**CAPITOLO 02**

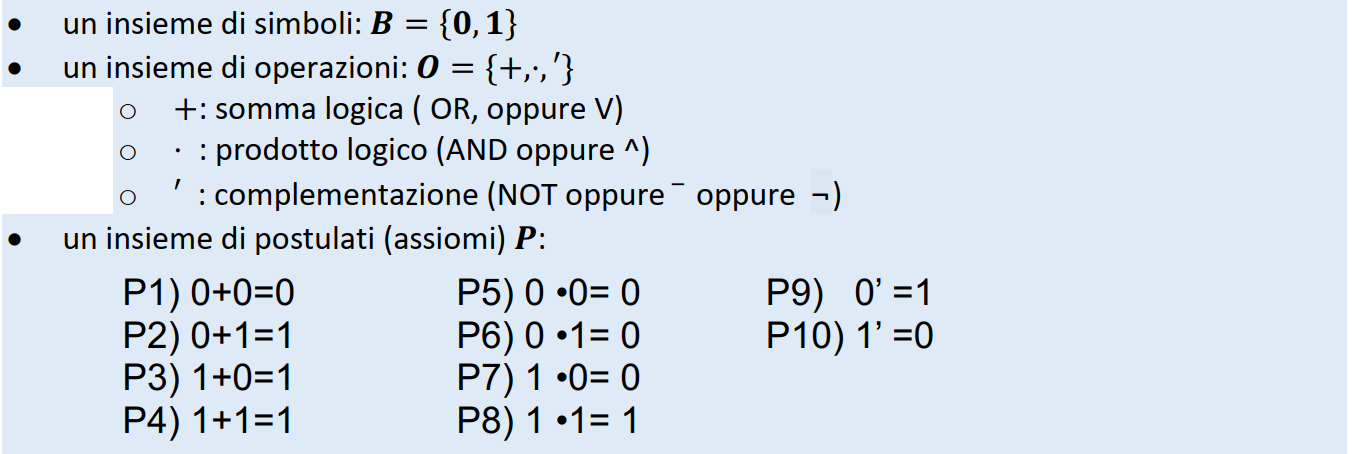
**ALGEBRA DI BOOLE O DI COMMUTAZIONE**

L’algebra di Boole fu introdotta nel 1847 dall’omonimo matematico e logico George Boole per fornire una rappresentazione matematica della logica. Attraverso l’algebra di Boole possiamo effettuare una progettazione logica con una descrizione strutturale (conseguentemente anche funzionale) indipendentemente dalla descrizione fisica. Fu poi applicata nel 1936 da Claude Shannon allo studio delle reti di commutazione telefonica

Fondamentale per la **SINTESI DELL’HARDWARE** e fondamento dell’**INTELLIGENZA ARTIFICIALE.** Si lavora ad un livello **superiore** al livello fisico, **indipendentemente** che l’informazione venga trasmessa da una grandezza fisica, elettronica, meccanica etc. la logica di commutazione (sinonimo di *di Boole*) si basa sul fatto che esistono segnali trasmessi ed elaborati di tipo elettronico, ma **trasformati** in natura digitale come valori 1(vero) e 0(falso) (tipicamente).

Vi sono due tipi di logiche:

|  |  |  |
| --- | --- | --- |
|  | Logica Positiva | Logica Negativa |
| Alto - Vero | 1 | 0 |
| Basso - Falso | 0 | 1 |

**DEFINIZIONE ALGEBRA DI BOOLE:** sistema matematico che descrive funzioni di variabili binarie, si compone di:

**COSTANTI:** 0 e 1 **VARIABILI:** un qualsiasi simbolo che può essere sostituito da una delle due costanti

**ESPRESSIONI:** stringhe di elementi dell’insieme B che soddisfa una delle seguenti regole:

* Una **COSTANTE** è un’espressione;
* Una **VARIABILE** è un’espressione;
* Se X è un’espressione allora il **COMPLEMENTO** di Xè un’espressione;
* Se X ed Y sono espressioni allora la **SOMMA LOGICA** di X e Y è un’espressione;
* Se X ed Y sono espressioni allora il **PRODOTTO LOGICO** di X e Y è un’espressione.

**PROPRIETÀ DI CHIUSURA: ;**

**PRINCIPIO DI DUALITÀ:** ogni espressione ha una forma duale ottenuta scambiando + con ▪ (o viceversa) e la costante 0 con la costante 1 (o viceversa) mantenendo i letterali invariati. Inoltre, ogni proprietà valida per un’espressione vale per la sua duale.

