# RETI LOGICHE

Le reti logiche corrispondono al livello di digital design del calcolatore:
livello di astrazione che studia i sistemi digitali a livello di componenti logici elementari, chiamati gate indipendentemente dalla loro realizzazione fisica = livello di reti logiche.

- **Punto di vista funzionale:** una rete logica *è una funzione logica* → è astrazione di un sistema digitale che *ha un certo n° N di segnali binari di ingresso e un n° M di segnali binari di uscita, e li mette in relazione.*
  *LA RERE LOGICA È UN MODELLO MATEMATICO*

Lavorando a livello di reti logiche posso considerare i segnali in modo separato o usarli come parole binarie

**TIPOLOGIA RETI LOGICHE:**

a) **Combinatorie:** ogni *segnale di uscita dipende SOLO dai valori degli ingressi in quell'istante*

$$Zi\ (t) = f\ (X1(t)\ ,\ldots,\ XN(t))$$

b) **Sequenziali:** *ogni segnale di uscita dipende dal valore degli ingressi in quell'istante e* dal valore che gli ingressi hanno assunto *negli istanti precedenti* → è una rete dotata di memoria

$$Zi\ (t) = f\ (X1(t)\ ,\ldots,\ XN(t)\ ,\ t)$$

- ***Asincrone o sincrone,*** cioè reti logiche che hanno un segnale di clock (segnale di sincronismo per gestire la memoria)

**PROPRIETA' RETI LOGICHE:**

1) **Proprietà di interconnessione:** interconnettendo più reti logiche aventi per ingresso segnali esterni o segnali d'uscita di altre reti logiche, ottengo sempre una rete logica

2) **Proprietà di decomposizione:** rete logica complessa può essere decomposta in reti logiche più semplici (fino ai gate)

3) **Proprietà di decomposizione in parallelo:** rete logica ad M uscite può essere decomposta in M reti logiche ad un'uscita, con ingressi condivisi

## DESCRIZIONE COMPORTAMENTALE RETI LOGICHE:

1) Descrizione a parole o linguaggio naturale
2) Descrizione a forme d'onda: come si evolvono i segnali nel tempo e gli ingressi
3) **Descrizione con tabelle di verità:** ***è la descrizione di tutte le configurazioni di uscita per ogni possibile configurazione di ingresso***

***Righe = configurazioni in ingresso (2^n)***
***Colonne = n° uscite***

**2 tipologie:**

a) **Completamente specificata:** se ogni valore della tabella assume il valore logico di vero o falso
b) **Non completamente specificata:** se esistono alcune configurazioni di ingressi che sono vietate oppure se le uscite sono assolutamente indifferenti per alcune configurazioni in ingresso

***OSS: ogni tabella della verità può essere rappresentata tramite una k-map***

4) **Descrizione con** mappe di Karnaough o **K-map**: tabella della verità in modo matriciale dove TUTTE le configurazioni in ingresso sono nelle righe o nelle colonne
   → **concetto di adiacenza legato alla distanza di Hamming:** *numero di bit di differenza tra due configurazioni binarie espresse bit a bit*

***Bit di parità*** = ci viene dato in input dal sistema e viene messo per rendere pari il n° di 1

5) **Descrizione con algebra di Boole:** sistema matematico chiuso che descrive funzioni di variabili binarie.

- **Simboli: B= {0,1}**
- **Operazioni: O= {+, *, '}**
- **Postulati:**

| Della somma logica | Del prodotto logico | Complementazione |
|---|---|---|
| *0+0=0* | *0*0=0* | *0'=1* |
| *0+1=1* | *0*1=0* | *1'=0* |
| *1+0=1* | *1*0=0* | |
| *1+1=1* | *1*1=1* | |

# TUTTE LE FUNZIONI LOGICHE RAPPRESENTABILI CON ALGEBRA DI BOOLE SI POSSONO RAPPRESENTARE CON RETI LOGICHE

- **Espressione per Boole:** qualsiasi stringa di elementi che soddisfa le seguenti regole:
  a) *Costante = espressione*
  b) *Variabile = espressione*
  c) *Se X è un'espressione → complemento è un'espressione*
  d) *Se X e Y sono espressioni → prodotto logico è un'espressione*
  e) *Se X e Y sono espressioni → somma logica è un'espressione*

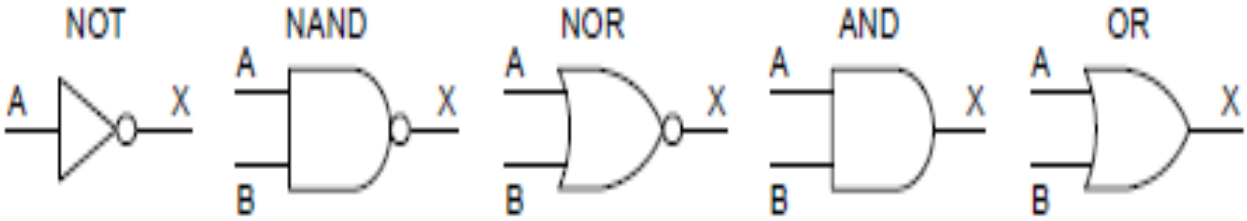
## Quante sono le possibili funzioni binarie (reti logiche) di N variabili indipendenti?

Con due ingressi, gli ingressi possono assumere 4 configurazioni, quindi si possono costruire 16 funzioni. Delle 16 possibili funzioni logiche di 2 variabili indipendenti, solo 6 meritano di essere nominate. 3 hanno un corrispondente diretto nelle espressioni logiche:

- **AND** (prodotto logico): l'uscita è vera solo se entrambi i termini sono veri.
- **OR** (somma logica): l'uscita è vera se almeno uno dei termini è vero.
- **EXOR** (disuguaglianza): l'uscita è vera se uno solo dei termini è vero, ma non entrambi.

