

Modulo: Approfondimenti sui Sistemi Aritmetici di un computer: tipo intero [P2_02]

Unità didattica: Rappresentazioni per complemento a 2 e per eccesso B [4-AT]

Titolo: Interi rappresentati per complemento a 2, per eccesso B

Argomenti trattati:

- ✓ Rappresentazione per complemento a 2 su n bit
- ✓ Rappresentazione per complemento alla base: cifre complementari
- ✓ Rappresentazione per eccesso B (B-biased) su n bit
- ✓ Tipi di dati interi in C (char, short int, long int) signed e unsigned
- ✓ Overflow d'intero

Prerequisiti richiesti: Sistema Aritmetico Intero

La **rappresentazione per complemento a 2** (r_{c2}) su **n bit** di un **intero** (rappresentabile) $k \in \mathbb{Z}$ si esprime con la formula

$$r_{c2}(k) = [2^n + k]_{\text{mod. } 2^n}$$

Per un **intero** $+k \geq 0$ ($k=0, 1, \dots, 2^{n-1}$) coincide col numero stesso:

$$r_{c2}(+k) = [2^n + (+k)]_{\text{mod. } 2^n} \equiv k$$

Invece per un **intero** $-k < 0$ ($-k=-(2^{n-1}-1), \dots, -1$) è data da

$$r_{c2}(-k) = [2^n + (-k)]_{\text{mod. } 2^n} \equiv 2^n - k$$

La **rappresentazione per complemento a 2** su **n bit** di un numero negativo si può calcolare anche tramite gli **operatori bitwise**:

➤ si complementa il numero positivo bit a bit ($0 \longleftrightarrow 1$)

➤ si addiziona 1 **modulo** 2^n .

Esempio

$$n=4, \quad k=5, \quad r_{c2}(-5)=?$$

$$r_{c2}(-k) = [2^n + (-k)]_{\text{mod. } 2^n} \equiv 2^n - k =$$

$$= [16 - 5] = 11_{10} = 1011_2$$

$$n=4, \quad -k = -5,$$

$$+5_{10} = 0101_2 \longleftarrow +k$$

$$\text{bitNOT}(+5_{10}) = 1010_2 +$$

$$\underline{1} =$$

$$1011 \longleftarrow r_{c2}(-5)$$

Dall'essere

$$r_{C2}(+k) + r_{C2}(-k) = 2^n \equiv 0 \pmod{2^n}$$

risulta che $r_{C2}(-k)$ si comporta come $-k$ in aritmetica **mod** 2^n ,
e pertanto entrambi gli algoritmi per $+$ e $-$ si riconducono ad
un'addizione aritmetica.

Esempio: sottrazione

$$h - k = [r_{C2}(h) + r_{C2}(-k)]_{\text{mod } 2^n}$$

$$n=4, \quad h-k = 7-5,$$

$$r_{C2}(+7) = 0111 +$$

$$r_{C2}(-5) = \underline{1011} =$$

$$\cancel{1}0010 = r_{C2}(+2)$$

Rappresentazione per complemento alla base: cifre complementari

Base 2	
0	1

Base 10	
0	9
1	8
2	7
3	6
4	5

Base 16	
0	F
1	E
2	D
3	C
4	B
5	A
6	9
7	8

h e k cifre complementari
 se, in base β , risulta
 $h + k = \beta - 1$

Rappresentazione per complemento alla base

$n=4$

$$\text{range } [i_{\min}, i_{\max}] = [-2^{n-1}, 2^{n-1} - 1]$$
$$= [-8, 7]$$

$\beta=2$	
0	1

$n=4$

$r_{C_2}(-7)$

$+7_{10}$		0	1	1	1	
compl.		1	0	0	0	
$+1$		0	0	0	1	
-7_{10}		1	0	0	1	+
$+7_{10}$		0	1	1	1	=
0	1	0	0	0	0	

