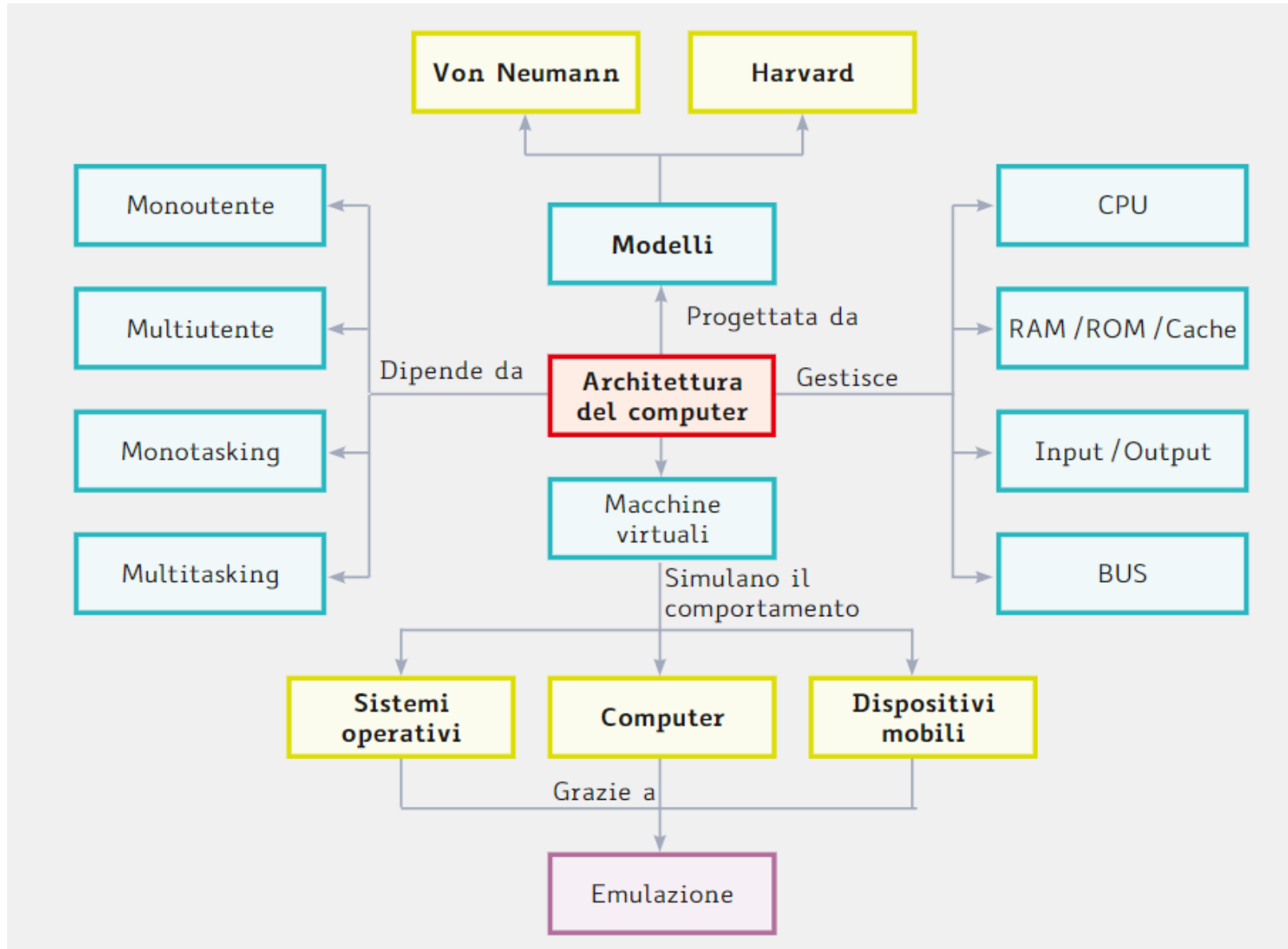


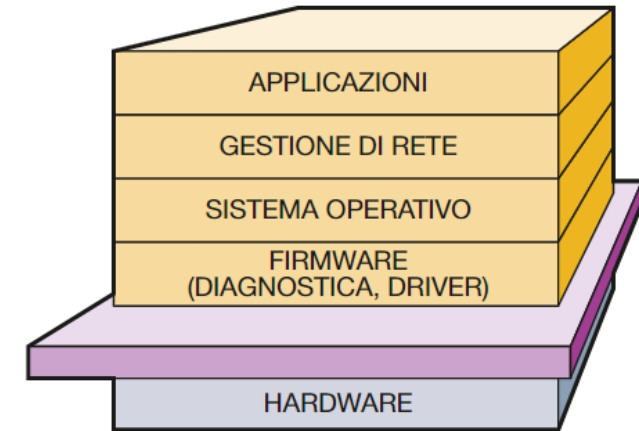
# Architetture dei sistemi di elaborazione

- Mappa concettuale
- Definizioni: architettura e legge di Moore
- Matematica dei circuiti digitali
- Progettazione di circuiti digitali
- Modelli di Von Neumann e di Harvard
- CPU
- Memorie
- BUS
- Assemblaggio di un calcolatore



Un sistema di elaborazione può essere definito come l'insieme di **hardware** e **software**.

Il confine tra i due elementi è dato da uno strato intermedio, chiamato **firmware**, formato dall'insieme dei programmi (sequenza di istruzioni) memorizzati direttamente sui circuiti elettronici (software cablato): il firmware è integrato direttamente in un componente elettronico programmato (es. UEFI - interfaccia informatica tra il firmware e il sistema operativo di un PC progettata per sostituire il BIOS a partire dal 2010 - su ROM).

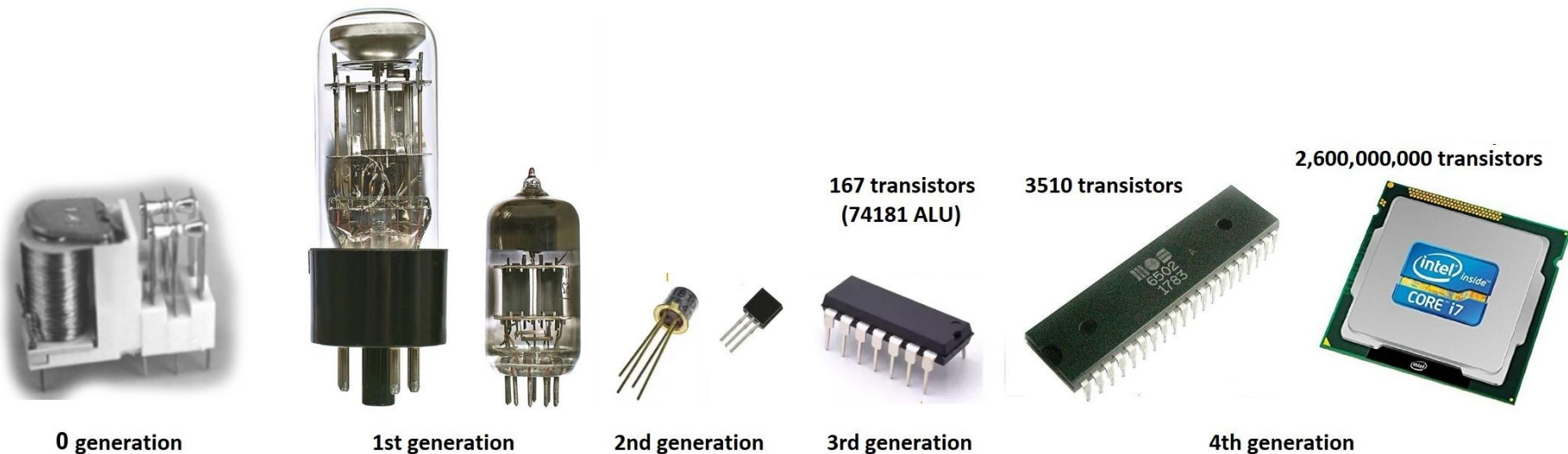


L'architettura dei computer studia le tecniche con le quali i componenti di un sistema di elaborazione vengono progettati e uniti logicamente tra loro.

L'architettura è l'insieme di concetti, metodologie e tecniche per definire, progettare e valutare un sistema. Mediante l'architettura dei calcolatori vengono progettati nuovi sistemi di elaborazione di diversa complessità formati da componenti fisici di tipo elettronico.

Evoluzione tecnologica dei «mattoni» costruttivi dei computer:

- **zero generazione:** i computer adottano dei relè elettromeccanici
- **prima generazione:** i computer adottano la tecnologia delle valvole termoioniche (tubi elettronici)
- **seconda generazione:** i computer adottano la tecnologia dei transistor
- **terza generazione:** i computer adottano la tecnologia dei primi circuiti integrati
- **quarta generazione:** i computer adottano la tecnologia dei circuiti integrati ad alta densità di componenti



## Legge di Moore – capacità di integrazione

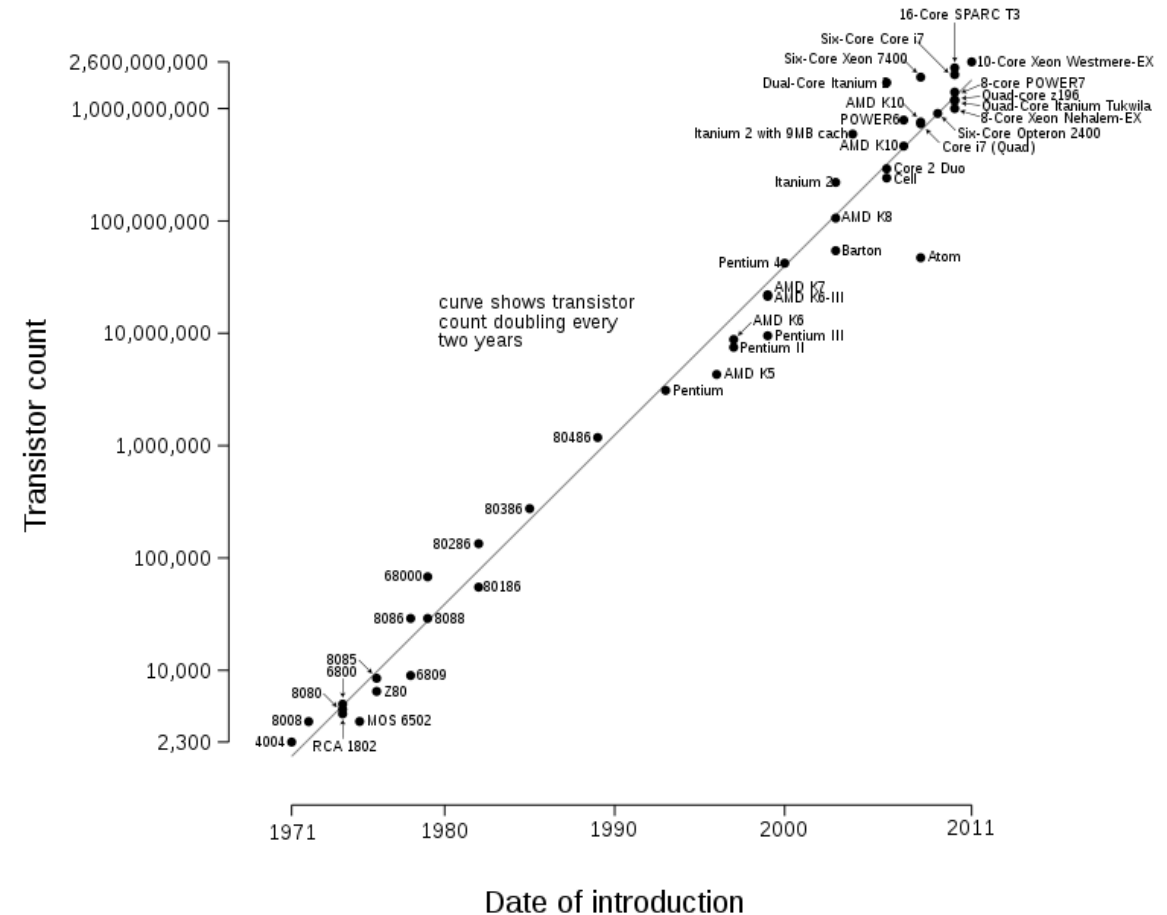
Nel 1965 Moore ipotizzò che il numero di transistori nei microprocessori sarebbe raddoppiato ogni 12 mesi circa.

Nel 1975 questa previsione si rivelò corretta e prima della fine del decennio i tempi si allungarono a due anni, periodo che rimarrà valido per tutti gli anni ottanta.

La legge, che verrà estesa per tutti gli anni novanta e resterà valida fino ai nostri giorni, viene riformulata alla fine degli anni ottanta ed elaborata nella sua forma definitiva, ovvero che il **numero di transistori nei processori raddoppia ogni 18 mesi.**

Questa legge è diventata il metro e l'obiettivo di tutte le aziende che operano nel settore come Intel e AMD.

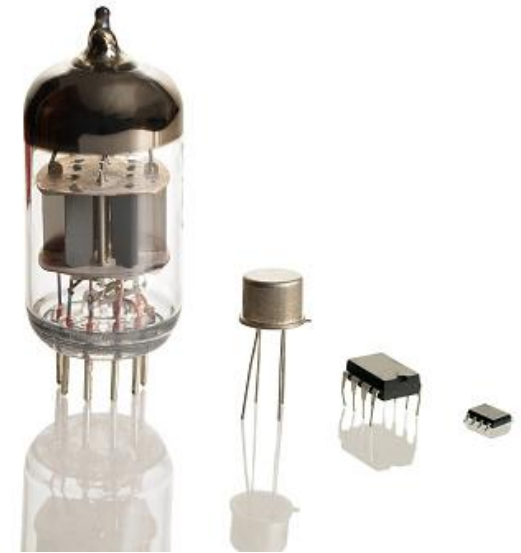
## Microprocessor transistor counts 1971-2011 & Moore's law



Le dimensioni dei sistemi di elaborazione sono progressivamente diminuite nel corso del tempo grazie all'aumento della **capacità di integrazione** degli elementi elettronici di cui sono composti.

La capacità di integrazione quantifica all'incirca quanti transistor sono in esso contenuti:

- **SSI** (Small Scale Integration): meno di 10 transistor
- **MSI** (Medium Scale Integration): da 10 a 100 transistor
- **LSI** (Large Scale of Integration): da 100 a 10.000 transistor
- **VLSI** (Very Large Scale Integration): da 10.000 a 100.000 transistor
- **ULSI** (Ultra Large Scale Integration): fino a 10 milioni di transistor

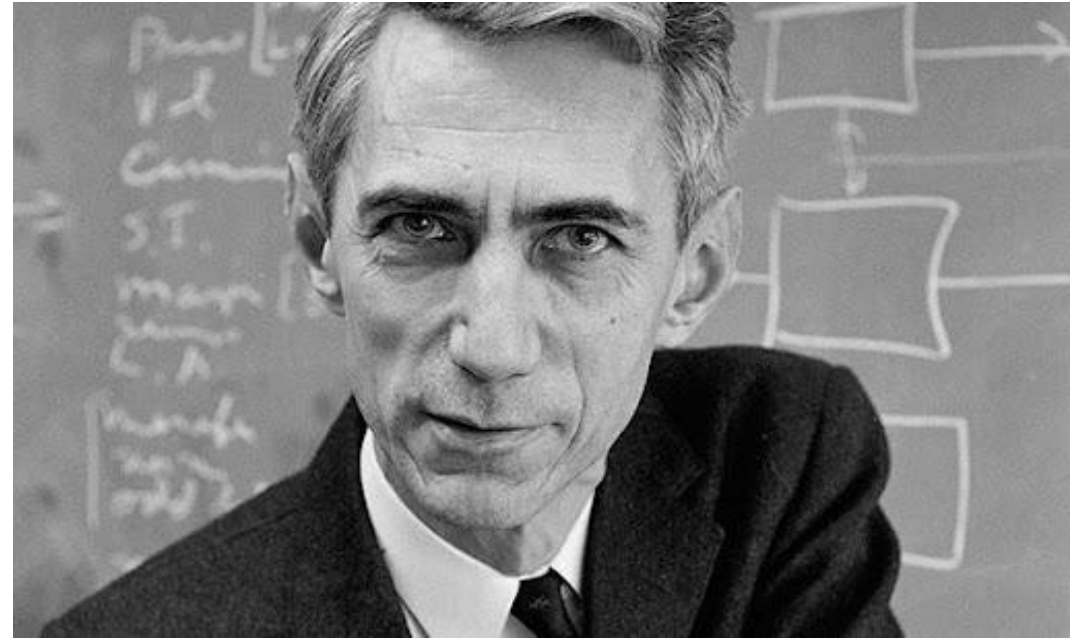


# Matematica dei circuiti digitali



Nel **1938**, studiando i circuiti elettrici a relè comunemente utilizzati nelle telecomunicazioni, il fisico e matematico americano **Claude Shannon** (1916-2002) del MIT, si rese conto che il loro funzionamento poteva essere descritto in termini logici utilizzando il calcolo proposizionale.

Con la tesi per il master scrive un importante lavoro sull'uso dell'algebra di Boole per progettare e ottimizzare i circuiti elettrici di commutazione a relè facendo intuire la superiorità dell'approccio digitale rispetto a quello analogico.



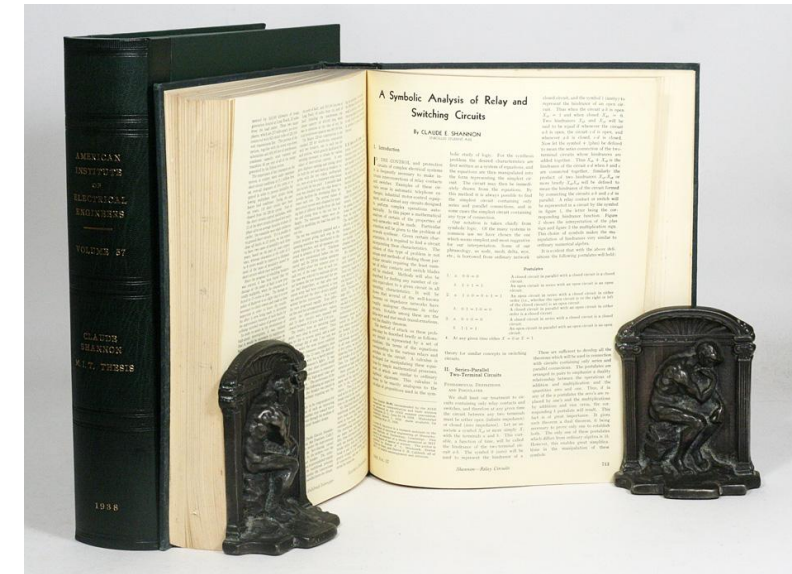
Nel **1948** pubblicò «A Mathematical Theory of Communication», probabilmente il lavoro più importante per tutta la storia della teoria dell'informazione.

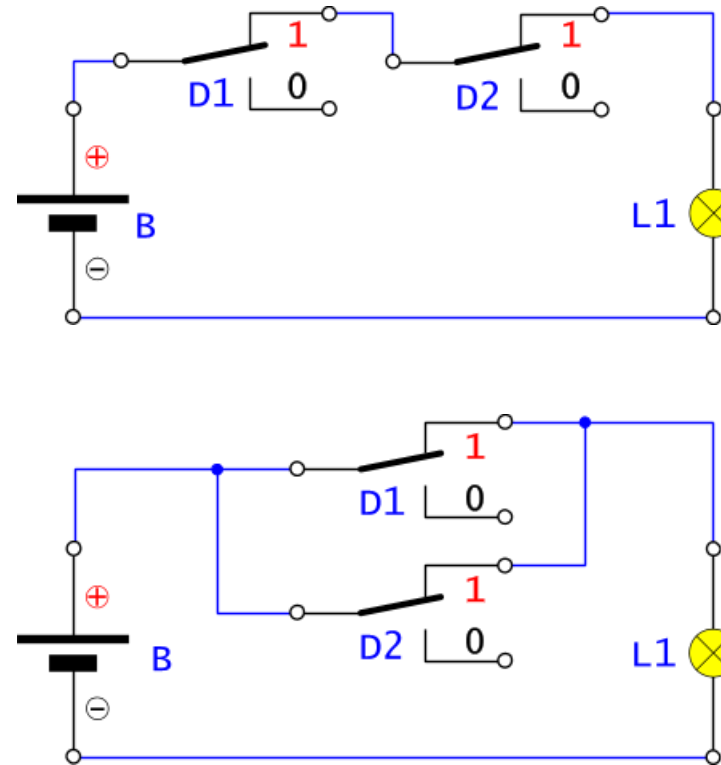
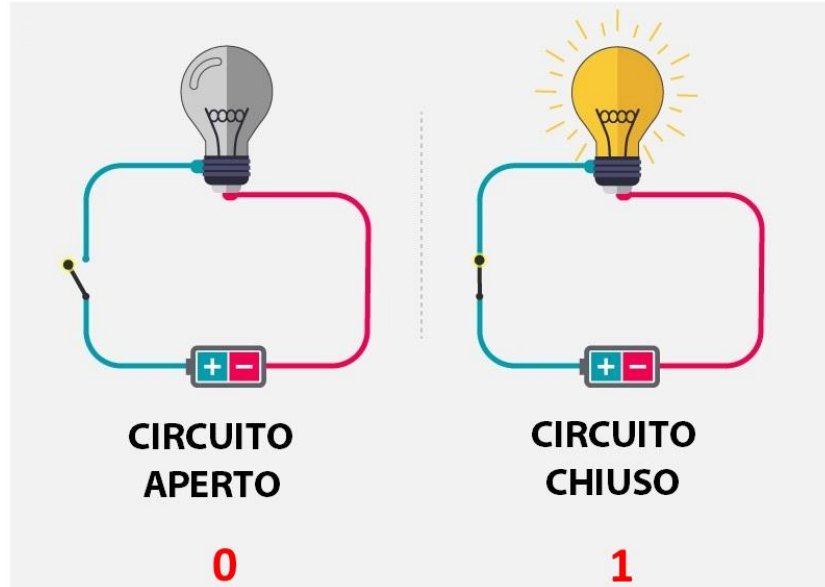
In questo articolo, Shannon teorizza che qualunque tipo di informazione, (testo, immagine, suono, etc.) possa essere trasmessa secondo le stesse leggi codificate in forma di 0 e 1, che prendono il nome di **bit** (contrazione di **b**inary **d**igit), termine suggerito dallo statistico J.Turkey e utilizzato per la prima volta proprio nell'articolo di Shannon. Oggi il bit costituisce un concetto atomico fondamentale per tutta l'informatica e per la teoria dell'informazione.

Nel lavoro del 1938 non si parla ancora di computer, ma ci si limita a menzionare complessi sistemi di controllo automatici. Nonostante questo, le sue idee si riveleranno fondamentali per lo sviluppo successivo dei calcolatori.

Il contributo fondamentale di Shannon fu soprattutto quello di introdurre un metodo sistematico per progettare reti logiche (si parla di *sintesi di reti logiche*) capaci di eseguire le operazioni logico-aritmetiche desiderate.

Egli mostrò come trasformare un algoritmo in un circuito elettrico costruito semplicemente con interruttori e relè di commutazione (oltre che da fili di collegamento e batterie).  
Le informazioni sono rappresentate essenzialmente mediante il passaggio o non passaggio di segnali elettrici, controllati semplicemente da interruttori aperti (0) e chiusi (1).





Serie di interruttori:  
**AND** logico

$$L1 = D1 \cdot D2$$

Parallelo di interruttori:  
**OR** logico

$$L1 = D1 + D2$$

Anche se già altri lo avevano preceduto nel mettere in evidenza il legame esistente tra logica proposizionale e circuiti elettrici (es. il filosofo e logico americano Charles Pierce nel 1886 e Paul Ehrenfest dell'Università di S.Pietroburgo nel 1910), Shannon fu il primo a sviluppare in modo sistematico l'argomento ricevendone la dovuta attenzione.

