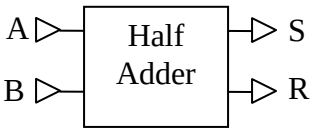
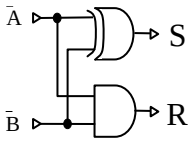
# Es. 05 - Soluzioni

1) **Addizionatore Half Adder (senza riporto in ingresso)**:

| A | B | S | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



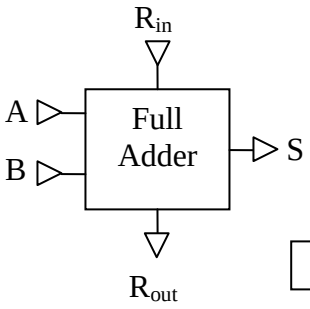



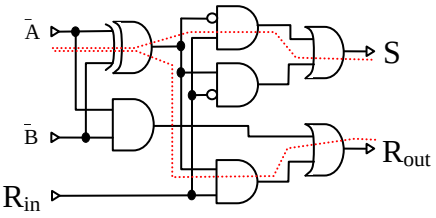

$S = A \oplus B \qquad R = A \cdot B$

N.Porte = 2 *Cammino Critico* S = 1 , R = 1

2) **Addizionatore Full Adder ( con riporto in ingresso )**:

| $R_{in}$ | A | B | S | $R_{out}$ |
|---|---|---|---|---|
| 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 |




N.Porte = 7 *Cammino Critico* S = 3 , R = 3

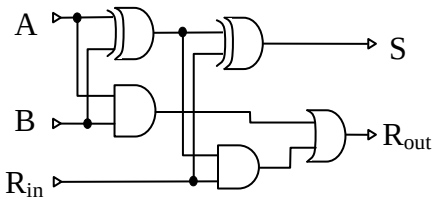
$S = \sim R_{in}\sim AB + \sim R_{in}A\sim B + R_{in}\sim A\sim B + R_{in}A\,B$
$= \sim R_{in}(\sim AB + A\sim B) + R_{in}(\sim A\sim B + A\,B)$
$= \sim R_{in}(A \oplus B) + R_{in}\sim (A \oplus B)$

$R_{out} = \sim R_{in}AB + R_{in}\sim AB + R_{in}A\sim B + R_{in}AB$
$= AB\,(\sim R_{in} + R_{in}) + R_{in}\,(\sim AB + A\sim B)$
$= AB + R_{in}(A \oplus B)$

Nota: **~A~B + A B** = ~A~B + A B = ~(~(~A~B + A B))
= ~(~(~A~B) ~ ( A B)) = ~((~~A+~~B) (~A+~B))
=~((A+B) (~A+~B)) = ~(A(~A+~B)+B(~A+~B))
=~(A~A+A~B+B~A+B~B) = ~(A~B+B~A) = **~ (A ⊕ B)**

**Semplificazione circuitale:**

$S = \sim R_{in}(A \oplus B) + R_{in}\sim (A \oplus B) = R_{in} \oplus (A \oplus B)$



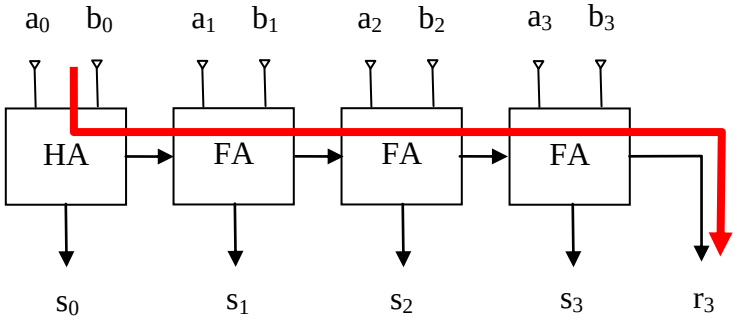

N.Porte = 5 *Cammino Critico* S = 2 , R = 3

***Nota**: le porte **AND e la OR possono essere pensate come "gate"** che lasciano "passare" il segnale presente su un terminale in funzione del segnale presente sull'altro. Diversamente **le porte XOR si comportano come "invertitori"**: il segnale presente su un terminale viene lasciato passare o invertito a seconda del segnale presente sull'altro.*

