Prestazioni dei Computer

Tempo di risposta e produttività

- **Tempo di risposta**: latenza, quanto ci vuole per eseguire una operazione.
- **Produttività**: Lavoro totale svolto per unità di tempo.
  In che modo sono influenzati?
- Sostituire il processore con una versione più veloce?
- Aggiungere più processori?

Tempo di risposta

Cosa determina le prestazioni di un programma?

- **Algoritmi**
  - Determina il numero di **operazioni** eseguite.
- **Linguaggio di programmazione**, **compilatore**, **architettura**
  - Determina il numero di **istruzioni macchina** eseguite per operazione.
- **Processore e memoria di sistema**
  - Determinano quanto *veloce* le istruzioni vengono eseguite.
- **Sistemi di I/O** (OS incluso)
  - Determina quanto velocemente le operazioni di I/O sono eseguite.

Misuriamo il tempo di risposta

Tempo trascorso:

- Tempo di risposta totale
  - Elaborazione, I/O, overhead del SO, tempo di inattività.
- **Tempo di CPU** (Clock):
  - Tempo speso elaborando un dato lavoro:
    - Sconti sul tempo di I/O, quote di altri lavori
  - Comprende il tempo CPU dell'utente + tempo CPU del sistema
  - I programmi sono influenzati dalle prestazioni della CPU e del sistema

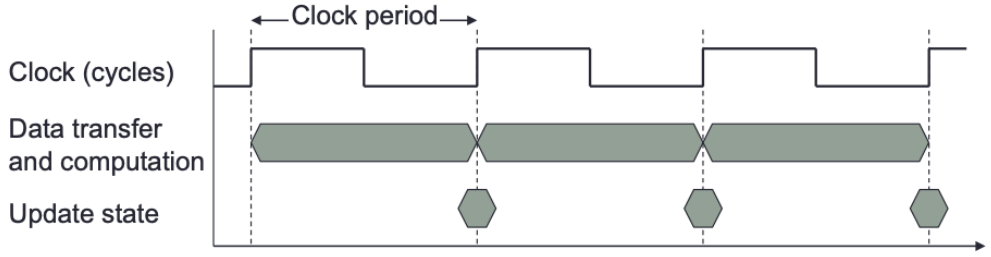
CPU Clock

È la frequenza operativa di un processore, cioè la velocità con cui la CPU può eseguire le istruzioni.

$$T = \frac{1}{f}$$

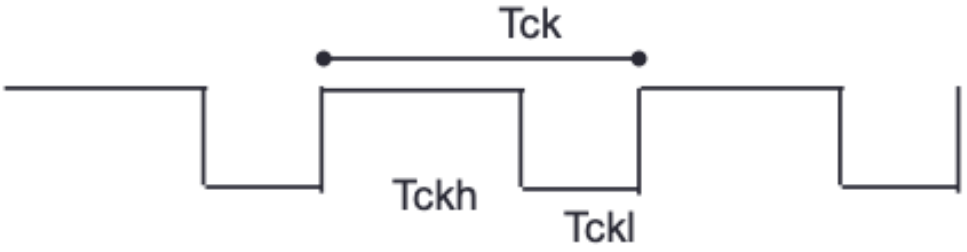
Dove:

- T è il **periodo di clock** (misurato in secondi)
- f è il suo inverso, la **frequenza del clock** (misurata in hertz, Hz)



Il **duty cycle** è la percentuale del tempo in cui il clock rimane alto.

$$duty_{cycle} = \frac{T_{clkh}}{T_{clk}}$$



Esempio:
Se la frequenza del clock della CPU è **2 GHz** (2 miliardi di cicli al secondo), il periodo di clock sarà:

$$T = \frac{1}{2 \times 10^9} = 0,5 \times 10^{-9} = 0,5 \text{ nanosecondi}(ns)$$

Ogni ciclo di clock dura **0,5 nanosecondi**.

$$\text{Clock Rate} = \frac{\text{Clock Cycles}}{\text{CPU Time}}$$

GHz:

$$1GHz = 10^9 Hz$$

ISA: Instruction Set Architecture.
Il **conteggio delle istruzioni** per un programma sono determinati dal programma, ISA e compilatore.
Il numero medio di **cicli per istruzione** (CPI), sono determinati dall'hardware della CPU. Se istruzioni differenti hanno CPI differenti, il CPI medio è influenzato dal mix di istruzioni.
Esempio:

- Computer A: $\text{Cycle Time} = 250ps, CPI = 2.0$
- Computer B: $\text{Cycle Time} = 500ps, CPI = 1.2$
- ISA identico

Quale è più veloce? Di quanto?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250ps = 1 \times 500ps \\ \text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500ps = 1 \times 600ps\end{aligned}$$

A è più veloce, di quanto?

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600ps}{1 \times 500ps} = 1.2$$

A è 1.2 volte più veloce di B.

Se classi di istruzione diverse richiedono un numero diverso di cicli:

$$\text{Clock Cycles} = \sum_{i=1}^{n}(\text{CPI}_i \times \text{Instruction Count}_i)$$

Media pesata dei CPI:

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n}\left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}}\right)$$

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles = 2×1 + 1×2 + 2×3 = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles = 4×1 + 1×2 + 1×3 = 9
  - Avg. CPI = 9/6 = 1.5

IPC: Istruzioni per ciclo:

$$\text{IPC} = \frac{\text{Instruction Count}}{\text{Clock Cycle}} = \frac{1}{\text{CPI}}$$

RIASSUMENDO:

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

- T_{sup} (*tempo di setup*) è il periodo in cui gli ingressi devono rimanere stabili prma del fronte del clock per poter essere campionati correttamente.
- T_h (*tempo di hold*) è il periodo in cui gli ingressi devono rimanere stabili dopo l'evento del clock

Power Trends

Nella tecnologia **CMOS IC**:

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

Rappresentazione dell'informazione

Rappresentazione binaria

Tutta l'informazione interna ad un computer è codificata con sequenze di due soli simboli: 0 e 1.
L'unità elementare di informazione si chiama *bit* (da *binary digit*).
Byte: 8 bit.
Word: sequenza di 32, 64, ... bits (4, 8, ... *Bytes*)

Sistema decimale posizionale (1)

La rappresentazione di un numero intero in base 10 è una sequenza di cifre scelte fra l'insieme $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
Il valore di una rappresentazione è dato da:
Parte intera:

$$a_N \cdot 10^N + a_{N-1} \cdot 10^{N-1} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0 +$$

Parte frazionaria:

$$a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} + a_{-3} \cdot 10^{-3} + \ldots$$

Sistema decimale posizionale (2)

$$\begin{aligned} 253 &= 2 \times 100 + 5 \times 10 + 3 \times 1 \\ &= 2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 \end{aligned}$$

Notazione in base 2 (1)

La rappresentazione di un numero intero in base 2 è una sequenza di cifre scelte fra $\{0, 1\}$:
Parte intera:

$$a_N \cdot 2^N + a_{N-1} \cdot 2^{N-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 +$$

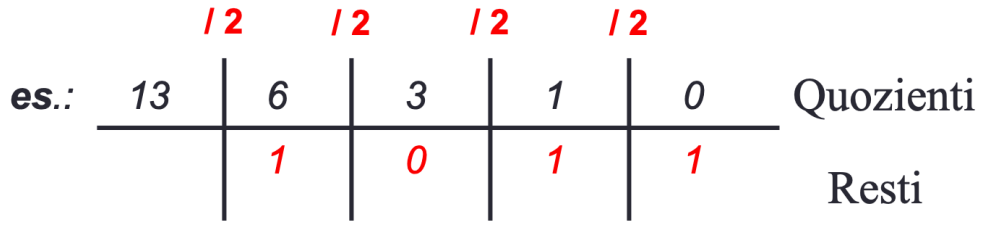
Parte frazionaria:

$$a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + a_{-3} \cdot 2^{-3} + \ldots$$

Notazione in base 2 (2)

$$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 + 0 = 6$$

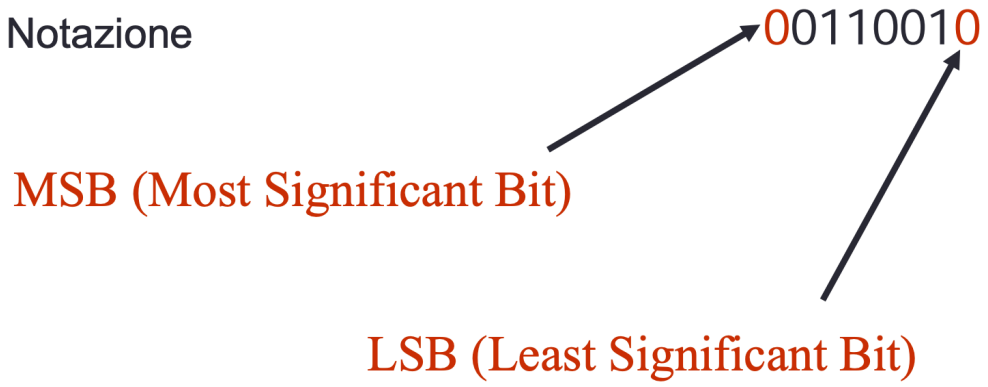
Conversione di interi: Base 10 -> Base 2



$13_{10} = (1101)_2$

La rappresentazione dei numeri all'interno di un computer

Gli interi positivi si rappresentano usando 4 o 8 byte.



