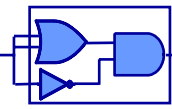
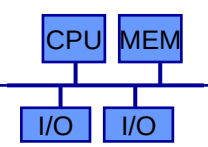


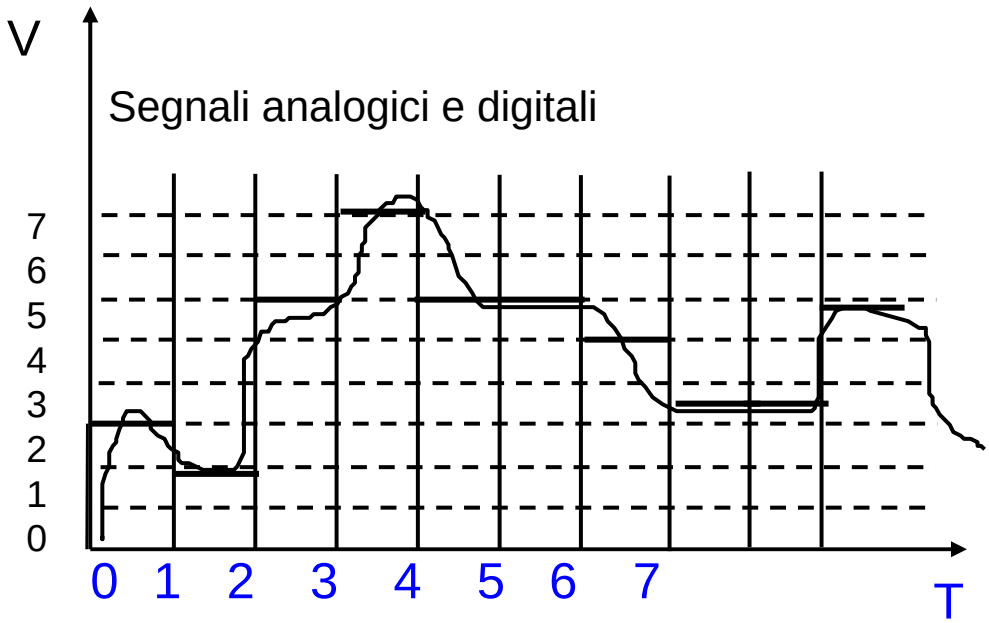
Reti Logiche

Capitolo 1: Introduzione alle reti logiche



Segnali digitali

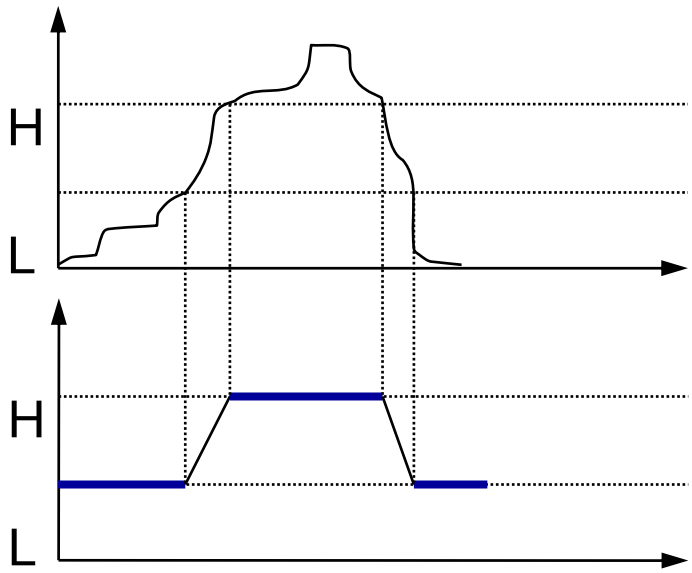
- **SEGNALE**: grandezza fisica che varia nel tempo e che puo' assumere un insieme di valori misurabili
- Segnale **analogico**:se può assumere infiniti valori all'interno di un dato intervallo
- Segnale **digitale** :se può assumere un numero finito di valori all'interno di un dato intervallo



- **Segnale binario**: si dice binario quando assume due soli valori.
- **Bit** variabile binaria che può assumere solo valori 1 e 0
- **Variabili binarie** sono variabili definite da configurazioni binarie di *n* bit, ossia stringhe di 1 e 0 lunghezza n (cifre binarie). un segnale digitale può essere rappresentato da una variabile binaria.
- *Con n bit si possono rappresentare $N=2^n$ configurazioni binarie diverse*

Segnale digitale binario

- Il segnale binario viene codificato con una variabile binaria ad 1 bit. I valori 1 e 0 definiscono i due stati del segnale binario;
- I due valori 1 e 0 (true, false) derivano dal campionamento del segnale mediante soglie;



- dipendentemente dalla **tecnologia** impiegata si usa una logica positiva o negativa a seconda se il valore H (alto) viene associato al valore logico 1 o 0:
- **LOGICA POSITIVA**
- (segnale attivo alto)
  - H (5V) --> 1 (true)
  - L (0V) --> 0 (false)

- **LOGICA NEGATIVA**
- (segnale attivo basso)
  - H (5V) --> 0 (false)
  - L (0V) --> 1 (true)

Codifica Binaria

- Con una variabile binaria ad n bit si rappresentano 2 ^n valori possibili

- Esistono diverse codifiche possibili:

- Codificare un segnale digitale che può assumere N valori con n bit secondo la **CODIFICA BINARIA** significa associare ad ognuno degli N valori la configurazione binaria che rappresenta in base 2 tale valore.

Ad esempio x=62 codifica binaria di x a 8 bit 00111110

n	N	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
9	512	
10	1024	1K
11	2048	2K
12	4096	4K
13	8192	8K
14	16384	16K
15	32768	32K
16	65536	64K

2^{10}	1K
2^{20}	1M
2^{30}	1G
2^{40}	1T

Altre codifiche

- **Distanza di Hamming** tra due configurazioni binarie e' il n. di bit aventi valore diverso di cifre binarie:

  es: 1001 1011 distanza 1;
  1001 1010 distanza 2

- **Codici a distanza di Hamming unitaria**: in cui simboli consecutivi dell'alfabeto rappresentato hanno distanza 1.

  es: **codice di Gray**

Simboli alfabeto	cod. Gray	cod. Binaria
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Parita’

- I codici **rilevatori d’errori** sono codici in cui e’ possibile rilevare se sono stati commessi errori nella trasmissione

  Codici ridondanti: in cui l’insieme dei simboli dell’alfabeto e’ minore dell’insieme di configurazioni rappresentabili col codice

- Codici con **bit di parità:** alla codifica binaria si aggiunge un bit di parità;
- (codice ridondante in quanto usa 1 bit in piu’ del necessario)

- **parità pari** rende pari il numero di 1 presenti nella parola (vale 1 se ci sono un n. dispari di 1)
- **parità dispari:** il contrario

- I codici di parità rilevano la presenza di un numero dispari di errori (e quindi di errori singoli) perché tutte le configurazioni rappresentate hanno distanza di Hamming minima pari a 2

- es. valore definito con 8 bit 11001011
- con 9 bit con parita’ (pari) 110010111**1**

Parita’

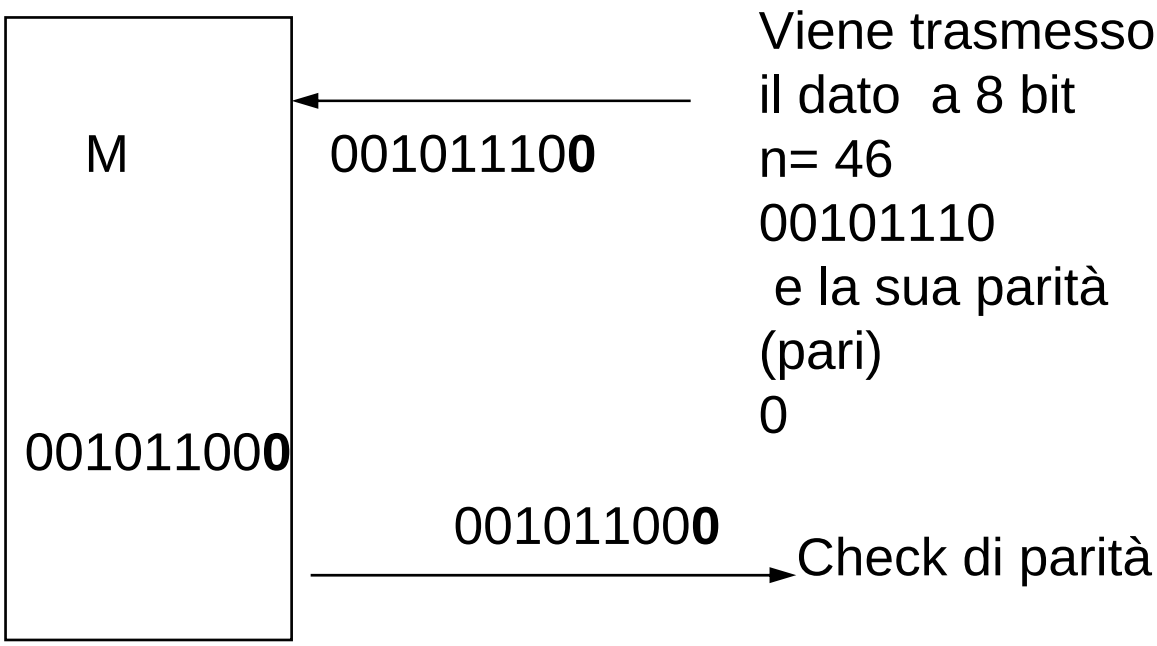
Simboli alfabeto	cod. Binaria	cod. Binaria con parità pari
0	000	000 0
1	001	001 1
2	010	010 1
3	011	011 0
4	100	100 1
5	101	101 0
6	110	110 0
7	111	111 1

Ad ogni simbolo dell’alfabeto corrisponde una configurazione a parità pari.

Le configurazioni a parità dispari non codificano alcun simbolo dell’alfabeto.

Se viene rilevata una configurazione a parità dispari significa che si e’ verificato un errore che ha alterato un numero dispari di bit (1, 5, 5, 7, ...).

Esempio



- 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 e' ammissibile per la codifica binaria con parità pari perché la somma dei bit a 1 e' dispari.
- Quindi viene rilevato un errore.

Sistemi di numerazione

- Un sistema di numerazione e' costituito da un insieme di simboli e di regole mediante il quale si rappresentano numeri.
- Le principali categorie di sistemi di numerazione sono la **posizionale** (a cui appartiene il sistema decimale) e la **additiva** (usata dagli antichi romani).

- I sistemi di numerazione posizionale sono caratterizzati da :

- una **base B** (numero naturale >1);

- un insieme di **B - 1 simboli** diversi {0, 1, 2, ..., B -1}.

Alla sequenza di n + k cifre :

$$b_{n-1},\ b_{n-2}, b_{n-3}, \ldots, b_0 . , b_{-1}, b_{-2},\ldots, b_{-k}$$

si associa il valore numerico che si ottiene dal polinomio:

base

Peso della cifra

$$b_{n-1} x\, B^{n-1} + b_{n-2} x\, B^{n-2} + b_{n-3} x\, B^{n-3} + \ldots + b_1 x\, B^1 + b_0 x\, B^0 + b_{-1} x\, B^{-1} + b_{-2} x\, B^{-2} + \ldots + b_{-k} x\, B^{-k}$$

Valore della cifra

$$N = \sum_{i=-k}^{n} b_i * B^i$$

Sistemi di numerazione

- Sistema di numerazione **decimale :**

  Base = **10**
  Simboli utilizzati **{0, 1, 2, 3, 4, 5, 6, 7, 8, 9,}**

- Sistema di numerazione **binario :**

  Base = **2**
  Simboli utilizzati **{0, 1}**

- Sistema di numerazione **ottale :**

  Base = **8**
  Simboli utilizzati **{0, 1, 2, 3, 4, 5, 6, 7}**

- Sistema di numerazione **esadecimale :**

  Base = **16**
  Simboli utilizzati **{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }**

Sistemi di numerazione

- Questo esempio mostra il numero 2001_{10} rappresentato in binario, ottale, decimale, esadecimale

Binary 1 1 1 1 1 0 1 0 0 0 1

$1\times2^{10} + 1\times2^{9} + 1\times2^{8} + 1\times2^{7} + 1\times2^{6} + 0\times2^{5} + 1\times2^{4} + 0\times2^{3} + 0\times2^{2} + 0\times2^{1} + 1\times2^{0}$

1024 + 512 + 256 + 128 + 64 + 0 +16 + 0 + 0 + 0 + 1

Octal 3 7 2 1

$3\times8^{3} + 7\times8^{2} + 2\times8^{1} + 1\times8^{0}$

1536 + 448 + 16 + 1

Decimal 2 0 0 1

$2\times10^{3} + 0\times10^{2} + 0\times10^{1} + 1\times10^{0}$

2000 + 0 + 0 + 1

Hexadecimal 7 D 1

$7\times16^{2} + 13\times16^{1} + 1\times16^{0}$

1792 + 208 + 1

Decimal	Binary	Octal	Hex
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
20	10100	24	14
30	11110	36	1E

Sistemi di numerazione

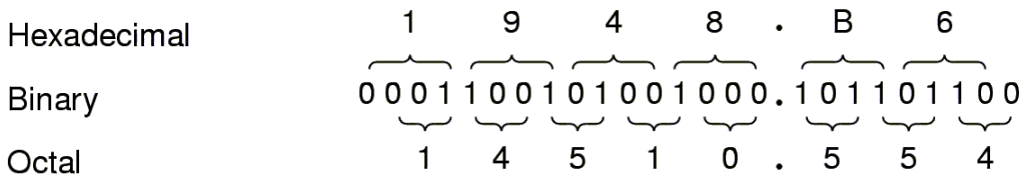
⇨ Conversione tra una base ed un'altra:

Il passaggio tra sistemi di numerazione le cui basi risultino esprimibili come potenze di uno stesso numero risulta particolarmente semplice.

