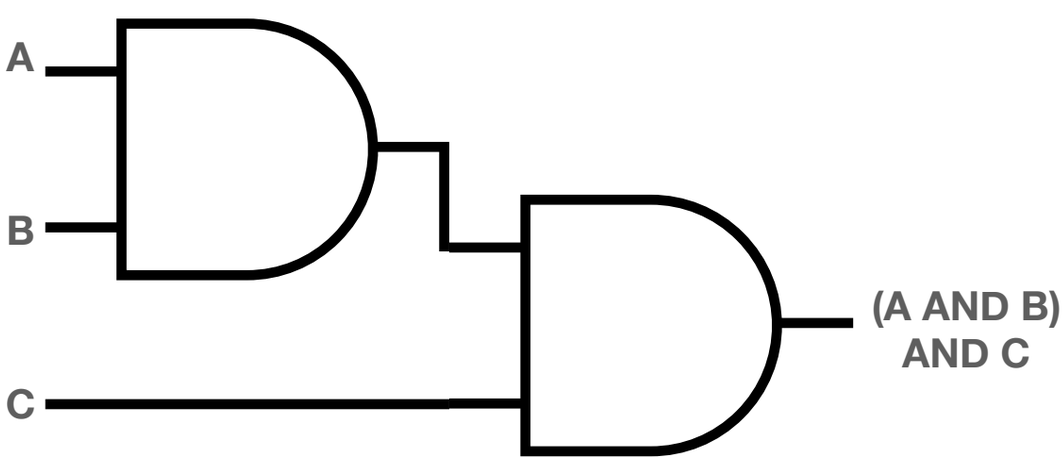
Programmazione e Architetture (Modulo B)

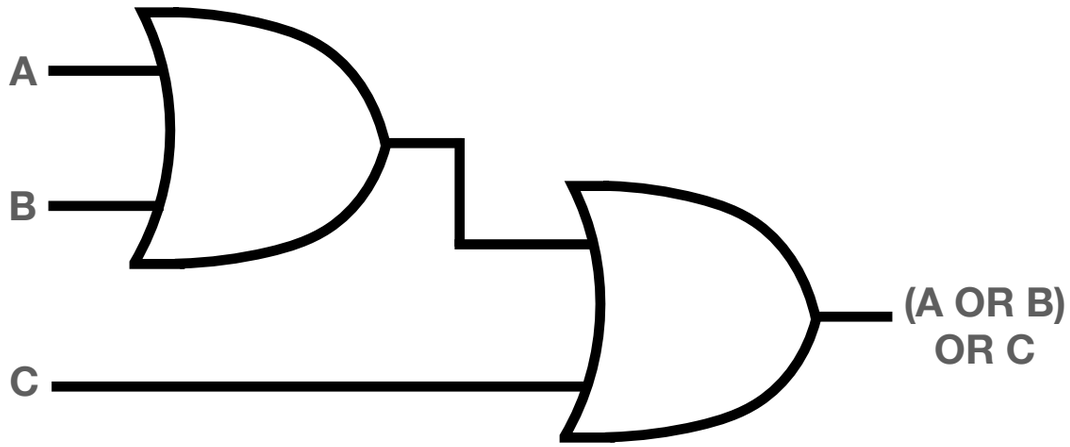
Lezione 3
Adder, multiplexer, demultiplexer

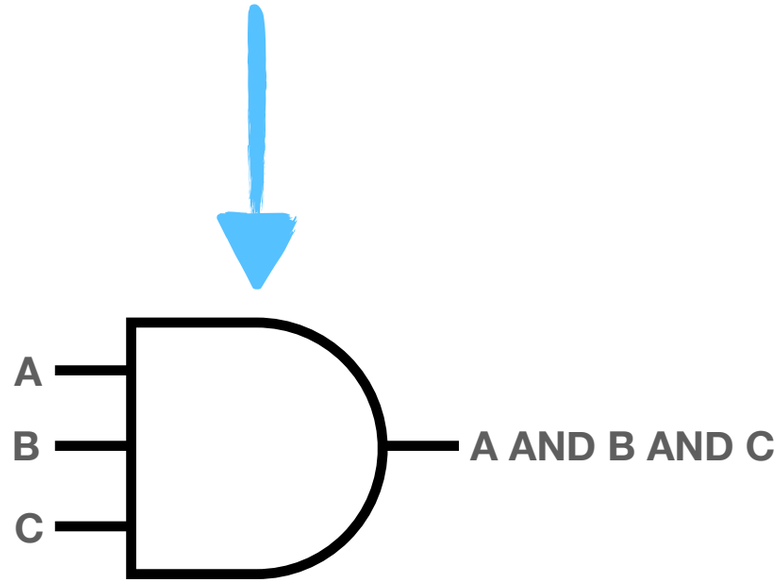
Luca Manzoni

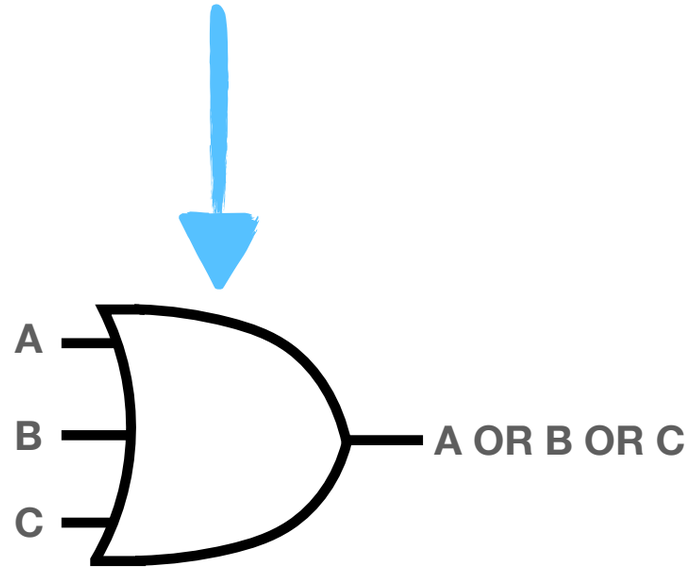
AND e OR con tre input

O un numero maggiore di 3









La stessa costruzione può essere generalizzata per avere un numero arbitrario di bit di input per le porte AND e OR

Circuiti per la somma

Addizione di numeri binari

Iniziamo a costruire una ALU

- Abbiamo visto sia come rappresentare numeri in base due che come funzionano le porte logiche
- Vogliamo ora unire queste nozioni per costruire un circuito composto solamente di porte logiche che ci permetta di effettuare la somma di due numeri (interi) in binario
- Analizziamo cosa succede sommando numeri di un solo bit
- Generalizzeremo poi a numeri composti di un qualsiasi numero di bit (e.g., 8, 16, 32, 64,…)

Half-adder

Somma di due bit

Sommiamo A e B, due numeri di un singolo bit

Input		Output
A	**B**	**Somma**
0	0	0
0	1	1
1	0	1
1	1	10

Unico caso in cui il risultato necessita di due bit. Il bit più significativo è quello che diventerà il riporto (in inglese **carry**)

Separiamo il primo bit dal valore del riporto/carry

Input		Output	
A	**B**	**Somma**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Avremo due output:
La cifra meno significativa e il riporto

