Circuiti combinatori notevoli e circuiti aritmetici

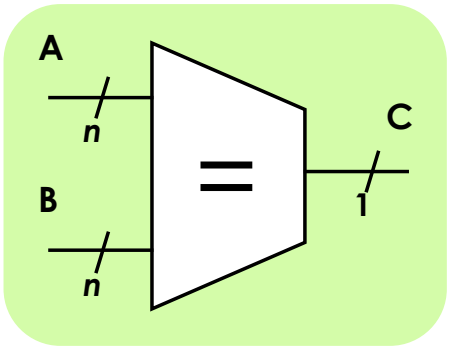
F. Pedersini

Dipartimento di Informatica
Università degli Studi di Milano

COMPARATORE

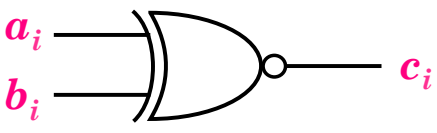
- Confronta **2** parole di **n** bit, verificando se sono uguali
  - IN: **2** gruppi di **n** bit
  - OUT: **1** bit

OUT = 1 se: A = B (bit a bit)
OUT = 0 se: A ≠ B



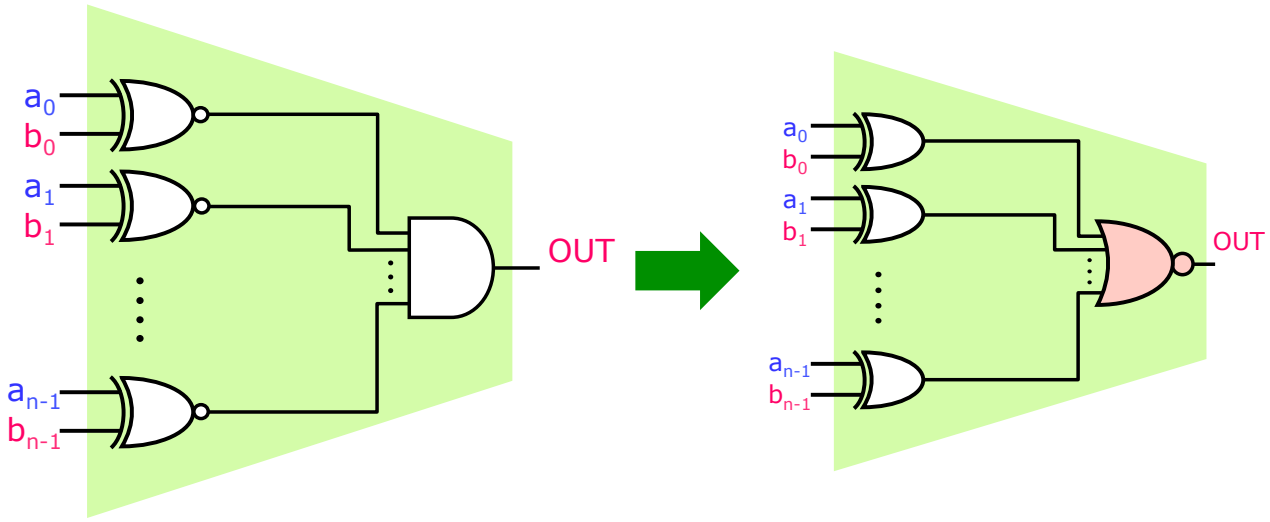
A_0	B_0	C_0
0	0	1
0	1	0
1	0	0
1	1	1

A_1	B_1	C_1
0	0	1
0	1	0
1	0	0
1	1	1



Comparatore

Comparatore a ***n* ingressi**: schema circuitale



n porte **XNOR** + 1 **AND** ad **n** ingressi

n porte **XOR** + 1 **NOR** ad **n** ingressi

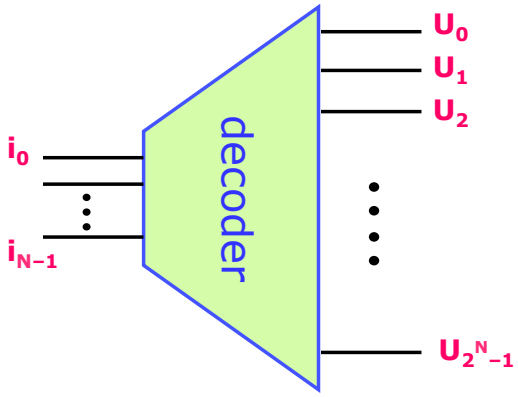
Decodificatore (decoder)

DECODER:

- **N** ingressi,
- $\mathbf{2^N}$ uscite

Dato un numero ***I*** espresso sugli ingressi ($0 \le I \le 2^N-1$),

- viene asserita (posta a "1") l'uscita U_I corrispondente
- tutte le altre uscite vengono poste a "0"



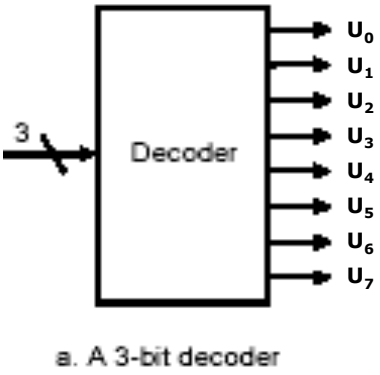
Applicazioni:

- *Utilizzato ad es. nelle memorie, per selezionare una cella mediante il suo indirizzo.*

La funzione decoder

Decoder a **3 ingressi ➔ 2^3=8 uscite**

Tabella di verità:



a. A 3-bit decoder

A	B	C	U_0	U_1	U_2	U_3	U_4	U_5	U_6	U_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Ogni uscita corrisponde ad uno dei mintermini

- 2^N uscite ←→ 2^N mintermini

Cammino critico ?

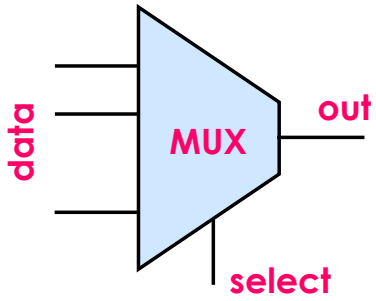
Multiplexer

MULTIPLEXER (MUX):

Operatore di **selezione**

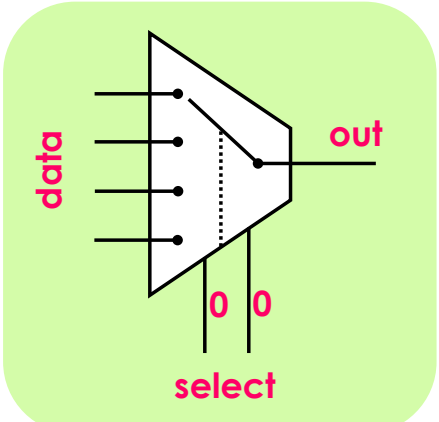
IN: **2^n** linee di input (**data**)
n linee di controllo (**select**)

OUT: **1** linea



Funzionamento:

Dato il numero ***i*** ($0 \le i \le 2^n-1$) presente sull'ingresso **select**,
il valore sulla linea di ingresso (**data**) *i-esima* viene propagato sull'uscita.



Quante linee di selezione? $\mathbf{k} = \log_2 n$

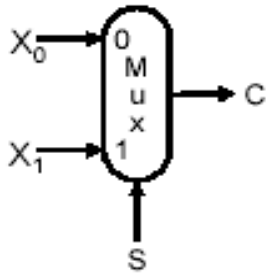
Linee di input: $n = 4$

Linee di controllo: $k = ceil(\log_2 4) = 2$

Multiplexer binario

MUX a **2 ingressi:** **n = 2, k = 1**

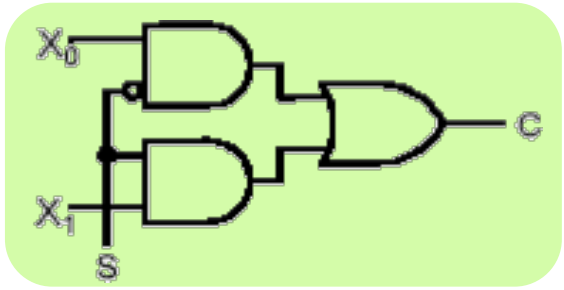
- Selezione S "apre" la porta opportuna
- Circuito logico a **3 ingressi, 1 uscita**



S	X_0	X_1	C
0	**0**	0	**0**
0	**0**	1	**0**
0	**1**	0	**1**
0	**1**	1	**1**
1	0	**0**	**0**
1	0	**1**	**1**
1	1	**0**	**0**
1	1	**1**	**1**

I forma canonica (SoP):

$$C = \overline{S}X_0\overline{X_1} + \overline{S}X_0X_1 + S\overline{X_0}X_1 + SX_0X_1 =$$
$$= \overline{S}X_0\left(\overline{X_1} + X_1\right) + SX_1\left(\overline{X_0} + X_0\right)$$
$$= \overline{S}X_0 + SX_1$$



