

# Architettura degli elaboratori I

## Lezione 1

### L'architettura MIPS

è un'architettura semplice ma potente, è possibile notarla anche a livello di assembler che vedremo in architettura II.

### La legge di Moore

la legge di Moore aveva indicato un lasso temporale di circa 18 mesi nel quale il numero di transistor e di conseguenza le prestazioni raddoppiavano, raddoppiano anche le capacità delle memorie DRAM.

Ai giorni nostri la legge di Moore non vale più in quanto si è arrivati a un limite fisico nella miniaturizzazione dei transistor ed è anche dovuto al fatto che la velocità di accesso alla memoria cresce molto più lentamente.

### L'obiettivo di un'architettura

L'obiettivo è di elaborare in modo adeguato un input per produrre l'output.

Un'architettura è composta dalle unità di ingresso che permettono al calcolatore di acquisire informazioni dall'ambiente esterno, per esempio sono tastiera, mouse, rete ecc.

L'architettura di elaborazione è l'organizzazione logica dei vari componenti interni della macchina ed il modo in cui essi cooperano per eseguire le operazioni richieste.

le unità di uscita, per esempio monitor, stampanti, che consentono al calcolatore di comunicare i risultati ottenuti dall'elaborazione all'ambiente esterno.

Un elaboratore permette di risolvere algoritmi, calcoli ed operazioni logiche.

### Operazioni elementari e codifica dell'informazione

Le operazioni elementari necessarie ad eseguire gli algoritmi sono le seguenti:

- Calcolo (somma, sottrazione, prodotto...)
- Controllo del flusso (if, for...)

L'informazione viene rappresentata utilizzando solamente due simboli in base 2, cioè 1 e 0, anche definibili come acceso e spento, infatti ogni elemento può assumere solamente due valori: bit (binary digit).

i calcoli ed i controlli sono eseguiti utilizzando esclusivamente le tre operazioni fondamentali della logica classica: AND, OR e NOT.

### I principi delle architetture

La "Universal Turing machine" è stata la prima macchina di esecuzioni di algoritmi creata.

Von Neumann negli anni 40 ha codificato i seguenti principi:

- Il dispositivo è essenzialmente una macchina di calcolo ed è dotato di un'unità devota ad eseguire i calcoli chiamata ALU
- i dati e le istruzioni sono memorizzati separatamente in una memoria read/write
- ci sarà un componente che gestisce tutto il sistema di elaborazione: trasferimento dei dati, comanda le operazioni e l'input/output. Questo componente è chiamato control unit (UC)
- il contenuto della memoria può essere recuperato in base alla sua posizione (detta indirizzo) e non è funzione del tipo di dato
- l'esecuzione procede sequenzialmente da un'istruzione alla seguente, nelle architetture più avanzate l'esecuzione procede sequenzialmente per gruppi di istruzioni.

## Struttura dell'elaboratore

Gli elementi principali di un elaboratore sono: la CPU (central processing unit) anche definita come unità centrale di elaborazione e la memoria principale o di lavoro.

Sulla motherboard di un elaborato possiamo trovare i collegamenti principali di un calcolatore:

Bus di sistema (dati, indirizzi e controllo)

Bus di I/O che permettono il collegamento ai dispositivi di input/output, alle memorie di massa, alla rete e ad altri dispositivi.

## Central processing unit

La CPU provvede ad eseguire le istruzioni che costituiscono i diversi programmi elaborati dal calcolatore.

La CPU deve anche eseguire le istruzioni, eseguire dei calcoli a seconda dell'istruzione e dei dati a disposizione.

## Elementi principali della CPU

- Banco di registri (register file) ad accesso rapido, in cui vengono memorizzati i dati di utilizzo più frequente, in quanto il tempo di accesso ai registri è circa dieci volte più veloce del tempo di accesso alla memoria principale. Il register file è composto dalla cache ed i registri.
- Il Program counter (PC) che contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzione
- L'Instruction register (IR) che contiene l'istruzione in corso di esecuzione
- L'Arithmetic logic unit (ALU) unità dedicata all'esecuzione delle operazioni aritmetico-logiche, i dati forniti all'ALU provengono direttamente dai registri interni alla CPU ma possono provenire anche dalla memoria, in quest'ultimo caso devono essere prima trasferiti in registri interni alla CPU.
- Unità di controllo che si occupa di controllare il flusso e determina le operazioni di ciascun blocco
- Esistono anche alcune unità aggiuntive per elaborazioni particolare come unità aritmetiche per dati in virgola mobile (FPU – floating point unit), sommatore ausiliari ecc.