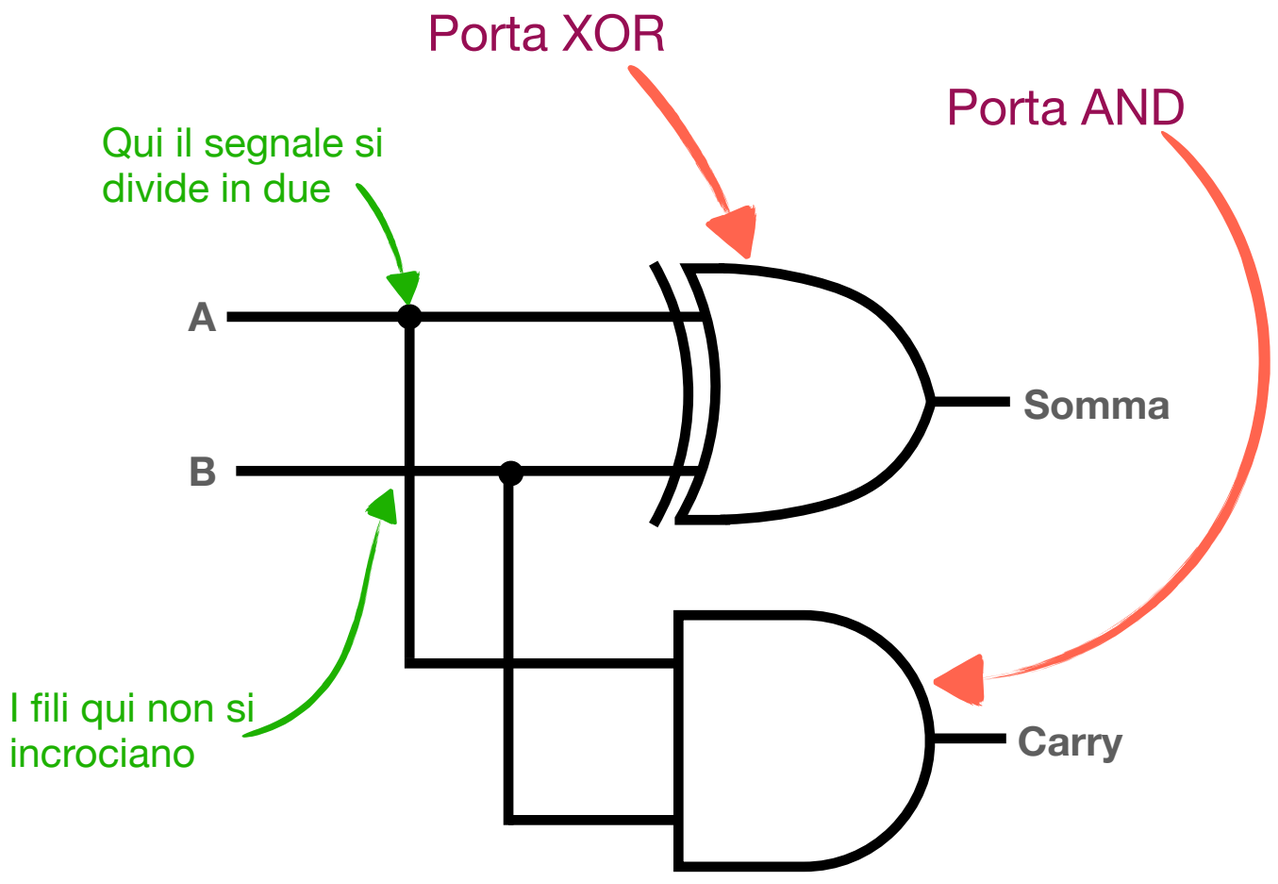
Half-adder

Somma di due bit

Input		Output	
A	B	Somma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Questo è lo stesso risultato dello XOR tra A e B

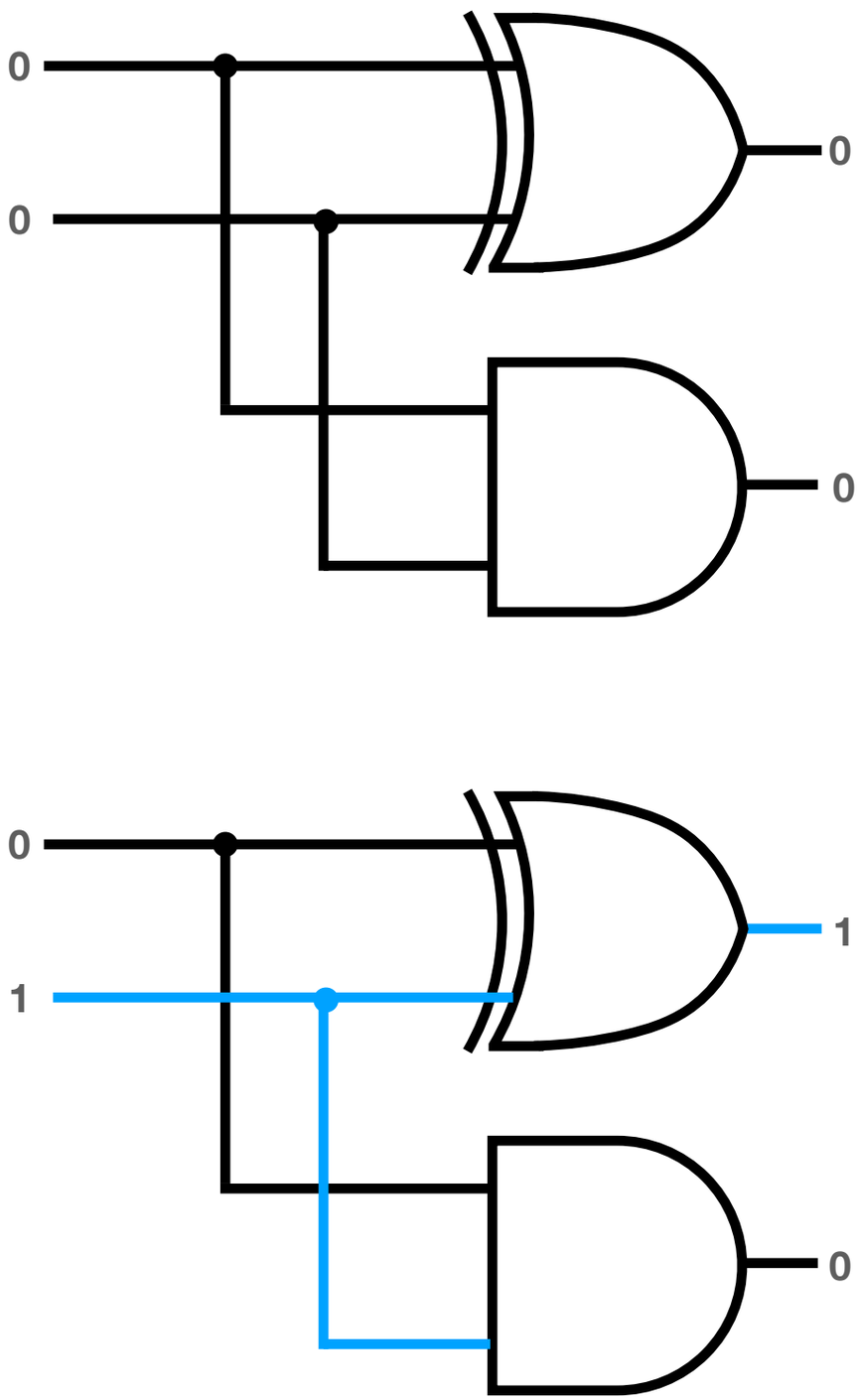
Questo è lo stesso risultato dell'AND tra A e B



