Lezione 6

Circuiti digitali notevoli: ALU

F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

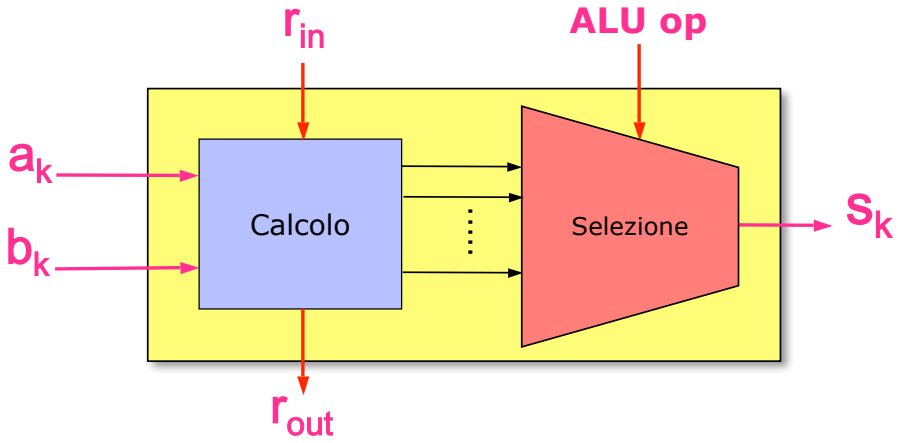
ALU: Arithmetic-Logic Unit

- Esegue le operazioni aritmetico-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
  - PowerPC G5, Athlon64: 64 bit
- Struttura modulare
  - Blocchi funzionali da 1 bit, replicati N volte
  - Blocchi da 4 bit

Struttura a 2 livelli di una ALU

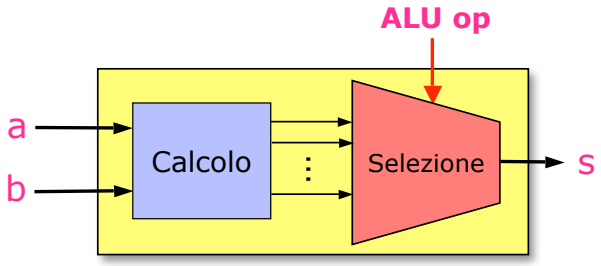
Struttura ALU per il bit k-esimo:

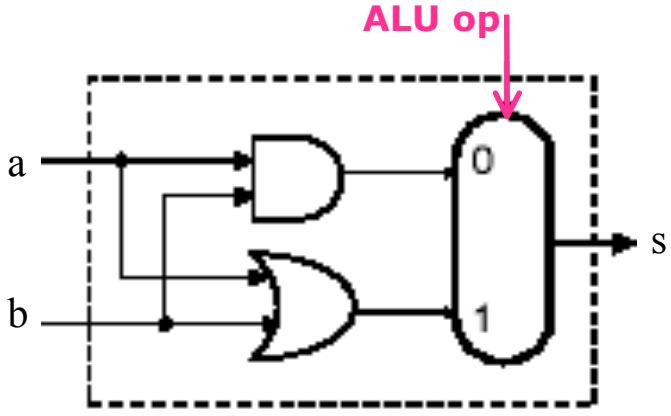
- Ingressi: Operandi: a_k, b_k
  Riporto in ingresso: r_{in}
  Selettore operazione: ALUop
- Uscite: Risultato: s_k
  Riporto in uscita: r_{out}



Progettazione della ALU

- Porta AND / OR
  - Selezionabile
- Componenti:
  - 1 porta AND
  - 1 porta OR
  - 1 Multiplexer (MUX)

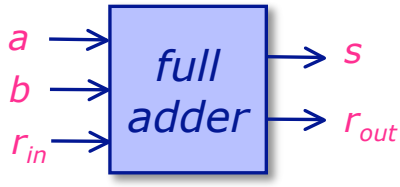




ALUop = 0 → s = AND(a,b)
ALUop = 1 → s = OR(a,b)

FULL Adder (1 bit)

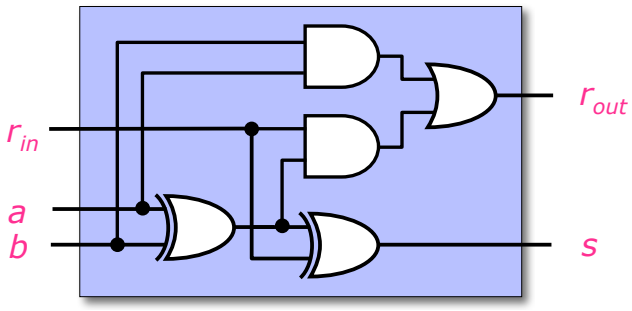
- Gestisce anche il riporto in ingresso
  - 3 ingressi: **a, b, r_{IN}**
  - 2 uscite: **s, r_{OUT}**



