Informatica B - Esercitazione 1

Codifica dell'informazione

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

Politecnico di Milano

Argomenti

Codifiche

Numeri interi

Aritmetica in complemento alla base

Esercizio da TdE

Codifiche

Alfabeti e 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 pozione e quella successiva è definito da una sequenza di moltiplicatori $b_1, b_2, b_3, \ldots$:

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

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.

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:

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

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

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

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

Esempi conversione da base 2 a base 10

Ricordando le potenze di 2 (1,2,4,8,16,32,64,128,...) la conversione diventa molto più facile. Basta sommare le potenze in corrispondenza degli uni:

$01101101_2 = 1 + 4 + 8 + 32 + 64 = 109_{10}$

$10010010_2 = 2 + 16 + 128 = 146_{10}$

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.

$$
\begin{array}{r|l}
\multicolumn{2}{c}{b = 2} \\
123 & 1 \\
61 & 1 \\
30 & 0 \\
15 & 1 \quad \uparrow 123_{10} = 1111011_2 \\
7 & 1 \\
3 & 1 \\
1 & 1 \\
0 &
\end{array}
$$

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

Codifiche in base 8 e 16

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

$$10100101_2 = 010\ 100\ 101_2 = 245_8$$

$$10100101_2 = 1010\ 0101_2 = A5_{16}$$

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 $N = \lceil \log_2 7 \rceil = \lceil 2.83 \rceil = 3$ 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).

ASCII

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

0+0	=0
0+1	=1
1+0	=1
1+1	=0 con riporto di 1

0-0	=0
1-0	=1
1-1	=0
0-1	=1 con prestito di 1

10 + 01	=11
01 + 01	=10
11 + 01	=00 con overflow
011 + 01	=100 dopo il padding

Esercizi di somma

Le somme si eseguono come in base 10. Se alla fine dell'operazione "avanza" un riporto, non possiamo inserire un nuovo bit per tenerne conto, ma dobbiamo segnalare che si è verificato un overflow, ovvero una condizione di errore.

$$\begin{array}{lr|l} \text{primo operando} & 1001 & + \\ \text{secondo operando} & 0101 & = \\ \text{riporti} & & \\ \hline \text{risultato} & & \end{array} \rightarrow \begin{array}{r|l} 1001 & + \\ 0101 & = \\ 1\_ & \\ \hline 0 & \end{array} \rightarrow \begin{array}{r|l} 1001 & + \\ 0101 & = \\ 01\_ & \\ \hline 10 & \end{array}$$

$$\rightarrow \begin{array}{r|l} 1001 & + \\ 0101 & = \\ 001\_ & \\ \hline 110 & \end{array} \rightarrow \begin{array}{r|l} 1001 & + \\ 0101 & = \\ 001\_ & \\ \hline 1110 & \end{array}$$

Esercizi di somma

1101	+
0101	=
1101_	
0010	OVF

0110	+
1010	=
1110_	
0000	OVF

0110	+
0111	=
110_	
1101	

10011	+
01101	=
11111_	
00000	OVF

010101	+
001010	=
00000_	
011111	

010101	+
001111	=
11111_	
100100	

Controllo esercizi di somma

Convertiamo le somme in base 10 per trovare eventuali errori:

- $1+4+16 \quad + \quad 1+4 = 21+5 = 26 > 2^4 \rightarrow$ OVF
- $2+4 \quad + \quad 2+8 = 6+10 = 16 = 2^4 \rightarrow$ OVF (con 4 bit rappresentiamo 16 valori, ma considerando lo zero 16 diventa il *diciassettesimo*)
- $2+4 \quad + \quad 1+2+4 = 6+7 = 13 = 1+4+8$
- $1+2+16 \quad + \quad 1+4+8 = 19+13 = 32 = 2^5 \rightarrow$ OVF
- $1+4+16 \quad + \quad 2+8 = 21+10 = 31 = 1+2+4+8+16$
- $1+4+16 \quad + \quad 1+2+4+8 = 21+15 = 36 = 4+32$

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.

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

10	da 10 a cpl2	cpl2
+3	1+2	011
+2	2	010
+1	1	001
0	0	000
-1	$2^3 - \vert -1 \vert = 8 - 1 = 7_{10} = 111_2$	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.

Proprietà cpl2

Rappresentiamo un numero negativo $-x$ con $N = 4$ bit:
$-x \to (2^N - |-x|)_2 = 10000_2 - x_2 = 1111_2 + 1_2 - x_2$

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

$$\begin{array}{c|c} 1111 & - \\ 0101 & = \\ \hline 1010 & \end{array} \qquad \begin{array}{c|c} 1010 & + \\ 0001 & = \\ \hline 1011 & \end{array}$$

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 si 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.

Metodi di conversione da base 10 a cpl2

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

1. Utilizzare la definizione: $(2^N - |x|)_2$:
   $N = 3; -2 \rightarrow (2^N - 2)_2 = (6)_2 \rightarrow 110$
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:
$1010_2 = 2^N - |x| = 2^N + x \rightarrow x = 1010_2 - 2^N = 10 - 16 = -6$

Gli stessi 3 metodi possono essere usati per convertire un numero negativo da cpl2 a base 10.

Se vogliamo convertire un numero positivo lo possiamo semplicemente convertire in binario puro e poi aggiungere zeri a sinistra (padding) fino a raggiungere il numero di bit desiderati (almeno uno per il segno):

$$18_{10} = (16 + 2)_{10} = 10010_2 = 010010_2$$

Se vogliamo convertire un numero negativo dobbiamo passare all'opposto e convertirlo in cpl2 (come sopra). Infine dovremo complementare il risultato e sommare uno. Il padding verrà fatto con degli uni:

$$-15_{10} = -(8 + 4 + 2 + 1)_{10} = -(1111_2) = -(01111_{cpl2}) = 10001_{cpl2} = 1110001_{cpl2}$$

Esempi di conversione da cpl2 a base 10

Se vogliamo convertire un numero da cpl2 a base 10 guardiamo il primo bit. Se vale zero abbiamo un numero positivo e lo convertiamo come se fossimo in binario puro:

$$010110_{cpl2} = 2 + 4 + 16 = 22_{10}$$

Se il primo bit vale uno abbiamo invece un numero negativo, dobbiamo quindi passare all'opposto (complementando e sommando uno) e convertire in base 10 per po i passare di nuovo all'opposto:

$$1001001_{cpl2} = -(0110111_{cpl2}) = -(1 + 2 + 4 + 16 + 32) = -55_{10}$$

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.

Somma senza riporto ne overflow

Calcoliamo $-7 + 2 = -5$ con $N = 4$ bit.

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ 4 bit sono sufficienti per rappresentare sia gli operandi che il risultato.

-7 è negativo, dunque convertiamo l'opposto (+7) in binario, poi complementiamo e sommiamo 1: $-7 \rightarrow 7_{10} = 0111_2 \rightarrow 1001_{cpl2} = -7$

+2 è positivo, quindi lo convertiamo direttamente (con padding per avere 4 bit): $+2 = 10_2 = 0010_{cpl2}$

Calcoliamo quindi la somma di -7 e +2:
$$\begin{array}{c|c} 1001 & + \\ 0010 & = \\ \hline 1011 & \end{array}$$

Operandi discordi non possono dare overflow.

Il risultato è negativo, dunque complementiamo, sommiamo 1 e convertiamo in base 10, poi passiamo all'opposto: $1011 < 0 \rightarrow 0101 = 5 \rightarrow -5$

$-5 = -7 + 2$ dunque non abbiamo fatto errori.

Somma con riporto ma senza overflow

Calcoliamo $7 - 2 = 7 + (-2) = +5$ con $N = 4$ bit.

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ i bit sono sufficienti sia per gli operandi che per il risultato.

$+7 = 0111_2 = 0111_{cpl2}$
$-2 \rightarrow 2_{10} = 010_2 \rightarrow 110_{cpl2} = 1110_{cpl2} = -2$

0111	+
1110	=
10101	

Operandi discordi non possono dare overflow.

Ignoriamo il riporto:
$0101_{cpl2} = +5$ Il risultato è corretto. Quando eseguiamo operazioni in cpl2 dobbiamo sempre ignorare l'ultimo riporto.

Somma senza riporto ma con overflow

Calcoliamo $7 + 2 = +9$ con $N = 4$ bit.

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ i bit sono sufficienti per gli operandi, ma non per il risultato: ci aspettiamo una situazione di overflow.

$+7 = 0111_2 = 0111_{cpl2}$
$+2 = 0010_2 = 0010_{cpl2}$

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

1001_2 è un numero negativo, dunque passiamo all'opposto 0111_{cpl2}, che convertito in base 10 vale $1 + 2 + 4 = 7$, dunque il risultato vale -7 ed è sbagliato (coerentemente con la situazione di overflow).

Non abbiamo riporto, cosa ci indica la presenza di overflow? Gli operandi sono concordi fra loro ma discordi dal risultato, dunque si è verificato overflow.

Somma con riporto e overflow

Calcoliamo $(-7) + (-2) = -9$ con $N = 4$ bit.

$-2^{N-1} = -8 \wedge 2^{N-1} - 1 = +7 \rightarrow$ i bit sono sufficienti per gli operandi, ma non per il risultato e 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	=
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;
   - 1111111 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.

Soluzione punto 2 - conversione

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:

- $34_8 = 000\ 011\ 100_{cpl2}$
- $34_{10} = 000100010_{cpl2}$
- $0A1_{16} = 0\ 1010\ 0001_{cpl2}$

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

- $1111111_{cpl2} = 111111111_{cpl2}$
- $0011011_{cpl2} = 000011011_{cpl2}$

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

000 011 100	+
111 011 110	=
000 111 00_	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	=
001 000 11_	riporti
011 011 110	risultato

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

Convertiamo il risultato in base 10:
$011011110_2 = 2 + 4 + 8 + 16 + 64 + 128 = 222_{10} = 161 + 34 + 27$

Calcoliamo il risultato di -413 + 324.

$-413_{10} = -(110011101_2) = -(0110011101_{cpl2}) = 1001100011_{cpl2}$
$324_{10} = 101000100_2 = 0101000100_{cpl2} = 0101000100_{cpl2}$

1 001 100 011	+
0 101 000 100	=
0 010 000 00_	riporti
1 110 100 111	risultato

Risultati discordi non danno overflow.

$1110100111_{cpl2} = -(0001011001_{cpl2}) = -(1+8+16+64)_{10} = -89_{10} = -413+324$