*Le altre 3 funzioni sono il complemento (negato) di queste tre: NAND, NOR e EXNOR.*

AND, OR, NOT assieme a EXOR, NOR, NAND e EXNOR sono detti "**gate elementari**" delle reti logiche e corrispondono direttamente all'algebra booleana.



# TEOREMI DELL'ALGEBRA DI BOOLE

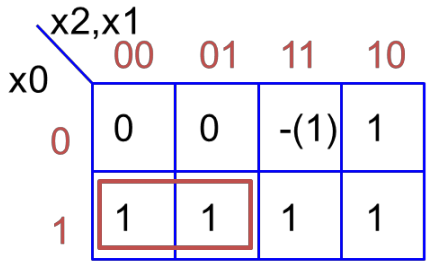
| | | | |
|---|---|---|---|
| Identità | (T1) X + 0 = X<br>(T1') X · 1 = X | Associativa | (T7) (X + Y) + Z = X + (Y + Z) = X + Y + Z<br>(T7') (X · Y) · Z = X · (Y · Z) = X · Y · Z |
| Elementi nulli | (T2) X + 1 = 1<br>(T2') X · 0 = 0 | Assorbimento | (T8) X + X · Y = X<br>(T8') X · (X + Y) = X |
| Idempotenza | (T3) X + X = X<br>(T3') X · X = X | Distribuzione | (T9) X · Y + X · Z = X · (Y + Z)<br>(T9') (X + Y) ·(X + Z) = X + Y · Z |
| Involuzione | (T4) ( X ')' = X | Combinazione | (T10) (X + Y) · ( X'+ Y) = Y<br>(T10') X · Y + X' · Y = Y |
| Complementarietà | (T5) X + X' = 1<br>(T5') X · X' = 0 | Consenso | (T11) (X + Y) ·( X' + Z) ·(Y + Z) = (X + Y)·(X'+Z)<br>(T11') X · Y + X' · Z + Y · Z = X · Y + X' · Z |
| Commutativa | (T6) X + Y = Y + X<br>(T6') X · Y = Y · X | De Morgan | (T12) (X + Y)' = X' · Y'<br>(T12') (X · Y)' = X' + Y' |

**FORMA MINIMA RETE LOGICA:**

non può essere ulteriormente semplificata.

La forma minima può essere trovata anche lavorando direttamente sulla K-map in forma grafica usando le stesse regole e modelli visuali chiamati ***raggruppamenti rettangolari.***

***→ trovare i raggruppamenti rettangolari = verificare dove esistono configurazioni adiacenti e si può applicare la regola della combinazione. L'adiacenza vale anche in forma ciclica tra la prima e l'ultima configurazione.***




Gli implicanti e gli implicati si possono vedere nelle k-map dove i raggruppamenti rettangolari che coprono le uscite pari a 1, indicano gli implicanti e gli implicati primi se non esistono altri raggruppamenti rettangolari che li coprono.

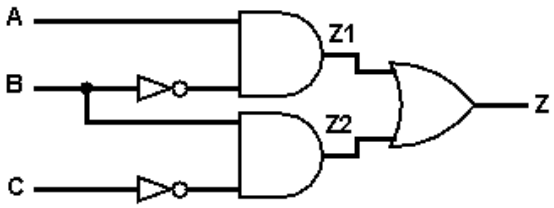
**ANALISI DI UNA RETE LOGICA:**

**Analisi** = *ottenere la descrizione comportamentale*, ossia ottenere la descrizione esaustiva mediante tabella della verità o espressione di Boole equivalente.

***→ analisi è univoca: algoritmo che permette di passare da schema logico a tabella:***

1) Si *nominano tutte le uscite dai gate logici*
2) Si *sostituiscono i gate elementari a partire dalle uscite con le corrispondenti espressioni della logica*, semplificando se necessario coi teoremi dell'algebra di Boole
3) Si fa la *valutazione dell'espressione per ottenere in modo esaustivo la tabella della verità*

**ESEMPIO:**




$Z_1 = AB$ e $Z_2 = BC$

$Z = Z_1 + Z_2 \rightarrow Z = AB + BC$

**Valutazione di un'espressione:** *ogni espressione di $n$ variabili descrive una funzione completamente specificata che può essere valutata attribuendo ad ogni variabile un valore assegnato ed applicando i postulati e teoremi dell'algebra ottenere la tabella della verità*

- **Complessità di una rete logica:**

**G = *numero di gate*** (porte logiche) ***necessari alla sintesi della rete*** (indicazione della dimensione della rete)
**L = *numero di livello di porte logiche che il segnale*** associato a un qualsiasi letterale ***di ingresso deve attraversare per raggiungere l'uscita***
**I = *numero di ingressi*** *totali alle porte logiche usate*

**SINTESI DI UA RETE LOGICA (COMBINATORIA):**

**Sintesi** = *progettare una rete logica* a partire da una descrizione comportamentale (con tabella della verità) fino al suo schema logico che ne rappresenta una descrizione strutturale.

a) **Sintesi minima:** minimo numero di gate elementari, livelli ed ingressi.

b) **Sintesi canonica:**
   - *Minimizza il numero di livelli (2)*
   - *Direttamente implementabile con strumenti automatici*
   - *Semplificabile a posteriori con algoritmi di Boole*

→ ***è la sintesi realizzata attraverso i mintermini o maxtermini***

**Definizioni:**

1) **Letterale:** ogni occorrenza di una singola variabile, sia in forma semplice Xi che complementata Xi'
2) **Implicante di una funzione f(X1…XN):** prodotto P(X') X' sottoinsieme di X di letterali tale che se P=1 IMPLICA f=1.

