Sommatore

- Dati due numeri naturali rappresentati su un solo bit, e il riporto della somma precedente, il circuito sommatore ne calcola la somma (compreso il riporto). Cioè in pratica somma i tre bit.

Tavola delle verità

A	B	Cin	S	Cout
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

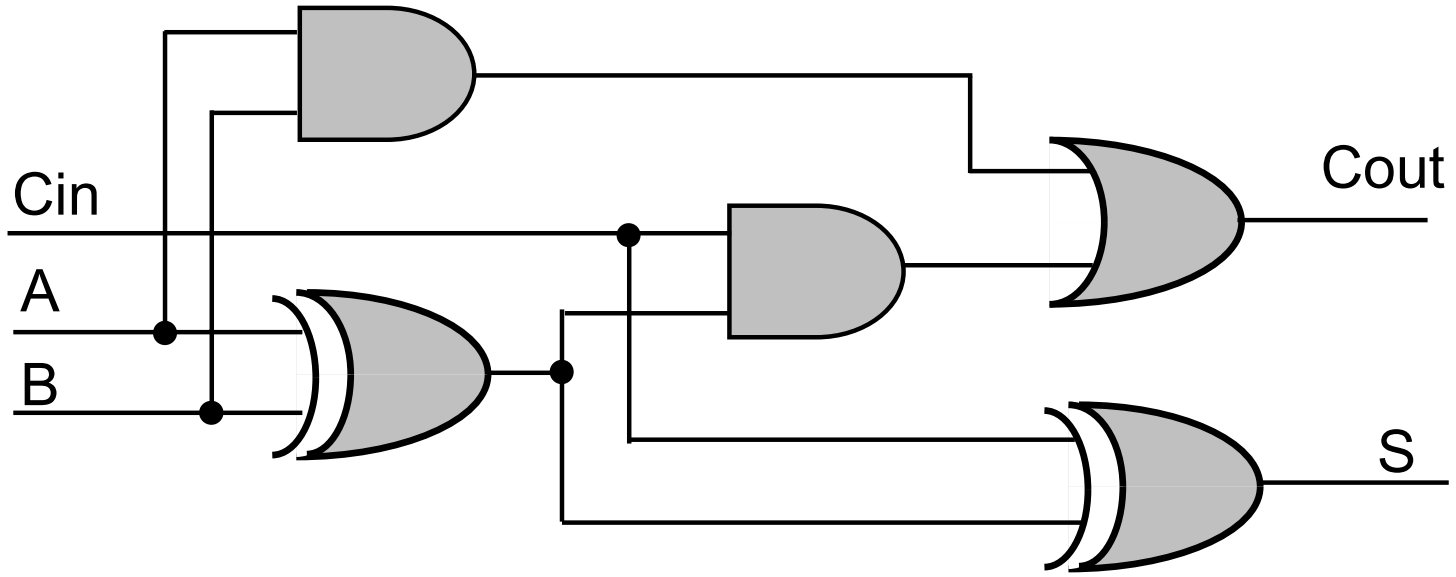
Cout =
/ABCin + A/BCin + AB/Cin + ABCin =
/ABCin + A/BCin + AB(/Cin + Cin) =
/ABCin + A/Bcin + AB =
AB + Cin (/AB+A/B) =
AB + Cin (A⊕B)

S= A ⊕ B ⊕ Cin

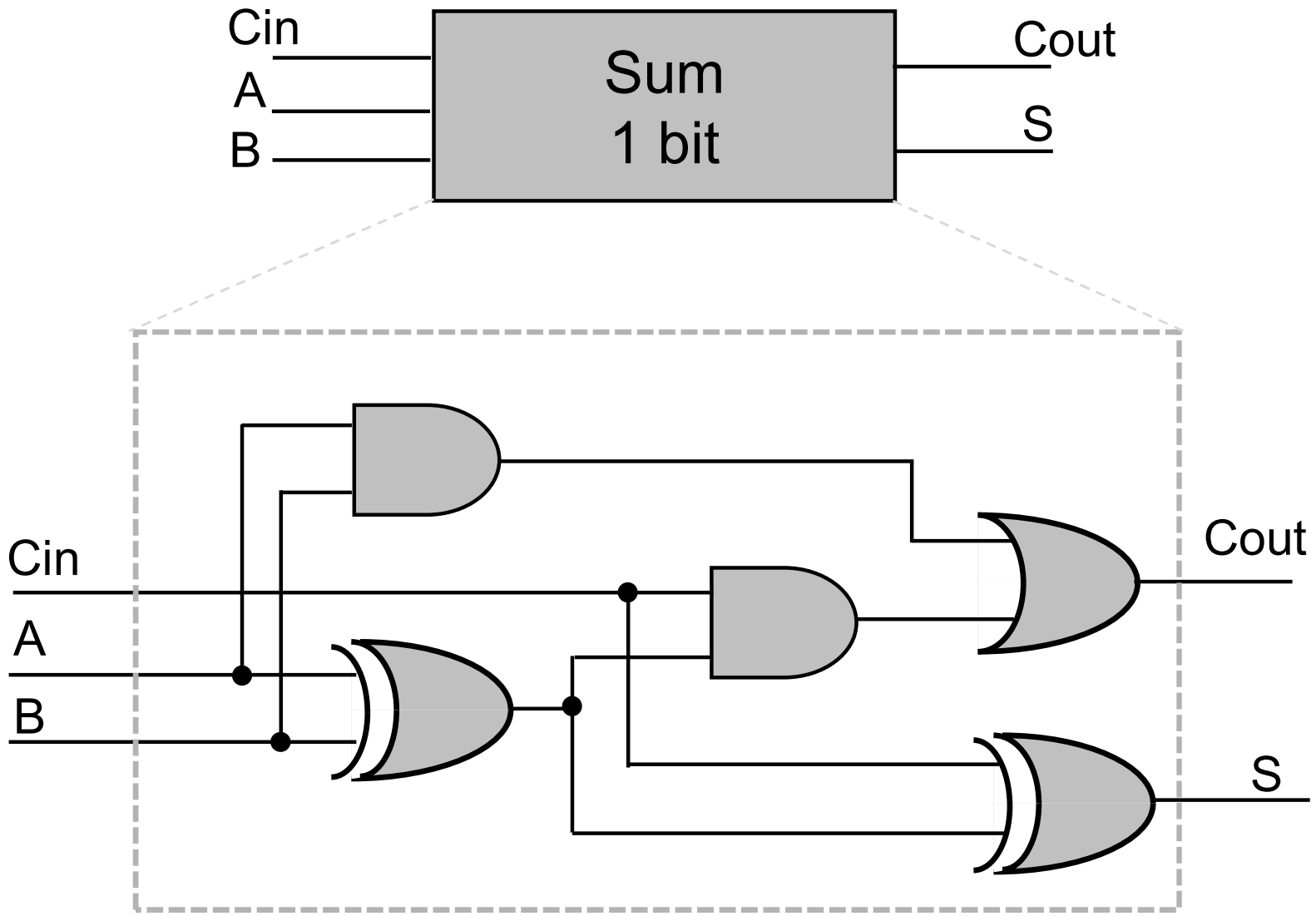
xor

Sommatore

- Cout = AB + /ABCin + A/BCin =
  AB + Cin (/AB+A/B) =
  AB + Cin (A⊕B)
- S= A ⊕B ⊕Cin