Questo circuito è chiamato "**half-adder**". Questo perché per fare la somma "completa" abbiamo bisogno di gestire anche un input addizionale: il bit di riporto proveniente dalla cifra precedente

Half-adder

Verifichiamo che il circuito funzioni



Full adder

Somma di tre bit

Sommiamo A e B, due numeri di un singolo bit, e un terzo bit C_{in}, che rappresenta un bit di carry in input

Input			Output
A	**B**	**C_{in}**	**Somma**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	10
1	0	0	1
1	0	1	10
1	1	0	10
1	1	1	11

Il pattern sembra un poco più complesso

Come prima spezziamo nella somma e nel bit di carry

Input			Output	
A	**B**	**C_{in}**	**Somma**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder

Somma di tre bit

Serve un circuito che verifichi la parità dei bit di input

Notiamo come il bit di somma è uno solamente quando il numero di bit di input è dispari

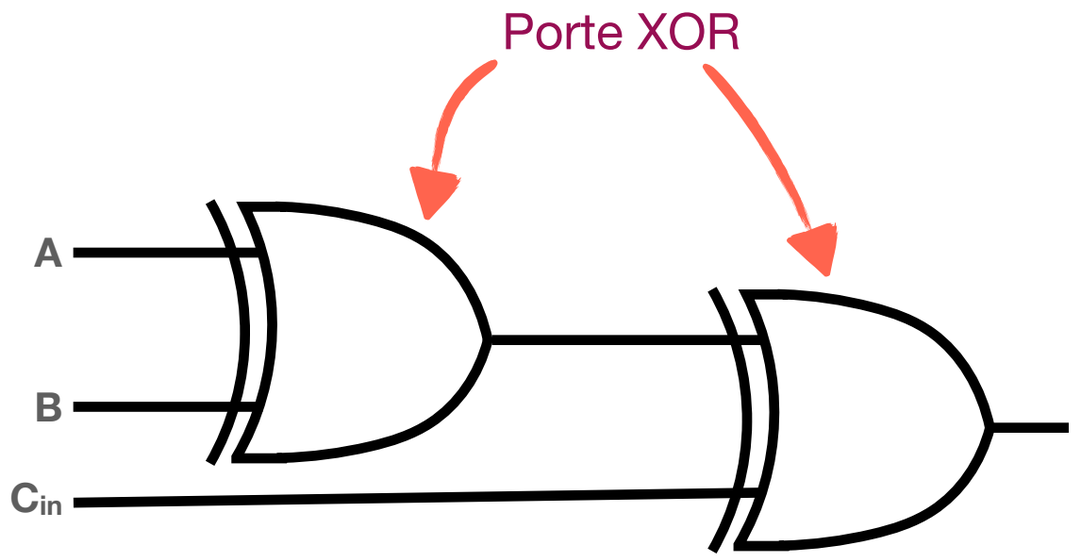
Mentre il bit di carry è uno solamente quando ci sono due o più bit di input a uno

Serve un circuito a soglia (i.e., 1 solo quando il numero di 1 in input supera una soglia)

Input			Output	
A	**B**	**C_{in}**	**Somma**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder

Parità



Questo è verificato quando ci sono esattamente 1 o 3 bit con valore 1

Input			Output
A	B	C_{in}	XOR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A XOR B è 1 quando esattamente uno tra A e B è uno

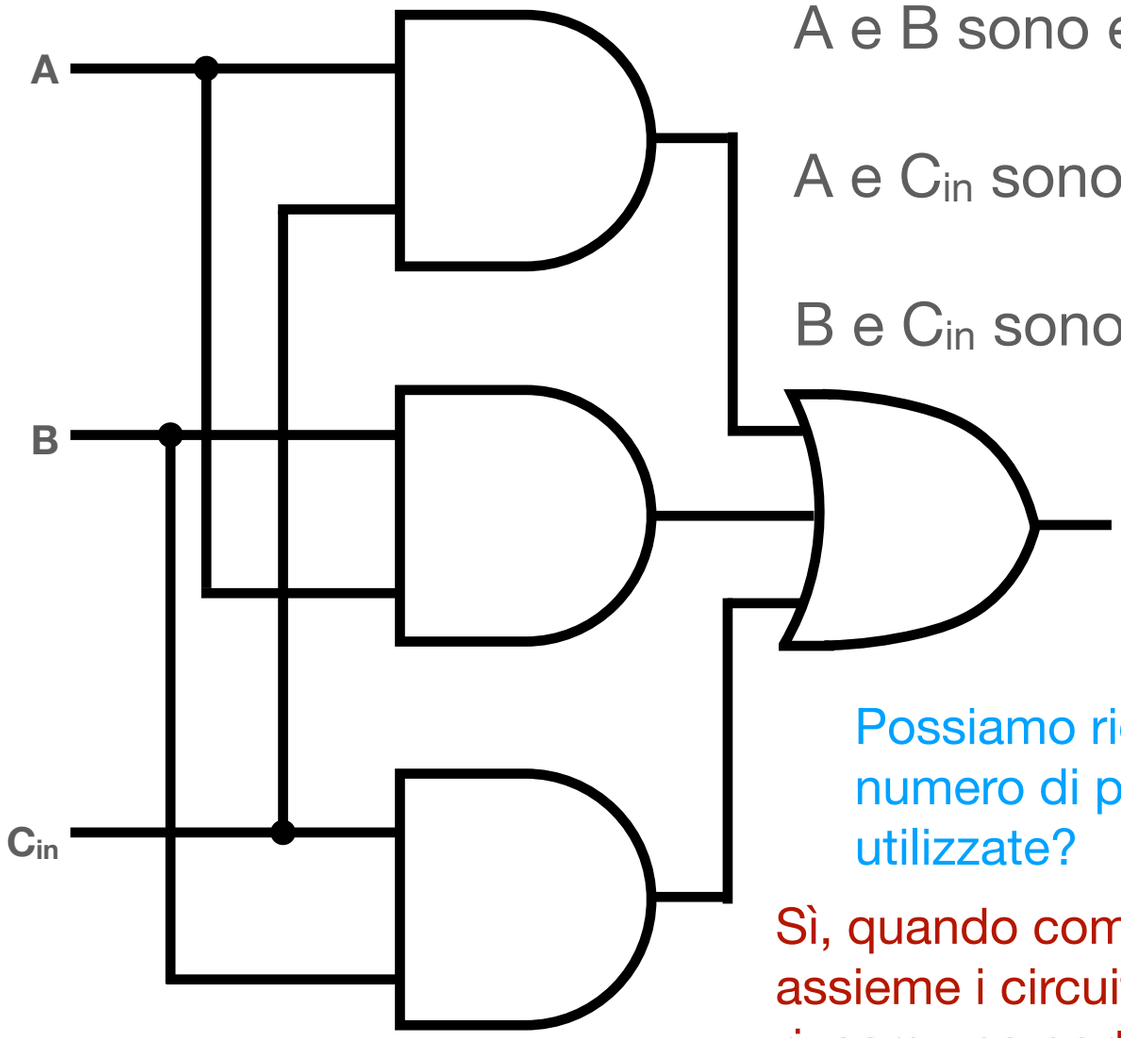
(A XOR B) XOR C è 1 quando esattamente uno tra C e A XOR B è 1. Ovvero:

Se C è 1 allora A XOR B deve essere 0 (entrambi zero o entrambi uno)

Se C è 0 allora A XOR B deve essere 1 (esattamente uno dei due è uno)

Full adder

Circuito a soglia



L'output è 1 se almeno una delle seguenti condizioni è verificata:

A e B sono entrambi 1

A e C_{in} sono entrambi 1

B e C_{in} sono entrambi 1

Possiamo ridurre il numero di porte logiche utilizzate?