Date due funzioni f(X1…XN) e g(X1…XN) si dice che **f copre g** (oppure g implica f) e si scrive se f(X1…XN)=1 quando g(X1…XN)=1. Se P è il prodotto di letterali e f copre P, si dice che P è un implicante di f.

**Implicante primo** di una funzione f: *un implicante di f che non è coperto da un altro implicante di f con meno letterali* → ***implicante primo essenziale****: se comprende almeno un mintermine che non è coperto da nessun altro implicante primo*

3) **Mintermine:** *un implicante in cui appaiono tutti i letterali possibili ossia tutte le variabili di ingresso, cioè X' o X.*
4) **Maxtermine:** *un punto Bx nello spazio booleano Bn tale per cui f(Bx) = 0.*

**Implicato:** somma di letterali che impongono 0 alla funzione, e nello stesso modo si parla di ***implicato primo essenziale***

**Nelle mappe di Karnaough si possono vedere gli implicanti, ossia quelle configurazioni di ingressi che implicano che la funzione valga 1 e naturalmente anche gli implicati.**

## LA SINTESI CANONICA

Esistono ***due tipi di sintesi canonica*** equivalenti e duali:

- ***SP:*** somma di prodotti → definita come ***la somma logica dei mintermini associati alle righe della tabella nelle quali l'uscita assume valore 1***
- ***PS:*** prodotto di somme → definita come ***il prodotto logico dei maxtermini associati alle righe della tabella nelle quali l'uscita assume valore 0***

**La sintesi canonica non è minima**, ma ***può essere minimizzata con diversi algoritmi*** tra i quali quelli che impiegano la semplificazione algebrica

Esistono *molti modi per realizzare in modo manuale e automatico la sintesi logica:*

1) usare le forme canoniche e minimizzare con i teoremi **dell'algebra di Boole**
2) usare le **mappe di Karnough** e determinare gli implicanti primi ed essenziali.

   → E' un metodo manuale che funziona solo per reti assai piccole

3) usare algoritmi di sintesi logica: il più famoso è **l'algoritmo di Quine Mc Cluskey.**

## ALGORITMO DI QUINE McCLUSKEY

È il *metodo più impiegato per una sintesi minima di una rete canonica con qualsiasi numero di ingressi e anche con indifferenze*. È un metodo di minimizzazione tabellare, facile da tradurre in algoritmo con il numero di variabili trattate teoricamente illimitato. ***Consideriamo l'algoritmo per SP ma esiste anche per PS.***

Si impiega ***la distanza di Hamming definita come il n° di bit diversi tra due variabili binarie:*** *con due variabili binarie a e a' di n bit, la distanza di Hamming vale d (numero intero tra 0 e n) se d è il numero di cifre binarie corrispondenti diverse tra le due variabili.*

→ **Ad es.:** *a = 00110010 e a' = 01110100* hanno distanza di Hamming pari a 2.

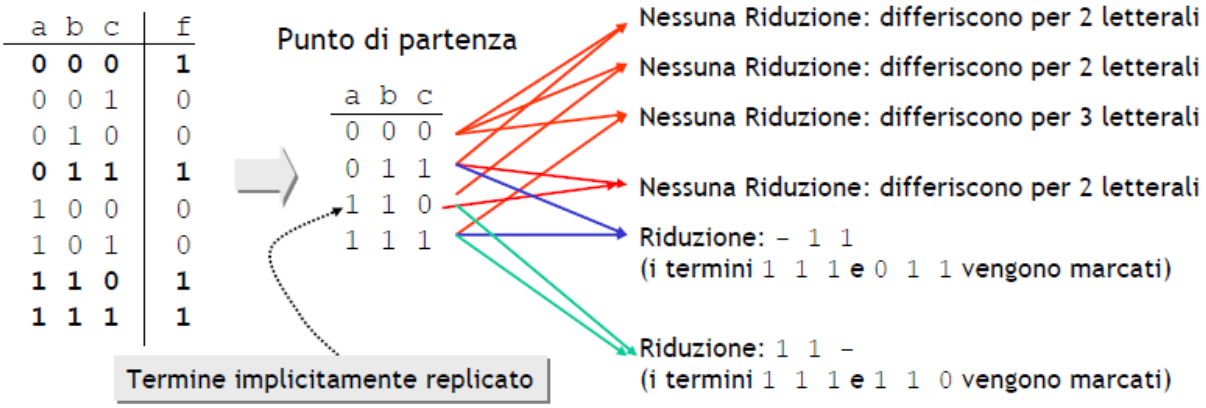
**L'algoritmo parte dalla tabella della verità, e si compone di due passi:**

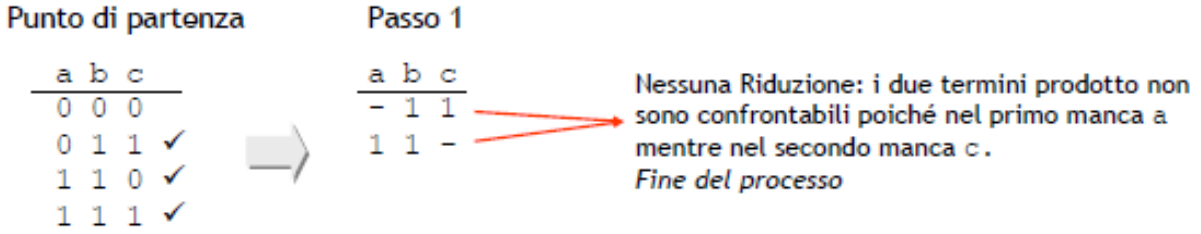
1) **calcola tutti gli implicanti primi**, a partire dai mintermini di una funzione
2) **cerca la copertura minima** ossia ottima perché minimizza i costi della rete che abbia la descrizione funzionale richiesta ma contenga il numero minimo di implicanti primi.

