Fondamenti di Informatica

Esercizi su rappresentazione binaria dei dati

(a.a. 2019/2020, prof. Gianluca Amato – Francesca Scozzari)

Esercizio 1

Siano dati i numeri 10.2 e 9.8 in base 10. Si convertano i numeri in base 5, si calcoli la loro somma (sempre in base 5) e si riconverta il risultato in base 10.

Soluzione

Iniziamo col convertire in base 5 le parti intere (10 e 9) dei due numeri, col metodo delle divisioni successive:

10 / 5 = 2 col resto di 0
2 / 5 = 0 col resto di 2 $\longrightarrow$ $10_{10}=20_5$

9 / 5 = 1 col resto di 4
1 / 5 = 0 col resto di 1 $\longrightarrow$ $9_{10}=14_5$

Successivamente convertiamo la parte frazionaria col metodo delle moltiplicazioni successive:

0.2 * 5 = 1.0 $\longrightarrow$ $0.2_{10}=0.1_5$

0.8 * 5 = 4.0 $\longrightarrow$ $0.8_{10}=0.4_5$

Dunque, riassumendo:

$$10.2_{10}=20.1_5 \quad , \quad 9.8_{10}=14.4_5$$

Calcoliamo la somma

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 20.1_5 \; + \\ 14.4_5 \; = \\ \hline 40.0_5 \end{array}$$

Dobbiamo riconvetire ora 40_5 in base 10. Da destra verso sinistra, le cifre in un numero in base 5 valgono $5^0=1$ e $5^1=5$ per cui $40^5=4\cdot 5^1+0\cdot 5^0=4\cdot 5=20$.

Note

Si noti che 10.2 + 9.8 = 20 (eseguendo l'operazione normalmente in base 10). Dunque convertire in numeri in base 5, effettuare la somma, e riconvertire in base 10 ci da lo stesso risultato di effettuare direttamente la somma in base 10. Questo ci rassicura sul fatto che non abbiamo commesso errori.

Ricordate inoltre che in un numero in base 5 possono comparire solo le cifre 0, 1, 2, 3 e 4. Se ad un certo punto dello svolgimento dell'esercizio vi ritrovare con un numero in base 5 contenente una cifra diverse, vuol dire che avete sbagliato!!! Questa osservazione vi può sembrare stupida, ma vi assicuro che più di una volta ho visto compiti con questo tipo di errori.

Esercizio 2

Convertire in base 5 i seguenti numeri in base 10: 173, -242.44, 10.1.

Convertire in base 10 i seguenti numeri in base 5: -43.23_5, 21.1_5, 412_5.

Esercizio 3

Sia dato un vettore di 8 numeri interi. Ogni intero è rappresentato in memoria su 16 bit in complemento a 2, in un sistema little-endian. Se il contenuto della memoria occupata dal vettore è:

0A 00 FF 00 01 00 10 00 00 10 01 01 FF FF BC 9A

dire qual è il contenuto di ogni elemento del vettore, in decimale o in ottale.

Soluzione

Consideriamo i primi due byte in memoria: 0A e 00. Assieme essi formano 16 bit e quindi costituiscono la codifica del primo elemento del vettore di interi. La prima cosa da fare è, partendo dai due numeri 0A e 00, ottenere un unico valore a 16 bit. Per questo bisogna considerare l'endianess del sistema: in un sistema big-endian, il byte 0A seguito dal byte 00 corrisponde al valore a 16 bit 0A00, in un sistema little-endian (come richiesto nell'esercizio) il valore a 16 bit è 000A. Questo valore è un numero intero codificato in complemento a 2: occorre capire se si tratta di un numero positivo o negativo. Trasformandolo in binario (e preservando tutti e 16 i bit), si ottiene

$000A_{16} = 0000\ 0000\ 0000\ 1010_2$

Ho inserito degli spazi nel numero binario per renderlo più leggibile. Il bit più a sinistra (che è il bit di segno) è 0 e quindi il numero è positivo. Allora 000A rappresenta proprio il numero $A_{16} = 10_{10} = 12_8$.

