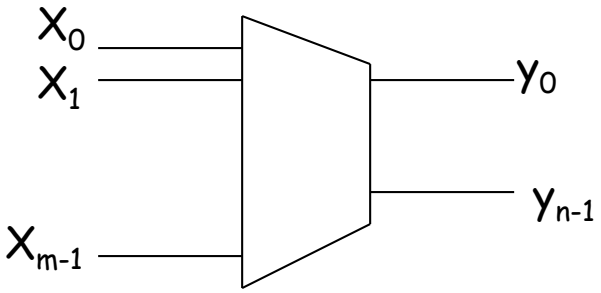
Reti combinatorie: moduli di base

Codificatore

- Realizza la funzione di **codifica binaria**, ossia associare ad ogni elemento di un insieme Γ composto da **m simboli**, una sequenza distinta di n bit
- Per ogni simbolo tale circuito genera il codice corrispondente

$$2^n \geq m$$

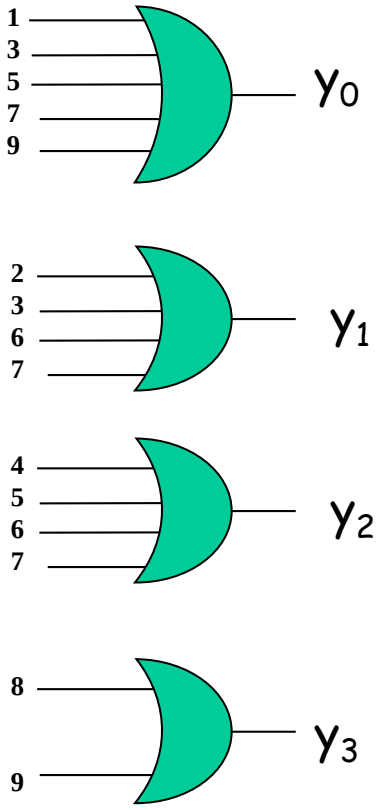
- m linee di ingresso $x_0,..,x_{m-1}$, n linea di uscita $y_0,..,y_{n-1}$
  - La linea x_i è associata al simbolo i-simo
  - Quando $x_i=1$, e $x_j=0$ ($j \neq i$), in uscita è presente il codice corrispondente al simbolo i-simo



Esempio

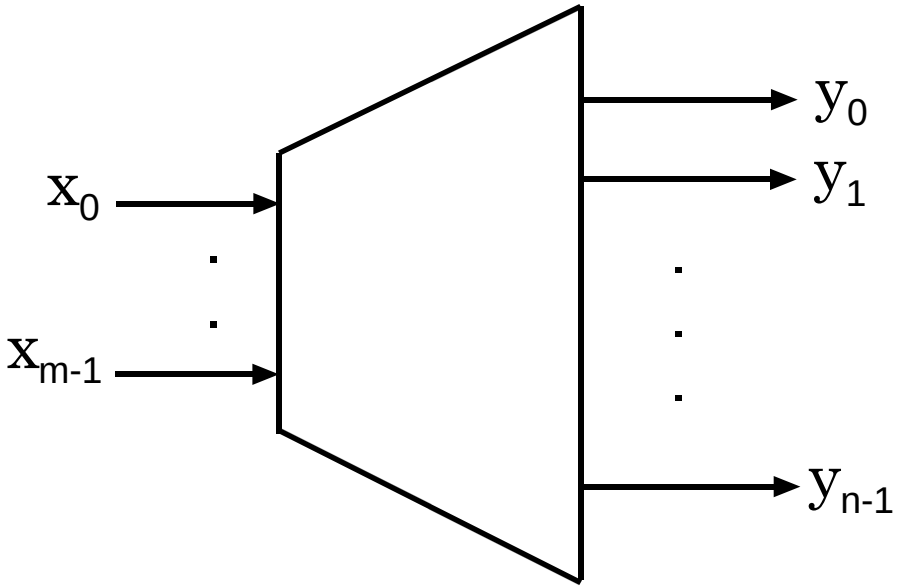
- Codifica cifre decimali in BCD

	$Y_3Y_2Y_1Y_0$
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001



Decodificatore

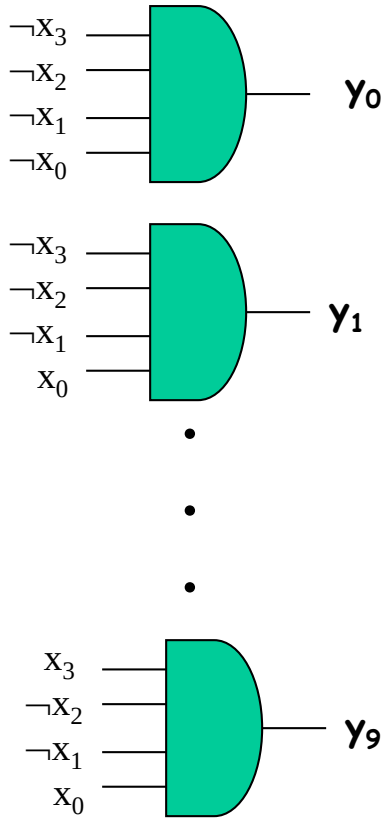
- Realizza la funzione inversa del codificatore, a partire da una parola di un codice in binario genera una uscita che identifica uno dei simboli dell'insieme Γ.
- Per ogni configurazione di ingresso, una sola uscita vale 1, le altre hanno valore 0



Esempio

- Decoder BCD-Cifre decimali (prima realizzazione)

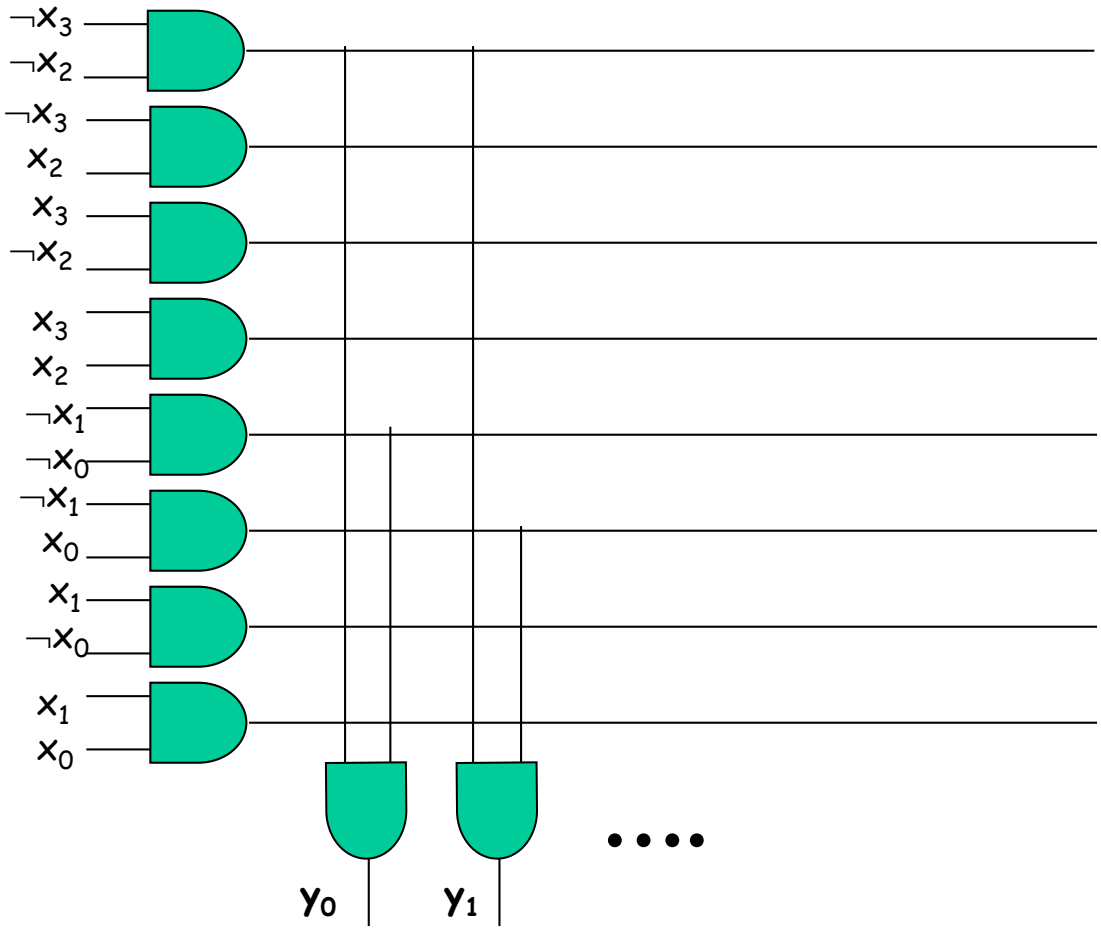
$x_3x_2x_1x_0$	$y_9y_8y_7y_6y_6y_5y_4y_3y_2y_1y_0$
0000	0 0 0 0 0 0 0 0 0 1
0001	0 0 0 0 0 0 0 0 1 0
0010	0 0 0 0 0 0 0 1 0 0
0011	0 0 0 0 0 0 1 0 0 0
0100	0 0 0 0 0 1 0 0 0 0
0101	0 0 0 0 1 0 0 0 0 0
0110	0 0 0 1 0 0 0 0 0 0
0111	0 0 1 0 0 0 0 0 0 0
1000	0 1 0 0 0 0 0 0 0 0
1001	1 0 0 0 0 0 0 0 0 0



