

Unità didattica: Operatori bitwise: esempi

[2-AC]

Titolo: Esempi di applicazione degli operatori bitwise

Argomenti trattati:

- ✓ Visualizzazione degli 8 bit di una variabile di tipo char
- ✓ Moltiplicazione e divisione intera per 2 mediante operatori di shift
- ✓ Scambio del contenuto di due variabili senza variabile d'appoggio
- ✓ Uso di costanti ottali o esadecimali per azzerare parte di una variabile
- ✓ Uso di “maschere” per estrarre alcuni bit da una variabile

Prerequisiti richiesti: **fondamenti del linguaggio C, operatori bitwise**

Come visualizzare in C il contenuto di una variabile **char**?

```
#include <stdio.h>
void main()
{unsigned char C;
  C='z';
  printf("char = %c\tdec = %d\thex = %02x\n",C,C,C);
}
```

output

char = z dec = 122 hex = 7a

```
printf("char = %C\tdec = %d\thex = %02x\n",C,C,C);
```

visualizza come carattere

visualizza in decimale

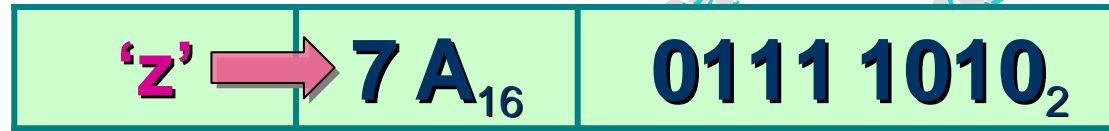
visualizza in esadecimale

come visualizzare in **binario**?

Esempio 1:

Come aggiungere all'output, mediante operatori bitwise, la visualizzazione degli 8 bit di un char?

esempio `Z='z';`



suggerimento:

1. separare le singole cifre esadecimali (gruppo di 4 bit);
2. convertire il valore di ogni cifra nella stringa di bit corrispondente.

1. separare le singole cifre esadecimali;

0111 1010

0000 0111 = 7

come?

`C=Z>>4;`

0111 1010

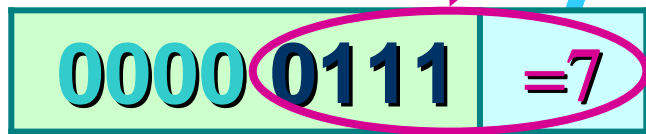
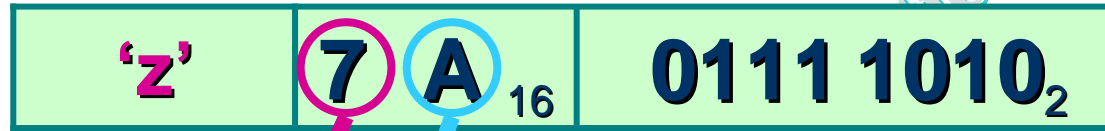
0000 1010 = 10

come?

`C=Z<<4; C=C>>4;`

2. convertire il valore di ogni cifra nella stringa di bit corrispondente;

esempio



```
C=Z>>4;  
...bit[C]
```

bit[7]



```
C=Z<<4;  
C=C>>4;  
...bit[C]
```

bit[10]

bit
"0000"
"0001"
"0010"
"0011"
"0100"
"0101"
"0110"
"0111"
"1000"
"1001"
"1010"
"1011"
"1100"
"1101"
"1110"
"1111"

3. usare il valore ottenuto direttamente come indice nell'array **bit** di stringhe;

versione semplificata

```
... void main()
{unsigned char bit[16][4]={ '0','0','0','0', '0','0','0','1', '0','0','1','0', '0','0','1','1',
                             '0','1','0','0', '0','1','0','1', '0','1','1','0', '0','1','1','1',
                             '1','0','0','0', '1','0','0','1', '1','0','1','0', '1','0','1','1',
                             '1','1','0','0', '1','1','0','1', '1','1','1','0', '1','1','1','1'};

unsigned char C, dx, sx;
C = 'z';
sx=C>>4;
dx=(C<<4); dx=dx>>4;
printf("char = %c\tdec = %d\thex = %02x\n",C,C,C);
printf("\t\tbin = %c%c%c%c %c%c%c%c\n",bit[sx][0],bit[sx][1],bit[sx][2],bit[sx][3],
                                             bit[dx][0],bit[dx][1],bit[dx][2],bit[dx][3]);
}
```

output

```
char = z    dec = 122    hex = 7a
            bin = 0111 1010
```

Esempio 2:

Gli operatori di shift (\ll , \gg) sono utili per descrivere l'effetto di moltiplicazioni o divisioni intere per 2 (base)

equivale a moltiplicare per 2

\ll 110 1001 1011 01010

0110 1001 1011 0101

\gg 00110 1001 1011 010

equivale a dividere per 2

```
...  
void main()  
{ short n=11;  
  printf("n >> 1 = %d\n",n>>1);  
  printf("n >> 2 = %d\n",n>>2);  
  printf("n >> 3 = %d\n",n>>3);  
  printf("n << 1 = %d\n",n<<1);  
  printf("n << 2 = %d\n",n<<2);  
  printf("n << 3 = %d\n",n<<3);  
}
```

