



# Architettura degli Elaboratori I

Corso di Laurea Triennale in Informatica

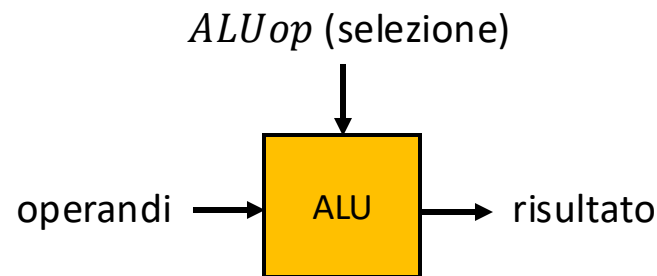
Università degli Studi di Milano

Dipartimento di Informatica "Giovanni Degli Antoni"

Edizione 2 (Cognomi H-Z), A.A. 2024-2025, Nicola.Basilico@unimi.it

# ALU: Unità Aritmetico-Logica

- In una CPU, la ALU è la «centrale» hardware che svolge le operazioni di calcolo, queste possono essere di tipo **aritmetico** (ad esempio, somme, sottrazioni, etc. ..) oppure di tipo **logico** (ad esempio AND e OR)
- A livello astratto la ALU è un **circuito combinatorio multifunzione** definito così:
  - INPUT: gli operandi e un codice di selezione, chiamato **ALUop** che identifica la funzione  $f$  (l'operazione richiesta alla ALU)
  - OUTPUT: il risultato dell'applicazione di  $f$  agli operandi



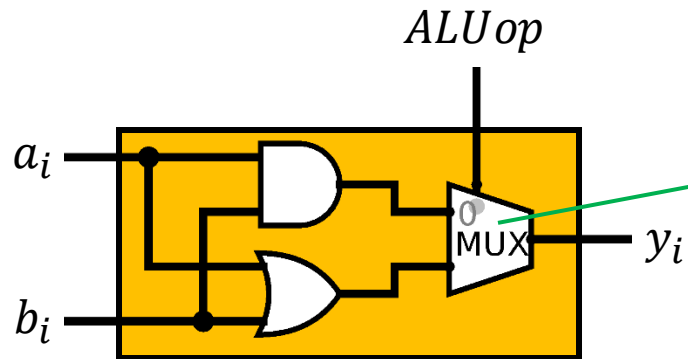
- Logica di progettazione:
  - **Modulare**: una ALU da  $n$  bit (per operandi e risultato) si progetta componendo ALU da 1 bit (moduli)
  - **Parallela**: dati gli operandi, la ALU calcola internamente tutte le funzioni di cui è capace, la selezione (multiplexer) ne porrà in uscita una sola

# ALU

- La ALU è un componente centrale nella Architettura di Von Neumann, è l'elemento responsabile della fase di **execute**
- Ogni operazione supportata è eseguita **in hardware** da un sotto-circuito combinatorio dedicato
- Le CPU moderne possono contenere diverse ALU per svolgere più operazioni in parallelo, ad esempio nelle GPU (CPU progettate specificatamente per calcoli finalizzati alla grafica)
- Le ALU possono essere molto sofisticate, in questo corso vedremo la ALU MIPS, una versione base che supporta queste operazioni tra due operandi  $a$  e  $b$ :
  - AND (logica)
  - OR (logica)
  - Somma (aritmetica)
  - Sottrazione (aritmetica)
  - Comparazione (logica)
  - Test di uguaglianza allo zero (logica)

# AND e OR

- Iniziamo col progettare una ALU elementare ad 1 bit in grado di svolgere le operazioni logiche di AND e OR tra due operandi di 1 bit che indichiamo con  $a_i$  e  $b_i$