Sintesi Multiplexer

*Esercizio: sintesi MUX a 2 ingressi in **II forma canonica***

$$Y = \left(S + X_0 + X_1\right)\left(S + X_0 + \overline{X_1}\right)\left(\overline{S} + X_0 + X_1\right)\left(\overline{S} + \overline{X_0} + X_1\right) =$$

$$\begin{cases} a = \overline{S} + X_1 \\ b = S + X_0 \end{cases}$$

$$= \left[\left(b + X_1\right)\left(b + \overline{X_1}\right)\right] \cdot \left[\left(a + X_0\right)\left(a + \overline{X_0}\right)\right] =$$

$$= \left[b + b\left(X_1 + \overline{X_1}\right) + X_1\overline{X_1}\right] \cdot \left[a\right] = ba =$$

$$= \left(S + X_0\right)\left(\overline{S} + X_1\right) = \quad \textit{(PoS)}$$

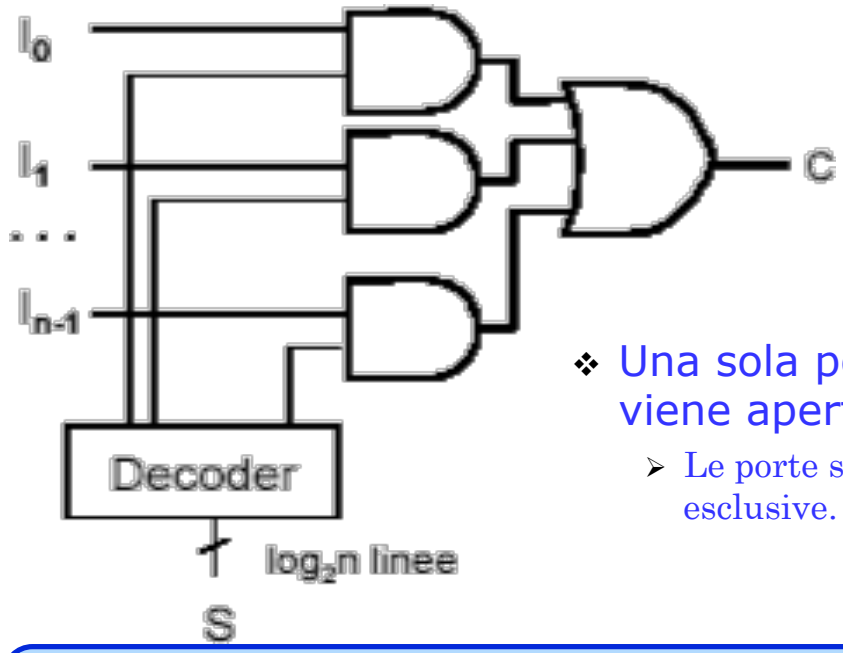
S	X_0	X_1	C
0	**0**	0	**0**
0	**0**	1	**0**
0	**1**	0	**1**
0	**1**	1	**1**
1	0	**0**	**0**
1	0	**1**	**1**
1	1	**0**	**0**
1	1	**1**	**1**

$$= S\overline{S} + SX_1 + \overline{S}X_0 + X_0X_1 =$$

$$= SX_1 + \overline{S}X_0 + X_0X_1\left(S + \overline{S}\right) = SX_1\left(1 + X_0\right) + \overline{S}X_0\left(1 + X_1\right) = SX_1 + \overline{S}X_0 \quad \textit{(SoP)}$$

MUX a ***n*** ingressi:

- si utilizza un DECODER: $\log_2(n)$ ingressi, ***n*** uscite



- Una sola porta alla volta viene aperta dal segnale S.
  - Le porte sono mutuamente esclusive.

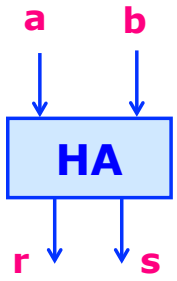
Circuiti aritmetici

circuiti combinatori che generano risultati
di operazioni aritmetiche su numeri binari

- somma
- moltiplicazione
- ALU

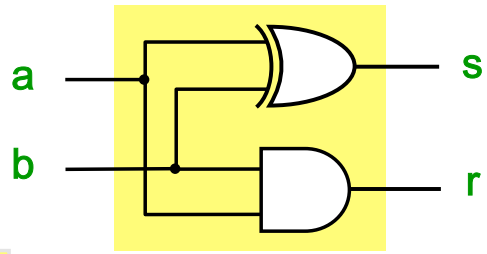
HALF Adder (1 bit)

- Somma aritmetica tra 2 parole di 1 bit
  - 2 ingressi: addendi: **a , b**
  - 2 uscite: somma: **s**
    riporto: **r**



SOMMA:
tabella della verità

a	b	somma	riporto
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

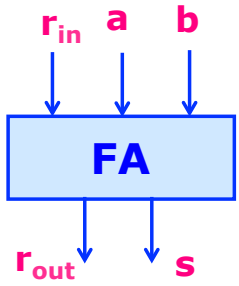


$s = a \oplus b$
$r = ab$

FULL Adder (1 bit)

Sommando numeri di più bit, c'è un problema:
va gestito anche il riporto in ingresso

- 3 ingressi: **a, b, r_{in}**
- 2 uscite: **s, r_{OUT}**



	a	b	r_{in}	r_{out}	s
m_0	0	0	0	0	0
m_1	0	1	0	0	1
m_2	1	0	0	0	1
m_3	1	1	0	1	0
m_4	0	0	1	0	1
m_5	0	1	1	1	0
m_6	1	0	1	1	0
m_7	1	1	1	1	1

SOP:

$$s = m_1 + m_2 + m_4 + m_7$$
$$r_{out} = m_3 + m_5 + m_6 + m_7$$

$$s = \bar{a}\bar{b}r_{in} + \bar{a}b\bar{r}_{in} + a\bar{b}\bar{r}_{in} + abr_{in}$$

$$r_{out} = ab\bar{r}_{in} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in}$$

Semplificando le espressioni

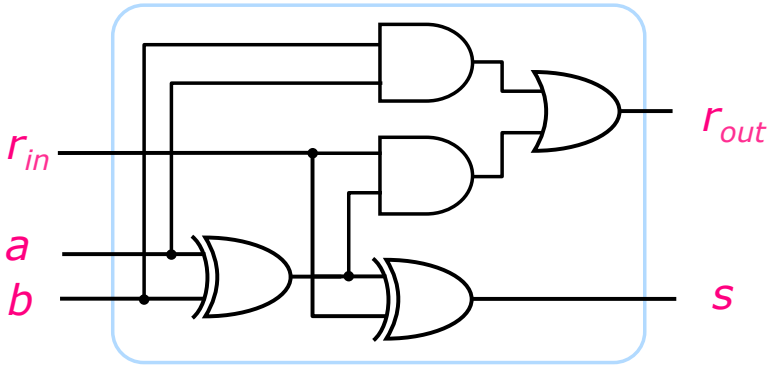
$$s = \bar{a}b\bar{r}_{in} + a\bar{b}\bar{r}_{in} + \bar{a}\bar{b}r_{in} + abr_{in} =$$
$$= (a \oplus b)\bar{r}_{in} + (ab + \bar{a}\bar{b})r_{in} =$$
$$= (a \oplus b)\bar{r}_{in} + \overline{(a \oplus b)}r_{in} =$$
$$= (a \oplus b) \oplus r_{in}$$

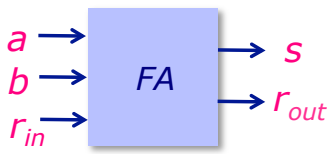
$$r_{out} = ab\bar{r}_{in} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in} =$$
$$= ab(r_{in} + \bar{r}_{in}) + (a\bar{b} + \bar{a}b)r_{in} =$$
$$= ab + (a \oplus b)r_{in} = \cdots = ab + (a + b)r_{in}$$

Dimostrate l'equivalenza

Implementazione circuitale

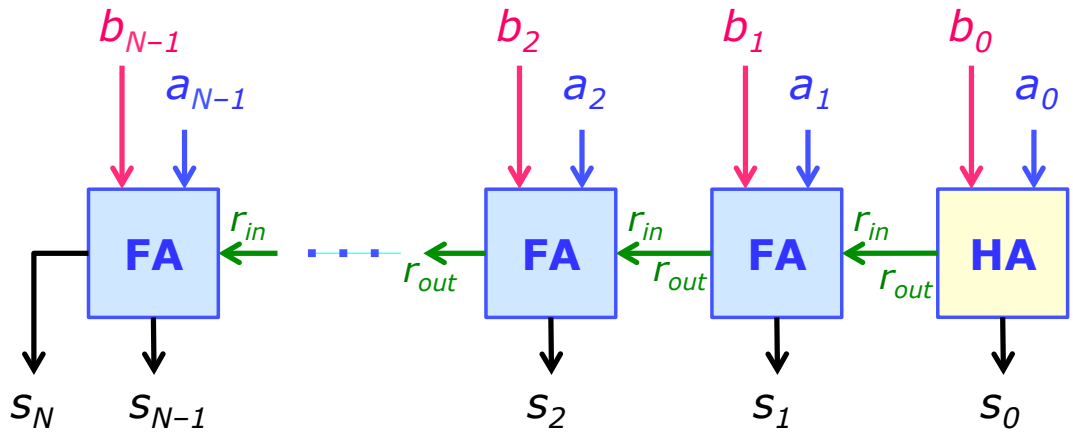
$$s = (a \oplus b)\bar{r}_{in} + \overline{(a \oplus b)}r_{in} = (a \oplus b) \oplus r_{in}$$
$$r_{out} = ab + (a \oplus b)r_{in}$$





