Corso di Calcolatori Elettronici I

Circuiti combinatori

ing. Alessandro Cilardo

Corso di Laurea in Ingegneria Biomedica

Funzioni combinatorie elementari

- AND, OR, NOT
- NAND ed NOR
- implicazione
- XOR ed EQ

Funzioni NAND e NOR

NAND: definita come **AND** seguita da **NOT** (negata della AND)

NOR: definita come **OR** seguita da **NOT** (negata della OR)

notazioni:

$$x \uparrow y = \overline{x \cdot y} = \overline{x} + \overline{y}$$

$$x \downarrow y = \overline{x + y} = \overline{x} \cdot \overline{y}$$

(da *De Morgan*)

NAND *e* **NOR** *a più ingressi:*

$$x_1 \uparrow x_2 \uparrow \ldots \uparrow x_n = \overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} + \overline{x_2} + \ldots \overline{x_n}$$

$$x_1 \downarrow x_2 \downarrow \ldots \downarrow x_n = \overline{x_1 + x_2 + \ldots + x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n}$$

Funzioni NAND e NOR

- NAND e NOR ***non*** godono della proprietà associativa

$$(x_1 \uparrow x_2) \uparrow x_3 \neq x_1 \uparrow (x_2 \uparrow x_3) \neq x_1 \uparrow x_2 \uparrow x_3$$

$$(x_1 \downarrow x_2) \downarrow x_3 \neq x_1 \downarrow (x_2 \downarrow x_3) \neq x_1 \downarrow x_2 \downarrow x_3$$

- E' possibile ottenere una NOT tramite NAND e NOR

$$x \uparrow 1 = \overline{x \cdot 1} = \overline{x} \qquad x \uparrow x = \overline{x \cdot x} = \overline{x}$$

…o anche…

$$x \downarrow 0 = \overline{x + 0} = \overline{x} \qquad x \downarrow x = \overline{x + x} = \overline{x}$$

Funzioni NAND e NOR

- Riassumendo, le NAND permettono di ottenere una NOT, una AND e, usando De Morgan, una OR
- Similmente per la NOR
- ricordiamo che {AND,OR,NOT} è un insieme funzionalmente completo, quindi →

{NAND} e {NOR} sono due
insiemi funzionalmente completi

Forme NAND e NOR di una funzione

- una forma elementare di tipo P si trasforma in una forma NAND a due livelli operando come segue:
  - tutti gli operatori si trasformano in NAND, rispettando le priorità;
  - le clausole costituite da un solo letterale vengono negate.

$$f = x_1 + x_2 + \ldots + x_n = \overline{\overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n}} = \overline{x_1} \uparrow \overline{x_2} \uparrow \ldots \uparrow \overline{x_n}$$

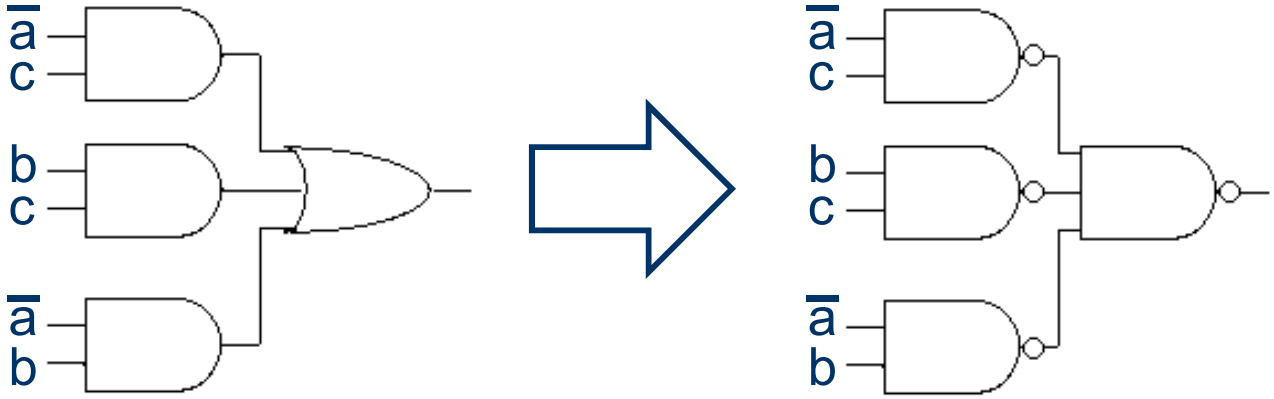
- Dualmente per la forma di tipo S

Forme NAND e NOR di una funzione

- Esempio

$$f = \bar{a}c + bc + \bar{a}b = \overline{\overline{\bar{a}c + bc + \bar{a}b}} = \quad \text{(De Morgan)}$$

$$= \overline{\overline{\bar{a}c} \cdot \overline{bc} \cdot \overline{\bar{a}b}} = (\bar{a} \uparrow c) \uparrow (b \uparrow c) \uparrow (\bar{a} \uparrow b)$$

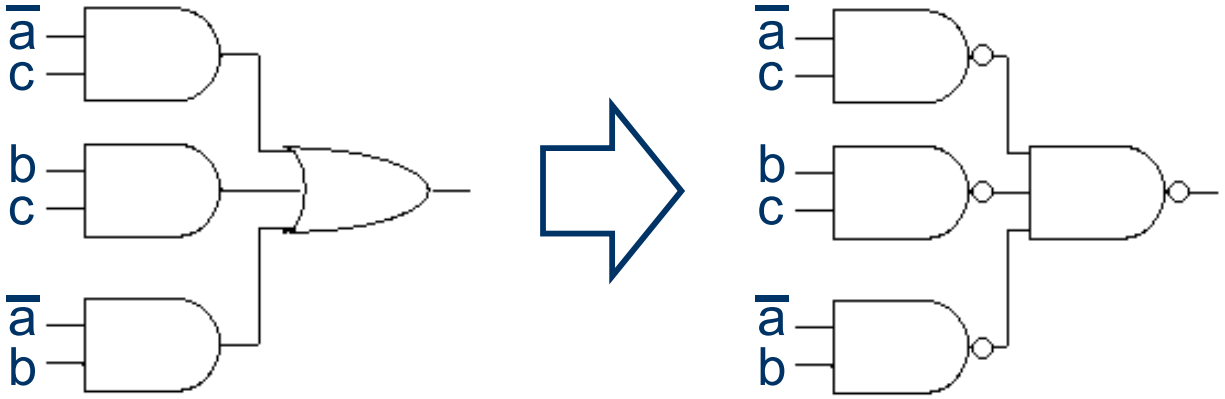


Forme NAND e NOR di una funzione

- Il procedimento è in effetti generalizzabile:
- *Da una forma somma di prodotti si può passare ad una forma* NAND *semplicemente sostituendo ogni operatore (+/·) con una porta* NAND *(o da una negazione se al posto di un prodotto è presente una singola variabile)*

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

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



Forme NAND e NOR di una funzione

- Dualmente:
- *Da una forma prodotto di somme si può passare ad una forma* NOR *semplicemente sostituendo ogni operatore (+/·) con una porta* NOR *(o da una negazione se al posto di un prodotto è presente una singola variabile)*

$$f = (\overline{a} + c) \cdot c \cdot (\overline{a} + \overline{c} + d) = \overline{\overline{(\overline{a} + c) \cdot c \cdot (\overline{a} + \overline{c} + d)}} =$$

$$\overline{\overline{(\overline{a} + c)} + \overline{c} + \overline{(\overline{a} + \overline{c} + d)}} = (\overline{a} \downarrow c) \downarrow \overline{c} \downarrow (\overline{a} \downarrow \overline{c} \downarrow d)$$

Funzione implicazione

- ha due ingressi, *a* e *b*, ordinati (*non è commutativa*)
- pari ad 1 se l'ingresso *a* **implica** l'ingresso *b*
  - "*non può accadere che a sia* 1 *senza che lo sia b*"

a	**b**	**y**
0	0	*1*
0	1	*1*
1	0	*0*
1	1	*1*

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

Forma S

$$a \Rightarrow b$$

Funzioni XOR e EQ

XOR (funzione OR esclusivo, anche detta somma modulo 2)

$$x \oplus y = \overline{x \equiv y} = x\bar{y} + \bar{x}y = (x + y)(\bar{x} + \bar{y})$$

x	*y*	x XOR y
0	0	*0*
0	1	*1*
1	0	*1*
1	1	*0*

EQ (funzione equivalenza)

$$x \equiv y = \overline{x \oplus y} = xy + \bar{x}\bar{y} = (x + \bar{y})(\bar{x} + y)$$

x	*y*	x EQ y
0	0	*1*
0	1	*0*
1	0	*0*
1	1	*1*

Funzione XOR

- Forme alternative:

$$x \oplus y = (x \uparrow \overline{y}) \uparrow (\overline{x} \uparrow y)$$

forma a 2 livelli (costo di porte **5** includendo anche la NOT)

$$x \oplus y = x(\overline{x} + \overline{y}) + y(\overline{x} + \overline{y}) =$$
$$= (x \uparrow (x \uparrow y)) \uparrow (y \uparrow (x \uparrow y))$$

forma a 3 livelli (costo di porte **4** riutilizzando un fattore comune)

Proprietà della XOR

- a differenza di NAND e NOR, la XOR non costituisce un insieme funzionalmente completo
- Permette di ottenere una NOT

$$x \oplus 1 = x \cdot 0 + \overline{x} \cdot 1 = \overline{x}$$

$$x \equiv 0 = x \cdot 1 + \overline{x} \cdot 0 = \overline{x}$$

Funzioni Parità e Disparità

- ***p(X)***: vera se e solo se il numero di '1' presenti in ***X*** è pari
- ***d(X)***: vera se e solo se il numero di '1' presenti in ***X*** è dispari

- si ha ovviamente: $d(X) = \overline{p(X)}$

- Per funzioni a due variabili:

$$d(x_1, x_2) = x_1 \oplus x_2$$
$$p(x_1, x_2) = x_1 \equiv x_2$$

Funzioni Parità e Disparità

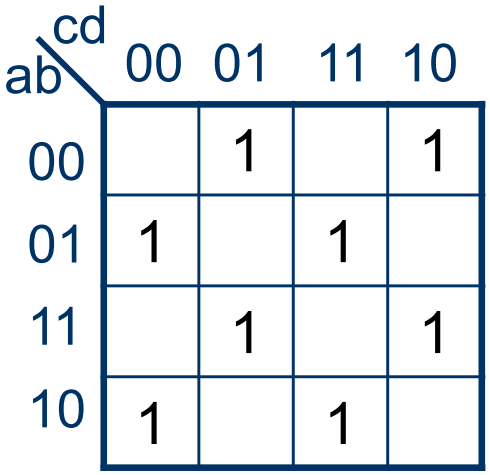
- Si noti che la forma minima di queste due funzioni coincide con la forma canonica di tipo P
  - Mappa di Karnaugh “a scacchiera”
- Per un numero di ingressi elevato non è praticabile usare la forma minima, che presenta un costo di porte pari a

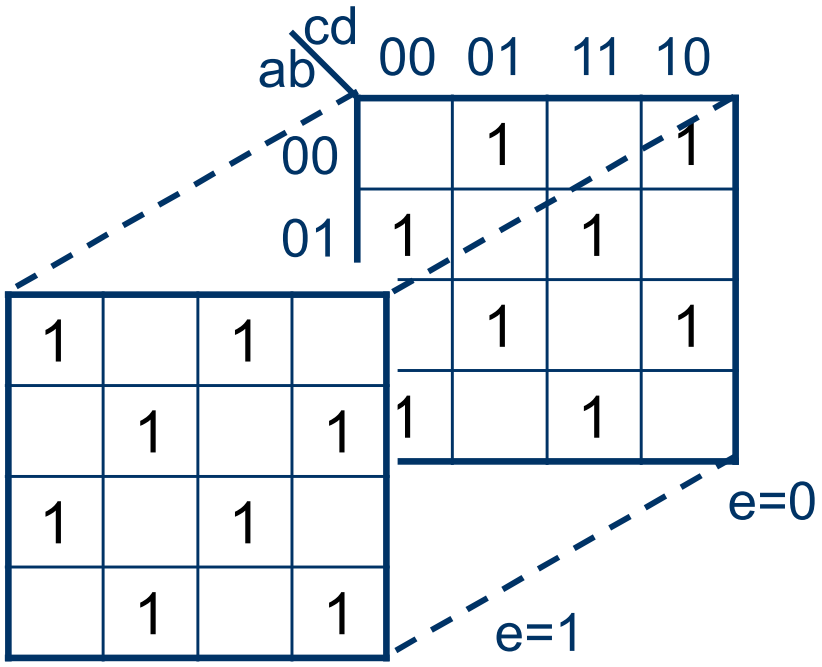
$$C_p = 2^{n-1} + 1$$

- Sono necessarie strutture a più di due livelli

Funzioni Parità e Disparità

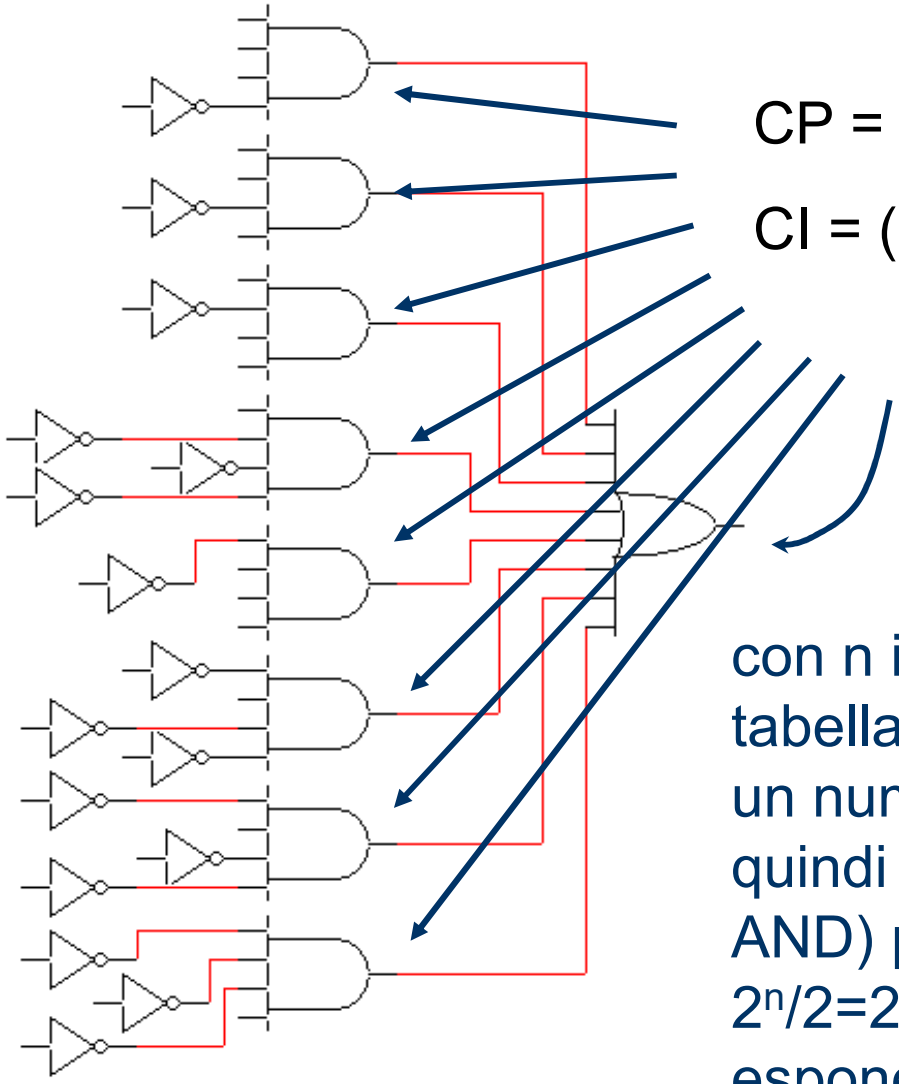
- Mappa di Karnaugh “a scacchiera”





Funzioni Parità e Disparità

a	*b*	*c*	*d*	y
0	0	0	0	*0*
0	0	0	1	*1*
0	0	1	0	*1*
0	0	1	1	*0*
0	1	0	0	*1*
0	1	0	1	*0*
0	1	1	0	*0*
0	1	1	1	*1*
1	0	0	0	*1*
1	0	0	1	*0*
1	0	1	0	*0*
1	0	1	1	*1*
1	1	0	0	*0*
1	1	0	1	*1*
1	1	1	0	*1*
1	1	1	1	*0*



$CP = 2^{n-1} + 1$

$CI = (n + 1)\ 2^{n-1}$

con n ingressi, la tabella di verità ha un numero di **1** (e quindi di porte AND) pari a $2^n/2=2^{n-1}$, (cresce esponenzialmente!)

