Corso di Calcolatori Elettronici I
A.A. 2012-2013

Circuiti combinatori notevoli

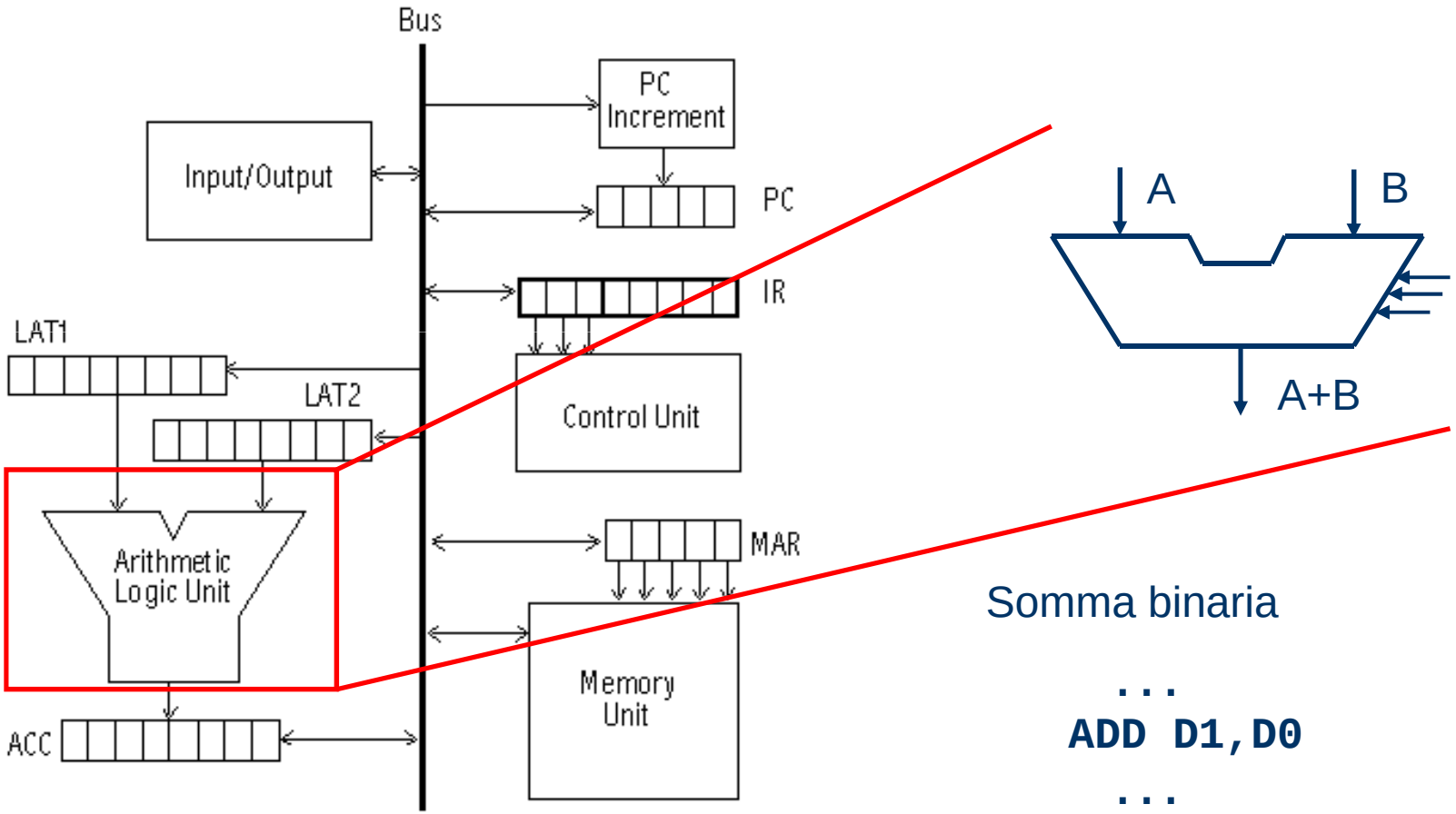
ing. Alessandro Cilardo

Accademia Aeronautica di Pozzuoli
Corso Pegaso V “GArn Elettronici”

Sommario

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

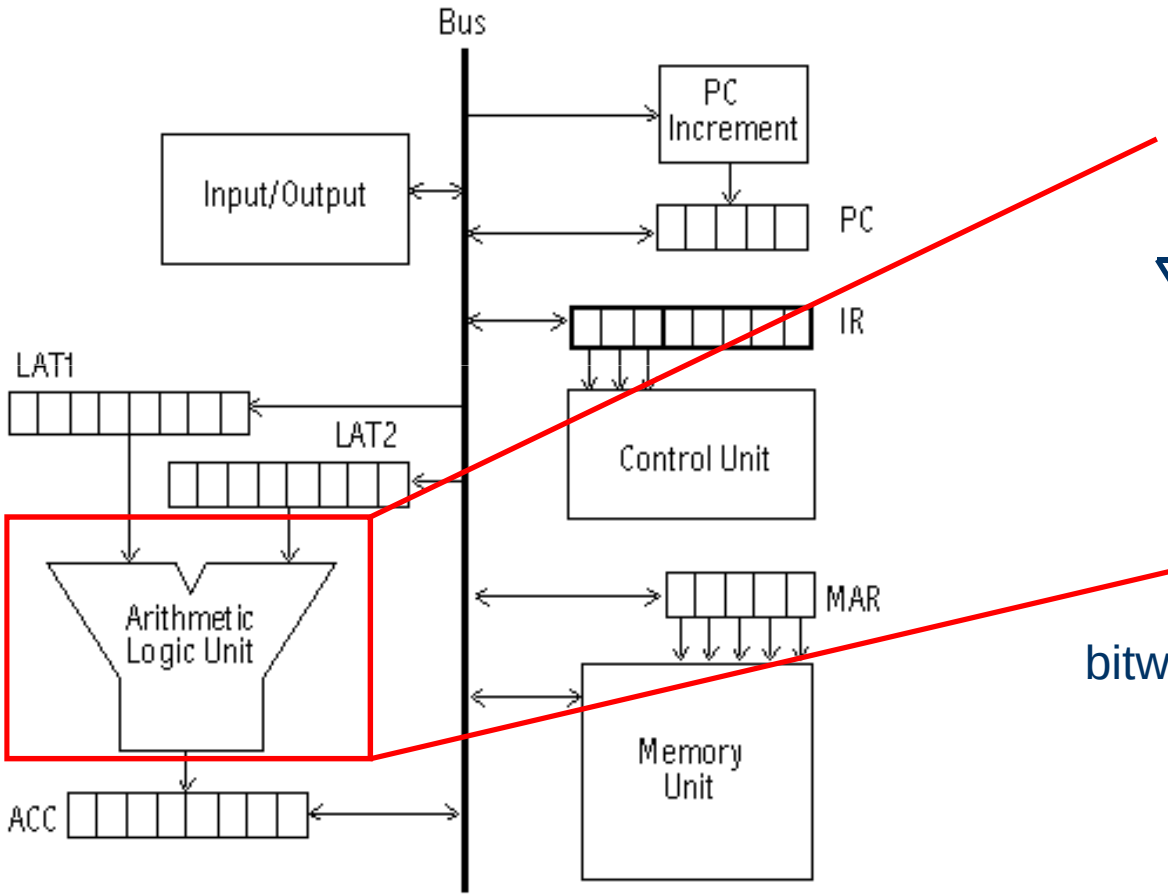
Esempi di circuiti combinatori

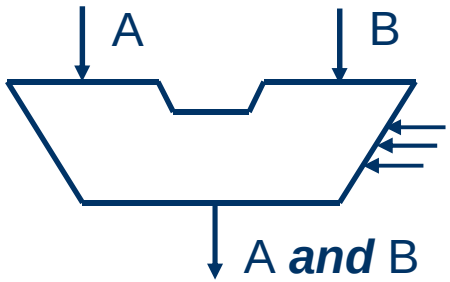


Somma binaria

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

Esempi di circuiti combinatori

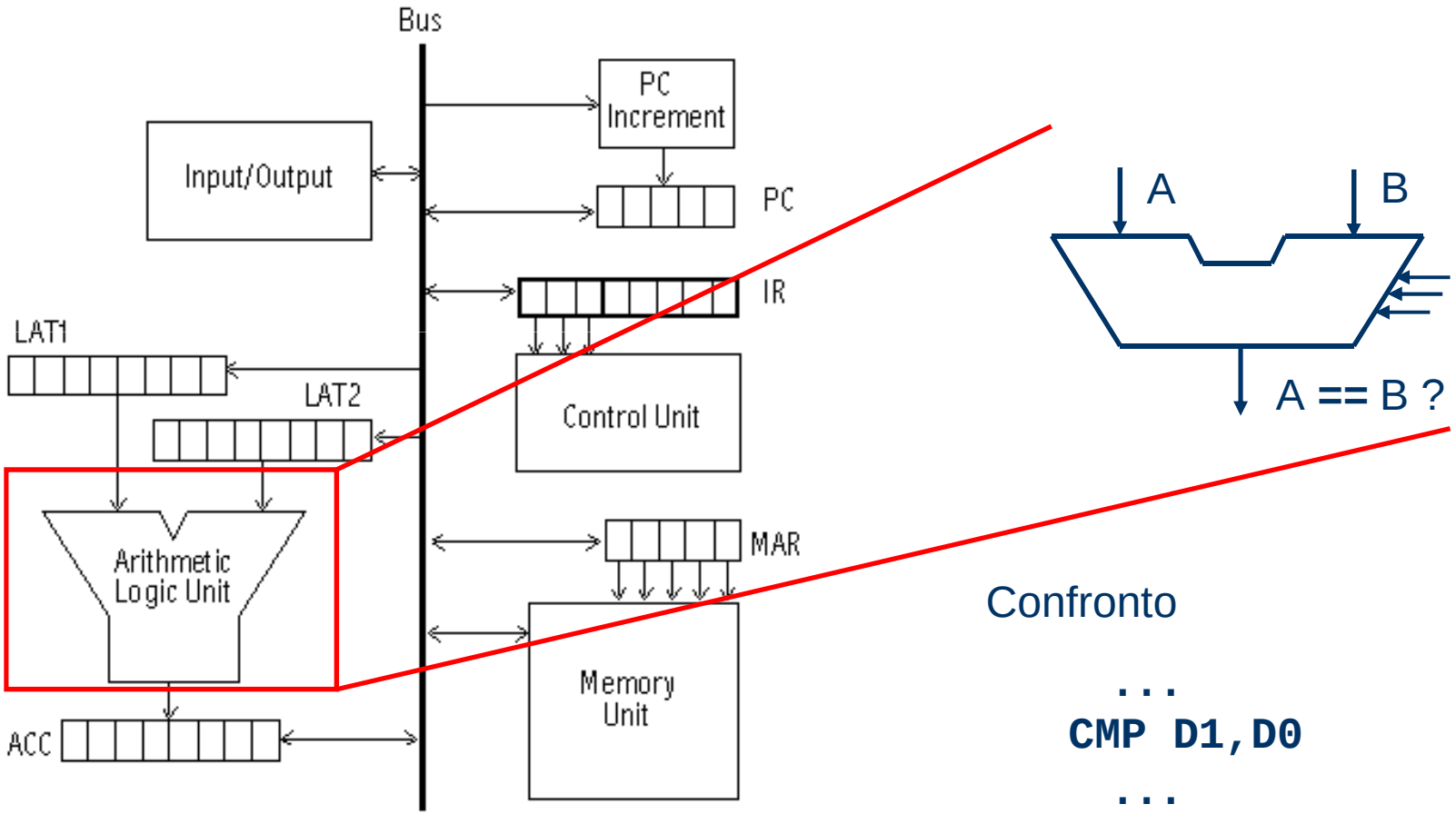




bitwise AND (AND *bit a bit*)

