Reti iterative

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

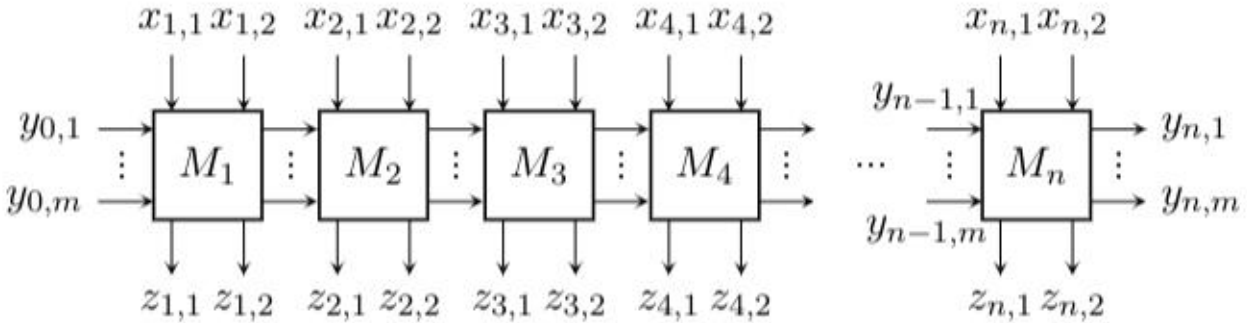
Reti combinatorie iterative

- I metodi di sintesi che abbiamo analizzato fino a questo punto permettono la sintesi di circuiti in cui sono poche le variabili in input
- Per realizzare una CPU, dobbiamo essere in grado di gestire dati a 16, 32, 64 bit
- Un circuito combinatorio realizzato a partire da un numero così grande di variabili può essere complesso da sintetizzare

- Possiamo organizzare i circuiti in maniera *iterativa*
- Ciò significa che uno stesso circuito elementare tratta un sottoinsieme dei bit dei dati, riducendo il numero di variabili
- Più circuiti elementari sono interconnessi tra loro, per calcolare la funzione finale

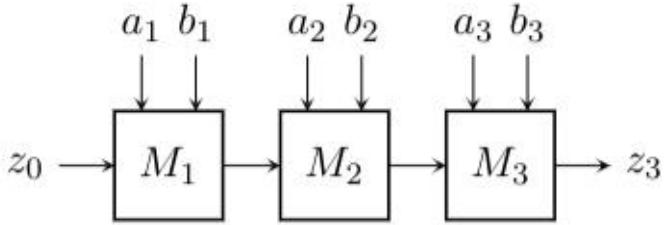
Reti combinatorie iterative

- Vettore $\boldsymbol{y}$: rappresenta le informazioni di stato trasferite da un modulo al successivo
  - L'ultimo modulo può esporre parte di questa informazione all'esterno, ad esempio per notificare dettagli circa il risultato finale dell'operazione
- Vettore $\boldsymbol{x}$: rappresenta il dato in input, decomposto tra i vari moduli
- Vettore $\boldsymbol{z}$: rappresenta l'output, calcolato iterativamente dai moduli

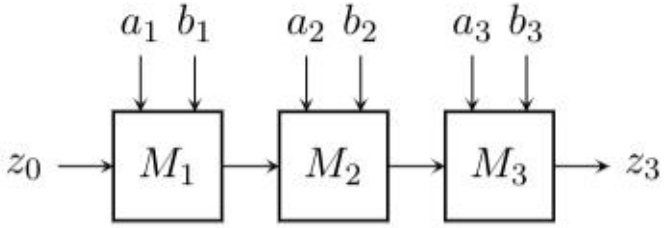


Comparatori

- I comparatori sono dei circuiti che confrontano il valore di due numeri, A e B, rappresentati in formato binario
- Il risultato di un comparatore determina se $A = B$
- Il confronto può essere effettuato su ciascuna coppia di bit in moduli separati
- Tuttavia, è necessario *propagare* il risultato della comparazione dai moduli precedenti



