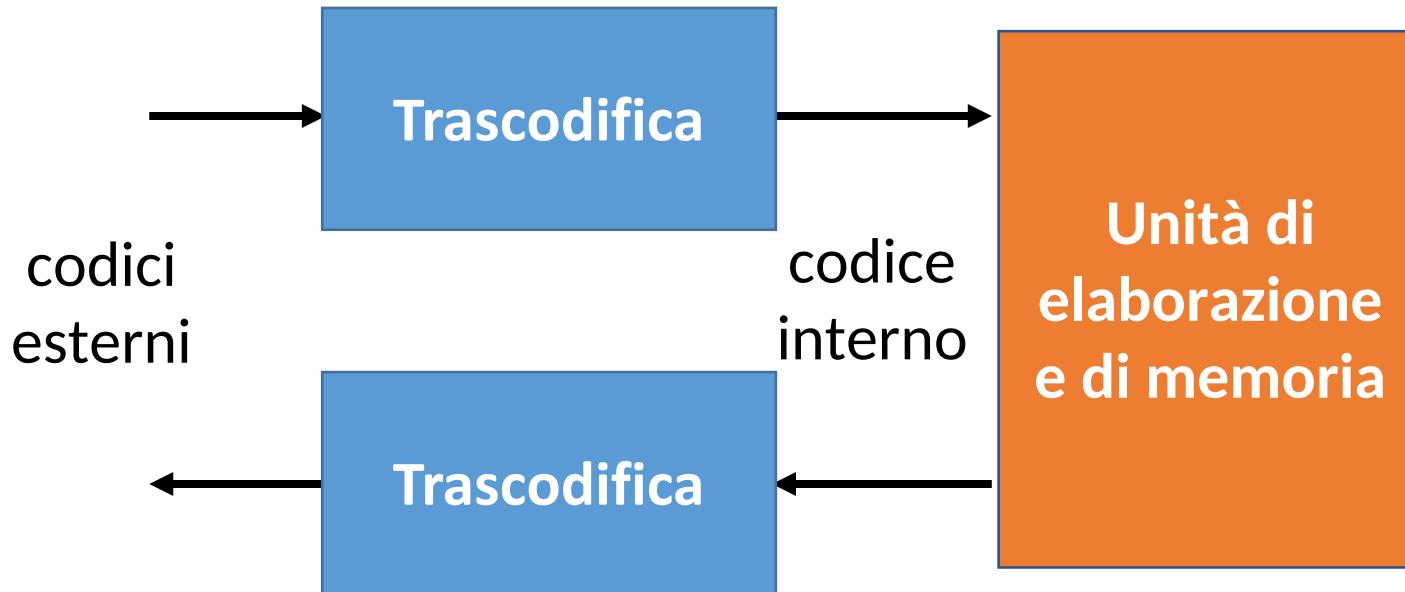
Lettura:

# Codice di Gray

- La codifica della posizione usata per il carrello è un **codice di Gray** (dal nome del suo inventore)
- In un codice di Gray, **due configurazioni che codificano informazioni adiacenti differiscono per il valore di un solo bit**. Questo non è vero col codice binario «standard» che abbiamo usato per elencare le  $2^n$  configurazioni di  $n$  bit, in cui ogni posizione è rappresentata dal numero binario corrispondente.
- **Usato per codificare la posizione** (lineare o circolare) e, più in generale, per
  - ridurre le sorgenti di errore nel convertire un segnale da analogico a digitale
  - per ridurre il numero di cambi necessari per eseguire un ciclo attraverso tutte le possibili configurazioni (es. risparmio energetico in un contatore realizzato con transistor CMOS )

Pos	$b_3$	$b_2$	$b_1$	$b_0$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

# La conversione di codice (trascodifica)



Il **codice interno** è di norma **non ridondante** per minimizzare il numero di bit da elaborare e da memorizzare.

Il **codice esterno** è di norma

- **ridondante**, per semplificare la generazione e l'interpretazione delle informazioni,
- **standard**, per rendere possibile la connessione di macchine (o unità di I/O) realizzate da costruttori diversi.

# Codici proprietari e codici standard

**Codice proprietario** - Codice scelto da un Costruttore per mettere in comunicazione macchine di sua produzione. L'uso di codici proprietari mira ad ottimizzare le prestazioni e a proteggere il mercato di certe macchine.

*Esempi: Linguaggio Assembler, Telecomando TV o condizionatore*

**Codice standard** - Codice scelto da norme internazionali (*de iure*) o dal Costruttore di una macchina ampiamente utilizzata sul mercato (*de facto*).

L'uso di codici standard nelle unità di I/O consente di collegare macchine realizzate da Costruttori diversi.