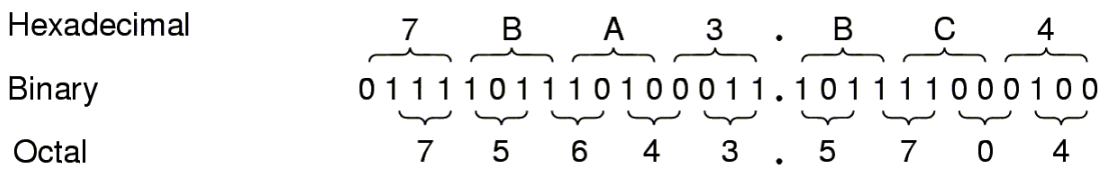
La conversione tra sistema di numerazione ottale, esadecimale e binario e' particolarmente semplice.

⇨ Binario ⟺ Ottale ⟺ Esadecimale

Example 1



Example 2



⇨ **Conversione Binario => Ottale :** Si raggruppino le cifre binarie a gruppi di tre a partire del punto decimale (punto binario) muovendosi sia a destra (parte decimale) che a sinistra (parte intera) del punto decimale. Si sostituisca ad ogni terna di bit una cifra ottale (0 .. 7) in base alla corrispondenza della tabella alla pagina precedente. Puo' essere necessario aggiungere uno o due bit per completare la terna piu' a sinistra o quella piu' a destra.

Sistemi di numerazione

- **Conversione Ottale => Binario :** Ogni cifra ottale e' sostituita dell'equivalente numero binario a tre' bit.

- **Conversione Binario => Esadecimale:** Si procede come nel caso Binario => Ottale operando su gruppi di 4 bit piuttosto che 3 bit poiche' una cifra esadecimale e' codificata in binario con 4 bit.

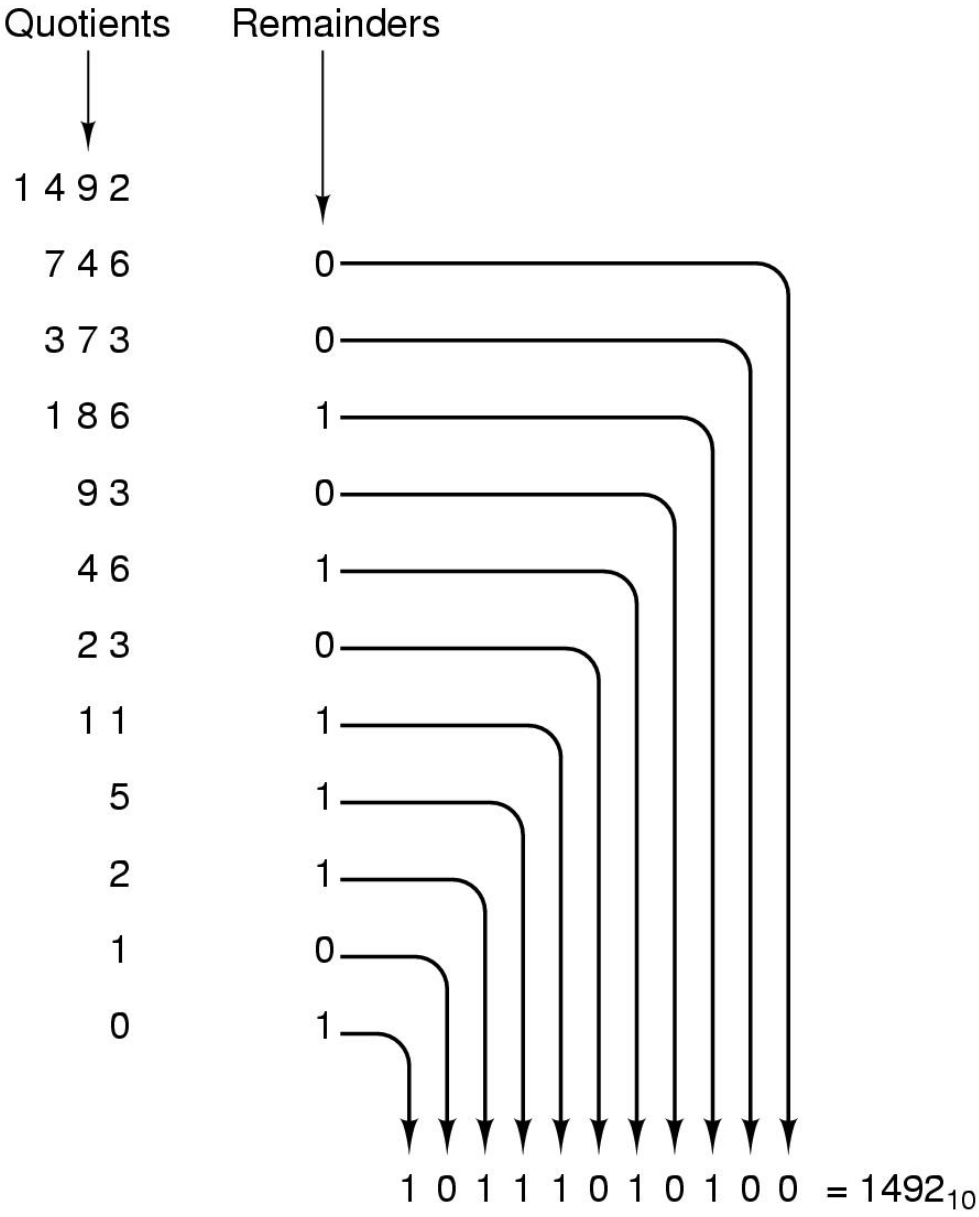
- **Conversione Esadecimale => Binario:** Ogni cifra Esadecimale e' sostituita dell'equivalente numero binario a 4 bit.

- **Conversione da base 10 ad un' altra base (base 2, 8, 16):** Esistono due modi per effettuare tale conversione.

- Il primo metodo consiste nel decomporre il numero in potenze di due. La decomposizione si ottiene sottraendo al numero la piu' grande potenza di due che risulti inferiore la numero stesso e ripetendo questa operazione sul risultato delle differenze fino ad ottenere zero. Il numero binario si ottiene assemblando 1 in corrispondenza delle potenze di due che si sono ottenute dal processo di decomposizione e assemblando 0 in corrispondenza delle potenze mancanti.

Sistemi di numerazione

- Il secondo metodo consiste, per ciò che riguarda la parte intera, nel procedere con divisioni successive per la nuova base a partire dal numero che si vuole trasformare. Il processo di divisione termina quando si ottiene come risultato 0. La sequenza dei resti rappresenta le cifre del numero espresso nella nuova base.



Sistemi di numerazione

- Se il numero possiede anche una parte decimale allora si converte la parte intera con il procedimento indicato in precedenza, mentra sulla parte decimale si opera effettuano moltiplicazioni successive per la nuova base. Al risultato di ogni moltiplicazione si sottrae la parte intera ottenuta, questa rappresenta una cifra del numero espressa nella nuova base, si ripete il processo fino ad ottenere zero o fino al raggiungimento della precisione desiderata.

Es : 127.235_{10} ==> base 2

127 ==> per divisione successiva => 1111111_2

0.235 ==> per moltiplicazioni successive:

0.235	* 2	0.47	0 =>	MSB
0.47	* 2	0.94	0	
0.94	* 2	1.88	1	
0.88	* 2	1.76	1	
0.76	* 2	1.52	1	
0.52	* 2	1.04	1	
0.04	* 2	0.08	0 =>	LSB
	...	...	...	

127.235_{10} => 1111111.0011110_2

Rappresentazione dei numeri con segno

⇨ **Rappresentazione dei numeri interi con segno :** Tre sono le principali forme di rappresentazione dei numeri interi con segno:

n = numero di bit;
N = valore numerico rappresentabile.

- ***Notazione Segno e modulo (signed magnitude);*** il bit piu' a sinistra e' utilizzato come bit di segno (0 +, 1 -) mentre i restanti bit rappresentano il valore assoluto del numero.

$$-2^{(n-1)} < N < 2^{(n-1)}$$

- ***Notazione in complemento a uno (one's complement);*** i numeri positivi vengono rappresentati con il bit più significativo a 0. I numeri negativi sono ottenuti dai numeri positivi effettuando il complemento ad uno. Anche in questa notazione il bit più significativo rappresenta il bit di segno (o +, 1 -). Il complemento ad uno di un numero si ottiene negando i bit che rappresentano il numero stesso.

$$-2^{(n-1)} < N < 2^{(n-1)}$$

- ***Notazione in complemento a due (two's complement);*** i numeri positivi sono ancora rappresentati come nel caso della notazione in complemento ad uno, con il bit più significativo a zero. I numeri negativi sono ottenuti effettuando il **complemento a due** dei numeri positivi. Anche in questo caso il bit piu' significativo rappresenta il bit di segno (0 +, 1 -). Il complemento a due di un numero si ottiene effettuando il complemento ad uno ed aggiungendo uno al risultato.

$$-(2^{(n-1)} + 1) < N < 2^{(n-1)}$$

Rappresentazione dei numeri con segno

- **Esempio di codifiche relative a parole di 4 Bit**

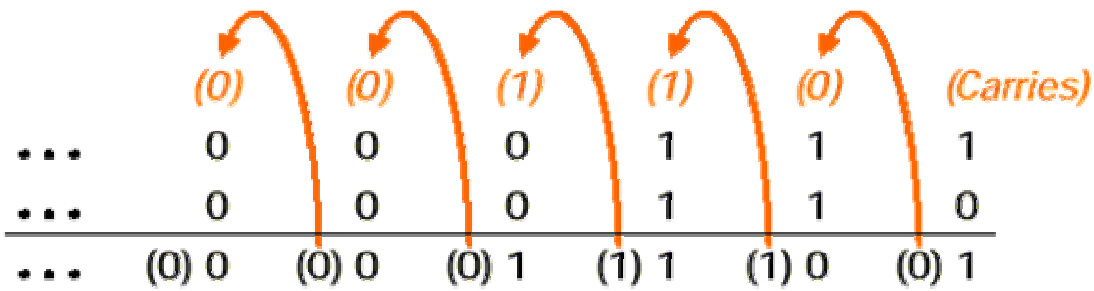
N decimal	- N signed magn.	- N 1's compl.	- N 2's compl.
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	-------
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-------	-------	1000

- Delle tre rappresentazioni la notazione in complemento a due e' quella attualmente utilizzata poiché non presenta la doppia codifica dello zero e permette, nelle operazioni aritmetiche, di trattare il bit di segno con le stesse regole con cui si opera sui restanti bit. Ciò implica una semplificazione delle reti atte ad effettuare le operazioni aritmetiche su numeri binari con segno.

Aritmetica binaria

- **Somma e sottrazione :** Si opera su numeri binari con le stesse regole con cui si effettua somma e sottrazione tra numeri decimali. Si procede operando bit a bit da destra verso sinistra tenendo conto degli eventuali riporti:

$$
\begin{aligned}
&00111_2 = 7_{10}\\
+&00110_2 = 6_{10}\\
&\text{----------}\\
&01101_2 = 13_{10}
\end{aligned}
$$



Aritmetica binaria

La differenza tra sottraendo e minuendo e' riconducibile alla somma del sottraendo con il minuendo cambiato di segno. Quindi anche l'operazione di sottrazione e' riconducibile alla somma.