Sommatore di parole di N bit

- Sommatore a propagazione di riporto
  - **IN**: **2** parole di **N** bit
  - **OUT**: somma di **N+1** bit



Cammino critico? (HA=1 ; FA=3)

$C = 3(N-1)+1 = 3N - 2$

Moltiplicazione binaria

Moltiplicatore **Moltiplicando**

1 1 0 1 1 x (27_{10})
1 1 1 = (7_{10})

1 1 1 1 1 1 ← *riporti*
1 1 0 1 1
1 1 0 1 1 –
1 1 0 1 1 – –

1 0 1 1 1 1 0 1 (189_{10})

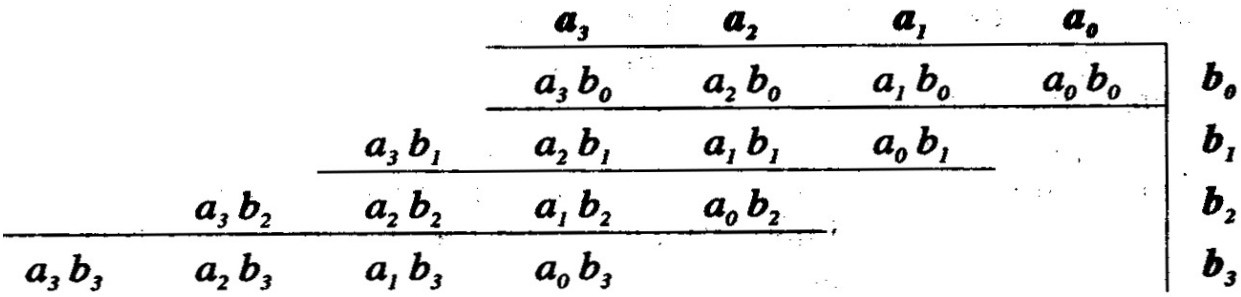
Prodotto

Come fare una moltiplicazione con circuiti logici?

Possiamo scomporre l'operazione in due stadi:

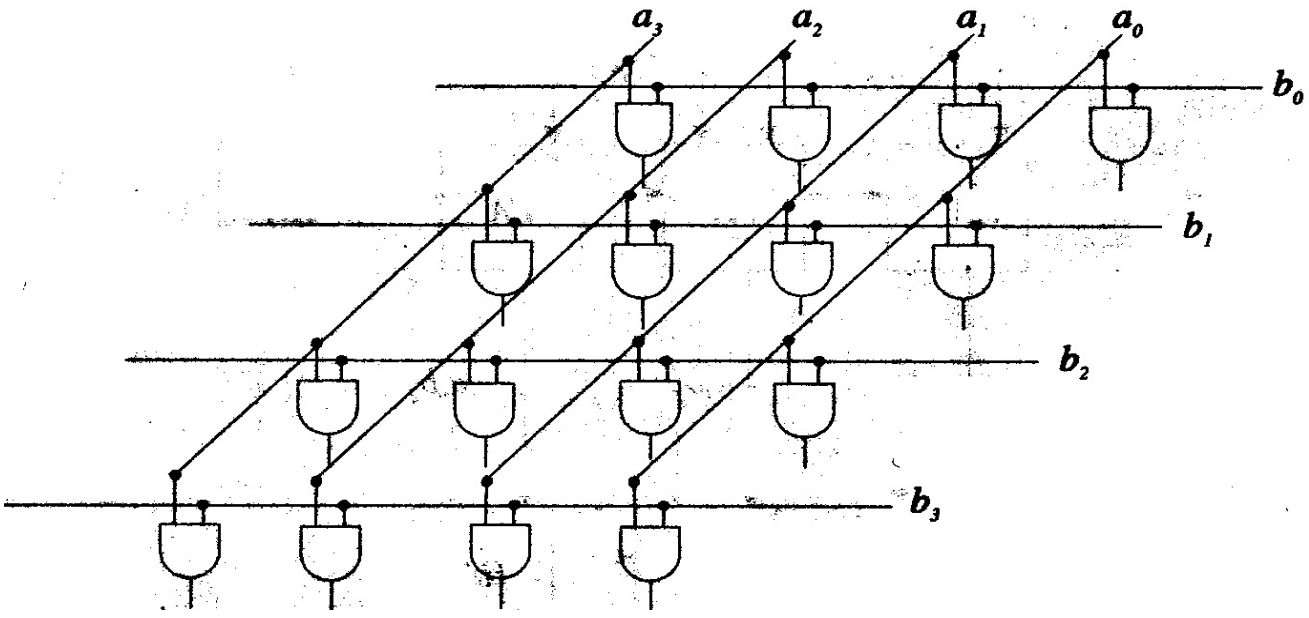
- Stadio 1: **prodotti parziali**
  - si mette in AND ciascun bit del moltiplicatore con i bit corrispondenti del moltiplicando
- Stadio 2: **somme**
  - si effettuano le somme (half/full adder) dei bit sulle righe contenenti i prodotti parziali