Sommatore



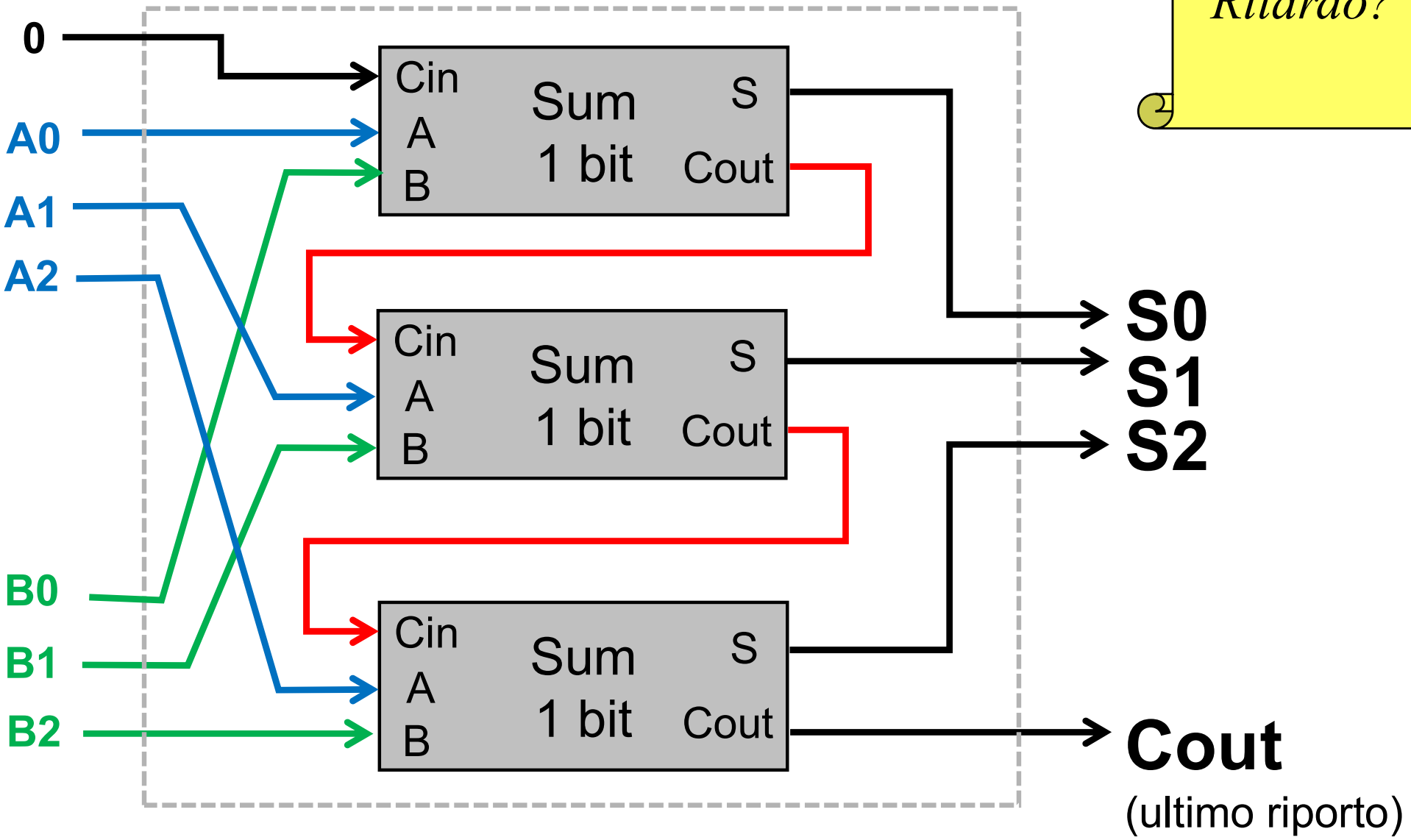
Sommatore binario naturale a n bit (n-bit adder)

- È la generalizzazione del sommatore completo: addizione di numeri naturali binari a n bit
- Ha in ingresso due numeri naturali A e B da $n \geq 1$ bit ciascuno
- In uscita presenta la somma a n bit dei due numeri interi A e B
- Tipicamente: ha un riporto in ingresso e un riporto in uscita
  - In uscita presenta A + B + (Riporto in ingresso)

