Codifica binaria

September 15, 2018

Outline

Rappresentazione dei numeri nel calcolatore

Introduzione

- La rappresentazione numerica consiste nell'utilizzare un insieme di simboli b_i (chiamati cifre o caratteri) per specificare un numero:

$$b_n \dots b_0$$

- Useremo la notazione

$$[\![b_n \dots b_0]\!]_B$$

per indicare il numero corrispondente se interpretiamo la serie di cifre in base B.

Introduzione

- Un sistema di rappresentazione numerica può essere *posizionale* o *non posizionale*.
- Nel primo caso (tra cui troviamo il sistema decimale), ogni cifra assume un peso diverso a seconda della posizione in cui è scritta. Nel secondo no.
- La rappresentazione unaria è un esempio di rappresentazione non posizionale:

Cifra unaria	Valore decimale
I	1
II	2
III	3
IIII	4
~~IIII~~	5
~~IIII~~ I	6

Rappresentazione decimale

- Il sistema decimale è posizionale; ad esempio questi due rappresentazioni sono differenti:

$$[\![123]\!]_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

$$[\![321]\!]_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0$$

Rappresentazione decimale

- Nelle notazioni posizionali, distinguiamo la base e le cifre.
- Nella rappresentazione decimale, la base è il numero 10, e le cifre possono essere da 0 a 9:

$$[\![\mathbf{b}_n \ \ldots \mathbf{b}_0]\!]_{10} = b_{N-1}10^{N-1} + \cdots + b_0 10^0 = \sum_{i=0}^{N-1} b_i \cdot 10^i.$$

dove i pesi sono calcolati come potenze della base, in questo caso dieci.

Rappresentazione in base arbitraria

E' possibile usare una qualsiasi base K (e.g., K=8 oppure K=16)

$$[\![\mathtt{b}_n \ \ldots \mathtt{b}_0]\!]_K = b_{N-1}K^{N-1} + \cdots + b_0K^0 = \sum_{i=0}^{N-1} b_i \cdot K^i.$$

esempio

$$[\![\mathtt{10}]\!]_{16} = 1 * 16 + 0 * 1 = 16$$

$$[\![\mathtt{24}]\!]_{8} = 2 * 8 + 4 = 20$$

Rappresentazione esadecimale (K=16)

Codifica molto frequente nelle applicazioni informatiche; le cifre di base sono 16:

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

quindi possiamo avere codifiche di questo tipo

$$[\![\texttt{1A}]\!]_{16} = 1 * 16 + 10 * 1 = 26$$

Codifica binaria naturale

Rappresentazione binaria

- Nella rappresentazione binaria naturale, la base è il numero 2 e le cifre (bit) possono essere 0 oppure 1:

$$[\![\mathtt{b}_n \ \ldots \mathtt{b}_0]\!]_2 = b_{N-1}2^{N-1} + \cdots + b_0 2^0 = \sum_{i=0}^{N-1} b_i \cdot 2^i.$$

  dove N è il numero di bit della cifra binaria.

- Ad esempio, avremmo

$$[\![\mathtt{101101}]\!]_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [\![\mathtt{45}]\!]_{10}$$

Conversione in codifica binaria - Metodo dei resti

Sia $m = 44$ il valore che vogliamo convertire in binario. Il metodo dei resti permette di calcolare la codifica binaria dividendo iterativamente per due:

Valore	Divisone per due	Risultato	Resto	nota
44	44/2	22	0	cifra più a destra
22	22/2	11	0	
11	11/2	5	1	
5	5/2	2	1	
2	2/2	1	0	
1	1/2	0	1	cifra più a sinistra

Codifica di 44: 101100

Notazione abbreviata

$2^{24} = 2^{4+20} = 16$ Mega (leggi “16 milioni ”)
$2^{35} = 2^{5+30} = 32$ Giga (leggi “32 miliardi”)
$2^{48} = 2^{8+40} = 256$ Tera (leggi “256 bilioni”)

Cambio del numero dei bit

- premettendo in modo progressivo un bit 0 a sinistra, il valore del numero non muta

$$[\![\texttt{100}]\!]_2 = [\![\texttt{0100}]\!]_2 = [\![\texttt{00100}]\!]_2 = ...[\![\texttt{000000000100}]\!]_2 = [\![4]\!]_{10}$$

- cancellando in modo progressivo un bit 0 a sinistra, il valore del numero non muta, ma bisogna arrestarsi quando si trova un bit 1!

$$7 = [\![\texttt{00111}]\!]_2 = [\![\texttt{0111}]\!]_2 = [\![\texttt{111}]\!]_2 - - - \mathit{STOP}!$$

Somma di numeri binari

La somma si esegue colonna per colonna come per le somme decimali. Attenzione che si ha un riporto già per `1 + 1 = 10`!

```
     riporto      1   -  -  -
4 (decimale)     (0)  1  0  0
7 (decimale)     (0)  1  1  1
                 ------------
11 (decimale)     1   0  1  1
```

Se avessi avuto a disposizione solo 3 bit per codificare il numero, questo sarebbe stato un errore di overflow.