**TEOREMI:**

* **IDENTITÀ: X + 0 = X X \* 1 = X**
* **ELEMENTI NULLI: X + 1 = 1 X \* 0 = 0**
* **IDEMPOTENZA: X + X = X X \* X = X**
* **INVOLUZIONE: (X’)’ = X**
* **COMPLEMENTARIETÀ: X + X’ = 1 X \* X’ = 0**
* **COMMUTATIVA: X + Y = Y + X X \* Y = Y \* X**
* **ASSOCIATIVA: (X + Y) + Z = X + (Y + Z) = (X \* Y) \* Z = X \* (Y \* Z) =**

**= X + Y + Z X \* Y \* Z**

* **ASSORBIMENTO: X + X \* Y = X X \* (X + Y) = X**
* **DISTRIBUTIVA: X \* Y + X \* Z = X \* (Y + Z) (X + Y) \* (X + Z) = X + (Y \* Z)**
* **COMBINAZIONE°: (X + Y) \* (X’ + Y) = Y X \* Y + X’ \* Y = Y**
* **CONSENSO: (X+Y) \* (X’+Z) \* (Y+Z) = X\* Y + X’\*Z + Y\*Z = X\*Y + X’ \*Z**

**(X+Y) \* (X’+Z)**

* **DE MORGAN: (X + Y)’ = X’ \* Y’ (X \* Y)’ = X’ + Y’**

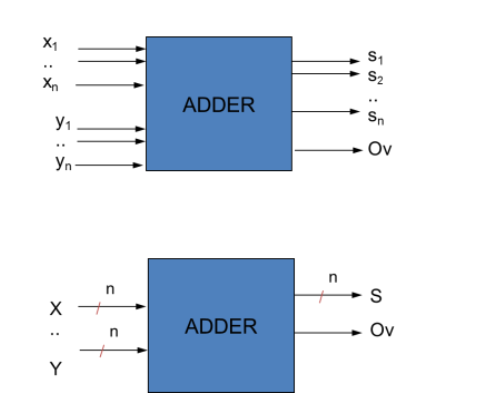
**LE RETI LOGICHE**

**LIVELLO DI RETI LOGICHE:** Livello base del modello del calcolatore, al di sopra del livello della pura elettronica. Implementa la microarchitettura; è il livello di astrazione che studia i sistemi digitali a livello di componenti logici elementari, detti **GATE**, indipendentemente dalla loro realizzazione fisica;

**RETE LOGICA:** rappresentazione astratta del modello matematico con cui si formalizza un sistema digitale; dal punto di vista puramente funzionale, è **l’astrazione di un sistema digitale** avente n segnali binari d’ingresso **X = {}**, m segnali binari di uscita **Z = {}**, ed una funzione logica *f* che le pone in relazione logica  **= *f (*)**;

**FUNZIONE LOGICA: *f (***  è una funzione logica che segue le leggi dell’algebra binaria di Boole. Più specificamente, la funzione logica booleana è un’applicazione su variabili booleane e si definisce come *f( … ,): {0,1}n →{0,1}*;

I segnali d’ingresso e di uscita possono essere singoli segnali binari oppure un insieme di segnali binari che formano delle **parole,** segnali digitali composti;

**DESCRIZIONE COMPORTAMENTALE DELL’ADDER**

**Rete Logica →**

**RTL →**

L’adder qui sopra può essere descritto a livello comportamentale a livello di:

- **Rete logica:** l’adder è una rete che ha 2n ingressi e somma n bit d’ingresso a n bit d’ingresso per creare n bit d’uscita ed un unico segnale OV di *overflow*;

- **RTL (R**egister **T**ransfer **L**evel): l’adder è una rete logica con due parole ad n bit in ingresso, X e Y, e con due uscite, una parola a n bit S e un OV ad 1 bit.

**LE PROPRIETÀ DELLE RETI LOGICHE**

**Interconnessione:** l’interconnessione di più reti logiche, aventi per **ingresso** segnali esterni o le uscite di altre reti logiche e per **uscite** segnali d’uscita esterni o ingressi di altre reti logiche, è ancora una rete logica;

**Decomposizione:** una rete logica complessa può essere decomposta in reti logiche più semplici, fino all’impiego di soli blocchi o gate elementari;

**Decomposizione in parallelo:** una rete logica ad m uscite può essere decomposta in m reti logiche a 1 uscita, aventi ingressi condivisi;

**RETI LOGICHE COMBINATORIE E SEQUENZIALI**

**RETI LOGICHE COMBINATORIE:** ogni segnale d’uscita dipende solo dai valori degli ingressi in un determinato istante di tempo t; ***f (***