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

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

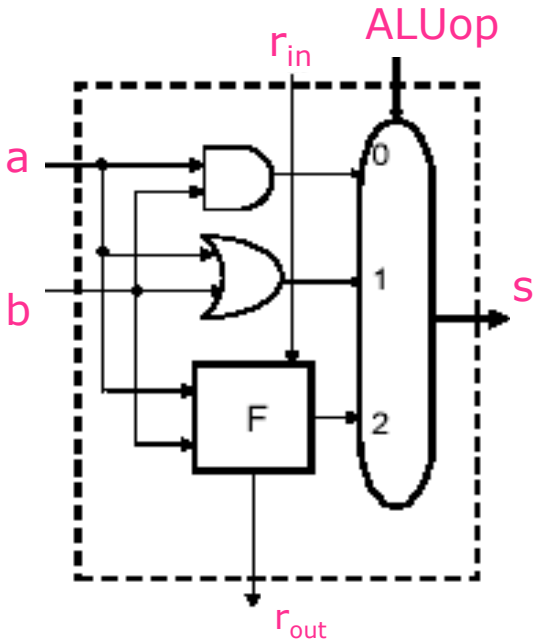
$$r_{out} = ab\bar{r}_{in} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in} = ab + (a \oplus b)r_{in}$$



ALU 1 bit

Operazioni:

- OR, AND, somma
- ALUop: 2 bit
  - 00: s = a and b
  - 01: s = a or b
  - 10: s = a + b + r_{in}

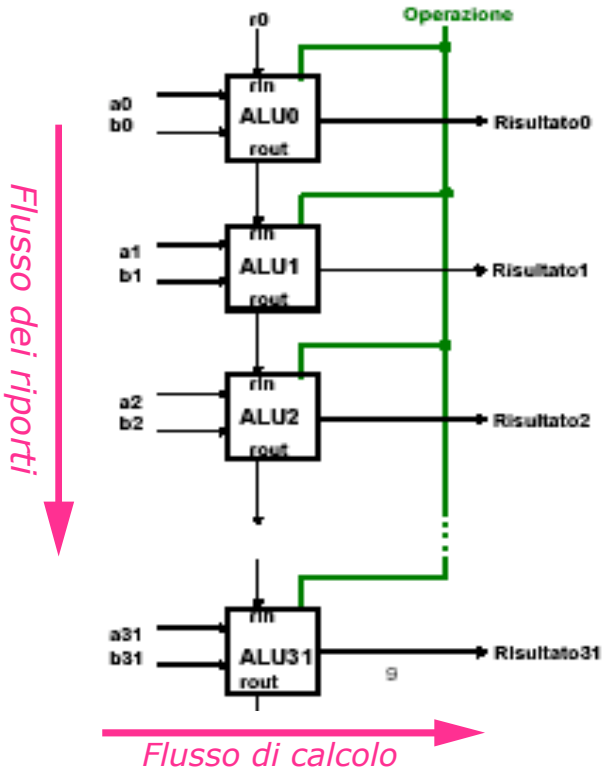


Sommario

- ALU su 1 bit:
  operazioni logiche e somma
- ALU su 32 bit:
  implementazione di sottrazione, confronto e test di uguaglianza

ALU a 32 bit

- Come collegare N ALU a 1 bit per ottenere ALU a N bit?
- ALU a 32 bit:
- ALU in parallelo, ma...
  - Propagazione dei riporti
  - Limite alla velocità di calcolo



Sottrazione

- **Sottrazione** → addizione dell'opposto:

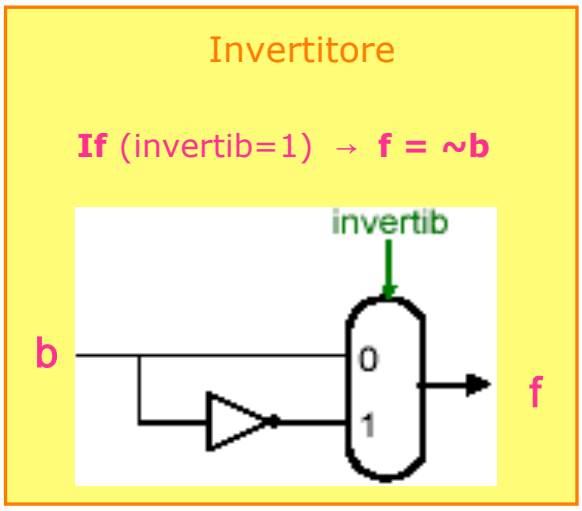
  $a - b = a + (- b)$

  - 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: circuito di inversione
  - Aggiunta della costante "1": pongo $r_{in}(0) = 1$