Codifica binaria dei numeri con segno

Codifica modulo e segno

- il primo bit a sinistra rappresenta il segno del numero (bit di segno),
- i bit rimanenti rappresentano il valore:
  - 0 per il segno positivo
  - 1 per il segno negativo
- esempi con n = 9 (8 bit + 1 bit per il segno)
  - $[\![\texttt{000000000}]\!]_2 = [\![\texttt{0}]\!]_{10}$
  - $[\![\texttt{000001000}]\!]_2 = + 1 \times 2^3 = [\![\texttt{8}]\!]_{10}$
  - $[\![\texttt{100001000}]\!]_2 = - 1 \times 2^3 = [\![\texttt{-8}]\!]_{10}$

Svantaggi del modulo e segno

- Il bit di segno è applicato al numero rappresentato, ma non fa propriamente parte del numero in quanto tale
- E' inefficiente: due codifiche per lo 0 ed è complesso da implementare

Codifica complemento a due

- Il C_2 è un sistema di rappresentazione simile al sistema binario naturale, ma il primo bit (quello a sinistra, il più significativo) ha peso negativo.
- Il bit più a sinistra è ancora chiamato bit di segno ma questa volta ha un significato numerico poiché il valore è calcolato tramite una espressione che lo comprende:

$$-1 \cdot b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_0 2^0 = -1 \cdot b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i \cdot 2^i.$$

Esempi di numeri in complemento a due

binario	formula per il calcolo del valore	decimale
$[\![\texttt{000}]\!]_2$	$-0\times2^2 + 0\times2^1 + 0\times2^0$	$[\![\texttt{+0}]\!]_{10}$
$[\![\texttt{001}]\!]_2$	$-0\times2^2 + 0\times2^1 + 1\times2^0$	$[\![\texttt{+1}]\!]_{10}$
$[\![\texttt{010}]\!]_2$	$-0\times2^2 + 1\times2^1 + 0\times2^0$	$[\![\texttt{+2}]\!]_{10}$
$[\![\texttt{011}]\!]_2$	$-0\times2^2 + 1\times2^1 + 1\times2^0$	$[\![\texttt{+3}]\!]_{10}$
$[\![\texttt{100}]\!]_2$	$-1\times2^2 + 0\times2^1 + 0\times2^0$	$[\![\texttt{-4}]\!]_{10}$
$[\![\texttt{101}]\!]_2$	$-1\times2^2 + 0\times2^1 + 1\times2^0$	$[\![\texttt{-3}]\!]_{10}$
$[\![\texttt{110}]\!]_2$	$-1\times2^2 + 1\times2^1 + 0\times2^0$	$[\![\texttt{-2}]\!]_{10}$
$[\![\texttt{111}]\!]_2$	$-1\times2^2 + 1\times2^1 + 1\times2^0$	$[\![\texttt{-1}]\!]_{10}$

Come si ottiene un numero in complemento a due

- L'inverso additivo (o opposto) di un numero rappresentato in C2 si ottiene:
  - Invertendo (negando) ogni bit del numero
  - Sommando 1 alla posizione meno significativa

Esempio

Calcolare il valore in C2 di $[\![\texttt{-11}]\!]_{10}$:

- Parto da $[\![\texttt{11}]\!]_{10}$ in complemento a 2: $[\![\texttt{01011}]\!]_2$
- Ne inverto i bit: $[\![\texttt{10100}]\!]_2$
- Aggiungo 1: $[\![\texttt{10100}]\!]_2 + [\![\texttt{1}]\!]_2 = [\![\texttt{10101}]\!]_2 = [\![\texttt{-11}]\!]_{10}$

Confronto rappresentazioni:

Dec	M & S	C2
127	01111111	01111111
126	01111110	01111110
…	...	...
2	00000010	00000010
1	00000001	00000001
+0	00000000	00000000
0	10000000	-
-1	10000001	11111111
-2	10000010	11111110
…	...	...
-126	11111110	10000010
-127	11111111	10000001
-128	-	10000000

Confronto rappresentazioni

- Binario naturale (*n* bit): $[0, 2^n)$
- Modulo e segno (*n* bit): $(-2^{n-1}, 2^{n-1})$
- Complemento a 2 (*n* bit): $[-2^{n-1}, 2^{n-1})$

Addizione e sottrazione di interi

Addizione di numeri naturali - no overflow

```
       riporto              1  1  1
 77 (decimale)      0  1  0  0  1  1  0  1  +
156 (decimale)      1  0  0  1  1  1  0  0  =

233 (decimale)      1  1  1  0  1  0  0  1
```

Addizione di numeri naturali - con overflow

```
                riporto
                perduto

      riporto     1  1  1  1  1  1
125 (decimale)       0  1  1  1  1  1  0  1  +
156 (decimale)       1  0  0  1  1  1  0  0  =

 25 (decimale)   (1) 0  0  0  1  1  0  0  1

             avrei bisogno di
            un altro bit..
```

Addizione di numeri in C2

```
      riporto                1  1  1
   77 (decimale)       0  1  0  0  1  1  0  1  +
 -100 (decimale)       1  0  0  1  1  1  0  0  =

  -23 (decimale)   .   1  1  1  0  1  0  0  1

                 non ho un
              riporto perduto
```

Addizione di numeri in C2 con overflow

```
      riporto          1     1  1  1
 77 (decimale)         0  1  0  0  1  1  0  1  +
 92 (decimale)         0  1  0  1  1  1  0  0  =

-87 (decimale)    .    1  0  1  0  1  0  0  1

 risultato        nessun riporto
 errato           perduto
```

Regole

- Si può avere overflow senza “riporto perduto”
  - Capita quando da due addendi positivi otteniamo un risultato negativo, come nell’esempio precedente
- Si può avere un “riporto perduto” senza overflow
  - Può essere un innocuo effetto collaterale
  - Capita quando due addendi discordi generano un risultato positivo (si provi a sommare +12 e -7)
  - Se gli addendi sono discordi, non si ha mai overflow