Esempio

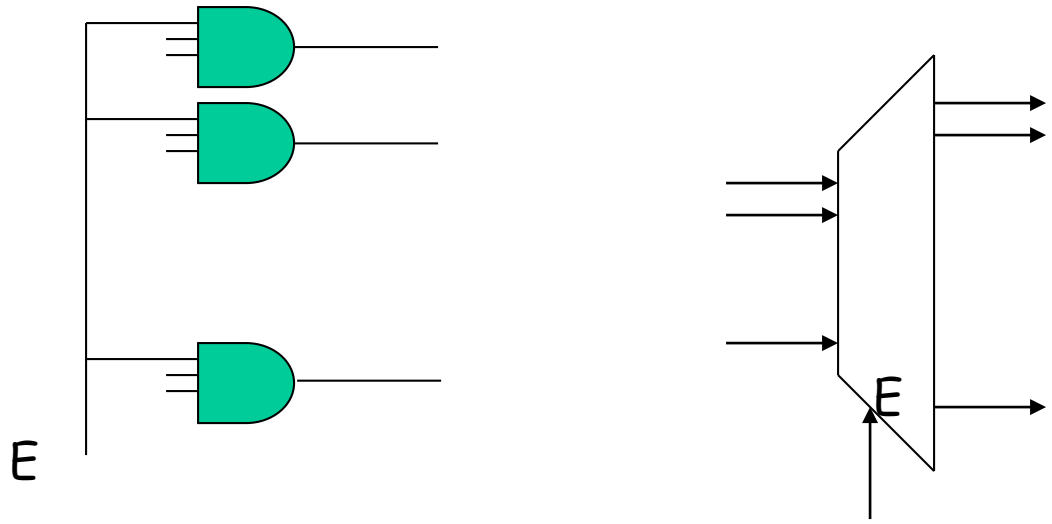
- Decoder BCD-Cifre decimali (seconda realizzazione)

$x_3x_2x_1x_0$	$y_9y_8y_7y_6y_6y_5y_4y_3y_2y_1y_0$
0000	0 0 0 0 0 0 0 0 0 1
0001	0 0 0 0 0 0 0 0 1 0
0010	0 0 0 0 0 0 0 1 0 0
0011	0 0 0 0 0 0 1 0 0 0
0100	0 0 0 0 0 1 0 0 0 0
0101	0 0 0 0 1 0 0 0 0 0
0110	0 0 0 1 0 0 0 0 0 0
0111	0 0 1 0 0 0 0 0 0 0
1000	0 1 0 0 0 0 0 0 0 0
1001	1 0 0 0 0 0 0 0 0 0



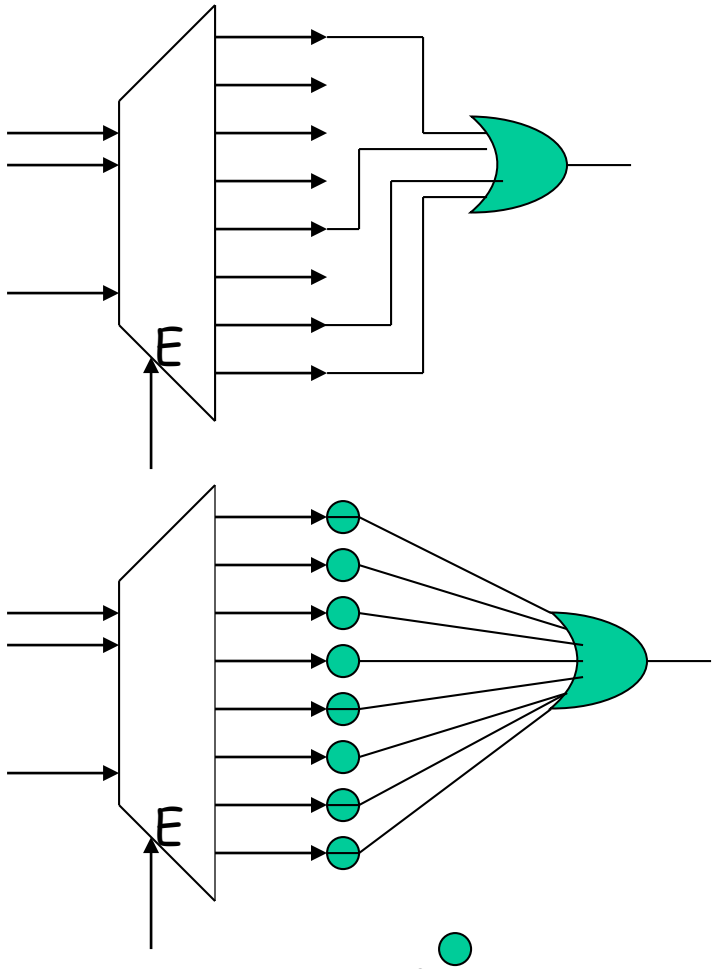
Decodificatore con enable

- E' dotato di un ulteriore ingresso di abilitazione E (detto anche *strobe*)
- Il decodificatore è abilitato (ossia il processo di decodifica ha luogo) solo quando E=1

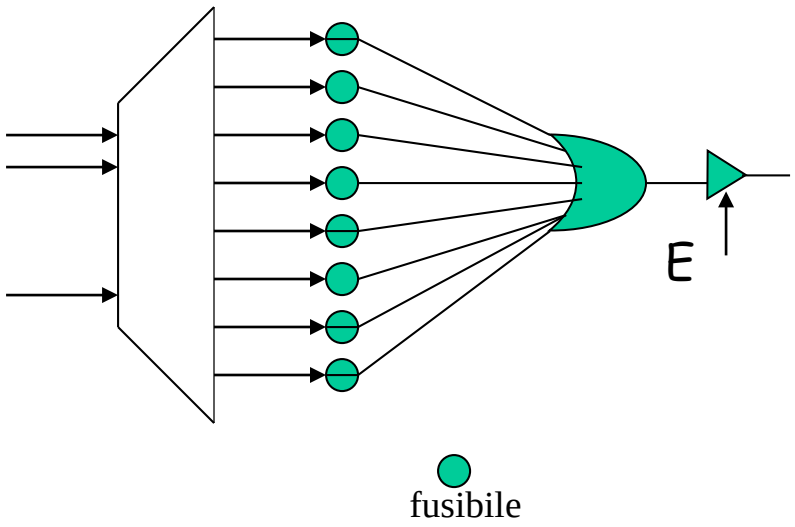


Realizzazione di funzioni tramite decoder

$x_2x_1x_0$	f
000	1
001	0
010	0
011	0
100	1
101	0
110	1
111	1

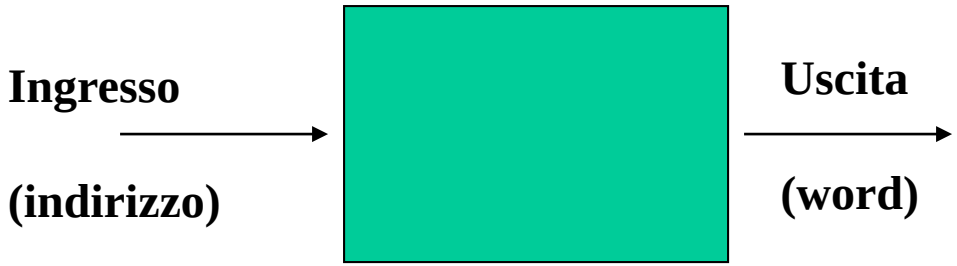


Realizzazione di funzioni tramite decoder (con Enable tree-state)