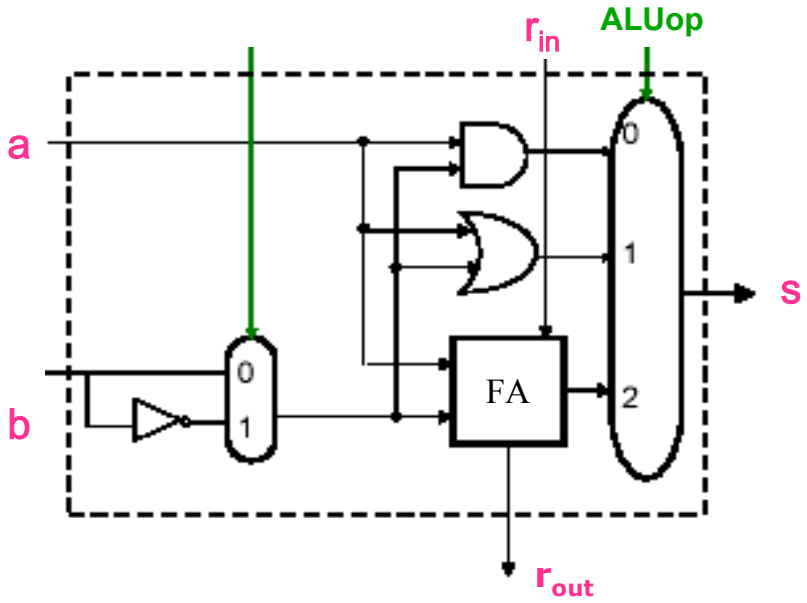
ALU - bit i-esimo

Operazioni: **AND , OR , + , –**

Propagazione riporti: $r_{in}(i) = r_{out}(i-1)$ i = 1, 2, 3, ...31

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

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



Comparazione (confronto)

- **Comparazione:**

  if **a** < **b** then **s** = 1 (s = 0...01)

  - Fondamentale per dirigere il flusso di esecuzione (test, cicli...)

    if (a<b) → s = [0 0 0 ... 0 **1**]

    else → s = [0 0 0 ... 0 **0**]

- Implementazione:
  - if ALUop = "comparazione"

    then s(i) = 0, i = 1, 2, 3, ... 31

    if (a < b) s(0) = 1

    else s(0) = 0

  Devo:
  - Imporre tutti i bit di **s** (tranne s_0) a 0;
  - Calcolare s_0 in base alla condizione a<b

Come sviluppare la comparazione?

- IDEA: in complemento a 2, il MSB della somma (bit di segno) = 1 per numeri negativi → $s_{MSB} = 1$

$$a < b \rightarrow a - b < 0 \rightarrow s_{MSB} = 1$$

Implementazione:

- Nuovo ingresso: **LESS**

  IF: ALUop = "comparazione" → $s_i = LESS_i$

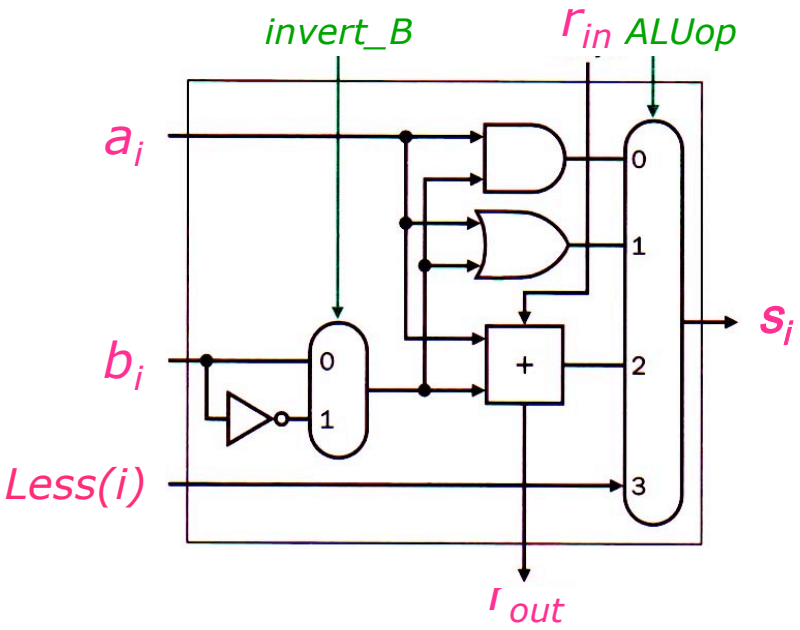
- Operazioni:
  - Calcolare la differenza (a – b) (senza mandarla in uscita)
  - Inviare l'uscita del sommatore del **MSB** a **LESS di ALU_0**

    $s_{31} \rightarrow LESS_0$
  - Questa uscita viene chiamata **segnale di set**

ALUi con comparatore

Less(i) ← 0 **i = 1,2,3, ….31**

Less(0) ← s_{31} **iff (a < b) & (S = comparazione)**



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**
- Supponendo il MSB dedicato al bit di segno…
  - 019 + 083 = 102
  - L'overflow modifica il **MSB** (in compl. a 2, dedicato al **segno**)
- **Overflow nella somma** quando:

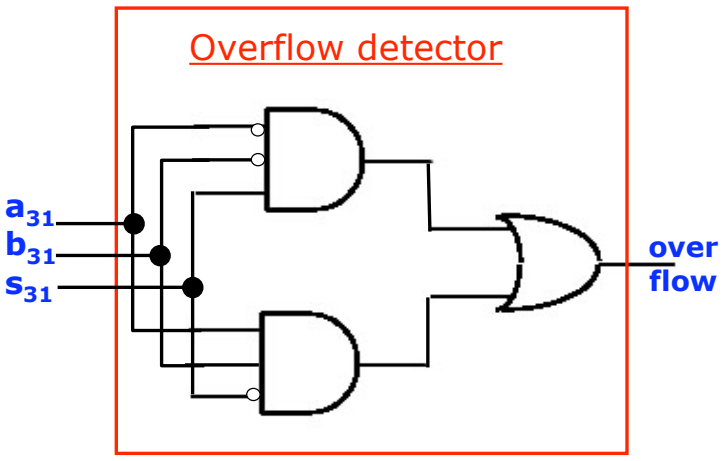
  a + b = s, a > 0, b > 0 → **MSB di a e b = 0, MSB di s = 1**

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

a_{31}	b_{31}	s_{31}	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



ALU_31 con Overflow detector

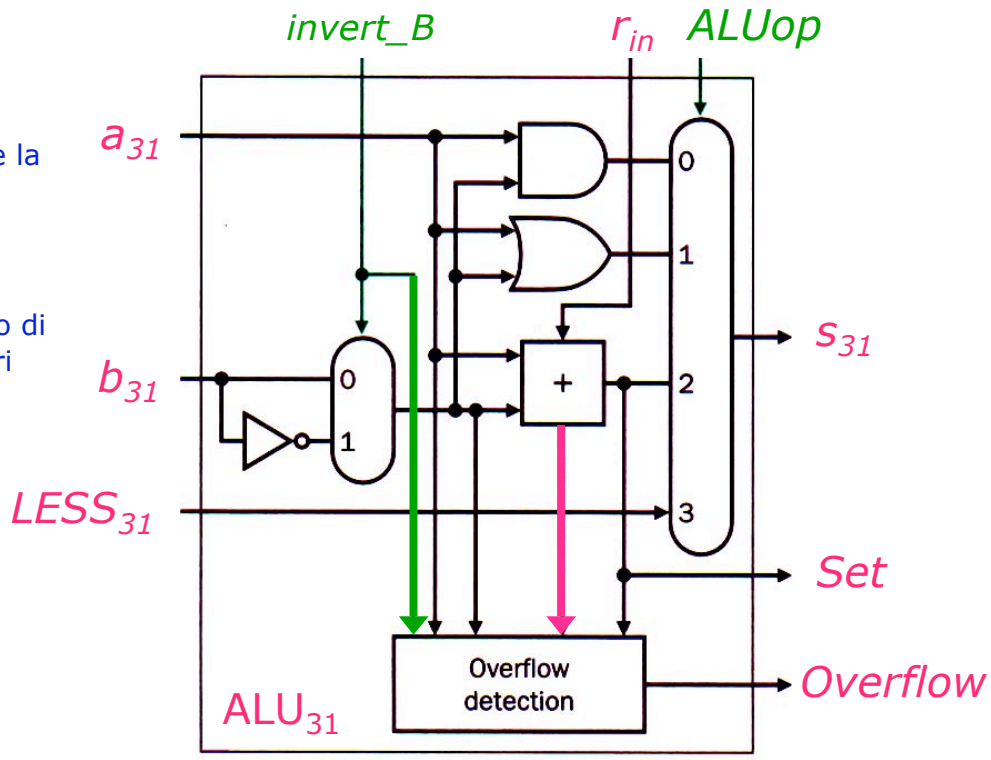
Altri ingressi:

invert_B:

per gestire anche la differenza tra numeri discordi

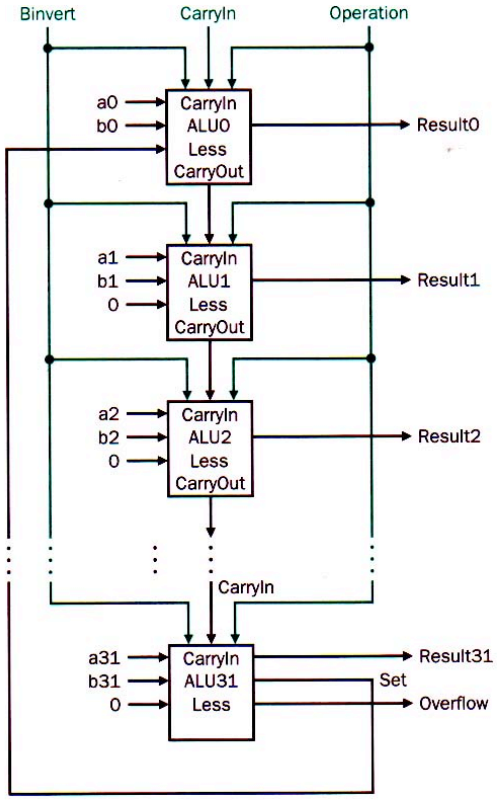
$r_{out}(31)$:

per gestire il caso di overflow tra interi "unsigned"



