

# Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

## Circuiti aritmetici

*Nino Cauli*



UNIVERSITÀ  
degli STUDI  
di CATANIA

Dipartimento di Matematica e Informatica

# Addizionatore ad 1 bit

Per **SOMMARE** numeri binari ad 1 bit:

$0 +$ $0 =$ <hr style="width: 100%; border: 0; border-top: 1px solid black; margin: 5px 0;"/> <span style="font-size: 2em;">0</span>	$1 +$ $0 =$ <hr style="width: 100%; border: 0; border-top: 1px solid black; margin: 5px 0;"/> <span style="font-size: 2em;">1</span>	$0 +$ $1 =$ <hr style="width: 100%; border: 0; border-top: 1px solid black; margin: 5px 0;"/> <span style="font-size: 2em;">1</span>	$1 +$ $1 =$ <hr style="width: 100%; border: 0; border-top: 1px solid black; margin: 5px 0;"/> <span style="font-size: 2em;">1 0</span>
<span style="font-size: 1.5em;">Addendi da un bit</span>		<span style="font-size: 1.5em;">Riporto in uscita</span> <span style="font-size: 1.5em;">Somma (1 bit)</span>	

**Figura 1.4** - Addizione di numeri a un bit

Il **RIPORTO IN USCITA** della cifre precedente viene assegnato come **RIPORTO IN ENTRATA** alla successiva

# Addizionatore ad 1 bit

- Un addizionatore tra due singoli bit può essere espresso da 2 funzioni logiche a tre ingressi (i due bit da sommare più il riporto in ingresso):
  - La prima calcola la somma tra i bit ed il riporto in ingresso
  - La seconda calcola il riporto in uscita
- Dalla tabella di verità si ricavano le espressioni logiche per somma e riporto in uscita

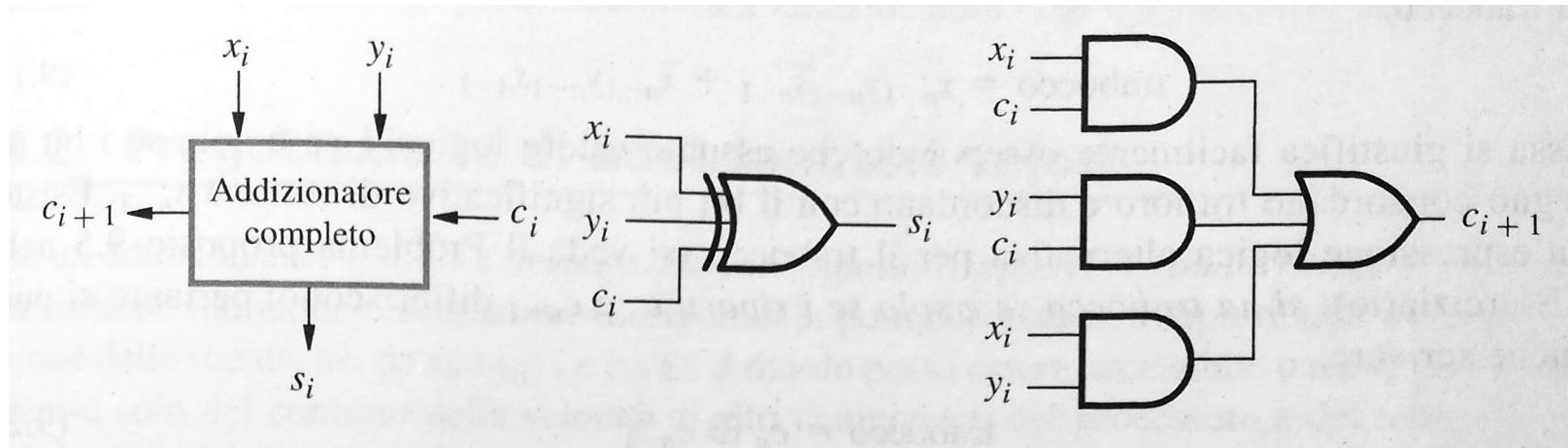
$x_i$	$y_i$	Riporto in ingresso $c_i$	Somma $s_i$	Riporto in uscita $c_{i+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i = \bar{x}_i \bar{y}_i r_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = x_i c_i + y_i c_i + x_i y_i$$