## Algebra di Boole

L'algebra booleana è un sistema di logica matematica a due stati, dove le variabili (dette *variabili booleane*) possono assumere solo due stati: vero (1) o falso (0).

L'algebra booleana venne ideata dal matematico inglese George Boole (1815 –1864) nel XIX secolo.

Nel XX secolo la logica booleana fu adottata da Claude Elwood Shannon per spiegare il funzionamento dei circuiti elettrici, poi elettronici e informatici.

Le principali operazioni binarie della logica booleana sono la somma, il prodotto e il complemento:

*addizione logica*

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

*moltiplicazione logica*

A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

*complemento logico*

A	$\bar{A}$
0	1
1	0

Ogni funzione booleana ha sempre un'espressione in **forma normale disgiuntiva (DNF)** composta da una o più clausole:

$C_1 + C_2 + \dots + C_n$  ogni clausola composta da una congiunzione di letterali  $C_i = ABC$


Un'espressione booleana in forma normale disgiuntiva può essere semplificata in **forma minimale**.

Un'espressione in forma minimale esprime la stessa funzione booleana, ha la stessa tavola di verità, ma è composta da un minor numero di clausole e letterali.

Per fare questa semplificazione basta applicare le leggi dell'**algebra booleana**.

*Esempio:*

$$ABC + AB = AB$$

*lo posso dimostrare confrontando le tabelle di verità* 

*Si può dimostrare anche applicando la Legge della dominanza:*

$$ABC + AB = AB(C+1) = AB1 = AB$$

*presa dalle leggi dell'algebra di Boole.*

A	B	C	ABC + AB	AB
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

Leggi dell'algebra di Boole:

Legge	forma AND	forma OR
Doppio complemento	$A = \overline{\overline{A}}$	
Elemento neutro	$1A = A$	$0 + A = A$
Legge di dominanza	$0A = 0$	$1 + A = 1$
Legge dell'idempotenza	$AA = A$	$A + A = A$
Legge dell'inversione	$A\overline{A} = 0$	$A + \overline{A} = 1$
Proprietà commutativa	$AB = BA$	$A + B = B + A$
Proprietà associativa	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Proprietà distributiva	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Legge dell'assorbimento	$A(A + B) = A$	$A + AB = A$
Leggi di De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\overline{B}$

(dimostratele a casa utilizzando le tabelle di verità)

Si studia l'algebra booleana poiché **le funzioni dell'algebra booleana sono isomorfe ai circuiti digitali**.  
In altre parole, un circuito digitale può essere espresso tramite un'espressione booleana e viceversa.

Poiché le variabili possono assumere solo i valori 0 o 1 una funzione booleana con  $n$  variabili di input ha solo  $2^n$  combinazioni possibili e può essere descritta dando una tabella, detta **tabella di verità**, con  $2^n$  righe.

*Ad esempio, con 2 variabili (A,B)  
avrò  $2^2 = 4$  possibili combinazioni  
dei valori assunti da questa variabili  
e di conseguenza la tabella di verità  
sarà composta da 4 righe*

ingressi		uscita
A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

Tutte le funzioni booleane possono essere espresse come combinazione di somma, prodotto e complemento logico [viste in slide 11].

Ad ogni funzione di base corrisponde una porta logica e quindi ogni espressione booleana può essere tradotta in un circuito digitale fatto di porte logiche. Tramite le proprietà dell'algebra booleana è possibile semplificare espressione booleane complesse riducendo di conseguenza la complessità del circuito.

Per passare dalla rappresentazione mediante tabella di verità alla notazione tramite espressione booleana è necessario:

- 1) identificare tutte le righe della tabella di verità che danno 1 in output;
- 2) per ogni riga con un 1 in output scrivere la configurazione delle variabili che la definiscono (tutte le variabili della configurazione saranno in AND tra loro)
- 3) collegare tramite OR tutte le configurazioni ottenute

La rappresentazione così ottenuta è detta in **prima forma canonica**.

*Esempio:*

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$\bar{A}\bar{B}C$

$\bar{A}B\bar{C}$

$A\bar{B}\bar{C}$

$ABC$

$$Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

Espressione in prima forma canonica ovvero in forma normale disgiuntiva (DNF)

Convenzioni notazionali:

- Una variabile con valore 1 è indicata dal suo nome
- Una variabile con valore 0 è indicata dal suo nome con sopra una barra
- L'operazione di AND booleano è indicato da un  $\cdot$  moltiplicativo oppure viene considerato implicitamente presente
- L'operazione di OR booleano è indicato da un  $+$



## Esercizi:

- 1) Utilizzando gli operatori booleani AND, OR, NOT, si scrivano le seguenti funzioni booleane (ricavate dalle rispettive tabelle di verità):
  - a) funzione booleana che ritorna in uscita il valore 1 se sono veri un numero dispari dei tre input
  - b) funzione booleana che ritorna in uscita il valore 1 se sono veri solo due di quattro input
  - c) funzione booleana che riceve in ingresso un numero binario su 3 bit e ritorna in uscita il valore 1 se e solo se il numero che c'è in ingresso è maggiore o uguale a quattro.
  - d) funzione booleana avente come ingressi due numeri binari X e Y su 2 bit, che ritorni il valore 1 se  $X > Y$

- 2) Applicando i teoremi dell'algebra di Boole, verificare la seguente equivalenza tra espressioni:

$$\bar{A} \bar{B} \bar{C} + B\bar{C} + A(B + \bar{B}\bar{C}) = A + \bar{C}$$

- 3) Applicando i teoremi dell'algebra di Boole, semplificare le seguenti espressioni e disegnarne la tavola di verità:

$$A B \bar{C} + AB + AC + C$$

$$\bar{A} \bar{B} C + A\bar{B} + \bar{A} \bar{B} + AB$$

$$A + AB + B + BC$$

- 4) Svolgere gli esercizi da 1 a 10 presenti nel link: <http://www.edutecnica.it/sistemi/boolex/boolex.htm>

# Soluzioni:

1a)

A	B	C	U
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$\begin{aligned} U &= \bar{A} \bar{B} C + \bar{A} B \bar{C} + A \bar{B} \bar{C} + ABC \\ &= \bar{A} (\bar{B} C + B \bar{C}) + A (\bar{B} \bar{C} + BC) \\ &= \bar{A} (B \oplus C) + A (\overline{B \oplus C}) \\ &= A \oplus (B \oplus C) \end{aligned}$$

1b)

A	B	C	D	U
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$$\begin{aligned} U &= \bar{A} \bar{B} C D + \bar{A} \bar{B} \bar{C} D + \bar{A} B C \bar{D} + \bar{A} \bar{B} \bar{C} D + A \bar{B} C \bar{D} + A B \bar{C} \bar{D} \\ &= \bar{A} D (\bar{B} C + B \bar{C}) + \bar{A} B C \bar{D} + A \bar{B} \bar{C} D + A \bar{D} (\bar{B} C + B \bar{C}) \\ &= (\bar{A} D + A \bar{D}) (\bar{B} C + B \bar{C}) + \bar{A} B C \bar{D} + A \bar{B} \bar{C} D \\ &= (A \oplus D) (B \oplus C) + (\bar{A} \bar{D} \oplus BC) \end{aligned}$$

1c)

A	B	C	U
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$\begin{aligned}
 U &= A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC \\
 &= A\bar{B}(\bar{C} + C) + AB(\bar{C} + C) \\
 &= A\bar{B}1 + AB1 \\
 &= A(\bar{B} + B) \\
 &= A
 \end{aligned}$$

1d)

X1	X0	Y1	Y0	U
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$\begin{aligned}
 U &= \bar{X1} \bar{X0} \bar{Y1} \bar{Y0} + X1 \bar{X0} \bar{Y1} \bar{Y0} + \\
 &\quad X1 \bar{X0} \bar{Y1} Y0 + X1 \bar{X0} \bar{Y1} Y0 + \\
 &\quad X1 \bar{X0} \bar{Y1} Y0 + X1 \bar{X0} Y1 \bar{Y0} = \\
 &= \bar{Y1} \bar{Y0} (X1 \oplus X0) + \\
 &\quad X1 \bar{Y1} (X0 \oplus Y0) + \\
 &\quad X1 X0 (Y1 \oplus Y0)
 \end{aligned}$$

# Progettazione di circuiti digitali

I componenti elettronici che formano un computer possono essere sintetizzati in due categorie principali:

- generatori di segnali
- porte logiche

## Generatori di segnali

I generatori di segnali sono componenti in grado di produrre un segnale periodico utile a sincronizzare gli elementi hardware presenti nei computer.

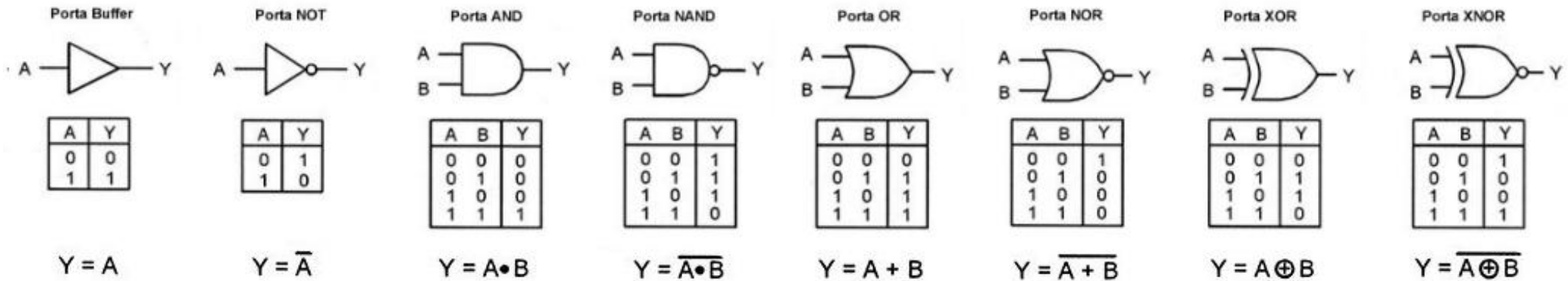


Ad esempio, un oscillatore al cristallo è un circuito elettronico che usa la risonanza meccanica di un cristallo piezoelettrico vibrante per ottenere un segnale elettrico a onda quadra caratterizzato da una frequenza molto precisa. Questa frequenza è comunemente usata per mantenere una sincronia (come negli orologi al quarzo), per ottenere un segnale di clock stabile per i circuiti integrati digitali.

Sopra è rappresentato un piccolo cristallo di quarzo a 16 MHz ermeticamente racchiuso in un package, usato come risuonatore in un oscillatore al quarzo.

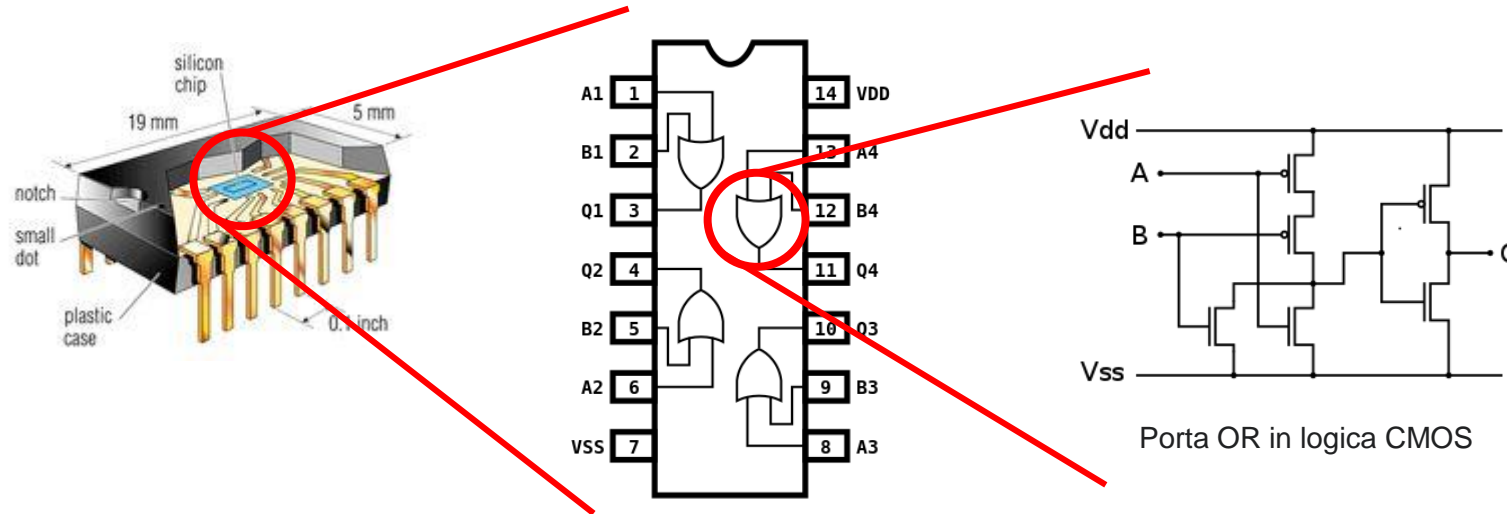
## Porte logiche

Sono dei circuiti elettronici elementari in grado di svolgere le operazioni logiche dell'algebra booleana basata sui valori logici VERO o FALSO (TRUE or FALSE). TRUE o FALSE corrispondono al passaggio o al non passaggio di corrente elettrica in tali circuiti e quindi livelli logici binari 0 e 1.



Ma come è fatta una porta logica?

In un circuito integrato possono coesistere diverse porte logiche, ad esempio prendiamo un componente al cui interno siano state implementate delle porte OR:



Ciascuna porta è fatta da un rete di transistor, di «interruttori programmabili»

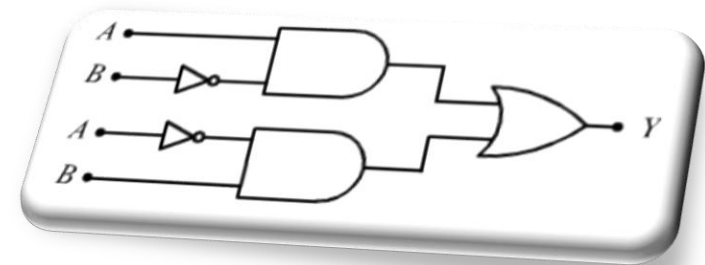
Attraverso l'assemblaggio di porte logiche vengono realizzati circuiti digitali, macchine elementari, suddivise in combinatorie e sequenziali.

Un circuito digitale può essere rappresentato con:

- tabella di verità
- funzione logica
- schema circuitale

ingressi			uscite	
A	B	C <sub>i</sub>	S	C <sub>o</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{A}\overline{B}C_i + \overline{A}B\overline{C}_i + A\overline{B}\overline{C}_i + ABC_i$$



Nella **progettazione di un circuito digitale** si seguono i seguenti step:

- 1) descrizione del comportamento del circuito
- 2) stesura della tabella di verità
- 3) determinazione della funzione logica corrispondente (in prima forma canonica o forma normale disgiuntiva)
- 4) eventuale semplificazione con le regole dell'algebra di Boole
- 5) disegno dello schema circuitale (includendo porte logiche e generatori di segnali)



## Esempio di progettazione di un circuito digitale: **SOMMATORE BINARIO**

L'addizione è la più elementare delle operazioni aritmetiche; l'addizione è più o meno la sola cosa che i computer fanno.

I sommatore binari sono reti combinatorie che ricevono in ingresso n bit degli addendi da sommare e generano in uscita i bit della somma binaria con il relativo riporto.

Si tratta, dunque, di un tipico esempio di rete combinatoria con ingressi multipli ed uscite multiple, strutturata in modo da seguire il meccanismo secondo cui avviene la somma binaria.

Esistono due possibili blocchi funzionali che eseguono la somma binaria:

- il semisommatore binario HA (**Half Adder**) senza riporto in ingresso ;
- il sommatore binario FA (**Full Adder**) con riporto in ingresso.

# HALF ADDER

La differenza tra la somma in decimale e la somma in binario è che per quest'ultimo sistema lo schema per la somma è molto più semplice:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

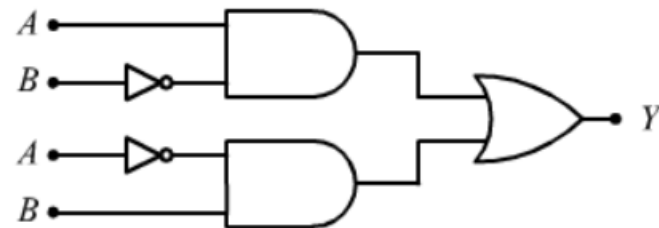
$$1 + 1 = 0 \text{ con riporto } 1$$

non è altro che la tabella della verità della porta XOR (a parte la questione del riporto che deve essere risolta):

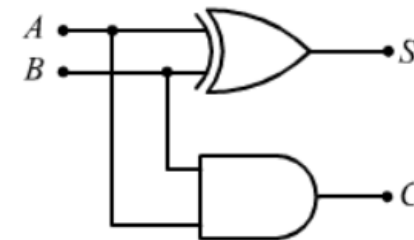
$$Y = A \oplus B = A\bar{B} + \bar{A}B$$



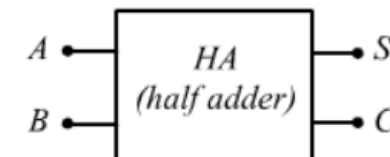
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



Per gestire anche il riporto questo circuito deve essere modificato in modo da risolvere la seguente tabella della verità (dove A e B sono i bit da sommare, S è il bit della somma (sum) e C è il bit del riporto (carry)):



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = A \oplus B$$

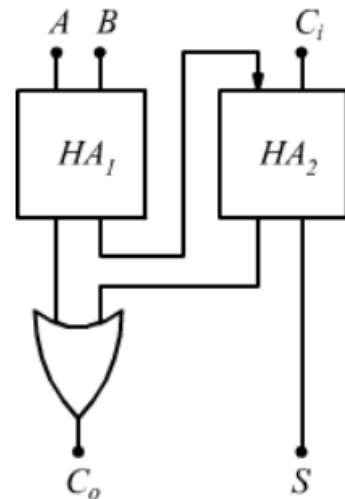
$$C = A \cdot B$$

FULL ADDER

Per ottenere la somma completa (full-adder) tra due numeri di più cifre, oltre ai bit dello stesso ordine occorre sommare anche il riporto eventualmente ottenuto dai due bit di ordine immediatamente inferiore. Per questo motivo il circuito full-adder si presenta con tre ingressi e due uscite.

I due **ingressi** sono costituiti dai due bit  $A_n$  e  $B_n$  da sommarsi e dal riporto  $C_i$  eventualmente ottenuto dalla somma dei due bit  $A_{n-1}$  e  $B_{n-1}$ .

Le due **uscite** sono composte dal bit di somma dei tre ingressi e dall'eventuale riporto  $C_o$  da inviare al full-adder successivo. Qui sotto è mostrato il simbolo logico, la tabella della verità e lo schema circuitale di un full-adder:



ingressi			uscite	
$A$	$B$	$C_i$	$S$	$C_o$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$S = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i$$

$$C_o = \bar{A}BC_i + A\bar{B}C_i + AB\bar{C}_i + ABC_i$$

Dalla tabella della verità è possibile dedurre la funzione logica della somma eseguita di questo circuito combinatorio e la funzione logica del riporto:

$$S = \overline{A}\overline{B}C_i + \overline{A}B\overline{C}_i + A\overline{B}\overline{C}_i + ABC_i$$

$$S = C_i(\overline{A}\overline{B} + AB) + \overline{C}_i(\overline{A}B + A\overline{B})$$

$$S = C_i(\overline{A \oplus B}) + \overline{C}_i(A \oplus B)$$

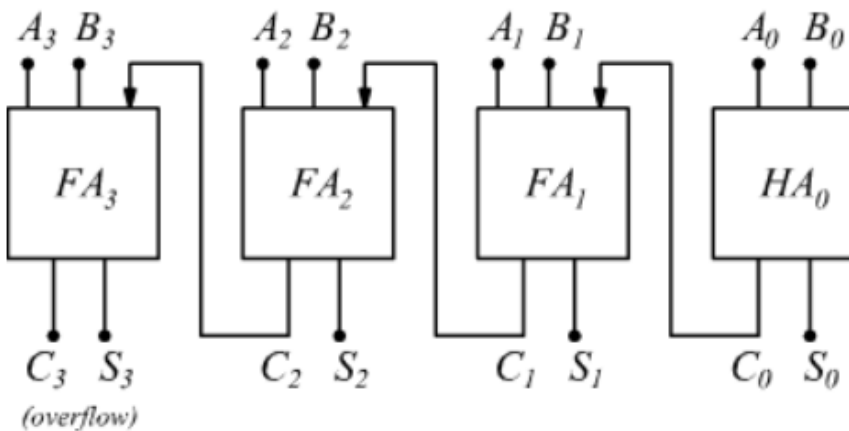
$$S = (A \oplus B) \oplus C_i$$

$$C_o = \overline{A}BC_i + A\overline{B}C_i + AB\overline{C}_i + ABC_i$$

$$C_o = C_i(\overline{A}B + A\overline{B}) + AB(\overline{C}_i + C_i)$$

$$C_o = C_i(A \oplus B) + AB$$

Quando si vogliono sommare numeri di più bit ciascuno, il metodo più semplice è quello di realizzare un sommatore parallelo come disegnato qui:



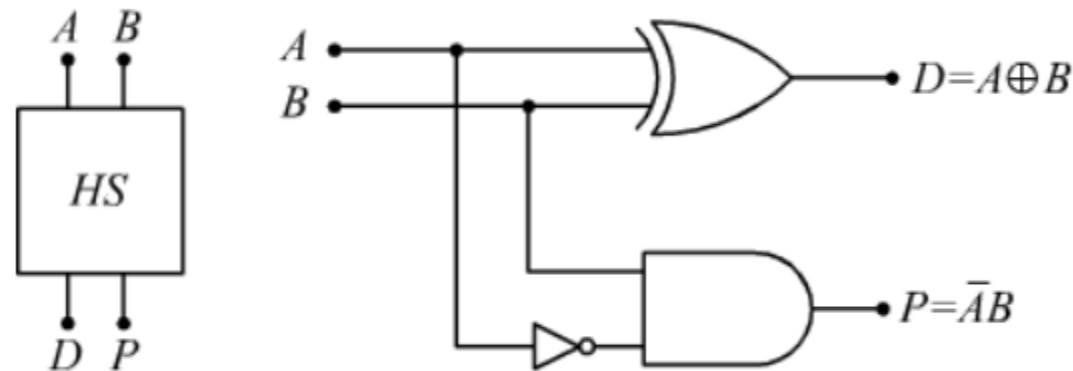
Esempio di progettazione di un circuito digitale: **SOTTRATTORE BINARIO**SEMISOTTRATTORE

A	B	D = A-B	P (prestito)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

La differenza (D)  $0 - 1 = 1$  con prestito (P) di 1 in quanto abbiamo dovuto prendere in prestito una cifra dalla posizione precedente. Prestandoci una cifra, la sottrazione diventa  $10 - 1$  che in binario è uguale a 1.

$$D = A\bar{B} + \bar{A}B = A \oplus B$$

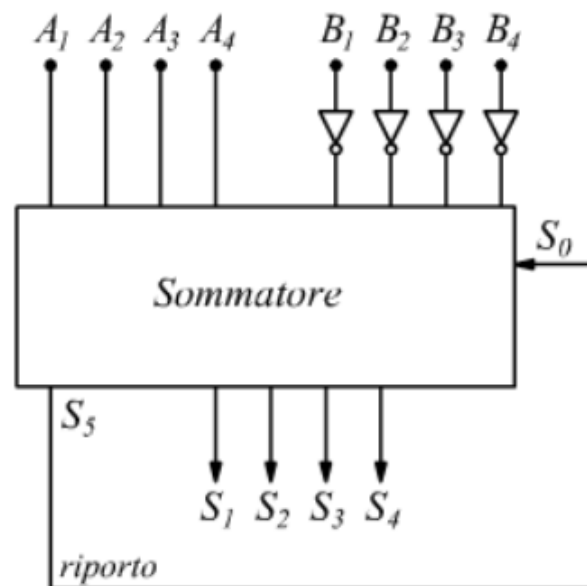
$$P = \bar{A}B$$



Bisogna dire che in caso di calcoli complessi, i sottrattori sono poco usati. Si preferisce effettuare l'operazione tramite una **addizione con complementazione**.