$7_{10} - 6_{10}$

$$
\begin{array}{rl}
 & 00111_2 = 7_{10} \\
- & 00110_2 = 6_{10} \\
\hline
 & 00001_2 = 1_{10}
\end{array}
$$

$7_{10} + (-6_{10})$

$$
\begin{array}{rl}
 & 00111_2 = 7_{10} \\
+ & 11010_2 = -6_{10} \\
\hline
 & 00001_2 = 1_{10}
\end{array}
$$

- **Condizione di overflow :** Quando si opera con numeri con segno si possono manifestare situazioni in cui il risultato di una operazione di somma non e' correttamente rappresentabile con il numero di bit a disposizione. Se ad esempio si opera con parole a 4 bit e si effettua 3 + 5 si verifica che, entrambi gli addendi risultano correttamente rappresentabili con 4 bit mentre il risultato della somma non risulta correttamente rappresentabile con 4 bit utilizzando la notazione in complemento a due. In questo caso il numero appare negativo poiché il bit di segno e' "perso" ed il suo posto e' stato preso da un bit del modulo "traboccato"

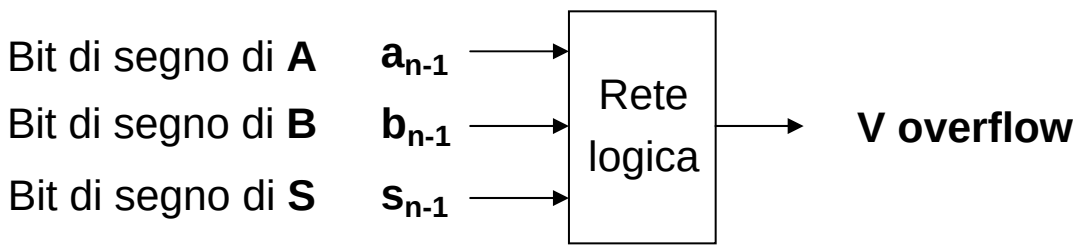
Aritmetica binaria

A (a_3, a_2, a_1, a_0)	0011_2	$= 3_{10}$
B (b_3, b_2, b_1, b_0)	$+0101_2$	$= 5_{10}$
---	---	---
S (s_3, s_2, s_1, s_0)	**1000_2**	**$= -8_{10}$ Overflow**

A (a_3, a_2, a_1, a_0)	1011_2	$= -5_{10}$
B (b_3, b_2, b_1, b_0)	$+1100_2$	$= -4_{10}$
---	---	---
S (s_3, s_2, s_1, s_0)	1 **0111_2**	**$= -8_{10}$ Overflow**

- Si può concludere dicendo che **la condizione di overflow** si manifesta quando la somma di due numeri positivi produce un risultato di segno negativo o quando la somma di due numeri negativi da luogo ad un risultato di segno positivo.

  In base a ciò si può realizzare una rete combinatoria che in base ai segni degli operandi e del risultato fornisce un'uscita che si attiva quando ha luogo la condizione di overflow.



Notazione in virgola mobile

- La notazione scientifica permette un **ampio** intervallo di rappresentazione dei numeri impiegando un numero **limitato** di di cifre. Permette quindi la separazione dell'intervallo di rappresentazione (range) dalla precisione.

$$N = f \times 10^{e}$$

f = mantissa (fraction);

e = esponente, intero;

$0.000001 = 0.1 \times 10^{-5} = 1.0 \times 10^{-6}$

$1941 = 0.1941 \times 10^{4} = 1.941 \times 10^{3}$

- **Il Ragne** e' determinato dal numero di cifre dell'esponente.
- **La precisione** e' determinata dal numero di cifre della mantissa.

Es :

mantissa f **:** tre cifre piu' segno $0.1 \leq |f| < 1$ e 0;

esponente e **:** due cifre piu' segno.

L' intervallo di rappresentazione che si ottiene e':

$$+0.100 \times 10^{-99} \leq |N| \leq +0.999 \times 10^{+99}$$

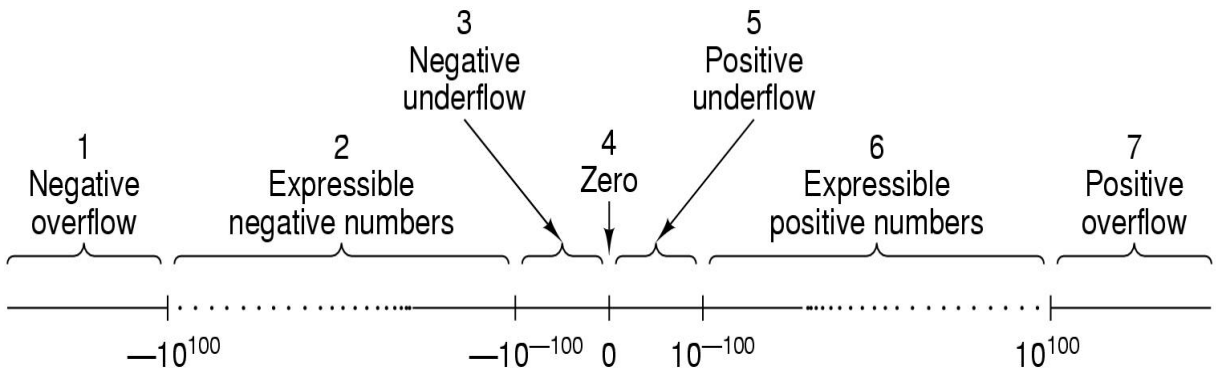
si estende per quasi **199 ordini di grandezza**.

Notazione in virgola mobile

Si rappresentano **179100 numeri positivi**, **179100 numeri negativi**, per un totale di **358201 numeri**, zero compreso.

- Quindi un insieme discreto di valori.

- La figura visualizza su una retta il range di rappresentazione che deriva dall'esempio



- La "distanza" assoluta tra un numero ed il successivo non e' costante sull'intervallo di rappresentazione.

+0.998 x 10 $^{+99}$ e **+0.999 x 10 $^{+99}$** **+0.001 x 10 $^{+99}$**

+0.998 x 10 $^{+0}$ e **+0.999 x 10 $^{+0}$** **+0.001 x 10 $^{+0}$**

- La "distanza" relativa si mantiene pressoché costante su tutto l'intervallo di rappresentazione.

Notazione in virgola mobile

- I calcolatori, per permettere la rappresentazione di numeri non interi fanno uso di una notazione analoga alla notazione scientifica, detta notazione in **Floating Point.**

- La forma generale per un numero in floating point e':

- $\mathbf{(-1)^{s} \times f \times 2^{e}}$

  S = segno del numero, 0 positivo, 1 negativo.

  f = modulo espresso in binario.

  e = esponente (con segno) espresso in binario.

- Per questioni di efficienza li modulo si esprime in **forma normalizzata**; spostando le cifre della mantissa fino ad ottenere la prima cifra significativa a sinistra del punto binario ed aggiustando di conseguenza il valore dell'esponente. In queste condizioni la cifra significativa ed il punto binario si suppongono implicitamente presenti e non vengo memorizzati. Ciò rende disponibile una cifra binaria in più per la rappresentazione delle cifra significative del numero. La stringa di bit cosi ottenuta e' indicata **Significand.**

- **All' esponente** viene sommata una **costante (Bias)** in modo che i valori positivi assumano stringhe binarie caratterizzate dal bit più significativo sempre a 1 mentre i valori negativi siano rappresentati da stringhe binarie che presentano sempre il bit più significativo ad 0.

Notazione in virgola mobile

$$N = (-1)^{S} \times (1 + \text{Significand}) \times 2^{(\text{exponent} - \text{Bias})}$$

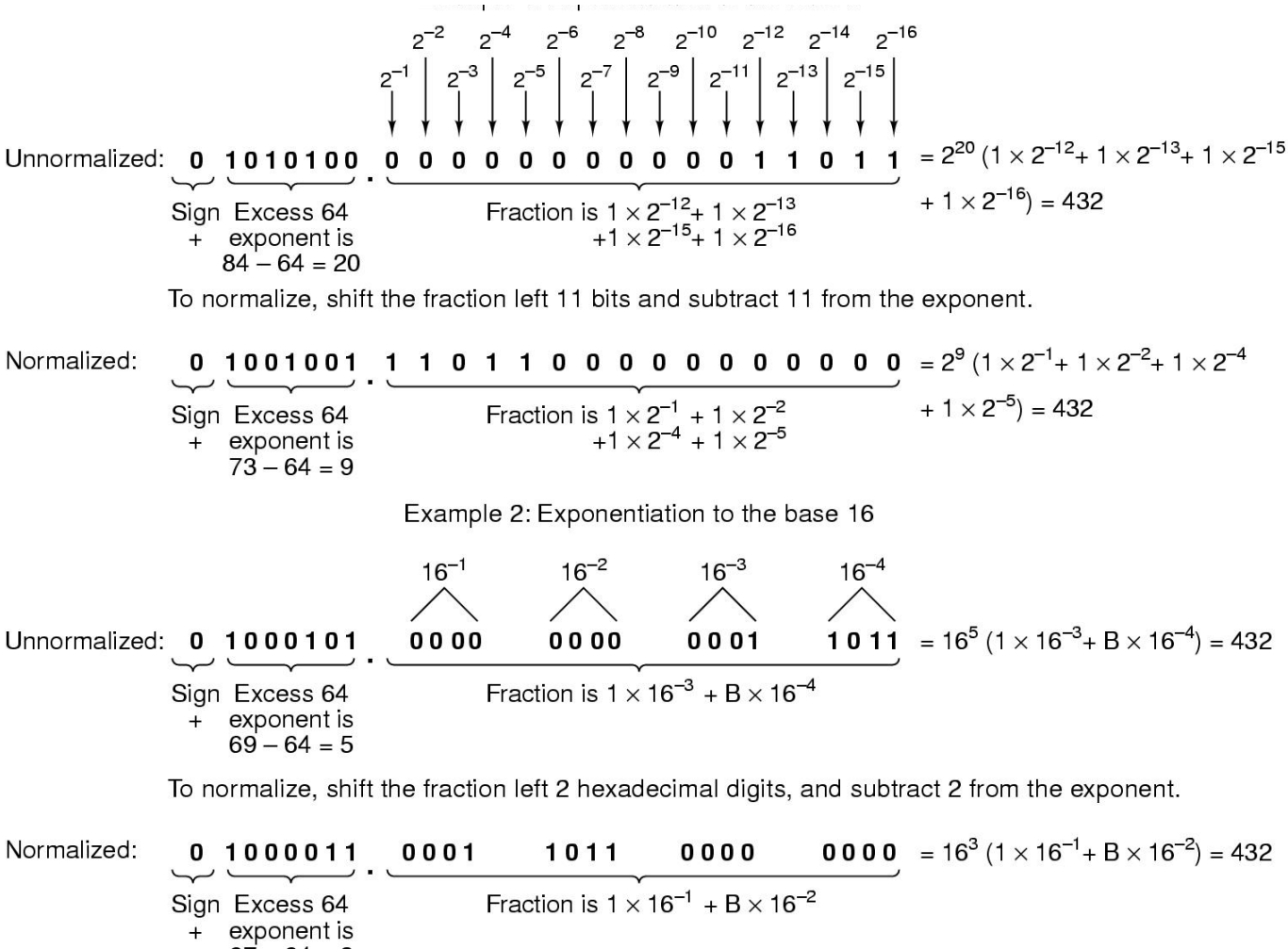
Ex:

Sign = 1 bit

Exponent = 7 bit

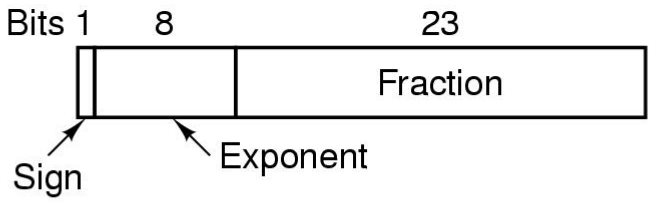
Fraction = 16 bit

Excess = 64

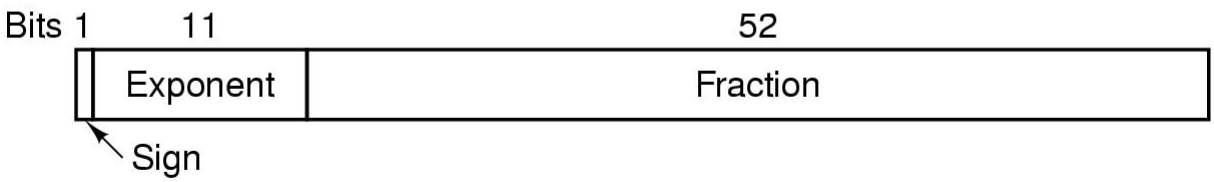


Notazione in virgola mobile

- **Lo standard IEEE 754**



(a)