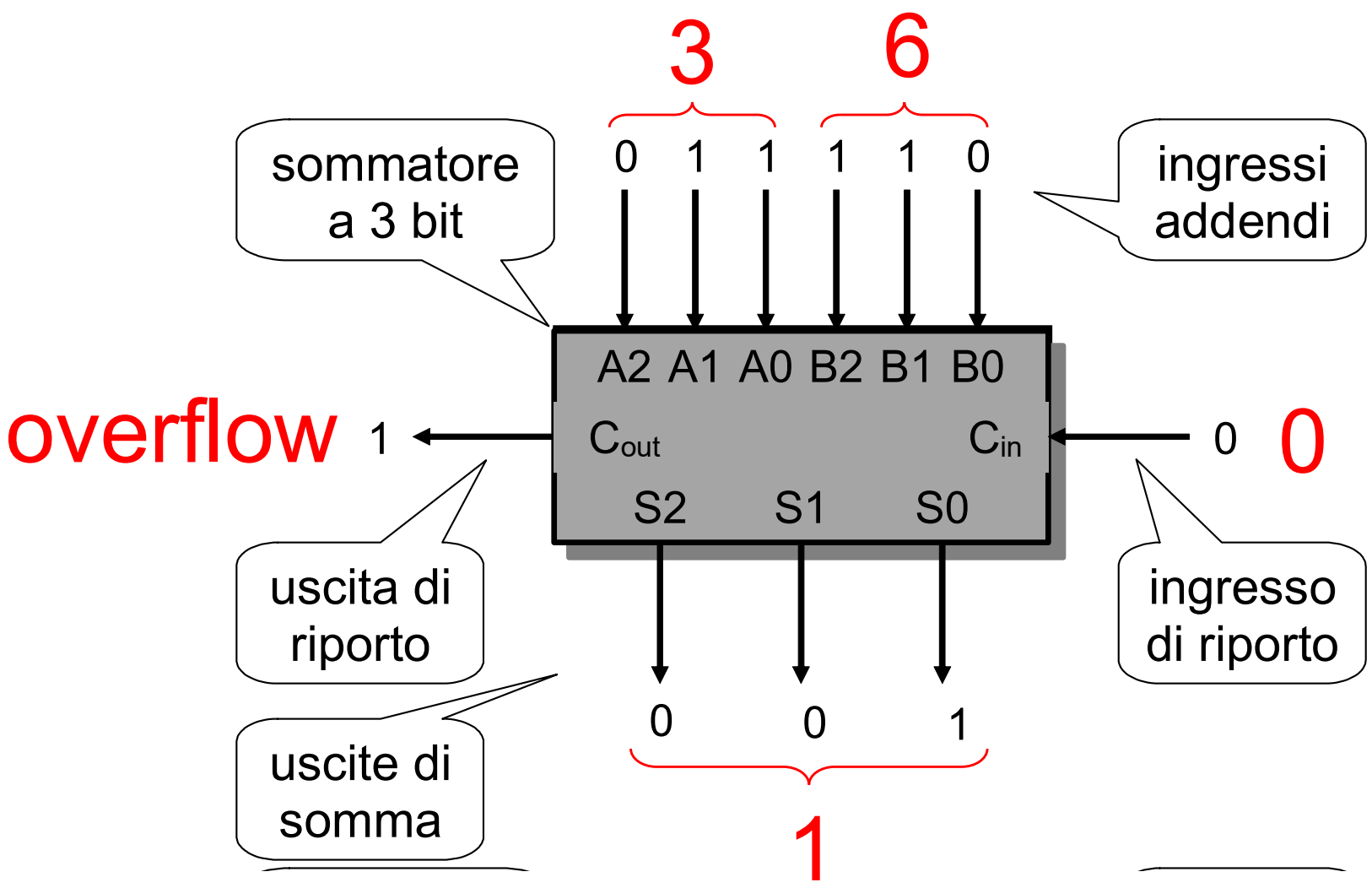
- Come possiamo realizzarlo?

Sommatore a 3 bit: realizzazione

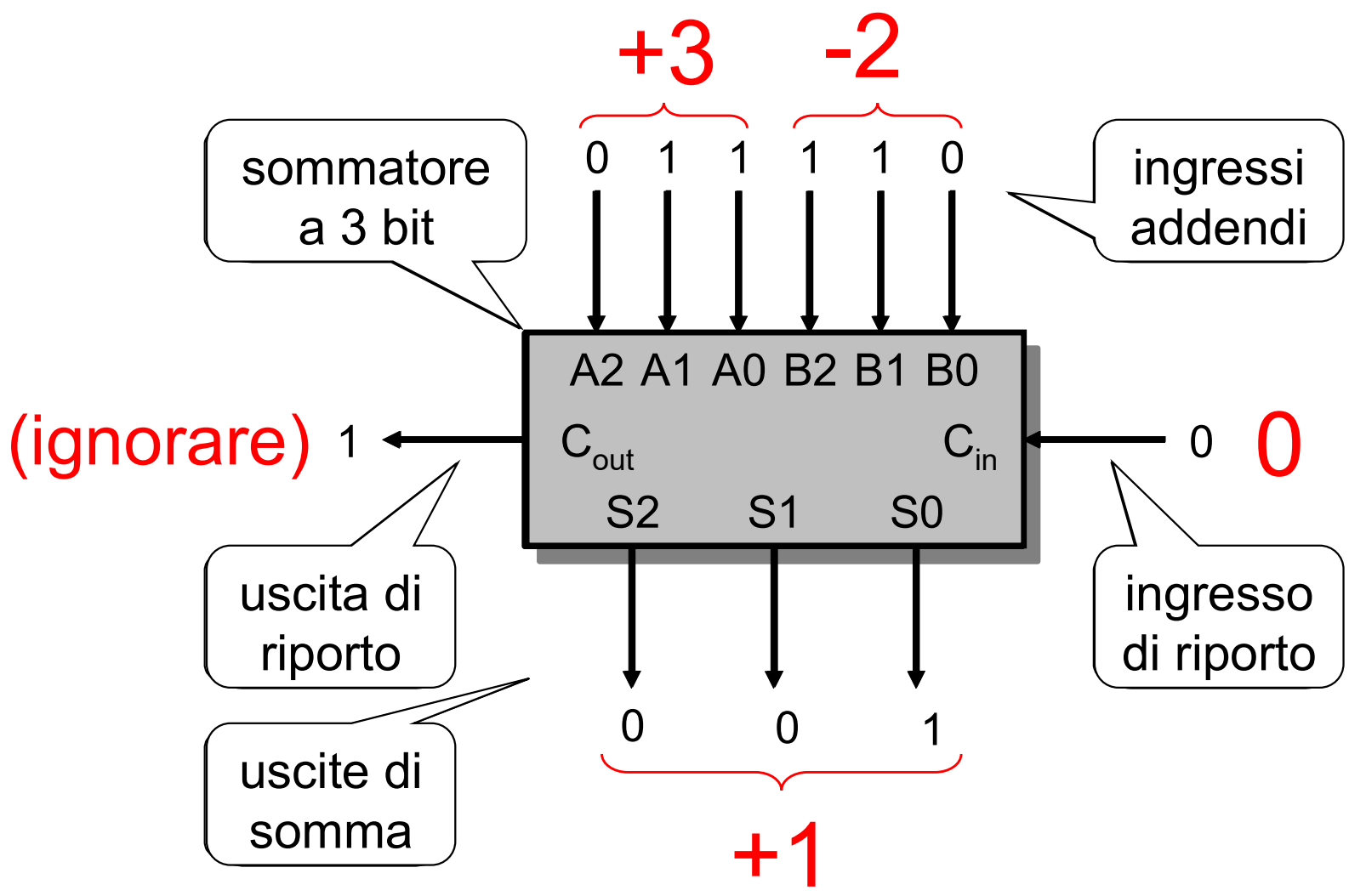
Ritardo?