## Ciclo di esecuzione di un'istruzione MIPS

- Prelievo istruzione (fase di fetch)
- Decodifica (decode)
- Calcolo (execute)
- Lettura/scrittura (memory)
- Write back

Esempio di istruzione:

Somma: 0x80000: addi \$s3, \$s2, 4

Somma il contenuto del registro \$s2 con la costante 4 e scrivi il risultato nel registro \$s3

### Fase di Fetch

Vengono prelevate dalla memoria principale le istruzioni e i dati, l'esecuzione è caricata in un registro speciale detto instruction register

### Fase di decodifica (decode)

La unità di controllo riconosce di che istruzione si tratta decodificandola e predispone la ALU per la sua esecuzione, in questa fase vengono anche recuperati gli operandi dal register file o dalla memoria.

### Fase di calcolo (execute)

La ALU svolge le operazioni aritmetico/logiche necessarie all'esecuzione dell'istruzione

### Fase di lettura/scrittura

Viene effettuato un accesso a memoria se l'istruzione lo prevede

Write back

Viene salvato il risultato della operazione nei registri oppure in memoria

La storia dell'elaboratore

## Lezione 2

### Il linguaggio

Per farsi capire da un calcolatore bisogna parlare la sua stessa lingua, un linguaggio è costituito dalla semantica (significato delle parole) e dalla sintassi.

Le parole del linguaggio sono rappresentate mediante simboli, quest'ultimi sono associati a dei suoni e sono le lettere dell'alfabeto.

Esistono diverse tipologie di alfabeti per rappresentare le stesse parole.

### Alfabeto Binario

L'alfabeto dei calcolatori è costituito da due simboli, che vengono rappresentati come 0 e 1, sono suddivisi a gruppi di 8 bit chiamati byte.

Un bit è l'unità di informazione base e può assumere solo due valori.

L'alfabeto italiano più i simboli possono essere tradotti dal linguaggio binario, per farlo è stata creata la codifica ASCII (American standard code for information interchange).

Ogni carattere corrisponde ad un gruppo di 8 bit, ogni numero, in base dieci, di questa tabella equivale ad un carattere, dal numero 0 al 31 sono i codici di controllo, dal 32 al 127 i caratteri dello standard ASCII, per i restanti fino al 255 troviamo gli extended ASCII.

L'unico problema di questa codifica è che non potevi decifrare tutte le lingue del mondo in quanto è stata sviluppata sul all'alfabeto americano, per esempio non sono indicate le lettere accentate.

Infatti, successivamente è stata sviluppata la codifica UNICODE che raccoglie la maggioranza delle lingue mondiali ed è anche costituito da un maggior numero di byte per un maggiore numero di caratteri codificati.

### Sistema di numerazione binario

#### Codifica binaria

Per poter rappresentare un numero maggiori di informazioni è necessario usare sequenze di bit. Esistono due tipologie di codifica binaria, quella implicita e quella esplicita.

Codifica binaria implicita, dato un insieme di elementi, associa ad ognuno di essi un valore tramite la codifica binaria senza fare riferimento ad un legame.

La codifica binaria esplicita associa ad ogni insieme binario un numero.

La codifica binaria è costituita da 0 e 1, pertanto non limita in alcun modo le funzionalità del calcolatore nel rappresentare valori a patto che di avere uno spazio sufficiente a rappresentare le informazioni.

Una stringa binaria non ha un significato univoco, può essere interpretata in modo differente in base al contesto e avere differenti significati.

### I numeri

gli informatici hanno provato a definire gli insiemi numerici matematici in linguaggio informatico per

esempio:

l'insieme dei numeri naturali non ha una definizione in campo informatico, mentre i numeri relativi vengono definiti come integer (int) e i numeri decimali, quelli con la virgola, solitamente sono definiti come real o float.