Funzioni Parità e Disparità

- La funzione disparità può essere scritta come sommatoria dei suoi mintermini *dispari*
- dualmente, la funzione parità è pari alla sommatoria dei suoi mintermini *pari*
- Le funzioni di costo assumono valori inaccettabili per un numero di ingressi alto:
  - Costo di porte: $CP = 2^{n-1} + 1$
  - Costo di ingressi: $CI = (n + 1)\, 2^{n-1}$

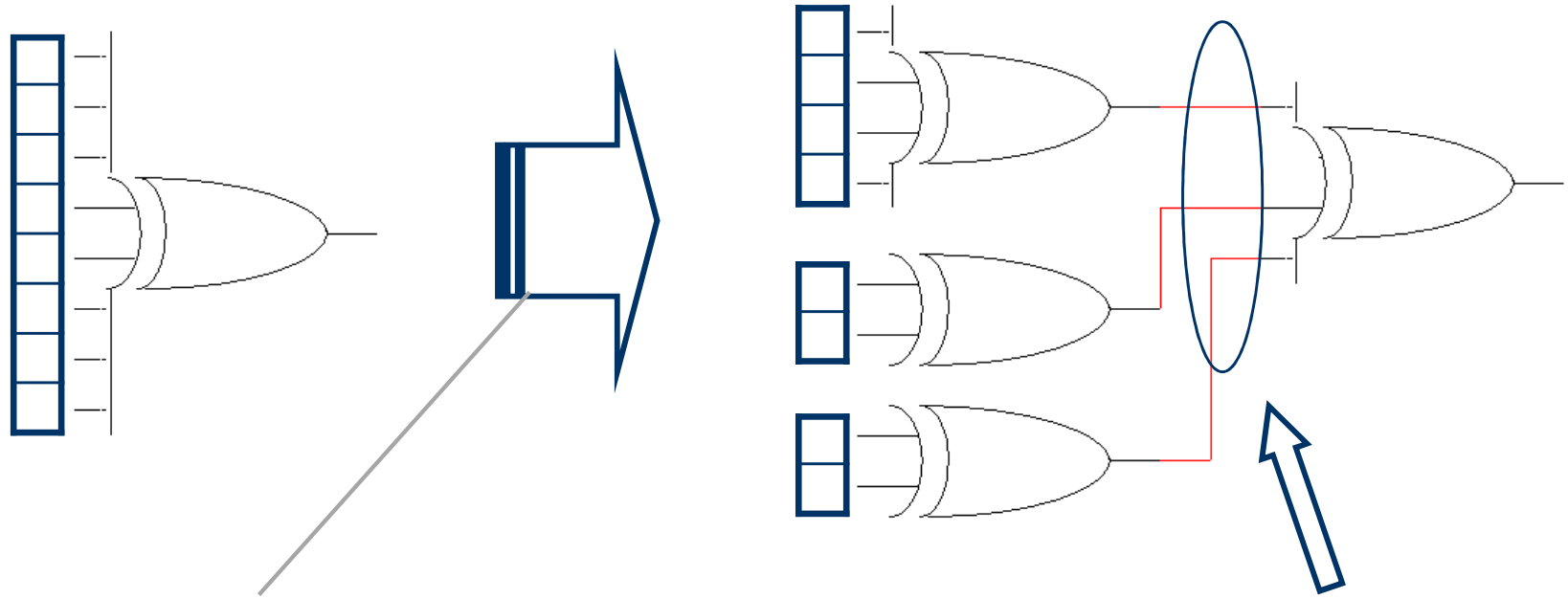
Funzioni Disparità

- La funzione di disparità (a differenza di quella di parità) gode della proprietà **associativa**

$$d(X) = d\left[d(X^{(1)}),\ldots,\ d(X^{(m)})\right]$$

- possiamo partizionare gli ingressi ***X*** in gruppi $\boldsymbol{X^{(i)}}$
- la funzione disparità sugli ingressi ***X*** è vera se e solo se è dispari il numero di gruppi $\boldsymbol{X^{(i)}}$ con un numero dispari di '1'
- Funzioni disparità a più ingressi si possono ottenere come un albero di funzioni disparità "più piccole"

Proprietà associativa della disparità



Proprietà associativa della XOR:si può scomporre una XOR di molti ingressi in una XOR di porte XOR aventi ciascuna pochi ingerssi

il numero di uscite alte è dispari se e solo se è dispari il numero di ingressi alti, indipendentemente da come questi sono partizionati

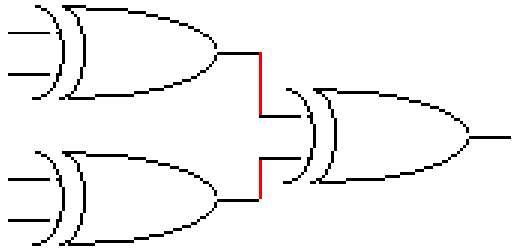
Funzioni Disparità

a	*b*	*c*	*d*	y
0	0	0	0	*0*
0	0	0	1	*1*
0	0	1	0	*1*
0	0	1	1	*0*
0	1	0	0	*1*
0	1	0	1	*0*
0	1	1	0	*0*
0	1	1	1	*1*
1	0	0	0	*1*
1	0	0	1	*0*
1	0	1	0	*0*
1	0	1	1	*1*
1	1	0	0	*0*
1	1	0	1	*1*
1	1	1	0	*1*
1	1	1	1	*0*

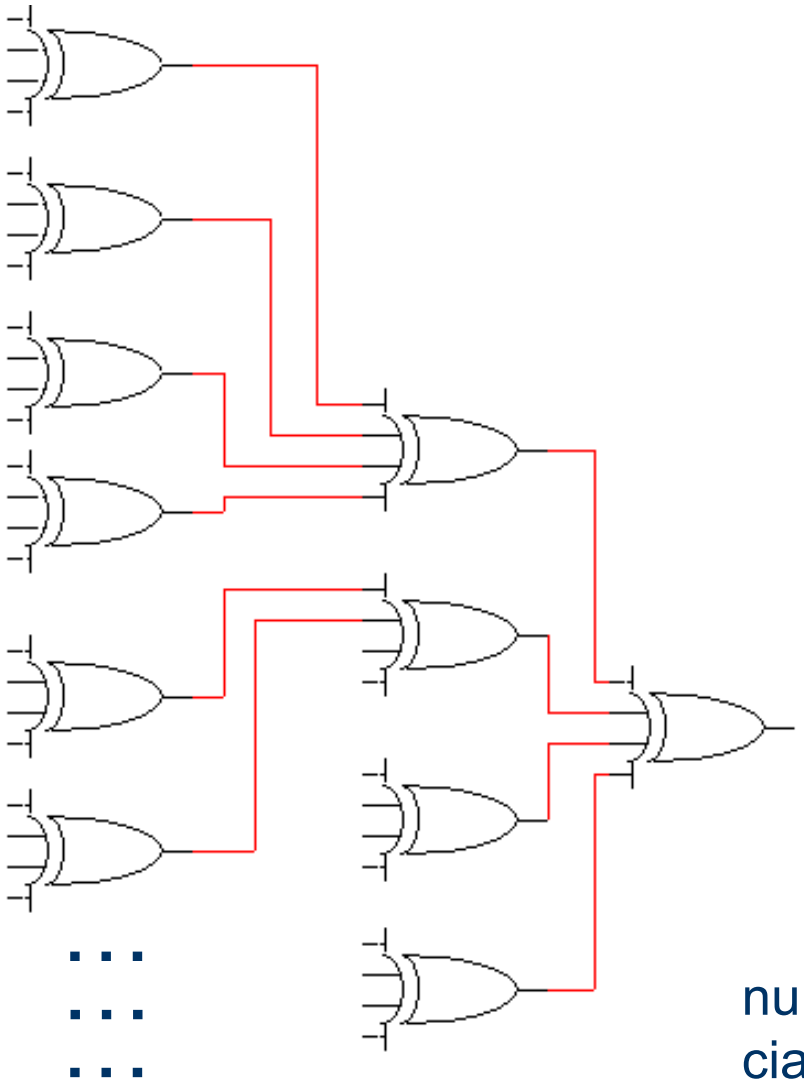
Una XOR a 4 ingressi può quindi essere ottenuta interconnettendo più XOR a 2 ingressi

Il numero di componenti necessarie non cresce più esponenzialmente con il numero di ingressi, ma **linearmente**!

Il **ritardo** della rete però tende ad aumentare, poiché in questo caso occorre attraversare un numero maggiore di porte tra ingresso e uscita



Funzioni Disparità



Esempio di albero di **3** livelli, composto soltanto di XOR di **4** ingressi

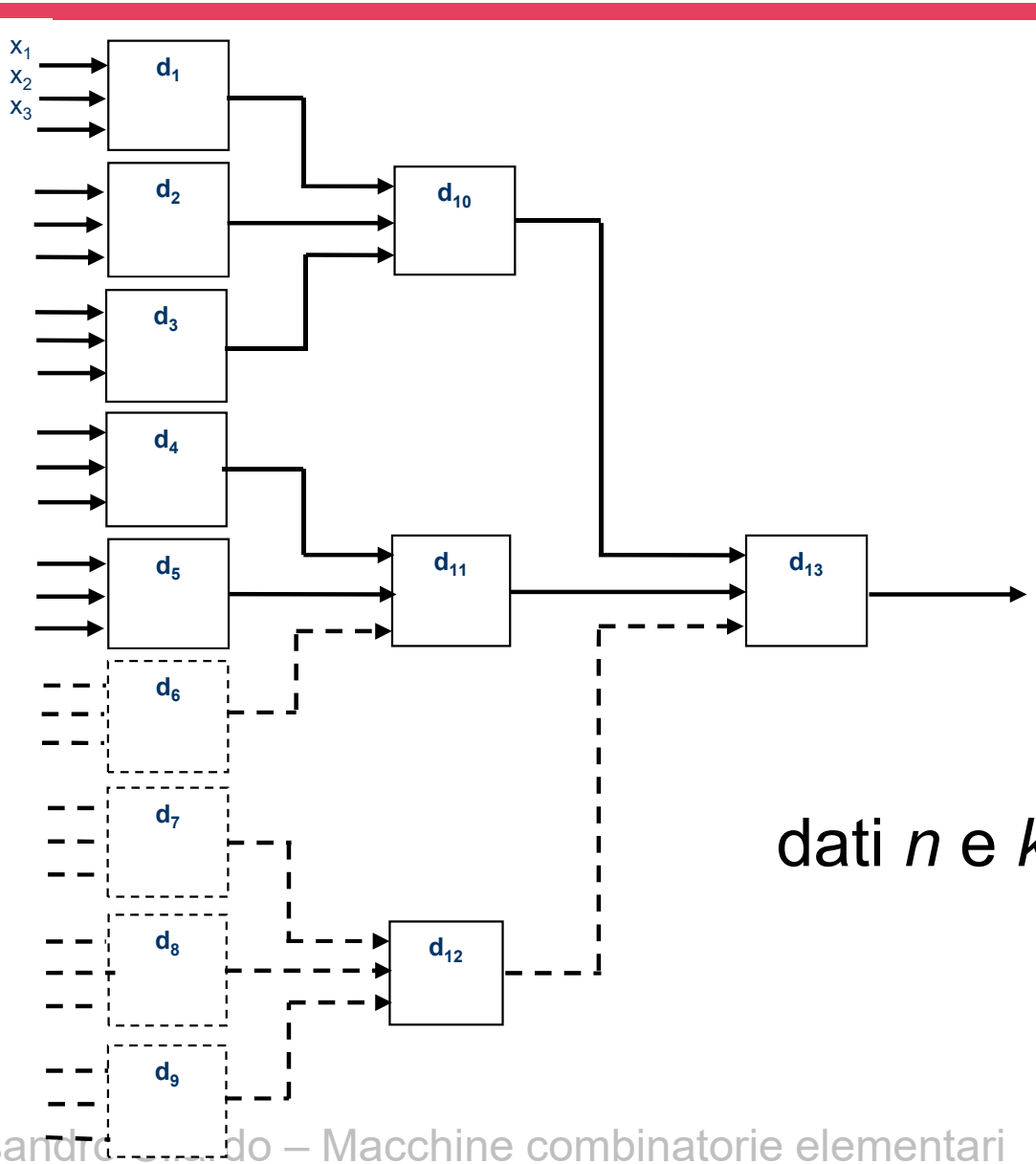
E' possibile ottenere una XOR complessiva avente 4 x 4 x 4 =

$$\mathbf{4^3} = 64 \text{ ingressi}$$

numero di ingressi di ciascuna XOR (4)

numero di livelli (3)

Rete ad albero per la disparità



- *n*: numero di ingressi complessivi
- *k*: numero di ingressi della singola funzione
- *l*: numero di livelli
- *q*: numero di porte

dati *n* e *k*:

$$l = \lceil \log_k n \rceil$$

$$q = \left\lceil \frac{n-1}{k-1} \right\rceil$$

Protezione dagli errori

Materiale facoltativo

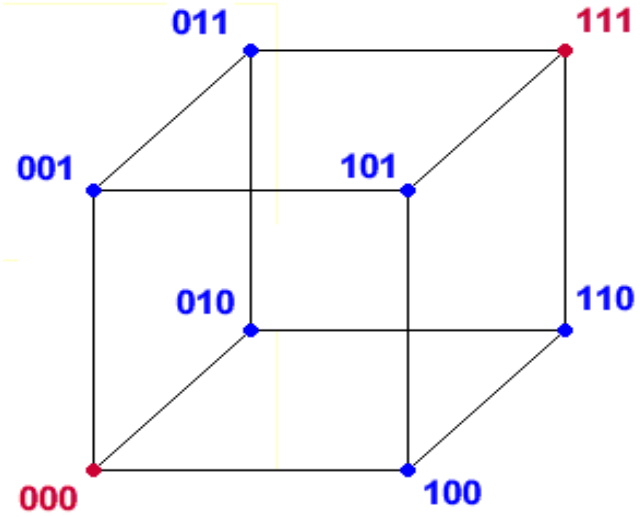
- In caso di codice non ridondante, non è possibile garantire alcuna forma di protezione da errori
  - Anche un errore su un singolo bit trasforma la parola originaria in una parola differente comunque ammessa nel codice.
  - Nei codici **ridondanti** questo può non avvenire in alcuni casi:

    Esempio: codice BCD (ridondante)

Protezione dagli errori

Materiale facoltativo

- Distanza di Hamming *(D)*
  - minimo numero di bit di differenza tra due parole codice
- **Teorema di Hamming**
  - *un codice a distanza minima D = d + 1 può <u>individuare</u> gli errori simultanei su d bit;*
  - *un codice a distanza minima D = 2 c +1 può <u>correggere</u> gli errori simultanei su c bit.*



- Il livello di protezione dall'errore cresce con la **ridondanza**
- Quando un codice di Hamming segnala un errore fornisce la **CERTEZZA** della sua presenza
- Quando invece non rileva errori, il codice garantisce un certo valore di **PROBABILITA'** con la quale la parola che risulta legale è effettivamente esente da errori

Protezione dagli errori

Materiale facoltativo

- Un esempio di codice ridondante con distanza minima D=d+1 si può semplicemente ottenere richiedendo una di queste due condizioni:
  - La codifica di qualsiasi simbolo abbia sempre un numero **pari** di ‘1’

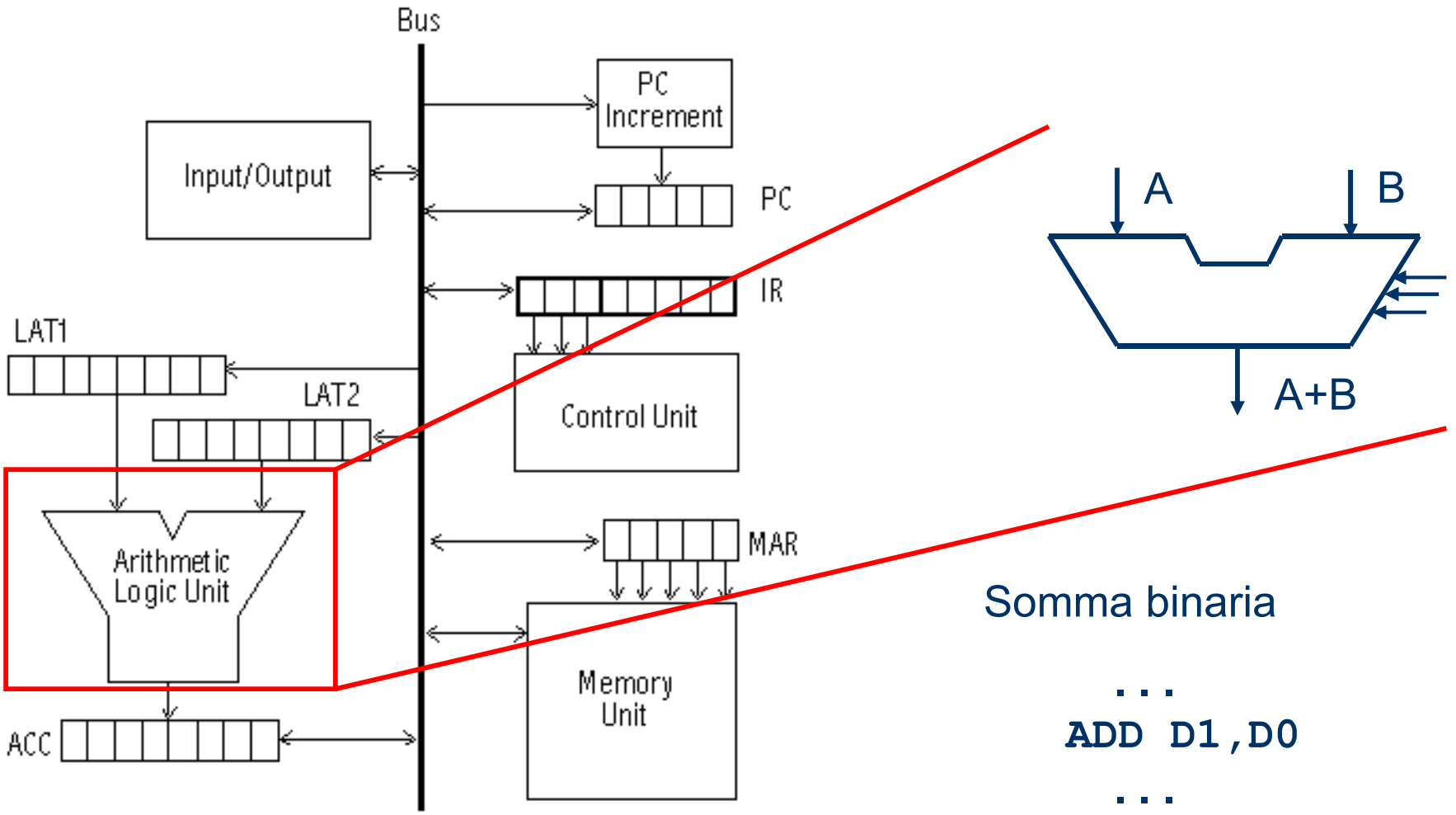
  oppure
  - La codifica di qualsiasi simbolo abbia sempre un numero **dispari** di ‘1’
- In ciascuno dei due casi, è evidente che basta aggiungere al più un ‘1’ per rispettare una delle due condizioni

