UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Esercitazione 2: Circuiti Aritmetici

Architettura dei calcolatori [MN1-1143]

Corso di Laurea in Ingegneria Informatica
(D.M.270/04) [16-215]
Anno accademico 2020/2021

Dott. Gianluca Brilli
gianluca.brilli@unimore.it
Prof. Marko Bertogna
Marko.bertogna@unimore.it

Esercizio 01

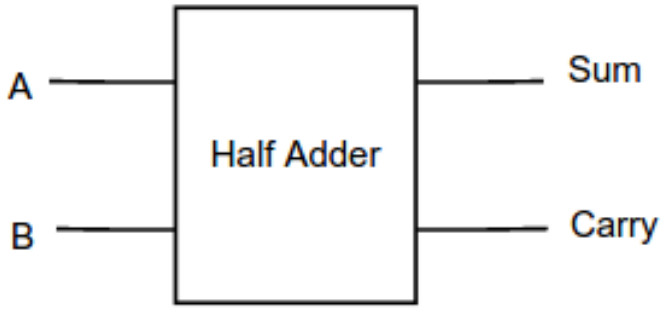
Half Adder

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

Truth Table

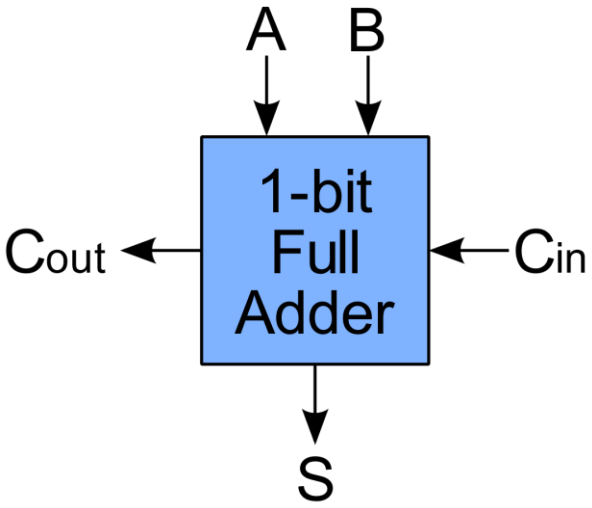
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Block Diagram



Esercizio 02
Full Adder

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



Pensate come tenere conto del carry in ingresso.

Esercizio 03

Full Adder a 8 bit

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

Estendere il full adder ad un bit per renderlo un full adder a 8 bit.

Esercizio 01 - Soluzione

Half Adder

Notiamo che dall'algebra di Boole, la somma vale 0 quando A e B sono entrambi 0 o quando sono entrambi 1:

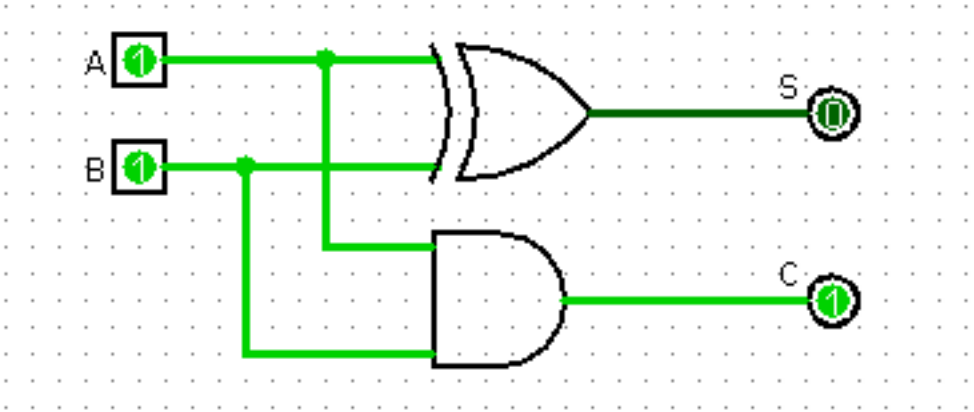
$$S = A \text{ xor } B$$

Il carry è presente solo quando A e B sono entrambi 1:

$$Co = A \text{ and } B$$

Esercizio 01 - Soluzione

Half Adder



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

Esercizio 02 - Soluzione
Full Adder

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:

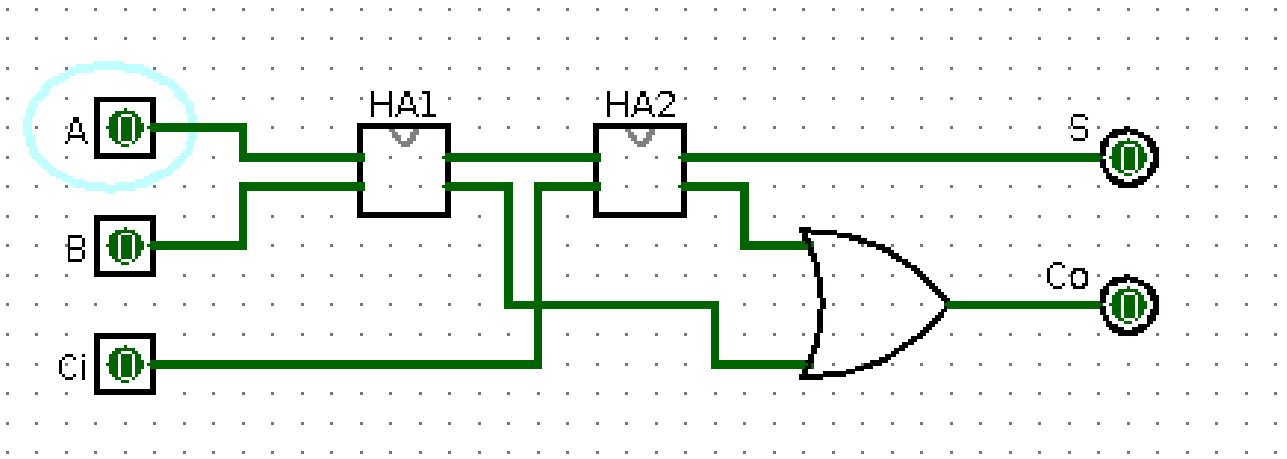
1) A e B sono entrambi 1, oppure

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

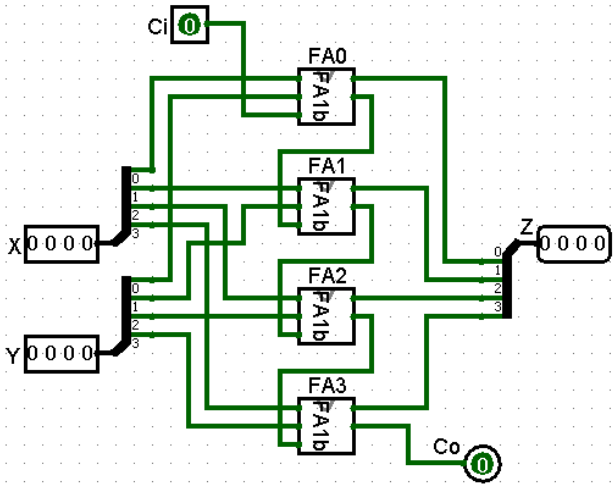
Esercizio 02 - Soluzione
Full Adder

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



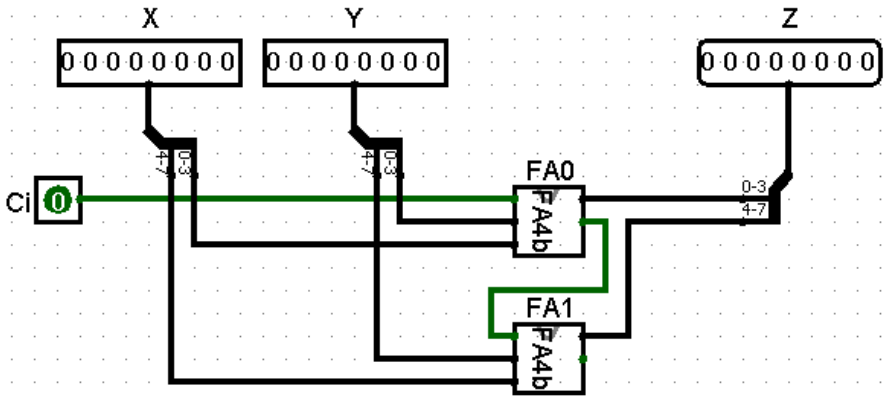
Esercizio 03 - Soluzione

Full Adder a 8 bit



Full Adder a 4 bit realizzato con 4 Full Adder a 1 bit.

Full Adder a 8 bit realizzato con 2 Full Adder a 4 bit.



Esercizio 04

Sottrattore

Partendo dal Full Adder a 1 bit estendiamolo aggiungendo anche funzionalità di sottrattore.

