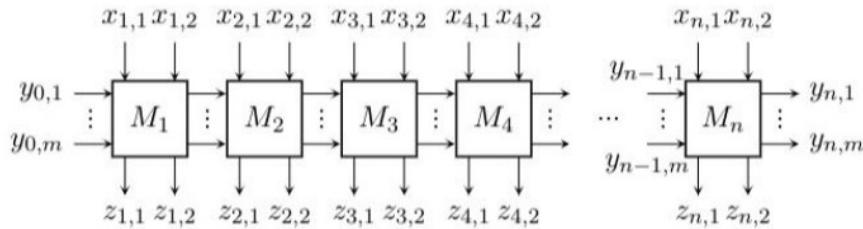


# Lezione 12 Reti iterative

giovedì 19 ottobre 2023 10:43

I processori moderni , implementano e gestiscono programmi fino a 512 bit . Quindi i circuiti combinatori visti finora non sono tanto "adatti" . Quindi si cerca di "spezzettare il problema" in vari sotto problemi, i quali verranno ricomposti alla fine. Quindi si prendono i bit in ingresso, quelli in uscita e li studio in sotto blocchi. **Attenzione all'interdipendenza tra i blocchi:**



Dove :

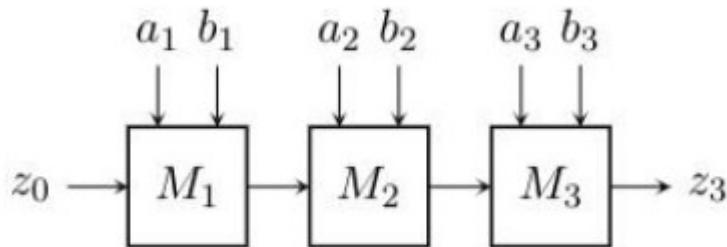
Vettore  $\mathbf{y}$ : rappresenta le informazioni di stato trasferite da un modulo al successivo

- L'ultimo modulo può esporre parte di questa informazione all'esterno, ad esempio per notificare dettagli circa il risultato finale dell'operazione

Vettore  $\mathbf{x}$ : rappresenta il dato in input, decomposto tra i vari moduli

Vettore  $\mathbf{z}$ : rappresenta l'output, calcolato iterativamente dai moduli

**Attenzione al tempo di propagazione : dipende dal numero dei blocchi .** Vediamo ora delle applicazioni : **il comparatore** : modulo hardware che confronta due parole binarie : quindi prende in input due input e ci dice se questi due numeri sono uguali o no .



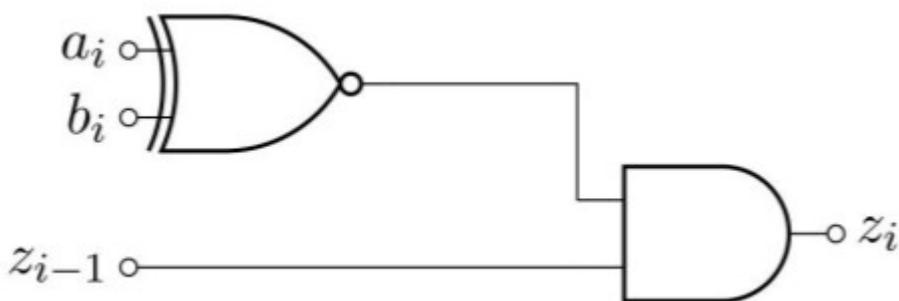
Da notare che  $z$  è un'informazione che viene propagata : nel nostro caso è il risultato del confronto. Quindi andiamo a costruire la tabella di verità :

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

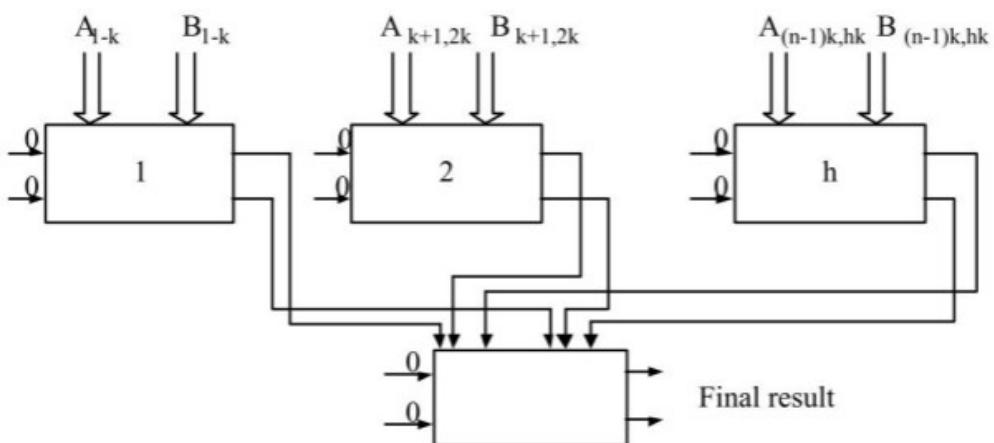
Se i bit precedenti sono uguali l'uscita vale 1 solo in due casi : devo propagare anche l'informazione che il modulo corrente sta osservando , solo se i due bit correnti sono 1 (carry in =1).