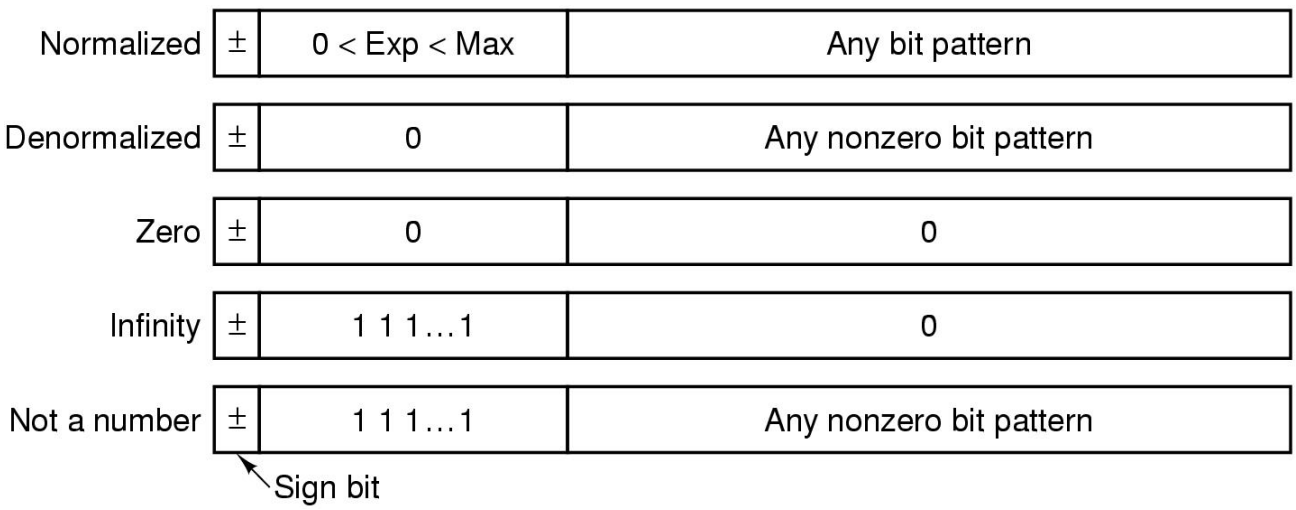
(b)

- La figura (a) rappresenta il formato in **singola precisione** mentre la figura (b) si riferisce a quello utilizzato per la **doppia precisione.**

Notazione in virgola mobile

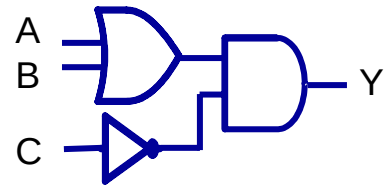
Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exp. system	Excess 127	Excess 1023
Exponent range	-126 to + 127	-1022 to + 1023
Smallest norm. number	2^{-126}	2^{-1022}
Largest norm. number	$\approx 2^{128}$	$\approx 2^{1024}$
Decimal Range	$\approx 10^{-38}$ to 10^{+38}	$\approx 10^{-308}$ to 10^{+308}
Smallest denorm. Numb.	$\approx 10^{-45}$	$\approx 10^{-324}$

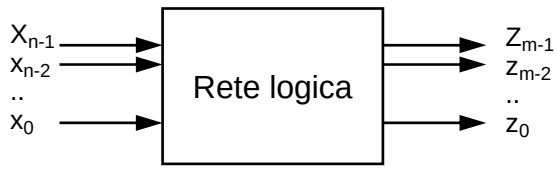
IEEE 754



Reti logiche

- Livello di astrazione che studia i sistemi digitali a livello di componenti LOGICI elementari ***indipendentemente dalla tecnologia con cui il sistema viene realizzato***. Il sistema digitale e' rappresentato
- come descrizione funzionale
  - segnali binari di ingresso e uscita
- come descrizione strutturale
  - blocchi logici elementari (GATE ELEMENTARI)



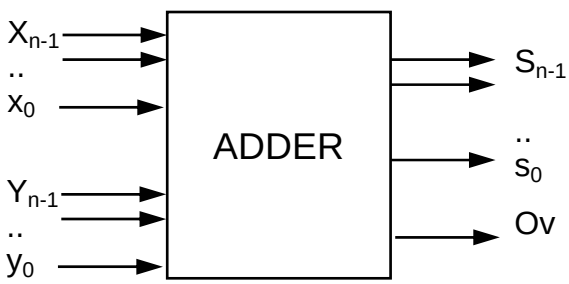


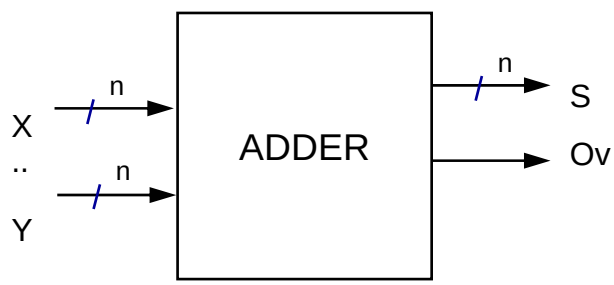
- **Rete logica**: rappresentazione astratta di un sistema digitale avente *n* segnali binari di ingresso ed *m* segnali binari di uscita.
- I segnali sono grandezze funzioni del tempo

$X = \{x_{n-1}(t), \ldots\ldots x_0(t)\}$ $\quad Z = \{z_{m-1}(t), \ldots\ldots z_0(t)\}$

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

- I segnali di ingresso ed uscita delle reti logiche possono essere singoli segnali binari (es. RESET) o segnali digitali composti in parole codificate come un insieme di segali binari





Reti logiche

Le reti logiche costituiscono l'hardware dei sistemi di elaborazione digitali

I fondamenti di analisi e sintesi delle reti logiche sono lo strumento essenziale per:

1) *la progettazione e realizzazione di ASIC* Application Specific Integrated Circuit

reti logiche complesse (porte, processori dedicati, controllori), spesso proprietari

2) *la progettazione e realizzazione della logica discreta nella progettazione a livello di sistema*

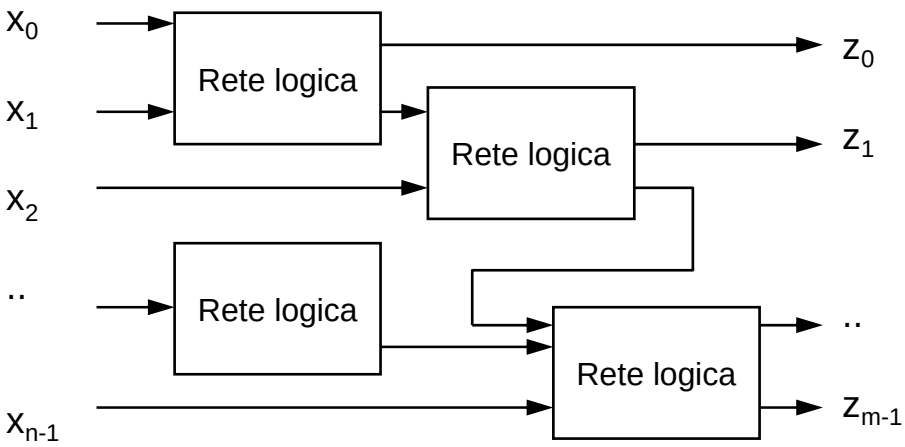
(interfacce tra componenti, logica nelle schede

di tipo generale, sistemi embedded con processori…)

3) *la progettazione e la conoscenza dei calcolatori elettronici* a livello di logica digitale e per comprendere i livelli di astrazione superiori

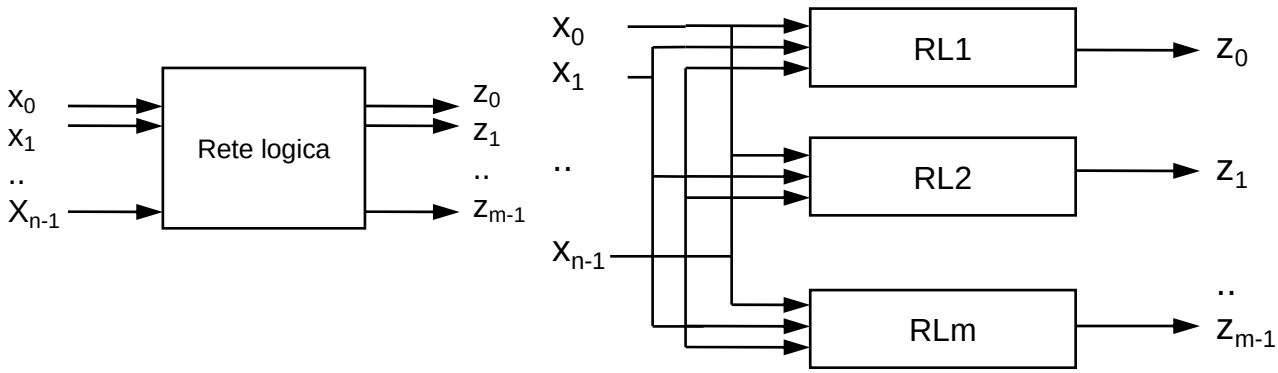
Proprietà delle reti logiche

- **Proprietà di 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, e' ancora una rete logica



- **Proprietà di decomposizione:** una rete logica complessa può essere decomposta in reti logiche più semplici (fino all'impiego di soli blocchi o gate elementari)

- **Proprietà di decomposizione in parallelo**: una rete logica a m uscite può essere decomposta in m reti logiche ad 1 uscita, aventi ingressi condivisi



Reti combinatorie e sequenziali

- Reti COMBINATORIE $z_i(t)=f(x_0(t),,,,x_{n-1}(t)\}$
- Reti SEQUENZIALI $z_i(t)=f((x_0(t),,,,x_{n-1}(t), t\}$
- **Rete combinatoria:** ogni segnale di uscita dipende solo dai valori degli ingressi in quell'istante
- **Rete sequenziale**: ogni segnale di uscita dipende dai valori degli ingressi in quell'istante e dai valori che gli ingressi hanno assunto negli istanti precedenti
- Rete combinatoria: rete senza memoria (l'uscita cambia "istantaneamente" dopo che l'ingresso e' cambiato)
- Rete sequenziale: rete con memoria; e' una rete in cui l'uscita cambia in funzione del cambiamento dell'ingresso e della specifica configurazione interna in quell'istante (STATO)

Esempi di reti combinatorie e sequenziali

Esempio di **rete combinatoria:**

Conversione di valori BCD su display a sette segmenti

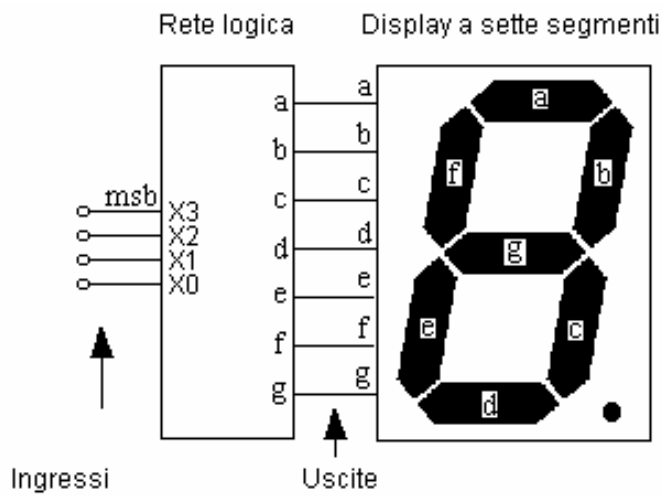
- Descrizione comportamentale (a parole) :

  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.

- Es : 15 decimale
  1111 binario
  0001 0101 BCD

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



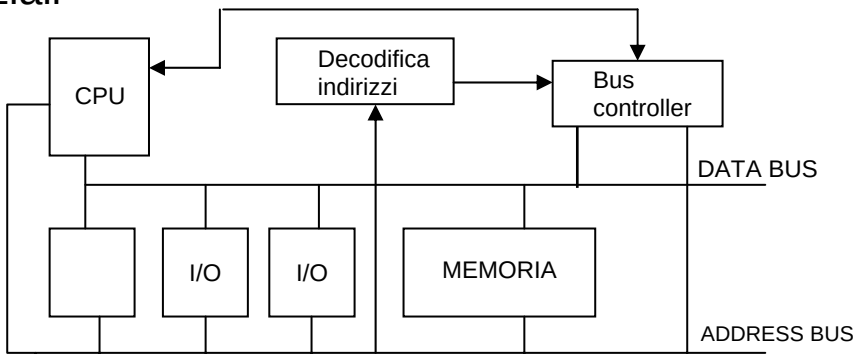
Esempio di **rete sequenziale:**

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 giu' 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 e' necessario conoscere il piano corrente che indica lo stato interno.