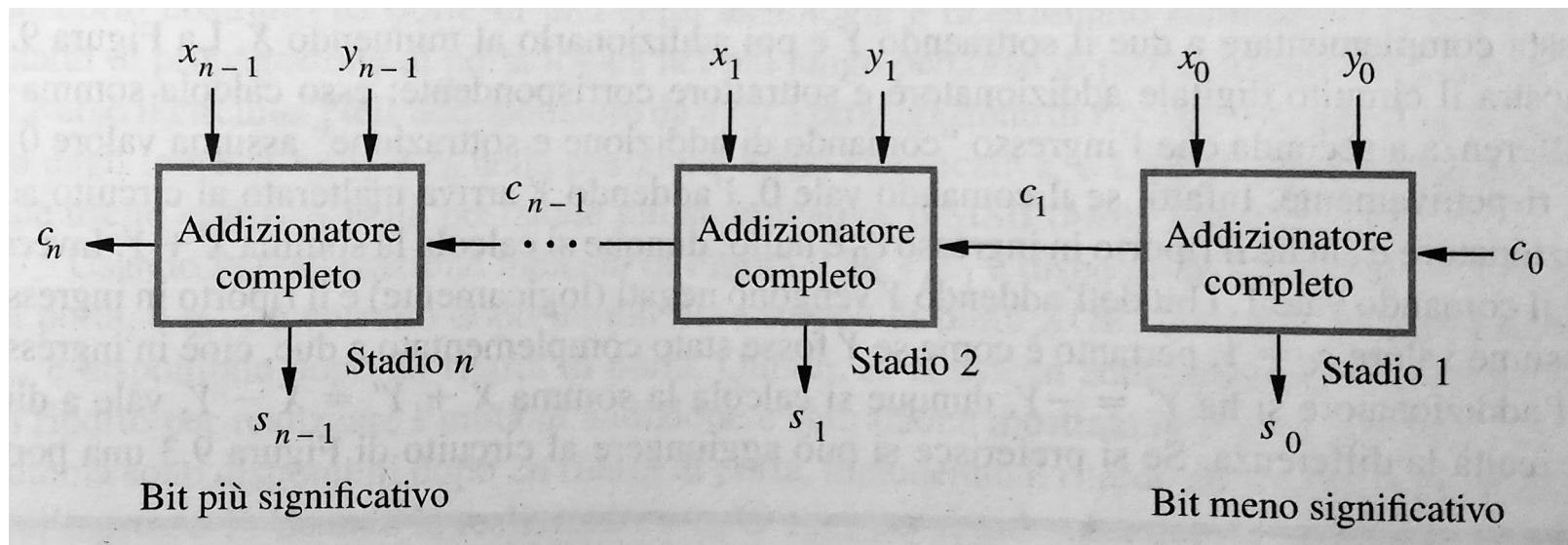
# Addizionatore completo (full adder)

- Unendo assieme in un singolo circuito le reti logiche per le funzioni di somma e riporto in uscita si ottiene l'addizionatore completo
- L'addizionatore completo prende in ingresso i due bit da sommare e il riporto in entrata e rende in uscita somma e riporto in uscita



# Addizionatore a propagazione di riporto

- Collegando una catena di  $n$  addizionatori completi in modo da propagare il riporto si ottiene un circuito in grado di sommare numeri binari di  $n$  bit
- Tale circuito è chiamato addizionatore a propagazione di riporto (ripple carry adder)



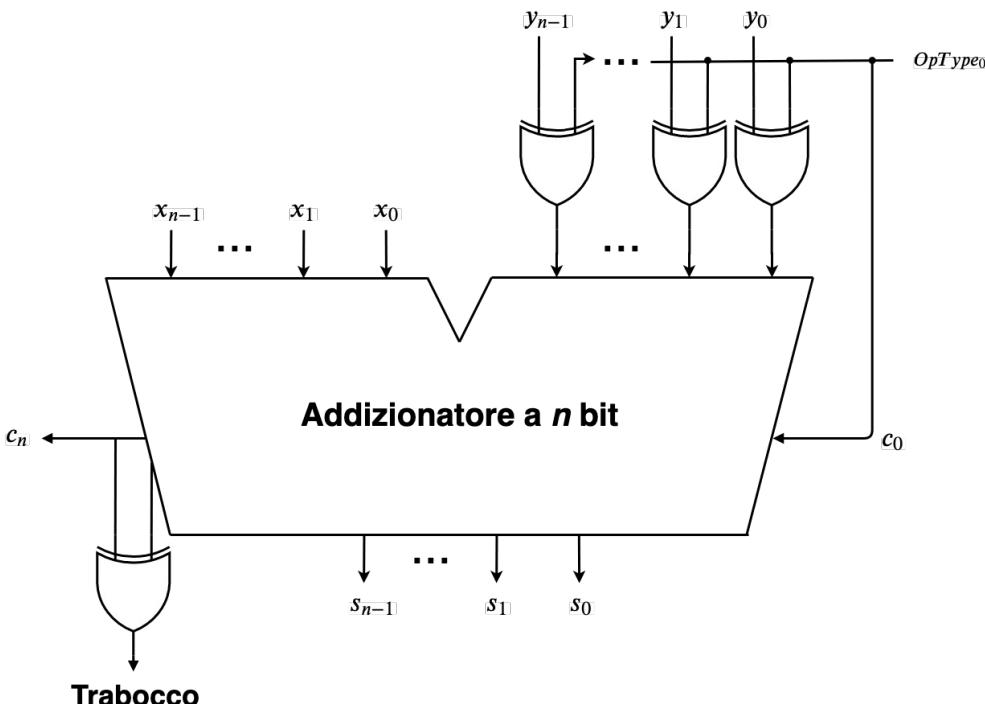
- L'addizione di due numeri in complemento a due corrisponde alla somma di due numeri binari naturali senza contare il riporto in uscita
- La sottrazione corrisponde ad un addizione complementando a due il sottraendo
- Bisogna però garantire che non avvenga trabocco
- Il calcolo del trabocco può essere espresso da una delle seguenti espressioni logiche:

$$\text{trabocco} = x_{n-1}y_{n-1}\bar{s}_{n-1} + \bar{x}_{n-1}\bar{y}_{n-1}s_{n-1}$$

$$\text{trabocco} = c_n \oplus c_{n-1}$$

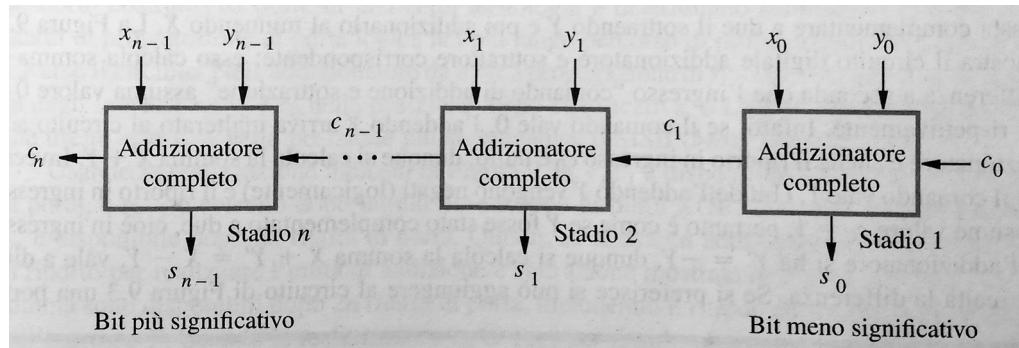
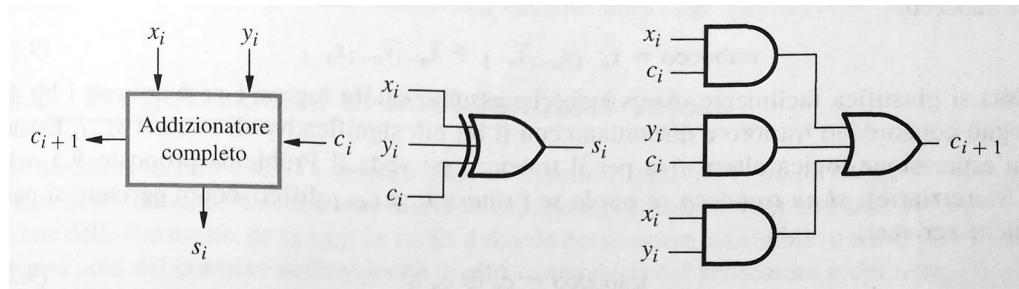
# Addizionatore algebrico a n bit

- Una unità logica per addizione e sottrazione può essere ottenuta usando un addizionatore a propagazione di riporto
- Si usa il bit  $OpType_0$  per complementare a due il sottraendo in caso di sottrazione
- Nel caso  $OpType_0 = 1$  si avrà un riporto in ingresso al bit meno significativo e il secondo addendo verrà complementato attraverso una catena di porte xor parallele ( $y_n \oplus OpType_0$ )
- Il trabocco viene calcolato come  $c_n \oplus c_{n-1}$



# Ritardi ripple carry adder

- Il ritardo totale di un circuito dipende dal ritardo del percorso più lento
- Assumiamo che ogni porta logica semplice introduce un ritardo fisso
- Un **Full Adder** genera  $s_i$  dopo **1 ritardo di porta**, mentre  $c_{i+1}$  dopo **2 ritardi di porta**
- Quindi in un **Ripple Carry Adder** a **n bit** il riporto  $c_n$  viene generato in **2n ritardi di porta**, mentre l'ultimo bit risultato  $s_{n-1}$  viene generato dopo **2n - 1 ritardi di porta**



# Funzioni di generazione e propagazione

- Il risultato di ciascun Full Adder dipende dal riporto calcolato dal Full Adder nella posizione anteriore:

$$s_i = x_i \oplus y_i \oplus c_i \quad c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

- Fattorizzando l'equazione del riporto si ottiene:

$$\begin{aligned} c_{i+1} &= x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i) c_i = G_i + P_i c_i \\ G_i &= x_i y_i \quad P_i = x_i + y_i \end{aligned}$$

- $G_i$  (funzione di generazione) e  $P_i$  (funzione di propagazione)** dipendono solo dagli ingressi  $x_i$  e  $y_i$  e possono essere calcolati tutti in parallelo in **1 ritardo di porta**

# Parallelizzare il calcolo del ritardo



È possibile calcolare i tutti ritardi  $c_i$  solo in funzione degli addendi X, Y e del riporto in ingresso  $c_0$ ?

