Esercitazioni su rappresentazione dei numeri e aritmetica

Salvatore Orlando
&
Marta Simeoni

Interi *unsigned* in base 2

- Si utilizza un alfabeto binario A = {0,1}, dove 0 corrisponde al *numero zero*, e 1 corrisponde al *numero uno*

  $d_{n-1}...d_1d_0$ con di $d_i \in \{0,1\}$

- Qual è il numero rappresentato ? $N = d_{n-1} \cdot 2^{n-1} + ... + d_1 \cdot 2^1 + d_0 \cdot 2^0$

- Quanti numeri sono rappresentabili su *n* bit?

00….00 = 0
00….01 = 1
00….10 = 2
…
01….11 = $2^n - 3$
10….00 = $2^n - 2$
…
11….11 = $2^n - 1$
100….00 = 2^n

Con sequenze di *n* bit sono rappresentabili 2^n numeri naturali (da 0 a 2^{n-1})

Interi *unsigned* in base 2

- I seguenti numeri naturali sono rappresentabili usando il numero di bit specificato ?

- 20_{10} su 5 bit ? SI
- 64_{10} su 6 bit ? NO
- 500_{10} su 9 bit ? SI
- 1025_{10} su 10 bit ? NO

Ricorda che:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
....

Conversione binario-decimale

- Esercizio: $1110101_2 = ???_{10}$

$$1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$

$$64 + 32 + 16 + 0 + 4 + 0 + 1$$

Soluzione: $1110101_2 = 117_{10}$

Conversione decimale-binario

- **Esercizio**: $100_{10} = ???_2$

100 : 2 = 50	resto 0
50 : 2 = 25	resto 0
25 : 2 = 12	resto 1
12 : 2 = 6	resto 0
6 : 2 = 3	resto 0
3 : 2 = 1	resto 1
1 : 2 = 0	resto 1

Soluzione: $100_{10} = 1100100_2$

Conversione dec-bin: metodo più pratico

- Scriviamo direttamente il numero decimale come somma di potenze di 2
- Per far questo, sottraiamo via via le potenze di 2, a partire dalle più significative

- **Esercizio**: $103_{10} = ???_2$

103 - 64 = 39	==>	2^6
39 - 32 = 7	==>	2^5
7 - 4 = 3	==>	2^2
3 - 2 = 1	==>	2^1
1 - 1 = 0	==>	2^0

Allora $103_{10} = 2^6 + 2^5 + 2^2 + 2^1 + 2^0$

Soluzione: $103_{10} = 1100111_2$

Ricorda che:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
....

Conversione binario-ottale e viceversa

- **Esercizio**: $10101111_2 = ???_8$

10	101	111
2	5	7

Soluzione: $10101111_2 = 257_8$

- **Esercizio**: $635_8 = ???_2$

6	3	5
110	011	101

Soluzione: $635_8 = 110011101_8$

Base 16

- Quali dei seguenti numeri esadecimali sono numeri sono corretti?

BED
~~CAR~~
938
DEAD
BEBE
A129
~~ACI~~
~~DECADE~~
~~BAG~~
~~DAD~~
~~4H3~~

Conversione binario-esadecimale e viceversa

- **Esercizio**: $101111101101_2 = ???_{16}$

$$\underline{1011}\quad \underline{1110}\quad \underline{1101}$$

$$B \qquad E \qquad D$$

Soluzione: $101111101101_2 = BED_{16} = 3053_{10}$

- **Esercizio**: $A3C9_{16} = ???_2$

$$A \qquad 3 \qquad C \qquad 9$$

$$\underline{1010}\quad \underline{0011}\quad \underline{1100}\quad \underline{1001}$$

Soluzione: $A3C9_{16} = 1010001111001001_2 = 41929_{10}$

Ricorda che:

$1_{10} = 1_{16} = 0001_2$
$2_{10} = 2_{16} = 0010_2$
.....
$9_{10} = 9_{16} = 1001_2$
$10_{10} = A_{16} = 1010_2$
$11_{10} = B_{16} = 1011_2$
$12_{10} = C_{16} = 1100_2$
$13_{10} = D_{16} = 1101_2$
$14_{10} = E_{16} = 1110_2$
$15_{10} = F_{16} = 1111_2$

