Esercitazione 02

Circuiti Aritmetici

Gianluca Brilli
gianluca.brilli@unimore.it

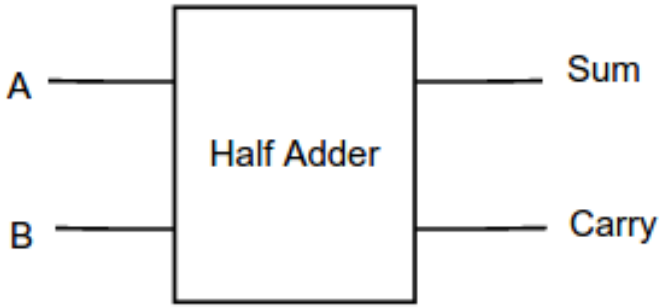
Esercizio 01

- Creare un nuovo sottocircuito chiamato "adder_1", e implementarvici un half adder utilizzando solo porte logiche (anche non elementari, come XOR)

Truth Table

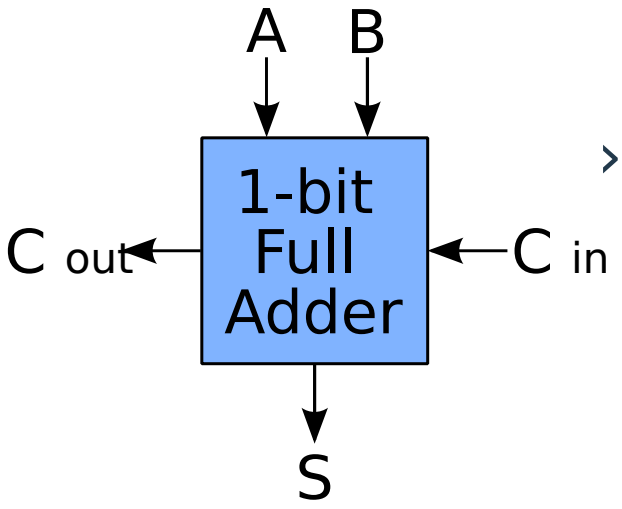
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Block Diagram



Esercizio 02

- Estendere l'half adder per renderlo un full adder e salvarlo in un sottocircuito.



- Pensate come tenere conto del carry in ingresso.

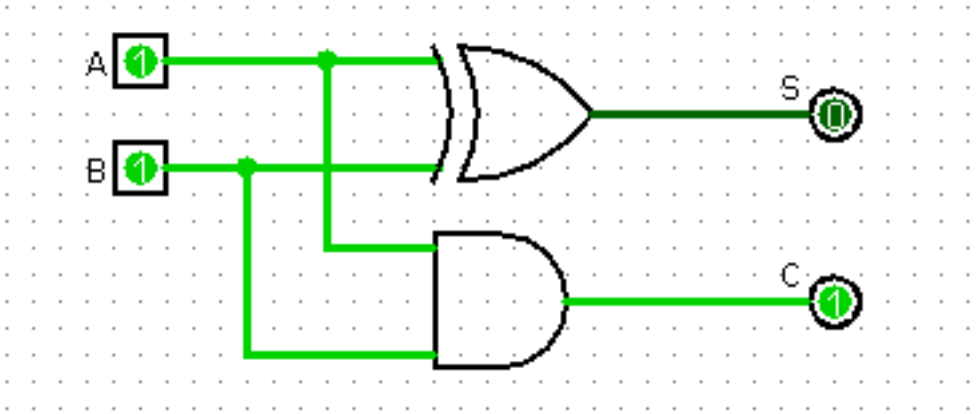
Esercizio 03

- Estendere il full adder ad un bit per renderlo un full adder a 4 bit.

Esercizio 01 – Soluzione (1)

- Notiamo che dall'algebra di Boole, la somma vale 0 quando A e B sono entrambi 0 o quando sono entrambi 1:
  - S = A xor B
- Il carry è presente solo quando A e B sono entrambi 1:
  - Co = A and B

Esercizio 01 – Soluzione (2)



- Teniamoci da parte l’adder in un sottociruito, ci servirà per i prossimi esercizi.

Esercizio 02 – Soluzione (1)

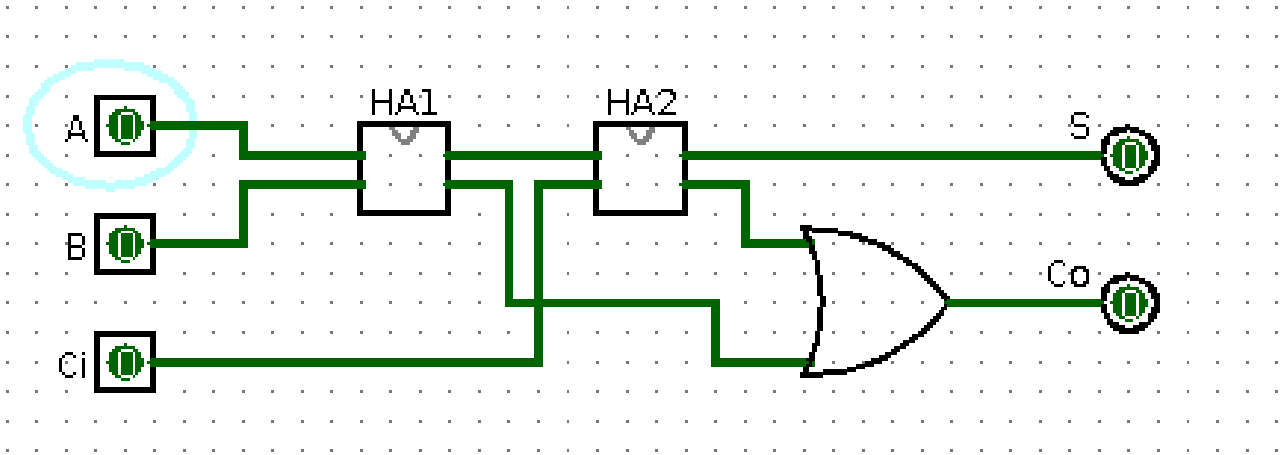
- Analizziamo il comportamento del circuito:
  - Dobbiamo realizzare la somma di due bit più il carry in ingresso, quindi:
    S = (A + B) + Ci
  - Abbiamo carry in uscita quando:

    1) A e B sono entrambi 1, oppure

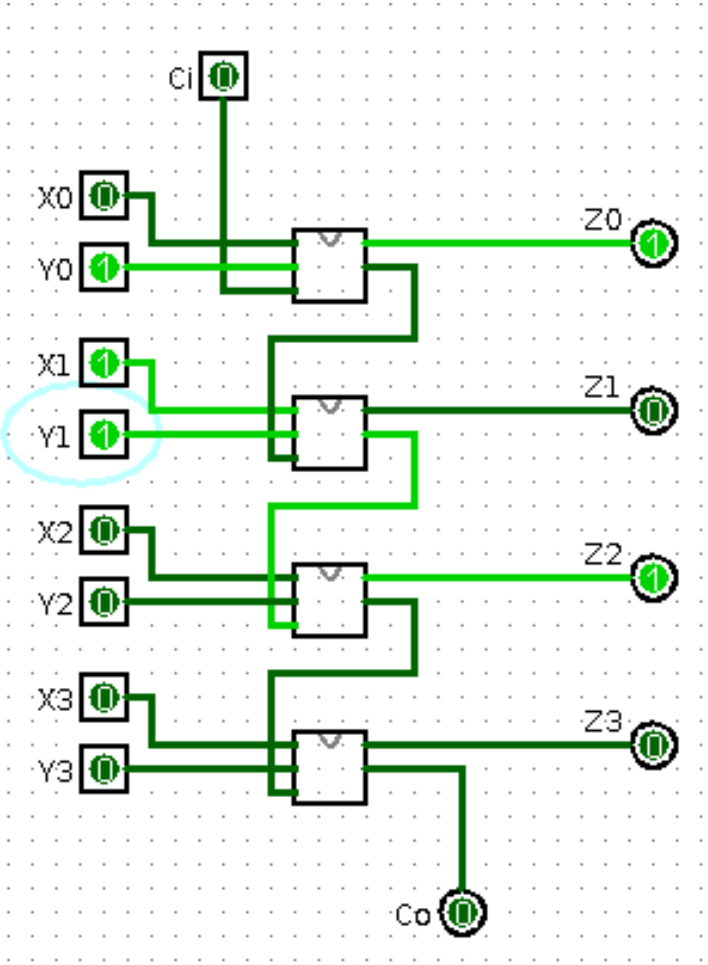
    2) A e B sommano 1 e c'è carry in ingresso
- S = (A + B) + Ci
- Co = A and B or (A + B) and Ci

Esercizio 02 – Soluzione (2)

- HA1 e HA2 sono gli Half Adder a 1 bit realizzati nell'esercizio precedente.



Esercizio 03 - Soluzione



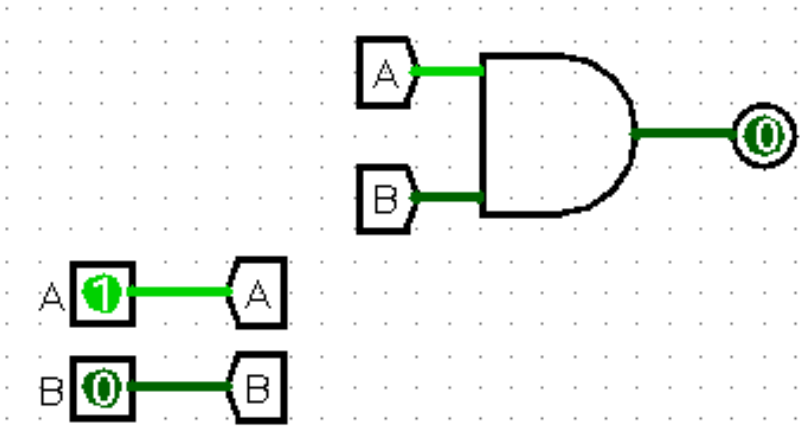
- I singoli blocchi sono i Full Adder a 1 bit dell'esempio precedente.
- Testiamo il funzionamento su un esempio:
- Y = 0011 +
  X = 0010 =

  Z = 0101

Esercizio 03 – Nota (1)

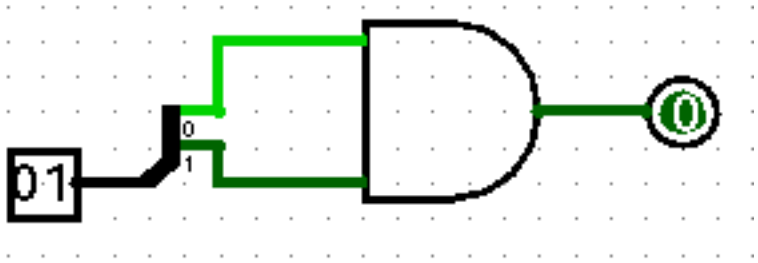
- Il circuito è corretto e funziona, ma notiamo che anche con solo 4 bit, abbiamo già un grande numero di pin e fili. Usiamo quindi alcune **feature avanzate di Logisim**:
- **Tunnel**. Realizzano fili virtuali mediante etichette testuali, utili per migliorare grandemente la leggibilità e la chiarezza di circuiti complessi, visto che riducono il numero di collegamenti visibili e consentono l'assegnamento di un nome rappresentativo a quelli nascosti.