- Espandendo iterativamente  $c_i$  fino ad arrivare a  $c_0$  si ottiene un'equazione per  $c_{i+1}$  in funzione solo dei vari P, G e di  $c_0$ :

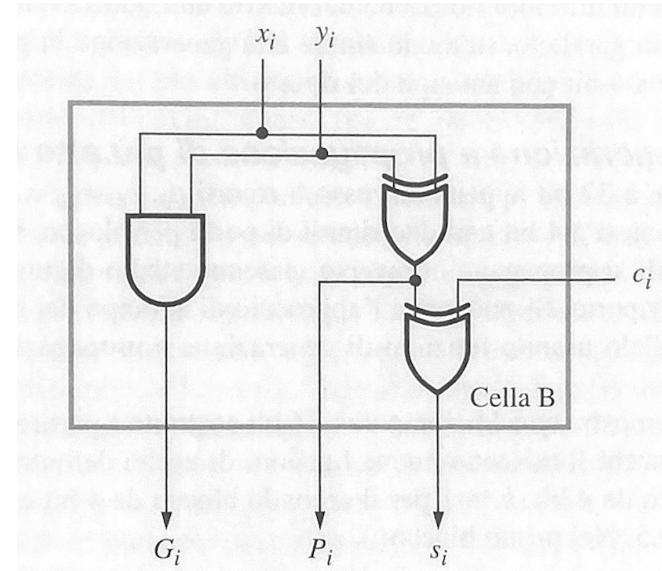
$$c_{i+1} = G_i + P_i c_i = G_i + P_i(G_{i-1} + P_{i-1} c_{i-1}) = G_i + P_i G_{i-1} + P_i P_{i-1} c_{i-1}$$

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \dots + P_i P_{i-1} \dots P_1 G_0 + P_i P_{i-1} \dots P_0 c_0$$

- In questo modo ciascun ritardo ci può essere calcolato in parallelo dopo 2 ritardi di porta

# Cella di stadio da un bit

- Si può modificare la cella di sommatore a 1 bit per dare in uscita anche le funzioni  $G_i = x_i y_i$  e  $P_i = x_i + y_i$
- $G_i$  si ottiene con una porta AND con ingressi  $x_i$  e  $y_i$
- $P_i$  si ottiene con una porta XOR con ingressi  $x_i$  e  $y_i$
- $s_i$  si ottiene con due porte XOR annidate con ingressi  $x_i$ ,  $y_i$  e  $c_i$



# Addizionatore con anticipo di riporto a 4 bit

- Usando 4 celle da un bit è possibile realizzare un **Addizionatore con anticipo di riporto** a 4 bit
- Il blocco “**Logica di anticipo del riporto**” genera i riporti come segue:

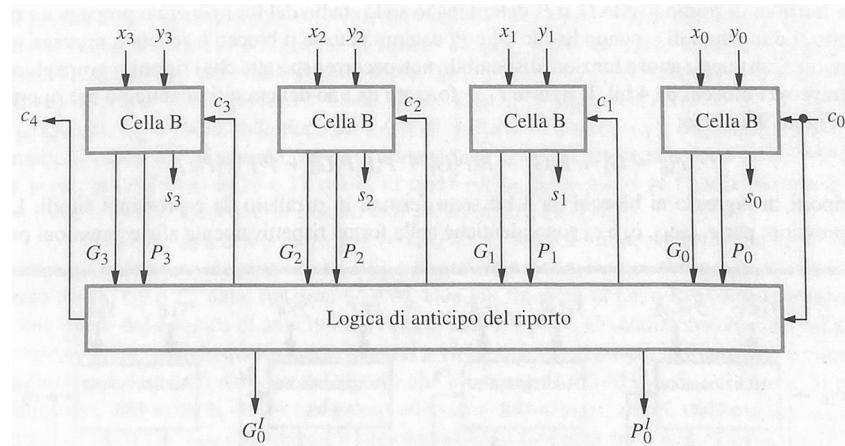
$$c_1 = G_0 + P_0 c_0$$

$$c_2 = G_1 + P_1 G_0 + P_1 P_0 c_0$$

$$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

$$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

- c4** viene generato dopo **3 ritardi di porta**, mentre il **risultato S** dopo **4 ritardi di porta**



# Addizionatore con anticipo a 2 livelli

- Con addizionatori con anticipo di riporto a più di 4 celle si incorre in valori di **fan-in troppo elevati** nelle porte di generazione dei riporti
  - Una soluzione è replicare l'idea di anticipo di riporto su **più livelli**
  - In un addizionatore a 4 bit si ha:
- $$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 = \mathbf{G}_k^I + \mathbf{P}_k^I c_0$$
- $$\mathbf{G}_k^I = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$
- $$\mathbf{P}_k^I = P_3 P_2 P_1 P_0$$
- Per ottenere un **addizionatore da 16 bit con anticipo di riporto**, si possono collegare **4 addizionatori a 4 bit** con un blocco di anticipo di riporto che riceve in ingresso i vari  $G_k^I$  e  $P_k^I$  e genera in parallelo  $c_4$ ,  $c_8$ ,  $c_{12}$  e  $c_{16}$

