Lezione 6

Semplificazione: mappe di Karnaugh
Circuiti digitali notevoli: ALU

Proff. A. Borghese, F. Pedersini

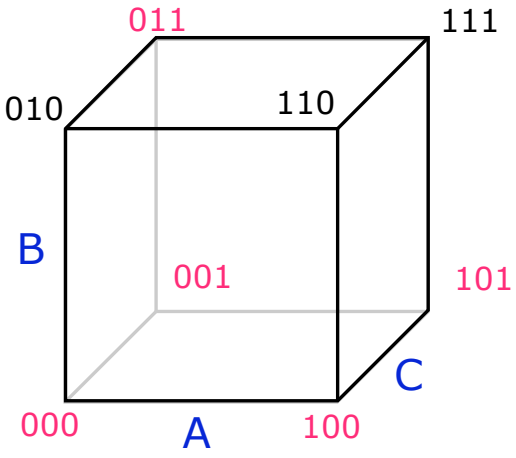
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Semplificazione: mappe di Karnaugh

- Tecnica di semplificazione, a partire dalla tabella di verità
  - Rappresentazione cubica di funzioni logiche a 3 variabili: F = *f*(a,b,c)
  - Muovendosi sui lati, la configurazione di variabili cambia di un solo bit
  - Distanza di HAMMING: ***d(v1, v2)*** = n. di bit diversi tra le sequenze

$$F = A \cdot B + B \cdot \overline{C}$$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

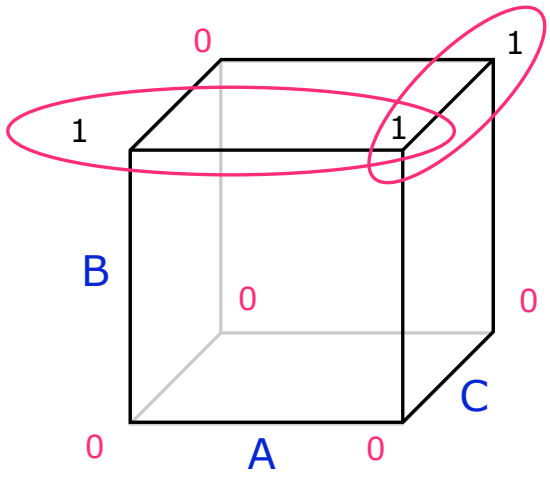


Semplificazione: mappe di Karnaugh

- **Copertura**: ricerca di tutti gli implicanti
- Se i vertici di un lato sono entrambi 1, l'implicante è indipendente dalla variabile corrispondente al lato

$$F = A \cdot B + B \cdot \overline{C}$$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



- Per **N>3** variabili, la rappresentazione diviene complessa...

Semplificazione: mappe di Karnaugh

- Rappresentazione piana della funzione:
  - "srotolo" il cubo
  - Codifica di Gray (codice riflesso) lungo ogni direzione

C \ AB	00	01	11	10
0	000	010	110	100
1	001	011	111	101

indipendente da a: **b~c**

indipendente da c: **ab**

C \ AB	00	01	11	10
0	0	1	1	0
1	0	0	1	0

F = ab + b~c

- Rappresentazione piana, utilizzabile per: **2 ≤ N ≤ 4**

B \ A	0	1
0	1	0
1	1	0

F = ~a

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

F = ab + cd + b~c~d

- Mappa di Karnaugh: rappresentazione piana e ciclica

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	0	1	0
11	1	0	1	1
10	0	0	1	0

