



# Progettazione di sistemi digitali

---

# CONVERSIONE DA BASE 10 A BASE b

dato N

- Dividere N per b
- Registriamo il resto
- Iterare il procedimento fino ad ottenere quoziente 0
- La sequenza di resti in ordine inverso da la rappresentazione in base b

ES.

$b_{10}$

$b_2$

62	111110
31	0
15	1
7	1
3	1
1	1
0	1

Per riconvertire da base b a base 10: metodo delle potenze associate

ES.

$b_3 : 2022$

$$2 \cdot 3^0 + 2 \cdot 3^1 + 0 \cdot 3^2 + 2 \cdot 3^3 = (26)_{10}$$

# OPERAZIONI IN BINARIO

## ADDITIONI

$$\begin{array}{r} 101 \\ + 001 \\ \hline 110 \end{array} \quad \begin{array}{l} 9 \\ 3 \\ + \text{riporto } (10)_2 \end{array} \quad \begin{array}{l} \text{classiche addizioni (come in 10)} \\ + \text{riporto } (10)_2 \end{array}$$

$$\begin{array}{r} 1010 \\ + 0111 \\ \hline 10001 \end{array} \quad \begin{array}{l} 10 \quad \text{con 4 bit: [0,15]} \\ 7 \\ 17 \end{array}$$

bit di OVERFLOW (superato l'intervallo di numeri rappresentabile con quel tot di bit disponibili)

## SOTTRAZIONI

da 2 rimane 1

$$\begin{array}{r} 1 \\ 01001 \\ - 0011 \\ \hline 0110 \end{array}$$

classiche sottrazioni + prestito (ricordando che  $10 = 2^3$ )

## MOLTIPLICAZIONI

$$\begin{array}{r} 1011 \\ \times 0011 \\ \hline 1011 \\ + 1011 \\ \hline 00100001 \end{array}$$

doppio bit

le moltiplicazioni danno un risultato con il doppio numero di bit rispetto ai fattori

$$\begin{array}{r} 1111 \\ \times 111 \\ \hline 1111 \\ 1111 \\ - \\ 1111 \\ \hline 01 \end{array}$$
$$(100)_2 = (4)_{10}$$

↳ in questo caso il riporto slitta di uno (100)<sub>pos.</sub>

aggiungo lo zero nell'uno  
+ +

## RAPPRESENTAZIONE DEGLI INTERI

Per rappresentare i numeri negativi vi sono tre diversi sistemi:

1. Rappresentazione in modulo e segno
2. Rappresentazione in complemento a 1
3. Rappresentazione in complemento a 2 (complemento alla base)

### MODULO e SEGNO

il bit più significativo rappresenta il segno 0 = positivo



INTERVALLO:  $[-2^{n-1} + 1, 2^{n-1} - 1]$  (un bit viene utilizzato per il segno, ne servono di più)

### COMPLEMENTO A 1 CA1

si parte dalla rappresentazione del numero positivo in binario

→ si complementa (invertire 1 con 0, in modo che la somma sia un n con tutti uno) per ottenere l'equivalente negativo

$$\text{ES. } 010 = 2$$

$$101 = -2$$

$$\text{INTERVALLO: } [-2^{n-1} + 1, 2^{n-1} - 1]$$

## COMPLEMENTO A 2

→ il MSB ha valore negativo

ov questo vuol sommato il modulo del resto della sequenza

$$-a_{n-1}b^{n-1} + \sum_{i=0}^{n-2} a_i b^i$$

ES.  $1011 = -8 + 2 + 1 = -5$   
 ↑  
 MSB: -8

INTERVALLO:  $[-2^{n-1}, 2^{n-1}-1]$

tabelle con 3 bit

	MS	CA1	CA2
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

Tra le possibili modalità di Rappresentazione la più valutata è il Ca2:

- In MS e CA1 vi è una ripetizione dello zero (più sequenze codificano lo zero), inoltre le operazioni non sono facili da realizzare in quanto bisogna tenere conto anche della magnitudo (la grandezza del modulo)
- Il Ca2 permette di codificare una cifra in più e una maggiore facilità di esecuzione
- Il CA2 è asimmetrica  $[-2^{n-1}, 2^{n-1}-1]$

## COMPLEMENTO A 2, OPERAZIONI

per rappresentare una cifra secondo il metodo a complemento CA2 bisogna:

1. scriverla in sistema binario
2. Calcolare l'ampiezza dell'intervallo (un bit in + dello standard)
3. se negativa: complementare il numero e aggiungere uno.
4. se positiva: aggiungere tanti zeri iniziali quanti ne richiede l'intervallo preso in considerazione.

ES.  $(20)_{10}$  :  $[-64, 63]$  6 bit       $(-12)_{10}$  :  $[-16, 15]$  5 bit

$$\begin{array}{r} 20 \\ 10 \\ 5 \\ 2 \\ 1 \\ 0 \\ 0 \end{array} \quad \begin{array}{l} (10100)_2 \\ (010100)_{CA2} \end{array}$$

$$\begin{array}{r} 12 \\ 6 \\ 3 \\ 1 \\ 0 \\ 1 \\ 0 \end{array} \quad \begin{array}{l} (1100)_2 = 12 \\ 10011 + \\ \hline (10100)_{CA2} \end{array}$$

## CALCOLARE L'OPPOSTO

1. complemento tutti i bit

2. aggiungo 1

1011 : negativo x uno iniziale  
(sottraendo ad una potenza di 2 tutte le potenze precedenti otengo 1 -> ovvero e' sempre + grande della somma delle potenze che l'au precedono)

METODO "ad occhio"

complemento tutti i bit fino all'1 meno significativo

escluso: 10110 → D1010  
↑LS

ES. 1011

$$\begin{array}{r} 1. \ 0100 \\ 2. \ 0100 \\ \hline & 1 \\ & \hline 0101 \end{array}$$

## ADDIZIONI

4 bit [-8, 7]

### addendi positivi

$$\begin{array}{r} 0^1 010 \\ + 0011 \\ \hline 0101 \end{array} \quad \begin{array}{r} 1^1 0^1 11 \\ + 0101 \\ \hline (1)000 \end{array} \quad \begin{array}{r} \rightarrow 0110 \\ + 0111 \\ \hline (1)101 \end{array} \quad \begin{array}{l} \text{la loro somma darebbe} \\ \text{valore fuori intervallo} \end{array}$$

errore: sommate 2 valori positivo ottenuto uno negativo

### addendi discordi

$$\begin{array}{r} 0110 \quad 6 \\ + 1011 \quad -5 \\ \hline (1)0001 \quad 1 \end{array} \quad \begin{array}{r} 0010 \quad 2 \\ + 1101 \quad -3 \\ \hline 1111 \quad -1 \end{array}$$

carry: non e' errore xche' il risultato e' corretto

### addendi negativi

$$\begin{array}{r} 1^1 011 \quad -5 \\ + 1110 \quad -2 \\ \hline \text{carry} \ 1001 \quad -7 \end{array} \quad \begin{array}{r} \rightarrow 1^1 011 \\ + 1010 \\ \hline 0101 \end{array}$$

errore: 2 numeri negativi danno un numero positivo

in CA2 e' facile comprendere gli errori (a differenza degli altri metodi)

## SOTTRAZIONI

→ prima calcolo l'opposto

→ poi somma

ES.  $A = 4 \quad B = 7 \quad A - B = ?$

4 bit  
[-8, 7]

$$4 = (0100)_2$$

$$7 = (0111)_2 \rightarrow \text{opposto: } 1001$$

$$\rightarrow \text{sommo: } 0100$$

$$\begin{array}{r} + 1001 \\ \hline 1101 \end{array}$$

-3

## ESTENDERE UN NUMERO

× estendere un numero di un tot di bit bisogna:

→ POSITIVO: aggiungere 0  $0100 \rightarrow 0000100$

→ NEGATIVO: aggiungere 1  $1001 \rightarrow 1111001$  grazie ad una proprietà potenze del 2 :

$$\begin{array}{ccc} 100 & = & 1100 \\ \uparrow & & \downarrow \\ -2^3 & & -2^4 \end{array} \quad -2^3 = -2^4 + 2^3$$

## RAPPRESENTAZIONE DEI REALI

Ovvero numeri molto grandi o molto piccoli, con la virgola

## CONVERTIRE n. CON LA VIRGOLA

- **PARTE INTERA:** la parte intera viene convertita secondo il classico procedimento di conversione binaria (divisioni successive per la base)
- **PARTE RAZIONALE** (dopo la virgola): la parte dopo la virgola segue sempre un sistema posizionale ma in questo caso le potenze di 2 sono negative
  1. Si moltiplica la parte razionale per due ( $0.xyz \times 2$ )
  2. Si itera la moltiplicazione sulla parte razionale del numero ottenuto fino a ottenere 0 o fino al n di cifre desiderato
  3. La rappresentazione finale della parte decimale è data dalle parti

interi del numero ottenute nel procedimento nell'ordine in cui sono state ottenute (dall'alto verso il basso).

es. 13,75

$$\rightarrow 13 : \begin{array}{c|c} 13 & 1101 \\ 6 \mid 1 & \\ 3 \mid 0 & \\ 1 \mid 1 & \\ 0 \mid 1 & \end{array}$$

$$13,75 = 1101,11$$

$$1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

$$\rightarrow 0.75 \quad 0.75 \cdot 2 = 1,5 \downarrow \quad 11$$

$$0.5 \cdot 2 = 1,0 \quad \text{c'e' zero mi fermo}$$

es. 13,45

$$13 = 1101$$

$$0.45 \cdot 2 = 0.9$$

$$0.9 \cdot 2 = 1.8$$

$$0.8 \cdot 2 = 1.6$$

$$0.6 \cdot 2 = 1.2$$

$$0.2 \cdot 2 = 0.4$$

$$0.4 \cdot 2 = 0.8$$

$$13,45 = 1101,011100 \text{ approssimazione}$$

↳ numero che va all'infinito ma bit finiti)

non otterro' mai zero

ma si ripetera' questa sequenza

mi fermo al

-, numero di cifre desiderate

2 modi x rappresentare i numeri reali = FIXED POINT o FLOATING POINT

## RAPPRESENTAZIONE A VIRGOLA FISSA (fixed point)

Sono l'equivalente del numero razionale decimale: viene utilizzato un totale di bit prefissati per la parte intera ed il restante per la parte razionale.

$n = \text{tot bit} \quad k \text{ bit} \times \text{la parte intera} \quad n-k \times \text{la parte decimali}$

Metodo rigido (non è adattabile alle diverse situazioni, come studi astronomici che hanno numeri molto grandi o studi microscopici che hanno numeri molto piccoli)

il numero si ottiene con il procedimento visto prima

## RAPPRESENTAZIONE A VIRGOLA MOBILE (floating point)

La rappresentazione a virgola mobile è l'equivalente della **notazione scientifica** ( $4,54 \times 10^5$ ), ovvero riportando un solo numero diverso da 0 prima della virgola e poi moltiplicare per una potenza di 10.

Il floating point si basa su una **tripla** di valori:

- s = Segno 1 BIT: (0 = +, 1 = -)
- e = Esponente in complemento a 2?
- m = Mantissa cifre dopo la virgola

in b=2  
sarà potenza  
di 2

NB: la cifra prima della virgola viene omessa, in quanto in notazione scientifica binaria corrisponde sempre ad 1 (nel sistema binario l'unica cifra diversa da 0). In questo modo si guadagna un bit per la mantissa così da migliorare la precisione.

es.  $13,45 \approx 1101,011100$

in notazione scientifica:  $1\text{ }101011100 \times 2^3$   
omessa

esponente del 2 (3) in complemento a 2 :  $e = 0011$

esponente  
↓  
 $13,45 = \begin{matrix} 0 & 0011 & 101011100 \\ \text{segno} & \text{mantissa} \end{matrix}$

Il valore del numero rappresentato è calcolabile come:  $(-1)^s \cdot 2^e \cdot 1.m$

Lo standard adottato x i floating point è IEEE 754

diversi tipi di precisione in base al numero di bit totali disponibili.

### IEEE 754

- 16 BIT	half precision	s=1	e=5	m=10	Bias = 15
- 32 BIT	single precision	//	e=8	m=23	Bias = 127
- 64 BIT	double precision	//	e=11	m=52	Bias = 1023

calcolare il bias :  $B = 2^{n-1} - 1$  (n è il numero di bit)

### Intervallo esponente

Per calcolare l'intervallo dell'esponente:  $[-2^{e-1}, 2^{e-1} - 1]$

1. si eliminano i due valori più piccoli  $(-2^{e-1} \text{ e } -2^{e-1} + 1)$  poiché vengono riservati per i numeri speciali dopo descritti.
2. si considera quindi:  $[-2^{e-1} + 2, 2^{e-1} - 1]$
3. si somma il bias, cioè  $2^{e-1} - 1$
4. e otteniamo  $[1, 2^e - 2]$  in binario  $[0...01, 1...10]$

PS. il bias è una quantità correttiva che serve per aggiustare un valore.

In questo caso il suo scopo è quello di portare tutti i valori che l'esponente può assumere (che include sia valori positivi per indicare numeri molto grandi che valori negativi per rappresentare numeri molto piccoli) a numeri positivi. In questo modo si eliminano la necessità di un bit per il segno e il problema del confronto tra numeri che altrimenti si avrebbe.

es. single precision (32 bit) → n bit e: 8

$[-128, 127] \rightarrow$  escludiamo i valori + piccoli =  $[-126, 127]$

→ aggiungiamo il bias  $[-126 + 127, 127 + 127] = [1, 254]$

I valori assunti dalla mantissa e dall'esponente dividono i numeri in divers categorie:

TIPO	ESPOENTE	MANTISSA
n. normali	$[1,2^e - 2]$	qualsiasi
zeri	00000	0
infiniti	$2^e - 1 - 1 \dots - 1$	0
n. denormali	00000	$m \neq 0$
NaN (not a number)	$2^e - 1 - 1 \dots - 1$ (s.p.=255)	

### convertire da decimale a float IEEE 754

- ① convertire in binario <sup>PARTE INTRA</sup> <sub>PARTE RAZIONALE</sub> (fixed point)
- ② normalizzare (notazione scientifica)
- ③ aggiungere il bias all'esponente  $E = e + \text{bias}$  e convertirlo
- ④ comporre il numero secondo IEEE 754 ( $s + E + m$ )

es. 13,45

$$\begin{array}{r} 13 \\ | \\ 6 \quad 0.45 \cdot 2 = 0.9 \\ | \quad 0.9 \cdot 2 = 1.8 \\ 3 \quad 0.8 \cdot 2 = 1.6 \\ | \quad 0.6 \cdot 2 = 1.2 \\ 0 \quad 0.2 \cdot 2 = 0.4 \\ \quad 0.4 \cdot 2 = 0.8 \end{array}$$

$$\text{binario} = 1101,011100$$

$$\text{normalizzato} = \underbrace{1.101011100}_{\text{`ometto}} \cdot 2^3$$

$$s \quad e \quad m$$
  
 $0 \quad 10010 \quad 101011100$

$$E = 3 + 15 = 18 = 10010$$

es.  $A = 26,42$

bit  
 $\langle 1,5,10 \rangle$

26	$0,42 \cdot 2 = 0,84$
13	$0,84 \cdot 2 = 1,68$
6	$0,68 \cdot 2 = 1,36$
3	$0,36 \cdot 2 = 0,72$
1	$0,72 \cdot 2 = 1,44$
0	$1,44 \cdot 2 = 0,88$

$$e = 4 + 15 = 19 = 10011$$

mi fermo quando raggiungo i tot di bit richiesti

$$11010,011010 \rightarrow 1,1010011010 \cdot 2^4$$

convertire da float IEEE 754 a decimale

- ① scrivere in binario e raggruppare i valori (s, e, m)
- ② determinare il segno
- ③ convertire l'esponente e sottrarre bias  $e = E - \text{bias}$
- ④ riscrive (o) m in binario fixed point (ricordare di aggiungere z)
- ⑤ calcolare valore decimale con potenze associate

es. CE94

  
1100 1110 1001 0100  
s e m  
negativo

$$E = 19 - 15 = 4$$

aggiungo 1

$$1,1010010100 \cdot 2^4$$

$$11010,010100 = 16 + 8 + 2, \frac{1}{4} + \frac{1}{16} = 26,3125$$

## OPERAZIONI CON FLOATING POINT

### ADDIZIONI E SOTTRAZIONI

1. confrontare gli esponenti e portarli allo stesso valore (solitamente il più piccolo al più grande)
2. spostare a sinistra la mantissa del numero a cui è stato modificato l'esponente per mantenerne il valore numerico
3. eseguire somma o sottrazione in base al segno degli addendi (se concordi somma, se discordi sottrazione) e, nel caso della sottrazione, determinare in base alla magnitudo delle mantisse chi è il minuendo e chi il sottraendo. Determinare il segno del risultato coerentemente alle osservazioni fatte.

$A = <0, 10011, 1010011010>$

$B = <1, 10100, 0010110101>$

confronto esponenti  $e_A = 19 - 15 = 4$   $e_B = 20 - 15 = 5$

porto  $e_A$  su  $e_B$  e modifico  $m_A$   $1,1010011010 \cdot 2^4 \rightarrow 0,11010011010 \cdot 2^5$

confronto segni e mantisse:

• i segni di A e B sono discordi: sottrazione

•  $m_B > m_A$  quindi B è minuendo e A è il sottraendo

• il segno del risultato è  $\ominus$  (perché B è negativo e  $B > A$ )

$$\begin{array}{r}
 1,00101010101 \\
 - 0,1101001101 \\
 \hline
 0,0101101000 \cdot 2^5
 \end{array}$$

normalizzare  $1,0110100000 \cdot 2^3$

riportare allo standard  $e = 3 + 15 = 18 = 10010$

$R = <1, 10010, 0110100000>$

il risultato ottenuto sarà un'approssimazione di quello che otterrei in base 10 perché nei vari passaggi potrei perdere un bit o non riuscire a rappresentarlo

## Moltiplicazione e divisione

1. sommare o sottrarre gli esponenti

2. moltiplicare o dividere le mantisse, normalizzando se necessario

3. determinare il segno (negativo se fattori discordi, positivo se concordi)

PS: quando vado a sommare gli esponenti devo sottrarre il bias, altrimenti otterrei un risultato con doppio bias (provenienti dai due esponenti) e quindi non rappresentabile.

es.  $A = <0, 10001, 0101000000>$  moltiplicazione

$B = <1, 10000, 0110000000>$

1,0101

determino il segno: negativo

$e_A = 17 - 15 = 2$   $e_B = 16 - 15 = 1$   $e_R = 3 + 15 = 18$

$R = <1, 10010, 1100111000>$

$\times 1,011$

10101

10101-

00000--

10101---

1,1100111 · 2<sup>3</sup>

NB.: l'underflow nelle operazioni con float si ha quando l'esponente non si può rappresentare (è o troppo grande o troppo piccolo)

## Altri codici numerici e alfanumerici

**Codice BCD** (*binary coded decimal*): è utilizzato per rappresentare singolarmente ogni cifra decimale. Le sequenze di cifre decimali non hanno valore numerico ma solo "grafico". Il più utilizzato è il BCD 8421, che associa ad ogni cifra decimale 4 bit. La rappresentazione è, però, non efficiente poiché vengono utilizzate solo 10 combinazioni delle 16 totali possibili.

1000  
3827 - 0111  
0011 '0010

**Codice ASCII** (*American standard code for information interchange*): è il codice alfanumerico standard. Utilizza 7 bit per la codifica di 128 caratteri. I tre bit più significativi (prefisso) indicano la tipologia di carattere (numero, lettera maiuscola, lettera minuscola, carattere di controllo).

Siccome la maggior parte dei calcolatori usa il byte (8 bit) il codice ASCII viene esteso di un bit:

- 0
- 1 (per produrre simboli addizionali come caratteri accentati o alfabeto greco)
- bit di parità

Il bit di parità è un bit di controllo utilizzato per individuare degli errori nella trasmissione ed elaborazione di dati.

Il bit viene aggiunto allo scopo di rendere il numero totale di 1 pari (parità pari) o dispari (parità dispari).

0111000  
0 dispari  
1 pari

Ad esempio, assumendo che venga utilizzata la parità pari, un messaggio viene inviato. Se la parità del carattere ricevuto non è pari, allora si è verificato almeno un errore).

# Circuiti combinatori

## PORTE LOGICHE

Un circuito logico è una rete di dispositivi elettronici che processa un valore discreto. L'unità di base dei sistemi digitali sono le porte logiche (un circuito combinatorio è composto da porte logiche interconnesse).

Una porta logica riceve in input uno o più valori binari e restituisce un output a sua volta binario. Trattandosi di circuiti digitali, quindi elettronici, il valore assunto dagli input e output binari è descritto dai valori di tensione della corrente (alta tensione:1, bassa tensione:0). Il segnale assume, perciò, valori continui che devono essere riassunti e approssimati a valori discreti. Per farlo i valori del segnale vengono misurati in tempi opportuni per distinguere con chiarezza quale valore tra 0 e 1 il segnale sta assumendo.



## LOGICA BINARIA

La logica binaria tratta variabili che possono assumere solo due valori e operazioni che ad esse possono essere applicate.

Le tre operazioni logiche fondamentali sono: OR, AND, NOT. Qui sotto ne vengono riportate i simboli, le porte logiche e le loro tabelle di verità.

NOT :  $\bar{x}$

x	y
0	1
1	0

Porta logica:



moltiplicazione logica  
AND :  $x \cdot y, xy, x \wedge y$

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Porta logica:  

A logic gate symbol with two input lines and one output line, showing a circle at the first input line.

somma logica  
OR :  $x + y, x \vee y$

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Porta logica:  

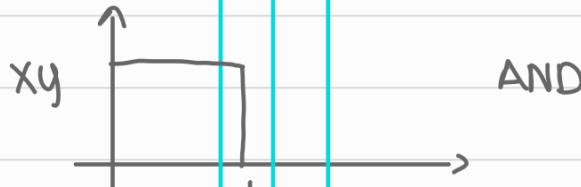
A logic gate symbol with two input lines and one output line, showing a circle at the second input line.