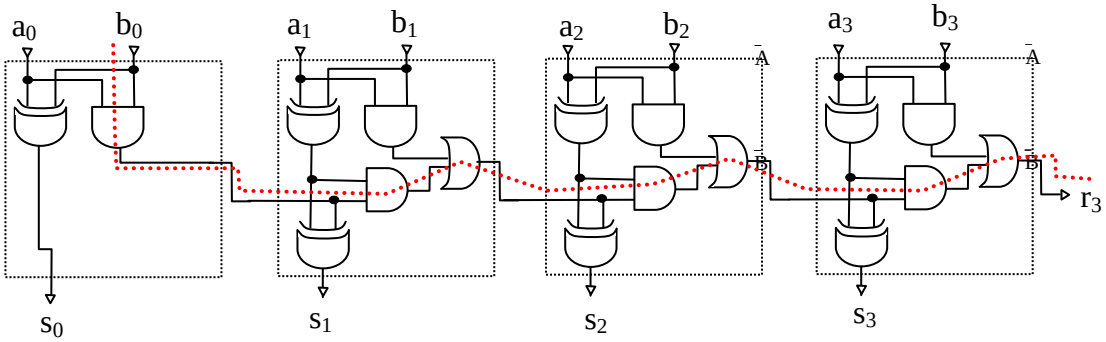
Massimo Marchi – marchi@dsi.unimi.it

### 3) Sommatore ad n bit (senza riporto in ingresso)

E' possibile realizzare un sommatore ad n bit usando un HA e n-1 FA collegati in cascata.



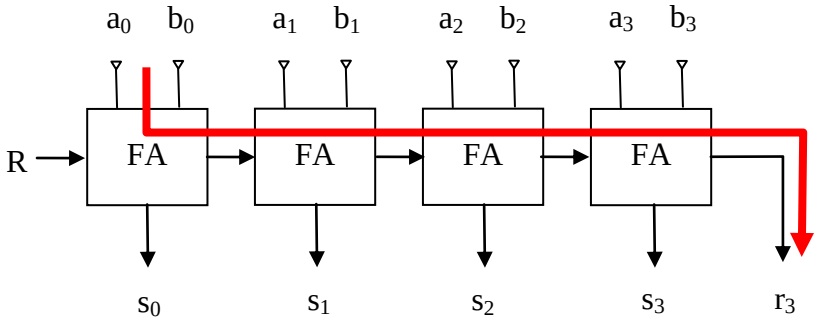

Il cammino critico è quello definito dalla propagazione dei riporti dal primo modulo all'ultimo. Se si esamina il circuito dei FA si nota che la porta XOR e le porta AND colegate direttamente agli ingressi $a_i$ $b_i$ si stabilizzano tutte nella prima unità di tempo. Ne segue che per propagare il riporto dall'ingresso $R_{in}$ all'ingresso $R_{out}$ di ogni sommatore occorrono due unità di tempo per ogni FA.




Il **cammino critico** quindi per questo sommatore è **1 + 2 * (n-1)** dove **n** è il numero di bit da sommare.

### 4) Sommatore ad n bit (con riporto in ingresso)

Normalmente si preferisce adottare un FA anche per il primo modulo. Questo permette di mettere in cascata più sommatori e di realizzare semplicemente un sottrattore binario in complemento a due (vedi più avanti).




In questo caso quindi il **cammino critico** è **3 + 2 * (n-1) = 1+2*n** poiché al primo FA occorrono 3 unità di tempo per generare il primo riporto.

Massimo Marchi – marchi@dsi.unimi.it

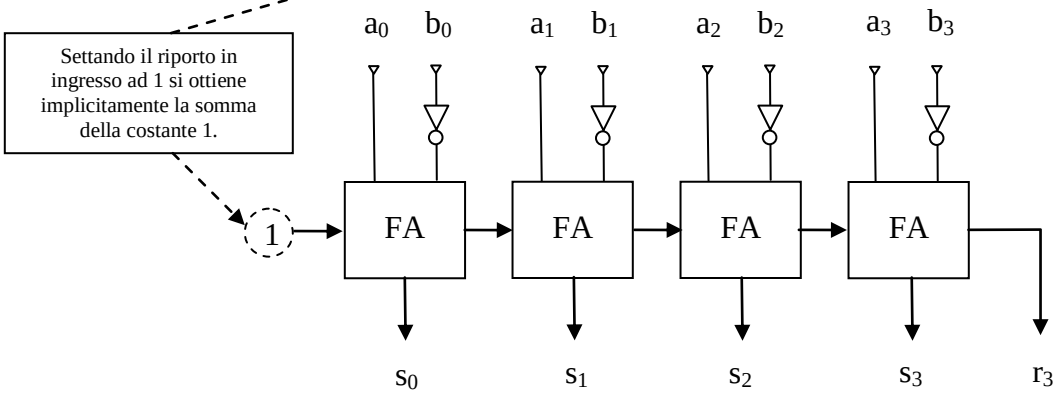
### 5) Sottrattori ad n bit

In binario la sottrazione ad n bit (con segno) può essere realizzata con una somma secondo la regola:

$$\mathbf{A - B = A + \sim B + 1}$$

dove **~B** è l'inversione bit a bit del secondo termine.

Adottando un sommatore con riporto in ingresso è possibile realizzare un sottrattore binario a n bit in questo modo:



6) **Problema dell'overflow**

Nel caso che il risultato di un addizione con segno ecceda il limite di rappresentazione della dimensione della parola si verifica un errore di segno (Overflow).

*Esempio:*

*$19_{10} + 15_{10} = 34_{10}$ se eseguita in aritmetica binaria con segno a 6 bit dà come risultato:*

| | | | | | | |
|---|---|---|---|---|---|---|
| *1* | *1* | *1* | *1* | *1* | | *R* |
| **0** | **1** | **0** | **0** | **1** | **1** | + |
| **0** | **0** | **1** | **1** | **1** | **1** | = |
| **1** | $^{1}$**0** | $^{1}$**0** | $^{1}$**0** | $^{1}$**1** | $^{1}$**0** | |

*$0\ 10011_2 + 0\ 01111_2 = 1\ 00010_2$ che in rappresentazione in complemento a due è un numero negativo , precisamente - $11110_2 = -30_{10}$ .*

*Analogamente si verifica lo stesso effetto di riporto errato sul bit di segno quando si sommano due numeri negativi in valore assoluto troppo grandi:*

Ex: $-17_{10} + -19_{10} = -35_{10}$ , *se eseguita in aritmetica binaria con segno a 6 bit dà come risultato:*

*$-17_{10} = -\ 10001_2$ (M&S) = $101111_2$ (CA2)*

*$-19_{10} = -\ 10011_2$ (M&S) = $101101_2$ (CA2)*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *1* | *1* | *1* | *1* | *1* | *1* | | *R* |
| | **1** | **0** | **1** | **1** | **1** | **1** | + |
| | **1** | **0** | **1** | **1** | **0** | **1** | = |
| ~~**1**~~ | $^{1}$**0** | **1** | $^{1}$**1** | $^{1}$**1** | $^{1}$**0** | $^{1}$**0** | |

*$1\ 01111_2 + 1\ 01101_2 = 0\ 11100_2$ che è un numero positivo , precisamente $+28_{10}$ .*

*Nel caso di somme miste, negativo con positivo e viceversa, questo problema non si pone poiché il risultato in valore assoluto sarà sempre al più grande come il più grande in valore assoluto dei due termini sommati.*

**Circuito per rilevare l'overflow:**

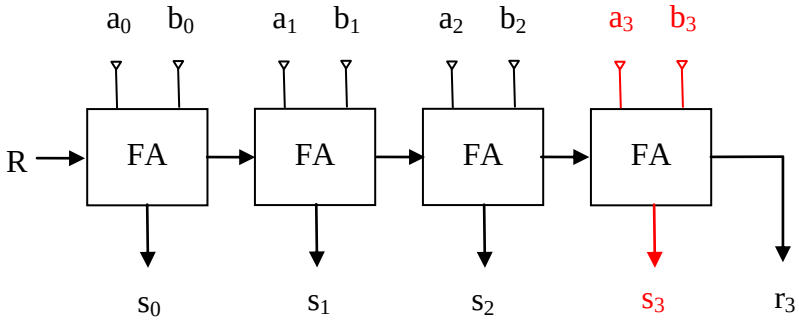
Da quanto visto sopra ne segue che l'overflow si verifica solo nei seguenti due casi:

*Segno di A = 0 e Segno di B = 0 e Segno del risultato = 1*

*oppure*

*Segno di A = 1 e Segno di B = 1 e Segno del risultato = 0*

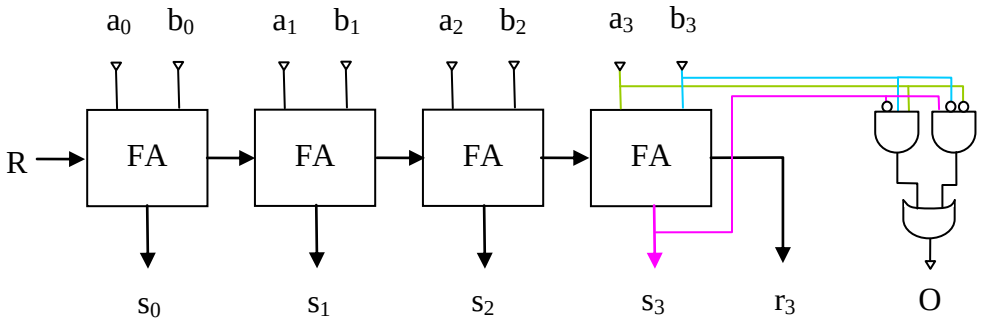
Tenendo presente il circuito Full-Adder a n bit,

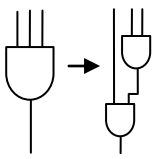



il circuito che realizza questo controllo deve sintetizzare la seguente forma tabellare:

| $s_{n-1}$ | $a_{n-1}$ | $b_{n-1}$ | Overflow |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

In forma SOP si può scrivere: $O = \sim s_{n-1}\, a_{n-1}\, b_{n-1} + s_{n-1} \sim a_{n-1} \sim b_{n-1}$



Per realizzare le porte AND a tre ingressi occorrono due porte in cascata. Ne segue che *il cammino critico di questo circuito per l'overflow* ha cammino critico 2 (le due porte AND a tre ingressi eseguite in parallelo) più la porta OR per un totale di 3.

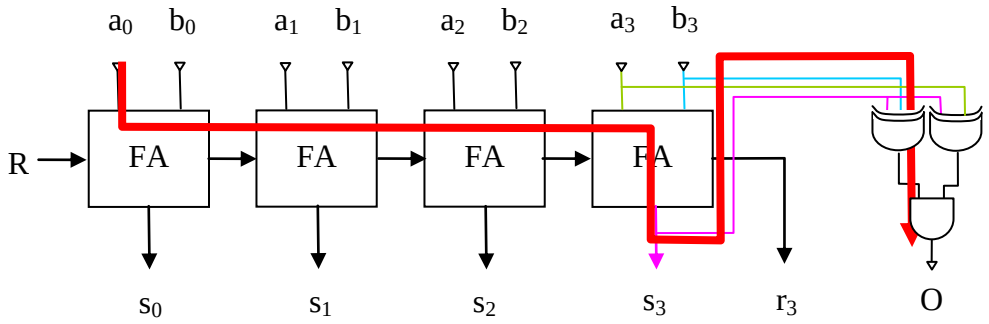
Massimo Marchi – marchi@dsi.unimi.it

Si può fare di meglio (assegnamo per comodità $S=S_{n-1}$ $A=a_{n-1}$ $B=b_{n-1}$):

$$O = \sim SAB + S\sim A\sim B = \sim SA\sim SB + S\sim AS\sim B = \sim SA\sim SB + 0 + 0 + S\sim AS\sim B$$
$$= \sim SA\sim SB + \sim SAS\sim B + S\sim A\sim SB + S\sim AS\sim B$$
$$= (\sim SB + S\sim B)\sim SA + S\sim A(\sim SB + S\sim B) = (S\oplus B)\sim SA + S\sim A(S\oplus B)$$
$$= (S\oplus B)(\sim SA + S\sim A) = (S\oplus B)(S\oplus A)$$

che ha cammino critico pari a **2**.

Ne segue che il cammino critico di un circuito Full-Adder con rilevamento dell'overflow definito dal tempo necessario per propagare i segnali in ingresso al circuito di rilevamento dell'overflow più il cammino critico del circuito stesso:




**Cammino critico:** primi n-1 stadi + ultimo stadio + Overflow = 1+2(n-1) + 2 + 2 = **2(n+1) + 1**

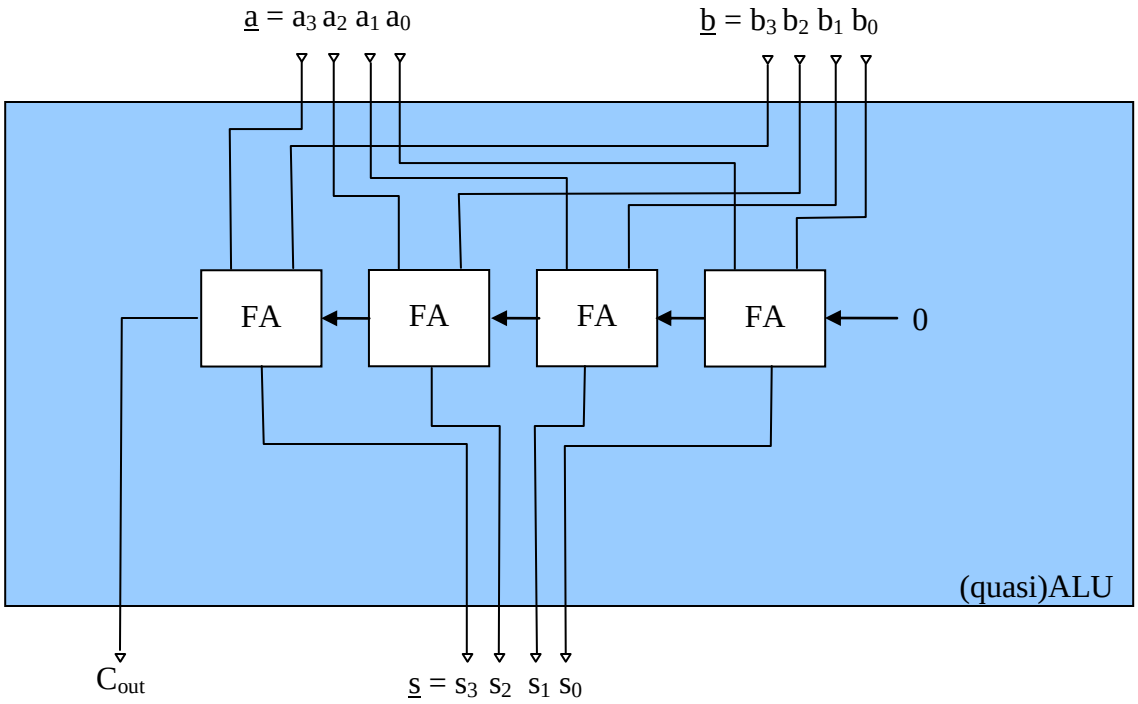
*Nota: Le porte XOR in questo caso si comportano come invertitori a comando: quando S è a 0 permettono ad A e B di giungere inalterati alla porta AND e di realizzare la parte alta della forma tabellare mentre quando S è uguale ad 1 invertono A e B realizzando la parte bassa, cioè ~A~B.*

*Nota: Il cammino critico del FA non passa più per l'ultimo resto poiché, dati i termini precedenti, il cammino critico per avere il bit di somma più il tempo del circuito di overflow, in totale 4, è maggiore del cammino critico per calcolare quest'ultimo resto, come visto sopra pari a 3.*

Massimo Marchi – marchi@dsi.unimi.it

## *7)* ALU: Arith metic Logic Unit. Un esempio a 4 bit

Consideriamo un sommatore a 4 bit (per semplicità senza CLA) e consideriamolo come un blocco unico che accetta in ingresso due parole da 4 bit **a** e **b** e restituisce in uscita 4 bit di risultato **s** ed il riporto dell'ultimo stadio $\mathbf{C_{out}}$ (*Carry Out*):




*Gli ingressi come l'uscita sono visualizzati con in bit più significativo, il bit di segno, a sinistra. Per semplicità, il sommatore è riportato con gli stadi invertiti. Lo stadio più a sinistra elabora ora i bit più significativi.*

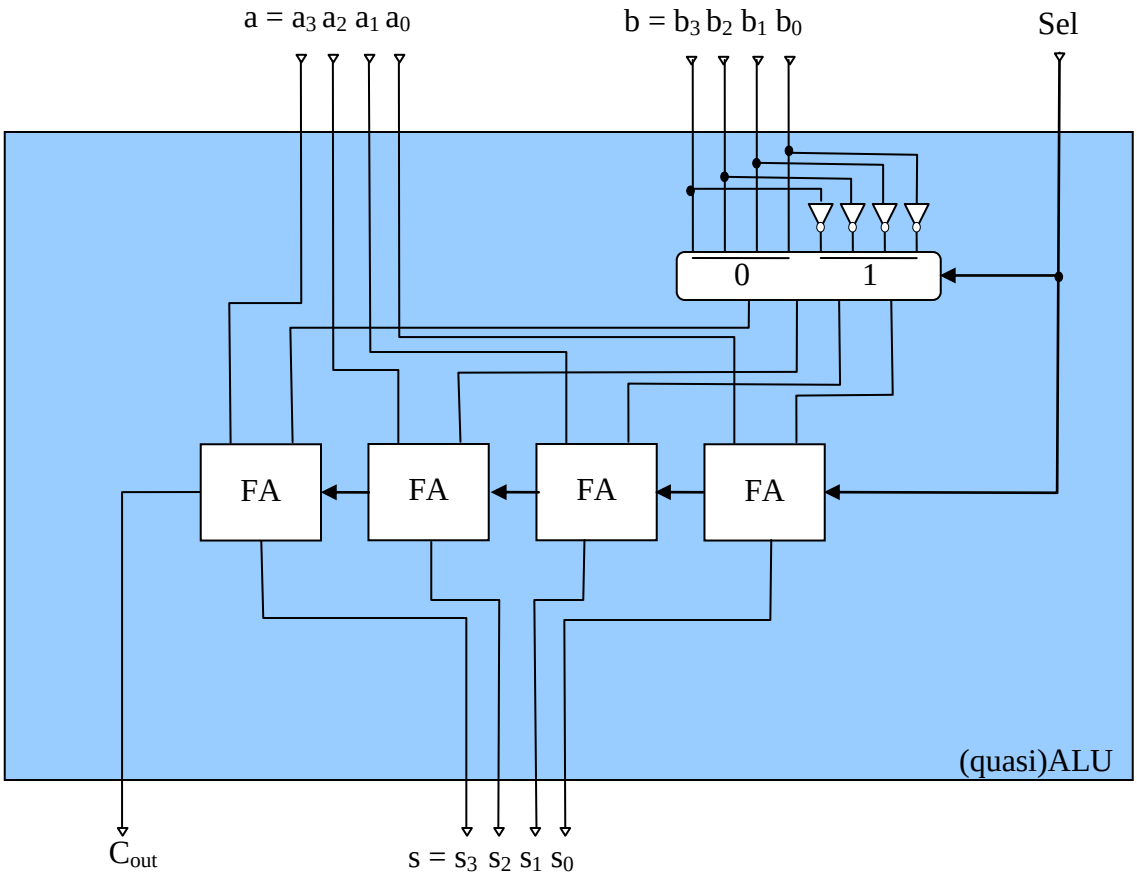
La (quasi)ALU così costruita realizza ovviamente una sola operazione, la somma, e quindi come tale non è necessaria nessuna linea di selezione dell'operazione da eseguire.

Supponiamo ora di voler aggiungere alla mia (quasi)ALU la possibilità di eseguire la sottrazione invece della somma a seconda del segnale presente su una linea di selezione **Sel**. Se **Sel=0** allora si desidera eseguire la somma **a + b** , se **Sel=1** allora si desidera eseguire la sottrazione **a – b**.
Date due parole binarie **a** e **b** si dimostra che:

$$\mathbf{a} - \mathbf{b} = \mathbf{a} + \mathbf{not}(\mathbf{b}) + \mathbf{1}$$

Porre il riporto in ingresso al primo stadio di un sommatore uguale a 1 equivale a sommare 1 al risultato finale. Ne segue che possiamo riorganizzare il circuito sommatore per eseguire all'occorrenza anche la sottrazione.

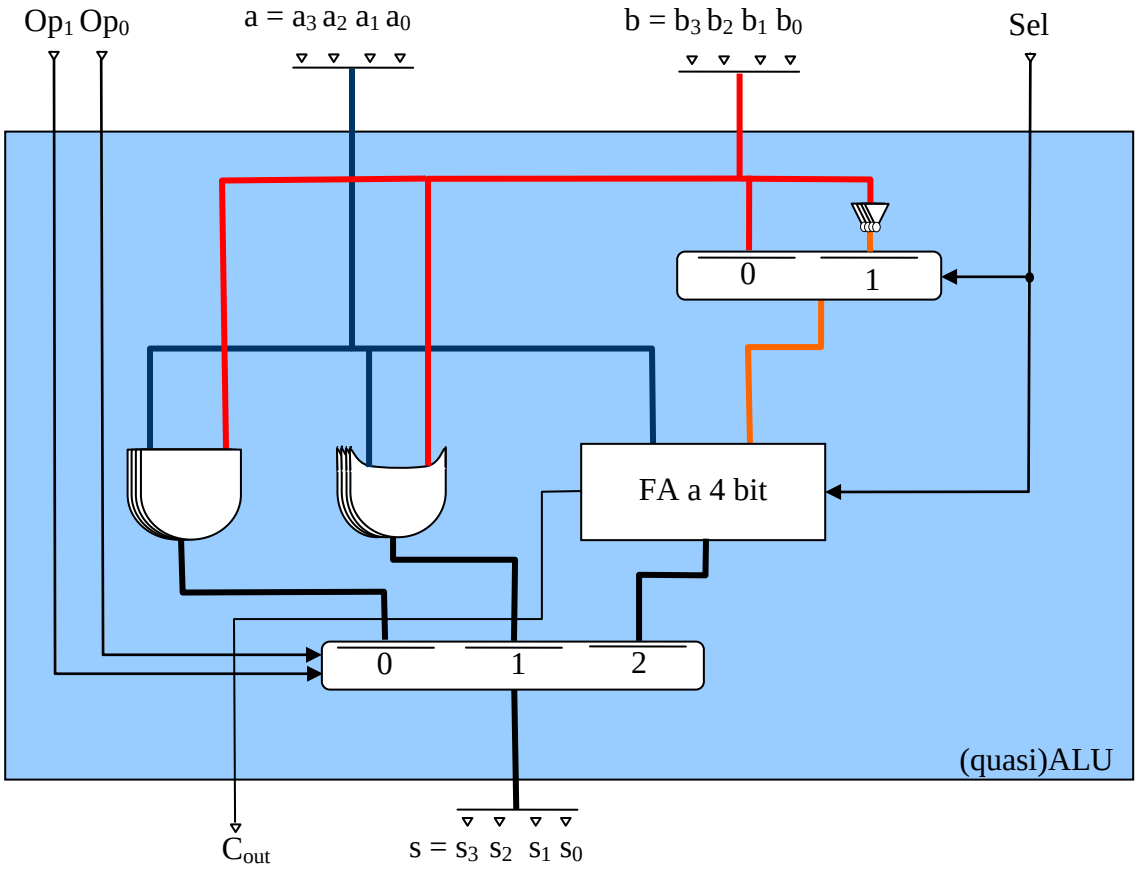
L'operazione di inversione *a comando* della parola **b** può essere ottenuta attraverso una batteria di porte NOT ed un selettore a 4 bit comandato da **Sel:**




La (quasi)ALU così progettata realizza le operazioni di somma e sottrazione.

Massimo Marchi – marchi@dsi.unimi.it

Sfruttando una batteria di porte AND, una batteria di porte OR e una nuova linea di selezione, questa volta a tre vie, possiamo aggiungere due nuove operazioni, AND e OR (*le linee in grassetto indicano bus a 4 bit,il FA come pure le prote AND e OR sono state raggruppate logicamente in blocchi*)




La (quasi)ALU così progettata può eseguire somme, sottrazioni, AND bit a bit ed OR bit a bit. Tramite i segnali di controllo **Sel**, **$Op_1$** e **$Op_0$**, i blocchi all'interno dell'ALU vengono *combinati* per ottenere in uscita il risultato dell'operazione desiderata. I segnali **$Op_1$** e **$Op_0$** selezionano quale risultato mandare verso l'uscita, se il risultato della AND nel caso **$Op_1$=0** e **$Op_0$=0,** o il risultato della OR per **01** oppure il risultato del sommatore per **10**, la somma o la differenza a seconda del segnale presente su **Sel** (nei casi AND e OR **Sel** non ha effetto).

Massimo Marchi – marchi@dsi.unimi.it

Supponiamo ora di voler dotare la (quasi)ALU della funzione di comparazione: date due parole di 4 bit **a** e **b**, **comp(a , b)** vale:

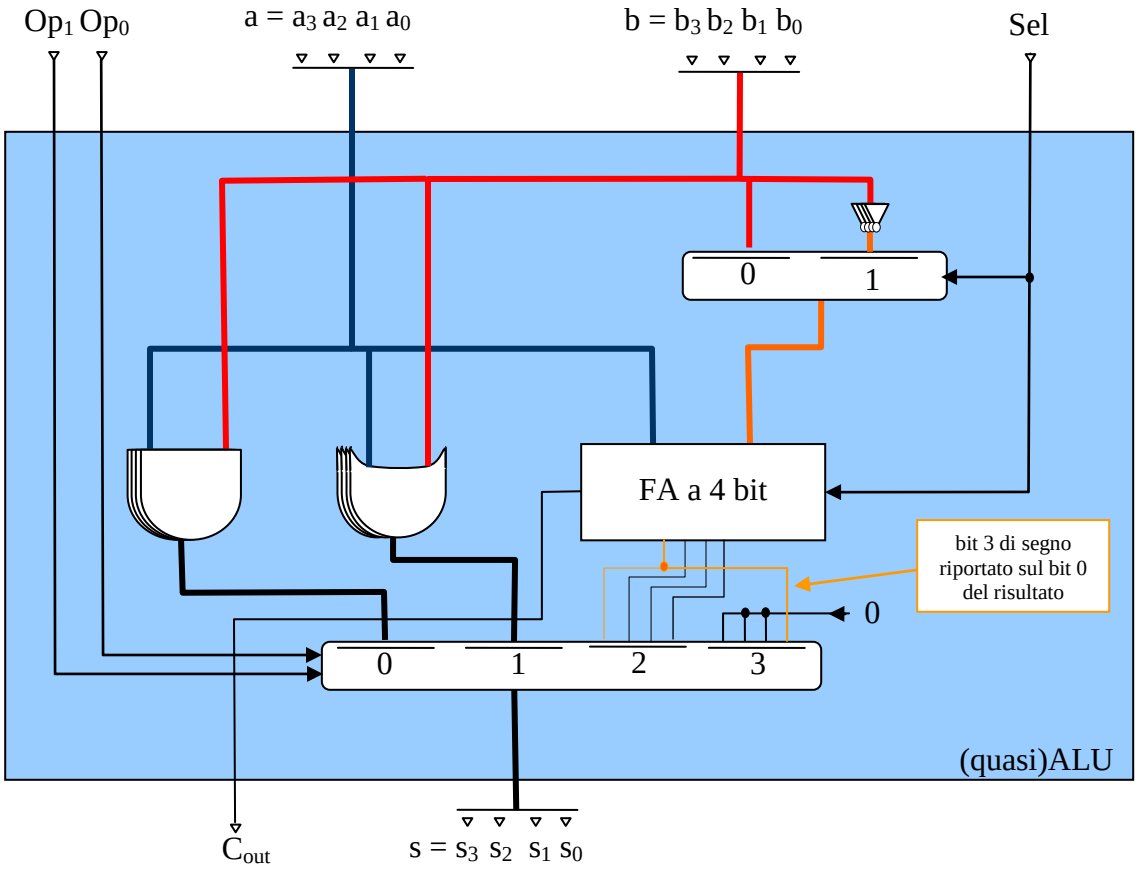
**1** se **a** < **b** , **0** se **a** >= **b**

Ora questo equivale a:

**1** se **a** - **b** < 0 , **0** se **a** – **b** >= 0

cioè **comp(a , b)** vale **1** se il risultato della differenza è negativo, **0** altrimenti. In altre parole, il risultato della comparazione è pari al valore del bit di segno della differenza tra le due parole **a** e **b**.

Per implementare questa funzione occorre che la ALU venga *programmata* per realizzare la differenza (controllo **Sel=1**) quindi, attraverso un nuovo percorso dei dati all'interno della ALU, il bit di segno deve essere riportato sul **bit 0** mentre tutti gli altri bit del risultato vanno posto a **0** *(dato il bit di segno* ***s*** *il risultato della operazione di comparazione è uguale a* ***000s****)*. Utilizziamo **$Op_1$=1** e **$Op_0$=1**, per indicare questo nuovo percorso:




La tabella delle operazioni eseguibili da questa (quasi)ALU è riassunta dalla tabella a fianco. Da notare che per le operazioni AND ed OR il valore del controllo **Sel** è ininfluente mentre la configurazione **Op=11** e **Sel=0** è senza significato (il risultato è il segno dell'addizione tra **a** e **b** ).

| $Op_1$ | $Op_0$ | Sel | Operazione |
|---|---|---|---|
| 0 | 0 | x | **a** AND **b** |
| 0 | 1 | x | **a** OR **b** |
| 1 | 0 | 0 | **a** + **b** |
| 1 | 0 | 1 | **a** - **b** |
| 1 | 1 | 1 | **comp(a,b)** |
| 1 | 1 | 0 | **????** |

Massimo Marchi – marchi@dsi.unimi.it