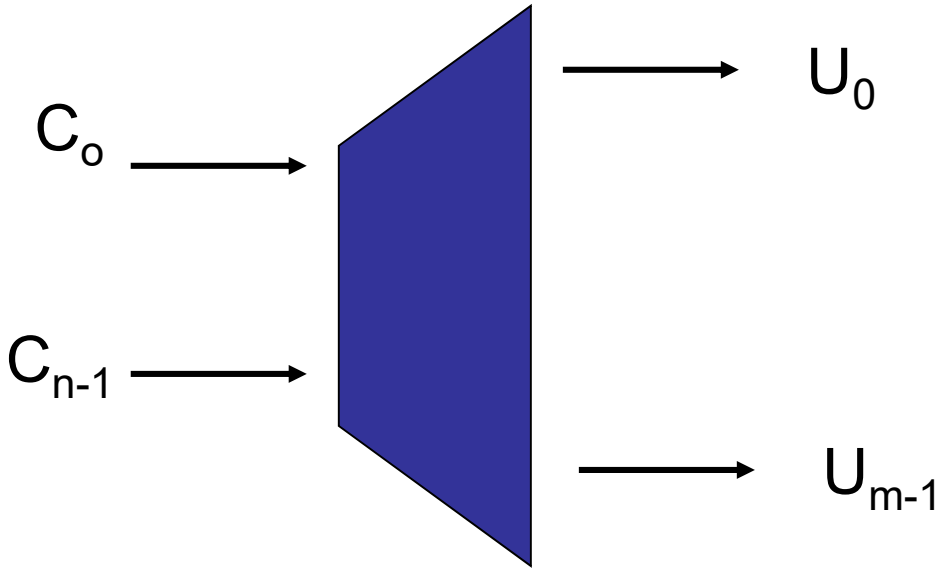
Decodificatore (decoder) 1 su m

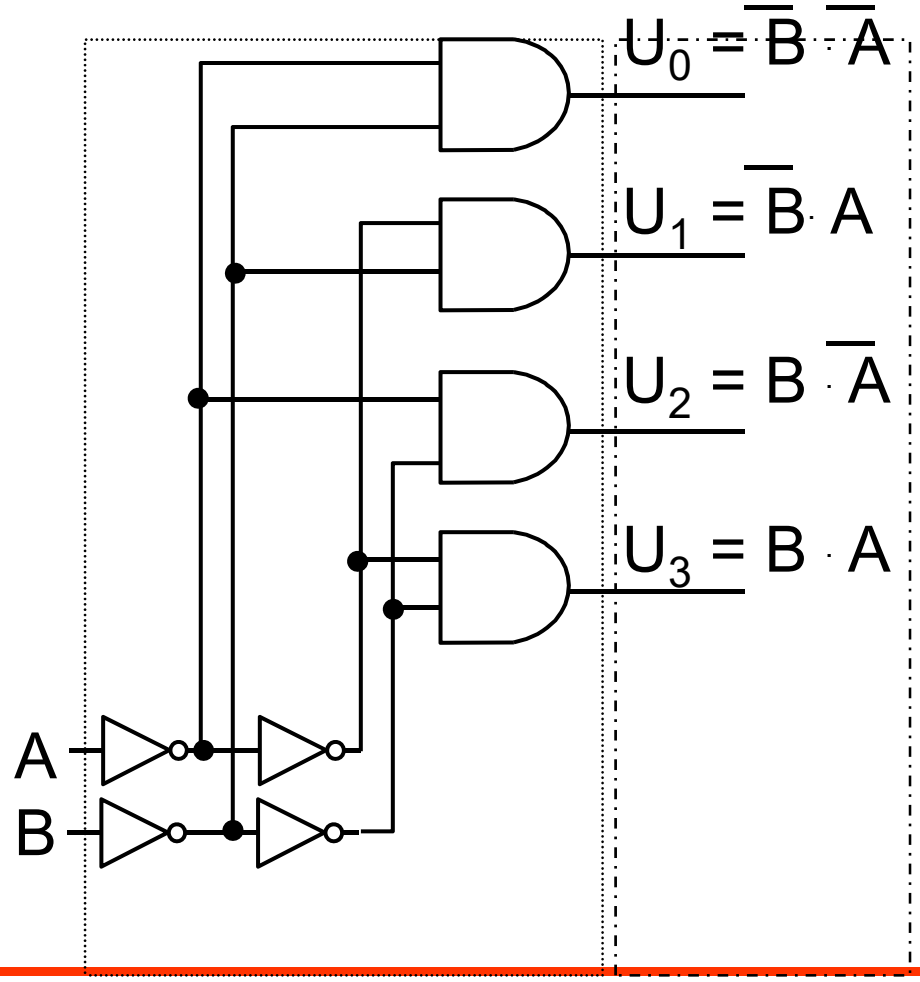
- Un decodificatore è una macchina che riceve in ingresso una parola codice (C) su n bit e presenta in uscita la sua rappresentazione decodificata (linee U_0, … U_{N-1}) su $m=2^n$ bit