La codifica di un numero binario è basata su una codifica posizionale come i nostri numeri interi, a differenza di quelli romani.

La codifica di un numero decimale associa ad ogni posizione della cifra un peso differenza che può essere espresso moltiplicando il numero per 10 alla potenza associata alla posizione del numero, per esempio  $764_{10} = 7 \times 10^2 + 6 \times 10^1 + 4 \times 10^0$ .

La stessa risoluzione vale anche per i numeri negativi ma utilizzando potenze negative, vale anche per i numeri binari ma si utilizzano le potenze di 2 al posto del numero 10.

Per convertire un numero da base 2 a 10, gli uni del numero binario vengono moltiplicati per 2 alla N in base alla posizione dell'uno, mentre per convertire da base 10 a base 2, il numero in base 10 viene diviso per 2 in modo ricorsivo finché non si ottiene 0, i resti dell'operazione presi dal basso verso l'alto identificano il numero binario associato, questi passaggi valgono anche per la conversione in base esadecimale.

### Le operazioni di somma

Per sommare due numeri binari tra di loro viene utilizzata la somma in colonna con il riporto in quanto il valore può valere solo 0 e 1.

### Complemento a due

Per poter rappresentare i numeri negativi tramite numeri binari, visto che i numeri negativi sono complementari ai numeri positivi, è stata ideata la codifica a complemento a 2, la prima cifra indica il bit di segno.

000 = 0            100 = -4

001 = 1           101 = -3

010 = 2           110 = -2

011 = 3           111 = -1

Nella codifica a complemento a due il numero negativo si ottiene cambiando gli 0 con 1 e viceversa e sommando 1.

### Codifica dei numeri relativi interi su N bit

una volta convertito il numero in base per esempio  $11_{10} = 1011_2$  bisogna a procedere a riempire tutti gli slot vuoti rimanenti con degli 0, in caso di codifica su 16 bit il risultato è il seguente: 0000 0000 0000 1011.

In caso di numeri negativi viene replicato il primo bit, quello del segno, perciò riempiamo gli slot con 1.

### Conversione da base 10 a base N della parte decimale

Come convertire la parte intera l'abbiamo già visto sopra basta dividere ricorsivamente il numero per la base desiderata da ottenere tenendo presente il resto.

Per la parte decimale procediamo a moltiplicarla per due in modo ricorsivo finché il risultato non è pari a 0 oppure finché gli slot di bit siano pieni. Pian piano che procediamo in modo ricorsivo teniamo conto del numero intero, quando risulta essere uno lo rimuoviamo e se esiste ancora parte decimale continuiamo a moltiplicare. Gli 0 e 1 tenuti da parte li andiamo a leggere dall'alto verso il basso.

$$\begin{array}{rcll}
0,625 * 2 = 1,250 = 1 + 0,250 & \Rightarrow & 1 & \downarrow \\
0,250 * 2 = 0,500 = 0 + 0,5 & \Rightarrow & 0 & \\
0,500 * 2 = 1,000 = 1 + 0,0 & \Rightarrow & 1 & \\
0,0000 & & & 
\end{array}$$

Per riottenere la parte decimale in base 10 andiamo a moltiplicare ogni bit uguale a 1 per la corrispettiva potenza di due negative.

### Tassonomia ed unità di misura

Hertz: numero di cicli al secondo nei moti periodi (clock)

MIPS: milioni di istruzioni per secondo

MFLOPS: milioni di istruzioni in virgola mobile al secondo

### Terminologia

Bit = binary digit

- 1 byte = 8 bit.
- 1kbyte = 1024 byte
- 1Mbyte = 1048,576 byte.
- 1Gbyte = 1073,741,824 byte.
- 1Tbyte = 1099,511,627,776 byte.

### Numeri Frazionari

Per rappresentare i numeri decimale (float) esistono due tipi di codifiche possibili:

**La rappresentazione a virgola fissa**, dove la posizione della virgola è già definita all'interno di una stringa.

**La rappresentazione a virgola mobile**, un numero razionale  $(N)_b$  è espresso come  $N = (-1)^s * m * B^e$

Dove con **s** indichiamo il segno (0 positivo, 1 negativo), con **m** la mantissa, cioè il numero frazionario su n bit in base B e con **e** indichiamo l'esponente, cioè il numero intero.