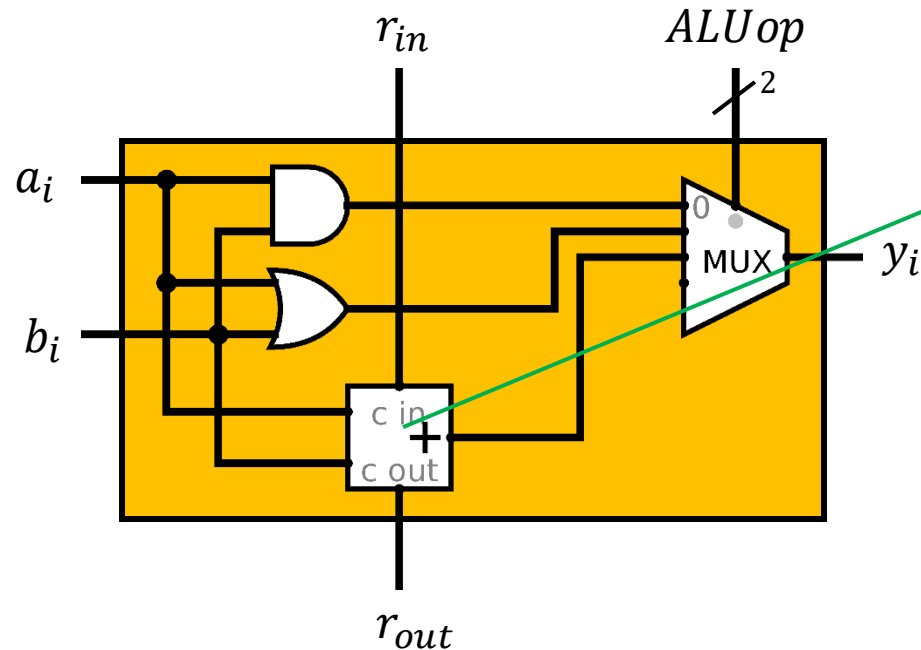
La selezione è implementata con un Multiplexer (in questo caso MUX a 1 bit)

- Parallelismo: la ALU calcola sempre sia AND che OR, ma solo una delle due viene posta in uscita attraverso la selezione
- $ALUop$  è, in questo caso, il **singolo** bit di selezione:  
se vale **0** in uscita avremo  $a_i b_i$ , se vale **1** avremo  $a_i + b_i$

$ALUop$	Funzione calcolata
0	$a_i \text{ AND } b_i$
1	$a_i \text{ OR } b_i$

# Somma (ADD)

- Estendiamo la ALU appena realizzata in modo che supporti anche la somma, sempre su 1 bit



Aggiungiamo un Full Adder: la ALU guadagna un nuovo input ( $r_{in}$ ) e un nuovo output ( $r_{out}$ )

La ALU ora richiede di selezionare una di tre diverse operazioni, il MUX deve essere a 2 bit

- Utilizzando questi moduli a 1 bit possiamo costruire una ALU a  $n$  bit che supporta AND, OR e Somma

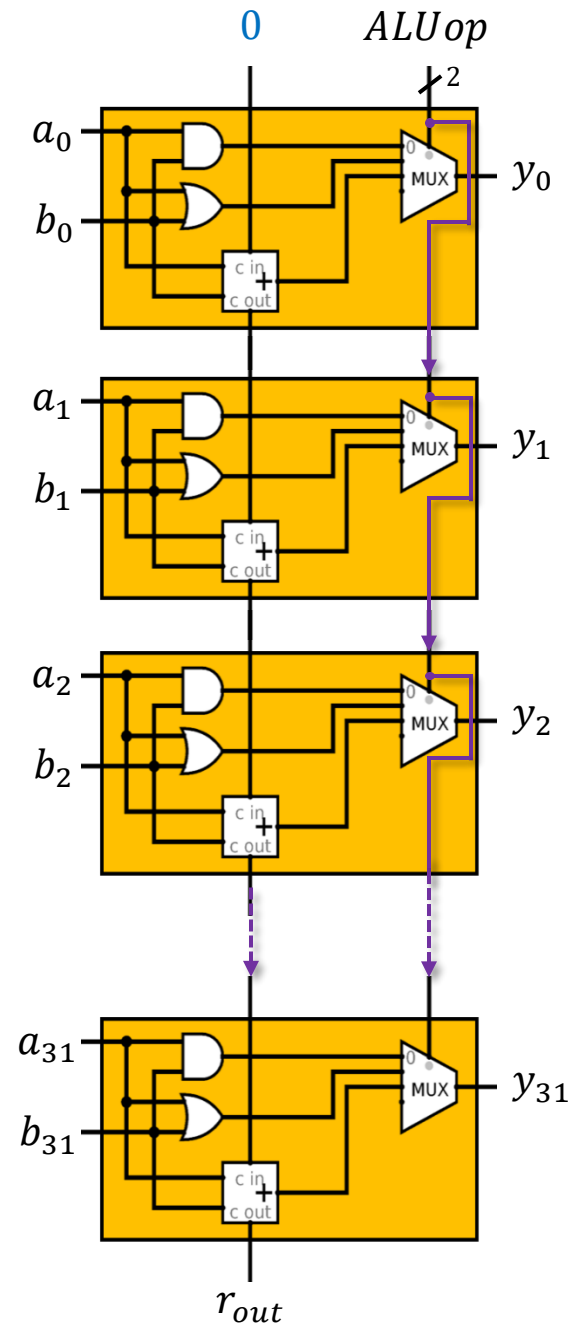
$ALUop$	Funzione calcolata
00	$a_i$ AND $b_i$
01	$a_i$ OR $b_i$
10	Somma ( $a_i + b_i$ )
11	<i>non utilizzato</i>