L'operazione aritmetica di sottrazione viene effettuata aggiungendo al minuendo (A) il **complemento a uno** del sottraendo ( $B_{C1}$ ) (invertendo quindi ciascun bit del sottraendo)

Il sottrattore è, in questo caso, simile ad un sommatore al quale uno dei due numeri giunge complementato.



L'eventuale riporto, ottenuto dalla somma dei bit significativi, deve essere aggiunto alla somma dei bit meno significativi.

Si deve osservare che il riporto si ha solo nel caso in cui il minuendo è maggiore del sottraendo e mai nel caso opposto.

caso **A** minuendo maggiore del sottraendo

complemento a 1 del sottraendo

$$\begin{array}{r}
 12- \quad 1100- \quad 1100+ \\
 5= \quad 0101= \quad 1010= \\
 \hline
 7 \qquad \qquad \quad 10110+ \\
 \text{bit di riporto} \quad \rightarrow 1= \\
 \hline
 0111
 \end{array}$$

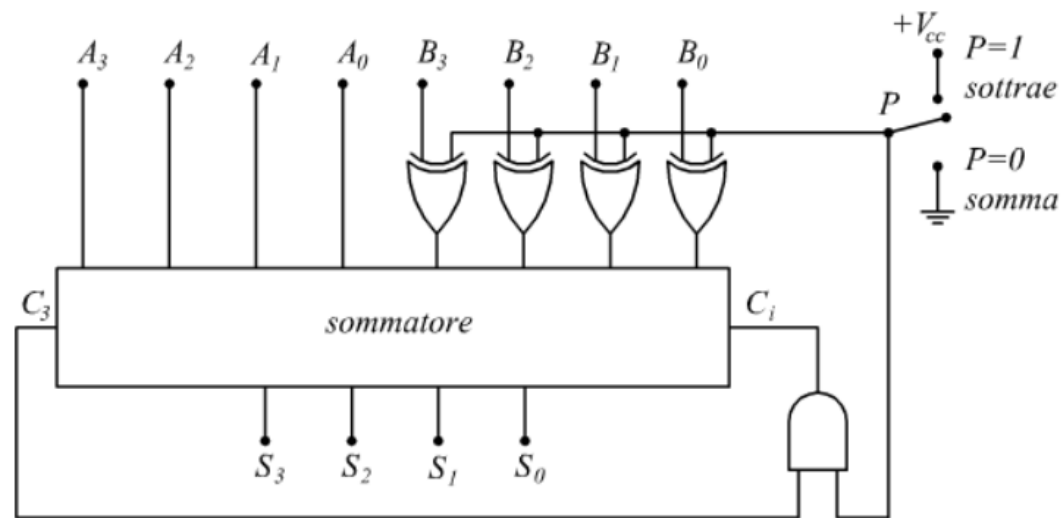
caso **B** minuendo minore del sottraendo

complemento a 1 del sottraendo

$$\begin{array}{r}
 8- \quad 1000- \quad 1000+ \\
 11= \quad 1011= \quad 0100= \\
 \hline
 -3 \qquad \qquad \quad 1100 \\
 \text{complementato a 1} \quad 0011
 \end{array}$$

Se minuendo < sottraendo, sappiamo già che il risultato è negativo come segno per cui basta prendere il risultato (1100) e complementarlo nuovamente a 1 per conoscere il modulo del numero:  $(0011)_2 = (3)_{10}$ .

Il circuito da realizzare deve quindi essere in grado di complementare eventualmente un addendo e poi deve tener conto del riporto causato dalla somma dei due bit più significativi. Partendo da un sommatore semplice, un circuito capace di eseguire la sottrazione con il metodo del complemento ad 1 può essere schematizzato dal seguente disegno:



L'uscita  $C_3$  costituisce il riporto eventuale ottenuto dalla somma di  $A_3$  e  $B_3$  mentre  $C_i$  è l'ingresso che deve essere sommato ad  $A_0$  e  $B_0$ .

Per fare in modo che il circuito possa funzionare da sommatore o da sottrattore occorre dotarlo di un comando che complementi o non complementi uno dei due nibble, ed inoltre impedisca o no la somma dell'ultimo riporto con i bit meno significativi.

Dallo schema si riconosce come l'operazione di complementazione ad 1 possa essere eseguita su un singolo bit da una **porta XOR**.



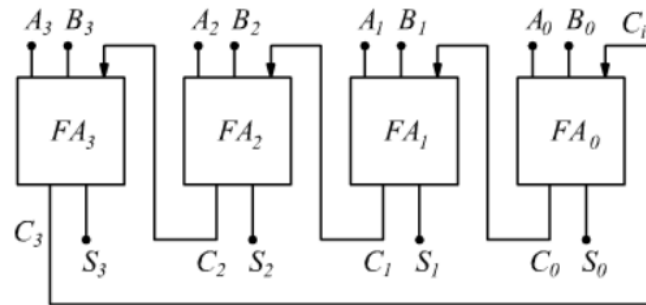
A	X	Y
0	0	0
0	1	1
1	0	1
1	1	0

Si vede come quando la variabile  $X=1$  la variabile  $A$  venga commutata (complementata ad 1) mentre quando  $X=0$  la variabile  $A$  non venga complementata:

ho una **porta NOT comandata** dall'ingresso  $X$ .

La variabile  $X$  applicata ad ogni singolo bit del sottraendo  $B$  sarà determinata dalla posizione del selettore  $P$ .

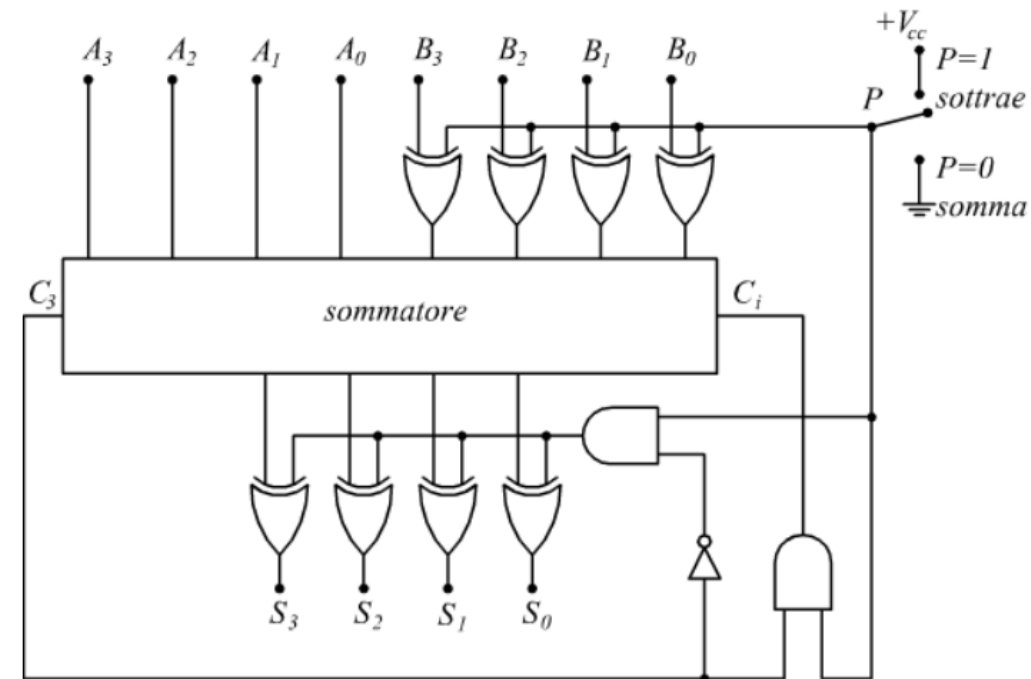
L'eventuale bit di riporto  $C_3$  dovrà coincidere con il riporto in ingresso  $C_i$  associato ai due bit meno significativi; si deduce che il dispositivo è costituito da 4 FA e non da 3 FA e un HA sul LSB come nel sommatore puro:



rimane solo una correzione da fare: nel caso in cui l'operazione sia una sottrazione ( $P=1$ ) ed il risultato sia eventualmente negativo ( $C_3=0$ ) deve essere eseguita una complementazione su quest'ultimo ( $S_{C1}$ ).

Questo può essere effettuato con la stessa struttura di porte prevista sull'operando B.

A lato viene dunque rappresentato un **circuito sommatore-sottrattore** completo:





Esempio di progettazione di un circuito digitale:

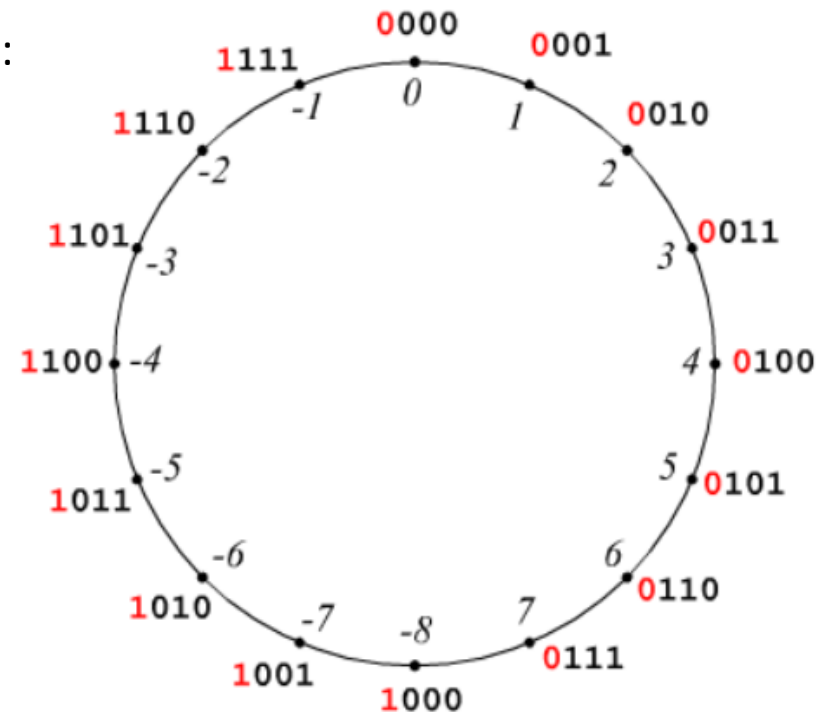
## SOTTRATTORE CON SISTEMA BINARIO IN COMPLEMENTO A 2

Solitamente nei sistemi digitali i **numeri interi relativi** vengono rappresentati con il metodo del **complemento a 2**, utilizzando il bit più pesante come bit di segno:

0 = numero positivo

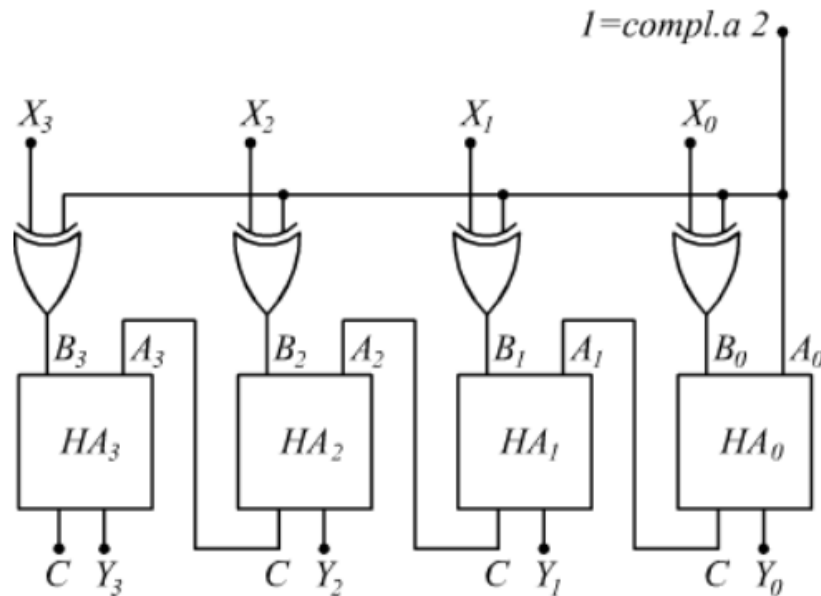
1 = numero negativo

quindi in un sistema a 4 bit avremo le seguenti eventualità:

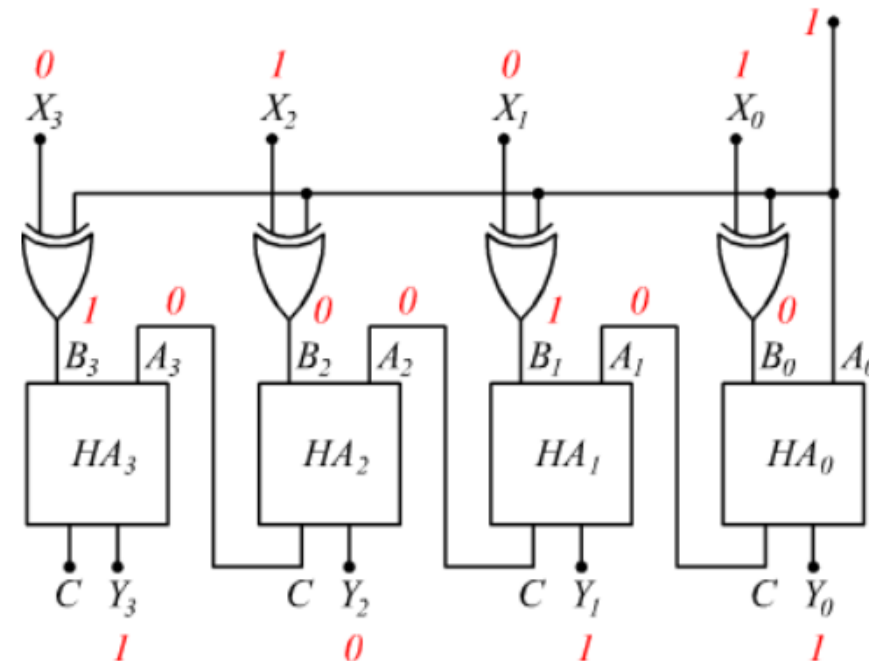


I circuiti sommatore sono indispensabili anche in questo caso. Il **circuito** disegnato sotto ci permette di eseguire il **complemento a 2 di un numero a 4 bit**.

Se il terminale  $A_0 = 1$  si ha la complementazione a 2 dei bit in ingresso  $X$  ( $X_{C2} = X_{C1} + 1$ ) che saranno rappresentati dai bit  $Y$  in uscita; se  $A_0 = 0$  il numero rimane invariato ( $X_3X_2X_1X_0 = Y_3Y_2Y_1Y_0$ ):

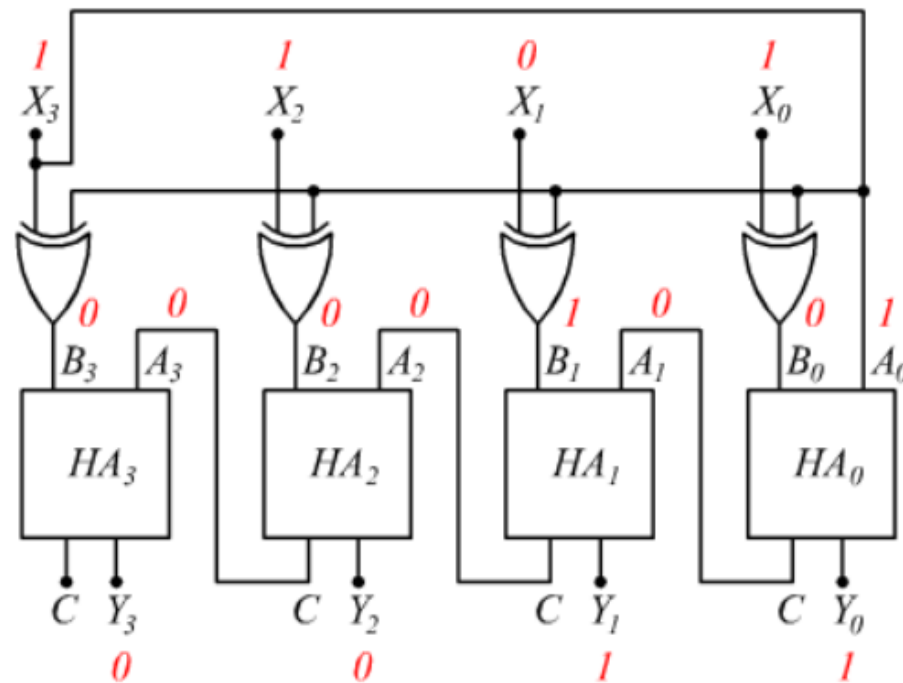


Supponiamo infatti di voler complementare il numero  $+5 = 0101$ , dopo aver posto  $A_0 = 1$  ecco cosa accade:



infatti  $1011 = -5$

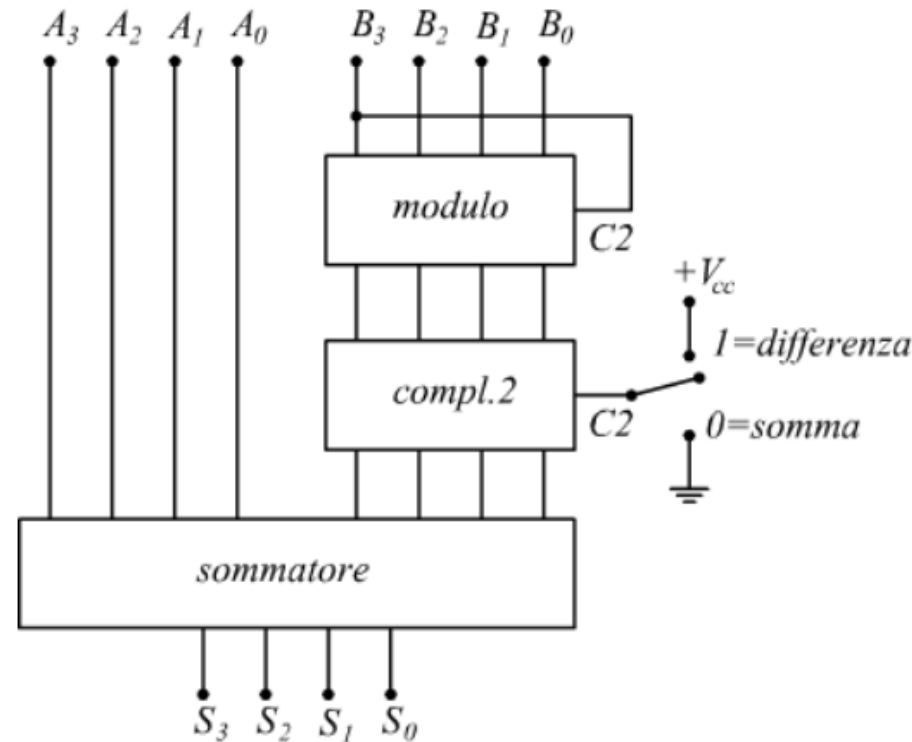
Il dispositivo precedente può anche essere facilmente usato per **ricavare il modulo del numero in ingresso**: se infatti siamo in presenza di un numero negativo il bit  $X_3$  può pilotare l'ingresso di complementazione  $A_0$ . Supponiamo, infatti di voler calcolare il modulo di  $1101_{C2}$  basterà (rispetto al circuito precedente) porre  $X_3=A_0$ :



Infatti  $1101_{C2} = -3$  e  $|-3| = 3 = 0011_{C2}$

Nel **sistema binario con rappresentazione a complemento a 2**, la differenza tra due numeri positivi avviene complementando a 2 il sottraendo e poi sommando quest'ultimo valore al minuendo.