La matrice dei prodotti parziali

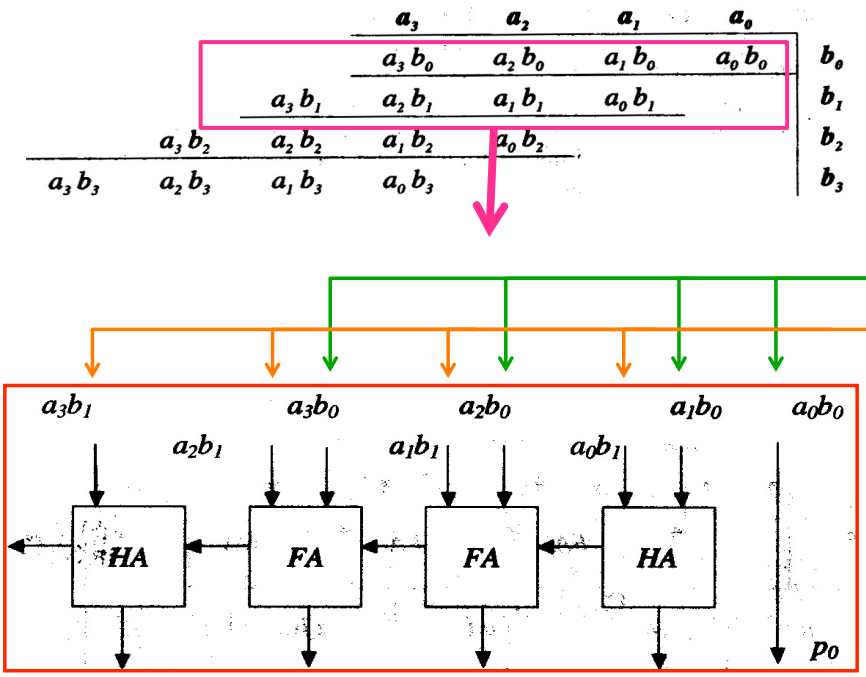


In binario i prodotti parziali sono degli AND

Il circuito che effettua i prodotti

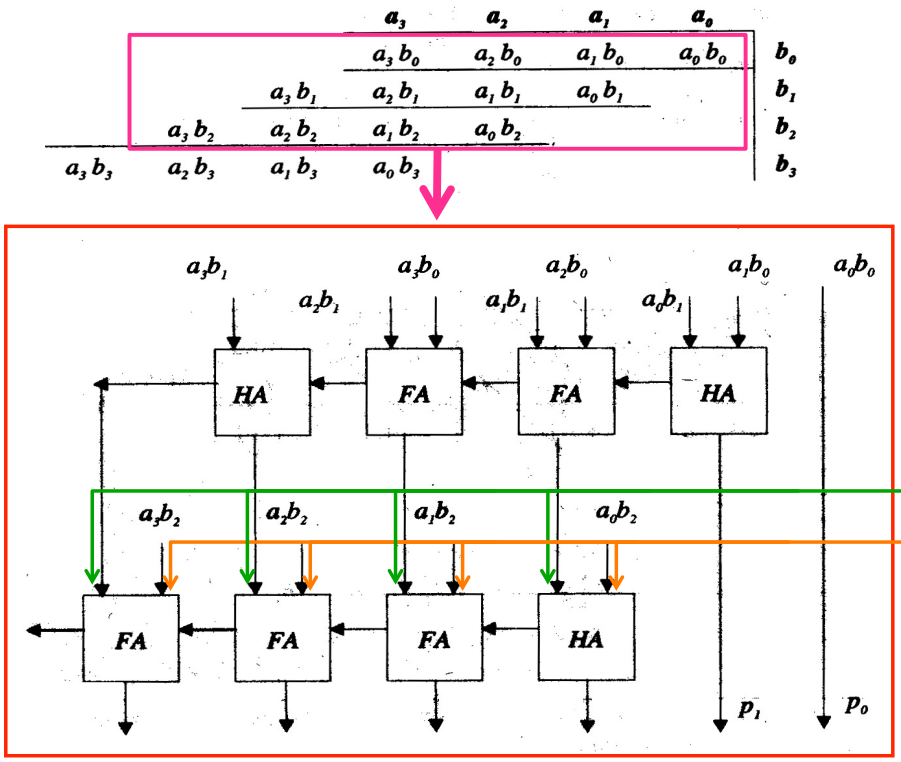


Somma – prime 2 righe dei prodotti parziali



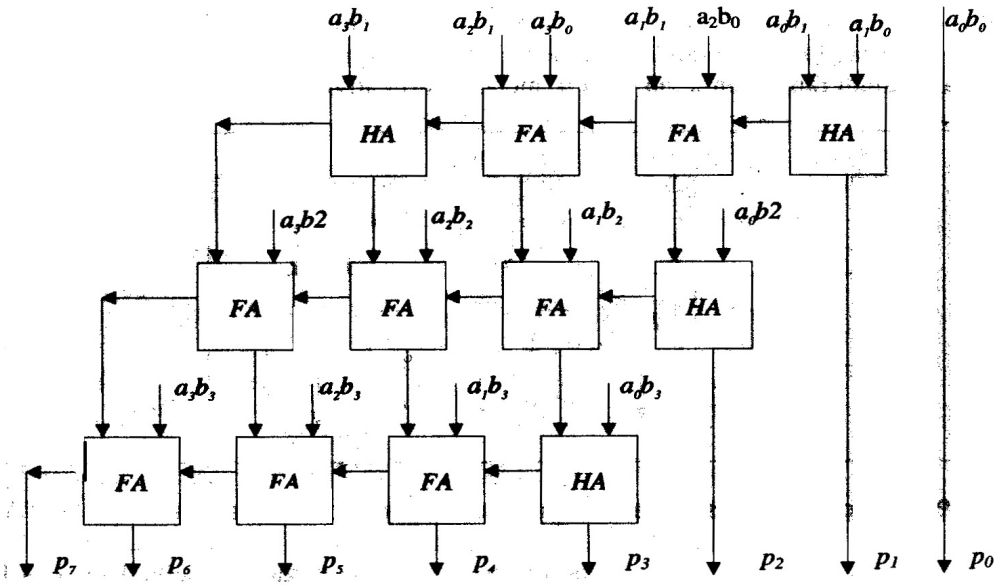

```
  1 0 1 1  x
  0 1 1 1  =
 ---------
    1 1 1
    1 0 1 1  +
  1 0 1 1 -
 ---------
1
  1 0 0 0 0 1  +
  1 0 1 1 - -
 ---------
1 0 0 1 1 0 1
```


Somma della terza riga




```
  1 0 1 1  x
  0 1 1 1  =
 ---------
    1 1 1
    1 0 1 1  +
  1 0 1 1 -
 ---------
1
  1 0 0 0 0 1  +
  1 0 1 1 - -
 ---------
1 0 0 1 1 0 1
```

Somma prodotti parziali – circuito completo



fattori A e B di **N bit** → prodotto P di **2N bit**
(A di **M** bit, B di **N** bit → prodotto P di **M+N bit**)

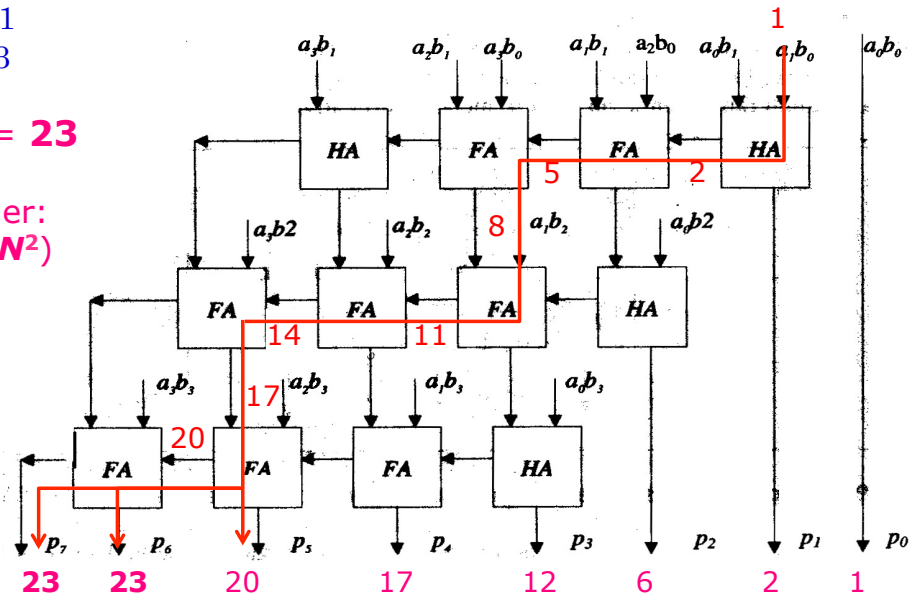
Valutazione del cammino critico

Sommatore a 4 bit: cammino critico

- **N = 4**
  - Ritardo AND = 1
  - Ritardo HA = 1
  - Ritardo FA = 3

Cammino critico = **23**

N–1 strati di ***N*** adder:
(cresce circa come N^2)



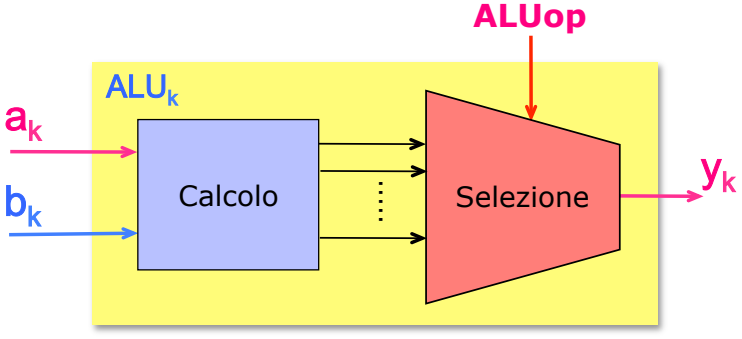
ALU: Arithmetic-Logic Unit

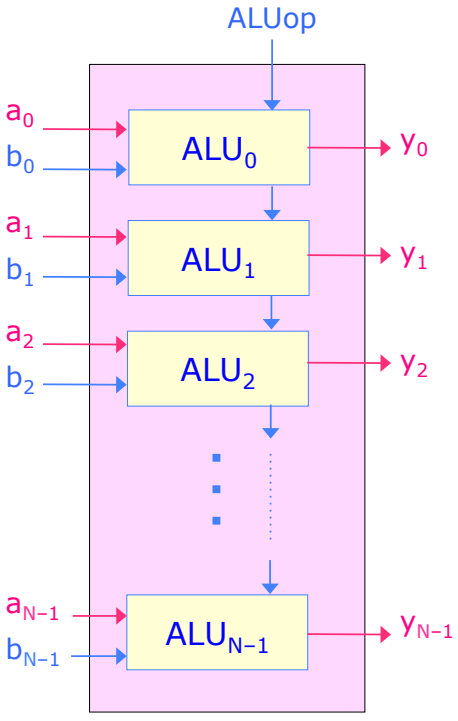
- Esegue le operazioni aritmetiche e logiche
  + , – , x , ÷ , ...
  and , or , not , xor , = , ≠ , ...
- Normalmente integrata nel processore
  - Inizio anni '90 → introduzione con il nome di co-processore matematico
  - Le ALU non compaiono solamente nei micro-processori
- E' un circuito combinatorio
  - Rappresentabile come insieme di funzioni logiche
- Opera su **parole (N bit)**
  - 6502, 8080, Z-80 8 bit
  - MIPS, 80386: 32 bit
  - AMD Athlon64, INTEL i3/5/7: 64 bit
- Struttura modulare
  - Blocchi funzionali da 1 bit, replicati N volte

ALU a N bit → N ALU a 1 bit (ALU elementari)

Struttura **ALU elementare**:

- Ingressi: Operandi: a_k, b_k
  Riporto in ingresso: r_{in}
  Selettore operazione: ALUop
- Uscite: Risultato: y_k
  Riporto in uscita: r_{out}