Sommatore intero binario naturale a 3 bit



Lo *stesso* circuito, ma interpretato come sommatore di **interi** a 3 bit in **CP2**



Sommatore: Overflow

Coi numeri naturali (senza segno):

- l'uscita finale Cout (l'ultimo riporto) è un bit che segnala l'overflow

Con i numeri in complemento a 2:

- L'uscita finale Cout non segnala overflow
  - (va semplicemente ignorata)
- L'overflow è segnalato invece da una funzione dei bit più significativi di A, B e S
  - (cioè quelli che rappresentano i segni dei tre numeri A B e S)
  - Regola, per 8 bit:
    se A_7 == B_7 ma =/= S_7 allora overflow
  - *Esercizio (facile):*
    costruire un circuito che restituisce 1 se c'è stato overflow

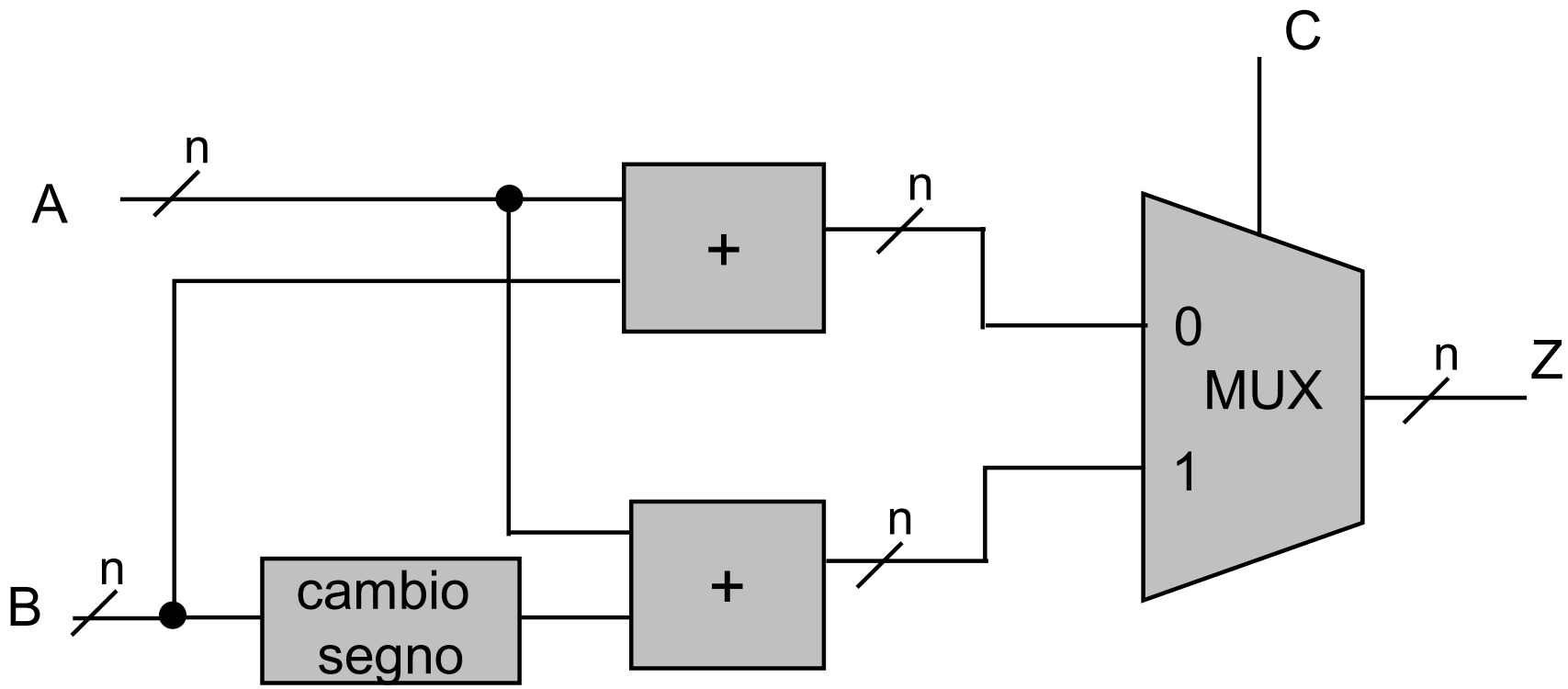
Semplice esempio di progetto con blocchi funzionali

- Si chiede di progettare un circuito digitale combinatorio, che abbia:
  - in ingresso due numeri binari naturali A e B da *n* bit ciascuno
  - in ingresso un segnale di comando C da un bit
  - in uscita un numero binario naturale Z da *n* bit tale che
    - Z = A + B , quando C = 0
    - Z = A – B , quando C = 1
- Si trascurano i riporti

- In pratica:
  - C rappresenta il comando (ed A e B i suoi operatori), in un ipotetico, minuscolo **Instruction Set** di una ALU composto da due sole istruzioni: { somma , sottrazione }
  - il circuito è una specie di una minuscola ALU capace di due sole op.
  - è il nostro primo passo da una calcolatrice verso un calcolatore

Soluzione 1

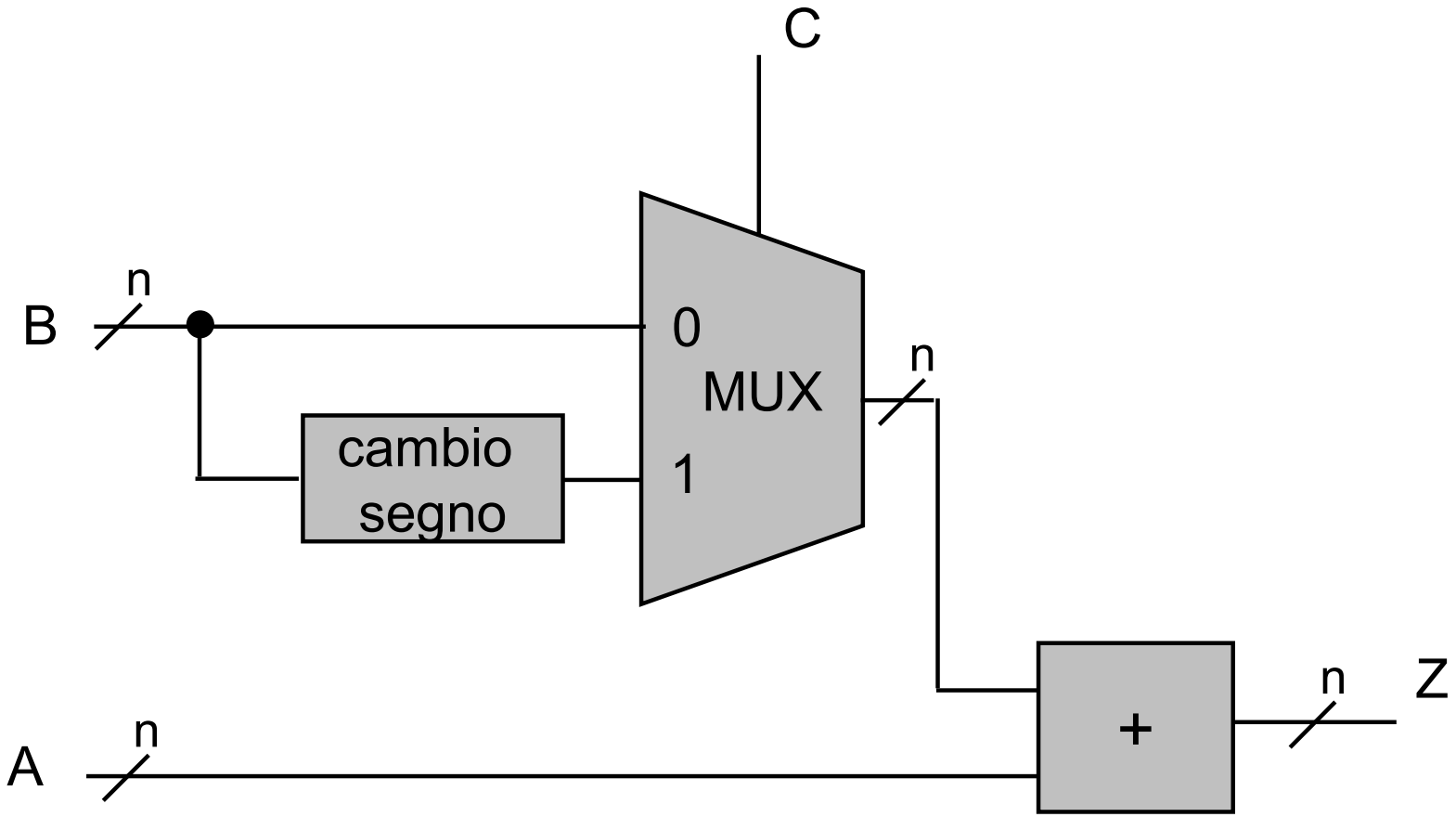
- Idea: usare un sommatore anche per la sottrazione… A – B = A + (– B)



