Operatori bitwise

[Articolo del sito: http://www.settorezero.com/wordpress/operazioni-algebriche-booleane-come-si-utilizzano-gli-operatori-and-or-xor-e-not-e-come-applicarli-ai-registri-ad-8-o-piu-bit/]

- 1 **AND**: Ottenere lo stato di un singolo bit / Impostare a zero un singolo bit
- 2 **OR**: Impostare ad uno un singolo bit di un registro rimanendo invariati gli altri bit
- 3 **XOR**: Invertire lo stato di un bit rimanendo invariati gli altri
- 4 **NOT**: Impostare a zero un singolo bit di un registro rimanendo invariati gli altri bit, oppure invertire lo stato di tutti i bit.

L'algebra di boole è molto utilizzata in informatica e nella programmazione dei microcontrollori perchè permette di effettuare operazioni sui singoli bit, settare/ricavare lo stato di alcuni bit di un registro (formato da più bit) utilizzando "maschere" ed effettuando opportune operazioni e tante altre cose utili. In questo articolo ci occuperemo degli operatori AND, OR, XOR e NOT. Rinfreschiamo un po' la memoria ricordando l'effetto di questi operatori su due bit con questa tabella:

Bit 1	Bit 2	and	or	xor
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

L'operatore **AND** in C viene anche scritto come **&** ed è detto **prodotto logico.** Quando viene effettuato l'AND tra due bit, il risultato varrà VERO (1) soltanto se entrambi i bit valgono 1, in tutti gli altri casi il risultato sarà FALSO.
L'operatore **OR** in C viene anche scritto come **|** (pipe) ed è detto **somma logica**. Quando viene effettuato l'OR tra due bit, il risultato varrà VERO se almeno uno dei due bit vale VERO.
L'operatore **XOR** (OR esclusivo) in C viene anche scritto come **^** (elevamento a potenza) ed è detto **somma modulo 2**. Quando viene effettuato l'XOR tra due bit, il risultato varrà VERO se soltanto uno dei due bit vale VERO.

Guardando la tabella in alto la comprensione dovrebbe risultare abbastanza semplice.

Abbiamo infine l'operatore **NOT** (negazione): in C viene anche scritto come **~** (tilde) ed accetta un solo operando, invertendone lo stato:

bit	not
0	1
1	0

> Il simbolo di ~ (tilde) si ricava tenendo premuto ALT e digitando in sequenza, sul tastierino numerico, i numeri 1-2-6 e quindi rilasciando ALT.

Vediamo invece adesso come sfruttare opportunamente tali operatori su **registri ad 8 bit** per ottenere ciò che vogliamo.

AND: Ottenere lo stato di un singolo bit

Supponiamo di avere un registro il cui valore decimale è 153, in binario tale valore equivale a 10011001. Bene, vogliamo sapere, con un'operazione logica, qual'è lo stato del bit n° 4, ad occhio, scrivendo il numero in formato binario vediamo subito che il bit n°4 vale 1:

Numero BIT:	8	7	6	5	4	3	2	1
Valore del BIT:	1	0	0	1	1	0	0	1

(153)

Ma come fare per dire al microcontrollore: controllami il registro e vedi se il bit n°4 vale 1 oppure zero? E' molto semplice: utilizzeremo il prodotto logico ed una **maschera**. Sappiamo che l'operatore AND ci restituisce 1 soltanto quando entrambi i bit valgono 1, utilizzeremo pertanto una maschera, ovvero un numero composto da 8 bit, in cui tutti i bit valgono zero e il bit n°4 vale 1:

Numero BIT:	8	7	6	5	4	3	2	1	
Valore del BIT:	1	0	0	1	1	0	0	1	(153)
Maschera:	0	0	0	0	1	0	0	0	
Risultato con &:	0	0	0	0	1	0	0	0	(8)

Abbiamo in pratica eseguito l'operazione:

```
10011001b & 00001000b
```

e abbiamo ottenuto

```
00001000b
```

(0vvero 8 in decimale, che è lo stesso valore della maschera!) come risultato. Mettiamo invece il caso che il bit che ci interessava ricavare (il n°4) valesse zero: il risultato dell'AND con la nostra maschera sarebbe stato zero. Quindi si capisce che : **effettuando l'AND di un valore (di un registro) con una maschera in cui tutti i bit sono impostati a zero tranne il bit n° N, il risultato sarà zero (falso) se il bit N valeva zero, il risultato sarà >0 (oppure, che è lo stesso: uguale al valore della maschera) se il bit N valeva 1.**