**RETI LOGICHE SEQUENZIALI:** ogni segnale d’uscita dipende dai valori degli ingressi nell’istante di tempo t e dai valori che hanno assunto gli ingressi negli istanti precedenti. ***f (***

**DESCRIZIONE DELLE RETI LOGICHE**

Descrivere una rete logica (con qualsiasi dei metodi elencati di seguito) è importante per:

* L’analisi, per dare una descrizione comportamentale
* La documentazione, per rappresentare in modo standard e codificato cosa fa un sistema
* Per la sintesi, per generare le reti in modo automatico al calcolatore

I metodi di descrizione sono:

1. **Descrizione a parole:** descrizione a parole in linguaggio naturale del comportamento della rete logica, immediato e semplice, ma poco formale e raramente porta alla sintesi;
2. **Tabelle di verità:** descrizione esaustiva di tutte le configurazioni in uscita per ogni possibile configurazione d’ingresso. È la descrizione più diretta ed è il punto di partenza più semplice per ottenere la sintesi automatica
3. **Mappe di Karnaugh:** altra rappresentazione delle tabelle di verità, forma più compatta;
4. **Espressioni dell’algebra booleana:** metodo formale impiegato come ingresso da sistemi di sintesi automatica;
5. **Schema logico:** descrizione strutturale, utile per la documentazione;
6. **Forme d’onda:** descrizione comportamentale in funzione del tempo, tipico delle RL sequenziali;
7. **Linguaggi di descrizione dell’hardware:** impiego dei linguaggi procedurali e paralleli come il VHDL.

**TABELLE DI VERITÀ**

**Tabella di verità:** descrizione di tutte le configurazioni di uscita per ogni possibile configurazione d’ingresso; rappresentata da una tabella che associa ad ogni possibile combinazione degli ingressi alle corrispondenti configurazioni delle uscite. Ha n(ingressi) + m(uscite) colonne e righe (corrispondenti alle possibili configurazioni in entrata).

**Tabella di verità completamente specificata →** se ogni valore della tabella assume il valore logico di vero o falso (1, 0 dipendentemente dal tipo di logica)

**Tabella di verità non completamente specificata →** se sono presenti condizioni di indifferenza (“-“ o “\*”) nel caso in cui l’uscita non sia specificata nella descrizione. Può capitare se

1. Alcune configurazioni di ingressi sono vietate;
2. Se le uscite sono indifferenti per alcune configurazioni d’ingresso.

**MAPPE DI KARNAUGH**

Le mappe sono una rappresentazione matriciale o geometrica della tabella di verità. Specificamente, le mappe di Karnaugh sono mappe in cui le configurazioni successive in ogni lato sono **adiacenti,** ovvero la loro distanza di Hamming (differenza bit a bit) uguale a 1 (ex. 1010 e 1110 hanno distanza di Hamming = 1).

**SCHEMA LOGICO E FORME D’ONDA**

Lo **schema logico** è una descrizione strutturale che rappresenta come è fatta la rete logica in termini di gate elementari o di aggregati e moduli di gate elementari in RTL.

Le **forme d’onda** rappresentano il legame consequenziale e temporale tra ingressi ed uscite, poco importante nelle RL combinatorie ma molto più importante nelle RL sequenziali.

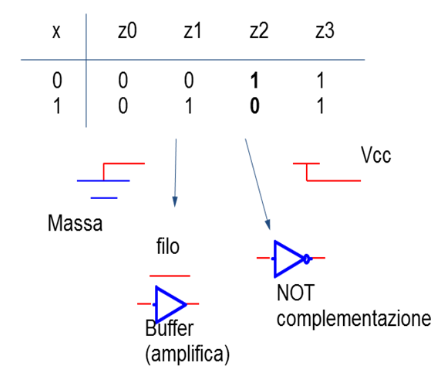
**FUNZIONI BINARIE E GATE ELEMENTARI**

Come detto precedentemente una tabella di verità di n variabili d’ingresso ha righe e conseguentemente dati n segnali sono possibili possibili reti logiche diverse.

