$$f(x_1,x_2,x_3,x_4) = \overline{x_3}\left(\overline{x_1 x_4}\right) + x_3\left(\overline{x_2\left(\overline{x_1 \overline{x_4}}\right)}\right) =$$

Rappresentarla con multiplexer con soltanto 2 linee di controllo

$$= \overline{x_3}\left(x_1 + \overline{x_4}\right) + x_3\left(x_2 + \left(x_1 \overline{x_4}\right)\right) =$$

$$= x_1 \overline{x_3} + \overline{x_3}\,\overline{x_4} + x_2 x_3 + x_1 x_3 \overline{x_4} =$$

$$= x_1 \overline{x_2} x_3 \overline{x_4} + x_1 \overline{x_2} \overline{x_3} x_4 + x_1 x_2 \overline{x_3} \overline{x_4} +$$

$$+ x_1 x_2 \overline{x_3} x_4 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} + \overline{x_1} x_2 x_3 x_4 +$$

$$+ x_1 \overline{x_2} x_3 x_4 + x_1 x_2 \overline{x_3} x_4 + \overline{x_1} x_2 x_3 \overline{x_4} +$$

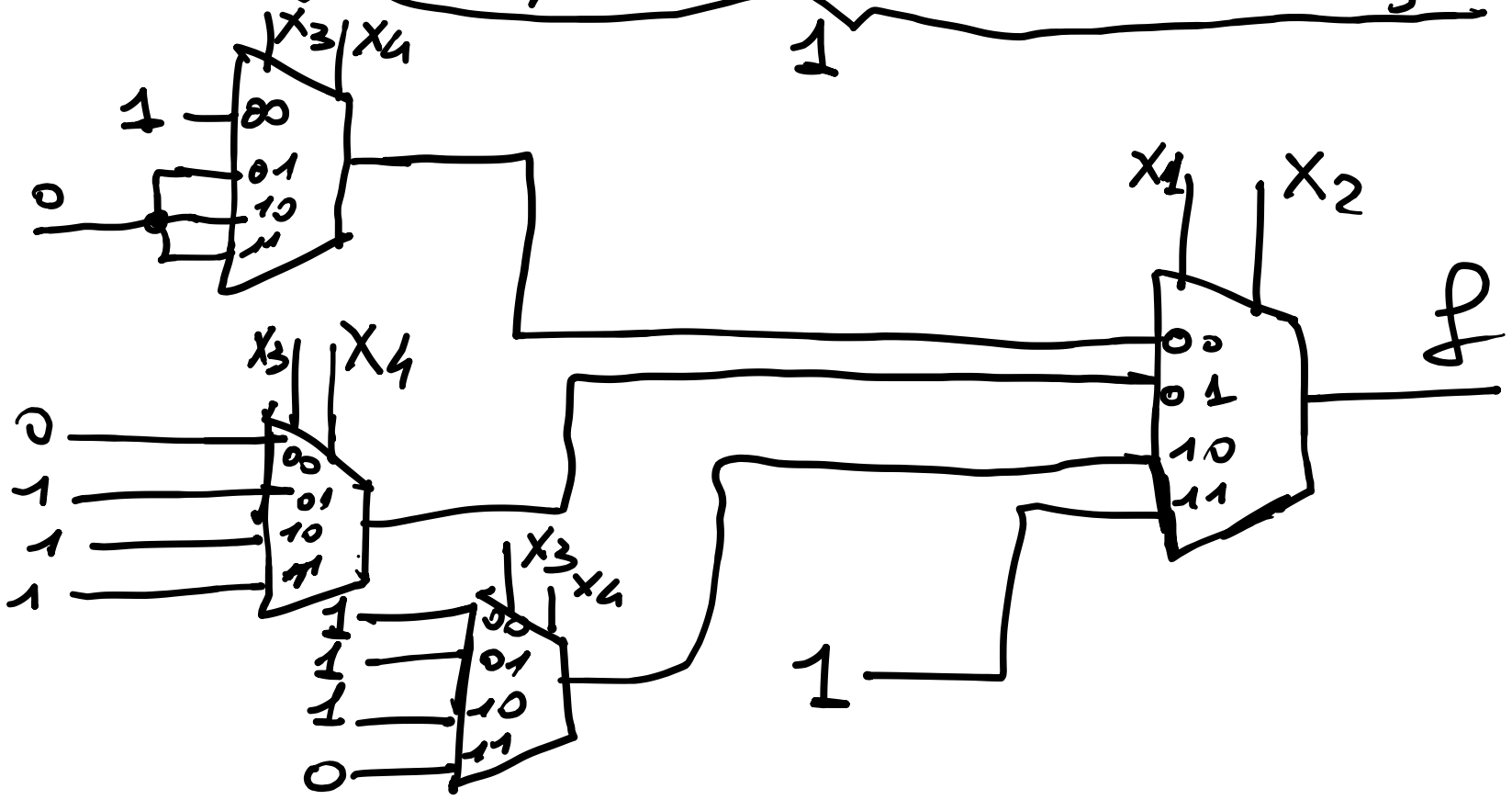
$$+ \overline{x_1} x_2 x_3 x_4 + x_1 x_2 x_3 \overline{x_4} + x_1 x_2 x_3 x_4 +$$

$$+ x_1 \overline{x_2} x_3 \overline{x_4} + x_1 x_2 x_3 x_4$$

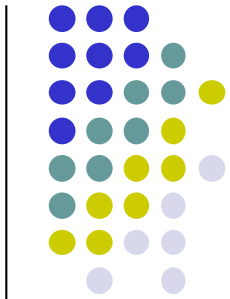
$$= x_1\bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3x_4 + x_1x_2\bar{x}_3\bar{x}_4 +$$
$$+ x_1x_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_2x_3x_4 +$$
$$+ x_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2x_3\bar{x}_4 +$$
$$+ \bar{x}_1x_2x_3x_4 + x_1x_2x_3\bar{x}_4 + x_1x_2x_3x_4 +$$
$$+ x_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 =$$