Comparatori



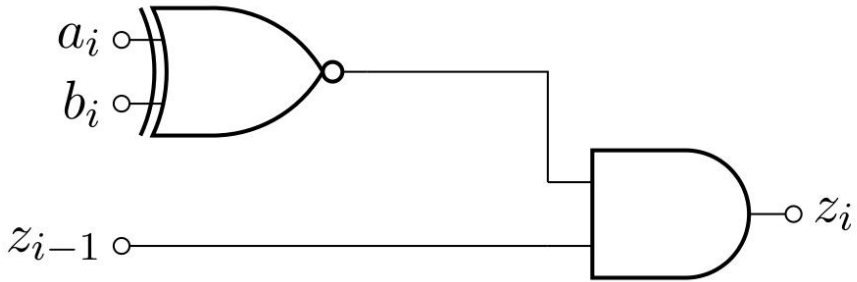
z_{i-1}	a_i	b_i	z_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- L'uscita è 1 se e solo se riceviamo un 1 dagli stadi precedenti (tutti i bit precedenti sono uguali) e se i due bit analizzati nel modulo corrente sono uguali
- I mintermini della funzione sono soltanto due:

$$z_i = z_{i-1}\overline{a}\,\overline{b} + z_{i-1}ab = z_{i-1}(a \odot b)$$

Comparatori

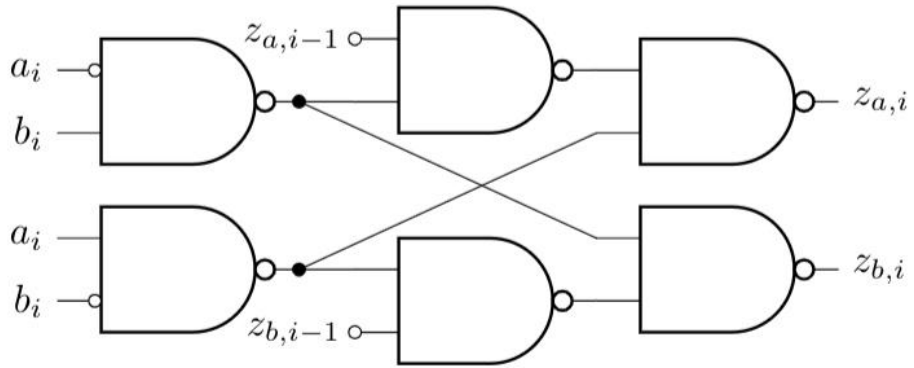
- La realizzazione circuitale del modulo M_i è immediata:



- È interessante però realizzare un comparatore che possa discriminare:
  - $A = B$: codifica di output 00
  - $A > B$: codifica 10
  - $A < B$: codifica 01

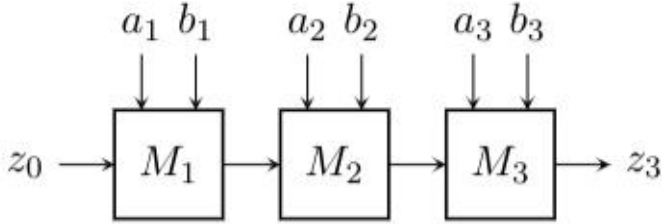
Comparatori

- Esercizio: mostrare che, data le configurazioni ammissibili, le espressioni minime per calcolare le uscite del modulo M_i corrispondono a:
  - $z_{a,i} = z_{a,i-1}(a_i + \overline{b_i}) + a_i\overline{b_i}$
  - $z_{b,i} = z_{b,i-1}(b_i + \overline{a_i}) + b_i\overline{a_i}$
- e che il circuito equivalente è:



Problema delle reti iterative

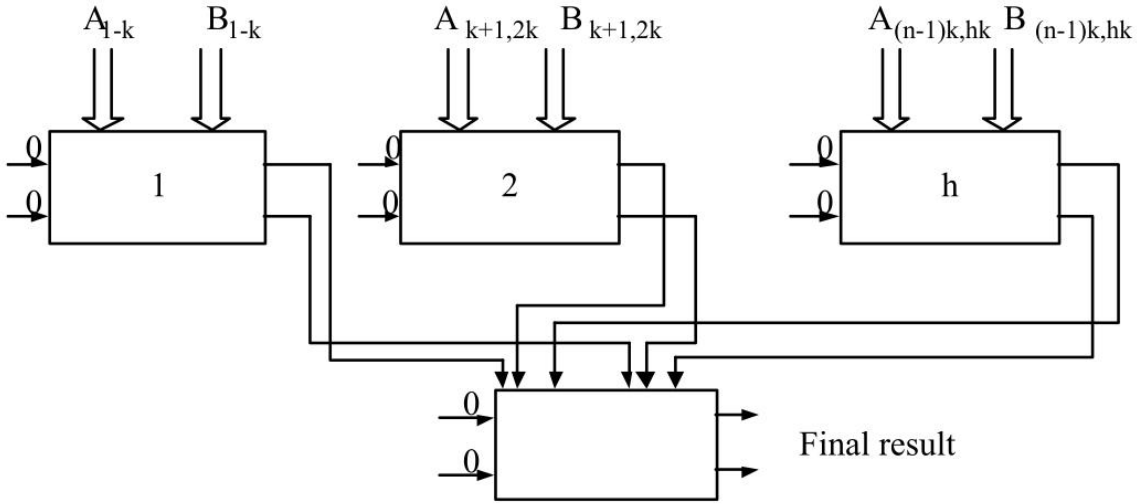
- Analizzando la struttura del comparatore realizzato è evidente quale sia il limite di queste reti