Reti logiche nei calcolatori elettronici

- I componenti fondamentali di un calcolatore elettronico sono reti sequenziali



- La CPU e' una rete sequenziale che passa attraverso stati interni leggendo decodificando ed eseguendo istruzioni. La memoria e' una rete sequenziale in cui lo stato interno dipende dal dato scritto precedentemente. Le periferiche di I/O sono macchine sequenziali più o meno complesse. La decodifica degli indirizzi e' una rete combinatoria.

- Nella CPU:
- CONTROL UNIT: e' una rete sequenziale che in base allo stato attuale e agli stati precedenti genera tutti i segnali di controllo interni e di interfaccia esterna della CPU

- DATA PATH: comprende tutti i blocchi funzionali impiegati nella esecuzione di una istruzione; unita' di calcolo (ALU e unita' floating point), registri, operandi e registri di stato; i registri sono reti sequenziali mentre la ALU e' una rete combinatoria

Descrizione delle reti combinatorie

1) **Descrizione comportamentale a parole:** descrizione a parole del comportamento della rete logica (poco formale e precisa)

2)**Tabelle di verita':** descrizione esaustiva di tutte le configurazioni di uscita per ogni possibile configurazione di ingresso

3) **Mappe:** altra rappresentazione delle tabelle della verita'

4) **Espressioni dell'algebra Booleana**

5) **Schema logico:** descrizione strutturale

6) **Forme d'onda:** descrizione comportamentale in funzione del tempo

7) **Linguaggi di descrizione dell'hardware**

- **Tabella di verita'**: tabella che associa tutte le possibili combinazioni degli ingressi alle corrispondenti configurazioni delle uscite e indica esaustivamente il comportamento della rete logica
- Se la rete combinatoria ha n ingressi e m uscite, allora la tabella di verita' ha (n+m) colonne e 2^n righe

$X_{n-1} \dots X_1 X_0$	$Z_{m-1} \dots Z_1 Z_0$
Tutte le possibili configurazioni di ingresso	Le uscite corrispondenti

- Oppure per la proprietà di decomposizione si possono definire tante tabelle quante sono le uscite

Tabelle di verita’

- Si dicono **COMPLETAMENTE SPECIFICATE** se ogni valore della tabella assume il valore logico di vero o falso (1, 0)

- Si dicono **NON COMPLETAMENTE SPECIFICATE** se contengono condizioni di indifferenza. Si verifica in due casi:

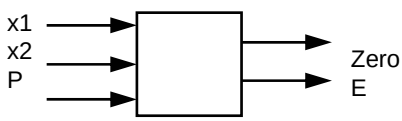
C.1) *se alcune configurazioni di ingressi sono vietate*

Es: conversione BCD 7 segmenti

x3	*x2*	*x1*	*x0*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	-	-	-	-	-	-	-
1	0	1	1	-	-	-	-	-	-	-
1	1	0	0	-	-	-	-	-	-	-
1	1	0	1	-	-	-	-	-	-	-
1	1	1	0	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-

C.2) *se le uscite sono indifferenti per alcune configurazioni di ingresso*

Esempio: progettare una rete che indichi se due ingressi binari sono entrambi uguali a zero, se il segnale di parità pari e’ corretto, altrimenti indichi errore



x1	x2	P	Zero	E
0	0	0	1	0
0	0	1	-	1
0	1	0	-	1
0	1	1	0	0
1	0	0	-	1
1	0	1	0	0
1	1	0	0	0
1	1	1	-	1

Mappe

- **Mappa:**

  Rappresentazione piu' compatta della tabella di verita'

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	-	-

a

- E' una rappresentazione matriciale della tabella in cui le righe indicano tutte le possibili configurazioni di un sottoinsieme delle variabili di ingresso e le colonne tutte le configurazioni delle variabili rimanenti, il valore nelle celle indica il valore dell'uscita nella configurazione corrispondente

- OGNI CELLA CORRISPONDE AD UNA CONFIGURAZIONE DELLE VARIABILI

- **Mappe di Karnaugh**: Mappe in cui le configurazioni successive in ogni lato sono ADIACENTI
- due configurazioni sono adiacenti se differiscono di un solo bit
- due celle sono adiacenti se corrispondono a configurazioni adiacenti

Mappe d Karnaugh

K-mappe a 2 variabili

x1 \ x0	0	1
0	0	0
1	1	1

K-mappe a 3 variabili

x2 \ x1,x0	00	01	11	10
0	0	0	-	1
1	1	1	1	1

K-mappe a 4 variabili

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

Criteri geometrici di adiacenza:

- 1 lato in comune
- un'estremita' di colonna
- un'estremita' di riga
- stessa posizione in sottomatrici adiacenti

K-mappe a 5 variabili

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

X4=0

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

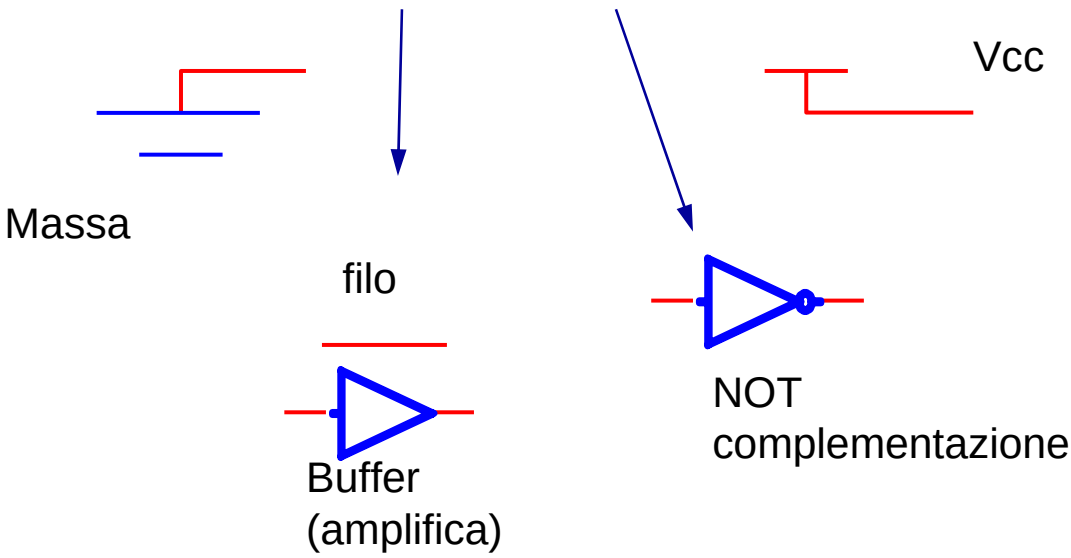
X4=1

Funzioni combinatorie e gate elementari

- Le reti logiche combinatorie sintetizzano funzioni combinatorie.
- Per ogni n, e' finito il numero di funzioni combinatorie di n variabili di ingresso. Alcune funzioni combinatorie elementari hanno una rappresentazione logica e grafica elementare (gate)

Funzioni di 1 sola variabile indipendente

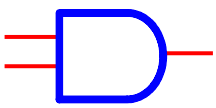
x	z3	z2	z1	z0
0	0	0	**1**	1
1	0	1	**0**	1



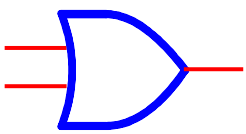
Funzioni di 2 variabili indipendenti

x1 x0	z0	z1	z2	z3	z4	z5	z6	z7
0 0	0	**0**	0	0	0	0	**0**	**0**
0 1	0	**0**	0	0	1	1	**1**	**1**
1 0	0	**0**	1	1	0	0	**1**	**1**
1 1	0	**1**	0	1	0	1	**0**	**1**

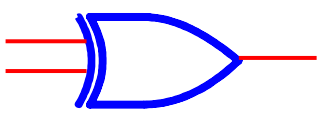
z1: AND vale 1 se e solo se tutti gli ingressi valgono 1
equivale al prodotto logico in logica positiva



z7: OR vale 1 se e solo se almeno uno degli ingressi vale 1
equivale alla somma logica in logica positiva



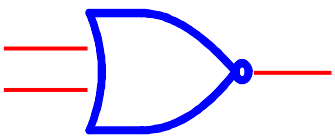
z6: EXOR vale 1 se e solo se x1 o x2 valgono 1 ma non entrambi



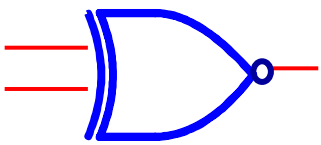
Funzioni di 2 variabili indipendenti

x1 x0	z8	z9	z10	z11	z12	z13	z14	z15
0 0	**1**	**1**	1	1	1	1	**1**	1
0 1	**0**	**0**	0	0	1	1	**1**	1
1 0	**0**	**0**	1	1	0	0	**1**	1
1 1	**0**	**1**	0	1	0	1	**0**	1

z8: NOR vale 1 se e solo se ne' x1 ne' x2 valgono 1
l'uscita e' il complemento di z7

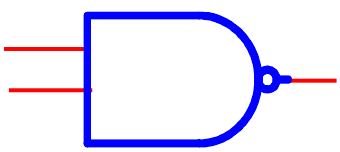


z9: EXNOR



EQUIVALENCE: vale 1 se e solo se x1 o x2 sono uguali
l'uscita e' il complemento di z6

z14: NAND



vale 0 se e solo se ne' x1 ne' x2 valgono 0
l'uscita e' il complemento di z1

Funzioni combinatorie

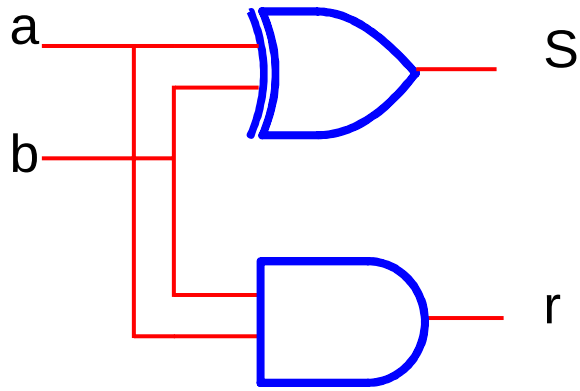
- Quante sono le possibili funzioni binarie di n variabili ?
- Tutte le combinazioni delle uscite per ogni configurazione di ingresso, ossia 2 elevato al numero delle possibili configurazioni di ingresso

$$N.\ conf = 2^{(2^n)}$$

- Una rete logica puo' essere descritta attraverso la interconnessione di GATE ELEMENTARI
- La descrizione strutturale corrisponde allo schema logico. Gli schemi logici rappresentano reti logiche **indipendenti** dalla loro realizzazione tecnologica;

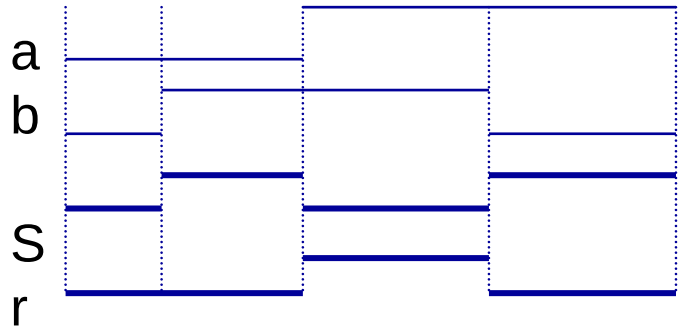
- **Esempio**: Progettare un HALF ADDER ossia un sommatore senza riporto di ingresso

b a	S	r
0 0	0	**0**
0 1	1	**0**
1 0	1	**0**
1 1	0	**1**



Forme d'onda

- Una rete logica puo' essere descritta dal punto di vista comportamentale anche con una forma d'onda che indica il legame tra l'input e l'output nel tempo



- Dal punto di vista logico le uscite cambiano *instantaneamente* con il cambiamento degli ingressi (nei gate ideali non ci sono ritardi)
- Gate reali: introducono ritardi e possibili variazioni non stazionarie delle uscite (glitch)

Non vengono considerati problemi di tecnologia come:

- *FAN-IN:* n. max di input di un gate logico
- *FAN-OUT:* n. di input a cui l'uscita del gate e' connessa
- *Ritardi:* si considerano gate a ritardo nullo e con connessioni a ritardo nullo
- *Consumi*