$$= \bar{x}_1\bar{x}_2(\bar{x}_3\bar{x}_4) +$$
$$+ \bar{x}_1x_2(\bar{x}_3x_4 + x_3\bar{x}_4 + x_3x_4) +$$
$$+ x_1\bar{x}_2(\bar{x}_3\bar{x}_4 + \bar{x}_3x_4 + x_3\bar{x}_4) +$$
$$+ x_1x_2(\bar{x}_3\bar{x}_4 + \bar{x}_3x_4 + x_3\bar{x}_4 + x_3x_4)$$

$$= \overline{x_1}\,\overline{x_2}\left(\overline{x_3}\,\overline{x_4}\right) +$$
$$+ \overline{x_1}\,x_2\left(\overline{x_3}\,x_4 + x_3\,\overline{x_4} + x_3 x_4\right) +$$
$$+ x_1\,\overline{x_2}\left(\overline{x_3}\,\overline{x_4} + \overline{x_3}\,x_4 + x_3\,\overline{x_4}\right) +$$
$$+ x_1 x_2\left(\underbrace{\overline{x_3}\,\overline{x_4} + \overline{x_3}\,x_4 + x_3\,\overline{x_4} + x_3 x_4}_{1}\right)$$



Memorie ROM (Read Only Memory)



- La realizzazione di qualsiasi funzione di *m* variabili richiede un numero di porte AND pari al numero dei suoi mintermini e di “prolungare” le uscite di tali porte ad una porta OR

- Fissare le linee prolungate equivale a *programmare* il comportamento della rete

- Tale programmazione potrebbe essere operata direttamente dal costruttore di integrati come mostrato nella figura che segue

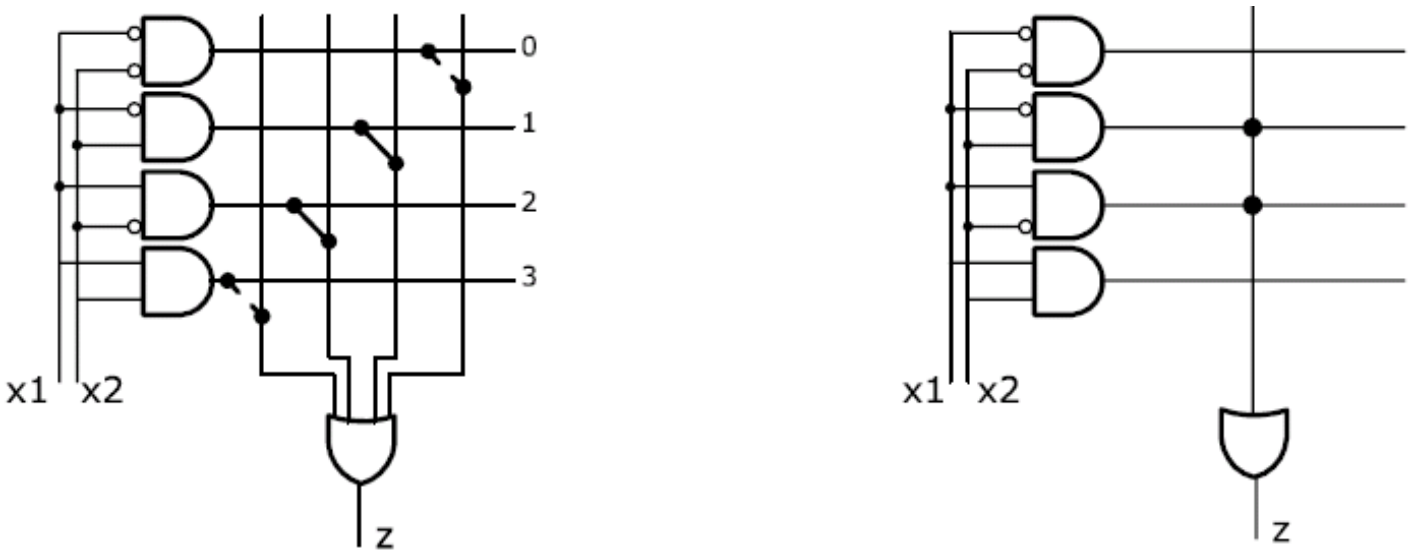
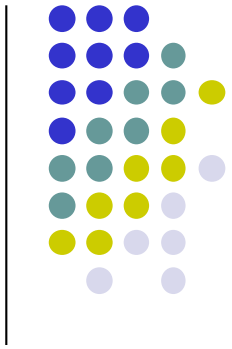


Figura 3.36 Matrice di contatti che consente di ottenere qualunque funzione logica dei due variabili. Nel caso specifico la rete ha come uscita $z = \overline{x}_1 x_2 + x_1 \overline{x}_2 = x_1 \oplus x_2$. A destra viene riportato lo schema sintetico che rappresenta la rete; i pallini indicano gli incroci in cui si ha contatto. Lo schema a sinistra ha valore di principio in quanto la porta OR avrebbe due ingressi "volanti", cosa sconsigliata perché non è definito il comportamento della porta stessa. Nella pratica la rete deve essere costruita in modo da dare continuità elettrica, verso massa o verso l'alimentazione, di ogni suo ingresso.



- In riferimento al circuito visto precedentemente in figura 3.33 che sfrutta un multiplexer per realizzare una funzione di *3* variabili, questa soluzione richiederebbe l'utilizzo di un integrato con *3+1=4* piedini invece di 8+*3+1=12*

- Con *12* piedini sarebbe possibile realizzare addirittura qualsiasi funzione di *11* variabili, con evidenti vantaggi per il progettista

- Estendendo tale soluzione in modo da fornire *m* ingressi e *k* uscite, si ottiene una rete che ad ogni configurazione degli *m* ingressi associa una configurazione delle *k* uscite

- Si ottiene così una memoria a sola lettura (ROM) di 2^m celle (ognuna di k bit), che ad ogni indirizzo (configurazione di ingressi) associa il contenuto della relativa cella (configurazione di uscite)

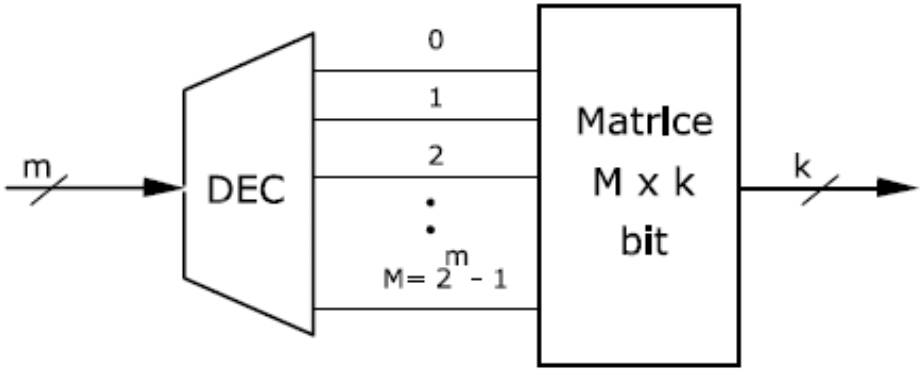


Figura 3.37 Schema a blocchi di una memoria ROM.