- Il tempo di calcolo della funzione può diventare inaccettabile se il numero di bit da processare è troppo elevato

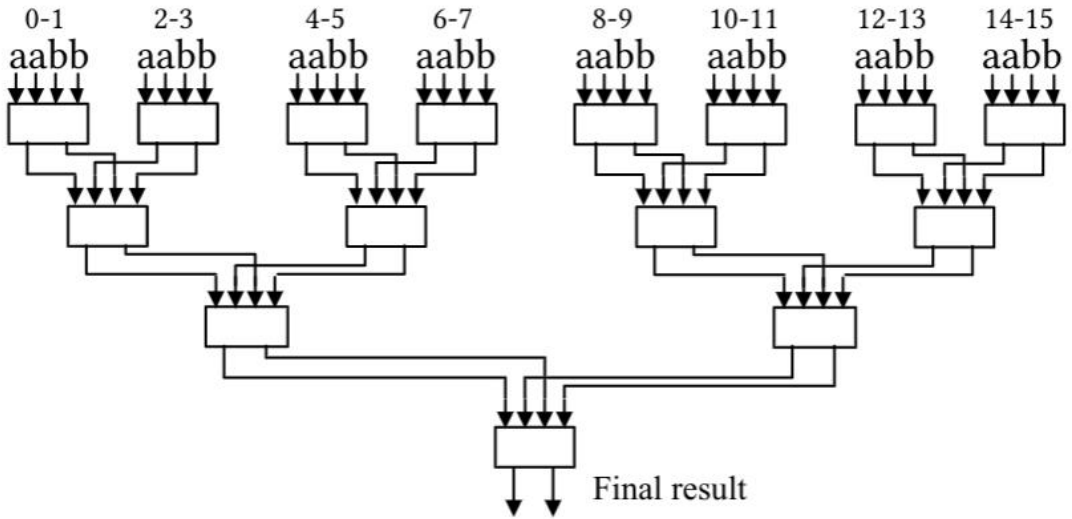
Comparatore veloce

- I bit dei numeri da confrontare vengono divisi in *h* blocchi di *k* bit
- Ciascun blocco viene confrontato da un comparatore iterativo dedicato
- Le uscite dei vari comparatori vengono processate da un comparatore aggiuntivo



Comparatore veloce ad albero

- Poiché il numero di bit è tipicamente una potenza di due, si possono organizzare i comparatori in una struttura ad albero a più livelli
- Ad esempio, per interi a 16 bit:

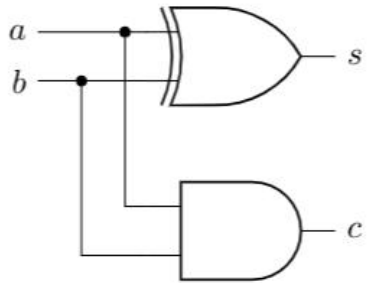


Half Adder

- Il circuito più semplice per effettuare una somma di operandi ad un solo bit deve calcolare il valore della somma e il valore del riporto:

$$s = a \oplus b \qquad c = a \cdot b$$

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full Adder

- Per calcolare la somma di un intero a n bit possiamo realizzare una rete iterativa composta da n sommatori
- Il circuito del modulo va modificato per considerare anche il riporto proveniente dai moduli precedenti

$$s_i = f_1(a_i, b_i, c_{i-1})$$

c_{i-1} \ a_ib_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$s_i = c_{i-1} \oplus a_i \oplus b_i$$

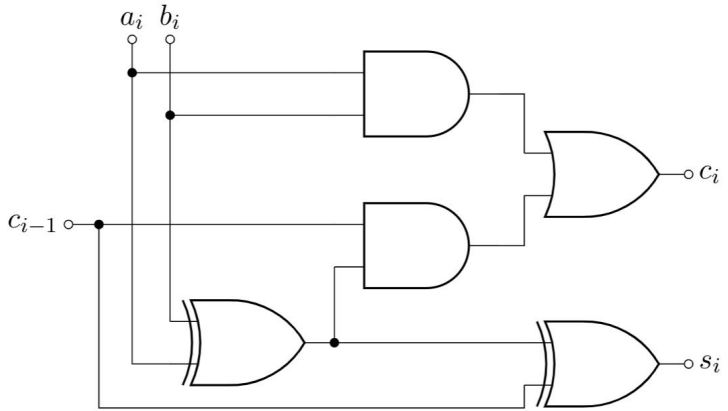
$$c_i = f_2(a_i, b_i, c_{i-1})$$

c_{i-1} \ a_ib_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$c_i = a_ib_i + c_{i-1}(a_i \oplus b_i)$$

