

Architettura degli elaboratori

Laboratorio



Linguaggio C – Ripasso operazioni bit a bit (bitwise)

Dr. Luca Dariz <luca.dariz@unife.it>
Prof. Matteo Manzali <matteo.manzali@unife.it>

Operazioni bitwise

- Agiscono sulla *rappresentazione binaria* delle variabili, su ogni singolo bit o a coppie (0b notazione binaria):

a = 6 = 0b00000110

b = 98 = 0b01100010

Diverse operazioni bitwise:

NOT	
b1	~b1
0	1
1	0

AND		
b1	b2	B1 & b2
0	0	0
0	1	0
1	0	0
1	1	1

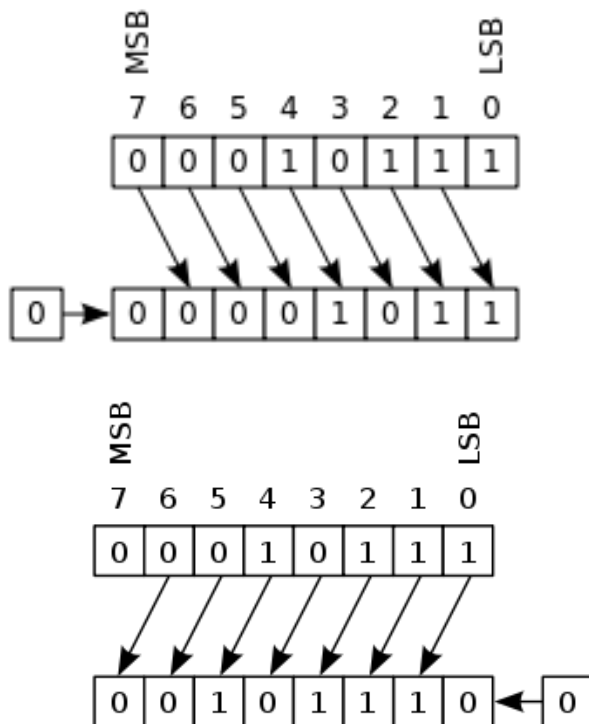
OR		
b1	b2	b1 b2
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
b1	b2	b1 ^ b2
0	0	0
0	1	1
1	0	1
1	1	0

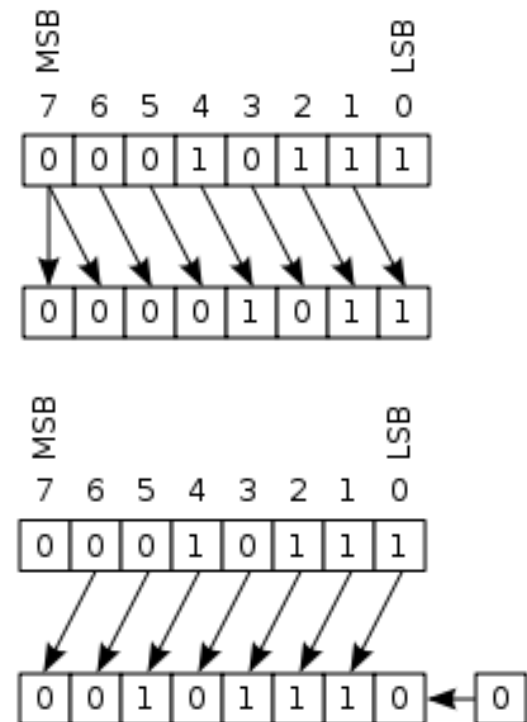
Operazioni bitwise

- Le operazioni di shift agiscono complessivamente su tutti i bit di una variabile, traslandoli a destra o sinistra

Shift logico



Shift aritmetico



Operazioni bitwise e shift in C

- Ufficialmente definite solo per interi senza segno

Operazione	Espressione in C
a negato bitwise	<code>~a</code>
a OR b	<code>a b</code>
a AND b	<code>a & b</code>
a XOR b	<code>a ^ b</code>
shift a destra di n bit	<code>a >> n</code>
shift a sinistra di n bit	<code>a << n</code>

- È comunque possibile forzare un cast tra tipi diversi, oppure operare direttamente su tipi signed