$11 : 2 = 5$

$11 : 4 = 2$

$11 : 8 = 1$

$11 \times 2 = 22$

$11 \times 4 = 44$

$11 \times 8 = 88$

$n \gg 1 = 5$

$n \gg 2 = 2$

$n \gg 3 = 1$

$n \ll 1 = 22$

$n \ll 2 = 44$

$n \ll 3 = 88$

Esempio 3:

Scambio del contenuto di due variabili senza variabile di appoggio

```
int x,y,temp;  
x=...; y=...;  
temp=x; x=y; y=temp;
```

swap di due variabili

Dalle proprietà di XOR: se X è un qualsiasi bit (cioè $X \in \{0, 1\}$)

$$X \wedge 1 = \sim X \quad (\text{cioè } X \wedge 1 \text{ inverte } X)$$

$$X \wedge 0 = X \quad (\text{cioè } X \wedge 0 \text{ lascia } X \text{ immutato})$$

si possono scambiare i valori di due variabili senza usarne una terza di appoggio.

```
int x,y;  
x=...; y=...;  
x=x^y;  
y=x^y;  
x=x^y;
```

x	0	0	0	0	0	1	1	0
y	1	0	1	1	1	0	1	0
x	1	0	1	1	1	1	0	0
y	0	0	0	0	0	1	1	0
x	1	0	1	1	1	0	1	0

Esempio 4:

Gli operatori bitwise possono servire per azzerare i bit meno significativi di una variabile qualunque sia la sua lunghezza

per azzerare gli ultimi 6 bit di **x**

```
char X; X=125;  
X = X & 0300
```

1	1	1	1	1	1	0	1
1	1	0	0	0	0	0	0

... però la costante dipende dalla lunghezza di **x**

0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

```
short X; X=29125;  
X = X & 0177700
```

```
X = X & ~077
```

azzerà* gli ultimi 6 bit di **x**

costante otta-
le (inizia con 0)

```
X = X & ~0xff
```

azzerà* gli ultimi 8 bit di **x**

costante
esadecimale

* ora è indipendente dal tipo di **x**

Esempio 5:

Gli operatori bitwise possono servire per estrarre alcuni bit di una variabile

Proprietà: se X è un qualsiasi bit

$$X \& 1 = X$$

$$X \& 0 = 0$$

$$X | 1 = 1$$

$$X | 0 = X$$

$$X \wedge 1 = \sim X$$

$$X \wedge 0 = X$$

Gli operatori sui singoli bit sono utili quando si vogliono estrarre particolari bit (mediante l'uso di maschere) dal valore di una variabile

X	0110 1001 1011 0101
$\&$	
mask	0000 0000 1111 0000
=	0000 0000 1011 0000

240

ling. C

```
if (X & 240)...
```

$$240_{10} = 2^7 + 2^6 + 2^5 + 2^4 = f0_{16}$$

Come costruire in C una maschera per estrarre alcuni bit da una variabile?

Es. 5.1

maschera che consente (tramite un **&**) di estrarre i **5** bit meno significativi

<---short--->

0000 0000 000**1 1111**

Come costruirla in C?

1 ...tramite **potenze di 2** (richiede `pow()` da `<math.h>`)

$$11111_2 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 2^5 - 1$$

2 ...tramite **costante decimale, ottale oppure esadecimale**

31 0 **037** 0 **0x1f**

3 ...tramite **operatori bitwise**

```
mask=0;
for (b=1;b<=5;b++) mask=mask<<1|1;
```

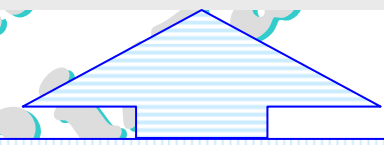
Come costruire in C una maschera per estrarre alcuni bit da una variabile?

Es. 5.2

maschera che consente (tramite un **&**) di estrarre i **5** bit più significativi

<---short--->

1111 1000 0000 0000



Come costruirla in C?

0000 0000 0001 1111

1 ... si costruisce prima **mask**

2 ... si slittano i bit verso sinistra di **11** bit (16 - 5)
mask << 11;

... in generale, per spostare dall'estrema destra all'estrema sinistra **n** bit in una variabile di tipo **type**

mask << (sizeof(type) * 8 - n);

perché ???