# Addizionatore con anticipo di riporto a 16 bit

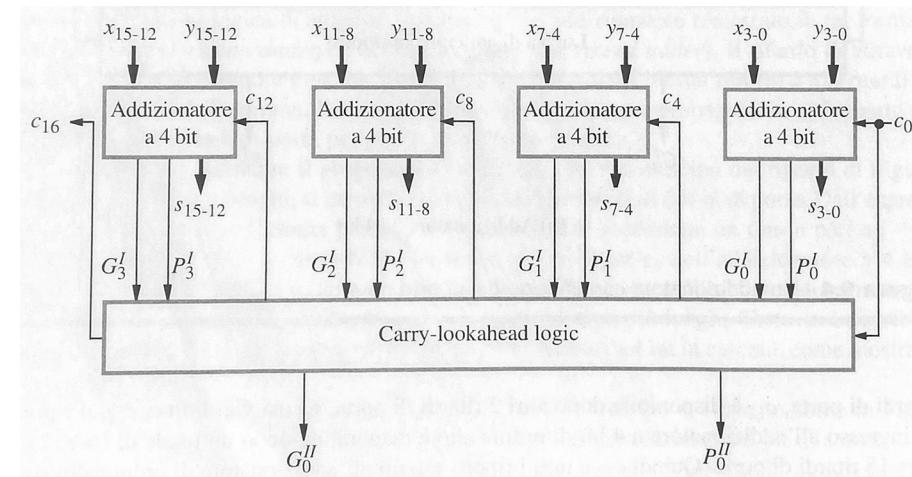
- Usando 4 addizionatori a 4 bit è possibile realizzare un **Addizionatore con anticipo di riporto** a 16 bit su due livelli
- Il blocco “**Logica di anticipo del riporto**” genera i riporti come segue:

$$c_4 = G_0^I + P_0^I c_0$$

$$c_8 = G_1^I + P_1^I G_0^I + P_1^I P_0^I c_0$$

$$c_{12} = G_2^I + P_2^I G_1^I + P_2^I P_1^I G_0^I + P_2^I P_1^I P_0^I c_0$$

$$c_{16} = G_3^I + P_3^I G_2^I + P_3^I P_2^I G_1^I + P_3^I P_2^I P_1^I G_0^I + P_3^I P_2^I P_1^I P_0^I c_0$$



- c16** viene generato dopo **5 ritardi di porta**, mentre il **risultato S** dopo **8 ritardi di porta**

# Moltiplicazione numeri senza segno

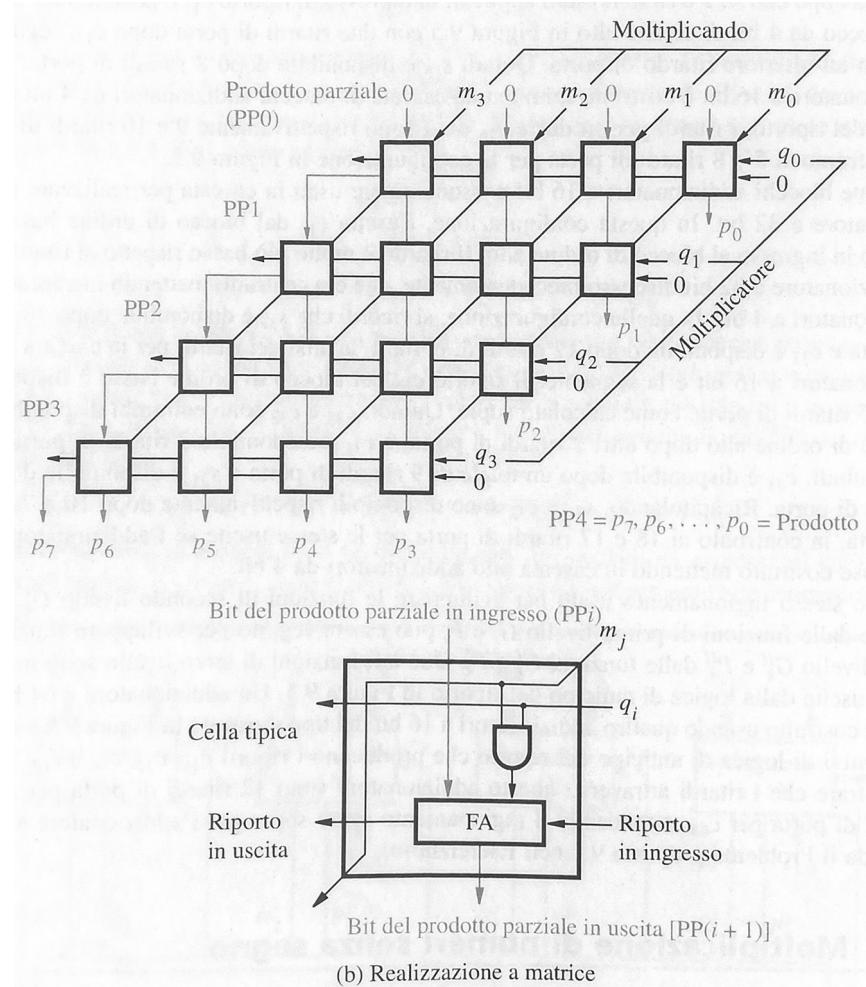
- Il metodo imparato a scuola per eseguire la moltiplicazione di due numeri vale anche per i numeri in formato binario
- Si moltiplica ciascuna cifra del moltiplicatore per il moltiplicando e si sommano i risultati fatti scorrere di una posizione ogni cifra
- Nel caso binario il Moltiplicando è moltiplicato solo per 0 o per 1
- Il prodotto di due numeri da n cifre è composto da  $2n$  cifre

$$\begin{array}{r} 1 & 1 & 0 & 1 \\ \times & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