ROM (Read Only Memory)

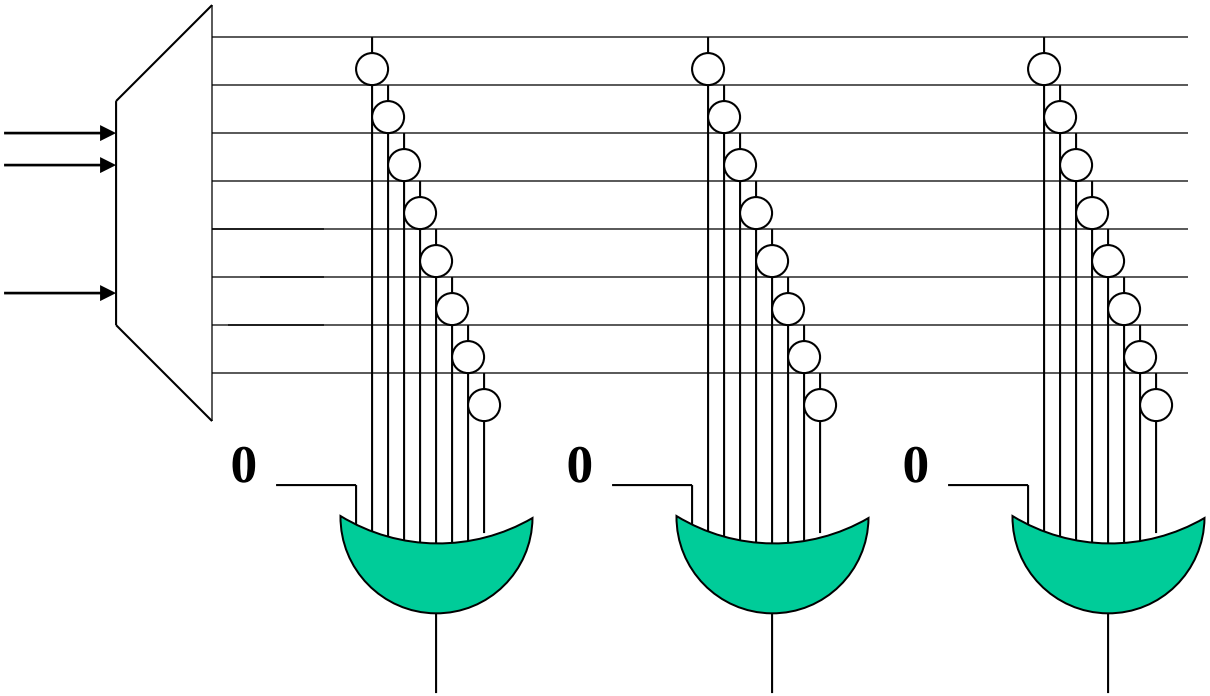
- Insieme di locazioni di memoria che possono essere lette specificandone l'indirizzo



- Una ROM è un **circuito combinatorio**
  (dato un ingresso c'è una sola uscita)

Schema logico di una ROM

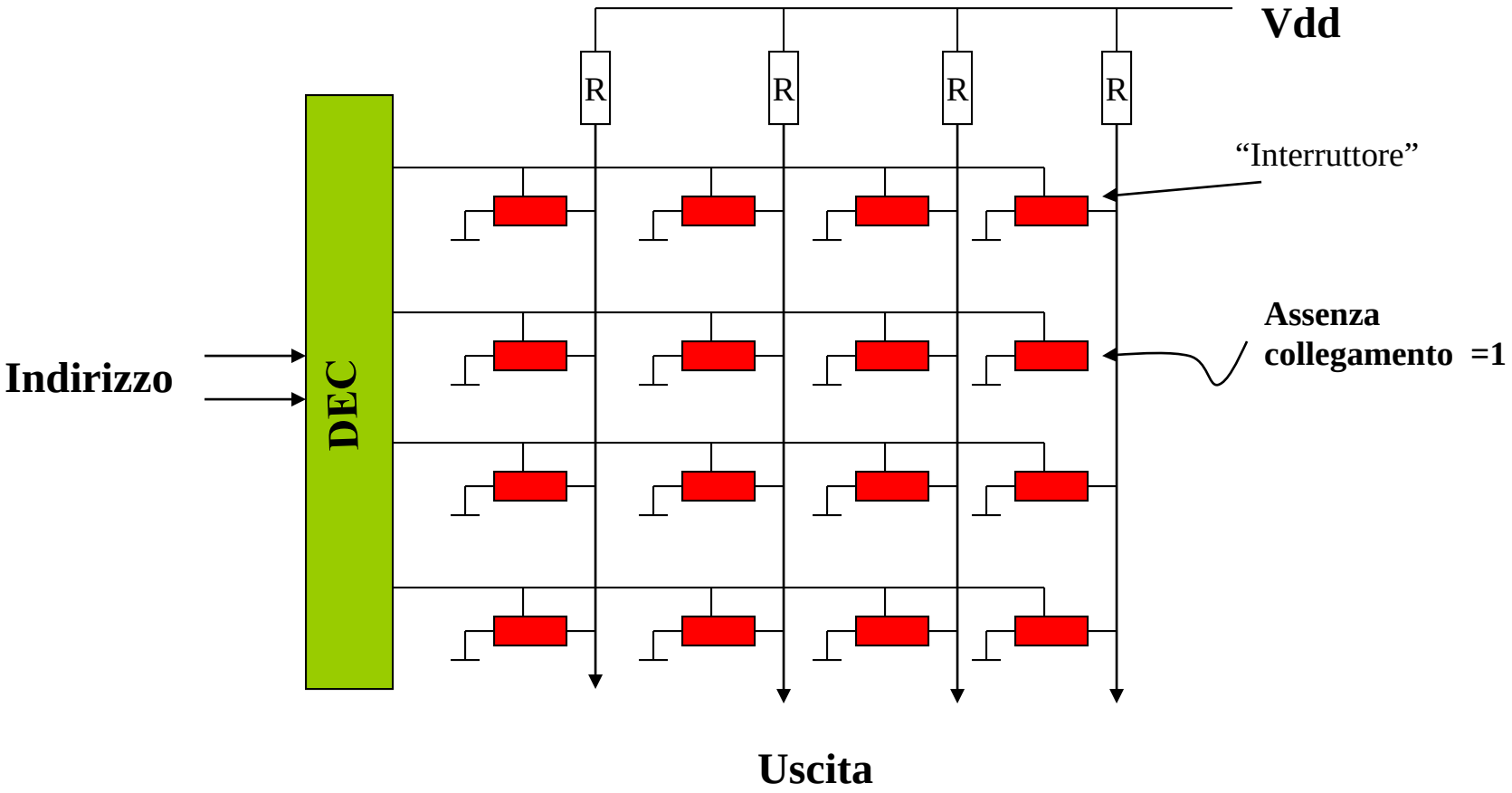
Funzioni di commutazioni realizzate come OR di mintermini



