



**UNIMORE**  
UNIVERSITÀ DEGLI STUDI DI  
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,  
Informatiche e Matematiche

# 7. Componenti Notevoli Combinatori

## Architettura dei calcolatori [MN1-1143]

*Corso di Laurea in INFORMATICA*  
(D.M.270/04) [16-215]  
Anno accademico 2022/2023

**Prof. Alessandro Capotondi**  
[a.capotondi@unimore.it](mailto:a.capotondi@unimore.it)

*È vietata la copia e la riproduzione dei contenuti e immagini in qualsiasi forma.*

*È inoltre vietata la redistribuzione e la pubblicazione dei contenuti e immagini non autorizzata espressamente dall'autore o dall'Università di Modena e Reggio Emilia.*

# Demultiplexer / Decoder (1/2)

- Il **demultiplexer** (*decoder*) realizza la funzione di smistare un singolo input in una delle  $n$  possibili uscite
- Formalmente il **demultiplexer** è una rete logica con 1 ingresso di dato,  $n$  segnali di controllo e  $2^n$  uscite: l'uscita contrassegnata dall'indice pari alla configurazione dei segnali di controllo riceve l'ingresso, mentre le altre non sono abilitate (normalmente poste a livello logico 0).



- Si dice anche **decoder** in quanto viene usato per decodificare un segnale binario (se si mantiene l'ingresso **EN** a 1).

# Demultiplexer / Decoder (1/2)

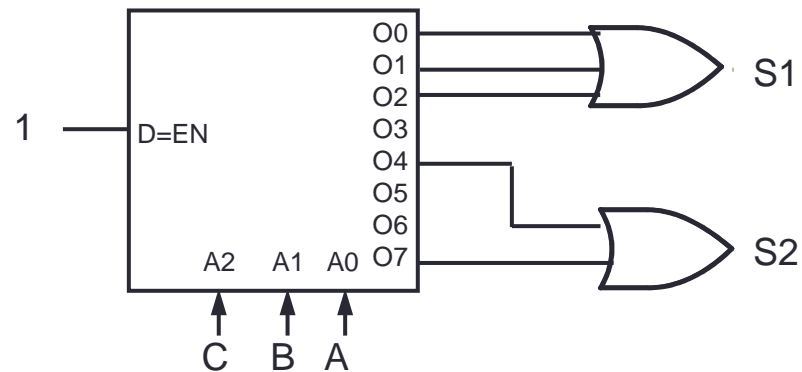
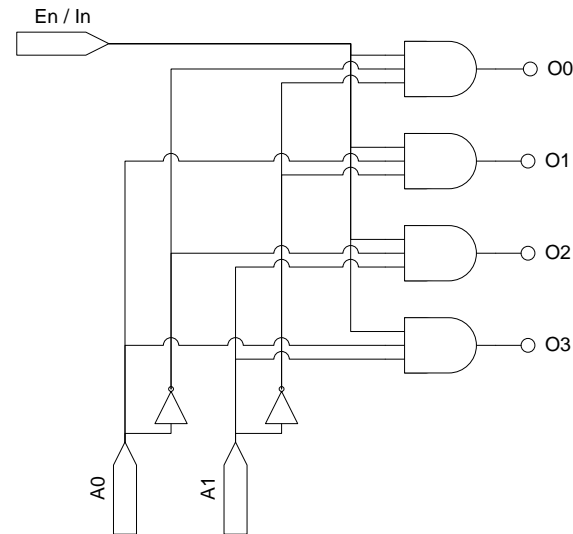
Può essere usato come  
generatore di *mintermini*:

## Esempio:

realizzare la rete logica

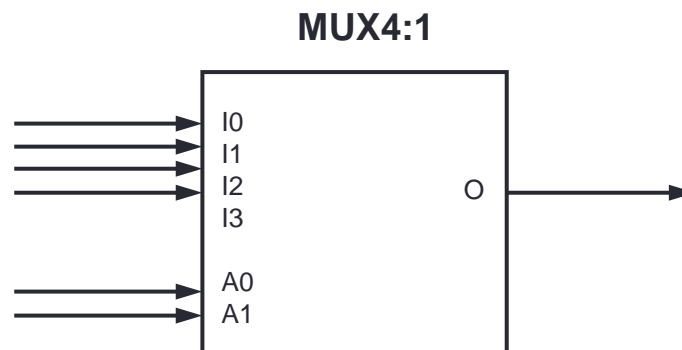
$$S1 = A'BC' + A'B'C + A'B'C'$$

$$S2 = AB'C' + ABC$$



# Multiplexer (1/3)

- **Multiplexer:** è quel blocco logico che permette di *deviare* su un'unica uscita un segnale proveniente da uno tra  $n$  possibili ingressi.
- Formalmente: è una rete logica avente  $2^n$  ingressi di tipo *dati* e  $n$  ingressi di tipo *segnali di controllo (o indirizzo)* ed 1 uscita: in ogni istante il dato presente all'ingresso selezionato (mediante la configurazione dei segnali di controllo) viene riportato in uscita

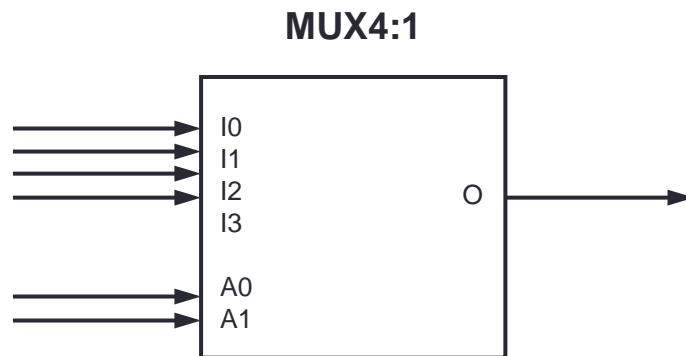


# Multiplexer (2/3)

- Sintesi attraverso la tabella della verità

$$O = I_3A_1A_0 + I_2A_1A_0' + I_1A_1'A_0 + I_0A_1'A_0'$$

*esegue la somma di tanti prodotti quanti gli ingressi (di dato), in cui ogni prodotto è un mintermine degli ingressi di controllo*