Full Adder

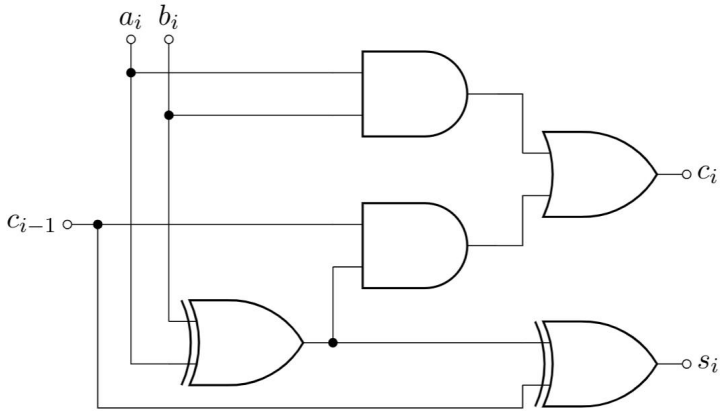
$$s_i = c_{i-1} \oplus a_i \oplus b_i \qquad c_i = a_i b_i + c_{i-1}(a_i \oplus b_i)$$

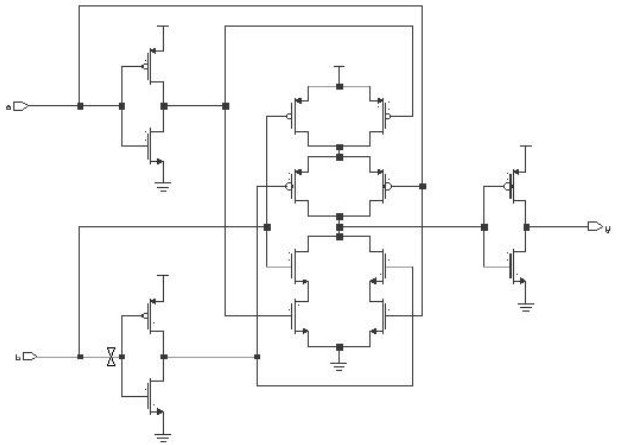


Full Adder

$$s_i = c_{i-1} \oplus a_i \oplus b_i$$

$$c_i = a_i b_i + c_{i-1}(a_i \oplus b_i)$$





- Problema: circuitalmente, la porta XOR è complessa e ha un tempo di propagazione elevato
- Il tempo di propagazione del carry in una rete iterativa *non è accettabile*

Carry Lookahead Adder

- Una possibile soluzione è quella di calcolare i bit di carry *in parallelo*
- Possiamo effettuare questa decomposizione:
  - $s_i = c_{i-1} \oplus a_i \oplus b_i$ e $c_i = a_i b_i + c_{i-1}(a_i \oplus b_i)$
  - Poniamo: $G_i = a_i b_i$ e $P_i = a_i \oplus b_i$
  - Possiamo riscrivere le equazioni come:
    - $s_i = P_i \oplus c_{i-1}$
    - $c_i = G_i + P_i c_{i-1}$