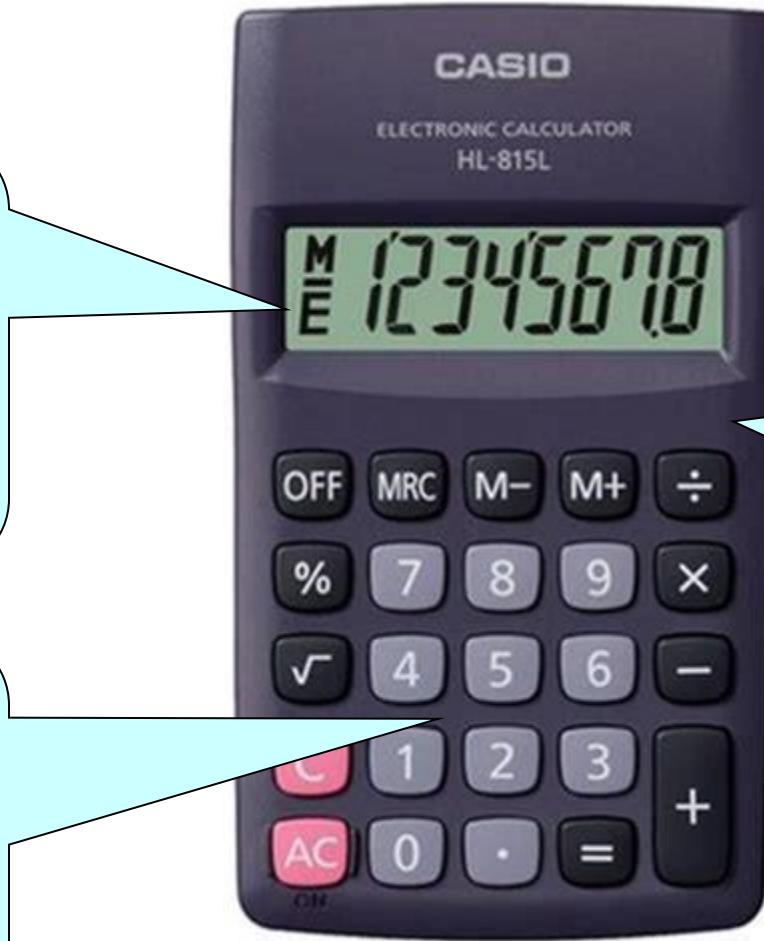
*Esempi: formato PDF, Stampanti e Calcolatori, Calcolatori e Calcolatori*

# La calcolatrice tascabile

Codice  
Ridondante  
7 segmenti  
per la  
visualizzazione  
dei dati

Codice  
Ridondante  
1 su N  
per la  
introduzione  
dei dati e  
dei comandi

Codice  
BCD o binario  
per la  
rappresentazione  
interna  
dei numeri



# Il codice ASCII a 7 bit (1967)

000	001	010	011	100	101	110	111
0000		SP	0	@	P	'	P
0001		!	1	A	Q	a	q
0010		"	2	B	R	b	r
0011		#	3	C	S	c	s
0100		\$	4	D	T	d	t
0101		%	5	E	U	e	u
0110		&	6	F	V	f	v
0111		'	7	G	W	g	w
1000		(	8	H	X	h	x
1001		)	9	I	Y	i	y
1010		*	:	J	Z	j	z
1011		+	;	K	[	k	{
1100		,	<	L	\	l	
1101		-	=	M	]	m	}
1110		.	>	N	^	n	~
1111		/	?	O	_	o	DEL

- 128 caratteri (33 di controllo)
- 0: 011 0000
- A: 100 0001; a: 110 0001
- DEL: 111 1111
- **American Standard Code for Information Interchange**
- Primo standard de iure per la codifica binaria dei caratteri, per le comunicazioni a distanza di telescriventi

# Codice ASCII esteso (8 e 16 bit)

- Estensione del codice ASCII a 8 bit, includendo nell'insieme dei caratteri rappresentati i simboli impiegati dalle lingue originate dal latino (in Europa...)
- **Standard Unicode (fino al 1995)**: ulteriore estensione (**16 bit**) che aveva lo scopo di permettere di codificare in binario con 2 byte i simboli di tutte le lingue conosciute
- Standard Unicode 2.0 e seguenti(dal 1996): lo spazio Unicode è stato ulteriormente esteso e adesso è **un codice a 21 bit**: come viene rappresentato in un calcolatore?