Altre possibilità?

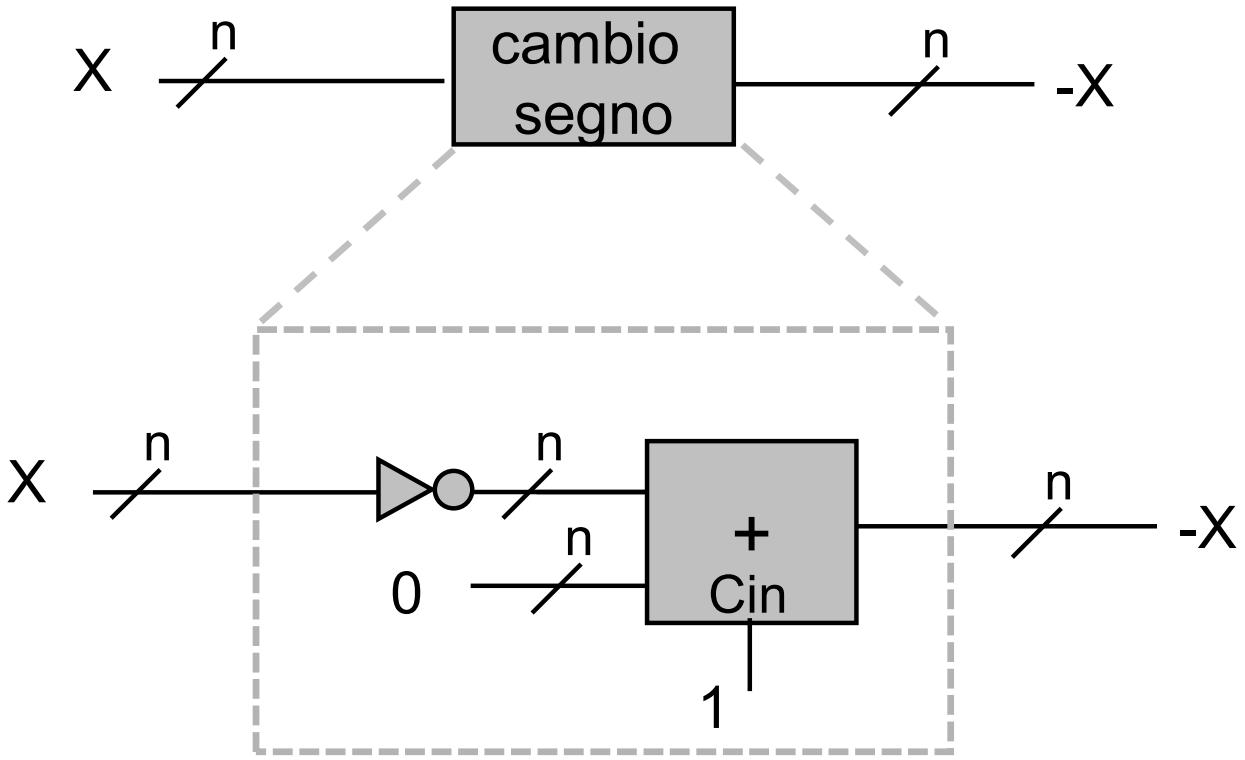
Soluzione 2

- Meno costosa della precedente: abbiamo risparmiato un sommatore!