**ATTENZIONE! se abbiamo n = 4 il numero di possibili reti logiche sarà .**

**FUNZIONI DI UNA VARIABILE INDIPENDENTE**

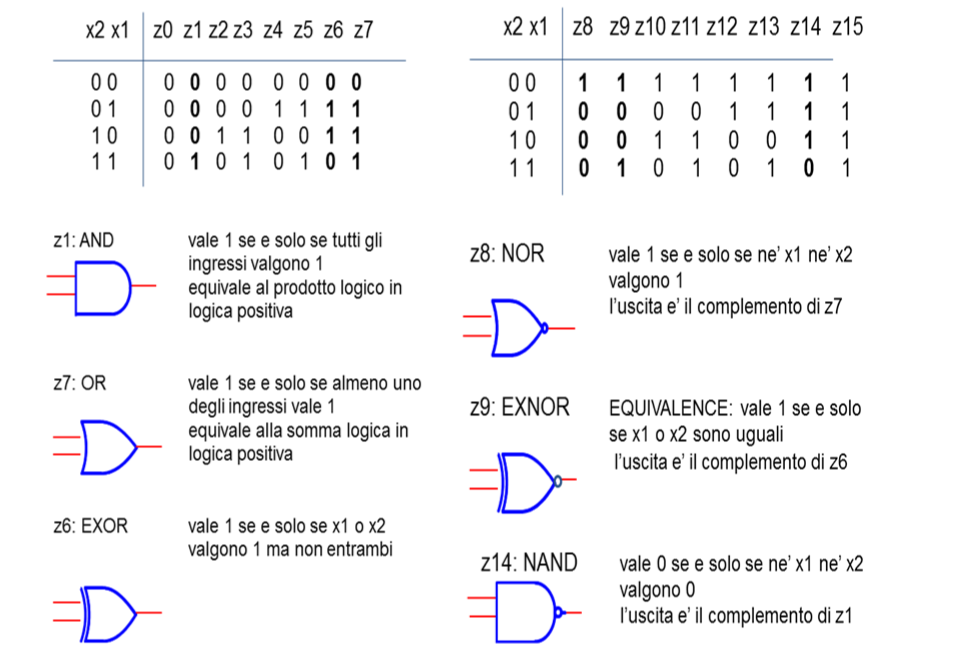
Data una tabella di verità con tutti i possibili ingressi e uscite di una sola variabile indipendente (2 ingressi, 4 uscite).



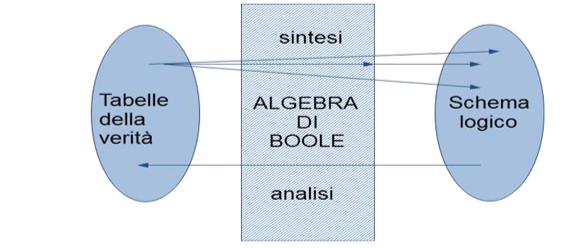
Otteniamo 4 funzioni logiche:

* Z0 → **massa**, qualsiasi sia il valore d’ingresso, l’uscita sarà sempre 0;
* Z1 → **filo conduttore**, qualsiasi sia il valore d’ingresso, l’uscita non cambia (in elettronica implica una funzione di amplificazione);
* Z2 → **invertitore o NOT**, qualsiasi sia il valore d’ingresso, fornisce il suo complemento (due NOT in cascata non cambiano il valore dell’ingresso);
* Z3 → **alimentazione**, qualsiasi sia il valore d’ingresso, l’uscita sarà sempre 1.

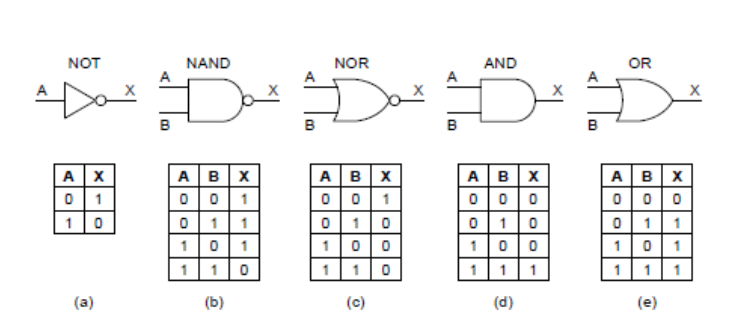
**FUNZIONI DI DUE VARIABILI INDIPENDENTI**



* **AND** equivale al **prodotto logico;**
* **OR** equivale alla **somma logica;**
* **EXOR** equivale alla disuguaglianza.

Queste 3 funzioni e le loro complementari sono dette **gate elementari.** Vi è una **diretta corrispondenza** tra gate elementari e l’algebra di Boole, i primi sono infatti gli elementi base delle reti logiche e sono stati progettati per corrispondere alle funzioni binarie più comuni → **TUTTE LE FUNZIONI LOGICHE RAPPRESENTABILI DALL’ALGEBRA DI BOOLE SI POSSONO RAPPRESENTARE CON LE RETI LOGICHE**.