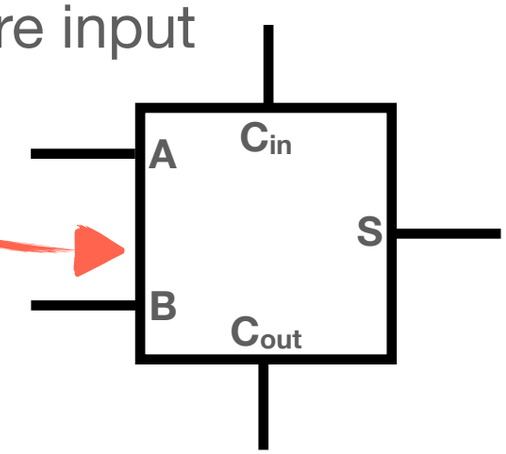
Sì, quando combiniamo assieme i circuiti possiamo riusare una porta XOR e rimuovere una porta AND

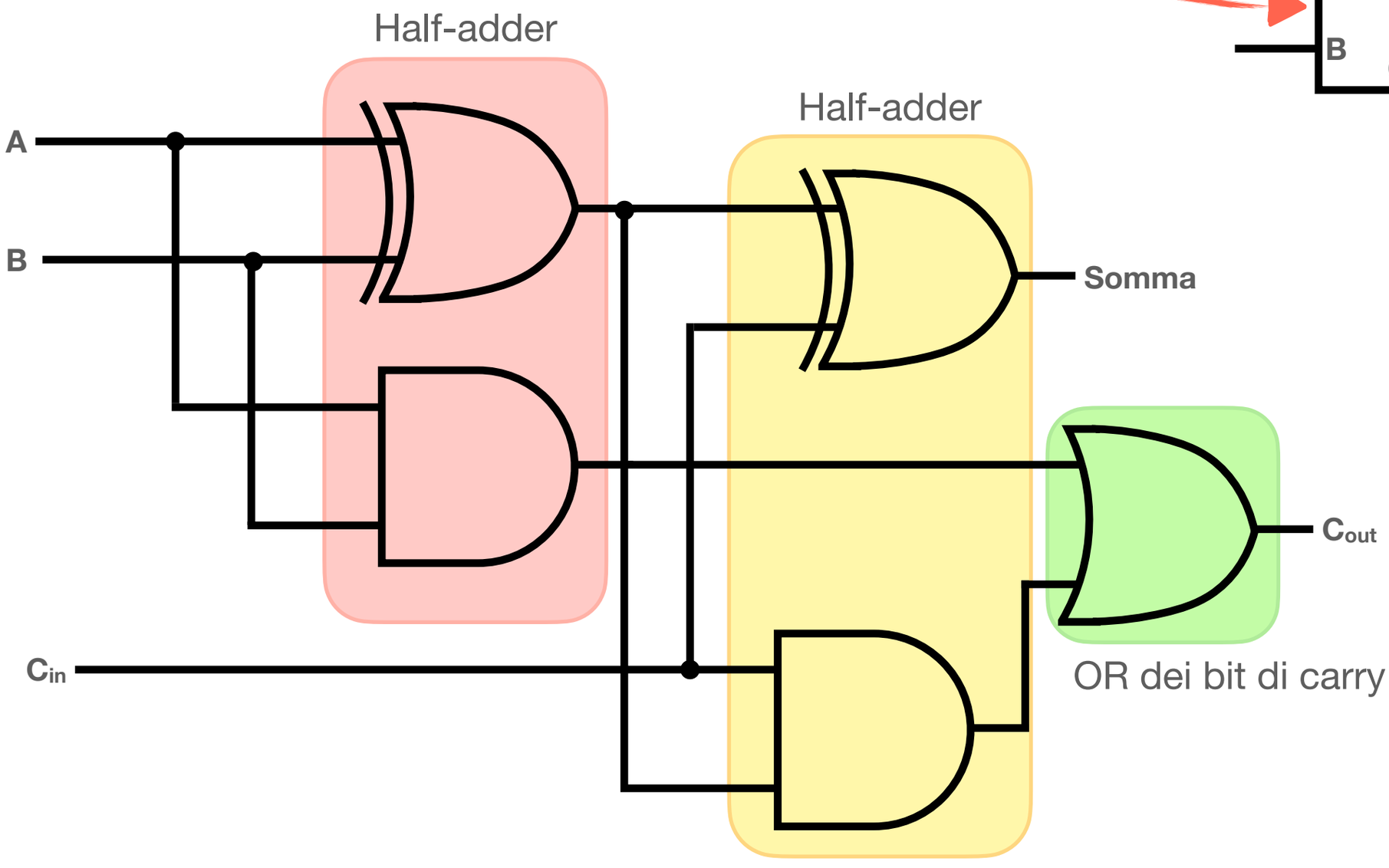
Input			Output
A	**B**	**C_{in}**	**≥2**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Full adder