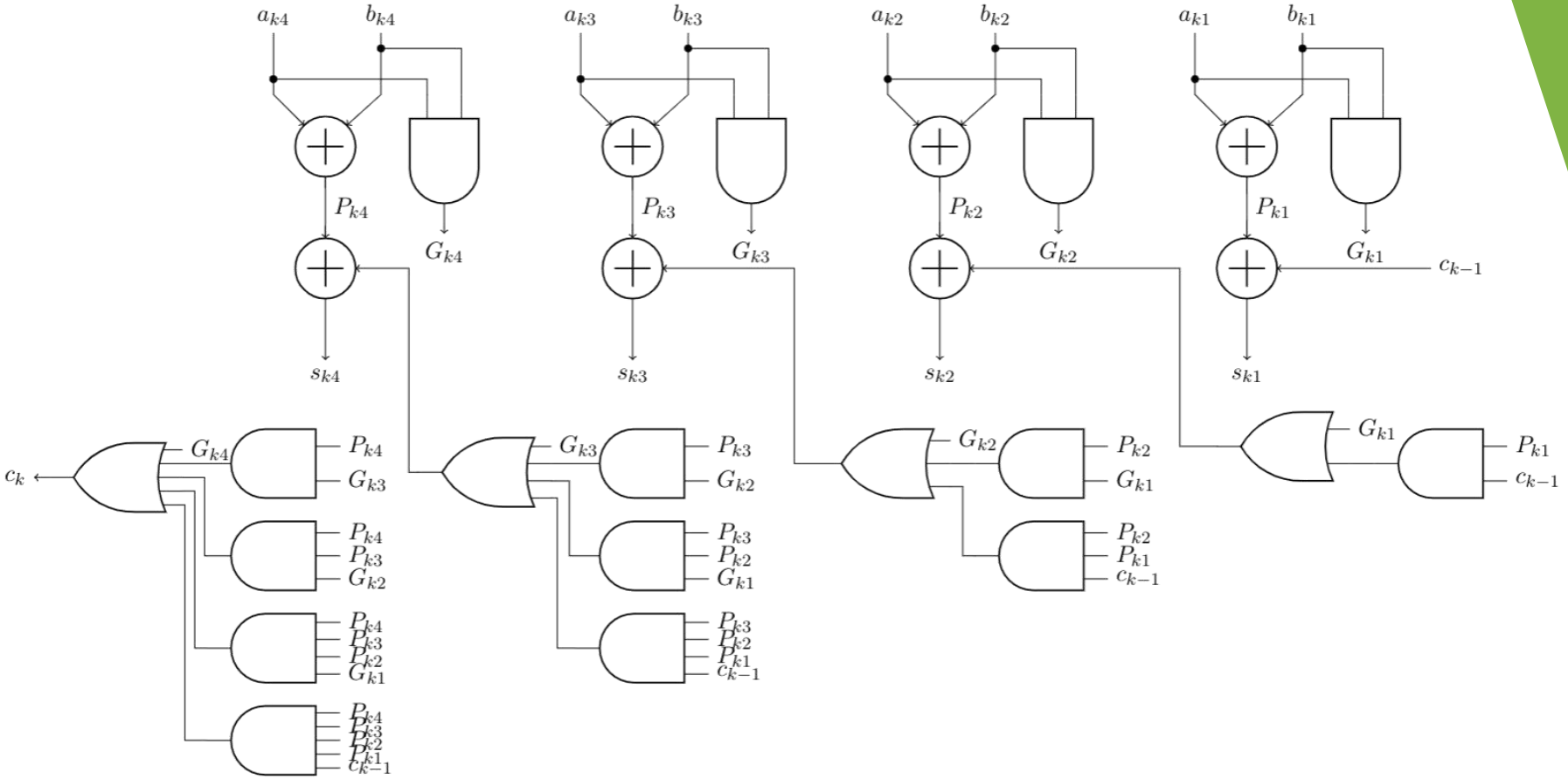
- Il termine G_i viene chiamato *generatore di carry*
- Il termine P_i è il *propagatore di carry*

Carry Lookahead Adder

- Supponiamo di dividere i numeri A e B in gruppi di 4 bit
- Ciascun gruppo k riceverà un bit di carry dal modulo precedente $k-1$ e ne invierà uno al modulo successivo $k+1$
- Se riusciamo a calcolare velocemente il bit di carry c_k da inviare al modulo successivo, abbattiamo il ritardo dovuto all'organizzazione iterativa

$$c_{k4} = G_{k4} + P_{k4}c_{k3} = G_{k4} + P_{k4}(G_{k3} + P_{k3}c_{k2}) =$$
$$= G_{k4} + P_{k4}(G_{k3} + P_{k3}(G_{k2} + P_{k2}c_{k1})) =$$
$$= G_{k4} + P_{k4}(G_{k3} + P_{k3}(G_{k2} + P_{k2}(G_{k1} + P_{k1}c_{k-1}))) =$$
$$= G_{k4} + P_{k4}G_{k3} + P_{k4}P_{k3}G_{k2} + P_{k4}P_{k3}P_{k2}G_{k1} + P_{k4}P_{k3}P_{k2}P_{k1}c_{k-1}$$

Carry Lookahead Adder



Shifter

- L'operazione di *shift* consiste nel muovere a destra o a sinistra i bit di una parola binaria
  - alcune cifre possono venire scartate
  - le posizioni lasciate libere possono essere riempite con degi zeri

- Esempio: `1101` diventa `1010` a causa di uno shift a sinistra di una posizione