(13) Moltiplicando M  
(11) Moltiplicatore Q  
(143) Prodotto P

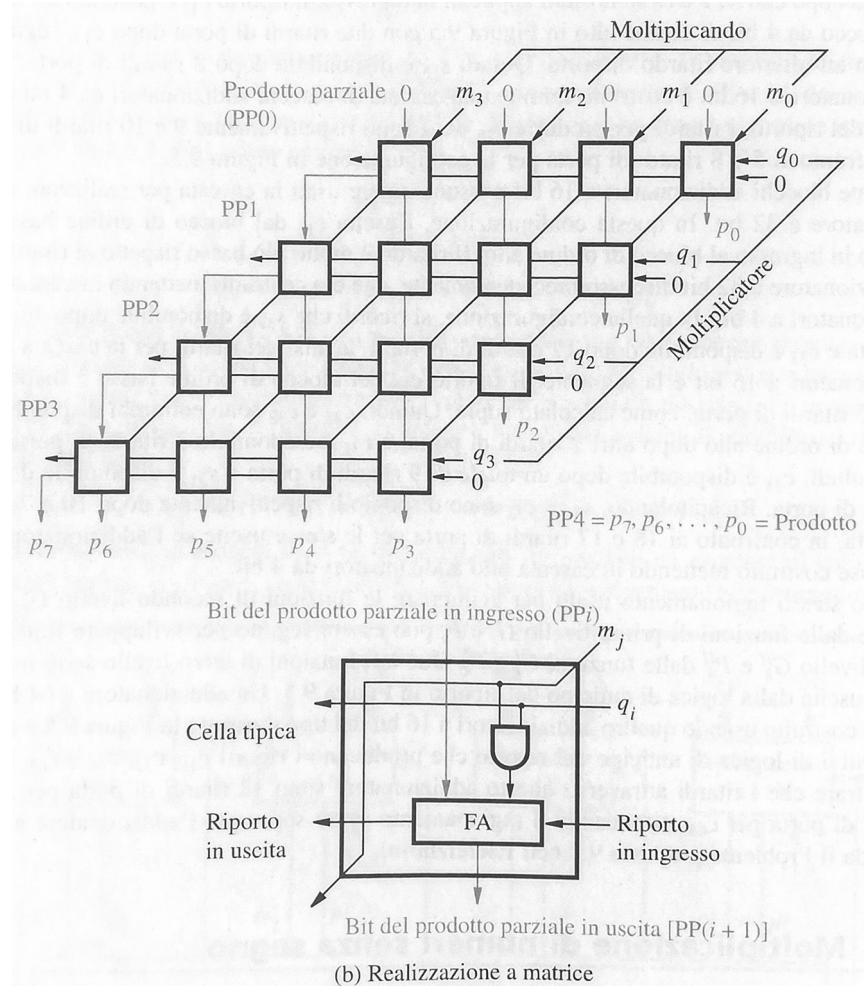
# Moltiplicazione a matrice

- Prendendo spunto dal metodo precedente si può realizzare un **circuito a matrice** in grado di eseguire la moltiplicazione
- Ogni riga** della matrice rappresenta un **prodotto parziale (PP)** tra le cifre del Moltiplicatore ed il Moltiplicando
- Ogni cella** della matrice è composta da un **Full Adder (FA)** ed una **porta AND**
- Gli ingressi del FA sono le cifre del PP della riga superiore e le cifre del Moltiplicando in AND con le cifre del Moltiplicatore