F = ab + b~c~d + ~bcd

Esercizi

Data: F = AC + BC + ~A~BC + AB~C

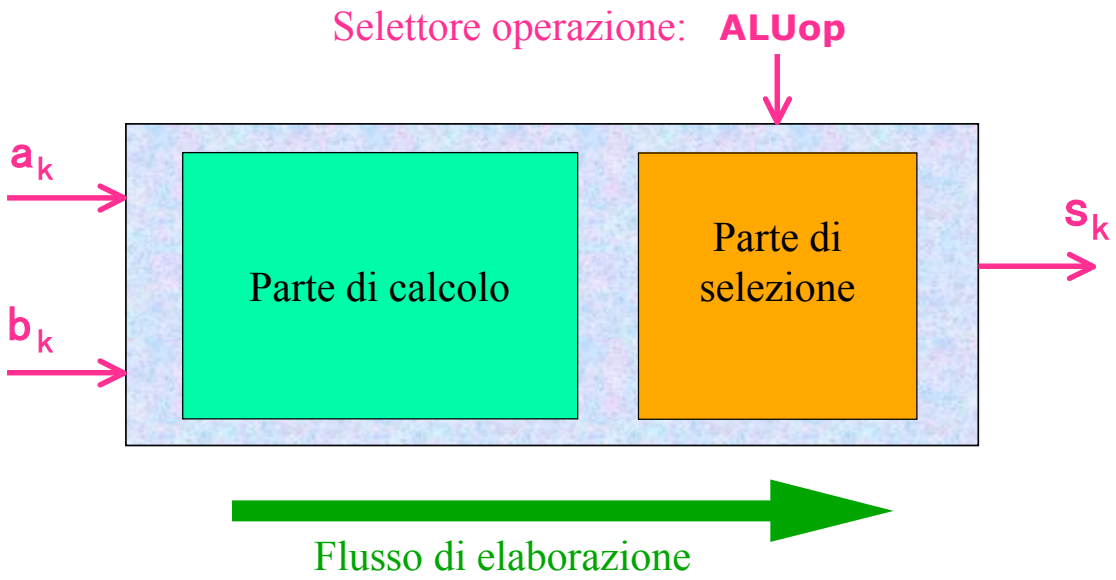
1. *Scrivere la tabella di verità di F*
2. *Determinare la I e la II forma canonica di F*
3. *Semplificare F mediante mappa di Karnaugh*
4. *Ottenere la stessa semplificazione mediante operazioni algebriche*

- *Calcolare la funzione F = f(A,B,C,D) tale che:*
  F=1 sse: il n. di ingressi='1' è 1 oppure 4
  1. *Calcolare la TT*
  2. *Det. la forma canonica più conveniente*
  3. *Semplificare la rappresentazione di F mediante mappe di Karnaugh*

ALU: Arithmetic-Logic Unit

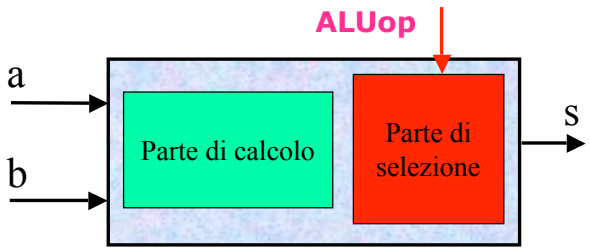
- Esegue le operazioni aritmetico-logiche
  + ; – ; x ; …
  and ; or ; not ; xor ; = ; ≠ ; …
- Normalmente integrata nel processore
  - Inizio anni '90 → introduzione con il nome di co-processore matematico
  - Le ALU non compaiono solamente nei micro-processori
- E' un circuito combinatorio
  - Rappresentabile 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

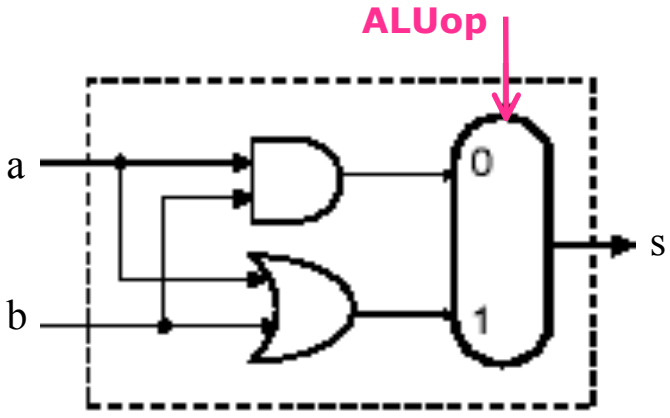
- Struttura ALU – k-esimo bit:



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)

HALF Adder (1 bit)