Nella forma normalizzata la parte intera ha solo una cifra significativa per esempio  $5.79 * 10^2$ , in base 2 la forma normalizzata inizia sempre con 1.xxx

### Virgola fissa e virgola mobile differenze

Il numero massimo e minimo rappresentabile lo definiamo con  $V_{min}$  e  $V_{max}$ .

Dato un numero X che si vuole rappresentare:

- l'errore assoluto  $e_A(x)$  è la differenza tra il numero x e la sua più vicina rappresentazione
- l'errore relativo  $e_R(x)$  è la differenza tra il numero x e la sua più vicina rappresentazione in percentuale del valore  $x = e_A(x)/x$

Virgola fissa	Virgola mobile
$R_{N, VF} = iii.fff$	$R_{N, VM} = i.fff \cdot 10^{ee}$
$V_{min} = 000.000, V_{max} = 999.999 \cong 10^3$	$V_{min} = 0, V_{max} = 9.999 \cdot 10^{99} \cong 10^{100}$
$\epsilon_A \leq 10^{-3}$	$\epsilon_A \leq 10^{-3} \cdot 10^{ee} \leq 10^{ee-3}$
$\epsilon_R \in \left[ \frac{10^{-3}}{10^3} = 10^{-6}, \frac{0.001}{0} \right)$	$\epsilon_R = \frac{10^{ee-3}}{m \cdot 10^{ee}} \cong 10^{-3}$

A parità di cifre utilizzare la rappresentazione in virgola mobile è in grado di rappresentare un numero più elevato di valori, la rappresentazione in virgola fissa ha una precisione assoluta costante e relativa variabile

mentre la rappresentazione in virgola mobile ha una precisione assoluta variabile con N e relativa approssimativamente costante.

Lo standard IEEE754 (standard for Floating Point Arithmetic)

È una standard definito per i numeri con la virgola a precisione singola: 32 bit

s		E (8 bit)						m (23 bit)																								
31	30						23	22																								0

Un numero normalizzato è così suddiviso:

- i bit da 0 a 22 sono la mantissa, la parte frazionaria del numero
- i bit da 23 a 30 sono l'esponente, il numero intero tra -126 e 127 memorizzato su 8 bit in eccesso 127, cioè memorizziamo il numero intero positivo  $E = e + 127$  anziché  $e$ .
- il bit 31 è il segno, 0 per positivo e 1 per negativo.

La mantissa viene rappresentata nella forma  $1.c_1c_2...c_{23}$ , per convenzione il bit  $c_0$  è sempre uguale a 1 e non si rappresenta.

I 23 bit di M rappresentano l'intervallo  $[1,2)$ .

Esempio: convertiamo  $17.375_{10}$ , dobbiamo determinare s, m ed E.

- Il numero è positivo  $\rightarrow s = 0$
- Converto la parte intera in binario:  $(17)_{10} = (10001)_2$
- Converto la parte frazionaria  $(.375)_{10} (= \frac{3}{2^3})$

$$\begin{array}{lll} 0.375 \cdot 2 = 0.75 & \text{parte intera} = 0 & \text{MSD} \\ 0.75 \cdot 2 = 1.5 & \text{parte intera} = 1 & \\ 0.5 \cdot 2 = 1.0 & \text{parte intera} = 1 & \end{array} \quad \left| \quad (.375)_{10} = (.011)_2 \right.$$

- Unisco i risultati:  $(10001.011)_2$
- Normalizzazione:  $(1.0001011)_2 \cdot (10)_2^4$
- Mantissa:  $m = 0001\ 0110\ 0000\ 0000\ 0000\ 000$
- Esponente:  $E = e + 127 = (4)_{10} + (127)_{10} = (131)_{10} = (10000011)_2$

Certi valori di m e di e sono utilizzati secondo diverse convenzioni definite dallo standard:

- se  $0 < E < 255$  (e appartiene  $[-126,127]$ )  $\rightarrow$  numero normalizzato
- se  $m = 0, E = 0 \rightarrow +0$  o  $-0$  a seconda di s
- se  $m = 0, E = 255 \rightarrow +\infty$  o  $-\infty$  a seconda di s
- se m diverso da 0,  $E = 255 \rightarrow \text{NaN}$ , non a number
- se m diverso da 0,  $E = 0 \rightarrow$  numero subnormalizzato

nel 2008 lo standard IEEE754 è stato aggiornato con nuovi formati

a 16 bit, anche detto mezza precisione, 1 bit per il segno, 5 bit per l'esponente con polarizzazione di 15 e 10 bit per la parte frazionaria

a 128 bit, ovvero quadrupla precisione, 1 bit di segno, 15 bit per l'esponente con polarizzazione di 262143 e 112 bit per la parte frazionaria.

google ha definito un nuovo formato chiamato Brain floating (bf16) basato sul formato fp32 ma utilizzando 16 bit al posto di 32, il quale utilizza 7 bit per la mantissa, 8 bit per l'esponente, con polarizzazione di 127, e uno per il segno.

### Lezione 3 – I circuiti logici

#### Le operazioni logiche fondamentali

all'interno di un elaboratore vengono effettuate solamente operazioni logiche

Le tre funzioni elementari sono NOT, AND e OR.

#### Rappresentazione delle funzioni logiche

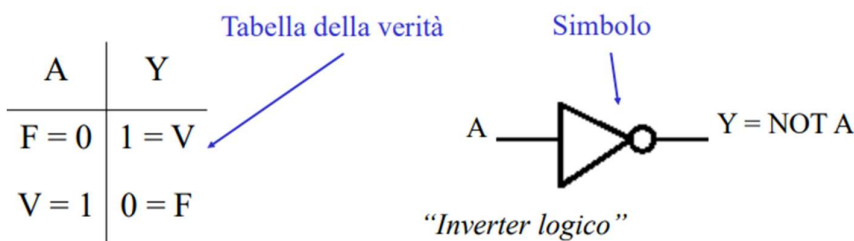
Tramite una tabella della verità possiamo definire tutte le possibili opzioni di un'operazione logica elencando tutti i possibili valori che essa può assumere per ogni valore preso in input e ogni valore di uscita.

Gli operatori logici, per esempio not and e or, vengono rappresentati in un circuito tramite le porte logiche.

#### L'operatore NOT

L'operatore NOT inverte il valore preso in input, può essere definito come un inverter logico, per esempio lo 0 diventa 1 e viceversa, la sua scrittura algebrica è la seguente  $NOT A = \bar{A} = !A$

Questa è la sua tabella della verità e la sua porta logica:



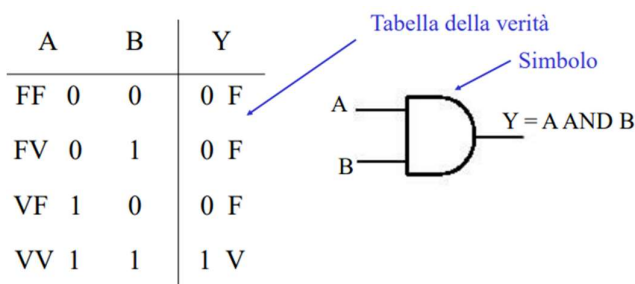
Concatenazione del NOT, possiamo indicare il not anche con un cerchio all'ingresso o all'uscita degli altri operatori, se messo all'ingresso vuol dire che esso è negato, mentre alla fine indica che il risultato deve essere negato.

#### L'operatore AND = Prodotto logico

L'operatore AND restituisce 1 (vero) solamente quando in input riceve due valori 1 (veri) mentre in tutti gli altri casi restituisce 0.

Scrittura algebrica  $Y = A \text{ AND } B = A * B = AB$

Questa è la sua tabella della verità e la sua porta logica:



## L'operatore OR = Somma logica

L'operatore OR restituisce 1 (vero) quando riceve in input almeno un valore 1 mentre restituisce 0 solamente quando riceve tutti valori pari a 0.