# AND, OR e ADD su 32 bit

- Collego 32 ALU da 1 bit utilizzando lo schema della propagazione dei riporti (approccio *bitwise*)
- Tutte le ALU lavorano in parallelo
- Il primo riporto è settato a 0
- $ALUop$  è sempre su 2 bit ed è lo stesso in tutte le ALU

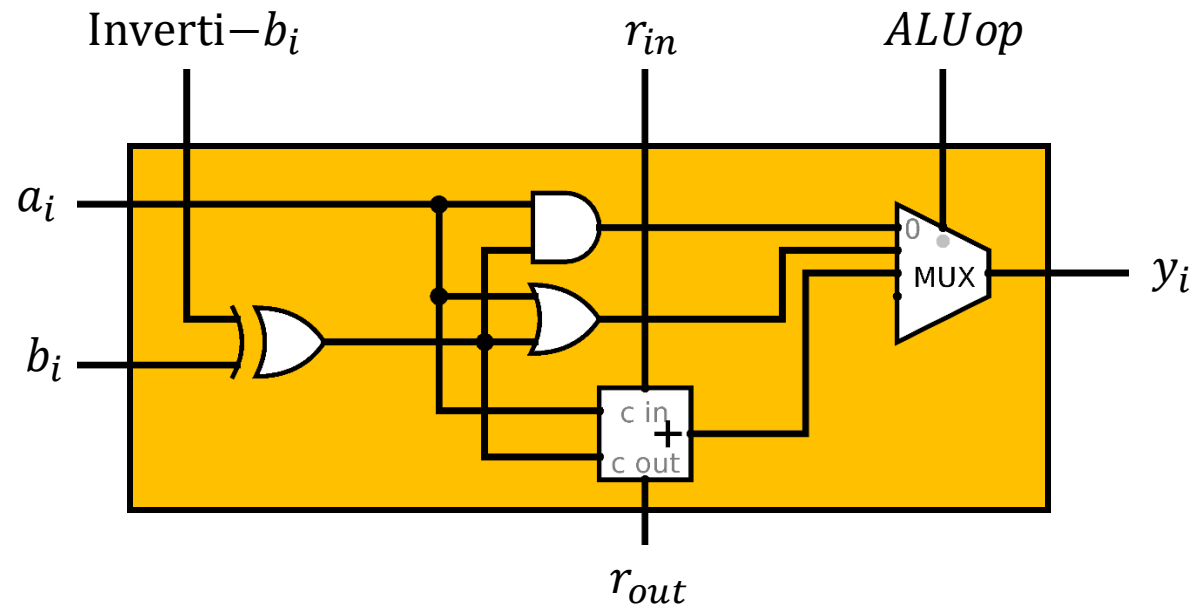
$ALUop$	Funzione calcolata
00	$a$ AND $b$ (bitwise)
01	$a$ OR $b$ (bitwise)
10	Somma ( $a + b$ )
11	<i>non utilizzato</i>