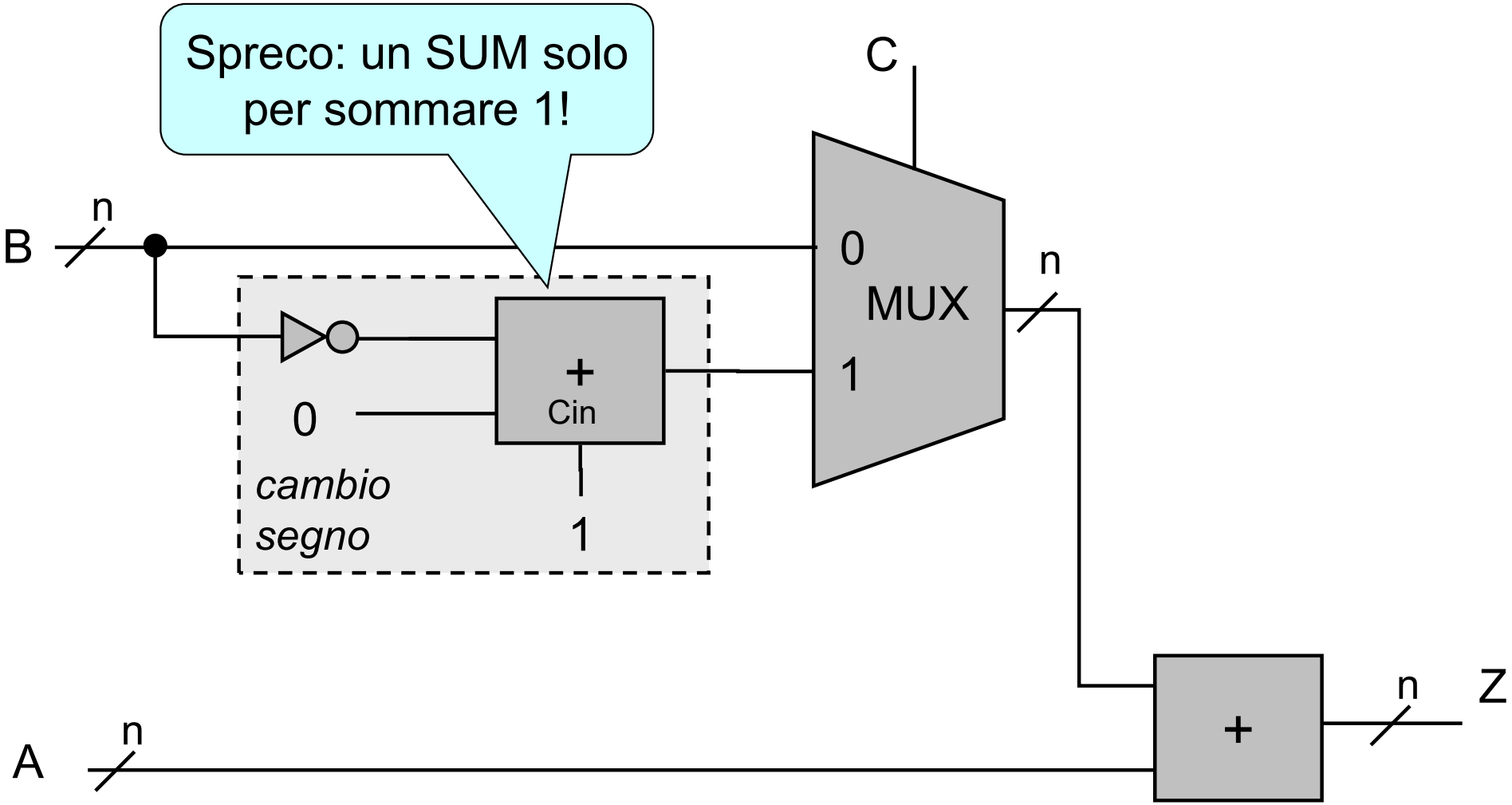


Flip del segno (complemento a due)

- Sottoproblema: circuito che cambia segno (in complemento a due)
  - Ricorda: cambiare segno = flip dei bit, e aggiungere 1

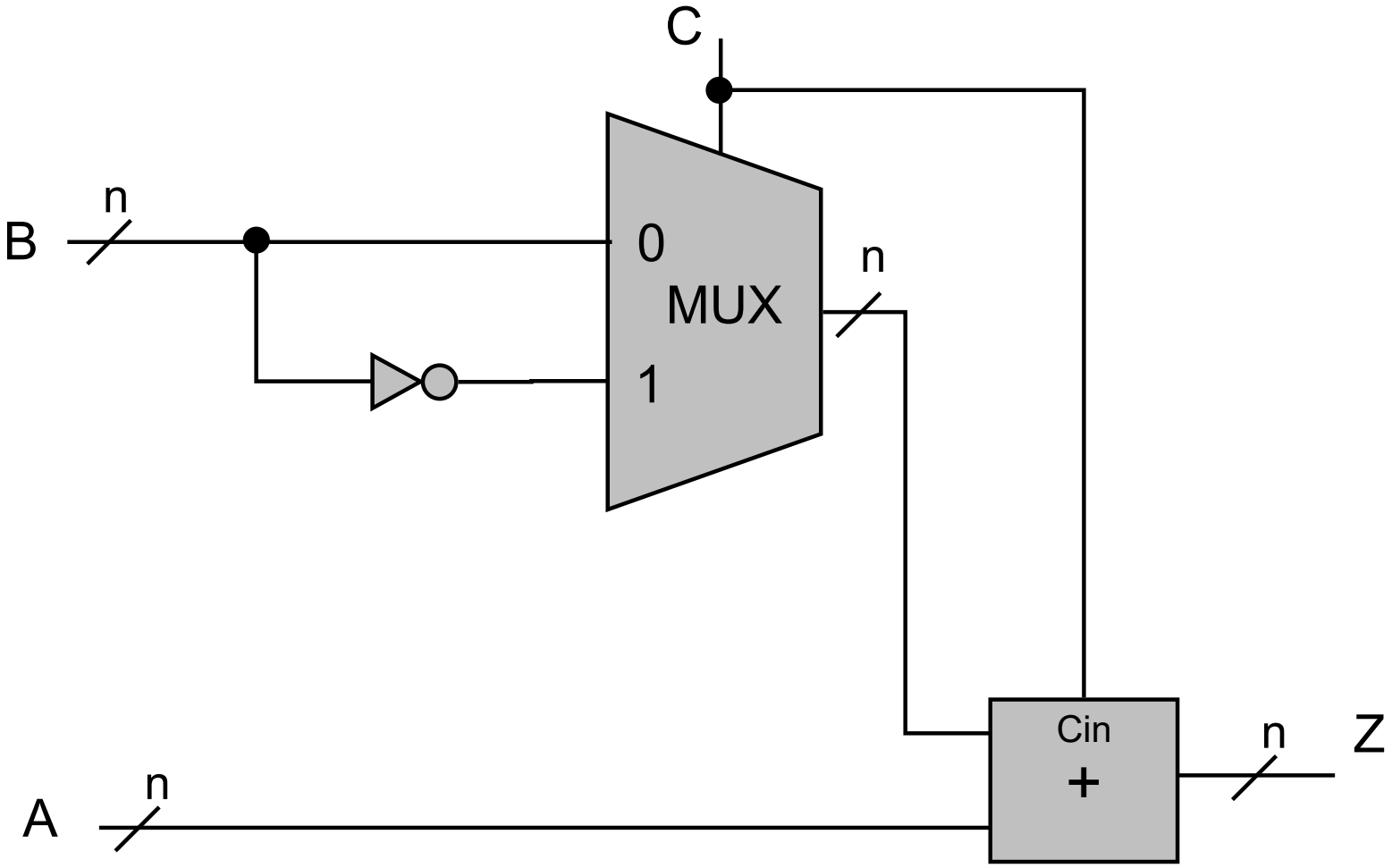


Soluzione 2 (raffinamento)



abbiamo finito secondo voi?