Rappresentazione per complemento alla base

n=4

Base 10	
0	9
1	8
2	7
3	6
4	5

range $[i_{\min}, i_{\max}] = [-10^{n-1}, 10^{n-1} - 1]$
 $= [-1000, 999]$

$r_{C_{10}}(-37)$

n=4					
+37₁₀		0	0	3	7
Compl.		9	9	6	2
+1		0	0	0	1
-37₁₀		9	9	6	3
+37₁₀		0	0	3	7
0	1	0	0	0	0

tipo intero

Rappresentazione per complemento alla base

n=4

Base 16	
0	F
1	E
2	D
3	C
4	B
5	A
6	9
7	8

range $[i_{\min}, i_{\max}] = [-16^{n-1}, 16^{n-1} - 1]$
 $= [-4096, 4095]$

$r_{C_{16}}(-3898)$

n=4

+3898 ₁₀		0	f	3	a
Compl.		f	0	c	5
+1		0	0	0	1
-3898 ₁₀		f	0	c	6
+3898 ₁₀		0	f	3	a
0	1	0	0	0	0

La **rappresentazione biased con bias B** di un intero (rappresentabile) **k** su **n bit**, **r_B(k)**, può esprimersi con la formula

$$r_B(k) = k + \boxed{B} = k + \boxed{2^{n-1} - 1} \equiv 0 \pmod{2^n}$$

Risulta $r_B(h) + r_B(\pm k) = h \pm k + 2B = h \pm k + 2^{n-1} - 2 \equiv \boxed{h \pm k - 2}$ serve correzione

invece di $r_B(h \pm k) = h \pm k + B = \boxed{h \pm k + 2^{n-1} - 1}$ correzione

Quindi $r_B(h \pm k) = r_B(h) + r_B(\pm k) + \text{correzione} \iff \text{correzione} = 2^{n-1} + 1$

Esempio 1

n=4, B=7,

$$r_B(+6) = +6 + 7 = 1101_2$$

$$r_B(-5) = -5 + 7 = 0010_2$$

Esempio 2: sottrazione

n=4, h - k = 7 - 5,

$$r_B(+7) = 1110 +$$

$$r_B(-5) = \underline{0010} =$$

$$\begin{array}{r} \times 0000 \quad (\text{mod } 2^n) \\ + 1001 = \text{correzione} \\ \hline 1001 = r_B(+2) \end{array}$$

Rappresentazione biased

n=4



range $[i_{\min}, i_{\max}] = [-(2^{n-1} - 1), 2^{n-1}]$
 $= [-7, 8]$

Bias $= 2^{n-1} - 1 = 7$

n=4

prova

+5₁₀				
+Bias				
= 12	1	1	0	0

-5₁₀				
+Bias				
= 2	0	0	1	0

$r_B(+5_{10}) = 12$	1	1	0	0
$r_B(-5_{10}) = 2$	0	0	1	0
$r_B(+5_{10}) + r_B(+5_{10})$	1	1	1	0
correzione	1	0	0	1
1	0	1	1	1
$r_B(0) \Leftrightarrow B =$	0	1	1	1

❑ La **rappresentazione per complemento a 2** è usata per memorizzare i **numeri interi** con segno.

❑ La **rappresentazione biased** è usata per il **campo esponente** di un numero reale floating-point.

❑ La **rappresentazione per segno e modulo** è usata per il **campo mantissa** di un numero reale floating-point.

Il **Sistema Aritmetico degli Interi (I)** di un computer usa la *rappresentazione per complemento a 2* dei numeri interi.

Interi nel linguaggio C:

il tipo **char**

rappresenta un intero su **n=8** bit (**1** byte)

$$\text{range} = [-2^7, 2^7 - 1] = [-128, +127]$$

il tipo **short int**

rappresenta un intero su **n=16** bit (**2** byte)

$$\text{range} = [-2^{15}, 2^{15} - 1] = [-32768, +32767]$$

il tipo **int**

rappresenta un intero su **n=32** bit (**4** byte)

$$\text{range} = [-2^{31}, 2^{31} - 1] = [-2147483648, +2147483647]$$

il tipo **long int**

rappresenta un intero su **n=64** bit (**8** byte)

indirizzi
a 64 bit

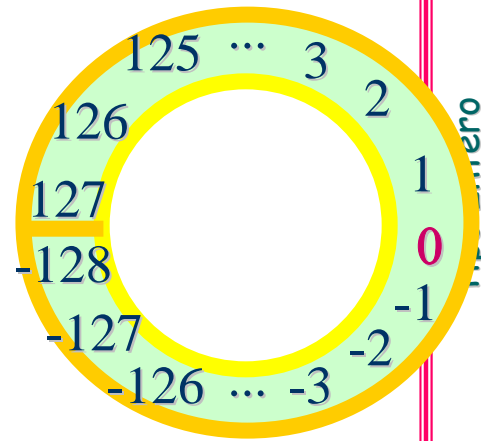
$$\text{range} = [-2^{63}, 2^{63} - 1] = [-9223372036854775808, \dots]$$

Quando si tenta di rappresentare un intero che non appartiene al range si dice che si è verificato un **OVERFLOW** d'intero (non è segnalato).