Interi *signed* in complemento a 2

- Come si riconosce un numero positivo da uno negativo?
  - Positivo $\Rightarrow$ bit più significativo 0
  - Negativo $\Rightarrow$ bit più significativo 1

- Su n bit sono rappresentabili 2^n *interi unsigned* (da 0 a 2^n-1)
- Sempre su n bit, quanti *interi signed* in complemento a 2 ?

```
0.......00  = 0
0.......01  = 1
      ...
01......11  = 2^(n-1)-1  (massimo)
10......00  = 2^(n-1)    (minimo)        - 2^(n-1)  = 2^(n-1) - 2^n
      ...
11......11  = 2^n - 1                    -1  = 2^n - 1 - 2^n
```

Dato $N>0$, il numero $-N$ si rappresenta su n bit con il numero $2^n - N$

$-1 \Rightarrow 2^n - 1$ (1.....1)

$-2^{n-1} \Rightarrow 2^n - 2^{n-1} = 2^{n-1}$ (10.....0)

Complemento a 2

- **Esercizio**: Rappresentare -35_{10} in complemento a 2

$00100011_2 = +35_{10}$

↓ Complemento a uno

```
11011100  +
       1  =
------------
11011101
```

Soluzione: $-35_{10} = 11011101_2$

Complemento a 2

- **Esercizio**: Rappresentare -35 in complemento a 2

$00100011_2 = +35_{10}$

↓

11011101_2

Inverti (complementa a 1) tutti i bit a sinistra del bit "1" meno significativo

Complemento a 2

- **Esercizio**: Quale numero decimale rappresenta il seguente numero binario in complemento a due?

$1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0000\ 1100_2$

↓

$0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 1111\ 0100_2 =$

$2^2 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8 = 500_{10}$

Soluzione: il numero è -500_{10}

Complemento a 2: somma e sottrazione

- **Esercizio**: eseguire $53_{10} - 35_{10}$ in complemento a due su 8 bit

$35_{10} = 00100011_2$ *complementando*: $-35_{10} = 11011101_2$

				11111101
53_{10} -		53_{10} +		00110101_2 +
35_{10} =		$(-35)_{10}$ =		11011101_2 =
———	⇒	———	⇒	———
18_{10}		18_{10}		$(100010010_2) \bmod 2^8$

$00010010_2 = 18_{10}$

Complemento a 2: somma e sottrazione

- **Esercizio**: eseguire $15_{10} - 38_{10}$ in complemento a due su 8 bit

$38_{10} = 00100110_2$ *complementando*: $-38_{10} = 11011010_2$

				00011110
15_{10} -		15_{10} +		00001111_2 +
38_{10} =		$(-38)_{10}$ =		11011010_2 =
———	⇒	———	⇒	———
-23_{10}		-23_{10}		$(011101001_2) \bmod 2^8$

↓

$00010111_2 = 23_{10}$

Overflow

- In quali dei seguenti casi si può ottenere overflow?
  - somma di due numeri con segno concorde? SI
  - somma di due numeri con segno discorde? NO
  - sottrazione di due numeri con segno concorde? NO
  - sottrazione di due numeri con segno discorde? SI

Esercizio

- Considerate i numeri esadecimali $x_1 = 7A$ $x_2 = 13$ $x_3 = FF$ $x_4 = C1$ $x_5 = 84$
- Scrivere i quattro numeri in codice binario a 8 bit

$x_1 = 0111\ 1010$ $x_2 = 0001\ 0011$ $x_3 = 1111\ 1111$

$x_4 = 1100\ 0001$ $x_5 = 1000\ 0100$

- Interpretare il codice binario in complemento a due ed eseguire le operazioni

$x_1 - x_2$; $x_3 + x_4$; $x_4 + x_5$; $x_4 - x_1$

```
  11111000
x1 01111010 +
-x2 11101101 =
_____________
(101100111) mod 2^8 = 01100111   overflow?
```