→ Forme d'onda

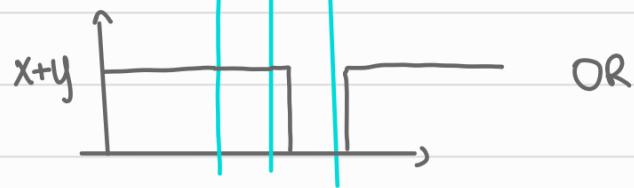


fissare i fronti d'onda  
(cambi da 0 a 1)



AND

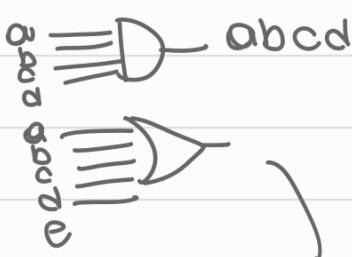
L. piccolo delay (ritardo) che serve per stabilizzare il valore dovuto all'entrata nella porta



OR

le porte logiche AND e OR possono avere anche più input di ingresso :

FAN IN



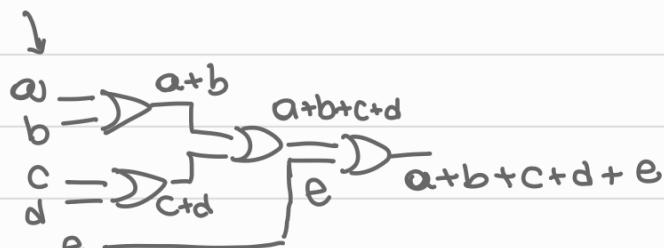
Schematizzazione

$$\begin{aligned} a &= D_{ab} \\ b &= D_{bc} \\ c &= D_{cd} \\ d &= D_{de} \end{aligned}$$

un livello

delay aggiuntivo

RIFARE



3 livelli = 3 ritardi

# ALGEBRA BOOLEANA

L'algebra booleana fornisce il supporto matematico per progettare i circuiti combinatori in maniera efficiente. Le relazioni tra gli input e output di variabili che attraversano porte logiche vengono descritte dalle equazioni (espressioni) booleane (derivanti dal matematico Boole).

Nell'algebra booleana viene utilizzato:

- un insieme di supporto o alfabeto di supporto (0,1)
- operatori OR, AND, NOT
- assiomi e proprietà che stabiliscono le regole da utilizzare per manipolare le espressioni.

## ASSIOMI E PROPRIETÀ

◦ ASSOCIAtività	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$
◦ COMMUTATIVITÀ	$x + y = y + x$	$xy = yx$
◦ DISTRIBUTIVITÀ	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
◦ COMPLEMENTO	$x + \bar{x} = 1$	$x\bar{x} = 0$
◦ ELEMENTO NEUTRO	$x + 0 = x$	$x \cdot 1 = x$
→ PROPIETÀ		
◦ EL. NULLIFICATORE	$x + 1 = 1$	$x \cdot 0 = 0$
◦ INVOLUZIONE (complemento)	$\bar{\bar{x}} = x$	
◦ IDEMPOTENZA	$x + x = x$	$x \cdot x = x$
◦ ASSORBIMENTO	$x + xy = x$	$x(x + y) = x$
◦ LEGGE DI DEMORGAN	$\overline{x+y} = \bar{x}\bar{y}$	$\overline{xy} = \bar{x} + \bar{y}$
◦ ASSORBIMENTO 2	$x\bar{y} + y = x + y$	$x(\bar{x} + y) = xy$

La caratteristica principale degli assiomi è la loro dualità, ossia la totale intercambiabilità tra AND e OR e tra 1 e 0 (senza cambiare il significato originale della funzione)

Una **funzione booleana** è descritta da:

- una variabile che denota la funzione
- al secondo membro: espressioni al secondo membro (variabili legate da operatori)

$$f = \bar{x}y + z$$

L'algebra booleana consente di **manipolare** una funzione booleana in modo da

ottenere forme equivalenti semplificate e conseguentemente minimizzare il numero di porte di un circuito.

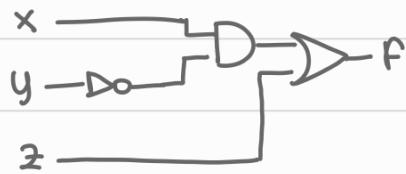
es. applicando alla f precedente la proprietà distributiva otteniamo

$$f = (x+z)(\bar{y}+z)$$

## Implementare una funzione con porte logiche

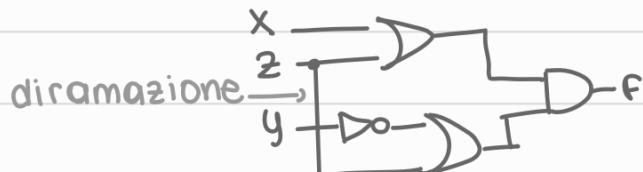
→ determinare il circuito

$$f = x\bar{y} + z$$



rappresentiamo f anche nell'altra forma:

$$f = (x+z)(\bar{y}+z)$$



## Realizzare una funzione con le tavole di verità

La tavola di verità è una tabella che riporta a sinistra tutte le possibili combinazioni di valori assunti dalle variabili e a destra i valori assunti dalla funzione.

$$f = x\bar{y} + z$$

$$f_1 = (x+z) \cdot (\bar{y}+z)$$

x	y	z	f	f <sub>1</sub>
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

① scrivere i valori binari delle variabili in modo ordinato

② comprendere se l'operatore esterno è OR o AND

→ OR = 1 quando almeno una variabile è 1  
quindi rintraccio tutte le combinazioni in cui z è 1 e poi quelle in cui  $x\bar{y}$  è 1.  
le restanti combinazioni varanno 0 per esclusione.

Tavole uguali e equivalenti



## METODO DELL'INDUZIONE PERFETTA

Utilizzato nella verifica dell'identità di due funzioni. Consiste nella costruzione della tavola di verità delle due funzioni di cui vogliamo verificare l'identità e confrontarle.

Questo metodo è applicabile solo alle funzioni con poche variabili.

## DUALITA:

Un'espressione E1 è duale di un'espressione E2 se differisce per gli operatori (AND e OR sono scambiati) e per i valori costanti (1 e 0 sono scambiati).

Secondo il principio di dualità un'equazione booleana rimane valida se si considera il duale di entrambi i lati della dell'uguaglianza.

### es. + verifica

$$x(y+z) = xy + xz \text{ duale} \quad \text{per ottenere la duale inverti operazioni}$$

$$x+yz = (x+y) \cdot (x+z) \rightarrow \text{verifichiamo la veridicità}$$

$$x+yz = xx + xz + xy + yz \quad xx=x \text{ idempotenza}$$

$$\text{"/} = x+xz + xy + yz \quad \text{assorbimento}$$

$$\text{"/} = x+xy+yz \quad \text{assorbimento}$$

$$x+yz = x+yz$$

$$x+xy = x(\overbrace{1+y}^{\text{nullificatore}}) = x \cdot 1 - x \quad \text{dim assorbimento}$$

L'espressione duale è equivalente a quella di partenza (hanno gli stessi valori nella tavola di verità). Perciò la funzione duale può risultare molto utile nelle verifiche di identità per passare da un'espressione più complessa alla sua duale più semplice.

$$f = x\bar{y} + z \rightarrow \tilde{f} = (x+\bar{y}) \cdot z$$

r → tilde

$\tilde{f}$  funzione duale

x	y	z	f	$\tilde{f}$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

## Complementare una funzione

Per ricavare algebricamente la funzione complementare si può:

- Usare le leggi di De Morgan

$$\overline{x+y} = \bar{x} \cdot \bar{y} \quad \overline{xy} = \bar{x} + \bar{y}$$

### es.

$$\bar{f} = \overline{xy + z} = \overline{xy} \cdot \bar{z} = (\bar{x} + \bar{y}) \bar{z}$$

- 1. Passare alla duale
- 2. Complementare le singole variabili (letterali)

es.

$$\tilde{f} = (x + \bar{y}) \cdot z \quad \text{complementare letterali}$$

$$\bar{f} = (\bar{x} + y) \cdot \bar{z}$$

## Semplificare una funzione

Un circuito è più efficiente quando ha meno porte o meno ingressi nel fan in di ogni porta (a parità di porte).

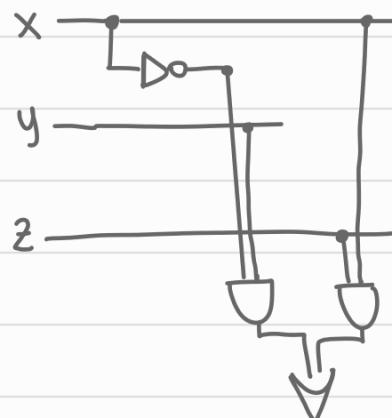
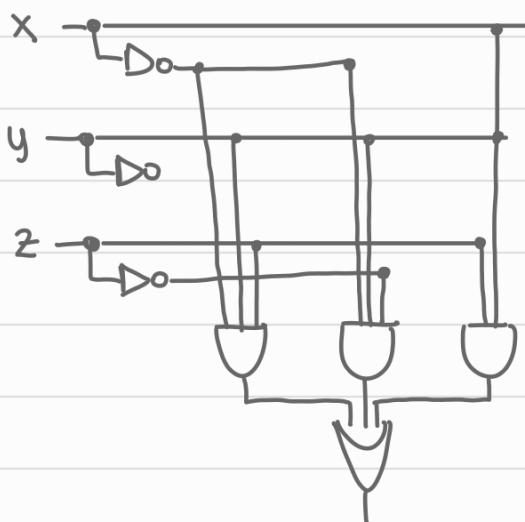
Per semplificare una funzione si devono applicare gli assiomi.

es.

$$f = \bar{x}yz + \bar{x}y\bar{z} + xy = \bar{x}y(z + \bar{z}) + xy = \bar{x}y + xy$$

Raggruppiamo

↳ è sempre 1 → el. nullo



## TEOREMA DEL CONSENSO

$$\bar{x}z + xy + yz = \bar{x}z + xy$$

↳ termine è ridondante e può essere eliminato

DIMOSTRAZIONE

$$\bar{x}z + xy + yz = \bar{x}z + xy + (x + \bar{x})yz = \quad \text{sempre } = 1 \text{ e } 1 \text{ nella moltiplicazione è nullo } (X \cdot 1 = X)$$

$$= \bar{x}z + xy + xy\bar{z} + \bar{x}yz = \quad \text{assorbimento}$$

$$= \bar{x}z + xy \quad \text{come volevansi dimostrare}$$

## Forma normale

Le forme normali in cui le funzioni booleane possono essere espresse sono :

- SOP *sum of products*:  $\bar{x}y + \bar{z}$  forma normale disgiuntiva (rete AND TO OR, le porte confluiscono in una OR finale)
- POS *product of sum*  $(x+y) \cdot \bar{z}$  forma normale congiuntiva (rete OR TO AND, le porte confluiscono tutte in una AND)

Tutte le espressioni booleane hanno entrambe queste forme.

La duale di una SOP è una POS e viceversa.

$$f = \bar{x}y + \bar{x}z \quad \text{SOP}$$

$$\tilde{f} = (\bar{x} + y)(\bar{x} + z) \quad \text{POS}$$

### procedimento per ottenere una normale

1. applicare leggi di DeMorgan fino a complementazione solo su singola variabile
2. applicare la proprietà distributiva  $x(y+z) = xy + xz \quad \text{SOP}$   
 $x + yz = (x+y)(x+z) \quad \text{POS}$
3. eliminare i termini ripetuti o ridondanti (attraverso l'idem potenza o l'assorbimento)

es.  $(\bar{a}+c) \cdot \bar{a}c + \bar{b} \cdot \bar{c}$

1. de morgan

FORMA SOP

$$(\bar{a}+c) \cdot \bar{a}c + b + \bar{c}$$

2. distributiva sop

$$0 = \underbrace{\bar{a}\bar{a}c}_{0 \cdot x = 0} + \underbrace{\bar{a}c \cdot c}_{= c \text{ idempotenza}} + b + \bar{c}$$

3. eliminazione

$$= \bar{a}c + b + \bar{c}$$

FORMA POS

$$\bar{a}c + b + \bar{c} = 1 + b = 1$$

$$(a+b+\bar{c})(\bar{c}+b+\bar{c})$$

$$a+b+\bar{c}+b = a+b+\bar{c}$$

$$\downarrow \text{idempotenza } b+b=b$$

tavola di verità: In una funzione POS per determinare la sua tavola di verità si studiano prima i valori per cui la funzione è 0 (ovvero quando almeno uno dei

termini è 0), tenendo conto che un termine somma è 0 solo se tutti letterali sono 0. Poi si completa con gli 1 per esclusione. Al contrario in una forma SOP si studia prima dove la funzione vale 1 (almeno un termine è 1) e poi per esclusione dove vale 0.

## Forme canoniche

Le forme canoniche sono tutte quelle espressioni in forma normale (pos o sop) nella quale appaiono solo maxtermini o mintermini.

- **mintermine:** termine prodotto in cui appaiono tutte le variabili della funzione in forma diretta o negata.
- **maxtermine:** termine somma nella quale sono presenti tutte le variabili in forma diretta o negata

Forma canonica SOP: appaiono solo mintermini

Forma canonica POS: appaiono solo maxtermini (tutti i termini sono maxtermini)

### tavola di verità

In una forma canonica ogni termine corrisponde ad un solo valore della tavola di verità (i mintermini, SOP, corrispondono agli 1 mentre i maxtermini, POS, corrispondono agli 0).

es. POS

$$f = (a + \bar{b} + c)(a + b + c)(\bar{a} + \bar{b} + c)$$

T <sub>1</sub>	a	b	c	f	la funzione vale 0
0	0	0	0	0	SOLO nelle combinazioni determinate
1	0	0	1	1	dai maxtermini
2	0	1	0	0	
3	0	1	1	1	OTTENUTE associando uno 0 alla variabile dritta
4	1	0	0	1	ed un 1 alla variabile negata
5	1	0	1	1	
6	1	1	0	0	
7	1	1	1	1	

espressione compatta della forma canonica :

$$M_0 \cdot M_2 \cdot M_6 = \text{AND}(M_0, M_2, M_6)$$

dove il numero decimale corrisponde alla posizione del maxtermine  
e quindi dello  $M_f$  nella T.V.

Viceversa dalla tavola di verità posso ricavare la forma canonica: le sequenze a cui corrisponde lo 0 della funzione rappresentano i maxtermini. Per scrivere ogni

maxtermine bisogna riportare al suo interno ogni variabile (in forma vera se le corrisponde lo 0 e in forma negata se le è associato l'1)

a	b	f	1. guardo gli 0
0	0	0	
0	1	0	2. COSTRUISCO i maxtermini
1	0	0	
1	1	1	$(a+b)(a+\bar{b})(\bar{a}+b)$

es. SOP

nella forma SOP ad ogni mintermine corrisponde un 1 nella tavola di verità.

Proviamo a ricavare l'espressione SOP della precedente funzione partendo dalla sua tavola di verità:

prendendo la tabella analizziamo dove f vale 1 per trovare i corrispettivi mintermini a cui sarà riportata una variabile dritta dove è associato l'1 e una negata se vi è lo 0

$$\bar{a}\bar{b}c + \bar{a}bc + a\bar{b}\bar{c} + \bar{a}\bar{b}\bar{c} + abc$$

forma compatta:  $m_1 + m_2 + m_4 + m_5 + m_7 \rightarrow OR(m_1, m_2, m_4, m_5, m_7)$

il numero decimale = numero binario  
corrispettivo ai valori che assumono le variabili

procedimento per ottenere una forma canonica

1. porto la funzione in forma normale
2. identifico il/i termine alla quale manca una variabile x e:

POS: termine somma sommo  $x\bar{x}$

SOP: termine prodotto moltiplico per  $(x+\bar{x})$

3. applico la proprietà distributiva

4. elimino i termini ripetuti

es. POS

$$(a+c)(\bar{b}+c)$$

$$(a+b\bar{b}+c)(\bar{b}+c+a\bar{a}) =$$

$$\hookrightarrow (a+b+c)(a+\bar{b}+c)(a+\bar{b}+c)(\bar{a}+\bar{b}+c)$$

proprietà distributiva

RIVEDERE

es. SOP

$$\bar{a} + \bar{b}c = \bar{a}(b + \bar{b})(c + \bar{c}) + \bar{b}c(a + \bar{a}) = \\ \bar{a}bc + \bar{a}b\bar{c} + \bar{a}\bar{b}c + \bar{a}\bar{b}\bar{c} + a\bar{b}c + \bar{a}\bar{b}c$$

## Circuito ed espressione minimale

Un circuito minimale presenta:

- il minor numero di porte
- a parità di numero di porte, il minor numero di ingressi per porta

L'espressione minimale che lo rappresenta algebricamente ha:

- il minor numero di termini (corrispettivo del numero di porte)
- a prità di numero di termini, il minor numero di letterali (corrispettivo dei fan in)

\* mappe di KARNOUGH

\* ANALISI E SINTESI

# OPERATORI UNIVERSALI: NAND E NOR

NAND



NOR

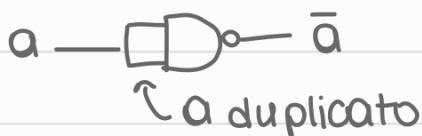


Gli operatori NAND e NOR sono chiamati operatori universali perché possono realizzare i tre operatori fondamentali NOT, OR, AND.

come NAND realizza gli operatori

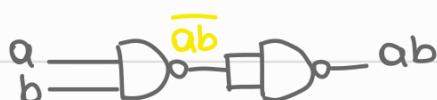
- NOT :

$$\bar{a} = \overline{aa}$$



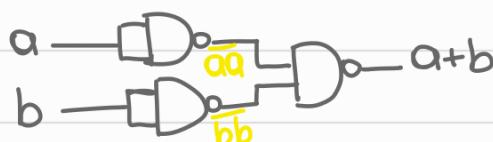
- AND :

$$ab = \overline{\overline{a}\overline{b}} = \overline{\overline{a}\overline{b}} \cdot \overline{\overline{a}\overline{b}}$$



- OR :

$$a+b = \overline{\overline{a} \cdot \overline{b}} = \overline{\overline{a} \cdot \overline{b}}$$



come NOR realizza i tre operatori

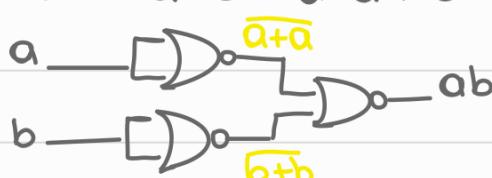
- NOT :

$$\bar{a} = \overline{a+a}$$



- AND :

$$ab = \overline{\overline{a}+\overline{b}} = \overline{\overline{a}+\overline{a}} + \overline{\overline{b}+b}$$



o OR:

$$a+b = \overline{\overline{a+b}} = \overline{\overline{a+b} + \overline{a+b}}$$



La peculiarità di questi operatori permette di realizzare circuiti utilizzando solo una tipologia di porta logica (NAND o NOR). Tuttavia, bisogna trovare l'espressione adatta in modo che il circuito non diventi troppo lungo e complesso.

per NAND  $\rightarrow$  SOP per NOR  $\rightarrow$  POS

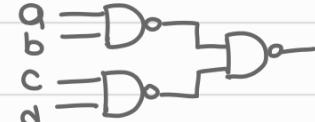
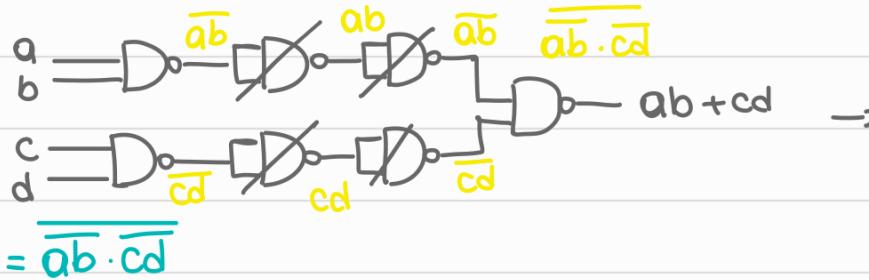
$\hookrightarrow$  si semplificano le porte

es.  $ab + cd$



tradizionale

$$ab + cd = \overline{\overline{ab} \cdot \overline{ab}} + \overline{\overline{cd} \cdot \overline{cd}} = \overline{\overline{ab} \cdot \overline{ab}} \cdot \overline{\overline{cd} \cdot \overline{cd}}$$



NAND con 3 variabili



XOR (o OR esclusivo)  $\oplus$

Restituisce 1 solo se una sola variabile vale 1

TAVOLA DI VERITA'

x y	$x \oplus y$
0 0	0
0 1	1
1 0	1
1 1	0

PORTA LOGICA



$$x \oplus y = \overline{x}y + x\overline{y}$$

proprietà'

$$x \oplus 0 = x, \overline{x \oplus y} = x \oplus \overline{x \oplus y}$$

XNOR

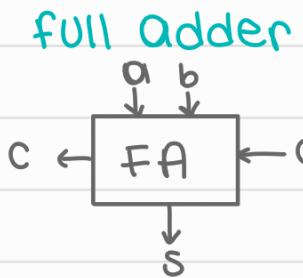
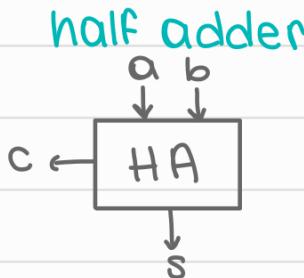
associatività:

$$(x \oplus y) \oplus z = x \oplus (y \oplus z)$$

# ADDIZIONATORE

esistono due tipologie di circuiti per realizzare un addizzionatore, ovvero un circuito aritmetico che esegue operazioni aritmetiche: Half adder e il Full Adder. **Half Adder** esegue la somma di due bit (gli addendi, senza curarsi del riporto). Il **Full adder**, invece, esegue la somma tra tre bit, due derivanti dagli addendi e uno è l'eventuale riporto della precedente colonna. Entrambi danno in uscita due bit: la somma e il riporto.