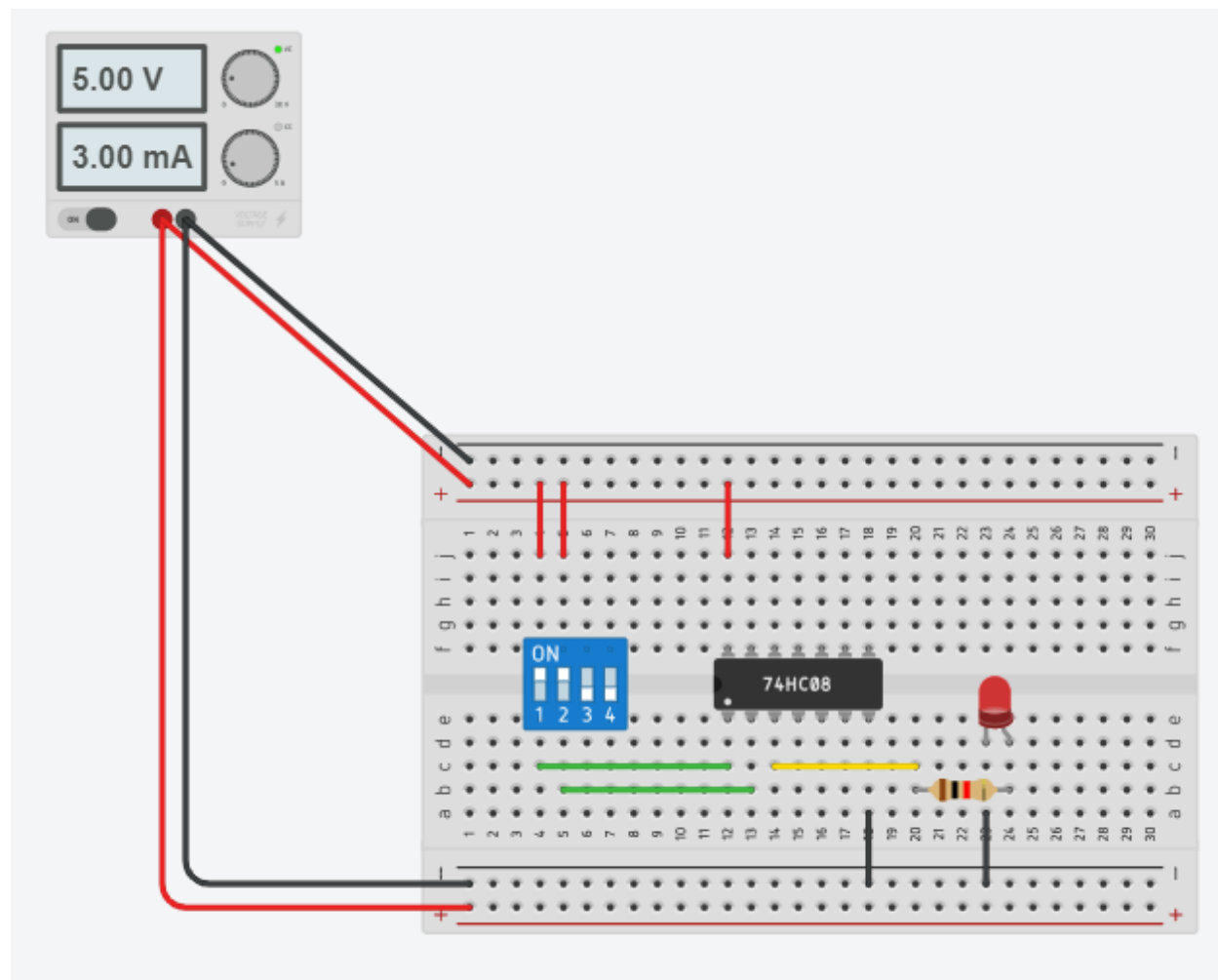
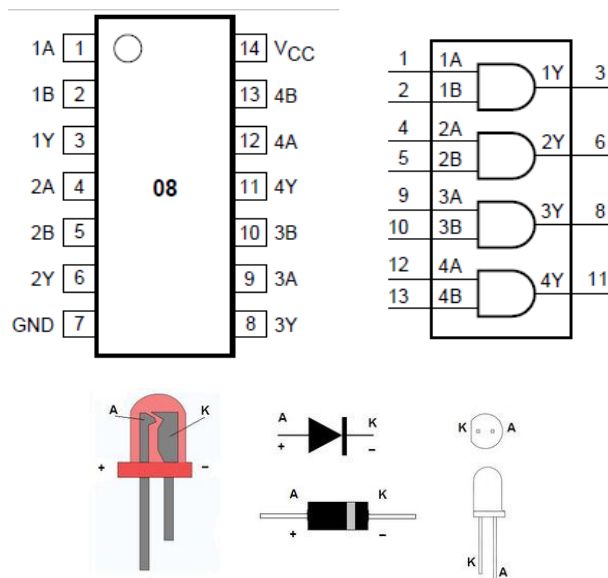
Supponiamo di voler effettuare l'operazione  $A \pm |B|$  con A numero naturale positivo e B numero positivo in complemento a 2. Ipotizzando di operare sempre in un sistema di 4 bit possiamo collegare i dispositivi precedenti nel seguente modo:



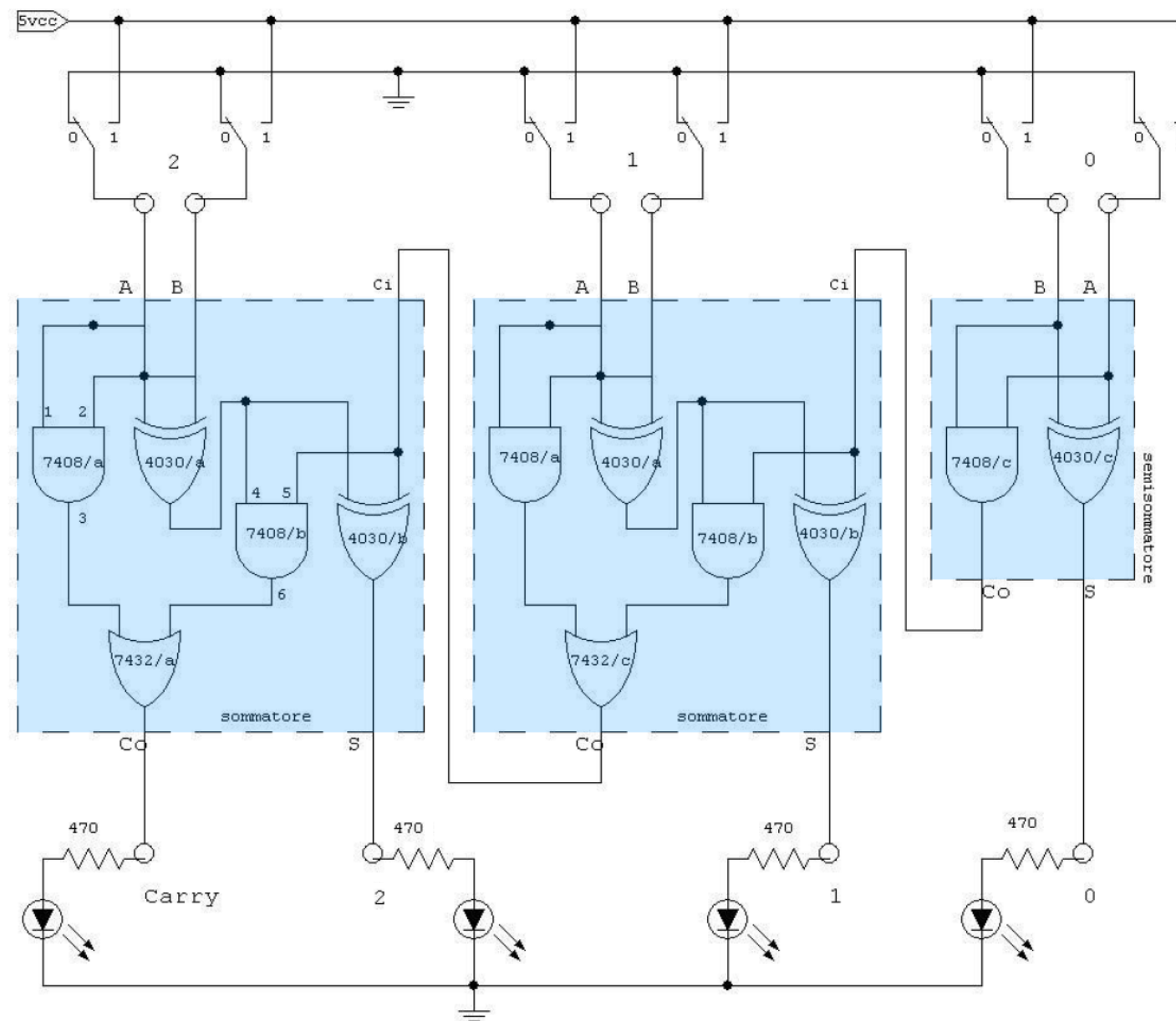
Con la piattaforma Tinkercad, realizziamo un circuito che utilizza una porta logica AND (useremo l'integrato 74HC08, porta AND quadrupla):

### Componenti:

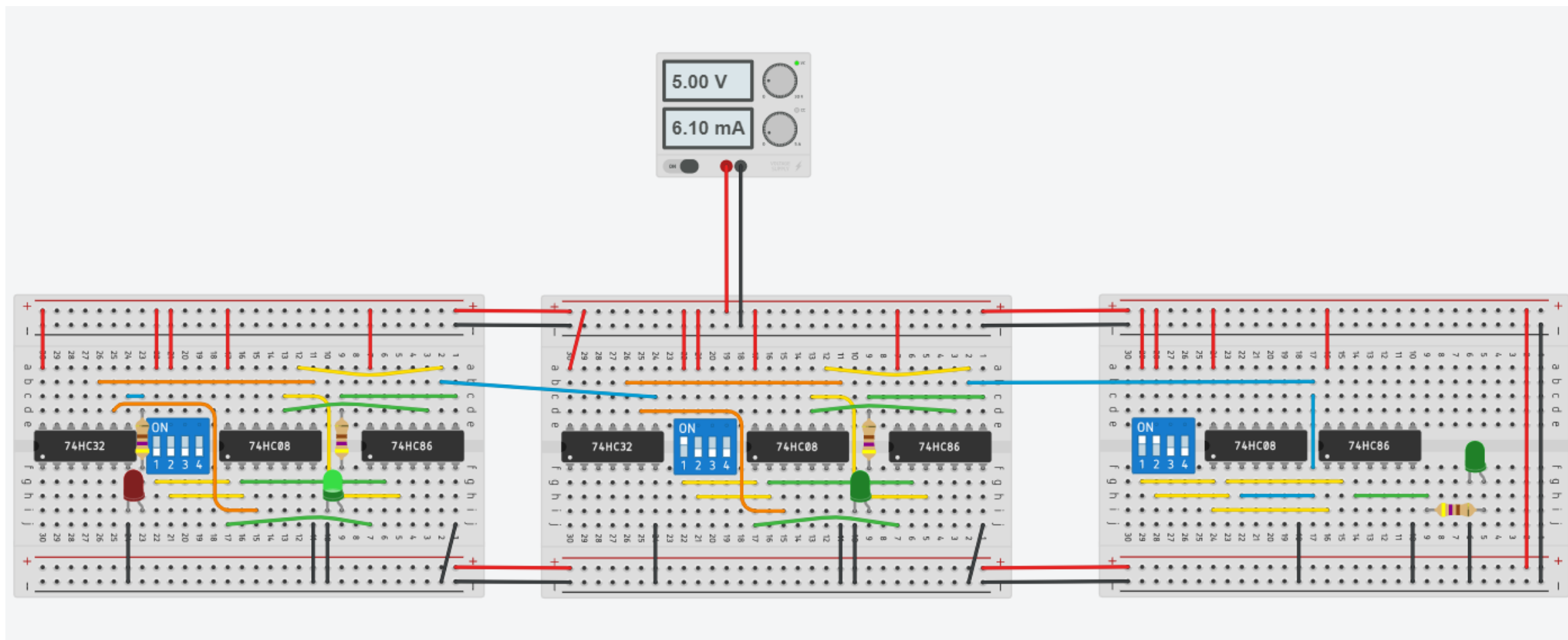
- 1 Alimentatore 5[V], 0.5[A]
- Interruttore SPST DIP x 4
- LED rosso
- Resistenza 1 K $\Omega$
- Porta AND quadrupla (componente 74HC08)



# Schema circuitale



Con la piattaforma Tinkercad, realizziamo il circuito:



## **Esercizi:**

Svolgere gli esercizi da 11 a 24 presenti nel link: <http://www.edutecnica.it/sistemi/boolex/boolex.htm>



# Modello Von Neumann e Harvard

## Modello di Von Neumann

Il modello di Von Neumann descrive il comportamento di una macchina che il suo inventore chiamò **stored-program computer** (esecutore sequenziale dotato di programma memorizzato).

Si tratta di uno schema a blocchi che illustra il modello di funzionamento di massima di un computer, ideato dal ricercatore ungherese Janus Neumann, naturalizzato americano (John von Neumann) dopo la fuga dal nazismo tedesco, durante la metà degli anni Quaranta del Novecento.

Il modello è stato ideato per la prima volta durante la progettazione del primo computer elettronico, chiamato **IAS machine** presso l'Institute for Advanced Study (IAS appunto), a Princeton negli Stati Uniti a cavallo tra il 1945 e il 1951. John von Neumann morì nel 1957 a soli 53 anni lasciando molti progetti che verranno sviluppati più tardi presso i centri di ricerca militari di Los Alamos, negli Stati Uniti.



## Stored-program computer

indica le istruzioni che la **CPU** deve eseguire.

Tali istruzioni sono collocate (*stored*) nella memoria del computer e l'insieme delle istruzioni rappresenta il programma (*program*) che deve essere eseguito.

Nella memoria risiedono, oltre alle istruzioni in linguaggio assembly dei programmi in corso di esecuzione, anche i dati sui quali tali programmi operano

rappresenta la **CPU** che compie azioni di elaborazione come, per esempio, prelevare o modificare il contenuto della memoria (*memory*), prelevare o modificare informazioni dai dispositivi di input/output fornendo informazioni in uscita oppure leggendo informazioni in ingresso (pensiamo per esempio alla tastiera o al monitor).

La CPU è in grado di eseguire le proprie azioni in modo sequenziale, cioè una alla volta, a una velocità assai elevata. La misura della velocità della CPU è strettamente legata alla **frequenza del clock** che indica il numero di azioni al secondo che essa deve compiere (*1 GHz rappresenta una frequenza di un miliardo di operazioni elementari (cicli macchina) svolte dalla CPU in un secondo!*)

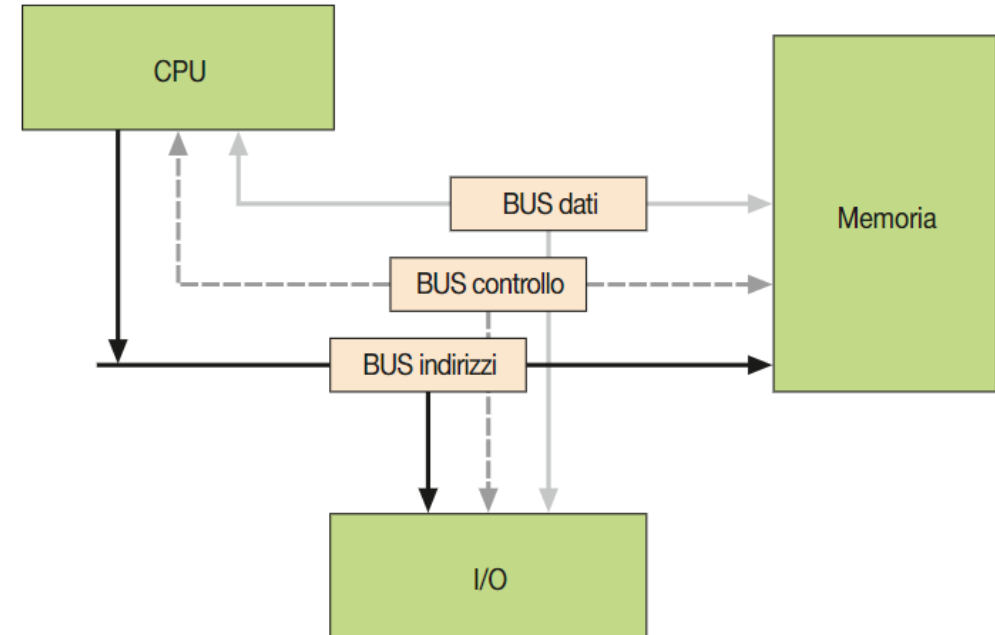
### Schema a blocchi del modello von Neumann

Il diagramma a blocchi evidenzia alcuni componenti principali:

- **CPU** (Central Processing Unit);
- **memoria centrale**;
- dispositivi o le interfacce di ingresso e uscita (input/output, **I/O**);
- i tre **BUS** che consentono di trasferire le informazioni, dove le frecce indicano il “verso” delle informazioni nella comunicazione. Le frecce bidirezionali segnalano la possibilità di scrittura e lettura da parte del dispositivo.

Non vi è una diretta comunicazione tra memoria e dispositivi di I/O; ogni trasferimento di informazioni dovrà necessariamente passare attraverso la CPU.

Dobbiamo tuttavia comprendere che lo schema a blocchi mostrato è una rappresentazione di massima e non descrive la struttura fisica del sistema.



### CPU (Central Processing Unit)

La CPU (*Central Processing Unit*) o più semplicemente il processore è l'elemento che svolge l'elaborazione dei dati, ed è rappresentata a livello fisico dal **microprocessore**.

Il processore ha il compito di preparare, elaborare ed eseguire istruzioni (è in grado di elaborare ed eseguire milioni di microistruzioni al secondo).

Tali istruzioni derivano spesso da un programma scritto in un linguaggio evoluto che è stato tradotto in un linguaggio comprensibile dalla macchina, chiamato appunto **linguaggio macchina**.

Nel linguaggio macchina viene definito il set di istruzioni (**instruction set**) fondamentali che un processore è in grado di eseguire.

In generale il **funzionamento di una CPU** può essere così schematizzato:

- la CPU estrae le istruzioni (in assembly) dalla memoria, le codifica (in linguaggio macchina) e le esegue; le istruzioni, in generale, possono comportare la manipolazione o il trasferimento dei dati;
- il trasferimento dei dati tra i vari componenti (per esempio memoria e I/O) avviene mediante i **BUS** di sistema;
- tutte le elaborazione descritte si susseguono in modo sincrono rispetto a un orologio di sistema chiamato **Clock**;
- durante ogni intervallo di tempo l'**unità di controllo**, contenuta nella CPU, stabilisce le operazioni da eseguire

**Attenzione:**

È importante saper distinguere questi due concetti:

**Frequenza del clock:**

unità di misura che identifica quanti **cicli macchina** esegue la CPU (microprocessore) in un secondo. Come già visto, il clock è un circuito temporizzatore, un oscillatore al quarzo che emette segnali a intervalli di tempo regolari generando dunque un'onda quadra caratterizzata da una certa frequenza.

Il numero di cicli macchina eseguiti in un secondo di tempo prende il nome di **Hertz (Hz)**, da cui:

<i>processore a</i>	<i>1 Hz = 1 ciclo macchina al secondo</i>
	<i>1 kHz = 1000 cicli macchina al secondo</i>
	<i>1 MHz = 1 milione di cicli macchina al secondo</i>
	<i>1 GHz = 1 miliardo di cicli macchina al secondo</i>

**MIPS** (*Million Instructions Per Second, milioni di istruzioni per secondo*):

numero di istruzioni al secondo eseguite dalla CPU. Ci indica il numero di istruzioni (in assembly) codificate ed eseguite (cioè elaborate) in un secondo. Possiamo affermare che la CPU impiega diversi cicli macchina per eseguire ciascuna istruzione in linguaggio assembly.

Siccome esistono istruzioni che impiegano tempi diversi per essere eseguite, non possiamo usare questa unità di misura per confrontare CPU diverse.

## Memoria

La memoria indicata nello schema viene anche denominata centrale o *main memory* e può essere sostanzialmente di due tipi: **RAM** (*Random Access Memory*) e **ROM** (*Read Only Memory*).

La memoria **RAM** è ad accesso casuale (o programmato), è volatile (non permanente) ed è riscrivibile. Viene usata per dati e programmi temporanei.

La memoria **ROM** è di sola lettura, è permanente e i dati in essa contenuti vengono memorizzati dal produttore oppure mediante la scrittura assai più lenta e costosa della lettura. Viene usata per memorizzare programmi non modificabili e per dati di avvio (**BIOS**).

La memoria è il dispositivo fisico che ha il compito di immagazzinare le istruzioni e i dati. È organizzata a locazioni o celle, ciascuna delle quali è in grado di memorizzare un **byte**; potremmo paragonare queste celle a tanti cassette all'interno dei quali possiamo immagazzinare un solo dato.

L'accesso alle celle di memoria avviene attraverso indirizzi numerici (*memory address*) che identificano la casella in cui si trova il dato. Tali indirizzi si chiamano indirizzi logici in quanto non fanno riferimento allo spazio fisico di memoria centrale in cui verrà caricato e allocato il dato. Il tempo necessario per leggere o scrivere un dato su una cella di memoria è dell'ordine del nanosecondo e si definisce **tempo di accesso**.

## Input/output

I dispositivi di input/output rappresentano i flussi di dati in ingresso e in uscita da e verso le principali periferiche, attraverso appositi circuiti di interfaccia che consentono di tradurre e interpretare grandezze fisiche e velocità di flusso diverse tra loro.

Possono essere distinti in dispositivi di **ingresso** e di **uscita**.

I primi consentono al sistema di elaborazione di acquisire segnali provenienti dal mondo esterno (rappresentati dai bit che vengono inviati e ricevuti da tali dispositivi; spesso rappresentano parole, numeri, suoni oppure immagini), mentre i secondi consentono al sistema di elaborazione di inviare segnali al mondo esterno.

La CPU gestisce la comunicazione con tali dispositivi in modo asincrono tramite un segnale particolare chiamato di **interrupt** (IRQ).

I dispositivi di I/O spesso sono solo delle interfacce (**controller**) con una periferica vera e propria. Il Controller è un dispositivo che si affianca a un dispositivo vero e proprio e gestisce il dialogo tra quest'ultimo e il BUS a esso collegato attraverso un **protocollo di comunicazione** rappresentato dall'insieme di regole che governano la comunicazione tra CPU e dispositivo.



## BUS

Possono essere paragonati ad autostrade sulle quali scorrono i bit.

Le dimensioni di questi elementi variano in relazione alla struttura fisica della CPU.

Nel modello Von Neumann sono rappresentate **3 tipologie di BUS**:

- il BUS dati (**data BUS**) consente la trasmissione dei dati dalla CPU agli altri elementi e viceversa (bidirezionale);
- il BUS indirizzi (**address BUS**) contiene l'indirizzo della cella di memoria o del dispositivo di I/O sul quale o dal quale la CPU ha deciso di operare (monodirezionale);
- infine il BUS di controllo (**control BUS**) trasporta gli ordini dalla CPU e restituisce i segnali di condizione dai dispositivi, per esempio quando i dati devono essere letti o scritti dai dispositivi e il dispositivo è pronto per riceverli oppure li ha effettivamente ricevuti.

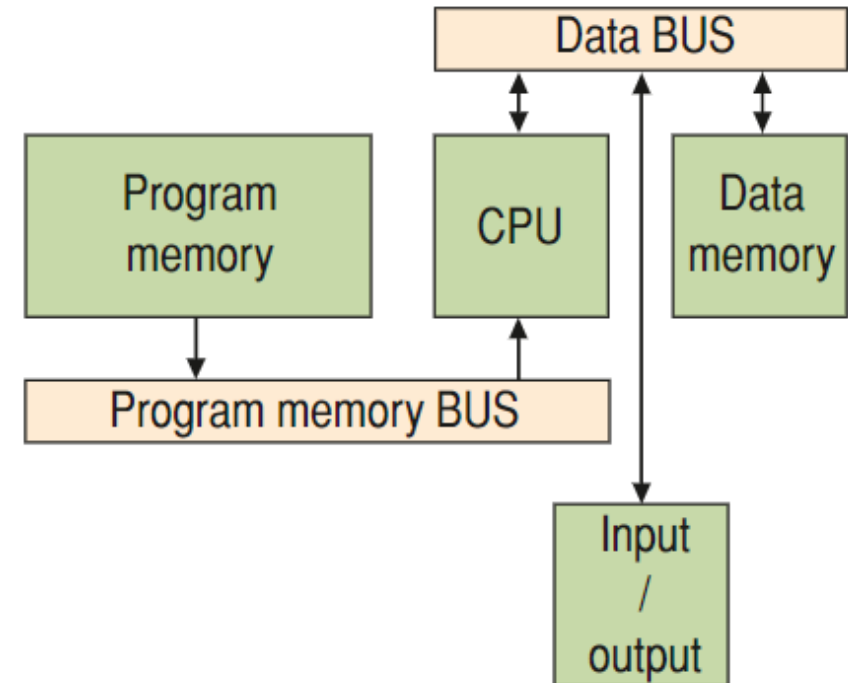
È importante sottolineare che il modello Von Neumann presentava non pochi limiti, e infatti oggi si parla di **architetture** diverse chiamate, appunto, **non Von Neumann**.

Nonostante questo, il modello è in parte valido ancora oggi come punto di partenza per comprendere, in estrema sintesi, la struttura e il funzionamento di una moderna **CPU**.

## Modello di Harvard

A differenza del modello di Von Neumann nel quale i dati e le istruzioni condividono la stessa memoria, il modello di Harvard dedica due memorie distinte per i dati e per le istruzioni. Si tratta di una soluzione più efficiente e ottimizzabile, ma tuttavia molto più costosa.

L'architettura Harvard è un modello applicato alla progettazione di processori specializzati come i **DSP** (*Digital Signal Processor*) che vengono utilizzati per il trattamento dei dati audio o video. Inoltre molti microcontrollori impiegati in applicazioni industriali utilizzano questa architettura, come i microcontrollori **PIC** (*Programmable Interface Controller*).