Somma di tre bit

Possiamo ora usare il full adder
come un blocco unico con tre input
e due output





Ripple adder

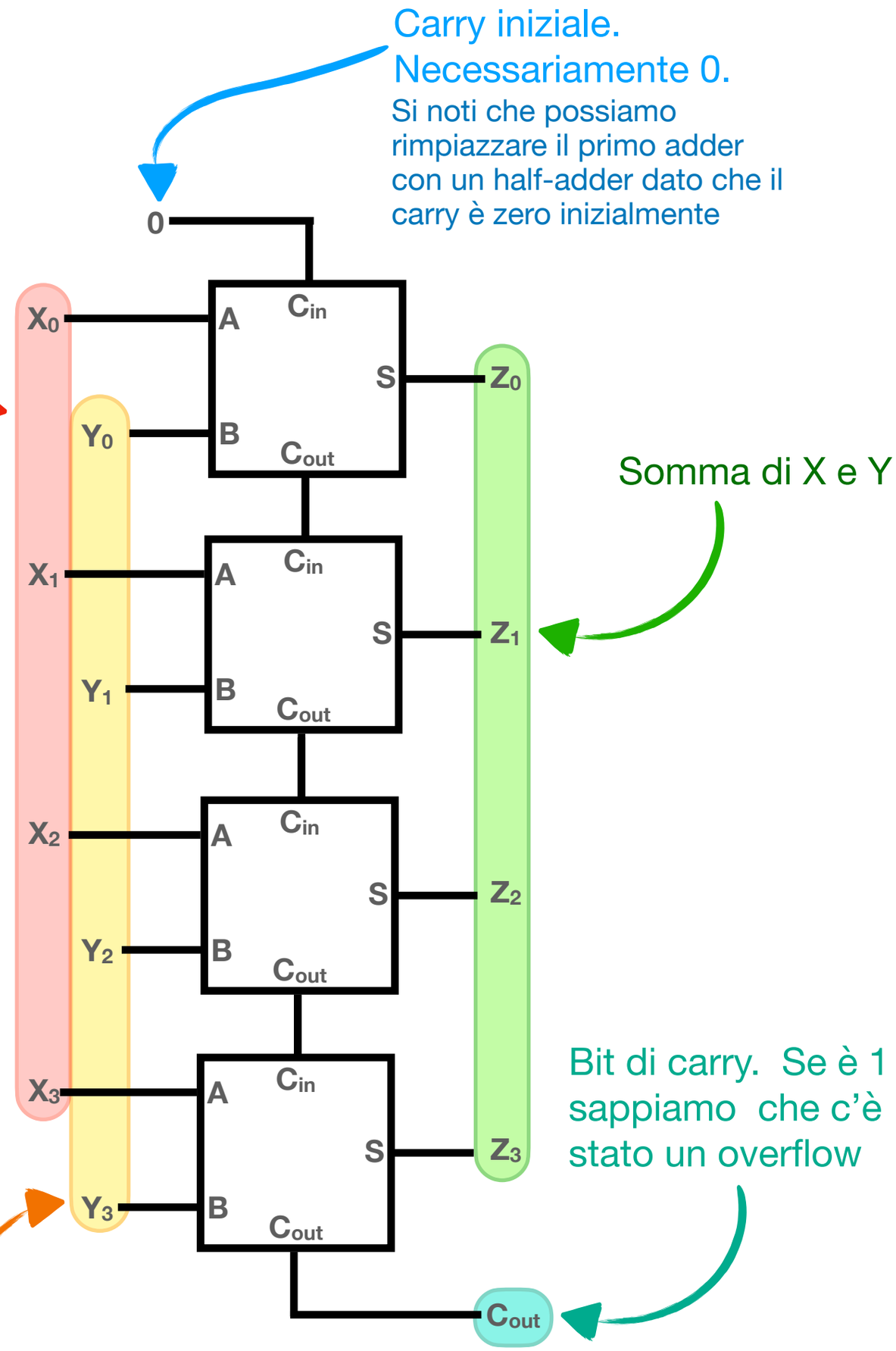
Estensione a più bit

Possiamo concatenare adder per ottenere un circuito in grado di fare addizioni di un numero qualsiasi di bit.

Ci basta fare in modo che il bit di carry di output di un adder sia collegato al bit di carry di input del successivo.

Quello che otteniamo è un **ripple adder**, dato che il carry si muove come "un'onda" dalle cifre meno significative a quelle più significative.

Esistono adder più efficienti, ma per i nostri scopi il ripple adder è sufficiente.



Multiplexer

Selezionare l'informazione

Principi e uso dei multiplexer

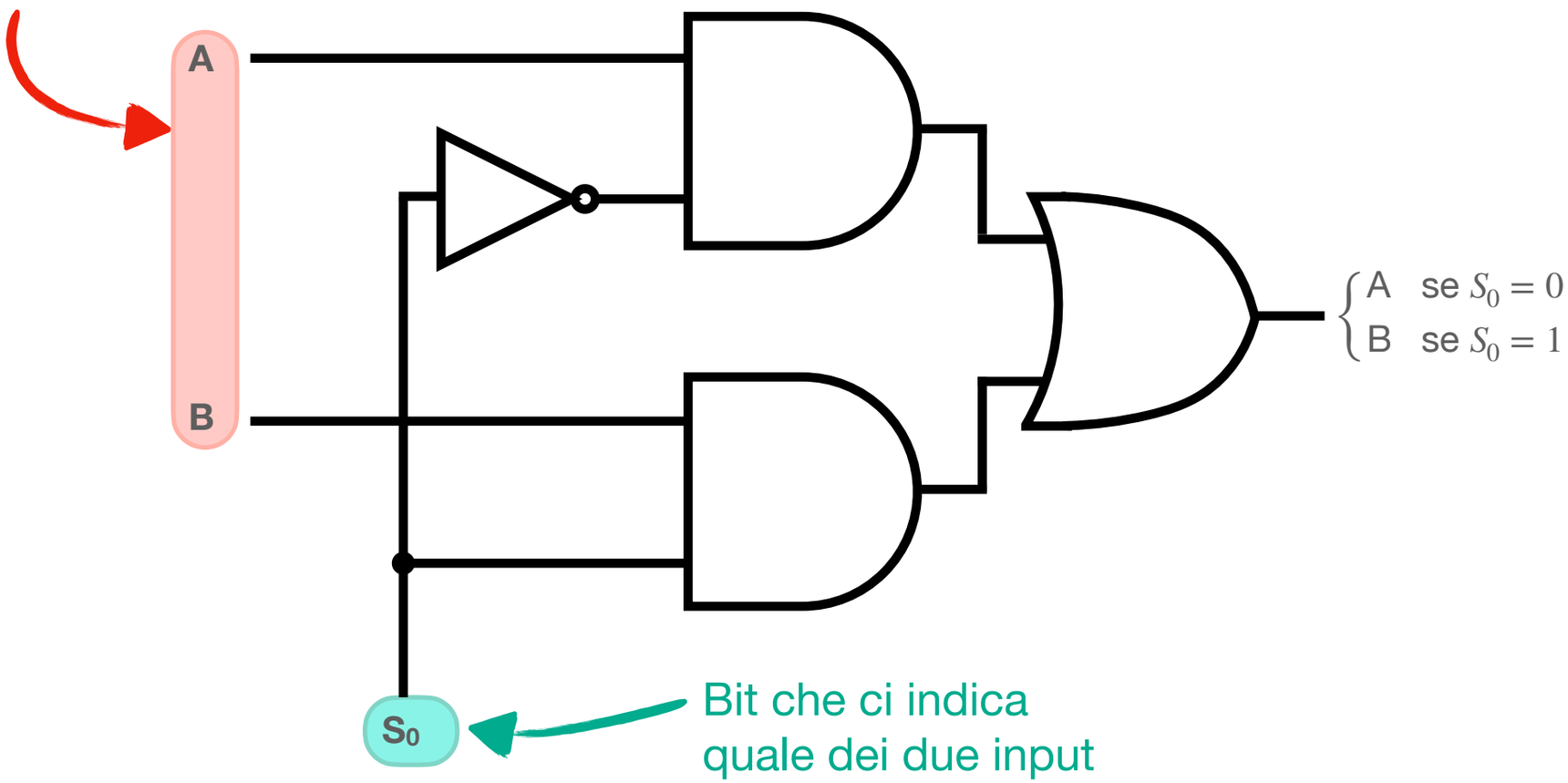
- Un multiplexer (selettore) è utilizzato per selezionare un input tra molti
- Supponiamo di dover scegliere tra $n = 2^m$ input (i.e., n è una potenza di due)
- Per esprimere quale degli n input scegliere abbiamo bisogno di $\log_2 n = \log_2 2^m = m$ bit addizionali per esprimere quale input dobbiamo selezionare (i.e., un "indirizzo" di quale input scegliere)
- Generalmente un multiplexer ha quindi $n + \log_2 n$ input e un singolo output

Multiplexer

Selezionare un input specifico

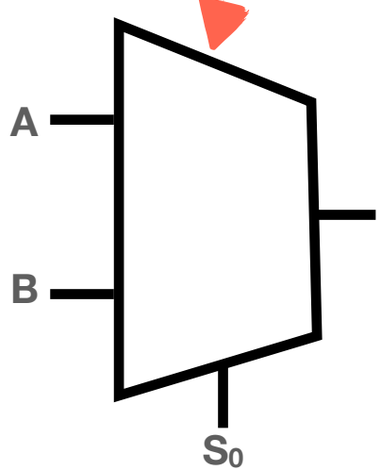
Possiamo usare un singolo bit per scegliere tra due diversi bit di output