- Considerando la possibilità di spostare un bit di una sola posizione, dobbiamo prevedere i seguenti "comandi" per un circuito:
  - SH=0 non effettuare alcun cambiamento, SH=1 effettua la traslazione
  - D=0 trasla a sinistra, D=1 trasla a destra

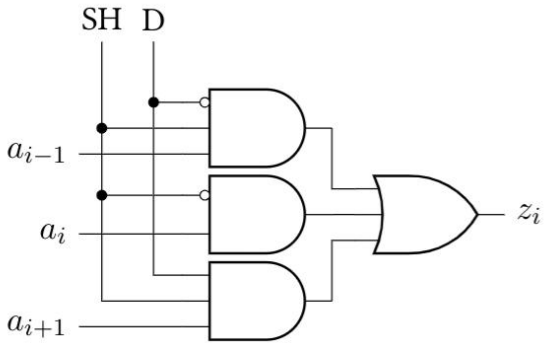
Shifter

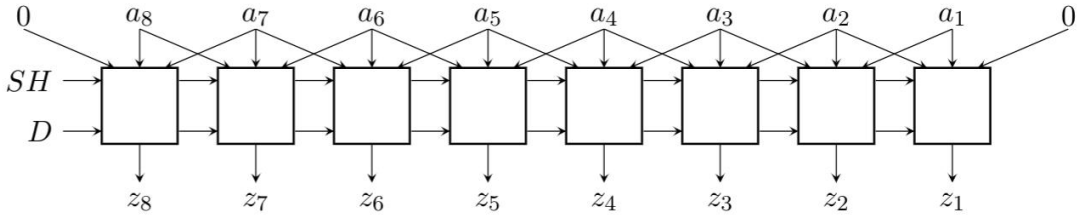
- Possiamo descrivere il circuito che calcola il valore del bit z_i in uscita con il seguente sistema:

$$z_i = \begin{cases} a_i & \text{se } SH = 0 \\ a_{i-1} \cdot \overline{D} + a_{i+1} \cdot D & \text{se } SH = 1 \end{cases}$$

- Pertanto l'equazione che calcola il valore del bit diventa:

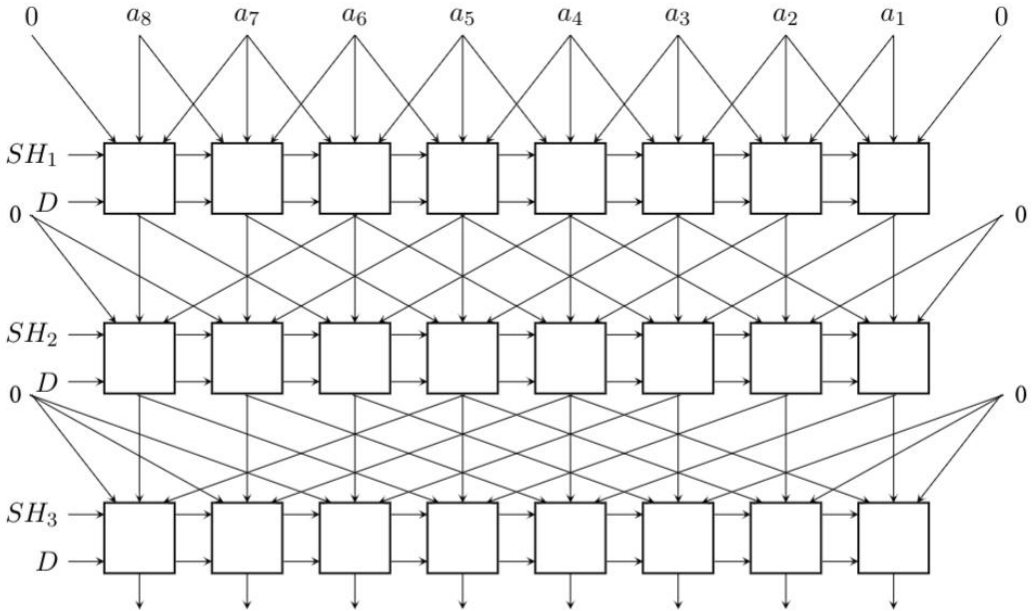
$$z_i = \overline{SH} \cdot a_i + SH \cdot (a_{i-1} \cdot \overline{D} + a_{i+1} \cdot D)$$