  → il codice ha ridondanza di 1 bit

Circuiti combinatori notevoli

- codificatori/decodificatori/transcodificatori
- multiplexer/demultiplexer
- addizionatori/sottrattori
- comparatori

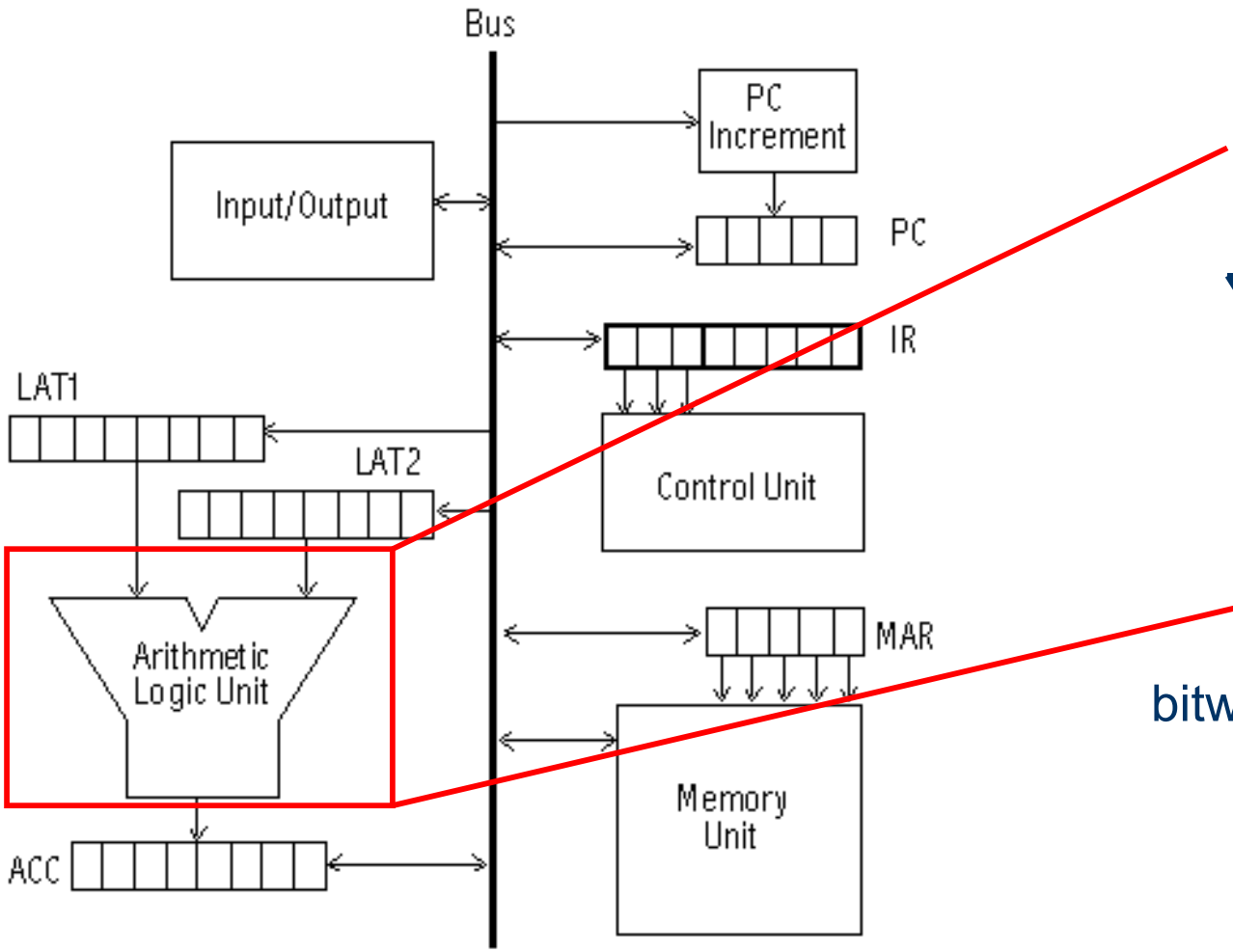
Esempi di circuiti combinatori

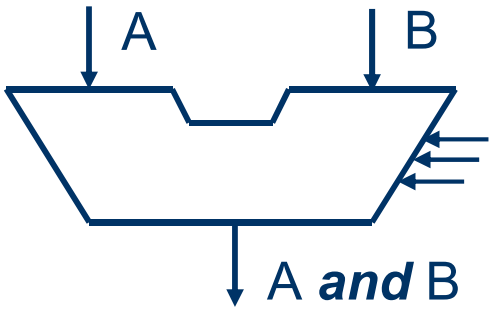


Somma binaria

```
...
ADD D1,D0
...
```

Esempi di circuiti combinatori

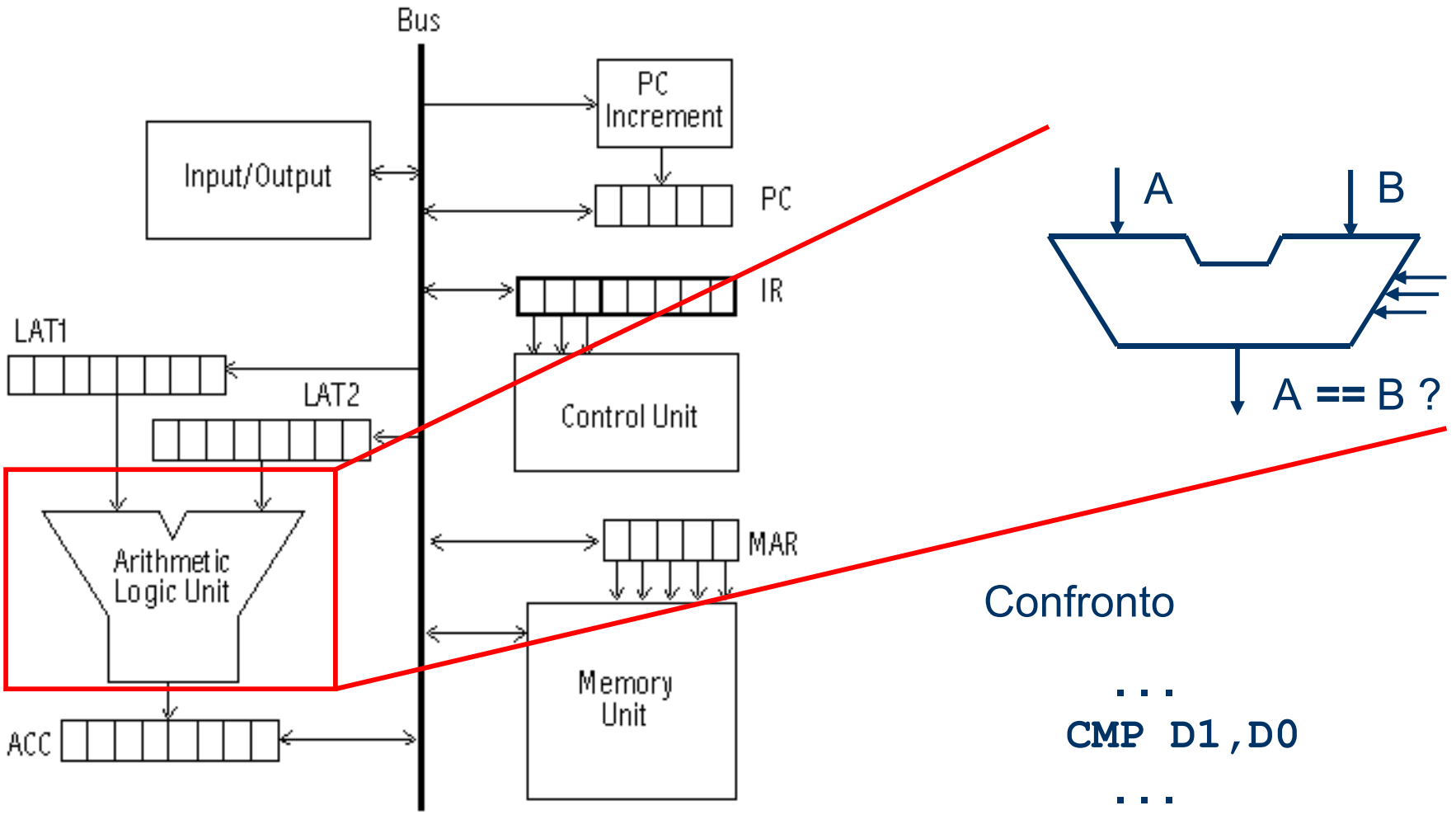




bitwise AND (AND *bit a bit*)

```
...
AND D1,D0
...
```

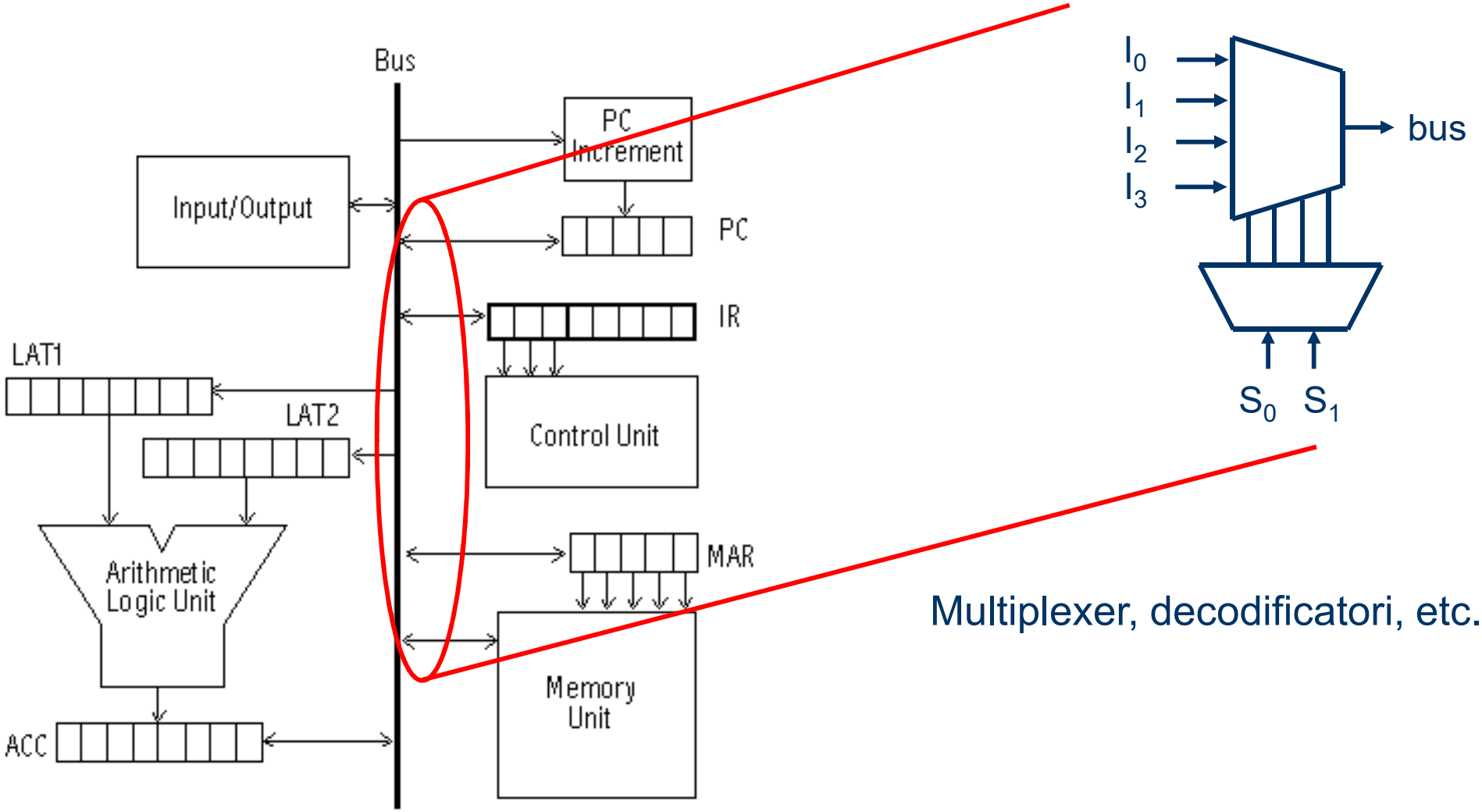
Esempi di circuiti combinatori



Confronto

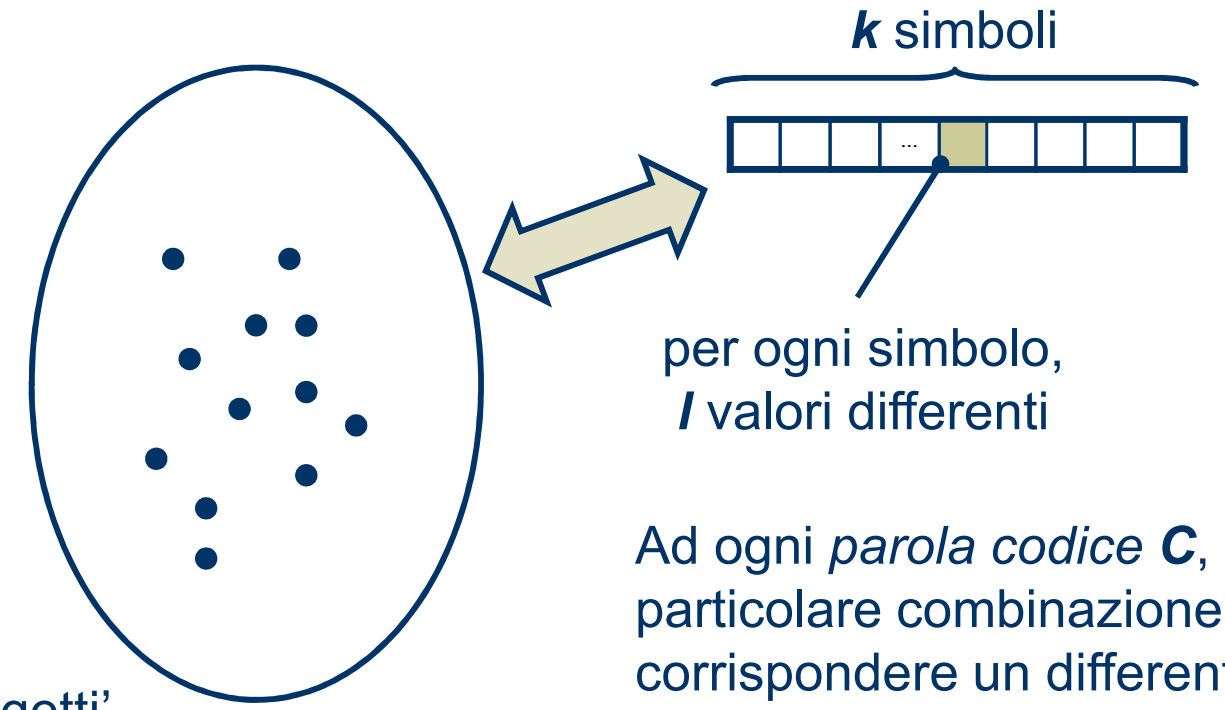
```
...
CMP D1,D0
...
```

Esempi di circuiti combinatori



Multiplexer, decodificatori, etc.

Codifica



n 'oggetti'

("alfabeto sorgente")

Ad ogni *parola codice* **C**, ovvero ad ogni particolare combinazione di **k** simboli, deve corrispondere un differente oggetto degli **n**

➔ $k^l \geq n$

- $k^l = n$: codici completi
- $k^l > n$: codici *ridondanti*, o incompleti

Esempi di codifica

- Esempi di codici intermedi completi: *Ottale, Esadecimale*
- Esempio di codice intermedio incompleto: *8-4-2-1 (BCD)*

ottale

v	o_2	o_1	o_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

esadecimale

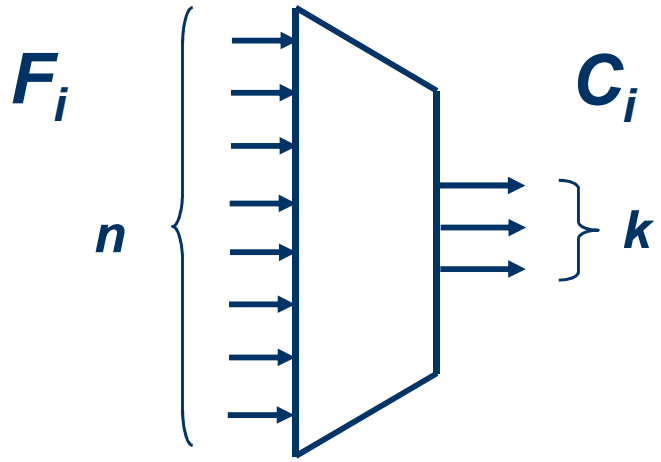
v	o_3	o_2	o_1	o_0	v	o_3	o_2	o_1	o_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	A	1	0	1	0
3	0	0	1	1	B	1	0	1	1
4	0	1	0	0	C	1	1	0	0
5	0	1	0	1	D	1	1	0	1
6	0	1	1	0	E	1	1	1	0
7	0	1	1	1	F	1	1	1	1

BCD

v	o_3	o_2	o_1	o_0	v	o_3	o_2	o_1	o_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0					
3	0	0	1	1					
4	0	1	0	0					
5	0	1	0	1					
6	0	1	1	0					
7	0	1	1	1					

Reti di codifica

- ***n*** ingressi *decodificati* F_i
  - solo uno degli ingressi può essere alto, rappresentando così uno degli ***n*** valori oggetto della codifica
- ***k*** uscite *codificate* C_i
- dal punto di vista algebrico, le C_i possono essere scritte come somme degli ingressi F_i per i quali la cifra ***i*** vale **1** nella codifica applicata (vedi esempio nel lucido successivo)