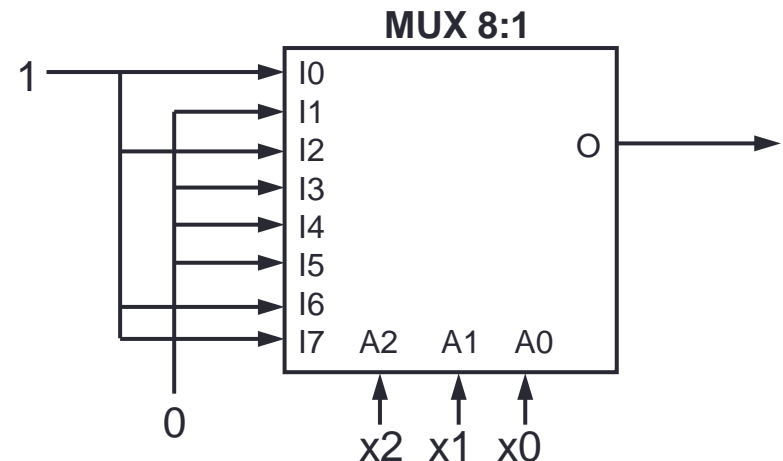
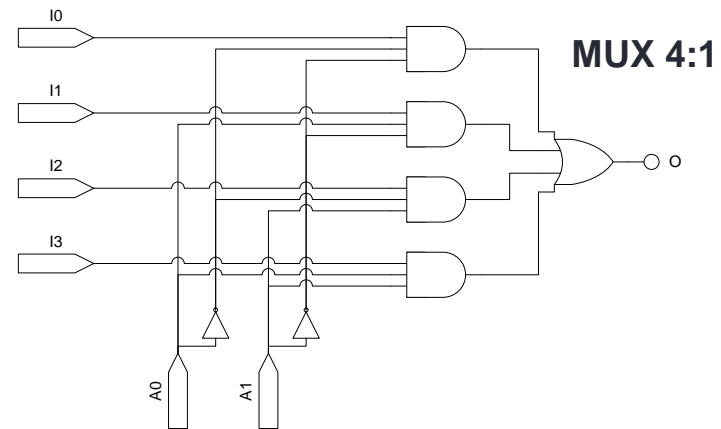
$A_1$	$A_0$	$I_3$	$I_2$	$I_1$	$I_0$	$O$
0	0	×	×	×	0	0
0	0	×	×	×	1	1
0	1	×	×	0	×	0
0	1	×	×	1	×	1
1	0	×	0	×	×	0
1	0	×	1	×	×	1
1	1	0	×	×	×	0
1	1	1	×	×	×	1

# Multiplexer (3/3)

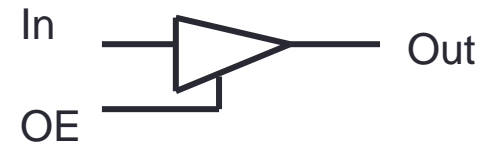
- E' possibile costruire un multiplexer N:1 mettendo in cascata vari livelli di multiplexer più piccoli

Il multiplexer non solo può essere usato come selettore, ma anche come «**generatore di tabelle della verità**»

- Esercizio:**  
realizzare  
 $F(x_0, x_1, x_2) = m_0 + m_2 + m_6 + m_7$   
(somma di 4 mintermini)



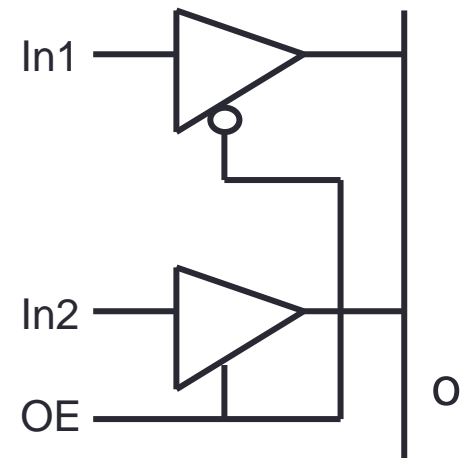
# Amplificatore Tri-State (1/3)



- Amplificatore tri-state: Generatore di segnale in terzo stato (Z)

In	OE	Out
X	0	Z
0	1	0
1	1	1

- l'uscita è uguale all'ingresso quando l'*output enable* (OE) è asserito;
  - Si può pensare ad uno switch (interruttore)
  - Si usa spesso per realizzare multiplexer distribuiti
- **Attenzione: per evitare corto circuiti bisogna che in ogni istante solo un tri-state sia abilitato.**

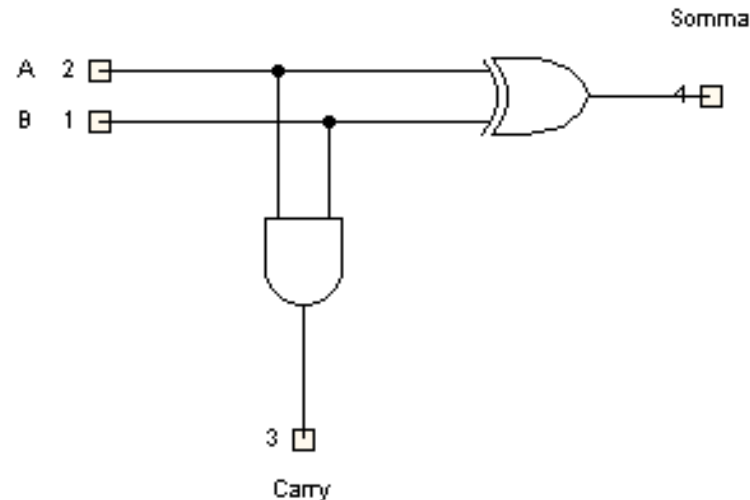




# Half adder (HA)

- Somma due bit in input, restituendo somma ed eventuale riporto.