- Se uso 4 byte (32 bit) posso rappresentare tutti i numeri da 0 a $2^{32} - 1$.

Numeri relativi

- **Modulo e segno**
  - Prima cifra 0 -> $+$
  - Prima cifra 1 -> $-$

$$+2 \leftrightarrow 010 \text{ e } -2 \leftrightarrow 110$$

Complemento a due

Esempio con 4 bit:
Partiamo da $+5 = 0101 (4 + 1)$

1. si invertono gli 1 con gli 0: 1010
2. si aggiunge 1: $1011 = -5$

$$1010 + 1 = 1011 = -5$$
$$-1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= -8 + 0 + 2 + 1 = -5$$

Conversione in base 8 da base 2

$$111000110101_2$$

In base 8:

$$\underbrace{111}\ \underbrace{000}\ \underbrace{110}\ \underbrace{101} = 7065_8$$

In base 16:

$$\underbrace{1110}\ \underbrace{0011}\ \underbrace{0101} = E35_{16}$$

BCD (Binary-Coded Decimal)

Si codificano in binario (4 bit) le singole cifre decimali.
254:

$$\underbrace{0010}_{2}\ \underbrace{0101}_{5}\ \underbrace{0100}_{4}$$

Rappresentazione in virgola fissa

Riservo X bit per la parte frazionaria.
Es:

$$\begin{aligned} 101.01 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 4 + 0 + 1 + 0 + 0.25 = 5,25 \end{aligned}$$

Problemi:

- *Overflow*
  - quando si sale al di sopra del massimo numero rappresenrtabile.
- *Underflow*
  - quando si scende al di sotto del minimo numero rappresentabile.

Rappresentazione in virgola mobile

Quando lavoro con *numeri molto piccoli* uso tutti i bit disponibili per rappresentare le cifre dopo la virgola.
Quando lavoro con *numeri molto grandi* le uso tutte per rappresentare le cifre in posizioni elevate.

- Ogni numero N è rappresentato da una coppia (*mantissa* M, *esponente* E) con il seguente significato

  $$N = M \times 2^E$$

  Esempio in base 10, con 3 cifre per la mantissa e 2 cifre per l'esponente:

  $$349\ 000\ 000\ 000 = 3.49 \times 10^{11}$$

  con la coppia $(3.49, 11)$ perché $M = 3,49$ e $E = 11$.

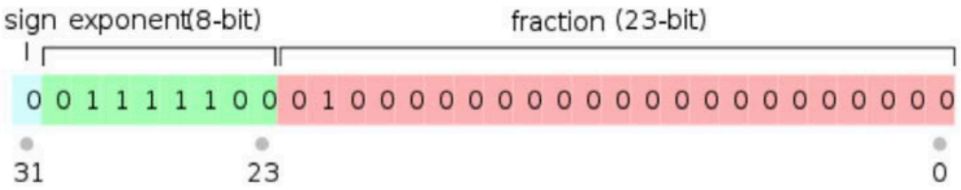
  $$0.000\ 000\ 002 = 2.0 \times 10^{-9}$$

  con la coppia $(2.0, -9)$ perché $M = 2.0$ e $E = -9$.

Standard IEEE 754

Si specificano 3 parametri:

- P: precisione o numero di bit che compongono la mantissa.
- E_{max}: esponente massimo.
- E_{min}: esponente minimo.
  Per la **precisione singola** (32 bit):
- $P = 23, E_{max} = 127$ e $E_{min} = -126$
- 1 bit di *segno*; 8 bit *esponente*



- La mantissa viene normalizzata scegliendo l'esponente in modo che sia sempre nella forma $1, xxx\ldots$
- L'esponente è *polarizzato*, ovvero ci si somma E_{max}
  - costante di polarizzazione o **bias**
  Esempio:

$$0.15625_{(10)} = \frac{1}{8} + \frac{1}{32} = 2^{-3} + 2^{-5} = 0.00101_{(2)}$$

  Normalizzazione della manitssa:

$$0.00101_{(2)} = 1.01_{(2)} \times 2^{-3}$$

- *Parte frazionaria della mantissa*: $.01_{(2)}$
- *Esponente*: -3
- *Esponente polarizzato*: $-3 + 127 = 124$

Per la **precisione doppia** (64 bit)

- $P = 52, E_{max} = 1023, E_{min} = -1022$
- 1 bit *segno*; 11 bit *esponente*
- *Parte frazionaria della mantissa*: $.01_{(2)}$
- *Esponente*: -3
- *Esponente polarizzato*: $-3 + 1023 = 1020$
  ...
  ...
  ...

Introduzione alle Reti Logiche

Reti logiche

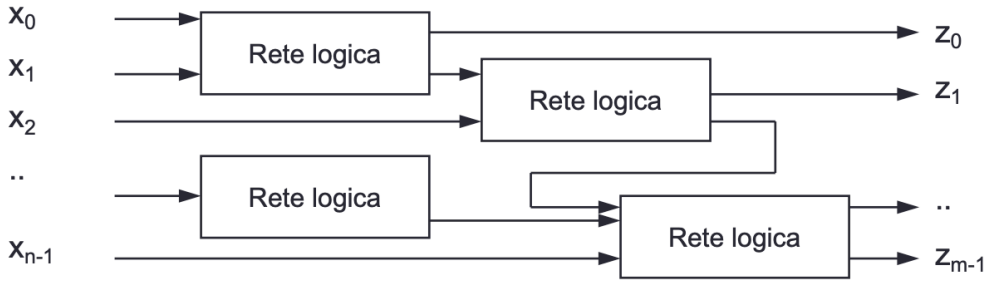
Sistema digitale avente n segnali binari di ingresso ed m segnali binari di uscita.
I segnali sono rigorosamente binari $(0/1)$.

I segnali sono grandezze funzioni del tempo

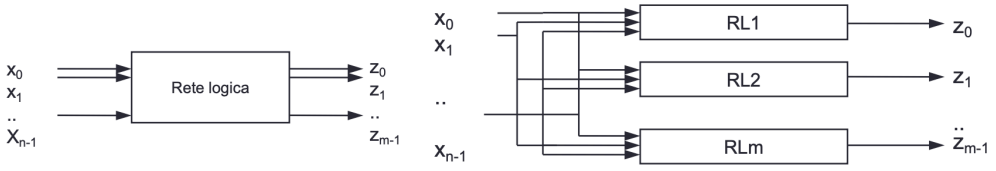
$$X = \{x_{n-1}(t), \ldots, x_0(t)\}$$

Proprietà delle reti logiche

- **Interconnessione**: l'interconnessione di più reti logiche, aventi per ingresso segnali esterni o uscite di altre reti logiche e per uscite segnali di uscita esterne o ingressi di altre reti logiche, è ancora una rete logica.



- **Decomposizione**: una rete logica complessa può essere decomposta in reti logiche più semplic.
- **Decomposizione in parallelo**: una rete logica a m uscite può essere decomposta in m reti logiche ad 1 uscita, aventi ingressi condivisi.

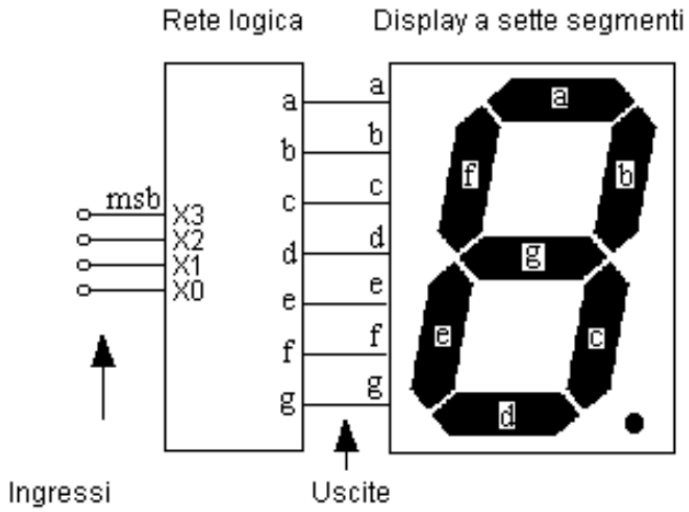


Reti combinatorie

- ogni segnale di uscita dipende solo dai valori degli ingressi in quell'istante.
- senza memoria, *non ha stato*, non ricorda gli ingressi precedenti, *transitori* a parte, basta conoscere gli ingressi in un istante per sapere esattamente quali saranno tutte le uscite nel medesimo istante.
  Esempio:
  Conversione di valori BCD su display a sette segmenti:
- Progettare una rete logica che permette la visualizzazione su un display a sette segmenti di un valore in codice BCD.
- **Codifica BCD**: impiego di 4 cifre binarie per la rappresentazione di un numero decimale da 0 a 9.