In pratica, in C indicando con REG il registro di cui vogliamo sapere il valore del bit n° N e con MASK il valore dato alla nostra maschera (con tutti i bit pari a zero tranne il bit n° N), avremmo scritto una cosa del genere:

```
if (REG & MASK) == MASK
{
// il bit è impostato a uno
}
else
{
// il bit è impostato a zero
}
```

OR: Impostare ad uno un singolo bit di un registro rimanendo invariati gli altri bit

Supponiamo di avere il nostro registro, in cui vogliamo impostare ad uno soltanto un singolo bit ma rimanendo invariati tutti gli altri. In questo caso ricorreremo all'operatore OR utilizzando una maschera in cui tutti i bit valgono zero, tranne il bit che intendiamo settare ad uno. Supponendo di voler operare sempre sul bit n°4, faremo ad esempio:

Numero BIT:	8	7	6	5	4	3	2	1
Valore del BIT:	1	0	0	1	0	0	0	1
Maschera:	0	0	0	0	1	0	0	0
Risultato con |:	1	0	0	1	1	0	0	1

Come vedete, il risultato è uguale a quello di partenza, tranne che per il fatto che il bit n°4 adesso vale 1. Ovviamente se il bit di partenza era 1, avremmo ottenuto uno comunque.

In C avremmo scritto:

```
REG = REG | MASK
```

XOR: Invertire lo stato di un bit rimanendo invariati gli altri

Abbiamo sempre il nostro registro, nel quale vogliamo invertire lo stato di un bit, ovvero lo vogliamo mettere ad uno se valeva zero, oppure a zero se valeva uno. Si utilizzerà sempre la stessa maschera (tutti zero, e il bit che ci interessa messo ad 1) ma utilizzando l'operatore XOR:

Numero BIT:	8	7	6	5	4	3	2	1
Valore del BIT:	1	0	0	1	0	0	0	1
Maschera:	0	0	0	0	1	0	0	0
Risultato con ^:	1	0	0	1	1	0	0	1

Come vedete lo stato del bit 4 è stato invertito (valeva zero prima dell'operazione, adesso vale 1).

In C:

```
REG = REG ^ MASK
```

NOT: Impostare a zero un singolo bit di un registro rimanendo invariati gli altri bit**,** oppure **invertire lo stato di tutti i bit.**

Se vogliamo invertire lo stato di tutti i bit di un singolo registro (mettere a zero quelli che valevano uno, e viceversa), l'operazione è semplice: si assegna al registro il valore del registro negato:

```
REG = NOT REG          oppure          REG = ~REG
```

Se invece vogliamo impostare a zero un singolo bit, rimamendo invariati gli altri? In questo caso in realtà non si utilizzerà una singola operazione ma due operazioni successive: un NOT e un AND. Abbiamo sempre il nostro registro di partenza, nel quale vogliamo impostare a zero il bit n°4:

Numero BIT:	8	7	6	5	4	3	2	1
Valore del BIT:	1	0	0	1	1	0	0	1

Bene, sfruttiamo sempre la nostra maschera con tutti zero, e soltanto 1 sul bit n°4, fatta in questo modo:

Numero BIT:	8	7	6	5	4	3	2	1
Maschera:	0	0	0	0	1	0	0	0

Effettuiamo la prima operazione: la negazione della maschera, in questo modo tutti i bit che erano zero varranno uno e viceversa:

Numero BIT:	8	7	6	5	4	3	2	1
Maschera:	0	0	0	0	1	0	0	0
NOT Maschera:	1	1	1	1	0	1	1	1

Adesso utilizzeremo l'AND sul nostro registro, combinandolo con la maschera negata:

Numero BIT:	8	7	6	5	4	3	2	1
Valore del BIT:	1	0	0	1	1	0	0	1
NOT Maschera:	1	1	1	1	0	1	1	1
Risultato con &:	1	0	0	1	0	0	0	1

Come vedete sono rimasti invariati tutti i bit, tranne il n°4 che adesso è "spento" (ovviamente sarebbe rimasto spento anche se lo era all'inizio…).

In C scriveremo:

```
REG = REG & ~MASK
```