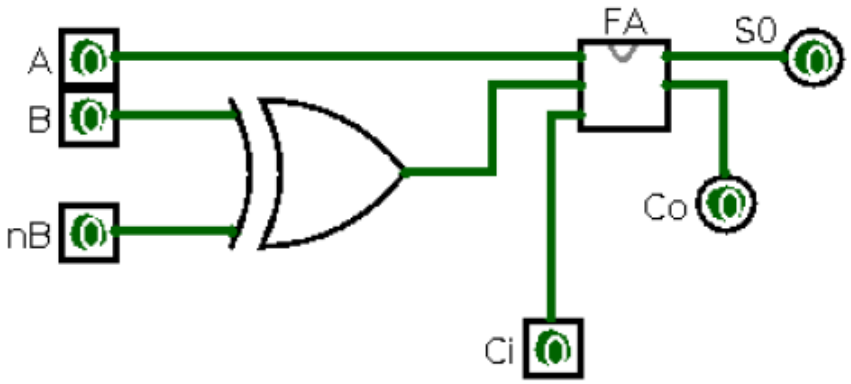
Aggiungere il complemento a 2 al Full Adder;

Infine estenderlo a più di un bit.

Esercizio 04 - Soluzione

Sottrattore

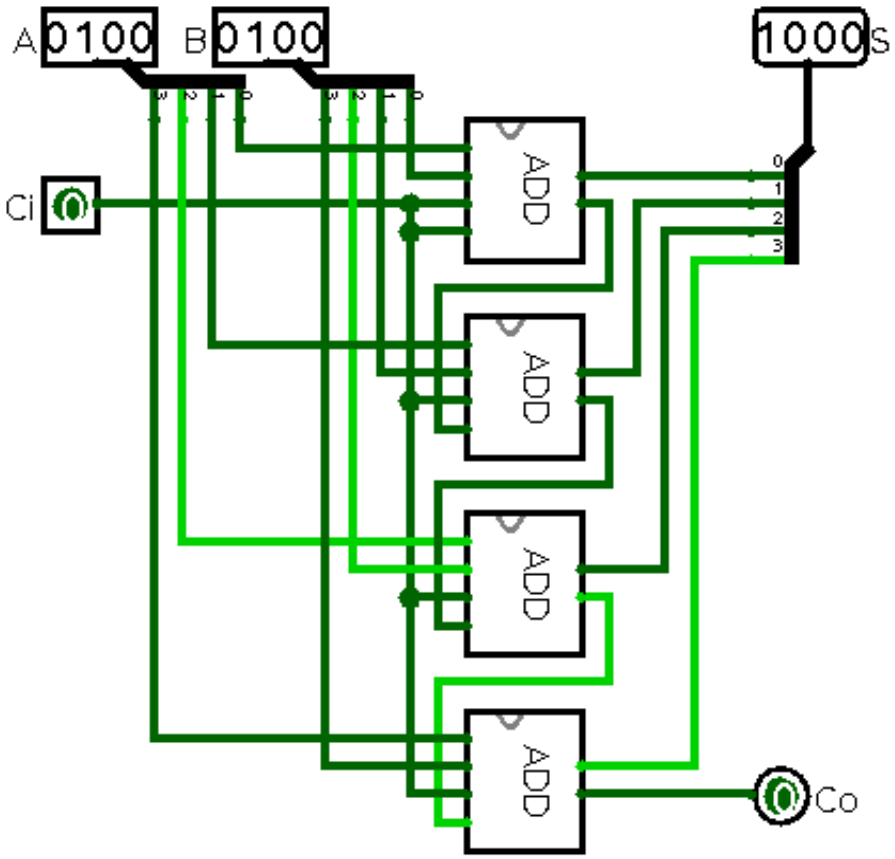
Sommatore-Sottrattore ad 1 bit: aggiungiamo la possibilità di negare B e sfruttiamo il carry in ingresso per sommare 1.



Notiamo infine che la porta XOR in questo caso la usiamo come inverter comandato da un segnale di controllo (nB).

Esercizio 04 - Soluzione

Sottrattore



Colleghiamo in cascata 4 Adder;