Reti di codifica

Può essere alto solo un ingresso alla volta
→ se uno è alto, gli altri devono necessariamente essere bassi

$$C_i = \sum_{j=0}^{n-1} F_j \cdot \tau_j$$

F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	C_2	C_1	C_0
-	-	-	-	-	-	-	1	0	0	0
-	-	-	-	-	-	1	-	0	0	1
-	-	-	-	-	1	-	-	0	1	0
-	-	-	-	1	-	-	-	0	1	1
-	-	-	1	-	-	-	-	1	0	0
-	-	1	-	-	-	-	-	1	0	1
-	1	-	-	-	-	-	-	1	1	0
1	-	-	-	-	-	-	-	1	1	1

τ_j

ad esempio:

$$C_1 = F_7 + F_6 + F_3 + F_2$$

Reti di codifica

Esempio di codificatore BCD

v	C_3	C_2	C_1	C_0	v	C_3	C_2	C_1	C_0
0	0	0	0	0	**8**	1	0	0	0
1	0	0	0	1	**9**	1	0	0	1
2	0	0	1	0					
3	0	0	1	1					
4	0	1	0	0					
5	0	1	0	1					
6	0	1	1	0					
7	0	1	1	1					

$$C_3 = F_8 + F_9$$

$$C_2 = F_4 + F_5 + F_6 + F_7$$

$$C_1 = F_2 + F_3 + F_6 + F_7$$

$$C_0 = F_1 + F_3 + F_5 + F_7 + F_9$$

Reti di codifica

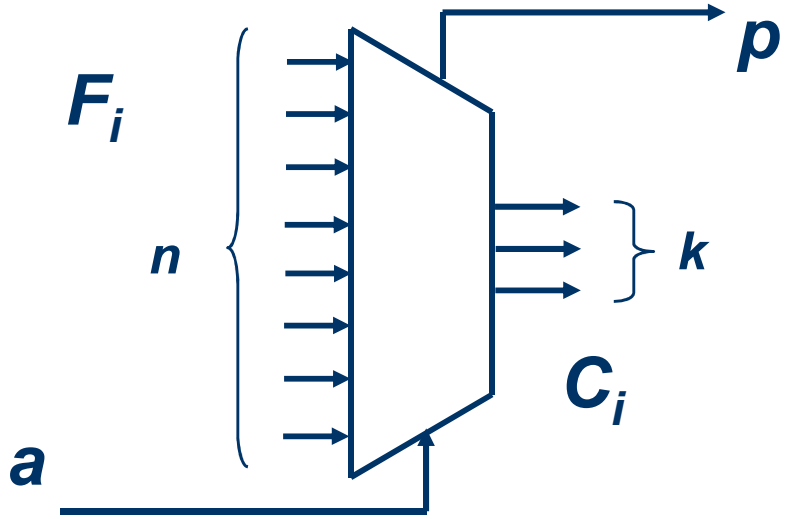
- Osservazione:
  - la condizione $F_0=1$, $F_i=0$ per ogni $i>0$ (corrispondente alla codifica in uscita '**00**…**0**' negli esempi precedenti) può essere confusa con il caso in cui *tutti* gli ingressi F_i sono zero
  - $F_i=0$ per ogni i (tutti gli ingressi pari a zero) non corrisponde a nessun valore codificato
- si aggiunge spesso un'ulteriore uscita p, che indica se l'uscita è *valida*:

$$p = \sum_{i=0}^{n-1} F_i$$

è la **OR** di tutti gli ingressi: alta se uno degli ingressi è alto, ovvero se è presente un ingresso da codificare

Reti di codifica

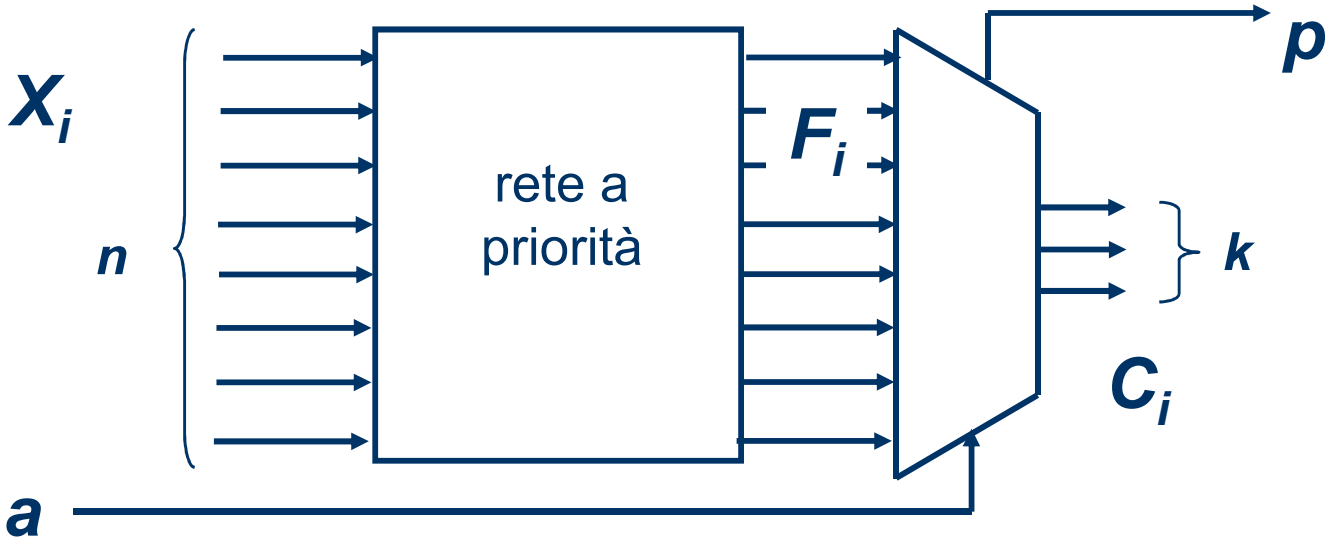
- Gli ingressi F_i possono essere abilitati da un ulteriore segnale ***a***
  - se pari a **0**, **maschera** gli ingressi, ovvero li rende inefficaci
- gli effettivi ingressi F'_i che vanno in ingresso al codificatore sono quindi $F'_i = F_i \cdot a$



Reti di codifica a priorità

Materiale facoltativo

- Un codificatore può essere preceduto da una **rete a priorità**
- più ingressi possono essere contemporaneamente alti
- viene codificato quello con priorità assegnata maggiore



Reti di codifica a priorità

Materiale facoltativo

- ***n*** ingressi X_i
- ***n*** uscite corrispondenti F_i, che rappresentano gli ingressi del codificatore
- fra gli ingressi è definita una priorità, ad esempio:
  - per fissare le idee: «X_i è prioritario su X_j se $i > j$»
- L'uscita F_i è alta se e solo se X_i è alto e *tutti gli altri ingressi prioritari su* X_i sono bassi.

$$F_{n-1} = X_{n-1}$$

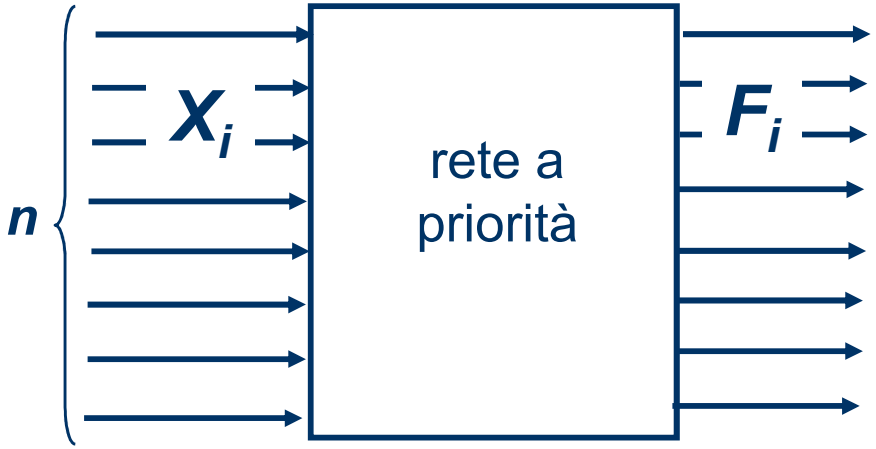
$$F_{n-2} = X_{n-2}\overline{X_{n-1}}$$

.....

$$F_i = X_i\overline{X_{i+1}}\ldots\overline{X_{n-1}}$$

.....

$$F_0 = X_0\overline{X_1}\ldots\overline{X_{n-1}}$$



Reti di codifica a priorità

Materiale facoltativo

- E' possibile un'ottimizzazione: per le F_i non è necessario realizzare le condizioni viste prima in maniera completa
- Ricordiamo che ciascuna uscita C_i è semplicemente una **OR** di alcuni degli ingressi F_i
- E' possibile sfruttare il fatto che le codifiche corrispondenti ad alcune combinazioni di ingresso rispettano implicitamente la condizione di priorità:
- Esempio:
  - 5 (**1**0**1**) contiene tutti gli '1' che sono presenti nella codifica 1 (00**1**), di 4 (**1**00) e di 0 (000)
  - 3 (0**11**) contiene tutti gli '1' di 1 (00**1**), di 2 (0**1**0) e di 0 (000)
  - 7 (**111**) contiene tutti gli '1' presenti nella codifica di tutti gli altri
    → sull'ingresso 7 non è necessario imporre la condizione di priorità

Reti di codifica a priorità

Materiale facoltativo

- L'equazione generale si semplifica quindi così:

$$F_7 = X_7 \qquad F_3 = X_3\overline{X_4}\,\overline{X_5}\,\overline{X_6}$$

$$F_6 = X_6 \qquad F_2 = X_2\overline{X_4}\,\overline{X_5}$$

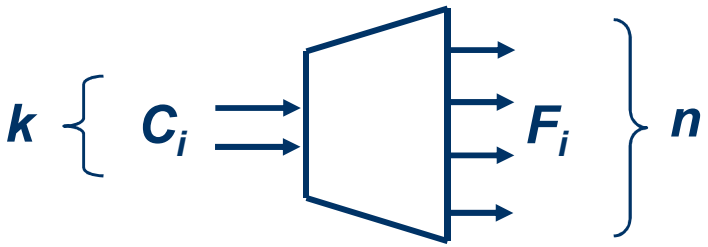
$$F_5 = X_5\overline{X_6} \qquad F_1 = X_1\overline{X_2}\,\overline{X_4}\,\overline{X_6}$$

$$F_4 = X_4 \qquad F_0 = X_0$$

Ad esempio, per F_5 non imponiamo esplicitamente la condizione **NOT**(X_7). Infatti, qualora si presentino entrambi X_7 (codifica **111**) e X_5 (**101**), *possiamo accettare* il fatto che F_5 e F_7 si alzino contemporaneamente: a causa della **OR** con cui generiamo le uscite codificate C_i, l'effetto sarà comunque quello di produrre la codifica a maggiore priorità **111**

Decodifica

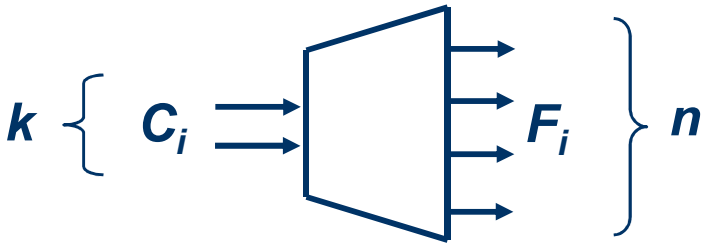
- Il processo di codifica è reversibile
- Decodifica
  - associa ad ogni parola codice rappresentata dagli ingressi C_i la sua *rappresentazione decodificata* F_i



una sola delle ***n*** uscite è alta:
rappresentazione decodificata di
uno degli ***n*** valori

Rete di decodifica

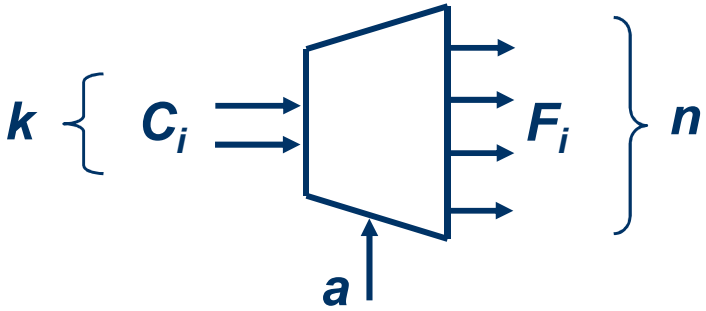
- ***k*** ingressi codificati C_i ed ***n*** uscite decodificate F_i
  - Ognuna delle uscite F_i può essere scritta come uno specifico *mintermine*, quello corrispondente al valore codificato in ingresso



ad esempio: $F_1 = C_0 \cdot \overline{C_1}$

alto se in ingresso si presenta la combinazione **01**

- Abilitazione ***a***
  - se **0**, rende **zero** le uscite, indipendentemente dagli ingressi



ad esempio: $F_1 = a \cdot C_0 \cdot \overline{C_1}$

alto se in ingresso si presenta la combinazione **01** *ed* ***a*** *è* **1**

Rete di decodifica

BCD

v	C_3	C_2	C_1	C_0	v	C_3	C_2	C_1	C_0
0	0	0	0	0	**8**	1	0	0	0
1	0	0	0	1	**9**	1	0	0	1
2	0	0	1	0					
3	0	0	1	1					
4	0	1	0	0					
5	0	1	0	1					
6	0	1	1	0					
7	0	1	1	1					

$$F_0 = \overline{C_3}\,\overline{C_2}\,\overline{C_1}\,\overline{C_0}$$

$$F_1 = \overline{C_3}\,\overline{C_2}\,\overline{C_1}\,C_0$$

$$F_2 = \overline{C_3}\,\overline{C_2}\,C_1\,\overline{C_0}$$

… … …

- per qualsiasi codifica è quindi sempre possibile scrivere le F_i semplicemente come *mintermini* (prodotti di tutti gli ingressi)
- La particolare codifica determina la specifica scelta dei mintermini

Decodificatore incompleto: BCD

$$F_0 = \overline{C_3}\,\overline{C_2}\,\overline{C_1}\,\overline{C_0}; \quad F_1 = \overline{C_3}\,\overline{C_2}\,\overline{C_1}C_0; \quad F_2 = \overline{C_2}C_1\overline{C_0};$$

$$F_3 = \overline{C_2}C_1C_0; \quad F_4 = C_2\overline{C_1}\,\overline{C_0}; \quad F_5 = C_2\overline{C_1}C_0;$$

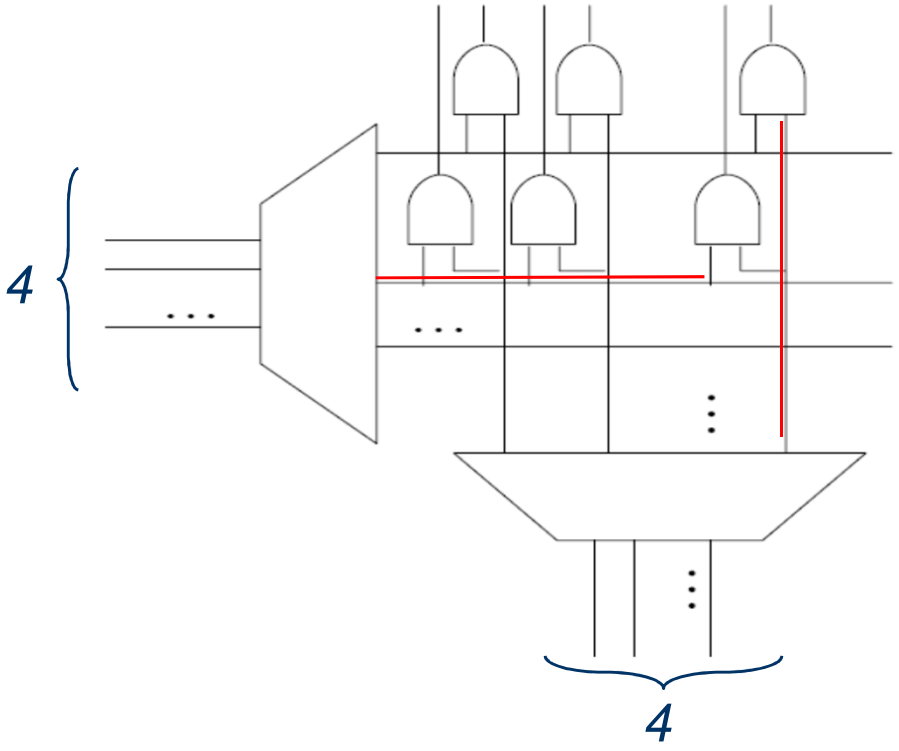
$$F_6 = C_2C_1\overline{C_0}; \quad F_7 = C_2C_1C_0; \quad F_8 = C_3\overline{C_0}; \quad F_9 = C_3C_0$$

C_1C_0 \ C_3C_2	00	01	11	10
00	F_0	F_4	-	F_8
01	F_1	F_5	-	F_9
11	F_3	F_7	-	-
10	F_2	F_6	-	-

- La mappa rappresenta 10 diverse funzioni, ciascuna con un unico 1 e sei punti di *don't care*
- La rete così ottenuta ha un numero di porte AND uguale alla rete realizzata come sottorete di quella completa, ma non tutte le sue porte sono a 4 ingressi
- e quindi ha un costo di ingressi ***Ci*** inferiore.