15	decimale
1111	binario
$0001\ 0101$	BCD

L'uscita $Z = \{a, b, \dots, g\}$ dipende in ogni istante dalla configurazione degli ingressi $\{x_3, x_2, x_1, x_0\}$.



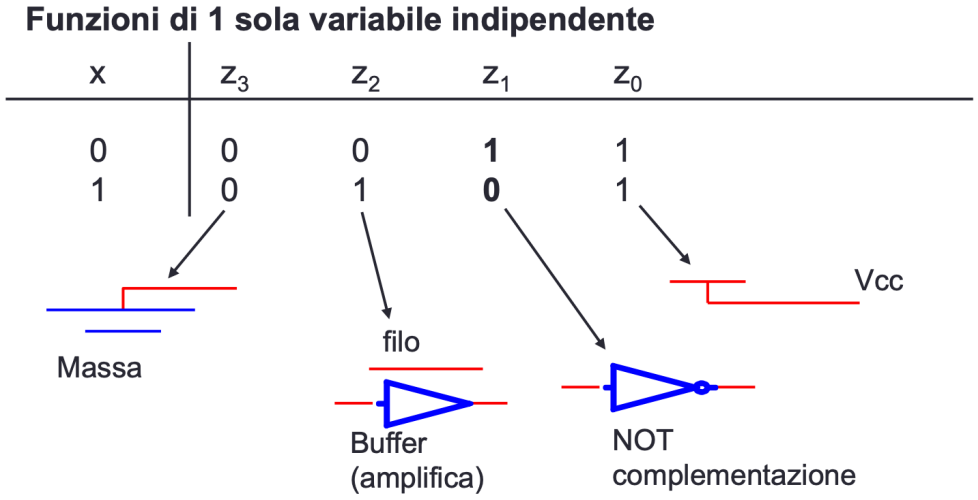
Descrizione reti combinatorie

- **Tabelle di verità**: associa le possibili combinazioni degli ingressi alle corrispondenti configurazioni delle uscite e indica il comportamento della rete logica.
  - Se la rete combinatoria ha n ingressi e m uscite, allora la tabella di verità ha $(n + m)$ colonne e 2^n righe.
  - **COMPLETAMENTE SPECIFICATE**: se ogni valore della tabella assume il valore logico di vero o falso.
  - **NON COMPLETAMENTE SPECIFICATE**: se contengono condizioni di indifferenza. Si verifica in due casi:
    - se alcune configurazioni di ingressi sono vietate.
    - se le uscite sono indifferenti per alcune configurazioni di ingresso.

Funzioni combinatorie e gate elementari

Le reti logiche combinatorie sintetizzano funzioni combinatorie.
Per ogni n, è finito il numero di funzioni combinatorie di n variabili di ingresso. Alcune funzioni

combinatorie elementari hanno una rappresentazione logica e grafica (*gate*).



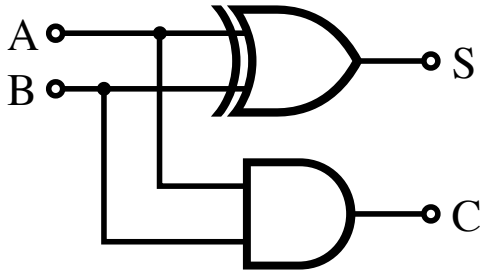
Funzioni di 2 variabili indipendenti

- **AND**
- **OR**
- **EXOR**
- **NOR**
- **EXNOR**
- **NAND**

  Quante sono le possibili funzioni binarie di n variabili?

  $$\mathrm{N.conf} = 2^{2n}$$

  Esempio di rete logica con gate:
- **HALF ADDER**, un sommatore senza riporto in ingresso.



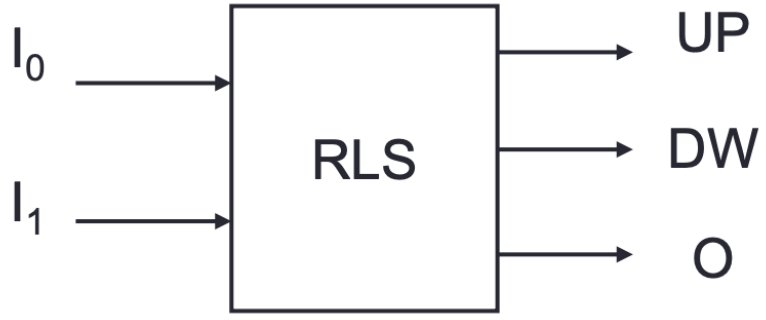
Reti sequenziali

Definizione: Circuiti digitali in cui l'uscita dipende non solo dagli ingressi attuali EE, ma anche dai valori assunti negli istanti precedenti. Hanno **memoria** e uno **stato** che riassume la sequenza degli ingressi precedenti.

- **Caratteristiche**:
  - L'uscita cambia in funzione del cambiamento degli ingressi e dello stato interno.
  - Due modalità per determinare l'uscita:
  1. Memorizzare **tutti** gli ingressi dall'accensione.
  2. Memorizzare uno **stato** che riassuma gli ingressi precedenti

Esempio:
Progettare la rete logica di gestione di un ascensore:

- La rete ha tre uscite *UP*, *DW* e *O*. *UP*, *DW* indicano le direzioni su e giù mentre *O* vale 1 se la porta deve essere aperta e 0 altrimenti. La rete ha come ingresso due segnali che indicano il piano $\{0, 1, 2, 3\}$ corrispondente al tasto premuto. Per calcolare l'uscita è necessario conoscere il piano corrente che indica lo stato interno.



Alcune reti sequenziali

- **Macchine a Stati Finiti (FSM)**:
  - Le reti sequenziali comprendono le **macchine a stati finiti (FSM)**, che giocano un ruolo cruciale nella progettazione di sistemi digitali.
  - Le FSM utilizzano elementi di retroazione, principalmente **flip-flop**, per memorizzare lo stato del sistema.
- **Registro di Stato**:
  - L'insieme dei flip-flop utilizzati in una FSM è chiamato **registro di stato**. Questo registro:
    - Memorizza lo **stato futuro** del sistema.
    - Presenta a valle lo **stato presente**, che è influenzato dagli ingressi e dallo stato corrente.
- **Caratteristiche delle FSM**:
  - Operano con un **unico segnale di clock**, il che significa che tutte le transizioni di stato avvengono in modo sincrono, facilitando la progettazione e la previsione del comportamento del sistema.
  - Le FSM possono essere **sincrone** o **asincrone**, ma le implementazioni più comuni utilizzano la sincronizzazione tramite clock.

Algebra di Boole

L'algebra di Boole è un sistema matematico fondamentale per descrivere funzioni di variabili binarie e costituisce la base della logica digitale. È definita da:

- **Insieme di Simboli**:
  - $(B = \{0, 1\})$
- **Insieme di Operazioni**:
  - $(O = \{+, \cdot, ' \})$
    - $(+)$: somma logica (*OR*)
    - $(\cdot)$: prodotto logico (*AND*)
    - $(')$: complementazione (*NOT*)

- **Postulati (Assiomi)**:
  - Rappresentano le regole fondamentali dell'algebra.

Proprietà di Chiusura

L'algebra di Boole è caratterizzata dalla proprietà di chiusura, che stabilisce che per ogni $(a, b \in B)$:

$$a + b \in B$$
$$a \cdot b \in B$$

Costanti e Variabili

- **Costanti**:
  - I simboli (0) e (1).
- **Variabile**:
  - Un simbolo che può assumere il valore di una delle costanti (0) o (1).

Espressioni nell'Algebra di Boole

Un'espressione è una stringa di elementi di (B) che segue queste regole:

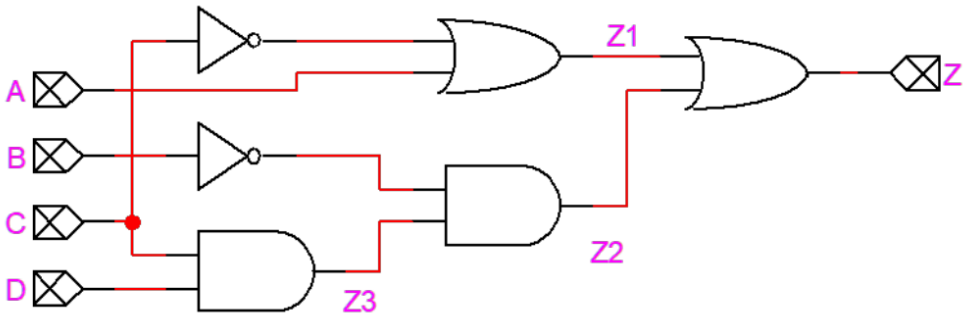
1. Una costante è un'espressione.
2. Una variabile è un'espressione.
3. Se (X) è un'espressione, allora il complemento di (X) è un'espressione.
4. Se (X, Y) sono espressioni, allora la somma logica di (X) e (Y) è un'espressione.
5. Se (X, Y) sono espressioni, allora il prodotto logico di (X) e (Y) è un'espressione.