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

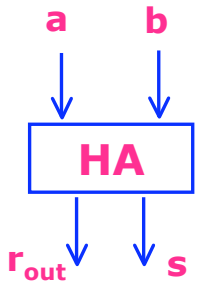
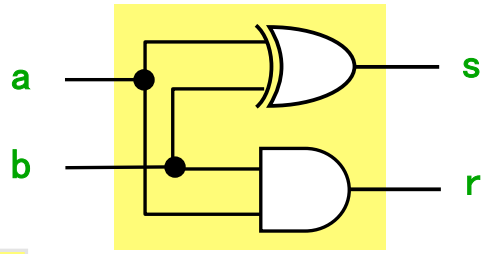


Tabella della verità			
a	b	somma	riporto
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

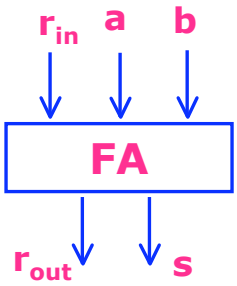


$s = a \oplus b$

$r = a \cdot b$

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

SOP:

$s = m_1 + m_2 + m_4 + m_7$

$r_{out} = m_3 + m_5 + m_6 + m_7$

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

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

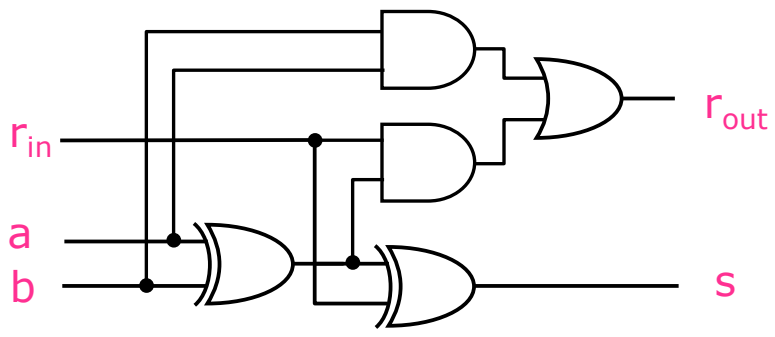
Semplificando le espressioni

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

$$r_{out} = ab\overline{r_{in}} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in} =$$
$$= ab\left(r_{in} + \overline{r_{in}}\right) + \left(a\bar{b} + \bar{a}b\right)r_{in} =$$
$$= ab + (a \oplus b)r_{in}$$