↳ sum      ↳ carry



## Half adder

in ingresso: 2 bit (uno per addendo)

in uscita: 2 bit (Sum e Carry)

### TAVOLA DI VERITÀ:

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

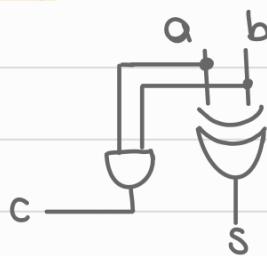
dalla tavola di verità è possibile notare che i valori del riporto C corrispondono ai valori assunti dall'AND tra gli ingressi (la funzione è uguale a 1 solo se entrambi gli ingressi sono 1):

$$C = ab$$

la somma S, invece, coincide con lo XOR (vale uno solo se i valori di ingresso sono diversi tra loro):

$$S = a \oplus b$$

### CIRCUITO HA:



## Full adder

n ingressi: 3 bit (2 da addendi e 1 dal riporto)

n uscite: 2 bit (sum e carry)

### TAVOLA DI VERITÀ:

a	b	c <sub>i</sub>	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

### mappe di Karnaugh

somma:			
b <sub>i</sub>	a	0	1
00	0	0	1
01	1	1	0
11	0	1	1
10	1	0	1

riporto:			
b <sub>i</sub>	a	0	1
00	0	0	0
01	0	1	1
11	1	1	1
10	0	1	1

$$C = ab + ac_i + bc_i$$

$$S = ab\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc =$$

$$c_i(\bar{a}\bar{b} + ab) + \bar{c}_i(a\bar{b} + \bar{a}b) = \\ c_i(\overline{a+b}) + \bar{c}_i(a+b) = \\ (a+b) \oplus c_i$$

Si può dimostrare sia dalla T.V. che dall'altra equazione dello XOR

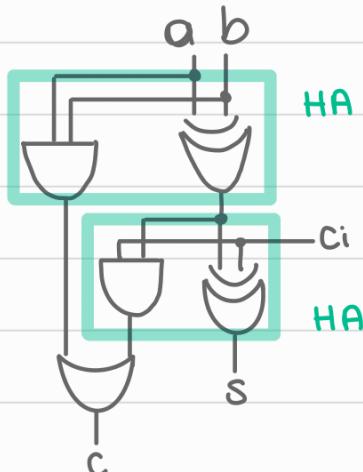
$$C = ab + ac_i + bc_i =$$

$$ab + c_i(a+b) =$$

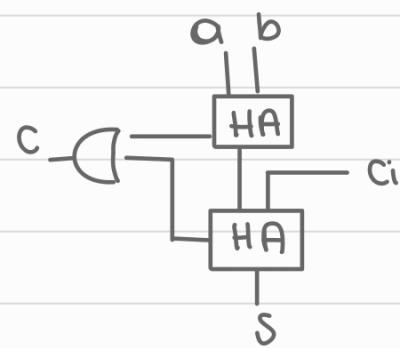
$$ab + c_i(a \oplus b)$$

la funzione da 1 se uno dei termini è 1. Nel secondo termine posso levare il caso in cui  $a=1 b=1$  danno  $C=1$  poiché è già stato preso in considerazione nell'AND del primo termine. L'OR tra  $a+b$  senza il caso  $1+1=1$  equivale allo XOR tra  $a$  e  $b$  ( $1+1=0$ )

### CIRCUITO FA:



il circuito FA può essere implementato utilizzando due HA



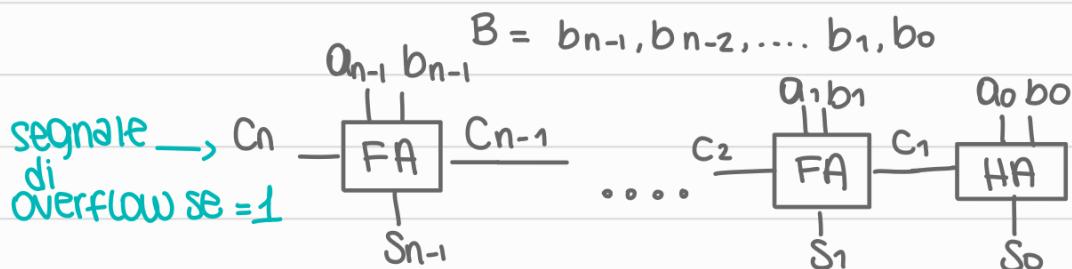
## Ripple Carry Adder

sommatore a propagazione di riporto

Il RCA è un circuito che esegue la somma tra due numeri binari di  $n$  cifre. È composto da un half adder e  $n-1$  full adder con i riporti connessi in cascata dal meno significativo al più significativo (l'uscita riporto dell'uno è connessa all'entrata riporto del successivo).

$$A = a_{n-1}, a_{n-2}, \dots, a_1, a_0$$

$$B = b_{n-1}, b_{n-2}, \dots, b_1, b_0$$



La composizione gerarchica dell'RCA consente la semplificazione della progettazione di un circuito sommatore di numeri a  $n$  bit. Ad esempio la somma di due numeri a 4 bit richiederebbe la semplificazione di una tabella di 256 righe (8 variabili in entrata).

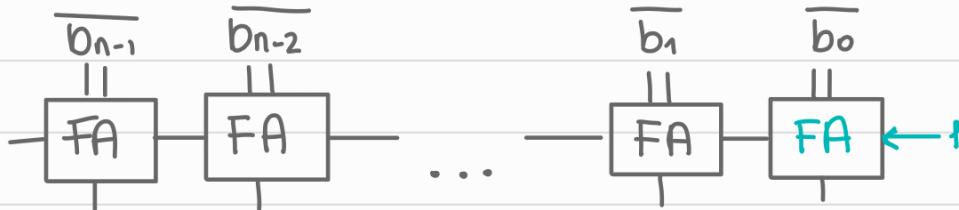
## Sommatori e sottrattori binari

Un sommatore binario può essere implementato per eseguire anche la sottrazione. Nel complemento a 2 la sottrazione tra due numeri corrisponde alla somma del primo con l'opposto dell'altro (il complemento dell'altro più uno).

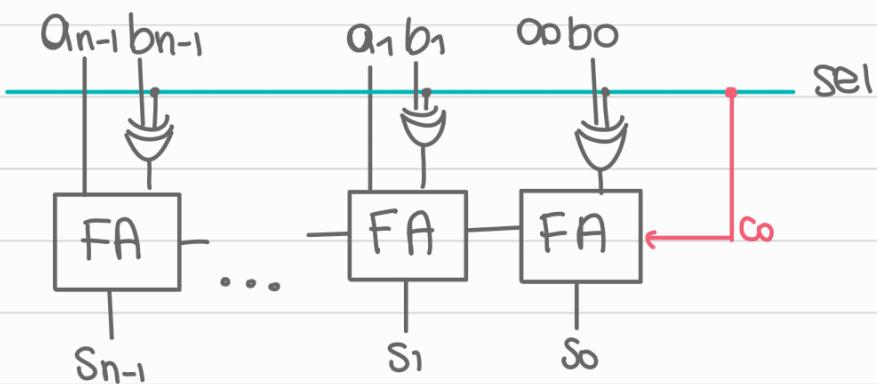
$$A - B = A + (-B) = A + (\bar{B} + 1)$$

Il ripple carry adder, tuttavia, da il risultato dopo  $n$  cicli, poichè il FA deve aspettare il riporto del FA precedente (quindi il tempo di esecuzione dipende dalla lunghezza degli operandi). Perciò eseguire un'ulteriore somma risulterebbe costoso a livello di tempo.

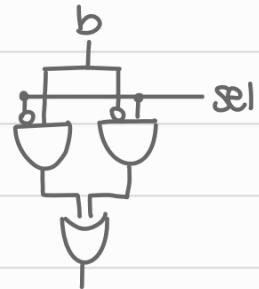
Per risolvere questa situazione l'Half adder viene sostituito da un Full adder con riporto di ingresso pari a 1.



Le due operazioni possono essere combinate in un unico circuito attraverso un sistema di selezione formate da porte XOR connesse agli ingressi del secondo addendo (b). Il segnale di selezione (sel) controlla l'operazione da eseguire: se sel=0 il circuito opera da sommatore, se sel=1 il circuito opera da sottrattore.



PORTE XOR



$$\overline{sel} \cdot b + \bar{b} \cdot sel = sel \oplus b$$

Il valore 1 di selezione fa sì che, attraversando le porte XOR B venga complementato (se  $b=1$  diventa 0 e se  $b=0$  diventa 1), sia che il riporto iniziale sia uno (aggiunge un uno).  
Co ↵

	b	sel	complemento
0	1	1	1
1	1	0	0
0	0	0	0
1	0	1	1

con 0 b rimane invariato

## PORTE AND con funzione Gating

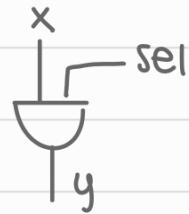
Le porte AND con funzione Gating selezionano il segnale da passare al circuito: quando il segnale è 1 il valore rimane invariato mentre quando è 0 viene azzerato indipendentemente dal suo valore iniziale. → il caso di prima utilizzava le porte XOR

T.V. AND

→	X	sel	Y
	0	1	0
	1	1	1
	0	0	0
	1	0	0

invariato  
azzerato

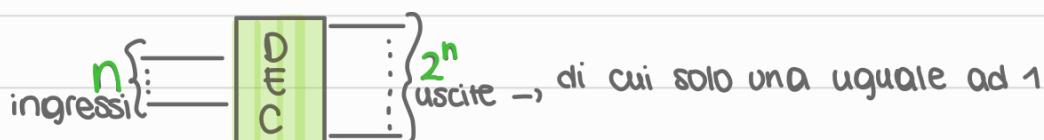
$$y \begin{cases} 0 & \text{se } sel=0 \\ x & \text{se } sel=1 \end{cases}$$



## DECODIFICATORE standard

Il decodificatore trasforma un codice ad  $n$  bit in un codice alla  $2^n$  bit. Viene chiamato decodificatore  $n-m$  dove  $n$  è il numero di ingressi mentre  $m$  è il numero di uscite.

Il decoder assegna un unico output ad ogni possibile combinazione di quei  $n$  bit (ogni codice binario di  $n$  bit può arrivare a rappresentare fino a  $2^n$  elementi distinti, a tutti questi possibili elementi il decodificatore assegna un output). Per ogni configurazione degli  $n$  ingressi una delle linee di uscita è attiva (pari a 1) mentre le altre sono disattivate (il decoder è solitamente usato per attivare un output specifico).



N.B. I decodificatori producono tutti i **mintermini** delle variabili in ingresso

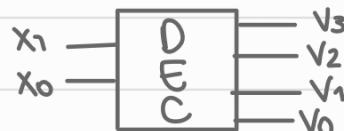
Per ciascuna combinazione **tutte le uscite sono nulle tranne quella il cui pedice corrisponde al numero binario in ingresso**, la quale viene posta uguale ad 1

Es. di un decodificatore 2-4:

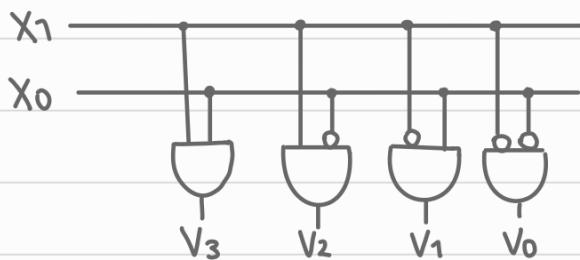
TAVOLA DI VERITÀ'

$X_1 X_0$	$V_3 \ V_2 \ V_1 \ V_0$	mintermini:
0 0	0 0 0 1	$V_3 = X_1 X_0$
0 1	0 0 1 0	$V_2 = X_1 \bar{X}_0$
1 0	0 1 0 0	$V_1 = \bar{X}_1 X_0$
1 1	1 0 0 0	$V_0 = \bar{X}_1 \bar{X}_0$

STRUTTURA:



CIRCUITO

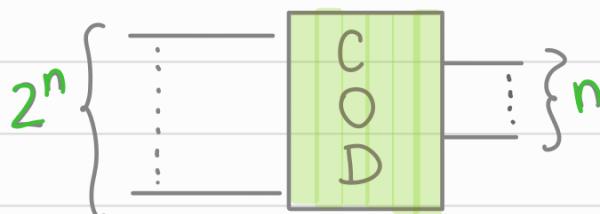


decoder 1-2

$$\begin{array}{ccc}
 A & \xrightarrow{\text{buffer}} & D_0 = \bar{A} \\
 & \xrightarrow{\text{buffer}} & D_1 = A
 \end{array}$$

## CODIFICATORE standard

Un codificatore esegue l'operazione inversa al decodificatore. È dotato di  $2^n$  ingressi e  $n$  uscite. Quando una linea di ingresso viene attivata, in uscita viene prodotto il codice binario corrispondente.

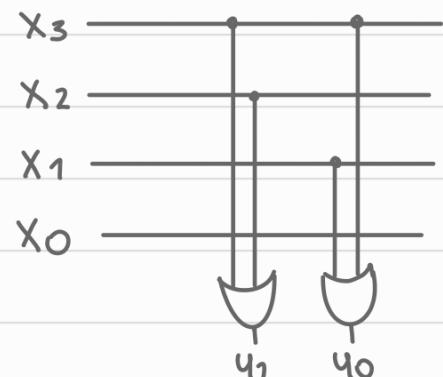


Solo un ingresso vale 1 mentre gli altri valgono 0. Nella tavola di verità vengono riportate solo le combinazioni nella quale una sola variabile è 1 (tutte le altre sarebbero dei don't care poiché non sono accettate come valori di ingresso)

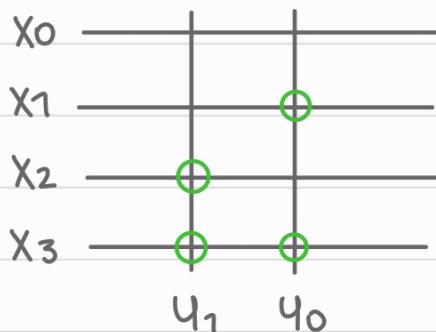
ES. codificatore 4-2

$X_3\ X_2\ X_1\ X_0$	$y_1\ y_0$	$y_1 = X_3 + X_2$	$y_0 = X_3 + X_1$
0 0 0 1	0 0		
0 0 1 0	0 1		
0 1 0 0	1 0		
1 0 0 0	1 1		

CIRCUITO

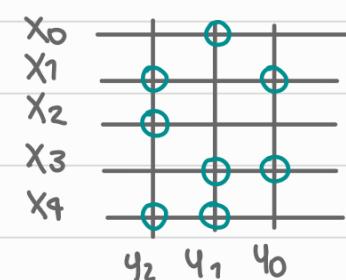


il codificatore puo' essere rappresentato e schematizzato attraverso una matrice di OR (simboleggiate da pallini vuoti)



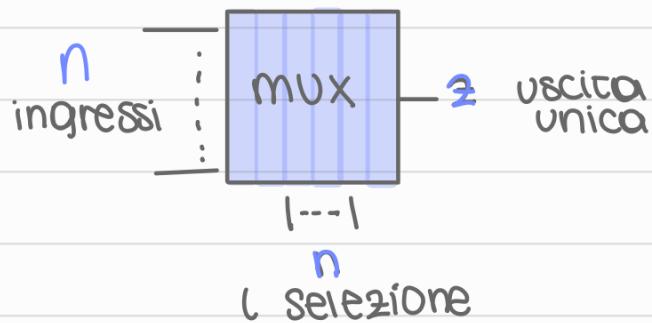
**codificatore non standard** = numeri binari delle uscite non corrispondono al pedice dell' entrata che vale 1

$X_4\ X_3\ X_2\ X_1\ X_0$	$y_2\ y_1\ y_0$	
0 0 0 0 1	0 1 0	
0 0 0 1 0	1 0 1	$y_2 = X_1 + X_2 + X_4$
0 0 1 0 0	1 0 0	$y_1 = X_0 + X_3 + X_4$
0 1 0 0 0	0 1 1	$y_0 = X_1 + X_3$
1 0 0 0 0	1 1 0	



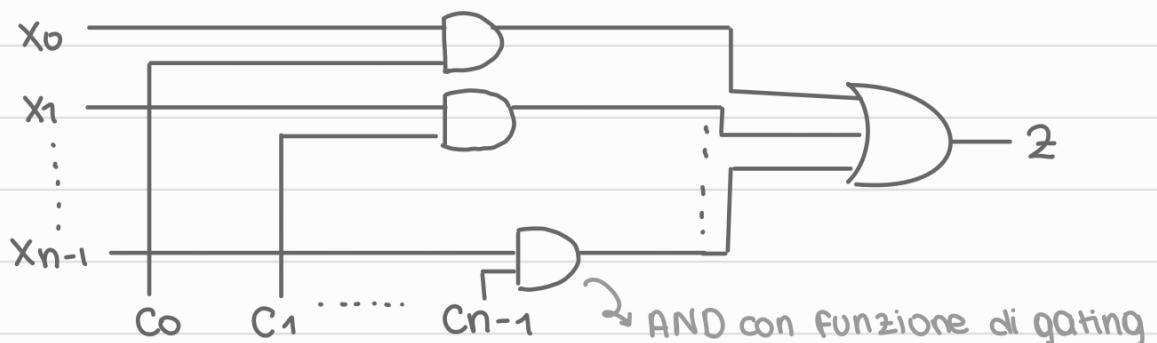
# MULTIPLEXER

il multiplexer o MUX è un circuito combinatorio che ha due o più input e un solo output selezionato in base alle linee di selezione (o di controllo). La sua funzione è quella di selezionare tra due o più linee di ingresso una sola linea da mandare in uscita (come output) in base agli ingressi di selezione.



in questo caso vi sono n linee di selezione di cui una sola vale 1  
CIRCUITO  
(solo una variabile si deve attivare)

Il mux è realizzato, a livello di porte logiche, con n porte AND che confluiscono in una porta OR.



Il MUX può essere implementato anche utilizzando un decodificatore, così da avere  $\log_2 n$  invece di n linee di selezione a (diminuire le linee di selezione sfruttando i mintermini ottenuti dal decodificatore).

Vi sono solitamente n linee di selezione per  $2^n$  input (linee di ingresso).



## ESEMPIO MUX 4-A-1

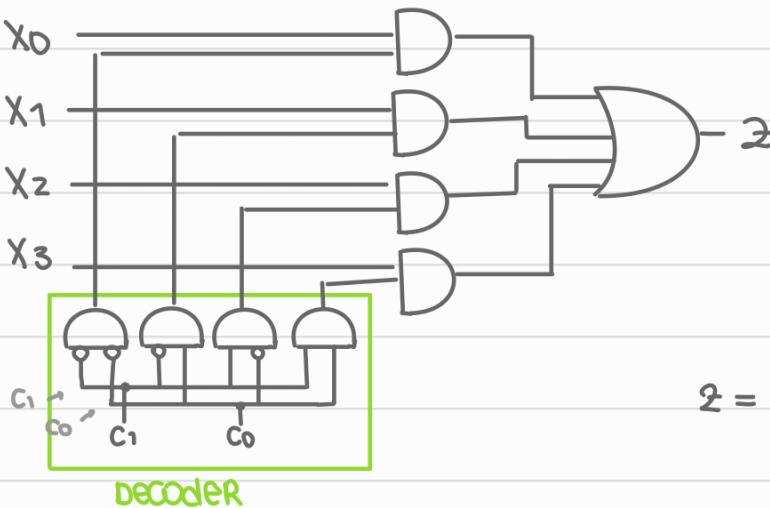


TAVOLA DI VERITA'

$C_1$	$C_0$	2
0	0	$X_0$
0	1	$X_1$
1	0	$X_2$
1	1	$X_3$

$$z = \bar{C}_1 \bar{C}_0 X_0 + C_1 \bar{C}_0 X_1 + \bar{C}_1 C_0 X_2 + C_1 C_0 X_3$$

## IMPLEMENTARE FUNZIONI BOOLEANE CON MUX

Attraverso l'utilizzo del multiplexer è possibile realizzare funzioni booleane ad  $n$  variabili:

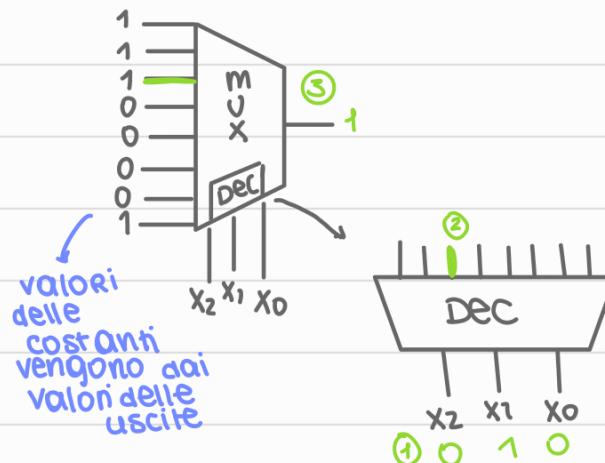
- Le variabili sono espresse con le linee di selezione ( $n$  variabili, sel)
- I valori assunti dalla funzione sono gli ingressi, sulle linee dati ( $2^n$  valori assunti, possibili combinazioni)

### ESEMPIO

TAVOLA DI VERITA' (di una f booleana)

$x_2$	$x_1$	$x_0$	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

REALIZZAZIONE CON MUX 8-A-1 CON DEC



- esempio :
- ① combinazione di variabili
  - ② si attiva un'uscita del dec (minitermine)
  - ③ in base ad essa viene mandata in uscita l'ingresso risultato della funzione

Un implementazione più efficiente di una funzione ad  $n$  variabili consiste nell'utilizzare non  $n$  linee di selezione e  $2^n$  ingressi, bensì  $n-1$  linee di selezione e  $2^{n-1}$  linee di ingresso.