# Unicode: UTF-8, UTF-16, UTF-32

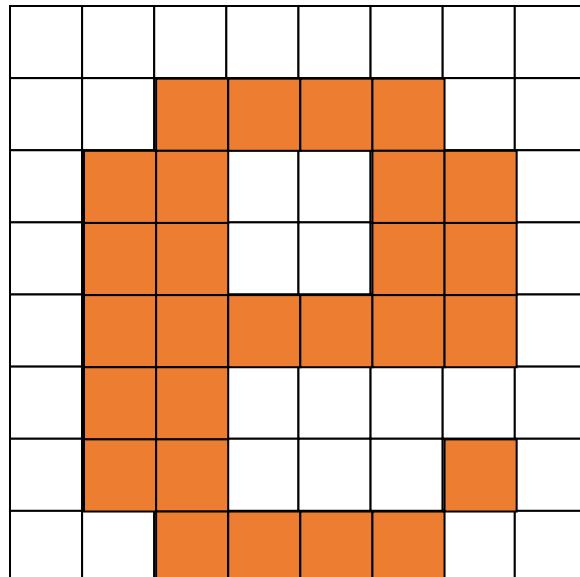
- Problema: le architetture e i linguaggi di programmazione sono progettati e ottimizzati per operare su gruppi di **byte** (= 8 bit) che seguono le potenze di 2, ovvero 1 byte, 2 byte, 4 byte, etc...
- Sono stati quindi definiti 3 standard per mappare un carattere Unicode da 21 bit in una sequenza di byte **non ridondanti** (ovvero farne **l'encoding**), detti *Unicode transformation format* (UTF):
  - **UTF-32**, ovvero usare 32 bit (4 byte) per codificare i 21 bit necessari per ogni carattere, aggiungendo 11 volte «0» a sinistra.
  - **UTF-16**, usare 2 byte (= 16 bit) per i **63,000 caratteri Unicode più comuni**, e 4 byte per i restanti caratteri
  - **UTF-8**, ovvero usare **1 byte per i 128 caratteri ASCII (retrocompatibile con ASCII)**, **2 byte per altri 1920 caratteri relativamente comuni** (lingue europee e arabe), 3 e 4 byte per gli altri caratteri più rari e per i simboli
- UTF-16 e UTF-8 sono esempi di codici a **lunghezza variabile**, in cui cioè la rappresentazione di elementi diversi dell'alfabeto originario è fornita da stringhe di lunghezze diverse (di solito usando stringhe più brevi per simboli più frequenti)

# Esempi UTF-32 e UTF-8

Carattere	Unicode	UTF-32	UTF-8
a	U+000061	$(00)_{16} = (0000\ 0000)_2$ $(00)_{16} = (0000\ 0000)_2$ $(00)_{16} = (0000\ 0000)_2$ $(61)_{16} = (0110\ 0001)_2$	$(0110\ 0001)_2 = (61)_{16}$  Unicode $\leq (7F)_{16}$ viene codificato con <b>un byte</b> ( <a href="#">come in codifica ASCII</a> )
à	U+0000E0	$(00)_{16} = (0000\ 0000)_2$ $(00)_{16} = (0000\ 0000)_2$ $(00)_{16} = (0000\ 0000)_2$ $(E0)_{16} = (1110\ 0000)_2$	$(1100\ 0011)_2 = (C3)_{16}$ $(1010\ 0000)_2 = (A0)_{16}$  Unicode $\leq (7FF)_{16}$ e $> (7F)_{16}$ viene codificato con <b>due byte</b> : il primo inizia sempre con 110, il secondo con 10
€	U+0020AC	$(00)_{16} = (0000\ 0000)_2$ $(00)_{16} = (0000\ 0000)_2$ $(20)_{16} = (0010\ 0000)_2$ $(AC)_{16} = (1010\ 1100)_2$	$(1110\ 0010)_2 = (E2)_{16}$ $(1000\ 0010)_2 = (82)_{16}$ $(1010\ 1100)_2 = (AC)_{16}$  Unicode $\leq (FFFF)_{16}$ e $> (7FF)_{16}$ viene codificato con <b>tre byte</b> : il primo inizia sempre con 1110, gli altri con 10

# Bit map: un codice ridondante per simboli alfanumerici

- Rappresentazione binaria dei simboli **grafici** dei caratteri
- Mentre nelle stampanti «ad impatto» basta il codice ASCII, per molti altri casi (getto d'inchiostro, laser, ma anche per mostrare a monitor) è necessario definire dei simboli grafici mediante matrici di pixel (**bitmap**)
- Ciascun elemento di una bitmap è rappresentato da un insieme di bit
  - Bianco/nero: 1 pixel -> 1 bit
  - Scala di grigi: 1 pixel -> 8 bit
  - Colori RGB: 1 pixel -> 3x8 bit
- **Font:** insieme dei simboli grafici caratterizzati da un certo stile



Es. di bitmap 8x8

# Codici “nascosti”