Esercizio 03 – Nota (2)

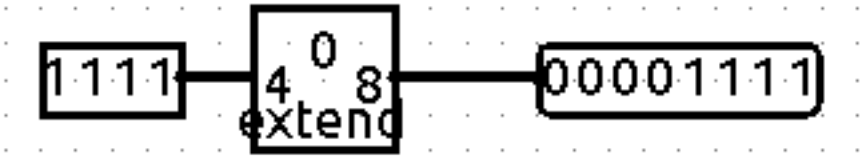


- **Splitter**. Permette di disegnare fili ad ampiezza maggiore di 1 bit.

Esercizio 03 - Nota

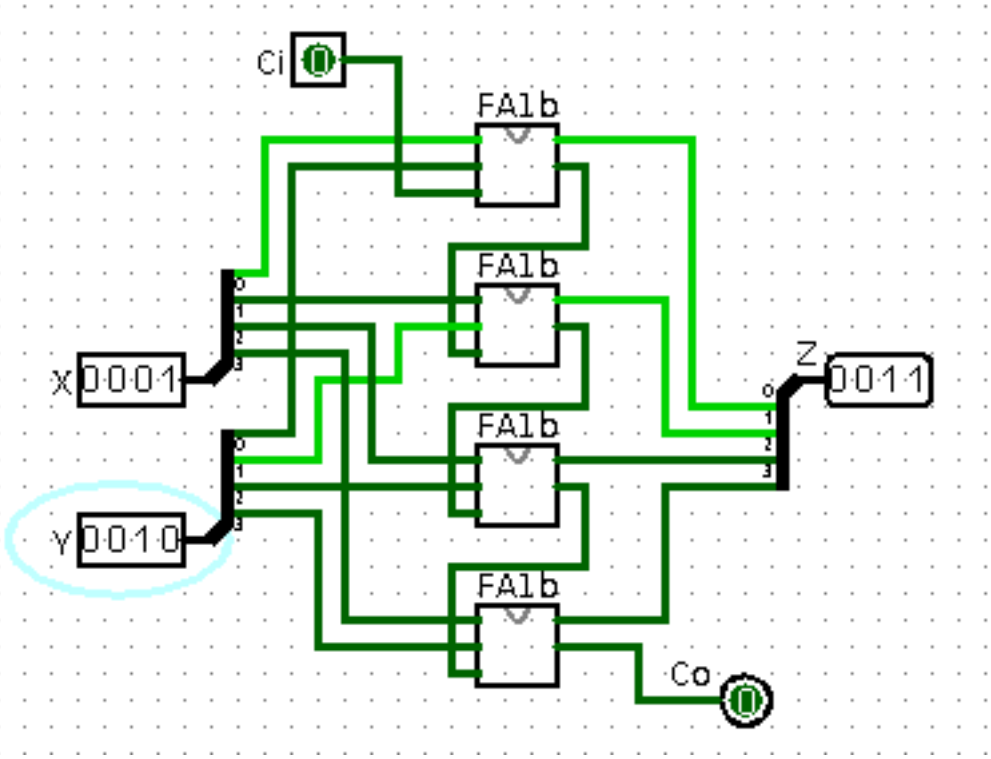


- **Extender**. Permette di estendere o troncare l'ampiezza di un filo.



Esercizio 03 - Nota

- Quindi organizziamo meglio il nostro Full Adder a 4 bit.

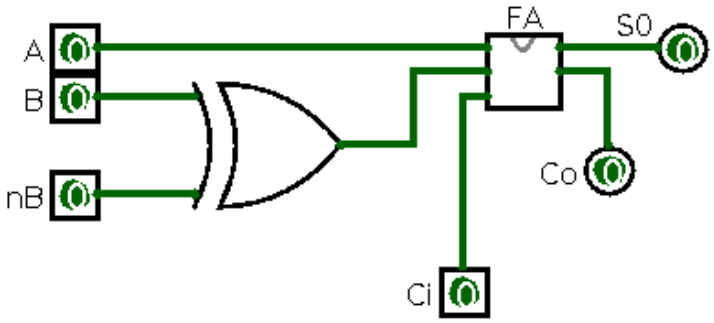


Esercizio 04

- Partendo dal Full Adder a 1 bit estendiamolo aggiungendo anche funzionalità di sottrattore.
- **Suggerimenti**:
  - Come aggiungiamo il complemento a 2 al nostro adder?
  - Una volta costruito il sottrattore ad 1 bit pensate a come estenderlo a 4 bit.