A partire da una tabella di verità possiamo ottenere diversi possibili schemi logici attraverso la sintesi, ma a partire dallo schema logico della rete, con l’analisi, possiamo ottenere una sola possibile tabella di verità

Dei due insiemi di gate che abbiamo visto i più utilizzati sono:

**ANALISI DI UNA RETE LOGICA**

1. Nominare le uscite dei gate logici;
2. Sostituire i gate elementari, a partire dalle uscite, con le corrispondenti espressioni logiche;
3. Semplificare con i teoremi dell’algebra di Boole;
4. Valutare l’espressione e ottenerne la tabella di verità.

**SINTESI DI UNA RETE LOGICA COMBINATORIA**

Fare la sintesi di una RL a partire dalla sua descrizione comportamentale significa **progettarla.**

Vi sono due modi per eseguire la sintesi di una RL combinatoria:

* **Sintesi minima:** sintesi che mira ad avere il minor numero di gate elementari possibili, difficilmente leggibile e molto sensibile a errori dovuti al ritardo di trasmissione;
* **Sintesi canonica:** sintesi facilmente leggibile e comprensibile, ma con un maggior numero di gate rispetto alla sintesi minima.

**DEFINIZIONI LETTERALI...**

Prima di parlare di sintesi canonica bisogna introdurre delle definizioni.

**LETTERALE:** ogni occorrenza di una singola variabile, sia in forma semplice che complementata;

**IMPLICANTE:** data un prodotto P(x’) x’ ⊆ X di letterali t.c. se P = 1 => *f* = 1;

**IMPLICANTE PRIMO:** implicante di *f* che non è coperto da altri implicanti con meno letterali;

**IMPLICANTE PRIMO ESSENZIALE:** implicante comprendente almeno un mintermine che non è coperto da nessun altro implicante primo;

**MINTERMINE:** implicante in cui compaiono tutti i letterali;

**IMPLICATO:** somma di letterali che impongono 0 alla funzione;

**MAXTERMINE:** data un maxtermine è un punto Bx dello spazio booleano tale per cui *f*(Bx)=0.

**CONCETTO DI COPERTURA DI UN LETTERALE:**

Date due funzioni f(x1,x2 ,..,xn ) e g(x1,x2,..,xn) si dice che f copre g (oppure g implica f) e si scrive f ⊃g se f(x1,x2,..,xn) =1 quando g(x1,x2,..,xn) =1; Se P è il prodotto di letterali e f copre P, si dice che P è un implicante di f.

**SINTESI CANONICA**

È realizzata attraverso i mintermini o i maxtermini e si ottiene osservando ogni riga della tabella di verità. Esistono due tipi di sintesi canonica:

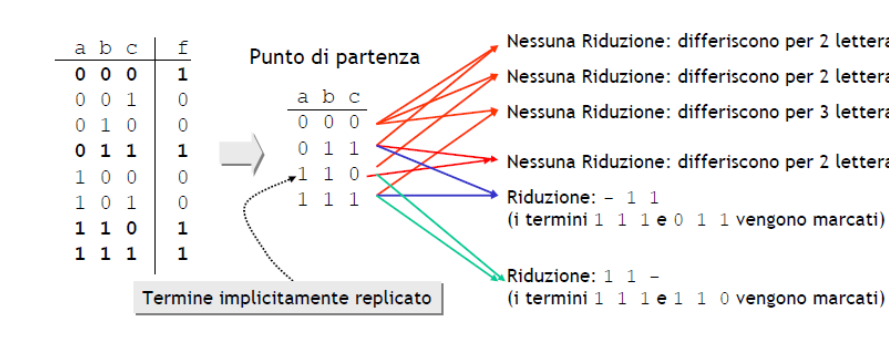
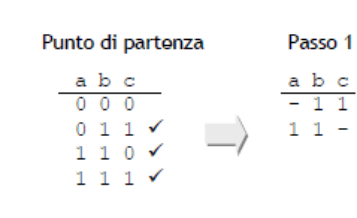
* **SP,** somma di prodotti, è definita come la somma logica dei mintermini associati alle righe della tabella nelle quali l’uscita assume il **valore 1**;
* **PS,** prodotto di somme, è definita come il prodotto logico dei maxtermini associati alle righe della tabella nelle quali l’uscita assume il **valore 0**.

La **sintesi canonica** non è **minima**, ma può essere sintetizzata attraverso appositi algoritmi (**Quine McCluskey**), il vantaggio di questa è di garantire la massima velocità in quanto impiega solo due livelli di gate S e P o P e S (dipendentemente dal tipo di sintesi).