Stesse operazioni di prima,  
ma ora sono su  $n = 32$  bit



# Sottrazione (SUB)

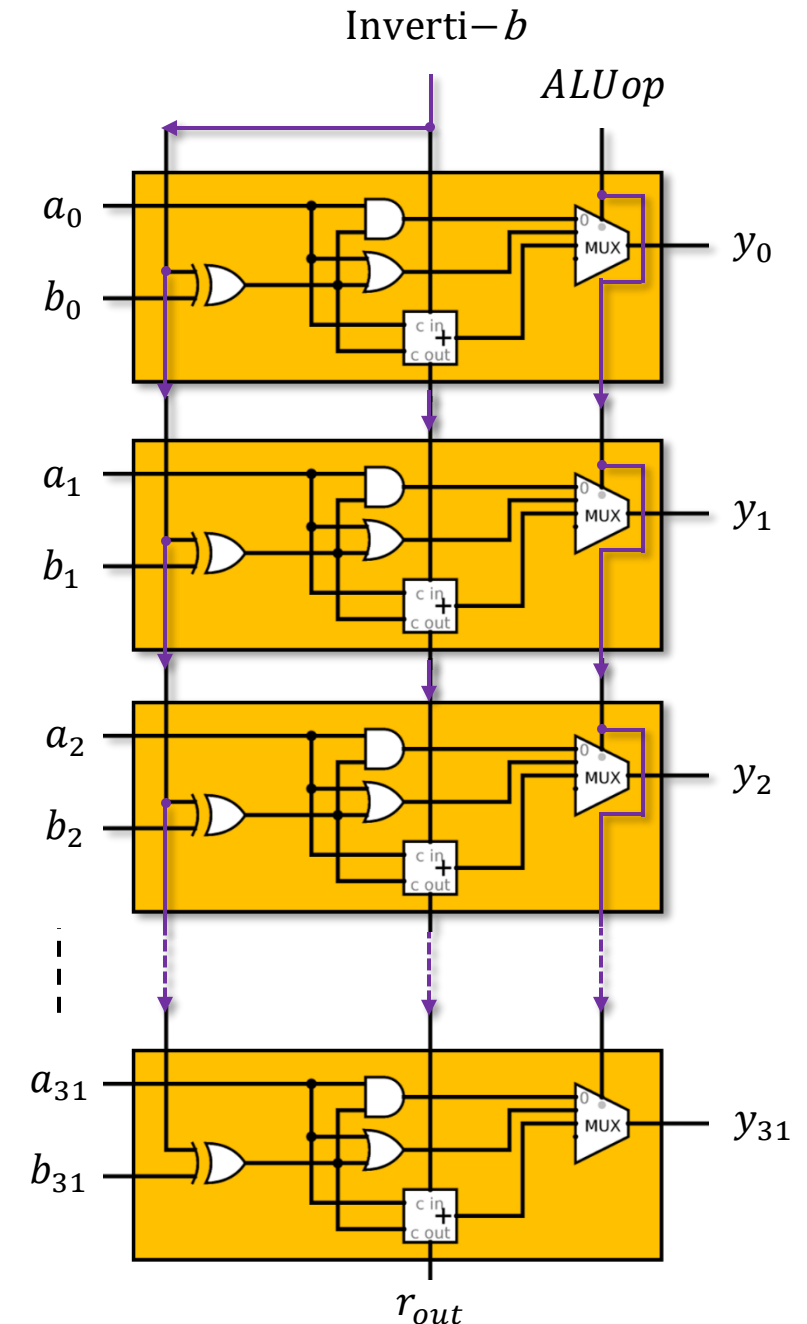
- Per supportare la sottrazione devo aggiungere la possibilità, in ogni full adder, di complementare a 1 l'operando  $b$  e aggiungere 1 alla somma
- Abbiamo già visto come fare: aggiungo un nuovo bit di controllo: Inverti- $b_i$ , se questo segnale viene posto a **1** l'operando  $b_i$  viene sostituito con  $\bar{b}_i$  (il suo complemento a 1)
- Posso fare la sottrazione settando Inverti- $b_i$  e  $r_{in}$  **entrambi** a **1**



# AND, OR, ADD e SUB su 32 bit

- Colleghiamo le 32 ALU da 1 bit usando sempre lo schema di propagazione dei riporti
- $ALUop$ : esattamente come prima (2 bit, lo stesso per tutte le ALU da 1 bit)
- Inverti- $b$ : è lo stesso in tutte le ALU ed è anche collegato al primo  $r_{in}$
- Settando Inverti- $b$  a **1** ottengo simultaneamente che:
  - Tutti i bit di  $b$  vengono invertiti quindi dentro la ALU  $b$  viene subito trasformato in  $\bar{b}$
  - Il primo riporto in ingresso è **1** quindi i sommatori svolgono  $1 + a + \bar{b}$  che in C2 significa  $a - b$

Inverti- $b$	$ALUop$	Funzione calcolata
0	00	$a$ AND $b$ (bitwise)
0	01	$a$ OR $b$ (bitwise)
0	10	Somma ( $a + b$ )
1	10	Sottrazione ( $a - b$ )
*	11	<i>non utilizzato</i>