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

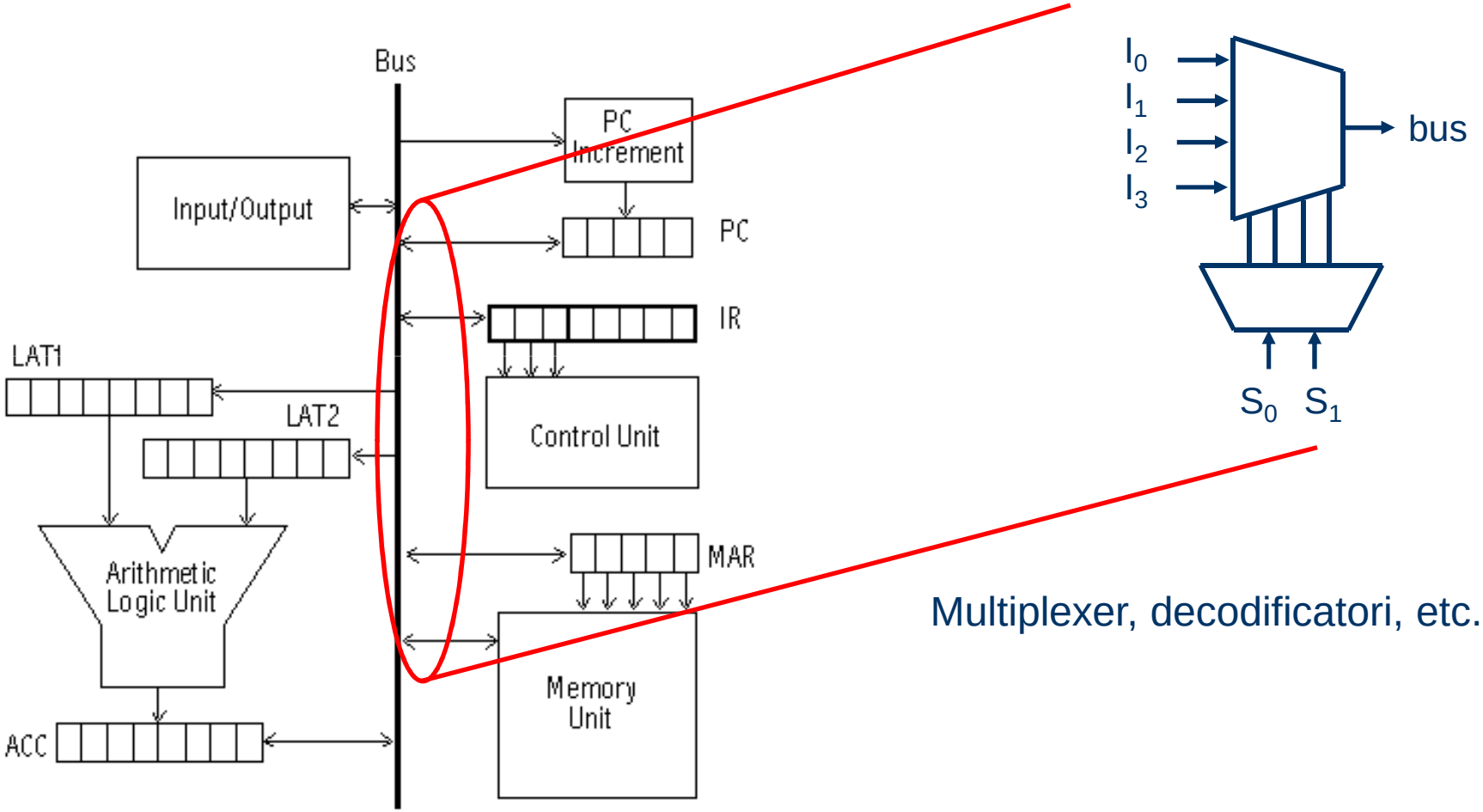
Esempi di circuiti combinatori



Confronto

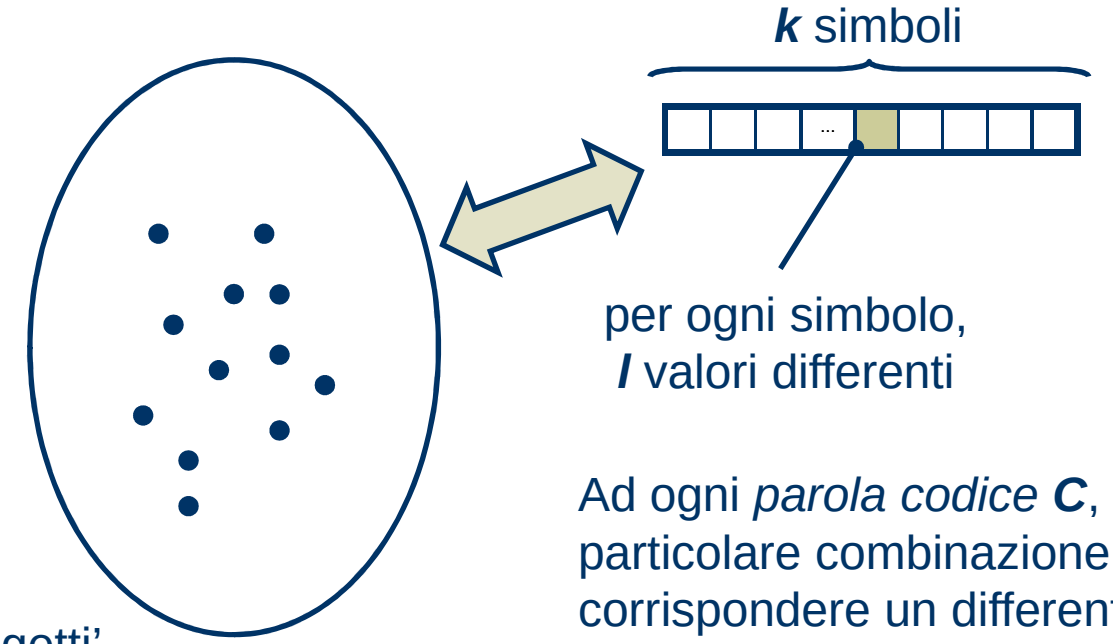
```
...
CMP D1,D0
...
```

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

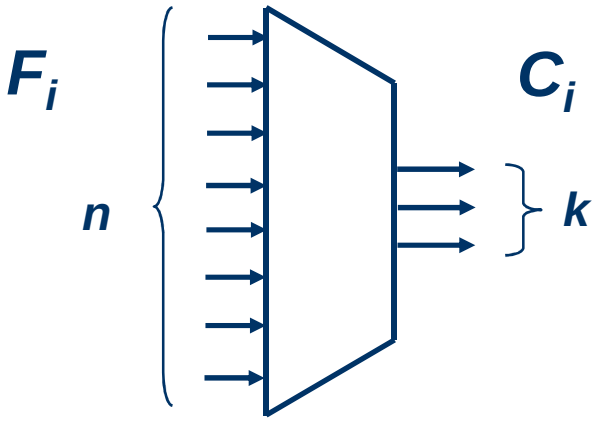
v	o_3	o_2	o_1	o_0	v	o_3	o_2	o_1	o_0
0	0	0	0	0	**8**	1	0	0	0
1	0	0	0	1	**9**	1	0	0	1
2	0	0	1	0	**A**	1	0	1	0
3	0	0	1	1	**B**	1	0	1	1
4	0	1	0	0	**C**	1	1	0	0
5	0	1	0	1	**D**	1	1	0	1
6	0	1	1	0	**E**	1	1	1	0
7	0	1	1	1	**F**	1	1	1	1

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

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

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

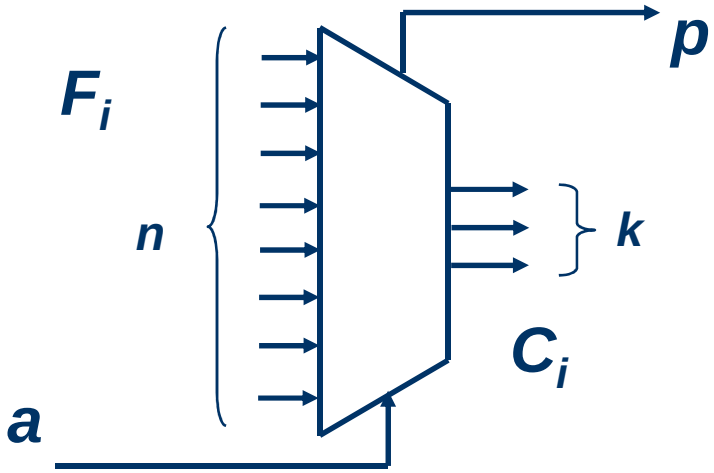
- Osservazione:
  - la condizione $F_0=1$, $F_i=0$ per ogni $i>0$ (corrispondente alla codifica in uscita '**00**…**0**' negli esempi precedenti) può essere confusa con il caso in cui *tutti* gli ingressi F_i sono zero
  - $F_i=0$ per ogni i (tutti gli ingressi pari a zero) non corrisponde a nessun valore 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

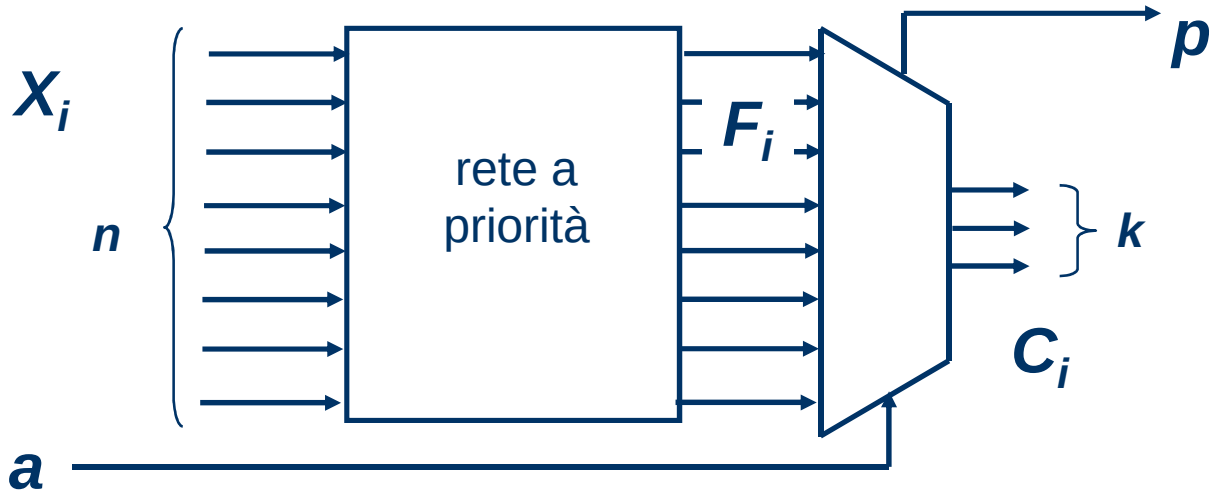
Reti di codifica

- Gli ingerssi 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

- 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

- ***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.