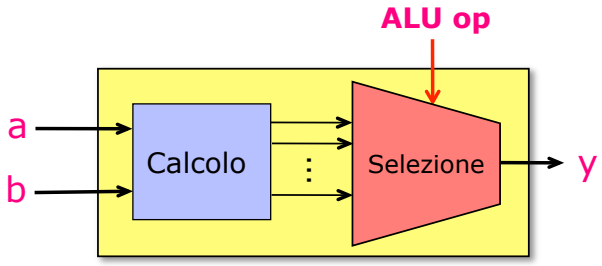
Progettazione della ALU

Operazioni logiche: **AND/OR**

- Selezionabile

❖ Componenti:

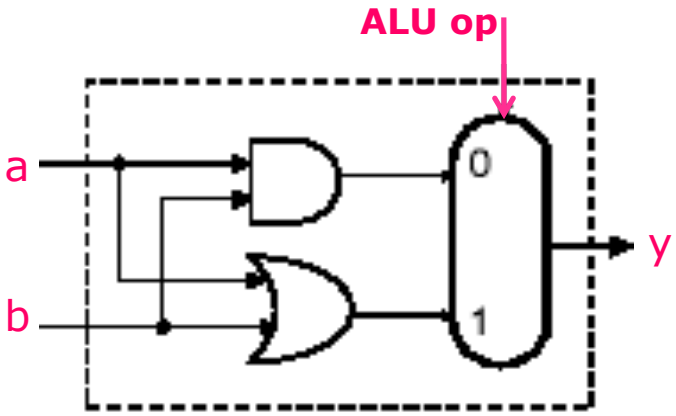
- 1 porta AND
- 1 porta OR
- 1 Multiplexer (MUX)



Selezione:

ALUop = 0 → y = **AND**(a,b)

ALUop = 1 → y = **OR**(a,b)

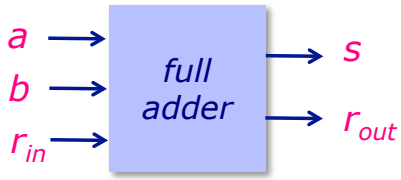


FULL Adder (1 bit)

Operazioni aritmetiche: **SOMMA**

❖ **FA: gestione dei riporti (in/out)**

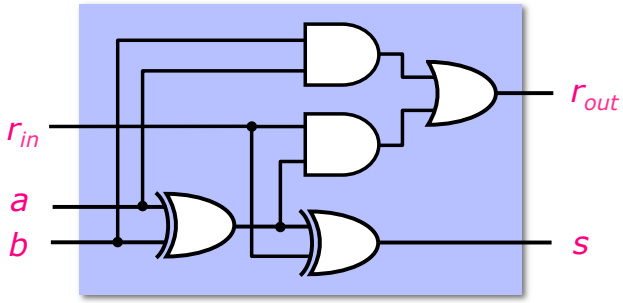
- 3 ingressi: **a, b, r_{IN}**
- 2 uscite: **s, r_{OUT}**



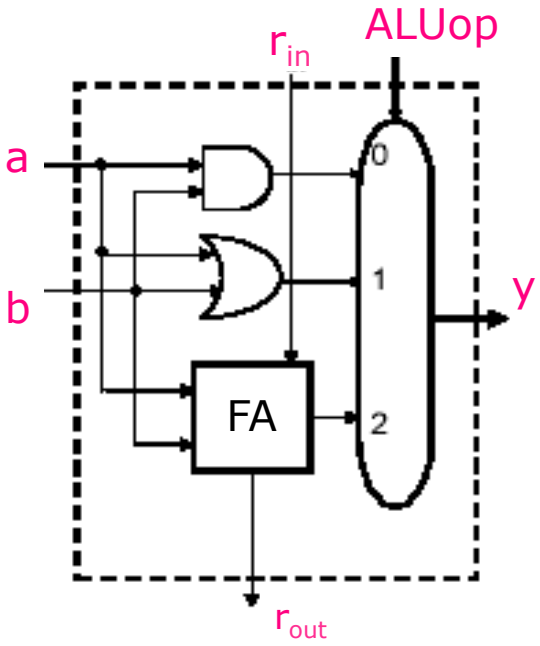
a	b	r_{in}	r_{out}	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

$$s = \bar{a}\bar{b}r_{in} + a\bar{b}\bar{r}_{in} + \bar{a}b\bar{r}_{in} + abr_{in} = a \oplus b \oplus r_{in}$$

$$r_{out} = ab\bar{r}_{in} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in} = ab + (a \oplus b)r_{in}$$



Operazioni: **OR, AND, somma**



Selezione:

- **ALUop** a 2 bit
  - **00: AND: s = a and b**
  - **01: OR: s = a or b**
  - **10: +: $s/r_{out} = a + b + r_{in}$**

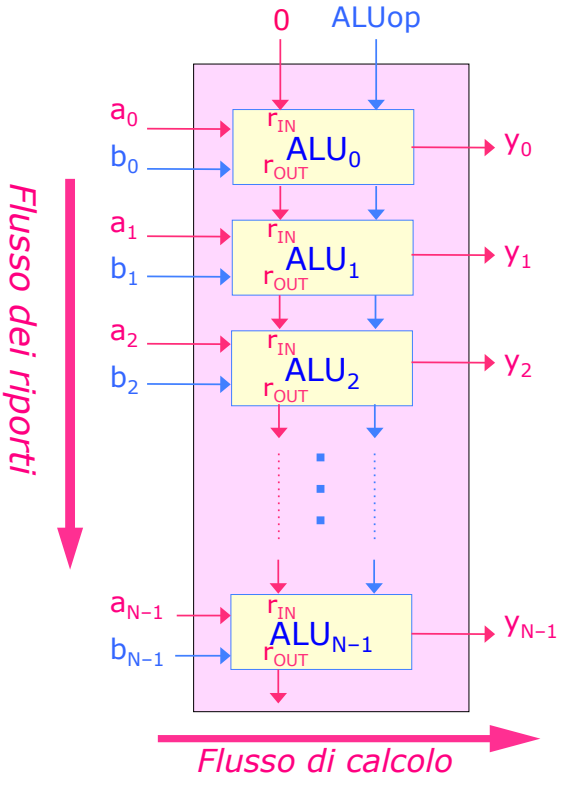
ALU a N bit

- *Come collegare N ALU a 1 bit per ottenere una ALU a N bit?*

ALU a ***N*** bit:

N ALU in parallelo
+ propagazione riporti

- Problema:
  il cammino critico si allunga
  → limite alla velocità di calcolo



Sottrazione

Sottrazione → addizione dell'opposto: **a – b = a + (– b)**

- Posso farlo con gli stessi circuiti dell'addizione, ma devo costruire **–b** a partire da **b**

Complemento a 2: –b = not(b) + 1

- Inversione logica bit-a-bit
- Aggiunta della costante "1": pongo $r_{in}(0) = 1$

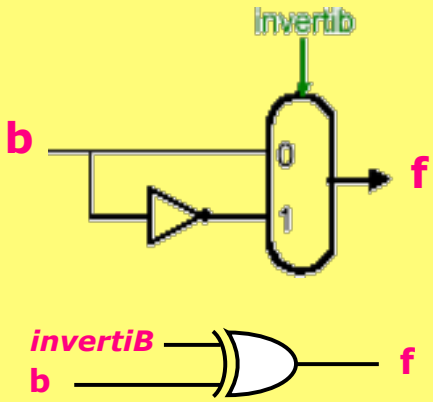
Inversione logica bit-a-bit

invertiB	b	f
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

→ **f = b *XOR* invertiB** →

Invertitore:

If (invertib=1) → **f = ~b**
else → **f = b**



ALU - bit i-esimo

ALU N bit – operazioni: AND, OR, +, –
→ N blocchi di ALU elementare: ALU_i

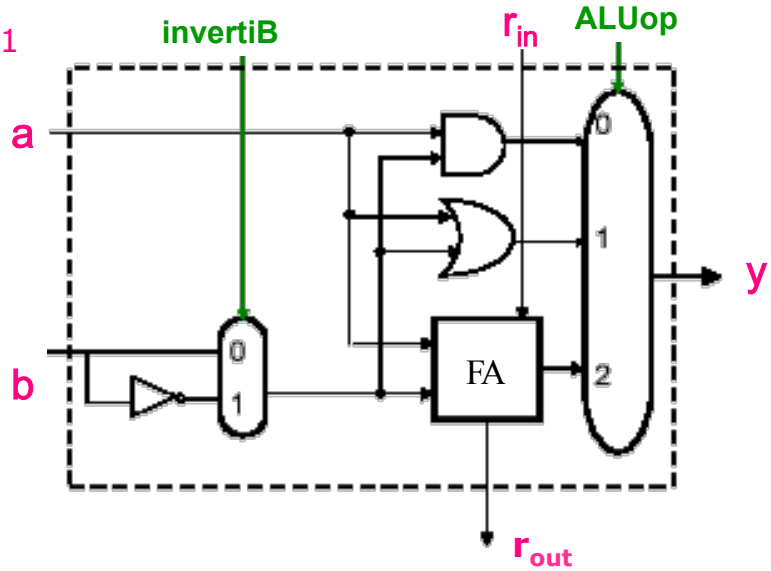
ALU_i (bit i-esimo)