Due bit e ne dobbiamo selezionare uno

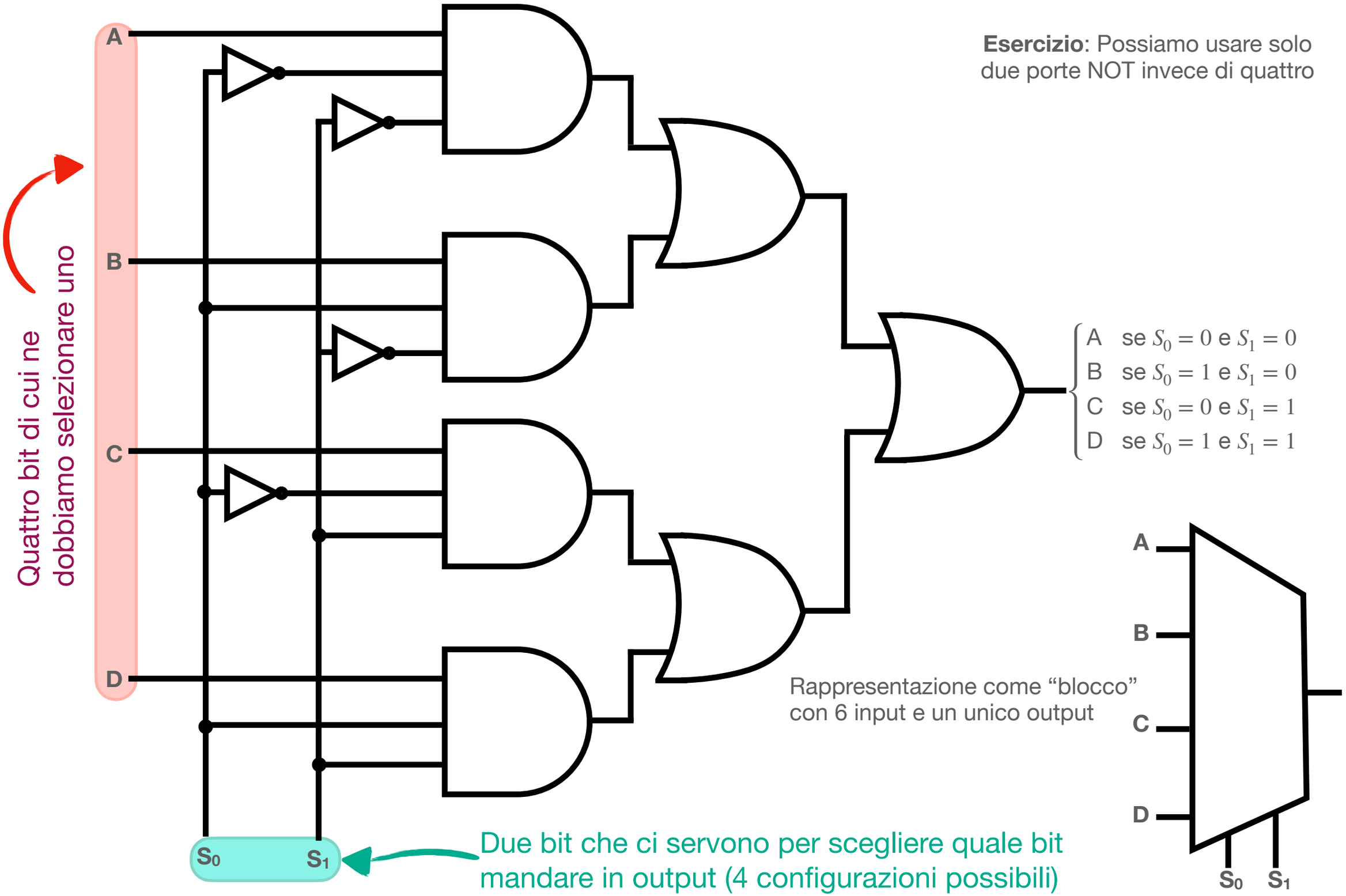


Bit che ci indica quale dei due input selezionare

Possiamo rappresentare un multiplexer 2-to-1 come un “blocco” con tre input e un singolo output



Multiplexer 4-to-1



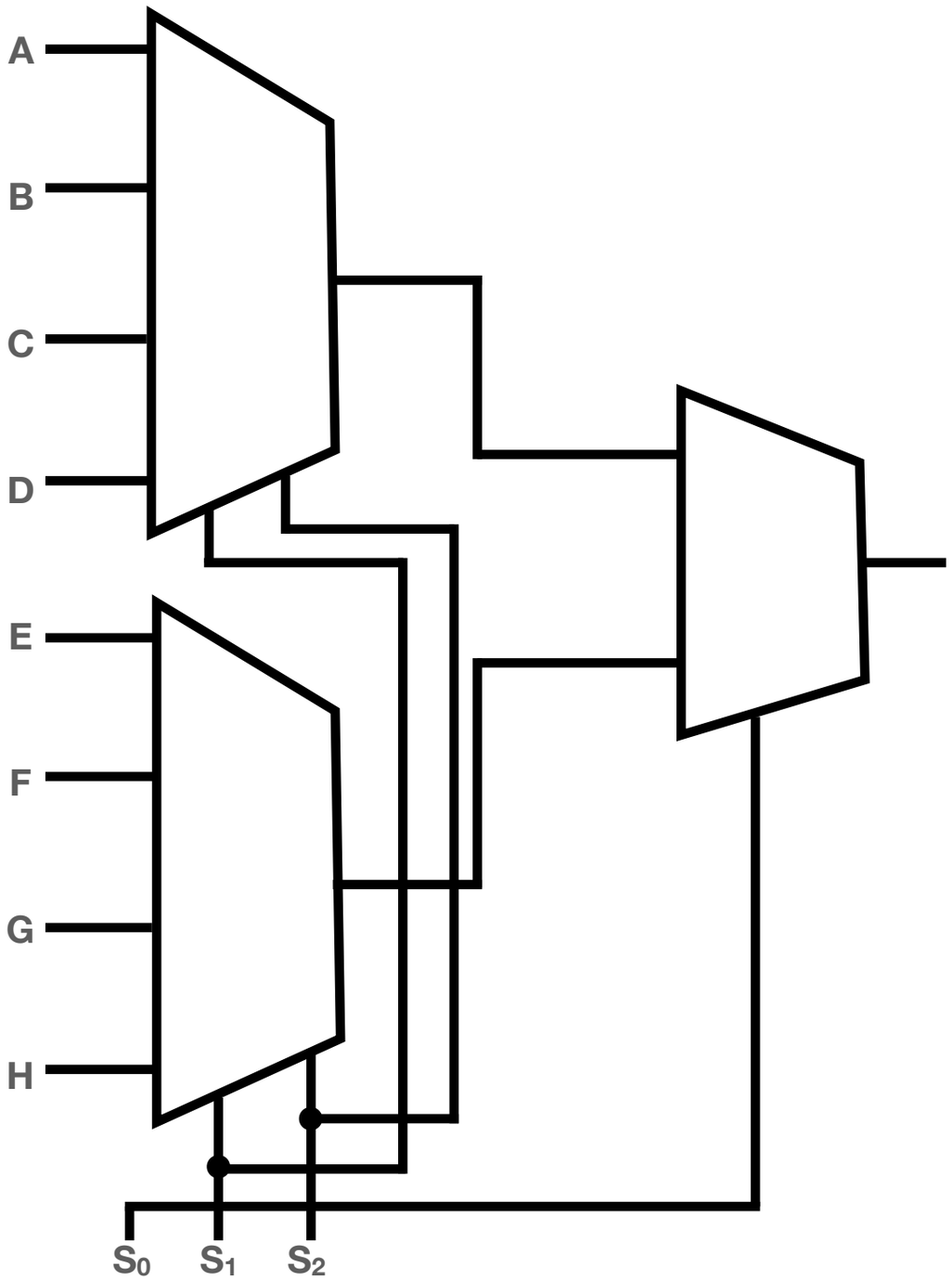
Esercizio: Possiamo usare solo due porte NOT invece di quattro