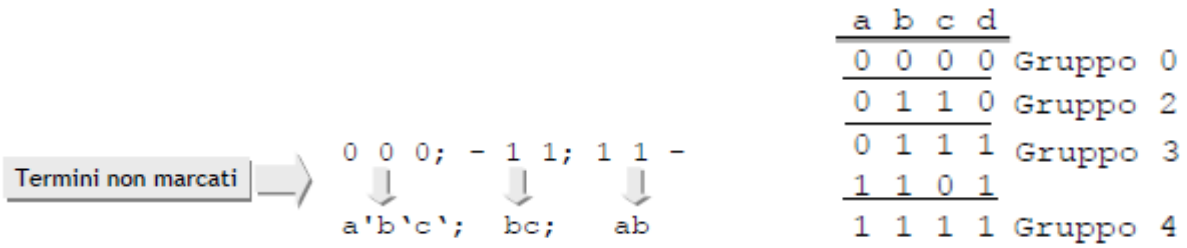
### Passo 1: ricerca degli implicanti primi

a) vengono *tabellati tutti i mintermini della funzione in forma binaria, in ordine crescente come per la tabella della verità*
b) si *applica in forma iterativa la semplificazione con la proprietà di combinazione:*
   **xiP + xi'P = (xi+xi')P = P provando tutti i letterali xi.**
   → *si semplificano in modo iterativo i termini che differiscono per un solo bit* (distanza di Hamming unitaria*) e si marcano*, in quanto essi hanno contribuito alla creazione di un implicante; quelli non marcati si possono ridurre

c) *si crea una nuova tabella con tutti i termini prodotto che derivano dalla semplificazione della prima tabella e si ripete il punto b)*
d) il processo termina quando non si possono più fare riduzioni




Si definisce il **gruppo come insieme Sij dei termini prodotto all'iterazione j con un numero di 1 pari a i;** ad ogni gruppo è collegata una etichetta (label) che indica l'insieme dei mintermini che esso copre.

**Algoritmo per Passo 1: implicanti primi;**

*j=0;*
*costruzione di $S_i^o$ (la partizione in cui ogni $S_i^o$ contenga tutti i mintermini con un numero di "1" pari a i );*
*repeat*
*for ( k= min(i); k<=(max(i)-1) ; k++)*
*{si confronta $S_k^j$ $S_{k+1}^j$;*
*se due configurazioni hanno dH=1 si genera un nuovo implicante (propr. combinazione)*
*si elimina la variabile con valore differente;*
*si marcano le configurazioni considerate e si pone il risultato in $S_k^{j+1}$;}*
*j=j+1;*
*until ( nessuna riduzione possibile);*

Tutte le configurazioni non marcate sono implicanti primi e potenzialmente essenziali: un implicante si ricorda che si dice ***implicante primo essenziale*** se comprende almeno un mintermine che non è coperto da nessun altro implicante primo.

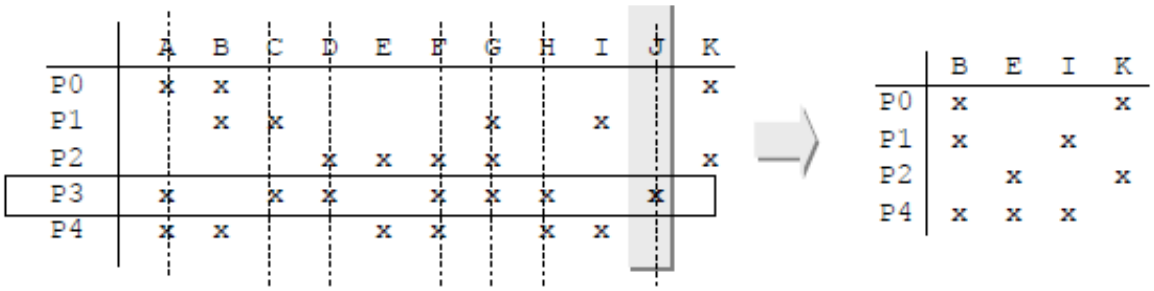
**Algoritmo per passo 2: copertura**

Per vedere se sono essenziali si devono ***trovare tra gli implicanti primi, l'insieme minimo di quelli che ottengono la copertura di tutto l'ON_SET***. Si usa una la **tabella degli implicanti** che è una *matrice binaria i cui indici di riga sono gli implicanti primi* e *le colonne sono i mintermini dell'ON_SET* e ***l'elemento Aij della tabella viene marcato se l'implicante i copre il mintermine j.***

| | 1 | 9 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| P0 | x | x | | | | | |
| P1 | | x | x | | x | | x |
| P2 | | | | x | x | x | x |