- **fusibile**

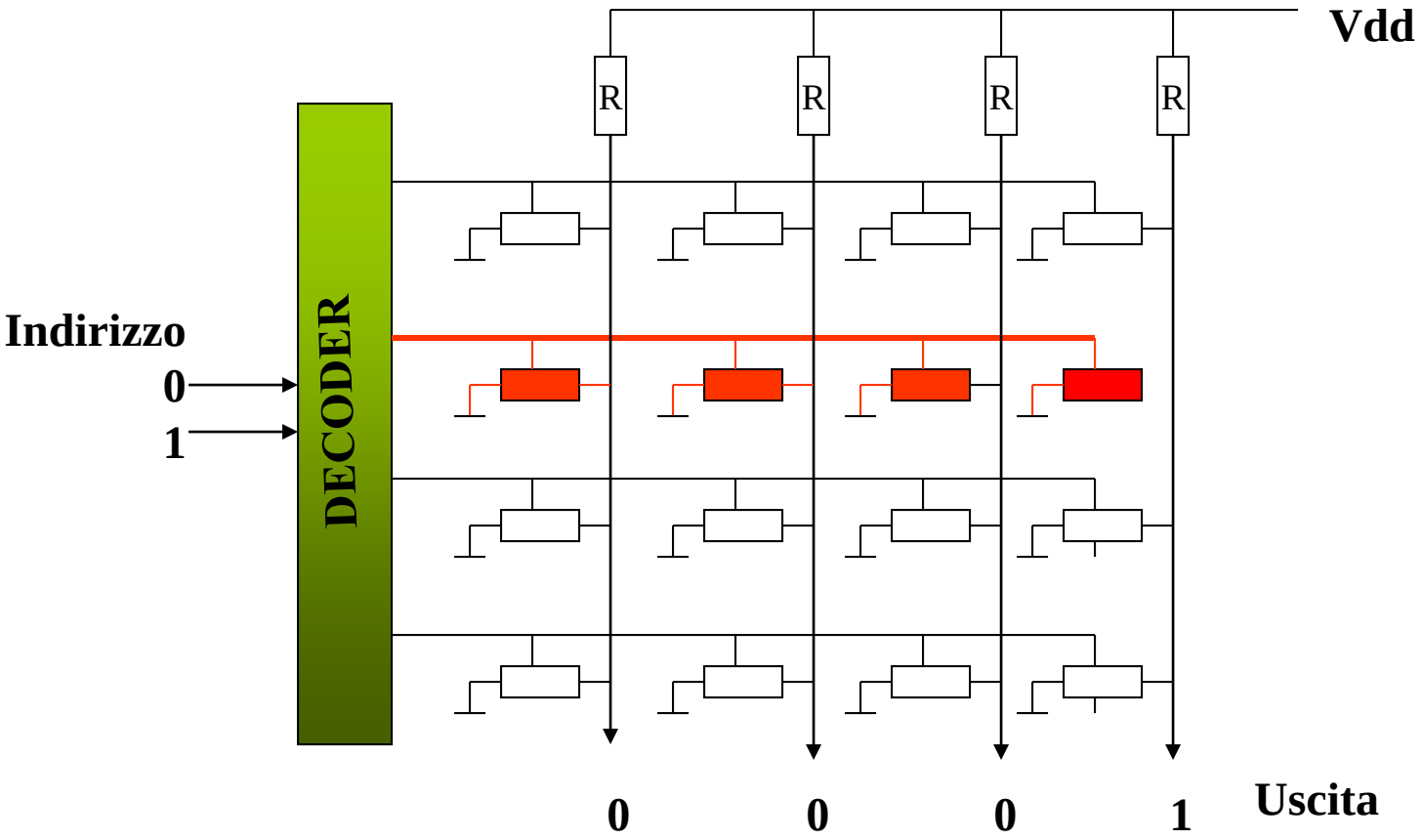
Implementazione ROM con C-MOS

- ROM 4x4 (numero parole x dimensione parola)

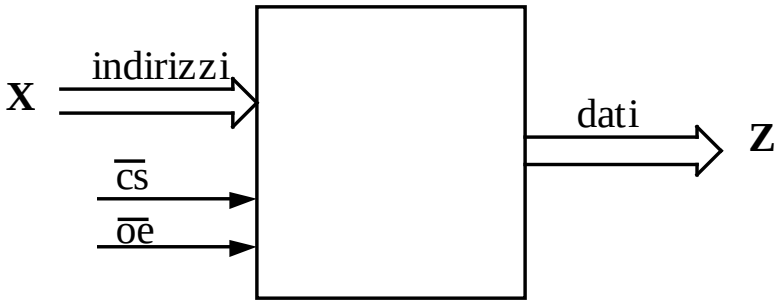


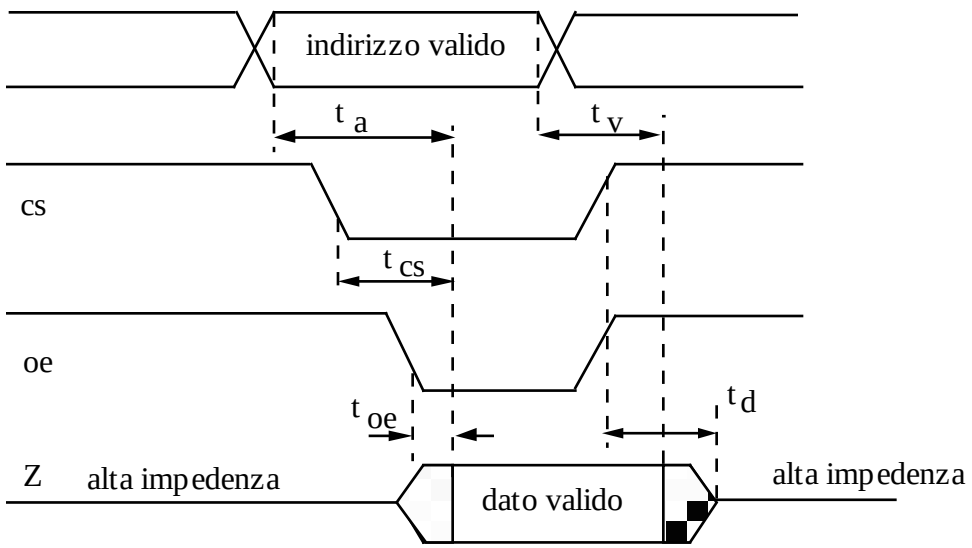
Implementazione ROM (2)

- Esempio, indirizzo 01, uscita=0001



ROM temporizzazioni

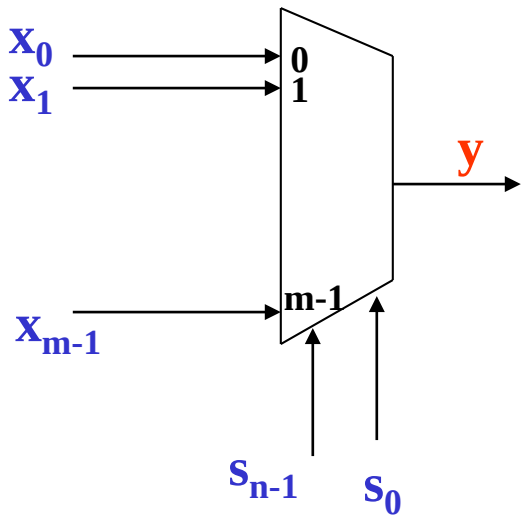




- t_a : tempo di propagazione dall'ingresso X all'uscita Z
- t_{cs}: tempo di propagazione dall'ingresso cs all'uscita Z
- t_{oe}: tempo di propagazione dall'ingresso oe all'uscita Z
- t_v: tempo di mantenimento dell'uscita da quando commuta X o cs o oe
- t_d: tempo di disabilitazione dell'uscita da quando commuta cs o oe

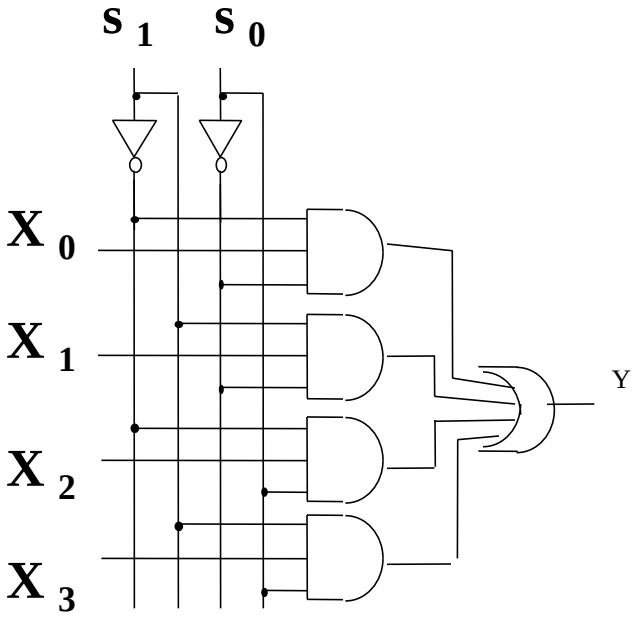
Multiplexer (MUX 2^n:1)