# Comparazione

- Attraverso la sottrazione possiamo ottenere anche il confronto di uguaglianza tra  $a$  e  $b$ : basta fare  $a - b$  e controllare se il risultato è zero
- **Spoiler alert!** nella struttura finale della ALU aggiungeremo un'uscita di 1 bit detta «bit di zero», che vale **1** ogni qualvolta il risultato calcolato dalla ALU è nullo ( $n$  zeri)
- Per fare i test di disuguaglianza aggiungiamo alla nostra ALU l'operazione logica SLT (Set Less Than):

$$\text{SLT}(a, b) = \begin{cases} \textcolor{red}{1}, & a < b \\ \textcolor{blue}{0}, & a \geq b \end{cases}$$

- Come si implementa all'interno della ALU?

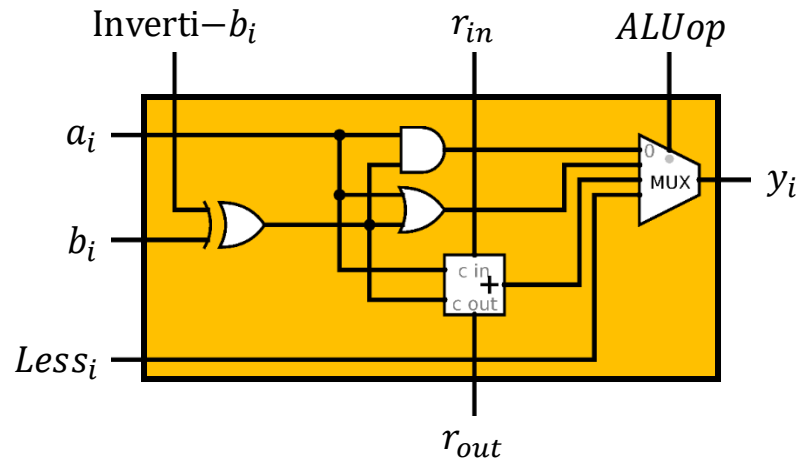
1. Calcoliamo  $a - b$
2. Se il risultato è  $< 0$  allora in uscita mando 000 ... 1
3. Altrimenti in uscita mando 000 ... 0

- Per verificare che il risultato della differenza sia negativo basta controllare il bit in uscita dal Full Adder nella ALU in posizione  $n - 1$  (l'ultima ALU) e cioè il **bit di segno del risultato**
- I bit dalla posizione 1 a  $n - 1$  (gli  $n - 1$  bit più alti) dell'uscita sono **sempre 0**

1. Setto le uscite  $y_1, y_2, \dots, y_{n-1}$  a 0
2. Calcolo  $a - b$
3. Setto l'uscita  $y_0$  a  $s_{n-1}$  (bit di somma in posizione  $n - 1$ )

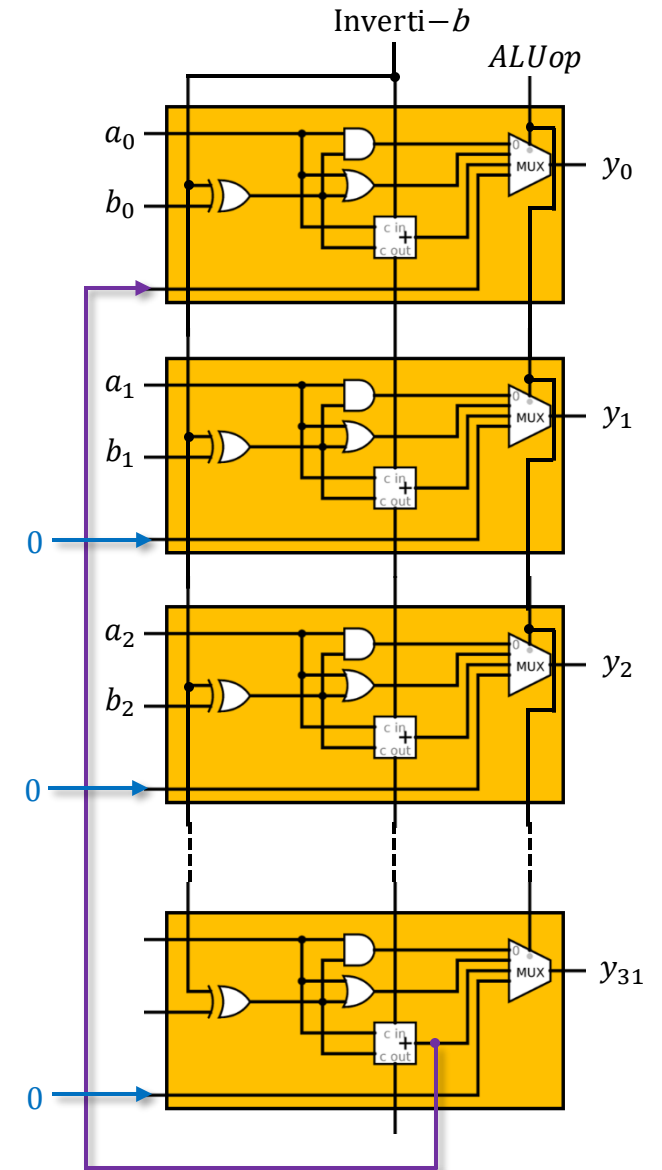
# Comparazione (SLT)

