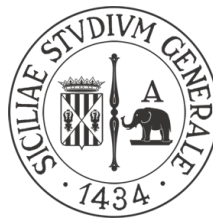


Corso di Architettura degli Elaboratori e Laboratorio (F-N)

Circuiti aritmetici

Massimo Orazio Spata

Dipartimento di Matematica e Informatica



UNIVERSITÀ
degli STUDI
di CATANIA

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

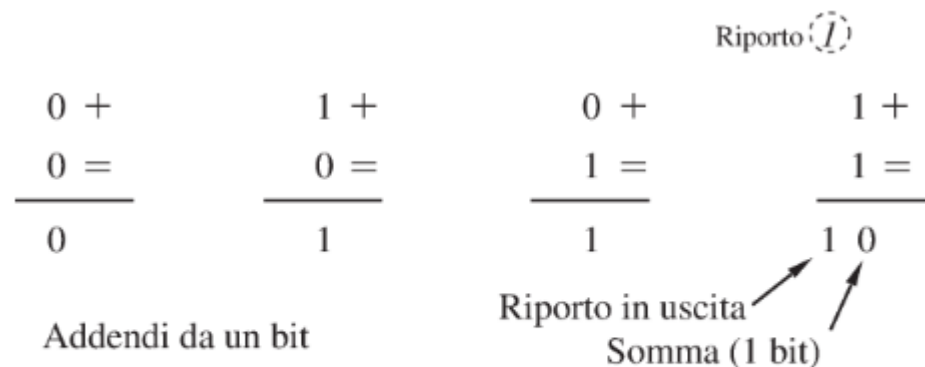


Figura 1.4 - Addizione di numeri a un bit

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

•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

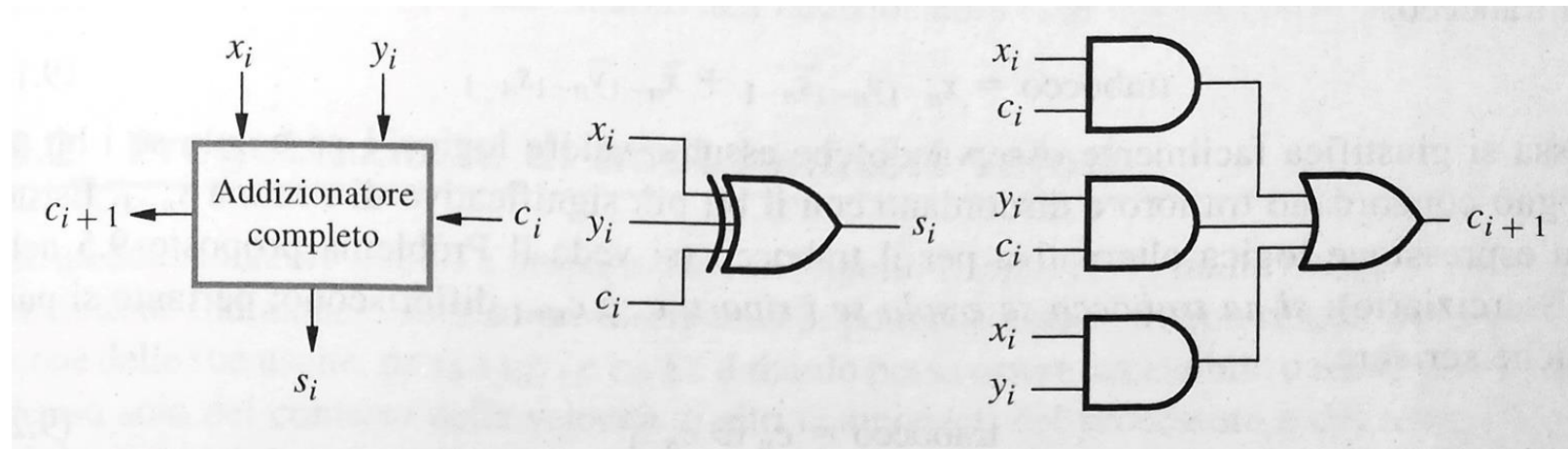
x_i	y_i	Riporto in ingresso c_i	Somma s_i	Riporto in uscita c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i = \bar{x}_i \bar{y}_i \bar{c}_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i \bar{c}_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i c_i + y_i c_i + x_i y_i$$

Addizionatore completo (full adder)