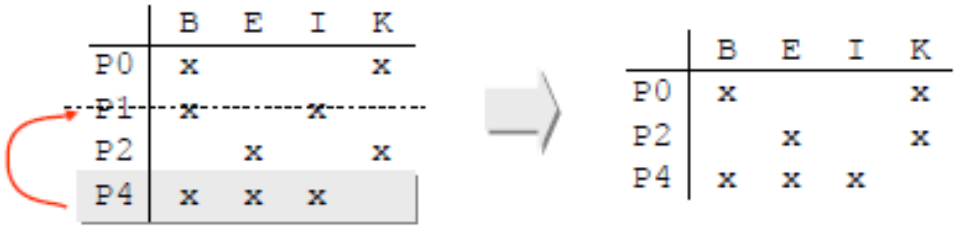
**La ricerca della copertura ottima dipende dalla funzione di costo → se gli implicanti hanno tutti lo stesso costo si minimizza solo la cardinalità della copertura.**

a) **Criterio di essenzialità:** ***si verifica se esistono implicanti primi ed essenziali***; se in una *colonna esiste una sola marcatura con X la riga è essenziale*. *Si eliminano le righe e le colonne essenziali.*



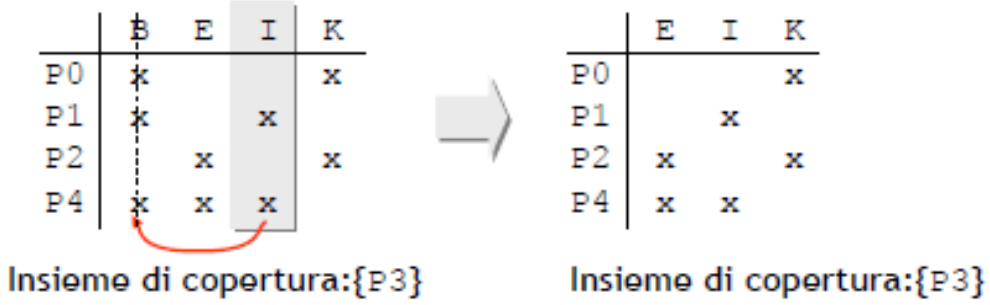

**P3 è implicante primo ed essenziale per J e copre anche A,C,D,F,G,H → Rimangono B,E,I, K da coprire.**

b) **Criterio di dominanza di riga:** ***un implicante Pi domina un implicante Pj se Pi copre almeno tutti i mintermini di Pj*** → *Pj può essere eliminato dalla tabella*




**P4 domina P1 → P1 può essere eliminato**

c) **Criterio di dominanza di colonna:** ***un mintermine Mi domina un mintermine Mj se ogni implicante che copre Mj copre anche Mi*** → *tengo solo Mj* che assicura tutte le coperture di Mi e *Mi viene eliminato dalla tabella*

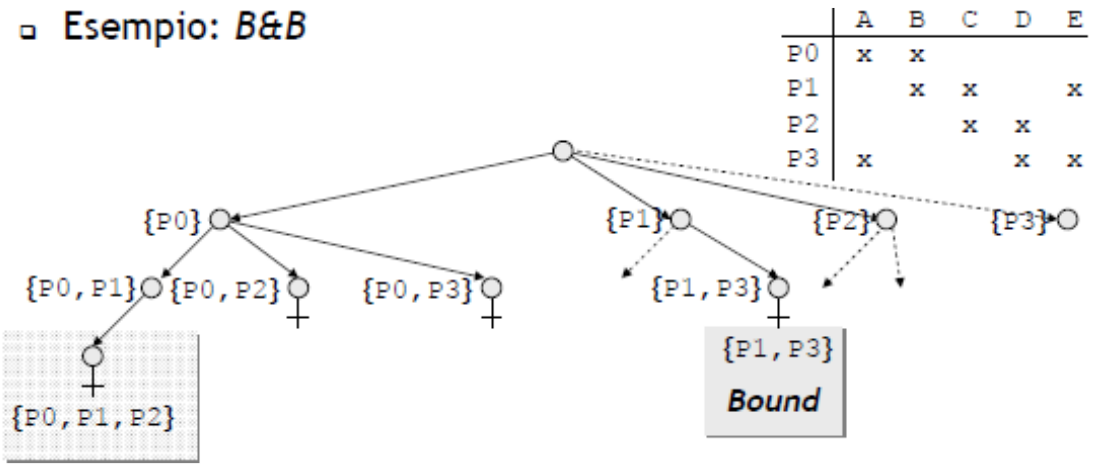



Al termine delle rimozioni di righe essenziali e delle righe e colonne dominate di ottiene la **tabella ciclica degli implicanti primi.**

- **Tabella vuota → insieme di copertura è stato trovato**

- **Tabella NON vuota → trovare una tra le soluzioni ottime usando algoritmo di ottimizzazione di Branch&Bound:**

  ogni scelta crea un ramo dell'albero delle scelte; se esso corrisponde all'implicante Po, si eliminano le righe e le colonne e si cerca di ridurre la tabella; altrimenti si itera fino a ottenere la soluzione di costo minore.