Propagazione riporti:
$r_{in}(i) = r_{out}(i-1)$ i=0, 1, … , N–1

Addizione: $r_{in}(0) = 0$, **invertiB = 0**

Sottrazione: $r_{in}(0) = 1$, **invertiB = 1**

InvB / ALUop	funzione
0 00	**and**
0 01	**or**
0 10	**+**
1 10	**–**



ALU a N-bit

Operazioni:
AND , OR , + , –

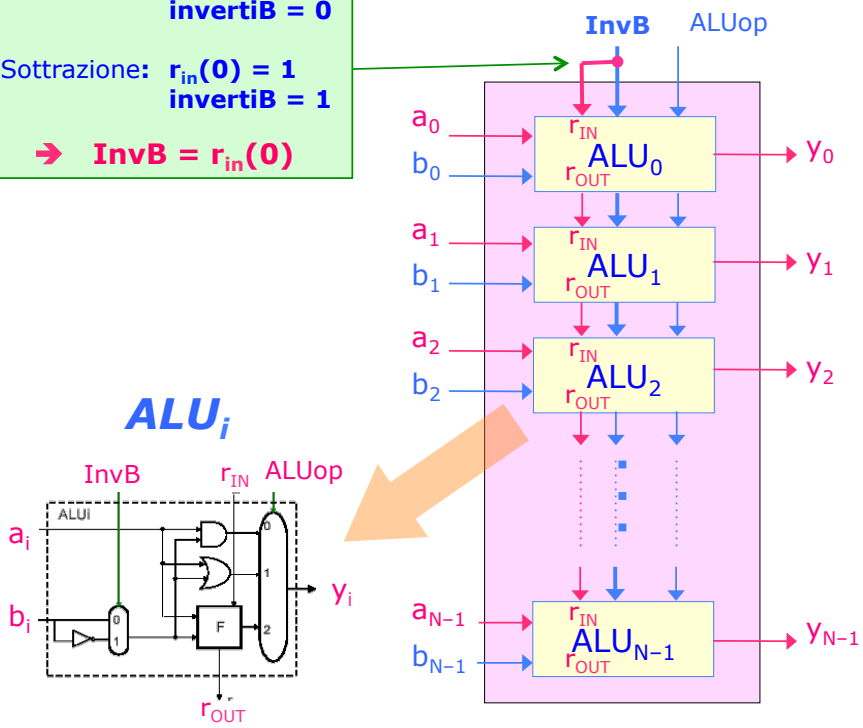
Addizione: $r_{in}(0) = 0$, **invertiB = 0**

Sottrazione: $r_{in}(0) = 1$, **invertiB = 1**

➔ **InvB = $r_{in}(0)$**

Codici operazione

InvB	ALUop	operaz.
0	00	AND
0	01	OR
0	10	+
1	10	–



Comparazione

Operazione fondamentale per dirigere il flusso di esecuzione (test, cicli...)

if a < b then y = 1 else y = 0
if (a<b) → y = [0 0 0 … 0 1]
else → y = [0 0 0 … 0 0]

Implementazione:

if ALUop = "comparazione" **then**
y(1) = y(2) = … = y(N–1) = 0
if (a < b) **then** *y(0) = 1*
else *y(0) = 0*

Devo:

- Imporre tutti i bit **y(1) … y(N–1)** a **0**;
- Calcolare **y(0)** in base alla condizione **a < b**

Come sviluppare la comparazione?

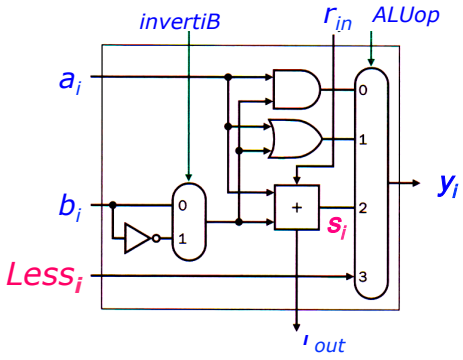
Idea: **a < b → a – b < 0 → $s_{N-1} = 1$**

- se a<b, allora la differenza: **a–b < 0**
- in complemento a 2, il MSB della somma (bit di segno) è **1 per numeri negativi:** **s = a–b < 0 →** **$s_{N-1} = 1$**

Implementazione:

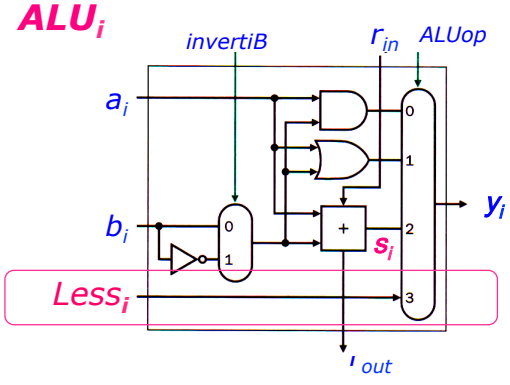
- Nuovo ingresso: **Less**
  IF: ALUop = "comparazione" → **s_i = $Less_i$**
- Operazioni:
  - Calcolare la differenza (a – b) (senza mandarla in uscita)
  - Inviare **0** a **LESS** di **ALU_1, ALU_2, …, ALU_{N-1}**
  - Inviare l'uscita del sommatore di **ALU_{N-1}** a **LESS** di **ALU_0**

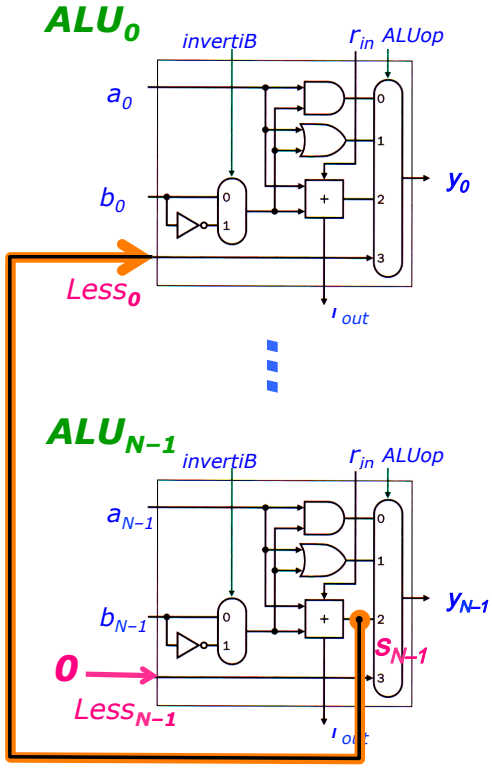
$Less_1$ … $Less_{N-1}$ ← 0
$Less_0$ ← s_{N-1}



Struttura ALU con comparatore

if (a < b) **and** (S = comparazione) **then**
ALUop = 11
invertiB = 1
Less(i) ← 0 (i = 1, 2, 3, …. N–1)
Less(0) ← s_{N-1}





Overflow

- Esempio decimale:
  - **a + b = c** dove **a,b,c tutti** codificati con 2 cifre decimali
  - **a = 19, b = 83**
  - **Overflow: 19 + 83 = (1)02**
- Nella rappresentazione binaria in complemento a 2, il MSB è dedicato al segno: 0: '+', 1: '–'

Esempio: **0**110 + **0**011 = **1**001
in complemento a 2: +6 +3

- L'overflow modifica il **MSB** (in compl. a 2, dedicato al **segno**)
- **Overflow nella somma** quando:

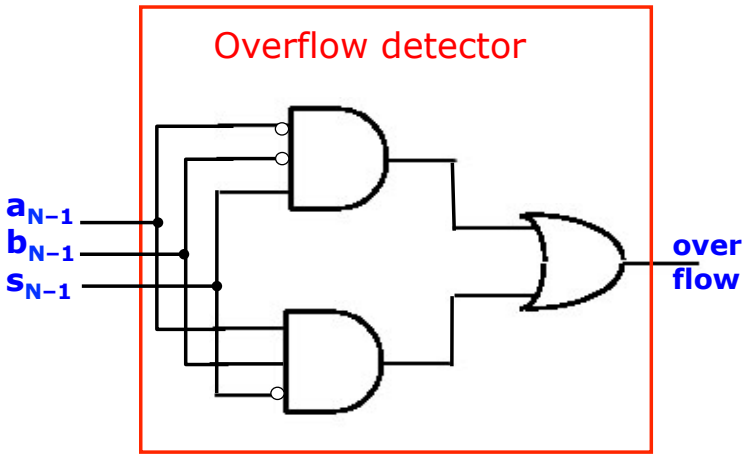
a + b = s, a > 0, b > 0 → **MSB di a e b = 0, MSB di s = 1**
a + b = s, a < 0, b < 0 → **MSB di a e b = 1, MSB di s = 0**

- Si può avere overflow con **a** e **b** di segno opposto ?