- A se $S_0 = 0$ e $S_1 = 0$
- B se $S_0 = 1$ e $S_1 = 0$
- C se $S_0 = 0$ e $S_1 = 1$
- D se $S_0 = 1$ e $S_1 = 1$

Rappresentazione come “blocco” con 6 input e un unico output

Multiplexer 8-to-1

E combinare multiplexer più piccoli



In generale possiamo combinare multiplexer più piccoli per generare multiplexer di dimensioni maggiori

8-to-1 multiplexer, con 11 input (dato che $8 + \log_2 8 = 11$) e un singolo output

16-to-1 multiplexer, usando tre 4-to-1 multiplexer, con 20 input ($16 + \log_2 16 = 20$)

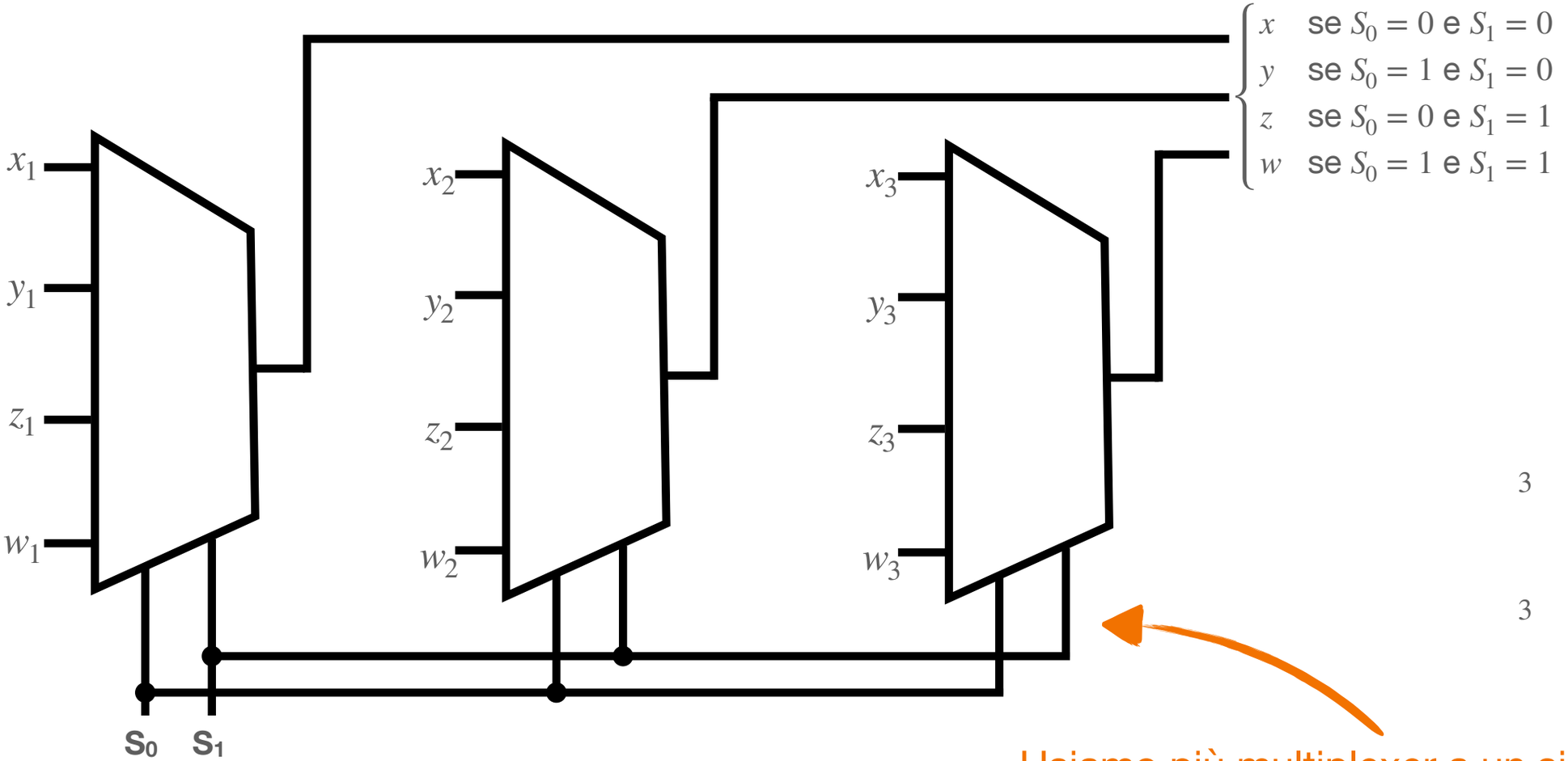
E via così…

Ma se invece di un singolo bit in output ne volessimo avere multipli?

Multiplexer con output multipli

Combinare le linee di indirizzo

Supponiamo di voler scegliere tra quattro sequenze di tre bit ciascuna:
$x = \langle x_1, x_2, x_3 \rangle$, $y = \langle y_1, y_2, y_3 \rangle$, $z = \langle z_1, z_2, z_3 \rangle$, $w = \langle w_1, w_2, w_3 \rangle$



Usiamo più multiplexer a un singolo bit collegati alle stesse linee di selezione per ottenere multiplexer che selezionano sequenze di bit