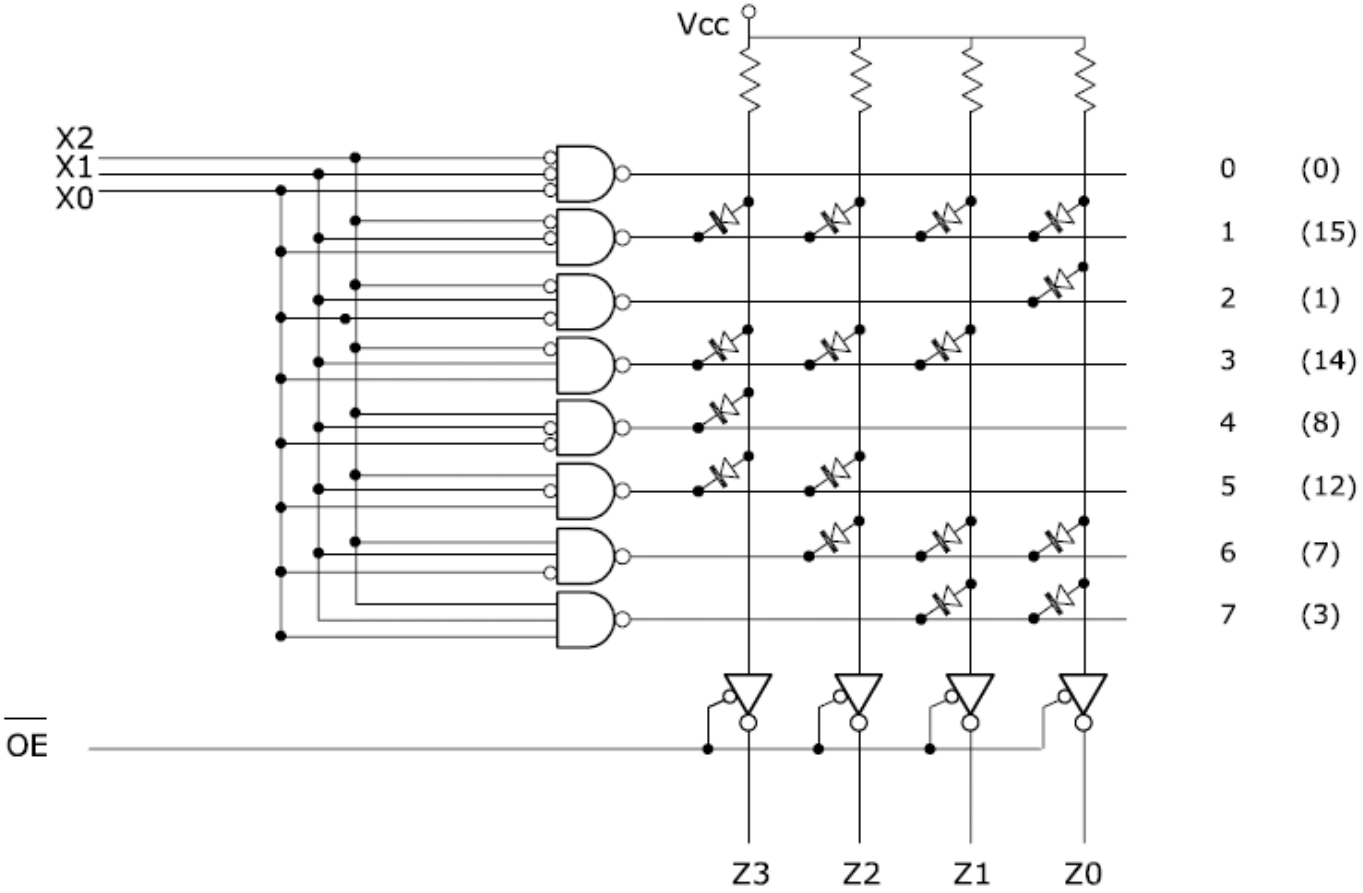
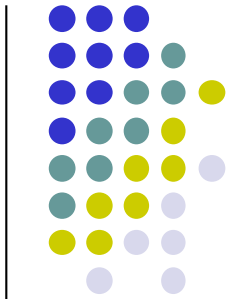


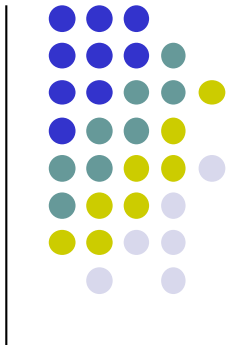
Figura 3.38 Schema di una ROM di 8 posizioni di 4 bit. A destra viene riportato l'indirizzo di ciascuna cella e, tra parentesi, il suo contenuto. Per capirne il funzionamento si consideri ad esempio il caso in cui l'indirizzo $x_2x_1x_0$ è 2. In tal caso solo la riga 2 della matrice è asserita bassa, per cui la tensione sull'anodo del diodo ad essa collegato (colonna z_0) risulta bassa (il diodo ha una caduta trascurabile). Gli altri diodi sulla colonna z_0 risultano interdetti. I diodi sulle restanti colonne sono in conduzione, ma la tensione sui loro anodi è alta essendo alta la tensione su tutte le righe diverse dalla 2. Si noti che la memoria ha un segnale di abilitazione delle uscite.