Circuito di riconoscimento dell'overflow

- 3 ingressi, tutti dalla ALU_{N-1}:
  - MSB di a, b e somma: a_{N-1} b_{N-1} s_{N-1}

a_{N-1}	b_{N-1}	s_{N-1}	**overflow**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Overflow

Overflow nella differenza:

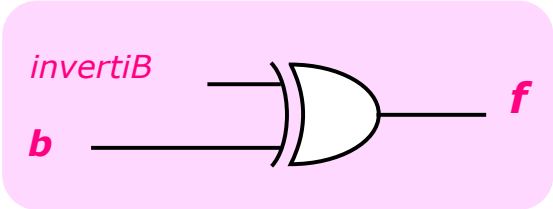
$$\underline{\mathbf{0}}110 - (-\underline{\mathbf{0}}011) = \underline{\mathbf{1}}001$$

- **Overflow nella differenza** quando:

a + (–b) = s,	a > 0, (–b) > 0 →	**MSB di a e (–b) = 0, MSB di s = 1**
a + (–b) = s,	a < 0, (–b) < 0 →	**MSB di a e (–b) = 1, MSB di s = 0**

\+ : **MSB** di **b**
– : **MSB** di **(–b)** = **MSB** di **NOT(b)**

➔ +/– : **MSB** di **f**

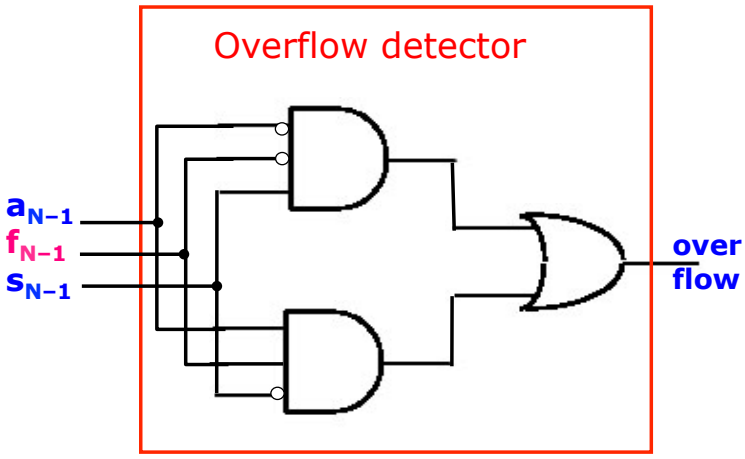


- Utilizzando **f**, in uscita dallo XOR, anziché **b**, il rivelatore di overflow funziona sia per la somma che per la differenza!

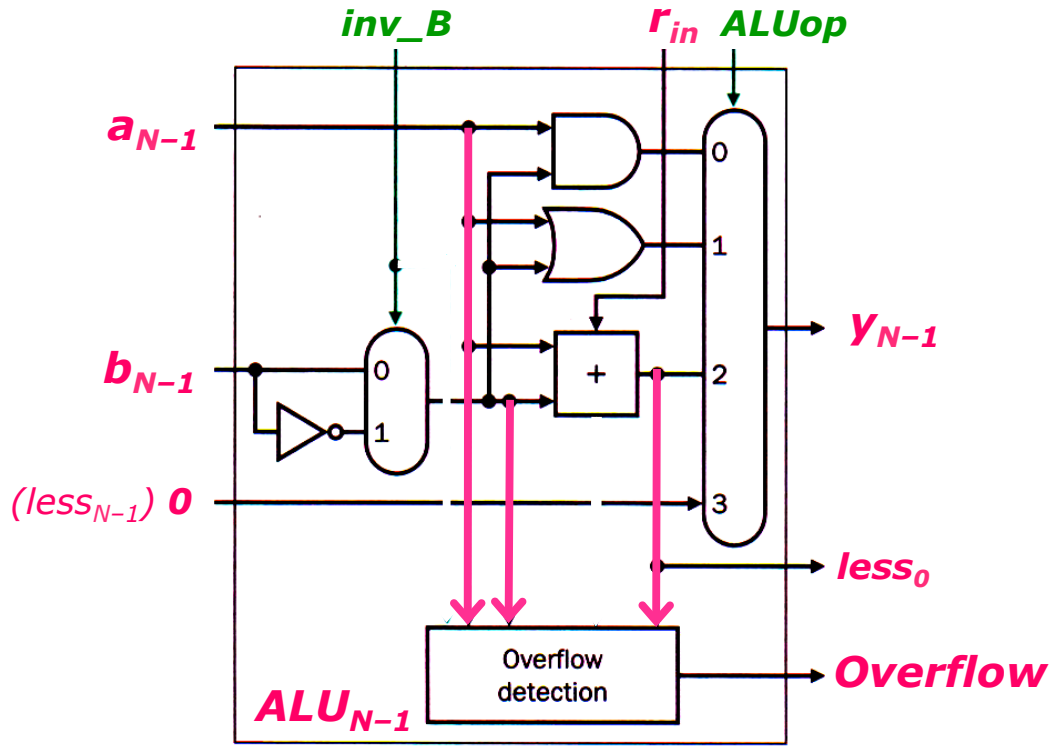
Circuito di riconoscimento dell'overflow

- 3 ingressi, tutti dalla ALU31:
  - MSB di a, b e somma: a_{31} b_{31} s_{31}

a_{N-1}	f_{N-1}	s_{N-1}	overflow
0	0	0	0
0	**0**	**1**	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	**1**	**0**	**1**
1	1	1	0



Struttura ALU_{N-1} con Overflow detector



ALU completa a 32 bit

ALU a 32 bit completa:

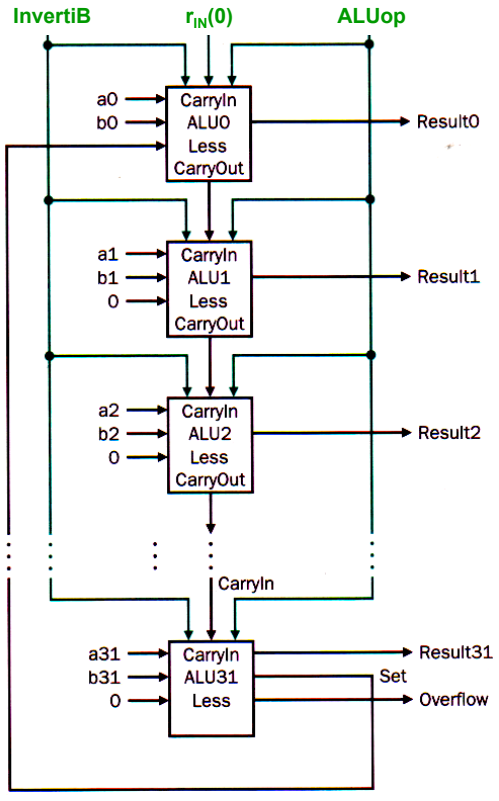
- *InvertiB e $r_{IN}(0)$ sono lo stesso segnale*

Test di uguaglianza:

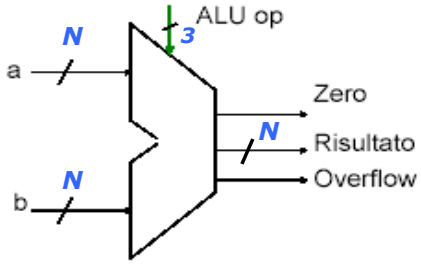
- *Operazioni necessarie*
  - Impostare una differenza.
  - Effettuare l' OR di tutti i bit somma.
  - Uscita dell' OR = 0 → i due numeri sono uguali

Operazioni possibili:

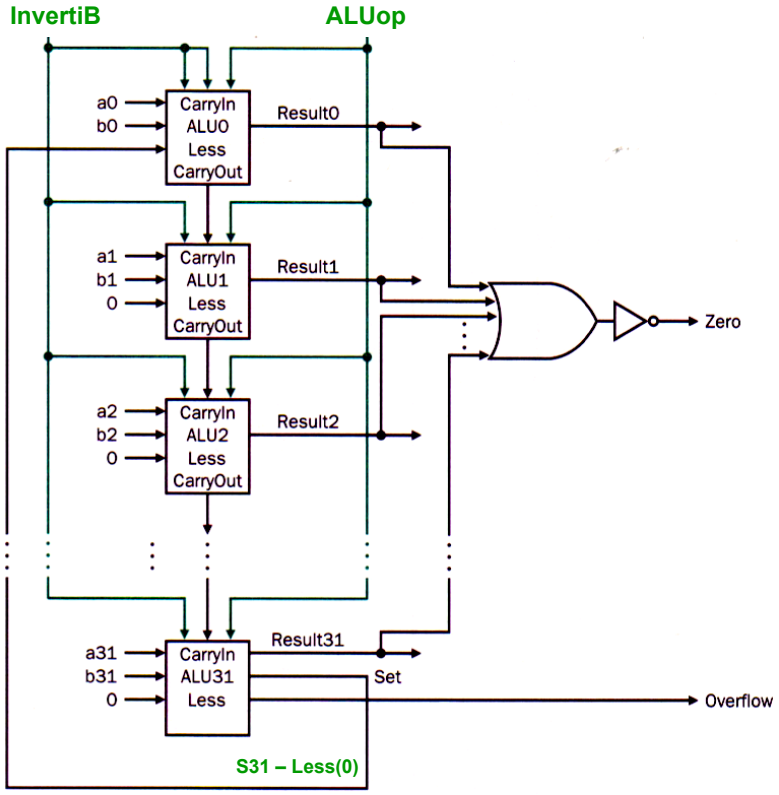
- AND
- OR
- Somma / Sottrazione (/ Uguaglianza)
- Comparazione