Funzione Completamente Specificata

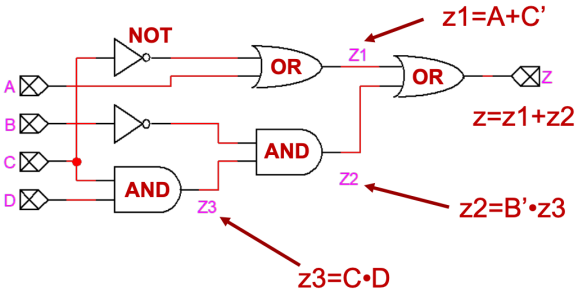
Ogni espressione di n variabili descrive una funzione completamente specificata, che può essere valutata attribuendo un valore a ciascuna variabile (0 o 1).

Analisi di uno schema logico

Dallo schema logico tramite le espressioni è possibile ricavare il comportamento di una rete logica.
Esercizio: Eseguire l'analisi del seguente schema



Esercizio: Eseguire l'analisi del seguente schema



z=z1+z2
z1=A+C'
z2=B'•z3
z3= C•D

z=z1+z2
z1=A+C'
z2=B'•C•D

z=A+C'+B'•C•D

Teoremi dell'algebra di Boole

Principio di dualità:

- ogni espressione algebrica presenta una forma **duale** ottenuta scambiando l'operatore *OR* con *AND*, la costante 0 con la costante 1 e mantenendo i letterali invariati.
- ogni proprietà vera per un'espressione è vera anche per la sua **duale**.
- il principio di dualità è indispensabile per trattare segnali attivi alti e segnali attivi bassi.
  - Logica negativa
  - Logica positiva

  Teorema di Identità:

$$(\mathrm{T1})\ X + 0 = X \qquad (\mathrm{T1'})\ X \cdot 1 = X$$

Teorema di Elementi nulli:
Utili nella sintesi di reti logiche: gli elementi nulli permettono di "lasciar passare" un segnale di ingresso in determinate condizioni.

$$(\mathrm{T2})\ X + 1 = 1 \qquad (\mathrm{T2'})\ X \cdot 0 = 0$$

Idempotenza:

$$(\mathrm{T3})\ X + X = X \qquad (\mathrm{T3'})\ X \cdot X = X$$

Involuzione:

$$(\mathrm{T4})\ (X')' = X$$

Complementarietà:

$$(\mathrm{T5})\ X + X' = 1 \qquad (\mathrm{T5'})\ X \cdot X' = 0$$

Proprietà commutativa:

$$(\mathrm{T6})\ X + Y = Y + X \qquad (\mathrm{T6'})\ X \cdot Y = Y \cdot X$$

Proprietò associativa:

$$(\mathrm{T7})\ (X + Y) + Z = X + (Y + Z) = X + Y + Z$$

$$(\mathrm{T7'})\ (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$$

Proprietà di assorbimento:

$$(\mathrm{T8})\ X + X \cdot Y = X \qquad (\mathrm{T8'})\ X \cdot (X + Y) = X$$

Proprietà distributiva:

$$(\mathrm{T9})\ X \cdot Y + X \cdot Z = X \cdot (Y + Z \qquad (\mathrm{T9'})\ (X + Y) \cdot (X + Z) = X + Y \cdot Z$$

Proprietà della combinazione:

$$(\mathrm{T10})\ (X + Y) \cdot (X' + Y) = Y \qquad (\mathrm{T10'})\ X \cdot Y + X' \cdot Y = Y$$

Proprietà del consenso:

$$(\mathrm{T11})\ (X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$$

$$(\mathrm{T11'})\ X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$$

Teorema di De Morgan:

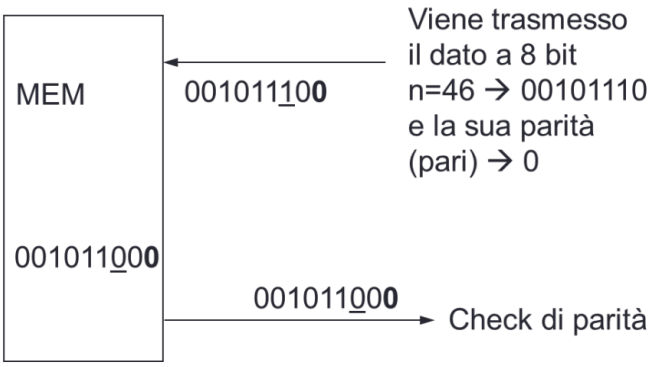
$$(\mathrm{T12})\ (X + Y)' = (X' + Y') \qquad (\mathrm{T12'})\ (X \cdot Y)' = (X' + Y')$$

generalizzabile per n variabili.

Parità

- I *codici rilevatori d'errori* sono codici in cui è possibile rilevare se sono stati commessi errori nella trasmissione.
- *Codici ridondanti*: in cui l'insieme dei simboli dell'alfabeto è minore dell'insieme di configurazioni rappresentabili col codice.
- *Codici con bit di parità*: alla codifica binaria si aggiunge un bit di parità.
- **Parità pari**: rende pari il numero di 1 presenti nella parola.
- **Parità dispari**: rende dispari il numero di 1 presenti nella parola.
  Esempio:

Vogliamo trasmettere il dato a *8 bit*: $n = 46 \rightarrow 00101110$, la sua parità *pari* $\rightarrow 0$.



- Supponiamo un errore di trasmissione durante la scrittura in memoria così che il numero memorizzato sia 001011000.
- Quando il dato viene riletto ed utilizzato viene fatto il check di parità e si verifica che quel numero non è ammissibile per la codifica binaria con parità pari *perché la somma dei bit a 1 è dispari*.

Sintesi di reti logiche combinatorie

La più semplice rappresentazione delle funzioni Booleane è attraverso la *forma canonica*, che può essere ottenuta da qualsiasi rete logica combinatoria.

Forma canonica SP

Somma di prodotti:
Teorema: una funzione di n variabili può essere rappresentata in un solo modo come somme di prodotti di n variabili (*mintermini*).

- **Minitermine**: prodotto logico di n letterali.
- Da ogni tabella si deriva la forma SP, prendendo in *OR* tutti i mintermini corrispondenti alle righe in cui l'uscita vale 1, in cui ogni variabile è in forma diretta se nella colonna appare il valore 1 ed in forma complementata se 0.
  Indipendentemente dalla complessità della rete logica da realizzare, la rete logica ottenuta dalla forma canonica è una rete molto veloce, in quanto composta da soli due livelli e mezzo (livello dei not).

r	a	b	S	R	
0	0	0	0	0	
0	0	1	1	0	
0	1	0	1	0	
0	1	1	0	**1**	r'ab
1	0	0	1	0	
1	0	1	0	**1**	ra'b
1	1	0	0	**1**	rab'
1	1	1	1	**1**	rab

R= r'ab +ra'b + rab' +rab

Forma canonica PS

Prodotto di somme:
Teorema: una funzione di n variabili può essere rappresentata in un solo modo come prodotto di

somme di n variabili (*maxtermini*).

- **Maxtermine**: somma logica di n letterali.
- Da ogni tabella si deriva la forma PS, prendendo in *AND* tutti i maxtermini corrispondenti alle righe in cui l'uscita vale 0, in cui ogni variabile e' in forma diretta se nella colonna appare il valore 0 ed in forma complementata se 1.

r	a	b	S	R	
0	0	0	0	0	(r+a+b)
0	0	1	1	0	(r+a+b')
0	1	0	1	0	(r+a'+b)
0	1	1	0	**1**	
1	0	0	1	0	(r'+a+b)
1	0	1	0	**1**	
1	1	0	0	**1**	
1	1	1	1	**1**	

  Dalla tabella precedente:
- $R = (r + a + b)(r + a + b')(r + a' + b)(r' + a + b)$

Funzioni non completamente specificate

(o **funzioni booleane incompletamente specificate**)
Se le uscite hanno condizioni di *indifferenza*.

x1	x2	Output
0	0	1
0	1	-
1	0	0
1	1	-

Sintesi e minimizzazione

Sintesi di reti logiche combinatori:

1. descrizione mediante tabella della verità
2. sintesi della espressione canonica SP o PS
3. corrispondenza 1 a 1 con schema logico

Normalmente una rete logica si dice in forma minima per indicare il minor numero di livelli e, a parità di livelli, il minor numero di gate e di ingressi dei gate.
Tecniche di minimizzazione:

- minimizzazione con manipolazione algebrica
- minimizzazione manuale (k-mappe)
- minimizzazione con algoritmi CAD o software appositi (Logisim)
  Perché minimizzare?
  Perché le forme canoniche richiedono troppi gate, troppo consumo di area.