Decodificatori composti

- Più decodificatori possono essere composti per realizzare decodificatori più grandi
  - Esempio: Semiselezione (1/16)

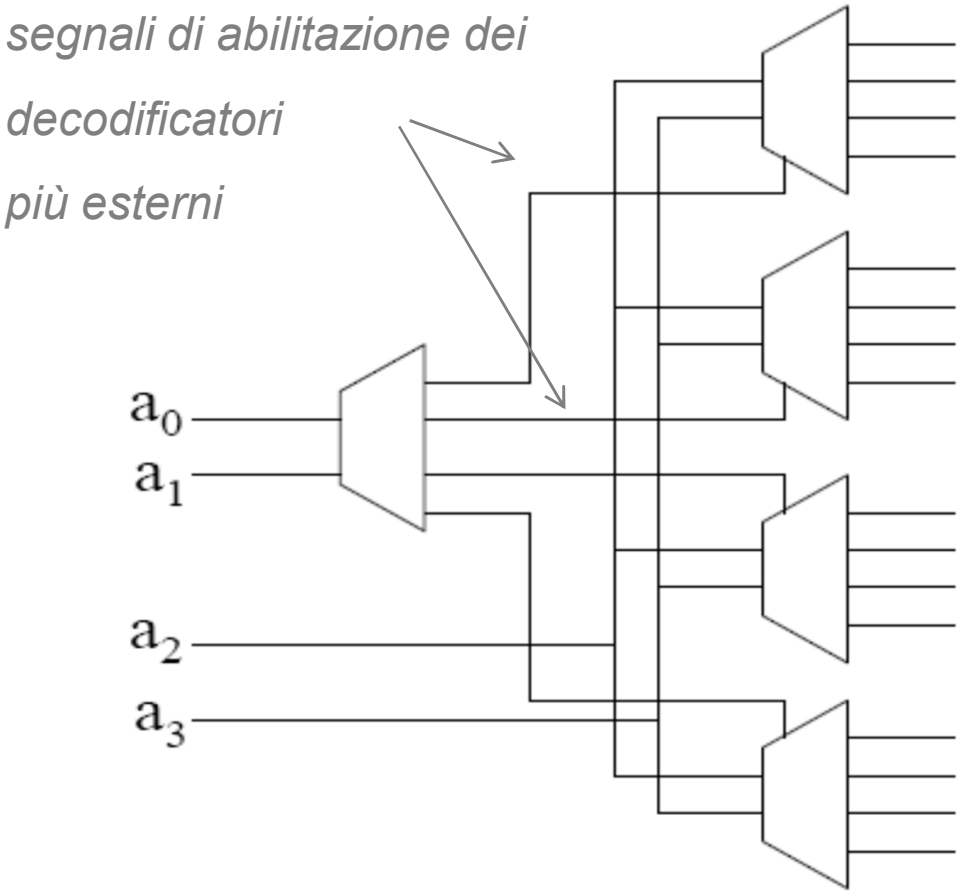


Gli ingressi (supponiamo ad esempio di averne 8 in tutto) sono divisi in due gruppi. Ciascun gruppo entra in uno dei due decodificatori (ad esempio, decod. 4:16).

In corrispondenza di ciascuno degli incroci delle uscite (in numero pari a 16x16=256), si colloca una AND, che vedrà in ingresso due 1 solo in corrispondenza di quella particolare combinazione degli ingressi che rende alta sia la linea verticale che quella orizzontale cui la AND è collegata. Si realizza così, ad esempio, un decodificatore complessivo 8:256.

Decodificatori composti

- Schema ad albero



Nello schema ad albero, alcuni segnali di ingresso sono collegati in maniera identica ai decodificatori più esterni (ingressi $\mathbf{a_2}$ ed $\mathbf{a_3}$ nell'esempio).

I restanti segnali di ingresso ($\mathbf{a_0}$ ed $\mathbf{a_1}$ nell'esempio) pilotano un ulteriore decodificatore, che a sua volta determina quale dei decodificatori esterni sia effettivamente attivo alzandone il corrispondente **segnale di abilitazione**

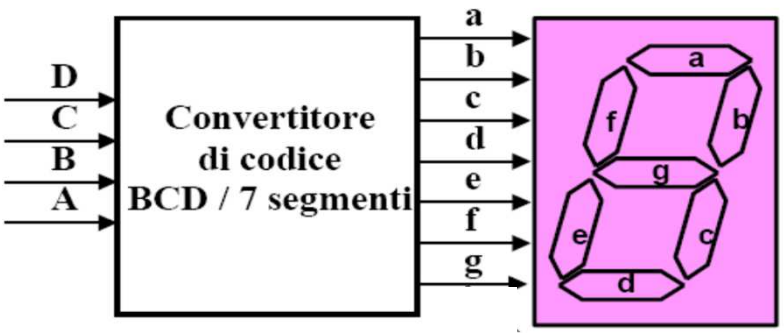
Nell'esempio, si ottiene complessivamente un decodificatore 4:16

Reti di transcodifica

Uno stesso dato può essere rappresentato mediante codici diversi

→ TRANSCODIFICATORE

- Macchina in grado di associare a ciascuna parola codice di C_1 la corrispondente parola codice di C_2
- Ad esempio:



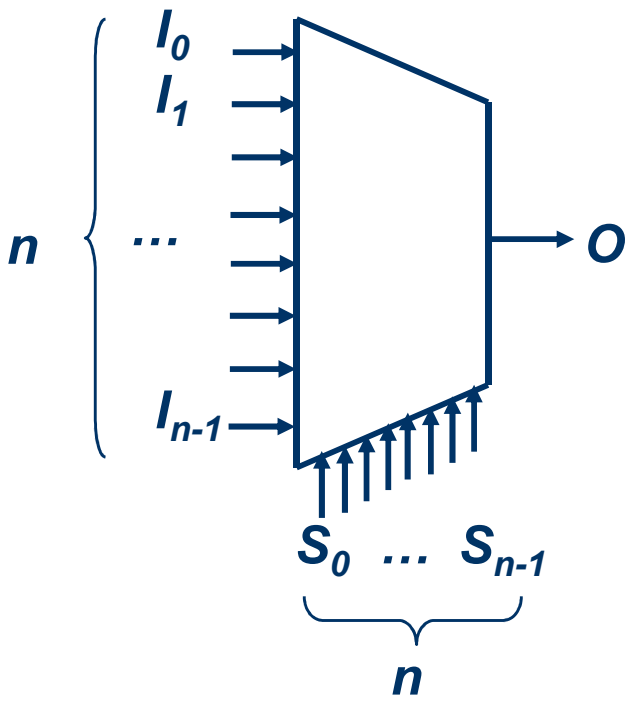
- la rete riceve in ingresso un codice decimale (ad esempio, a quattro bit 8-4-2-1) e fornisce in uscita un codice a 7 bit associato a ciascuna cifra decimale, che indica la corrispondente combinazione dei segmenti da illuminare

Multiplexer (Mux) lineare

- Multiplexer Binario lineare
  - ***n*** ingressi "dati" (primari) I_i
  - ***n*** ingressi "indirizzi" (secondari, o di selezione) S_i decodificati, quindi mutuamente esclusivi
  - una sola uscita ***O***, **uguale**

  all'ingresso dati I_i indicato dall'

  ingresso di selezione S_i attivo



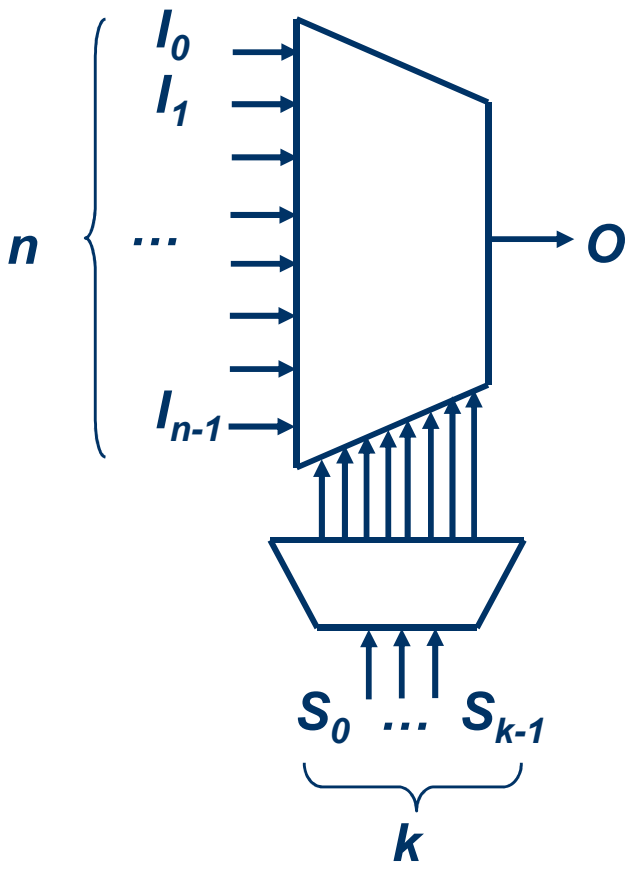
- Permette di scegliere (**selezionare**) quale degli ingressi I_i collegare all'uscita

$$O = \sum_{i=0}^{n-1} I_i S_i$$

Multiplexer (Mux) codificato

- Multiplexer Binario codificato
  - ***n*** ingressi “dati” (primari) I_i
  - $k = \log n$ ingressi “indirizzi” (secondari, o di selezione) S_i codificati
  - una sola uscita ***O***, uguale

  all’ingresso dati I_i dove ***i*** è codificato dall’ingresso indirizzi.



$$O = \sum_{i=0}^{n-1} I_i \cdot P_i(S_{n-1},\ldots,S_0)$$

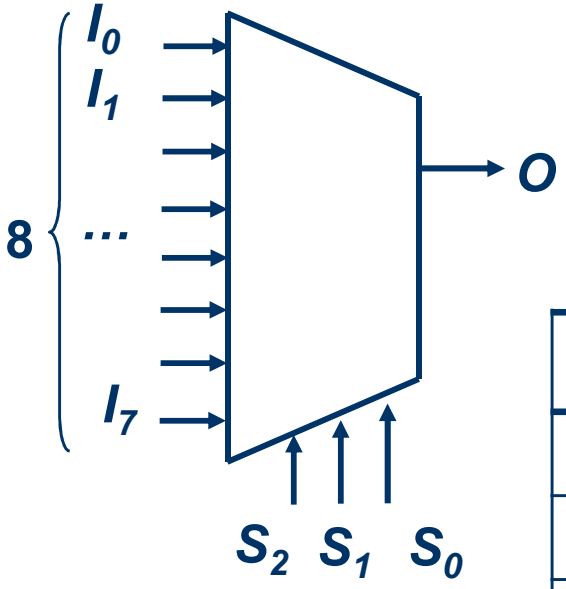
mintermine sulle ***k*** variabili S_i

Esempio

- Esempio:

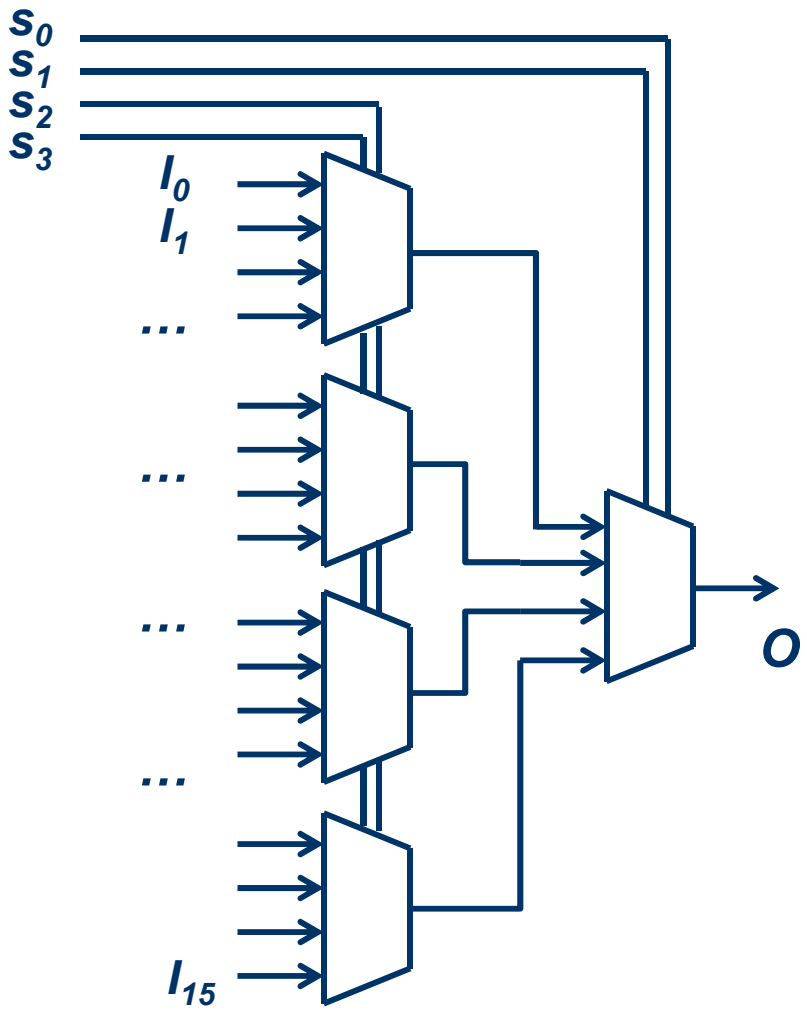
  MUX8 codificato

- A seconda di quale delle 8 possibili combinazioni è assunta dai tre segnali di selezione S_j, l'uscita è collegata ad uno degli **8** ingressi I_j, ovvero ne assume lo stesso valore



S_2	S_1	S_0	O
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Multiplexer composti

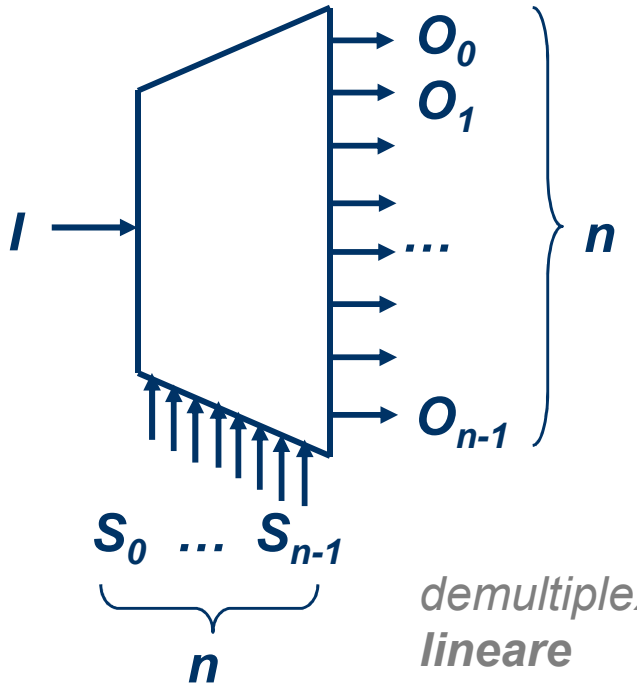


Esempio: Un **MUX16** può essere realizzato attraverso cinque **MUX4**.

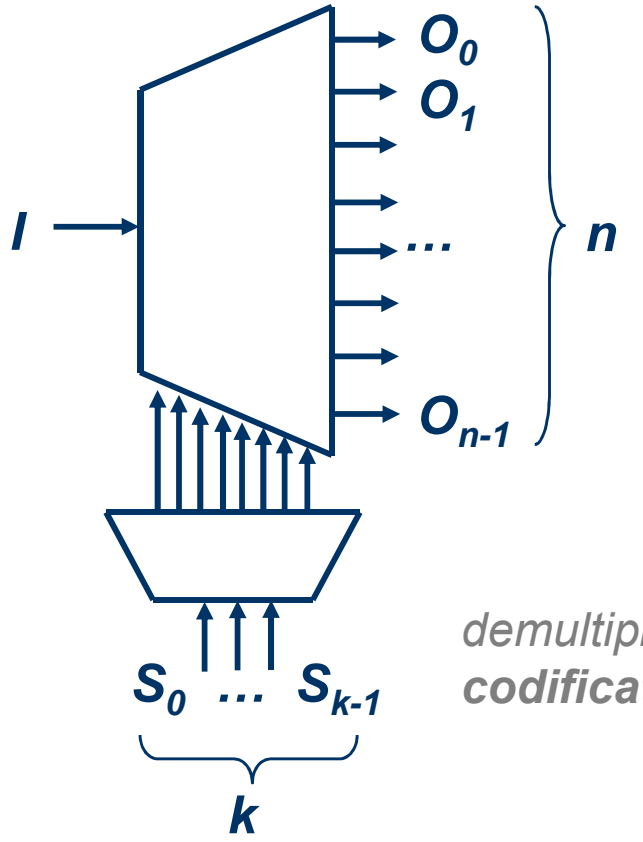
Due segnali di selezione (s_2 ed s_3 nell'esempio) sono applicati in parallelo ai quattro mux in ingresso, in maniera che ognuno scelga solo uno dei quattro segnali dati I_i (diversi per ciascun mux) che gli arrivano in ingresso. Gli altri due ingressi di selezione (s_0 ed s_1 nell'esempio) consentono di scegliere come uscita finale O una delle uscite dei quattro mux in ingresso.