**Algoritmo Per passo 2 di copertura:**

*1 si identificano implicanti primi essenziali primari;*

*2 si esaminano dominanze di righe e colonne;*

*3 se esistono implicanti primi essenziali secondari si torna al passo 2*

*4.si applica il B&B.*

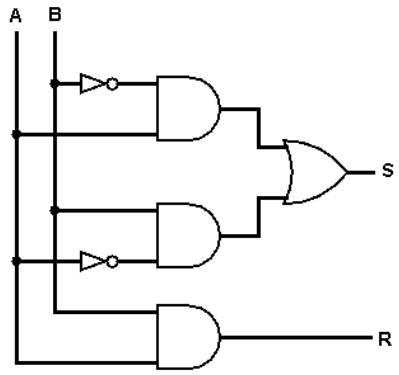
## RETI LOGICHE COMBINATORIE SEMPLICI

- **HALF ADDER (SOMMATORE):** la somma binaria è molto semplice, in quanto ogni cifra può assumere solo i valori 0 e 1, quindi è facile elencare le possibilità:

0+0 = 0 (nessun riporto)
0+1 = 1 (nessun riporto)
1+0 = 1 (nessun riporto)
1+1 = 0 (riporto di 1)

| a | b | R | s |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

→ **sintesi canonica SP che ne deriva:**

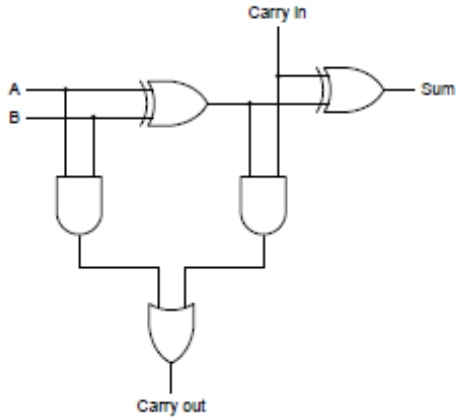



$S = \bar{a}b + a\,\bar{b}$ ed $r = ab$

**OSS: Nei calcolatori si usa il simbolo # per indicare che il segnale ha valore logico invertito e ha un significato logico vero quanto vale 0 e falso quando vale 1 (logica negativa).**

- **FULL ADDER:**

| A | B | Carry in | Sum | Carry out |
|---|---|---|---|---|
| 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 |




## RETI LOGICHE A LIVELLO DI RTL (REGISTER TRANSFER LEVEL)

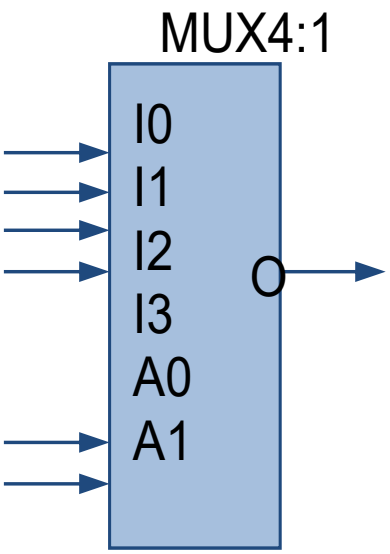
È la sintesi di blocchi logici elementari con cui realizzare per composizione ed interconnessione reti logiche e sistemi complessi. I blocchi più comuni che si impiegano nei calcolatori spesso con n° di input e output parametrici sono molti:

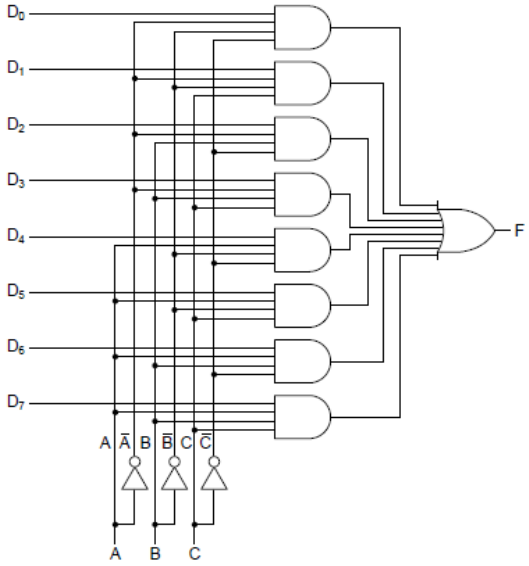
- **Multiplexer**
- **Decoder**
- **Adder, ALU**
- **….**

## 1) IL MULTIPLEXER:

È quel blocco logico che permette di ***derivare verso un'unica uscita*** un ***segnale proveniente da uno tra N possibili ingressi.***

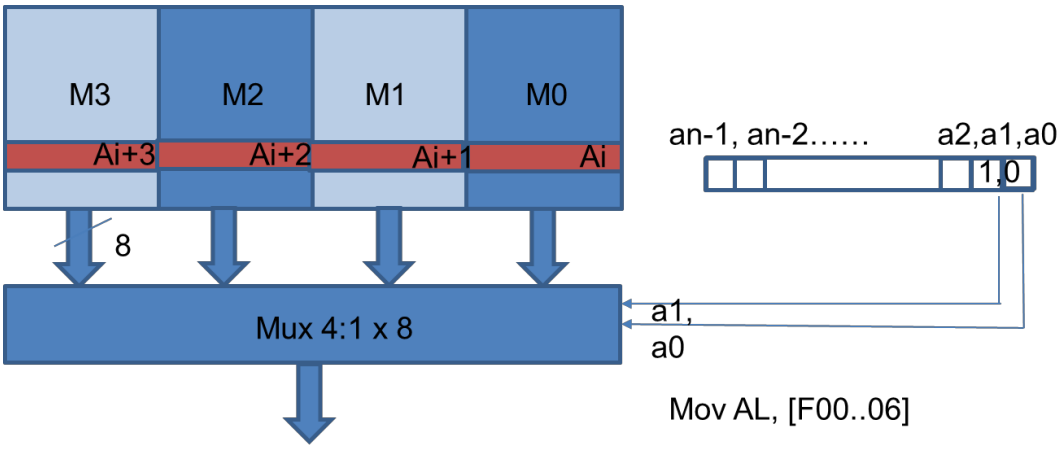
→ ***il MULTIPLEXER è una rete logica avente $2^n$ ingressi di tipo dati e $n$ ingressi di tipo segnali di controllo (o indirizzo) ed 1 uscita: in ogni istante il dato di input corrispondente alla configurazione dei segnali di controllo viene posto in uscita.***




Nei multiplexer la forma pseudo-canonica (relativamente ai soli segnali di controllo) è anche la forma minima (non è semplificabile).