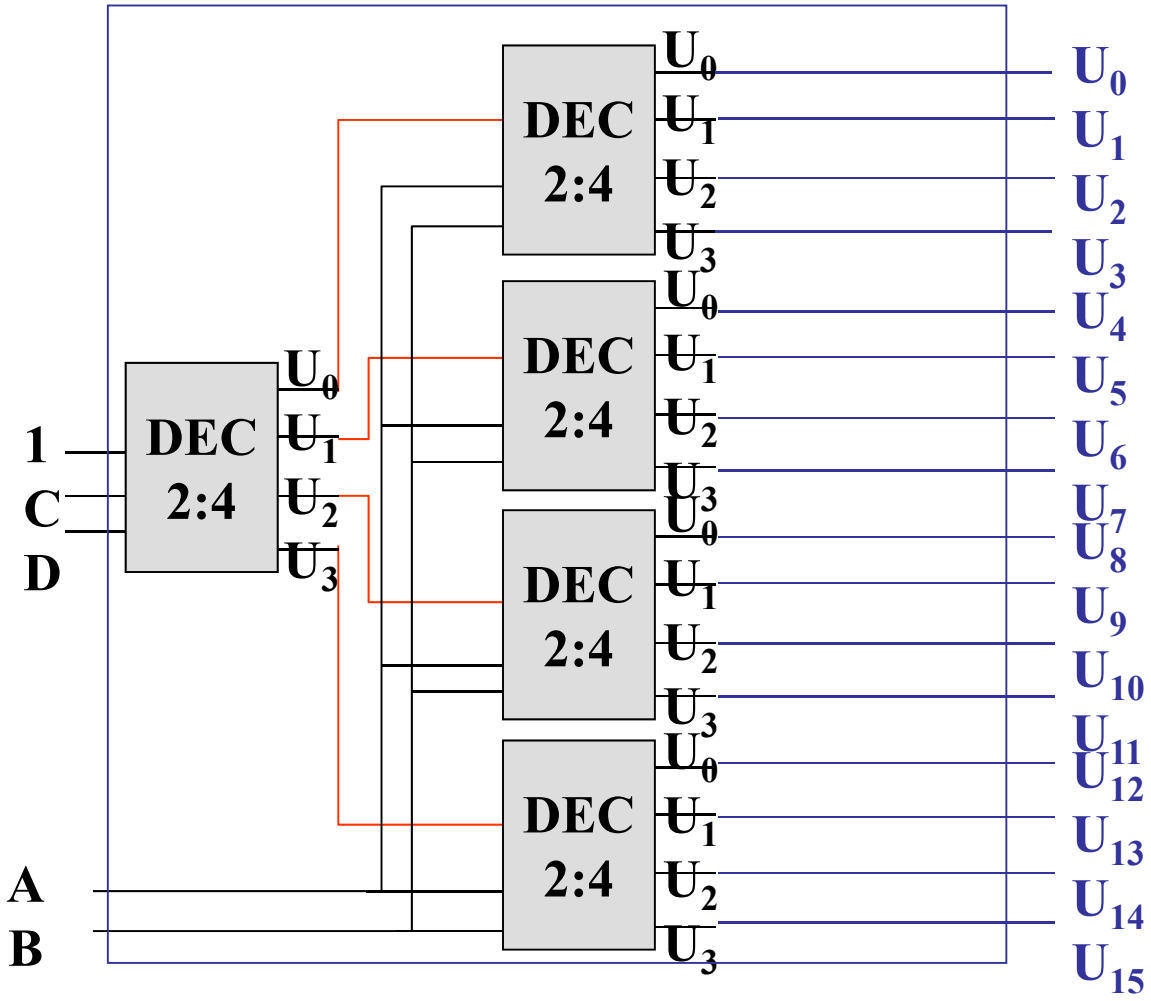
Decoder 1 su 4

Esempio: decoder 1:4

	B	A	U_0	U_1	U_2	U_3
1	0	0	1	0	0	0
2	0	1	0	1	0	0
3	1	0	0	0	1	0
4	1	1	0	0	0	1



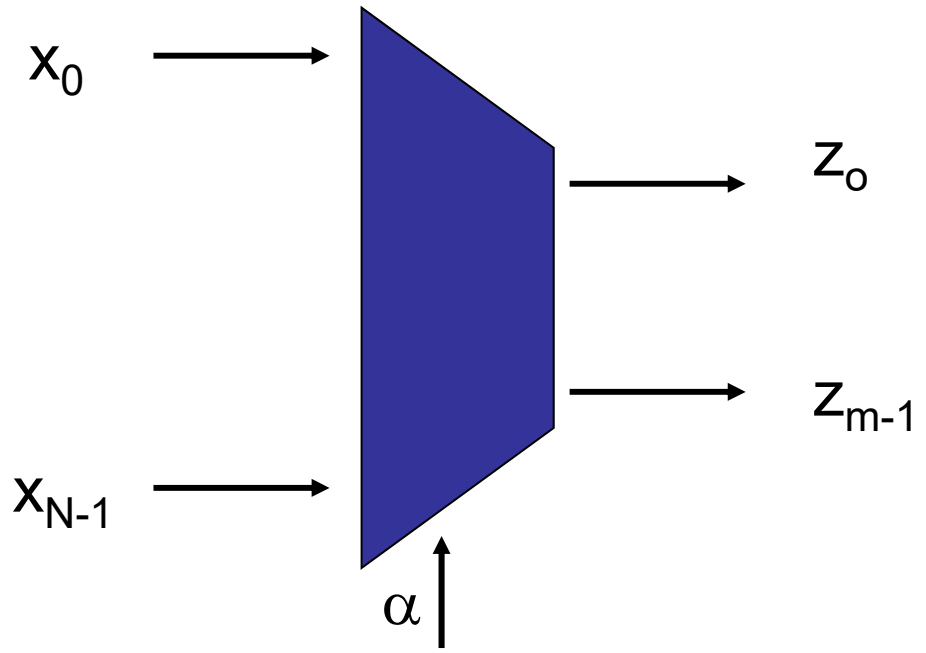
Composizione modulare di Decoder 4:16



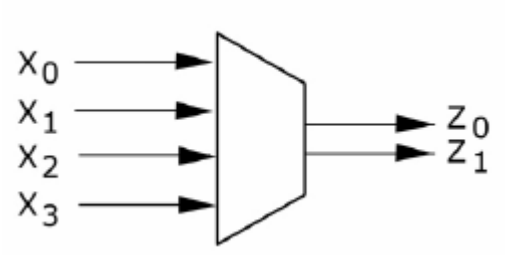
Encoder o codificatore

- Un codificatore riceve in ingresso una rappresentazione decodificata (linee x_0, … x_{m-1}) e fornisce in uscita una rappresentazione con un codice a lunghezza fissa di n bit
- L'uscita è la parola codice associata a x_i
  se $x_i=1$
  ed $\alpha=1$ (abilitazione)
- Vincolo su ingressi:

  $x_i \cdot x_j = 0$ per $i \neq j$



Codificatore a 4 ingressi



x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Codificatore 8-4-2-1

- $z_3 = x_8 + x_9$
- $z_2 = x_4 + x_5 + x_6 + x_7$
- $z_3 = x_2 + x_3 + x_6 + x_7$
- $z_3 = x_1 + x_3 + x_5 + x_7 + x_9$

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

- Espressioni ottenute considerando opportunamente le configurazioni di ingresso *dont'care*

Arbitro di priorità

- Un codificatore può essere preceduto da una “rete a priorità” che, in caso di più ingressi contemporaneamente alti, filtra quello con priorità assegnata maggiore

- Rete 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 Y_i è alta se e solo se X_i è alto e tutti gli altri ingressi prioritari su X_i sono bassi.