CPU

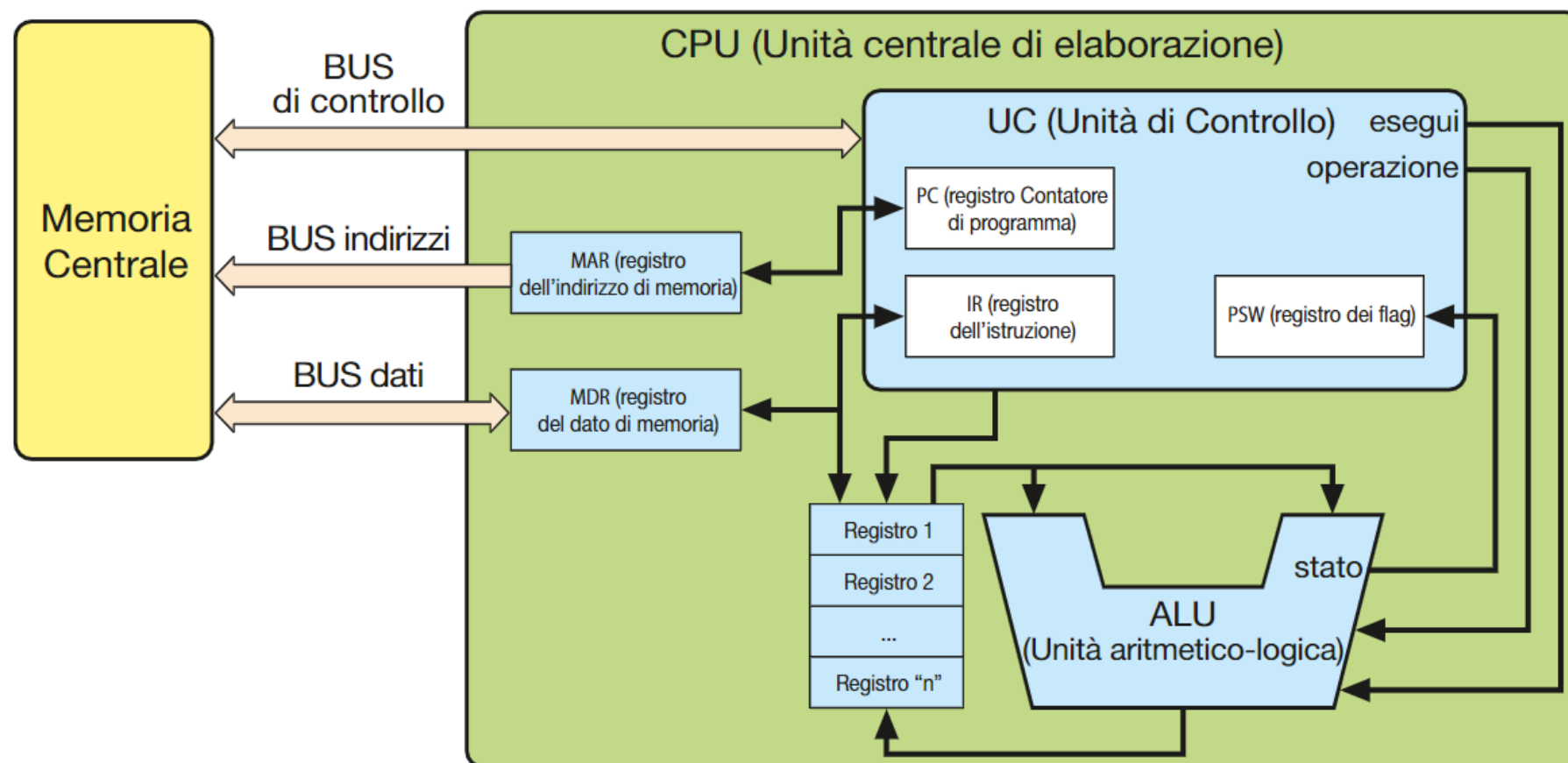
Il microprocessore (componente fisico che realizza la CPU - Central Processing Unit) svolge fondamentalmente tre funzioni:

- leggere i dati nella memoria del computer;
- elaborare le istruzioni macchina e i calcoli matematici del programma informatico caricato in memoria (elaborazione dati) eseguendo quindi calcoli aritmetici, operazioni logiche (Algebra di Boole) e salti;
- sovrintende a tutte le operazioni del sistema, organizzando i flussi di dati da e verso i dispositivi di input/output del computer (generando i segnali necessari al funzionamento dei circuiti collegati al sistema).

In base a queste funzioni, la CPU è composta dai seguenti componenti:

- **Registri / Memoria Centrale**, memorie ad accesso rapido della CPU utilizzabili dal processore per effettuare i calcoli e per il controllo dell'esecuzione di un programma;
- **Unità aritmetico logica (ALU - Arithmetic Logic Unit)**, a cui è affidata l'esecuzione dei calcoli logico-matematici;
- **Unità di controllo (CU)**, la componente della CPU che esegue e coordina l'esecuzione dei programmi informatici.

L'architettura interna di un'unità centrale di elaborazione (CPU), può essere schematizzata come nella figura che segue, secondo il modello di Von Neumann:



Nello schema, oltre alla memoria centrale (RAM), possiamo individuare i seguenti elementi funzionali della CPU:

- unità di controllo (*CU, Control Unit*);
- registro PC (*Program Counter*);
- registro IR (*Instruction Register*);
- registro PSW (*Process Status Word*);
- ALU (*Arithmetic Logic Unit*);
- registri generali;
- registro MAR (*Memory Address Register*);
- registro MDR (*Memory Data Register*);
- BUS di controllo;
- BUS indirizzi;
- BUS dati

**BUS**

insieme di linee, ciascuna delle quali in grado di trasmettere un bit di informazione tra due elementi elettronici. Il segnale che viene trasferito è di tipo logico, secondo la codifica binaria di un'informazione.

**Registro**

una particolare cella di memoria contenuta all'interno della CPU. La dimensione dei registri si esprime in bit e dipende dall'architettura specifica della CPU. In generale la funzione dei registri è quella di memorizzare temporaneamente dei dati.

## BUS interno

collega tutti gli elementi che fanno parte della CPU; nello schema sono individuabili dal colore nero delle frecce. Si tratta di un BUS generalmente di controllo, senza distinzione tra dati e indirizzi e non è da confondere con i BUS esterni o di sistema che verranno illustrati in seguito.

## Unità di controllo (CU)

sovrintende al funzionamento del processore e ha il compito di attivare tutte le azioni necessarie per l'esecuzione delle istruzioni.

È il blocco che invia i comandi esecutivi all'ALU in base alla decodifica dell'istruzione, e decide l'incremento dell'indirizzo di memoria contenuto nel registro PC in modo da predisporre all'esecuzione dell'istruzione.

## Registri interni

Un registro è paragonabile a una lavagna sulla quale viene scritta un'informazione per un breve periodo di tempo. Per memorizzare un'informazione per un periodo più lungo può essere usata una cella di memoria, paragonabile a un quaderno o a un blocco note.

Diversamente dalle celle di memoria i registri non possiedono un indirizzo ma un nome specifico.

È importante fare una distinzione tra i registri accessibili dal programmatore (**registri generali**) e altri che invece risultano inaccessibili al programmatore, in quanto vengono usati direttamente dalla CPU per le proprie operazioni di controllo (**registri di stato e di controllo**).

## Registri di stato e di controllo

Sono registri interni **non** accessibili dal programmatore, usati dal processore per controllare il proprio funzionamento e dal sistema operativo per il controllo dei programmi in esecuzione.

Sono usati dall'hardware o manipolabili con speciali istruzioni macchina eseguibili in modalità riservata dal SO.

### **MDR** (Memory Data Register)

È un registro interno collegato direttamente al BUS dati attraverso un **buffer bidirezionale tri-state**.

Il registro non è visibile al programmatore e contiene i **dati** che la CPU vuole inviare o ricevere dalla **memoria** o dai dispositivi di **I/O**.

Possiamo immaginare il registro MDR come una sorta di memoria di transito dove vengono immagazzinati temporaneamente tutti i dati scambiati con la memoria, prima di essere smistati presso gli altri registri interni.

### **MAR** (Memory Address Register)

È un registro interno collegato direttamente al **BUS indirizzi**. Non è visibile al programmatore e contiene gli **indirizzi** necessari alla selezione della cella di memoria oppure al dispositivo di I/O coinvolto nell'operazione.

Durante la fase di fetch di un'istruzione il MAR contiene l'indirizzo della locazione di memoria in cui si trova l'istruzione che deve essere codificata, mentre durante la fase di execute, se si tratta di un'istruzione che fa riferimento alla memoria, contiene l'indirizzo dell'operando che deve essere letto dalla RAM.

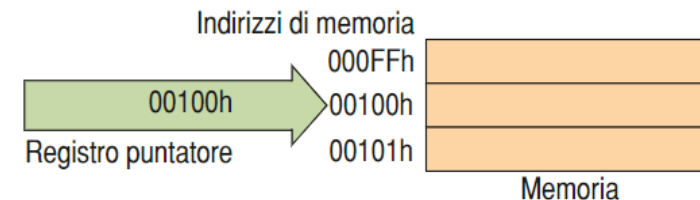


### IR (Instruction Register)

È il registro interno che riceve il **codice operativo** dell'istruzione prelevata durante la fase di fetch. È invisibile al programmatore e contiene temporaneamente il codice operativo dell'istruzione durante la sua codifica.

### PC (Program Counter)

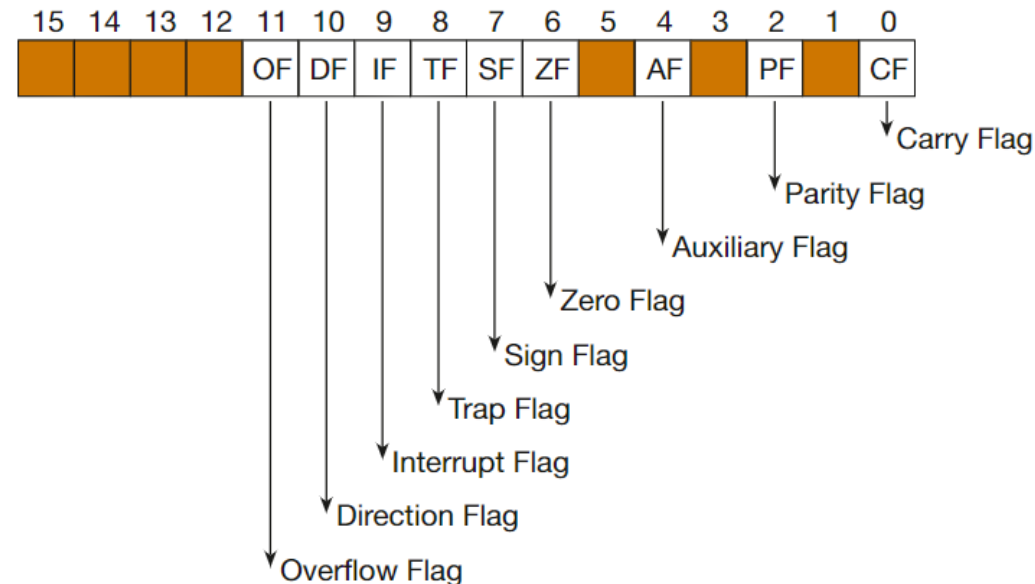
È un registro interno **accessibile parzialmente dal programmatore** che lo può usare per modificare il flusso sequenziale del programma. Il registro **PC** (nelle CPU Intel si chiama **IP**, *Instruction Pointer*) contiene, in ogni fase di avanzamento del programma, l'**indirizzo di memoria in cui si trova l'istruzione successiva da eseguire**. Una modifica del contenuto di questo registro provoca infatti un "salto" all'indirizzo indicato. È un registro di tipo **puntatore** nel senso che il dato contenuto in esso si riferisce, quindi punta, a un dato che si trova in memoria all'indirizzo corrispondente al valore contenuto nel registro stesso. La figura sotto illustra il concetto enunciato. Normalmente il contenuto di questo registro viene incrementato automaticamente dalla CU ogni volta che l'esecuzione di un'istruzione è completata, in modo da potere leggere in memoria l'istruzione successiva.



**PSW** (Program Status Word – parola di stato del programma)

Questo registro interno, chiamato anche registro dei **flag** (in italiano bandiere), non ha un significato nel suo complesso; ciascun bit che lo compone si comporta come una bandierina di segnalazione che fornisce informazioni sul risultato delle operazioni aritmetico-logiche dell'ultima istruzione eseguita (**codici di condizione**).

L'ALU, a ogni operazione aritmetico-logica, aggiorna il contenuto del registro PSW. Le informazioni contenute in questo registro sono essenziali per la costruzione degli algoritmi dei programmi. Grazie alle flag possiamo infatti realizzare le strutture fondamentali della programmazione (condizione, iterazione, sottoprogrammi): valutando il contenuto dei bit del registro PSW, il codice di un programma può alterare il flusso sequenziale dello stesso per realizzare i costrutti di scelta e iterazione e le chiamate ai sottoprogrammi.



I flag principali del PSW sono:

- **ZERO (ZF):** contiene 1 (true) quando il risultato dell'ultima operazione è uguale a 0, mentre contiene 0 (false) quando il risultato dell'operazione è diverso da 0. Nei linguaggi evoluti di programmazione la condizione:

se  $(a = b)$  diventa a livello di linguaggio macchina:  $a - b$

Se il risultato della sottrazione è zero significa che i due elementi (a e b) sono uguali, pertanto ZF viene attivato (1).

- **CARRY (CF):** contiene true quando si verifica un riporto dal bit più significativo del risultato durante le quattro operazioni aritmetiche basilari (somma, sottrazione, moltiplicazione, divisione). Se la somma del bit più significativo determina un riporto questo bit non viene sommato al bit successivo ma viene memorizzato nel CF per segnalare che l'operazione nel suo complesso ha determinato un riporto. Viene utilizzato per due diverse ragioni: verificare se un elemento è maggiore o minore di un altro, oppure verificare se un numero non può essere contenuto in un registro.
- **OVERFLOW (OF):** contiene true quando il segno del risultato dell'ultima operazione aritmetico-logica discorda dal segno degli operandi, cioè quando per esempio il prodotto di due numeri negativi dà luogo a un nuovo valore negativo. Segnala inoltre la presenza di un overflow rispetto alla precisione quando il risultato ha un'ampiezza superiore alla capacità del risultato.
- **PARITY (PF):** contiene true quando nel risultato dell'ultima operazione aritmetico-logica il numero di bit a 1 del risultato è pari. Viene usato principalmente negli algoritmi di comunicazione per la gestione del rilevamento degli errori di comunicazione.
- **SIGN (SF):** contiene true quando il risultato dell'ultima operazione aritmetico-logica è negativo, copiando nel flag SF il bit più significativo del risultato che riflette il segno secondo la codifica in complemento a 2.

## Registri generali

I registri generali sono registri interni non specializzati **visibili** al programmatore, destinati a ospitare temporaneamente i dati in corso di elaborazione o indirizzi di memoria e sono usati per contenere gli operandi e risultati parziali o il calcolo di indirizzi.

I dati contenuti nei registri possono provenire dalla memoria o da altri registri, vanno all'ALU per l'elaborazione e da qui tornano ai registri dai quali vengono riportati in memoria.

### Registri utente (R1, R2, ... Rn)

Sono registri a disposizione del programmatore per trasferirvi i dati dalla memoria centrale (RAM) e per memorizzare i risultati intermedi delle elaborazioni. Il numero e le caratteristiche dei registri utente varia da processore a processore e, nel caso di processori con un solo registro, questo viene chiamato **accumulatore**.

Il programmatore che ha visibilità dei registri utente è solo il programmatore che scrive programmi in linguaggio macchina o in linguaggio assembler. Nel caso di un programma scritto in un linguaggio ad alto livello è invece il compilatore che ha la visibilità dei registri utente.

### SP (Stack Pointer)

Registro che esiste solo nei processori che gestiscono lo **stack**, ovvero una zona di memoria usata in modalità **pila** (*stack*) ovvero LIFO (Last In First Out), per gestire procedure attive,...

Questo registro viene modificato dalle speciali istruzioni *Push* e *Pop* con le quali è possibile inserire o prelevare dati dallo stack.

## ALU (Arithmetic Logic Unit)

È la parte della CPU dedicata alle operazioni aritmetiche e logiche che esegue le trasformazioni sui dati. Poiché tutte le istruzioni e i dati sui quali opera la CPU sono codificate in forma numerica (binaria), in realtà tutte le elaborazioni devono passare attraverso l'ALU, in quanto qualsiasi operazione può essere ricondotta a un'operazione aritmetico-logica.

Per esempio, se vogliamo confrontare due caratteri alfabetici tra loro, siccome vengono codificati secondo la tabella ASCII in ordine progressivo (A = 41h, B = 42h, C = 43h ... a = 61h, b = 62h, c = 63h...), l'ALU effettua una semplice operazione di sottrazione tra i due valori: il settaggio del Flag di Carry indicherà se il primo carattere è maggiore o minore del secondo.

La ALU interviene su due operandi che possono essere contenuti nei registri interni o provenire dalla memoria attraverso l'MDR e produce un risultato che può essere salvato su un registro interno o in memoria attraverso l'MDR.

Qualunque operazione logica può essere realizzata disponendo di un insieme di operatori logici, quali per esempio AND, OR, XOR, NOT. Ad esempio:

- **AND** può essere usata per rimuovere bit (es. 01010111 AND 00001111 rimuove i 4 bit più alti)
- **OR** viene usata per unire i bit (es. 0110 OR 0001 = 0111)
- **XOR** viene usata per selezionare i bit che hanno avuto una variazione (es. 0101 XOR 0111 = 0010)
- **NOT** viene usata per capovolgere i bit, ovvero per fare il complemento a 1

Lo stesso concetto può essere esteso anche all'aritmetica individuando come insieme minimo la coppia addizione + negazione.

Ogni altra operazione aritmetica deriva da questa coppia di operatori:

- la **sottrazione** si ottiene in sostanza con la negazione del sottraendo addizionato al minuendo, quindi possiamo dire che  $a - b$  equivale ad  $a + (-b)$ .
- controllando il bit di segno del risultato di una sottrazione si può implementare anche l'operazione di **confronto**;
- la **moltiplicazione** è invece un ciclo di addizioni del moltiplicando eseguite per un numero di volte pari al moltiplicatore;
- la **divisione** è una ripetizione di sottrazioni del divisore e così via;

Per costruire una semplice ALU a 4bit, sarà sufficiente prendere il circuito sommatore/sottrattore mostrato in slide 36 e aggiungere le porte AND, OR, NOT, XOR collegando questi componenti in ingresso a un multiplexer che permetta di volta in volta di scegliere l'operazione da effettuare.

Il **comportamento** dell'unità aritmetico-logica può essere chiamato **binario ad accumulo**.

Indica che i dati sui quali opera sono espressi in codice binario e il risultato delle operazioni eseguite viene sempre accumulato in uno dei due operandi (lo vedremo scrivendo in codice Assembly).

Supponiamo per esempio di voler sommare il contenuto di due ipotetici registri generali R1 e R2. Se R1 contiene 5 e R2 contiene 10 effettuando la somma  $R1 + R2$  ( $5 + 10$ ), il registro posto a sinistra nell'addizione conterrà il risultato che andrà a sostituire il precedente valore in esso contenuto. Possiamo così sintetizzare il concetto enunciato:

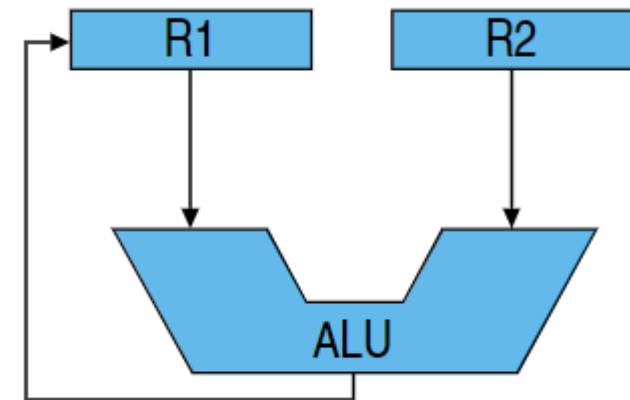
ADD R1, R2

equivale a:

$R1 = R1 + R2$

In tal modo il registro R1 si comporta da **accumulatore**; operando in questo modo l'ALU risparmia trasferimenti di dati tra registri ottenendo prestazioni superiori.

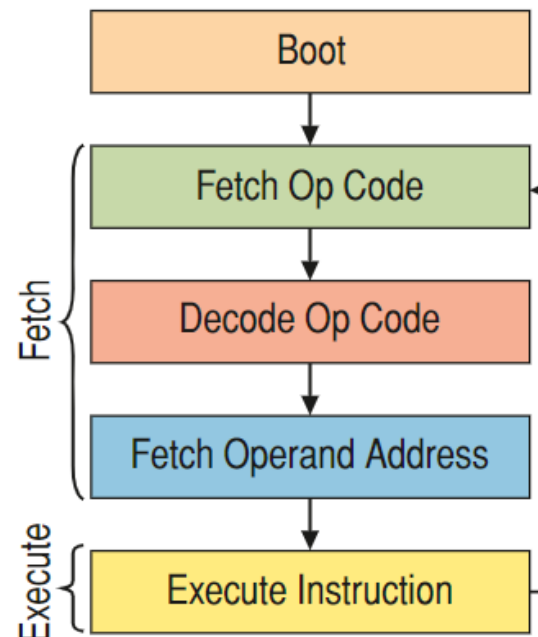
La figura a lato mostra il concetto di accumulo riportato nell'esempio:



## Il ciclo macchina

Il funzionamento di una CPU inizia con il **prelevamento** dalla memoria del codice macchina dell'istruzione da eseguire; tale operazione viene eseguita dalla Control Unit. L'istruzione prelevata viene trasferita in un registro specifico e quindi **codificata**. Dopo aver codificato, cioè tradotto l'istruzione, la CPU emette i segnali necessari **all'esecuzione** dell'istruzione. Il procedimento appena descritto, attraverso il quale la CPU esegue un'istruzione, prende il nome di **ciclo macchina**, che può essere idealmente suddiviso in quattro parti:

1. fase di Fetch dell'istruzione;
2. fase di Decode dell'istruzione;
3. fase di Fetch degli operandi;
4. fase di Execute.





**Fetch (prelevamento) dell'istruzione**

fase in cui la CPU deve reperire l'istruzione da eseguire. In questa fase la CPU deve dialogare con la memoria RAM per ottenere il codice macchina dell'istruzione da eseguire.

Il prelevamento avviene in questo modo:

- l'indirizzo di memoria della prima istruzione di un programma è memorizzato nel registro **PC** (Program Counter). La Control Unit (**CU**) legge dunque il contenuto del PC e lo trasferisce nel registro **MAR**;
- il contenuto del **MAR** viene impostato dalla CPU sul **BUS indirizzi**;
- la CPU ordina sul **BUS di controllo**, tramite la linea di richiesta di lettura dato (segnale **leggi**), il dato (l'istruzione in questo caso) situato all'indirizzo di memoria RAM richiesto;
- la memoria preleva il dato richiesto e lo presenta sul **BUS dati** a tutti i dispositivi connessi a questo BUS. Il dato viene quindi ricopiato nel registro **MDR**;
- la CPU preleva il dato dall'**MDR** e lo sposta nel registro interno **IR** (Instruction Register) per la decodifica;
- il **BUS di controllo** e il **BUS indirizzi** vengono rilasciati (ovvero lasciati liberi di essere richiesti da altri dispositivi);
- la CU incrementa il contenuto del registro **PC** in modo che esso "punti" all'istruzione successiva.

Le istruzioni sono, automaticamente, eseguite in sequenza; la prossima istruzione da eseguire è quindi normalmente quella posta nella parola di memoria successiva all'istruzione in esecuzione. Il contenuto del PC può però essere modificato dall'esecuzione dell'**istruzioni di salto**, quando è necessario trasferire il controllo da altre posizioni del programma.

Risulta chiaro che per **mandare in esecuzione un programma**, il sistema operativo deve:

- 1) Collocare il programma da eseguire in memoria RAM a partire da una certa locazione;
- 2) Inserire nel PC il valore della prima istruzione del programma da eseguire. Questo indirizzo si chiama punto d'ingresso (Entry Point) del programma.