- Nelle ROM il costruttore stabilisce i contatti su richiesta dell'utente, senza possibilità di ulteriori modifiche

- Nelle PROM (Programmable ROM), l'utente può programmare ponendo bit a 0 attraverso la fusione del relativo fusibile

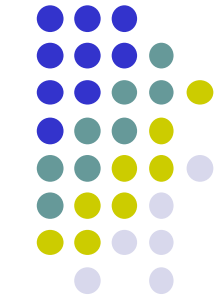
Figura 3.39 Schematizzazione della modalità di interconnessione tra righe e colonne per le ROM (a sinistra) e per le PROM (a destra).



- Nelle EPROM (Erasable PROM) attraverso una tecnologia che fa uso di piastrine di quarzo e raggi ultravioletti, è possibile ripristinare la programmabilità cancellando quella precedente (conveniente per prototipi di laboratorio e bassi volumi di produzione)

- Nelle EEPROM (Electrically Erasable PROM) la riprogrammazione è consentita per via elettrica, per cui sono adatte per l'impiego in apparati capaci di modificarne il contenuto

Matrici di logica programmabili



- La matrice programmabile che abbiamo appena visto nelle ROM è detta di OR, perché realizza l'OR dei contatti stabiliti

- Entrando nel dettaglio del decodificatore, possiamo identificare al suo interno una matrice simile detta di AND che esegue l'AND dei contatti stabiliti

- Rivediamo più in dettaglio lo schema circuitale corrispondente alla precedente Figura 3.37

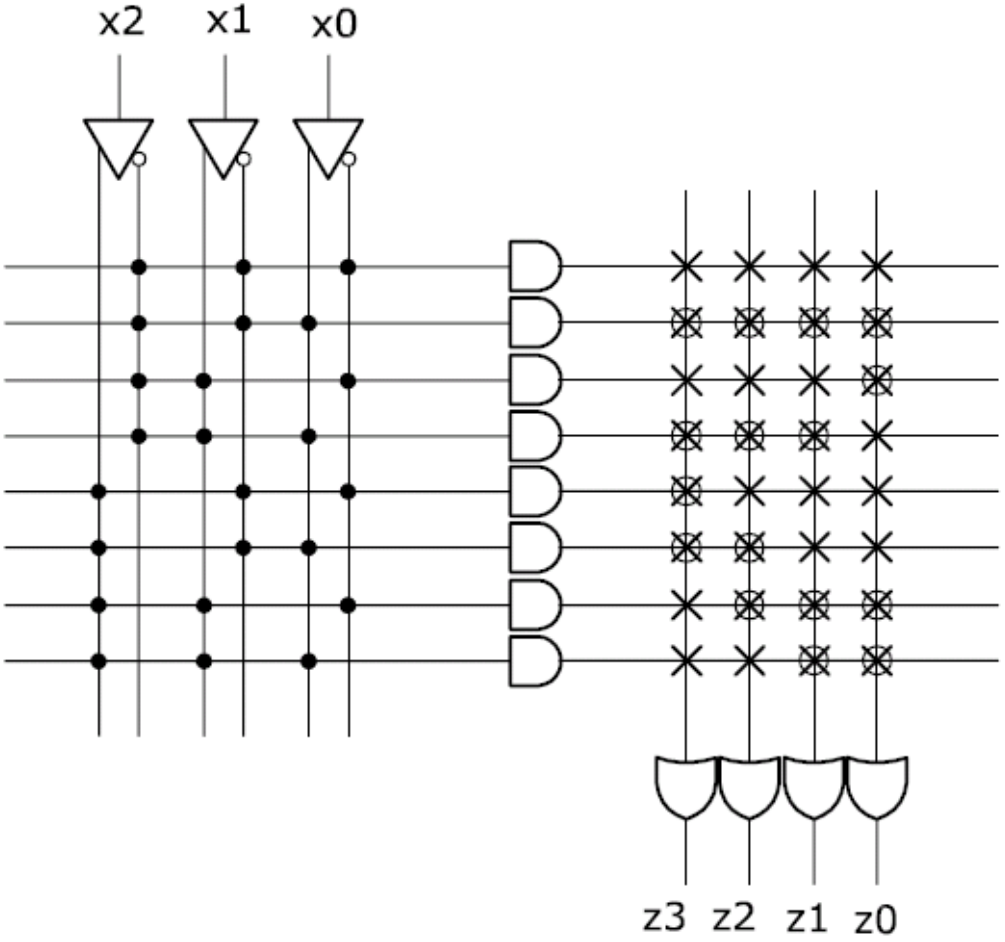
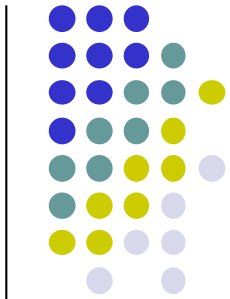


Figura 3.40 Schema di una PROM 8 per 4. Gli incroci della matrice programmabile sono stati indicati con "×". Quelli in cui è tracciato anche un cerchio sono quelli in cui c'è continuità elettrica, ovvero contengono un 1. La programmazione corrisponde a quella di Figura 3.38. Si noti il nuovo simbolo usato per la rappresentazione di un segnale e del suo negato.



A seconda della programmabilità o meno di tali matrici abbiamo:

- PROM: matrice di AND fissa e di OR programmabile
- PLA: matrice di AND programmabile e di OR programmabile
- PAL: matrice di AND programmabile e di OR fissa