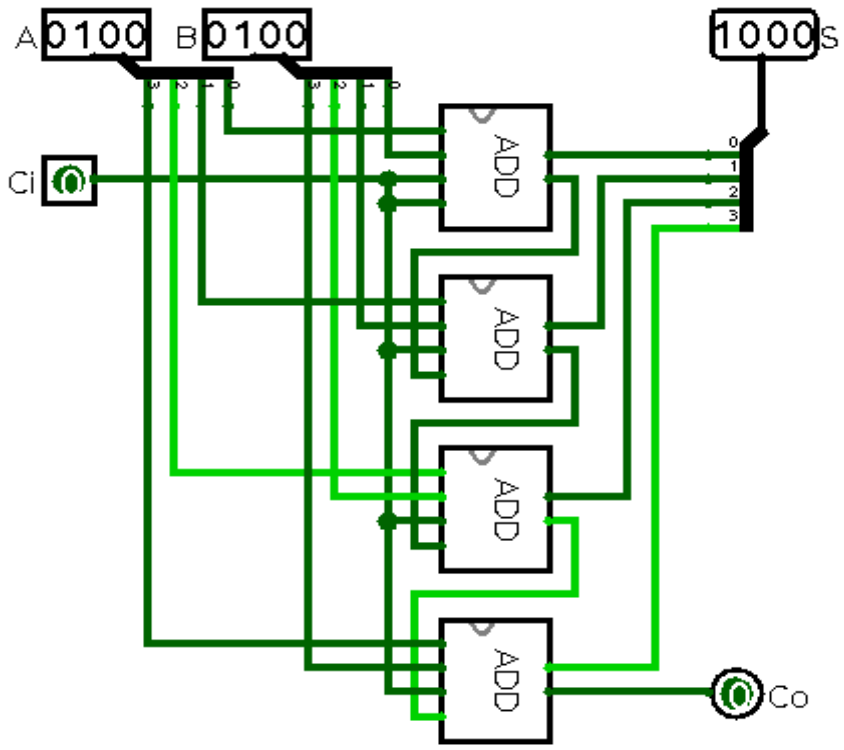
Esercizio 04 – Soluzione A

- Sommatore-Sottrattore ad 1 bit: aggiungiamo la possibilità di negare B e sfruttiamo il carry in ingresso per sommare 1.



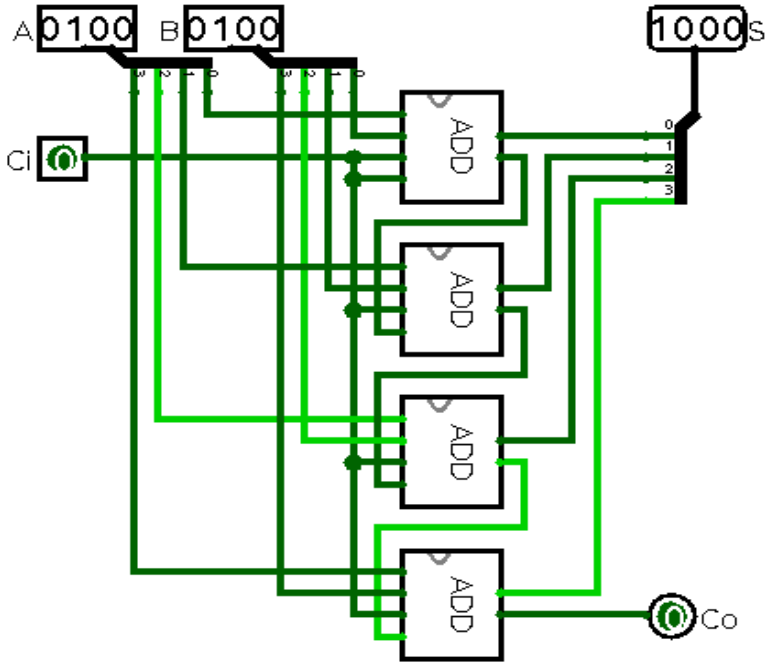
- Notiamo che la porta XOR agisce come inverter comandando da un segnale di controllo (nB).

Esercizio 04 – Soluzione B



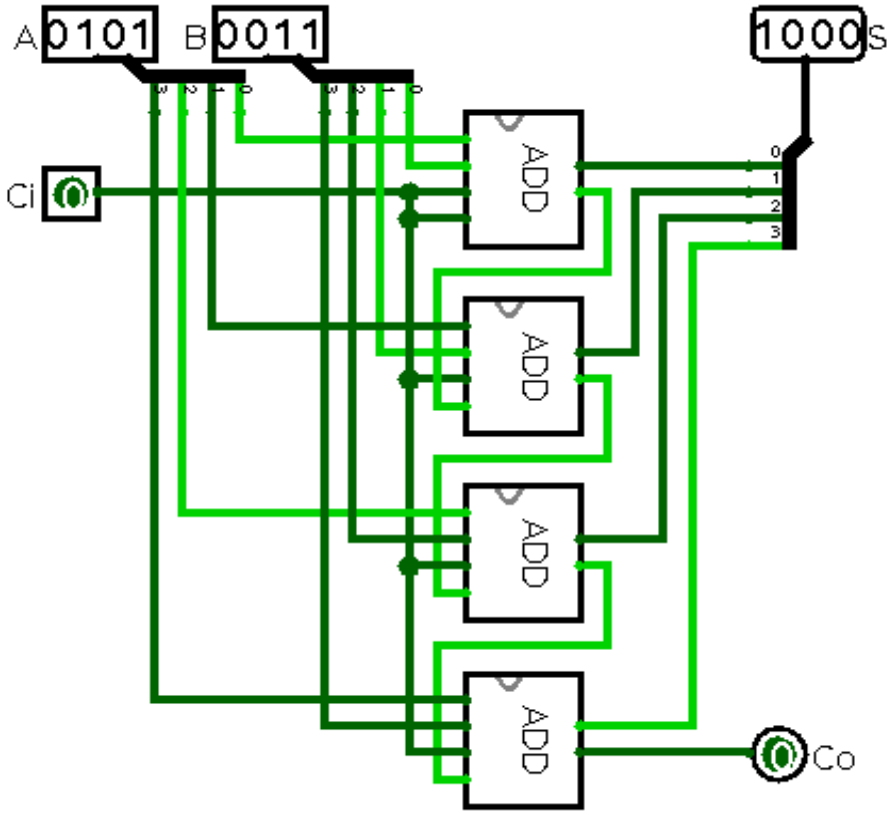
- Colleghiamo in cascata 4 Adder.
- Il Carry in ingresso al primo Adder è collegato al negB di ogni Adder.
- In questo modo usiamo un solo bit per decidere somma o sottrazione.