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

Attenzione al primo modulo : ingressi devono essere configurato/forzato ad 1 , in quanto non ci è stato nessun confronto . Dal punto di vista circuitale :



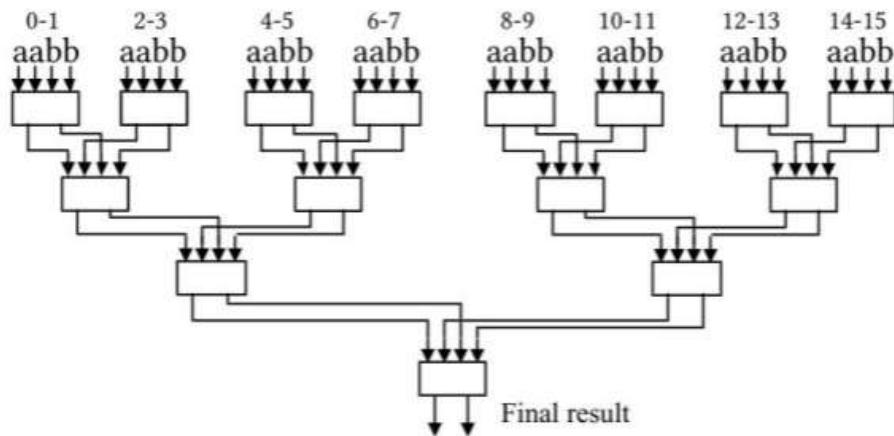
Quanto detto finora si ha solo uguaglianza , ma in generale si devono vedere anche se i numeri sono maggiori o minori . Di queste reti ( o meglio quelle con n arbitrario di moduli) il problema è il tempo di propagazione , che si stabilizzerà dopo . Posso velocizzare?? Si potrei parallelizzare : parte dei calcoli svolti in parallelo : **comparatore veloce** : confronto i bit a coppia ( h blocchi di k bit) ed ogni blocco è internamente un comparatore :



Il problema è il numero di input alla rete sottostante : è abbastanza elevato . Questa potrebbe essere una soluzione , ma visto che spesso si usano parole con lunghezza di potenze di 2 , potrei pensare ad una logica ad albero : **comparatore ad albero** : divido il mio circuito , calcolando in

parallello il risultato ed aggregando il risultato in parallelo :

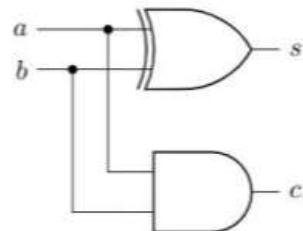
Ad esempio, per interi a 16 bit:



Oltre ai comparatori, ci servono anche i sommatori : circuiti elementari che permette di calcolare la somma ad un solo bit , inoltre genera un'informazione di riporto (carry-in) che non viene presa in considerazione dal modulo successivo : vediamo l'**half adder** :

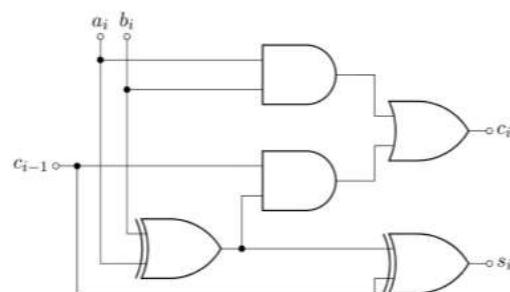
$$s = a \oplus b \quad c = a \cdot b$$

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



Questo è un circuito molto veloce , ma non lo osso usare per fare il sommatore ad n bit, in quanto genero l'informazione di stato, ma non viene preso in considerazione. Vediamo ora quello completo : il **full adder (considera il riporto precedente)** :