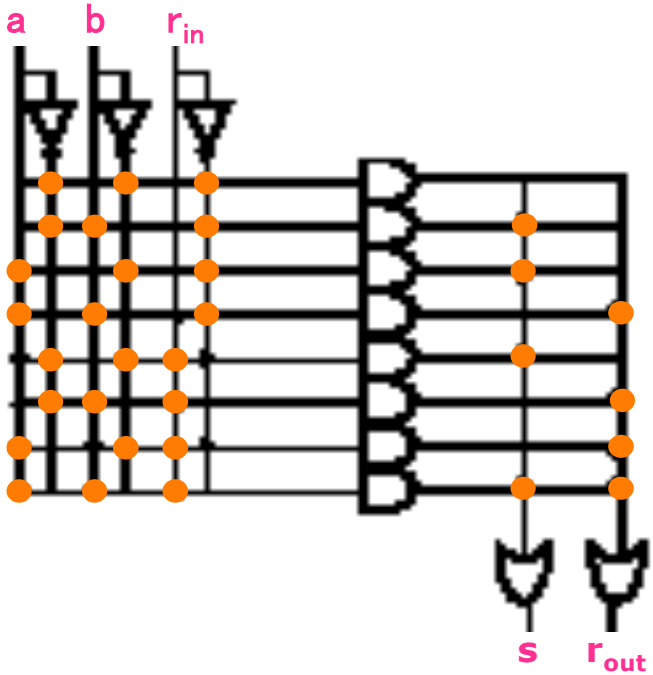
Implementazione circuitale

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

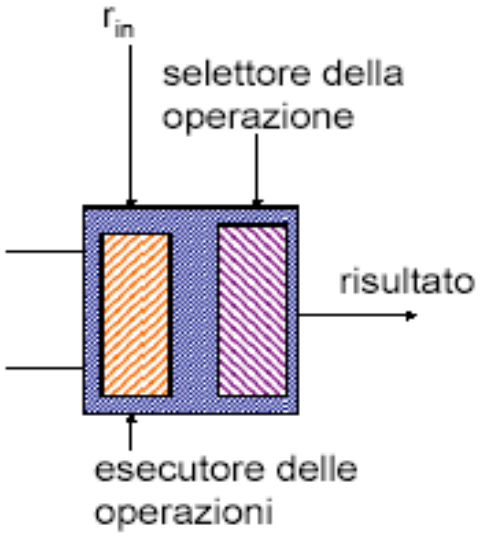


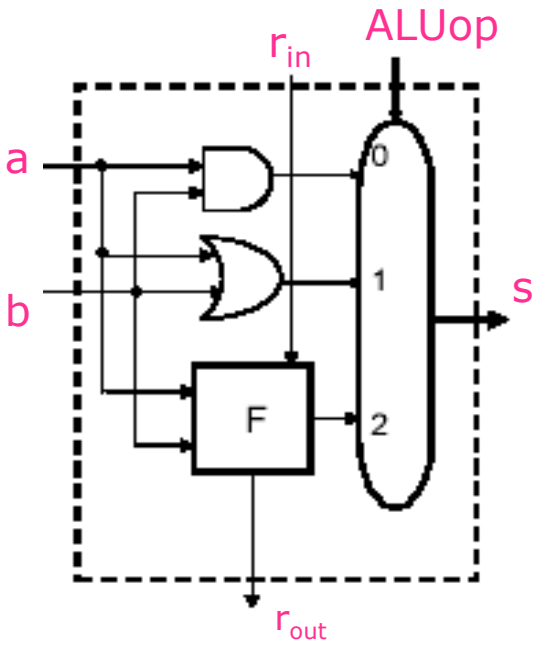
- Approccio **SOP**
  - Costruisco i mintermini
  - Li sommo alle uscite

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



ALU 1bit con OR,AND,Adder



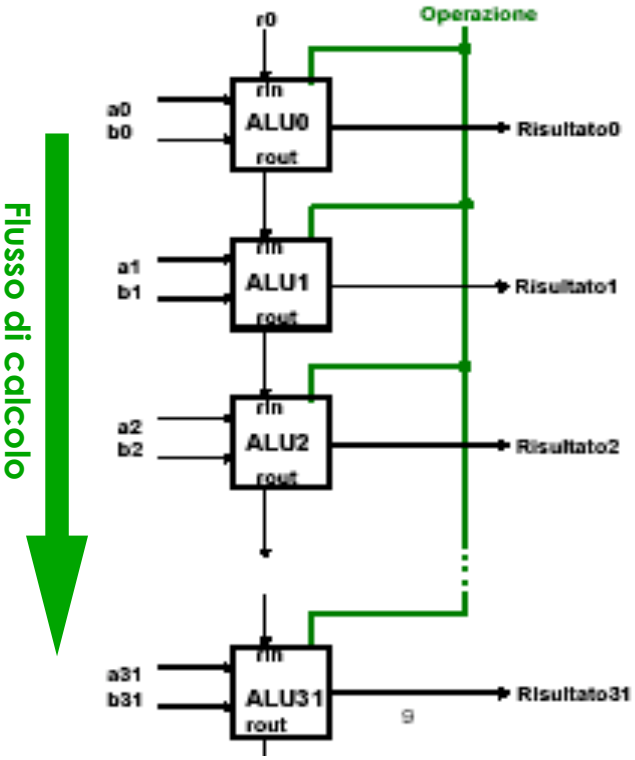


Sommario

- 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 le ALU a 1 bit per ottenere ALU a 32 bit
- ALU in parallelo, ma...
  - Propagazione dei riporti
  - Limite alla velocità di calcolo



Sottrazione

- Sottrazione → addizione dell'opposto:

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

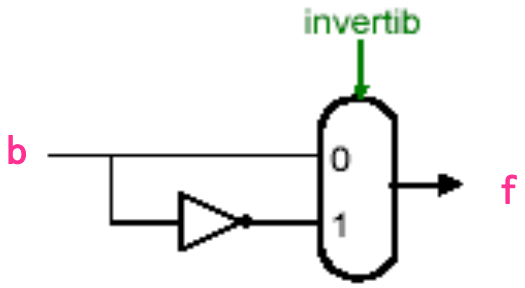
- Complemento a 2:

  $- b = \sim b + 1$

- Realizzazione:
  - Inversione logica
  - Aggiunta della costante "1"