Si ottiene eliminando una linea di selezione (un input della funzione) ed esprimendo la  $y$  in funzione della variabile eliminata:

$x_0$  diventa variabile di entrata (f deve essere espresso in  $x_0$ )

dividiamo in casistiche

e leggiamo f in

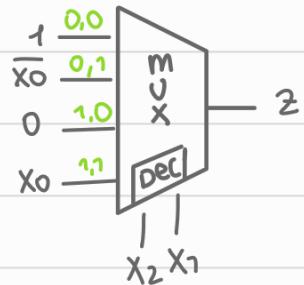
confronto ad  $x_0$

(variabile tolta dalle  
linee di selezione)

$x_2$	$x_1$	$x_0$	$y$
0,0	0	0	1 ↪ y sempre = 1
0,0	0	1	1
0,1	0	1	1 ↪ $\bar{x}_0$
0,1	1	1	0
1,0	1	0	0 ↪ y sempre = 0
1,0	1	1	0
1,1	1	0	0 ↪ $x_0$
1,1	1	1	1

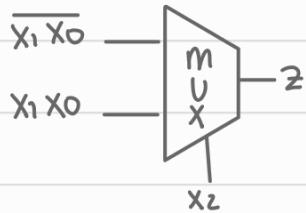
linee di controllo  
↓ 2 sei

MUX 4-A-1



$$Z = 1 \cdot \bar{x}_2 \bar{x}_1 + \bar{x}_0 \cdot \bar{x}_2 x_1 + 0 \cdot x_2 \bar{x}_1 + x_0 \cdot x_2 x_1$$

leviamo un'altra variabile! ( $x_1$ )



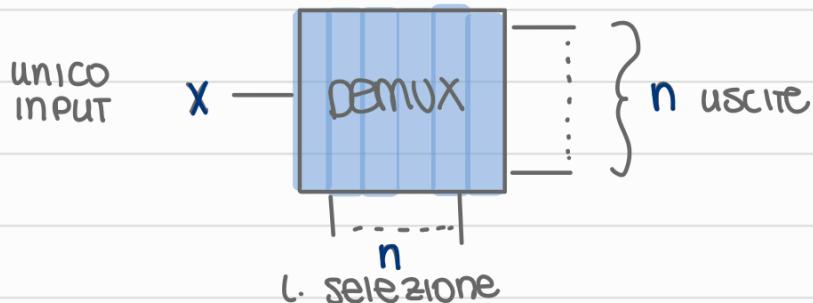
$$Z = \bar{x}_1 x_0 \cdot \bar{x}_2 + x_1 x_0 \cdot x_2$$

siccome SOP 0 e' il  
complemento e 1 e' il  
normale

- \* TRANSCODIFICATORI
- \* COMPARATORE LOGICO
- \* ALU

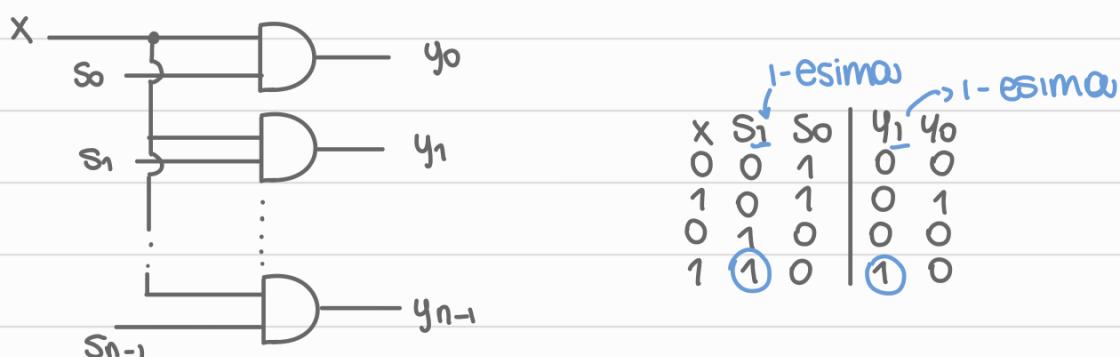
# DEMULTEPLEXER

Il demultiplexer o DE-MUX è una rete combinatoria che svolge una funzione logica inversa rispetto a quella del multiplexer, ovvero quella di inviare il segnale preso in input solo sull'uscita selezionata in base a delle linee di controllo. Il de-mux prende in ingresso una sola linea dati e  $n$  linee di selezione di cui solo una vale 1 e restituisce in output  $n$  linee, di cui solo la linea  $i$ -esima prende il valore della linea dati se la linea  $i$ -esima vale 1.



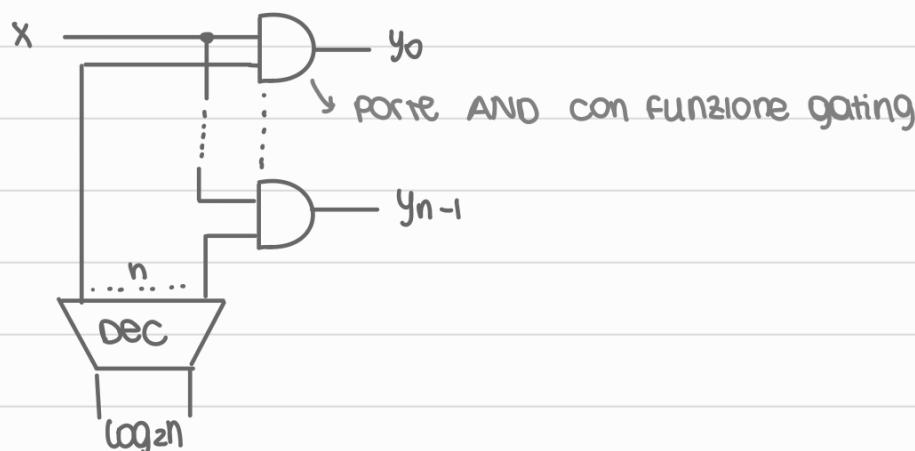
## CIRCUITO

A livello di porte logiche il de-mux è composto da  $n$  porte AND, collegate ciascuna all'input e ad una delle linee di selezione.



Il De-multiplexer può essere realizzato utilizzando un decodificatore: in tal modo i segnali di controllo sono le uscite del decodificatore. Così vi sono  $\log_2 N$  segnali di selezioni ed  $n$  uscite. I mintermini prodotti dal decodificatore vengono utilizzati con funzione di gating (messe in porte and con l'ingresso).

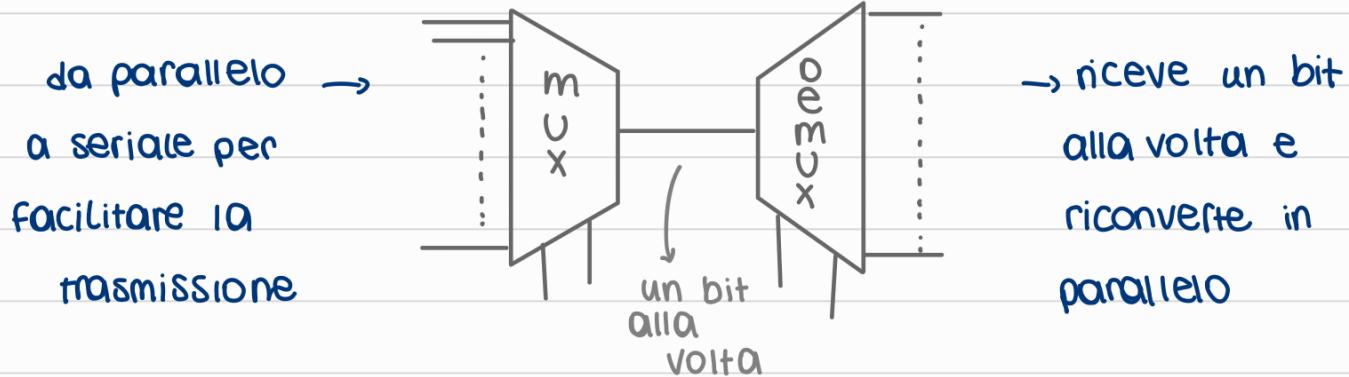
Il segnale di ingresso viene mandato solo all'uscita il cui pedice è uguale al numero decimale che i segnali codificano in binario.



## Mux e Demux in cascata

Il multiplexer e il demultiplexer possono essere utilizzati per facilitare la trasmissione di informazioni, convertendo da parallelo a seriale (mux) o da seriale a parallelo (demux) in base all'esigenza.

Nel caso in cui, ad esempio, bisogna trasmettere un'informazione in parallelo ma non si hanno sufficienti canali è possibile utilizzare un mux e un demux in cascata: è possibile trasmettere in serie delle informazioni originariamente in parallelo (mux), per poi riconvertirle in parallelo attraverso il demux.



# TRANSCODIFICATORE

Un transcodificatore è un dispositivo che effettua la transcodifica di un'informazione, ovvero il trasferimento di dati con una certa codifica ad una codifica diversa.

Da BCD a display a 7 segmenti

Rappresentare le cifre da 0 a 9

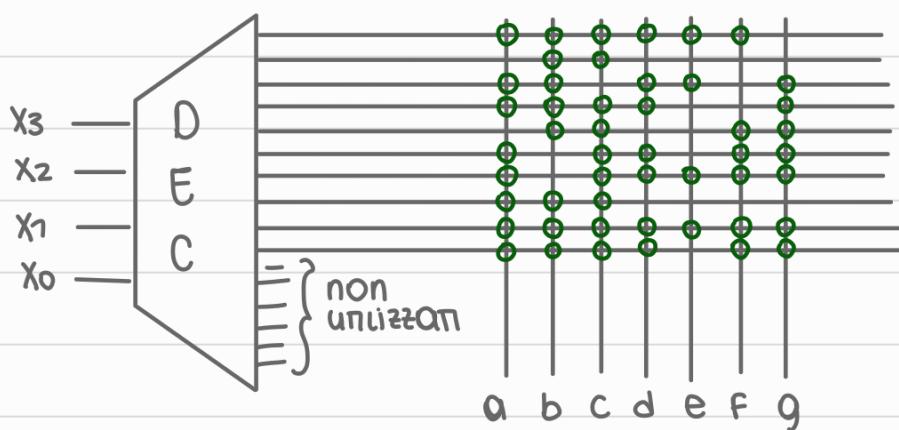
- BCD: codice standard per la rappresentazione del decimale in 4 cifre binarie.  $x_3 x_2 x_1 x_0$

- Display a 7 segmenti:

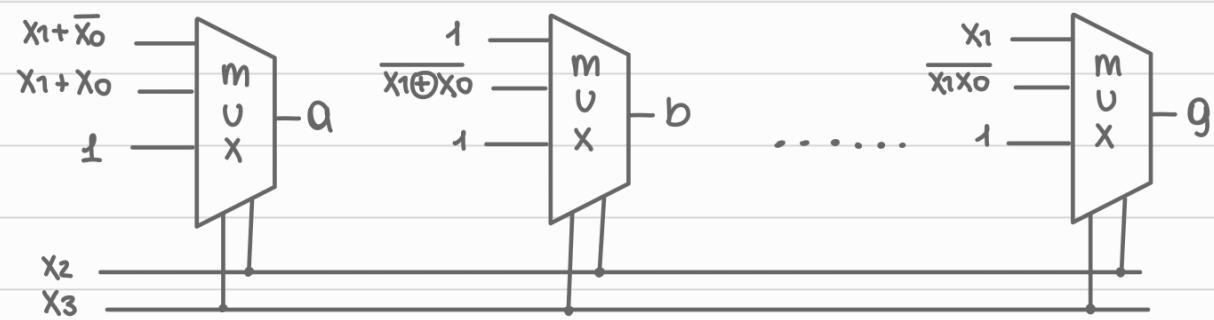


$x_3$	$x_2$	$x_1$	$x_0$	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0
2	0	0	1	0	1	1	0	1	1	0
3	0	0	1	1	1	1	1	1	0	1
4	0	1	0	0	0	1	1	0	0	1
5	0	1	0	1	1	0	1	1	0	1
6	0	1	1	0	1	0	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0
8	1	0	0	0	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	1	1
$\delta \left\{ \begin{array}{l} 1 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 1 \\ 1 \\ 1 \end{array} \right\}$										
non considerate in BCD										

- Decodificatore e codificatore in cascata.



- Con mux 4 a 1

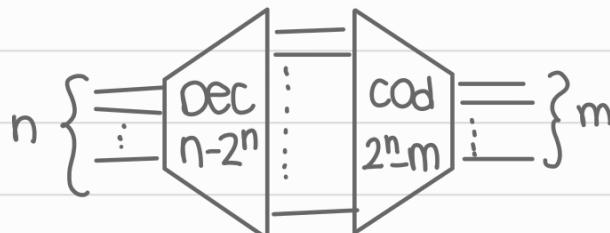


# ROM - Read only memory

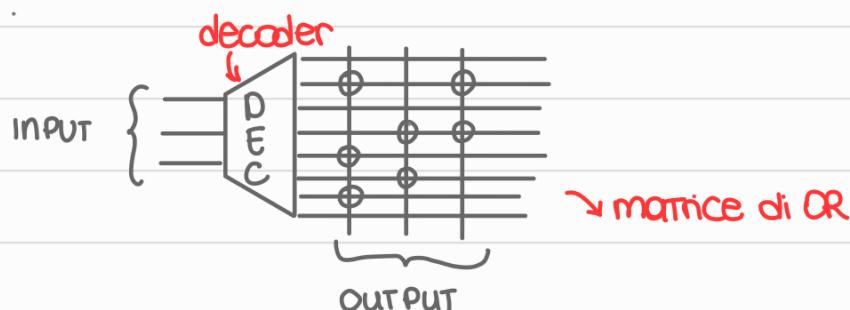
Una ROM è un dispositivo con  $n$  ingressi e  $m$  linee di uscita.

La ROM è composta da un decodificatore e da un codificatore in cascata: una matrice di AND (decodificatore) le cui entrate sono gli input e le uscite sono le entrate di una matrice di OR (codificatore), le cui uscite sono gli output della ROM.

Schema a blocchi



implementazione con decoder + matrice di OR



## Funzioni booleane con la ROM

Tramite la ROM è possibile realizzare funzioni booleane in forma canonica sop (il decoder genera tutti i possibili mintermini che poi vengono selezionati nelle uscite e messi in OR fra di loro).

### esempio

3 input e 3 uscite

$$y = \begin{cases} y_2 = 1 & \text{se } x \text{ e' dispari} \\ y_1 = 1 & \text{se } x > 5, x \text{ binario} \\ y_0 = 1 & \text{se } -1 \leq x \leq 2, x \text{ in CA2} \end{cases}$$

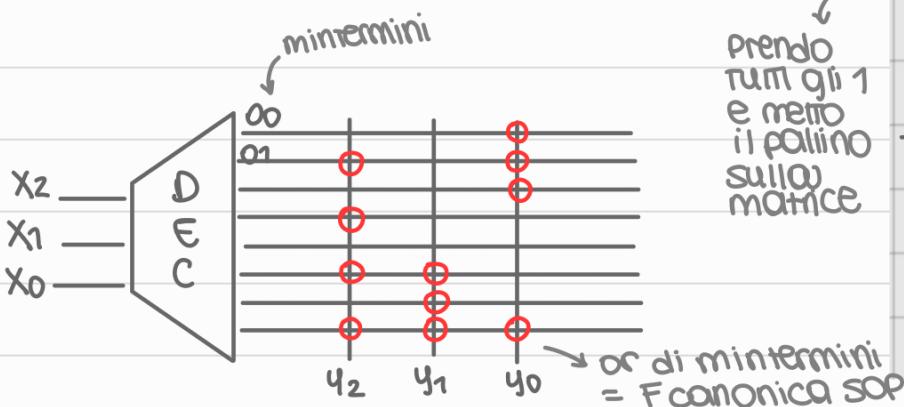
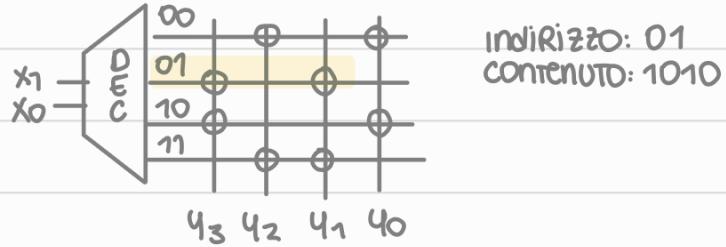


tavola di verità:

x <sub>2</sub>	x <sub>1</sub>	x <sub>0</sub>	y <sub>2</sub>	y <sub>1</sub>	y <sub>0</sub>
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	1	1	1

*In più:* il nome Read Only Memory deriva dal fatto che la ROM può essere vista come una memoria non riscrivibile (di sola lettura). Infatti, è costituita di celle di dimensione fissata, ognuna con il proprio indirizzo (11, 10 ...), come una memoria, ma una volta saldati i diodi i valori memorizzati non possono essere cambiati se non creando un nuovo circuito.



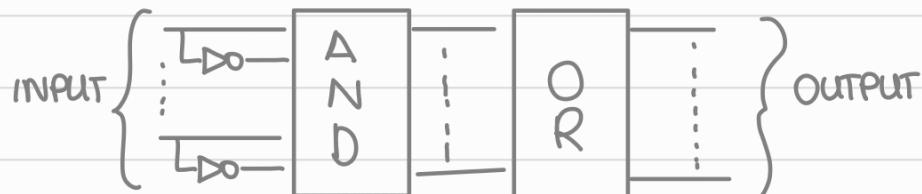
# PLA - Programmable Logic Array

integraz.

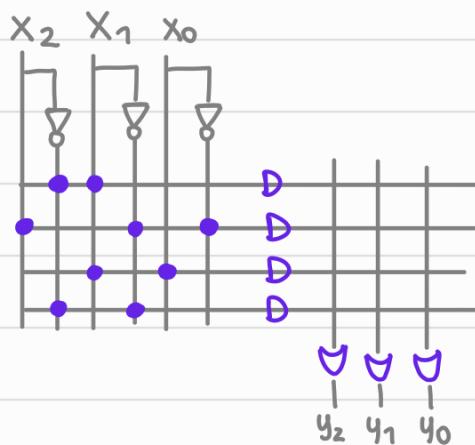
Il PLA è un circuito combinatorio con  $n$  ingressi e  $m$  uscite e tre stadi interni:

- inversione: tutti gli ingressi vengono invertiti, così da avere ogni ingresso e il suo complementare
- matrice di AND:
- matrice di OR:

Schema a blocchi



CIRCUITO



## Funzioni booleane con PLA

Con il PLA è possibile realizzare espressioni booleane nella loro forma minimale sop, in modo più economico ed efficiente di una ROM. Prima vengono realizzati i termini prodotto attraverso la matrice di AND selezionando tra le variabili o il loro complemento, e poi vengono congiunti in OR attraverso la matrice di OR finale.

### ESEMPIO

Riprendendo la tavola di verità dell'esempio della ROM

① OBTENIAMO LE ESPRESSIONI MINIMALI SOP DELLE USCITE

$$y_2 = x_0$$

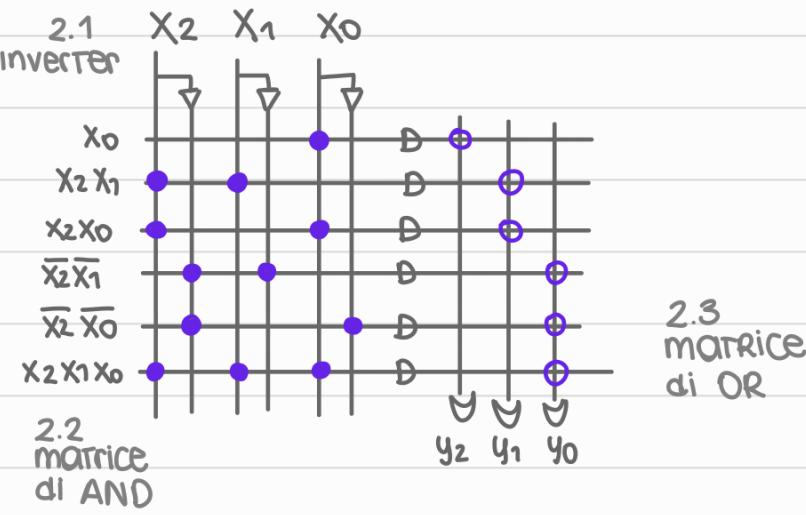
	00	01	11	10
$x_2 \backslash x_1 x_0$	0	0	0	0
0	0	0	0	0
1	0	1	1	1

$$y_1 = x_2 x_0 + x_2 x_1$$

	00	01	11	10
$x_2 \backslash x_1 x_0$	0	1	1	0
0	1	1	0	1
1	0	0	1	0

$$y_0 = \bar{x}_2 \bar{x}_1 + \bar{x}_2 \bar{x}_0 + x_2 x_1 x_0$$

## ② PLA



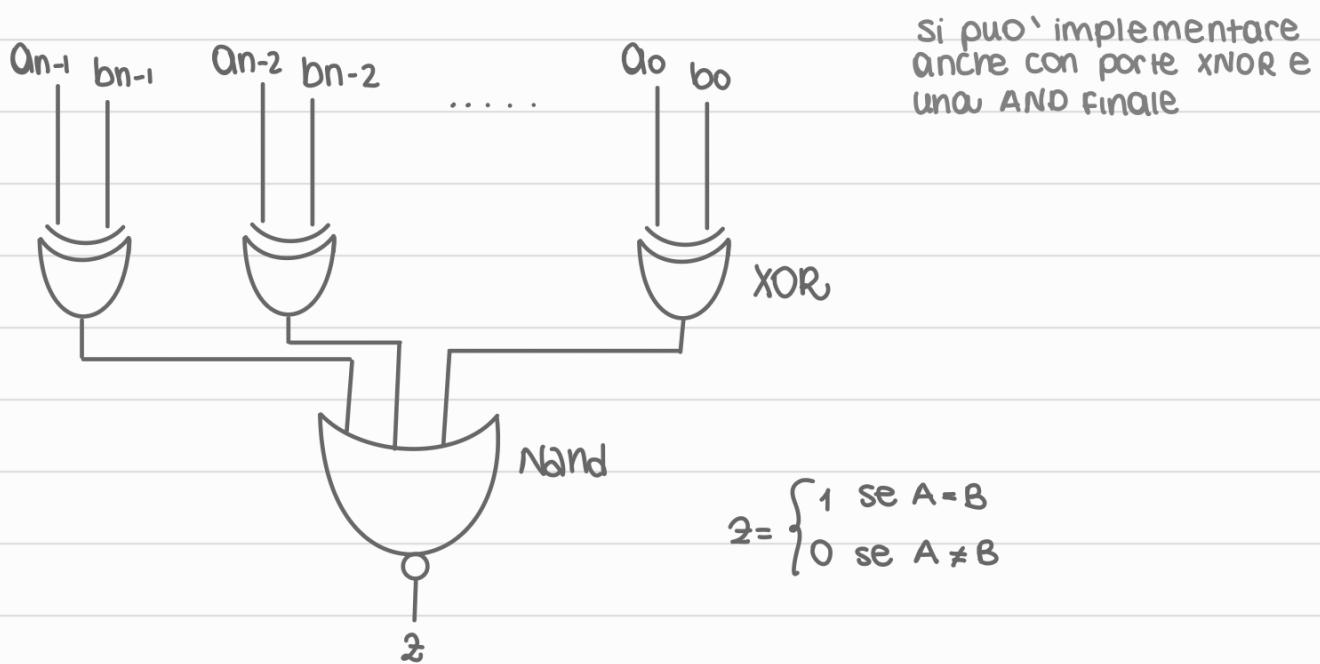
# COMPARATORE

Esistono due tipologie di comparatori:

- comparatore **logico**: confronta due sequenze di bit per verificarne l'uguaglianza
- comparatore **aritmetico**: confronta la magnitudine di due numeri (se un numero A è minore alla sequenza B)

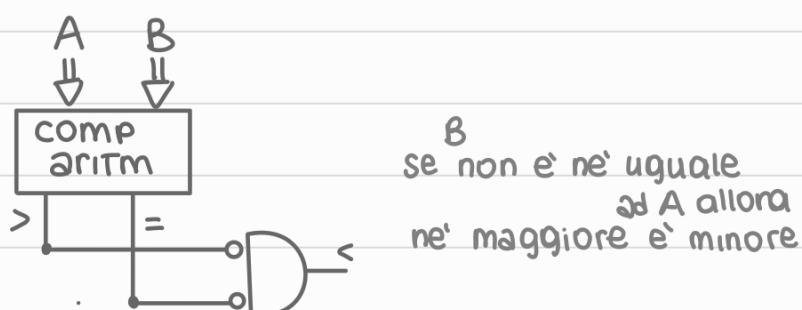
## Comparatore logico

Il comparatore logico confronta sequenze di  $n$  bit, utilizzando porte XOR per comparare ogni coppia di bit (la quale restituisce 1 se i bit sono diversi), per poi far passare tutti gli output attraverso una porta NOR, che restituirà 1 se le due sequenze sono uguali e 0 se anche solo una coppia di bit differisce.