**SINTESI MINIMA – ALGORITMO DI QUINE McCLUSKEY**

L’algoritmo di Quine McCluskey è il metodo più impiegato per la sintesi minima di una rete canonica di un qualsiasi numero di ingressi. Vi sono principalmente due passi:

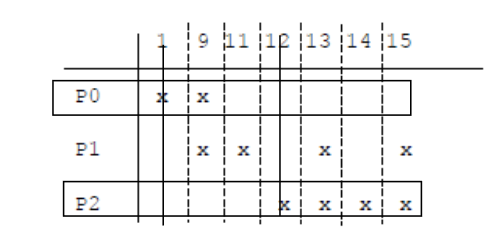
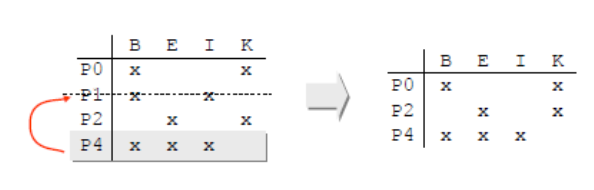
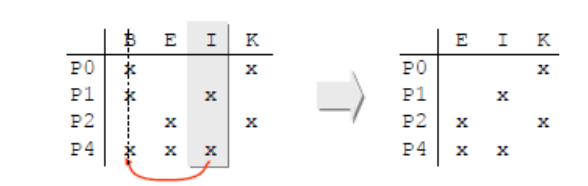
1. Ricerca degli implicanti primi:

* Mettere in una tabella tutti i mintermini in forma binaria, e in ordine crescente (ergo tutte le combinazioni che danno 1, ma anche indifferenza);
* Dividiamo i mintermini in sezioni in base al loro numero di uni;
* Applichiamo la proprietà di combinazione tra i mintermini di una sezione e quella successiva. ATTENZIONE! La proprietà di combinazione di può applicare solo se le due parole hanno distanza di Hamming **unitaria**;
* Si crea così una nuova tabella con tutti i termini di prodotto che derivano dalla semplificazione della prima tabella (dove le due parole sono diverse, porremo un’indifferenza) e ripetiamo il passaggio precedente;
* Terminiamo quando non si possono più fare riduzioni.

1. Copertura:

Non tutti gli implicanti primi trovati dal primo passo sono anche essenziali. Per trovare gli implicanti primi essenziali dobbiamo ricorrere alla **tabella di copertura**:

* Formiamo la tabella in cui sulle righe metteremo gli implicanti primi che abbiamo trovato con il passo precedente, mentre sulle colonne metteremo i valori corrispondenti agli implicanti che abbiamo preso in considerazione nel primo passo (solo quelli a cui corrisponde un 1);
* Applichiamo i criteri di semplificazione:

1. **Criterio di essenzialità:** se un valore è coperto da un solo implicante, allora quell’implicante sarà essenziale;
2. **Criterio di dominanza di riga:** un implicante Pi domina un implicante Pj se copre almeno tutti i mintermini di Pj;
3. **Criterio di dominanza tra colonne:** un mintermine Mi domina un mintermine Mj se ogni implicante che copre Mj copre anche Mi.

**RETI LOGICHE COMBINATORIE SEMPLICI**

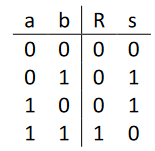
**SOMMATORE (HALF ADDER)**

L ’half adder è un esempio di rete logica combinatoria semplice.

Per eseguire qualsiasi somma in ogni base bisogna sapere il risultato della somma tra due cifre.

La somma binaria ha solo quattro casistiche:

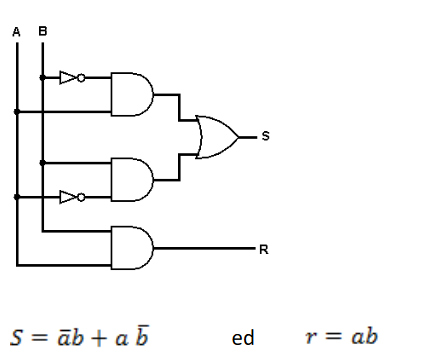
0 + 0 = 0 (con riporto 0)

0 + 1 = 1 (con riporto 0)

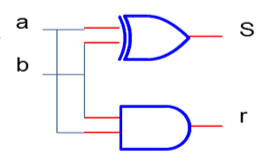
1 + 0 = 1 (con riporto 0)

1 + 1 = 0 (con riporto 1)

In tabella di verità sarebbe così →



che porta al seguente schema logico (tramite sintesi canonica SP) →

oppure, impiegando la logica booleana notiamo che il riporto non è altro che l’AND tra **a** e **b** mentre il risultato non è altro che lo XOR di **a** e **b** →

**RETI LOGICHE A LIVELLO RTL**

La sintesi a livello RTL è la sintesi di blocchi logici elementari (ma più complessi dei singoli gate) con cui realizzare per composizione e interconnessione reti logiche e sistemi più complessi.