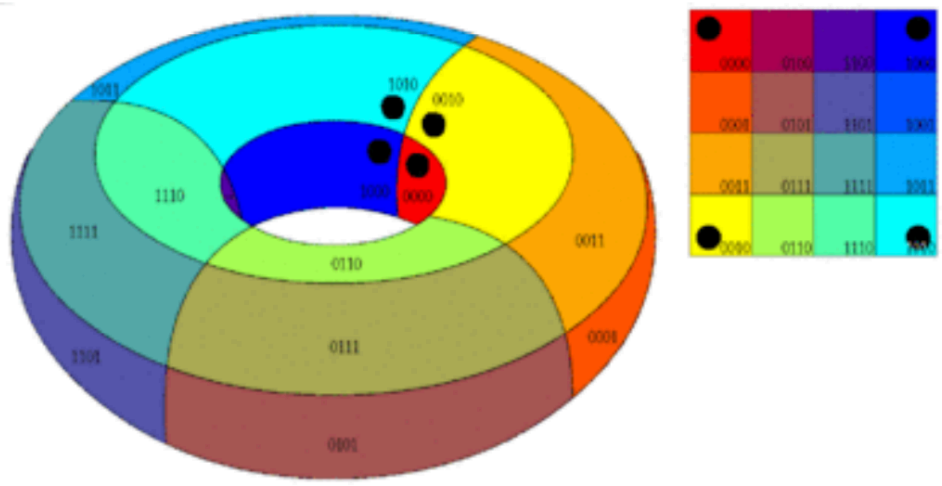
Mappe

Rappresentazione più compatta della tabella di verità, tramite matrici.
Le *righe* indicano tutte le possibili configurazioni di un sottoinsieme delle variabili di ingresso e le *colonne* tutte le configurazioni delle variabili.
Il valore nelle celle indica il valore dell'uscita nella configurazione corrispondente.

Mappe di Karnaugh

x_1x_0 \ x_3x_2	00	01	10	11
00	1	0	1	-
01	0	1	1	-
10	1	1	1	-
11	1	1	-	-

Le mappe vanno viste come «arrotolate» su se stesse. La prima riga risulta «adiacente» all'ultima riga. Stessa cosa per le colonne.



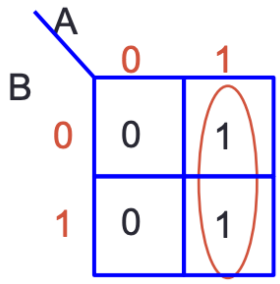
Minimizzazione con mappe di Karnaugh

Ogni casella della mappa è adiacente a caselle corrispondenti a *mintermini* (*maxtermini*) aventi distanza di Hamming unitaria dal *mintermine* (*maxtermine*) corrispondente alla casella considerata.

$$F = AB' + AB = A(B + B') = A \quad \text{proprietà distributiva}$$

A	B	Output
0	0	0
0	1	0
1	0	1

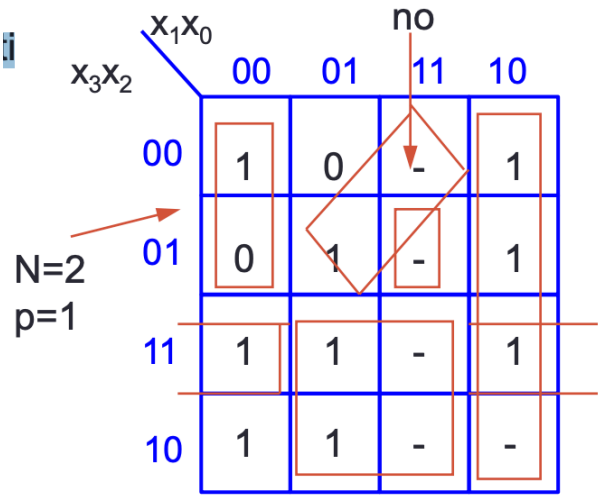
A	B	Output
1	1	1



Raggruppamenti rettangolari

Si dice **raggruppamento rettangolare** di ordine p una parte di una mappa a n variabili costituita da 2^p elementi (con $p \leq n$) tali da avere $n - p$ coordinate uguali fra loro, e di far assumere alle restanti p coordinate tutte le possibili configurazioni.
Ogni raggruppamento ha all'interno p celle adiacenti.

$$\text{RR ordine } 0 \rightarrow 1 \text{ cella}$$
$$\text{RR ordine } 1 \rightarrow 2 \text{ cella}$$
$$\text{RR ordine } 2 \rightarrow 4 \text{ cella}$$



Un Raggruppamento Rettangolare (*RR*) nel quale la funzione assume sempre valore 1 si dice **implicante** della funzione.
In modo duale, un *RR* nel quale la funzione assume sempre valore 0 si dice **implicato** della funzione.

- Si dice **copertura degli 1** un insieme di implicanti che contengono tutti gli 1 della funzione ed indifferenze.
- Un implicante non contenuto in nessun implicante di dimensioni maggiori prende il nome di **implicante primo**.
- **Implicanti essenziale**: un implicante primo contenente almeno un mintermine non contenuto in nessun altro implicante primo
- Ogni implicante essenziale deve essere contenuto nella *somma minima*. Vale il duale per gli implicati.

Una **copertura di 1** indica una forma SP.
Una **copertura di 0** indica una forma PS.

Complessità - Velocità

Per valutare la complessità di una rete logica in termini di complessità e velocità si utilizzano 3 indicatori:

- $N_{gate} =$ numero di gate,
- $N_{conn} =$ numero di connessioni,
- $N_{casc} =$ numero massimo di gate disposti in cascata
  Complessità: funzione *crescente* di N_{gate}, N_{conn}
  Velocità di elaborazione: funzione *decrescente* di N_{casc}

Forme normali e minime

Una espressione si dice:

- **normale SP** se è data dalla somma di prodotti non necessariamente di n variabili.
- **normale PS** se è data dal prodotto di somme non necessariamente di n variabili.
  Una espressione normale è equivalente alla forma canonica ma minimizzata.

Sintesi minima

(di costo minimo)

- minor numero di livelli
- minimo numero di gate
- minimo numero di connessioni
  l'espressione **minima normale** e non ridondante si ottiene con una *copertura* usando il **numero minimo di RR di ordine massimo**.
- Ordine massimo: minor numero di ingressi
- Minimo numero di RR: minimo numero di gate
- Forma normale irridondante: solo implicazioni essenziali
  Forma minima PS ed SP sono diverse.

Sintesi di reti combinatorie complesse

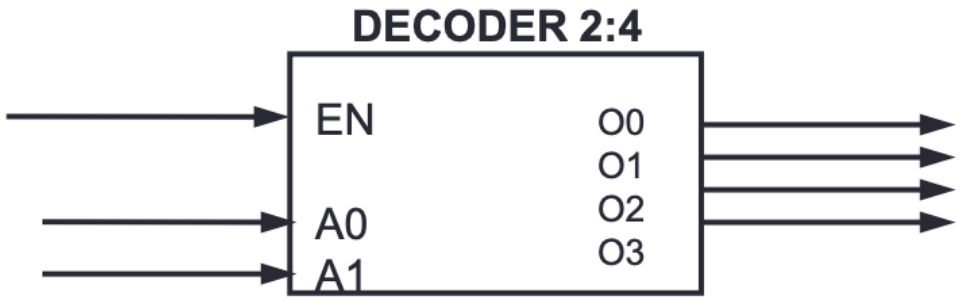
Esistono tecniche ed algoritmi per la sintesi automatica a più livelli:

- *Manipolazione algebrica*, ad esempio usando sistematicamente la proprietà distributiva.
- *Algoritmi di sintesi logica*.
- *CAD tools*.
- *Metodi empirici*.