Esercizio 04 – Soluzione C



- Somma: carry a 0
- Sottrazione: carry a 1

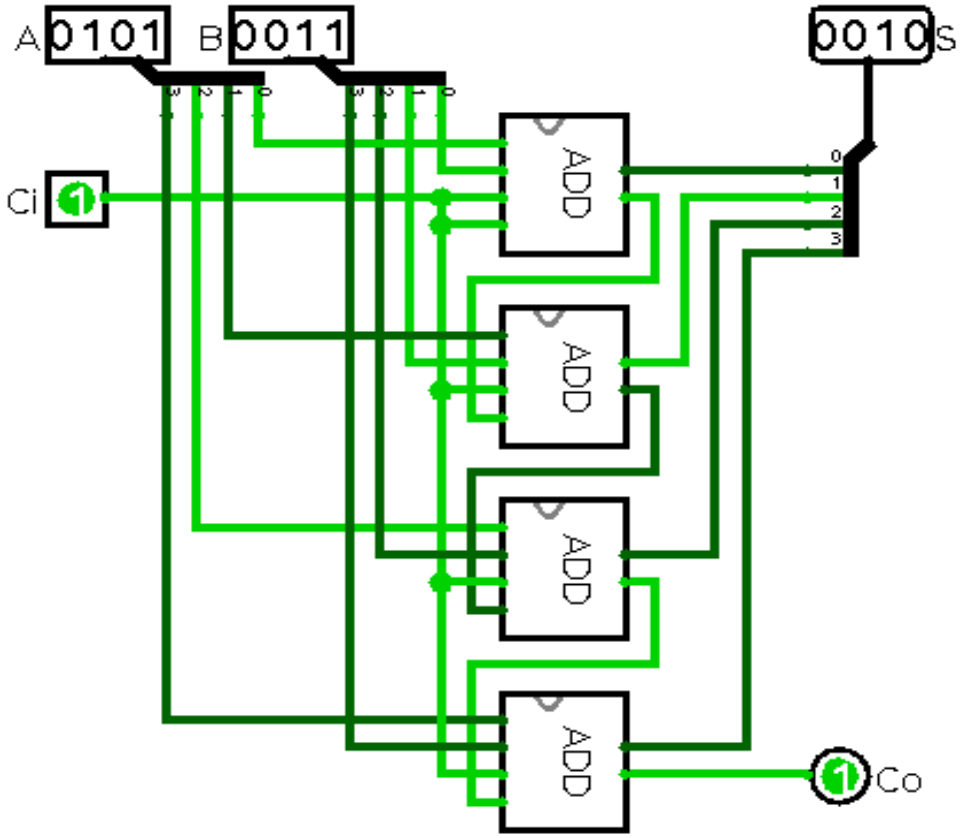
Esercizio 04 – Test 1



- 5 + 3 = 8
- A = 0101 +
  B = 0011 =

  S = 1000

Esercizio 04 – Test 2

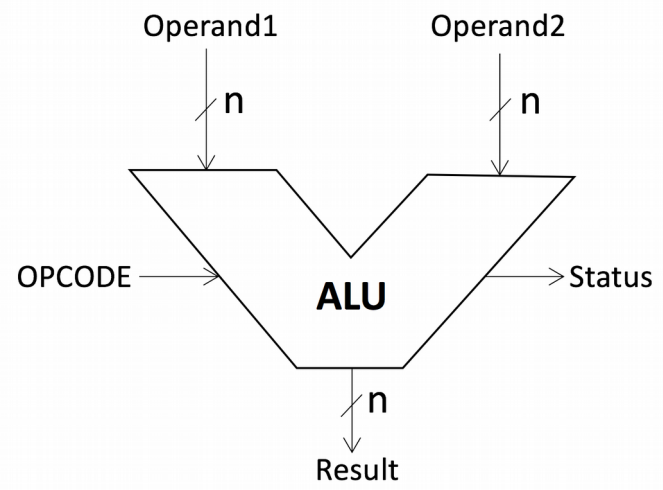


- 5 - 3 = 2
- A = 0101 -
  B = 0011 =

  S = 0010

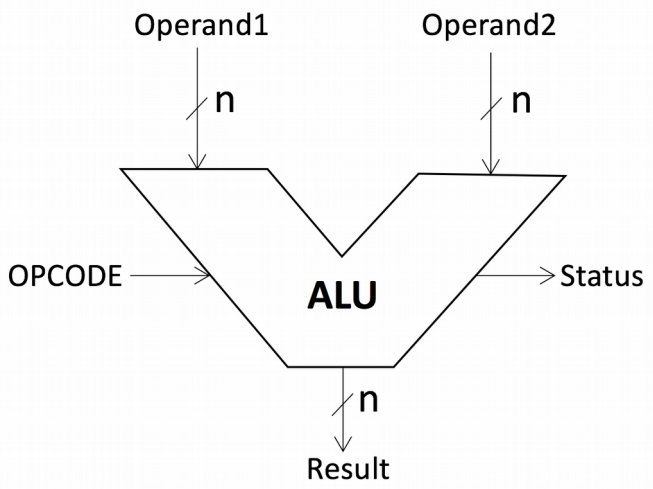
Esercizio 05 (1)

- Sfruttando quello che abbiamo costruito con gli esercizi precedenti, andiamo a costruire un **ALU a 8 bit**, Per semplicità partiamo costruendo una versione a 1 bit e poi estendiamola.



Esercizio 05 (2)

- In particolare facciamo in modo che siano implementate le seguenti operazioni:



Opcode	Mnemonic
00	AND A, B
01	OR A, B
10	ADD A, B
11	SUB A, B

Esercizio 05 (3)

- Facciamo in modo che la nostra ALU lavori con numeri **interi con segno** a n-bit, in **complemento a due**.
- Quindi avendo n-bit a disposizione abbiamo 2^n possibili combinazioni.
- Supponendo di essere nel caso a 4-bit, allora abbiamo 16 combinazioni.