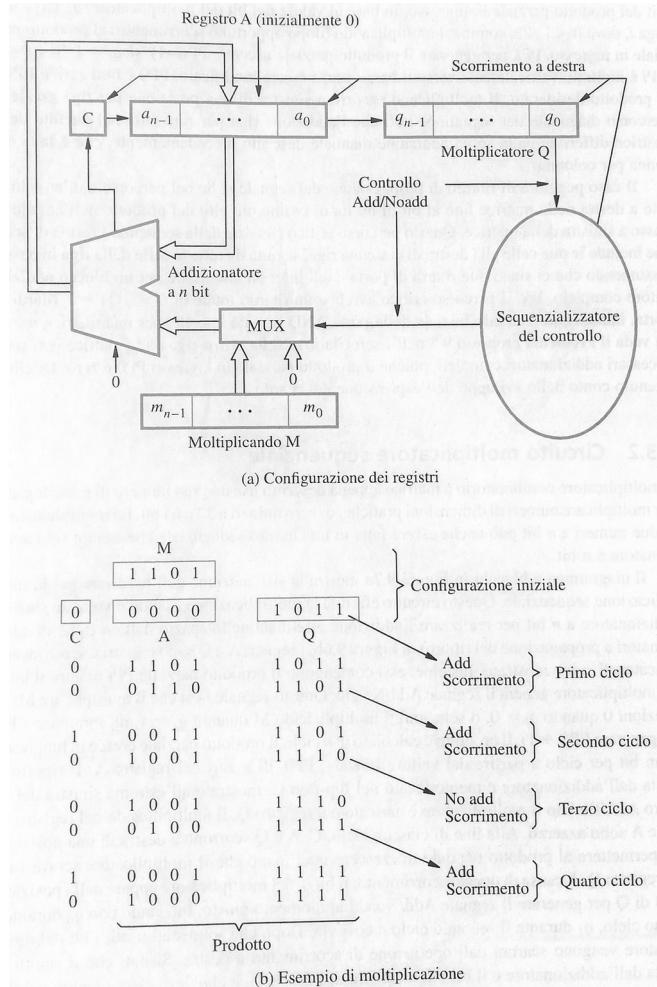
# Moltiplicazione a matrice

- Il circuito a matrice somma i suoi elementi per colonne
- Ciascuna colonna inoltra i riporti in uscita alla colonna successiva
- Le uscite delle celle sui lati destro e inferiore formano il **prodotto finale (PP4)**
- Circuito formato da un gran numero di porte  $n^2$  Full Adder e  $n^2$  porte AND



# Circuito moltiplicatore sequenziale

- La moltiplicazione di numeri senza segno può essere realizzata sequenzialmente usando solo **un Addizionatore a n bit e due registri a scorrimento**
- Ad ogni ciclo, l'Addizionatore effettua la somma tra il Moltiplicando (o un array di zeri) e un prodotto parziale fatto scorrere a destra di una posizione
- I due registri a scorrimento contengono:
  - **Inizialmente:** Registro A = 0 e Registro Q = Moltiplicatore
  - **Alla fine:** Registro A || Registro Q = Prodotto



# Circuito moltiplicatore sequenziale

- Algoritmo:

**A = 0**

**Q = Moltiplicatore**

**M = Moltiplicando**

**Per n cicli:**

**se  $q_0 = 1$ :**

**A = A + M**

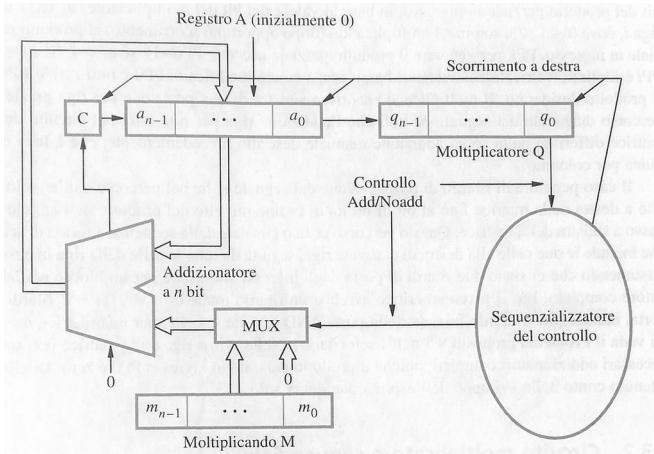
**altrimenti:**

**A = A + 0**

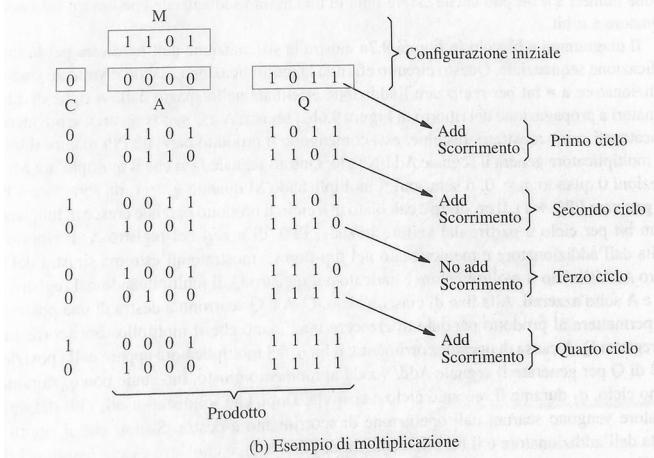
**c = riporto**

**c || A || Q scorrono verso destra di una posizione**

- Dopo n cicli i due registri **A || Q** concatenati conterranno il **Prodotto finale**