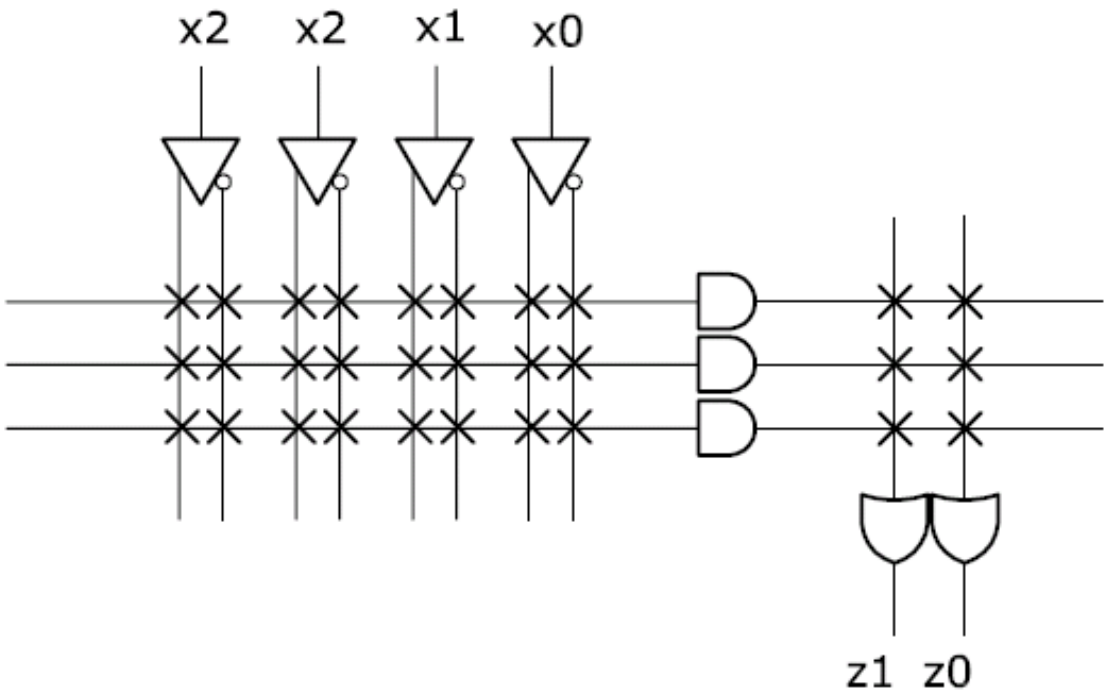
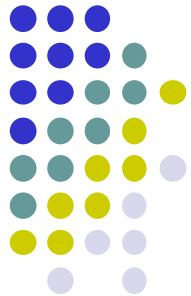


Figura 3.41 Schema di un dispositivo PLA a quattro ingressi e due uscite. Il dispositivo prevede una matrice di AND e una matrice di OR, ambedue programmabili in campo.

Unità aritmetiche e logiche (ALU)

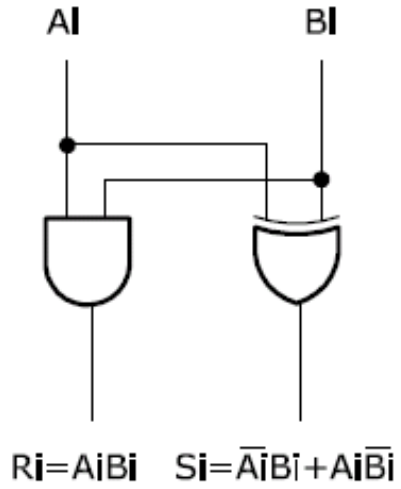


- Abbiamo già considerato l'aritmetica binaria
- Come già sottolineato, le costanti logiche *0* e *1* non hanno niente a che fare con le corrispondenti cifre binarie
- Le tabelline aritmetiche contengono comunque gli stessi simboli *0* e *1*
- Esse possono essere interpretate come tabelle di verità di funzioni logiche corrispondenti
- E' possibile quindi realizzare tramite circuiti logici
- Iniziamo a identificare una serie di componenti utili per costruire una piccola ALU semplificata

Semisommatore (Half Adder)

- Calcola la somma di *2* bit e l'eventuale riporto

Ai	Bi	Si	Ri
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



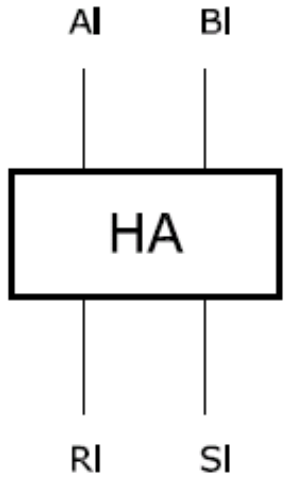


Figura 3.46 A sinistra viene riportata la tabellina aritmetica della somma e del riporto di due bit. La tabella in questione può essere interpretata come la tabella di verità delle due funzioni logiche S_i e R_i. Al centro viene data la rete corrispondente. A destra viene data una schematizzazione della rete come blocco funzionale. Il blocco è stato indicato come HA (da *Half Adder*), per *Semisommatore*.

Somma di due numeri interi

- Per sommare bit a bit *2* numeri interi di *n* bit $A=A_{n-1}\ldots A_0$ e $B=B_{n-1}\ldots B_0$ bisogna sommare bit a bit tenendo conto del riporto dovuto alla somma dei precedenti
- Per ogni coppia di bit la somma è effettuata da un circuito chiamato *Sommatore completo (Full Adder)*, che estende il semisommatore mediante l'aggiunta di un ingresso recante il riporto generato dalla somma dei bit precedenti

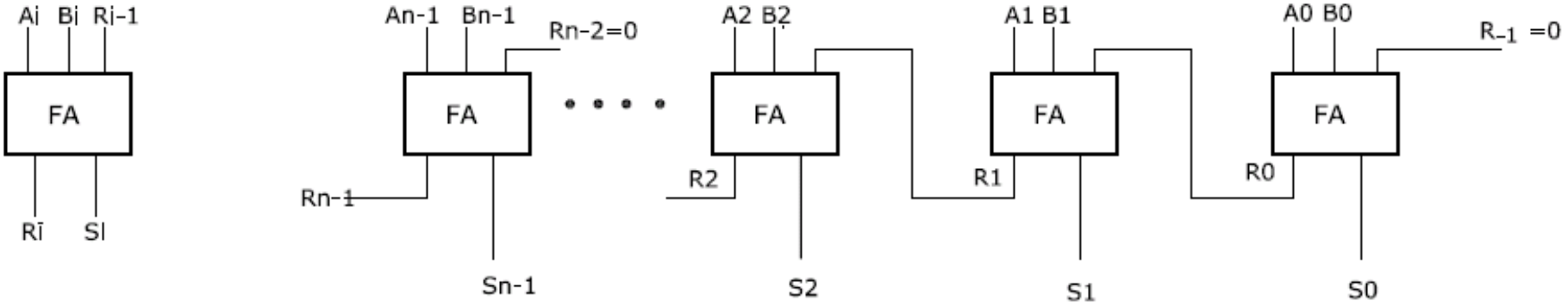
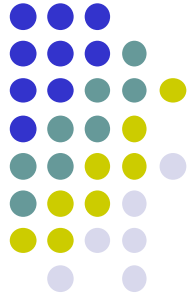