## Comparatore aritmetico

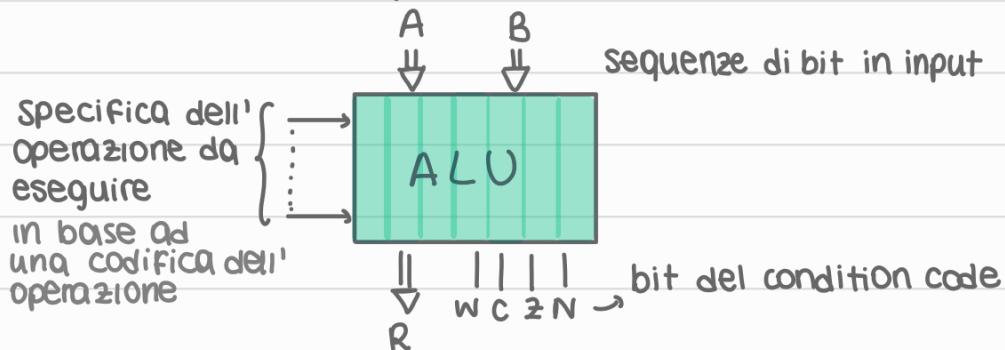
Il comparatore aritmetico confronta la magnitudine tra due numeri A e B. Per verificare se  $A < B$ , si controlla che A non è né maggiore né uguale a B, utilizzando una porta AND con gli ingressi negati. Per controllare l'uguaglianza si utilizza un comparatore logico. Mentre per verificare che  $A > B$  si utilizza un adder: infatti,  $A > B$  può essere riscritto come  $A - B < 0$  e quindi facendo la differenza tra i due numeri se il risultato è negativo (MSB vale 1) A è minore, altrimenti è maggiore.



$$\text{OUTPUT} = \begin{cases} A = B \\ A < B \\ A > B \end{cases}$$

# ALU - Arithmetic Logic Unit

L' ALU è uno dei blocchi digitali fondamentali. Esso svolge sia le operazioni logiche (su singoli bit o bitwise), come AND e OR, sia le operazioni aritmetiche di base, come somma/sottrazione, moltiplicazione e divisione.

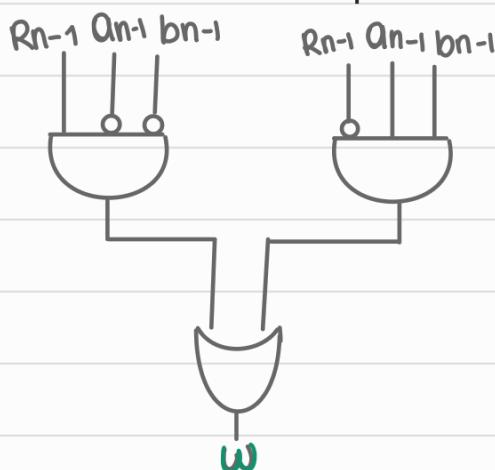


Le alu sono dotate anche di 4 output aggiuntivi, i bit del condition code (bit di controllo):

- **W** - overflow
- **N** - risultato negativo
- **Z** - risultato pari a 0
- **C** - riporto in uscita

## W - overflow

In Ca2 l'overflow si verifica se i due numeri hanno lo stesso segno e il risultato viene discorde da questi due. Perciò bisogna controllare il most significant bit dei due operandi e del risultato per verificare se risultano diversi.

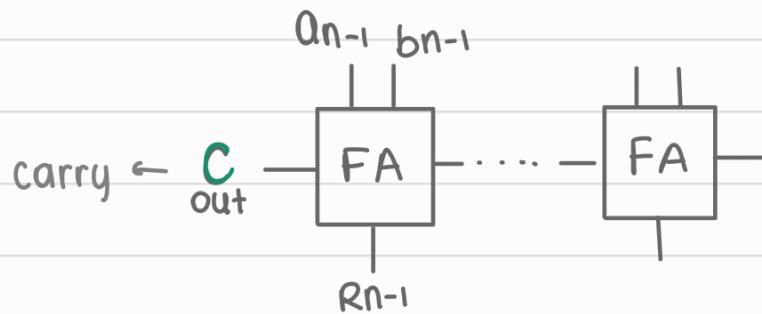


Per implementarlo si utilizzano due porte AND nella quale entrano gli operandi e il risultato complementando una volta i primi e una volta il secondo, le quali confluiscono infine in una porta OR.

Nel caso di rappresentazione standard binaria il segnale di overflow è dato dal carry.

## C - riporto in uscita

Per controllare la presenza del carry (cioè del riporto in uscita) basta verificare il **carry out** del full adder più significativo.



## Z - risultato è 0

Per controllare se è il risultato è pari a zero si implementa un **comparatore logico** che confronta il risultato con zero (gli output somma di ogni adder sono in XOR con 0 e poi in una NOR).

## N - negativo

Per verificare se il risultato è negativo è sufficiente controllare il **bit più significativo** del risultato, il quale è 1 se il numero è negativo.

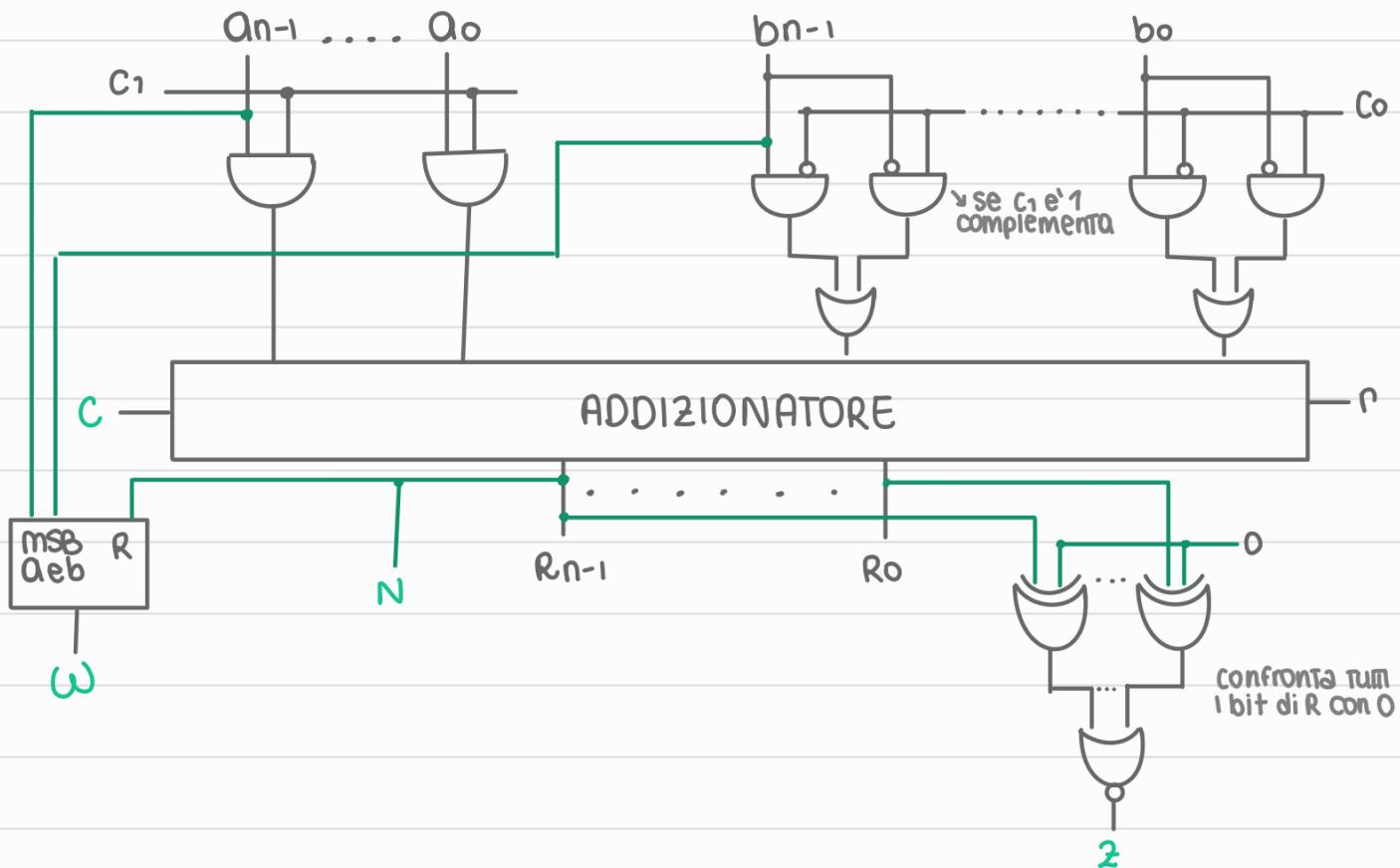
## esempio di ALU

- addizionatore tra A e B
- 3 linee di controllo :
  1. C1 pone l'operando A a 0 (se C1= 0 allora a=0)
  2. C0 complementazione logica di B (se C0= 1 allora B=  $\bar{B}$ )
  3. r incrementa di 1

TAVOLA di verità

C <sub>1</sub>	C <sub>0</sub>	r	RISULTATO
0	0	0	B
0	0	1	B+1
0	1	0	$\bar{B}$
0	1	1	$\bar{B}+1 = -B$
1	0	0	A+B
1	0	1	A+B+1
1	1	0	$A+\bar{B} = A-(B+1)$
1	1	1	$A+\bar{B}+1 = A-B$

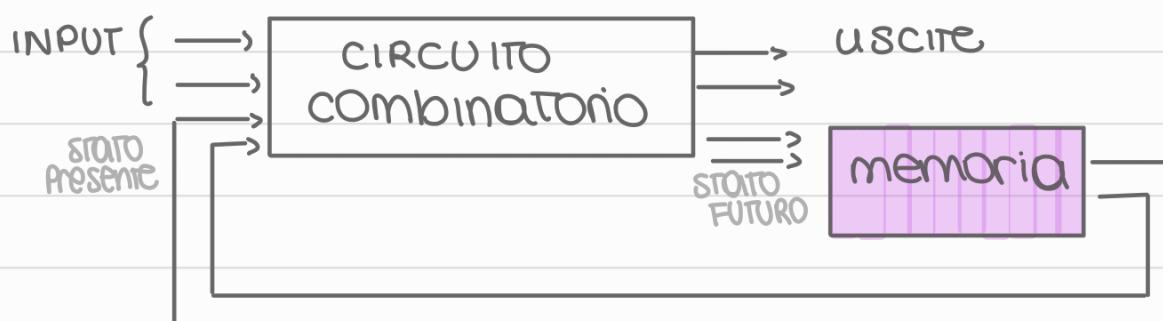
## CIRCUITO



# circuiti sequenziali

I circuiti sequenziali, a differenza dei circuiti combinatori, hanno la capacità di memorizzare informazioni binarie che definiscono istante per istante lo stato del circuito. Le uscite, in un dato istante, non dipendono solo dagli ingressi ma anche dalle uscite degli stati precedenti.

I circuiti sequenziali sono costituiti da un circuito combinatorio e degli elementi di memoria, come illustrato nel seguente schema a blocchi:



## Circuito combinatorio:

- riceve degli input (ingressi)
- produce uscite (output) e uscite che vanno ad alimentare la memoria
- la memoria a sua volta alimenta il circuito combinatorio (ne costituisce degli ingressi)

## Elementi di memoria:

la memoria è costituita da porte logiche e linee di *feedback* (*meccanismo retroattivo*) che riportano in ingresso lo stato delle uscite (l'uscita funge da ingresso di una delle porte che ha prodotto quell'uscita)

Fondamentale nei circuiti sequenziali è il concetto di tempo. Le uscite dipendono non solo dagli ingressi ma anche dal tempo (ovvero dagli stati precedenti). Vi è, infatti, una distinzione tra tempo presente e tempo futuro (che sarà determinato dagli ingressi e dallo stato del tempo presente). *Concetto di stati presenti e stati futuri*

# — LATCH —

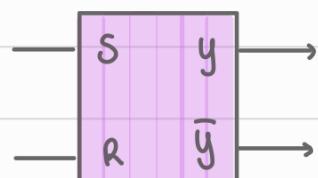
è l'unità di base dei circuiti sequenziali. È in grado di memorizzare (mantenere) un bit di informazione fino a quando non riceve un nuovo segnale di input. E' un componente asincrono, ovvero i valori delle uscite cambiano ogni volta che uno o più ingressi cambiano (sensibilità immediata al cambiamento degli ingressi).

## LATCH SET - RESET (SR)

Il latch sr è il tipo più semplice di latch ed è costruito a partire da due porte NOR con collegamenti incrociati. E' caratterizzato da due input S e R , da un'uscita Y e la sua complementata  $\bar{Y}$ .



rappresentazione circuitale



Il latch sr ha tre funzioni essenziali:

- stato di **set**: quando S è 1, l'uscita Y=1
- stato di **reset**: quando R è 1, l'output Y=0
- **hold (memorizzazione)**: se entrambi gli input sono pari a zero il latch si trova in una fase di memorizzazione durante la quale rimane invariato il valore dell'uscita rispetto allo stato precedente.
- **stato indefinito**: se entrambi gli input valgono 1, il latch entra in una condizione di funzionamento anomalo, in cui entrambe le uscite sono uguali a 0, in contrasto con l'ipotesi per cui sono una il complemento dell'altra.

S	R	$y(t+1)$	→ STATO FUTURO
0	0	$y(t)$	memorizzazione
0	1	0	RESET
1	0	1	SET
1	1	-	non ammissibile

tavola di verita' per ricavare la formula

$s \ r \ y$	$y_{\text{FUTURO}}$	$\bar{s} \bar{r}$	00	01	11	10
0 0 0	0	1	0	0	-	1
0 0 1	1	0	0	0	-	1
0 1 0	0	1	1	1	0	-
0 1 1	0	0	1	1	0	-
1 0 0	1	1	0	1	-	1
1 0 1	1	0	1	0	-	1
1 1 0	-	0	1	1	0	-
1 1 1	-	0	0	1	0	-

$$y = \bar{r}(y+s)$$

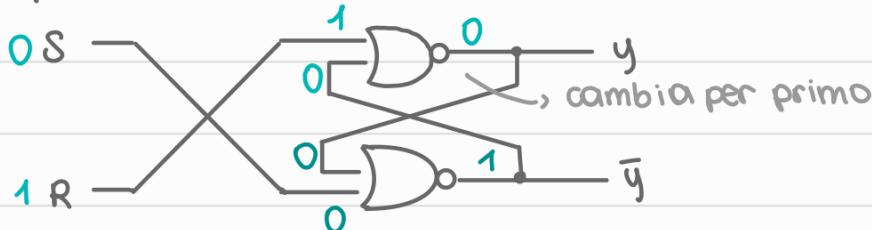
$$y = \overline{r+y} = \overline{r+\overline{s+y}} = \overline{r}(s+y)$$

come lo ha fatto la prof

ESAMINIAMO OGNI STATO:

reset

porta lo stato del latch a 0

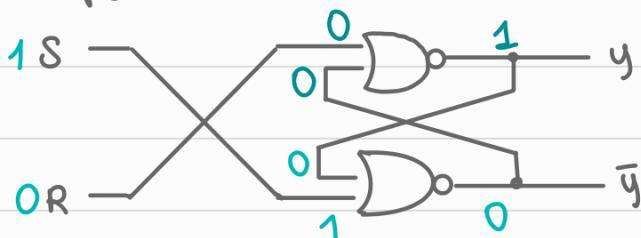


$$y=1 \quad s=0 \quad r=1 \rightarrow y=0$$

$$y = \bar{r}(s+y) = 0(0+1) = 0$$

set

imposta lo stato del latch a 1



$$y=0 \quad s=1 \quad r=0 \rightarrow y=1$$

$$y = \bar{r}(s+y) = 1(1+0) = 1$$

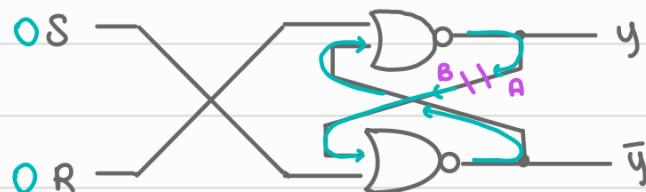
$$\textcircled{1} \quad \bar{y} = \overline{s+y} = \overline{1+0} = 0$$

$$\textcircled{2} \quad y = \overline{R+\bar{y}} = \overline{1+0} = 0$$

## MEMORIZZAZIONE

$S=0 \ R=0 \ Y=Y$  mantiene lo stato precedente

Per dimostrare che il latch mantiene lo stato precedente "tagliamo" il filo che collega l'uscita  $y$  all'entrata della porta NOR. Otteniamo così due punti (A e B). Se, partendo dal punto B, arriviamo al punto A con valore invariato, allora il latch ha mantenuto lo stato precedente.



• CASO 1  $B=0$

$$Y = \bar{R}(S+Y) = \bar{R}(S+0) = 1(0+0) = 0 = A$$

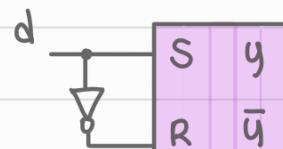
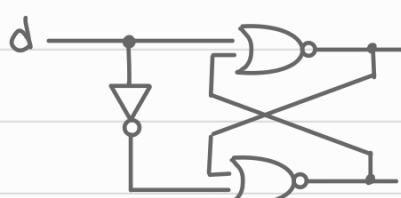
• CASO 2  $B=1$

$$Y = \bar{R}(S+Y) = 1(0+1) = 1 = A$$

Perciò quando gli input sono impostati entrambi a 0 lo stato precedente, indipendentemente da quale esso sia viene memorizzato, mantenuto.

## LATCH DELAY

Per evitare lo stato indefinito del latch sr, viene introdotto un nuovo tipo di latch: il latch D (delay). Il latch d ha un solo input e due output  $Y$  e  $\bar{Y}$ . Il latch di tipo D è un'implementazione del latch SR, ottenuto sostituendo ai due input S e R un unico input, chiamato D, e il suo complemento. Questo significa che le combinazioni in cui le entrate hanno lo stesso valore sono eliminate (di conseguenza è eliminata anche la combinazione 11 che pone il latch in uno stato indefinito).



## DESCRIZIONE DEL FUNZIONAMENTO

siccome sono uno il complementare dell'altro ho 2 casi (01, 10) :

d	Y
0	0
1	1

Reset  
Set

l'uscita assume lo stesso  
valore di D.

# GATED LATCH

Nel gated latch è presente un ingresso supplementare, un segnale di controllo che assicura che gli ingressi cambino valore solo in tempi discreti predefiniti. Spesso viene utilizzato il Clock come segnale di abilitazione, il quale è un segnale periodico.

Il segnale di clock è posto in AND con tutti gli ingressi in modo tale che essi vengano considerati (passino) solo quando il clock ha valore 1 (*edge triggered*).

In questo modo il latch non è sensibile a tutti i cambiamenti degli ingressi (come invece accade senza questo segnale), ma diventa *trasparente* (lascia passare i valori di ingresso) solo quando viene abilitato dal clock (funzione di gating, da cui il nome del tipo di latch).

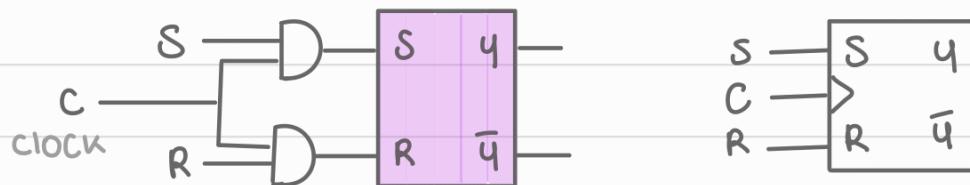


Tavola di verita' con clock

C	S	R	q
0	-	-	y
1	0	0	y
1	0	1	0
1	1	0	1
1	1	1	Indefinita

mem  
mem  
reset  
set  
Indefinita

diagramma temporale :

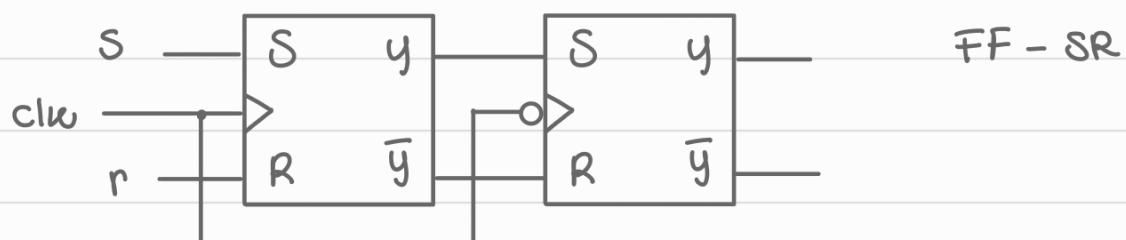


# FLIP-FLOP

La funzionalità del gated latch ha luogo solo se il clock è a 1 per pochissimo tempo. Infatti, nonostante le informazioni sono immagazzinate solo quando il clock è a 1, possono avvenire delle variazioni indesiderate durante un solo ciclo (dovute al fatto che il latch riceve tutti i valori trasmessi mentre è abilitato). Un modo per eliminare queste variazioni è minimizzare tempo in cui il clock è a 1. Tuttavia, questo sistema potrebbe portare ad errori dovuti ai ritardi di propagazione e potrebbe non esserci abbastanza tempo affinchè tutti i componenti del circuito tengano il passo.