Esempio: sapendo che il tipo **char** rappresenta un intero su **n=8** bit

$$\text{range} = [-2^7, 2^7 - 1] = [-128, +127]$$

```
void main()
{ char c;
  c= 127; printf("+127 = %+6d,\thex = %04x\n",c,c);
  c= 128; printf("+128 = %+6d,\thex = %04x\n",c,c);
  c= 130; printf("+130 = %+6d,\thex = %04x\n",c,c);
  c=-128; printf("-128 = %+6d,\thex = %04x\n",c,c);
  c=-129; printf("-129 = %+6d,\thex = %04x\n",c,c);
  c=-130; printf("-130 = %+6d,\thex = %04x\n",c,c);}
```



+127	=	+127	hex = 007f
+128	=	-128	hex = ff80
+130	=	-126	hex = ff82
-128	=	-128	hex = ff80
-129	=	+127	hex = 007f
-130	=	+126	hex = 007e

$\left. \begin{matrix} = i_{max} \\ > i_{max} \end{matrix} \right\}$
 $\left. \begin{matrix} = i_{min} \\ < i_{min} \end{matrix} \right\}$

L'overflow non è segnalato!

Il risultato (di tipo char) è ottenuto in **aritmetica modulo 2^8**

con gcc vers. 4.6.1

```
1  #include <stdio.h>
2  int main()
3  { char c;
4      c= 127; printf("+127  =%+6d, \thex = %02hhx\n", c, c);
5      c= 128; printf("+128  =%+6d, \thex = %02hhx\n", c, c);
6      c= 130; printf("+130  =%+6d, \thex = %02hhx\n", c, c);
7      c=-128; printf("-128  =%+6d, \thex = %02hhx\n", c, c);
8      c=-129; printf("-129  =%+6d, \thex = %02hhx\n", c, c);
9      c=-130; printf("-130  =%+6d, \thex = %02hhx\n", c, c);
10
11  return 0;
12  }
13
```

```
/home/rizzardi/Scrivania/ProgLab
+127 = +127,    hex = 7f
+128 = -128,    hex = 80
+130 = -126,    hex = 82
-128 = -128,    hex = 80
-129 = +127,    hex = 7f
-130 = +126,    hex = 7e

Process returned 0 (0x0)    execution time : 0.003 s
Press ENTER to continue.
```

Esempio

```
#include <stdio.h>
#include <stdlib.h>
void main()
{short s; unsigned short u;
 printf("immetti signed short..");
 scanf("%hd",&s); u=s;
 printf("signed = %d\nunsigned = %d\n",s,u);
}
```



immetti signed short..-13
signed = -13
unsigned = 65523

Come spiegare i risultati?

$$\begin{array}{r} 65523+ \\ 13= \\ \hline 65536 \end{array}$$

È la rappresentazione per complemento a 2 di -13

È lo zero dell'aritmetica mod. 2^{16}

2^{16}

Il seguente programma [con C a 32 bit*] ...

04.15

```
#include <stdio.h>
#include <math.h> /* eventualmente per usare pow(x,y) */
void main()
{
    int inp, inm; short snp, snm; long lnp, lnm;
    /* assegna un valore alle variabili intere */
    scanf("%d",&lnp); /* oppure ... lnp=pow(2,...)-...; */
    lnm=-lnp; snp=lnp; snm=-snp; inp=lnp; inm=-inp;
    printf("(+)long   int = %+11d,\thex = %08x\n",lnp,lnp);
    printf("(-)long   int = %11d,\thex = %08x\n\n",lnm,lnm);
    printf("(+)short int = %+11d,\thex =          %04hx\n",
                                                    snp,snp);
    printf("(-)short int = %11d,\thex =          %04hx\n\n",
                                                    snm,snm);
    printf("(+)      int = %+11d,\thex = %08x\n",inp,inp);
    printf("(-)      int = %11d,\thex = %08x\n\n",inm,inm);
}
```

... produce

* C a 32 bit: long int coincide con int

(prof. M. F.)