Facciamo un altro esempio: gli ultimi due byte sono BC e 9A. Essendo in un sistema little-endian, essi rappresentano il valore 9ABC, che è a sua volta un numero intero nella codifica in complemento a 2.

$9ABC_{16} = 1001\ 1010\ 1011\ 1100_2$

Si tratta di un valore negativo, quindi per conoscere il numero effettivamente rappresentato da 9ABC occorre effettuare l'operazione di complemento a 2. Abbiamo

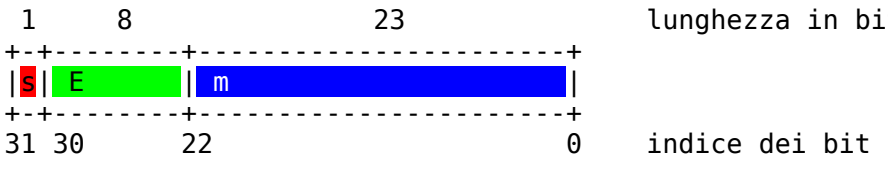
complemento a 2 di $1001\ 1010\ 1011\ 1100_2 = 0110\ 0101\ 0100\ 0011_2 + 1 = 0110\ 0101\ 0100\ 0100_2$

Riportando il numero in una base più leggibile, otteniamo $6544_{16} = 62504_8 = 25924_{10}$ (ricordiamo però che il numero è negativo!). Il risultato dell'esercizio è:

1. in base 10: 10, 255, 1, 16, 4096, 257, -1, -25924.
2. in base 8: 12, 377, 1, 20, 10000, -1, -62504.

Esercizio 4

Lo standard IEEE 754 specifica in maniera esatta come rappresentare dei numeri in virgola mobile. Il formato a 32 bit (precisione singola) dello standard IEEE 754 è il seguente:



S è il bit di segno. E è l'esponente nella notazione in "eccesso 127" (ovvero i numeri da 1 a 254 nel campo E corrispondono agli esponenti da -126 a 127 rispettivamente). I valori 0 e 255 del campo E hanno un significato speciale che è possibile ignorare ai fini della risoluzione di questo esercizio. Il campo m è una stringa di bit che rappresenta la sequenza di cifre dopo la virgola. Tutte le mantisse sono normalizzate in modo che il numero prima della virgola sia 1, per cui per un dato *m* il valore matematico corrispondente è *M*=1.*m*. Maggiori dettagli sono reperibili alla pagina di Wikipedia http://it.wikipedia.org/wiki/IEEE_754.

Dire qual è la rappresentazione in memoria dei numeri 13.25, 0.3, 3.625 e -2.1, usando lo standard IEEE 754 con precisione singola, in un sistema big endian.

Soluzione

Consideriamo il primo numero, 13.25. Prima di tutto bisogna convertire il numero in binario. Usando i soliti metodi, si ottiene $13.25_{10} = 1101.01_2$. Adesso riscriviamo il numero binario usando la notazione scientifica, normalizzando in modo che la mantissa sia della forma 1.xxxxxx. Abbiamo

$1101.01_2 = 1.10101_2 * 2^3$

Dunque, poiché il numero è positivo, abbiamo il bit s = 1 ed m = 10101000000....0 (ovvero 10101 con l'aggiunta di 0 fino ad arrivare a 23 bit). Per quanto riguarda il campo E, esso è la codifica binaria dell'esponente 3 a cui si somma il valore 127, ovvero la codifica binaria di 130 che è 10000010. Alla fine abbiamo la codifica a 32 bit:

$0\,10000010\,10101000000000000000000_2 = 41540000_{16}$.

Poiché non vogliamo semplicemente la codifica a 32 bit di 13.25 ma la sua rappresentazione in memoria, dobbiamo capire come 41540000_{16} viene memorizzato in una sequenza di byte consecutivi. Dato che il sistema è big-endian, i byte vengono memorizzati in memoria dal più al meno significativo, per cui la rappresentazione in memoria sarà 41_{16}, 54_{16}, 00_{16}, 00_{16} .