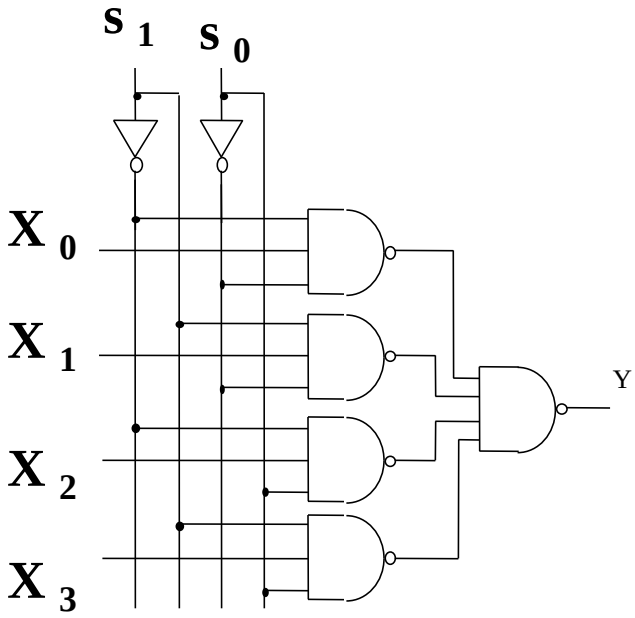
- Ingressi
  - $m=2^n$ ingressi dati
  - n ingressi di selezione (controllo)
- Uscita
  - Una fra le m, a seconda del controllo