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

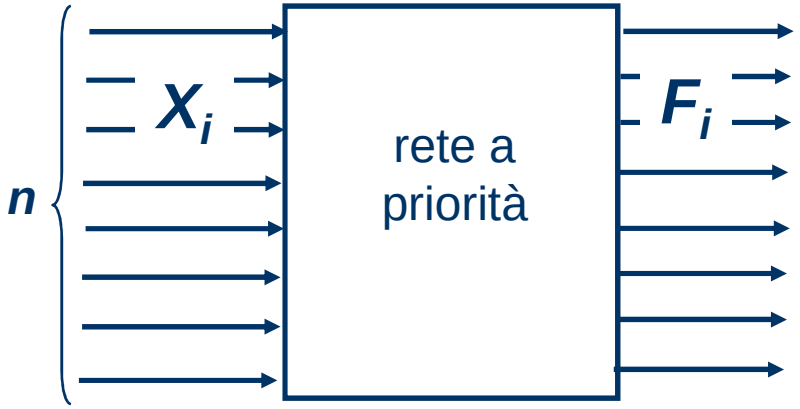
$$F_{n-2} = X_{n-2}\overline{X_{n-1}}$$

$$\ldots\ldots$$

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

$$\ldots\ldots$$

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



Reti di codifica a priorità

- 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

- L'equazione generale si semplifica quindi così:

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

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

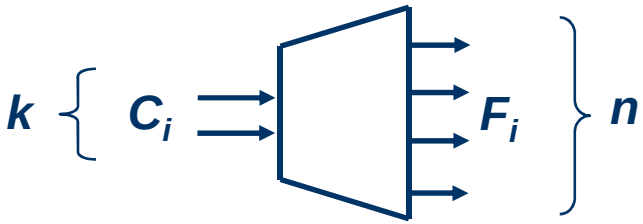
$$F_5 = X_5\overline{X_6} \qquad F_1 = X_1\overline{X_2}\,\overline{X_4}\,\overline{X_6}$$

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

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

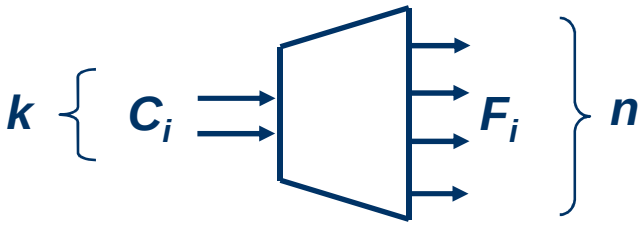
- Il processo di codifica è reversibile
- Decodifica
  - associa ad ogni parola codice rappresentata dagli ingressi C_i la sua *rappresentazione decodificata* F_i



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

Rete di decodifica

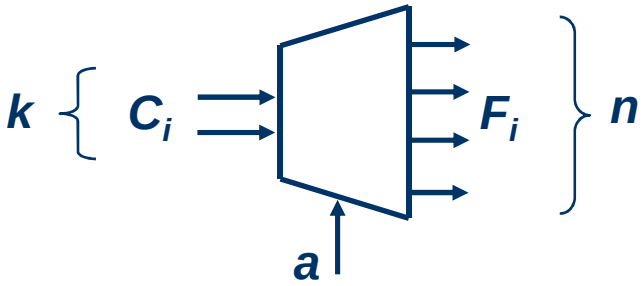
- ***k*** ingressi codificati C_i ed ***n*** uscite decodificate F_i
  - Ognuna delle uscite F_i può essere scritta come uno specifico *mintermine*, quello corrispondente al valore codificato in ingresso



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					

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

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

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

$$\ldots \quad \ldots \quad \ldots$$

- 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

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

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

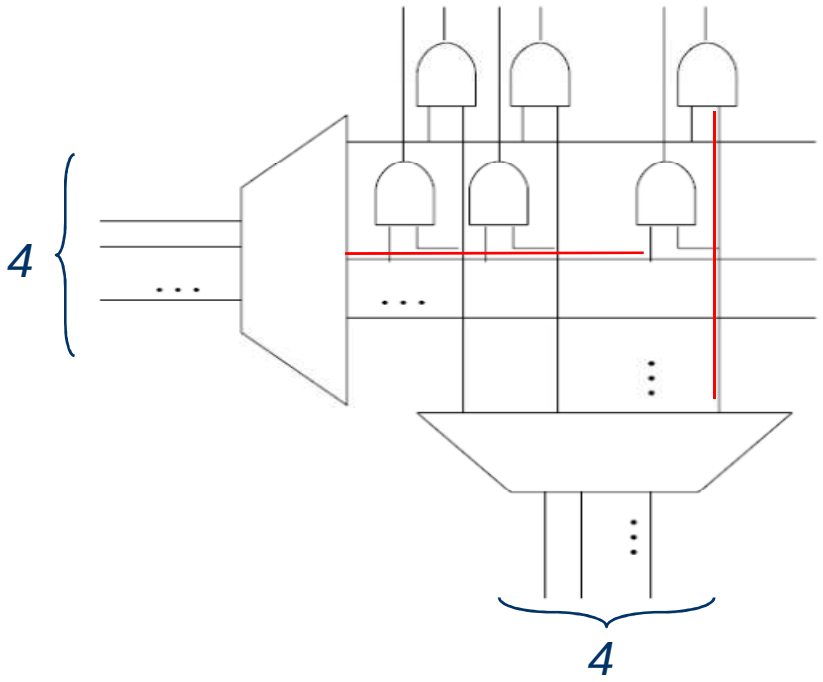
$$F_6 = C_2C_1\overline{C_0}; \quad F_7 = C_2C_1C_0; \quad F_8 = C_3\overline{C_0}; \quad F_9 = C_3C_0$$

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)

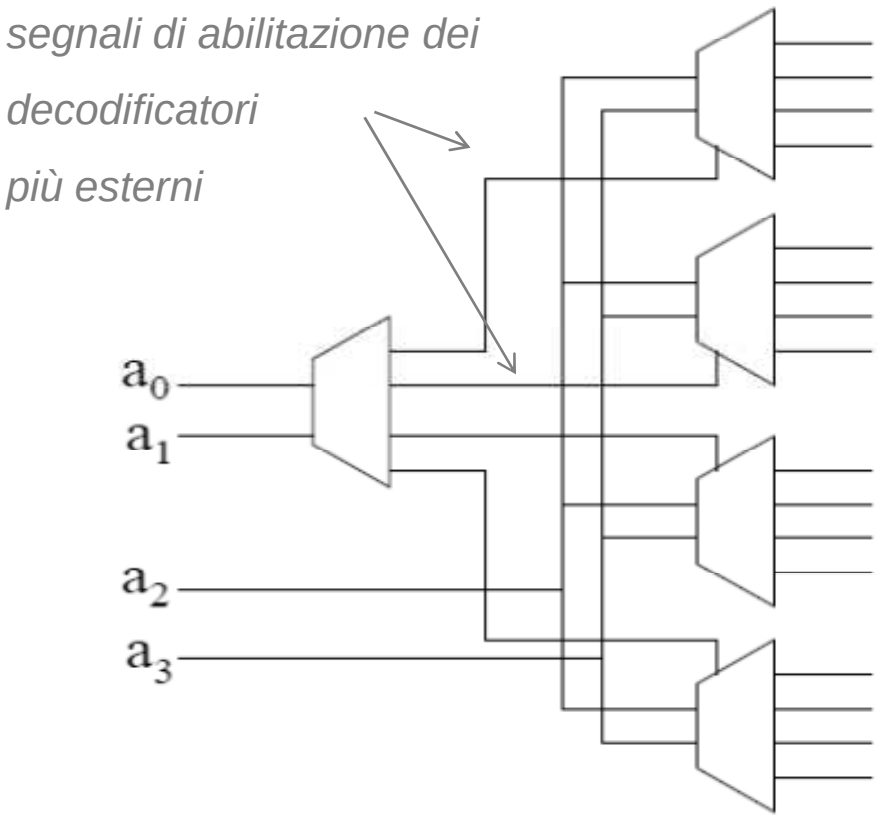


Gli ingressi (supponiamo ad esempio di averne 8 in tutto) sono divisi in due gruppi. Ciascun gruppo entra in uno dei due decodificatori (ad esempio, decod. 4:16).

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**

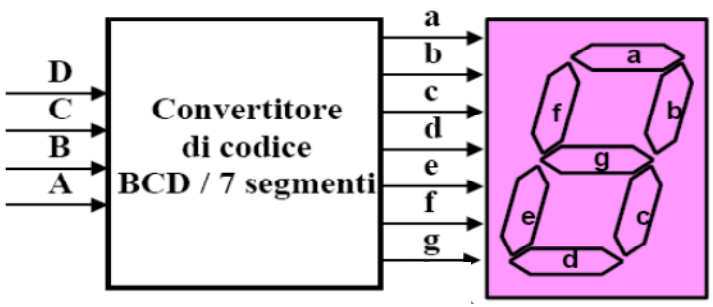
Nell'esempio, si ottiene complessivamente un decodificatore 4:16

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

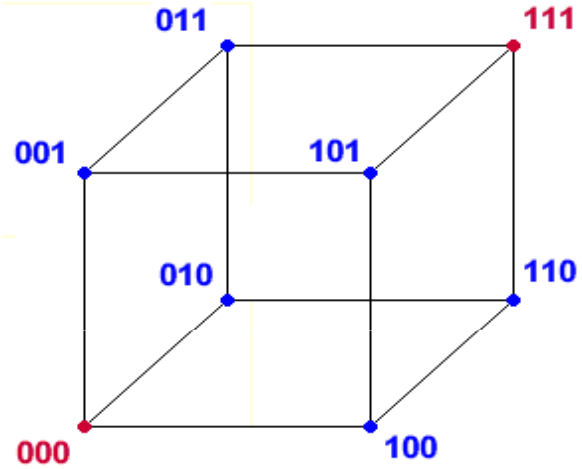
Protezione dagli errori

- 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

- 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

- 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

Funzioni di parità/disparità

- Il controllo di errore sulle uscite può essere effettuato tramite una funzione di parità/disparità

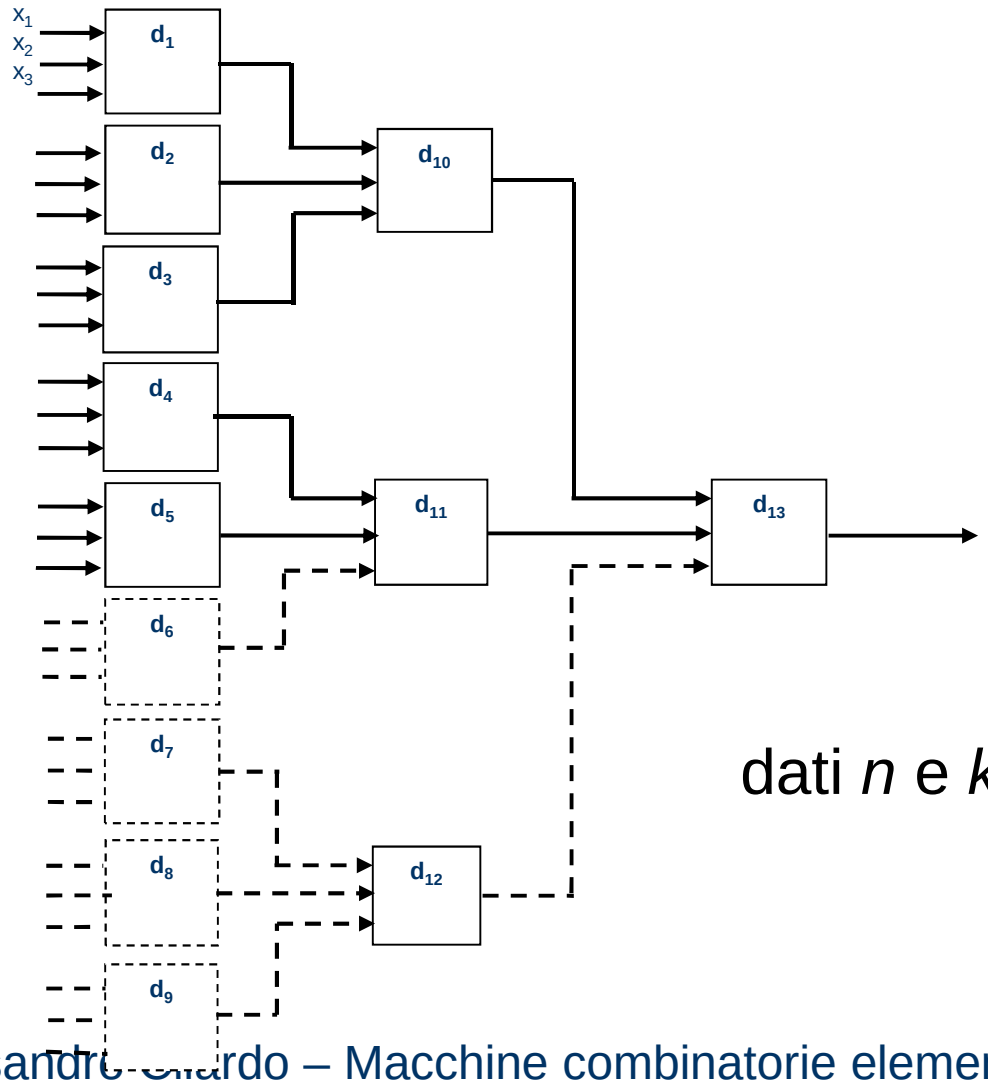
$$p(X) = \overline{d(X)}$$

- 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

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*:

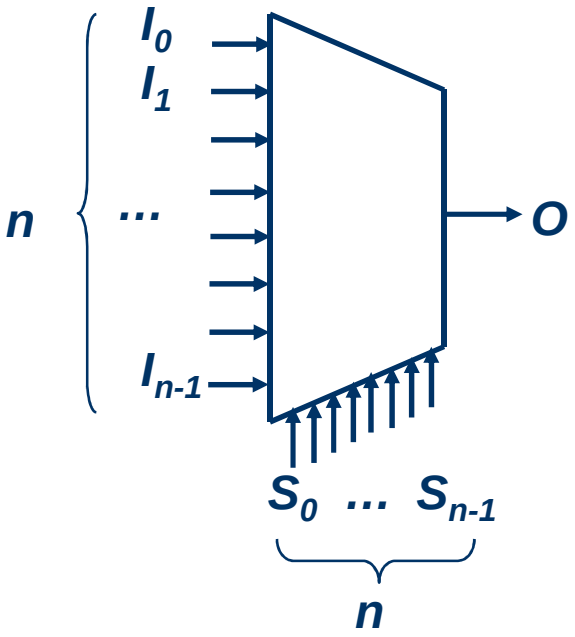
$$l = \lceil \log_k n \rceil$$

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

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



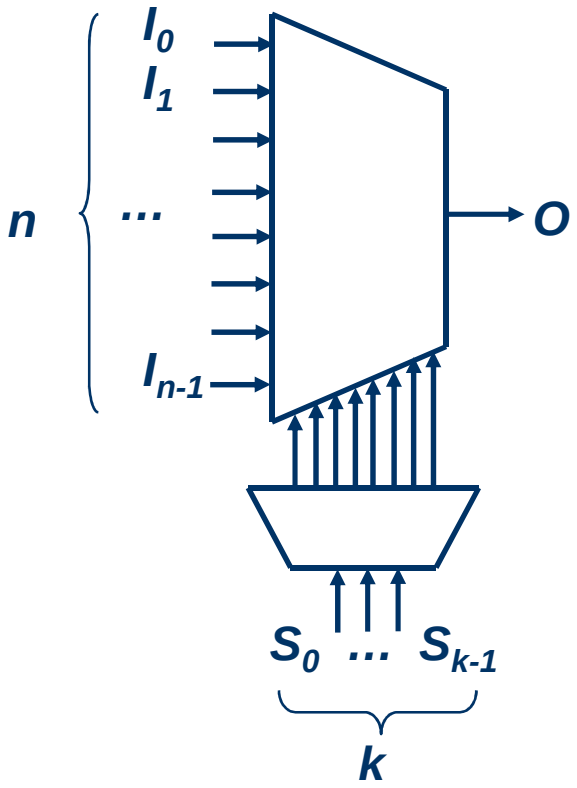
- Permette di scegliere (**selezionare**) quale degli ingressi I_i collegare all’uscita

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

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.

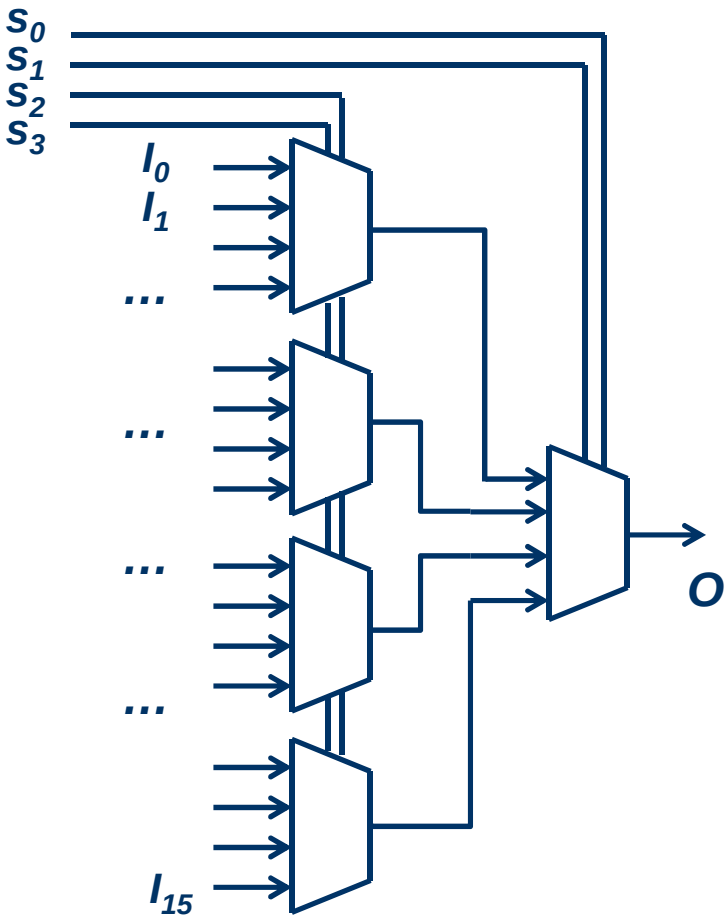


$$O = \sum_{i=0}^{n-1} I_i \cdot P_i(S_{n-1},\ldots,S_0)$$

mintermine sulle ***k*** variabili S_i

Multiplexer composti

- Realizzazione di un MUX16 attraverso MUX4

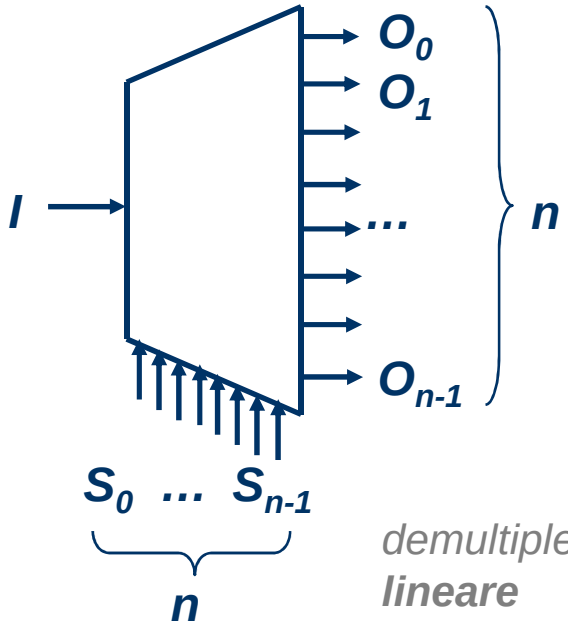


Un **MUX16** può essere realizzato attraverso cinque **MUX4**.

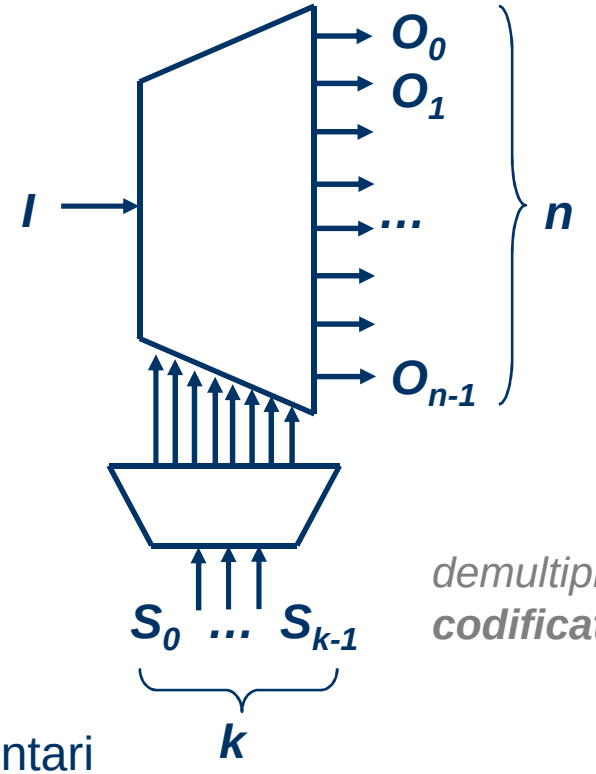
Due segnali di selezione (***s*****$_2$** ed ***s*****$_3$** nell'esempio) sono applicati in parallelo ai quattro mux in ingresso, in maniera che ognuno scelga solo uno dei quattro segnali dati ***I*****$_i$** (diversi per ciascun mux) che gli arrivano in ingresso. Gli altri due ingressi di selezione (***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