Demultiplexer (demux)

- Consentono di collegare l'ingresso ***I*** ad una delle ***n*** uscite
- come i mux, possono essere ***lineari*** o ***codificati***
- è possibile costruire demux più grandi partendo da demux piccoli secondo uno schema ad albero, come per i mux



demultiplexer ***lineare***



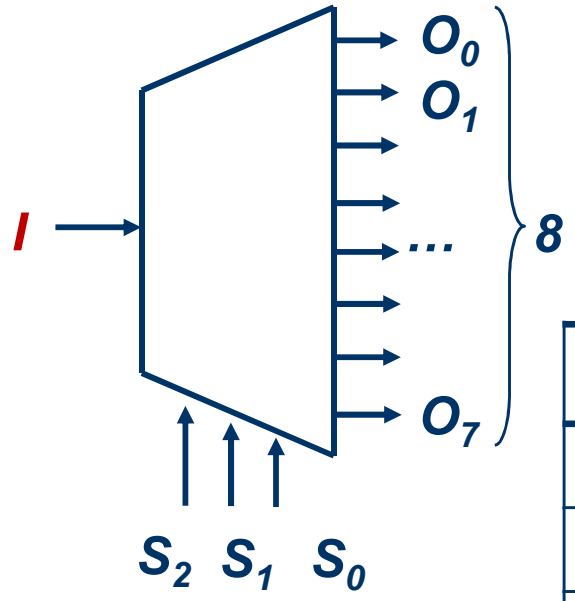
demultiplexer ***codificato***

esempio

- Esempio:

  DEMUX8 codificato

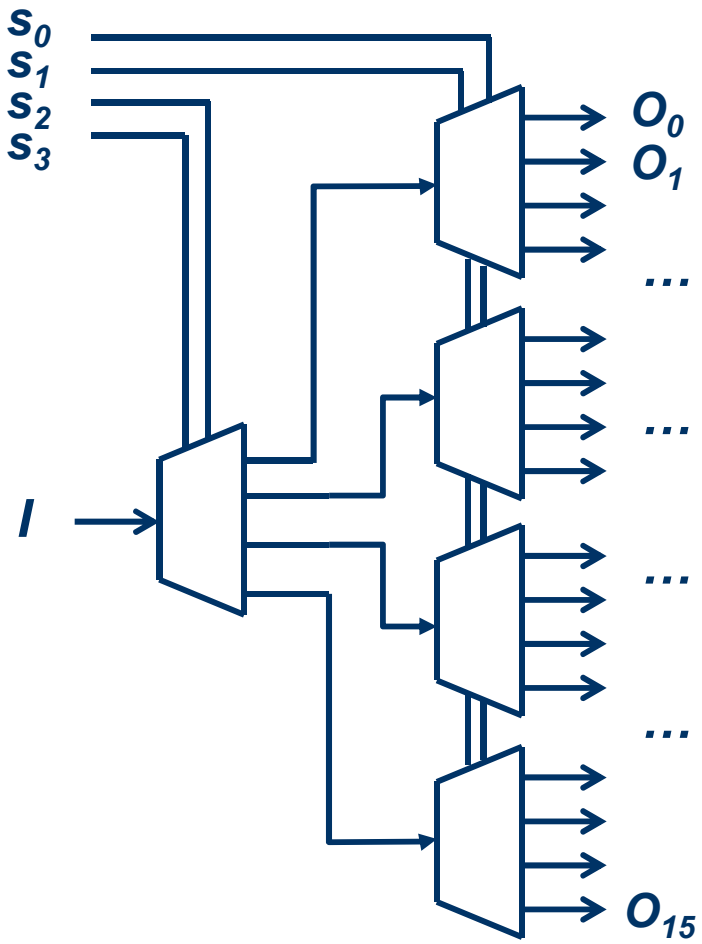
- Ciascuna uscita è sempre zero, tranne nel caso in cui il valore codificato dai tre segnali di selezione S_i sia pari al pedice j dell'uscita O_j. In questo caso, l'uscita è pari al valore dell'ingresso I



Esempio: uscita O_2

S_2	S_1	S_0	O_2
0	0	0	0
0	0	1	0
0	1	0	I
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Demultiplexer composti

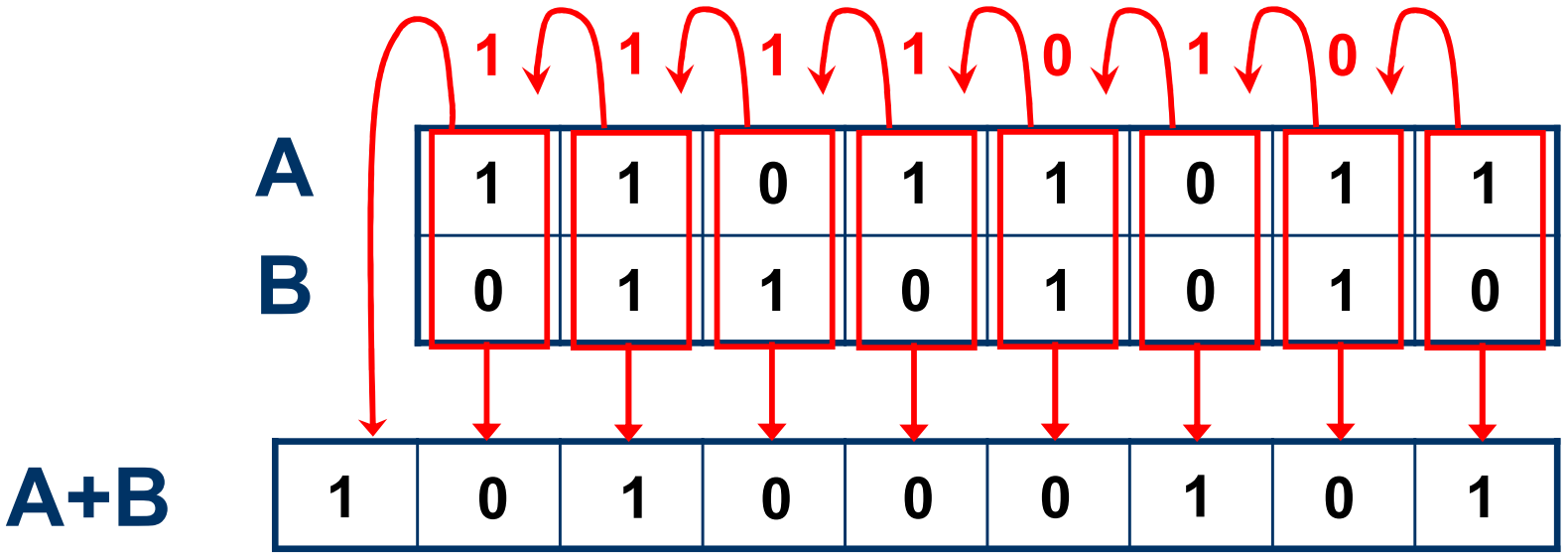


Esempio: Un **DEMUX16** può essere realizzato attraverso cinque **DEMUX4**.

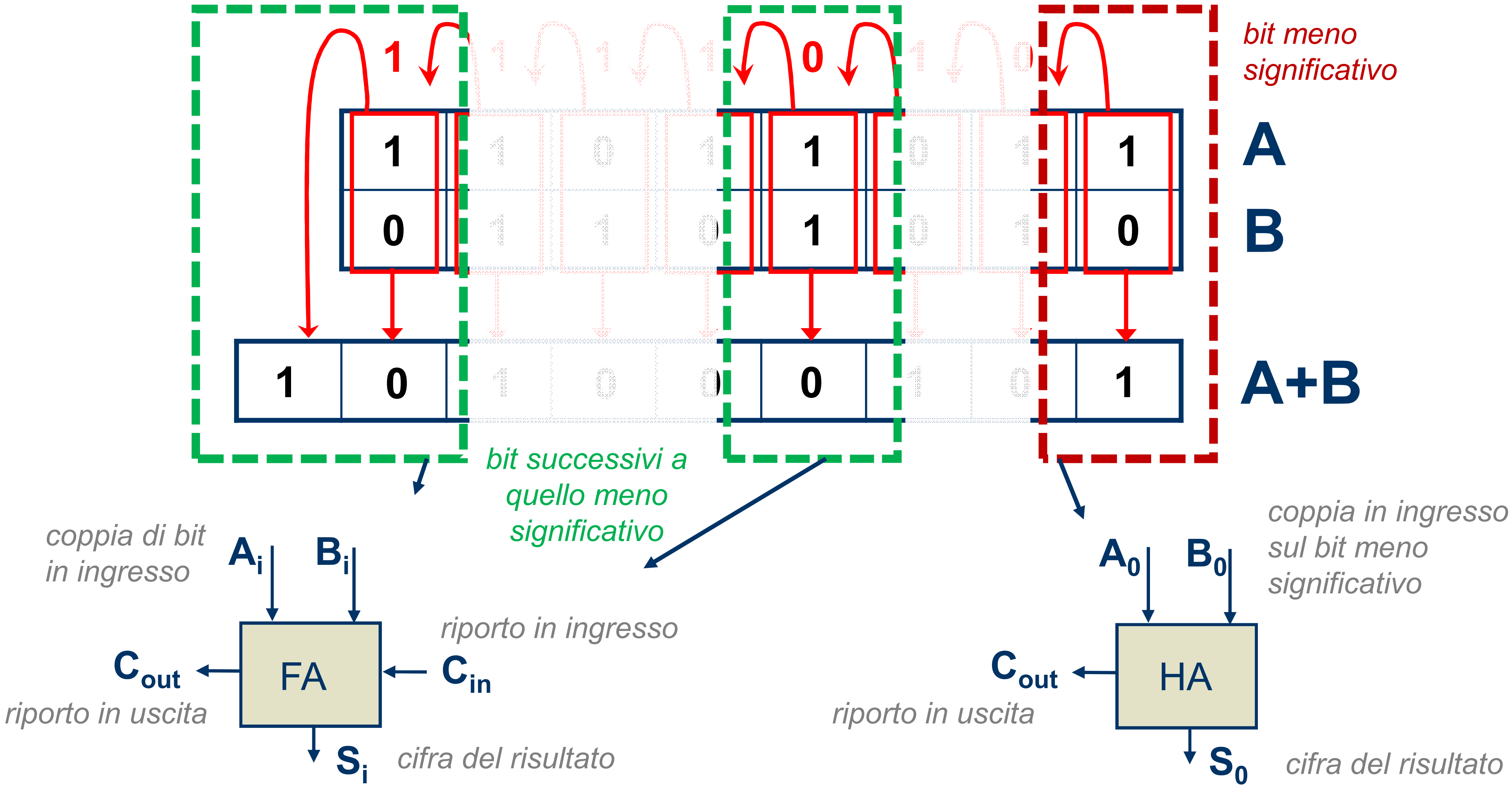
Due segnali di selezione (s_2 ed s_3 nell'esempio) sono applicati al demux in ingresso, che avrà tre uscite basse ed una uguale all'ingresso I. Gli altri due ingressi di selezione (s_0 ed s_1 nell'esempio) sono applicati in parallelo ai quattro demux di uscita. Le uscite esterne O_i saranno tutte basse, tranne quella corrispondente alla particolare combinazione di s_0/s_1 ed s_2/s presente in ingresso. Questa particolare uscita O_i sarà uguale all'ingresso I

Addizione

- Sfruttiamo la natura **iterativa** dell'operazione:
  - ogni coppia di bit in ingresso, più il riporto entrante da destra, possono essere processati da una diversa cella che realizza la stessa funzione

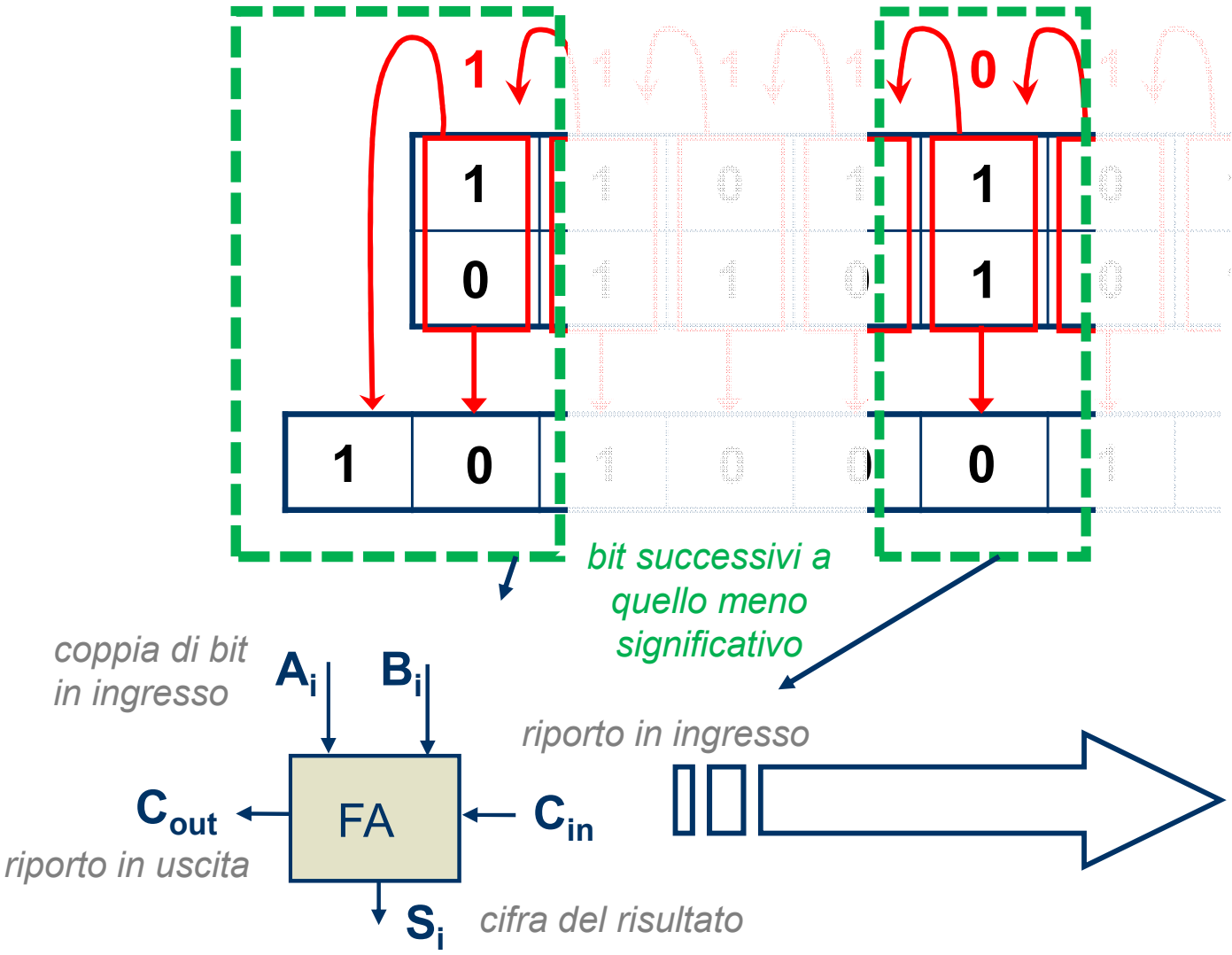


Addizione



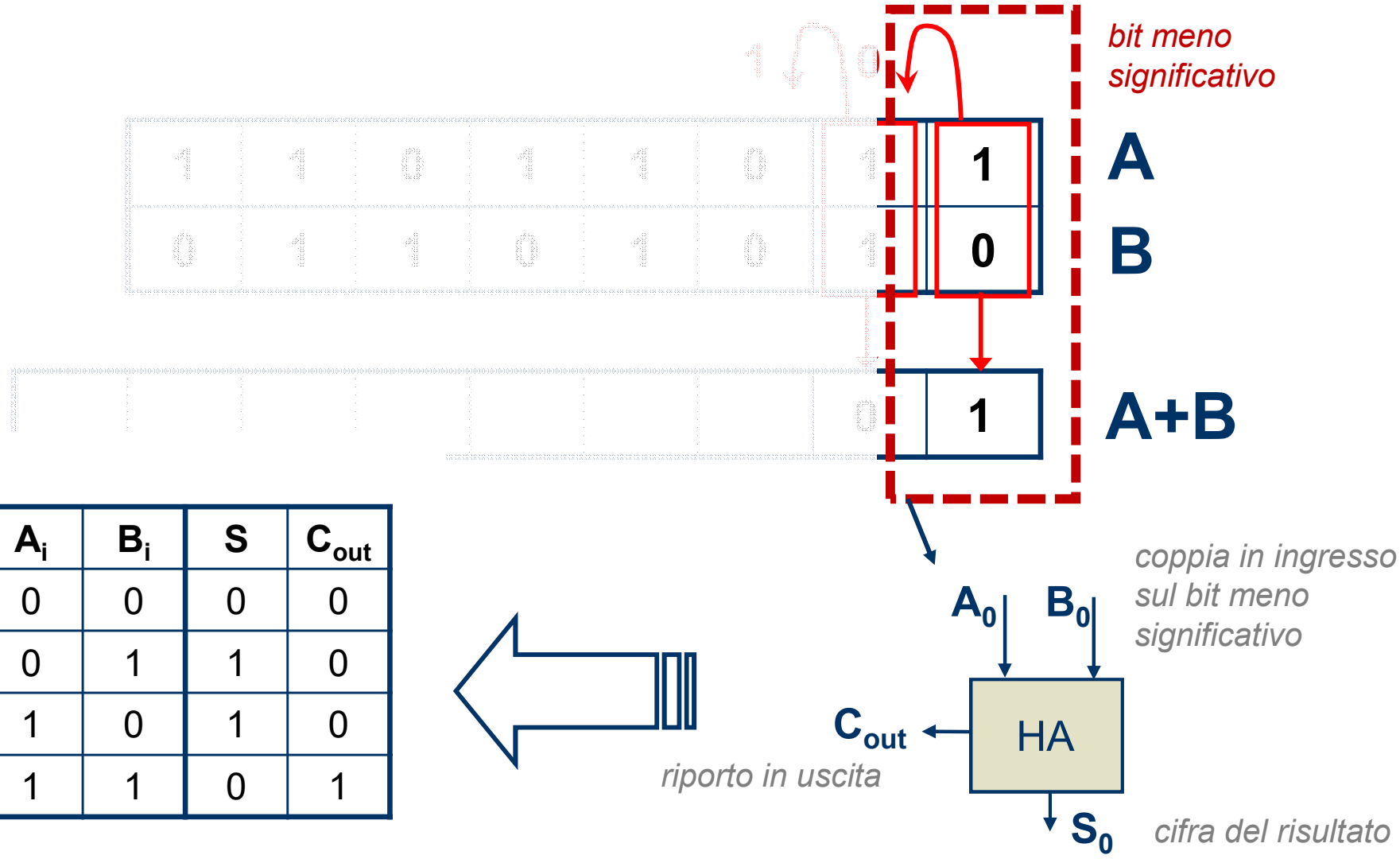
bit meno significativo
1
0
1
1
1
0
1
1
0
A
B
1
0
0
1
A+B
bit successivi a quello meno significativo
coppia di bit in ingresso
A_i
B_i
FA
C_{out}
riporto in uscita
C_{in}
riporto in ingresso
S_i
cifra del risultato
coppia in ingresso sul bit meno significativo
A_0
B_0
HA
C_{out}
riporto in uscita
S_0
cifra del risultato