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



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

$a_i b_i$	00	01	11	10
$c_{i-1}$	*	*	*	*

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

$a_i b_i$	00	01	11	10
$c_{i-1}$	*	*	*	1

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

	00	01	11	10
0	0	1	0	1
1	1	0	1	0

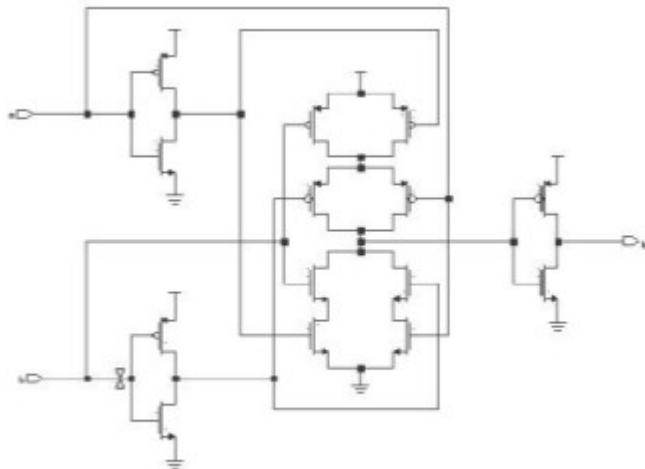
$$c = f_2(a_i, b_i, c_{i-1})$$

	00	01	11	10
0	0	0	1	0
1	0	1	1	1

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

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

Da notare che il riporto è più lento della somma perché aggiungo una porta and ( prodotto): anche se ho un termine in comune , calcolo due pezzi di due funzioni diverse : risparmio componenti elettronici . In questo circuito si usa un doppio xor binario rispetto ad uno ternario per via del costo elettronico in componenti :



Inoltre il tempo di propagazione di questa porta è spaventoso ! Possiamo migliorare questo circuito ? Andiamo a riorganizzare la mia rete iterativa per andare a calcolare un qualche risultato in parallelo ( il bit di carry) : cerco di sbirciare il carry per evitare perdite di tempo : **il carry lookahead adder** (carry in parallelo alla somma):

$$s_i = c_{i-1} \oplus a_i \oplus b_i \text{ e } c = a_i b_i + c_{i-1}(a_i \oplus b_i)$$

$$\text{Poniamo: } G_i = a_i b_i \text{ e } P_i = a_i \oplus b_i$$

Possiamo riscrivere le equazioni come:

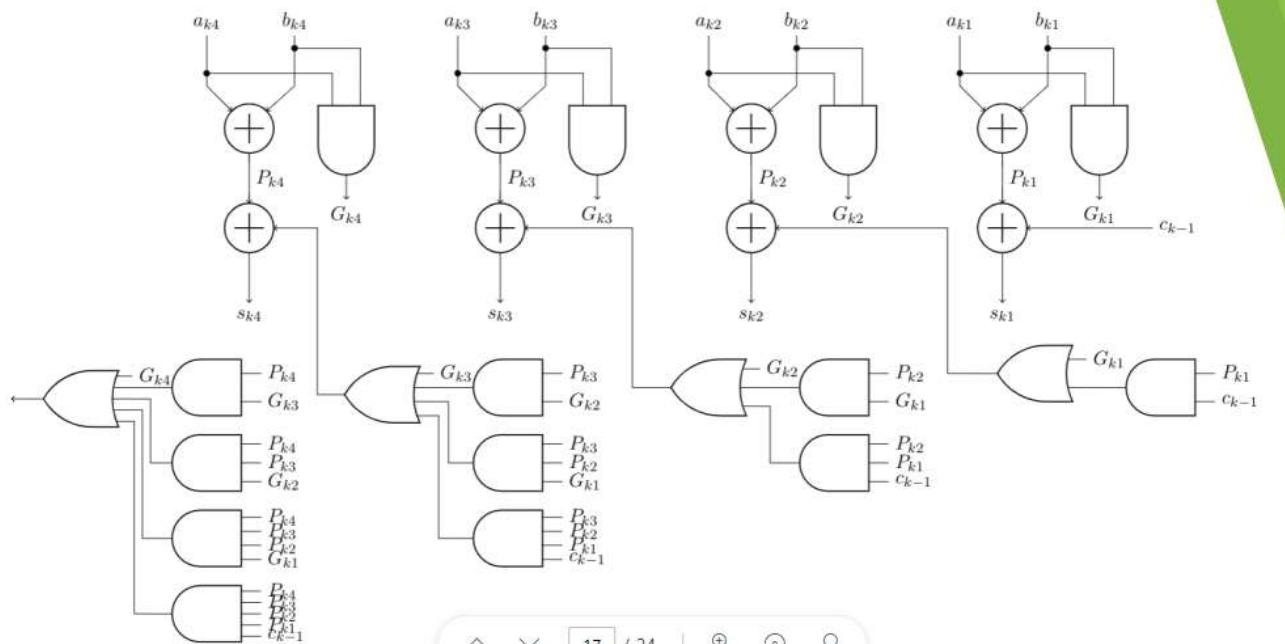
- $s_i = P_i \oplus c_{i-1}$
- $c = G_i + P_i c_{i-1}$

Dove Gi è il generatore di riporto , mentre Pi è il propagatore.