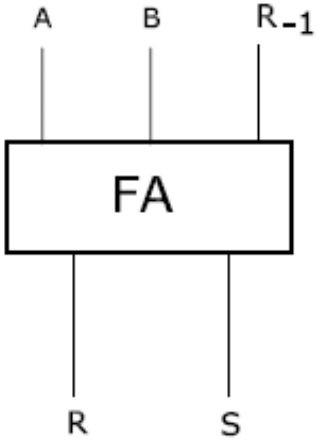
Figura 3.47 Schema di un sommatore di parole di n bit, detto sommatore di *ripple*, costruito impiegando la cella detta *Sommatore completo* (FA).

Sommatore completo (Full Adder)



- Vediamo più in dettaglio lo schema di un sommatore completo
- La somma dei *3* bit A_i,B_i e R_{i-1} (riporto precedente) è pari ad 1 solo se è dispari il numero di essi pari ad *1*, ossia quando è *1* l'OR esclusivo dei tre bit
- Quindi $S_i=A_i \oplus B_i \oplus R_{i-1}=(A_i \oplus B_i) \oplus R_{i-1}$
- Il riporto successivo R_i vale *1* se A_i e B_i generano direttamente riporto o se uno di essi è pari ad *1* ed il riporto precedente in ingresso è pari ad *1*
- Quindi $R_i=A_iB_i+(A_i \oplus B_i)R_{i-1}$
- Lo schema circuitale corrispondente è il seguente

A_i	B_i	R_{i-1}	S_i	R_i
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



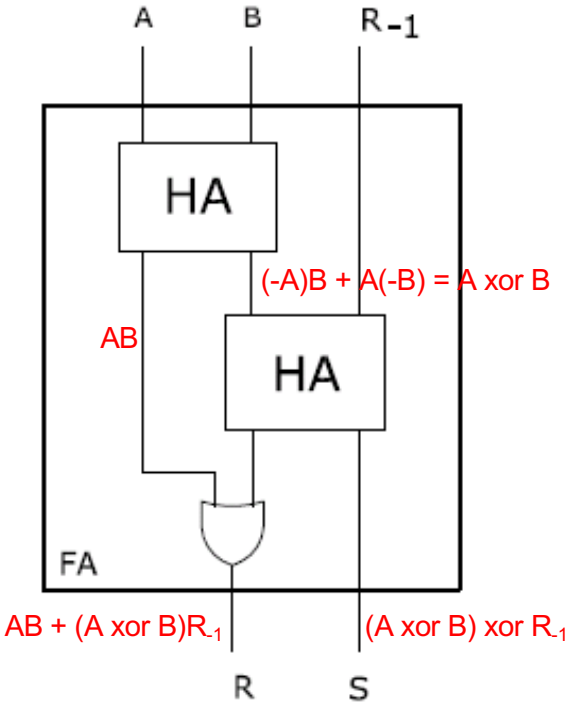
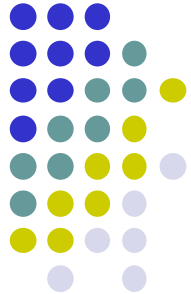


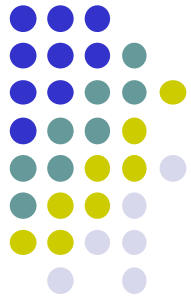
Figura 3.48 A sinistra la tabella di verità per il sommatore completo. Al centro la schematizzazione come blocco funzionale. A destra il sommatore completo costruito con due semisommatori.

Calcolo anticipato del riporto



- Uno dei problemi del circuito sommatore in Figure 3.47 (slide 233) è il tempo di commutazione
- Infatti il riporto può propagarsi dal bit meno significativo fino all'ultimo
- Se τ è il tempo di commutazione di una singola porta, il sommatore completo impiega tempo $\tau_R=3\tau$ per calcolare il riporto e $\tau_S=2\tau$ per la somma
- Quindi il tempo di calcolo per un circuito sommatore di *n* bit è $3\tau(n-1)+2\tau=(3n-1)\tau$
- Tale tempo di commutazione può risultare eccessivo, per cui sono state messe a punto tecniche per il calcolo anticipato del riporto

Una ALU semplificata



- Se rappresentiamo gli interi in complemento a *2*, per effettuare la sottrazione *A-B* basta effettuare la somma *A+(-B)*

- Per ottenere la rappresentazione di *–B* basta introdurre quel poco di logica che serve a complementare *B* bit a bit e a sommare *1*

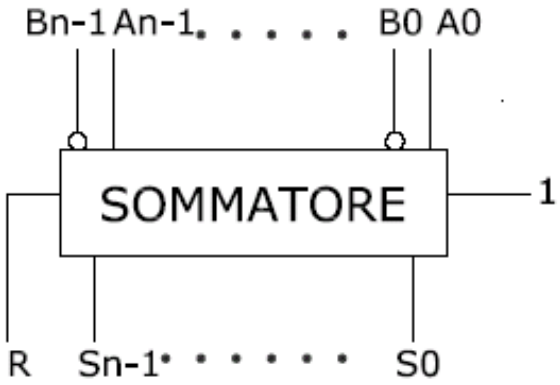
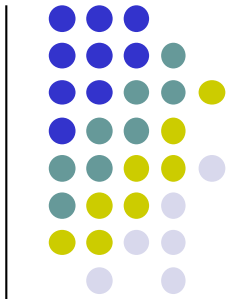


Figura 3.55 Schema della rete che effettua la somma usando un addizionatore.