ALU a 32 bit: struttura finale



ALUop	funzione
000	and
001	or
010	+ (add)
110	– (sub)
111	set less than



Semplificando le espressioni

$$s = \bar{a}b\bar{r}_{in} + a\bar{b}\bar{r}_{in} + \bar{a}\bar{b}r_{in} + abr_{in} =$$

$$= (a \oplus b)\bar{r}_{in} + (ab + \bar{a}\bar{b})r_{in} =$$

$$= \boxed{(a \oplus b)\bar{r}_{in} + \overline{(a \oplus b)}r_{in} =}$$

$$= \boxed{(a \oplus b) \oplus r_{in}}$$

$$r_{out} = ab\bar{r}_{in} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in} =$$

$$= ab(r_{in} + \bar{r}_{in}) + (a\bar{b} + \bar{a}b)r_{in} =$$

$$= \boxed{ab + (a \oplus b)r_{in}} = \cdots = \boxed{ab + (a + b)r_{in}}$$

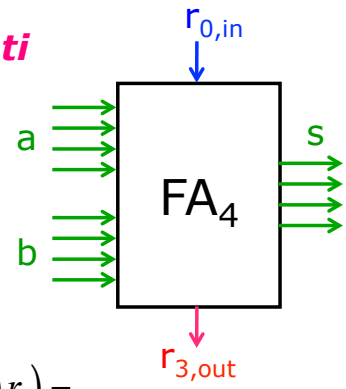
Dimostrate l'equivalenza

Sommatori ottimizzati

*I sommatori a **propagazione di riporto** sono **lenti***

Cammino critico (tutti FA): **C = 3N**

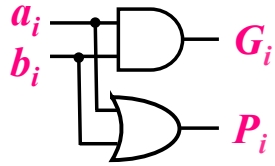
Responsabile della lentezza: il calcolo del riporto



Modulo sommatore a 4 bit:

Calcolo diretto del riporto in uscita: $r_{3,OUT}$

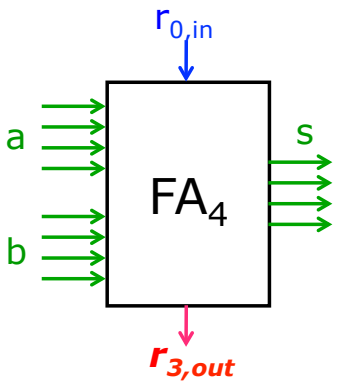
$$r_{3,OUT} = a_3b_3 + (a_3+b_3)r_2 = a_3b_3 + (a_3+b_3)(a_2b_2 + (a_2+b_2)r_1) =$$
$$= a_3b_3 + (a_3+b_3)(a_2b_2 + (a_2+b_2)(a_1b_1 + (a_1+b_1)r_0)) =$$
$$= a_3b_3 + (a_3+b_3)(a_2b_2 + (a_2+b_2)(a_1b_1 + (a_1+b_1)(a_0b_0 + (a_0+b_0)r_{IN}))) =$$
$$(G_i = a_ib_i\ ,\ P_i = a_i + b_i),$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 \cdot r_{0,IN}$$

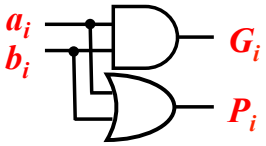


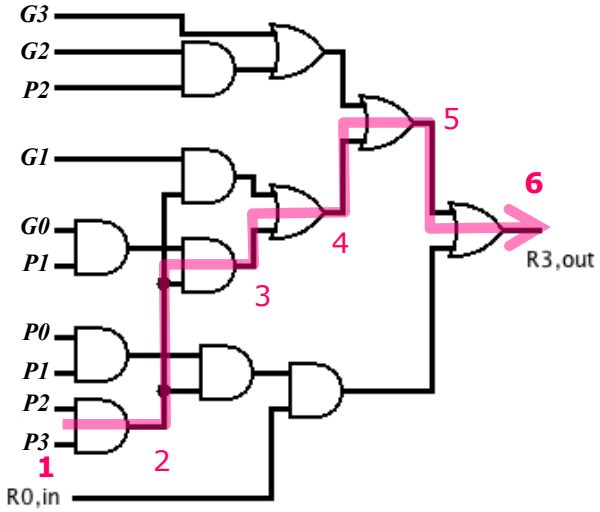
Addizionatori ottimizzati

Considero l'uscita più critica: $r_{3,out}$

$$r_{3,OUT} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 \cdot r_{0,IN}$$







Cammino critico (con porte a 2 ingressi) : **C = 6** (N=16 ➔ **C=24**)

A propagazione di riporto: **C ≈ 3N = 12** (N=16 ➔ **C=48**)

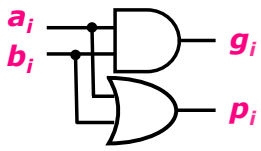
Struttura sommatore a blocchi

Struttura modulare

Come collegare i moduli tra loro?

Termine di generazione: $g_i = a_i b_i$

Termine di propagazione: $p_i = a_i + b_i$

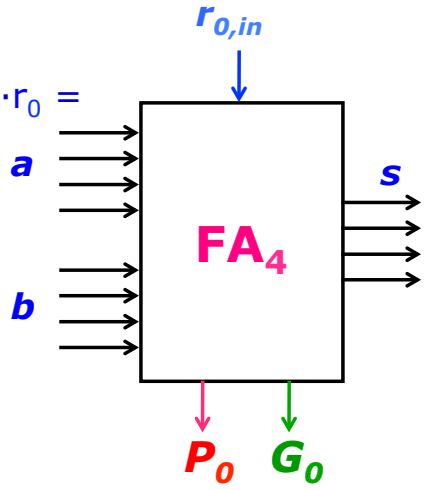


$r_3 = g_3 + p_3 r_2 =$

$= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) =$

$= (g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0) + p_3 p_2 p_1 p_0 \cdot r_0 =$

$= \mathbf{G_0 + P_0 \cdot r_0}$



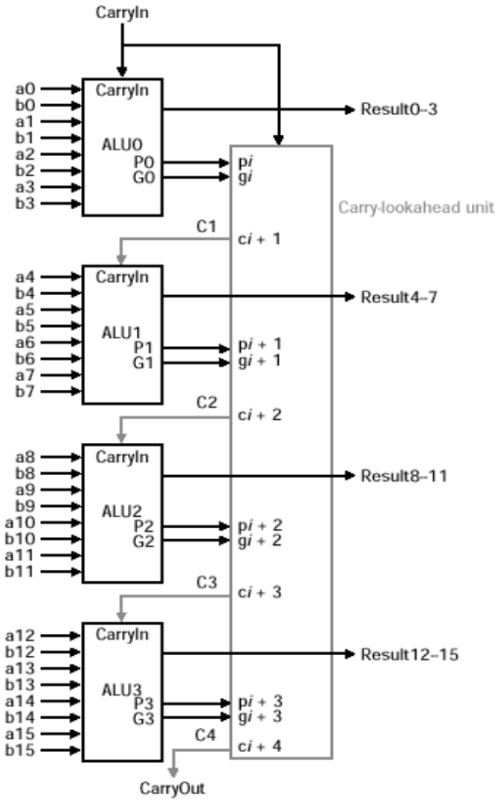
dove:

$P_0 = p_3 p_2 p_1 p_0$

$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$CC = 6$

Sommatore ad **anticipazione di riporto (CLA)**



Anticipazione di riporto *(Carry Look-Ahead):*

$C_1 = G_0 + P_0 \cdot r_0$

$C_2 = G_1 + P_1 \cdot C_1 =$
$= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot r_0$

$C_3 = G_2 + P_2 \cdot C_2 =$
$= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 +$
$+ P_2 \cdot P_1 \cdot P_0 \cdot r_0$

$r_{out} = C_4 = G_3 + P_3 \cdot C_3 =$
$= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 +$
$+ P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot r_0$

Cammino critico: 6

Cammino critico: C = 6+6 = 12

Propagazione riporto: C = 3N = **48**