### Decode dell'istruzione

fase interna alla CPU durante la quale avviene l'interpretazione dell'istruzione e la preparazione dei dispositivi necessari. In questa fase infatti, il codice macchina dell'istruzione viene codificato dalla CU in operazioni da eseguire da parte della CPU. L'interpretazione può avvenire attraverso due modalità:

- 1) circuiti logici già predisposti al momento di costruzione del processore secondo la logica cablata dal circuito: le funzioni eseguibili sono prefissate fisicamente e il sistema comporta una certa rigidità;
- 2) attraverso microistruzioni contenute in un'apposita parte della ROM secondo la logica microprogrammata. In questo caso, l'interpretazione avviene cercando nella ROM la sequenza di passi elementari di cui è composta l'istruzione da interpretare.

La CU deve riconoscere il codice dell'istruzione, scomporla nelle diverse parti e poi eseguirla (execute dell'istruzione).

### Fetch degli operandi

In base alla codifica dell'istruzione il processore riconosce se è necessario o meno prelevare **dalla memoria o da un registro interno** un altro dato per completare l'esecuzione dell'istruzione. In tal caso viene eseguita un'operazione di lettura, dalla memoria o da un registro, chiamata appunto Fetch degli operandi.

### Execute dell'istruzione

Nella fase di execute la Control Unit (CU) invia segnali che rappresentano opportuni comandi per l'esecuzione.

L'esecuzione delle istruzioni coinvolge l'unità aritmetico logica (ALU) e i registri R1, R2, ... Rn.

Le operazioni eseguite dall'ALU agiscono sui dati contenuti nei **registri utente** R1, R2, ... Rn e in memoria centrale; il risultato delle operazioni è depositato nei registri o in una cella di memoria.

## Esempio di ciclo macchina

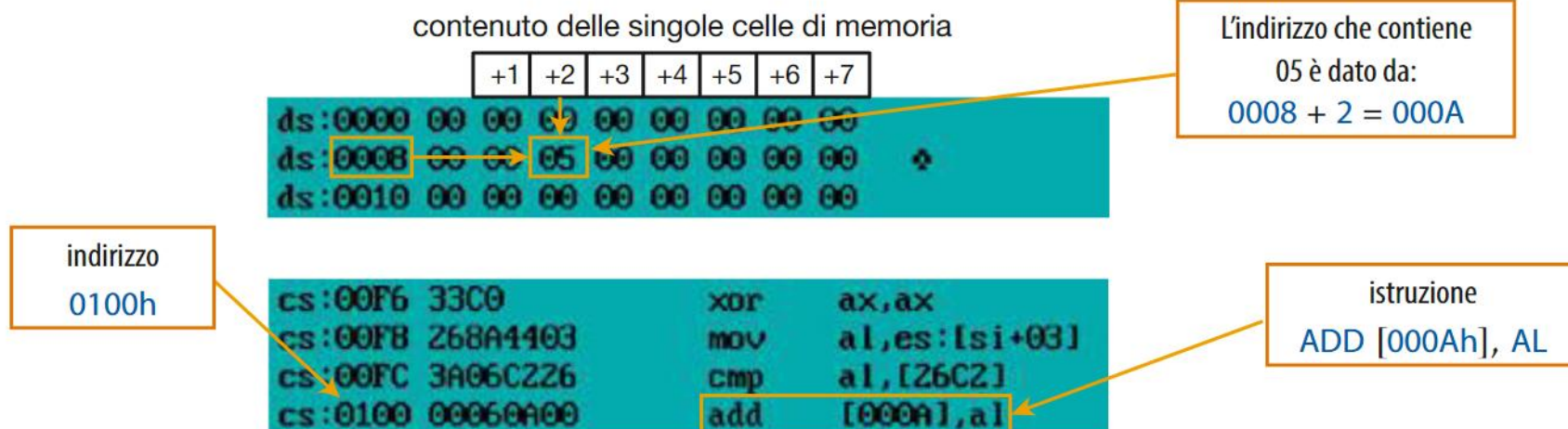
Somma tra il contenuto della cella di memoria di indirizzo `[000Ah]` e il contenuto del registro `AL`.  
Il processore deve eseguire la seguente istruzione assembly:

`ADD [000Ah],AL`

Usiamo una rappresentazione **Memory dump** della memoria, che mostra 8 celle per ciascuna riga. La memory dump identifica lo stato in cui si trovano le celle della memoria in un determinato momento. In alcuni casi viene anche mostrato il corrispettivo ASCII del contenuto della singola cella di memoria.

In questo caso, nella cella di indirizzo 000Ah (cioè due celle più a destra rispetto all'indirizzo base 0008h), è memorizzato il valore 05h:

la lettera "h" posta dopo la cifra meno significativa indica, in linguaggio assembly, un numero espresso in base di numerazione esadecimale



### Fetch

Viene prelevata l'istruzione presente all'indirizzo contenuto nel registro **PC** (ad esempio: **0100h**). Si tratta di una sequenza di 6 byte che rappresentano il codice macchina dell'istruzione: **00 06 0A 00 00 00**. Il codice dell'istruzione viene inserito nel registro **IR** (Instruction Register).

### Decode

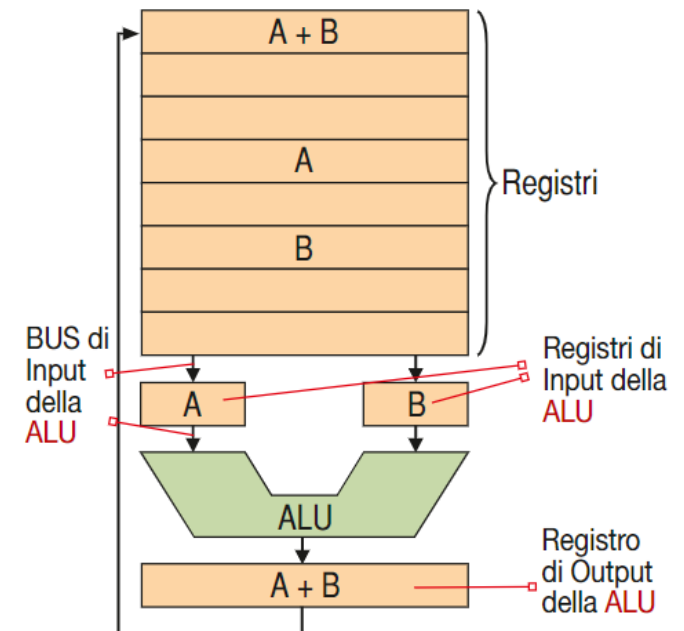
Viene decodificata l'istruzione il cui codice macchina è presente nel registro IR.

### Fetch degli operandi

Viene prelevato il contenuto della cella **000Ah**, cioè **05h**.

### Execute

Il valore recuperato dal passo precedente (**05h**) viene sommato con il contenuto del registro **AL** (poniamo che contenga **0Fh**). Il risultato viene posto nuovamente nella cella di indirizzo **000A** che conterrà **14h** al termine dell'esecuzione dell'istruzione. L'operazione richiede sicuramente più cicli di **data path**, soprattutto per la necessità di recuperare operandi dalla memoria. Nella figura a lato è rappresentato il "tipico" data path per una CPU. Il data path è una sezione della CPU che raggruppa l'ALU e i registri. Il passaggio di due operandi attraverso l'ALU e la memorizzazione del risultato in un nuovo registro viene detto **ciclo di data path**. Ogni istruzione viene eseguita in uno o più cicli data path, per esempio una divisione necessita di molti cicli per la sua esecuzione. La velocità con cui viene compiuto un ciclo di data path contribuisce significativamente a determinare la velocità della CPU.



Lo schema di ciclo macchina appena visto era valido per le CPU degli anni '80; attualmente, grazie all'incremento delle prestazioni dei chip integrati, si sono diffuse nuove e più complesse tecniche denominate generalmente con la dicitura **architetture non Von Neumann**. Tuttavia il principio di funzionamento resta in linea di massima ancora valido.

Le due tecnologie di riferimento per la costruzione di microprocessori sono:

- **CISC** (Complex Instruction Set Computer)

I processori costruiti secondo l'architettura CISC sono nati con la necessità di avere un **numero elevato di istruzioni diverse**, di tipo anche complesso, per **semplificare il compito dei programmatori** e per disporre di **programmi più compatti che utilizzino minore memoria**. All'interno delle CPU realizzate secondo tale architettura è presente una memoria di tipo ROM che contiene una serie di microcodici (set di operatori elementari che descrive il comportamento di una determinata istruzione assembly) ciascuno dei quali permette di eseguire all'interno del microprocessore stesso, un'azione elementare. Per eseguire le istruzioni è necessario prima di tutto trasformarle in una serie di istruzioni scritte in microcodice; in tal modo quelle più semplici richiederanno meno istruzioni in microcodice.

I processori CISC più conosciuti sono la famiglia di CPU della Intel: 80286, 80386, 80486, Pentium, Celeron.

- **RISC** (Reduced Instruction Set Computer)

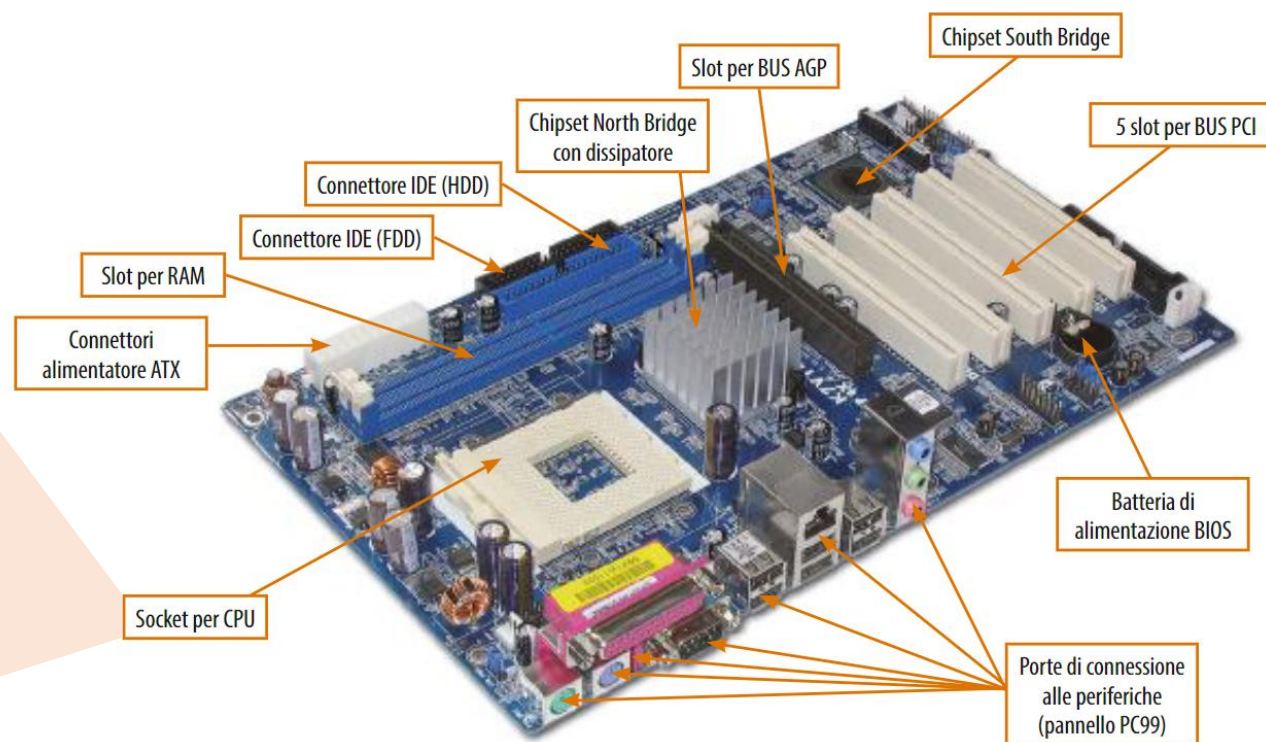
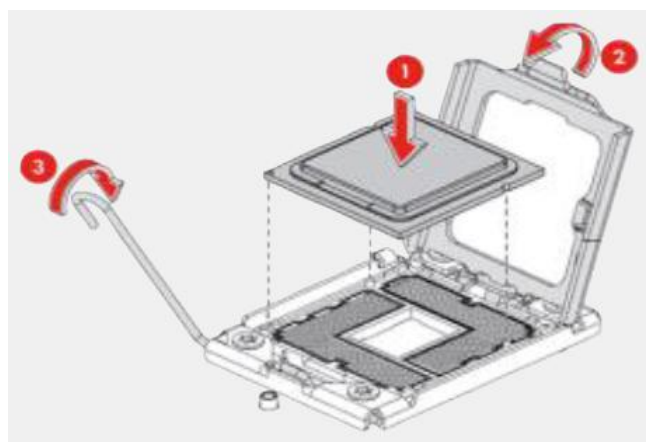
Il concetto costruttivo di un microprocessore RISC è invece la forte **riduzione del numero di istruzioni** in modo da poter conciliare la velocità del microprocessore con l'esecuzione di queste. Il fine principale della struttura RISC è quello di **produrre processori ad alta velocità e dal costo ridotto**, data la minore complessità del progetto. Lo svantaggio della tecnologia RISC è il fatto che per essi sono stati sviluppati sistemi operativi a minore diffusione rispetto a quelli sviluppati per i CISC, come Windows. Inoltre, una conseguenza dell'architettura RISC è la **maggiore complessità dei programmi**: se i processori riconoscono una quantità molto bassa di istruzioni, il programmatore deve sopperire con il software per far svolgere a essi operazioni complesse. In questo caso diventa praticamente obbligatorio studiare, di ogni porzione di codice, il metodo per renderla più veloce; l'ottimizzazione del codice diviene di primaria importanza nello sviluppo dei sistemi di tipo RISC.

Attualmente si stanno ormai diffondendo tecnologie ibride, denominate CISC/RISC, come quella dell'architettura Nehalem (IntelCore i).



Il microprocessore (che, come già detto, è il componente fisico che realizza la CPU) si trova sulla scheda madre (**motherboard**), quel componente che consente di mettere in comunicazione tra loro i diversi componenti e questi ultimi con la CPU.

La CPU può essere montata su un apposito alloggiamento di tipo Socket **ZIF** (Zero Insertion Force) ovvero un alloggiamento particolare nel quale non è necessario usare la propria forza per collocare la CPU in sede. Per facilitare il suo inserimento viene infatti usata una piccola leva che, una volta sollevata, permette l'inserimento del processore senza alcuna pressione e, una volta riabbassata, mantiene il processore sul suo supporto.



Nel **1971** la Intel commercializzò il **primo microprocessore**, il **4004**, realizzato dal fisico italiano (vicentino) **Federico Faggin**, che elaborava a una frequenza di 740 kHz. Faggin fondò poi ZiLOG, presso cui venne progettato lo Z80 (di seconda generazione).

La **prima generazione** di microprocessori fu quella commercializzata tra il 1971 e il 1973. Fra questi va citato l'8080, il primo processore a 8 bit.

A partire dal 1973 comparve la **seconda generazione** di microprocessori, prodotti con le più avanzate tecnologie NMOS (MOS di tipo n). Tra essi i più noti sono lo Z80 di Zilog, i 6800 e 6809 di Motorola e l'alternativa Intel costituita dall'8085. L'inserimento dei microprocessori nei personal computer ha fatto sì che tali circuiti fossero prodotti in grandi quantità, rendendoli più economici.

Nel 1978 sono apparsi i microprocessori a 16 bit, che costituiscono la **terza generazione** di questo tipo di circuiti. Sul mercato vennero commercializzati l'8086 di Intel con un rendimento dieci volte superiore all'8085, lo Z8000 Zilog e l'MC68000 Motorola. La diffusione dei personal computer si ebbe quando IBM scelse l'8086 per i suoi PC. Il successo di vendita dei PC portò allo sviluppo di software con molteplici applicazioni in molti campi. Questo fu talmente importante da determinare il fatto che uno dei principali obiettivi dei microprocessori sviluppati successivamente da Intel fosse quello di essere compatibili a livello di software, in modo da poter eseguire qualsiasi programma realizzato per funzionare con l'8086.



Nel 1985, con la comparsa degli MC68020 e MC68030 Motorola e degli 80386 e 80486 Intel, si cominciò a parlare di **quarta generazione** di microprocessori. Tali circuiti, realizzati con tecnologie CMOS (MOS complementare) consentono di lavorare a frequenze superiori a 50 MHz, con un consumo di energia molto ridotto.

La **quinta generazione** apparve nel 1993 con il processore Intel Pentium, con architetture che consentivano di raggiungere 300 MHz di frequenza.

Due anni più tardi con l'avvento del Pentium Pro, Pentium II e Pentium III si raggiunsero velocità ancora superiori fino a circa 1 GHz: era la **sesta generazione** di microprocessori.

La **settima generazione** arrivò nel 2000: con l'avvento di un nuovo sistema operativo Windows (Windows 2000) si affiancò una nuova microarchitettura denominata *netburst* implementata sul Pentium IV che raggiungeva i 3,75 GHz di frequenza di lavoro.

L'**ottava generazione** fu quella dei processori della serie Intel Itanium, adatti a workstation e server molto costosi, con una microarchitettura di tipo CISC.

La **nona generazione** si diffuse nel 2006 con i processori della serie Core: si tratta di **processori multicore**, con frequenze che si aggirano attorno ai 3 GHz, ma con prestazioni superiori date dall'elaborazione parallela.

La **decima generazione** è quella dei processori di architettura Intel Nehalem (Core i7 extreme), dotati di più core (processori) che consentono l'esecuzione simultanea di più istruzioni.

La tabella seguente illustra i principali processori rispetto all'anno di riferimento:

GENERAZIONE	PROCESSORE	MICRO- ARCHITETTURA	BUS DATI	BUS INDIRIZZI	FREQUENZA DI CLOCK	ANNO
Prima	8086	-	16	20	10 Mhz	1980
Seconda	80286	-	16	24	20 Mhz	1982
Terza	80386	i386	32	32	50 Mhz	1985
Quarta	80486	i486	32	32	100 Mhz	1989
Quinta	Pentium	P5	64	32	300 Mhz	1993
Sesta	Pentium Pro Pentium II Pentium III	P6	64	36	1 Ghz	1995
Settima	Pentium IV	Netburst	64	36	3,72 Ghz	2000
Ottava	Itanium Itanium-2	Itanium	64	64	1,66 Ghz	2002
Nona	CPU Core 2	Core (Penryn)	64	64	3 Ghz	2006
Decima	CPU Core i7 extreme	Nehalem	64	64	3,2 Ghz	2010

Il **Core** rappresenta il nucleo o nocciolo della CPU, cioè il vero e proprio “nucleo elaborativo”. Tutto il resto della CPU è rappresentato dal package, o guscio, che lo contiene.

Il nucleo è collegato elettricamente ai pin di contatto che andranno a innestarsi nel socket. Normalmente le dimensioni di tale componente a confronto con tutto il package, sono molto ridotte.

Dall'anno 2005 circa, i produttori di processori (Intel e Amd) si sono resi conto che non era più possibile innalzare le frequenze operative dei propri processori, e hanno deciso di puntare tutto sulle architetture che presentano più nuclei nello stesso package, chiamate **MultiCore Architectures**. Sfruttando il parallelismo, cioè le elaborazioni dell'istruzione da entrambi i core, si ottengono risultati analoghi rispetto all'utilizzo di più processori fisicamente separati posti sulla stessa scheda madre.

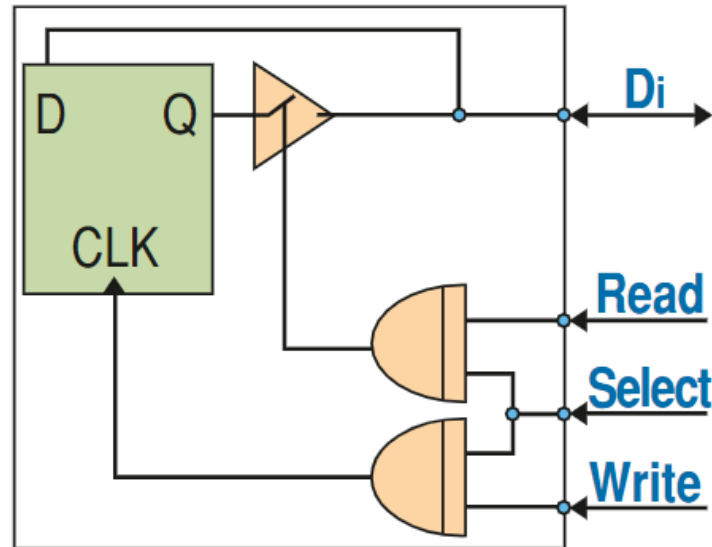
Nel 2007 debuttano i primi sistemi multi core, a 4 core (QuadCore) e dal 2010 i nuclei diventano 8 con la tecnologia Intel i7. Se confrontiamo per esempio la CPU Celeron 430 (1,8 GHz) con l'Intel Pentium 4 (3,00 GHz), appare semplice definire il processore più veloce: il secondo ha una frequenza di clock maggiore, pertanto risulta più veloce. Se invece confrontiamo la CPU Pentium 4 (3,00 GHz) con il Core 2 Duo E6600 (2,4 GHz), ne deduciamo che il secondo ha prestazioni migliori in quanto, nonostante abbia una frequenza di clock inferiore, possiede un doppio nucleo (Dual Core); pertanto sarà in grado di risolvere due calcoli per volta, a differenza del Pentium 4, grazie al parallelismo della sua architettura.

È tuttavia bene ricordare che **le frequenze di clock dei diversi Core della stessa CPU non possono essere sommate**. Infatti la CPU E6600 non elabora a 4,8 GHz, ma sono i due Core che elaborano ciascuno a 2,4 GHz.

Memorie

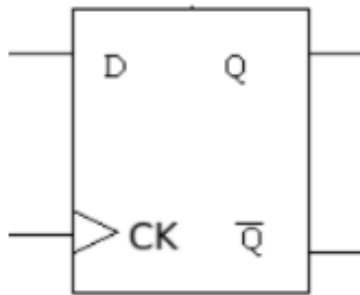
La memoria contiene informazioni espresse in binario, cioè formate da bit che possono essere 0 oppure 1.

Esistono diverse tecnologie per la realizzazione dei circuiti di memoria, ma il modello a **flip-flop** ci consente di comprenderne in modo immediato il significato. Lo schema sotto descrive il funzionamento di un singolo elemento di memoria, dal punto di vista elettronico, in grado di immagazzinare un solo bit.

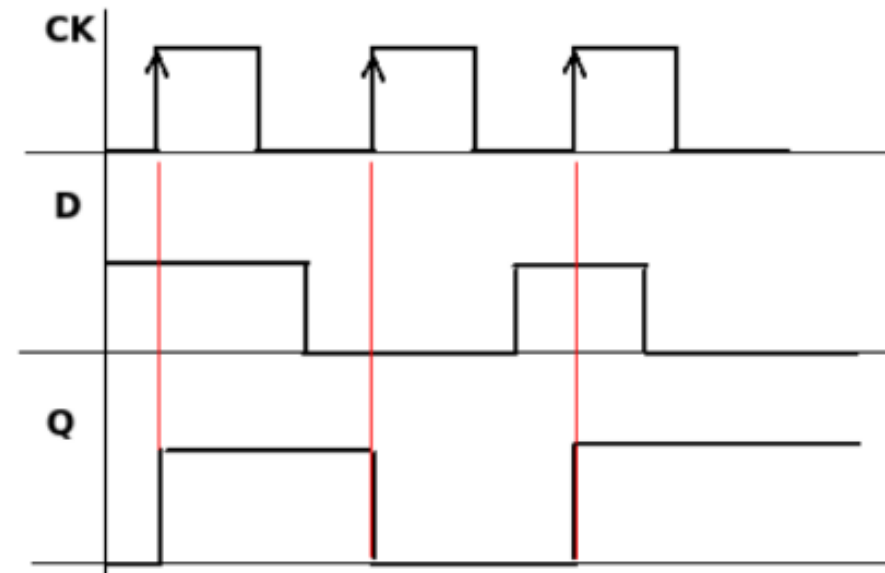


L'elemento di memoria vero e proprio, cioè quello che memorizza il bit, è formato da un flip-flop di tipo D (elemento in verde nell'immagine precedente). Il termine D deriva dall'inglese *delay* (che significa ritardo). Ha un ingresso per il dato **D** (*data*), un ingresso di sincronizzazione **CLK** (*clock*) e un'uscita **Q** (*quit*).