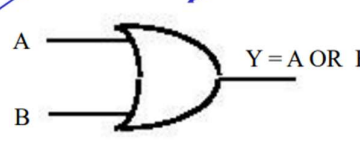
Scrittura algebrica  $Y = A \text{ OR } B = A + B$

Questa è la sua tabella della verità e la sua porta logica:

A	B	Y
FF 0	0	0 F
FV 0	1	1 V
VF 1	0	1 V
VV 1	1	1 V

Tabella della verità

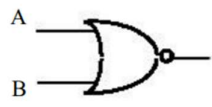
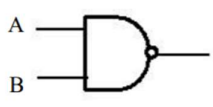
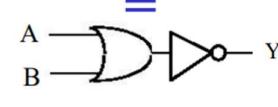
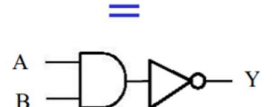
Simbolo



$Y = A \text{ OR } B$

Esistono anche gli operatori negati NOR e NAND, che sono dei normali operatori OR e AND ma con l'uscita negata.

Queste sono le loro tabelle di verità:

A	B	OR(A,B)	Y	Operatore OR negato	A	B	AND(A,B)	Y	Operatore AND negato
0	0	0	1		0	0	0	1	
0	1	1	0		0	1	0	1	
1	0	1	0		1	0	0	1	
1	1	1	0		1	1	1	0	
$\text{"Not(Or(A,B))"}$				$Y = \overline{A + B} = \overline{A + B}$	$\text{"Not(And(A,B))"}$				$Y = \overline{A B}$

## Precedenza degli operatori

1- NOT 2- AND 3- OR

$Y = A + B\hat{C}$  va intesa come  $Y = A + (B(\hat{C}))$   $\hat{C} = C$  negato

Le porte logiche possono avere N ingressi, anche avendo più ingressi valgono le stesse tabelle di verità ma con più casi. Per esempio, un AND a 4 ingressi ha 16 soluzioni ma solo una è 1, l'or è il contrario, 15 vere una falsa.

## Il Transistor

Le porte logiche sono costituite da transistor, è un componente con funzionamento lineare (amplificazione) o binario (interruttore).

Il transistor nel caso di interruttore tra emettitore e collettore è comandato dalla tensione sulla base, esistono due casi estremi, quando la tensione alla base è bassa C ed E sono isolati (spento) mentre quando la tensione alla base è alta C ed E sono collegati (acceso).

## La tecnologia CMOS – Complementary-MOS

MOS = Metal – Oxide Semiconductor



I vantaggi della tecnologia CMOS sono:

- Tensione di alimentazione flessibile
- Consumo bassissimo, in quanto consuma solo nella transizione, in condizioni statiche il consumo è praticamente nullo

È possibile vederlo come un ponte levatoio che si alza e abbassa tra Drain e Source ed è pilotato dal gate, consuma energia solo nell'operazione di abbassamento o alzamento.

La porta NOT in CMOS, anche detta inverter logico vengono utilizzati due MOS complementari

**Rivedere pezzo di lezione**

### **La funzione**

La funzione è una relazione che associa ad ogni elemento dell'insieme X, detto dominio, associa ad un elemento dell'insieme Y, detto codominio, indicando con  $Y = F(x)$

Le nostre funzioni saranno tipicamente relazioni tra ingressi e uscite multidimensionali (più variabili in ingresso e più variabili in uscita)

### **Le funzioni logiche**

Sono delle funzioni a N ingressi e M uscite, per ciascuna delle  $2^n$  combinazioni degli ingressi vengono definite delle M uscite.

La funzione logica ha domini finiti, infatti si possono calcolare tutti i punti dove la quale è definita.

La funzione logica è implementata da un'opportuna combinazione di porte logiche (NOT, AND, OR) che prende gli N ingressi e li trasforma nell'uscita desiderata.

La funzione può essere rappresentata in tre modi:

- Circuito
- Tabella della verità (Truth table, TT)
- Espressione simbolica

Tramite l'espressione logica possiamo ricavare la tabella della verità

Data l'espressione:  $F = A \text{ AND } B \text{ OR } B \text{ AND NOT } C = AB + B\bar{C}$

Questa espressione si legge:  $F = (A \text{ AND } B) \text{ OR } (B \text{ AND } (!C))$

Ricaviamo la tabella delle verità:

A B C	A and B	B and (!C)	F
0 0 0	0	0	0
0 0 1	0	0	0
0 1 0	0	1	1
0 1 1	0	0	0
1 0 0	0	0	0
1 0 1	0	0	0
1 1 0	1	1	1
1 1 1	1	0	1

Anche dal circuito è possibile risalire alla tabella della verità facendo le opportune prove di ingresso.