ALU completa a 32 bit

- InvertiB e $r_{IN}(0)$ sono lo stesso segnale
- Si può ancora ottimizzare



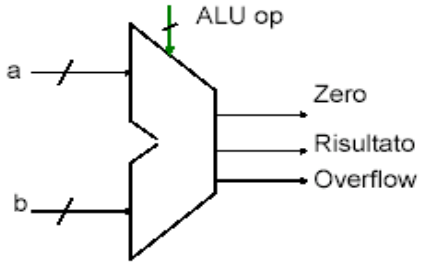
Test di uguaglianza

- *Esempio: istruzione Assembly:*

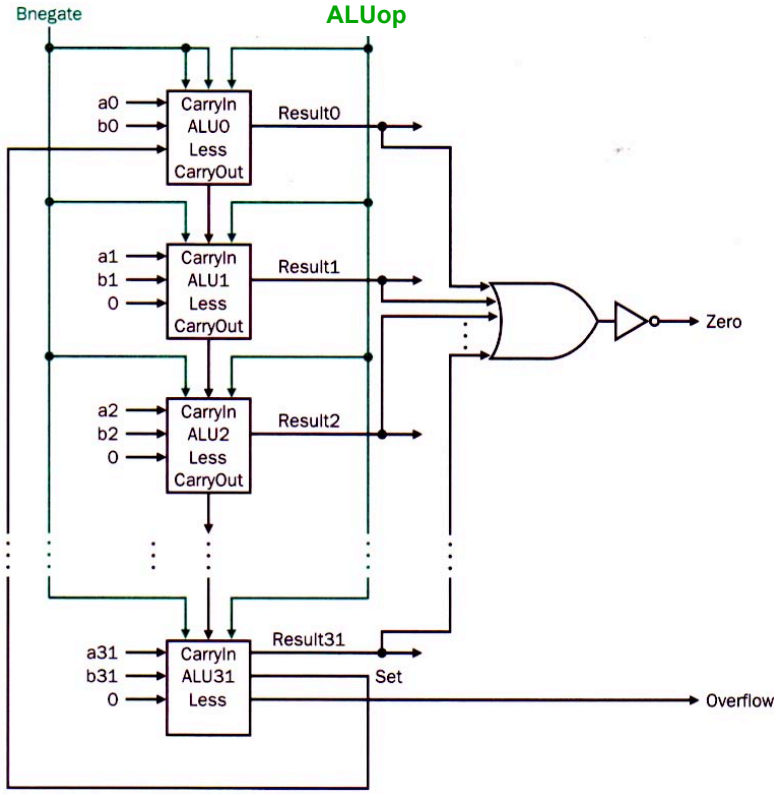
```
beq rs, rt label      # if (rs – rt) = 0 , salta
```

- *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
  - Comparazione
  - Test di uguaglianza