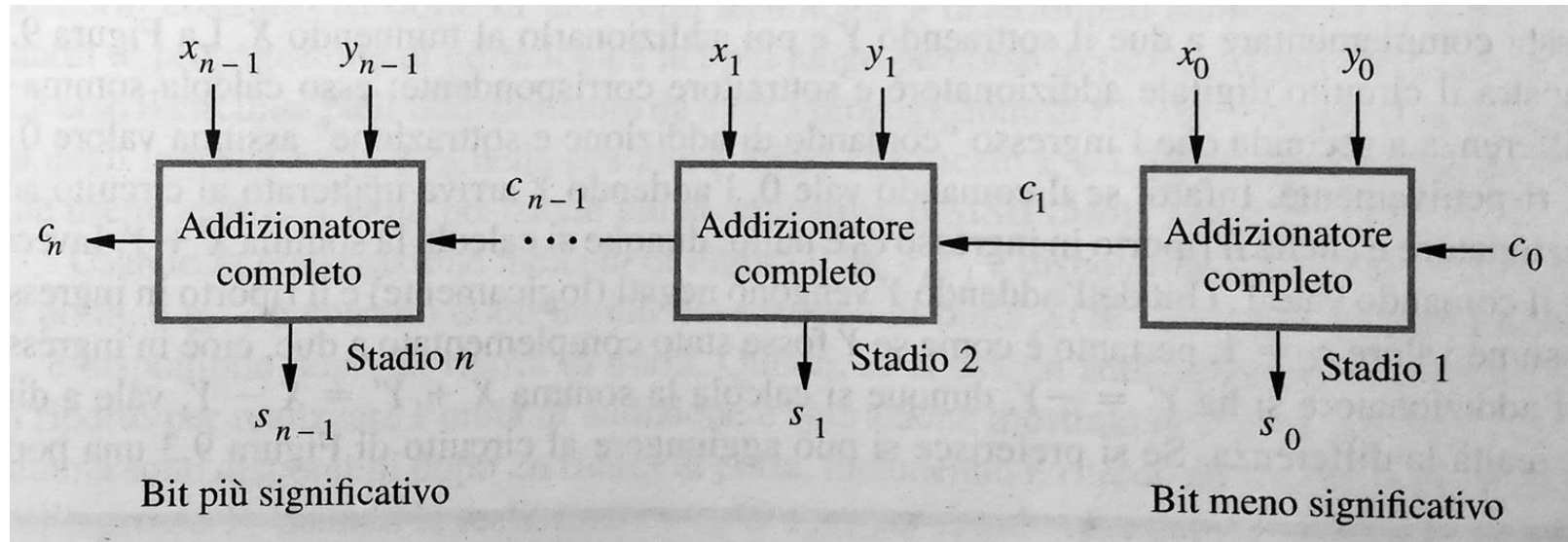
Unendo assieme in un singolo circuito le reti logiche per le funzioni di somma e riporto in uscita si ottiene l'addizionatore completo

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:

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

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

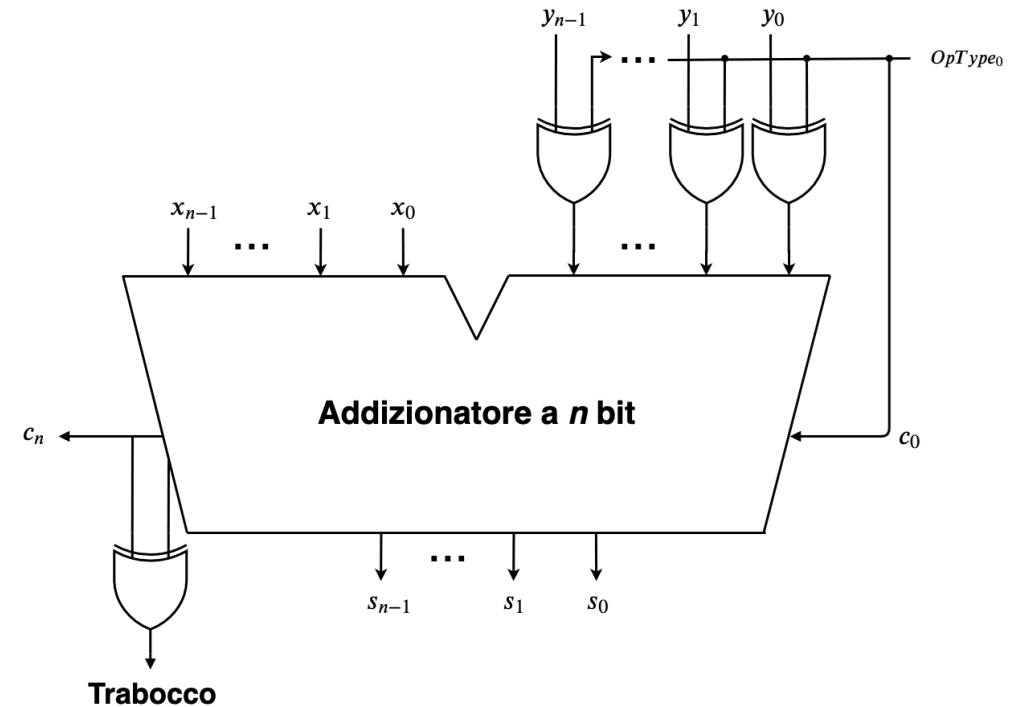
Addizionatore algebrico a n bit

•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 $c_n \oplus c_{n-1}$



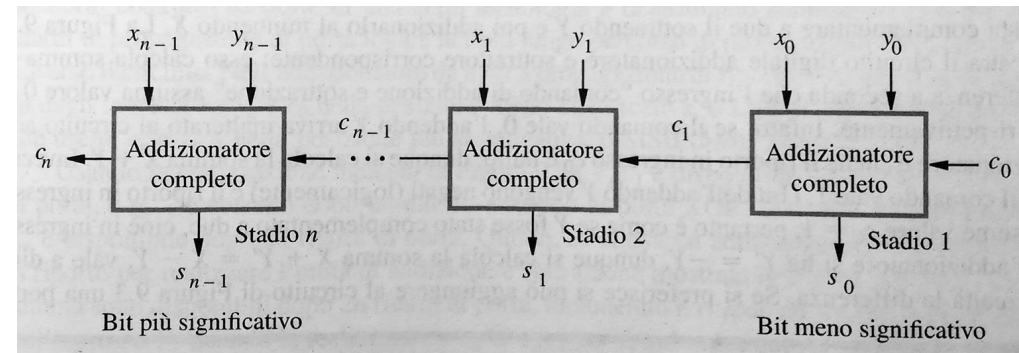
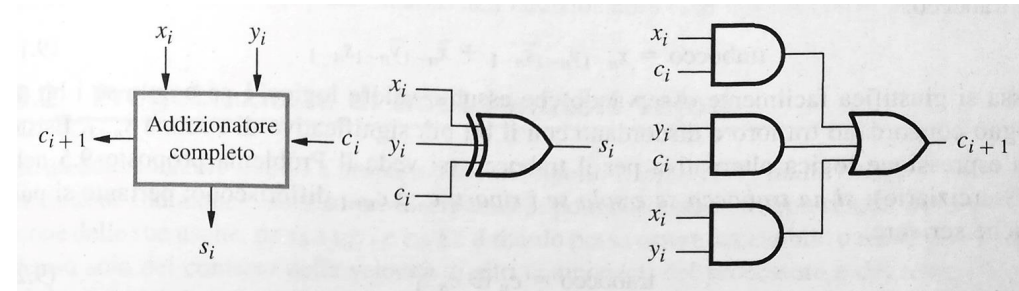
Ritardi ripple carry adder

• 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**



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

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

Fattorizzando l'equazione del riporto si ottiene:

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i) c_i = G_i + P_i c_i$$

$$G_i = x_i y_i \qquad P_i = x_i + y_i$$

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**

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** :

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

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

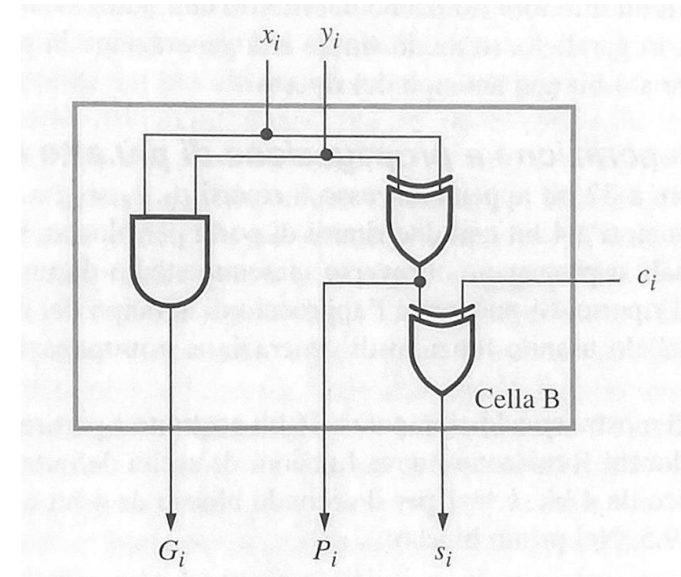
•In questo modo ciascun riporto ci può essere **calcolato in parallelo dopo 2 ritardi di porta**

.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:

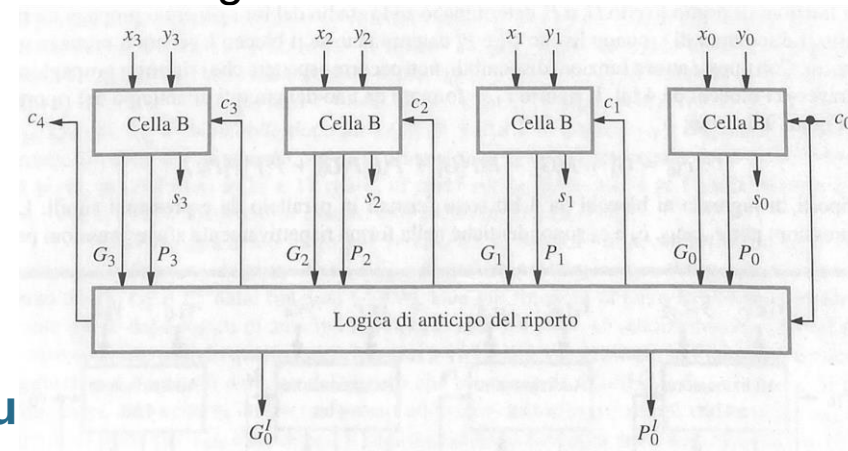
$$c_1 = G_0 + P_0 c_0$$

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

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

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

• **c_4** viene generato dopo **3 ritardi di porta**, mentre il **risu**



• 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:

$$c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0 = \mathbf{G_k^I} + \mathbf{P_k^I}c_0$$

$$\mathbf{G_k^I} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

$$\mathbf{P_k^I} = P_3P_2P_1P_0$$

• 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:

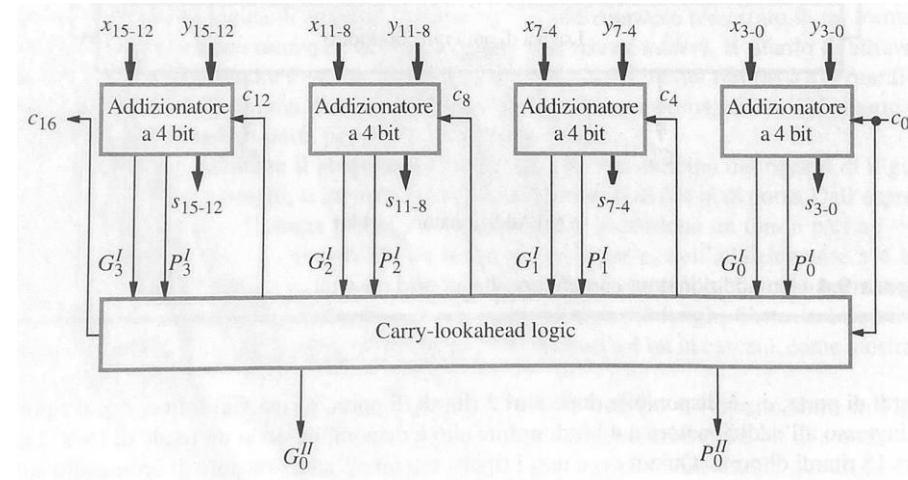
$$c_4 = G_0^I + P_0^I c_0$$

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

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

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

• **c_{16}** 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

			1	1	0	1		(13) Moltiplicando M	
		×	1	0	1	1		(11) Moltiplicatore Q	
						1	1	0	1
				1	1	0	1		
			0	0	0	0			
		1	1	0	1				
	1	0	0	0	1	1	1	1	
									(143) Prodotto P

Circuito moltiplicatore sequenziale

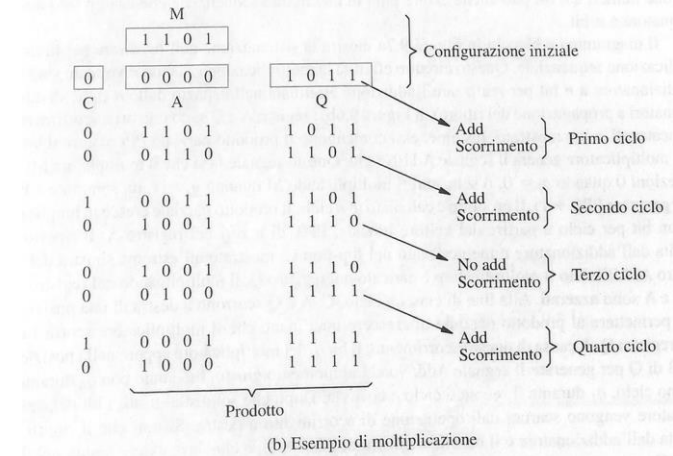
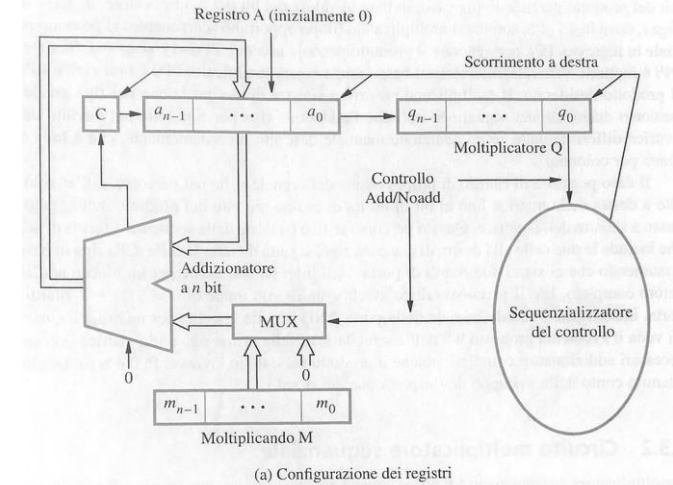
• La moltiplicazione di numeri senza segno può essere realizzata sequenzialmente usando solo **un Addizionatore a n bit e due registri a scorrimento**