Per le altre soluzioni, si può consultare uno dei vari convertitori on-line del formato IEEE 754, come http://www.h-schmidt.net/FloatConverter/IEEE754.html.

Note

Il procedimento per convertire gli altri numeri è simile a quello per convertire 13.25. L'unica eccezione è 0.3, che è problematico perché la sua rappresentazione binaria è periodica:

$0.3_{10} = 0.0\overline{1001}_2 = 1.001\overline{1001}_2 * 2^{-2}$

Quando la mantissa viene codificata secondo lo standard IEEE, deve essere troncata ai primi 23 bit, dunque abbiamo m = 00110011001100110011001 10011001. Volendo arrotondare al valore rappresentabile più vicino invece che troncare, notando che il primo bit che non rientra in m è 1, si deve incrementare di uno il valore dei primi 23 bit della mantissa, ottenendo 00110011001100110011010 (la cosa è analoga in base 10: ad esempio 12.346 viene arrotondato in 12.35 quando si vogliono preservare solo due cifre decimali). La versione arrotondata è quella che viene calcolata dal convertitore on-line di cui sopra.

Esercizio 5

Determinare la rappresentazione in memoria (eventualmente approssimata) dei numeri 11.3125 e 98.3, usando

- lo standard IEEE 754 con precisione singola (32 bit)
- la rappresentazione in virgola fissa su 32 bit con 8 bit per la parte frazionaria

- la rappresentazione in virgola fissa BCD su 32 bit con 8 bit per la parte frazionaria

Esercizio 6

Determinare, se possibile, la codifica in memoria (sequenza di byte) dei numeri -1, 65535, 32768, -32768, -16384, 256 e 184, utilizzando la rappresentazione in complemento a 2 su 16 bit in un sistema little-endian.

Soluzione

Prima di tutto, ricordiamo che, utilizzando la rappresentazione in complemento a 2 su 16 bit, si possono rappresentare i numeri da $-2^{15} = -32768$ a $2^{15}-1 = 32767$. Quelli al di fuori di questo intervallo (nel nostro caso 32768 e 65535) non possono essere rappresentati.

Cominciamo dunque a convertire il numero -1 in complemento a due. Per essere più compatti lavoreremo in base 16 invece che in base 2. Occorre calcolare il complemento a 2 di 1. In base 16, questo vuol dire calcolare 0xFFFF + 0x1 – 0x1 = 0xFFFF. Spezzando in due byte, abbiamo 0xFF e 0xFF. Proviamo ora col numero 184. Applicando il metodo delle divisioni successive, poiché 184 = 16 * 11 + 8, abbiamo 184 = 0xB8, ovvero 0x00B8 su 16 bit. Poiché il numero è positivo, 0x00B8 è già la rappresentazione corretta in complemento a 2. Visto che stiamo lavorando in un sistema little-endian, il valore 0x00B8 viene spezzato in due byte mettendo prima il valore meno significato (0xB8) e poi quello più significativo (0x00). Riassumendo, abbiamo:

1. -1 → 0xFF 0xFF
2. 65536 → non rappresentabile
3. 32768 → non rappresentabile
4. -32768 → 0x00 0x80
5. -16384 → 0x00 0xC0
6. 256 → 0x00 0x01
7. 184 → 0xB8 0x00

Esercizio 7

Nel linguaggio di programmazione Java su processori Intel (con architettura little-endian) il tipo short è codificato su 16 bit in notazione in complemento a 2. Si consideri una zona contigua di memoria con il seguente contenuto, espresso come sequenza di byte:

```
0  0  132 0 254 255 255 255 0 0 0 128 231 139 3 130
```

Sapendo che questa zona di memoria contiene un array di 8 valori di tipo short, determinare il contenuto dell'array.

Soluzione

La sequenza di valori è: 0 132 -2 -1 0 -32768 -29721 -32253.