- Estendo il modulo ALU da 1 bit aggiungendo un nuovo input  $Less_i$
- Questo bit viene passato sull'uscita quando il selettore  $ALUop$  vale **11**, cioè la configurazione di selezione che fino ad ora era *non utilizzata* e che ora **assegniamo a SLT**



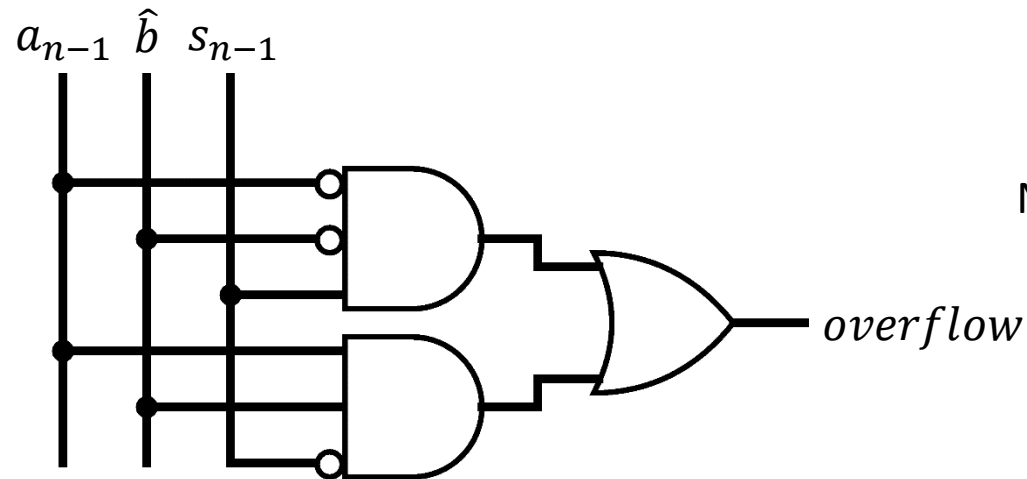
## Per fare la comparazione:

- Setto  $Less_1, Less_2, \dots, Less_{n-1}$  a 0
- Setto Inverti- $b$  a 1, così i sommatore svolgono  $a - b$
- Setto  $Less_0 = s_{n-1}$  (il bit di segno del risultato di  $a - b$ )
- Setto  $ALUOp$  a 11