Tra i più comuni vi troviamo:

- Multiplexer & Demultiplexer;

- Amplificatore Tri-State;  
- Comparatore;

- ALU.

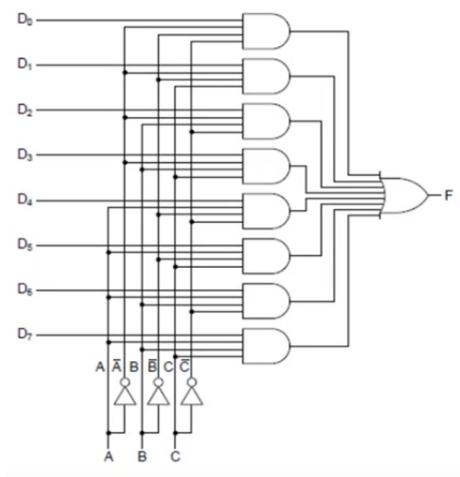
**MULTIPLEXER**

Rete logica avente:

* ingressi di tipo dato;
* ingressi di selezione;
* uscita.

**Funzionamento →** il multiplexer darà in uscita l’ingresso corrispondente alla combinazione degli ingressi di selezione

Ex. Multiplexer a 4 ingressi, se la combinazione degli ingressi di selezione è 10, l’ingresso selezionato sarà quello corrispondente a 2;

Ex. MUX 8 ingressi

**DEMULTIPLEXER**

Rete logica avente:

- ingresso;

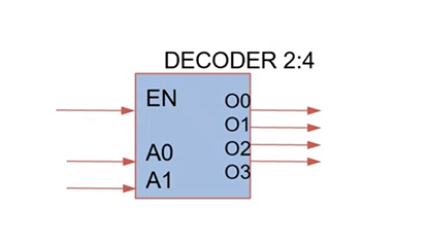
- ingressi di selezione;

- uscite.

**Funzionamento →** accetta un segnale di ingresso e lo trasferisce in una delle possibili **2^n** uscite in base agli **n** ingressi di selezione. In questo caso abbiamo un ingresso EN (**enable**) che se fisso a 1 attiva il decoder.

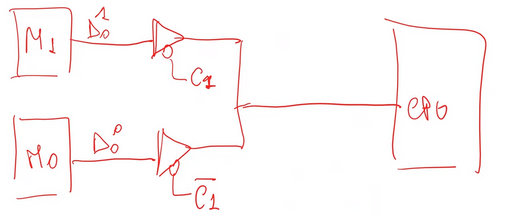
Il demux prende i segnali di controllo (A0, A1) e, in base alla loro configurazione, fornisce il valore 1 in uscita.

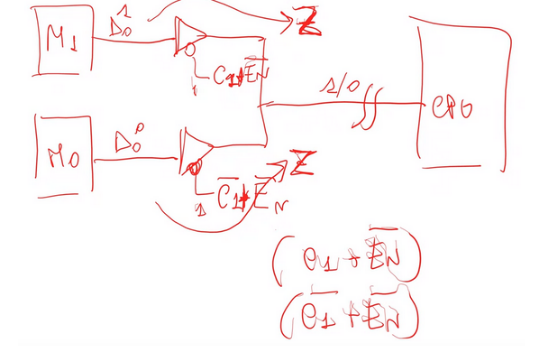
Ex. Se ci fossero, in entrata, 00, in uscita ci sarebbe 0001, per indicare la prima uscita, mentre se ci fosse 11, ci sarebbe 1000, per indicare l’ultima uscita.



**AMPLIFICATORE TRI-STATE**

L’amplificatore tri-state è una rete logica speciale che va al di là dell’algebra di Boole. È un blocco che ha una particolare uscita, un particolare ingresso e un segnale di attivazione attivo basso (ergo in logica negativa). Quando l’output enable è a 0, allora vuol dire che l’input passa normalmente, come se fosse un circuito chiuso. Se OE è 1, allora è come se ci fosse un circuito aperto e non passa nulla in uscita. Questo circuito è molto utile, poiché può essere usato come un multiplexer, che mi permette di mettere sul bus dei dati solo alcuni segnali.