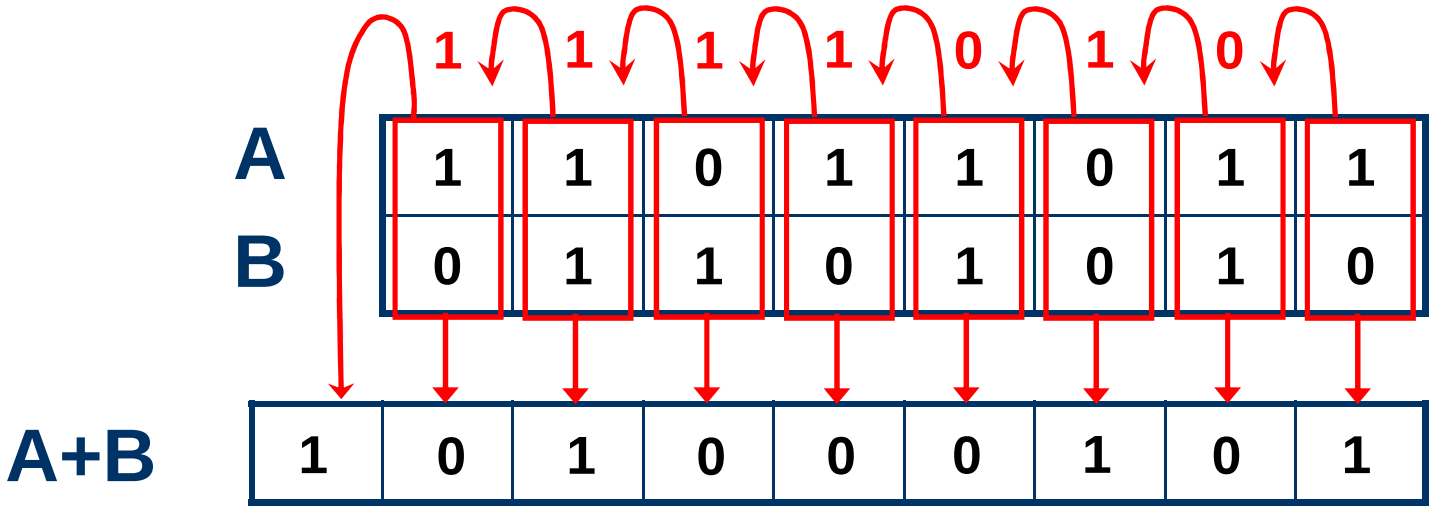
demultiplexer ***lineare***



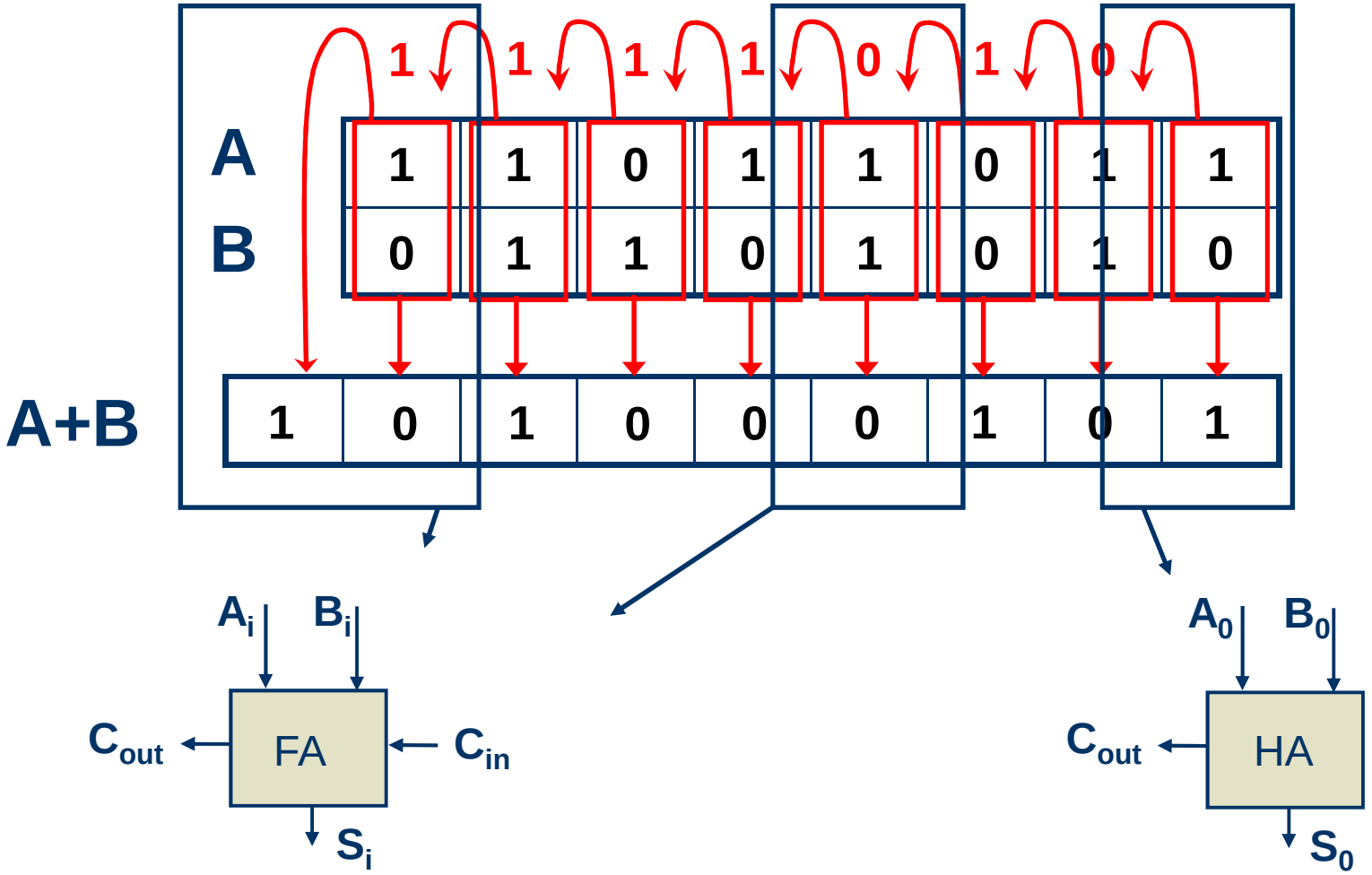
demultiplexer ***codificato***

Addizione

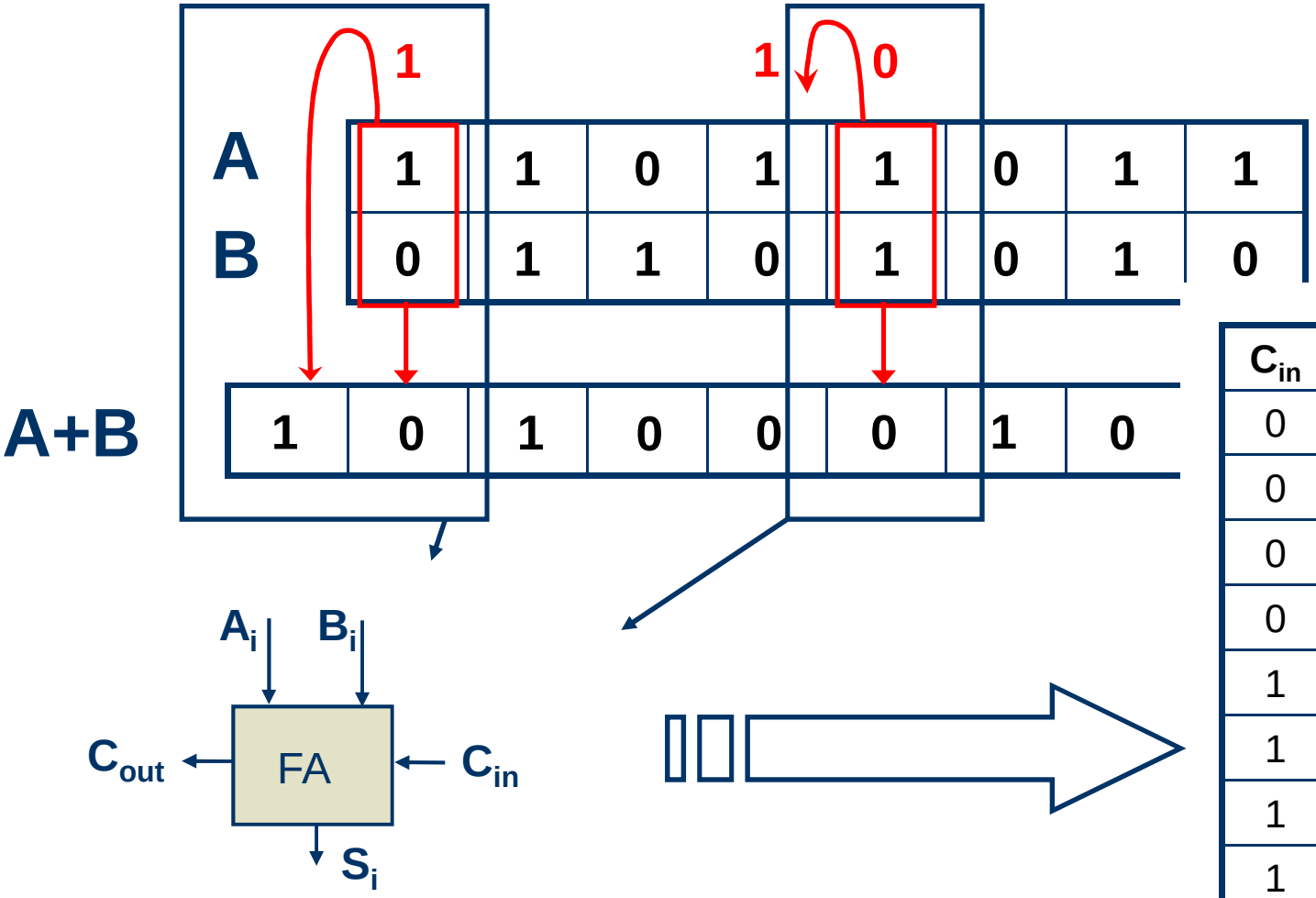
- Anche in questo caso è possibile sfruttare la natura **iterativa** dell'operazione



Addizione

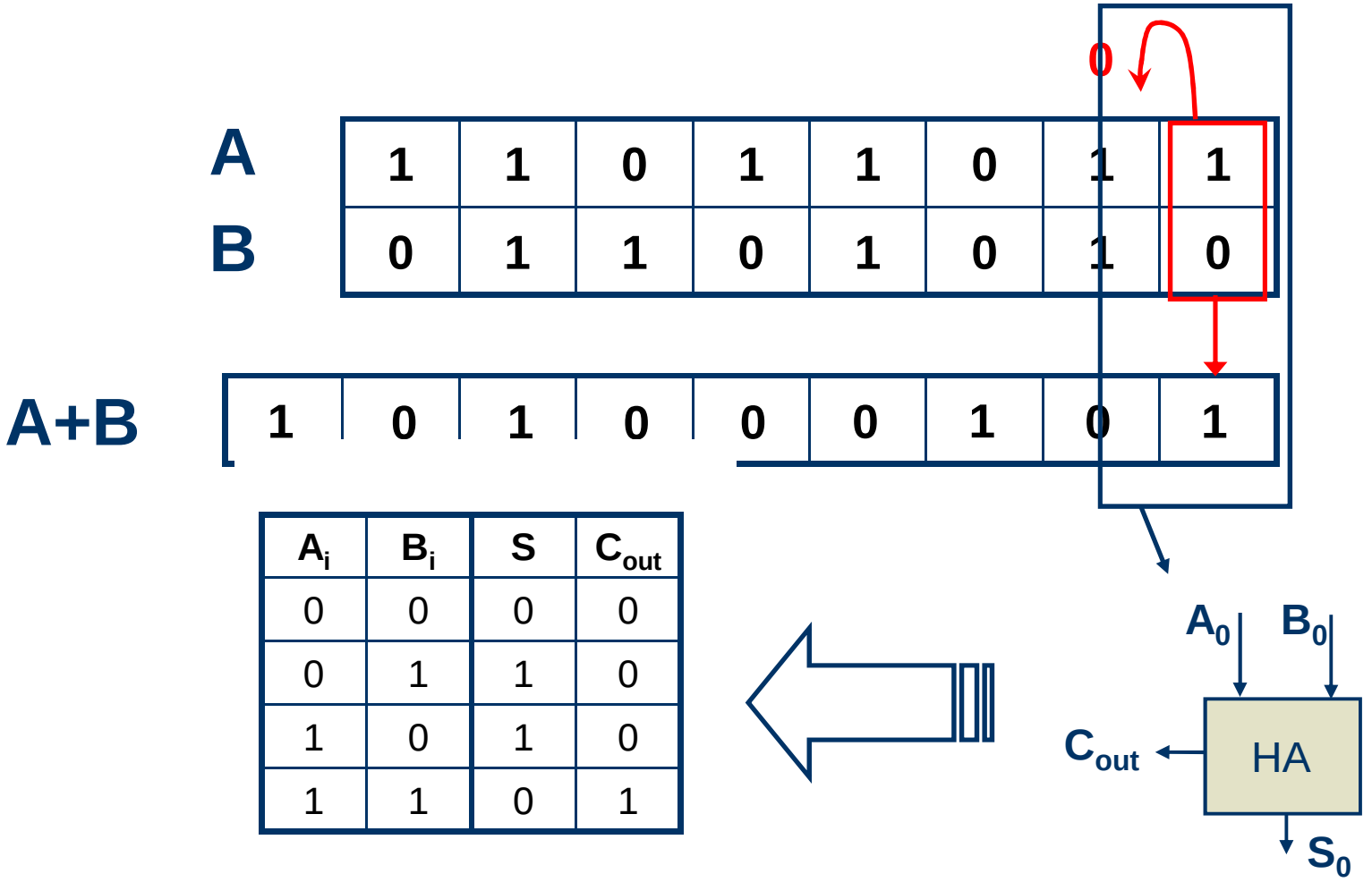

1 1 1 1 0 1 0
A
1 1 0 1 1 0 1 1
B
0 1 1 0 1 0 1 0
A+B
1 0 1 0 0 0 1 0 1
A_i B_i
C_{out}
FA
C_{in}
S_i
A_0 B_0
C_{out}
HA
S_0

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



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

Addizionatore binario

- Per il semiaddizionatore valgono le eguaglianze:

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

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

- Per l'addizionatore completo valgono le eguaglianze:

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

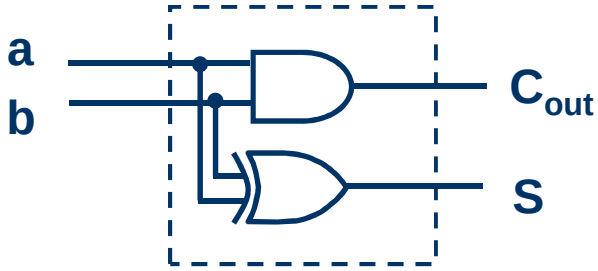
$$C_{out} = \bar{a}bC_{in} + a\bar{b}C_{in} + ab\overline{C_{in}} + abC_{in} = ab + bC_{in} + aC_{in} = ab + C_{in}(a+b)$$

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:

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

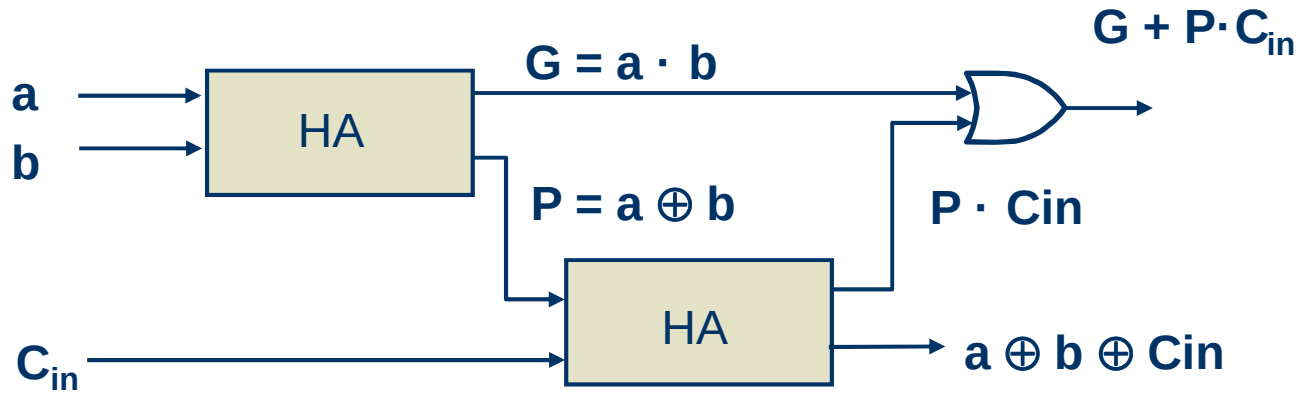
$$C_{out} = ab + C_{in}(a \oplus b) = G + C_{in}P$$

Addizionatore binario

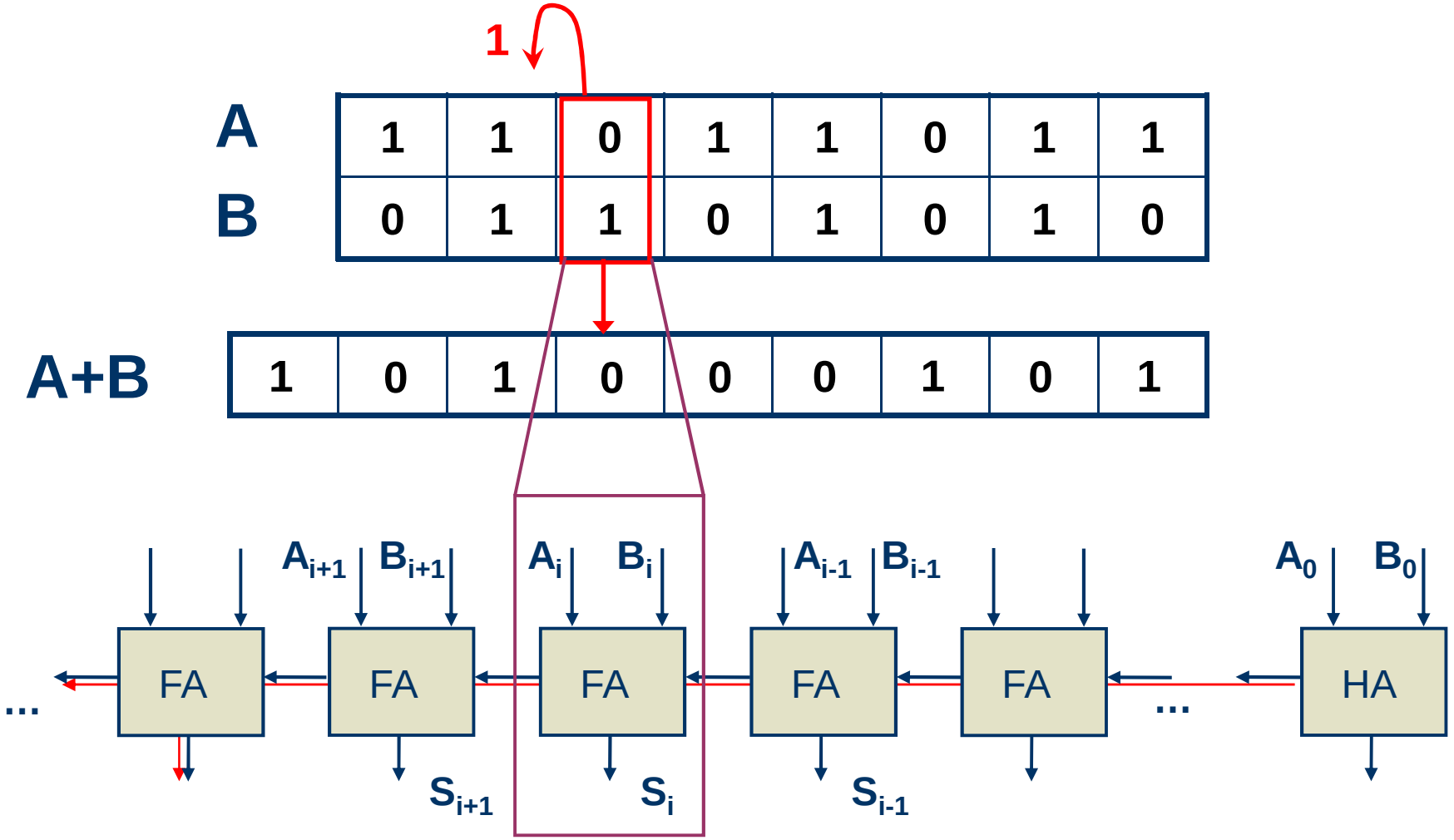
- Pertanto, un addizionatore completo può essere ottenuto a partire da due semiaddizionatori:

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

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



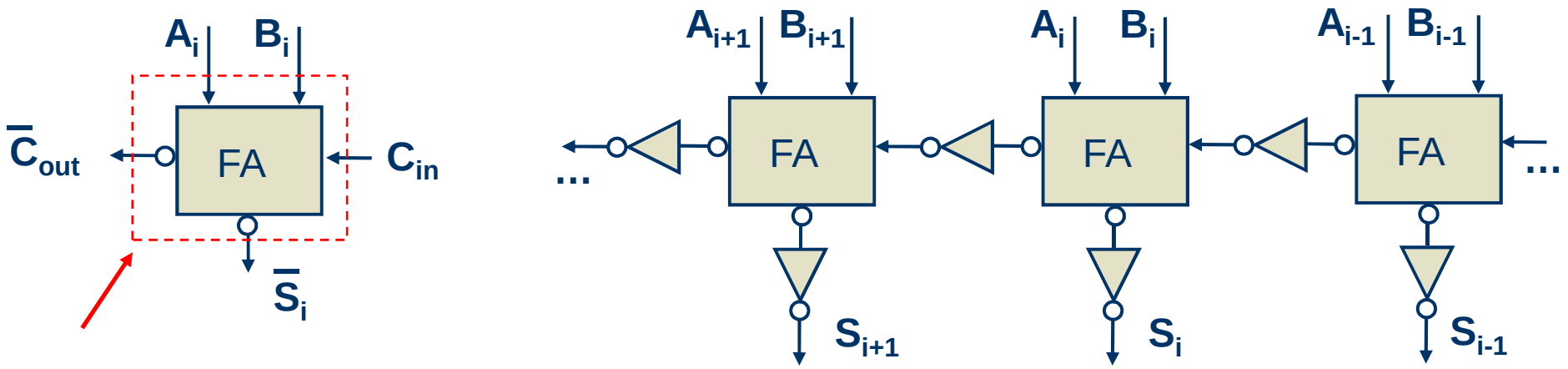
Addizionatore a propagazione



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



Addizionatore a propagazione

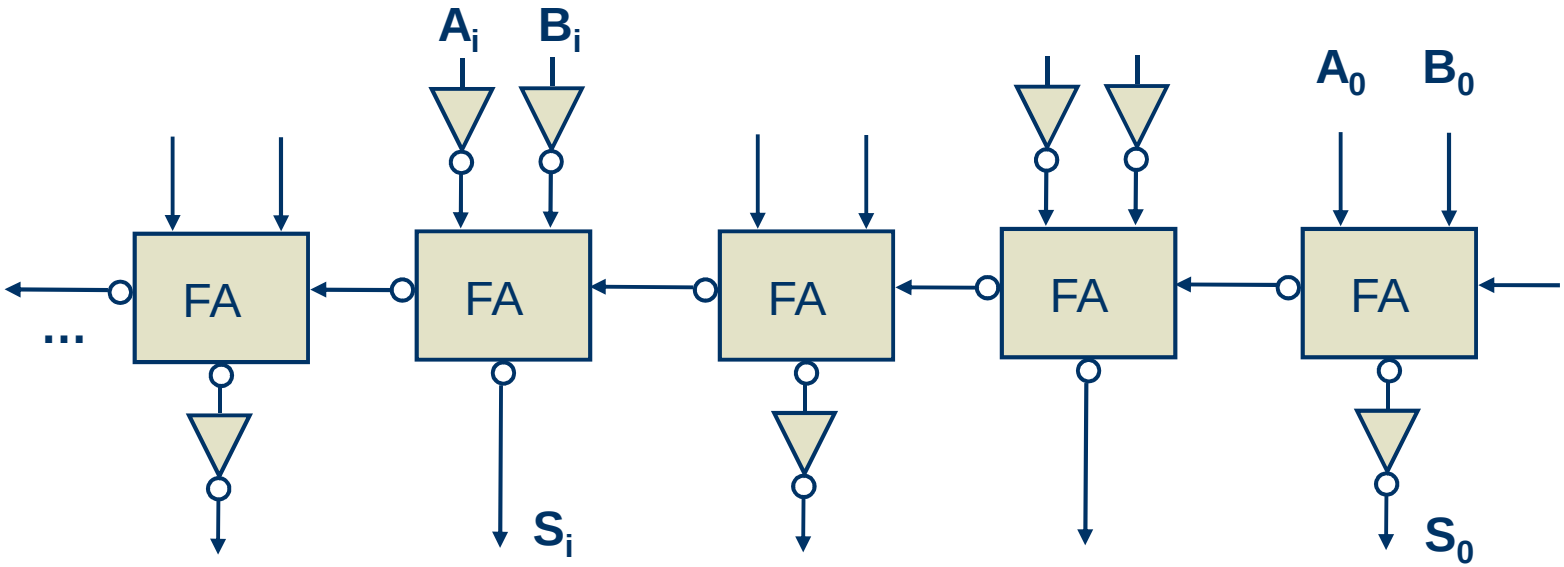
Materiale facoltativo

osservando che:

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

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

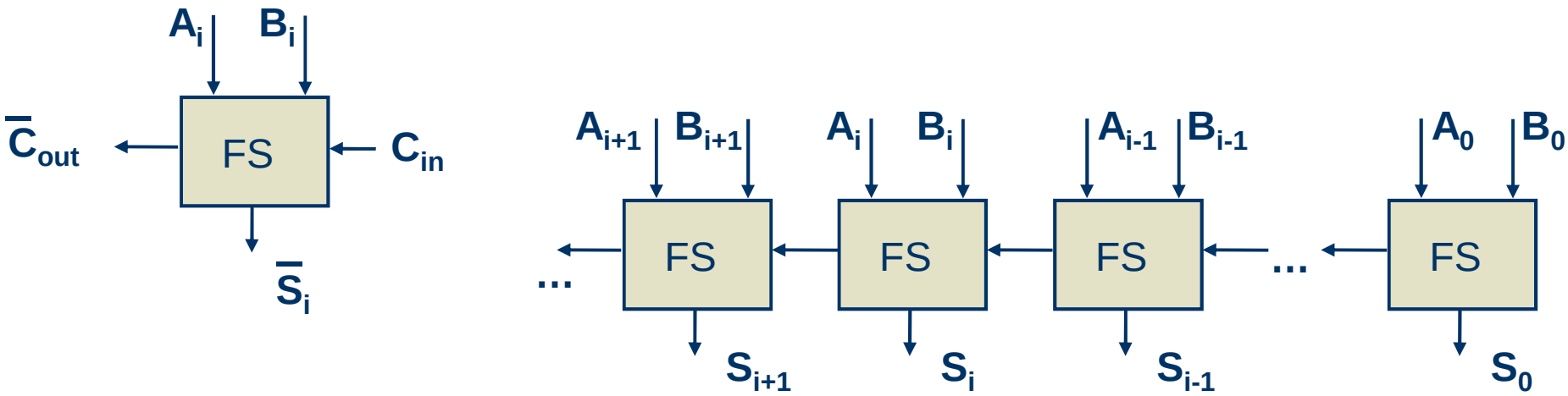
è possibile ottenere uno schema più veloce (meno porte lungo il cammino di propagazione del riporto):



Sottrattore

Materiale facoltativo

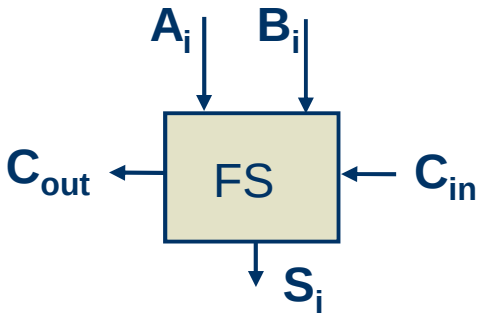
- stessa idea usata nell'addizionatore
- questa volta si propaga un "*prestito*" invece del riporto
- il prestito si *sottrae* alla coppia di cifre correnti
- i bit del sottraendo sono negati: 0 → 0, 1 → -1



Sottrattore

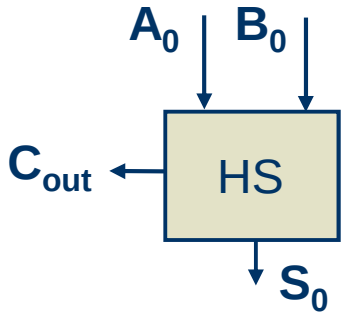
Materiale facoltativo

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:

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

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

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

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

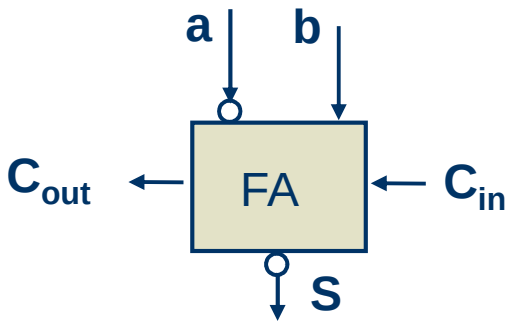
$$C_{out} = \bar{a}b$$

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

Sottrattore

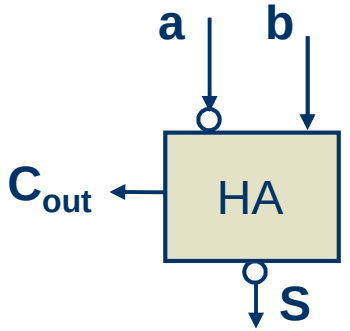
Materiale facoltativo

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



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

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

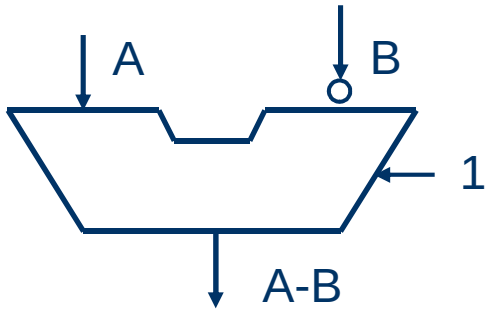


$$\overline{S} = \overline{a} \oplus b$$

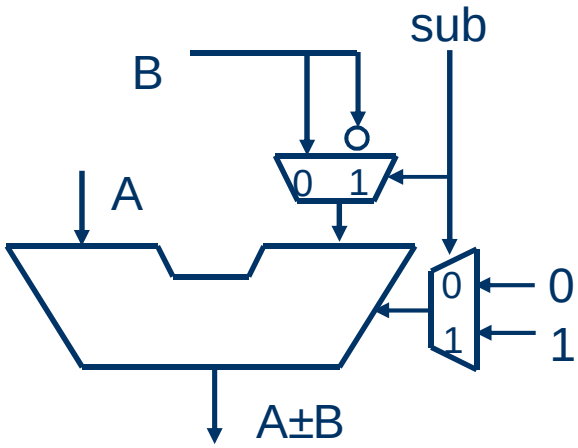
$$C_{out} = \overline{a}b$$

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

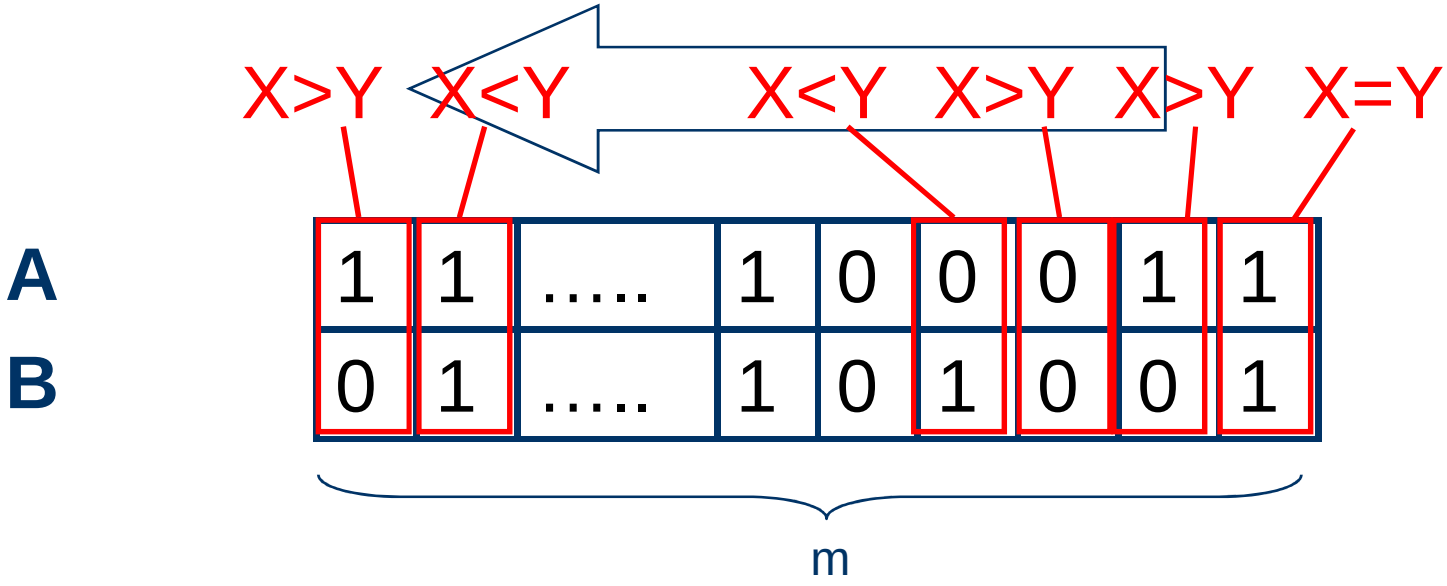


addizionatore/sottrattore programmabile:
sub=0 → addizione
sub=1 → sottrazione

Componenti aritmetici: comparatore binario

Materiale facoltativo

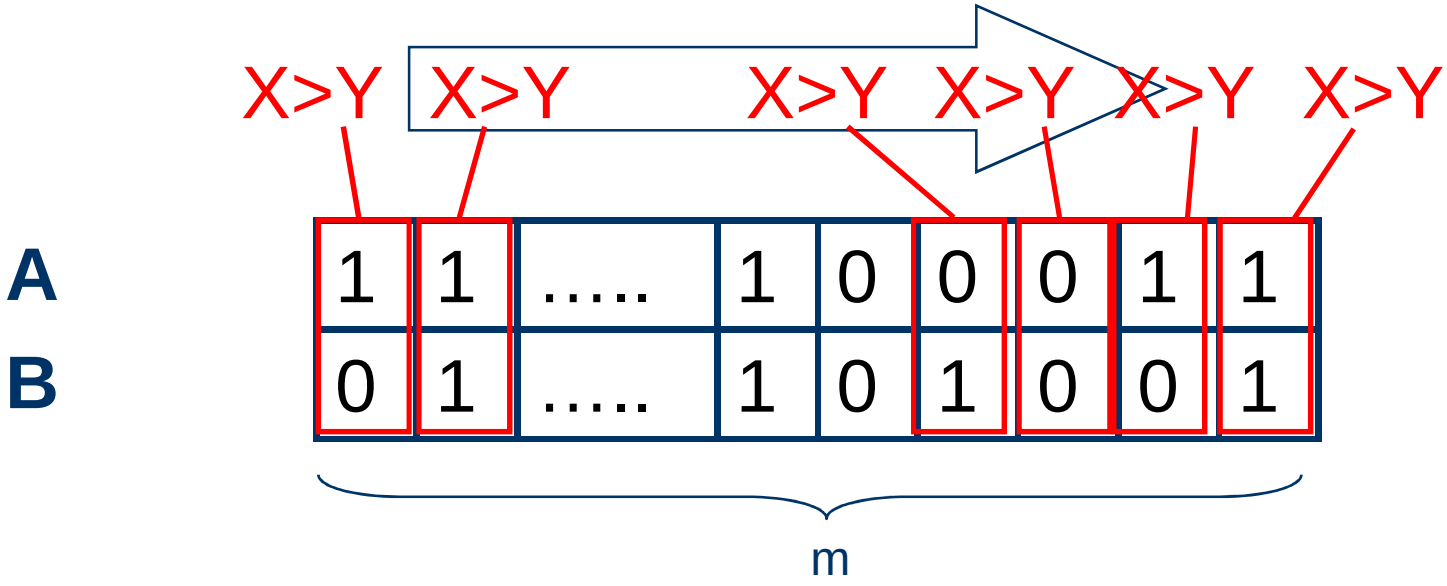
- confronto su numeri binari
- può essere effettuato sia da destra
  - l'ultima differenza trovata ci dice quale dei due numeri **A** e **B** è più grande…..