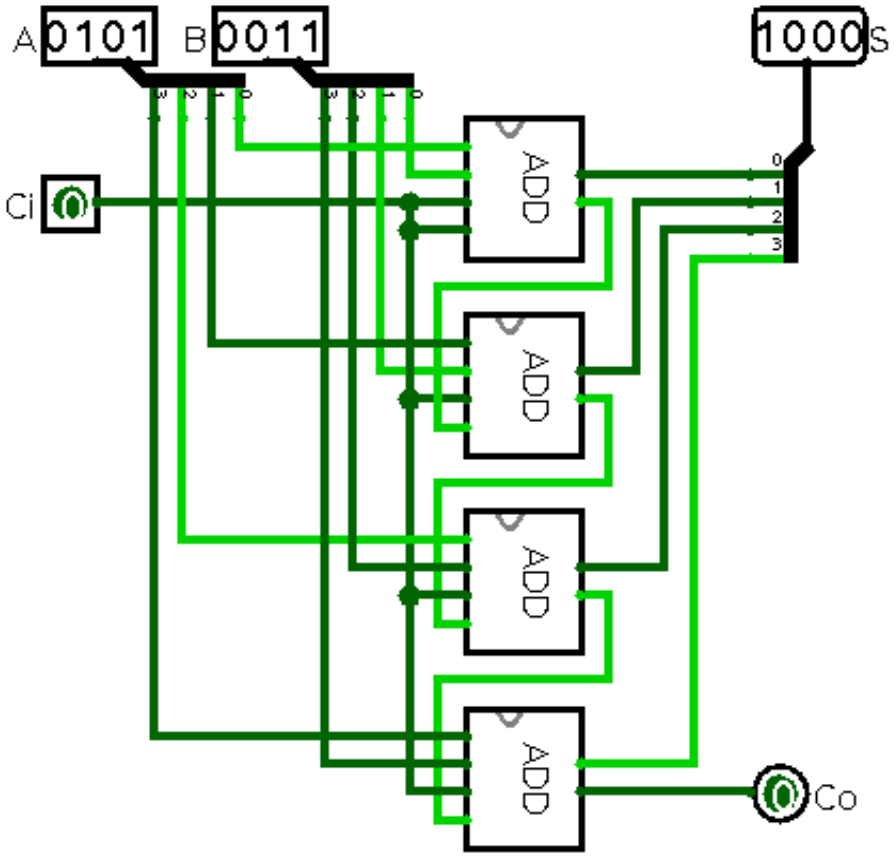
Il carry in ingresso al primo Adder è collegato al negB di ogni Adder;

In questo modo usiamo un solo bit per decidere somma o sottrazione.