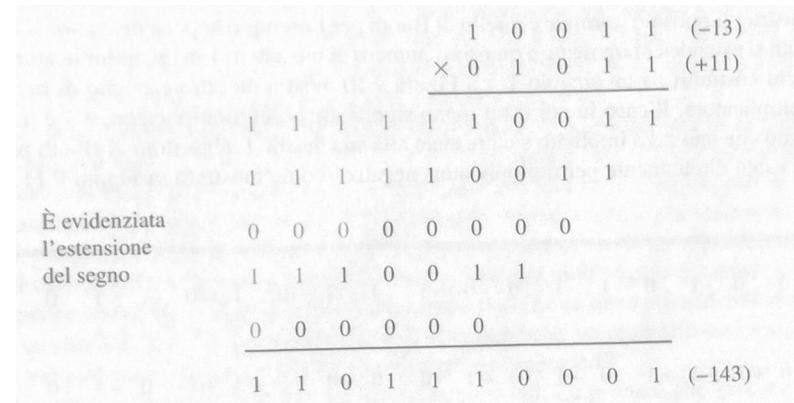


(a) Configurazione dei registri



# Moltiplicazione di numeri con segno

- Per moltiplicare numeri con segno in complemento a 2 bisogna apportare delle modifiche all'algoritmo di moltiplicazione
- Soluzione diretta:
  - **Moltiplicatore positivo:**
    - **Moltiplicando positivo:** si procede normalmente
    - **Moltiplicando negativo:** Bisogna estendere il segno quando si riporta il moltiplicando in tabella (in figura)
  - **Moltiplicatore negativo:** si fa il complemento a 2 di Moltiplicando e Moltiplicatore per ritornare al caso di Moltiplicatore positivo
- Soluzione più generale ed efficiente:
  - **Algoritmo di Booth**



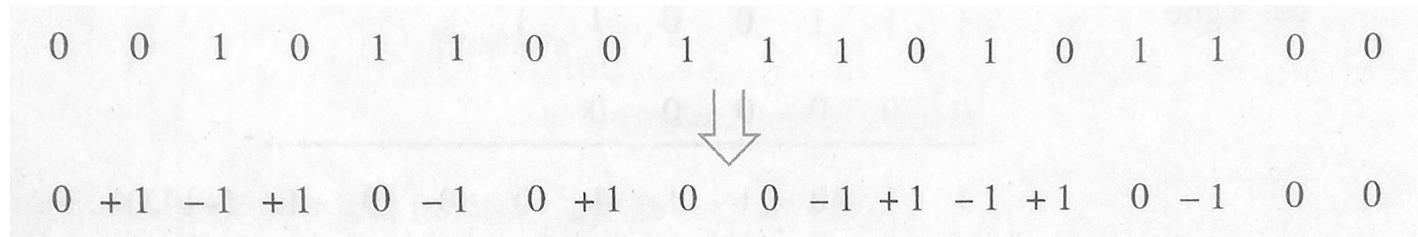
The diagram illustrates a binary multiplication process. At the top, the multiplicand  $(-13)$  is shown as  $100011$ , and the multiplier  $(+11)$  is shown as  $01011$ . A horizontal line separates the numbers from the multiplication steps below. The first step shows the multiplicand  $100011$  multiplied by the multiplier's least significant bit (0), resulting in  $000000$ . The second step shows the multiplicand shifted left (1000110) multiplied by the next bit (1), resulting in  $1000110$ . This pattern continues for all bits of the multiplier. An annotation "È evidenziata l'estensione del segno" points to the sign extension of the multiplier's bits (0, 1, 0, 1, 1) in the second row. The final result is  $11000001$ , which is the binary representation of  $-143$ .

# Algoritmo di Booth

- Nell'algoritmo di Booth si ricodifica il Moltiplicatore come somma e sottrazione di potenze di 2
- Caso semplice in cui il Moltiplicatore contiene una sequenza contigua di 1:
  - $Q = 0011110 = 0100000 + 1111110 = 0100000 - 0000010 = 2^5 - 2^1$
  - $P = M * Q = M * 2^5 + M * -2^1 = M * 2^5 + -M * 2^1$
  - Il prodotto è uguale al moltiplicando fatto scorrere di 5 posizioni a sinistra + il complemento del moltiplicando fatto scorrere di 1 posizione a sinistra
- Generalizzabile per qualsiasi Moltiplicatore:
  - $Q = 1100111011 = 1100000000 + 0000111000 + 0000000011 = 0000000000 + 1100000000 + 0001000000 + 1111111000 + 0000000100 + 1111111111 = -2^8 + 2^6 - 2^3 + 2^2 - 2^0$
  - $P = -M * 2^8 + M * 2^6 + -M * 2^3 + M * 2^2 + -M * 2^0$

# Algoritmo di Booth

- Possiamo quindi rappresentare il moltiplicatore ricodificato come sequenza di 0, 1 e -1:



- Durante la moltiplicazione, se il bit del Moltiplicatore è:
  - 0**: si somma al prodotto parziale una sequenza di 0
  - 1**: si somma al prodotto parziale il Moltiplicando
  - 1**: si somma al prodotto parziale il complemento del moltiplicando

Moltiplicatore		Versione del moltiplicando selezionata dal bit $i$
Bit $i$	Bit $i - 1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

# Algoritmo di Booth esempi



## Confronto tra algoritmo semplice e Booth

## Booth con Moltiplicatore negativo

$$\begin{array}{r}
 \begin{array}{r}
 0 & 1 & 1 & 0 & 1 & (+13) \\
 \times 1 & 1 & 0 & 1 & 0 & (-6) \\
 \hline
 \end{array}
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{r}
 \begin{array}{r}
 0 & 1 & 1 & 0 & 1 \\
 0 & -1 & +1 & -1 & 0 \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{r}
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{r}
 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0
 \end{array}$$