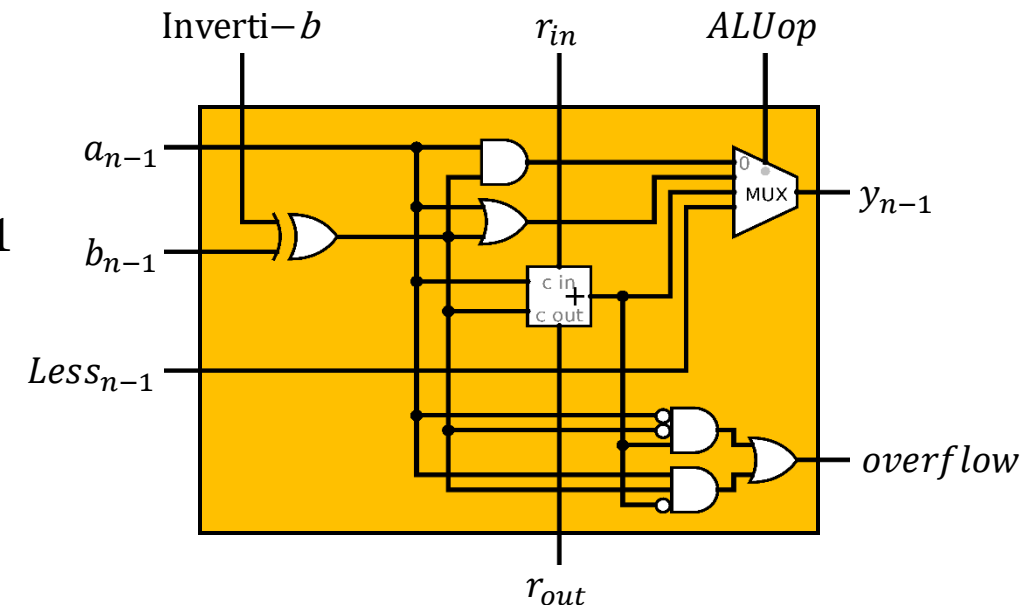
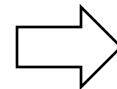


# Overflow

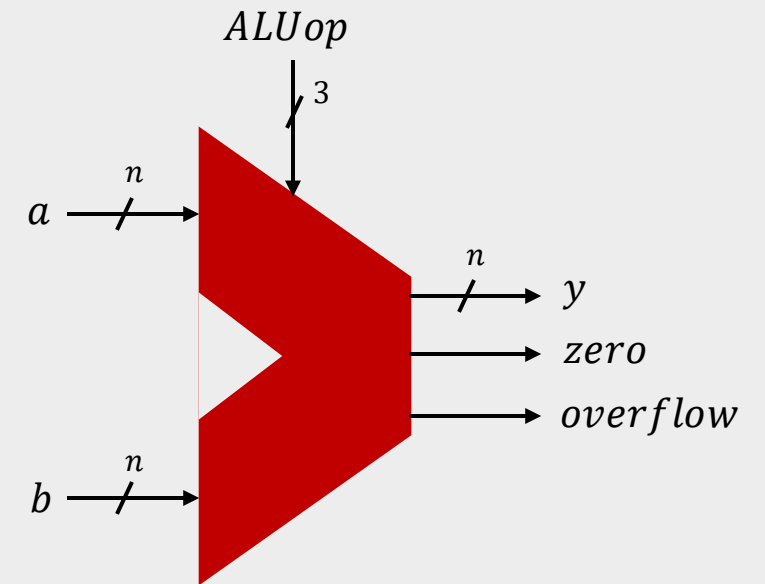
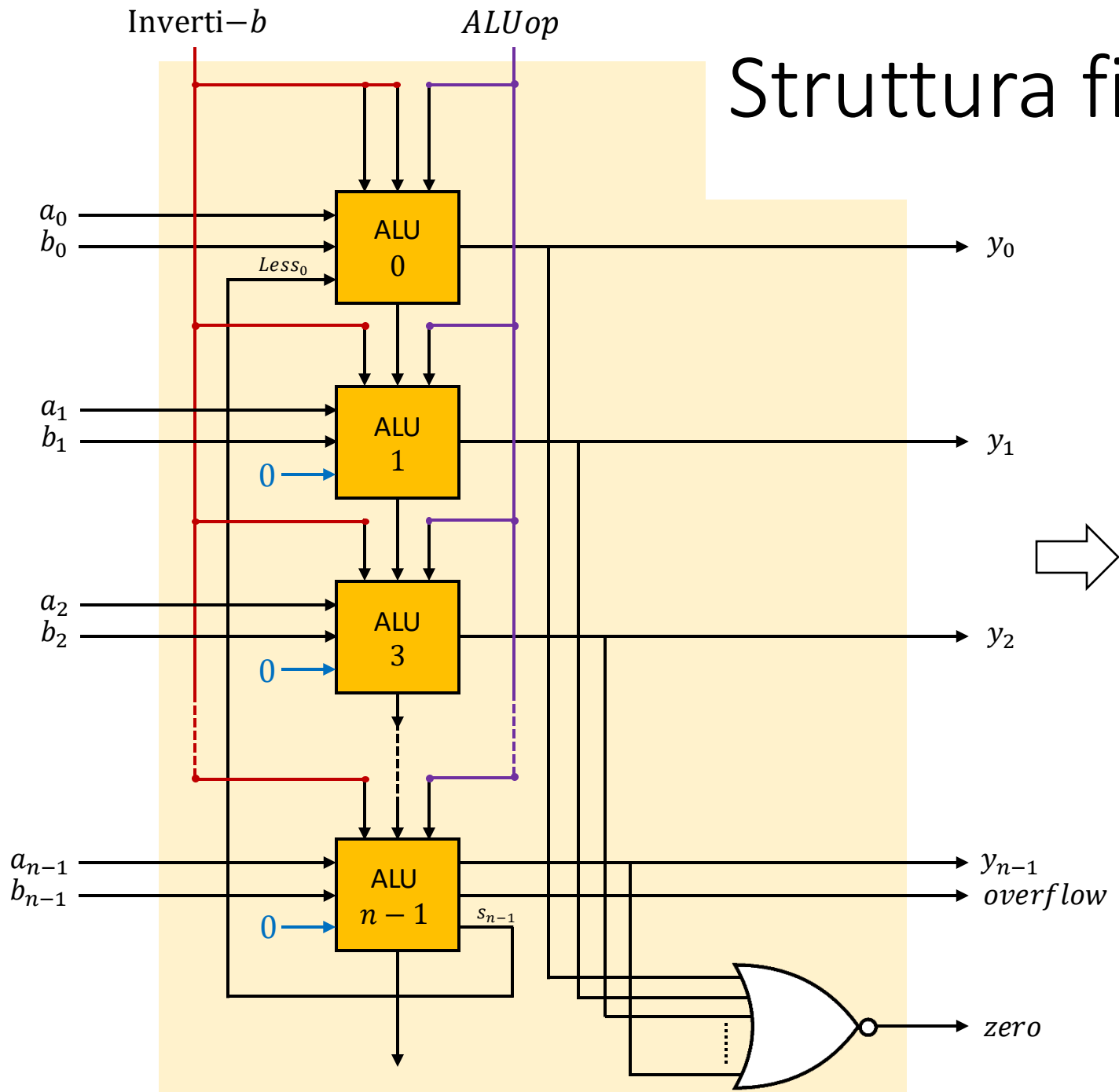
- Come riconoscere la presenza di overflow?
- Richiamo: in C2 l'overflow identifica in due modi
  1. Gli ultimi due riporti sono diversi
  2. **Sommando 2 numeri positivi ottengo un negativo oppure sommando 2 negativi ottengo un positivo**
- Possiamo implementare il secondo metodo con un semplice circuito combinatorio che prende in input i bit di segno di  $a$ ,  $b$  e del risultato della somma  $s$ : **questi bit stanno tutti nella ALU  $n - 1$ !**
- **ATTENZIONE!** Nella ALU  $n - 1$  il bit di segno di  $b$  non è  $b_{n-1}$  ma  $b_{n-1} \oplus \text{Inverti-}b = \hat{b}$