Un Flip Flop master slave è composto da due latch collegati in serie, il master (a sinistra) e lo slave (a destra) entrambi collegati al segnale di clock, il quale è negato per l'ingresso allo slave. L'output del master è collegato all'input dello slave. Il master legge i valori di input quando il valore del clock è a 1, mentre lo slave è abilitato solo quando il clock è a 0. Perciò un segnale di input non può passare immediatamente come accadeva per il latch, ma l'output sarà verificato solo nella seconda fase del ciclo del clock (= 0).

In questo modo è possibile allungare i tempi di lettura dei valori degli input e di commettere meno errori a causa di variazioni.



il flip flop riesce a ignorare efficacemente le possibili variazioni indesiderate poichè i due latch sono abilitati in fasi opposte del ciclo di clock.

I FF sono **edge triggered**, ovvero sono sensibili ai fronti d'onda del clock (cioè subiscono variazioni solo durante la salita o la discesa del segnale):

- *positive edge triggered* - sensibile al fronte d'onda positivo
- *negative edge triggered* - sensibile al fronte d'onda negativo

Esempio per comprendere meglio il funzionamento dell'ff e la sua efficacia.

\* non molto affidabile



## JK FLIP FLOP

Nel flip flop JK viene aggiunta una nuova funzionalità che rende accettabile la combinazione 11 in entrata (rendere produttivi questi ingressi).

CIRCUITO:

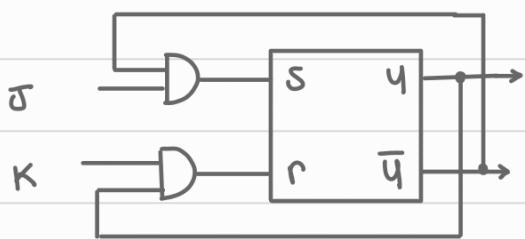


TAVOLA DI VERITA'

$\bar{J}K$	$y$
00	$y$ mem
01	0 RESET
10	1 SET
11	$\bar{y}$ Complemento (mantiene stato precedente)

equazione booleana:

$y_{JK}$	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$y = \bar{J}\bar{K} + J\bar{K}$$

$$y = \bar{r} \cdot (s + y) = \bar{J}\bar{K} + J\bar{K}$$

$$s = \bar{y}J \quad r = Ky$$

## T- FLIP FLOP

Il flip flop T (toggle), ottenuto dal flip flop JK ponendo  $j=k$ , o mantiene il valore memorizzato o lo complementa

CIRCUITO:

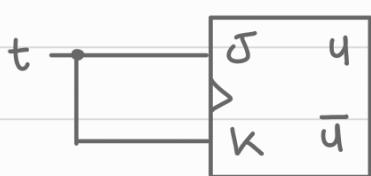
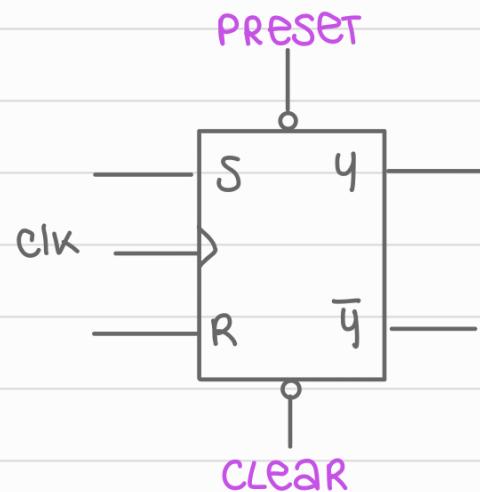


TAVOLA DI VERITA':

t	$y$
0	$y$
1	$\bar{y}$

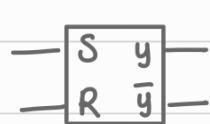
# FLIP FLOP + INGRESSI ASINCRONI

I Flip Flop possono essere dotati anche da due ulteriori ingressi asincroni, chiamati **preset** e **clear**, che funzionano in modo asincrono rispetto al clock: essi cioè settano o resettano il flip flop in modo istantaneo, indipendentemente dagli usuali input o dal clock (non sono filtrati dalla porta And di gating del clock).



# scheda RIASSUNTIVA di FF

## FF-SR



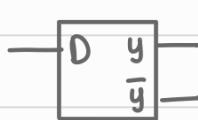
sr	y
00	0
01	1
10	1
11	-

y	4	sr
00	0	00
01	1	10
10	1	01
11	-	00

funzione  
di  
eccitazione

= inverso tavole  
di verità  
utili x  
sintesi

## FF-D

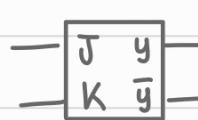


d	y
0	0
1	1

y	4	d
00	0	0
01	1	1
10	0	0
11	1	1

funzione  
di  
eccitazione

## FF-JK

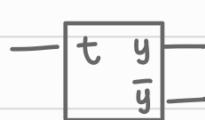


JK	y
00	0
01	1
10	1
11	0

y	4	JK
00	0	00
01	1	10
10	1	01
11	0	00

funzione  
di  
eccitazione

## FF-T



t	y
0	0
1	1

y	4	t
00	0	0
01	1	1
10	1	1
11	0	0

funzione  
di  
eccitazione

Le tabelle di verità includono:

- valori di ingresso
- valori di uscite degli istanti precedenti
- valori delle uscite degli istanti futuri.

# AUTOMA A STATI FINITI con OUTPUT

RAPPRESENTAZIONE GRAFICA delle transizioni di stato

L'automa a stati finiti è definito da 6 elementi:

- $\Sigma$  alfabeto: è un insieme finito di simboli (input)
- $\Sigma$  : insieme finito di stati
- $\delta$  : funzione di transizione (archi) da STATO a STATO

$$\delta: Q \times \Sigma \rightarrow Q \text{ azione}$$

- $V$  : insieme finito di simboli in uscita
- $\lambda$  : funzione di transizione da stato ad output

$$\rightarrow \lambda_{mealy} = Q \times \Sigma \rightarrow V$$

$$\rightarrow \lambda_{moore} = Q \rightarrow V$$

Vi sono due modelli di automa: l'automa di Mealy e quello di Moore.

## → mealy

- archi: ENTRATE + USCITE (etichettati)
- nodi: STATI

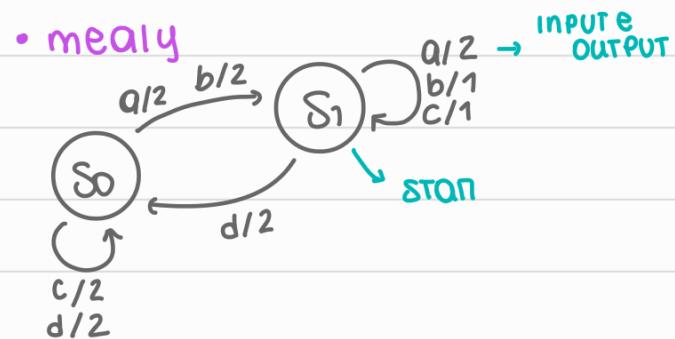
## → moore

- archi: ENTRATE
- nodi: STATI + USCITE (associati)

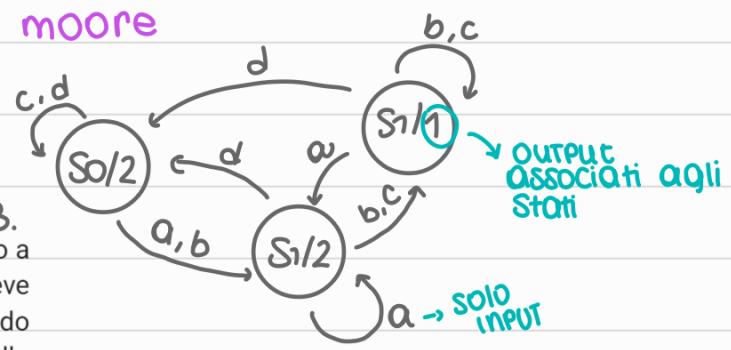
## esempio

	a	b	c	d
s <sub>0</sub>	s <sub>1/2</sub>	s <sub>1/2</sub>	s <sub>0/2</sub>	s <sub>0/2</sub>
s <sub>1</sub>	s <sub>1/2</sub>	s <sub>1/1</sub>	s <sub>1/1</sub>	s <sub>0/2</sub>

↳ tabella stati futuri



## • moore



N.B.  
se vi è uno stato associato a diverse uscite ogni coppia deve essere trattata allo stesso modo  
(collegata a tutte le uscite dello stato)

# MINIMIZZAZIONE

**STATI EQUIVALENTI:** due stati si dicono equivalenti se a fronte di ogni simbolo di ingresso transitano sullo stesso stato successivo e producono lo stesso output.

due stati equivalenti possono essere uniti in un unico stato equivalente.

- Passo 1: eliminare stati irraggiungibili dallo stato iniziale

- Procedimento: **Tabella Triangolare**

Confronto ogni stato con tutti gli altri (solo i seguenti nella tabella dell'automa, per non confrontare più volte gli stessi stati). mettere X

1. Se gli output sono diversi -> gli stati non sono equivalenti

2. se gli output sono uguali -> confronto gli stati futuri

- se sono diversi: scrivo nella cella le coppie di stati futuri

- se sono uguali ~~gli stati sono equivalenti~~

3. Infine controllare le coppie nelle celle: mettere X

- se non risultano equivalenti -> non sono equivalenti mettere O

- se risultano equivalenti -> lo sono anche le coppie della cella

**ESEMPIO :**

	a	b				
S <sub>1</sub>	S <sub>2</sub> /0	S <sub>6</sub> /0	S <sub>2</sub> X			
S <sub>2</sub>	S <sub>1</sub> /0	S <sub>3</sub> /1	S <sub>3</sub> X S <sub>4</sub> S <sub>7</sub>			
S <sub>3</sub>	S <sub>0</sub> /0	S <sub>3</sub> /1	S <sub>4</sub>	X	X	X
S <sub>4</sub>	S <sub>3</sub> /1	S <sub>7</sub> /0	S <sub>5</sub>	X	X	X
S <sub>5</sub>	S <sub>2</sub> /0	S <sub>7</sub> /0	S <sub>6</sub>	X	X	X
S <sub>6</sub>	S <sub>3</sub> /1	S <sub>7</sub> /0	S <sub>7</sub>	X	X	X
S <sub>7</sub>	S <sub>4</sub> /0	S <sub>6</sub> /0		S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>

*sono equivalenti*

*non riscrivo S<sub>7</sub>, S<sub>5</sub> che sono proprio gli stati che sto analizzando*

*dal secondo stato in poi*

*fino al penultimo stato*

Vi sono casi in cui l'equivalenza non è così evidente e perciò bisogna farsi anche un grafo:

esempio 2:

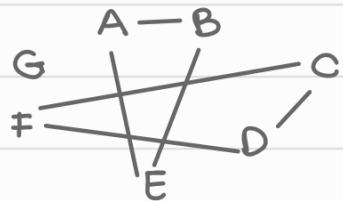
	0	1
A	G/00	C/01
B	G/00	D/01
C	D/10	A/01
D	C/10	B/01
E	G/00	F/01
F	F/10	E/01
G	A/01	F/11

B	CD			
C	X	X		
D	X	X	AB	
E	CF	DF	X	X
F	X	X	FD AE	CF BE
G	X	X	X	X
	A	B	C	D
	E			F

→ ogni equivalenza dipende da un'altra coppia

grafo:

collego tutti gli stati associati tra di loro x vedere se godono di simmetria e transitività tra di loro



$$A' = \{A, B, E\} \text{ tutti collegati fra loro}$$

$$C' = \{C, D, F\}$$

riscrivendo la tabella:

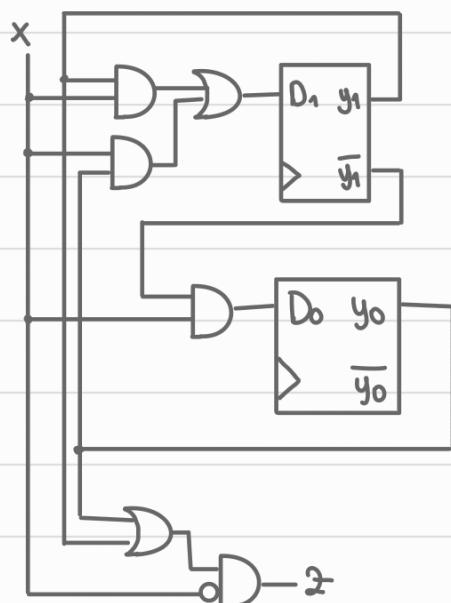
	0	1
A'	G/00	C'/01
C'	C'/10	A'/01
G	A'/01	C'/11

# ANALISI reti sequenziali

Partire dal circuito per arrivare all'automa

1. Scrivere espressioni booleane delle entrate ai flip flop (funzioni di eccitazione) e delle uscite
2. Tavola degli stati futuri, in cui sono inseriti:
  - Ingressi
  - Valori delle funzioni di eccitazione
  - Output
  - Espressioni delle uscite
  - Stati attuali
  - Stati futuri
3. Disegnare il diagramma rete sequenziale (automa a stati finiti)

ESEMPIO:



① espressioni booleane

$$d_1 = x y_1 + x y_0 = x(y_1 + y_0)$$

$$d_0 = x \bar{y}_1$$

$$z = (y_0 + y_1) \bar{x}$$

Ricordando la t.v. dell'FF d

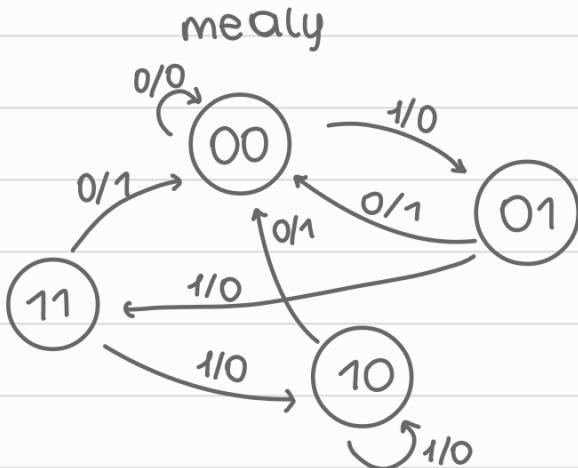
d	1	0	0
0	0	1	0
1	1	1	1

② TAVOLA STATI FUTURI

x	y <sub>1</sub>	y <sub>0</sub>	d <sub>1</sub>	d <sub>0</sub>	z	y <sub>1</sub>	y <sub>0</sub>	STATI FUTURI
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0
0	1	1	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1
1	0	1	1	1	0	1	1	1
1	1	0	1	0	0	1	0	0
1	1	1	1	0	0	1	0	0

### 3 Diagramma di stato : AUTOMA A STATI FINITI

$2 \text{ FF} = 2^2 \text{ STATI}$



X ottenere l'automa  
si confrontano stati  
attuali e futuri e  
l'input che ne determina  
la transizione

### 4 Facoltativo : ASTRAZIONE e TABELLA

Con astrazione si intende assegnare dei nomi simbolici alle variabili binarie (stati, uscite, input)

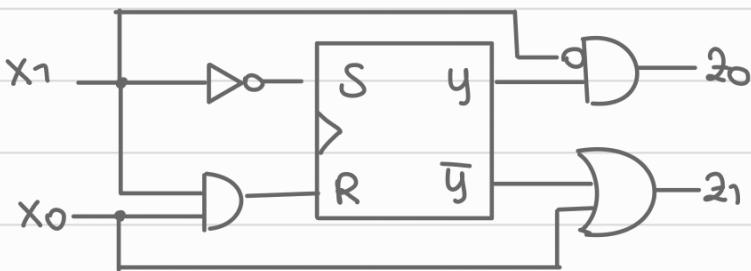
• STATI .

$$\begin{array}{ll} 00 \rightarrow Q_0 & \text{oppure } 00 \rightarrow A \\ 01 \rightarrow Q_1 & 01 \rightarrow B \\ 10 \rightarrow Q_2 & 10 \rightarrow C \\ 11 \rightarrow Q_3 & 11 \rightarrow D \end{array}$$

→ TABELLA

	0	1	INPUT
A	A/0 B/0		uscite
B	A/1 D/0		
C	A/1	C/0	
D	A/1	C/0	
STATI			

## esempio 2



### 1. espressioni booleane

$$S = \bar{x}_1$$

$$R = x_1 x_0$$

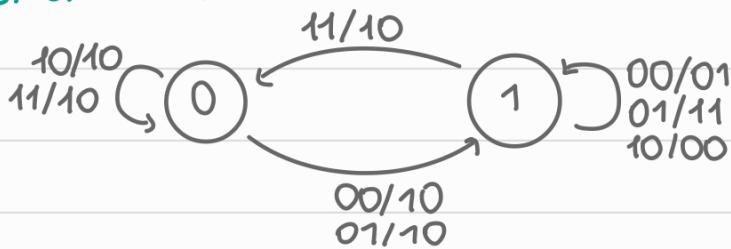
$$z_0 = \bar{x}_1 y$$

$$z_1 = \bar{y} + x_0$$

### 2. tavola degli stati futuri

$x_1 x_0 y$	S	r	$z_1$	$z_0$	y
0 0 0	1	0	1	0	1
0 0 1	1	0	0	1	1
0 1 0	1	0	1	0	1
0 1 1	1	0	1	1	1
1 0 0	0	0	1	0	0
1 0 1	0	0	0	0	1
1 1 0	0	1	1	0	0
1 1 1	0	1	1	0	0

### 3. AUTOMA



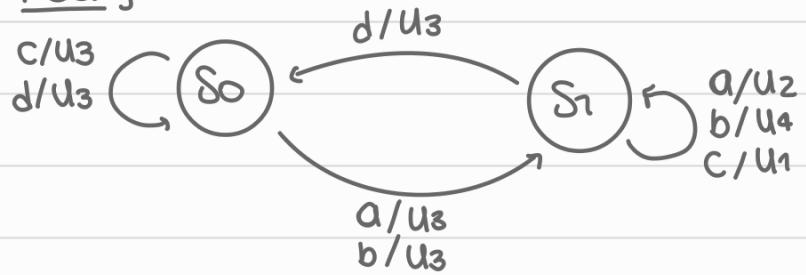
### 4. astrazione / codifica

stati

$$0 \rightarrow s_0, 1 \rightarrow s_1$$

ingressi	uscite
$00 \rightarrow a$	$00 \rightarrow u_1$
$01 \rightarrow b$	$01 \rightarrow u_2$
$10 \rightarrow c$	$10 \rightarrow u_3$
$11 \rightarrow d$	$11 \rightarrow u_4$

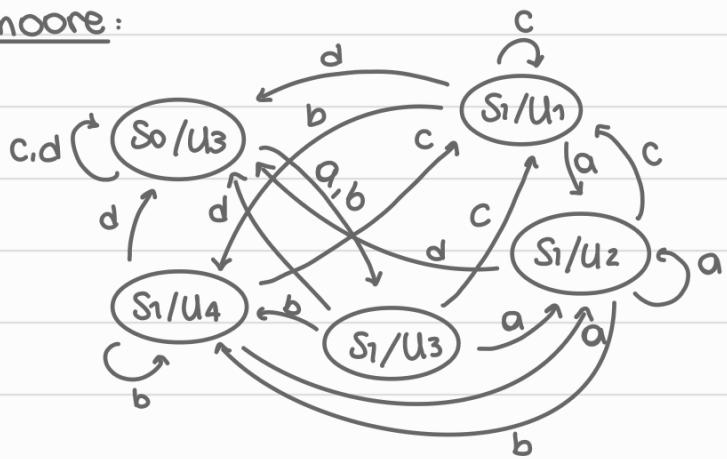
riscriviamo gli automi con la codifica  
mealy:



tabella

	$a$	$b$	$c$	$d$
$S_0$	$S_1/U_3$	$S_1/U_3$	$S_0/U_3$	$S_0/U_3$
$S_1$	$S_1/U_2$	$S_1/U_4$	$S_1/U_1$	$S_0/U_3$

moore:



# SINTESI reti sequenziali

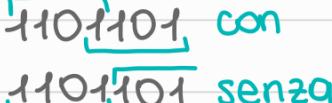
Partire dalla specifica verbale per arrivare al circuito

- realizzare il diagramma di stato (tramite tabella o automa)
- Fare la codifica binaria degli stati ( e se necessario anche di ingressi e uscite)
- Realizzare tavola degli stati futuri e scegliere il flip flop (utilizzare le funzioni di eccitazione per trovare i valori di ingressi dei flip flop )
- Trovare le esprerssioni booleane che descrivono le funzioni di escitazione e le uscite
- Disegnare la rete

esempio :

specifiche verbali: *in ingresso si riceve una sola linea x. Riconoscere se si riceve la sequenza 1101 e in questo caso produrre in uscita z = 1*

a questo punto si possono distinguere due casi:

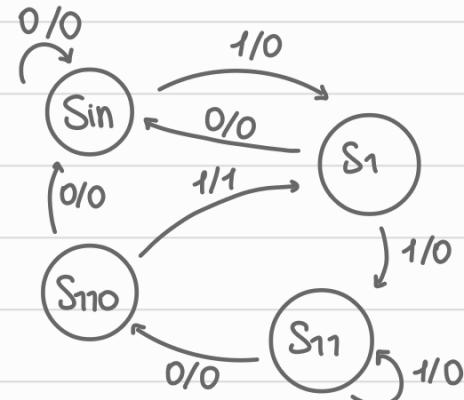
- con sovrapposizione: l'ultimo bit riconosciuto è anche il primo bit della sequenza successiva  