Esercizio 05 (4)

- Quindi in complemento a due, il range che possiamo rappresentare è il seguente:

Senza Segno	Con Segno	Binario
0	**0**	0000
1	**1**	0001
2	**2**	0010
3	**3**	0011
4	**4**	0100
5	**5**	0101
6	**6**	0110
7	**7**	0111
8	**-8**	1000
9	**-7**	1001
10	**-6**	1010
11	**-5**	1011
12	**-4**	1100
13	**-3**	1101
14	**-2**	1110
15	**-1**	1111

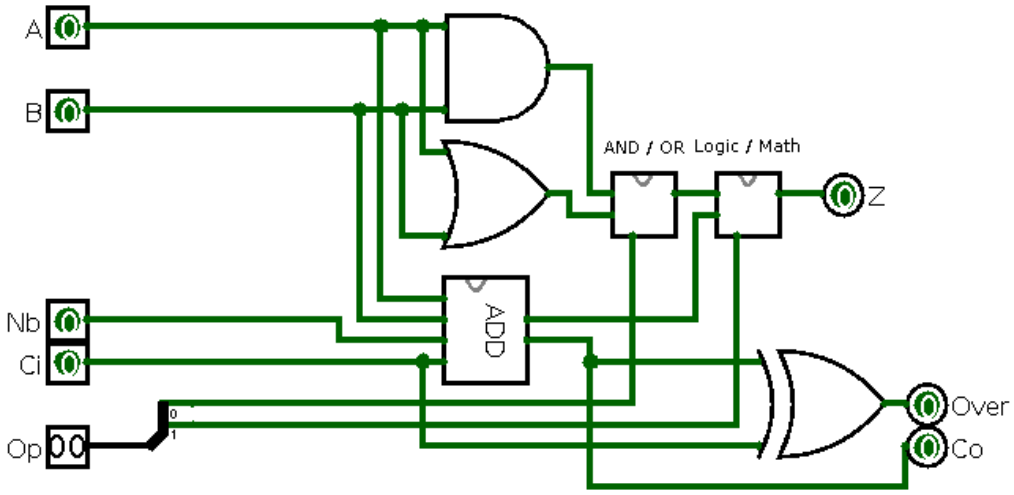
$$[-2^{n-1}, 2^{n-1} - 1]$$

Esercizio 05 (5)

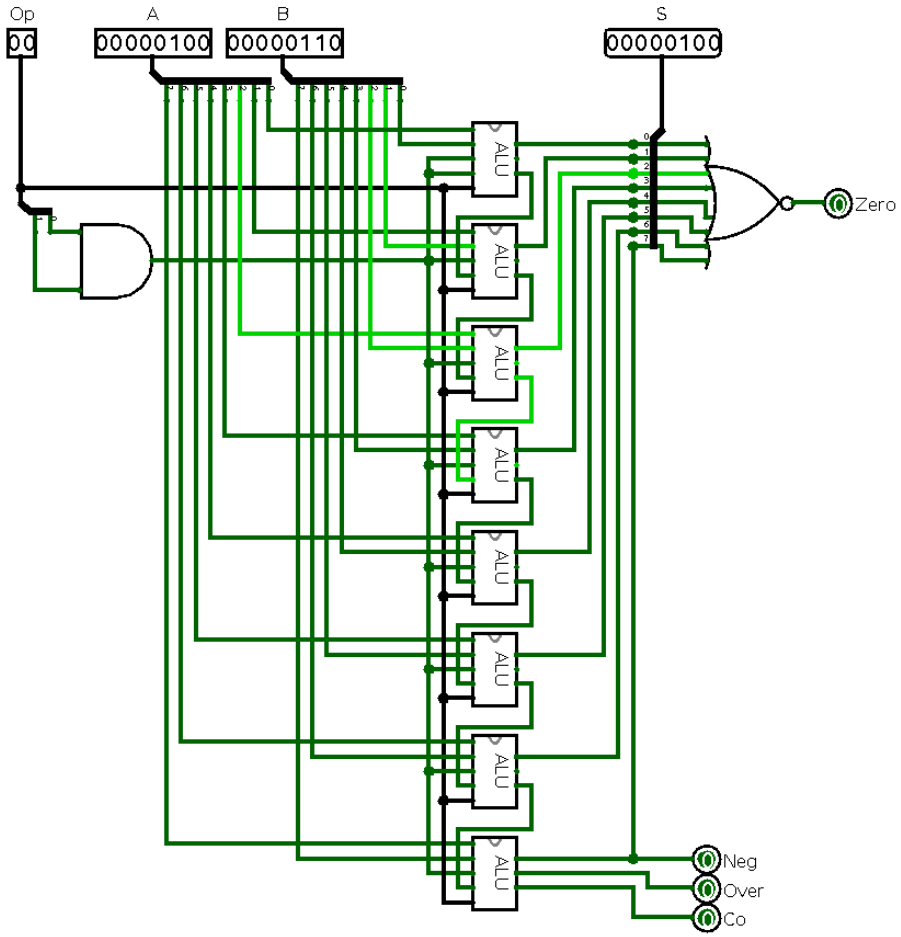
- Facciamo inoltre in modo che la nostra ALU sia dotata dei seguenti flags:
  - ***Zero (Z)***: indica se il risultato dell'operazione è zero;
  - **Negativo (N)**: indica se il risultato dell'operazione è un numero negativo;
  - **Carry Out (Co)**: non è rappresentabile come numero unsigned;
  - ***Overflow (O)***: non è rappresentabile come numero signed.