Facciamo alcuni esempi. La prima coppia di byte (0 0) codifica ovviamente il numero 0. La seconda coppia (132 0) rappresenta il numero 0*256+132=132. Siccome il bit più significativo (nella codifica a 16 bit) è posto a 0, il valore ottenuto è positivo e ci fermiamo qui. Consideriamo invece l'ultima coppia (3 130). Essa codifica il numero 130*256+3=33283. Siccome 33283 ha il bit più significativo posto ad 1, bisogna calcolarne il complemento a 2. Poiché 65536-33283=32253,

cambiando segno si ottiene il risultato cercato.

Esercizio 8

Nel linguaggio di programmazione Java il tipo int è codificato su 32 bit in notazione in complemento a 2. Si consideri una zona contigua di memoria con il seguente contenuto, espresso come sequenza di byte:

```
0  0  140 0 254 255 255 255 0 0 0 128 231 139 3 130
```

Sapendo che questa zona di memoria contiene un array di 4 valori di tipo int, determinare il contenuto dell'array.

Esercizio 9

Determinare, se possibile, la rappresentazione in memoria delle seguenti stringhe:

- “Ciriciao gente!” in ASCII e UTF-8
- “Tanto tuonò che piovve.” in ASCII e ISO-8859-1 (Latin-1)

Le virgolette non fanno parte della stringa da codificiare.

Soluzione

Consultando la tabella dei codice ASCII, ad ogni carattere associamo il corrispondente codice. Anche lo spazio va codificato opportunamente. In particolare, il codice numerico corrispondente allo spazio “standard” è il 32.

Pertanto, la rappresentazione in memoria di “Ciriciao gente!” è la seguente sequenza di byte:

- 67 105 114 105 99 105 97 111 32 103 101 110 116 101 33

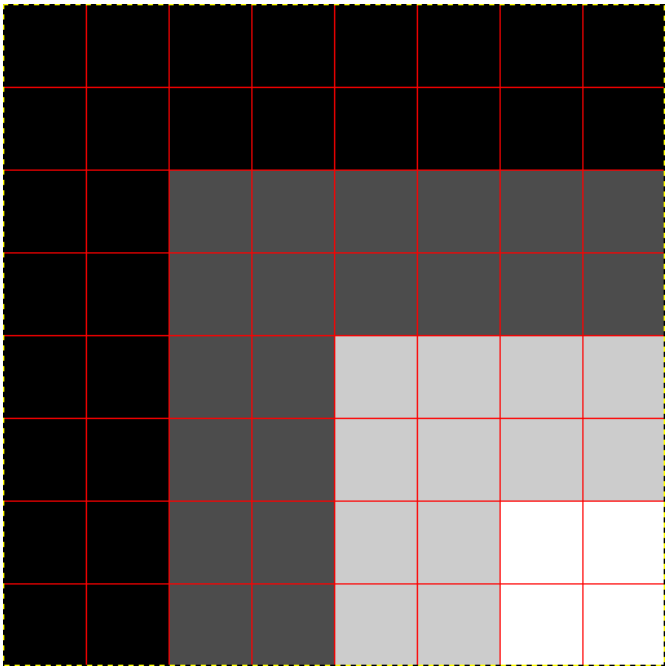
Siccome tutti i codici ASCII standard sono anche codici UTF-8 validi, la rappresentazione in memoria usando UTF-8 è identica.

Per la frase “Tanto tuonò che piovve.” si noti innzanzitutto che le o accentata non fa parte del set di caratteri ASCII, quindi non è possibile codificare la stringa in ASCII. In Latin-1 invece non ci sono problemi. Consultando la tabella del codice Latin-1, otteniamo la sequenza di byte

- 84 97 110 116 111 32 99 104 101 32 112 105 111 118 118 101 46

Esercizio 10

Si consideri l'immagine di 8 x 8 pixel che segue. Le linee rosse non fanno parte dell'immagine ma delimitano i singoli pixel.



Supponiamo di voler memorizzare questa immagine in memoria, in formato a scala di grigi con 2 bit per pixel. Scrivere la sequenza di byte corrispondente all'immagine.