- senza sovrapposizione  


## 1. diagramma di stato

definire uno stato iniziale in cui ancora non si ha memorizzato nulla

	0	1
Si <sub>n</sub>	Si <sub>n</sub> /0	Si <sub>n</sub> /0
Si <sub>1</sub>	Si <sub>1</sub> /0	Si <sub>1</sub> /0
Si <sub>11</sub>	Si <sub>11</sub> /0	Si <sub>11</sub> /0
Si <sub>110</sub>	Si <sub>110</sub> /0	Si <sub>110</sub> /1

Riconosciuto con sovrapposizione  
il 1° bit  
specificare significato



## 2. codifica

$$4 \text{ stati} = 2 \text{ bit} = 2 \text{ FF}$$

Si<sub>n</sub> = 00 ingressi e uscite sono già

Si<sub>1</sub> = 01 espressi in binario non

Si<sub>11</sub> = 10 necessitano di codifica

Si<sub>110</sub> = 11

### 3. TAVOLA degli STATI FUTURI

$x$	$y_1$	$y_0$	$y_1$	$y_0$	$z$	$s_1$	$u_1$	$j_0$	$k_0$
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0
0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0	1	1
1	0	0	1	0	0	0	0	1	0
1	0	1	1	0	1	0	1	0	1
1	1	0	0	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1	0

$y$	$Y$	$JK$
0	0	00
0	1	01
1	0	10
1	1	11

scegliere FF : 2 JK

### 4. espressioni booleane con K-mappe

$J_1 = y_0 \cdot x$					
$y_1 y_0$	$x$	00	01	11	10
0	0	0	0	$\delta$	$\delta$
1	0	1	$\delta$	$\delta$	$\delta$

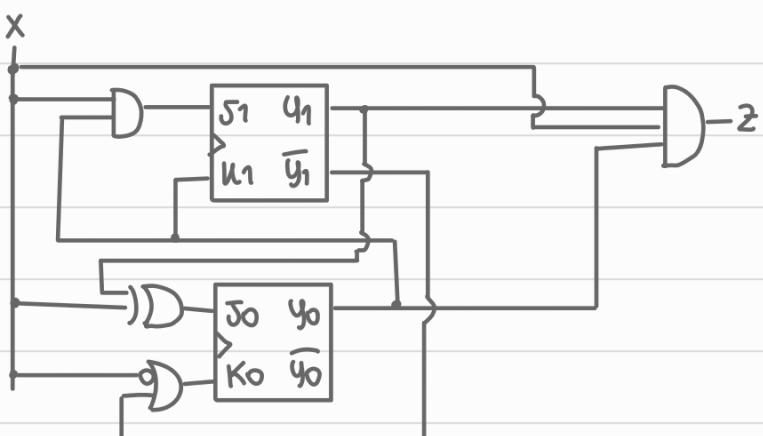
$u_1 = y_0$					
$y_1 y_0$	$x$	00	01	11	10
0	0	0	$\delta$	1	0
1	0	$\delta$	$\delta$	1	0

$j_0 = x\bar{y}_1 + y_1\bar{x} = x \oplus y_1$					
$y_1 y_0$	$x$	00	01	11	10
0	0	0	$\delta$	$\delta$	1
1	0	1	$\delta$	$\delta$	0

$k_0 = \bar{x} + \bar{y}_1$					
$y_1 y_0$	$x$	00	01	11	10
0	0	$\delta$	1	1	$\delta$
1	0	$\delta$	1	0	$\delta$

$z = x y_1 y_0$  c'è un solo 1, è un mintermine

### 5. CIRCUITO



## ESEMPIO 2:

**Specifiche verbali:** Realizzare l'automa per una macchina distributrice che riceve in input 10 o 20 centesimi fino ad arrivare a 50 centesimi e specificare il resto.

2 possibilità:

memorizzare il resto  
**SOVRAPPOSIZIONE**

avere un secondo output per indicare la presenza

o meno di resto (può essere solo di 10 c)  
**NO SOVRAPPOSIZIONE**

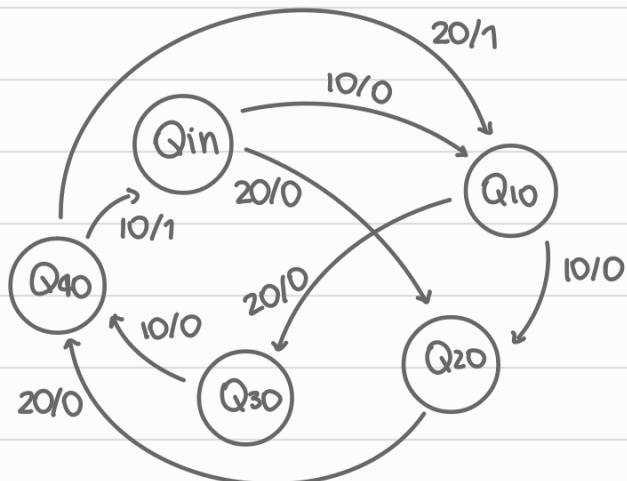
### 1. DIAGRAMMA DI STATO

INPUT = 10 c, 20 c

OUTPUT =  $\begin{cases} 0 \\ 1 \text{ raggiunti i } 50\text{c totali} \end{cases}$

	10	20
Qin	Q <sub>10</sub> /0	Q <sub>20</sub> /0
Q <sub>10</sub>	Q <sub>20</sub> /0	Q <sub>30</sub> /0
Q <sub>20</sub>	Q <sub>30</sub> /0	Q <sub>40</sub> /0
Q <sub>30</sub>	Q <sub>40</sub> /0	Qin/1
Q <sub>40</sub>	Qin/1	Q <sub>10</sub> /1

senza sovrapposizione  
sarebbe Qin/1 → x segnalare il resto



### 2. CODIFICA

INPUT :  $X = \begin{cases} 0 & 10\text{ c} \\ 1 & 20\text{ c} \end{cases}$

q<sub>2</sub> q<sub>1</sub> q<sub>0</sub>

STATI : 000 = Qin  
001 = Q<sub>10</sub>  
010 = Q<sub>20</sub>  
011 = Q<sub>30</sub>  
100 = Q<sub>40</sub>

### 3. TAVOLA STATI FUTURI

X

### ESEMPIO 3:

**SPECIFICA VERBALE:** Progettare l'automa riconoscitore delle sequenze STO e OTO. Prende in input le lettere S, T, O e produce in output 1 quando riconosce le sequenze con sovrapposizione.

STO  
OTO

### 1. DIAGRAMMA DI STATO

INPUT: O, S, T

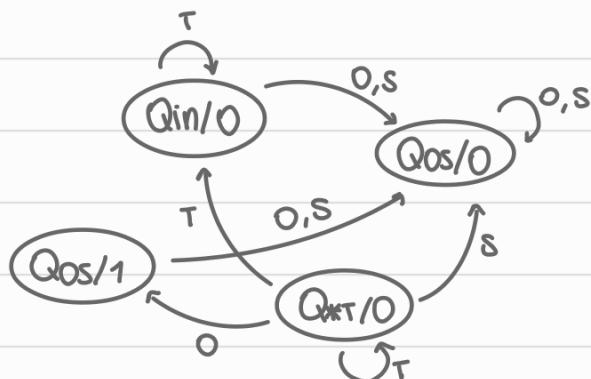
OUTPUT: 0, 1

	O	S	T
Qin	Qo/0	Qs/0	Qin/0
Qo	Qo/0	Qs/0	QOT/0
Qs	Qo/0	Qs/0	QST/0
QOT	Qo/1	Qs/0	Qin/0
QST	Qo/1	Qs/0	Qin/0

possiamo minimizzare l'automa

$$QOT = QST \rightarrow Qo = Qs$$

	O	S	T
Qin	Qo/0	Qs/0	Qin/0
QoS	Qo/0	Qs/0	QOT/0
Q*T	Qo/1	Qs/0	Qin/0



### 2. CODIFICA

INPUT:  $x_1 x_0$

$$\begin{array}{ll} 0\ 1 &= O \\ 1\ 0 &= T \\ 1\ 1 &= S \end{array}$$

STATI:  $q_1\ q_0$

$$\begin{array}{ll} 0\ 0 &= Qin \\ 0\ 1 &= Qos \\ 1\ 1 &= Q*T \end{array}$$

### 3. TAVOLA degli STATI FUTURI

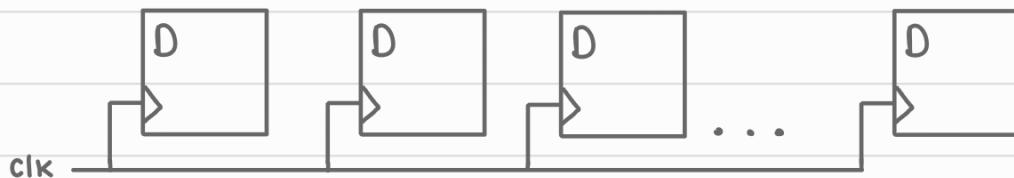
$x_1$	$x_0$	$u_1$	$u_0$	$y_1$	$y_0$	$z$
0	0	0	0	δ	δ	
0	0	1	1			
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	δ	δ	δ
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	1	1	0
1	0	1	0	δ	δ	δ
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	δ	δ	δ
1	1	1	1	0	1	0

# REGISTRI

Un registro è un insieme di  $n$  Flip-flop in cui memorizzare delle informazioni ad  $n$  bit (mentre i ff memorizzano un bit di informazione).

Le operazioni adoperate su un registro sono:

- mantenimento delle informazioni (memorizzazione)
- caricamento del valore di informazione



→ RAPPRESENTAZIONE schematica



## CARICAMENTO DELLE INFORMAZIONI:

il caricamento delle informazioni all'interno dei registri può avvenire in due diverse modalità:

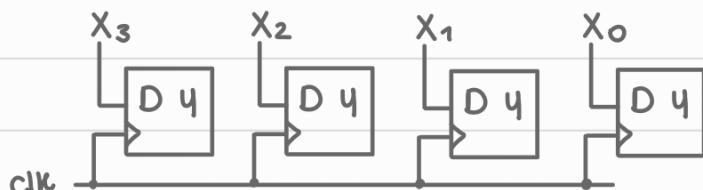
- caricamento parallelo (tutti i flipflop ricevono contemporaneamente il valore)
- caricamento seriale (il segnale è unico e si propaga a mano a mano in tutti i flip flop)

## PARALLELO

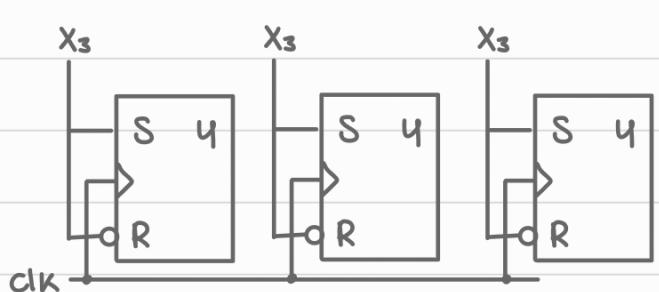
I registri con caricamento parallelo ricevono in input  $n$  linee dati (tante linee dati quanti ff lo costituiscono), le quali vengono caricate quando permesso dalla sensibilità del clock.

Per controllare cosa è stato caricato sul registro basta prendere l'uscita dei ff.

### REGISTRO di 4 FF - d



### REGISTRO di 4 FF - SR



↳ combinazioni in cui sono l'una il complemento dell'altra = FF-d

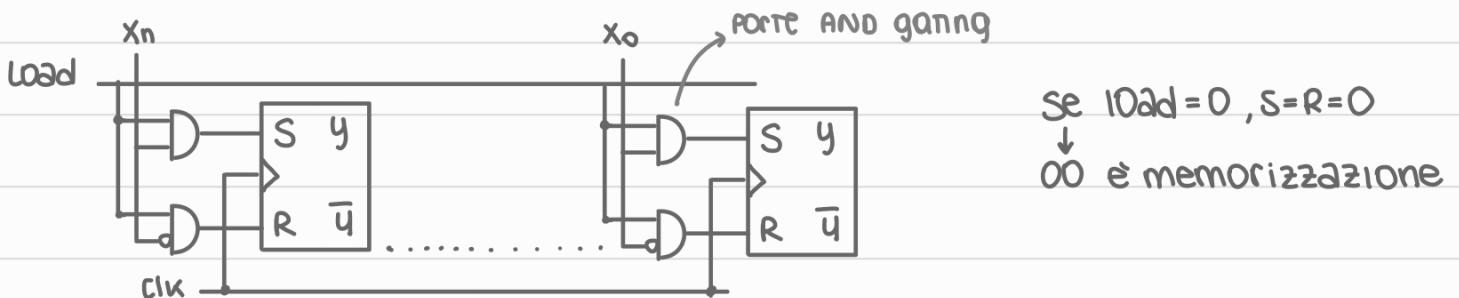
In tal modo per mantenere i valori all'interno dei ff basta mantenere uguali i valori degli ingressi. Tuttavia, è possibile, aggiungendo un ulteriore segnale di controllo chiamato **load**, distinguere la fase del caricamento da quella della memorizzazione.

(utilizzando delle porte and gating) Quando i load vale 1 i valori vengono caricati, mentre se vale zero vengono mantenuti i valori precedenti (memorizzazione).

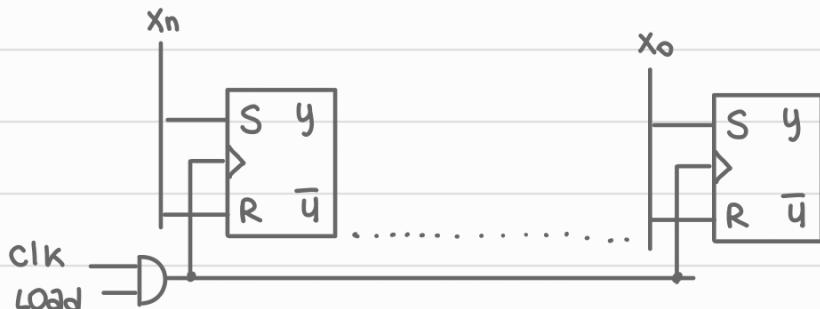
$$\text{Load} = \begin{cases} 1 & \text{CARICAMENTO} \\ 0 & \text{MEMORIZZAZIONE} \end{cases}$$

Il segnale di Load può essere associato sia agli input, che direttamente al clock.

### Load sugli INPUT FF-SR

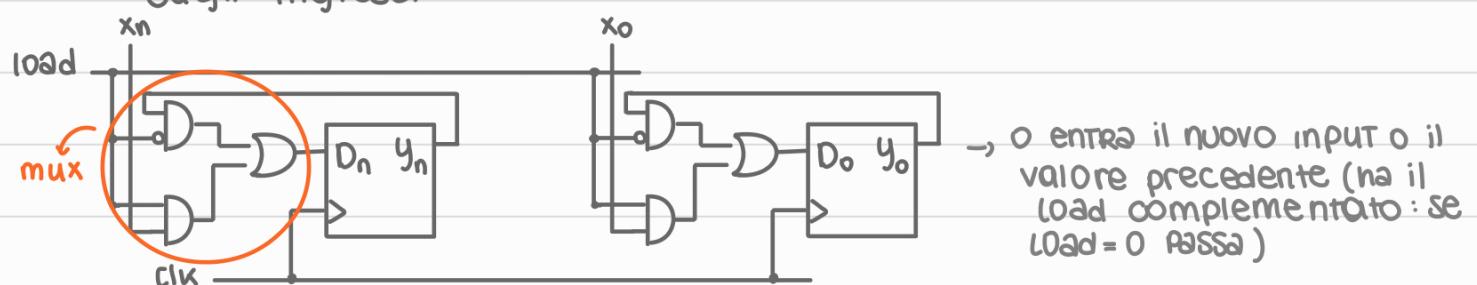


### Load sul CLOCK FF-SR

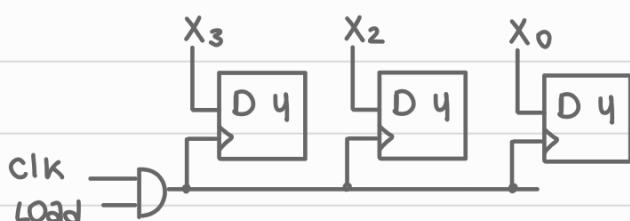


### FF-d con segnale Load

→ sugli ingressi



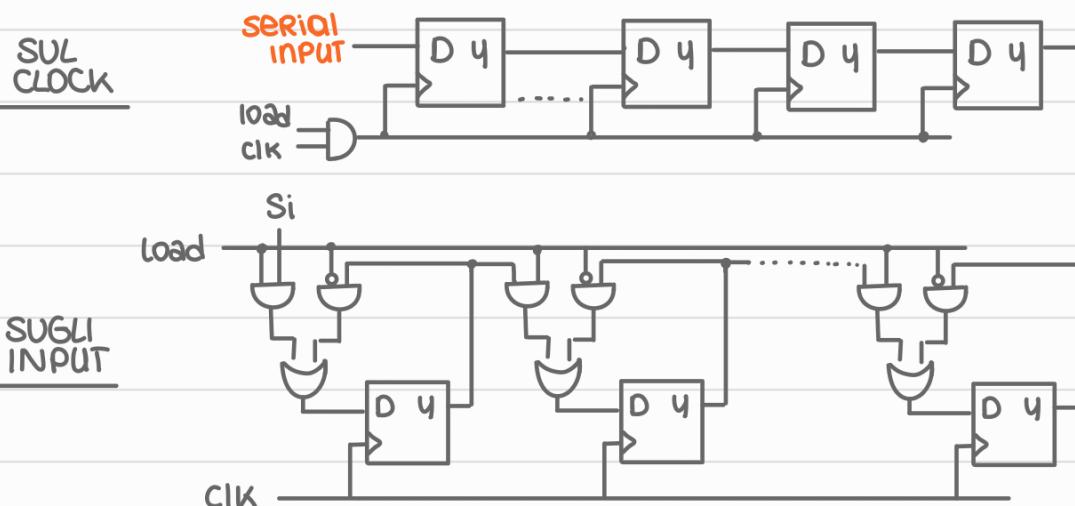
→ sul CLOCK



# SERIALE

I registri a caricamento seriale (o *shift registers*) ricevono una sola linea di input, il **serial input**, che viene progressivamente memorizzata nei vari Flip flop. Tale registro è composto logicamente da una serie di ff connessi in cascata, in cui l'uscita di ogni flip flop è connessa all'entrata del successivo.

Anche in questi registri viene utilizzato il segnale di **load** per assicurare che lo scorrimento dell'input si fermi, interrompendo il caricamento, e inizi la fase di memorizzazione. (utilizzato per scegliere se scorrere o memorizzare). Se load vale 1 i bit memorizzati sul registro shiftano, se è 0 rimangono inalterati.



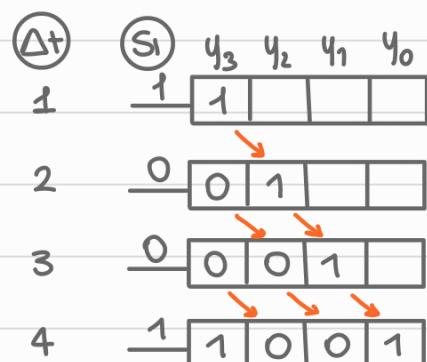
i valori caricati vengono riproposti in sequenza dall'unica uscita del registro.

## Come avviene il caricamento?

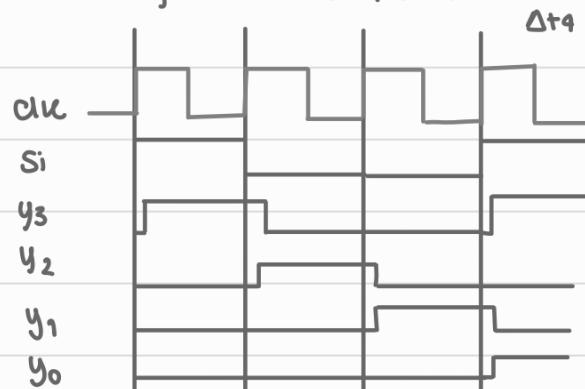
per caricare  $n$  bit in  $n$  ff il load deve essere impostato ad 1 e ad ogni fronte d'onda del clock deve arrivare il nuovo valore da memorizzare, il quale ad ogni nuovo ciclo viene propagato al flip flop successivo. Caricati tutti i bit il load torna a 0.

## esempio

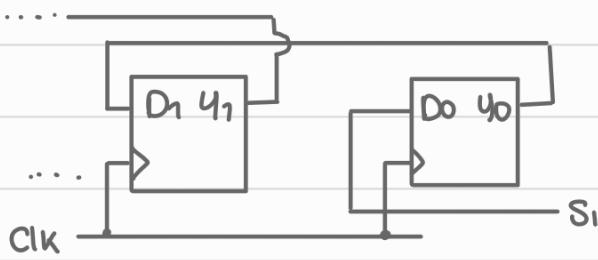
per caricare 1001 con scorrimento a destra:  
inizio dall'ultima cifra



## diagramma temporale

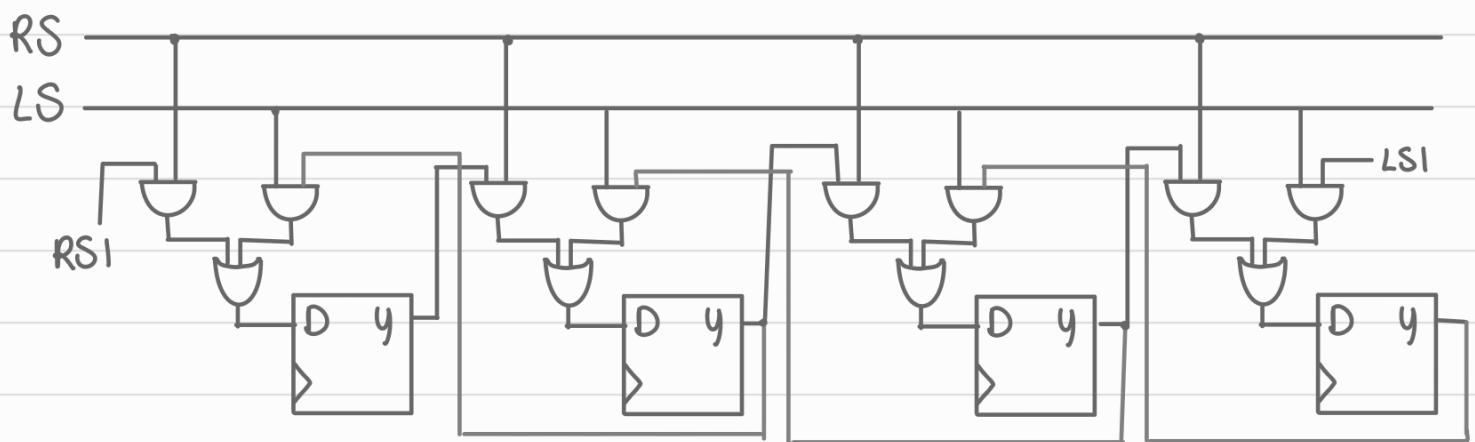


## scorrimento a sinistra



## REGISTRO BIDIMENSIONALE

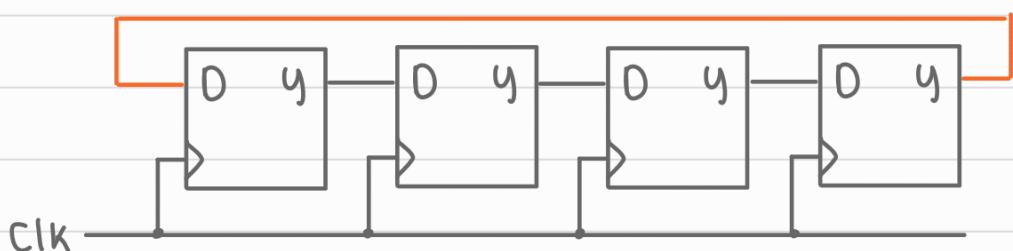
Nei registri bidimensionali vi sono due possibili direzione di scorrimento, destra o sinistra, e due input (right serial input e left serial input). Inoltre i segnali di controllo sono 2 (right shift e left shift) e non possono valere 1 contemporaneamente. Quando RS vale 1 si verifica (abilita) lo scorrimento a destra, mentre se SR vale 1 si verifica lo scorrimento a sinistra.