Laboratorio: visualizzare la rappresentazione binaria di un numero naturale mediante gli operatori bitwise.

```
#include <stdio.h>
#define n_len 16 // intero short
void bit_short(short n, unsigned char bit[n_len])
{
    short j;
    j=n_len-1;
    do {bit[j--]=n&1; n=n>>1;
        } while (n != 0 && j>=0);
    if (j >= 0)
        {do {bit[j--]=0;
            } while (j>=0);
        }
}
main()
{
    short numero; unsigned char k, bit[n_len];
    printf("numero = "); scanf("%d", &numero);
    bit_short(numero, bit);
    printf("in binario =");
    for (k=0; k<n_len; k++)
        (k%4 == 0) ? printf(" %1u", bit[k]) : printf("%1u", bit[k]);
}
```

numero = 13
in binario = 0000 0000 0000 1101

numero = 31005
in binario = 0111 1001 0001 1101

espressione condizionale in C

`(k%4 == 0) ? printf(" %1u", bit[k]) : printf("%1u", bit[k]);`

È possibile con una stessa funzione C visualizzare i bit di qualsiasi tipo?

È possibile con una stessa funzione **C** visualizzare i bit di qualsiasi tipo? **SI ! ...**

usare
union

```
union word32bit {
    long    la;
    short   sa[2];
    char    ca[4];
} w;
```

... ma bisogna stabilire **come sono allocati i byte della memoria per ciascun tipo di dato ...**

sono le stesse locazioni di memoria interpretate nei 3 tipi!

indirizzo _b +3	Ca ₃	0100 0100	+ bit -	0100 0100	+	0100 0100
indirizzo _b +2	Ca ₂	0100 0011	sa ₁	0100 0011	bit	0100 0011
indirizzo _b +1	Ca ₁	0100 0010	+ bit -	0100 0010	1a	0100 0010
indirizzo _{base}	Ca ₀	0100 0001	sa ₀	0100 0001	-	0100 0001
byte in memoria		char		short		long

```
// estrae_bit.c - operatori bit a bit
#include <stdio.h>
#include <stdlib.h> // per usare exit()
#include <math.h> // eventualmente per usare pow(x,y)
#define MAX_LEN 32 // numero bit di intero long
void bit_show(short , char [], short []);
void main()
{short menu, bit[MAX_LEN], k; unsigned char len;
```

```
union word32bit
{
    long L;
    short S[2];
    char C[4];
} word;
```

```
do
{
    puts("\n\n\nselezione");
    puts("\n[0] uscita programma");
    puts("\n[1] rappresentazione binaria di intero char");
    puts("\n[2] rappresentazione binaria di intero short");
    puts("\n[3] rappresentazione binaria di intero long");
    fflush(stdin); scanf("%hd",&menu);
```

...

```

switch( menu )
{
    case 0 : exit(0);
    case 1 :
        len=sizeof(char);
        printf("immettere intero char C ");
        fflush(stdin); scanf("%d",&(word.C[0]));
        puts("char in decimale, esadecimale e binario");
        printf("C=%+10hd, hex=%02x",word.C[0],word.C[0]);
        bit_show(sizeof(char),word.C,bit); break;
    case 2 :
        len=sizeof(short);
        printf("immettere intero short S "); // o S=pow(2,5);
        fflush(stdin); scanf("%hd",&(word.S[0]));
        puts("short in decimale, esadecimale e binario");
        printf("S=%+10hd, hex=%04hx",word.S[0],word.S[0]);
        bit_show(sizeof(short),word.C,bit); break;
    case 3 :
        len=sizeof(long);
        printf("immettere intero long L "); // o L=pow(2,5);
        fflush(stdin); scanf("%ld",&(word.L));
        puts("long in decimale, esadecimale e binario");
        printf("L=%+10d, hex=%08lx",word.L,word.L);
        bit_show(sizeof(long),word.C,bit); break;
    default : exit(1);
}

```

...

```
// visualizza i bit
for (k=8*len-1; k>=0; k--)
    (k%4 == 0) ? printf("%1d ",bit[k]) : printf("%1d",bit[k]);
} while (menu !=0);
}
```

```
//-----
void bit_show(short len, char ch[], short bit[MAX_LEN])
{
    short j,jc; char c;
    for (j=0; j<MAX_LEN; j++) bit[j]=0;
    // estrae i bit
    for (jc=0; jc<len; jc++)
        { c=ch[jc];
          for (j=0; j<8; j++)
              { bit[j+8*jc]=c&1;
                c=c>>1;
              }
        }
}
```

selezione

[0] uscita programma

[1] rappresentazione binaria di intero char

[2] rappresentazione binaria di intero short

[3] rappresentazione binaria di intero long

1

immettere intero char C 13

char in decimale, esadecimale e binario

C= +13, hex=0d 0000 1101

Esercizi

1

Dopo aver estratto i bit da una variabile X intera (tipo `char`, `short` o `long`) calcolare il relativo valore intero dalla formula:

$$\text{Val}_X = b_{n-1}2^{n-1} + \dots + b_22^2 + b_12^1 + b_02^0$$

dove b è l'array dei bit di X . Confrontare il risultato con il valore della variabile X dichiarata una volta *signed* ed un'altra *unsigned*.

2

Scrivere una function C per estrarre dalla variabile intera A i k bit più significativi o meno significativi, dove A , k sono parametri di input usando: (1) una maschera; (2) l'operatore di shift (\gg o \ll) (3) prodotto o divisione per potenza di 2.