Esercizio 05 – Soluzione Passo Passo (1)

- Realizziamo l'ALU a 1 bit tramite i blocchi che ci siamo già costruiti:

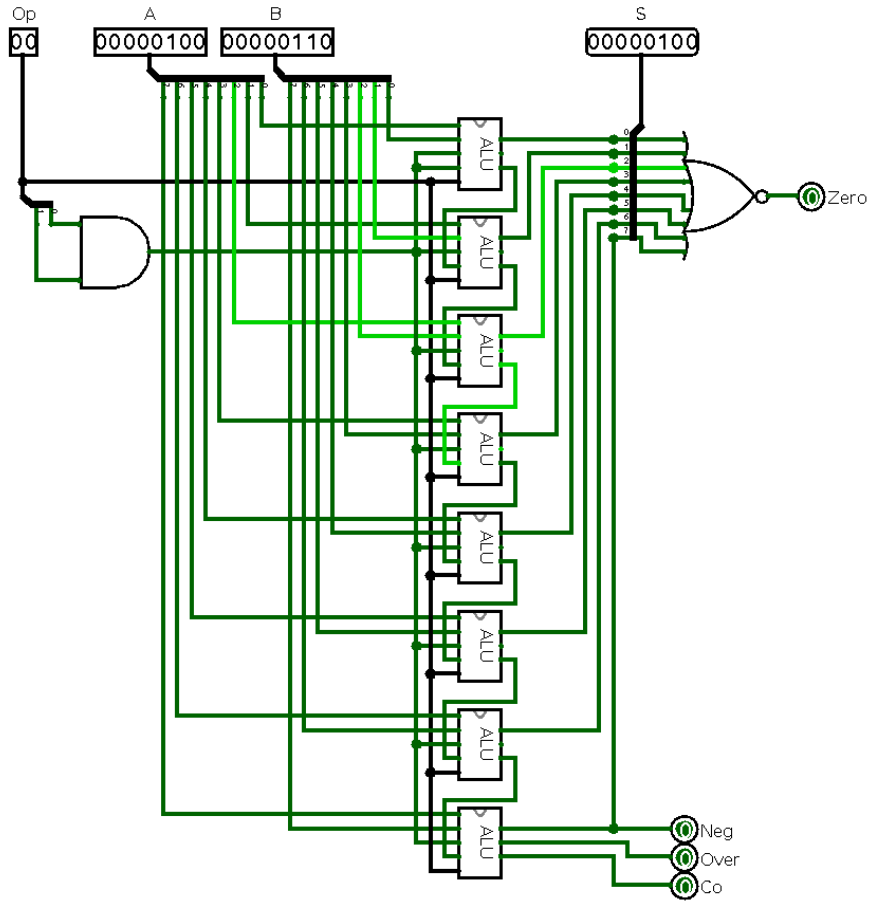


Esercizio 05 – Soluzione Passo Passo (2)



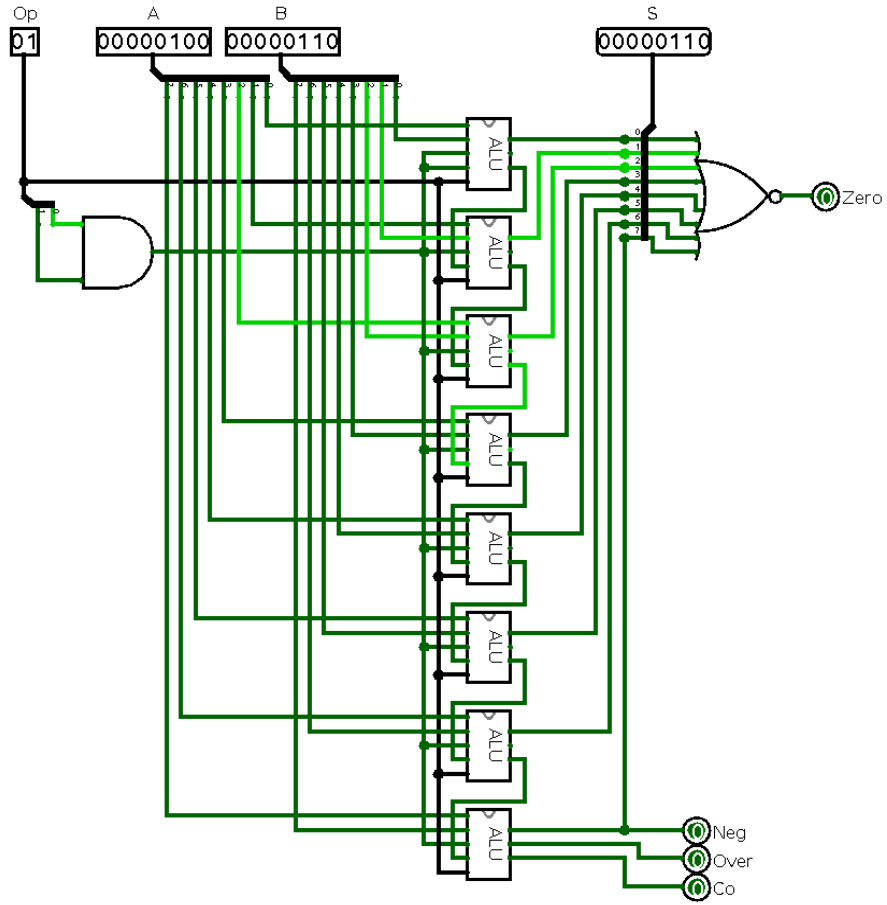
- Tramite l'ALU a 1 bit andiamo a realizzare un' ALU a 8 bit:

Esercizio 05 – Collaudo (1)