Soluzione

Prima di tutto bisogna capire come codificare ognuno dei pixel con una combinazione di 2 bit. Tipicamente, quando si tratta di immagini in scala di grigio, la combinazione di tutti bit a 0 indica il nero, e quella con tutti i bit a 1 indica il bianco. Adottando questa convenzione abbiamo 00 = nero, 01 = grigio scuro, 10 = grigio chiaro, 11 = bianco. L'immagine viene quindi codificata dalla seguenete sequenza di bit:

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 01 01 01 01 00 00 01 01 01 01 01 01
00 00 01 01 10 10 10 10 00 00 01 01 10 10 10 10 00 00 01 01 10 10 11 11 00 00 01 01 10 10 11 11

Ho lasciato gli spazi che separano le coppie di bit per facilitare la lettura. Adesso, visto che non si vuole una sequenza di bit ma di byte, occorre separare i bit a gruppi di 8, ottenendo la seguente sequenza (che volendo, ma non obbligatoriamente, si può convertire in altra base): 00000000, 00000000, 00000000, 00000000, 00000101, 01010101, 00000101, 01010101, 00000101, 10101010, 00000101, 10101010, 00000101, 10101111, 00000101, 10101111.

Note

La differenza tra sistemi big-endian e little-endian non gioca alcun ruolo in questo esercizio.

Esercizio 11

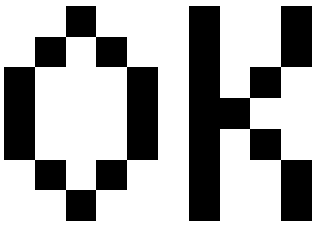
Quello che segue è il contenuto in esadecimale di un file .pnm. Disegnare su foglio l'immagine codificata in questo file. In particolare, si disegni una matrice di caselle con un numero di colonne e righe uguale alla larghezza e altezza dell'immagine, poi si colori ogni casella in maniera opportuna. Nel caso non si disponga di penne del colore giusto, descrivere a parole i colori utilizzati.

```
50 36 0a 23 20 43 52 45 41 54 4f 52 3a 20 47 49 4d 50 20 50 4e 4d 20 46 69 6c 74 65 72 20
56 65 72 73 69 6f 6e 20 31 2e 31 20 63 6f 6d 70 69 74 6f 20 0a 31 30 20 37 0a 32 35 35 0a
ff ff ff ff ff ff 00 00 00 ff ff ff ff ff ff ff ff ff ff 00 00 ff ff ff ff ff ff ff 00 00
ff ff ff 00 00 00 ff ff ff 00 00 00 ff ff ff ff ff ff ff 00 00 ff ff ff ff ff ff ff 00 00
00 00 00 ff ff ff ff ff ff ff ff ff 00 00 00 ff ff ff ff 00 00 ff ff ff ff 00 00 ff ff ff
00 00 00 ff ff ff ff ff ff ff ff ff 00 00 00 ff ff ff ff 00 00 ff 00 00 ff ff ff ff ff ff
00 00 00 ff ff ff ff ff ff ff ff ff 00 00 00 ff ff ff ff 00 00 ff ff ff ff 00 00 ff ff ff
```

```
ff ff ff 00 00 00 ff ff ff 00 00 00 ff ff ff ff ff ff ff 00 00 ff ff ff ff ff ff ff 00 00
ff ff ff ff ff ff 00 00 00 ff ff ff ff ff ff ff ff ff ff 00 00 ff ff ff ff ff ff ff 00 00
```

Soluzione

Questa è l'immagine 10x7 rappresentata nel file PNM. Tutti i punti sono o bianchi o neri (le differenti tonalità di grigio sono un artefatto dello zoom dell'immagine).



Domande teoriche

Domanda 1
Descrivere il metodo di rappresentazione dei numeri reali in virgola mobile.

Domanda 2
Spiegare la differenza tra tecniche di compressione lossy e lossless, facendo degli esempi per entrambi. Un file immagine non compresso è lungo 181 byte. Una volta compresso in formato JPEG diventa lungo 477 byte. Come si spiega il fenomeno?