Quando il clock (ingresso CLK) ha il fronte di salita, il flip-flop aggiorna lo stato Q, trasferendo lo stato dell'ingresso D in uscita Q. In tutti gli altri casi il flip-flop conserva lo stato logico corrente (mantiene inalterata l'uscita Q) indipendentemente dallo stato D, da sue eventuali variazioni. In pratica Q cambia soltanto quando il clock sale a 1. Non a caso D significa *delay* ossia ritardo.



CK	D	Q
0	X	$Q_0$
1	X	$Q_0$
↑	0	0
↑	1	1



Nella cella di memoria di slide 77 vi sono tre ingressi di controllo denominati: **Read**, **Select** e **Write**.

Il segnale **Di** è invece il dato che può essere memorizzato oppure letto dalla memoria.

Quando il segnale sul data BUS **Di** passa da uno stato a un altro, cioè commuta, la memoria del flip-flop (come già visto) continua a conservare il suo valore.

La memorizzazione (**scrittura**) avviene quando vengono attivati (cioè posti a 1) gli ingressi **Write** e **Select**.

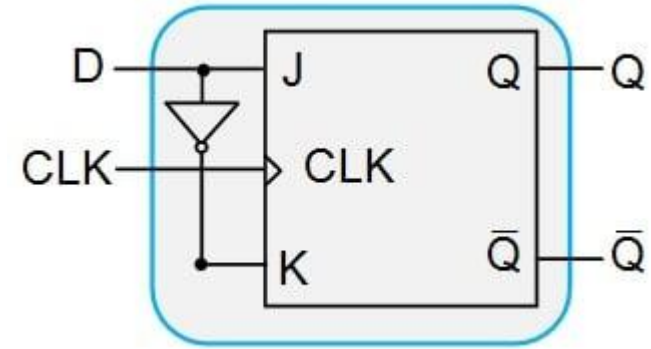
I segnali **Read** e **Write** provengono dal **control BUS**, mentre il segnale di selezione **Select** proviene dall'**address decoder**, un circuito di decodifica degli indirizzi.

La **lettura** avviene quando vengono attivati (cioè posti a 1) gli ingressi **Read** e **Select**. La logica costruttiva del BUS di controllo fa sì che i segnali Read e Write non possono mai essere contemporaneamente attivi.

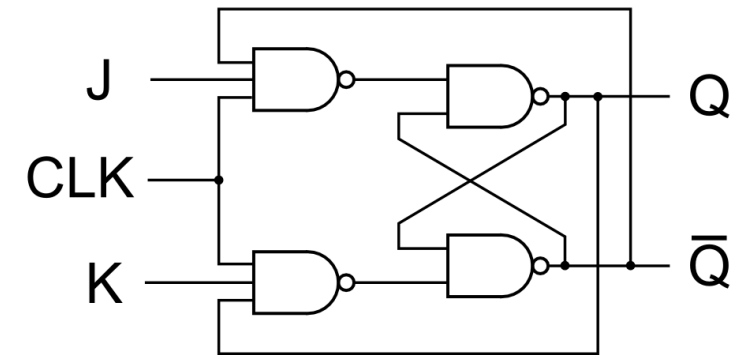
Le operazioni di accesso alle celle di memoria coinvolgono sempre blocchi di byte, quindi dobbiamo immaginare il circuito precedente come parte di altri sette elementi atti a formare un byte. Infatti il singolo bit non è accessibile singolarmente ma solo come parte di un blocco di elementi che vengono processati sempre insieme. Ciascun bit di una cella di memoria è collegato a un diverso filo conduttore del data BUS in modo tale che tutti i bit vengano trasferiti contemporaneamente. Quando la cella viene letta ogni elemento impone il suo contenuto sul rispettivo filo conduttore del data BUS. Allo stesso modo, quando ciascun elemento viene memorizzato nella cella, carica al suo interno il valore del rispettivo bit ricevuto dal filo conduttore del data BUS.

Con la piattaforma Tinkercad, realizziamo una cella di memoria per memorizzare 1 bit con flip-flop D.

Per realizzare un flip-flop D si parte dal circuito di un flip-flop JK con ingressi collegati come in figura accanto:



Dove il flip-flop JK è realizzato secondo questo schema con porte NAND:



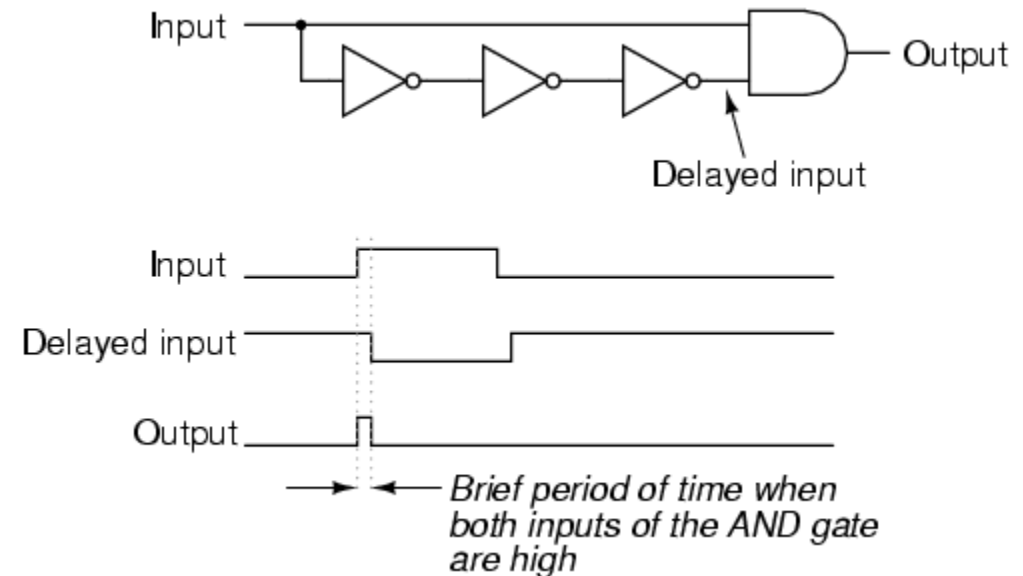


Se vogliamo che l'uscita del flip-flop D venga aggiornata sul fronte di salita del segnale di clock (CLK), è necessario che all'ingresso realizziamo un circuito in grado di generare un rapidissimo impulso in corrispondenza del fronte di salita del clock. Per farlo giochiamo con i ritardi di propagazione delle porte (iniziamo a vedere le porte nel caso reale e non più teorico, in cui la risposta era immediata).

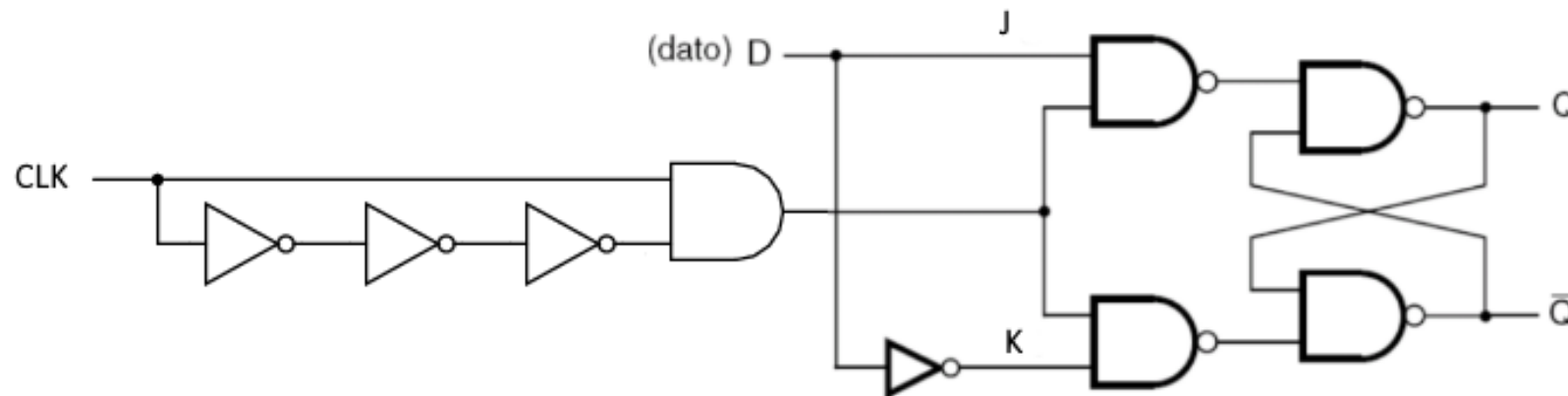
Il segnale di clock è applicato alla porta AND in due momenti successivi, direttamente sul primo ingresso e attraverso una serie di porte NOT sul secondo ingresso.

Tutte le porte generano un ritardo di propagazione tra l'applicazione del segnale di ingresso e la risposta dell'uscita; questo ritardo viene sfruttato utilizzando una porta a singolo ingresso come la porta NOT.

La porta AND darà in uscita un impulso a 1 solo quando i suoi ingressi sono a 1 e quindi ampio tanto quanto il ritardo che riesco a creare aggiungendo un numero dispari di porte NOT in serie ad uno dei suoi ingressi.



Mettendo insieme i vari componenti otterremo:



dove le porte NOT possono essere realizzate anch'esse con porte NAND:

Desired NOT Gate



$$Q = \text{NOT}(A)$$

NAND Construction



$$= A \text{ NAND } A$$

Truth Table

Input A	Output Q
0	1
1	0

Un computer tipicamente contiene differenti tipi di memoria, essenzialmente appartenenti a tre diverse categorie:

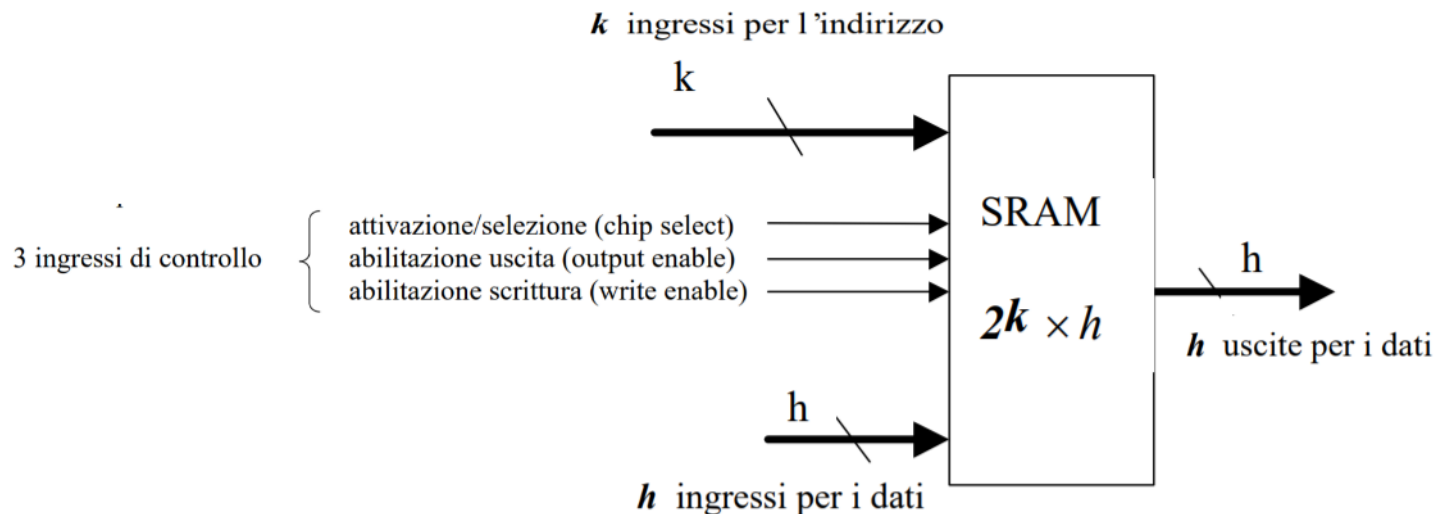
- **RAM**
- **ROM**
- **Cache**

## RAM

Il termine RAM (Random Access Memory) indica che in tali memorie è possibile accedere in **qualsunque** locazione di memoria e per qualunque tipo di accesso (lettura o scrittura). La caratteristica principale delle RAM è il fatto che l'informazione in esse contenuta rimane solo quando vengono alimentate (**volatilità**). Le RAM si suddividono in due sottocategorie chiamate RAM dinamiche (DRAM) e RAM statiche (SRAM).

La **RAM statica** è un'unità che memorizza un gran numero di parole in un insieme di **flip-flop**, opportunamente connessi, mediante un sistema di indirizzamento e trasferimento (lettura/scrittura) di parole.

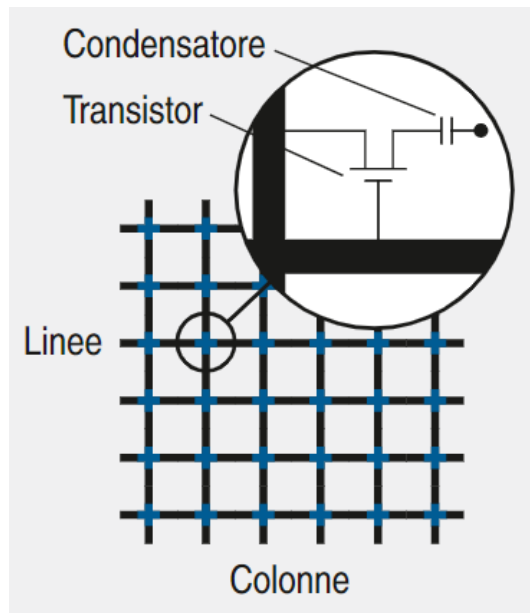
Se ciascun indirizzo di memoria è costituito da  $k$  bit, posso far riferimento a  $2^k$  **registri** di memoria con indirizzi nell'intervallo  $0 \dots 2^k - 1$ . Ciascuno dei  $2^k$  registri è formato da un gruppo di  $h$  flip-flop che formano una **parola** di  $h$  bit (ciascun flip-flop memorizza un bit).



Nella memoria statica (SRAM) il bit memorizzato viene mantenuto per un tempo arbitrario (finché c'è alimentazione) e, a differenza delle DRAM, nelle SRAM non occorre il rinfresco periodico dei dati.

È una memoria molto veloce con tempi di accesso attualmente dell'ordine delle decine di nanosecondi (da 5 a 10 ns). Generalmente, per la loro velocità, vengono utilizzate per realizzare memoria cache.

Nella **RAM dinamica** un bit viene espresso sotto forma di stato di carica di **un transistor**, non è stabile come nella SRAM e quindi va "rinfrescato" periodicamente. Usando meno transistor (perché non devo implementare flip-flop), la densità di bit memorizzati in una DRAM è molto più alta rispetto ad una SRAM e quindi la DRAM ha, a parità di capacità, un costo inferiore.



I dati sono immagazzinati, sotto forma di carica elettrica, in celle costituite da un transistor e da un condensatore.

Le celle sono ordinate secondo uno schema a matrice. È uno schema che prevede delle righe (Row line o Word line) collegate al gate del transistor, e delle colonne (Bit line) a loro volta organizzate in banchi.

Poiché i condensatori perdono nel tempo il loro stato di carica, è necessario effettuare un'operazione di ricarica periodica chiamata refresh (rinfresco).

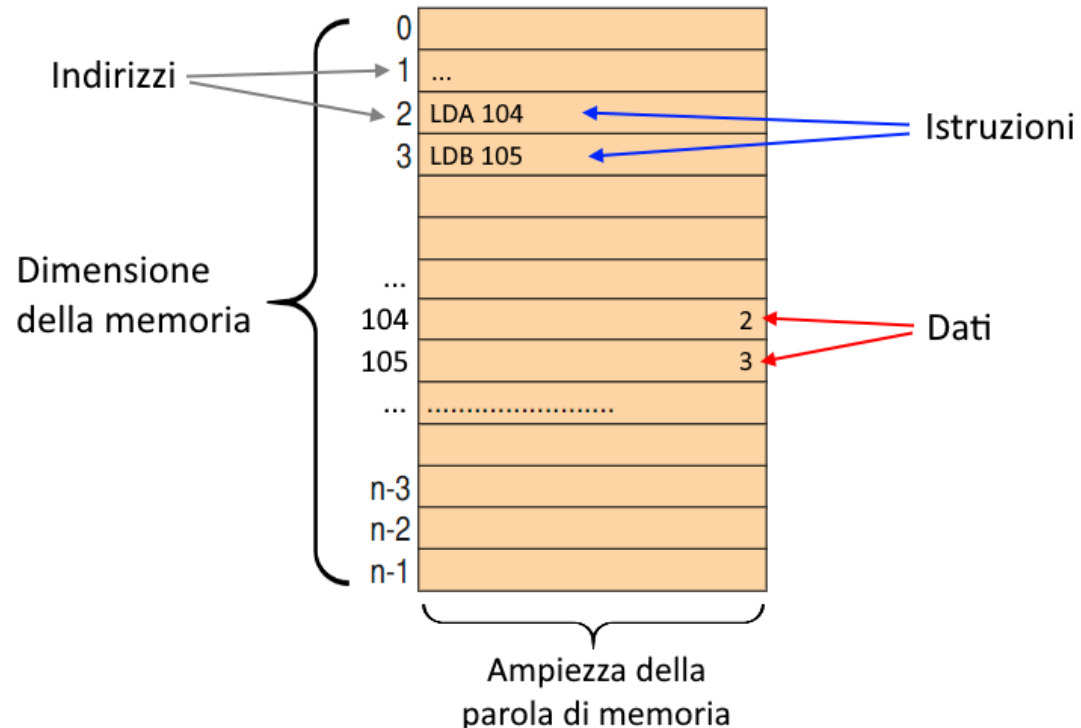
La memoria RAM dinamica è la memoria principale del nostro computer. È caratterizzata da tempi di accesso che variano tra i 20 ns e i 70 ns. La DRAM più conosciuta è quella sincrona (SDRAM), la cui caratteristica principale è la capacità di operare in sincronismo con il clock del BUS di sistema. Per quanto detto sinora, le SDRAM sono, attualmente, le uniche memorie in grado di dialogare con i BUS a frequenze molto elevate.

## ROM

Con il termine ROM (Read Only Memory) indichiamo una categoria di memorie accessibili solo in lettura. In realtà, attualmente esistono particolari ROM che possono essere anche riscritte. In ogni caso tutte le ROM sono caratterizzate dal fatto che l'informazione in esse contenuta permane anche quando manca la corrente. Le memorie ROM vengono in genere utilizzate per memorizzare programmi e dati di configurazione essenziali per il funzionamento del computer che devono essere memorizzati anche quando il computer è spento. Esistono differenti tipi di ROM:

- **ROM non programmabili.** Esse vengono prodotte già inglobando il programma o i dati.
- **PROM** (*Programmable ROM*). Normalmente sono vuote al loro interno e possono essere programmate successivamente attraverso appositi programmatori di PROM, tuttavia non possono essere più modificate nel contenuto.
- **EPROM** (*Erasable Programmable ROM*). Normalmente sono vuote al loro interno e possono essere programmate attraverso appositi programmatori di EPROM. A differenza delle PROM, la programmazione può avvenire più volte, a patto di cancellare la vecchia programmazione tramite raggi UV (ultravioletti). Sono identificabili per la presenza di una finestrella posta nella parte superiore del circuito, come vediamo nella figura a lato, che permette di ricevere i raggi ultravioletti.
- **EEPROM** (*Electrical Erasable Programmable ROM*). Sono identiche alle EPROM, dalle quali differiscono solo per il fatto che la cancellazione della vecchia programmazione è realizzata tramite corrente elettrica.

La memoria centrale (RAM) è costituita da un insieme di celle della medesima ampiezza, dette **parole di memoria**. L'ampiezza della parola di memoria, cioè il numero di bit per parola, cambia da computer a computer: valori tipici sono 32 o 64 bit (valori maggiori per sistemi ad elevate prestazioni). Le parole di memoria sono logicamente organizzate in un array di celle numerate a partire da 0. La posizione di una parola di memoria nell'array costituisce l'**indirizzo** di quella parola. L'indirizzo di valore massimo precisa la **dimensione** della memoria detta anche **spazio di indirizzamento** e il suo valore dipende dalle caratteristiche della CPU, dei BUS e della scheda madre. La quantità di celle presenti nella memoria centrale si misura in byte o nei suoi multipli, in quanto ciascuna cella contiene 1 byte.



Lo spazio di indirizzamento è definito dal numero di parole indirizzabili e dipende esclusivamente dal numero di bit dell'indirizzo e non dalla dimensione delle celle di memoria (parole di memoria). Il calcolo dello spazio di indirizzamento si ottiene a partire dal numero di fili conduttori del BUS indirizzi individuando tutte le possibili disposizioni con la ripetizione di due elementi (0 e 1). Sapendo che a ogni filo conduttore corrisponde un bit otteniamo il risultato applicando la formula seguente:

$$\text{spazio di indirizzamento} = 2^n$$

dove **n** è il numero di bit del BUS indirizzi.

Dalla formula si ricava allora che:

Numero bit del BUS indirizzi	Spazio di indirizzamento
10	1 k di celle ( = 1024 celle)
16	64 k celle
20	1 M celle (RAM dei processori fino all'80286)
32	4 G celle (RAM dei processori fino al Pentium)
36	64 G celle (RAM dei processori fino al Pentium IV)
40	1 T celle (RAM dei processori fino all'Athlon 64)
64	16 E celle (RAM dei processori fino al Core i7)

dove:

1 k (kilo) = 1024

1 M (mega) = 1024 k

1 G (giga) = 1024 M

1 T (tera) = 1024 G

1 P (peta) = 1024 T

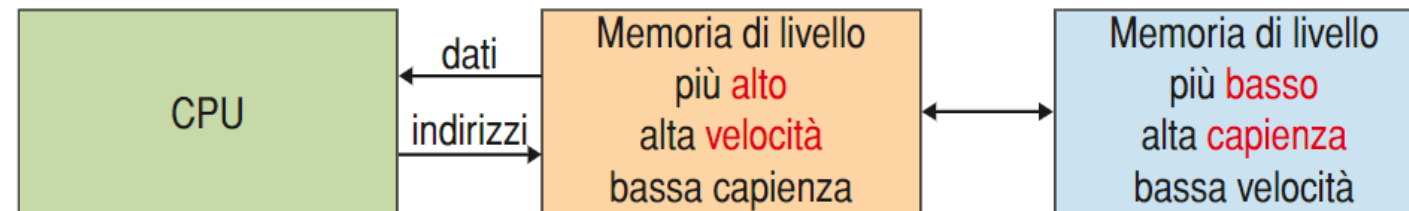
1 E (exa) = 1024 P



Nell'architettura Von Neumann il canale di comunicazione tra la CPU e la memoria è il punto critico del sistema ed è chiamato collo di bottiglia.