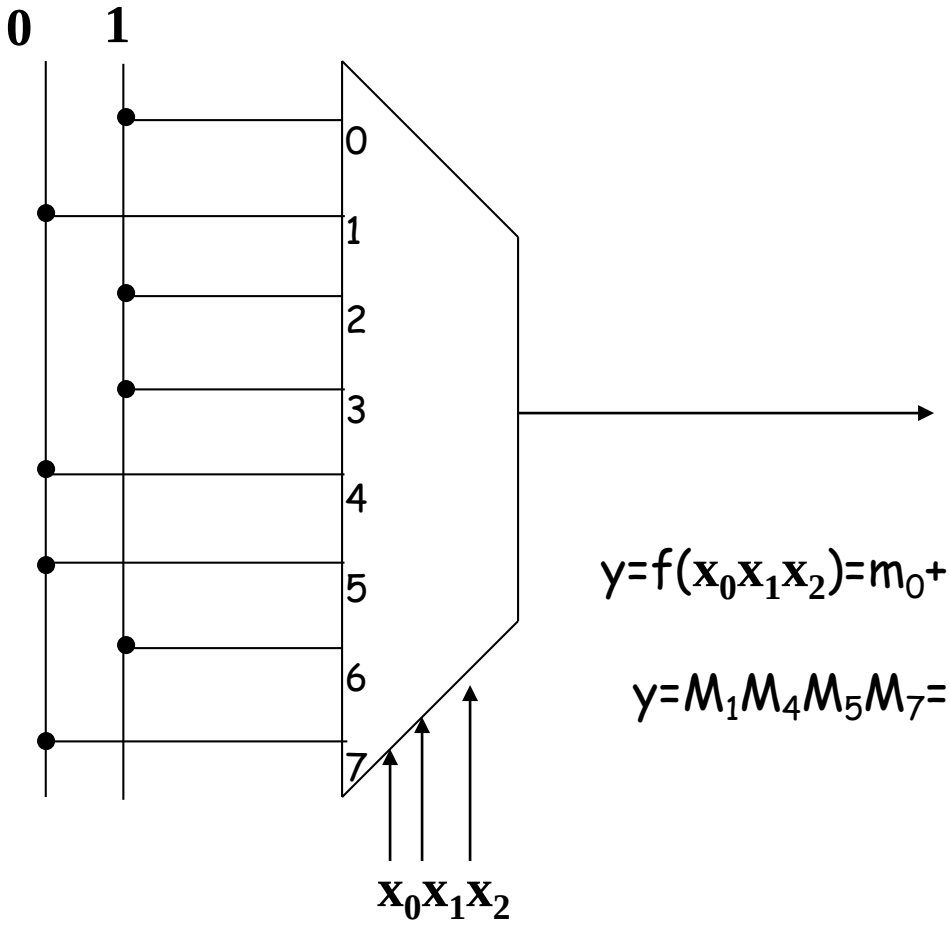
S	y
0	X_0
1	X_1
..	..
2^n-1	X_{2^n-1}

MUX 4-2





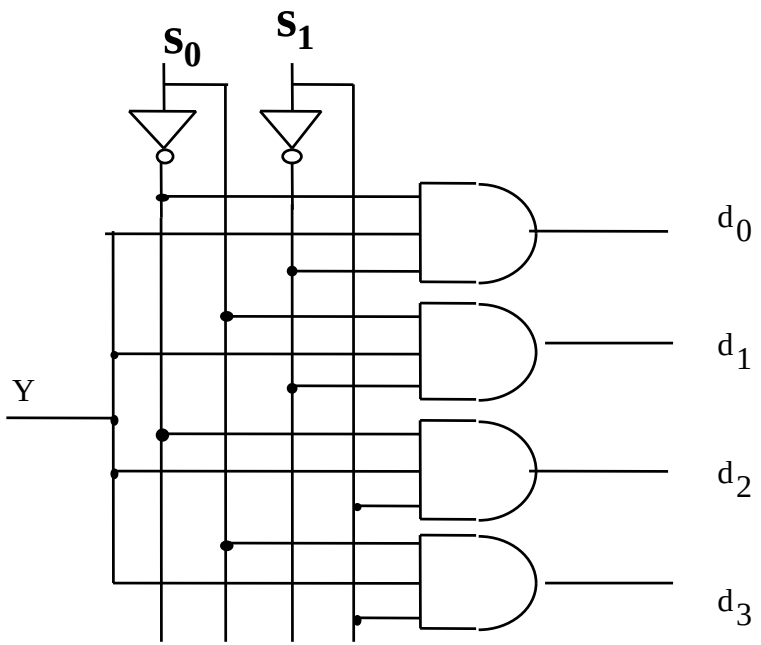
MUX - Generatore di funzioni



$y=f(x_0x_1x_2)=m_0+m_2+m_3+m_6=\Sigma(0,2,3,6)$

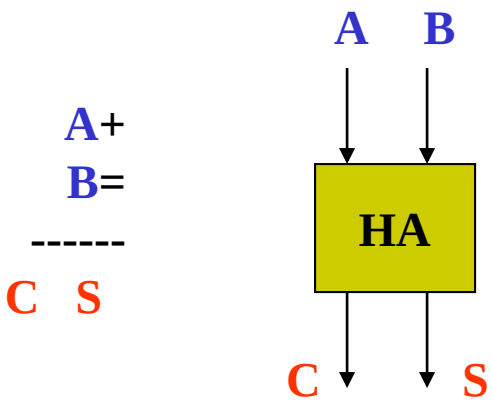
$y=M_1M_4M_5M_7=\Pi(1,4,5,7)$

DEMUX 2-4



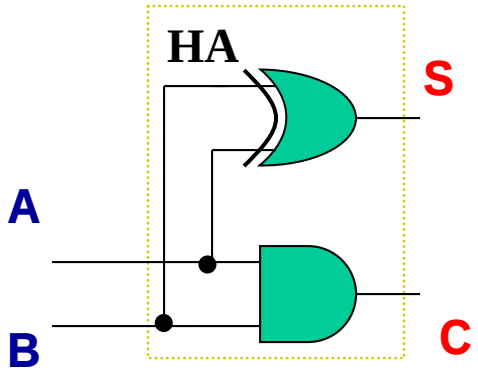
Half Adder - Semisommatore

- Ingresso 2 bit, uscita 2 bit



In	Out
A B	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

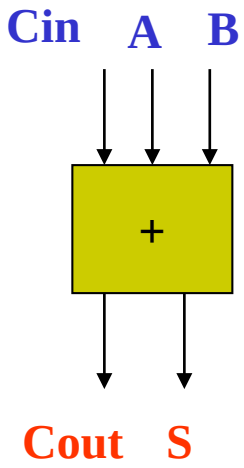
C=AB
S=(not A)B + A(not B)=A⊕B



Full Adder – Addizionatore completo

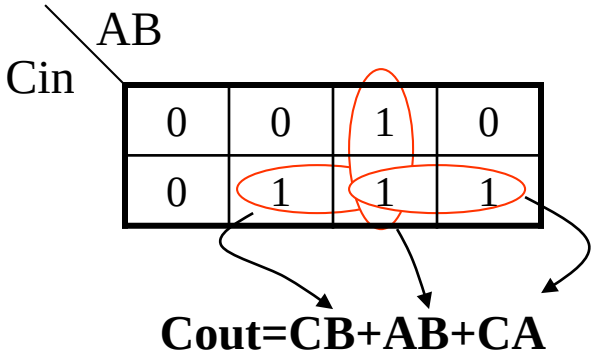
Cin+
A+
B=

Cout S

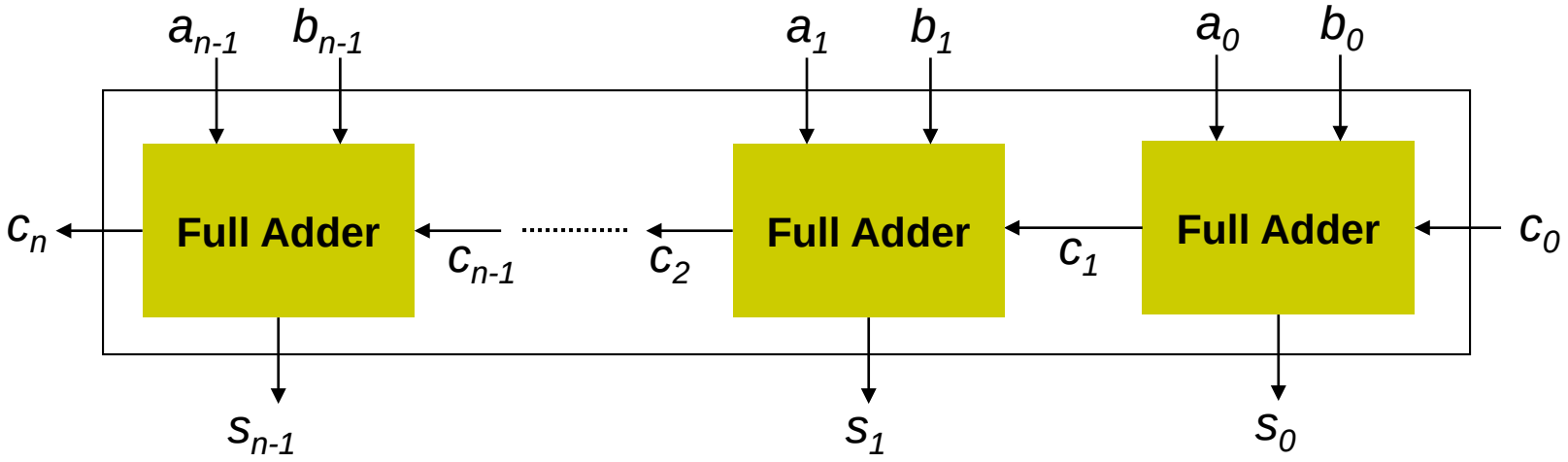


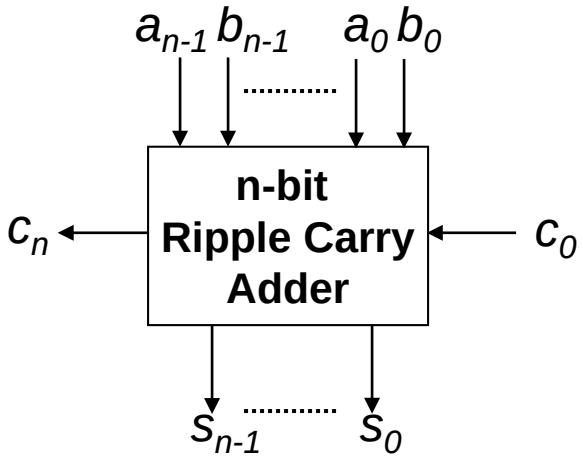
In			Out	
A	**B**	**Cin**	**Cout**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S vale 1 solo quando un numero dispari di bit di ingresso vale 1. Quindi,
S=A⊕B ⊕C



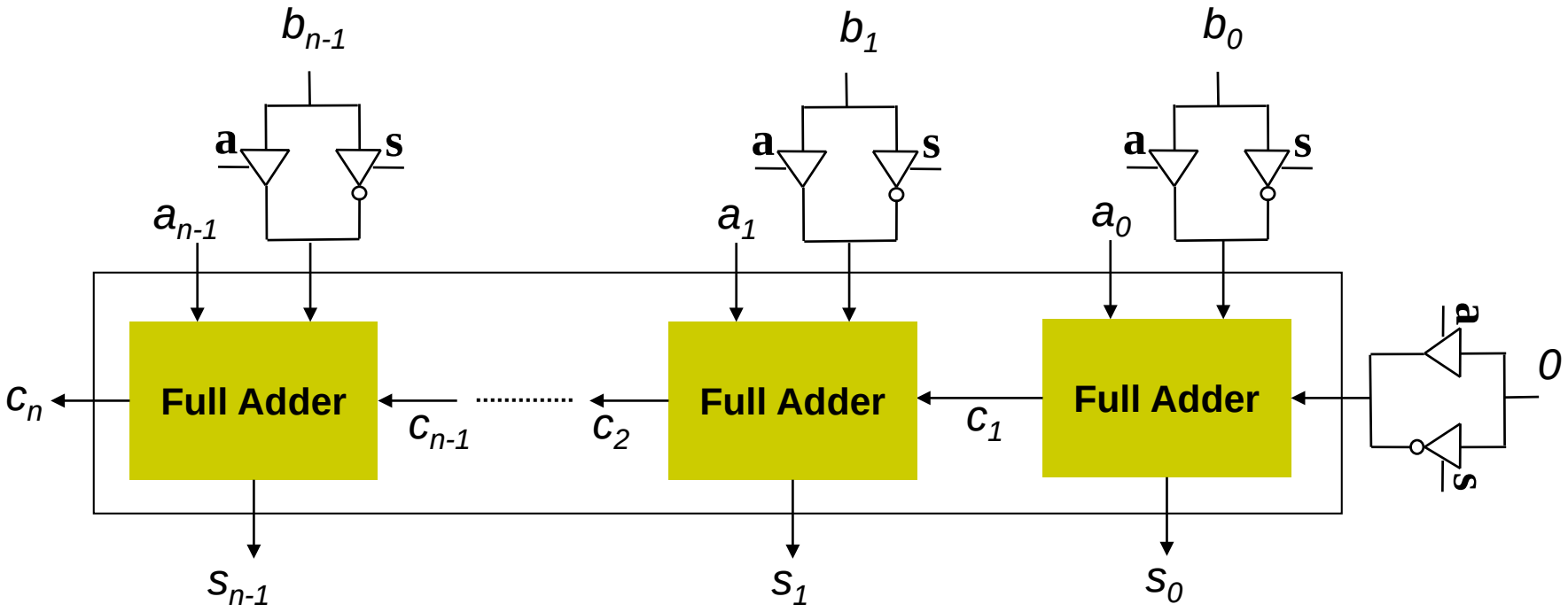
Ripple Carry Adder (RCA)