In questo circuito a posto di prendere coppie di bit , ne prendiamo 4 coppie di bit :

$$\begin{aligned}
 c_{k4} &= G_{k4} + P_{k4} c_{k3} = G_{k4} + P_{k4}(G_{k3} + P_{k3} c_{k2}) = \\
 &= G_{k4} + P_{k4}(G_{k3} + P_{k3}(G_{k2} + P_{k2} c_{k1})) = \\
 &= G_{k4} + P_{k4}(G_{k3} + P_{k3}(G_{k2} + P_{k2}(G_{k1} + P_{k1} c_{k-1}))) = \\
 &= G_{k4} + P_{k4} G_{k3} + P_{k4} P_{k3} G_{k2} + P_{k4} P_{k3} P_{k2} G_{k1} + P_{k4} P_{k3} P_{k2} P_{k1} c_{k-1}
 \end{aligned}$$

Il circuito è :



Vediamo ora lo **shifter**: muove a dx o sx i bit di una parola binaria . Decido inoltre la cifra che faccio entrare : 0 oppure 1 . Vediamo un esempio :

0100-> shift a sx di una posizione -> 1000

**Shiftare a sx si moltiplica per  $2^x$ ; a destra invece divido per  $2^x$ .**

1010-> shift a dx di una posizione > 0101

**Attenzione alla cifra che entra : se interi positivi entra uno 0 (shift logico) ,mentre se cp2 entra un 1 (shift aritmetico).**

1010-> shift a dx -> 1101

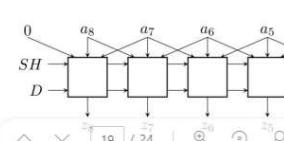
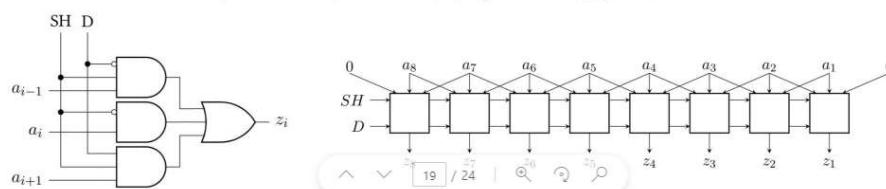
Il circuito è:

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

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

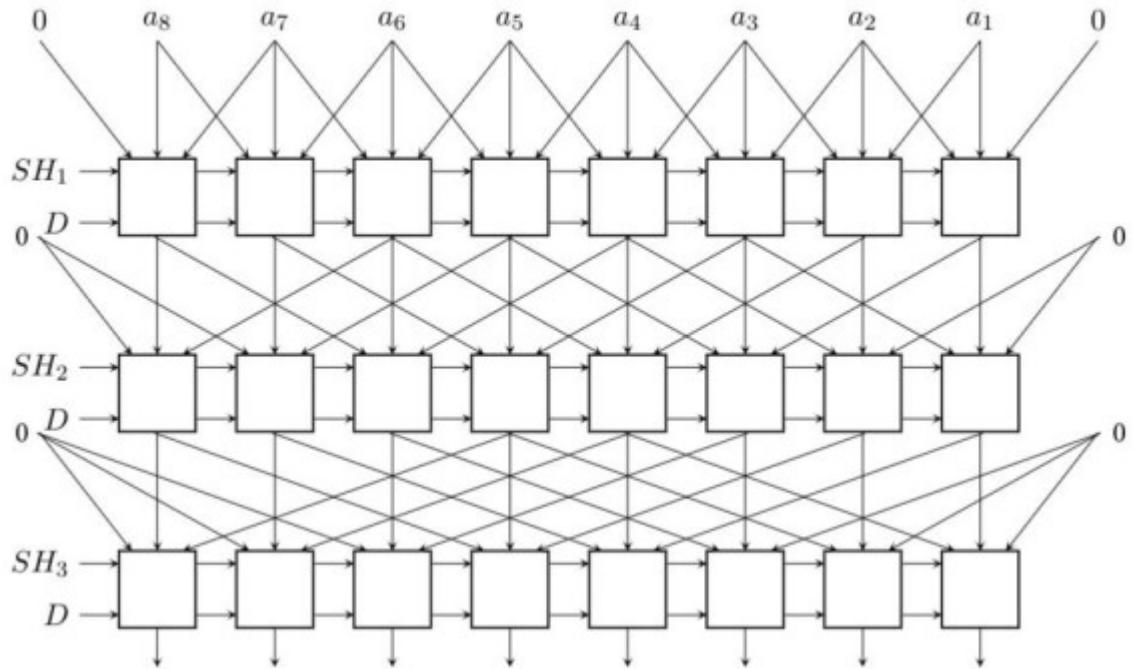
- Pertanto l'equazione che calcola il valore del bit diventa:

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

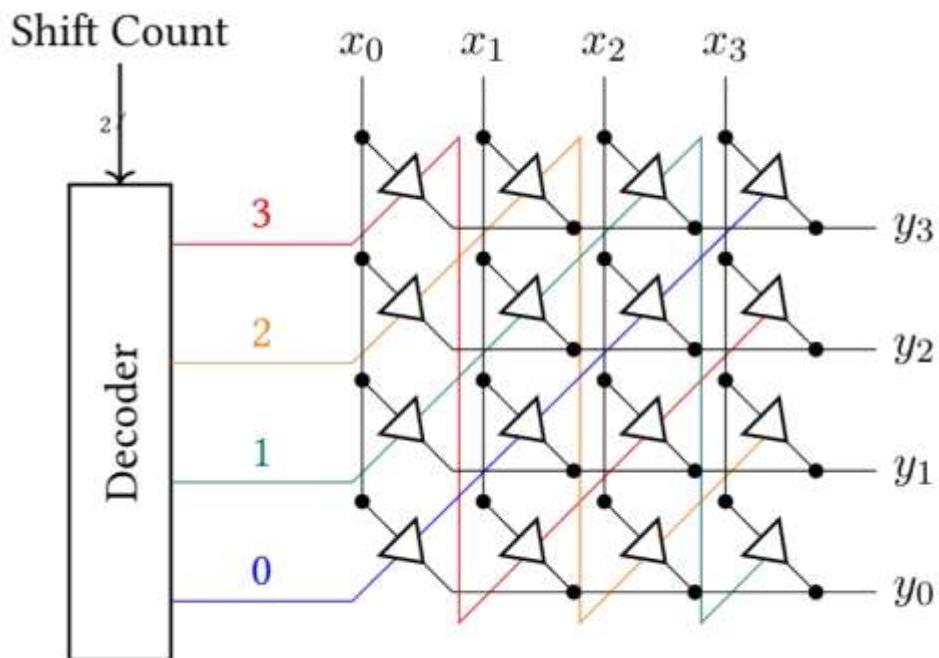


**Se shift=0 non fai shift , d=0 trasla a dx ;d=1 trasla a sx**

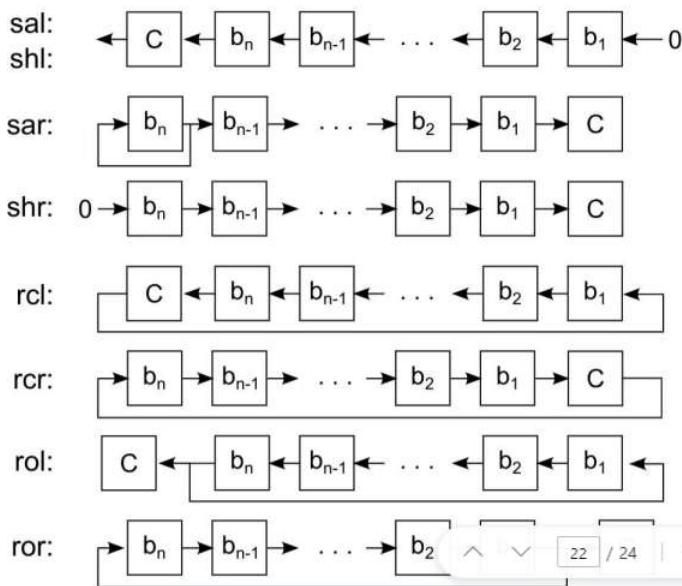
Quanto detto finora vale solo per shift di una posizione, ma per shiftare più posizioni? **Lo faccio multi livello :**



Da notare che questo circuito è molto complesso dal punto di vista funzionale e circuitale : si può migliorare?? Si , usando il **barrel shifter** : componenti e livelli ridotti . Non sfrutta la decomposizione in reti iterative, ma usa interruttori (buffer tiri state) :



Il decoder sceglie quale canale verrà usato. Tipicamente le operazioni che può fare questo dispositivo sono le seguenti :



- 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 carry*