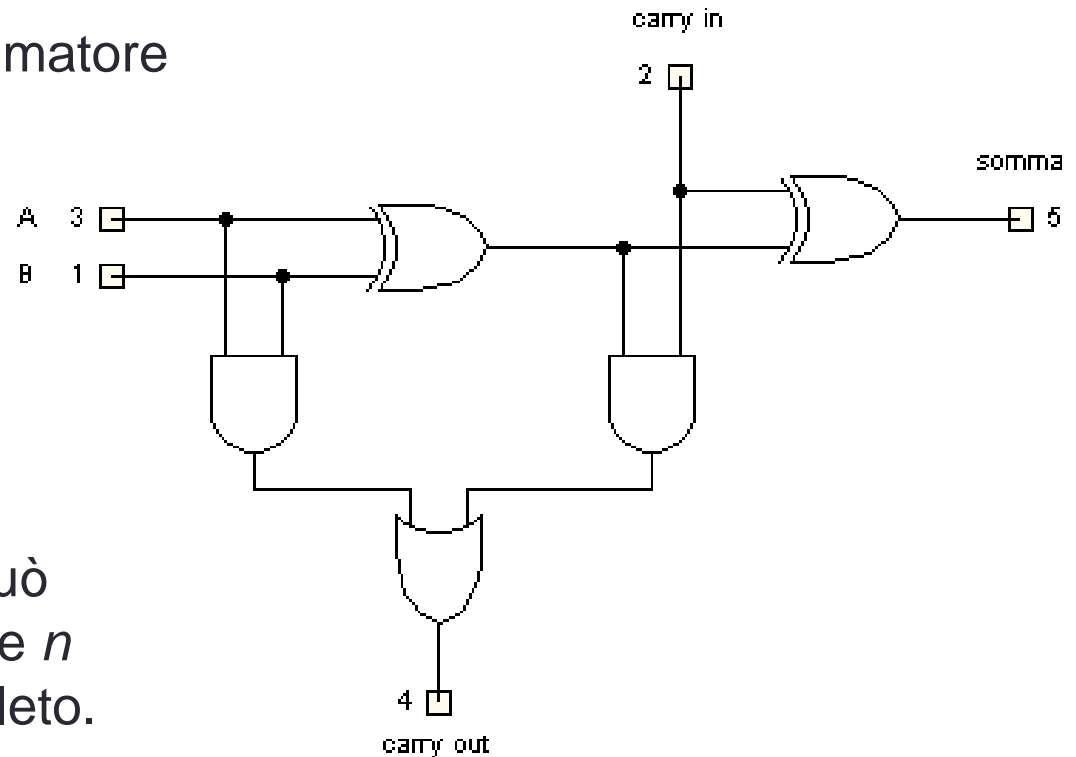
A	B	S	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- E' possibile realizzare un sommatore che tenga conto del riporto in ingresso (*Full Adder*) con le tecniche di sintesi tradizionali.
- Per maggior modularità verrà realizzato mediante la composizione di due *Half Adder (HA)*. Il riporto dell'operazione precedente viene sommato alla somma dei due bit attuali. E' necessario poi comporre opportunamente i riporti di queste due somme.

# Full Adder (FA)

- La versione finale del sommatore completo a 1-bit è:



- Un sommatore a  $n$  bit si può ottenere replicando in serie  $n$  volte un sommatore completo.
- Il riporto (*carry out*) di un bit si usa come *carry in* del sommatore completo alla sua sinistra (cifra più significativa).

# Sommatore ad N-bit

- La soluzione modulare ottenuta collegando in cascata  $N$  *full-adder* ha il vantaggio di essere chiara, facilmente replicabile, probabilmente ottimizzata dal punto di vista del numero di porte complessivo.
- In alternativa si può realizzare un sommatore generando direttamente le  $N$  uscite mediante **forme minime SP (o PS)**. In tal caso si otterrebbe una soluzione con molte **più porte logiche**, meno modulare **ma più veloce (2 livelli e mezzo)**.
- La soluzione mediante *full-adder* ha infatti un difetto, legato alla necessità di propagare il *carry* dal bit meno significativo a quello più significativo. **Rete è più profonda.**

# ALU

- L'ALU (*Arithmetic Logic Unit* o *unità aritmetico logica*) è un circuito combinatorio in grado di svolgere opportune operazioni aritmetiche e/o logiche su due operandi.
- L'operazione non è fissa ma è selezionabile mediante opportuni segnali.
- Normalmente una ALU fornisce in uscita il risultato della operazione e alcuni FLAG, che descrivono il risultato (negativo/zero) o identificano eventuali errori (es., overflow) o situazioni di interesse (es., carry).
- L'elenco delle operazioni eseguite viene definita in fase di progetto e non è fissata a priori.