• 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

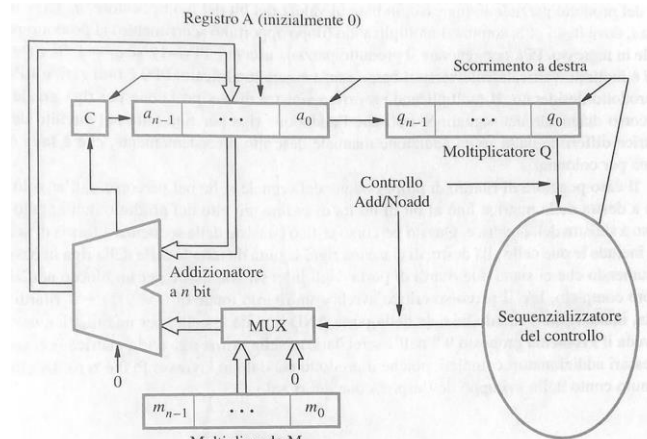
altrimenti:

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

		M												
		1 1 0 1												
0			0 0 0 0				1 0 1 1				} Configurazione iniziale			
C			A				Q							
0			1 1 0 1				1 0 1 1				} Add Scorrimento			
0			0 1 1 0				1 1 0 1							
1			0 0 1 1				1 1 0 1				} Add Scorrimento			
0			1 0 0 1				1 1 1 0							
0			1 0 0 1				1 1 1 0				} No add Scorrimento			
0			0 1 0 0				1 1 1 1							
1			0 0 0 1				1 1 1 1				} Add Scorrimento			
0			1 0 0 0				1 1 1 1							
										Prodotto				

(b) Esempio di moltiplicazione

.Soluzione diretta:

.Moltiplicando positivo: si procede normalmente

.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

						1	0	0	1	1	(-13)
						×	0	1	0	1	(+11)
						<hr/>					
	1	1	1	1	1	1	0	0	1	1	
	1	1	1	1	1	0	0	1	1		
È evidenziata l'estensione del segno	0	0	0	0	0	0	0	0			
	1	1	1	0	0	1	1				
	0	0	0	0	0	0					
	<hr/>										
	1	1	0	1	1	1	0	0	0	1	(-143)

• 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 a due del moltiplicando fatto scorrere di 1 posizione a sinistra

• Generalizzabile per qualsiasi Moltiplicatore:

$$.Q = 1100111011 = 1100000000 + 0000111000 + 0000000011 = 0000000000 + 1100000000 + 0001000000 + 1111111000 + 0000000100 + 1111111111 = -2^8 + 2^6 - 2^3 + 2^2 - 2^0$$

$$.P = -M * 2^8 + M * 2^6 + -M * 2^3 + M * 2^2 + -M * 2^0$$

•L'algoritmo di Booth può essere usato per ottenere una **ricodifica più efficiente**

•La **ricodifica bit-pair** raggruppa **in coppie i bit del moltiplicatore** ricodificato con Booth

•**Ciascuna coppia di bit** del moltiplicatore viene **riscritta come un singolo coefficiente** da moltiplicare al moltiplicando:

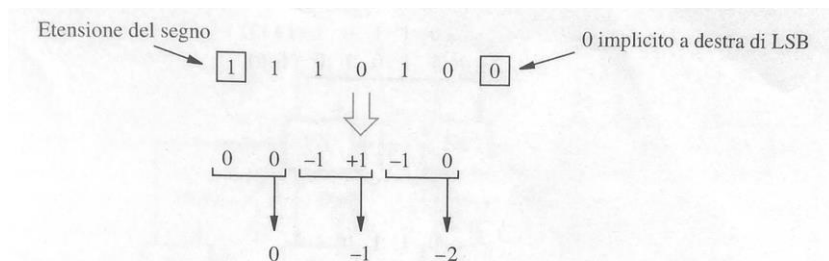
$$(+1, -1) \rightarrow 2M - M \rightarrow 1 * M \rightarrow (0, +1)$$

$$(-1, +1) \rightarrow -2M + M \rightarrow -1 * M \rightarrow (0, -1)$$

$$(+1, 0) \rightarrow 2M + 0M \rightarrow 2 * M \rightarrow (0, +2)$$

Ecc..

•In questo modo **il numero di addizioni** da eseguire durante la moltiplicazione **viene dimezzato**



(a) Esempio di ricodifica bit-pair derivata da ricodifica di Booth

Coppia di bit del moltiplicatore		Bit del moltiplicatore sulla destra $i - 1$	Versione del moltiplicando selezionata in posizione i
$i + 1$	i		
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

(b) Tabella delle selezioni di versione del moltiplicando

•Anche per la divisione di interi prendiamo come esempio l'algoritmo imparato a scuola:

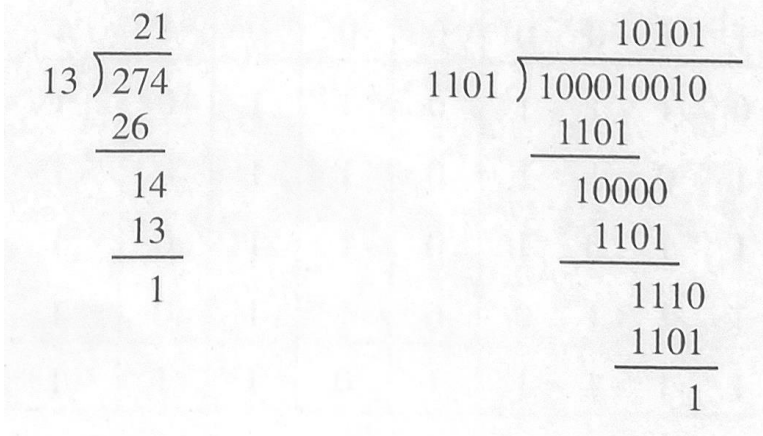
1.Si allinea il divisore con il dividendo a partire da sinistra

2.Si controlla se il divisore è contenuto nel dividendo e nel caso si calcola la divisione parziale tra le cifre allineate:

•si aggiunge il risultato al quoziente

•Si trascrive il resto in basso e gli si aggiunge la prossima cifra del dividendo

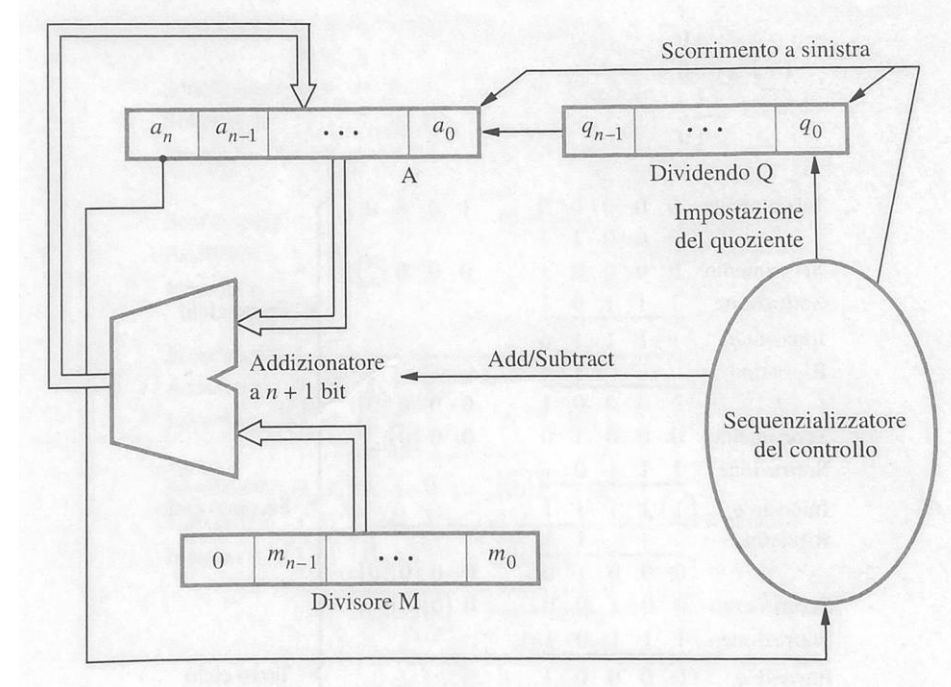
3.Si allinea il divisore e si riprende dal punto 2



Two handwritten examples of integer division are shown. The left example is in decimal: $13 \overline{)274}$. The quotient is 21, with a remainder of 1. The steps shown are: 13 goes into 27 twice (26), leaving a remainder of 14; 13 goes into 14 once (13), leaving a remainder of 1. The right example is in binary: $1101 \overline{)100010010}$. The quotient is 10101, with a remainder of 1. The steps shown are: 1101 goes into 10001 once (1101), leaving a remainder of 1000; 1101 goes into 10000 once (1101), leaving a remainder of 1110; 1101 goes into 11100 once (1101), leaving a remainder of 1.

Divisione con ripristino

- Il circuito sequenziale per la divisione può essere realizzato con **un registro (M)**, **2 shift register (A e Q)** e **un addizionatore a $n+1$ bit**
- Eseguire n volte i seguenti 3 passi:
 - Fare scorrere A e Q a sinistra di una posizione
 - Sottrarre M da A e porre il risultato in A
 - Se il segno di A è 1, porre q_0 a 0 e sommare M ad A; altrimenti, porre q_0 a 1
- All'inizio **M** contiene il Divisore, **A** contiene 0 e **Q** contiene il Dividendo
- Alla fine **A** contiene il Resto e **Q** contiene il Quoziente



Divisione senza ripristino

.L'algoritmo di divisione può essere semplificato eliminando il passo di ripristino

.L'algoritmo senza ripristino si divide in due stadi:

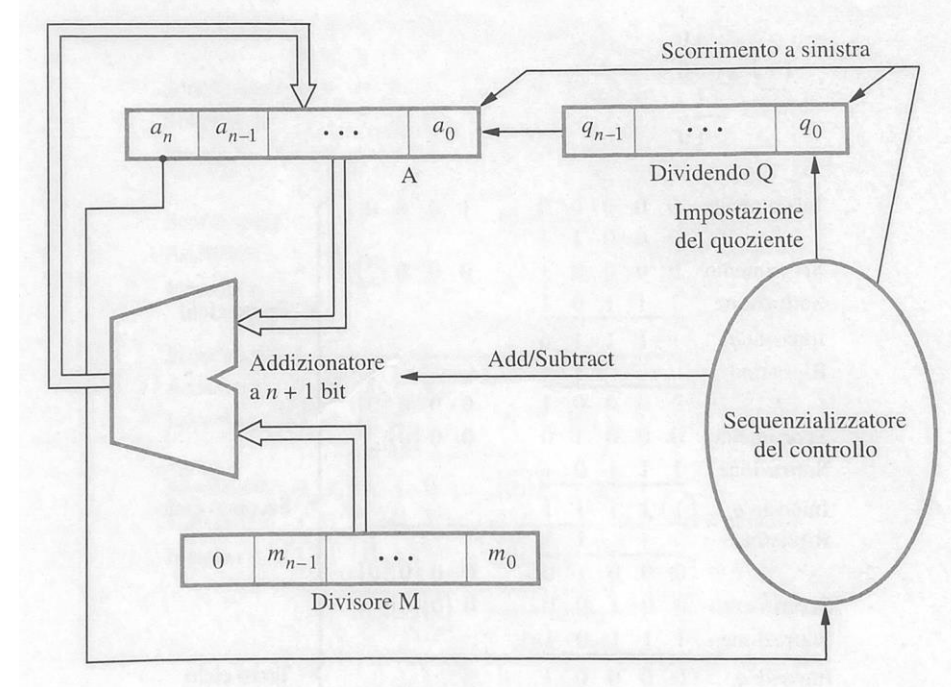
.Stadio 1: Eseguire n volte i seguenti 2 passi

.Se il segno di A è 0, fare scorrere A e Q a sinistra di una posizione e sottrarre M da A ; altrimenti, fare scorrere A e Q a sinistra di una posizione e sommare M ad A

.Se il segno di A è 0, porre q_0 a 1; altrimenti, porre q_0 a 0

.Stadio 2: passi:

.Se il segno di A è 1, sommare M ad A



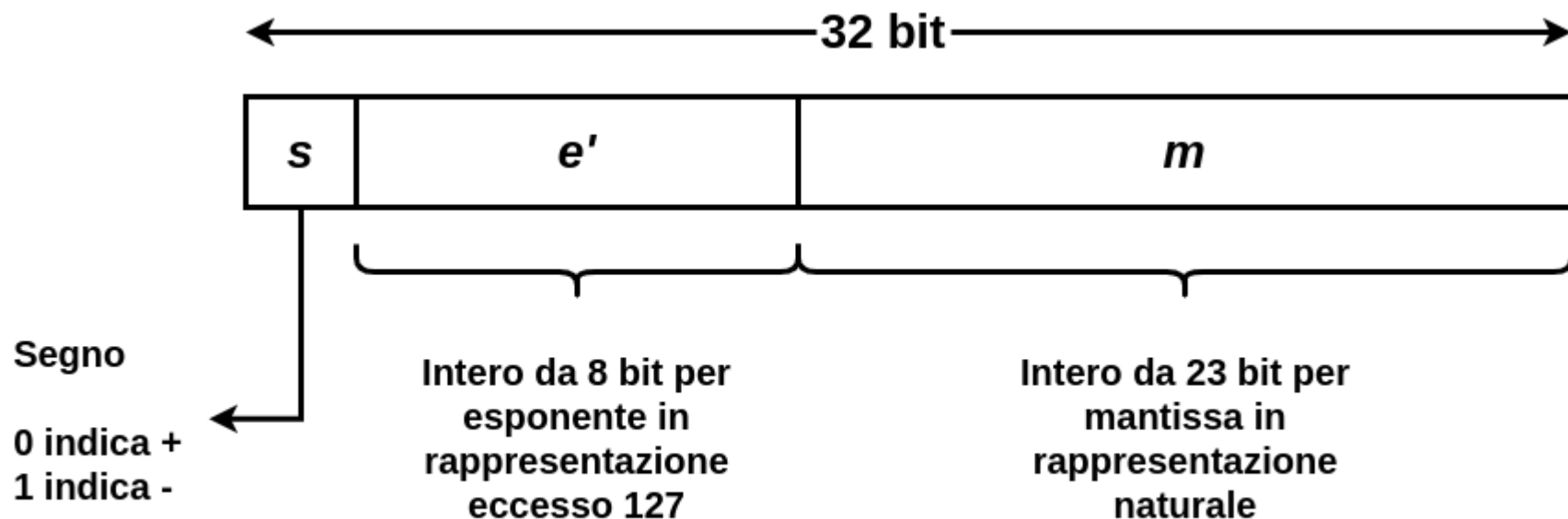
Un numero binario in virgola mobile può quindi essere rappresentato:

- Un **SEGNO** s per il numero
- La **MANTISSA** m (bit significativi escluso il bit più significativo)
- Un **ESPONENTE** e con segno in base 2

$$\text{Valore rappresentato} = \pm 1, m \times 2^e$$

Formato precisione singola (32 bit)

Standard **IEEE 754** numeri **32 bit**



$$0 \leq e' \leq 255$$

$$e = e' - 127$$

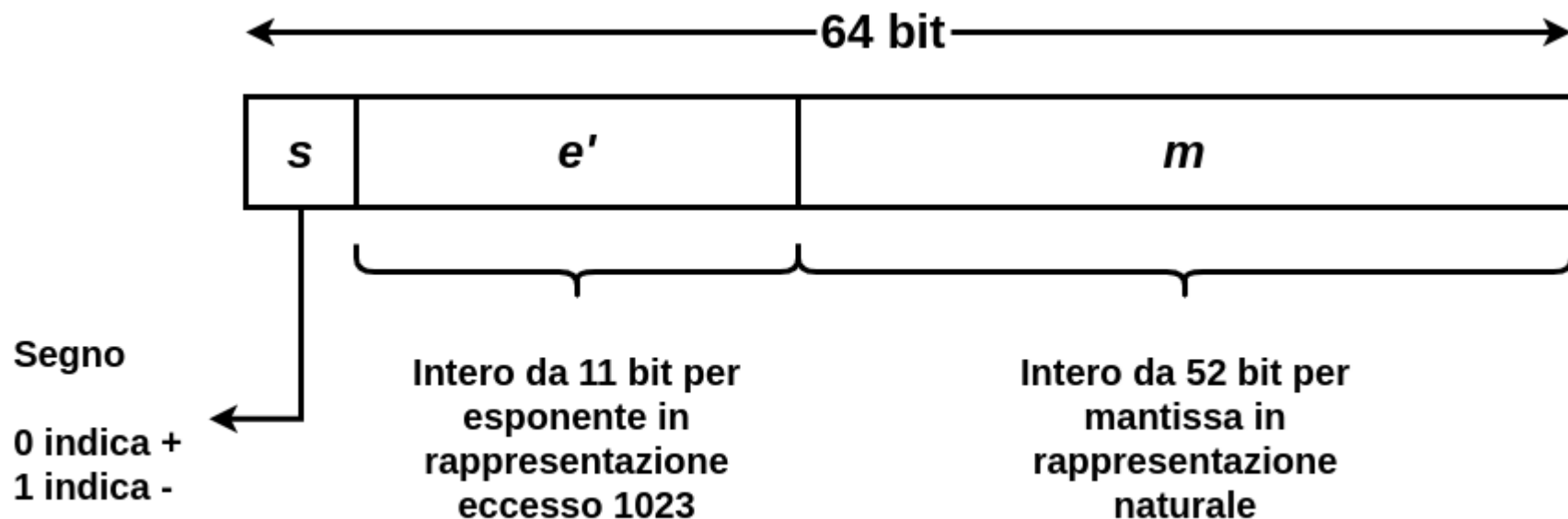
Valori speciali: $e' = 0$, $e' = 255$

Intervallo esponente: $-126 \leq e \leq 127$

Fattore di scala nell'intervallo: $[2^{-126}, 2^{127}]$

Formato precisione doppia (64 bit)

Standard **IEEE 754** numeri **64 bit**



$$0 \leq e' \leq 2047 \quad e = e' - 1023$$

Valori speciali: $e' = 0$, $e' = 2047$

Intervallo esponente: $-1022 \leq e \leq 1023$

Fattore di scala nell'intervallo: $[2^{-1022}, 2^{1023}]$

Alcuni valori dell'esponente sono speciali:

$.e' = 0, m = 0$ rappresenta lo **0 esatto**

$.e' = 255(2047), m = 0$ rappresenta l'infinito ∞

$.e' = 0, m \neq 0$ rappresenta la **forma non normale**

$.e' = 255(2047), m \neq 0$ rappresenta **Not a Number NaN**

1. Scegli il numero con esponente più piccolo e fai scorrere la sua mantissa a destra di un numero di passi uguale alla differenza degli esponenti
2. Poni l'esponente del risultato uguale all'esponente più grande
3. Addiziona i fattori interi (1 || Mantissa) tenendo conto del segno
4. Se il numero risultante non è in forma normale, normalizzalo

1. Addiziona gli esponenti dei fattori e toglì 127 alla somma, ottenendo l'esponente provvisorio del prodotto
2. Moltiplica i fattori interi dei fattori ottenendo segno e fattore intero provvisorio del prodotto
3. Se il numero risultante non è in forma normale, normalizzalo

1. Sottrai l'esponente del divisore dall'esponente del dividendo e aggiungi 127 alla differenza, ottenendo l'esponente provvisorio del rapporto
2. Calcola il quoziente del fattore intero del dividendo rispetto a quello del divisore, ottenendo segno e fattore intero provvisorio del rapporto
3. Se il numero risultante non è in forma normale, normalizzalo