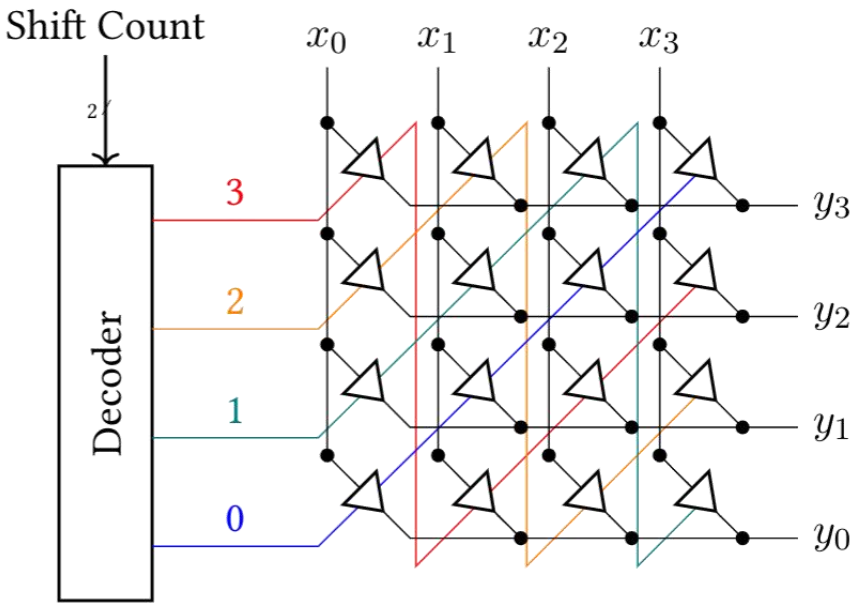
Shifter

- Per supportare una traslazione di più posizioni, possiamo costruire una rete iterativa di shifter
- Utilizzando un vettore di controllo **SH** possiamo controllare i vari livelli

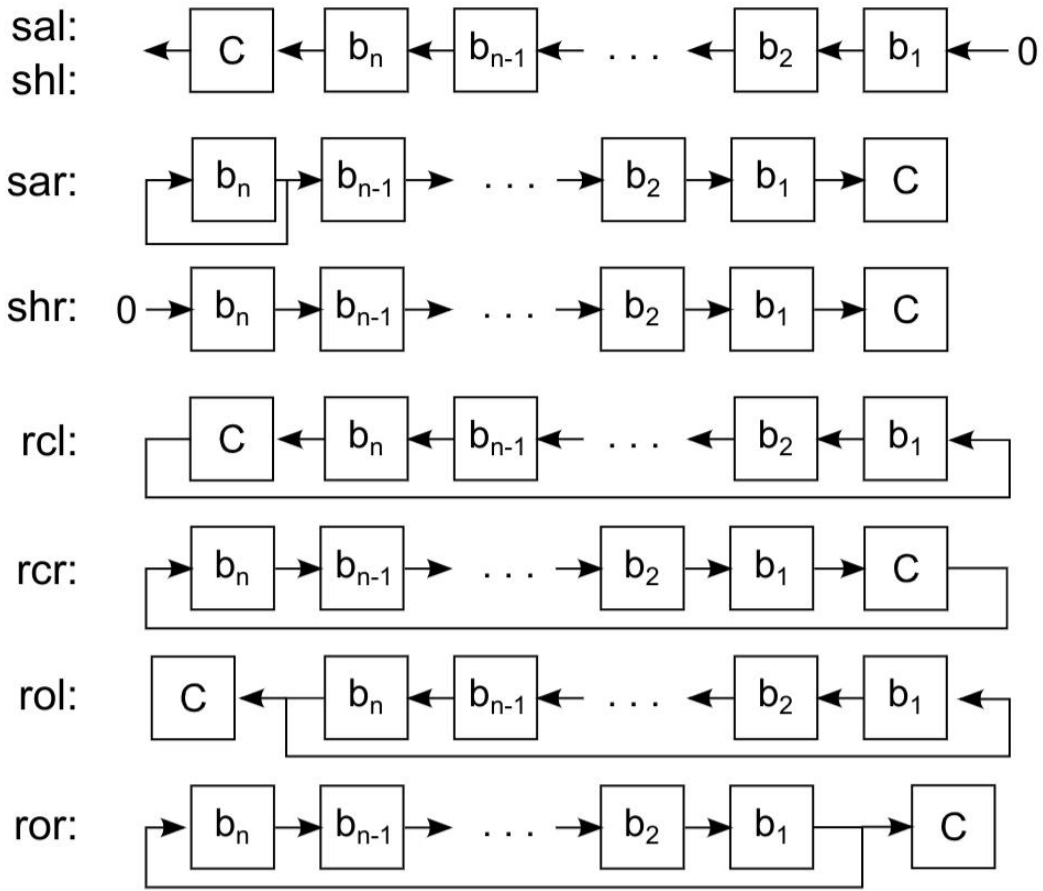


Barrel Shifter

- Il costo della rete può essere elevato
- Si può utilizzare una rete basata su *interruttori* posti in posizioni predefinite
- Attivando gli interruttori corretti, è possibile implementare in tempo costante tutte le traslazioni a destra e a sinistra