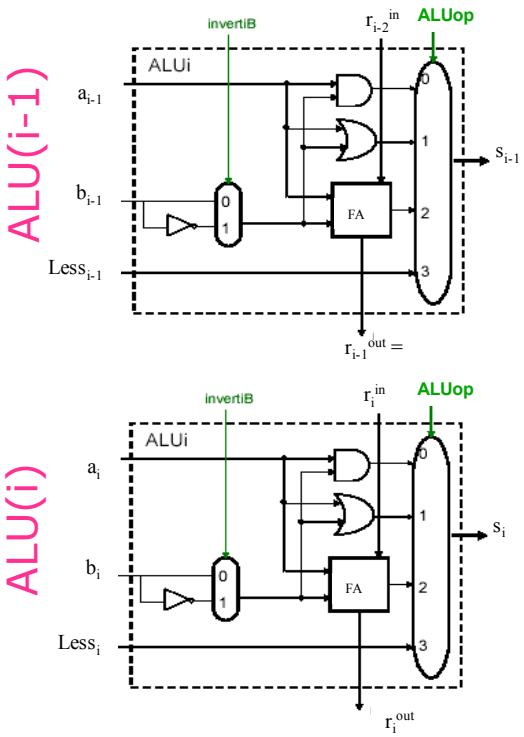
ALU a 32 bit: struttura finale

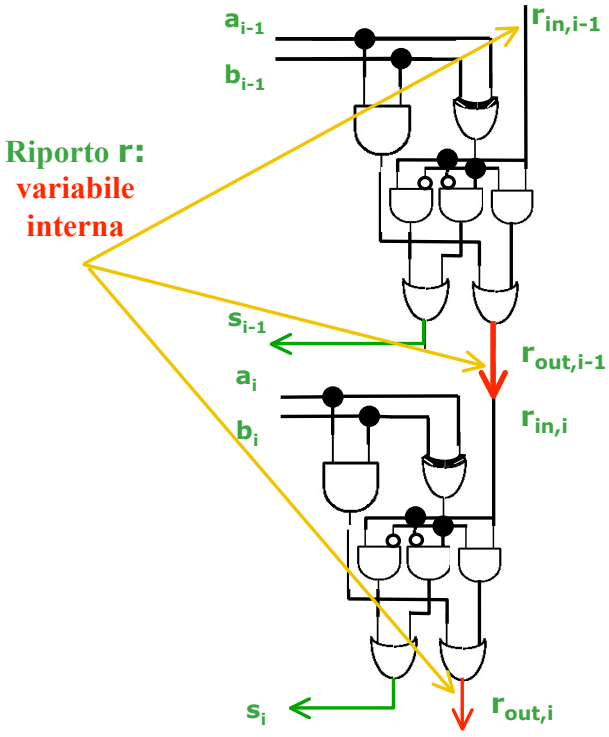


ALUop	funzione
000	and
001	or
010	+ (add)
110	– (sub)
111	set less than



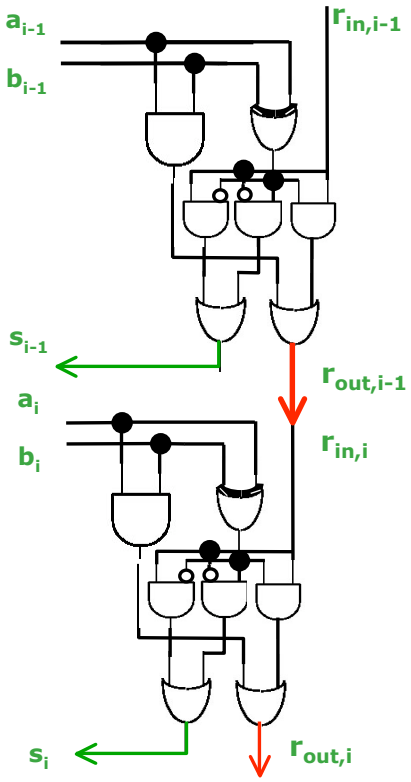
Propagazione del riporto





Cammini critici

- Per ogni stadio:
  - Somma: $C_S = 3$
  - Riporto: $C_R = 3$
- Per **N** stadi:
  - Somma: $C_S = 3$
  - Riporto: $C_R = 3 \cdot N$
    - *N = 4 bit → C_R=12*



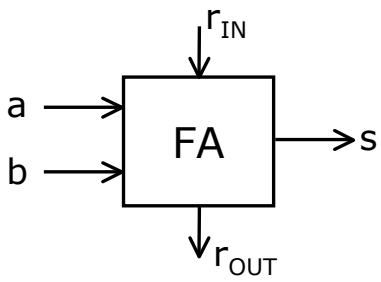
I problemi del full-adder

- Full Adder con propagazione di riporto è lento
  - Il riporto si propaga sequenzialmente
    - caratteristica dell'algoritmo di calcolo
  - La commutazione dei circuiti non è istantanea
    - caratteristica fisica dei dispositivi
- Soluzioni
  - modificare i dispositivi
  - modificare l'algoritmo
- Sommatori ad anticipazione di riporto

Anticipazione di riporto

- **Anticipazione di riporto** (*carry look-ahead*)
  - Approccio per diminuire la latenza della somma
  - Propagazione di riporto: $t_L = 3N$

- Principio di funzionamento:
  - Si genera un riporto in uscita quando ho almeno due "1" sui tre ingressi (r_{in}, a, b)



1 1 0 0 0 ← riporto
1 1 0 1 +
1 0 0 =
---
1 0 0 0 1

Propagazione e generazione

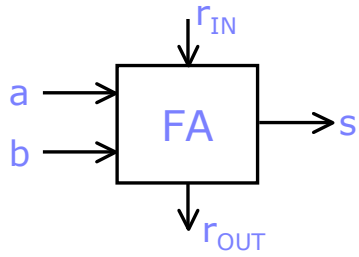
- Ho riporto quando ho almeno 2 dei 3 ingressi (r_{in}, a, b) = **"1"**

- Due casi possibili:
  - **GENERAZIONE:** g_i
    Viene generato un riporto allo stadio ***i***,
    per qualsiasi r_{in}, se:

    $g_i = a_i b_i$; $g_i = 1 \rightarrow r_{i,out} = 1$

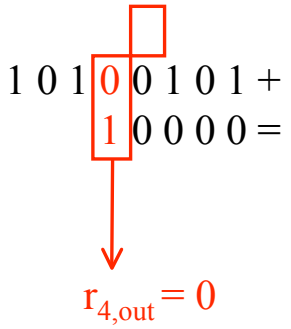
  - **PROPAGAZIONE:** p_i
    Viene generato un riporto allo stadio ***i***,
    se $r_{in} = 1$ e (**a** OR **b**) = 1

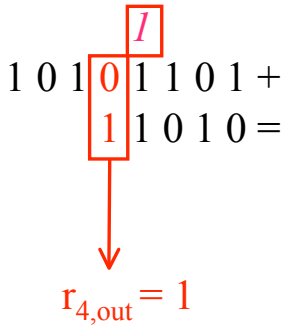
    $p_i = (a_i + b_i)$; $p_i\, r_{i,in} = 1 \rightarrow r_{i,out} = 1$



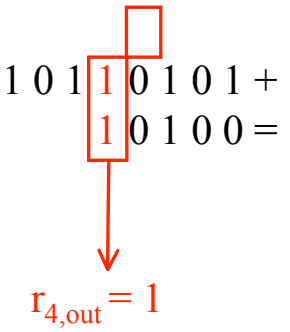
Esempio

- Calcolo: $r_{4,out}$
  - supponiamo $r_{0,in} = 0$





Propagazione:
$p_4 r_{3,out} = (a_4+b_4) r_{3,out} = 1$



Generazione:
$g_4 = a_4 b_4 = 1$

- Quindi: $r_{4,out} = (a_4+b_4) r_{3,out} + a_4 b_4$

Sviluppo della funzione logica riporto

Dato che:

$r_{i,out} = a_i b_i + (a_i + b_i)\, r_{i,in} = g_i + p_i\, r_{i,in}$ $\qquad r_{i-1} = r_{i-1,out} = r_{i,in}$

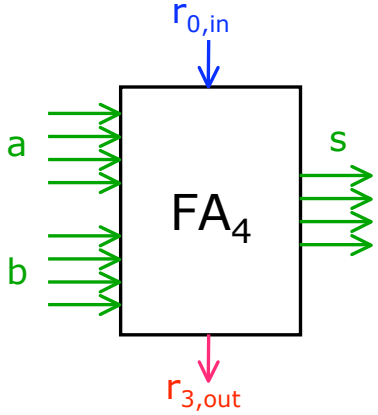
- Ricavo $r_{3,out}$ come funzione degli ingressi: a_i , b_i , $r_{in,0}$:

$$r_{0,out} = g_0 + p_0 \cdot r_{0,in}$$

$$r_{1,out} = g_1 + p_1 r_{0,out} = g_1 + p_1 g_0 + p_1 p_0\, r_{0,in}$$

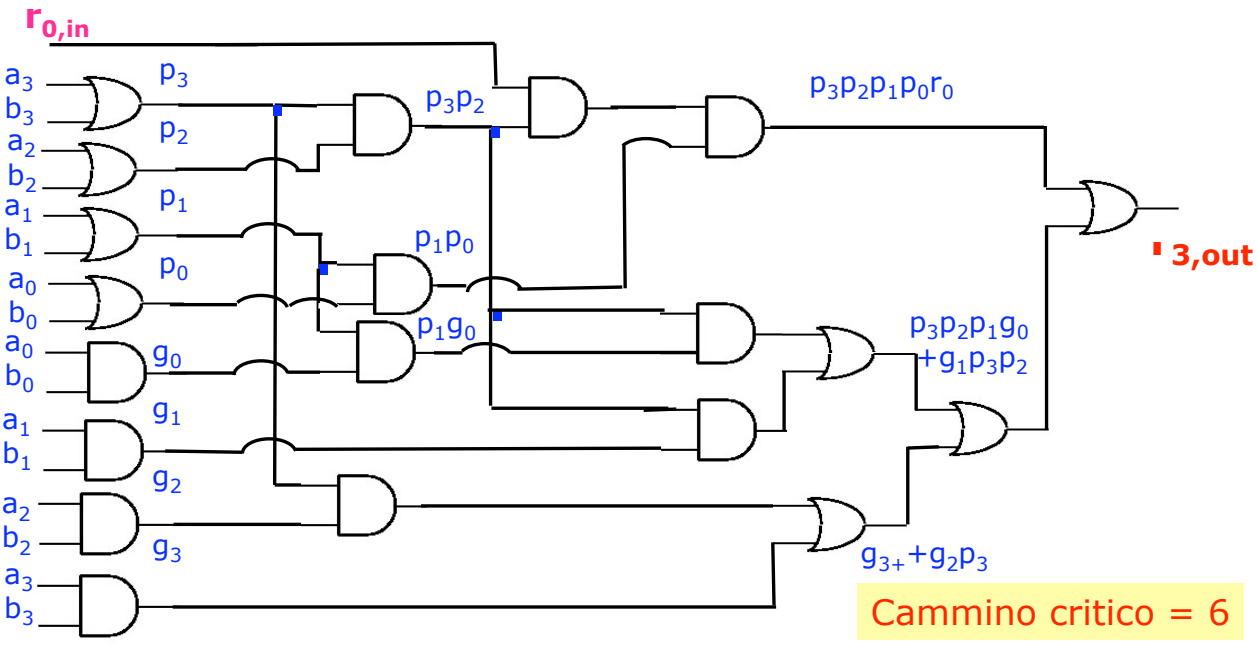
$$r_{2,out} = g_2 + p_2 r_{1,out} = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0\, r_{0,in})$$
$$= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0\, r_{0,in}$$