Esempio 1: $\text{pow}(2, 0) = 2^0 = 1$

(+)long	int	=	+1,	hex	=	00000001
(-)long	int	=	-1,	hex	=	ffffffff
(+)short	int	=	+1,	hex	=	0001
(-)short	int	=	-1,	hex	=	ffff
(+)	int	=	+1,	hex	=	00000001
(-)	int	=	-1,	hex	=	ffffffff

Esempio 2: $\text{pow}(2, 5) = 2^5 = 32$

(+)long	int	=	+32,	hex	=	00000020
(-)long	int	=	-32,	hex	=	ffffffe0
(+)short	int	=	+32,	hex	=	0020
(-)short	int	=	-32,	hex	=	ffe0
(+)	int	=	+32,	hex	=	00000020
(-)	int	=	-32,	hex	=	ffffffe0

Esempio 3: $\text{pow}(2, 15) - 1$

(+)long	int	=	+32767,	hex	=	00007fff
(-)long	int	=	-32767,	hex	=	ffff8001
(+)short	int	=	+32767,	hex	=	7fff
(-)short	int	=	-32767,	hex	=	8001
(+)	int	=	+32767,	hex	=	00007fff
(-)	int	=	-32767,	hex	=	ffff8001

Esempio 4: $\text{pow}(2, 15)$

(+)long	int	=	+32768,	hex	=	00008000
(-)long	int	=	-32768,	hex	=	ffff8000
(+)short	int	=	-32768,	hex	=	8000
(-)short	int	=	-32768,	hex	=	8000
(+)	int	=	+32768,	hex	=	00008000
(-)	int	=	-32768,	hex	=	ffff8000

overflow d'intero del tipo short

Esempio 5: $\text{pow}(2, 31) - 1$

(+)long	int	=+2147483647,	hex	=	7fffffff	
(-)long	int	=-2147483647,	hex	=	80000001	
(+)short	int	=	-1,	hex	=	ffff
(-)short	int	=	1,	hex	=	0001
(+)	int	=+2147483647,	hex	=	7fffffff	
(-)	int	=-2147483647,	hex	=	80000001	

Esempio 6: $\text{pow}(2, 31)$

(+)long	int	=-2147483648,	hex	=	80000000	
(-)long	int	=-2147483648,	hex	=	80000000	
(+)short	int	=	+0,	hex	=	0000
(-)short	int	=	0,	hex	=	0000
(+)	int	=-2147483648,	hex	=	80000000	
(-)	int	=-2147483648,	hex	=	80000000	

overflow d'intero del tipo int

Esempio 7: `pow(2, 32)`

(+)long	int	= -2147483648	hex	= 80000000
(-)long	int	= -2147483648	hex	= 80000000
(+)short	int	= +0	hex	= 0000
(-)short	int	= 0	hex	= 0000
(+)	int	= -2147483648	hex	= 80000000
(-)	int	= -2147483648	hex	= 80000000

Esempio 8: `pow(2, 399)`

(+)long	int	= +2147483647	hex	= 7fffffff
(-)long	int	= -2147483647	hex	= 80000001
(+)short	int	= -1	hex	= ffff
(-)short	int	= 1	hex	= 0001
(+)	int	= +2147483647	hex	= 7fffffff
(-)	int	= -2147483647	hex	= 80000001

overflow d'intero del tipo int

Esercizi

1

Scrivere una *function* C che, fissato il numero n di bit, calcoli la rappresentazione di un intero:

- per complemento a due (C2);
- biased.

2

Conoscendo la rappresentazione degli interi in C, **riscrivere** la *function C* (vers. 2) per l'addizione binaria di due interi mediante gli operatori bitwise a partire dall'algoritmo seguente:

Algoritmo “addizione binaria”

```
rip:=1;  
while rip>0  
    sum:=bitXOR(op1,op2);  
    rip:=bitAND(op1,op2);  
    rip:=leftSHIFT(rip,1);  
    op1:=sum; op2:=rip;  
endwhile
```

Se l'addizione binaria deve valere per tutti gli interi (**interi positivi e negativi**) rappresentati per complemento a 2, quale accorgimento va usato nella traduzione in C dell'algoritmo ... e perché.

```
/home/rizzardi/Scrivania/ProgLab2/

add1 = -7
add2 = 2
bitwise su interi char

addizione: -7 + 2

risultato = (+) -5      (bitwise) -5

Process returned 0 (0x0)   execution time : 76.468 s
Press ENTER to continue.
```