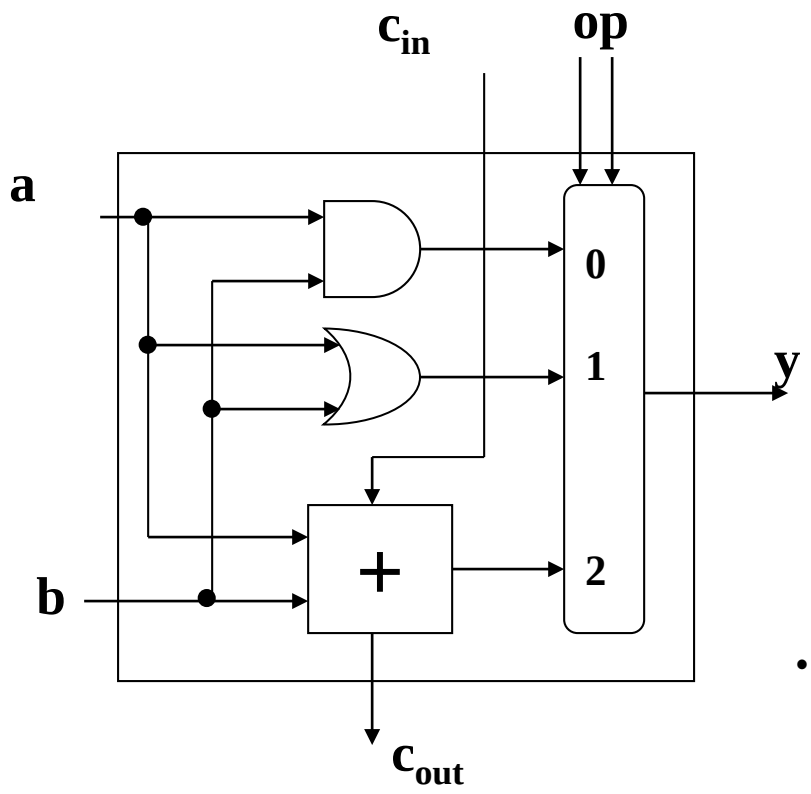


Il tempo per ottenere il risultato è pari ad nTc, dove Tc è il tempo di propagazione del riporto

Addizionatore/Sottrattore



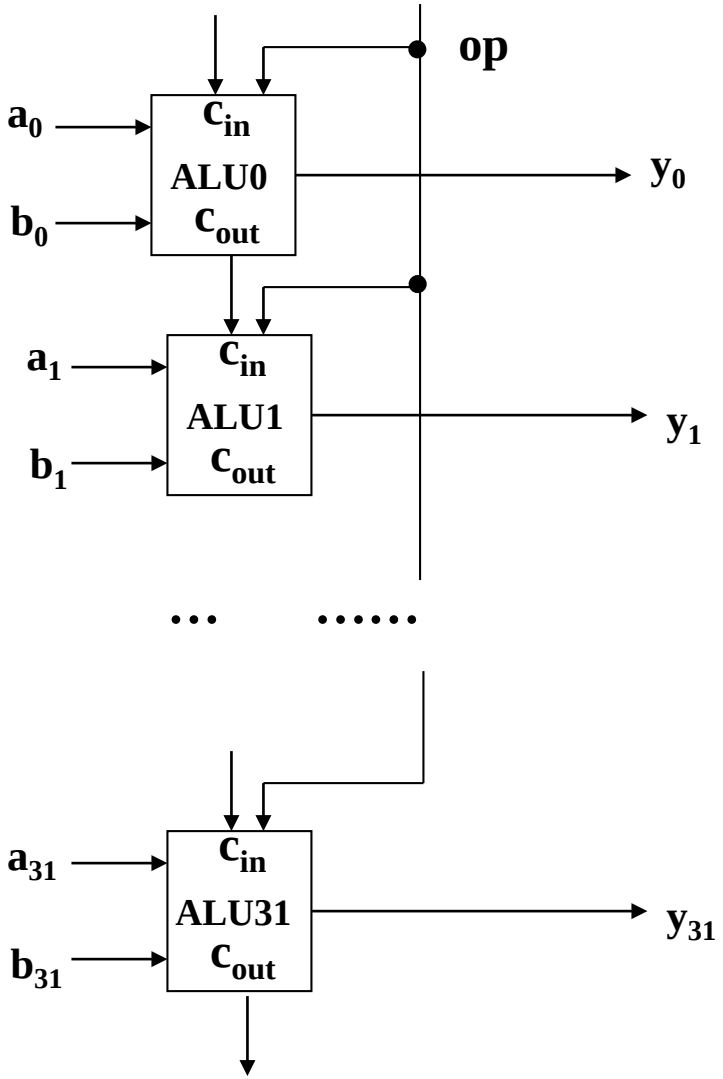
ALU (bit slice)



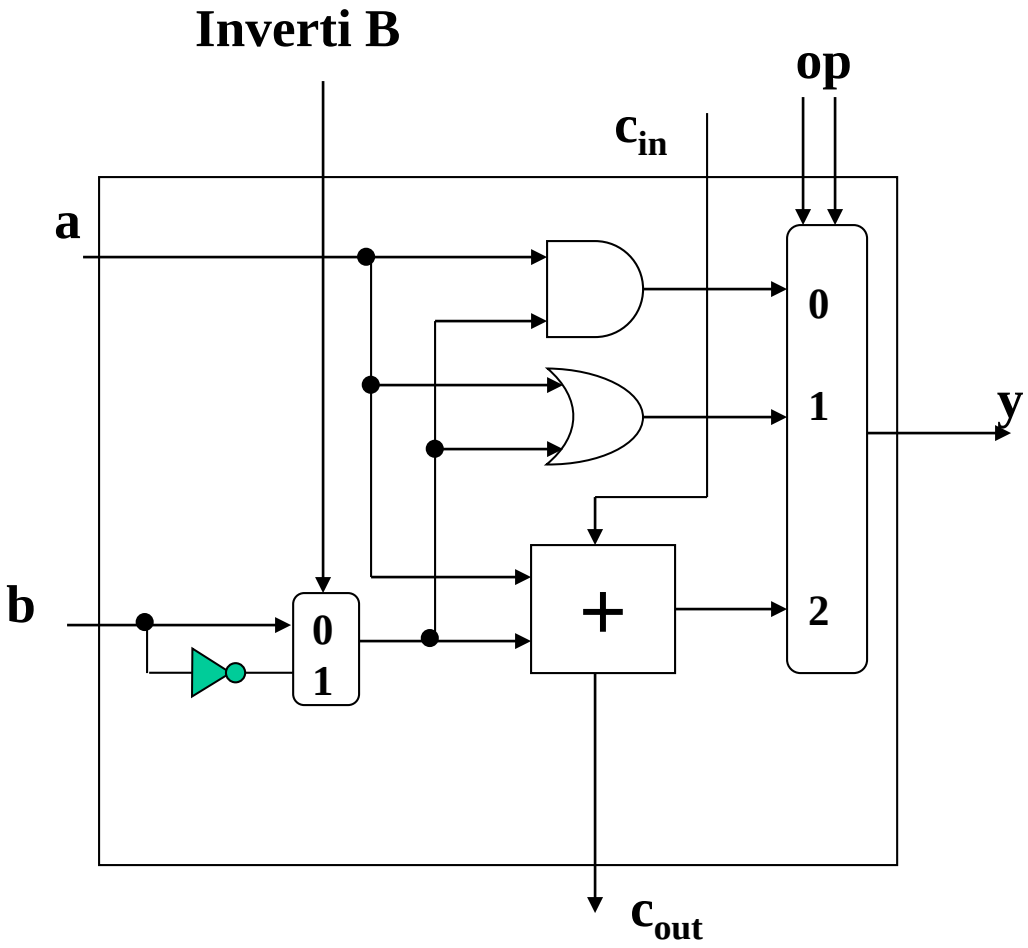
op	y
0 0	a AND b
0 1	a OR b
1 0	(a+b+cin) mod 2
11	??

- **op** seleziona il tipo di operazione (la configurazione 11 non è ammessa-prevista)

ALU a 32 bit (bit slice)