Addizione



C_{in}	A_i	B_i	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Addizione


bit meno
significativo
1
0
1
A
B
A+B
A_i B_i S C_{out}
0 0 0 0
0 1 1 0
1 0 1 0
1 1 0 1
riporto in uscita
A_0
B_0
coppia in ingresso
sul bit meno
significativo
C_{out}
HA
S_0
cifra del risultato

Addizionatore binario

- Per il semiaddizionatore valgono le eguaglianze:

$$S = a \oplus b = D(a,b) = a\overline{b} + \overline{a}b$$

$$C_{out} = a \cdot b$$

- Per l'addizionatore completo valgono le eguaglianze:

$$S = a \oplus b \oplus C_{in} = D(a,b,C_{in}) = \overline{a}\,\overline{b}C_{in} + \overline{a}b\overline{C_{in}} + a\overline{b}\,\overline{C_{in}} + abC_{in}$$

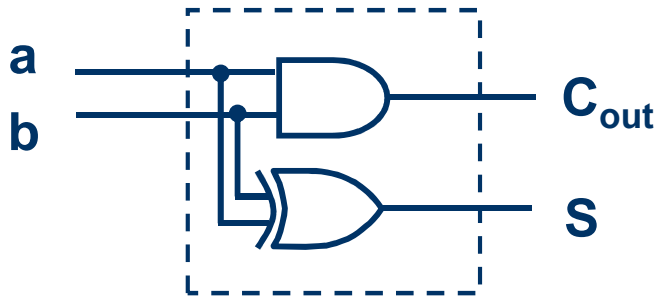
$$C_{out} = \overline{a}bC_{in} + a\overline{b}C_{in} + ab\overline{C_{in}} + abC_{in} = ab + bC_{in} + aC_{in} = ab + C_{in}(a+b)$$

Addizionatore binario

- Semiaddizionatore (Half-Adder):

$$S = a \oplus b = D(a,b) = a\bar{b} + \bar{a}b$$

$$C_{out} = a \cdot b$$



Addizionatore binario

- Addizionatore completo (Full-Adder):

  E' possibile esprimere le uscite in funzione di ***P*=(*a* ⊕ *b*)** e ***G* =*ab***
- E' facile verificare che:

$$C_{out} = ab + bC_{in} + aC_{in} = ab + abC_{in} + \bar{a}bC_{in} + abC_{in} + a\bar{b}C_{in} =$$

$$ab + \bar{a}bC_{in} + a\bar{b}C_{in} = ab + C_{in}(a \oplus b)$$

- Le equazioni del Full-Adder diventano quindi:

$$S = (a \oplus b) \oplus C_{in} = P \oplus C_{in}$$

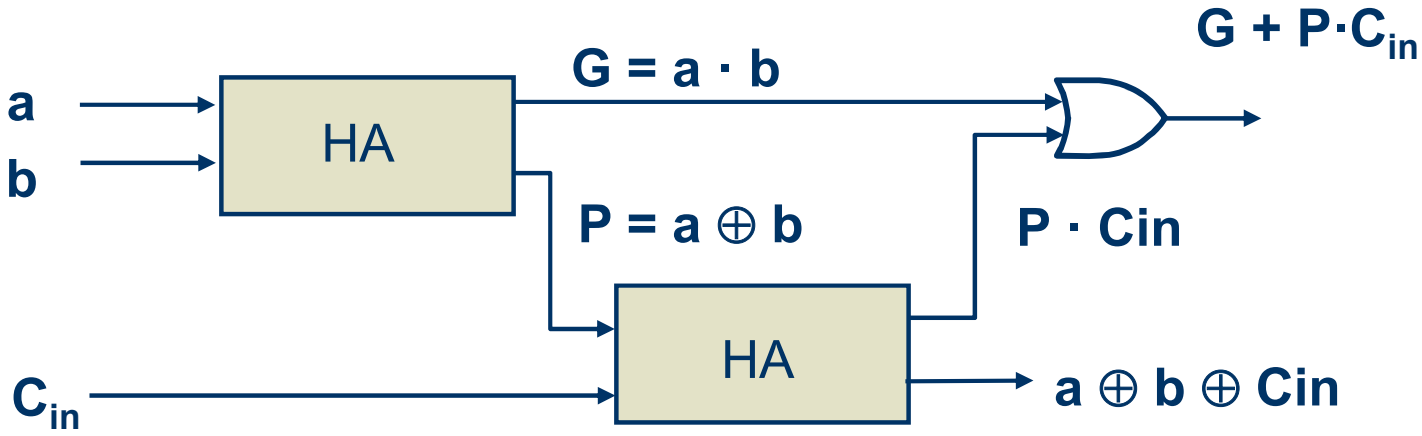
$$C_{out} = ab + C_{in}(a \oplus b) = G + C_{in}P$$

Addizionatore binario

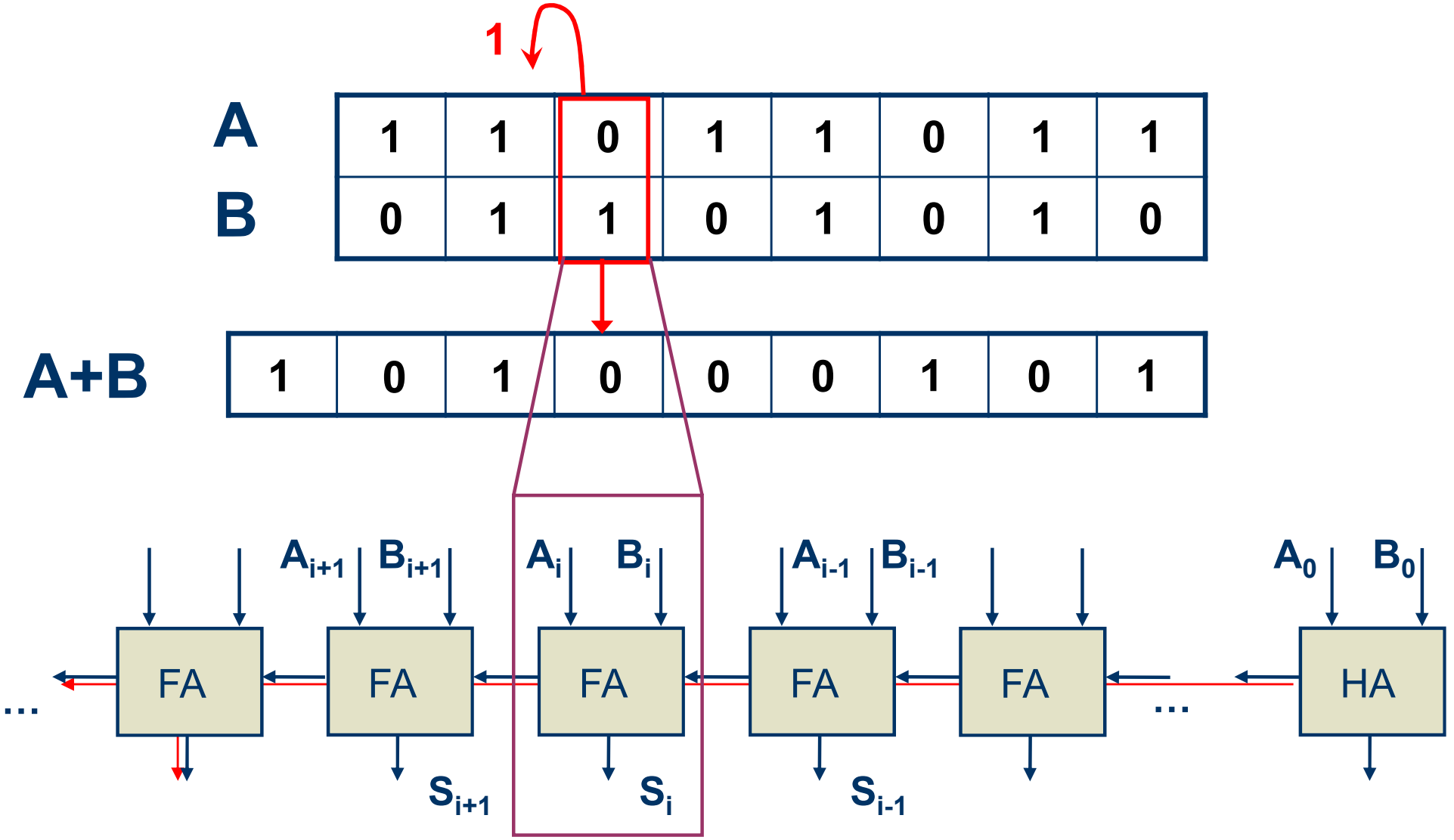
- Pertanto, un addizionatore completo può essere ottenuto a partire da due semiaddizionatori:

$$S = (a \oplus b) \oplus C_{in} = P \oplus C_{in}$$

$$C_{out} = ab + C_{in}(a \oplus b) = G + C_{in}P$$



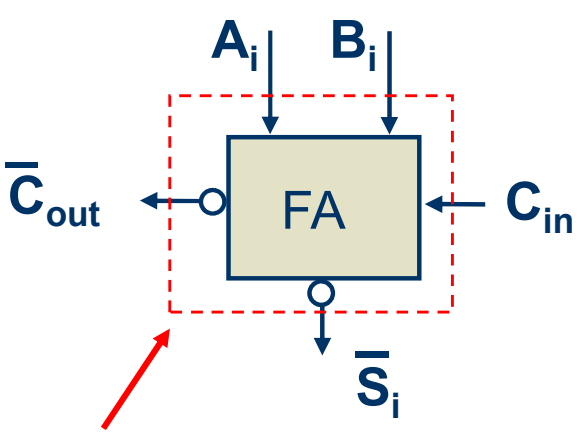
Addizionatore a propagazione



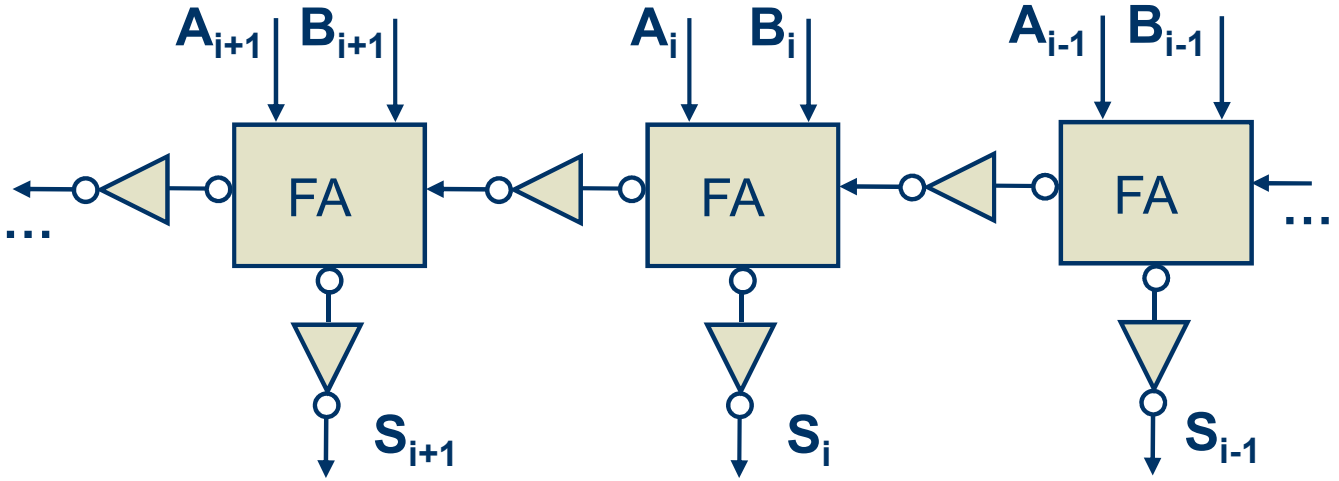
Addizionatore a propagazione

Materiale facoltativo

- In alcune tecnologie è più semplice realizzare circuiti che calcolino i negati delle funzioni ***S*** e C_{out}
- Occorrono quindi delle **NOT** aggiuntive per ottenere i valori corretti di somma e riporto



componente implementabile



Addizionatore a propagazione

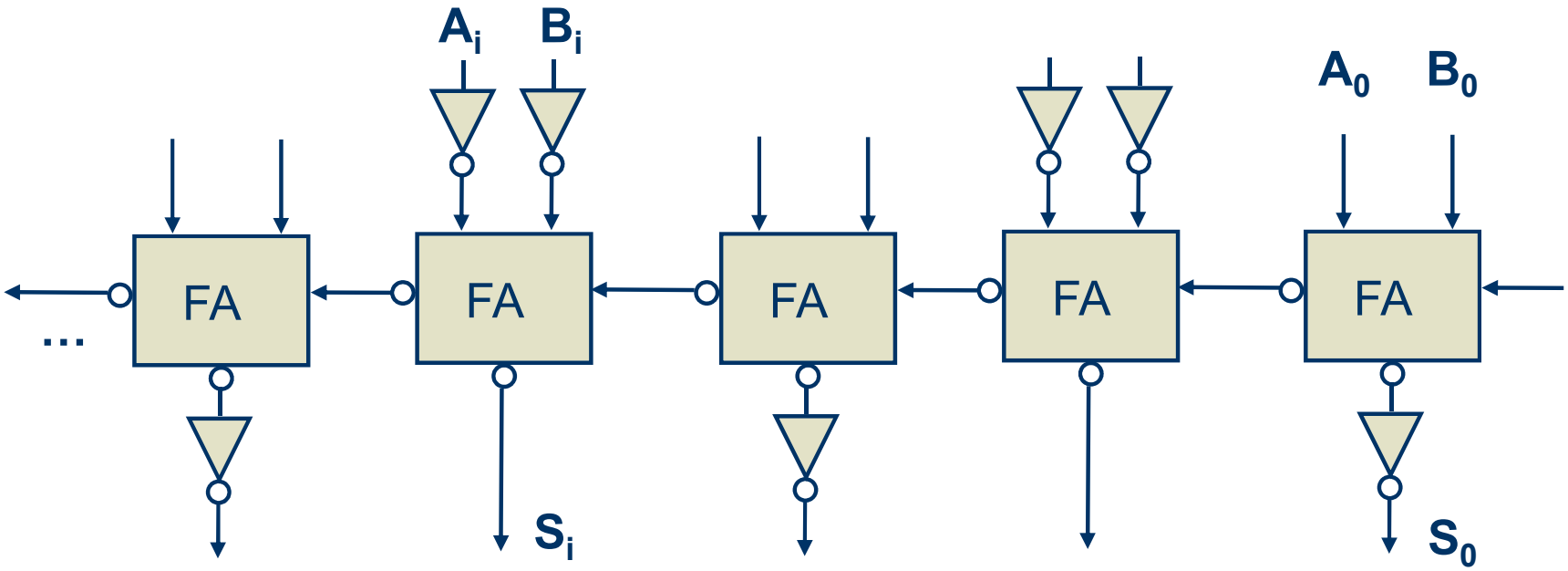
Materiale facoltativo

osservando che:

$$\overline{S} = \overline{S(a,b,C_{in})} = S(\overline{a},\overline{b},\overline{C_{in}})$$

$$\overline{C_{out}} = \overline{C_{out}(a,b,C_{in})} = C_{out}(\overline{a},\overline{b},\overline{C_{in}})$$

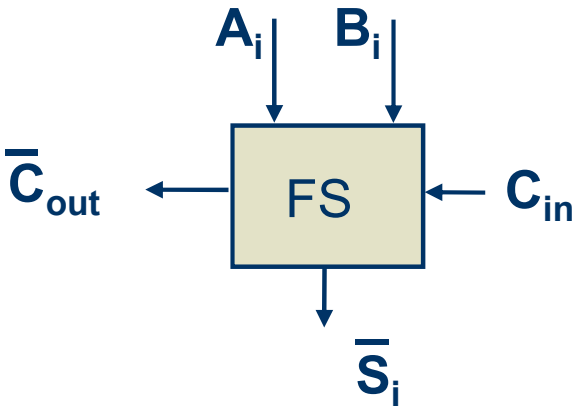
è possibile ottenere uno schema più veloce (meno porte lungo il cammino di propagazione del riporto):

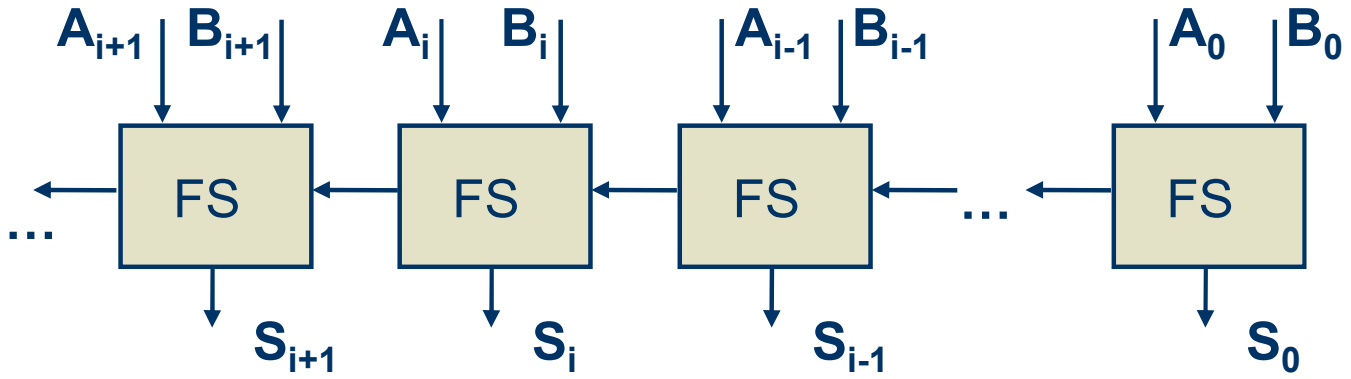


Sottrattore

Materiale facoltativo

- stessa idea usata nell'addizionatore
- questa volta si propaga un "*prestito*" invece del riporto
- il prestito si *sottrae* alla coppia di cifre correnti
- i bit del sottraendo sono negati: 0 → 0, 1 → -1