Invertitore

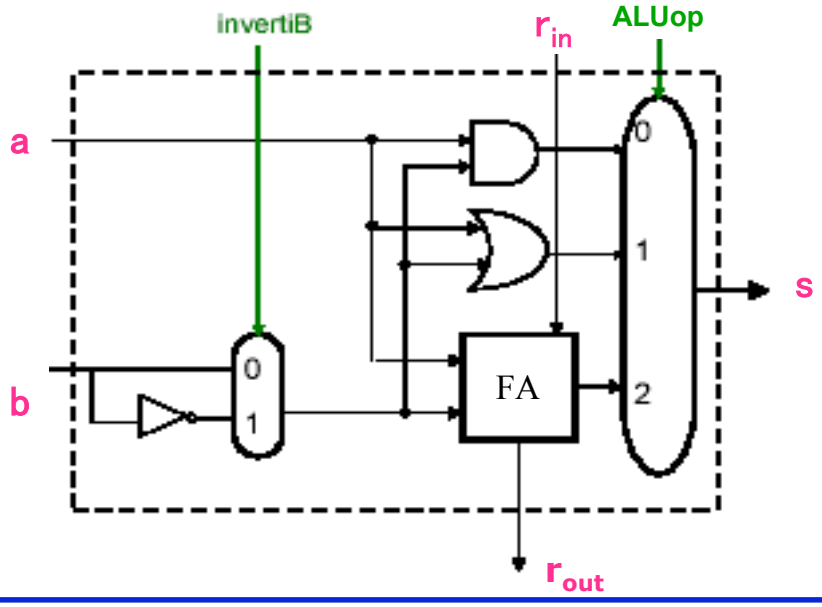
If (invertib=1) → **f = ~b**



ALUi

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

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



Comparazione (confronto)

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

if a < b **then** s = 1

Dove:

- s(i) = 0, i = 1, 2, 3, ... 31
- s(0) = 1 if (a < b) & (**ALUop** = "comparazione")

if (a<b) → s = [0 0 0 … 0 **1**]

else → s = [0 0 0 … 0 **0**]

Come sviluppare la comparazione?

a < b → a – b < 0

- MSB della somma (bit di segno) = 1 → $s_{MSB} = 1$
- 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$_0$**

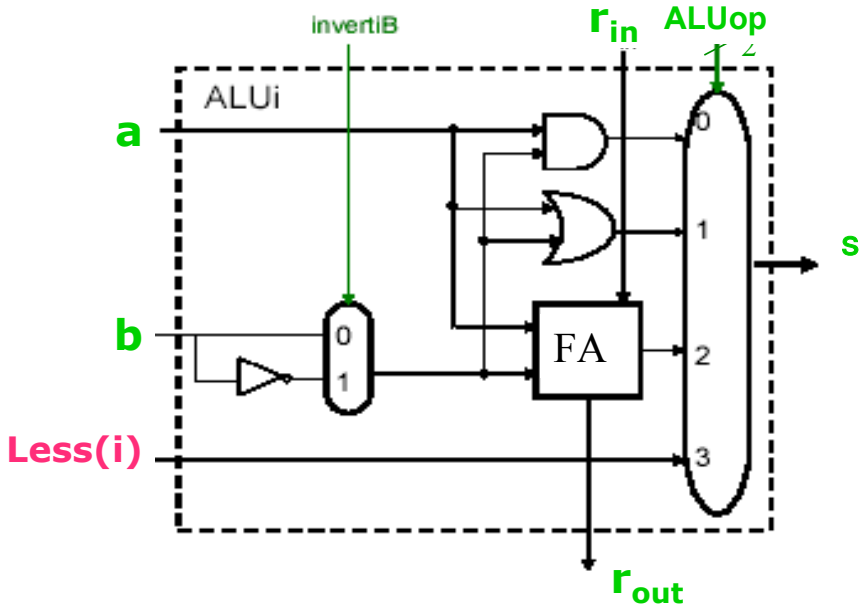
$$s_{31} \rightarrow LESS_0$$

  - 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:

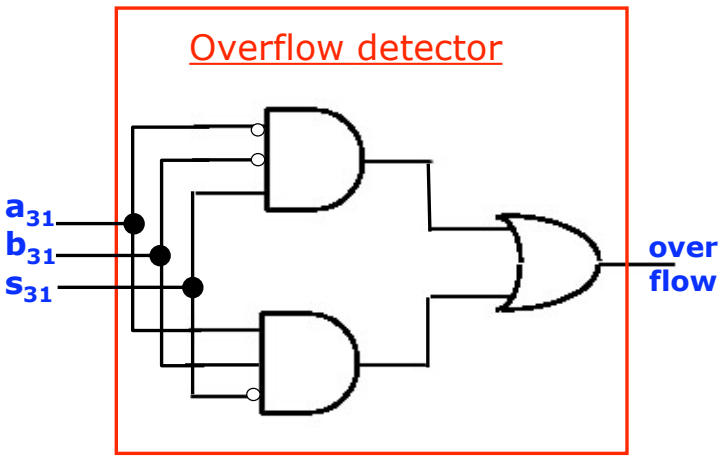
  a + b = s, a > 0, b > 0 → **MSB di a e b = 0, MSB di s = 1**

  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 ?

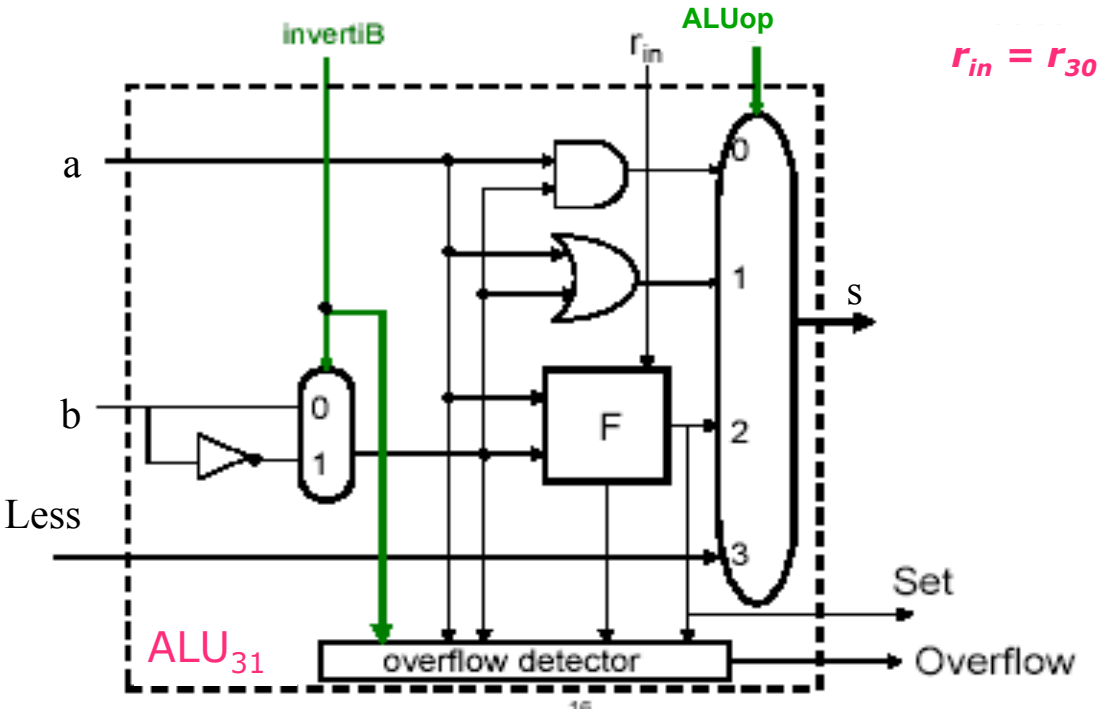
Circuito di riconoscimento dell'overflow