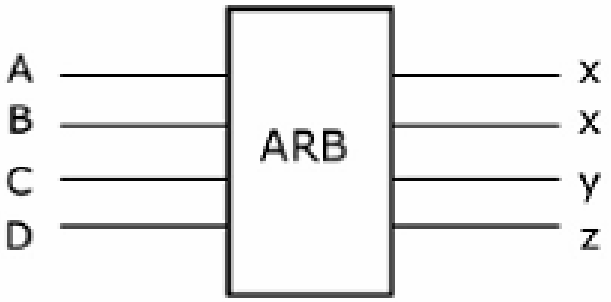
$$F_1 = X_1$$

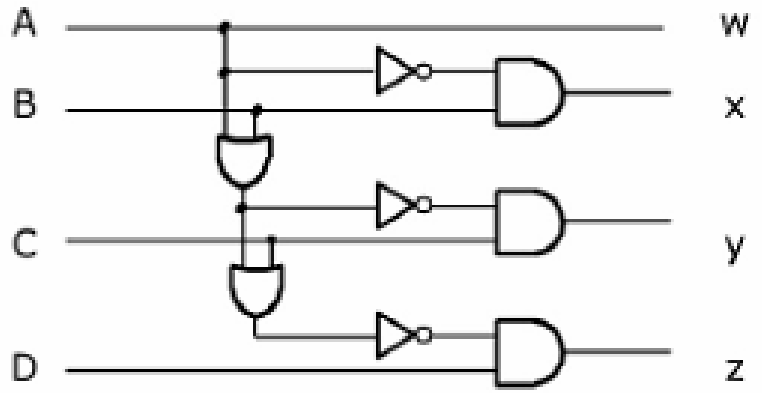
$$F_2 = X_2 \overline{X_1}$$

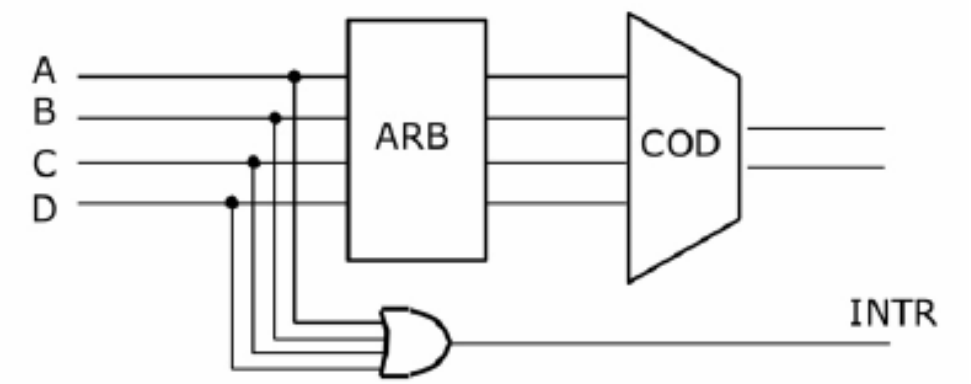
……

$$F_n = X_n \overline{X_{n-1}} \ldots \overline{X_1}$$

Arbitro di priorità a 4 ingressi

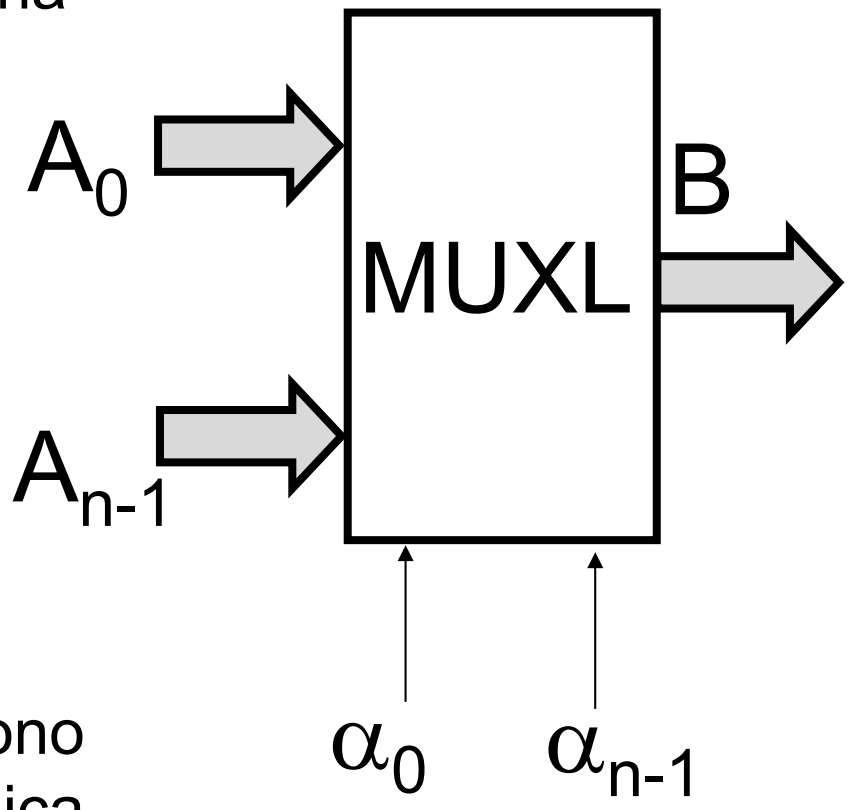






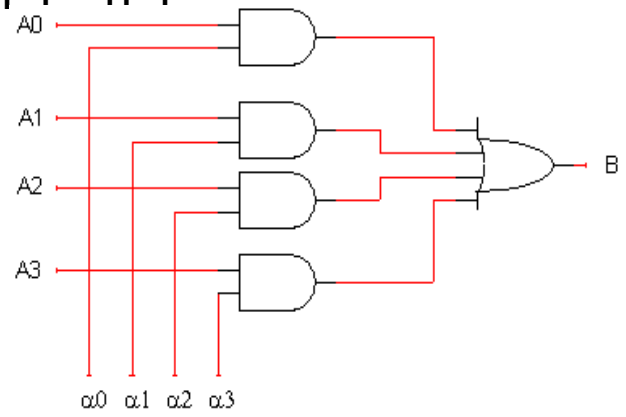
Multiplexer lineare

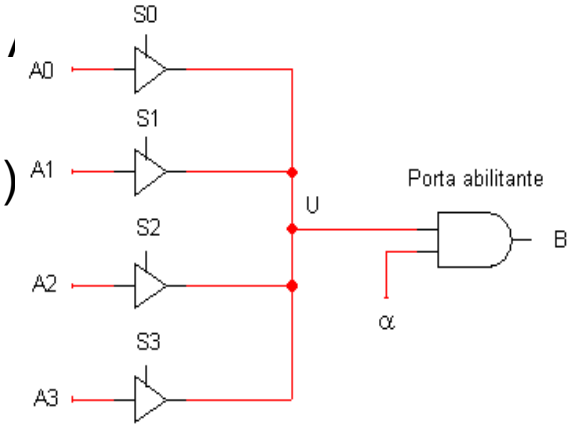
- Un *Multiplexer lineare (*ML) è una macchina con:
  - n ingressi-dati (A_0,…,A_{n-1})
  - n segnali binari di selezione (α_0,…, α_{n-1}), **dei quali al più uno è attivo**
  - una uscita-dati B, che assume
    - valore A_i se è attivo α_i
    - neutro se nessuna delle selezioni è attiva
- utilizzata quando più linee devono essere convogliate verso un'unica linea di uscita (bus)



Multiplexer lineare - realizzazioni

- $B = A_0 \alpha_0 + A_1\alpha_1 + \ldots + A_{n-1}\alpha_{n-1}$, n=4
  - **Realizzazione I**
    - Con porte AND e OR
  - **Realizzazione II**
    - Con porte 3-state
      - S=1, restituisce il valore di A
      - S=0, restituisce un'alta impedenza (apre il circuito)

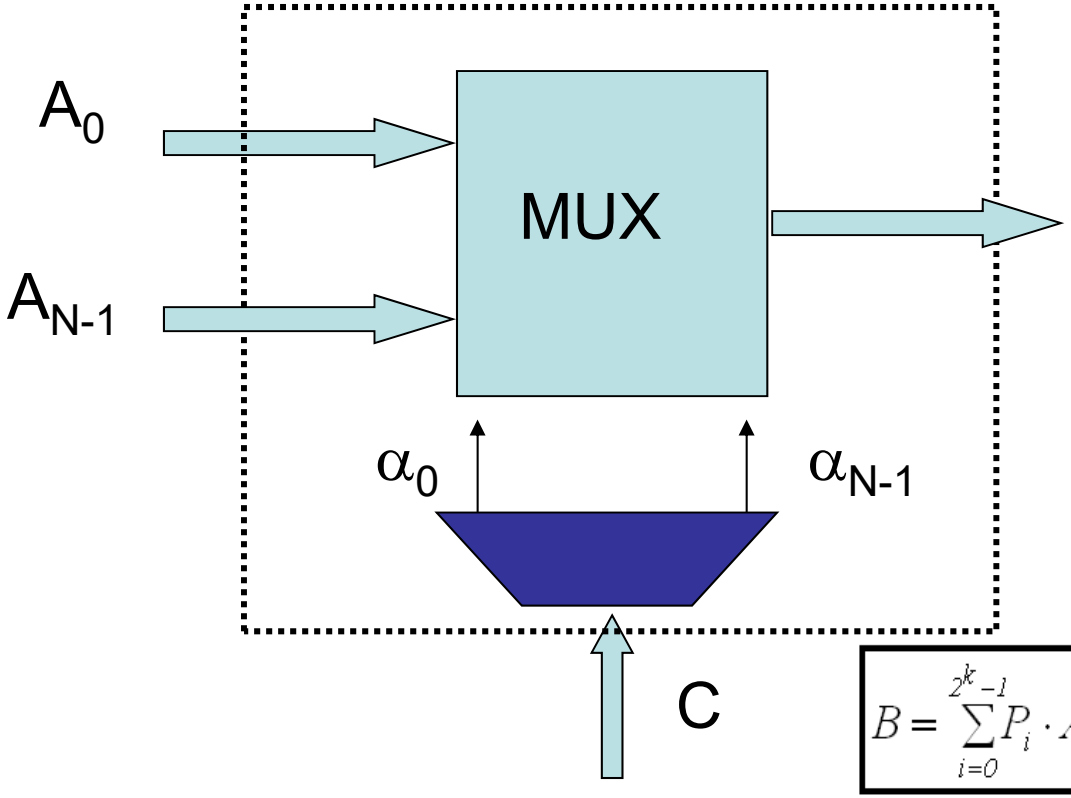




S	Uscita
0	z
1	0
0	z
1	1

Multiplexer (indirizzabile)

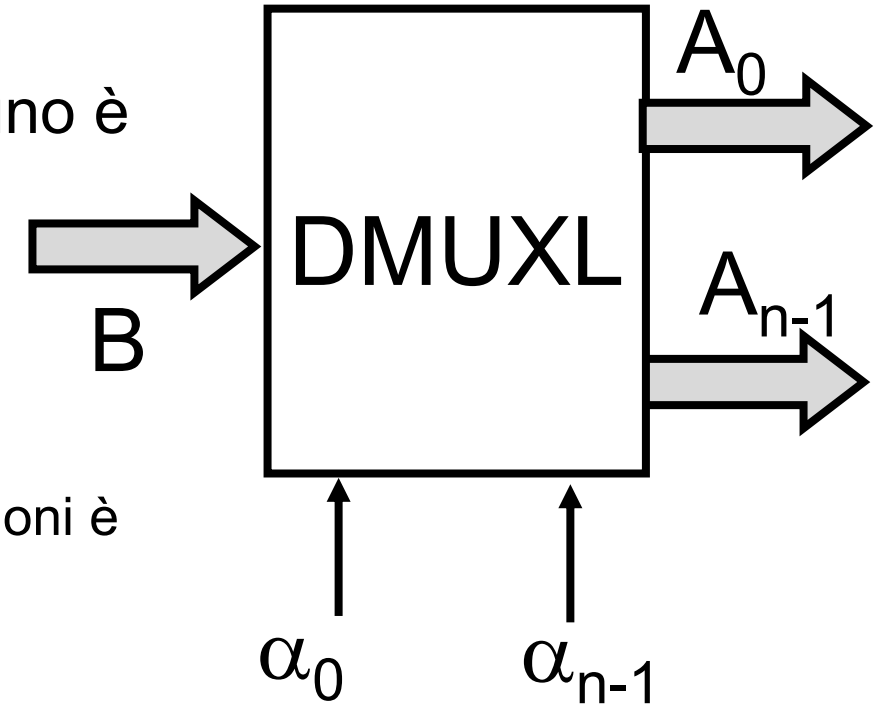
- Multiplexer Lineare i cui segnali di abilitazione sono collegati con le uscite di un decodificatore



$$B = \sum_{i=0}^{2^k-1} P_i \cdot A_i \quad \text{con } P_i \text{ mintermine delle } C$$

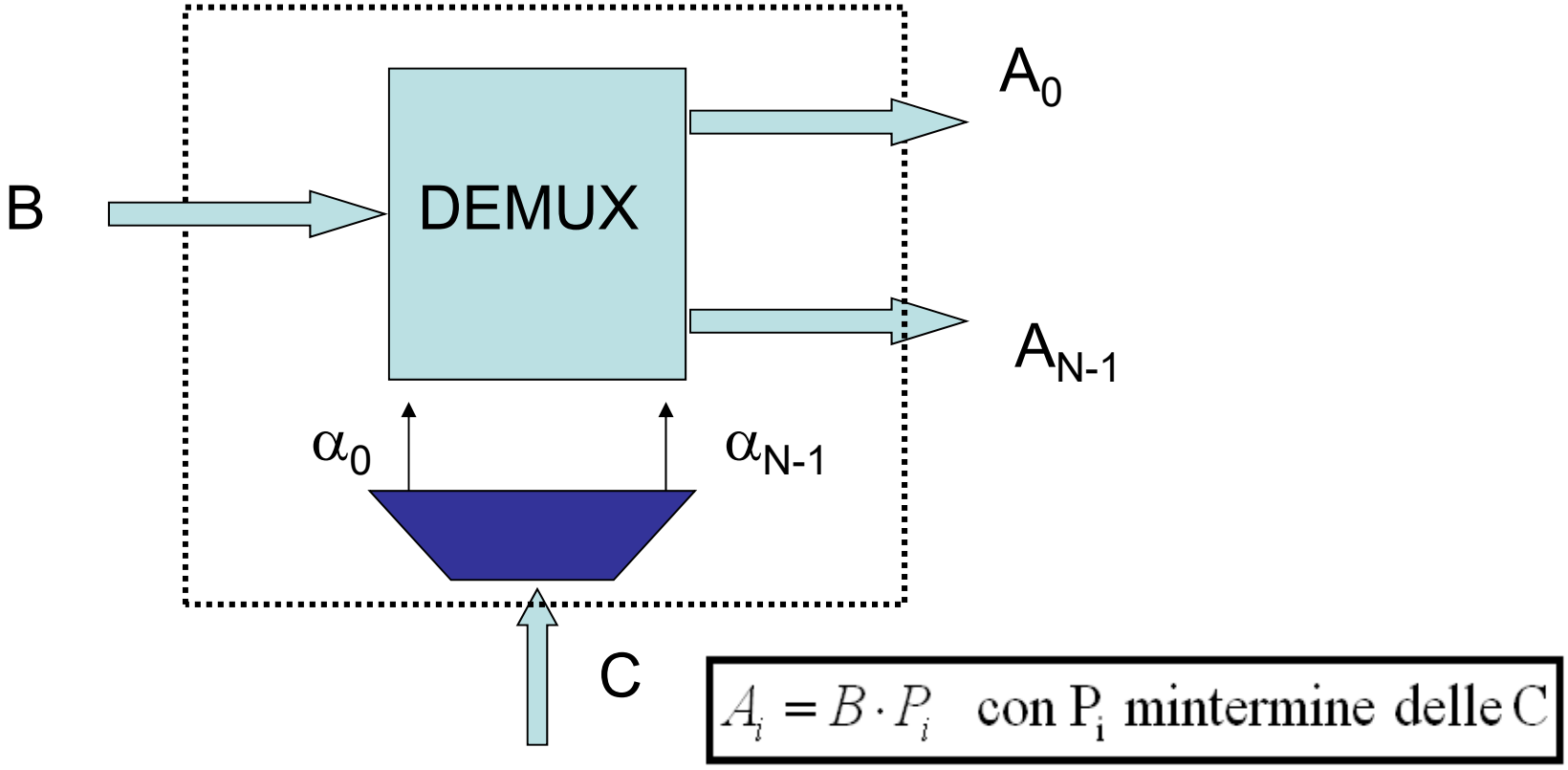
Demultiplexer lineare

- Un *Demultiplexer Lineare* è una macchina con:
  - 1 ingresso-dati B
  - n segnali binari di selezione ($\alpha_0,\ldots, \alpha_{n-1}$), dei quali al più uno è attivo
  - n uscite-dati ($A_0,\ldots,A_{n-1}$), con
    - A_i=B se è attivo α_i
    - neutro se nessuna delle selezioni è attiva



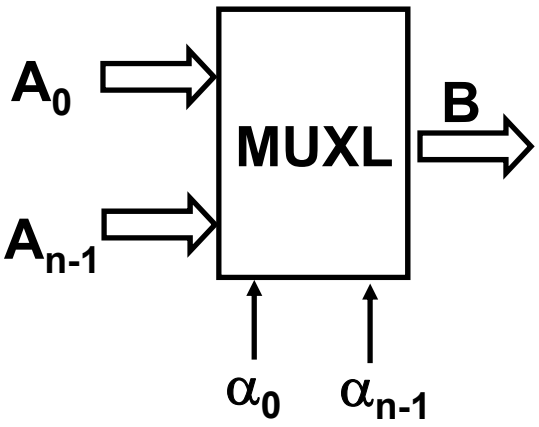
Demultiplexer (indirizzabile)

- Demultiplexer Lineare i cui segnali di abilitazione sono collegati con le uscite di un decodificatore

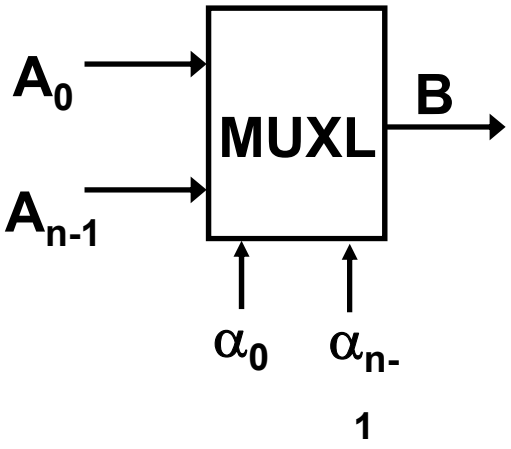


Multiplexer binario

- Se i dati A_i e B sono vettori di bit, che viaggiano su un bus
  - si parla genericamente di multiplexer o demultiplexer

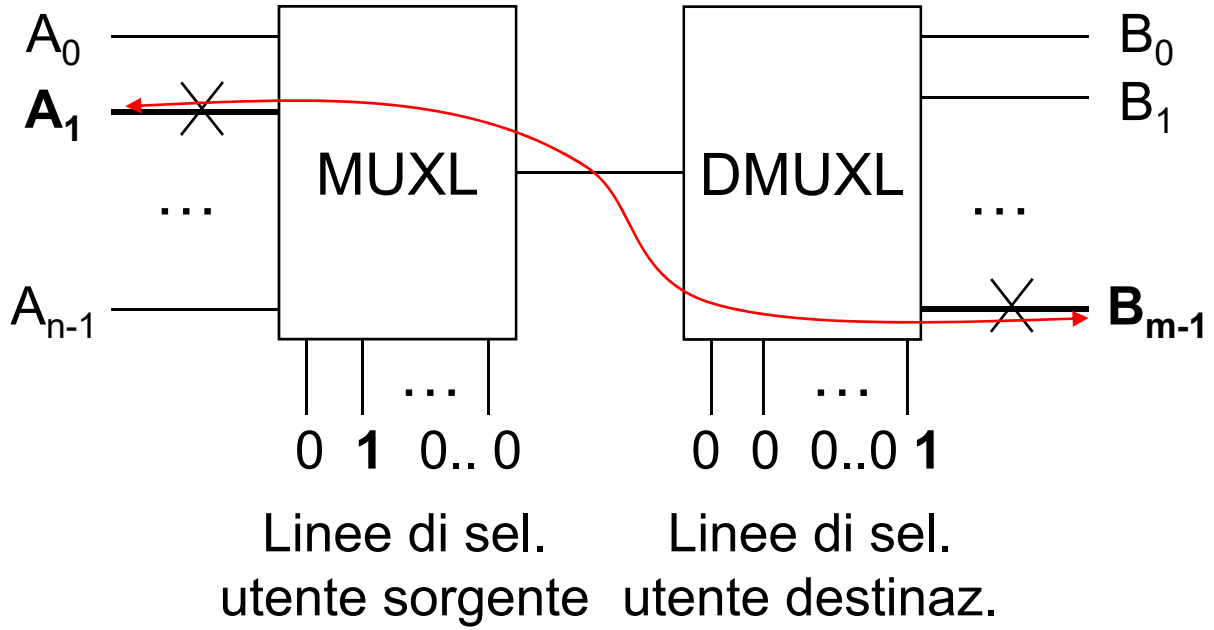


- Se i dati A_i e B sono singoli bit
  - si parla di multiplexer o demultiplexer binario



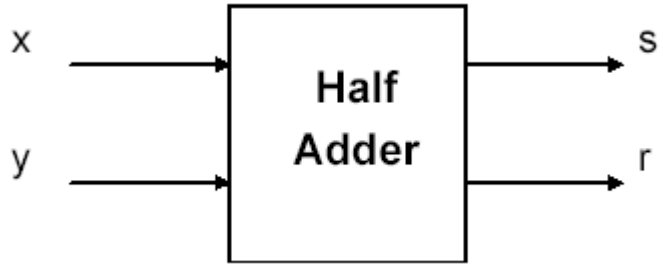
MuxI/DmuxI: un esempio

- Supponiamo di avere un “centralino telefonico” in cui *n* utenti sorgente vogliono parlare con *m* utenti destinazione
  - **vincolo:** l’utente di destinazione abilitato deve sentire solo l’utente sorgente abilitato



L’utente A_1 è abilitato a parlare con l’utente B_{m-1}

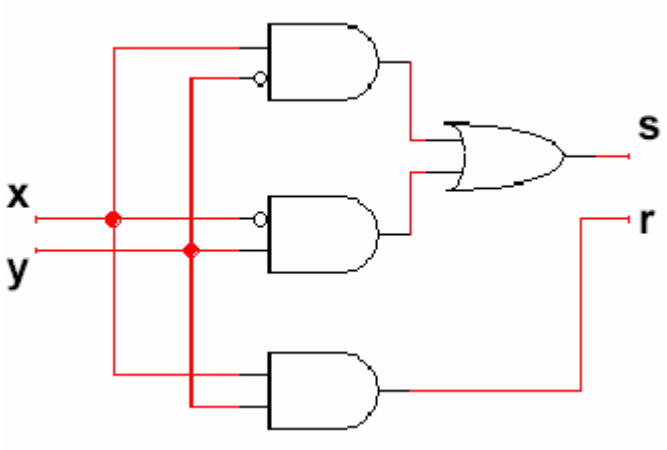
Half Adder

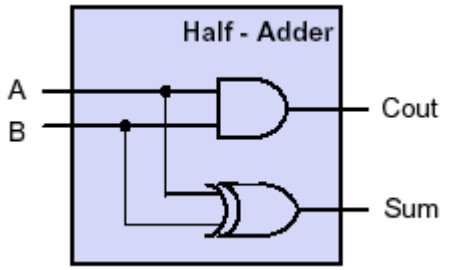


x	y	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

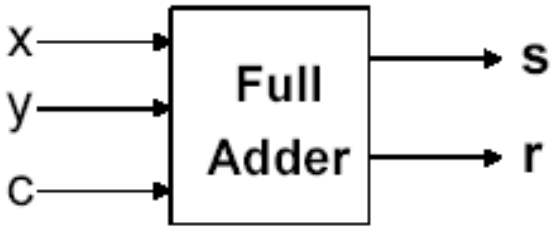
$S = \overline{X}Y + X\overline{Y} = X \text{ xor } Y$

$r = X\,Y$

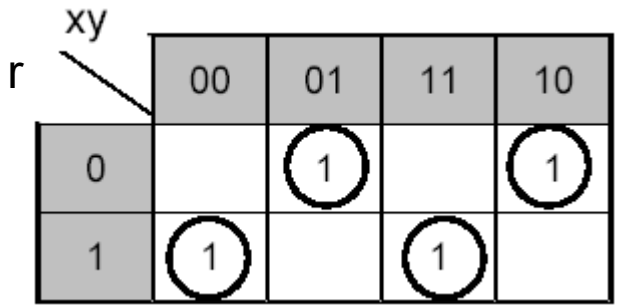


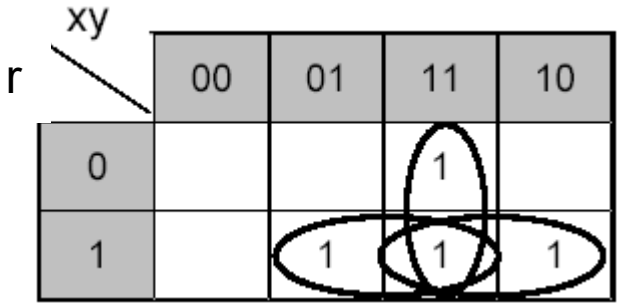


Full Adder (1/2)



X	Y	r	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

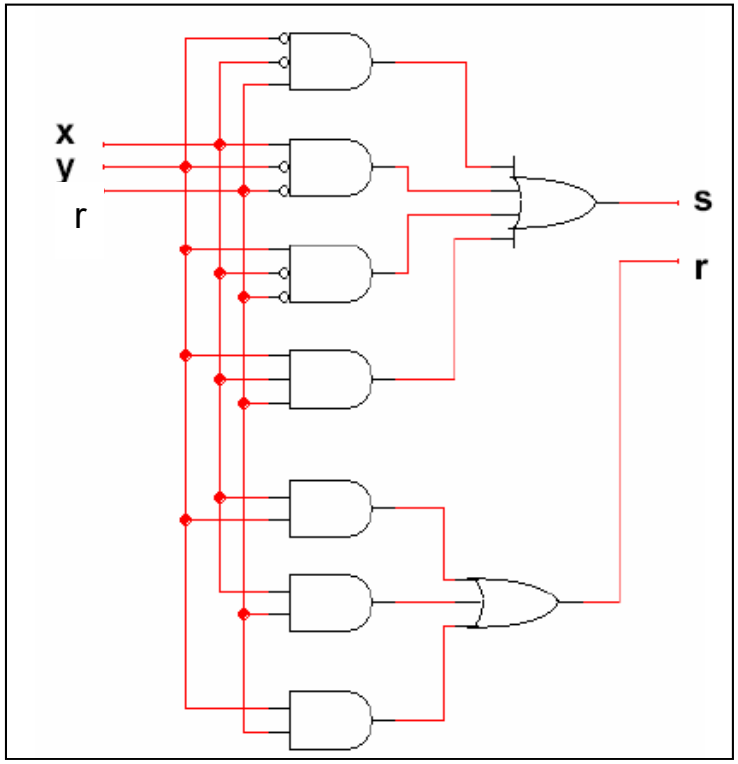




Full Adder (2/2)

$$S = (\overline{X}\,\overline{Y}\,r) + (\overline{X}\,Y\,\overline{r}) + (X\,\overline{Y}\,\overline{r}) + (X\,Y\,r)$$

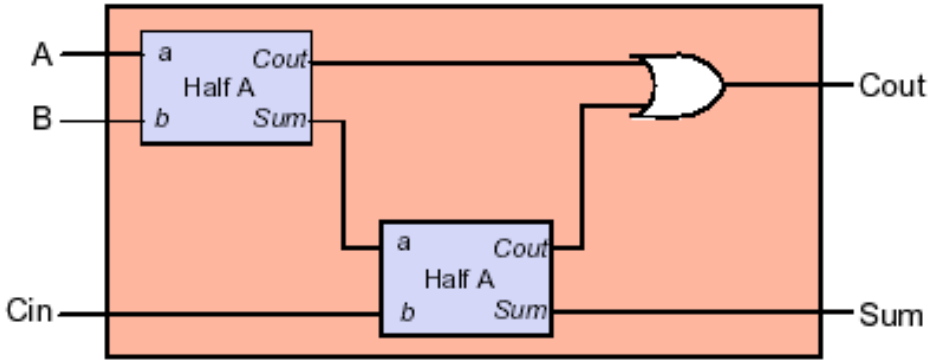
$$R = \overline{X}Yr + X\overline{Y}r + XY\overline{r} + XYr = XY + Yr + Xr$$



Full Adder con 2 Half Adder

- Gli **output** sono uguali all'Half - Adder, cioè la somma (Sum) ed il riporto (Cout). La presenza del riporto in ingresso si spiega con la possibilità di collegare N Full - Adder in serie per ottenere Full - Adder a N bit.
- La logica del dispositivo è la seguente:
  - Sum = (A ⊕ B) ⊕ Cin
  - Cout = A * B + (A ⊕ B) * Cin

  e il conseguente schema logico è rappresentato in Figura.
- Si può descrivere il FA utilizzando la definizione di HA.



Addizionatore binario

- E’ possibile isolare il fattore **(a ⊕ b)**

- Rielaborando le precedenti espressioni è quindi possibile ottenere le seguenti espressioni per l’addizionatore completo:

$$S = (a \oplus b) \oplus r = H \oplus r$$

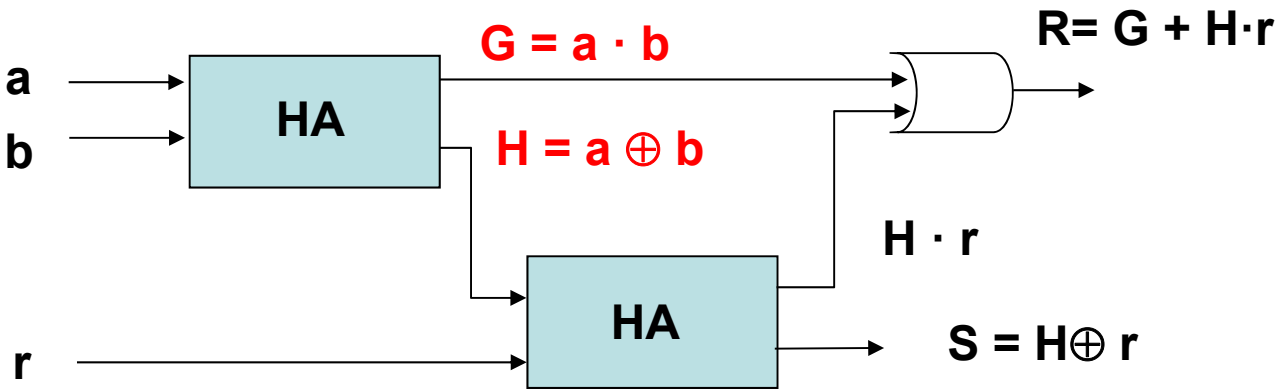
$$R = ab + r(a \oplus b) = G + rH$$

Addizionatore binario

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

$$S = (a \oplus b) \oplus r = H \oplus r$$

$$R = a \cdot b + r \cdot (a \oplus b) = G + r \cdot H$$



Addizionatore binario: riporto

- Le diverse componenti dell'espressione di R assumono un significato particolare:
  - **G = a·b** "**riporto generato**": indica la creazione di un riporto all'interno dell'addizionatore binario
  - **P = H = a⊕b** "**riporto propagato**": indica se, in presenza di un riporto in ingresso, lo stesso verrò propagato in uscita
  - Il riporto in uscita può quindi essere espresso come **R=G+Pr**

Addizionatore binario

- Per il semiaddizionatore valgono le eguaglianze

$$H = a \oplus b = d(a,b) = a\bar{b} + \bar{a}b$$

$$G = a \cdot b$$

- Similmente per l'addizionatore completo valgono le eguaglianze

$$S = a \oplus b \oplus r = d(a,b,r) = \bar{a}\bar{b}r + \bar{a}b\bar{r} + a\bar{b}\bar{r} + abr$$

$$R = \bar{a}br + a\bar{b}r + ab\bar{r} + abr = ab + br + ar = ab + r(a+b)$$

Addizionatori binari

$$n_i = \overline{r_i}$$

non-riporto

Indica assenza di riporto in ingresso

$$K_i = \overline{a_i} \cdot \overline{b_i}$$

Riporto “killed”

Indica che, indipendentemente dalla presenza di un riporto entrante, il riporto in uscita sarà comunque zero

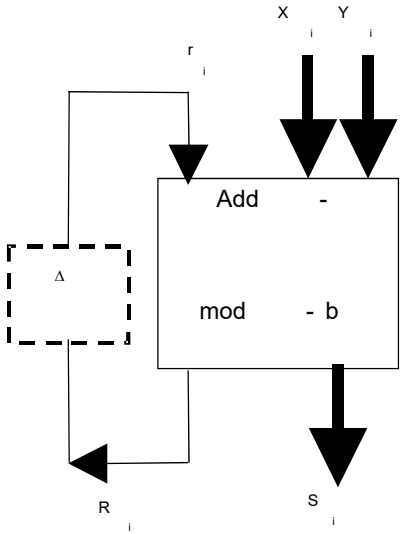
$$N_i = K_i + P_i \cdot n_i$$

Propagazione del non-riporto

Indica assenza di riporto in uscita

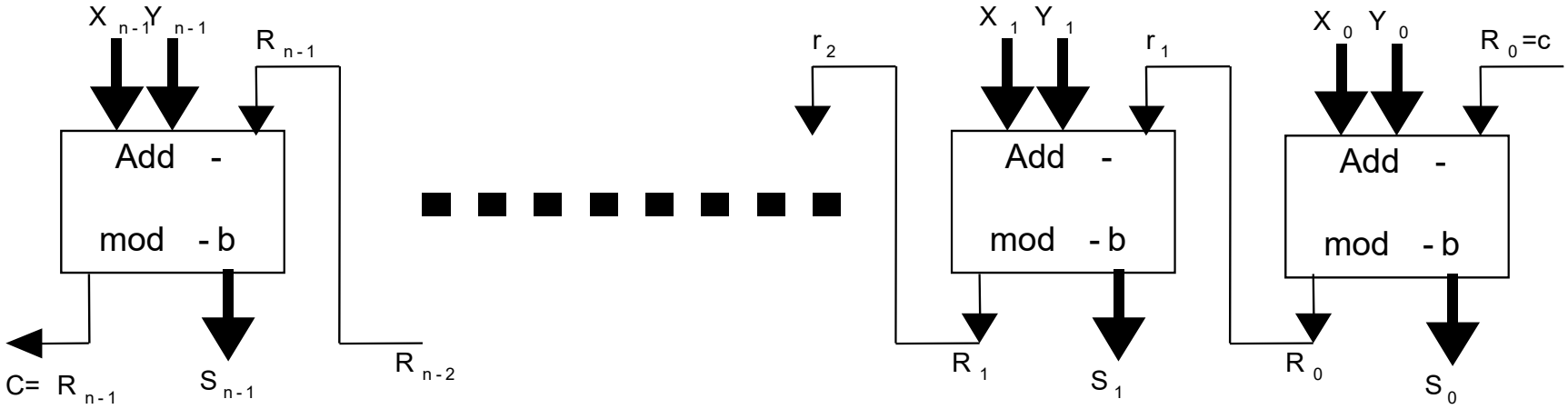
Addizionatori seriali

- Usa un unico addizionatore operante sulla singola cifra
- Opera in momenti successivi su cifre diverse degli addendi
- Richiede un blocco “con memoria”
- E’ lento rispetto ad addizionatori che lavorano in parallelo sulle diverse cifre degli addendi



Addizionatore binario parallelo

- Opera sulle cifre degli addendi in parallelo
- …anche se il riporto deve propagarsi attraverso l'intera struttura
- Richiede un numero maggiore di risorse rispetto all'addizionatore seriale



Addizionatore parallelo: tempo di risposta

- Gli addizionatori ottenuti collegando in cascata ***n*** addizionatori di cifra sono anche chiamati addizionatori a propagazione del riporto (*carry-ripple* o *carry-propagate*)
- ε = tempo di risposta di uno stadio
- Allo stadio i, il riporto uscente o è **generato** o è **ucciso** o è **propagato**
- Tempo di ritardo complessivo: **Limite inferiore** ε (in tutti gli stadi il riporto è generato o ucciso)
- Tempo di ritardo complessivo: **Limite superiore** $n\varepsilon$ (un riporto entrante nel primo stadio che è propagato in tutti gli stadi)
- Tempo di ritardo complessivo = $k\varepsilon$ ($k \leq n$), dove k è la più lunga catena di condizioni di propagazione.

Adder carry lookahead

$$r_{k+j} = G_{k+j-1} + G_{k+j-2}P_{k+j-1} + \ldots + G_k P_{k+1} \ldots P_{k+j-1} + r_k P_k P_{k+1} \ldots P_{k+j-1}$$

r_{k+j} è alto se è verificata la condizione di generazione nell'ultimo stadio

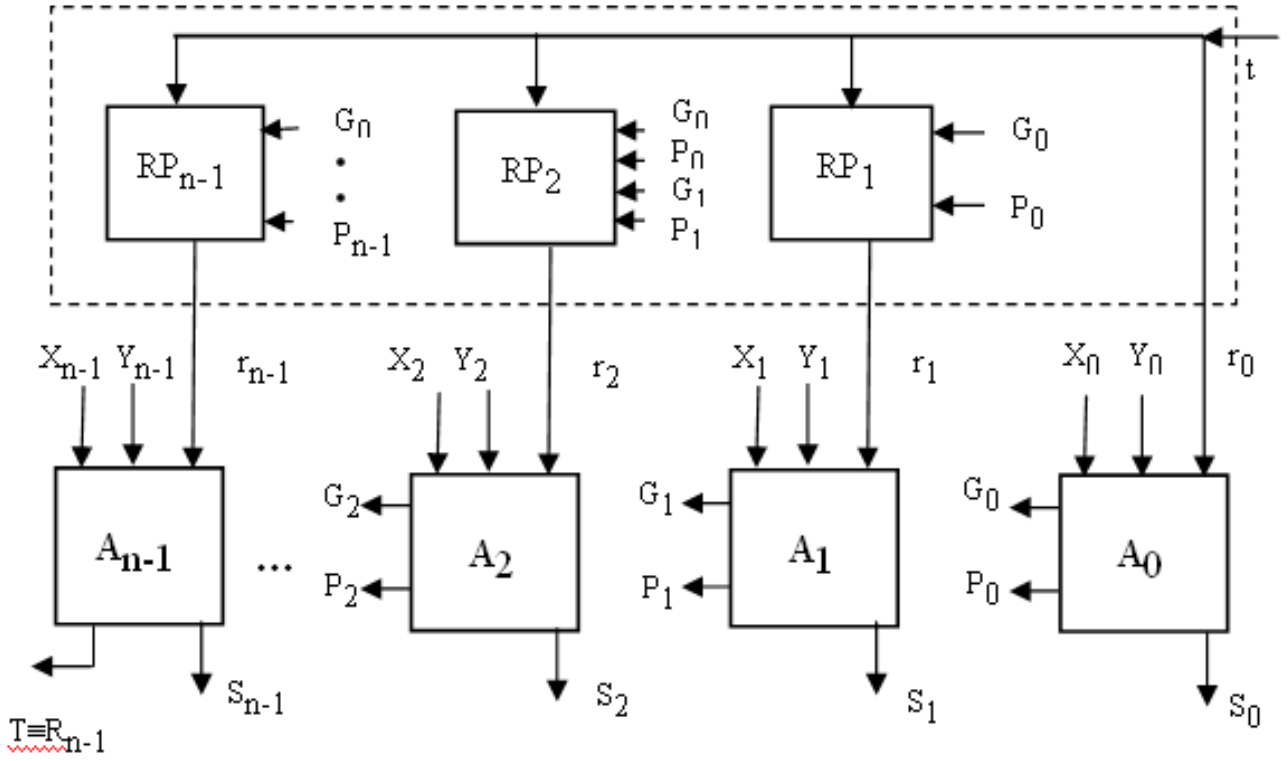
….
….

…oppure se è verificata la condizione di generazione G_k e se questa viene propagata dagli stadi dal (k+1)-esimo fino all'ultimo

…oppure è pari al riporto entrante r_k se questo viene propagato in tutti gli stadi

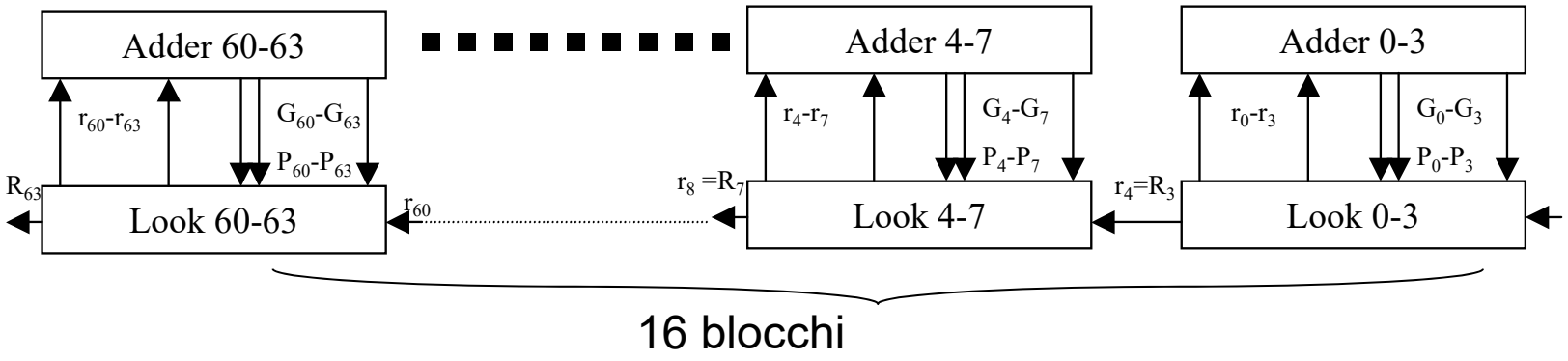
Addizionatori carry lookahead

- L'espressione precedente può essere realizzata nella maniera riportata in figura

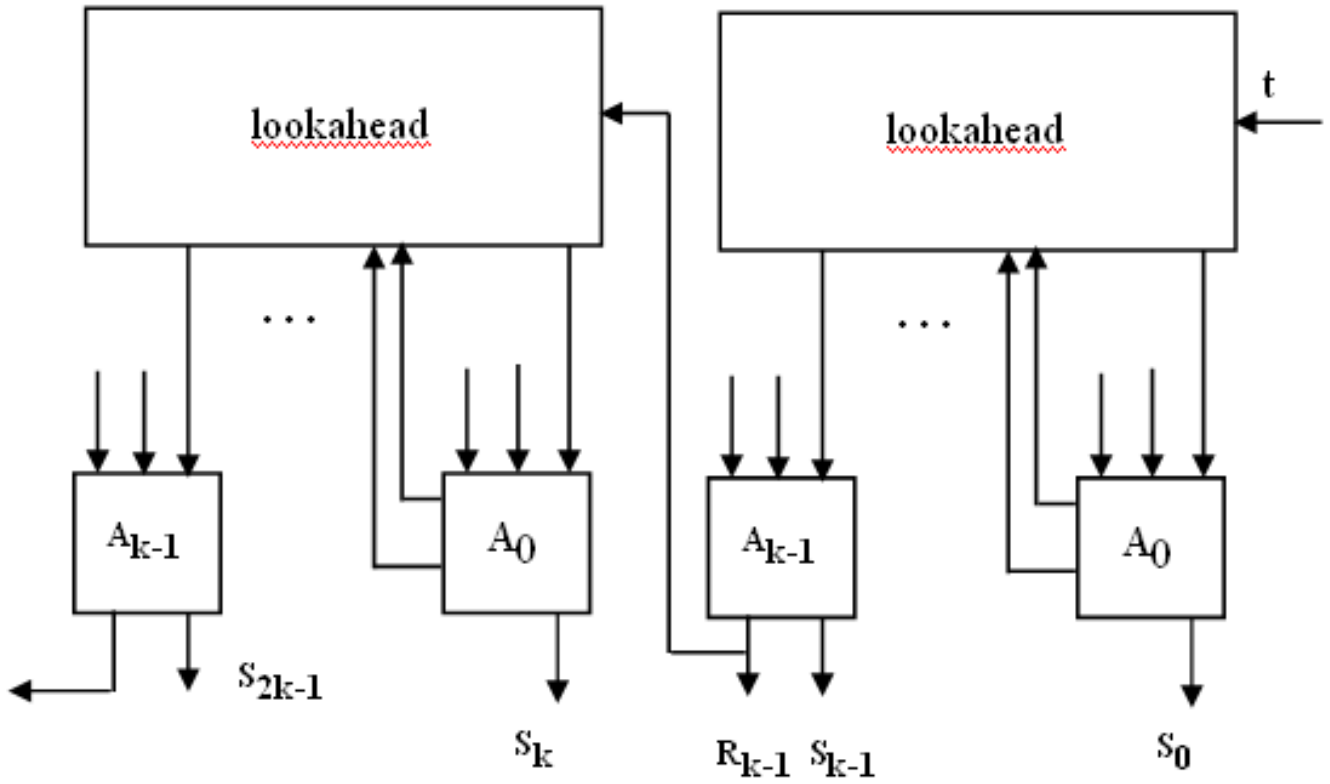


Addizionatori carry lookahead

- L'idea di base negli addizionatori carry lookahead è quella di calcolare la relazione tra r_k ed r_{k+j} separatamente per ogni gruppo di cifre $k+1 \ldots k+j$
- Una rete a livello superiore valuta la propagazione del carry tra *gruppi* di cifre
- Ad esempio, per un adder a 64 bit suddiviso in blocchi di quattro cifre (bit), la parte di "*look*" lungo cui si propaga il riporto è lunga 64/4=16 stadi

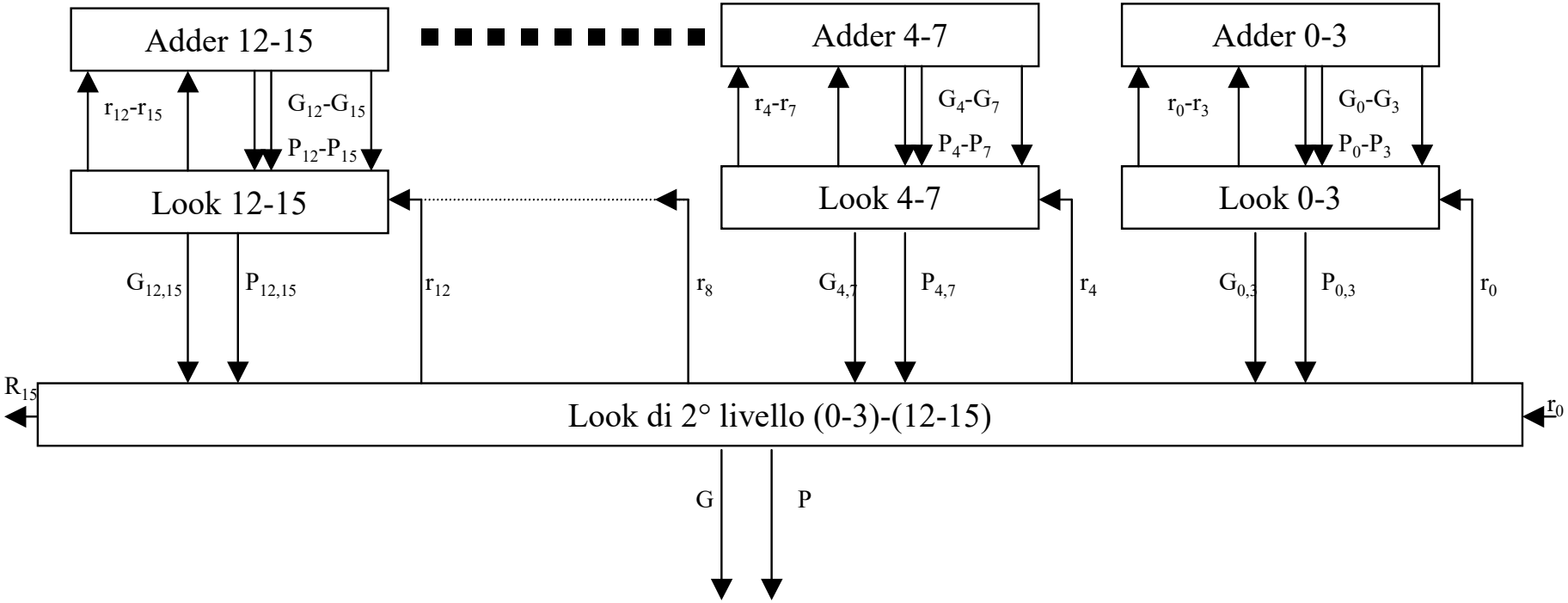


Addizionatori carry lookahead



Addizionatori carry lookahead

- La rete di lookahead può a sua volta essere realizzata a più livelli, secondo lo stesso principio visto prima



Porte di parola

- Porte con abilitazione:
  - B=αA = α **AND** A
- Parola:
  - Vettore di bit
  - V = {v_o, v_1, …, v_n}
- Porta di parola con abiitazione:
  - α V = {α v_o,…, α v_n}
- Porta generica di parola:
  - A **AND** B =
    = {a_0 **AND** b_0,…, a_n **AND** b_n}