Aritmetica dei Calcolatori 2

Laboratorio di Architettura

13 aprile 2012

Operazioni logiche bit a bit

AND

IN		OUT
A	*B*	*A* AND *B*
0	0	0
0	1	0
1	0	0
1	1	1

OR

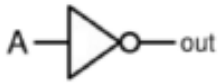
IN		OUT
A	*B*	*A* OR *B*
0	0	0
0	1	1
1	0	1
1	1	1

XOR

IN		OUT
A	*B*	*A* XOR *B*
0	0	0
0	1	1
1	0	1
1	1	0

NOT

IN	OUT
A	NOT *A*
0	1
1	0

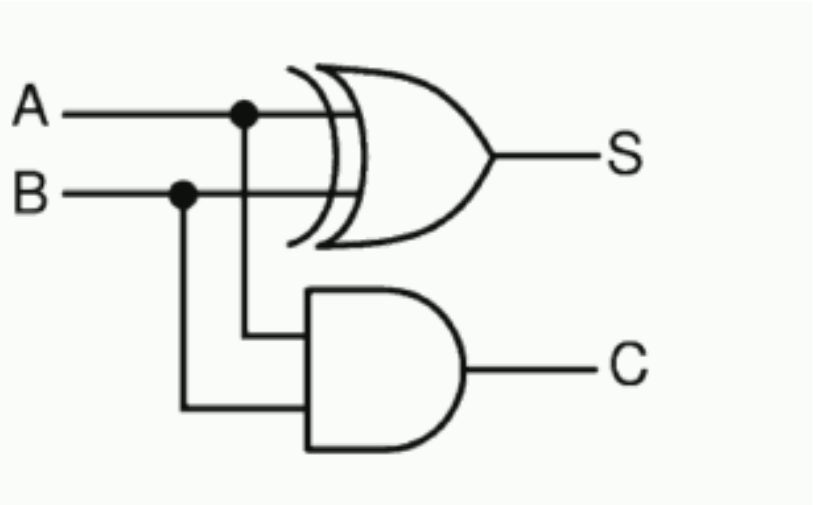


In C: & | $\wedge$ $\sim$ operano bit a bit!

Half Adder

Implementiamo la somma bit a bit:

IN		OUT	
A	*B*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

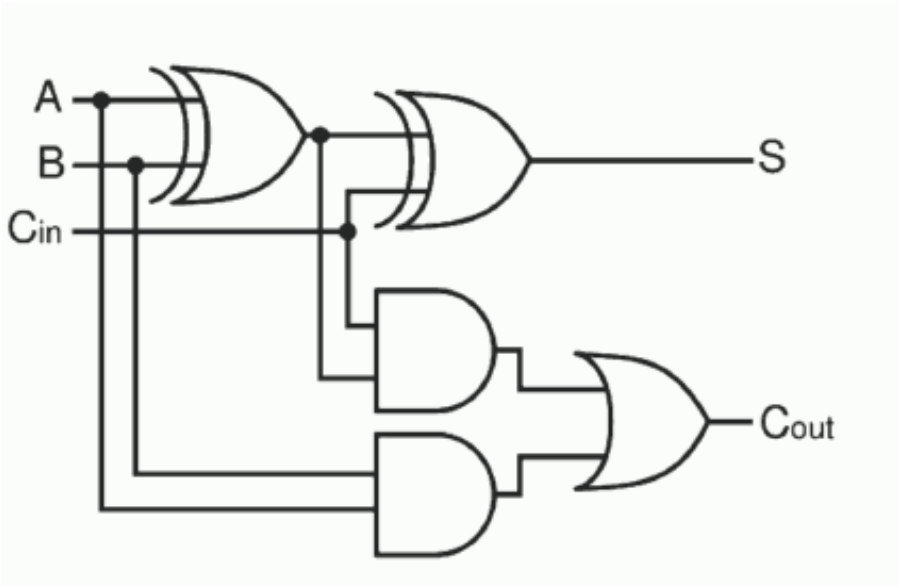


Half Adder riceve due bit *A* e *B* in input, calcola il bit di somma *S* e il bit di riporto *C*.

Full Adder

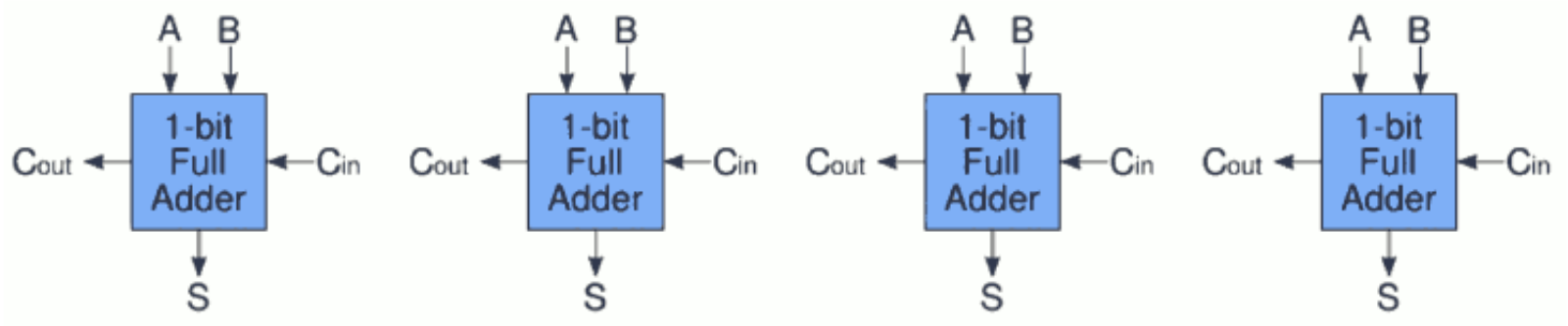
Ora, consideriamo il bit di riporto C_{in}, calcolato da una somma precedente:

IN			OUT	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full Adder riceve tre bit A, B e C_{in} in input, calcola il bit di somma S e il bit di riporto C_{out}.

Somma binaria

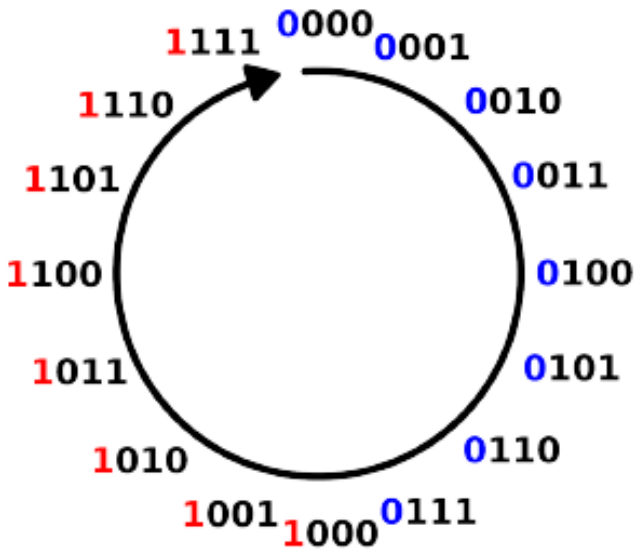


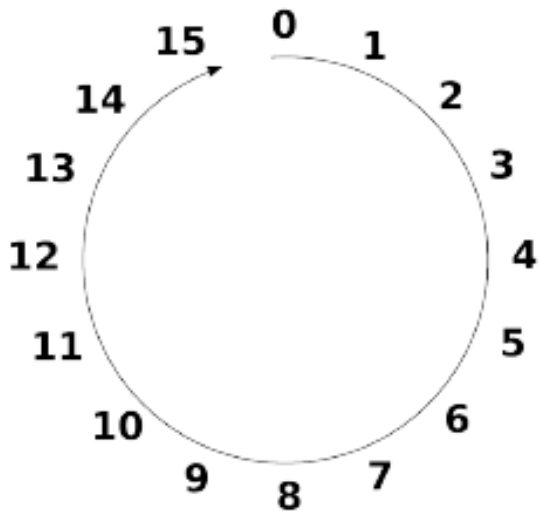
Per ottenere un sommatore a N bit, basta collegare N Full Adder: in questo modo si esegue la classica somma in “colonna”.

Rappresentazione binaria (senza segno)

Come si rappresentano i numeri interi senza segno?

Abbiamo già visto come si rappresentano i numeri interi non negativi:
con 4 bit possiamo rappresentare 16 numeri: $[0, 2^4 - 1]$

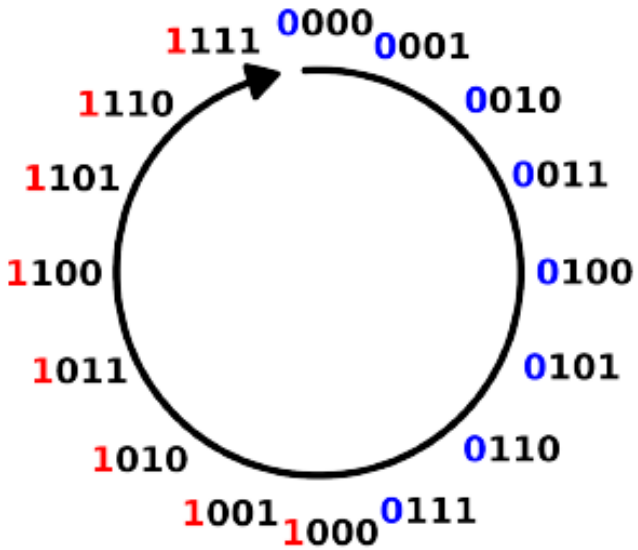


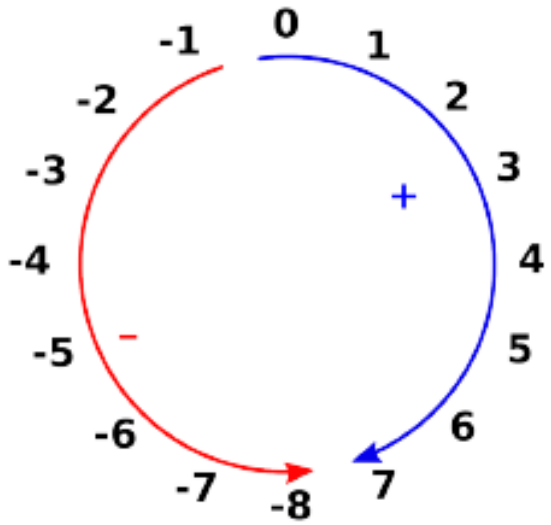


Rappresentazione binaria (con segno)

Come si rappresentano i numeri interi con segno?

Per memorizzare il segno, si utilizza il bit più significativo:
1 per i numeri negativi, 0 per lo zero e i numeri positivi.
In questo modo si possono rappresentare gli interi in $[-2^3, 2^3 - 1]$





Esempio

Per convertire un numero con segno in base 2 utilizzando N bit:

- osservare il segno
- convertire il suo modulo in base 2
- se negativo, calcolare il suo complemento a 2: negare ogni bit e sommare 1.

Esempio:

- $(-28)_{10} = -(28)_{10}$
- $(28)_{10} = (011100)_2$
- $(011100)_2 \rightarrow (100011)_2$
  $(100011)_2 + (000001)_2 = (100100)_2$

Estensione del segno

Immaginiamo di avere due interi con segno, rappresentati su un numero di bit differente, vogliamo calcolarne la somma.

Esempio:
$(30)_{10} = (00011110)_2$
$(-28)_{10} = (100100)_2$

Prendiamo il numero rappresentato con il minor numero di bit: consideriamo il bit più significativo e lo copiamo a sinistra tante volte quanti sono i bit mancanti per arrivare alla rappresentazione dell'altro numero.

Esempio:
$(30)_{10} = (00011110)_2$
$(-28)_{10} = (100100)_2 \rightarrow (11100100)_2$

Ora possiamo fare la somma:

$$\begin{array}{ll} (00011110)_2 & \\ (11100100)_2 & \\ (00000010)_2 & = (2)_{10} \end{array}$$

Riassumendo:

- Cambio di segno
  Dato un numero in notazione binaria, per cambiare il segno basta calcolare il complemento a due (negare bit a bit e sommare 1)
- Estensione del segno
  Dato un numero in notazione binaria su N bit, portarlo a M bit, con M>N

Operazione di shift

Data una sequenza di bit, lo shift a destra di s bit equivale a una divisione per 2^s, mentre lo shift a sinistra di s bit equivale a una moltiplicazione per 2^s.

in C:

```
n >> s;  // shift a destra di s posizioni
n << s;  // shift a sinistra di s posizioni
```

Moltiplicazione I

Se osserviamo l'algoritmo di moltiplicazione in colonna:

```
   123
 x 456
 =====
   738  (this is 123 x 6)
  615   (this is 123 x 5, shifted one position to the left)
+ 492   (this is 123 x 4, shifted two positions to the left)
 =====
 56088
```

```
     1011   (this is 11 in decimal)
   x 1110   (this is 14 in decimal)
   ======
     0000   (this is 1011 x 0)
    1011    (this is 1011 x 1, shifted one position to the left)
   1011     (this is 1011 x 1, shifted two positions to the left)
 + 1011     (this is 1011 x 1, shifted three positions to the left)
 =========
  10011010  (this is 154 in decimal)
```

Quindi, nel caso binario, la moltiplicazione si può implementare come successione di somme e shift.

Moltiplicazione II

Nel libro di testo trovate un esempio di moltiplicazione $(3)_{10} \times (2)_{10}$:

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ no operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ no operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Esercitazione

Implementare le funzioni:

1. int fullAdder(int A, int B, int Cin);
   Implementazione del Full Adder: somma 3 bit memorizzati rispettivamente nel bit meno significativo (LSB) di A, B, Cin, e ritorna un intero con :

somma nel bit di indice 0 (LSB)
carry nel bit di indice 1

2. int wordAdder(int X, int Y);
   Sfrutta la funzione fullAdder, ritorna la somma binaria di X e Y, operandi interi con segno da considerare a 8 bit. Nel bit successivo a MSB del valore di ritorno (considerando 8 bit) memorizza il carry bit calcolato.
3. int mult(int X, int Y);
   Sfrutta la funzione wordAdder, ritorna il prodotto di X e Y, operandi interi con segno da considerare a 8 bit. A quanti bit potrá essere il valore di ritorno? Non può usare somme ($+$) o prodotti ($*$)!!!.

Requisiti

Requisiti fondamentali:

- Potete usare gli operatori bit a bit ($\& \mid \wedge \sim$) e lo shift ($>>$ e $<<$), ma non somma ($+$) e prodotto ($*$). Si puó usare l'incremento per i contatori dei cicli for (i++).
- Il prototipo delle funzioni non va cambiato per nessun motivo.
- Non devono essere presenti ulteriori funzioni oltre le 3 definite nella slide precedente.
- Non é richiesto alcun input interattivo.

Test finale

Come test per le funzioni implementate, si scriva una funzione main() che chiami wordAdder e mult fornendo 5 esempi, stampare a video parametri e risultato

```
me@mylaptop:~$ ./es2
Addizione:
          12 +         -73 =         -61
         115 +         -89 =          26
         -37 +         100 =          63
          83 +          98 =         181
         -65 +          -3 =         -68
Moltiplicazione:
          12 *         -73 =       -8832
         115 *         -89 =      -10235
         -37 *         100 =       -3700
          83 *          98 =        8134
         -65 *          -3 =         195
```

Consegna

L'esercitazione da consegnare è formata da:

1. un file in linguaggio C di nome es2.c, adeguatamente commentato
2. un file in formato testo di nome es2.out contenente la cattura dell'output del programma di test Usate la redirezione verso file:

```
me@mylaptop:~$ ./es2 > es2.out
```

Oppure copiate l'output e incollate su file di testo.

- NON usate la cattura di immagini da schermo: copiate ed incollate usando un editor di testo
- NON includete file BINARI o creati con WORD, EXCEL...
- NON verranno considerate mail con piu' di 2 file allegati!

Spedite tutto a arc1@fe.infn.it entro le ore 23:59:59 di giovedí 26 aprile.
L'oggetto della mail deve essere nella forma:
LAB1-N#esercitazione-#gruppo
(es: LAB1-N2-99)