Esercizio (continua)

```
  11111111
x3 11111111 +
x4 11000001 =
_____________
(111000000) mod 2^8 = 11000000   overflow?
```

```
  10000000
x4 11000001 +
x5 10000100 =
_____________
(101000101) mod 2^8 = 01000101   overflow?
```

```
  10000000
x4 11000001 +
-x1 10000110 =
_____________
(101000111) mod 2^8 = 01000111   overflow?
```

Esercizio - caso particolare

- Si ricorda che l'opposto del numero negativo più piccolo su *n bit* non può essere rappresentato in complemento a due
  - codifica non simmetrica
- Supponiamo quindi
  - di lavorare con rappresentazioni in complemento a due su 3 bit
  - di dover effettuare la sottrazione x-y

dove $y=100_2$ è il minimo numero rappresentabile ($y=-4_{10}$)

Esercizio: calcolare x-y usando il solito algoritmo, dove $x=001_2$

```
  000
X 001 +
-Y 100
_____
(0101) mod 2^3 = 101   OVERFLOW, anche se
                       riporti concordi !!
```

Il complemento a due non ha effetto

Abbiamo infatti sottratto un numero negativo da un numero >= 0 ⇒ il segno atteso è positivo

Esercizio - caso particolare (continua)

- **Esercizio**: calcolare x-y, dove $x=111_2$
  - poiché $x=111_2$ allora $x = -1_{10}$
  - non dovremmo avere overflow, poiché vogliamo effettuare la somma algebrica di due numeri con segno discorde
  - il risultato da ottenere è $x-y=-1_{10} - (-4_{10}) = 3_{10}$

```
  100
X 111 +
-Y 100
_____
(0011) mod 2^3 = 011 = 3_10
```

Il complemento a due non ha effetto

corretto

NO OVERFLOW , anche se riporti discordi !!

Abbiamo infatti sottratto un numero negativo da un numero negativo => in questo caso non si può verificare overflow

Procedura generale per determinare l'OVERFLOW

Alla luce dell'esempio precedente, guardare solo ai due ultimi riporti per controllare l'OVERFLOW potrebbe quindi portare a risultati erronei

Operazione	Segno 1º operando	Segno 2º operando	Segno atteso
Somma	+	+	+
Somma	+	-	qualsiasi
Somma	-	+	qualsiasi
Somma	-	-	-
Sottrazione	+	+	qualsiasi
Sottrazione	+	-	+
Sottrazione	-	+	-
Sottrazione	-	-	qualsiasi

Solo in questi casi si può verificare un OVERFLOW.

Possiamo controllarlo confrontando il bit di segno del risultato con il segno atteso.

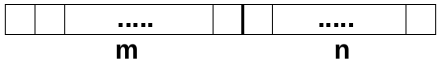
Numeri con la virgola (virgola fissa)

Data una base B, si assegnano:

n *cifre* per rappresentare la parte intera

m *cifre* per rappresentare la parte frazionaria

In base B=2, abbiamo quindi m+n *bit* per parte intera e frazionazia



Esempio:

$d_{n-1}...d_1d_0 \cdot d_{-1}...d_{-m}$

Qual è il numero rappresentato in base B?

$$N = d_{n-1} \bullet B^{n-1} + ... + d_1 \bullet B^1 + d_0 \bullet B^0 + d_{-1} \bullet B^{-1} + ... + d_{-m} \bullet B^{-m}$$

Virgola fissa

Esercizio: $23.625_{10} = ???_2$

(usare la rappresentazione in virgola fissa con n=8, m=8)

Conversione parte intera:

23 : 2 = 11	resto 1
11 : 2 = 5	resto 1
5 : 2 = 2	resto 1
2 : 2 = 1	resto 0
1 : 2 = 0	resto 1

Conversione parte frazionaria:

0.625 x 2 = 1.25	parte intera 1
0.25 x 2 = 0.50	parte intera 0
0.50 x 2 = 1	parte intera 1