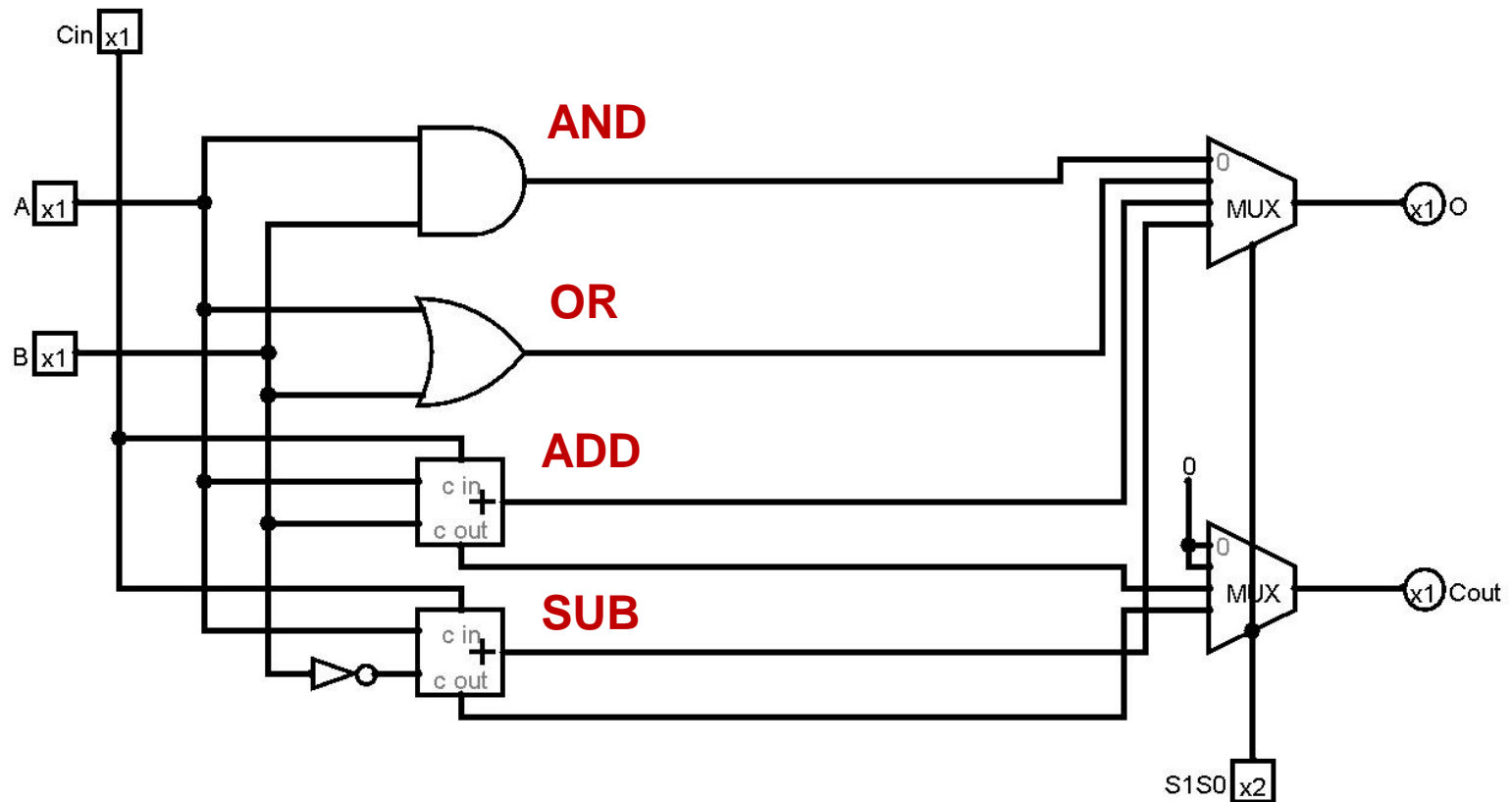
# ALU

- Per motivi didattici, vedremo una implementazione che prevede solamente 4 operazioni
- Per realizzare in modo semplice una ALU è possibile inserire in parallelo tutti i circuiti logici che effettuano le operazioni prescelte e selezionare il risultato desiderato mediante un multiplexer

# ALU a 1-bit

- Combinando opportunamente un sommatore e alcune porte logiche è possibile realizzare delle Unità Aritmetico-Logiche (ALU) che consentono di eseguire operazioni su singoli bit. L'uscita dell'unità viene comandata da un multiplexer che seleziona l'operazione da compiere, a seconda dei segnali di controllo che gli vengono forniti.
- Le 4 operazioni considerate sono:
  - AND logico
  - OR logico
  - Somma (con/senza carry in ingresso)
  - Sottrazione (con/senza carry in ingresso)

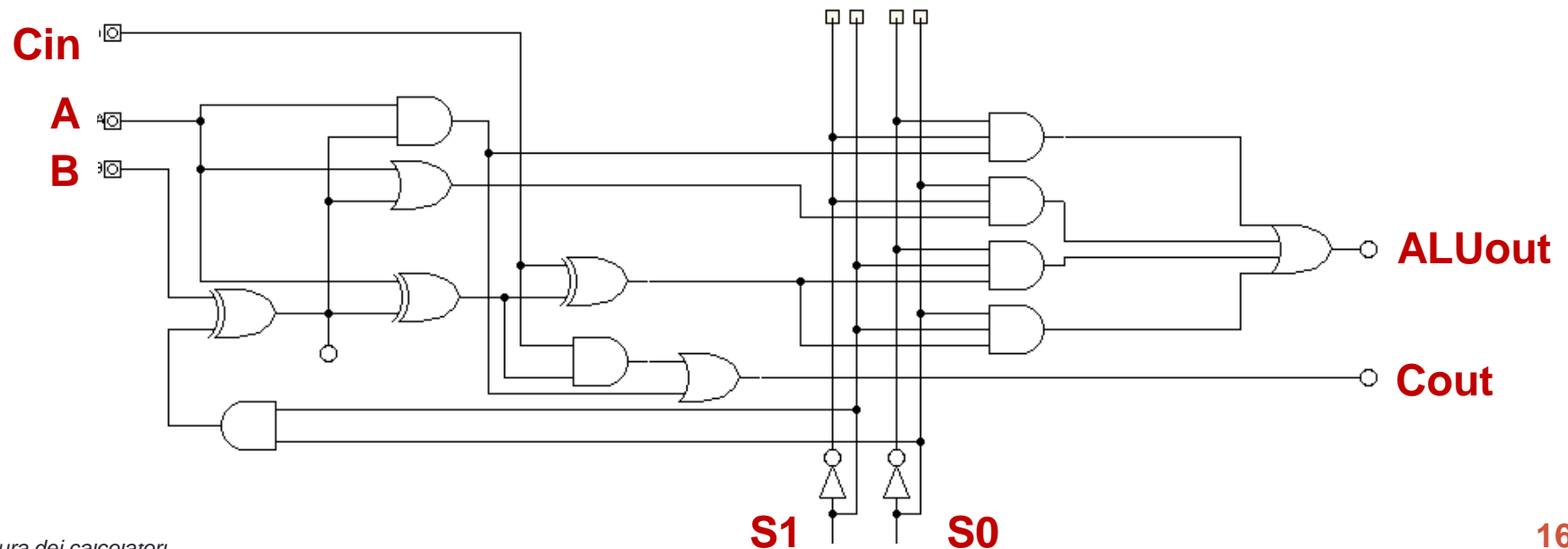
# Schema logico ALU a 1-bit



- *Sottrazione:*  $A - B = A + \text{compl.2 di } B = A + B' + 1$   
→ nego B e aggiungo 1 dal carry-in (solo quello iniziale del primo bit)