Somma: carry a 0
Sottrazione: carry a 1

Esercizio 04 - Soluzione

Sottrattore



5 + 3 = 8

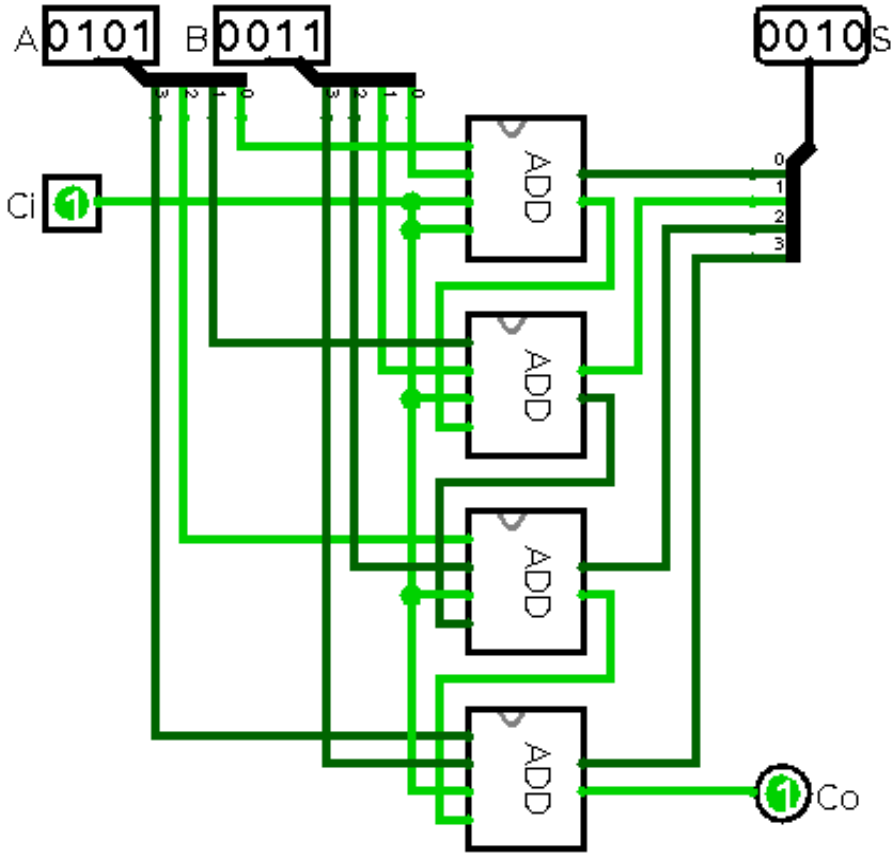
A = 0101 +
B = 0011 =

_____

S = 1000

Esercizio 04 - Soluzione

Sottrattore



5 – 3 = 2

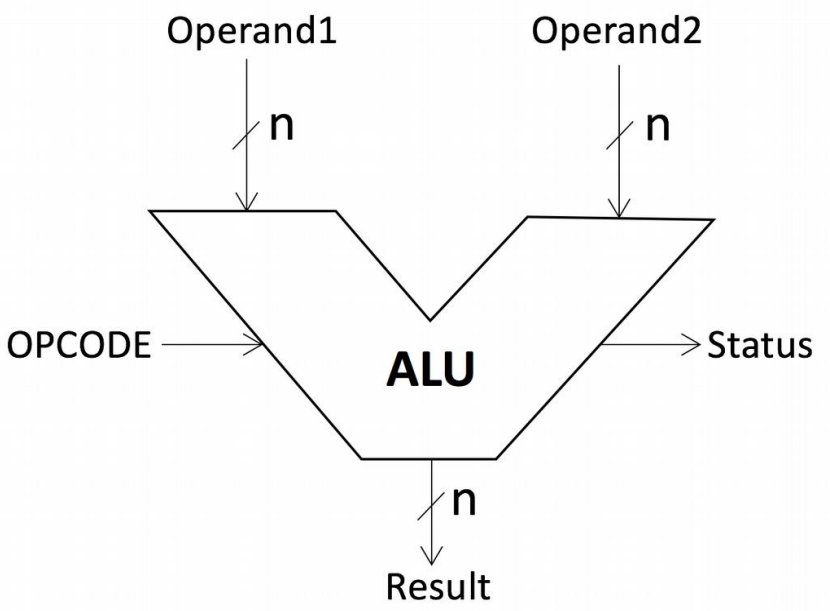
A = 0101 -
B = 0011 =

_____

S = 0010

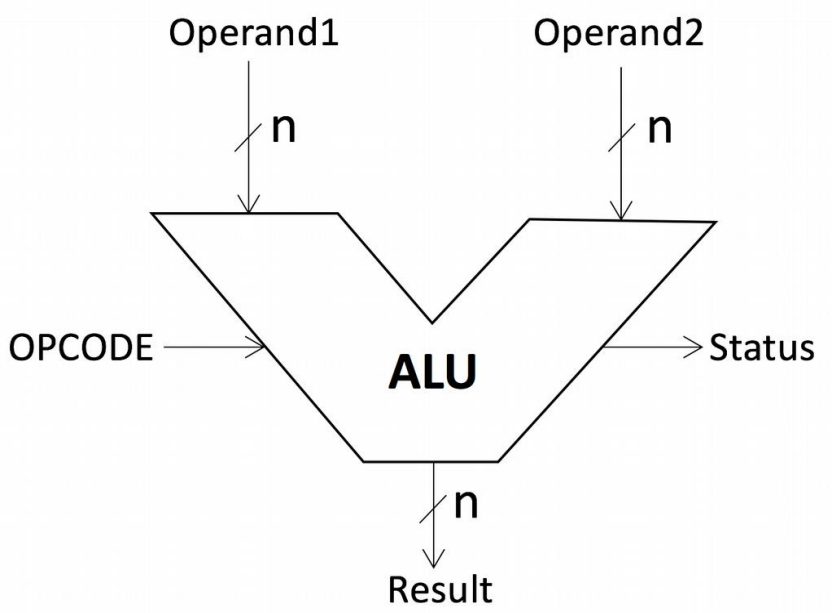
Esercizio 05
ALU

Sfruttando ciò che abbiamo realizzato tramite gli esempi precedenti, andiamo a costruire un ALU a 8 bit. Per semplicità partiamo costruendo una versione a 1 bit e poi estendiamola.



Esercizio 05
ALU

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
ALU

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
ALU

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

Esercizio 05
ALU

Facciamo inoltre in modo che la nostra ALU sia dotata dei seguenti flags:

1) **Zero (Z)**: indica se il risultato dell'operazione è zero;
2) **Negativo (N)**: indica se il risultato dell'operazione è minore di zero;
3) **Carry-Out (Co)**: indica se il numero è rappresentabile o meno come numero unsigned;
4) **Overflow (O)**: indica se il numero è rappresentabile come numero signed.

Esercizio 05 - Soluzione
ALU

Realizziamo la nostra ALU a 1 bit tramite i blocchi che ci siamo già costruiti: