

Informatica B - Esercitazione 1

Codifica dell'informazione

Stefano Cereda
stefano.cereda@polimi.it
02/10/2018

Politecnico di Milano



Codifiche

Numeri interi

Aritmetica in complemento alla base

Esercizio da TdE

Codifiche

Una codifica è una regola arbitraria che consente di dare un significato a dei simboli.

L'insieme di simboli (e.g. cifre) utilizzabili definisce l'alfabeto.

Dato un alfabeto con S simboli, le possibili combinazioni di lunghezza L sono $C = S^L$.

Viceversa, per rappresentare C combinazioni tramite un alfabeto di S simboli avremo bisogno di combinazioni di lunghezza $L = \lceil \log_S C \rceil$ (arrotondamento per eccesso).

In informatica si usano alfabeti di 2 caratteri, ogni carattere viene detto **bit**. 8 bit vengono chiamati **byte**.

Sistemi di numerazione

Possiamo usare diversi sistemi per rappresentare una certa quantità. Diversi sistemi hanno diverse caratteristiche che possono rendere più o meno semplice il calcolo.

Nel 1202 Leonardo Fibonacci introduce in Europa, con il *Liber abaci*, il sistema numerico decimale indo-arabico ed i relativi metodi di calcolo.



Sistema di numerazione posizionale in base 10

Il sistema di numerazioni di tutti i giorni è un sistema **posizionale** in **base 10**.

Un sistema di numerazione è **posizionale** quando ogni simbolo (cifra) utilizzato assume un significato diverso a seconda della posizione che occupa nella notazione.

Il rapporto fra il valore che una cifra assume in una data posizione e quella successiva è definito da una sequenza di moltiplicatori b_1, b_2, b_3, \dots :

$$c_4 c_3 c_2 c_1 = c_4(b_3 b_2 b_1) + c_3(b_2 b_1) + c_2 b_1 + c_1$$

Nel caso più semplice i moltiplicatori sono tutti uguali $b_1 = b_2 = b_3 = b$ e la formula si riduce a:

$$c_4 c_3 c_2 c_1 = c_4 b^3 + c_3 b^2 + c_2 b + c_1$$

Il numero b si chiama **base** del sistema di numerazione.

Codifiche in base 2, 8 e 16

Possiamo inserire la base della codifica come pedice di un numero per indicarne la base.

Utilizzando come basi i valori 2, 8 e 16 otteniamo tre codifiche molto utilizzate in ambito informatico.

Utilizzando la formula precedente è molto semplice ricavare il valore nella base 10:

$$1234_{10} = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 = 1234_{10}$$

$$1010_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 10_{10}$$

$$147_8 = 1 \cdot 8^2 + 4 \cdot 8^1 + 7 \cdot 8^0 = 103_{10}$$

$$B2A_{16} = 11 \cdot 16^2 + 2 \cdot 16^1 + 10 \cdot 16^0 = 2858_{10}$$

Codifica in base 2

La codifica in base 2 (o binaria pura) è molto utilizzata in informatica perché può essere rappresentata tramite presenza o assenza di tensione elettrica.

Per passare dalla base 10 alla base 2 utilizziamo l'algoritmo delle **divisioni ripetute**: dividiamo ripetutamente il numero per 2 fino ad arrivare a zero, la lettura dei resti (in ordine inverso) ci darà il numero in base 2.

$b = 2$	
123	1
61	1
30	0
15	1
7	1
3	1
1	1
0	

$\uparrow 123_{10} = 1111011_2$

In generale, dividendo ripetutamente per b possiamo passare dalla base 10 alla base b .

Le basi 8 e 16 sono molto utilizzate come modo compatto per rappresentare cifre binarie. Infatti, è molto semplice passare da base 2 a base 8 (o 16), basta tradurre a gruppi di tre (o quattro) cifre per volta partendo dalla cifra meno significativa:

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

$$10100101_2 = 215_8 = 55_{16}$$

Rappresentazione di enumerazioni

Tramite una opportuna codifica possiamo rappresentare delle enumerazioni tramite cifre binarie.

Ad esempio, consideriamo l'insieme dei giorni della settimana. Dati $M = 7$ valori distinti da rappresentare, abbiamo bisogno di

$$N = \lceil \log_2 7 \rceil = \lceil 2.83 \rceil = 3 \text{ bit.}$$

lun	0	0	0
mar	0	0	1
mer	0	1	0
gio	0	1	1
ven	1	0	0
sab	1	0	1
dom	1	1	0

Rimane una sequenza non utilizzata (111).

Una delle codifiche per enumerazione più importanti è quella ASCII.

<https://it.wikipedia.org/wiki/ASCII> (notare la colonna CEC)

- 128 caratteri (256 ASCII esteso)
- Caratteri di *comando*, *alfanumerici* e *simboli*
- Proprietà interessanti: $'A' + 1 = 'B'$ e $'c' + ('A' - 'a') = 'C'$

Somme e sottrazioni in base 2

$$\begin{array}{r|l} 0+0 & =0 \\ 0+1 & =1 \\ 1+0 & =1 \\ 1+1 & =0 \text{ con riporto di } 1 \end{array}$$

$$\begin{array}{r|l} 0-0 & =0 \\ 1-0 & =1 \\ 1-1 & =0 \\ 0-1 & =1 \text{ con prestito di } 1 \end{array}$$

$$\begin{array}{r|l} 10 + 01 & =11 \\ 01 + 01 & =10 \\ 11 + 01 & =00 \text{ con overflow} \\ 011 + 01 & =100 \text{ dopo il padding} \end{array}$$

Numeri interi

Notazione in modulo e segno

Utilizzando la codifica binaria vista fin'ora, con N bit possiamo utilizzare 2^N valori: da 0 a $2^N - 1$

Come possiamo rappresentare numeri negativi?

La notifica in modulo e segno utilizza il **MSB** (most significant bit) per indicare il segno:

$$0101_2 = +101_2 = +(1 + 4)_{10} = +5_{10}$$

$$1101_2 = -101_2 = -(1 + 4)_{10} = -5_{10}$$

Con N bit useremo un bit per il segno, i rimanenti per il modulo. Possiamo quindi rappresentare numeri nell'intervallo $-2^{N-1} + 1 \leq x \leq +2^{N-1} - 1$

Abbiamo infatti due possibili codifiche per il numero zero:

$$+0 = +0000 = 00000$$

$$-0 = -0000 = 10000$$

Notazione in complemento alla base

La notazione in modulo e segno spreca un valore, inoltre il calcolare avrebbe bisogno di un circuito per effettuare la somma ed un altro per effettuare la sottrazione.

La **notazione in complemento alla base** risolve questi limiti: con N bit ci permette di rappresentare 2^N valori, precisamente da -2^{N-1} a $+2^{N-1} - 1$.

La rappresentazione in complemento a 2, su N bit, di un numero x_{10} è definita come:

- x_2 se $x \geq 0$ (msb = 0)
- $(2^N - |x|)_2$ se $x < 0$ (msb = 1 in base 2, dimostrare)

Abbiamo quindi una sola rappresentazione per il numero 0.

Attenzione: il msb indica il segno solo se siamo in base 2, **non è un bit di segno**.

Esempio cpl2 su 3 bit

$$N = 3 \rightarrow -2^2 \leq x \leq 2^2 - 1 \rightarrow -4 \leq x \leq 3$$

10	da 10 a cpl2	cpl2
+3	$1+2$	011
+2	2	010
+1	1	001
0	0	000
-1	$2^3 - -1 = 8 - 1 = 7_{10} = 111_2$	111
-2	$8 - 2 = 6_{10} = 110_2$	110
-3	$8 - 3 = 5$	101
-4	$8 - 4 = 4$	100

Notare che msb **non è il segno**: cambiandolo non si ottiene il numero opposto.