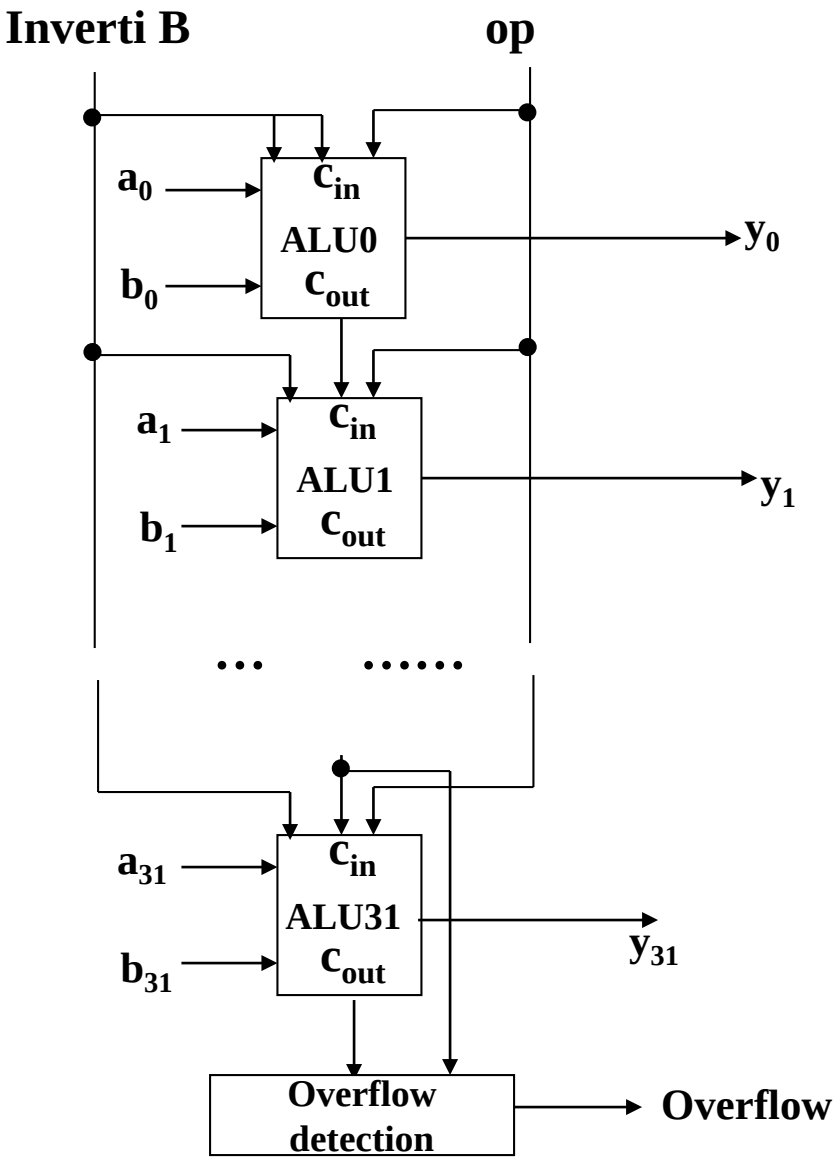
ALU (bit slice)



op	InvertiB	c_{in}	y
0 0	0	-	a AND b
0 0	1	-	A AND (NOT b)
0 1	0	-	a OR b
0 1	1	-	A OR (NOT b)
1 0	0	0	(a+b+cin) mod 2
1 0	1	1	(a-b)*

* = rappresentazioni in complemento a 2

ALU a 32 bit



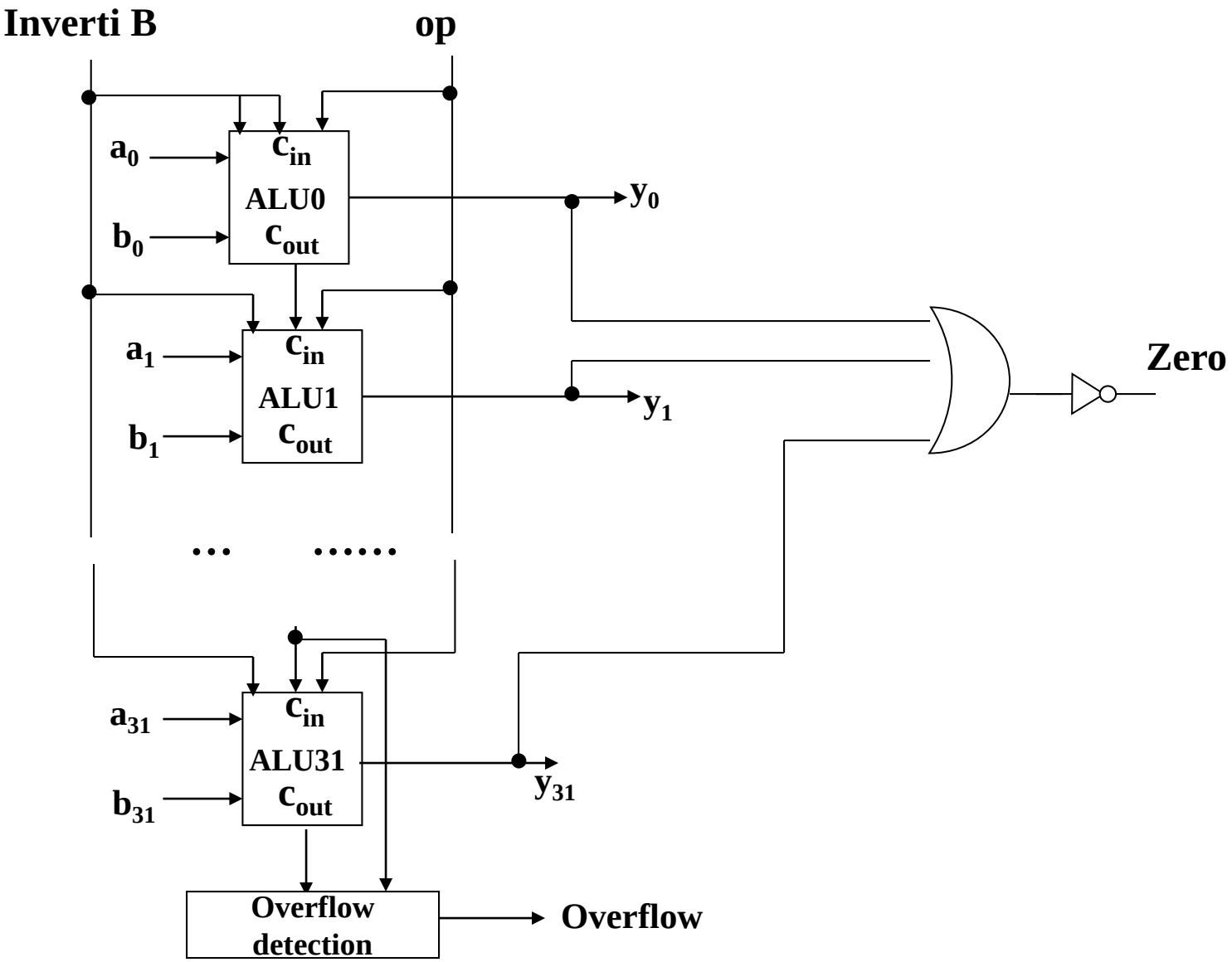
op	Inverti B	y
0 0	-	A AND B
0 1	-	A OR B
1 0	0	A + B
1 0	1	A-B

Per stabilire se si verifica overflow
È sufficiente confrontare se
in corrispondenza del MSB, $c_{in} \neq c_{out}$

Supporto ALU per i salti

- Vogliamo ampliare la ALU in modo che sia in grado di rilevare la condizione a=b
- Tale condizione è utile per far eseguire istruzioni in modo condizionato (jump)
- Indichiamo con Zero la variabile binaria cosi definita:
  - Zero=1 se e solo se a=b
- Per calcolare Zero osserviamo che a=b <-> a-b=0
  - Pertanto Zero=1 se e solo se tutti i bit dell'operazione a-b sono nulli. Ossia, Zero coincide col mintermine m_0 definito sulgli n bit r_0 … r_{n-1} che rappresentatno la differenza.
  - Zero=m_0= (not r_0)(not r_1)…(not r_{n-1})= not (r_0+r_1 .. +r_{n-1})

ALU a 32 bit


Inverti B
op
a_0
b_0
c_{in}
ALU0
c_{out}
y_0
a_1
b_1
c_{in}
ALU1
c_{out}
y_1
Zero
a_{31}
b_{31}
c_{in}
ALU31
c_{out}
y_{31}
Overflow detection
Overflow

Progetto di un sommatore con operandi a due bit

$a_1b_1a_0b_0$	$r_{out}\ s_1s_0$
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 0 1
0 0 1 1	0 1 0
0 1 0 0	0 1 0
0 1 0 1	0 1 1
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	0 1 1
1 0 1 1	1 0 0
1 1 0 0	1 0 0
1 1 0 1	1 0 1
1 1 1 0	1 0 1
1 1 1 1	1 1 0

Sintesi

Confronto con approccio iterativo

Commento sulle operazioni aritmetiche

- Sottrazione: si può implementare come addizione con operandi rappresentati in complemento a due

- Moltiplicazione: si può implementare come somme successive

- Divisione: si può implementare come sottrazioni successive

Quindi tutte le operazioni si potrebbero implementare solo con il circuito addizionatore, anche se poi le moltiplicazioni e le divisioni si realizzano, per motivi di velocità, con circuiti sequenziali ad hoc.