Una tabella della verità ed/o un'espressione possono essere rappresentate da diversi circuiti ma che restituiscono gli stessi risultati, essi sono definiti circuiti equivalenti.

### Manipolazione Algebrica

Seguendo alcune proprietà qui sottoelencate possiamo semplificare le nostre funzioni logiche, alcune di queste proprietà derivano dalla matematica.

Nell'algebra di Boole vale il principio di dualità, il duale di una funzione booleana si ottiene sostituendo AND ad OR e OR ad AND, gli 0 ad 1 e viceversa, come esempi possiamo vedere le proprietà di identità ed elemento nullo.

	AND	OR
Identità	$1 \wedge X = X$	$0 \vee X = X$
Elemento nullo	$0 \wedge X = 0$	$1 \vee X = 1$
Idempotenza	$X \wedge X = X$	$X \vee X = X$
Inverso	$X \wedge \neg X = 0$	$X \vee \neg X = 1$
Commutativa	$X \wedge Y = Y \wedge X$	$X \vee Y = Y \vee X$
Associativa	$(X \wedge Y) \wedge Z = X \wedge (Y \wedge Z)$	$(X \vee Y) \vee Z = X \vee (Y \vee Z)$
Distributiva	(di AND risp. ad OR) $X \wedge (Y \vee Z) = X \wedge Y \vee X \wedge Z$	(di OR risp. ad AND) $X \vee (Y \wedge Z) = (X \vee Y) \wedge (X \vee Z)$
Assorbimento I	$X \wedge (X \vee Y) = X$	$X \vee (X \wedge Y) = X$
Assorbimento II	$X \wedge (\neg X \vee Y) = X \wedge Y$	$X \vee (\neg X \wedge Y) = X \vee Y$
De Morgan	$\neg(X \wedge Y) = \neg X \vee \neg Y$	$\neg(X \vee Y) = \neg X \wedge \neg Y$

Commutativa  $x y z = y x z = z x y$        $x+y+z = y+x+z = z+x+y$

Distributiva      AND rispetto ad OR      OR rispetto ad AND  
 $x(yh+yz) = xyh+xz$        $xh + yz = (xh+y)(xh+z)$

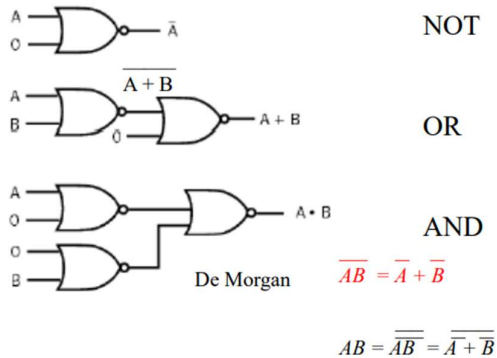
De Morgan       $\overline{xyz} = \bar{x} + \bar{y} + \bar{z}$        $\overline{x+y+z} = \bar{x} \bar{y} \bar{z}$

## Porta universale NOR

La porta NOR, così come la porta NAND, sono dette universali perché sono in grado di rappresentare le tre porte logiche fondamentali NOT, OR e AND.

Ha il comportamento di una porta logica OR ma con l'uscita negata

Qui sotto possiamo vedere come sono rappresentate le tre porte logiche fondamentali tramite l'utilizzo di sole porte logiche NOR



## Lezione 4

### I Circuiti combinatori

I circuiti combinatori sono circuiti logici in cui l'uscita è determinata univocamente dai segnali in entrata, essi sono formati da due o più porte logiche, i circuiti non hanno memoria, temporizzazione o loop di feedback, il loro funzionamento è istantaneo.

Un circuito combinato esegue un'operazione assegnata logicamente da un'espressione booleana o da una tabella di verità.

Alcuni esempi di circuiti combinatori sono: il multiplexer, il decoder.

**Mintermine:** un implicante che contiene tutte le N variabili della funzione associato agli 1 della funzione

**Maxtermine:** Contiene tutte le N variabili della funzione ed è tale che il loro prodotto logico è uno 0 della funzione