Rappresentiamo un numero negativo $-x$ con $N = 4$ bit:

$$-x \rightarrow (2^N - | -x |)_2 = 10000_2 - x_2 = 1111_2 + 1_2 - x_2$$

$1111 - x$ possiamo ottenerlo semplicemente invertendo tutti i bit di x :

1111	-	1010	+
0101	=	0001	=
1010		1011	

Oltre ad usare la definizione, possiamo convertire un numero negativo con questo metodo: convertire l'opposto in binario puro, invertire tutti i bit e sommare 1.

Oltre ad avvantaggiare noi umani, questa proprietà fa sì che il calcolatore necessiti solo di un circuito di addizione (più quello banale di complementazione) per effettuare sia somme che sottrazioni!

Notare che invertire l'opposto e sommare 1 equivale a copiarlo da lsb a msb fino al primo 1 (compreso) ed invertire i bit rimanenti.

Per convertire un numero negativo da base 10 a cpl2 abbiamo quindi tre metodi:

1. Utilizzare la definizione: $(2^N - |x|)_2$:
 $N = 3; -2 \rightarrow (2^3 - 2)_2 = (6)_2 \rightarrow 110$
2. Convertire l'opposto in base 2, invertire i bit e sommare 1:
 $N = 3; -2 \rightarrow +2_{10} = 010_2 \rightarrow 101 \rightarrow 110$
3. Convertire l'opposto in base due, ricopiarlo da *lsb* verso *msb* fino al primo 1 (compreso), copiare i restanti bit complementati:
 $N = 4; -6 \rightarrow +6_{10} = 0110_2 \rightarrow 1010$

Controlliamo l'ultimo risultato utilizzando la definizione di cpl2:

$$1010_2 = 2^N - |x| = 2^N + x \rightarrow x = 1010_2 - 2^N = 10 - 16 = -6$$

Aritmetica in complemento alla base

Dati due numeri in base 10 da sommare algebricamente, dobbiamo innanzitutto controllare il numero di bit necessari:

- Se N è assegnato, dobbiamo verificare che sia sufficiente.
 $(-2^{N-1} \leq x \leq 2^{N-1} - 1)$
- Altrimenti dobbiamo calcolare il valore minimo capace di rappresentare **entrambi** i valori.

$$(-7) + (+2) = -5 \text{ con } N = 4$$

$$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow \text{i bit sono sufficienti}$$

$$-7 \rightarrow 7_{10} = 0111_2 \rightarrow 1001_{cpl2}$$

$$+2 \rightarrow 2_{10} = 0010_2 \rightarrow 0010_{cpl2}$$

$$\begin{array}{r|l} 1001 & + \\ 0010 & = \\ \hline 1011 & \end{array}$$

$$1011 < 0 \rightarrow 0101 = 5 \rightarrow -5$$

Somma con riporto ma senza overflow

$$(+7) + (-2) = +5 \text{ con } N = 4$$

$$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow \text{i bit sono sufficienti}$$

$$+7 \rightarrow 7_{10} = 0111_2 \rightarrow 0111_{cpl2}$$

$$-2 \rightarrow 2_{10} = 0010_2 \rightarrow 1110_{cpl2}$$

$$\begin{array}{r|l} 0111 & + \\ 1110 & = \\ \hline 10101 & \end{array}$$

Ignoriamo il riporto:

$$0101 > 0 \rightarrow 0101 = 5 \rightarrow +5 \text{ Il risultato è corretto}$$

Somma senza riporto ma con overflow

$$(+7) + (+2) = +9 \text{ con } N = 4$$

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ i bit sono sufficienti per gli operandi, ma avremo una situazione di overflow

$$+7 \rightarrow 7_{10} = 0111_2 \rightarrow 0111_{cpl2}$$

$$+2 \rightarrow 2_{10} = 0010_2 \rightarrow 0010_{cpl2}$$

$$\begin{array}{r|l} 0111 & + \\ 0010 & = \\ \hline 1001 & \end{array}$$

$$1001 < 0 \rightarrow 0111 = 7 \rightarrow -7$$

Non abbiamo riporto, cosa ci indica la presenza di overflow?

$$(-7) + (-2) = -9 \text{ con } N = 4$$

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ i bit sono sufficienti per gli operandi, ma avremo una situazione di overflow

$$-7 \rightarrow 7_{10} = 0111_2 \rightarrow 1001_{cpl2}$$

$$-2 \rightarrow 2_{10} = 0010_2 \rightarrow 1110_{cpl2}$$

1001		+
1110		=
<hr/>		
10111		

Ignoriamo il riporto: $0111 > 0 \rightarrow 0111 = 7 \rightarrow +7$

Operandi concordi ma risultato discorde indicano che siamo in una condizione di overflow. (Operandi discordi non daranno mai overflow)

Esercizio da TdE

1. Si dica quale dei seguenti cinque numeri è il maggiore motivando la risposta:
 - 34 in base 8;
 - 34 in base 10;
 - 0A1 in base 16;
 - 111111 in Complemento a 2;
 - 0011011 in Complemento a 2.
2. Si indichi quindi il numero n di bit che permette di codificare tutti i numeri A - E in Complemento a 2 e li si codifichi tutti in Complemento a 2 usando n bit, riportando i passaggi fondamentali dei calcoli.
3. Si eseguano (riportando i calcoli) le seguenti operazioni utilizzando n bit e si indichino eventuali bit di carry (riporto) e overflow:
 - $A - B$;
 - $C + B + E$.

Convertiamo tutti i valori in base 10.

- $34_8 = 3 \cdot 8 + 4 = 24 + 4 = 28_{10}$
- 34_{10}
- $0A1_{16} = 10 \cdot 16 + 1 = 161_{10}$
- 1111111_{cpl2} è negativo, passiamo all'opposto e convertiamolo:
 $0000001_2 = 1_{10}$, dunque il numero vale -1
- 0011011_{cpl2} è positivo, lo convertiamo direttamente:
 $0011011_2 = 1 + 2 + 8 + 16 = 27_{10}$

Dunque il maggiore è $0A1_{16}$.

Notare che non era necessario convertire tutti i numeri: A è minore di B per la base e D è negativo.

Soluzione punto 2 - numero di bit necessari

Con N bit possiamo rappresentare da -2^{N-1} a $2^{N-1} - 1$.

Per il numero minore (-1) basterebbero solo 2 bit, controlliamo dunque il maggiore, sapendo che ci servono almeno 5 bit (ultimo numero):

N	min	max
5	-16	15
6	-32	31
7	-64	63
8	-128	127
9	-256	255

Dunque ci servono 9 bit.

Convertiamo i primi 3 numeri con il metodo delle divisioni ripetute (o con le tabelle per base 8 e 16) e aggiungendo zeri a sinistra fino ad arrivare a 9 bit:

- $34_8 = 000\ 011\ 100_{cpl2}$
- $34_{10} = 000100010_{cpl2}$
- $0A_{16} = 0\ 1010\ 0001_{cpl2}$

Per gli ultimi due numeri aggiungiamo uni o zeri a sinistra fino ad arrivare a 9 bit:

- $1111111_{cpl2} = 11111111_{cpl2}$
- $0011011_{cpl2} = 000011011_{cpl2}$

$A - B = A + (-B)$ dunque troviamo $-B = 111011110_{cpl2}$

000 011 100		+
111 011 110		=
<hr/>		
000 111 000		riporti
111 111 010		risultato

Operandi discordi non possono dare overflow.

Controlliamo il risultato: 111111010_{cpl2} è negativo, quindi convertiamo l'opposto. $000000110_2 = 2 + 4 = 6_{10}$

Dunque il risultato vale -6, che è corretto: $A - B = 28 - 34 = -6$

Soluzione punto 3 - $C + B + E$

010 100 001	+
000 100 010	+
000 011 011	=
<hr/>	
001 000 110	riporti
011 011 110	risultato

Gli operandi sono concordi con il risultato, dunque non si è verificato overflow.

Convertiamo il risultato in base 10:

$$011011110_2 = 2 + 4 + 8 + 16 + 64 + 128 = 222_{10} = 161 + 34 + 27$$