Descrizione VHDL della rete half_adder

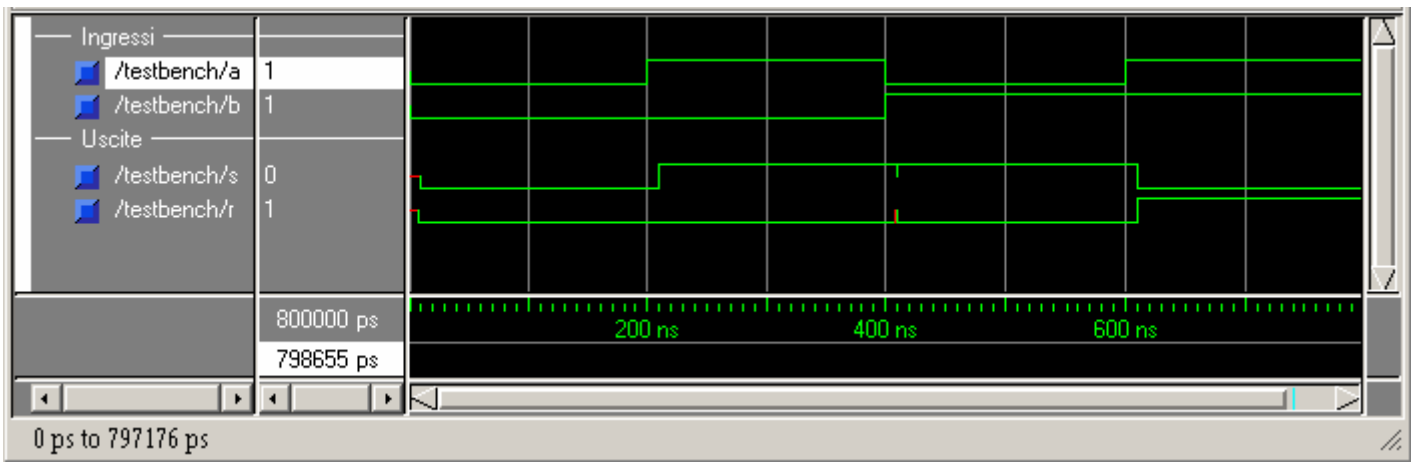
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_adder is
    Port ( a : in std_logic;
           b : in std_logic;
           S : out std_logic;
           R : out std_logic);
end half_adder;

architecture Struct of half_adder is
begin

           S <= a xor b;
           R <= a and b;

end Struct;
```

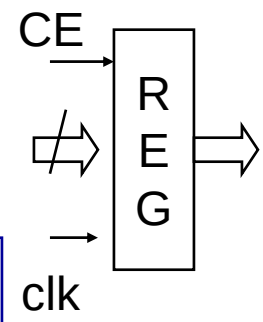


Descrizione comportamentale con VHDL

- VHDL **V**HSIC **H**ardware **D**escription **L**anguage
  (Very High Speed Integrated Circuits)
  Standard IEEE 1076 (1987) e 1164 (1991)

- Linguaggio di descrizione dell'hardware usato per:
  - design specification (per progetti complessi, dichiarazione dei requisiti, specifica delle prestazioni)
  - design capture (come ingresso per il CAD\CAE)
  - design simulation (sia funzionale che temporale)
  - design documentation (standard DoD americano)

- Esempio di rete sequenziale: il **Registro**
- descrizione a parole: e' una rete logica che se CE=1, quando il segnale di clock va alto, campiona i segnali di ingresso e li porta in uscita, mantenendoli per tutto il tempo in cui il segnale di clock non torna ad 1.



Definizione dell'interfaccia

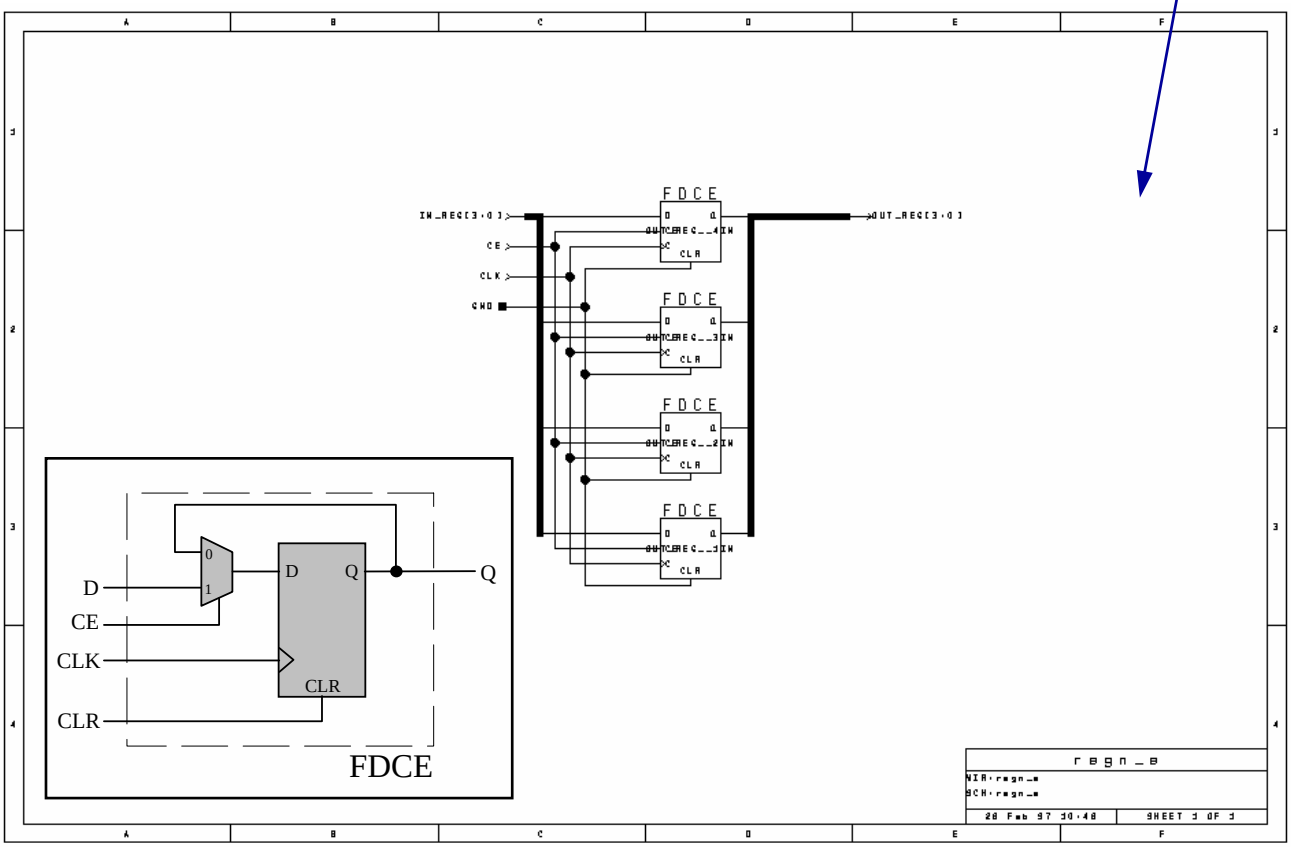
```
entity regn_e is
generic (N : integer := 4);
port (  signal IN_REG        : in vlbit_1d(N-1 downto 0);
        signal CE, CLK       : in vlbit;
        signal OUT_REG       : out vlbit_1d(N-1 downto 0));
end regn_e;
```

Esempio