**INTERFACCIA CON LA MEMORIA:**




→ ***memoria con 4 banchi "in parallelo" ognuno da 1 byte per ottenere una parola da 32 bit.*** Supponiamo che ***la CPU*** abbia un modello di indirizzamento non allineato

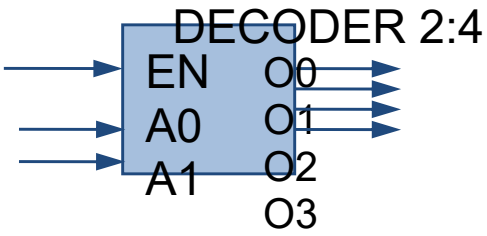
e ***voglia leggere solo 1 byte a partire da un indirizzo che termina con 06h*** (o con 02 o con A o con E). ***In questi 4 casi il byte si trova nel terzo banco M2*** (così come nel banco M0 troviamo gli indirizzi che terminano per 0 per 4 per 8 o per C etc..). → il ***bus degli indirizzi viene diviso in due parti***:

- **la parte alta** o più significativa *per selezionare l'indirizzo che potenzialmente va a tutti i banchi*
- **la parte bassa** per *selezionare quale banco ad un MUX per leggere/scrivere*

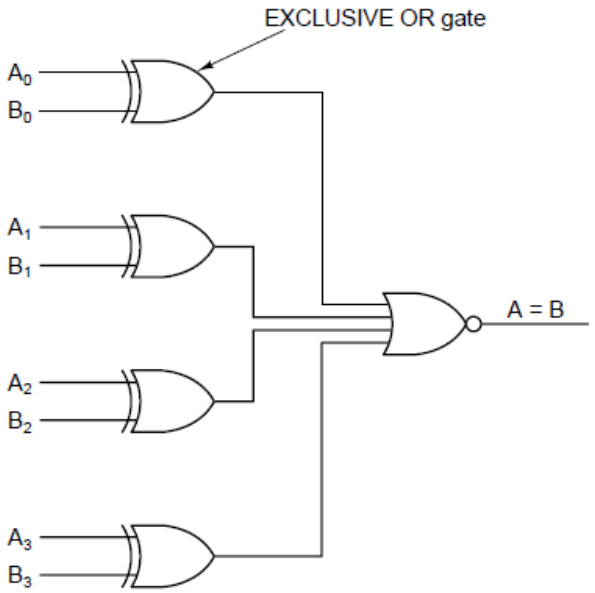
**l MUX può essere usato come dispositivo per realizzare una qualsiasi funzione canonica di tipo SP** → ***impiegare per rappresentare direttamente la tabella della verità basta collegare a "1" i mintermini della funzione canonica SP ed a "0" i prodotti di tutti i letterali che danno uscita 0.***

2) **DEMULTIPLEXER:** *contrario del MUX → accetta un segnale in serie e lo trasferisce in una delle possibili N uscite a seconda dei* **$\log_2 n$ *ingressi*** *di selezione.*
3) **DECODER:** ***rete logica che asserisce un valore 1 a una sola delle possibili $2^n$ uscite in base agli n ingressi.*** *È come il de-multiplexer con ingresso fisso a 1.* **→ usato per decodificare un segnale binario in codice 1/n. Molto usato nei bus controller del calcolatore per decodificare i bit di indirizzo verso i segnali di selezione (chiamati chip select) delle memorie e dei dispositivi di input/output.**

***Il decoder/de-multiplexer è una rete logica con 1 ingresso di dato (fissato ad 1 nel decoder), $n$ 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 e forniscono il valore 0.***

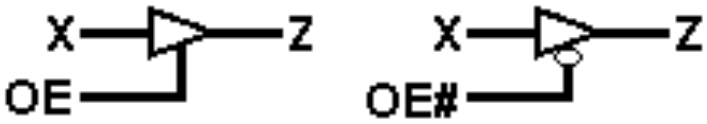


4) **COMPARATORE**: blocco logico che ha uscita uguale ad uno se le due parole di ingresso sono uguali.




5) **AMPLIFICATORE TRI-STATE:** ***è una rete logica che va al di là dell'algebra di Boole. È un generatore di segnale in terzo stato (Z = alta impedenza):***

- **se NON è abilitato il segnale di controllo OE emula** ***la presenza di un circuito aperto***
- **se OE è abilitato,** ***lascia passare il segnale di ingresso***