Soluzione 2 (ottimizzata)



Estensione di un numero: aggiungere cifre a sinistra

- A volte, vogliamo passare da una rappresentazione di un numero a *n* bit ad una rappresentazione a *m* bit, con *m* > *n*
- Come si sa, *aggiungere zeri a sinistra* non modifica un numero:

  $|11011|_2 = 27$

  $|\mathbf{000}11011|_2$ = ancora 27
  - Questa operazione si chiama **estensione (con zero)**
- Ma attenzione: per estendere un numero in **CP2** senza cambiarlo occorre aggiungere a sinistra: **0** se il MSB è **0**, ma **1** se il MSB è **1**:

  $|01101|_2 = +13$

  $|\mathbf{000}01101|_2$ = ancora +13

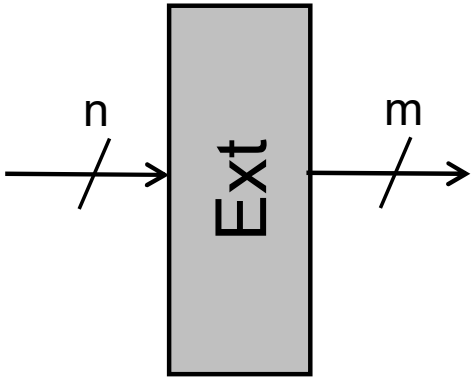
  $|11001|_2 = -7$

  $|\mathbf{111}11001|_2$ = ancora -7

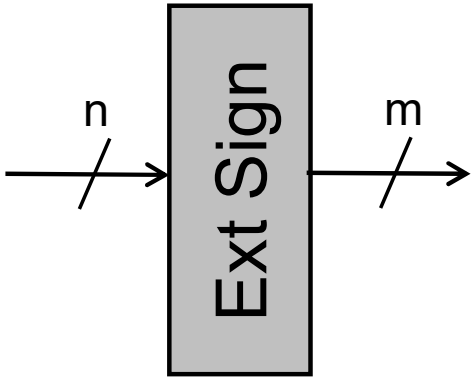
  ←verificare!
  - Questa operazione si chiama **estensione in segno**
- Quali circuiti implementano queste due operazioni?

Blocchi funzionali per estensione

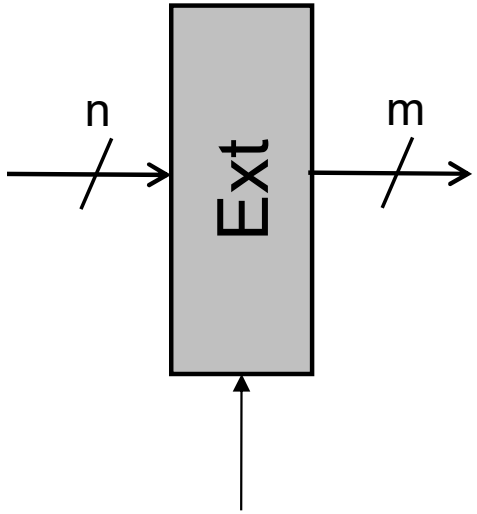
Estensione con zero



Estensione in segno



Estensione a scelta

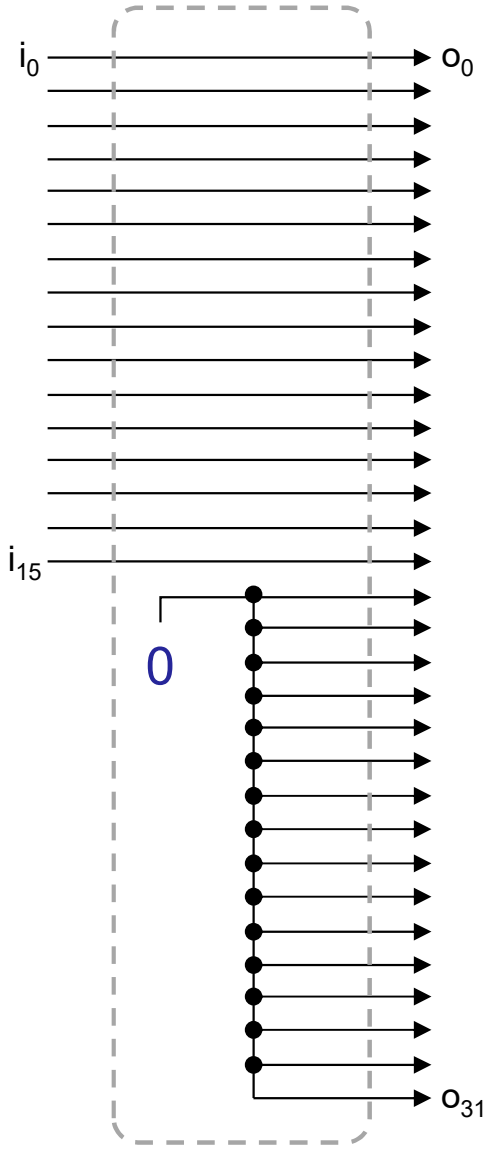


se 0: con zero
se 1: in segno

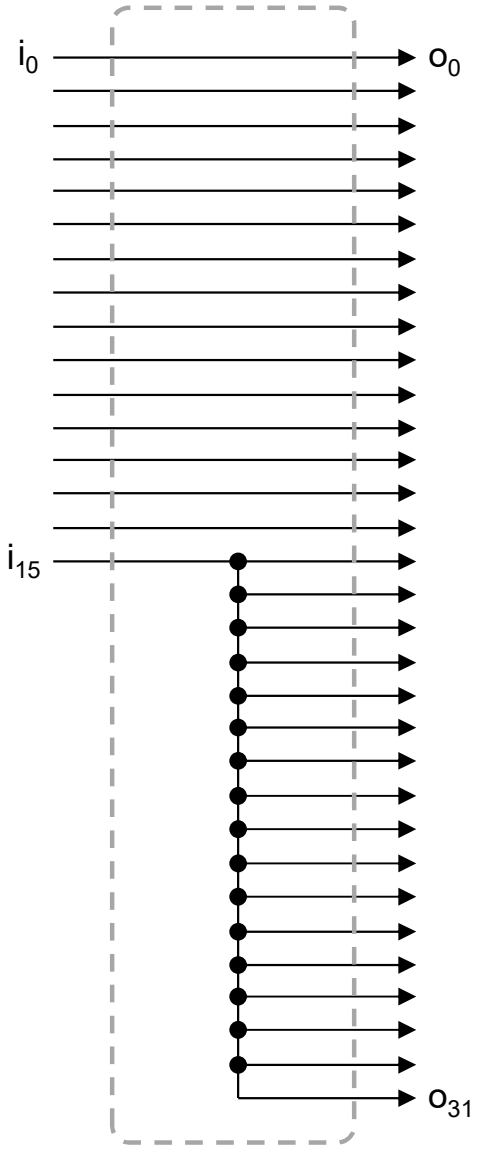
(con m>n)