```
architecture bhv of regn_e is
begin
  PROCESS
  BEGIN
    wait until (prising (CLK));
      if (CE='1') then
     OUT_REG <= IN_REG;
      end if;
  END PROCESS;
 end bhv;
```

Descrizione comportamentale

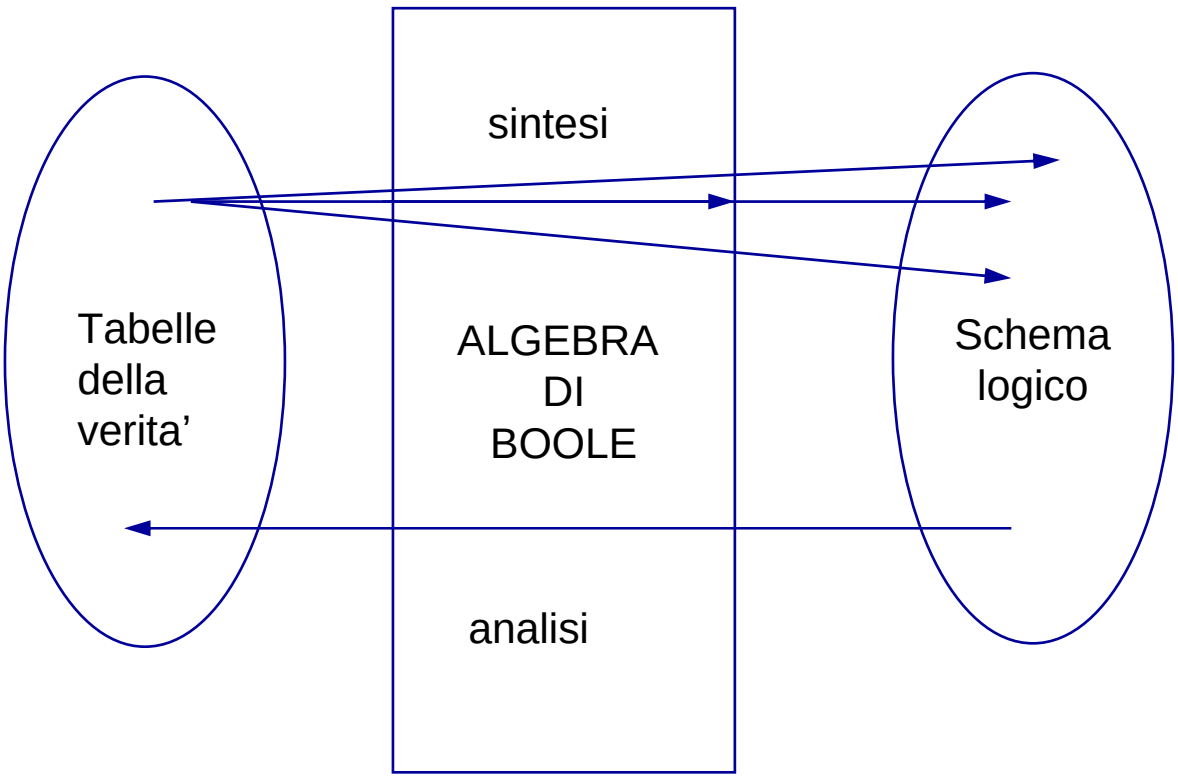
Dopo la sintesi logica con una data libreria…..



Algebra di Boole

- Uno strumento potente di rappresentazione delle reti logiche combinatorie e' data dalle espressioni dell' ALGEBRA DI BOOLE o ALGEBRA DI COMMUTAZIONE

- e' il sistema matematico usato per la sintesi e per l'analisi, per passare dalle tabelle della verità allo schema logico e viceversa



Algebra di Boole

- L'algebra di Boole e' un sistema matematico che descrive funzioni di variabili binarie: e' composto da
  - un insieme di simboli **B={0,1}**
  - un insieme di operazioni **O={+,•,'}**
    - \+ somma logica
    - • prodotto logico
    - ' complementazione
  - un insieme di postulati (assiomi) **P:**

P1) 0+0=0	P5) 0 •0= 0	P9) 0' =1
P2) 0+1=1	P6) 0 •1= 0	P10) 1' =0
P3) 1+0=1	P7) 1 •0= 0	
P4) 1+1=1	P8) 1 •1= 1	

Proprieta' di chiusura:

per ogni a, b $\in$B

$a+b \in B$

$a \bullet b \in B$

- **COSTANTI** dell'algebra: le costanti 0 ed 1
- **VARIABILE:** un qualsiasi simbolo che puo' essere sostituito da una delle due costanti

Funzioni Booleane

- Una **funzione completamente specificata** *di n variabili* $f(x_{n-1},..., x_1,x_0)$ e' l'insieme di tutte le possibili coppie formate da un elemento di B^n (dominio) e da un elemento di B (codominio).

- La tabella della verita' e' un tipico modo per descrivere una funzione dell'algebra di Boole.

- Esiste corrispondenza 1:1 tra una tabella della verita' e funzione Booleana.

$f(x2,x1,x0):BxBxB \rightarrow B$

x2	x1	x0	f(x2,x1,x0)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- **Complementazione** : il valore complementato di A si indica come A' oppure A.
- A e A' non sono due variabili diverse: ci si riferisce a questi due simboli come a due **letterali**; letterale e' una variabile logica associata o meno al simbolo di inversione di valore. A e A' sono due diversi letterali di una stessa variabile.
- Il simbolo • del prodotto logico viene spesso omesso.

Espressioni Booleane

Un'**espressione** secondo l'algebra di Boole e' una stringa di elementi di B che soddisfa una delle seguenti regole:

- una costante e' un'espressione;
- una variabile e' un'espressione;
- se X e' un'espressione allora il complemento di X e' un'espressione;
- se X,Y sono espressioni allora la somma logica di X e Y e' un'espressione;
- se X,Y sono espressioni allora il prodotto logico di X e Y e' un'espressione.

TEOR: *ogni espressione di n variabili descrive una funzione completamente specificata che puo' essere* ***valutata*** *attribuendo ad ogni variabile un valore assegnato.*

es: dalla tabella della verita' precedente:

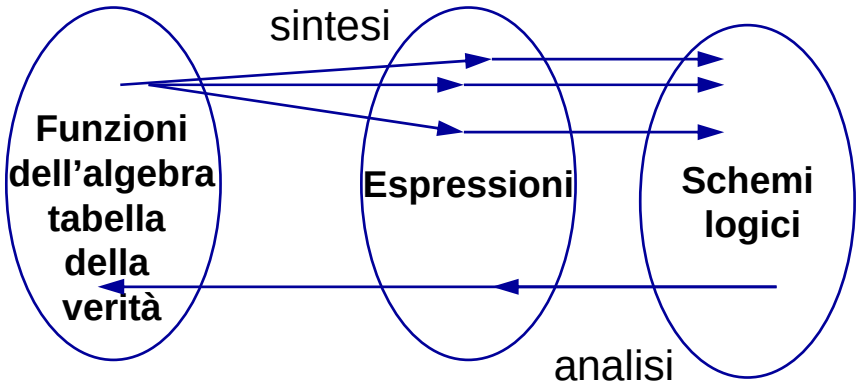
f(x3,x2,x1)= x1'x2'x3+x1x2'x3'+x1x2x3

⇨ Se ogni espressione definisce univocamente una funzione non e' vero il contrario: per ogni funzione esistono piu' espressioni che la descrivono e si dicono logicamente **equivalenti.** Espressioni equivalenti godono delle proprietà simmetrica, riflessiva e transitiva

TEOR: *una espressione di n variabili descrive in maniera univoca uno schema logico di AND, OR e NOT*

Analisi di uno schema logico

- Dallo schema logico tramite le espressioni e' possibile ricavare il comportamento di una rete logica

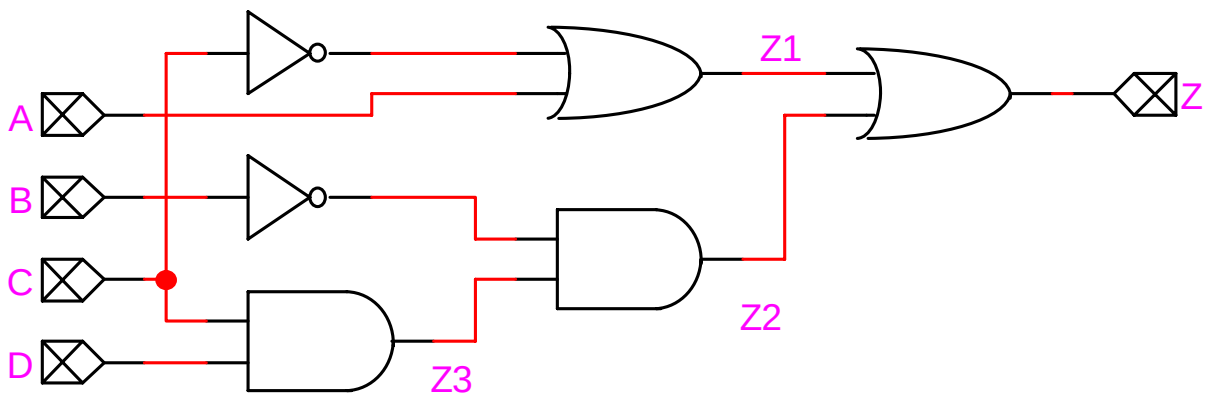


Analisi:

1) nominando tutte le uscite dei gate logici

2) per sostituzione a partire dalle uscite si ottiene una funzione Booleana delle sole variabili di ingresso

Esercizio: Eseguire l'analisi del seguente schema



Teoremi dell'algebra di Boole

- Si possono dimostrare per
  - induzione perfetta
  - induzione matematica
  - manipolazione algebrica

Principio di Dualita':

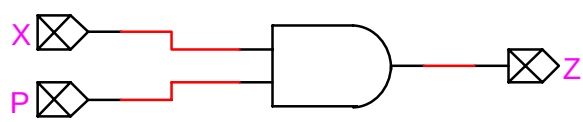
- *ogni espressione algebrica presenta una forma duale ottenuta scambiando l'operatore OR con AND, la costante 0 con la costante 1 e mantenendo i letterali invariati.*
- *ogni proprieta' vera per un' espressione e' vera anche per la sua duale.*
- il principio di dualita' e' indispensabile per trattare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

Teor. di Identità

- (T1) $X + 0 = X$ (T1') $X \cdot 1 = X$

Teor. di Elementi nulli

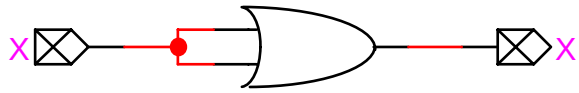
- (T2) $X + 1 = 1$ (T2') $X \cdot 0 = 0$
- sono molto utili nella sintesi di reti logiche: gli elementi nulli permettono di "lasciar passare" un segnale di ingresso in determinate condizioni
- ad es: progettare una rete logica che fornisca in uscita il valore di X se un pulsante P viene premuto altrimenti l'uscita valga sempre 0



Teoremi dell'algebra di Boole

Idempotenza

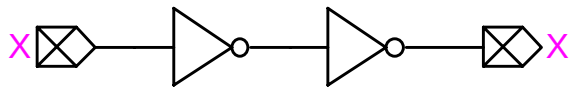
- (T3) $X + X = X$
- (T3') $X \cdot X = X$



si usa per l'amplificazione dei segnali ed eliminazione disturbi

Involuzione

- (T4) $(X ')' = X$

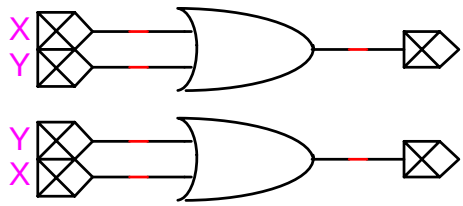


Complementarieta'

- (T5) $X + X' = 1$
- (T5') $X \cdot X' = 0$

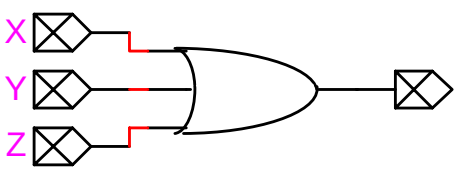
Proprieta' commutativa

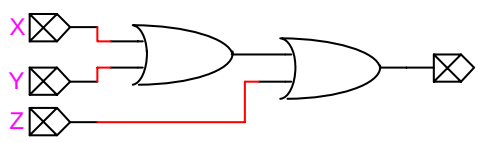
- (T6) $X + Y = Y + X$
- (T6') $X \cdot Y = Y \cdot X$



Proprieta' associativa

- (T7) $(X + Y) + Z = X + (Y + Z) = X + Y + Z$
- (T7') $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$





Teoremi dell'algebra di Boole

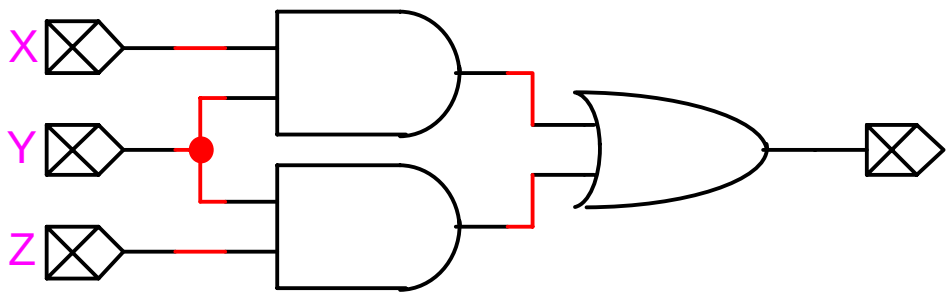
Proprieta' di assorbimento

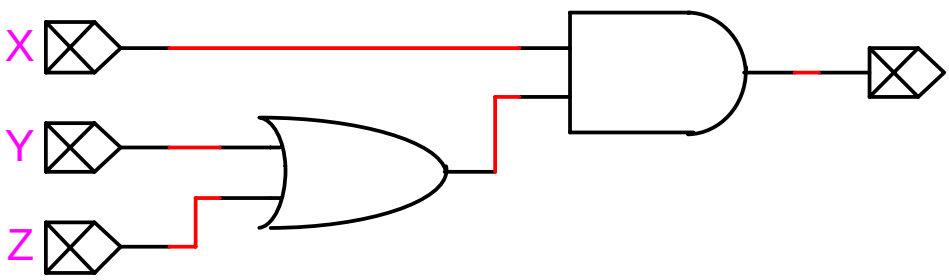
- (T8) $X + X \cdot Y = X$
- (T8') $X \cdot (X + Y) = X$

permette di minimizzare il n. di gate

Proprieta' distributiva

- (T9) $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$
- (T9') $(X + Y) \cdot (X + Z) = X + Y \cdot Z$





Teoremi dell'algebra di Boole

Proprieta' della combinazione

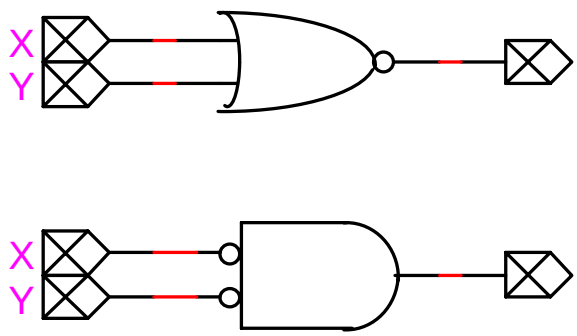
- (T10) $(X + Y) \cdot (X' + Y) = Y$
- (T10') $X \cdot Y + X' \cdot Y = Y$

Proprieta' del consenso

- (T11) $(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$
- (T11') $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$

Teorema di De Morgan

- (T12) $(X + Y)' = (X' \bullet Y')$
- (T12') $(X \bullet Y)' = (X' + Y')$
- vale per n variabili



Esercizi

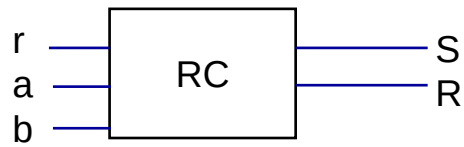
Esercizio 1: calcolare la tabella di verita' per una rete logica che fornisca la parita' pari per un qualsiasi dato in ingresso a 3 bit.

Esercizio 2: dato un segnale a 4 bit (con ultimo bit di parita') indicare la tabella di verita' di una rete logica che verifichi se la parita' e' corretta.

Esercizio 3: compilare le mappe di Karnaugh per il display a 7 segmenti.

Esercizio 4: compilare la mappa di Karnaugh per la rete logica dell'esercizio 2 .

Esercizio 5: Il FULL ADDER: progettare la rete logica che esegua l'addizione ad 1 bit con riporto di ingresso e di uscita



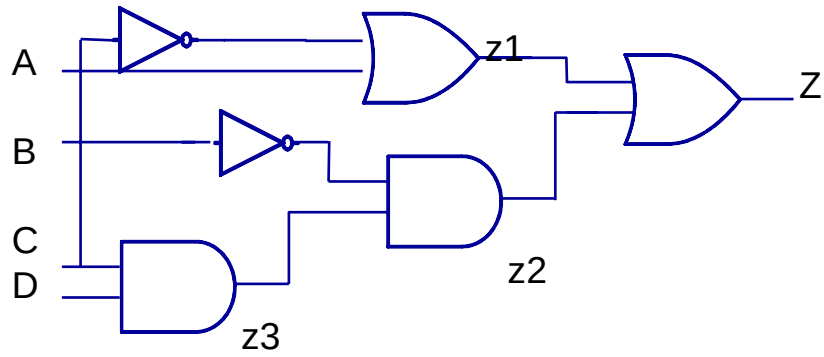
Esercizio 6: valutare la seguente espressione

z= A'B +C((A'+B')+ C') + B

quanto vale Z se (A,B,C)=(0,0,1)?

Esercizi

Esercizio 7: analizzare e semplificare la rete logica impiegando le regole dell'algebra Booleana



Esercizio 8: scrivere la tabella di verita' della seguente rete logica:

- Il riporto d'uscita vale 1 se almeno un ingresso vale 1 e il riporto di ingresso vale 1 oppure se il riporto di ingresso vale 0 ma entrambi gli ingressi valgono 1.
- Semplificare con le regole dell'algebra Booleana

Esercizio 9: Viene fornita una assicurazione: se uomo e o ha meno di 30 anni o ha piu' di 30 anni ed ha figli; se ha piu' di 30 anni ma non ha figli e ,o e' uomo o e' sposato ;se ha piu' di 30 anni ma non ha figli e non e' sposato

Valutazione: Riceve una assicurazione una donna con figlio non sposata e con meno di 30 anni?

Soluzioni

Esercizio 1:

x2	x1	x0	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Esercizio2:

x2	x1	x0	P	E
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Esercizio 5:

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
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Descrizione vhdl del full_adder

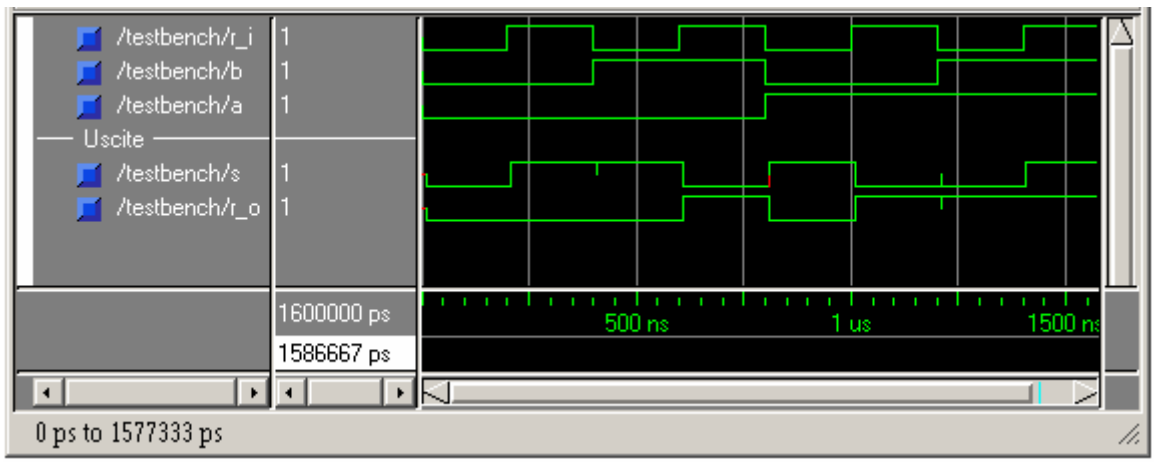
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is
    Port ( a        : in     std_logic;
           b        : in     std_logic;
           r_i      : in     std_logic;
           s        : out    std_logic;
           r_o      : out    std_logic);
end full_adder;

architecture Data_flow of full_adder is

begin
           s     <= a xor b xor r_i;
           r_o   <= (a and b) or (b and r_i) or (r_i and a);

end Data_flow;
```