- 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



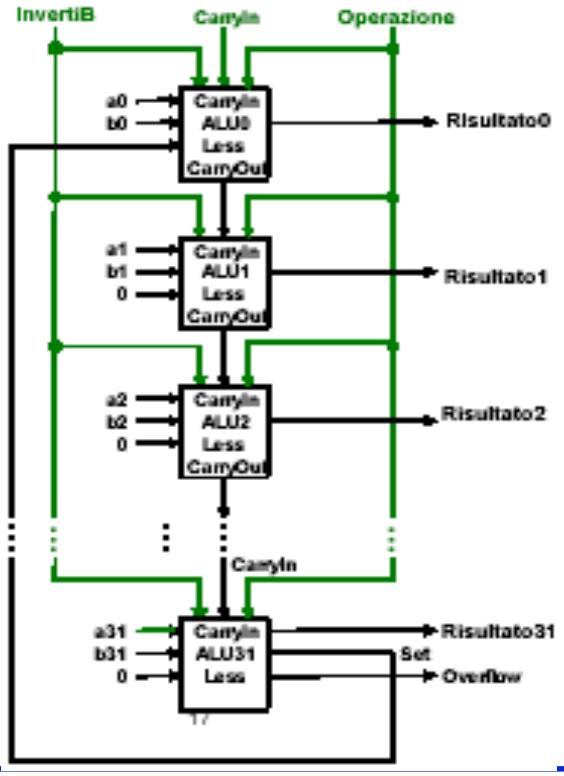
ALU31 con overflow detector



$r_{in} = r_{30}$

ALU completa a 32 bit

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



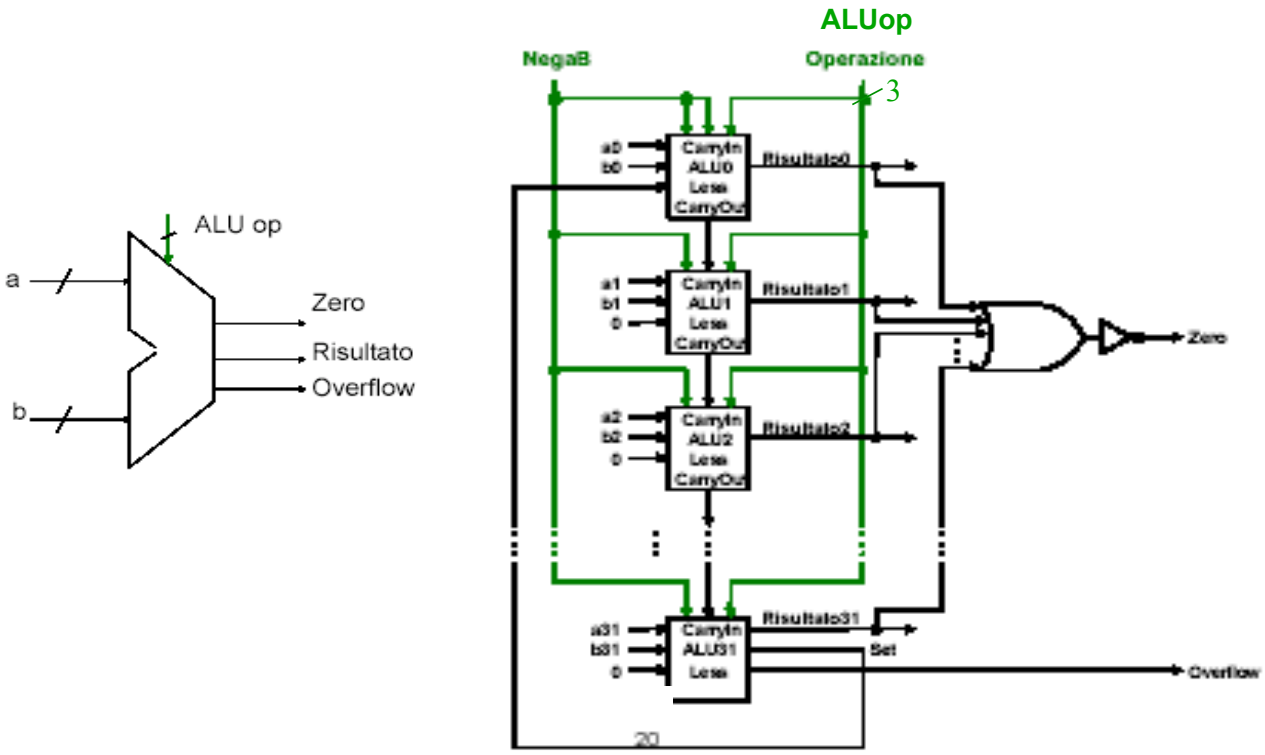
Test di uguaglianza

- *Esempio: istruzione Assembly:*

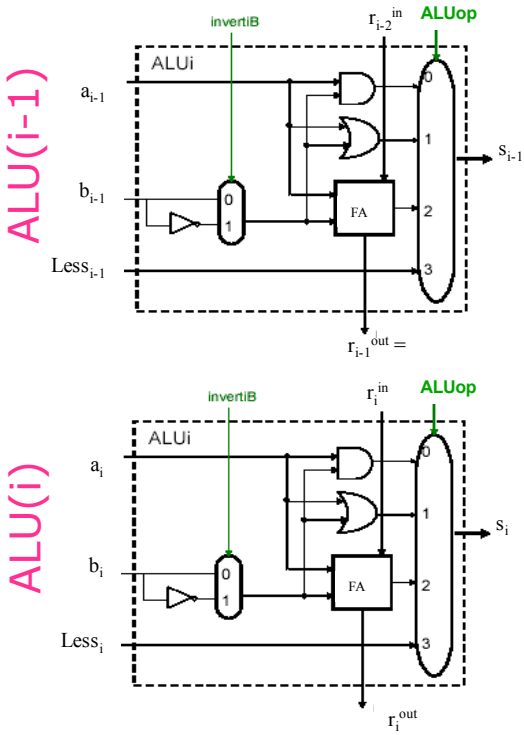
```
beq rs, rt label      # if (rs – rt) = 0 , salta
```