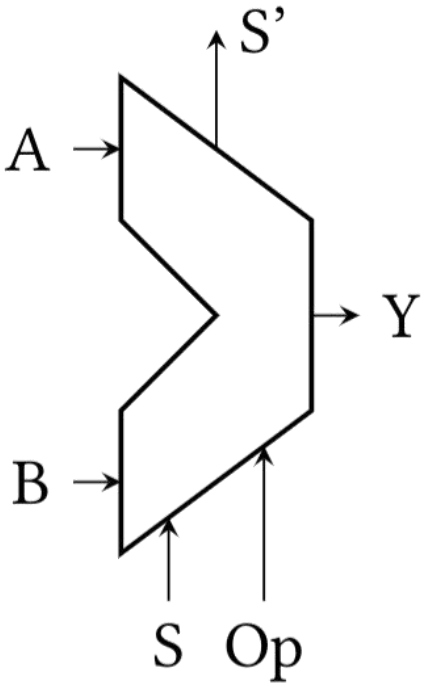
Operazioni di shift tipiche



- Uno shifter non implementa soltanto traslazioni a destra e a sinistra
  - le operazioni viste fino ad ora prendono il nome di *shift logico*
- Altre operazioni tipiche sono:
  - rotazioni
  - shift *aritmetico*
- In alcuni casi, viene preso in considerazione anche un *bit di carry*

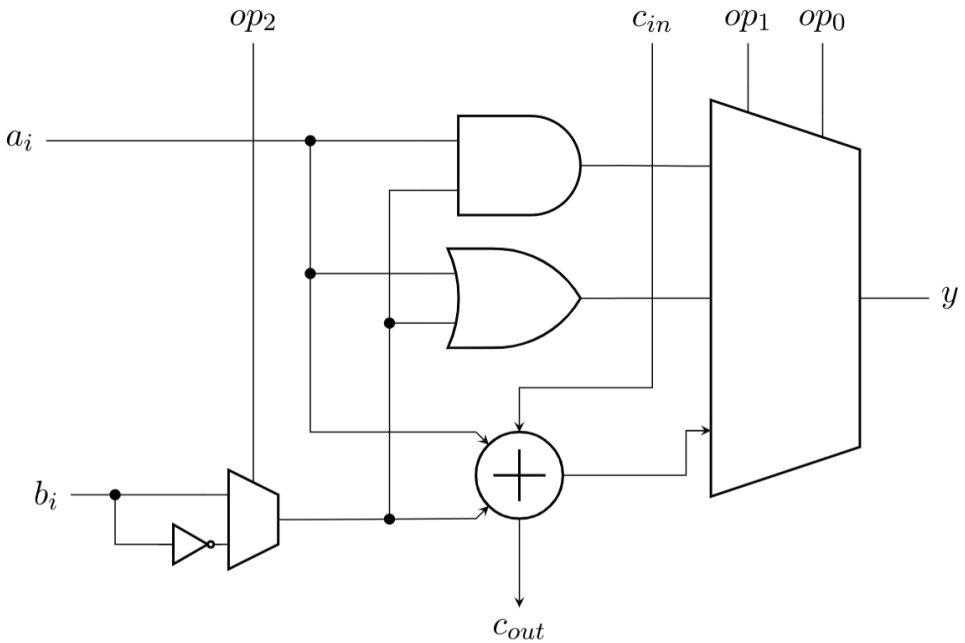
Unità Logico Aritmetica (ALU)

- Molte operazioni operano su parole binarie di dimensione fissa
- Per risparmiare spazio, è possibile *accorpare* le varie operazioni in un singolo circuito iterativo, chiamato ALU
  - La struttura delle reti iterative è la stessa, cambia soltanto la logica per implementare l'operazione
- Mediante alcuni segnali di controllo, si specifica qual è l'operazione che si vuole eseguire sulle parole in input



Modulo di una ALU

- Un modulo come quello in figura permette di implementare varie operazioni, a seconda dei bit di controllo forniti in input



op_0	op_1	op_2	c_{in}	y
0	0	0	—	$a \cdot b$
1	0	0	—	$a + b$
0	1	0	—	$a \oplus b \oplus c_{in}$
0	1	1	0	$a + \overline{b}$
0	1	1	1	$a - b$ (complemento a 2)