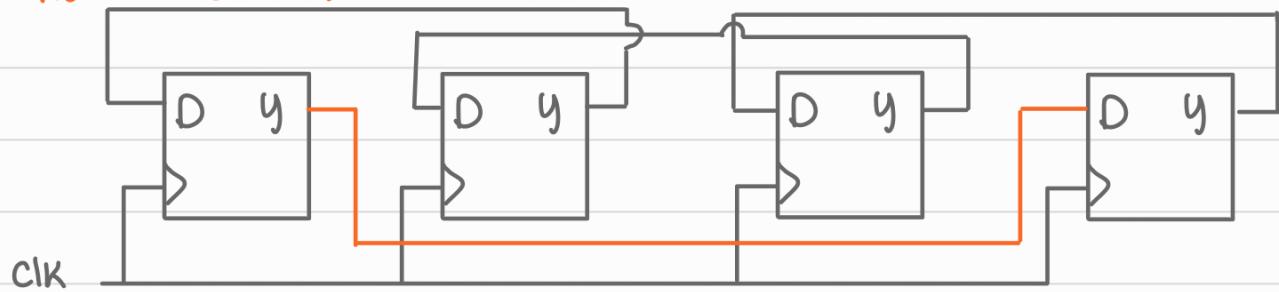
## ROTAZIONE

Nei registri a rotazione (circolari) l'uscita dell'ultimo flip flop è collegata all'entrata del primo e non è presente il serial input. In tal modo l'ultimo bit uscente non viene perso. La rotazione viene utilizzata per accedere ad insiemi di bit che sono in una determinata posizione senza, però, perderne il contenuto.

## ROTAZIONE A DESTRA



## ROTAZIONE A SINISTRA

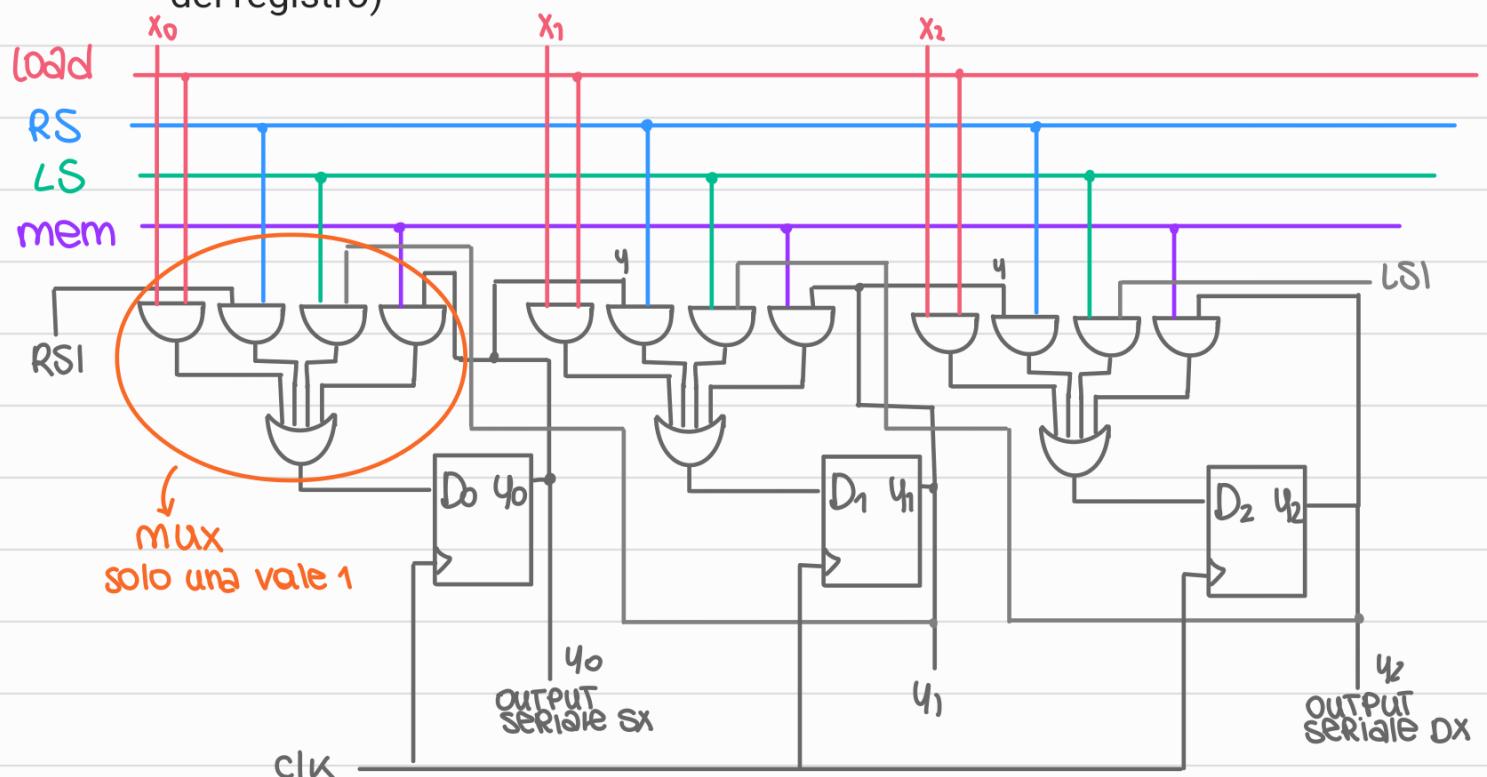


## REGISTRO UNIVERSALE

Nel registro universali vengono messi assieme il caricamento parallelo, quello seriale, sia a destra che a sinistra, e la rotazione a destra e sinistra.

INPUT:

- caricamento parallelo: **load**
- scorrimento a destra: **RS (right shift)**
- scorrimento a sinistra: **LS (left shift)**
- memorizzazione: **mem** (utilizzato per mantenere i valori caricati all'interno del registro)



# REGISTRI CONTATORE

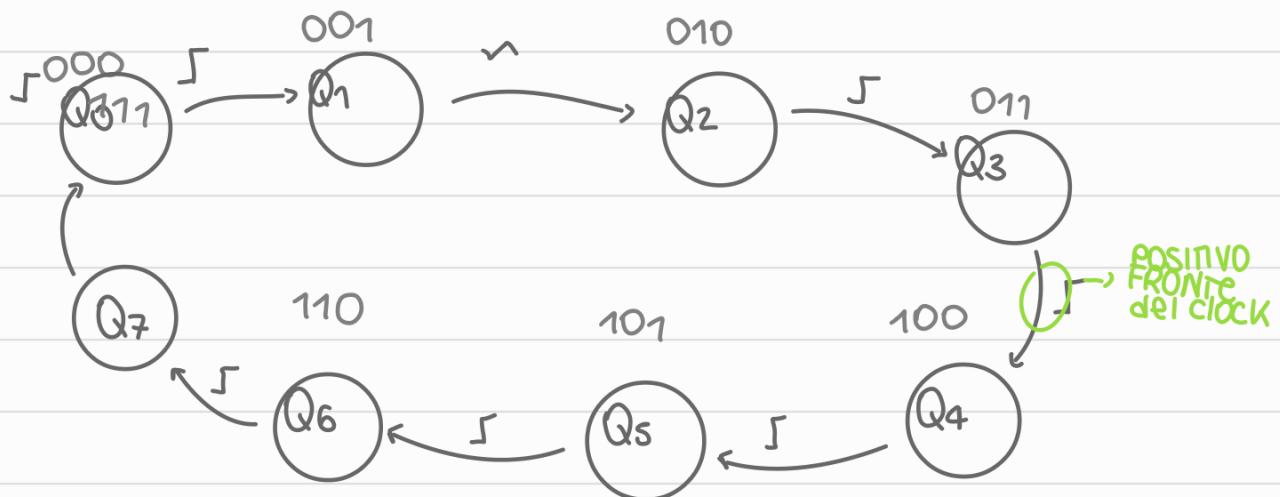
Un contatore è un registro utilizzato per contare il numero di occorrenze di un determinato evento, sempre modulo di un numero naturale.

Un modulo  $2^n$  è composta da n Flip flop.

## Contatore modulo 8

Il contatore mod 8, che è un caso particolare dei contatori mod  $2^n$ , conta da 0 a 7. Partendo da 0 ad ogni fronte d'onda ascendente del clock incrementa di 1 il suo valore fino ad arrivare a 7 e poi ricomincia.

- AUTOMA



- Codifica dell'automa

Codifica binaria stato : 3 bit  $\rightarrow$  valore numerico binario

- Sintesi :  $8 = 2^3 \rightarrow 3$  FlipFlop

$y_2$	$y_1$	$y_0$	$y_2$	$y_1$	$y_0$	$j_2$	$k_2$	$j_1$	$k_1$	$j_0$	$k_0$
0	0	0	0	0	1	0	8	0	8	1	8
0	0	1	0	1	0	0	8	1	8	8	1
0	1	0	0	1	1	0	8	8	0	1	8
0	1	1	1	0	0	1	8	8	1	8	1
1	0	0	1	0	1	8	0	0	8	1	8
1	0	1	1	1	0	8	0	1	8	8	1
1	1	0	1	1	1	8	0	8	0	1	8
1	1	1	0	0	0	8	1	8	1	8	1

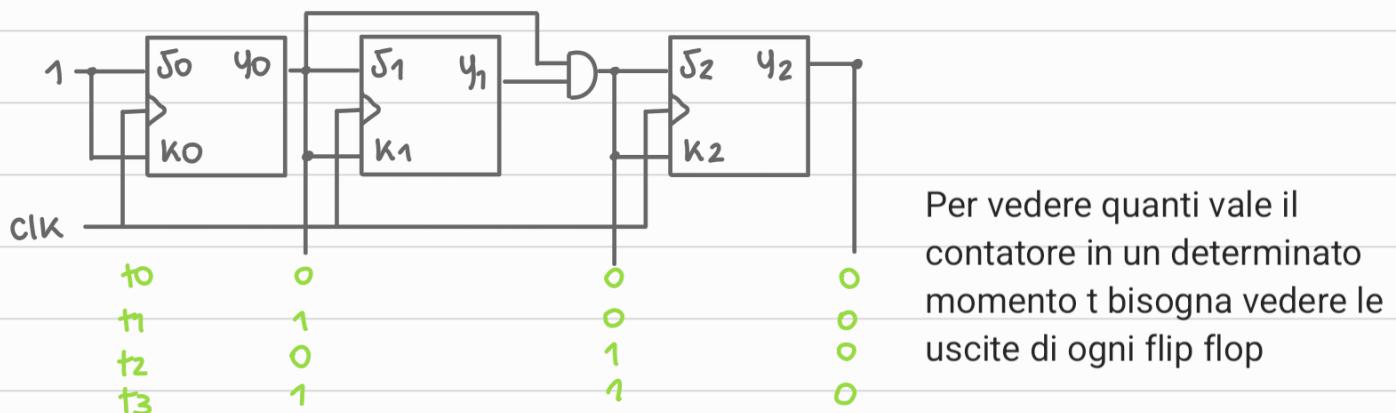
$$j_0 = k_0 = 1$$

$$j_1 = k_1 = y_0$$

$$j_2 = k_2 = y_1 y_0$$

$j_2$ K-mappa		00	01	11	10
0	0	0	1	0	0
1	8	8	8	8	8

## • CIRCUITO

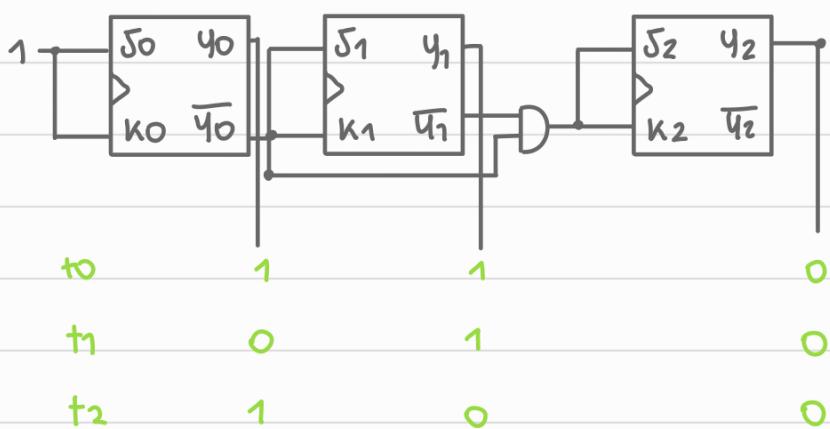


Per aggiungere cifre (aggiungere un flip flop), rimanendo nei contatori di tipo  $2^n$ , è sufficiente aggiungere un flip flop finale le cui entrate sono il risultato dell'and tra tutte le uscite precedenti (in tale modo l'ultima cifra viene complementata solo se tutte le uscite precedenti sono pari ad 1).

## Contatore alla rovescia

Anche chiamato down counter, conta i numeri da  $2^n$  a 0. Viene realizzato facendo entrare nei flip flop le uscite complementate del precedente e non quelle vere. Tuttavia, per visualizzare il conteggio bisogna sempre far riferimento alle uscite non complementate).

*Prendendo come riferimento sempre un contatore mod 8*

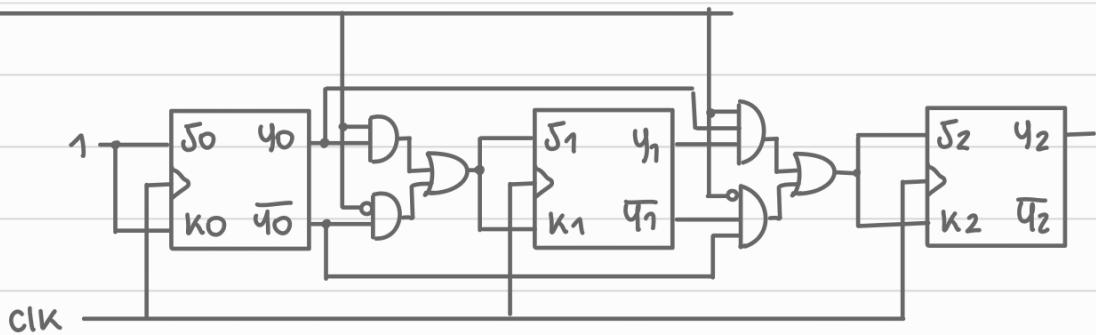


## Contatore avanti e indietro

E' un contatore nella quale è possibile determinare la direzione del conteggio (per l'appunto, o avanti o indietro), attraverso un segnale di controllo **dir**. Se dir vale 1 il conteggio va in avanti, se al contrario vale 0 il conteggio è alla rovescia.

DIR { 1 avanti  
0 indietro

DIR



### Contatore con modulo n (diverso da $2^n$ )

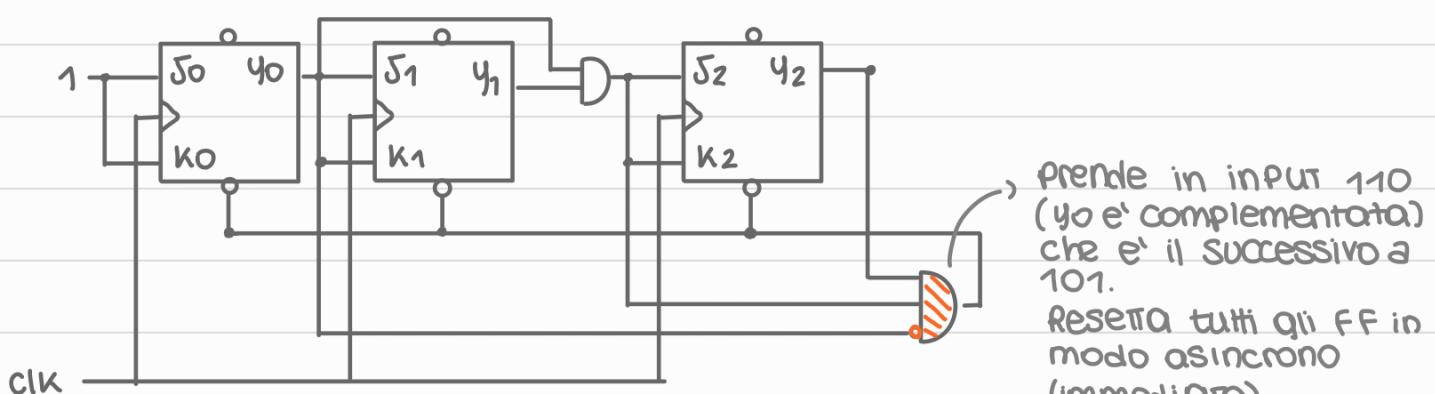
Per realizzare un contatore che conti fino a un numero n diverso da una potenza di 2, vi sono due strade percorribili:

- creare un circuito ad hoc, seguendo il procedimento di sintesi
- **contatore preselezionabile**, che permette di scegliere il valore di inizio e di fine del contatore (soluzione modulare). In questo caso vengono utilizzati **dei flip flop con ingressi asincroni**, ovvero che non dipendono dal fronte del clock. Il segnale asincrono è dato da una porta And che prende in input la sequenza successiva all'ultima presa in considerazione (complementando gli 0 della sequenza e lasciando invariati gli 1, in modo tale che l'and equivale ad 1 solo in quella specifica sequenza, abilitando così il segnale di asincrono di CLEAR che resetta tutti i ff).

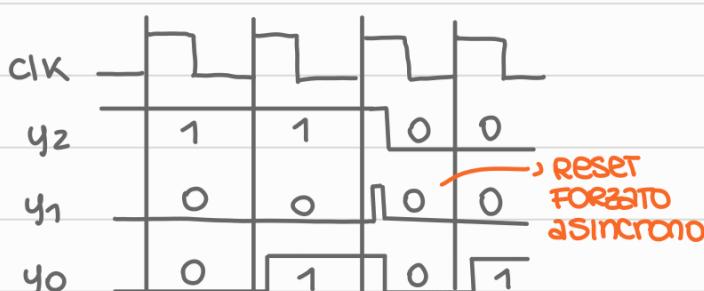
### ESEMPIO

### Contatore mod 6

Conta i numeri da 0 a 5, che in codifica binaria equivalgono ai numeri da 000 a 101 (a differenza del mod 8 non si prendono in considerazioni i numeri da 110 a 111).



### diagramma temporale



# FULL ADDER sequenziale

Nel Ripple carry adder tradizionale (combinatorio) per ottenere il risultato bisogna aspettare che ogni full adder abbia prodotto il risultato poiché l'fa successivo deve considerare anche il riporto in uscita del precedente. Perciò l'attesa diventa di  $n$  tempi se ci sono  $n$  full adder.

Un'altra modalità di progettazione consiste nel realizzare un full adder sequenziale con un flip flop che memorizza il riporto:

- input: a,b
- output: somma
- il riporto viene memorizzato (dallo stato)



## Automa

$S_0 = \text{RIPORTO } 0$

$S_1 = \text{RIPORTO } 1$

	a <sub>i</sub>	b <sub>i</sub>				
$S_0$	00	01	10	11		
	$S_0/0$	$S_0/1$	$S_0/1$	$S_1/0$		
$S_1$	$S_0/1$	$S_1/0$	$S_1/0$	$S_1/1$		

## TAVOLA STATI FUTURI

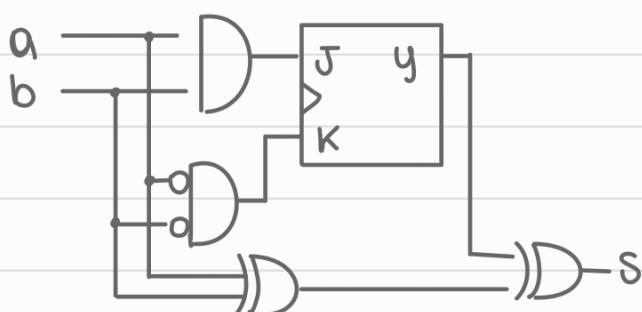
a	b	y	$\gamma$	S	J	K
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	1	1	0	0	0
1	1	0	1	0	1	0
1	1	1	1	1	0	0

$$K = \bar{a}\bar{b}$$

$$J = ab$$

$$S = a \oplus b \oplus y$$

CIRCUITO :



Il comparatore aritmetico confronta la magnitudo tra due numeri A e B. Per verificare se  $A < B$ , si controlla che A non è né maggiore né uguale a B, utilizzando una porta AND con gli ingressi negati. Per controllare l'uguaglianza si utilizza un comparatore logico. Mentre per verificare che  $A > B$  si utilizza un adder: infatti,  $A > B$  può essere riscritto come  $A - B < 0$  e quindi facendo la differenza tra i due numeri se il risultato è negativo (MSB vale 1) A è minore, altrimenti è maggiore.