# ALU a 1-bit

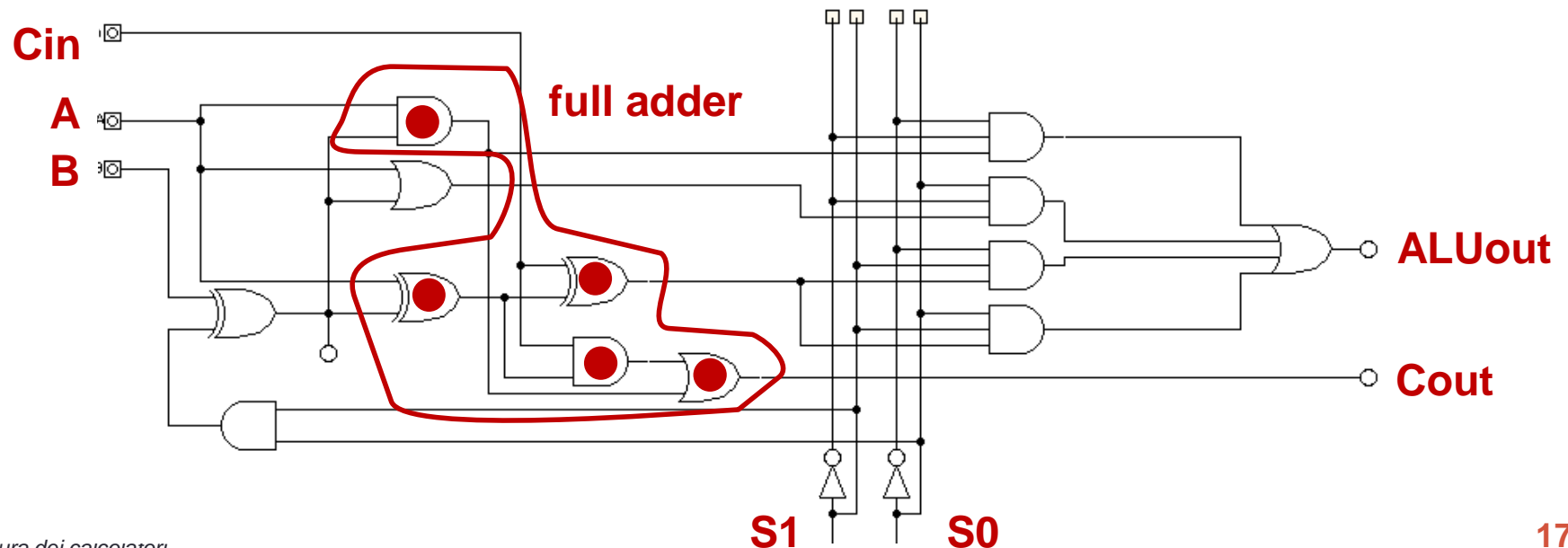
- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:





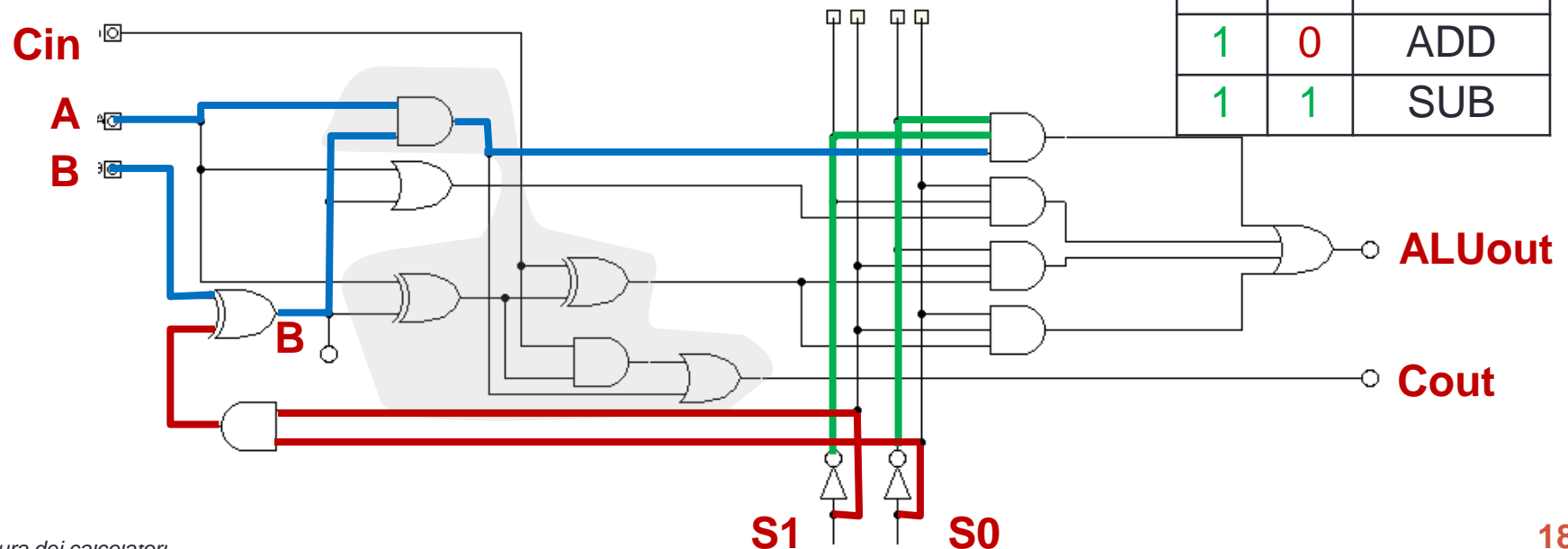
# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione



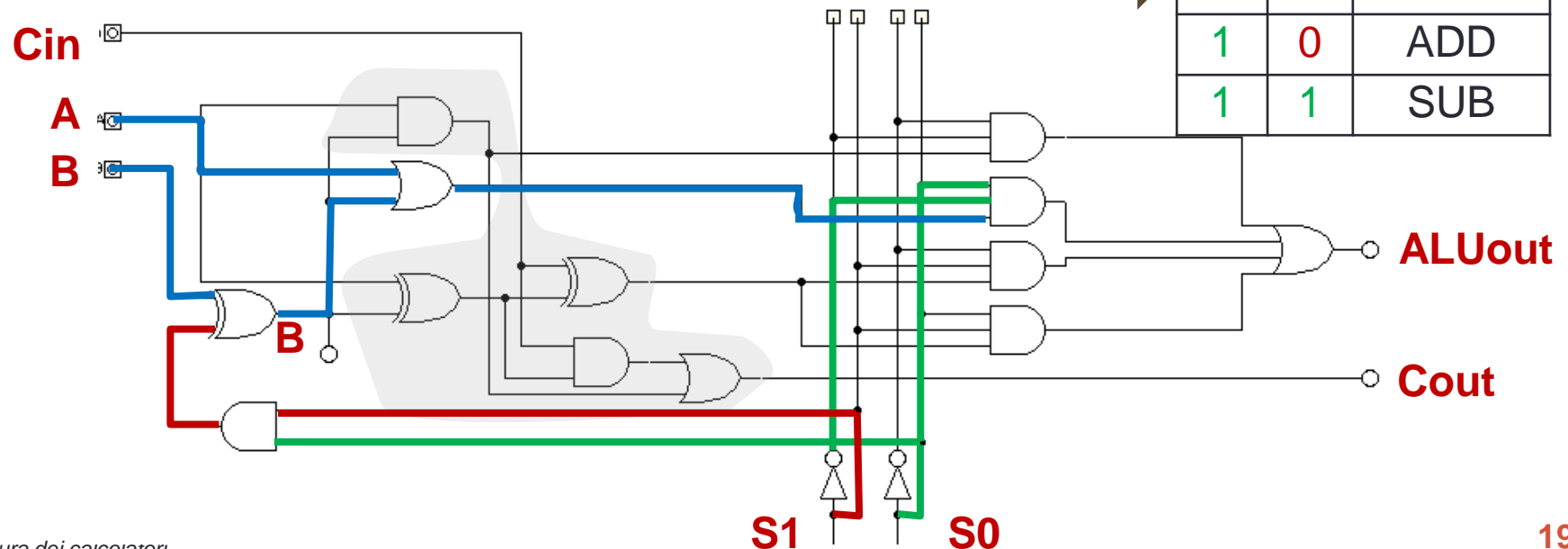
# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione
  - La porta XOR può essere vista come un invertitore comandato da un segnale di controllo



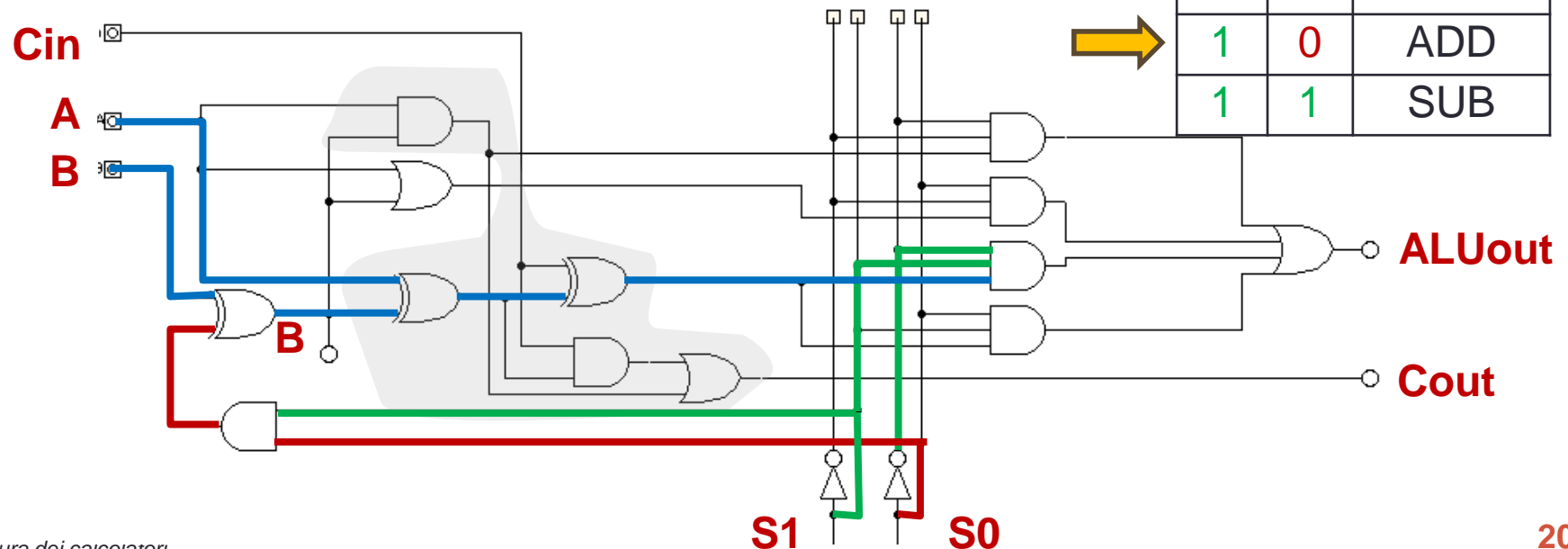
# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione
  - La porta XOR può essere vista come un invertitore comandato da un segnale di controllo