**OE# attivo basso:** l'uscita è uguale all'ingresso quando l'output enable è asserito (basso se OE#)

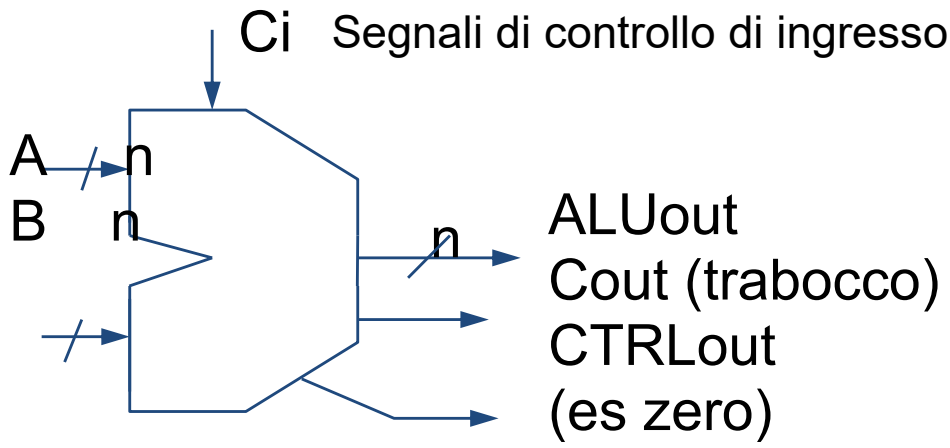
## PROGETTO DI UNA ALU

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.*

Usando una sintassi assembly a due ingressi ed una uscita con sintassi: **<op, sorg1, sorg2, dest>, le tipiche operazioni di ALU sono:**

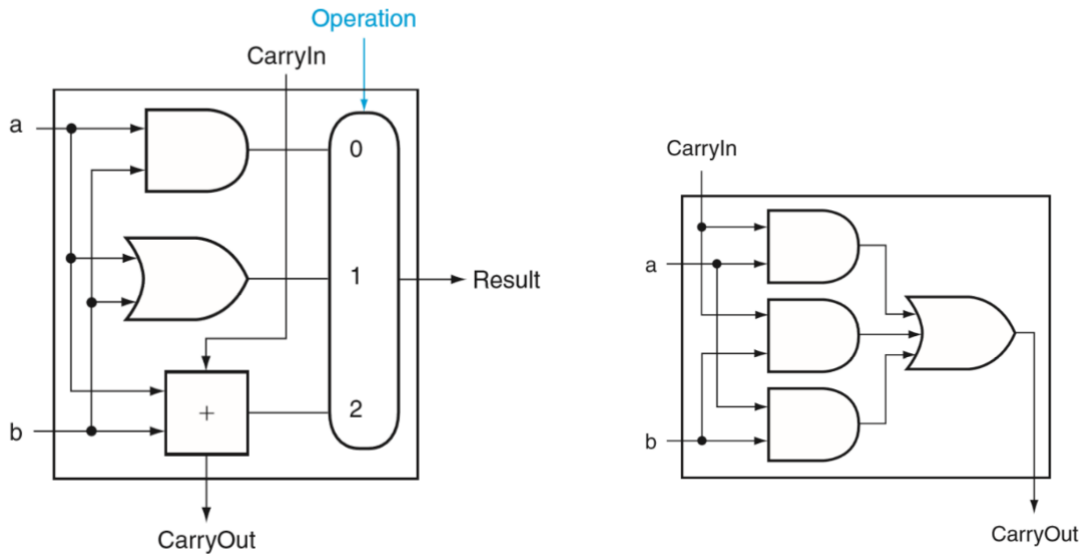
- somma add r1,r2,r3
- sottrazione sub r1,r2,r3
- prodotto logico and r1,r2,r3
- somma logica or r1,r2,r3
- negazione not r1,r2
- shift shr/shl r1,r2,r3
- confronto

***Tipico schema di una ALU:***



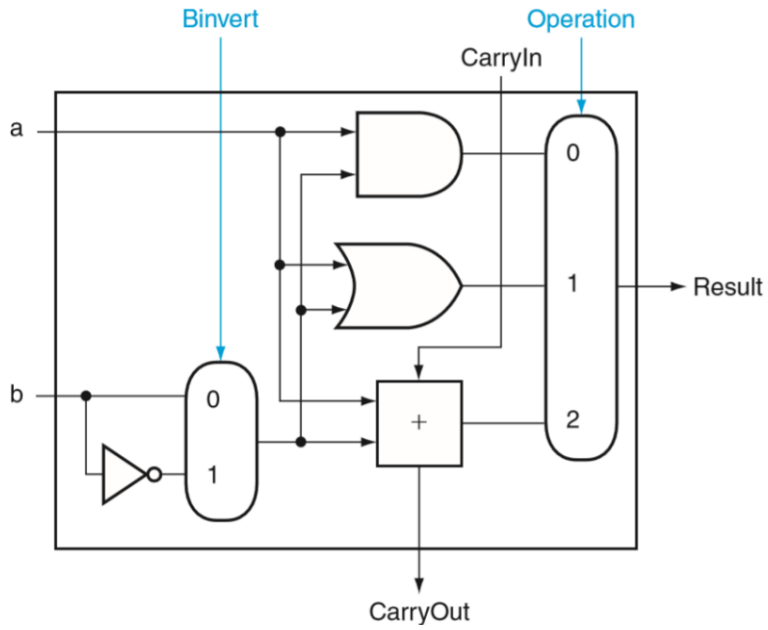

All'interno la rete logica è decomposta in reti logiche semplici per ogni funzione (poi spesso tutta l'implementazione è ottimizzata per avere il minor numero di gate in cascata).

**ALU AD 1 BIT:** ***compie 2 operazioni logiche ed 1 aritmetica***




***Gli ingressi a e b vanno a tre blocchi***, ad un **OR, ad un AND o a un FULL-ADDER** che ***crea la somma ed il riporto.*** **Le tre uscite AND, OR e SOMMA vanno ad 1 multiplexer che ha 4 ingressi (di cui 1 non usato) ed 1 uscita**.

**OSS: Se si volesse realizzare anche la sottrazione: inverto il secondo operando e sommo 1 (complemento a 2).**




Inoltre se **I** ha la possibilità di invertire anche A si ottengono anche altre funzioni logiche come il NAND e il NOR.