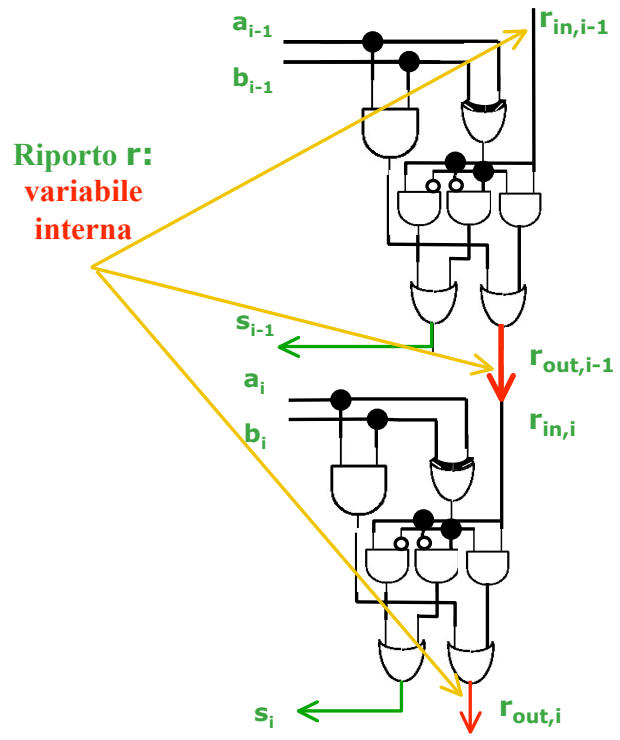
- *Operazioni necessarie*
  - Impostare una differenza.
  - Effettuare l' OR di tutti i bit somma.
  - Uscita dell' OR = 0 → i due numeri sono uguali
- *Operazioni possibili:*
  - AND
  - OR
  - Somma / Sottrazione
  - Comparazione
  - Test di uguaglianza

ALU a 32 bit: struttura finale



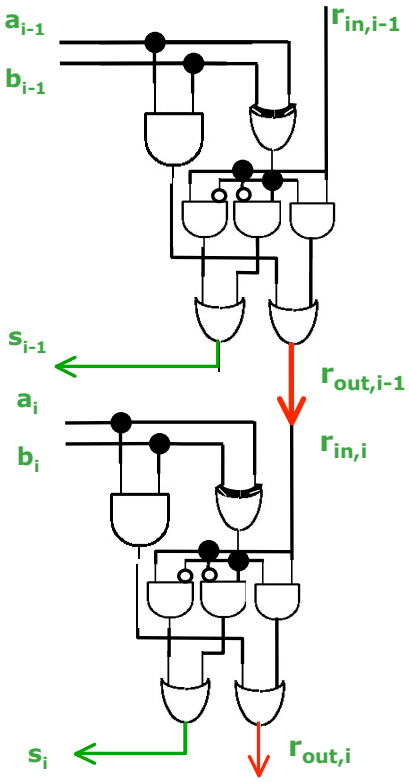
Circuito della somma





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