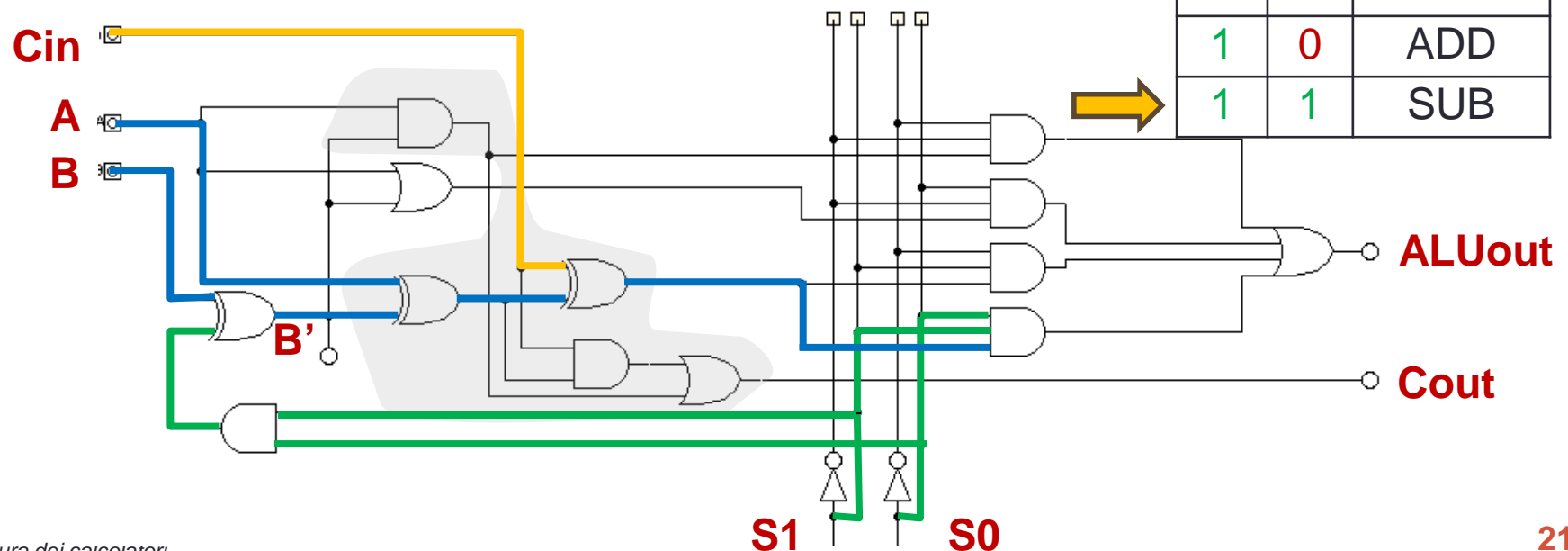
# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione
  - La porta XOR può essere vista come un invertitore comandato da un segnale di controllo



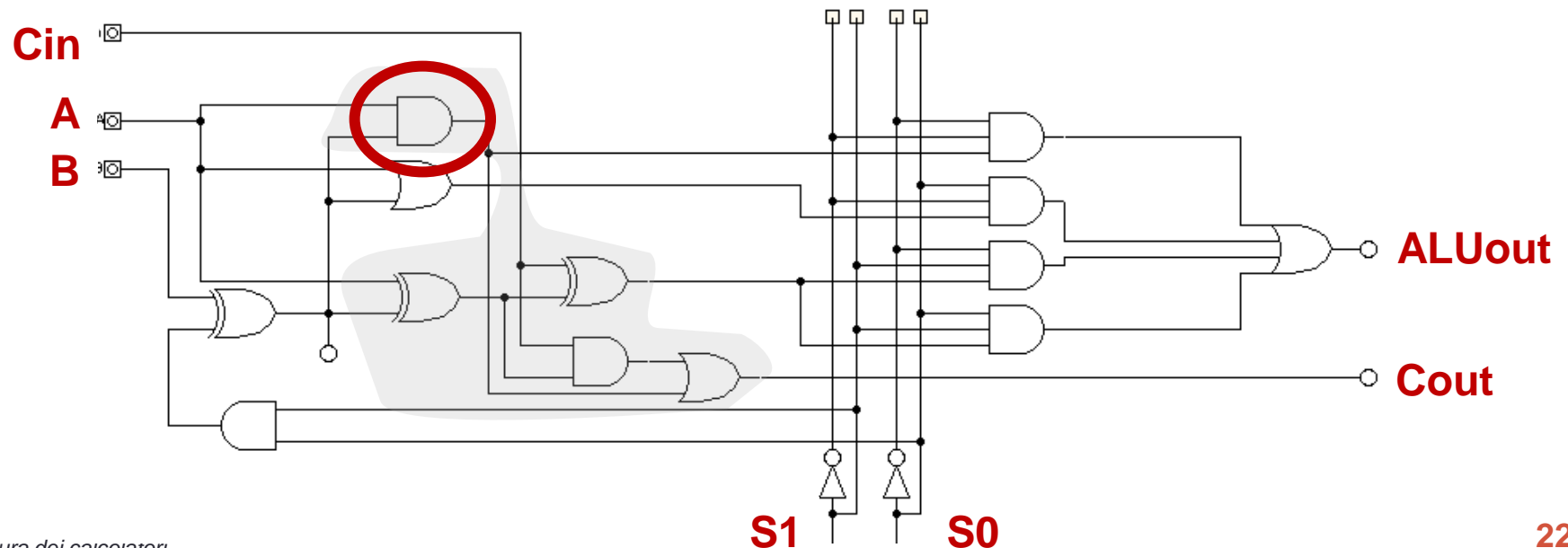
# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione
  - La porta XOR può essere vista come un invertitore comandato da un segnale di controllo