Blocchi funzionali per estensione (qui, da 16 a 32 bit): implementazione

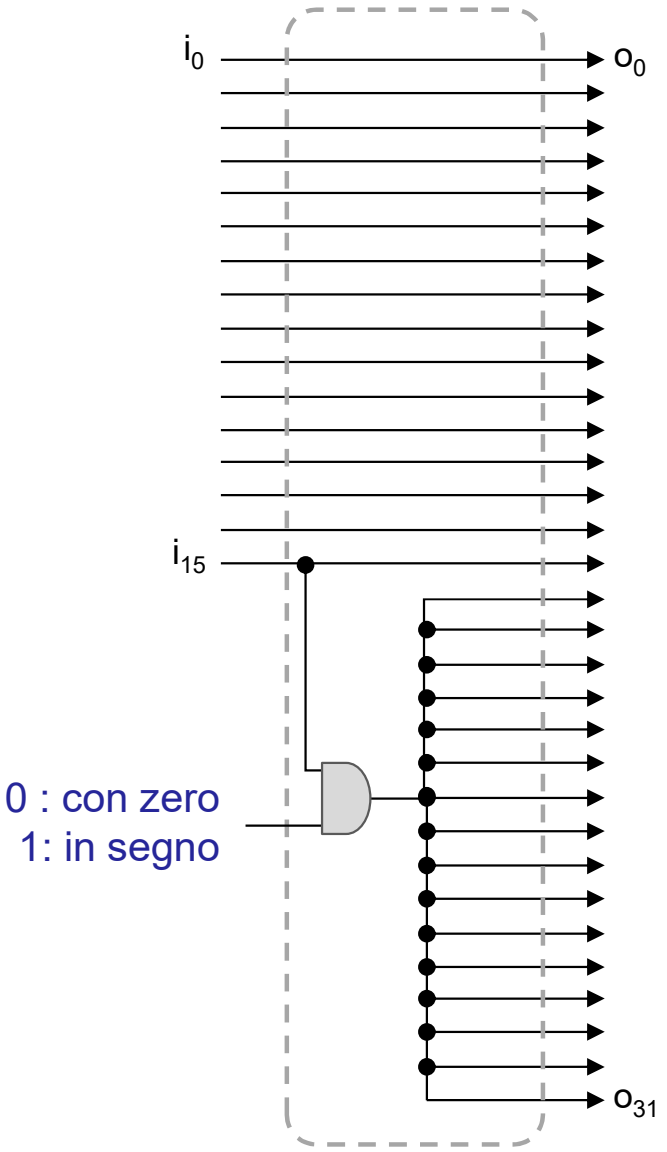
Estensione con zero



Estensione in segno



Estensione a scelta

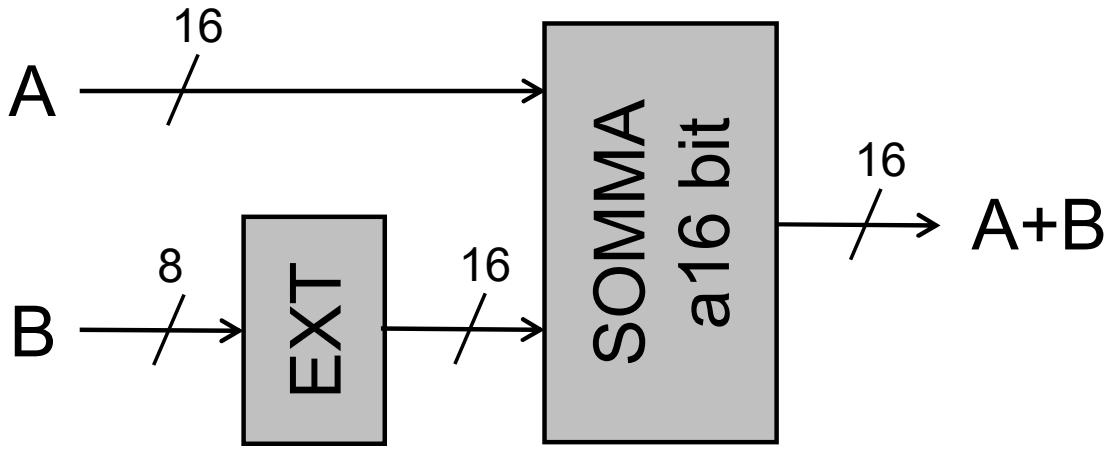


Esempio

- Come sommare un numero A a 16 bit con un numero B a 8 bit? (senza segno)
- Risposta:

Esempio

- Come sommare un numero A a 16 bit con un numero B a 8 bit? (senza sengo)
- Risposta:



Left Shift (moltiplicazione per 2, 4, 8 …)

analogo in base 10:
1320 = **132** x 10
13200 = **132** x 100

- *Left-shift* (di *n*): spostare le cifre binarie *n* posti a sinistra
  - le *n* cifre più a sinistra scompaiono, e da destra compaiono *n* **zeri**
- Ricorda:
  uno *shift* a sinistra in base 2 di 1 = raddoppio del numero
  uno *shift* a sinistra in base 2 di *n* cifre = moltiplicazione per 2^n
- Notazione: <<
- Esempio:

$|000110111|_2 = 27$

$|000110111|_2 << 1 = |001101110|_2 = 54 \quad (= 27x2)$

$|000110111|_2 << 2 = |011011100|_2 = 108 \; (= 27x4)$

$|000110111|_2 << 3 = |110111000|_2 = 216 \; (= 27x8)$

- **Vale anche in CP2?**

Left Shift (moltiplicazione per 2, 4, 8 …)

analogo in base 10:
1320 = **132** x 10
13200 = **132** x 100

- *Left-shift* (di *n*): spostare le cifre binarie *n* posti a sinistra
  - le *n* cifre più a sinistra scompaiono, e da destra compaiono *n* **zeri**
- Ricorda:
  uno *shift* a sinistra in base 2 di 1 = raddoppio del numero
  uno *shift* a sinistra in base 2 di *n* cifre = moltiplicazione per 2^n
- Notazione: <<
- Esempio:

$|00011011|_2 = 27$

$|00011011|_2 << 1 = |00110110|_2 = 54 \quad (= 27x2)$

$|00011011|_2 << 2 = |01101100|_2 = 108 \; (= 27x4)$

$|00011011|_2 << 3 = |11011000|_2 = 216 \; (= 27x8)$

- **SI! Vale anche in CP2!**

$|11111011|_2 = -5$

$|11111011|_2 << 1 = |11110110|_2 = -10$

$|11111011|_2 << 2 = |11101100|_2 = -20$