Operazioni bitwise

- NOT

NOT	
b1	~b1
0	1
1	0

a = 00000110
~a = 11111001

- AND

AND		
b1	b2	B1 & b2
0	0	0
0	1	0
1	0	0
1	1	1

a 00000110 &
b 01100010 =
a & b 00000010

Operazioni bitwise

- OR

OR		
b1	b2	b1 b2
0	0	0
0	1	1
1	0	1
1	1	1

```
a      000000110 |
b      011000010 =
a | b  01100110
```

- XOR

XOR		
b1	b2	b1 ^ b2
0	0	0
0	1	1
1	0	1
1	1	0

```
a      000000110 ^
b      011000010 =
a ^ b  011001100
```

Operazioni bitwise in C

`a = 0b00000110 = 0x06 → maschera`

`b = 0b01100010 = 0x62 → byte da mascherare`

- Mascheramento bit:

`c = b & a; → c = 0b00000010 = 0x02`

- Inversione bit:

`c = ~b; → c = 0b10011101 = 0x9D`

Operazioni bitwise in C

`a = 0b00000110 = 0x06 → maschera`

`b = 0b01100010 = 0x62 → byte da mascherare`

- Set bit:

`c = b | a; → c = 0b01100110 = 0x66`

- Reset bit:

`c = b & (~a); → c = 0b01100000 = 0x60`

Operazioni di shift in C

Utili per ottimizzare alcune operazioni matematiche:

- Moltiplicazione per potenze di 2 (shift a sinistra)

`a = 0b000000100 = 0x04 = 4`

`b = 0b000000110 = 0x06 = 6`

`c = b<<a; → c = 0b01100000 = 0xC0 = 192 = 6x16`

`shift sx di 4 = moltiplicazione per  $2^4=16$`

- Divisione per potenze di 2

`a = 0b000000010 = 0x02 = 2`

`b = 0b00011000 = 0x18 = 24`

`c = b>>a; → c = 0b00000110 = 0x06 = 6 = 24/4`

`shift dx di 2 = divisione per  $2^2=4$`

Rappresentazione binaria

- **Esercizio 1:**
Creare una funzione che, dato un intero senza segno `uint32_t`, ne stampi la rappresentazione binaria.

Suggerimento: usare operazioni di mascheramento e shift
- Questa funzione sarà molto comoda negli esercizi successivi!

Ripasso floating point

- Secondo lo standard IEEE-754 un numero in virgola mobile a precisione singola/doppia è rappresentato in 32/64 bit di cui, a partire da MSB:
 - 1 bit di segno
 - 8 o 11 bit di esponente
 - 23 o 52 bit di mantissa

Un numero reale x è rappresentato da segno S , esponente E e mantissa M secondo la relazione:

$$x = (-1)^S \cdot 2^E \cdot M$$

dove l'esponente è codificato come $E - 127$ e la mantissa è la parte decimale del numero $M = 1.xxxxxxxxxxxxx$

Rappresentazione numeri floating point

- Esercizio 2:
Scrivere una funzione che estragga S, E ed M da un numero floating point (a scelta tra precisione singola o doppia), e una funzione che dati S, E ed M crei il numero floating point corrispondente. Scrivere almeno 2 funzioni di test per ogni funzione.
- Suggerimento: usare un cast tra puntatori per passare da una *rappresentazione binaria* all'altra siccome un cast di variabili converte il *valore*. Ad esempio, per ottenere in *xi* la rappresentazione binaria di un float *xf*:

```
float xf=1.0;
uint32_t xi;
uint32_t *ptr_helper;

ptr_helper = (uint32_t*)&xf;
xi = *ptr_helper;
```

Ottimizzazione operazioni

- Esercizio 3:

L'architettura a basso consumo MSP430 non ha un'istruzione dedicata per la moltiplicazione tra interi. Si implementi in C una versione ottimizzata della moltiplicazione tra interi senza segno riducendola a somma di moltiplicazioni per potenze di 2, ovvero:

$$c = a \cdot b = a \cdot (b_n 2^n + b_{n-1} 2^{n-1} + \dots + b_1 2^1 + b_0 2^0)$$

dove i coefficienti b_n corrispondono alla rappresentazione in base 2 di b . Si implementi anche una versione che supporti interi con segno. Si implementino almeno 2 test per ogni funzione.

Suggerimento: nella versione signed può essere comodo trattare separatamente il bit di segno