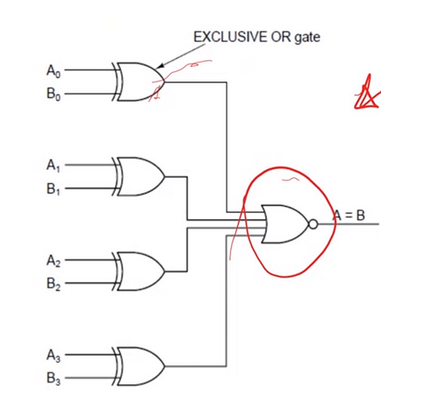
Ex. Ho due blocchi di memoria M0 e M1 e voglio che solo uno dei due raggiunga la CPU, tramite l’amplificatore tri-state, collegato per una memoria a C1 e per un’altra a C1’ mi permette di far passare solo i dati da una delle due memorie alla volta. 



Ma se a questo punto collego un enable globale (di circuito) a C1 e C1’, l’espressione degli ingressi di controllo cambia, diventando:

(C1 + E’), (C1’ + E’).

Se, quindi, E = 0, C1 e C1’ vanno a 1, pertanto su entrambe le uscite non passa nulla (alta impedenza **Z**), pertanto il circuito è completamente staccato.

**COMPARATORE**

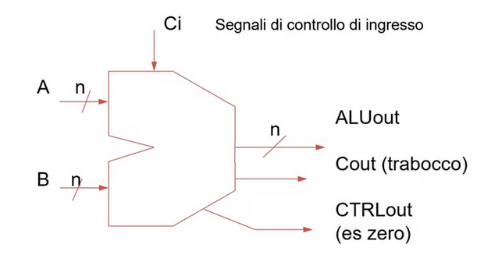
Il comparatore è utile quando bisogna testare condizioni sui nostri dati.

Per esempio, come facciamo a capire se A = B?

“Booleanamente” il problema è molto semplice, dato che lo XOR va esattamente questo (a 2 bit).

Se la parola è composta da più di due bit (n bit), basterà semplicemente collegare tanti XOR quanti bit e collegare i bit delle due parole a ciascuno di questi XOR a due a due.

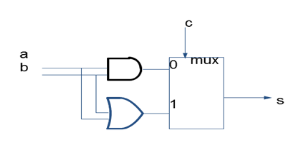
**ALU**

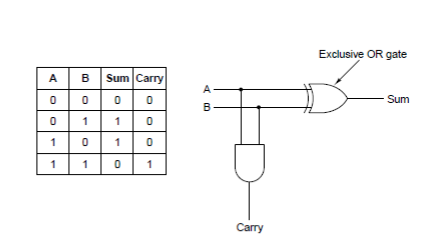
Una delle reti logiche combinatorie più importati nelle CPU è la ALU (arithmetic logic unit), ossia l’unità che in base a segnali di controllo dell’istruzione esegue un’operazione aritmetica o logica tra gli operandi.

Nell’immagine:

* **A** e **B** sono ingressi (parole a n bit);
* **ALUout** è l’output della ALU (sempre a n bit);
* **Cout** è il riporto;
* **CTRLout** è l’uscita di controllo;
* **Ci** impone segnali di controllo in ingresso, in base all’istruzione passata dalla CPU.

All’interno la ALU è decomposta in tante piccole reti logiche semplici per ogni funzione. Ad esempio, per ottenere l’AND e l’OR logico avremo uno schema simile a questo



Mentre per la somma avremo una rete logica un po’ più complessa. 

Partendo dall’ half adder che

somma i due bit meno

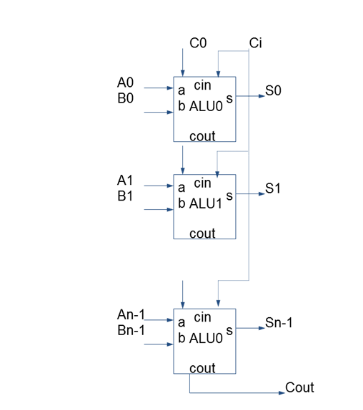
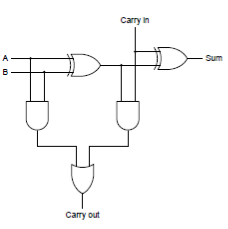
significativi di una parola

il risultato della mia somma

andrà in uscita, mentre,

in cascata ad altri half adder propagherò il resto

ottenendo dunque un **full adder**.

Mettendo più **full adder** in cascata (tanti quanti sono i bit da sommare) otterremo un sommatore 

che propaga il riporto.