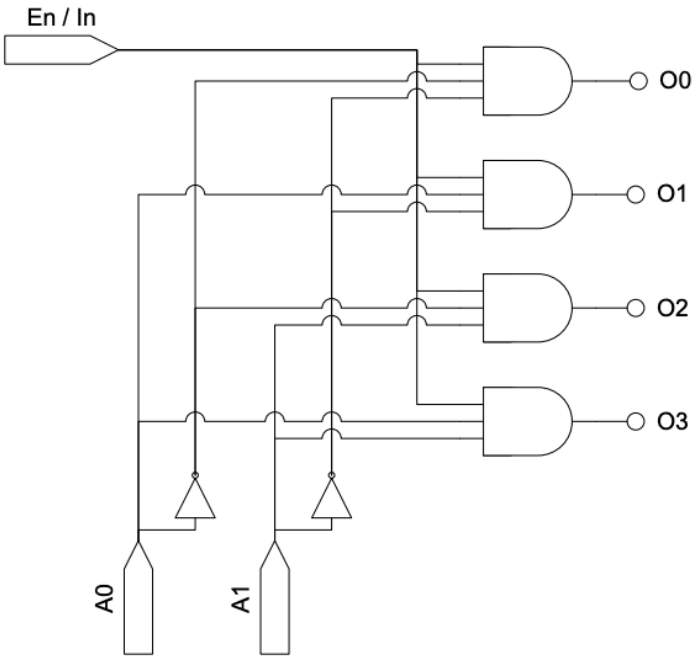
Componenti Notevoli Combinatori

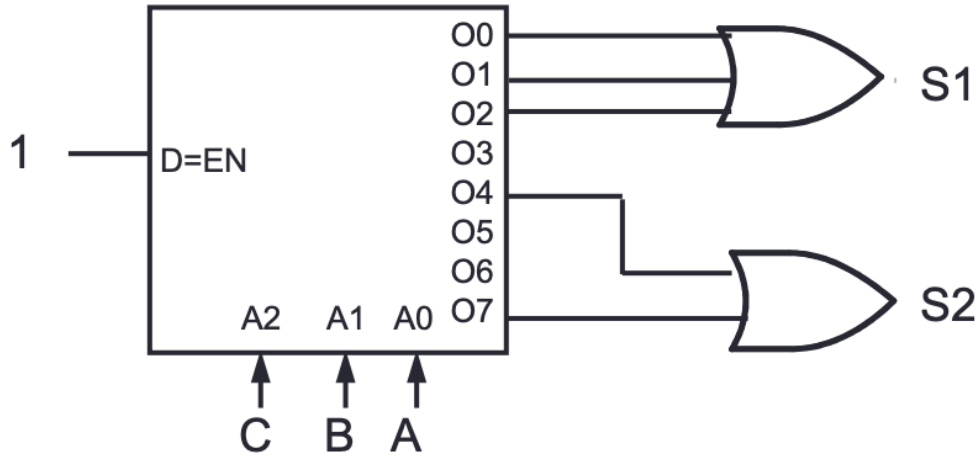
Demultiplexer/Decoder

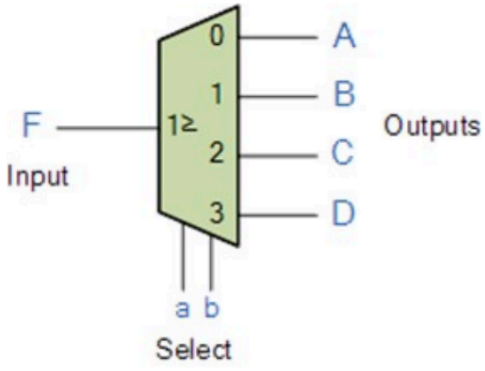
Il **demultiplexer**(*decoder*) smista un singolo input in una delle n possibili uscite.
È una rete logica con 1 ingresso, n segnali di controllo e 2^n uscite: l'uscita contrassegnata dall'indice pari alla configurazione dei segnali di controllo riceve l'ingresso, mentre le altre non sono abilitate (normalmente poste a livello logico 0).
Si dice anche *decoder* in quanto viene usato per decodificare un segnale binario (se si mantiene l'ingresso EN a 1).



Può essere usato come generatore di *mintermini*.
Esempio:
Realizzare la rete logica
$S_1 = A'BC' + A'B'C + A'B'C'$
$S_2 = AB'C' + ABC$

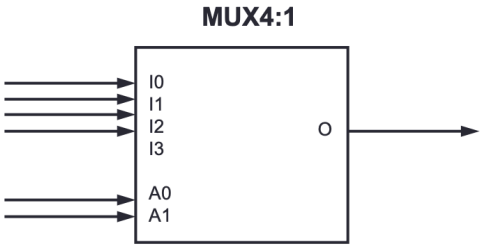


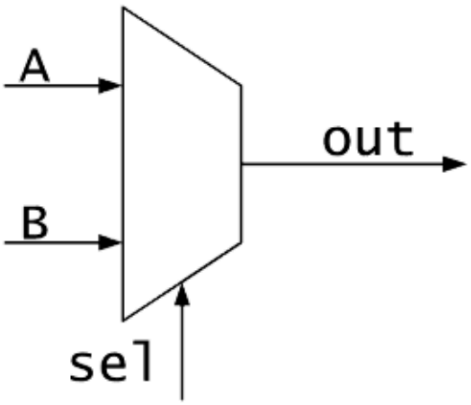




Multiplexer

Il **multiplexer** è un dispositivo logico che consente di deviare su un'unica uscita un segnale proveniente da uno tra n ingressi. Funziona come un selettore, permettendo di scegliere quale segnale di ingresso deve essere inviato all'uscita in base ai segnali di controllo.





uando i segnali di controllo vengono configurati, il multiplexer identifica quale ingresso deve essere attivato e invia il segnale corrispondente all'uscita. Questo consente di ridurre il numero di linee necessarie per trasmettere i dati, facilitando il routing delle informazioni nei circuiti digitali.

Esempio:

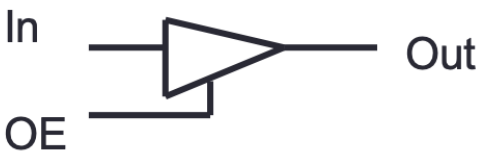
In un multiplexer 4:1 (che ha 4 ingressi dati e 2 segnali di controllo), la configurazione dei segnali di controllo determina quale ingresso sarà passato all'uscita. La tabella di verità per un multiplexer 4:1 è la seguente:

S1	S0	Uscita (Y)
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Amplificatore Tri-State

Supponiamo che:

- IN sia l'ingresso del buffer.
- OE sia il segnale di abilitazione.
- OUT sia l'uscita.



IN (ingresso)	OE (abilitazione)	OUT (uscita)
X	0	Z (alta impedenza)

IN (ingresso)	OE (abilitazione)	OUT (uscita)
0	1	0
1	1	1

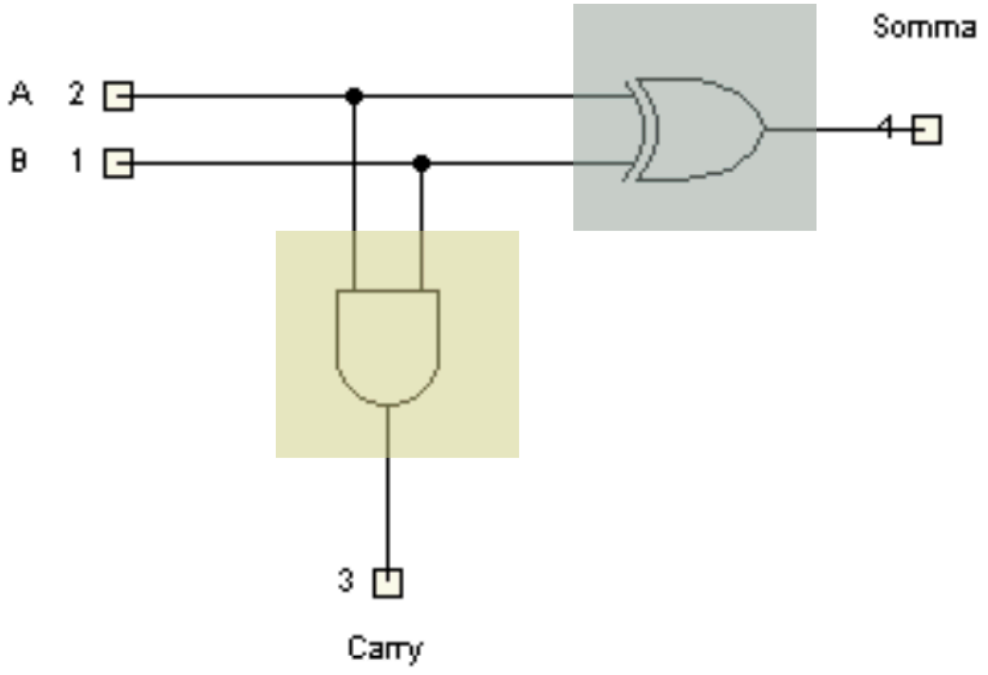
L'*impedenza* è una grandezza fisica che rappresenta l'opposizione che un circuito elettrico offre al passaggio della corrente alternata, ed è una generalizzazione della resistenza elettrica per circuiti in corrente continua (DC) e alternata (AC).
L'impedenza, indicata con Z, è misurata in **ohm** (Ω), è composta da due elementi:

- *Resistenza*(R): la parte dell'impedenza che oppone una resistenza costante alla corrente, 1. indipendentemente dalla frequenza;
- *Reattezza*(X): la parte dell'impedenza che varia con la frequenza e dipende da componenti come induttori e condensatori.

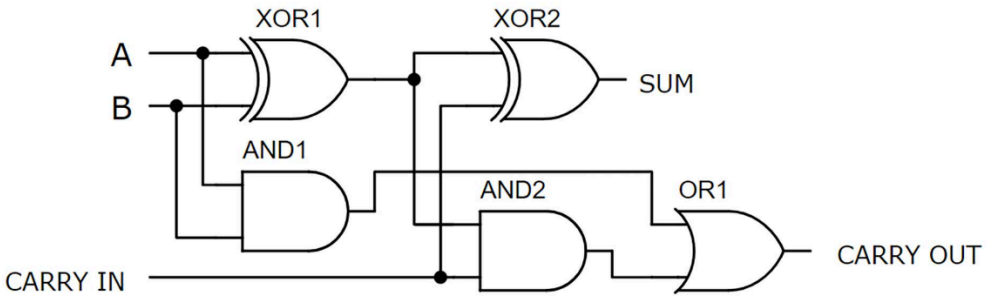
Half adder (HA)

Somma due bit in input, restituendo somma ed eventuale riporto.