La tecnologia consente di realizzare processori sempre più veloci e memorie sempre più capienti, tuttavia la velocità di accesso delle memorie non è adeguata alla crescita repentina delle CPU. La soluzione ottimale per un sistema di gestione della memoria dovrebbe garantire costi minimi con capacità massime e bassi tempi di accesso al dato.

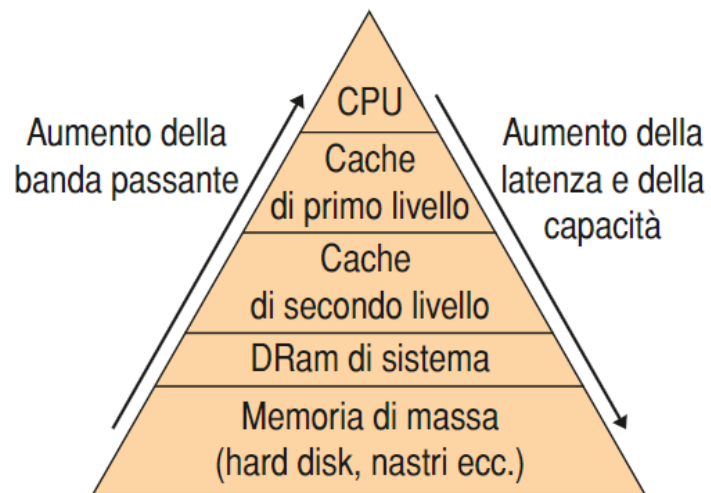
La soluzione che è stata escogitata prevede l'uso di memorie con tempi di accesso diversi, organizzate secondo gerarchie: con l'uso di tecnologie differenti possiamo soddisfare al meglio ciascuno dei requisiti.



La memoria all'interno della scheda madre di un PC è organizzata in livelli gerarchici: ogni livello è caratterizzato da una dimensione crescente e da un tempo di accesso decrescente.

La memoria RAM è molto più lenta della CPU, per cui per migliorare le prestazioni vengono combinati tipi di memoria veloce con tipi di memoria più capienti ma lente.

La CPU legge e scrive i dati in modo diretto sulla cache di primo livello, quindi via via sulle memorie inferiori secondo lo schema seguente. La **cache** rappresenta una memoria temporanea che memorizza un insieme di dati che possono essere successivamente recuperati su richiesta, ad altissima velocità. L'origine del nome deriva appunto dal fatto che la memoria cache e il suo utilizzo sono trasparenti al programmatore, quindi nascosti.



Quando la cache riceve una richiesta dalla CPU, potrebbe non possedere i dati necessari: si parla in questo caso di **cache miss**, cioè di dato mancato nella cache. Solitamente il cache miss avviene quando un dato viene richiesto per la prima volta dalla CPU alla partenza del sistema, oppure quando si tratta di una quantità di dati che non possono essere contenuti all'interno della cache.

Quando invece il dato viene trovato nella cache si parla di **cache hit**, per indicare il successo nella lettura.

Quindi, le memorie cache fanno da tramite tra la CPU e la memoria RAM compensando il deficit legato alla lentezza della RAM. Esistono diversi livelli di cache. Quella di primo livello spesso risiede all'interno del microprocessore ed è più potente e veloce della versione integrata sulla motherboard. Le cache mantengono separati dati e istruzioni secondo l'architettura Harvard, consentendo un accesso simultaneo.

BUS

Il BUS può essere identificato da un insieme di linee che conducono elettricità, ciascuna delle quali collegata a un pin di un dispositivo. Per collegare due dispositivi che usano 32 piedini per comunicare, il BUS sarà formato da 32 linee costituite da altrettanti fili conduttori. Questo in ragione del fatto che ciascuna linea del BUS può trasmettere solo un segnale.



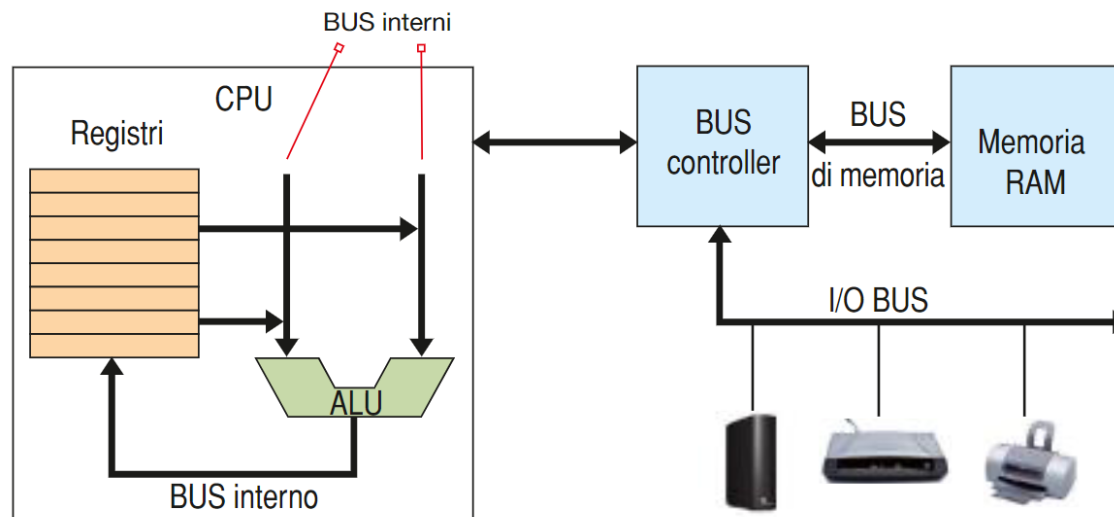
I BUS si possono dividere in **due categorie**:

- BUS interno alla CPU, trasporta dati verso e dall'ALU;
- BUS esterno alla CPU, trasporta dati da e verso la memoria e i dispositivi di I/O.

Si definisce **protocollo del BUS** l'insieme di regole precise che consentono ai vari dispositivi, rappresentati anche da schede particolari, di comunicare attraverso il BUS di sistema.

Il dispositivo che intende inviare un segnale sulla linea del BUS non fa altro che attivare un voltaggio particolare che identifica il dato inviato, secondo le specifiche definite dal protocollo.

Il segnale viaggia sulla linea e può essere ricevuto da tutti gli altri dispositivi collegati al BUS: possiamo dire che il protocollo del BUS regola la trasmissione e la ricezione dei dati.



In generale possiamo affermare che il dispositivo in grado di iniziare un trasferimento dei dati è denominato **master**, mentre un dispositivo che inizia la comunicazione solo su comando del master assume il nome di **slave**.

Alcuni dispositivi sono in grado di comportarsi sia come master sia come slave.

In un certo istante può esistere un solo master per cui è necessario regolare l'accesso a un BUS attraverso un'operazione chiamata **arbitraggio**.

Il numero di linee di un BUS è detto larghezza del BUS ed è direttamente proporzionale alle prestazioni: con **n** linee a disposizione un BUS può infatti indirizzare **2<sup>n</sup>** diverse locazioni di memoria.

Per **aumentare le prestazioni dei BUS** i costruttori hanno adottato due tecniche che presentano tuttavia degli inconvenienti:

- diminuzione della durata del ciclo di BUS aumentando in tal modo il numero di bit trasferiti al secondo. È di difficile implementazione perché i segnali di linee diverse si muovono a velocità diverse, presentando incompatibilità con i modelli precedenti;
- aumento della larghezza del BUS, ottenendo in tal modo un numero maggiore di linee a discapito della velocità. In realtà questo finora è l'approccio più diffuso.

Possiamo sintetizzare di seguito le **caratteristiche principali di un BUS in termini prestazionali**:

- **bit rate**: numero di bit al secondo (b/s) trasmessi attraverso il canale;
- **larghezza**: numero di linee indipendenti per la trasmissione di dati;
- **velocità**: frequenza del ciclo di BUS per un BUS sincrono;
- **banda**: numero (massimo) di Byte al secondo (B/s) che si possono trasmettere attraverso il canale.

Possiamo suddividere i BUS in due categorie principali: **sincroni** e **asincroni**.

### **BUS sincrono**

Possiede in ingresso un segnale proveniente da un oscillatore che ne sincronizza le operazioni al microprocessore. Il segnale su questa linea è un'onda quadra generata dal clock di sistema. Ha una struttura più semplice, grazie alla discretizzazione indotta dai cicli di clock, tuttavia rende difficile sfruttare al massimo le prestazioni dei dispositivi a esso connessi, rendendo le prestazioni globali vincolate al dispositivo più lento che lo utilizza

### **BUS asincrono**

Il BUS asincrono non è dotato di un clock principale. I cicli del BUS possono essere della lunghezza necessaria e non devono essere uguali per tutti i dispositivi. Esso è caratterizzato da una semplicità strutturale in quanto non esiste un segnale di clock del BUS, e sfrutta al meglio le prestazioni di ciascun dispositivo che lo utilizza. Possiede una circuiteria di corredo complessa che deve gestire la sincronizzazione: **MSYN** (*Master SYNchronization*) e **SSYN** (*Slave SYNchronization*):

- **MSYN**: attivato dal master (in questo caso, CPU) quando è pronto per ricevere i dati, disattivato quando ha completato la lettura dei dati;
- **SSYN**: attivato dallo slave (in questo caso, memoria) quando i dati sono pronti sul BUS, disattivato quando il master disattiva MSYN al termine dell'operazione.

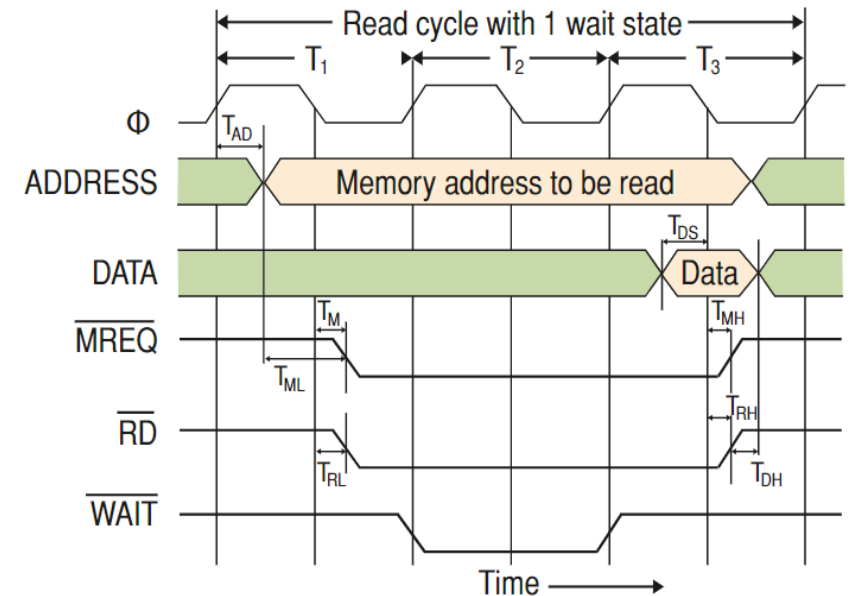
## CICLO DI LETTURA DA MEMORIA CON BUS SINCRONO

In questo ciclo avviene la **lettura** di un dato dalla memoria RAM da parte della CPU mediante un BUS sincrono. L'operazione avviene in un numero definito di cicli di clock, chiamati  $T_1$ ,  $T_2$  ecc. Il grafico che segue mostra le operazioni necessarie per eseguire tale operazione.

Vediamo il significato dei principali segnali coinvolti:

- $\Phi$ : clock del BUS;
- **ADDRESS**: linee di indirizzo controllate dal master (CPU);
- **DATA**: BUS dei dati condivise da master e slave
- **MREQ**: segnale di controllo generato dalla CPU che indica il tipo di indirizzo (0 = memoria, 1 = dispositivo di I/O);
- **RD**: tipo di operazione richiesta dalla CPU (0 = lettura, 1 = scrittura);
- **WAIT**: linea di controllo generata dallo slave, in questo caso dalla memoria (0 = attesa, 1 = dato valido). Il segnale **WAIT** generato dalla memoria informa la CPU che il dato è stabile e leggibile. In caso contrario la CPU attende un successivo ciclo  $T$ , fino a quando il segnale non è attivo (alto).

Se prendiamo un BUS sincrono con frequenza di clock a 40 MHz, il ciclo del BUS è di 25 ns ( $T = 1/f$ ). La lettura dalla memoria richiede 40 ns dal momento in cui l'indirizzo è stabile, con 3 cicli di BUS per leggere una parola.





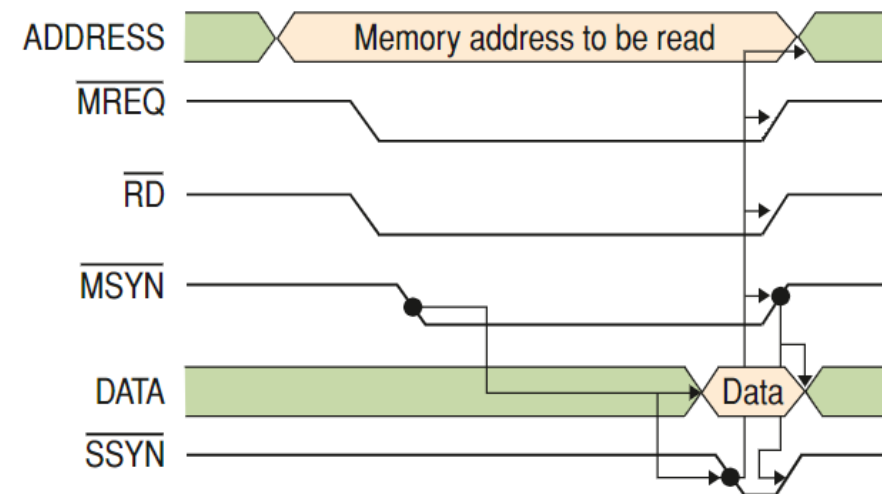
1. Nel primo ciclo **T** ( $T_1$ ), la CPU attiva l'indirizzo della cella "leggere" (segnale **ADDRESS**) dopo un tempo  $T_{AD}$ ;
2. Sempre nel primo ciclo **T**, dopo un istante chiamato  $T_{ML}$ , viene attivato il segnale **MREQ** (attivo basso) che attiva i chip della memoria. Contemporaneamente viene attivato dalla CPU il segnale **RD** che indica alla memoria che si tratta di un'operazione di lettura;
3. Sul fronte di discesa dell'istante  $T_2$  la CPU verifica il segnale **WAIT** proveniente dalla memoria. Se tale segnale è basso significa che la memoria non ha ancora risposto con il dato richiesto;
4. Sul fronte di discesa di  $T_3$  la CPU verifica ancora una volta il segnale **WAIT**: se attivo (alto) significa che il dato è disponibile sul BUS dati (DATA). Prima del fronte di salita di  $T_3$  la CPU legge il dato presente sul BUS dati (DATA). Dopo alcuni istanti  $T_{MH}$  e  $T_{RH}$  la CPU disattiva i segnali di attivazione della memoria (**MREQ**) e di lettura (**RD**).

## CICLO DI LETTURA DA MEMORIA CON BUS ASINCRONO

In questo ciclo avviene la **lettura** di un dato dalla memoria RAM da parte della CPU mediante un BUS asincrono. La lettura, che avviene secondo la sincronizzazione fra **master** (CPU) e **slave** (memoria), è realizzata attraverso un **full handshake**.

Il grafico che segue mostra le operazioni necessarie per eseguire tale operazione.

1. Inizialmente la CPU attiva l'indirizzo della cella "leggere" (segnale **ADDRESS**).
2. La CPU attiva il segnale che informa la memoria che si tratta di una lettura (**RD**).
3. La CPU attiva il segnale **MSYN**.
4. Quando il dato è reso disponibile, la memoria attiva il segnale **SSYN** in risposta a MSYN.
5. I segnali **MSYN**, **RD** e **MREQ** vengono disattivati dalla CPU in risposta a SSYN
6. **SSYN** viene negato in risposta alla negazione di MSYN



È necessario regolare l'accesso a un BUS attraverso un'operazione chiamata **arbitraggio** che può essere fondamentalmente di tre tipi:

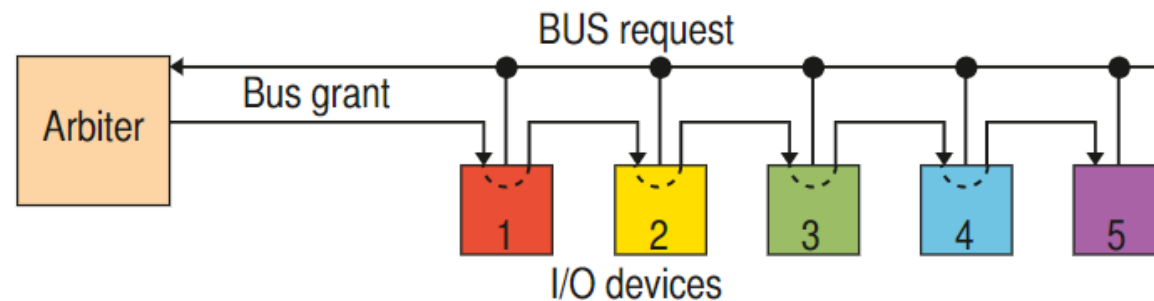
- **arbitraggio centralizzato (daisy chaining a un livello);**
- **arbitraggio centralizzato (daisy chaining con più livelli di priorità);**
- **arbitraggio distribuito**

## Arbitraggio centralizzato (daisy chaining a un livello)

La concessione dell'accesso al BUS avviene per un solo dispositivo alla volta ed è gestita da un dispositivo chiamato **arbitro del BUS** che, attraverso due linee, si collega con i potenziali master che potrebbero richiedere il controllo del BUS. La sequenza delle operazioni dello schema è così sintetizzata:

- il dispositivo master che vuole utilizzare il BUS attiva la linea BUS request;
- quando la linea BUS request viene attivata, l'arbitro attiva la linea BUS grant (segnale di concessione del bus) appena è disponibile;
- la linea BUS grant viene propagata da un dispositivo a quello successivo solo se esso non ha attivato la linea BUS request;
- il dispositivo che ha attivato la linea BUS request deve attendere che la linea BUS grant diventi attiva prima di poter utilizzare il BUS.

in un meccanismo Daisy Chaining il dispositivo fisicamente più vicino all'arbitro ha più priorità rispetto ai dispositivi più lontani.

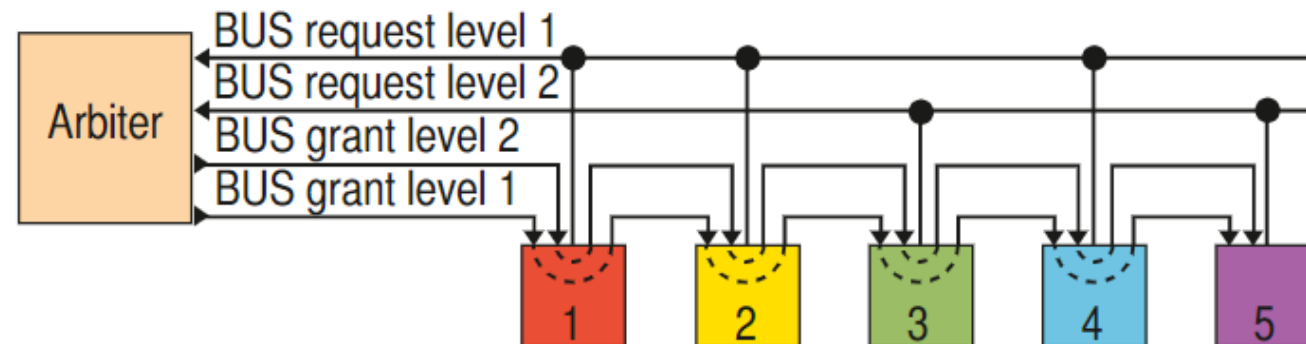


Per avere più livelli di priorità, caratterizzati ognuno da una propria linea di richiesta del bus e una linea di grant, devo utilizzare uno schema a:

### Arbitraggio centralizzato (daisy chaining a più livelli)

Anche in questo caso la concessione dell'accesso al BUS avviene per un solo dispositivo alla volta ed è gestita da un dispositivo chiamato arbitro del BUS. La sequenza delle operazioni dello schema è così sintetizzata:

- ciascun livello è indipendente e funziona come descritto in precedenza tranne quando le due (o più) linee BUS request sono attive contemporaneamente perché, in tal caso, l'arbitro attiva solo la linea BUS grant di livello più basso (di priorità più alta).

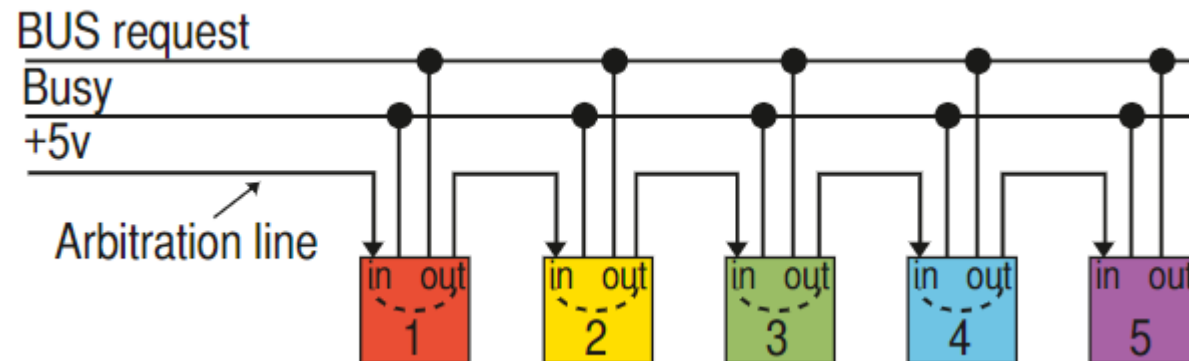


## Arbitraggio distribuito

- non esiste l'arbitro del BUS: tutti i dispositivi regolano da soli l'accesso esclusivo al BUS;
- ogni unità funzionale ha priorità fissata e diversa, espressa da un codice univoco associato all'unità stessa. Se dunque più dispositivi hanno bisogno del bus, questo viene preso da chi ha priorità maggiore.

Questo tipo di arbitraggio utilizza solo 3 linee:

- **BUS request**, linea di richiesta, che effettua la richiesta del bus;
- **busy**, è attivata dal master che ne fa richiesta;
- **arbitration line**, linea di arbitraggio, che collega tutti i dispositivi. È usata per arbitrare il bus ed è alimentata a 5 volt.



Quando nessuno richiede il bus, la linea di arbitraggio si propaga per tutti i dispositivi, mentre se un dispositivo lo richiede, dovrà controllare (sulla linea Busy) se il bus è libero (Busy=0) e se il segnale della linea di arbitraggio (IN) è attivato; in questo caso nega OUT in modo che tutti i dispositivi successivi vedranno IN negato e a loro volta negheranno OUT. In questo modo il dispositivo con priorità maggiore che ha fatto richiesta, avrà il possesso del bus, attiverà la linea di Busy e OUT, e inizierà il proprio trasferimento.

La sequenza delle operazioni dello schema riportato precedentemente è così sintetizzata:

- il dispositivo master che vuole utilizzare il BUS nega la linea **arbitration out**;
- il dispositivo master che vuole utilizzare il BUS attende che la linea **arbitration in** diventi attiva e che la linea busy venga negata, quindi attiva la linea busy, attiva la linea **arbitration out** e utilizza il BUS;
- una volta terminato l'uso del BUS, il dispositivo master che utilizza il BUS nega la linea **busy** permettendo così a un nuovo dispositivo di prenotare l'utilizzo del BUS.