Soluzioni

Esercizio 6:

Dalla valutazione ne deriva che z=1

Esercizio 7:

z=z1+z2 z1=A+z4 z4=C’ z2=z5z3

z3=CD z5=B’

z=A+C’+B’CD

pe il Teor. di assorbimento

z=A+C’+C’B’D+B’CD

z=A+C’+B’D

Esercizio 9: sia A uomo, B sposato, C >30 annui, D figli

f= A(C’+CD)+CD’ (A+B)+CB’D’

VALUTAZIONE: no

Esercizi Proposti

- **Esercizi di logica combinatoria tratti da Cap. II di R.Katz – Contemporary Logic Design.**

- 1. (Gate Logic) Draw schematics for the following functions in terms of AND, OR, and inverter gates.

- **Vedere: 2.1.1 Boolean Operations Revisited, Pag. 41.** R.Katz – Contemporary Logic Design.

  a) X (Y + Z)

  b) X Y + X Z

  c) (X (Y + Z))'

  d) X' + Y' Z'

  e) W (X + Y Z)

- 2. (Gate Logic) Draw the schematics for the following functions using NOR gates and inverters only:

  a) [X' + (Y + Z)']'

  b) [(X' + Y')' + (X' + Z')']'

- 3. (Gate Logic) Draw the schematics for the following functions using NAND gates and inverters only:

  a) [X (Y' Z')]'

  b) XY + XZ

Esercizi Proposti

- 7. (Laws and Theorems of Boolean Algebra) Prove the following simplification theorems using the first eight laws of Boolean algebra:
- **Vedere: 2.2.1 Laws and Theorems of Boolean Algebra, Pag . 50.** R.Katz – Contemporary Logic Design.

  a) (X + Y) (X + Y') = X

  b) X (X + Y) = X

  c) (X + Y') Y = XY

  d) (X + Y) (X' + Z) = XZ+X'Y
- 8. (Laws and Theorems of Boolean Algebra) Verify that OR and AND are duals of each other.

  a) NOR and NAND are duals of each other.

  b) XNOR and XOR are duals of each other.

  c) XNOR is the complement of XOR: (XY' + X'Y)' = X'Y' + XY
- 9. (Laws and Theorems of Boolean Algebra) Prove, using truth tables, that

  ⇨ XY + YZ + X'Z = XY +X'Z

Esercizi Proposti

- 10. (Laws and Theorems of Boolean Algebra) Use DeMorgan's theorem to compute the complement of the following Boolean expressions:

  a) A (B + CD)

  b) ABC + B(C' + D')

  c) X' + Y'

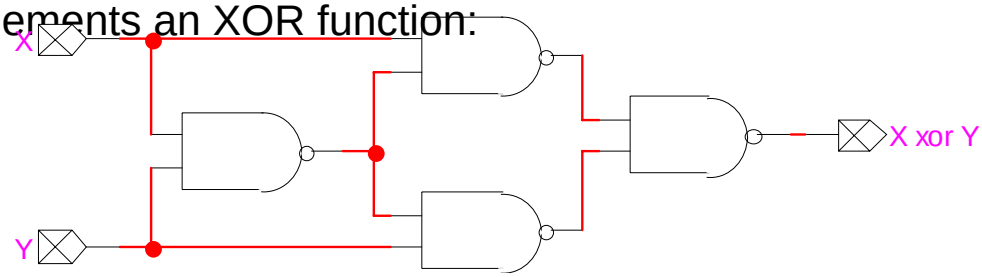
  d) X + YZ'

  e) (X + Y)Z

  f) X +(XY)'

  g) X(Y+ZW' + V'S)

- 12. (*L*aws and Theorems of Boolean Algebra) Using Boolean algebra, verify that the schematic of Figure implements an XOR function:



- 13. (Boolean Simplification) Simplify the following functions using the theorems of Boolean algebra. Write the particular law or theorem you are using in each step. For each simpli\xde ed function you derive, how many literals does it have?

Esercizi Proposti

a) f(X,Y) =XY + XY'

b) f(X,Y) =(X + Y) (X + Y′)

c) f(X,Y,Z)= YZ' + X'YZ + XYZ

d) f(X,Y,Z)=(X + Y) (X′ + Y + Z) (X′ + Y +Z′)

)f(W,X,Y,Z)=X + XYZ + X′YZ + X′Y +WX + W′X

⇨ 14.(Boolean Simplification) Consider the function f(A,B,C,D) = (AD +A′C) [B′(C+BD′)].

a) Draw its schematic using AND, OR, and inverter gates.

b) Using Boolean algebra, put the function into its minimized form and draw the resulting schematic.

⇨ 15.(Canonical Forms) Consider the function f(A,B,C,D) = Σ m(0,1,2,7,8,9,10,15).

⇨ **Vedere: 2.2.2 Conversion Between Canonical Forms** R.Katz – Contemporary Logic Design

a) Write this as a Boolean expression in canonical min-term form.

b) Rewrite the expression in canonical maxterm form.

c) Write the complement of f in "little *m*" notation and as a canonical min-term expression.

d) Write the complement of f in "big *M*" notation and as a canonical maxterm expression.

Esercizi Proposti

- 18.(Boolean Simplification) Use Karnaugh maps (K-maps) to simplify the following functions in sum of products form. How many literals appear in your minimized solutions?
- **Vedere: 2.1.1 Boolean Operations Revisited, Pag 42; 2.2.2 Conversion Between Canonical Forms** R.Katz – Contemporary Logic Design

a) $f(X,Y,Z) = \Pi\ M(0,1,6,7)$

b) $f(W,X,Y,Z) = \Pi\ M(1,3,7,9,11,15)$

c) $f(V,W,X,Y,Z) = \Pi\ M(0,4,18,19,22,23,25,29)$

d) $f(A,B,C,D) = \Sigma\ m(0,2,4,6)$

e) $f(A,B,C,D) = \Sigma\ m(0,1,4,5,12,13)$

f) $f(A,B,C,D,E) = \Sigma\ m(0,4,18,19,22,23,25,29)$

g) $f(A,B,C,D,E,F) = \Sigma\ m(3,7,12,14,15,19,23,27,28,29,31,35,39,44,45,46,48,49,50,52,53, 55,56,57,59)$.

Esercizi Proposti

- 19. (Boolean Simplification) Determine the minimized realization of the following functions in the sum of products form:
- **Vedere: 2.2.4 Don't Cares and the Terminology of Canonical Forms, Pag 65; 2.3.3 Don't Care in K-maps, Pag. 74. 2.3.5 Example Application of the Step-by-Step Algorithm, Pag. 81.** R.Katz – Contemporary Logic Design
  - $f(W,X,Y,Z) = \Sigma m(0,2,8,9) + \Sigma d(1,3)$
  - $f(W,X,Y,Z) = \Sigma m(1,7,11,13) + \Sigma d(0,5,10,15)$
  - $f(V,W,X,Y,Z) = \Sigma m(2,8,9,10,13,15,16,18,19,23) + \Sigma d(3,11,17,22)$
  - $f(V,W,X,Y,Z) = \Sigma m(0,1,2,9,13,16,18,24,25) + \Sigma d(8,10,17,19)$
- 20. (Boolean Simplification) Use the K-map method to find the minimized product of sums expressions for the following Boolean functions:
  - $f(A,B,C) = A \oplus B \oplus C$
  - $f(A,B,C) = AB + BC + AC$
  - $f(A,B,C,D) = \sum m(1,3,5,7,9) + \sum d(6,12,13)$
  - $f(A,B,C,D) = \prod M(0,1,6,7)$
  - $f(A,B,C,D) = \sum m(0,2,4,6)$

Esercizi Proposti

- 21. (Positive and Negative Logic) Show the following:
- **Vedere: 2.2.3 Positive Versus Negative Logic, Pag. 61.** R.Katz – Contemporary Logic Design
  -)A positive logic AND is equivalent to a negative logic OR.
  -)A positive logic NOR is equivalent to a negative logic NAND.
  -)A positive logic XOR is equivalent to a negative logic XNOR.
  -)A positive logic XNOR is equivalent to a negative logic XOR.
- 26. (Combinational Logic Design) Consider a five-input Boolean function that is asserted whenever exactly two of its inputs are asserted.
- **Vedere: 2.3.6 K-Maps Revisited: Five – and Six – Variable Function, Pag. 83.** R.Katz – Contemporary Logic Design
  -)Construct its truth table.
  -)What is the function in sum of products form, using "little *m*" notation?
  -)What is the function in product of sums form, using "big *M*" notation?
  -)Use the Karnaugh map method to simplify the function in sum of products form.

Esercizi Proposti

- 28. (Combinational Logic Design) In this chapter, we've examined a 2-bit binary adder circuit. Now consider a 2-bit binary subtractor, defined as follows. The inputs *A*, *B* and *C*, *D* form the two 2-bit numbers *N*1 and *N*2. The circuit will form the difference *N*1 - *N*2 on the output bits *F* (most significant) and *G* (least significant). Assume that the circuit never sees an input combination in which *N*1 is less than *N*2. The output bits are don't cares in these cases.
  -)Fill in the four-variable truth table for *F* and *G*.
  -)Fill in the K-map for the minimum sum of products expression for the functions *F* and *G*.
  -)Repeat to find the minimum product of sums expression for *F* and *G*.
- 30. (Combinational Logic Design) Design a combinational circuit with three data inputs *D*2, *D*1, *D*0, two control inputs *C*1, *C*0, and two outputs *R*1, *R*0. *R*1 and *R*0 should be the remainder after dividing the binary number formed from *D*2, *D*1, *D*0 by the number formed by *C*1, *C*0. For example, if *D*2, *D*1, *D*0 = 111 and *C*1, *C*0 = 10, then *R*1, *R*0 = 01 (that is, the remainder of 7 divided by 2 is 1). Note that division by zero will never be requested.
  -)Fill in truth tables for the combinational logic functions R1 and R0.
  -)Derive minimized sum of product realizations of these functions using the Karnaugh map method.
  -)Draw a circuit schematic that implements R1 and R0 using NAND gates only. You may assume any fan-in gates that you need.