A	B	Somma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full Adder (FA)

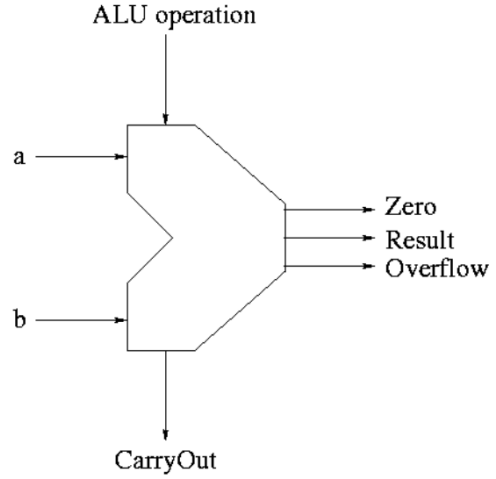


Realizzato con due *Half Adder*.
Un sommatore a n bit si può ottenere replicando in serie n volte un sommatore completo.

ALU

Arithmetic Logic Unit: - L'ALU è un circuito combinatorio che esegue operazioni aritmetiche e logiche su due operandi. Le operazioni possono essere selezionate tramite segnali di controllo.

- **Funzioni Principali**:
  - **Operazioni Aritmetiche**: Somma, sottrazione, moltiplicazione, divisione.
  - **Operazioni Logiche**: AND, OR, NOT, XOR.
  - **Output di FLAG**: Indicano informazioni sul risultato, come zero, overflow, e carry out.
- **Implementazione**:
  - Utilizzo di circuiti logici paralleli per ciascuna operazione.
  - Un **multiplexer** seleziona il risultato dell'operazione desiderata in base ai segnali di controllo.

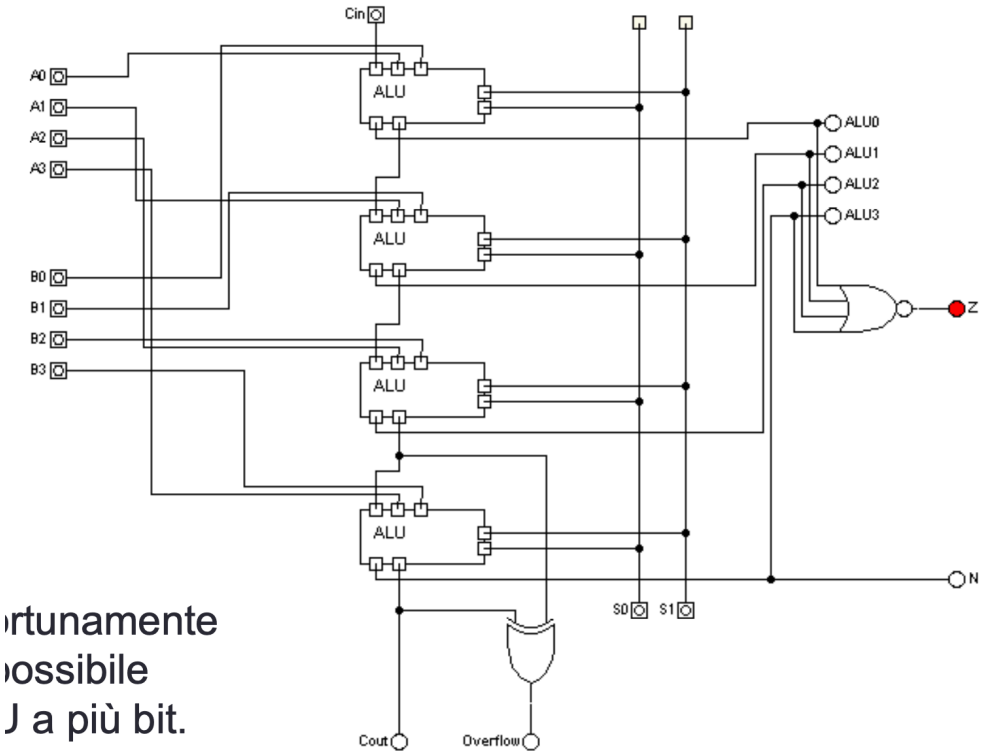


ALU a 1-bit

L'uscita dell'unità viene comandata da un multiplexer che seleziona l'operazione da compiere, a seconda dei segnali di controllo che gli vengono forniti.

- *AND* logico
- *OR* logico
- *Somma* (con/senza carry in ingresso)
- *Sottrazione* (con/senza carry in ingresso)
  Sottrazione $A - B = A + \text{complemento 2 di } B = A + B' + 1$
- Nego B e aggiungo 1 dal carry-in (solo quello iniziale del primo bit)

ALU a 4-bit



rtunamente
ossibile
J a più bit.

Sintesi di reti logiche sequenziali

Reti sequenziali

Reti logiche in cui in ogni istante le uscite (e il comportamento interno) dipendono dalla configurazione degli ingressi in quell'istante e dalle configurazioni degli ingressi negli istanti precedenti, quindi posseggono **memoria** nel proprio **stato** interno.

Aggiornamento dello Stato:

L'aggiornamento dello stato presente a quello futuro avviene in base alla logica del circuito e al tipo di rete sequenziale.

Tipi di Reti Sequenziali

1. **Reti Sequenziali Asincrone**:
   - **Definizione**: Le variazioni delle configurazioni di ingresso vengono sentite e possono modificare lo stato e le uscite in qualsiasi istante.
   - **Caratteristiche**:
     - Non richiedono un clock per aggiornare lo stato.
     - Possono rispondere immediatamente ai cambiamenti degli ingressi, rendendole più reattive ma potenzialmente instabili.
   - **Esempi**: Circuiti di controllo di flusso, sistemi di automazione.
2. **Reti Sequenziali Sincrone**:
   - **Definizione**: Le variazioni delle configurazioni di ingresso modificano lo stato e le uscite solo in presenza di un opportuno evento di sincronizzazione.

- **Caratteristiche**:
  - Utilizzano un segnale di clock per gestire l'aggiornamento dello stato.
  - L'uscita e il nuovo stato vengono calcolati e stabilizzati solo al fronte di un clock.
- **Esempi**: Flip-flop, registri, contatori.

Clock

Il **clock** è un segnale periodico utilizzato nei circuiti digitali e nei sistemi a microprocessore per sincronizzare e coordinare tutte le operazioni. Può essere visto come un "metronomo" che scandisce il ritmo, garantendo che tutti i componenti del sistema operino in armonia e nei tempi corretti.
Albero di clock: L'albero di clock è una struttura utilizzata nei circuiti digitali per *distribuire il segnale di clock* in modo uniforme a tutte le parti del sistema, minimizzando il *ritardo di propagazione* (*clock skew*). Garantisce che tutti i componenti operino in armonia.

Memoria binaria bistabile

Sono elementi fondamentali delle reti sequenziali, capaci di mantenere il valore 1 o 0, quindi un singolo bit di memoria.

Latch SR (Set-Reset)

è un circuito semplice che usa due porte logiche NAND o NOR per memorizzare un bit. Ha due ingressi:

- **Set (S)**: imposta l'uscita a 1.
- **Reset (R)**: imposta l'uscita a 0.
- **Quit (Q)** e **Complementary Quit (Q')**: rappresentano rispettivamente il valore del bit memorizzato e il suo complemento.

Sintesi a porte NOR

S (Set)	R (Reset)	Q (Uscita)	Q' (Complemento)
0	0	Mantiene	Mantiene
0	1	0	1
1	0	1	0
1	1	Indefinito	Indefinito

- $S = 0, R = 0$: Il latch mantiene il valore precedente.
- $S = 0, R = 1$: L'uscita Q viene resettata a 0.
- $S = 1, R = 0$: L'uscita Q viene impostata a 1.
- $S = 1, R = 1$: Condizione proibita o indefinita, poiché porta a un conflitto logico in cui sia Q che Q' potrebbero essere 0, violando il principio che Q e Q' devono sempre essere opposti.

Sintesi a porte NAND

S (Set)	R (Reset)	Q (Uscita)	Q' (Complemento)
1	1	Mantiene	Mantiene
1	0	1	0
0	1	0	1
0	0	Indefinito	Indefinito

- $S = 1, R = 1$: Il latch mantiene il valore precedente.
- $S = 1, R = 0$: L'uscita Q viene resettata a 0.
- $S = 0, R = 1$: L'uscita Q viene impostata a 1.
- $S = 0, R = 0$: Condizione proibita o indefinita.

Latch S-R sensibile a livello (del clock)

Risponde agli ingressi **Set (S)** e **Reset (R)** solo quando il segnale di controllo o **clock (CLK)** si trova in uno stato specifico, ossia **livello alto** (1) o **livello basso** (0). Questo latch è detto "sensibile a livello" perché monitora e reagisce agli ingressi **S** e **R** finché il clock rimane al livello attivo.