$$r_{3,out} = g_3 + p_3 r_{2,out} =$$
$$= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0\, r_{0,in}) =$$
$$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 \cdot r_{0,in}$$



Anticipazione di riporto (4 bit)

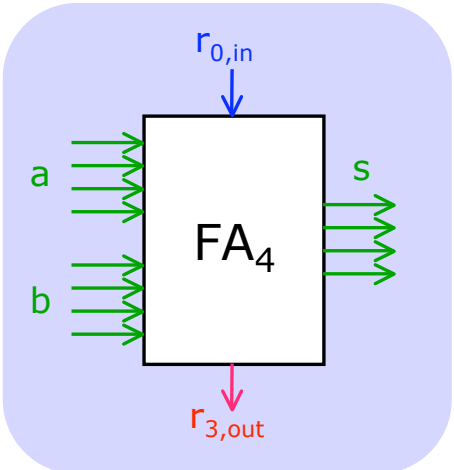
$r_{3,out} = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_{0,in}) =$
$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 \cdot r_{0,in}$



Addizionatori modulari

- Moduli elementari, collegabili in cascata.
  - Complessità del circuito tollerata per piccoli ***n***
    (es. *n*=4)

- Cammino critico C:
  - M moduli da 4 bit: **C = 6·M**
    N = 16 bit → M = N/4
    → C = 6·N/4 = **24**
  - A propagazione di riporto:
    N = 16 bit
    → C = 3·N = **48**



Struttura sommatore a blocchi

- Vogliamo 32 bit → 8 sommatori elementari
  - Come collegarli tra loro?

$r_3 = g_3 + p_3 r_2 =$

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

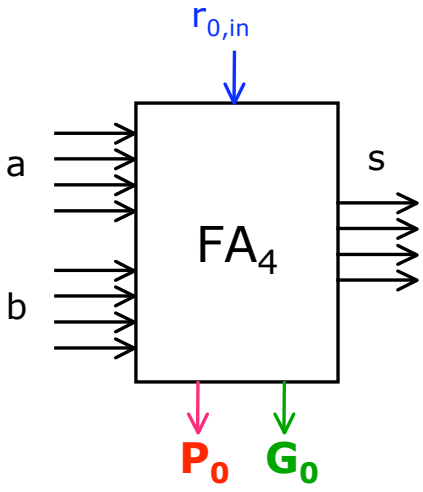
$= (g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0) + p_3 p_2 p_1 p_0 \cdot r_0 =$

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

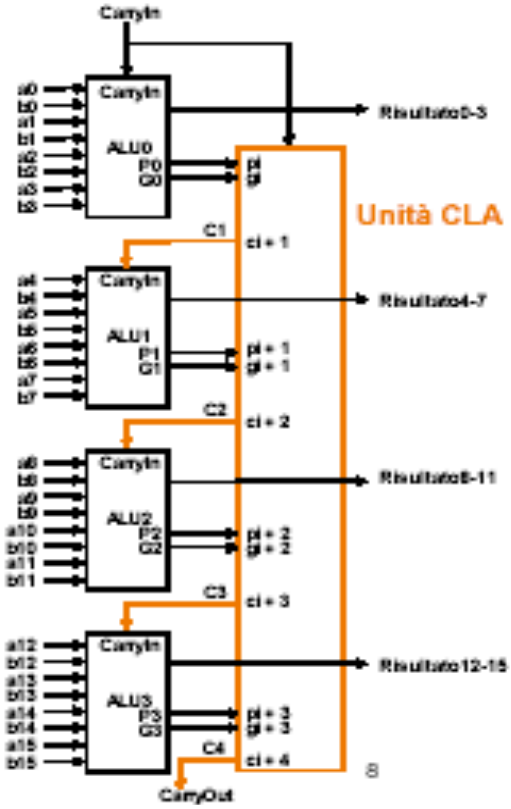
dove:

$\mathbf{P_0} = p_3 p_2 p_1 p_0$

$\mathbf{G_0} = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$



Struttura di un sommatore su 16 bit



$C_1 = G_0 + P_0 \cdot r_0$

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

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

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

Cammino critico = 6+6 = 12
- CLA + prop: 6M = 24
- Prop: 3N = 48