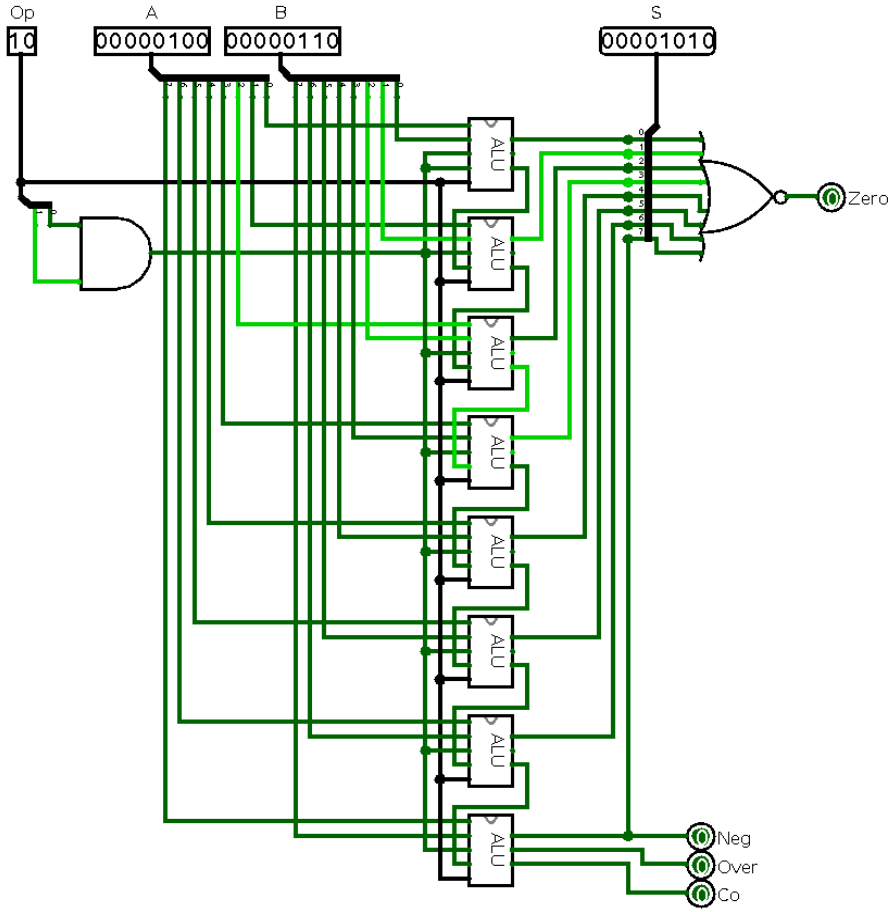
- Testiamo l'operazione 4 **AND** 6, che in binario diventa:
- 00000**1**00 **AND** 00000**11**0 = 00000**1**00

Esercizio 05 – Collaudo (2)



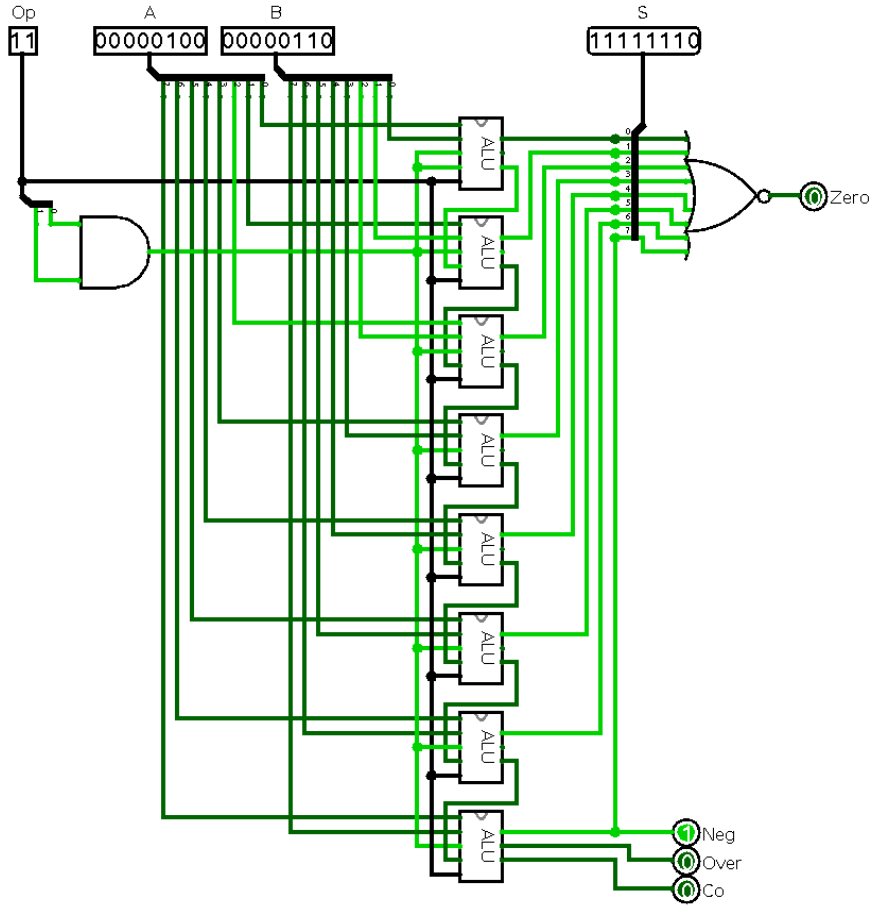
- Testiamo l'operazione 4 **OR** 6, che in binario diventa:
- 00000**1**00 **OR** 00000**11**0 = 00000**11**0

Esercizio 05 – Collaudo (3)



- Testiamo l'operazione 4 + 6, che in binario diventa:
- 00000**1**00 +
  00000**11**0 =
  0000**1**0**1**0

Esercizio 05 – Collaudo (4)



- Testiamo l'operazione 4 + 6, che in binario diventa:
- 00000**1**00 - 00000**11**0 = **1111111**0