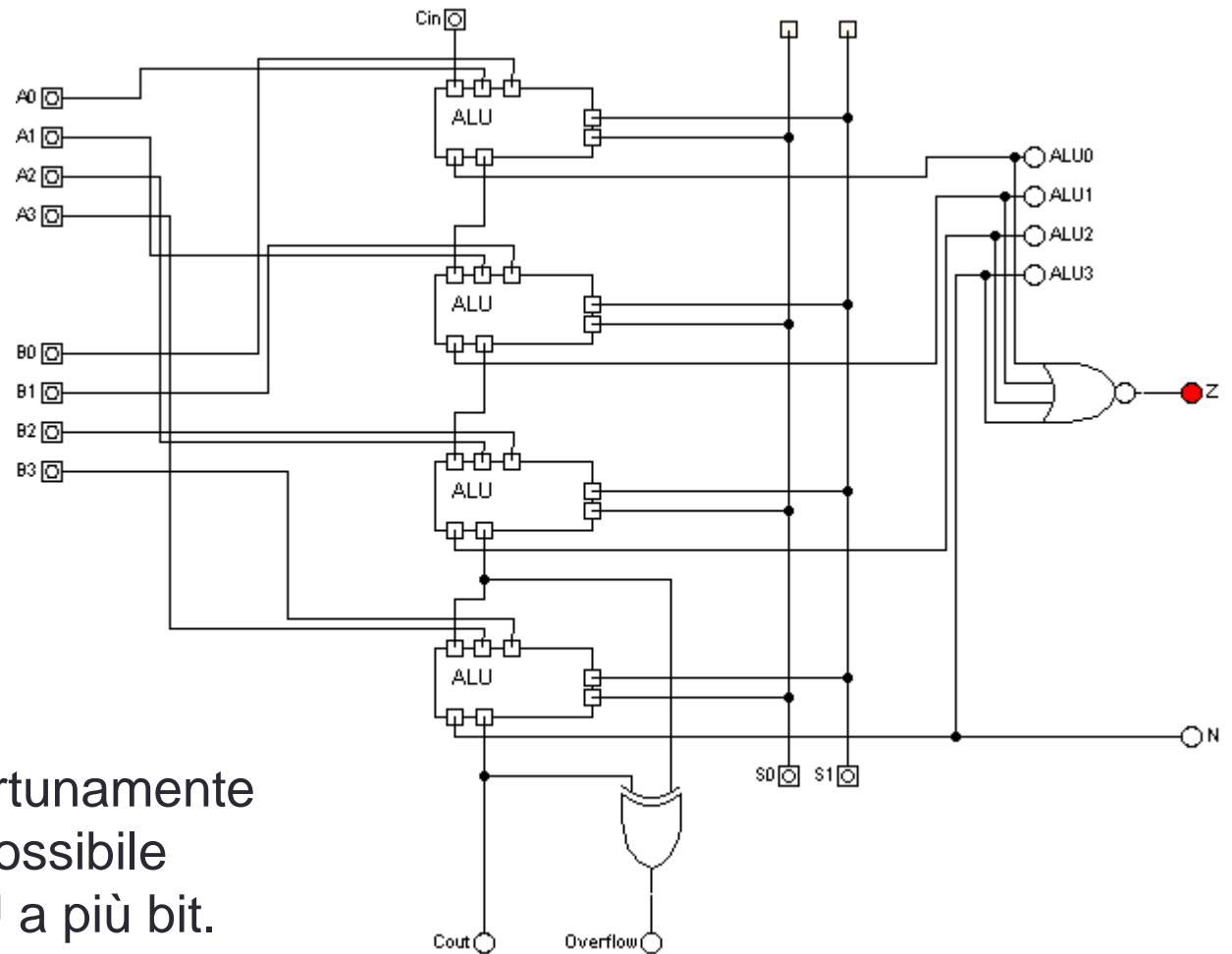


# ALU a 1-bit

- Una versione ottimizzata dell'ALU può essere generata sfruttando le seguenti considerazioni:
  - E' sufficiente un unico *full-adder* per somma e sottrazione
  - La porta XOR può essere vista come un invertitore comandato da un segnale di controllo
  - L'AND tra A e B è necessario sia come operazione che come generazione della prima parte del *carry*



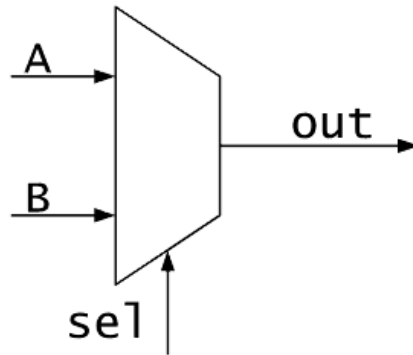
# ALU a 4-bit



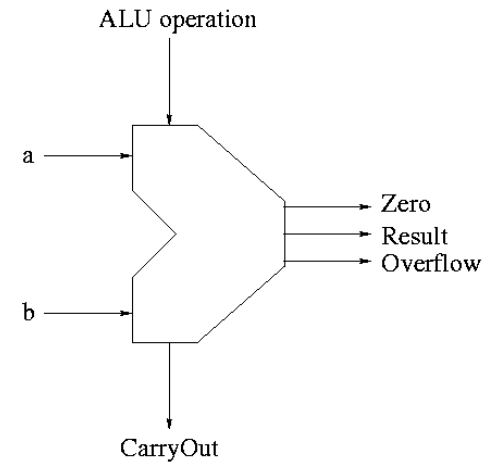
- Combinando opportunamente più ALU a 1-bit è possibile realizzare una ALU a più bit.

# Altri simboli in uso nella pratica

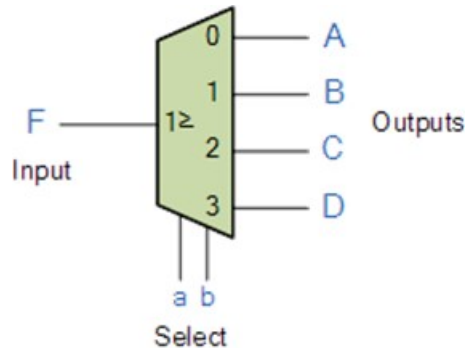
- Multiplexer:



- ALU:



- Demux:





# Evaluation (it is your moment)

- Oppure usa il QR code

Collegati

<https://menti.com>

Inserisci il codice

7497 3572



<https://www.menti.com/alca17dgcy9a>