Nella ALU  $n - 1$



## Struttura finale (con bit di zero)



<i>ALUop</i>	Funzione calcolata
000	$a$ AND $b$ (bitwise)
001	$a$ OR $b$ (bitwise)
010	ADD( $a, b$ )
110	SUB( $a, b$ )
111	SLT( $a, b$ )

# Varianti della ALU

- ALU per operazioni in floating point?

**Esempio:** somma

- Supponiamo di avere due numeri in formato IEEE 754:  $N_1 = \langle s_1, E_1, m_1 \rangle$ ,  $N_2 = \langle s_2, E_2, m_2 \rangle$
- Come calcolo  $N_1 + N_2$ ? Posso semplicemente sommare i 3 campi usando la ALU introdotta precedentemente? **NO!** Serve un circuito più complesso: ALU-FP (ALU per operazioni in floating point)
- Come si svolge  $99.5 + 0.85$ ?
- $99.5 + 0.85$  normalizzando diventa:  $9.95 \times 10^1 + 8.5 \times 10^{-1}$
- Allineo gli esponenti de-normalizzando (temporaneamente) il numero con esponente più basso:  
$$9.95 \times 10^1 + 0.085 \times 10^1$$
- Sommo le mantisse:  $9.95 + 0.085 = 10.035$
- Normalizzo il risultato:  $1.0035 \times 10^2$