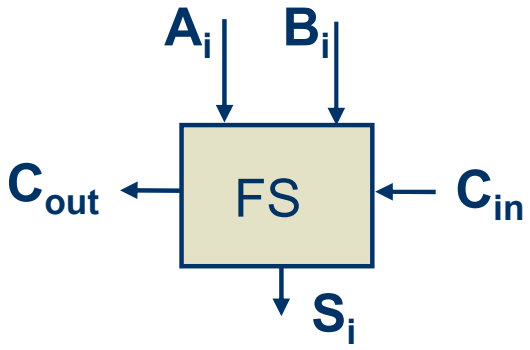




Sottrattore

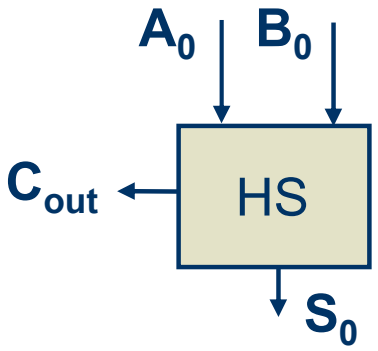
Materiale facoltativo

C_{in}	A_i	B_i	S	C_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1



S e C_{out} codificano la somma degli ingressi, considerando però C_{in} e B_i con peso **-1**, ed A_i con peso **1**

L'uscita ***S*** ha peso **1**, mentre il riporto uscente C_{out} ha, in questo caso, peso pari a **-2**



A_0	B_0	S	C_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Sottrattore

Materiale facoltativo

- Equazioni:

$$S = \bar{a}\bar{b}C_{in} + \bar{a}b\overline{C_{in}} + a\bar{b}\overline{C_{in}} + abC_{in} = a \oplus b \oplus C_{in}$$

$$C_{out} = \bar{a}\bar{b}C_{in} + \bar{a}b\overline{C_{in}} + \bar{a}bC_{in} + abC_{in} = \bar{a}b + \bar{a}C_{in} + bC_{in}$$

C_{in}	a	b	S	C_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

$$S = a\bar{b} + \bar{a}b = a \oplus b$$

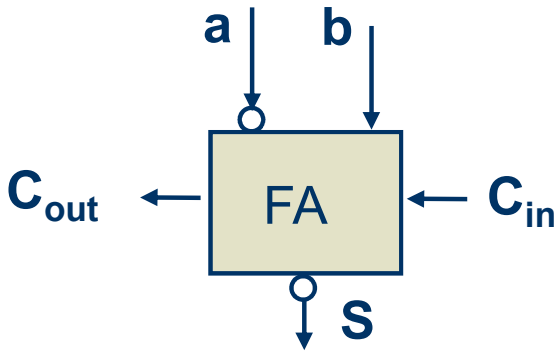
$$C_{out} = \bar{a}b$$

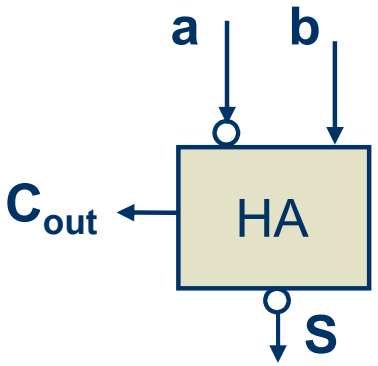
a	b	S	C_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Sottrattore

Materiale facoltativo

- Un sottrattore può quindi essere ottenuto a partire da un addizionatore negando opportunamente alcuni ingressi ed uscite, come mostrato in figura





$$\overline{S} = \overline{a} \oplus b \oplus C_{in}$$

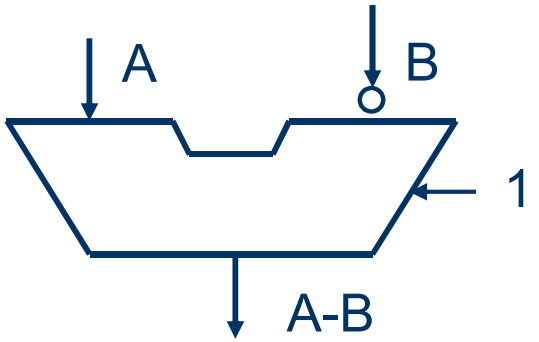
$$C_{out} = \overline{a}b + \overline{a}C_{in} + bC_{in}$$

$$\overline{S} = \overline{a} \oplus b$$

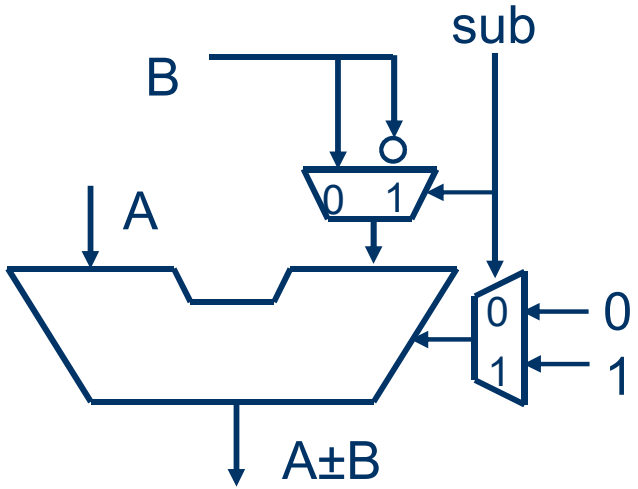
$$C_{out} = \overline{a}b$$

Addizionatori usati come sottrattori

- usando la rappresentazione in complementi a 2, un addizionatore può essere usato per eseguire la sottrazione:
  - A-B → A + (-B)



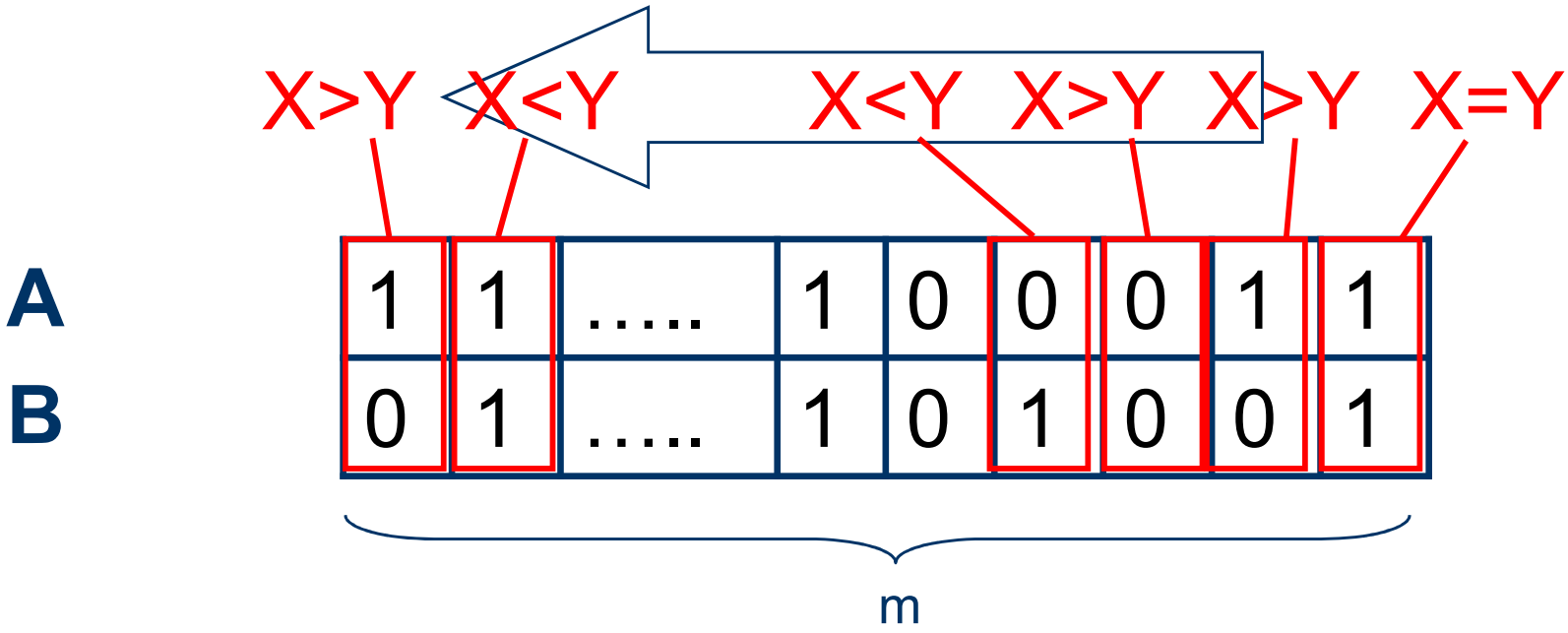
sottrattore



addizionatore/sottrattore programmabile:
sub=0 → addizione
sub=1 → sottrazione

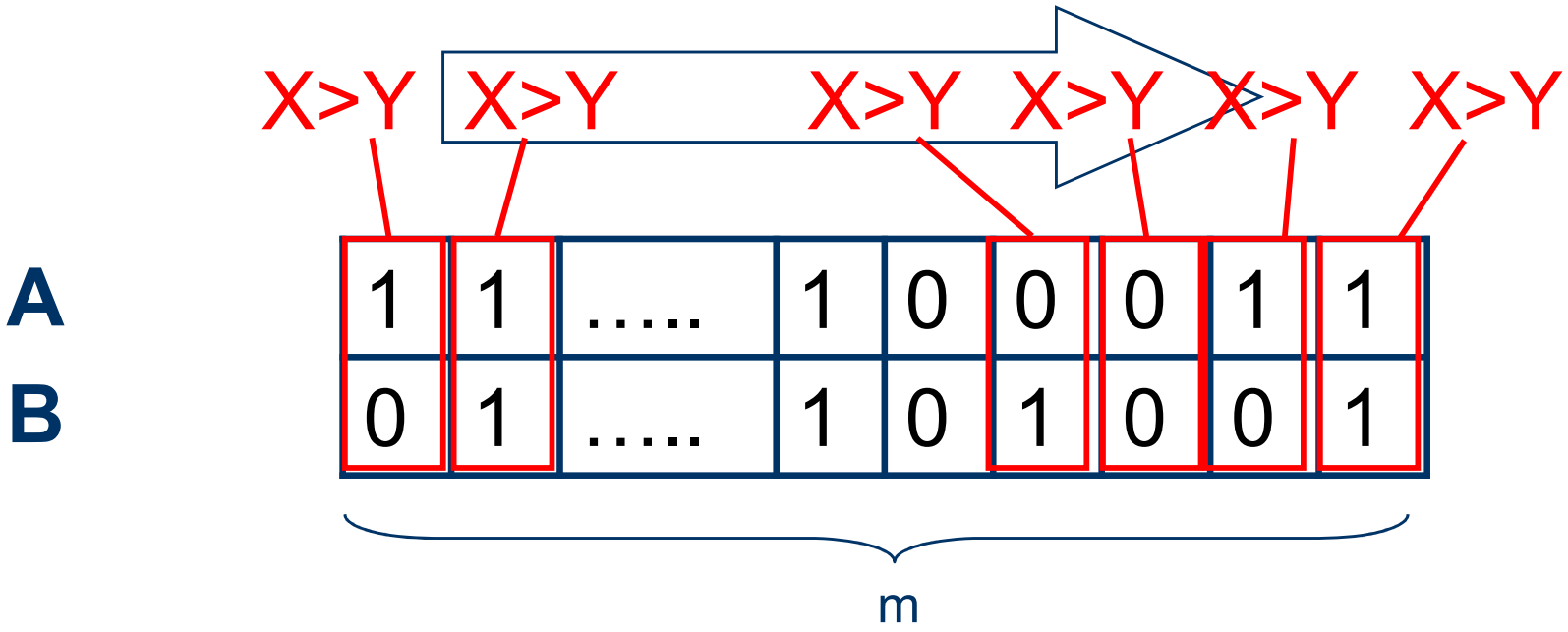
Comparatori

- confronto su numeri binari
- può essere effettuato sia da destra
  - l'ultima differenza trovata ci dice quale dei due numeri **A** e **B** è più grande…..



Comparatori

- …che da sinistra
  - la prima differenza trovata ci dice quale tra i due numeri **A** e **B** è più grande



Comparatori

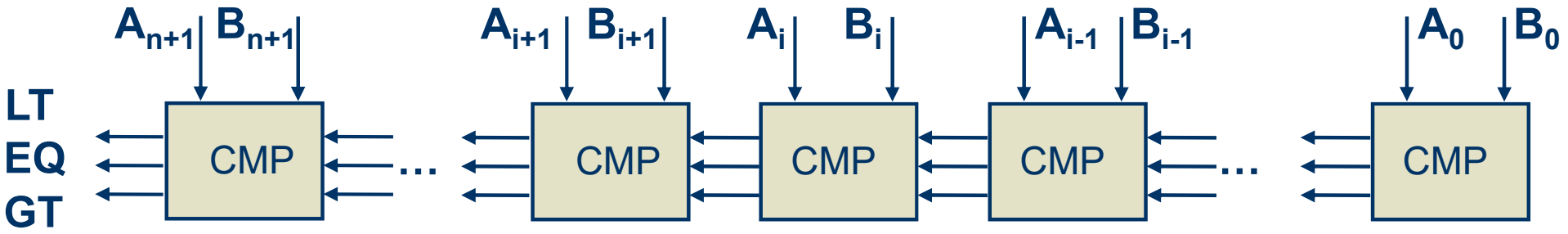
- In entrambi i casi, il confronto si può effettuare “localmente” sulle singole coppie di bit da confrontare x_i,y_i,,
- …tenendo però conto dell’esito del confronto parzialmente effettuato a destra o a sinistra

A	1	1	…..	1	0	0	0	1	1
B	0	1	…..	1	0	1	0	0	1

Comparatori

Materiale facoltativo

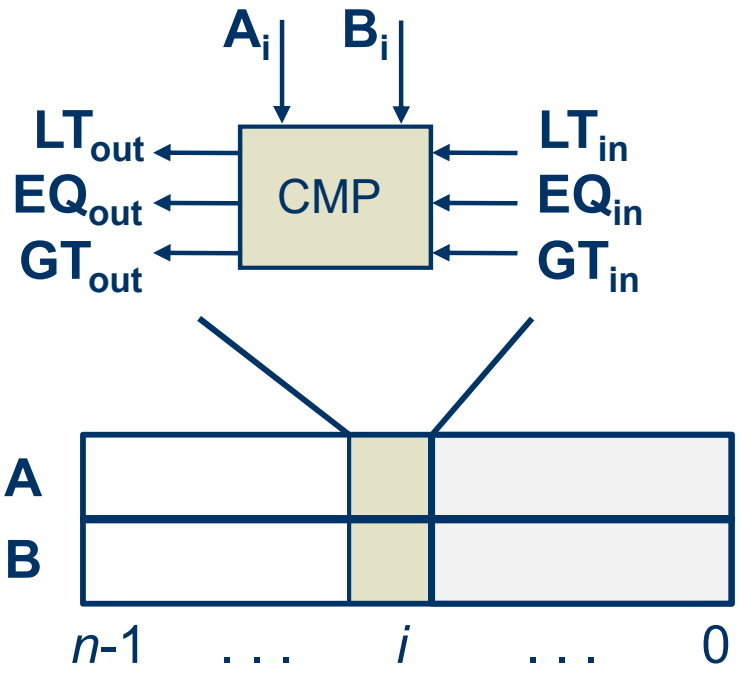
- Scegliamo, ad esempio, di progettare il comparatore con confronto da destra
- Come l'addizionatore, il comparatore è ottenuto collegando *n* componenti elementari tutti uguali
  - il *comparatore binario* (indicato con **CMP** nella figura)
- *ingressi*: **A**, **B**: numeri binari senza segno da confrontare
- *uscite*: **LT**: alto se A<B, **GT**: alto se A>B, **EQ** alto se A=B



Comparatore binario

Materiale facoltativo

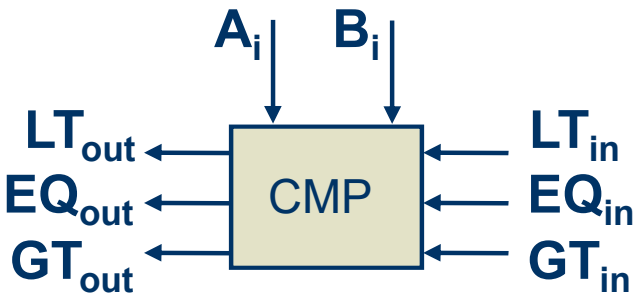
- $\mathbf{A_i}$ e $\mathbf{B_i}$: coppia *i*-esima di bit in ingresso
- $\mathbf{LT_{in}}$, $\mathbf{EQ_{in}}$, $\mathbf{GT_{in}}$: indicano l'esito del confronto parziale fatto a destra della coppia *i*-esima di bit (esclusa)
- $\mathbf{LT_{out}}$, $\mathbf{EQ_{out}}$, $\mathbf{GT_{out}}$: indicano l'esito del confronto parziale compresa la coppia *i*-esima di bit



Comparatore binario

Materiale facoltativo

- osservazione: $\mathbf{LT_{in}}$, $\mathbf{EQ_{in}}$, $\mathbf{GT_{in}}$: sono mutuamente esclusivi (se uno di essi è alto, gli altri due sono certamente bassi)



LT_{in}	A_i	B_i	LT_{out}	EQ_{out}	GT_{out}
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

EQ_{in}	A_i	B_i	LT_{out}	EQ_{out}	GT_{out}
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	1	0

GT_{in}	A_i	B_i	LT_{out}	EQ_{out}	GT_{out}
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	0	1

Tabelle di verità per le tre uscite del Comparatore Binario, corrispondenti a ciascuno dei tre possibili casi in ingresso (considerati separatamente)