- Invece di utilizzare la rete bloccata precedente, conviene prevedere linee di controllo per poter usare lo stesso circuito per somme e sottrazioni

- Nello schema che stiamo per vedere
  - la linea c_0 seleziona *B* o il suo complemento
  - la linea c_1 seleziona *A* o *0*

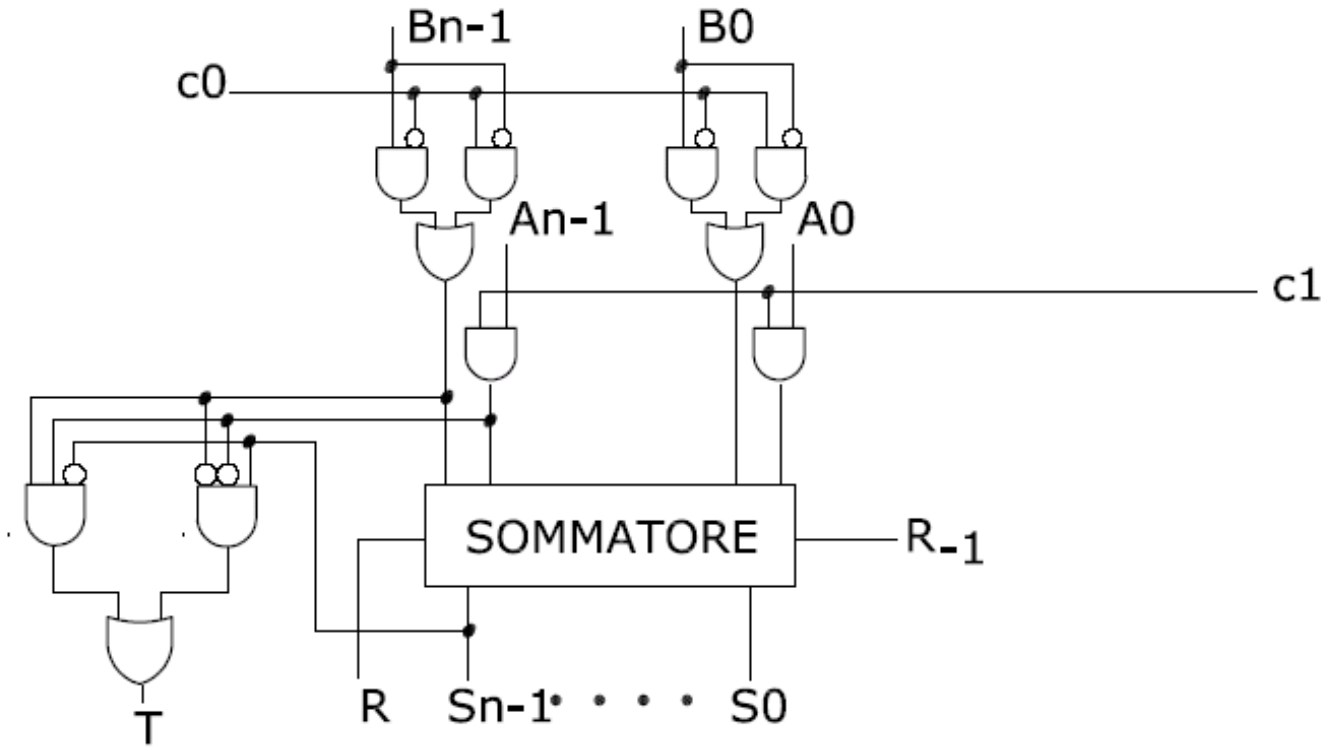
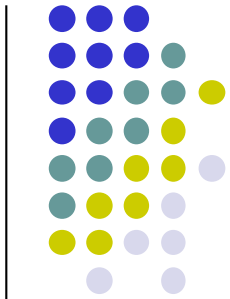


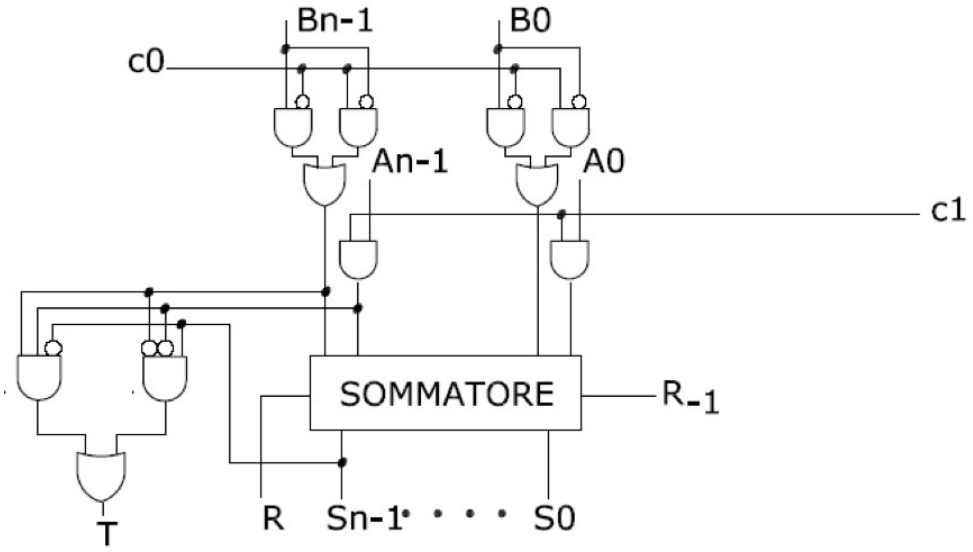
Figura 3.56 Dettaglio della rete che effettuata la sottrazione. L'ingresso B può essere presentato al sommatore in forma complementata o no a seconda di c_0. L'altro ingresso, controllato tramite la linea c_1, è A oppure 0. Rispetto alla Figura 3.55 è stata aggiunta la logica per il controllo del trabocco di cui si parla più avanti.