Decoder e Demultiplexer

Decoder Binario

Da un valore all'attivare uno specifico output

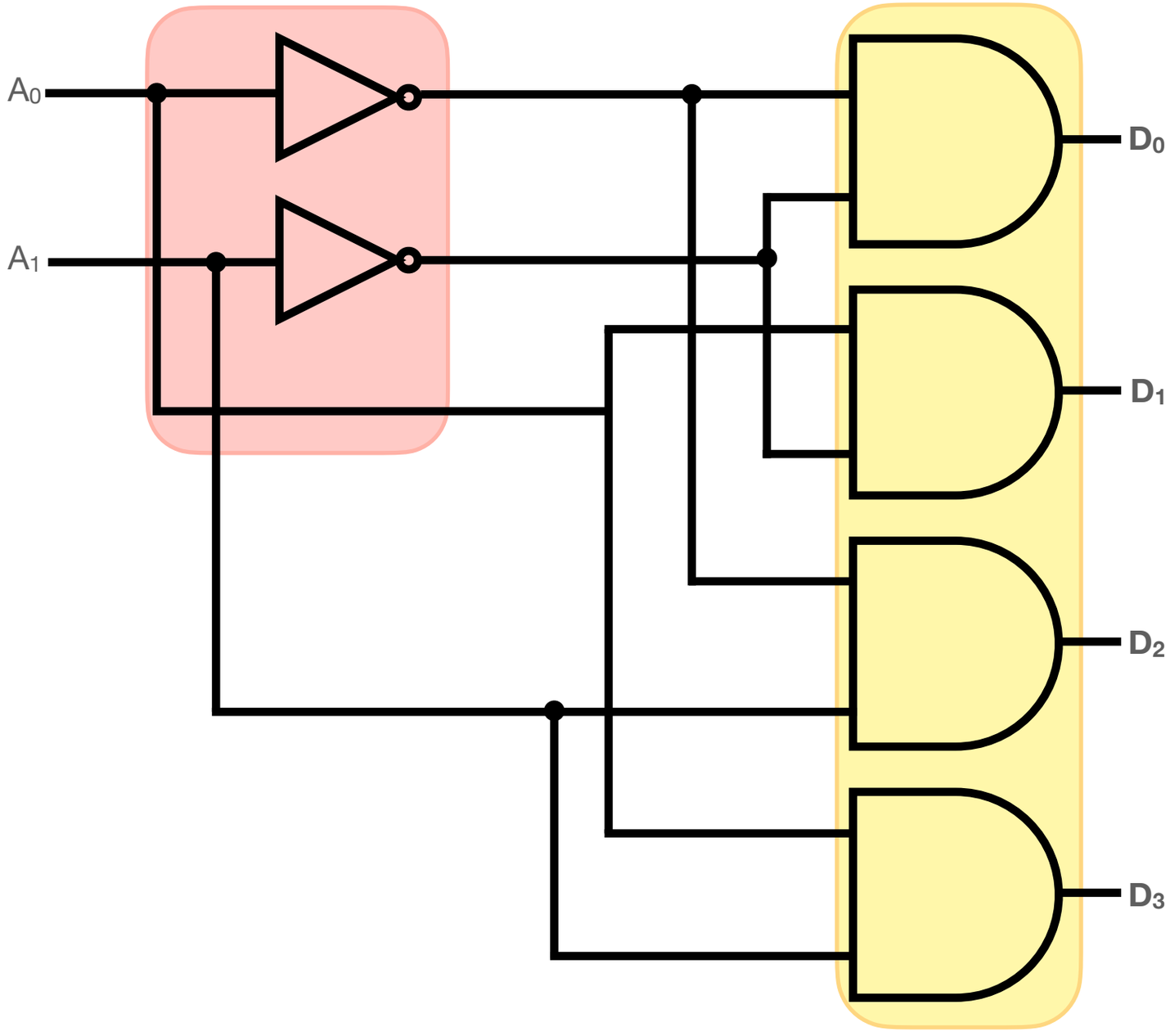
- Dati n input che ci rappresentano un numero binario
- E 2^n output (che possiamo pensare numerati da 0 a $2^n - 1$)
- Vogliamo mettere a **uno** solo l'output corrispondente al numero binario in input
- Per esempio se abbiamo 3 input e $2^3 = 8$ output e otteniamo come input 101_2, corrispondente a 5 in base 10, vogliamo mettere a uno l'output numero 5 e tutti gli altri a zero
- L'output è "one-hot", ovvero esattamente un bit è a uno e tutti gli altri sono a zero

Decoder Binario

Da un valore all'attivare uno specifico output

Per ogni input abbiamo sia l'input che la sua versione negata

Mettendo in AND tutte le possibili combinazioni otteniamo che solo un output sarà a uno



Input		Output
A	B	(4 bit)
0	0	1000
0	1	0100
1	0	0010
1	1	0001

Esattamente un solo bit a uno

Demultiplexer

Dirigere un input verso uno tra tanti output

- Nel multiplexer abbiamo molteplici input e un unico output
- Il demultiplexer invece ha un unico input e molteplici output e l'input sarà "ridiretto" verso uno di quelli in base a degli input usati come "selettori"
- Per questo useremo il decoder binario che abbiamo appena introdotto
- Possiamo usare il decoder binario anche per costruire dei multiplexer

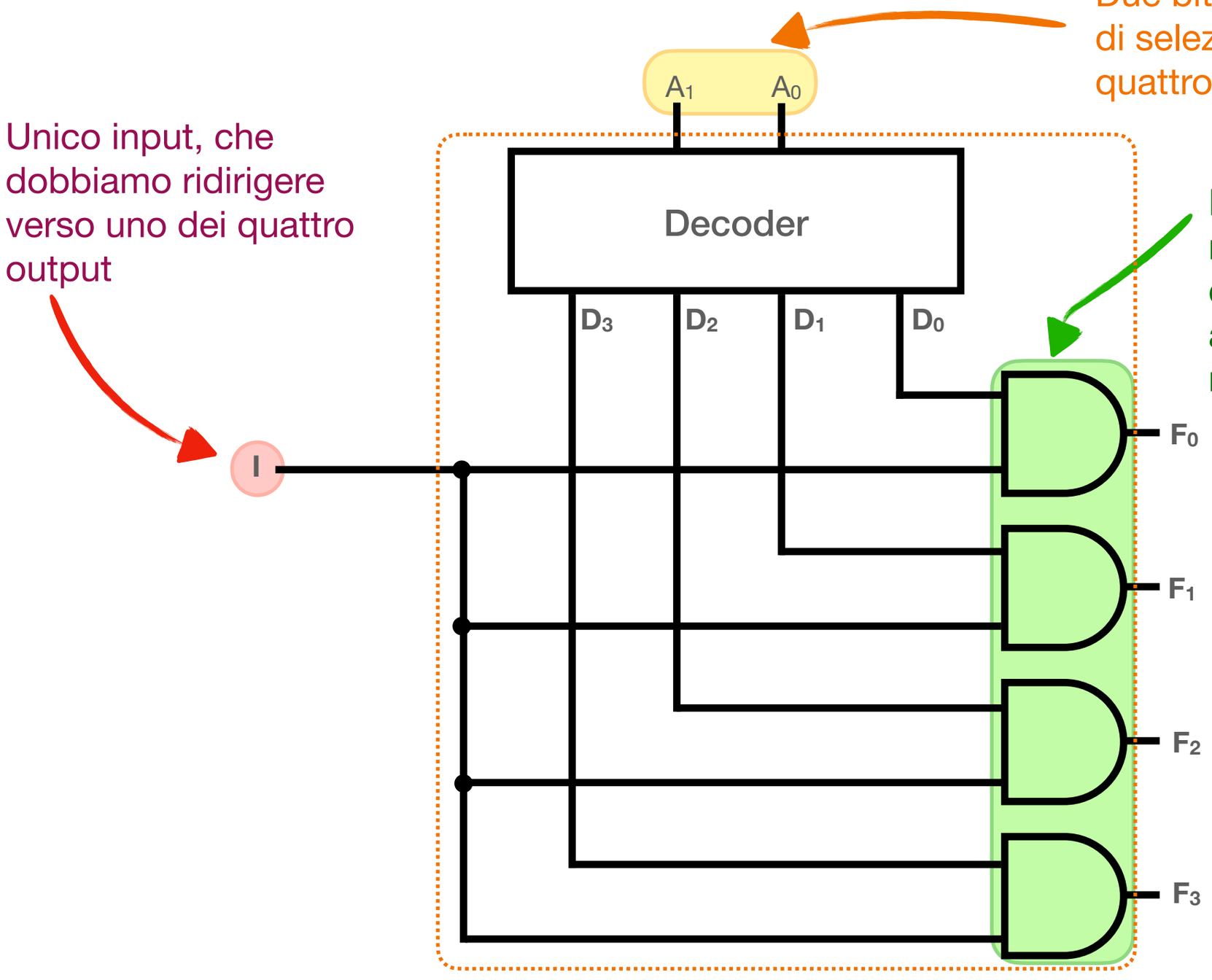
Demultiplexer

Attivare una specifica linea di output

Due bit ci permettono di selezionare tra quattro output distinti

Unico input, che dobbiamo ridirigere verso uno dei quattro output

L'uso di porte AND fa in modo che tutti gli output non corrispondenti alla linea attivata dal decoder siano necessariamente zero



Rappresentazione come "blocco" con 3 input e 4 output