Soluzione: $23.625_{10} = 00010111.10100000_2$

Numeri con virgola mobile

- Un numero reale R può essere scritto in base B come $R = \pm m \bullet B^e$
  - m = mantissa
  - e = esponente
  - B = base
- Esempi con B = 10
  - $R1 = 3.1569 \times 10^3$
  - $R2 = 2054.00035 \times 10^{-6}$
  - $R3 = 0.1635 \times 10^2$
  - $R4 = 0.0091 \times 10^{-12}$
- **Notazione scientifica**: $m = 0 . d_{-1}...d_{-k}$
- **Notazione scientifica normalizzata**: $m = d_0 . d_{-1}...d_{-k}$ con $d_0 \neq 0$

Numeri binari in virgola mobile

Rappresentando mantissa ed esponente in binario in notazione scientifica normalizzata si ottengono numeri del tipo:

$\pm 1.\ ss....s \bullet 2^{yy...y}$

Si osservi che:

Spostare la virgola (punto) a destra di *n* bit significa decrementare di *n* l'esponente

es: $0.01 \bullet 2^3 = 1.0 \bullet 2^1$
Infatti $1 \bullet 2^{-2} \bullet 2^3 = 1 \bullet 2^1$

Spostare la virgola (punto) a sinistra di *n* bit significa incrementare di *n* l'esponente

es: $100.011 \bullet 2^3 = 1.00011 \bullet 2^5$
Infatti $(1 \bullet 2^2 + 1 \bullet 2^{-2} + 1 \bullet 2^{-3}) \bullet 2^3 = (1 \bullet 2^0 + 1 \bullet 2^{-4} + 1 \bullet 2^{-5}) \bullet 2^5$
$2^5 + 2^1 + 2^0 = 2^5 + 2^1 + 2^0$

Numeri FP

- Esercizio: $10_{10} = ???_2$ FP

$10_{10} = 1010_2 = 1010.0_2 \bullet 2^0 = 1.01 \bullet 2^3$

- Esercizio: $151.25_{10} = ???_2$ FP

$151_{10} = 128 + 16 + 4 + 2 + 1 = 10010111_2$

$0.25_{10} \times 2 = 0.50_{10}$ parte intera 0
$0.50_{10} \times 2 = 1_{10}$ parte intera 1
$0.25_{10} = 0.01_2$

Quindi $= 151.25_{10} = 10010111.01_2 = 1.001011101_2 \bullet 2^7$

Numeri FP

Una volta fissato il numero di bit totali per la rappresentazione dei numeri razionali rimane da decidere

Quanti bit assegnare per la mantissa ?
(maggiore è il numero di bit e maggiore è l'accuratezza con cui si riescono a rappresentare i numeri)

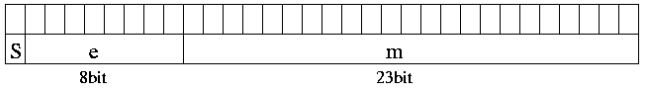
Quanti bit assegnare per l'esponente ?
(aumentando i bit si aumenta l'intervallo dei numeri rappresentabili)

OVERFLOW: si ha quando l'esponente positivo è troppo grande per poter essere rappresentato con il numero di bit assegnato all'esponente

UNDERFLOW: si ha quando l'esponente negativo è troppo grande (in valore assoluto) per poter essere rappresentato con il numero di bit assegnato all'esponente

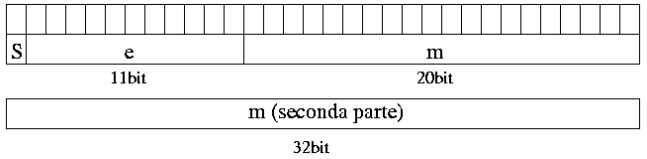
Numeri FP in standard IEEE 724

Standard IEEE754: Singola precisione (32 bit)



si riescono a rappresentare numeri $2.0_{10} \bullet 2^{-38} \div 2.0_{10} \bullet 2^{38}$

Standard IEEE754: Doppia precisione (64 bit)



si riescono a rappresentare numeri $2.0_{10} \bullet 2^{-308} \div 2.0_{10} \bullet 2^{308}$