Si noti che

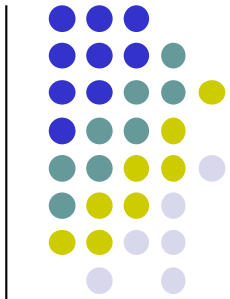
- per $c_1c_0R_{-1}=010$ la rete esegue la complementazione di B
- $c_1c_0R_{-1}=101$ *e* $c_1c_0R_{-1}=110$ producono risultati non particolarmente interessanti
- per tutte le altre combinazioni la rete effettua somme aritmetiche

La seguente tabella sintetizza il comportamento della rete



c_1	c_0	R_{-1}	Risultato	Commento
0	0	0	$S = 0 + B = B$	Selezione di B
0	0	1	$S = 0 + B + 1 = B + 1$	Incremento di B
0	1	0	$S = 0 + \overline{B} = \overline{B}$	Complementazione di B
0	1	1	$S = 0 + \overline{B} + 1 = -B$	Cambio segno di B
1	0	0	$S = A + B$	Somma $A + B$
1	0	1	$S = A + B + 1$	
1	1	0	$S = A + \overline{B} = A - B - 1$	
1	1	1	$S = A + \overline{B} + 1 = A - B$	Differenza $A - B$

Tabella 3.4 Operazioni effettuate dalla rete di Figura 3.56 a seconda degli ingressi di controllo c_1, c_0 e R_{-1}.

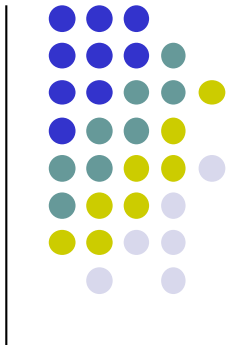


- Spieghiamo infine il ruolo delle *3* porte per il controllo del trabocco (overflow)

- Sommando due numeri si possono avere due risultati inaspettati:
  1. si sommano due interi positivi e il risultato è negativo
  2. si sommano due interi negativi e il risultato è positivo

- Ciò si verifica quando si ha una situazione di overflow

- Per rilevare l'overflow bastano quindi due porte AND a *3* ingressi collegate come in Figura 3.56



- In conclusione abbiamo ottenuto una piccola ALU
- A partire da un semplice sommatore, con l'aggiunta di poca logica è capace di svolgere addizioni, sottrazioni e un'operazione logica
- Riesce inoltre a verificare la correttezza del risultato
- Nella pratica non tutte le operazioni si riescono a fare con un'ALU combinatoria
- Può essere infatti che occorra eseguire una sequenza di operazioni attraverso reti sequenziali (che vedremo tra breve)
- E' il caso tipico delle operazioni in virgola mobile e della relativa unità deputata alla loro esecuzione, denominata FPU (Floating Point Unit)
- Una schematizzazione sintetica dell'ALU ottenuta è la seguente

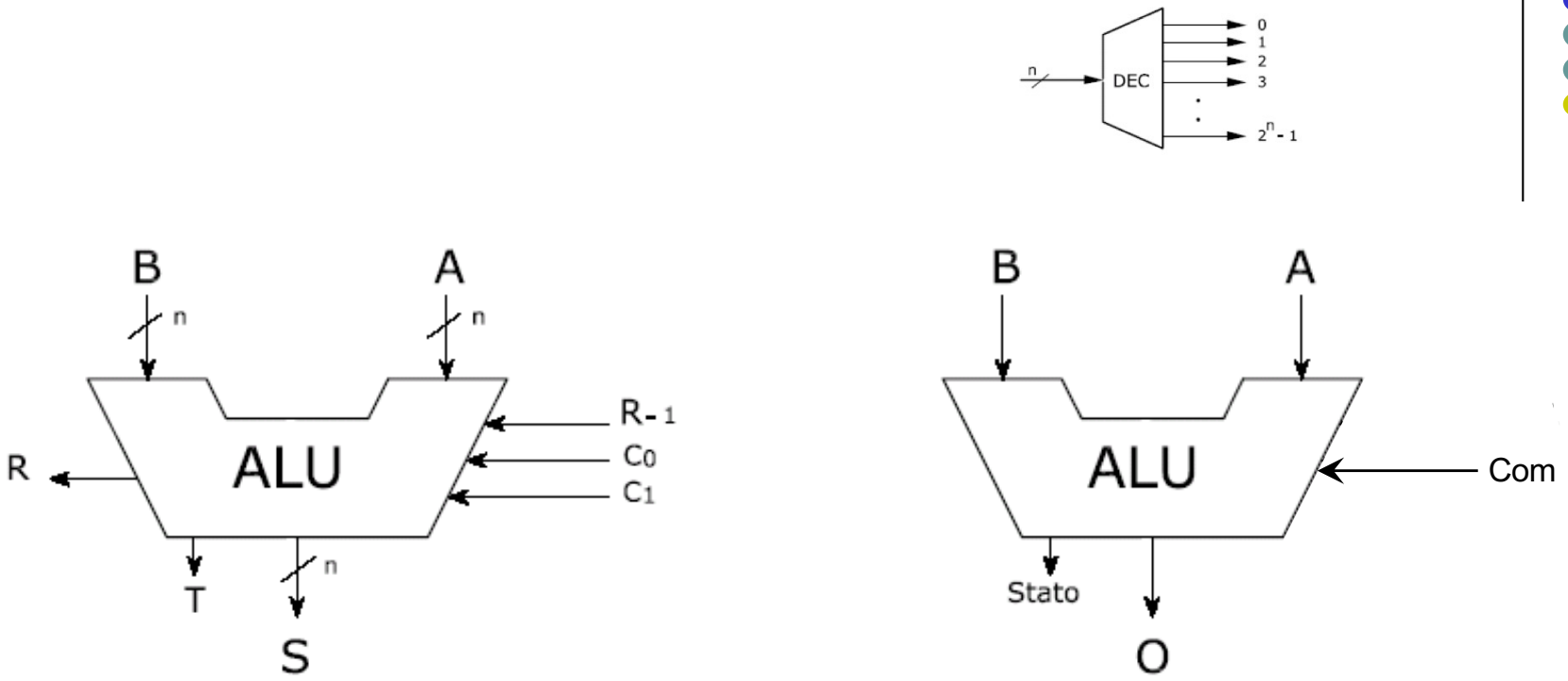


Figura 3.57 A sinistra viene riportata la schematizzazione della rete di Figura 3.56. A destra viene data la schematizzazione di una generica ALU: Com1, Com2, ..., ComN sono le linee di controllo che selezionano le varie operazioni. Qui le linee di comando sono esplicitate. Da un punto di vista realizzativo conviene portare alla ALU i comandi in forma codificata e prevedere sull'ALU un decodificatore per selezionare le singole linee.