CLK (Livello)	S (Set)	R (Reset)	Q (Uscita)	Q' (Complemento)
0	X	X	Mantiene	Mantiene
1	0	0	Mantiene	Mantiene
1	0	1	0	1
1	1	0	1	0
1	1	1	Indefinito	Indefinito

D Latch

Memorizza un bit di informazione in base al valore dell'ingresso **D** (Data) quando il segnale di controllo o **clock (CLK)** è nel livello attivo (solitamente alto). Il D Latch è molto utile per evitare stati ambigui, come avviene invece nel latch S-R, grazie alla sua struttura che ha solo un ingresso dati.

CLK (Clock)	D (Data)	Q (Uscita)	Q' (Complemento)
0	X	Mantiene	Mantiene
1	0	0	1
1	1	1	0

- $CLK = 0$: il latch mantiene il valore precedente di Q.
- $CLK = 1$: il latch "segue" il valore di D, cioè $Q = D$.

Flip-Flop D

Derivato dal *D-Latch* è il flip flop più usato per memorizzare dei segnali il cui valore è significativo – e quindi deve essere campionato – solo in un dato istante.

CLK (Clock)	D (Data)	Q (Uscita)	Q' (Complemento)
↑	0	0	1
↑	1	1	0

- ↑: indica un fronte di salita del clock.
- D: valore che viene memorizzato su Q al fronte di salita di CLK.

Flip-Flop e Latch

Latch

Un **latch** è un dispositivo bistabile sincrono trasparente, in grado di memorizzare o meno segnali di ingresso in base a un segnale di abilitazione (clock o enable).
Funzionamento:

- La transizione di stato avviene mentre il clock è attivo (alto).
- Il latch è trasparente agli ingressi quando l'enable è attivo, quindi ogni cambiamento negli ingressi si riflette nell'uscita durante questo periodo.

Flip-Flop

Un **flip-flop** è un dispositivo bistabile che non possiede la proprietà di trasparenza.
Funzionamento:

- Il cambiamento dell'uscita non dipende direttamente dagli ingressi, ma è il risultato di un cambiamento (edge-triggered) di un ingresso di controllo sincrono (clock) o asincrono (preset o clear).
- I flip-flop si definiscono bistabili sincroni a commutazione sul fronte, poiché la transizione di stato avviene solo in corrispondenza di un evento significativo del clock (fronte di salita o di discesa), in base agli ingressi in quel momento.

Flip flop master/slave

È una configurazione che utilizza due flip-flop (master e slave) per garantire la sincronizzazione delle operazioni e prevenire problemi di instabilità nel circuito. Questa architettura è particolarmente utile nei sistemi digitali per gestire i dati in modo sicuro e affidabile.

- **Master Flip-Flop**: Riceve i dati in ingresso e li memorizza quando il segnale di clock è attivo (alto). Durante questo tempo, il master è sensibile ai cambiamenti degli ingressi.
- **Slave Flip-Flop**: Memorizza l'uscita del master. Il slave si aggiorna solo quando il segnale di clock è inattivo (basso), quindi è insensibile alle variazioni degli ingressi durante questo

intervallo.

Flip-Flop JK

È un'estensione del flip-flop SR e presenta un comportamento più versatile, eliminando la condizione indeterminata presente nel flip-flop SR quando entrambi gli ingressi sono attivi.

J	K	Q (Next State)	Descrizione
0	0	Q	Mantieni stato
0	1	0	Reset (Q = 0)
1	0	1	Set (Q = 1)
1	1	Q'	Toggle (inverti stato)

Applicazioni

- **Contatori**: I flip-flop JK possono essere utilizzati per costruire contatori binari.
- **Circuiti di Memoria**: Utilizzati nei registri di memoria per memorizzare bit di dati.
- **Circuiti di Stato**: Utilizzati nei sistemi sequenziali per gestire stati e transizioni.

Flip-Flop T (Toggle)

È un tipo di flip-flop bistabile che cambia stato (toggle) ogni volta che riceve un impulso sul segnale di clock, se l'ingresso T è attivo. È particolarmente utile nei circuiti di conteggio e in altre applicazioni che richiedono una semplice alternanza tra due stati.

T	Q (next state)	Descrizione
0	Q	Mantieni stato
1	Q'	Toggle (inverti stato)

Applicazioni

- **Contatori**: Utilizzato per costruire contatori binari a 1 bit. Collegando più flip-flop T in cascata, è possibile realizzare contatori che incrementano di 1 ad ogni impulso di clock.

Quando usarli?

- **S-R latch** sono poco usati come blocchi funzionali (e comunque all'interno dei JK e D).
- **Flip Flop T** sono molto usati (realizzati con JK o D) all'interno dei contatori o per ricordarsi l'evoluzione di un contesto interno al sistema di elaborazione in due stati possibili.
- **Flip Flop JK** e **D** sono entrambi i più usati: con JK si realizzano funzioni più complesse con meno logica esterna, ma richiedono più pin. In VLSI si usano più i D (componenti base della memoria).

Registri

Elemento di memoria in cui n flip-flop vengono controllati dallo stesso clock, formando una unità in grado di memorizzare parole composte da n bit.

Caratteristiche Principali

- **Composizione**: Un registro è formato da nn flip-flop, dove ogni flip-flop memorizza un singolo bit. Pertanto, un registro a nn bit può memorizzare parole di nn bit.
- **Controllo Sincronizzato**: Tutti i flip-flop all'interno di un registro vengono controllati da un segnale di clock comune, garantendo che le operazioni di scrittura e lettura siano sincronizzate.
- **Segnali di Controllo**:
  - **Input Enable (o Chip Select, CS)**: Questo segnale consente di attivare la fase di memorizzazione. Quando CS è attivo, il registro è in grado di ricevere dati e memorizzarli.
  - **Output Enable**: Questo segnale rende visibile l'uscita della parola memorizzata. Se l'output enable è attivo, il contenuto del registro viene presentato all'uscita.

Funzionamento dei Registri

1. **Memorizzazione dei Dati**: Quando il segnale di Input Enable è attivo e viene fornito un dato in ingresso, i flip-flop del registro memorizzano il dato alla prossima transizione del clock.
2. **Lettura dei Dati**: Quando il segnale di Output Enable è attivo, il contenuto del registro è visibile all'uscita. Questo permette ad altri circuiti di leggere i dati memorizzati.

Reti asincrone e sincrone

Nella progettazione dei circuiti digitali, si predilige l'uso di **reti sincrone** rispetto alle **reti asincrone** per vari motivi legati alla stabilità, all'affidabilità e alla facilità di progettazione.

Reti Sincrone

Nelle reti sincrone, tutte le operazioni sono coordinate da un segnale di clock comune. Questo segnale fornisce un riferimento temporale uniforme per le transizioni di stato.

- **Vantaggi**:
  - **Coordinamento**: La sincronizzazione riduce il rischio di conflitti e incertezze nei dati, rendendo le operazioni più prevedibili.
  - **Progettazione Semplificata**: I designer possono pianificare il comportamento del circuito basandosi su un ciclo di clock ben definito.
  - **Resistenza alle Alee**: Le reti sincrone sono meno sensibili alle alee, o corse critiche, che possono causare risultati errati.
- **Componenti**: Nelle reti sincrone, componenti come i **flip-flop D** sono frequentemente utilizzati. Questi dispositivi memorizzano lo stato dell'ingresso al fronte del clock, garantendo che il cambiamento di stato avvenga in modo controllato.

- **Segnali Asincroni**: Anche se sono sincrone, possono includere segnali asincroni come **clear** e **preset**, che possono forzare il flip-flop a uno stato specifico immediatamente, indipendentemente dal clock.

Reti Asincrone

Definizione: Le reti asincrone non dipendono da un segnale di clock comune. Le transizioni di stato possono avvenire in qualsiasi momento, in base ai cambiamenti degli ingressi.

- **Svantaggi**:
  - **Sensibilità alle Alee**: Le reti asincrone possono essere soggette a problemi di corsa critica, dove segnali diversi possono arrivare a componenti critici in momenti diversi, portando a risultati inaffidabili.
  - **Progettazione Complessa**: La mancanza di un riferimento temporale uniforme rende la progettazione e la verifica più difficili.
  - **Difficoltà di Sincronizzazione**: Risulta complicato garantire che tutti i componenti reagiscano simultaneamente a un cambiamento degli ingressi.

Automa a stati finiti

Automata di Mealy

- **Definizione**: Uscite dipendono dallo stato corrente e dagli ingressi attuali.
- **Caratteristiche**:
  - Uscite cambiano immediatamente in risposta a variazioni negli ingressi.
  - Rappresentazione delle uscite associata alle transizioni.
- **Vantaggi**:
  - Maggiore reattività agli ingressi.
- **Esempio**: Controllo di porte che si apre/chiude in base al pulsante premuto.

Automata di Moore

- **Definizione**: Uscite dipendono solo dallo stato corrente.
- **Caratteristiche**:
  - Uscite cambiano solo con un cambiamento di stato.
  - Rappresentazione delle uscite associata agli stati.
- **Vantaggi**:
  - Maggiore stabilità nelle uscite.
- **Esempio**: Semaforo che mostra il colore in base allo stato attuale (rosso, verde, giallo).