Comparatore binario

Materiale facoltativo

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



Comparatore binario

Materiale facoltativo

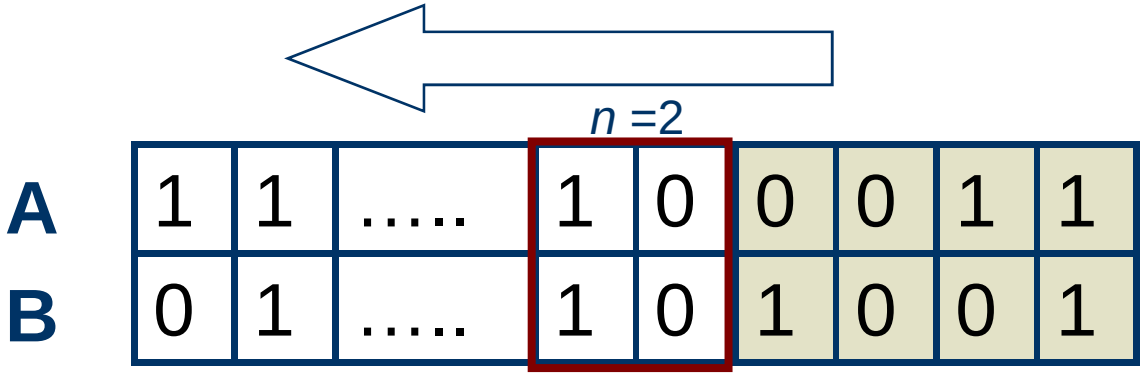
- In entrambi i casi, il confronto si può effettuare “localmente” sulle singole coppie di bit da confrontare x_i,$y_{i,}$,
- …tenendo però conto dell’esito del confronto parzialmente effettuato a destra o a sinistra

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

Comparatore binario

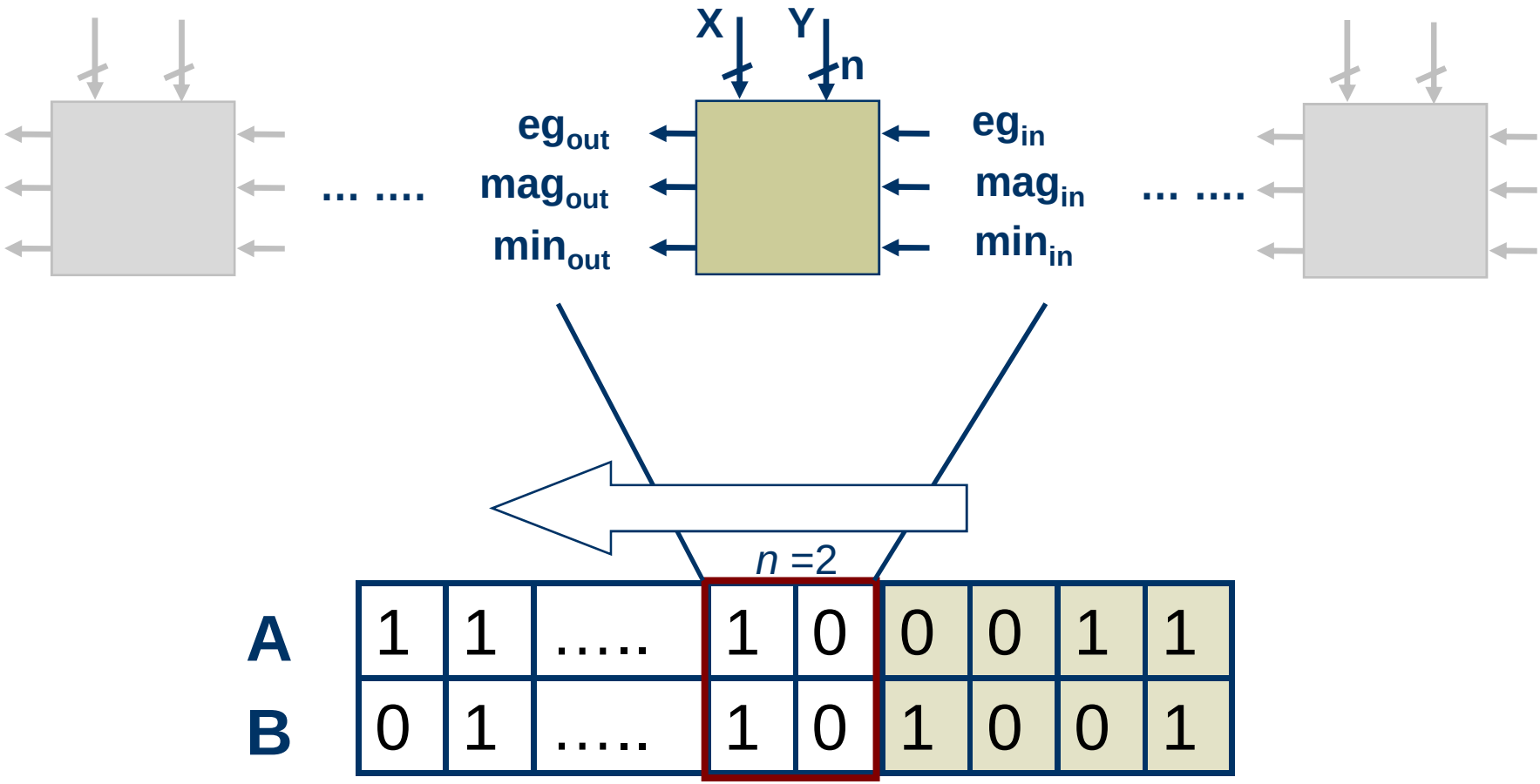
Materiale facoltativo

- possiamo raggruppare in *cifre* gruppi di *n* bit consecutivi (ad esempio *n* =2) da confrontare



Comparatore binario

Materiale facoltativo



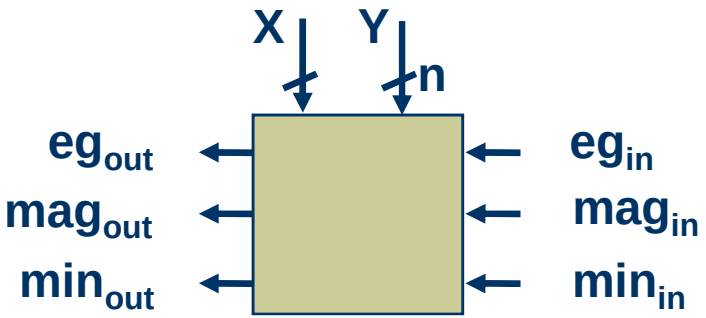
Comparatore binario

Materiale facoltativo

$eg_{out} := (X = Y)$ AND eg_{in}

$mag_{out} := (X > Y)$ OR [$(X = Y)$ AND mag_{in}]

$min_{out} := (X < Y)$ OR [$(X = Y)$ AND min_{in}]



X ed ***Y*** sono due cifre di ***n*** bit (***n*** =2 nell'esempio)

eg_{in}, **mag_{in}**, e **min_{in}** ci dicono se i due numeri parzialmente confrontati a destra sono uguali, maggiori o minori l'uno dell'altro

eg_{out}, **mag_{out}**, **min_{out}** propagheranno la stessa informazione a sinistra

Comparatore binario

Materiale facoltativo

- Deriviamo l'espressione di eg_{out}
  - partiamo dall'eguaglianza $X=Y$

$$(X = Y) = \prod_{i=0}^{n-1} (X_i \equiv Y_i) = \prod_{i=0}^{n-1} (\overline{X_i \oplus Y_i}) = \overline{\sum_{i=0}^{n-1} (X_i \oplus Y_i)}$$

ricordiamo che la XOR è la negata della EQ

- considerando anche il segnale **eg_{in}**, proveniente da destra:

$$eg_{out} := \left[\prod_{i=0}^{n-1} (X_i \equiv Y_i) \cdot eg_{in} \right] \Rightarrow eg_{out} := \overline{\left[\sum_{i=0}^{n-1} (X_i \oplus Y_i) \right]} + \overline{eg_{in}}$$

Comparatore binario

Materiale facoltativo

- La realizzazione in logica a due livelli *and-or (nand)* non è conveniente
- La funzione (*X*=*Y*) non presenta mintermini adiacenti

Y \ X	00	01	11	10
00	1			
01		1		
11			1	
10				1

Mappa di Karnaugh per *f* = *(X=Y)* con *X* e *Y* ingressi di due variabili ciascuno

Comparatore binario

Materiale facoltativo

- Deriviamo l'espressione di mag_{out}
- Osserviamo che, per le singole cifre binarie, si ha

$$X_i > Y_i \quad \Leftrightarrow \quad X_i \cdot \overline{Y_i} = 1$$

- quindi, per ottenere il confronto tra ingressi a più cifre è possibile "partire" confrontando le cifre da sinistra e procedendo via via verso destra fino a trovare eventuali differenze

$$\begin{aligned} mag_{out} := {} & X_{n-1} \cdot \overline{Y_{n-1}} + \\ & (X_{n-1} \equiv Y_{n-1}) \cdot (X_{n-2} \cdot \overline{Y_{n-2}}) + \\ & (X_{n-1} \equiv Y_{n-1}) \cdot \ldots \cdot (X_1 \equiv Y_1) \cdot X_0 \cdot \overline{Y_0} + (X = Y) \cdot mag_{in} \end{aligned}$$

Comparatore binario

Materiale facoltativo

- Per ottenere la terza uscita min_{out}, se necessaria, si può procedere analogamente a mag_{out} oppure semplicemente osservare che

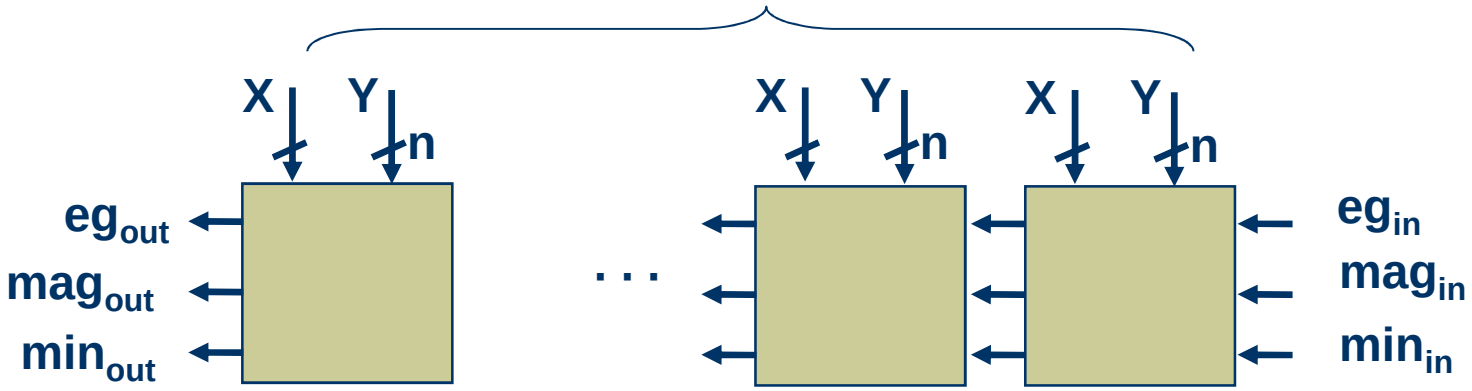
$$\min_{\text{out}} := \overline{eg_{out}} \cdot \overline{mag_{out}}$$

- Si noti che la componente $X_i \equiv Y_i$ è ripetuta nelle tre funzioni e può essere calcolata una sola volta

Comparatore binario

Materiale facoltativo

- un confronto tra stringhe di *k* bit può essere effettuato con un unico comparatore se $k \leq n$ numero di bit di *X* e *Y*
- Altrimenti, possono essere collegati in catena {[*k*/*n*]} comparatori, come in figura, procedendo da destra a sinistra



Comparatore binario

Materiale facoltativo

- Sarebbe stato possibile progettare il circuito anche partendo da sinistra
- La logica con cui generare le uscite $\mathbf{eg_{out}}$, $\mathbf{mag_{out}}$, $\mathbf{min_{out}}$ è differente
- tuttavia, in ogni caso il circuito fisico da realizzare deve coprire l'intera lunghezza degli operandi, quindi il costo realizzativo ed il ritardo saranno simili