Approfondimento 1

rappresentazione di interi e floating point

In generale con n bit si possono rappresentare 2^n valori diversi

Per gli interi con n bit, posso rappresentare valori da 0 a $2^n - 1$

<u>esempio</u>: n=6 bits, il massimo che posso rappresentare è
1 1 1 1 1 1 = 32+16+8+4+2+1= 63 con $2^6 = 64$

però vogliamo rappresentare sia interi positivi che negativi
e per farlo usiamo la rappresentazione in complemento a 2 che si
calcola così:

0 1 1 0 1 0 => complemento => 1 0 0 1 0 1 => +1 =>1 0 0 1 1 0

per rappresentare sia i valori positivi che i negativi, possiamo decidere che i valori minori di 2^{n-1} rappresentino i positivi e quelli maggiori e uguali di 2^{n-1} siano i negativi

esempio: sia n=6 bit, quindi i valori più piccoli di
32 saranno positivi e quelli più grandi o uguali di 32 saranno negativi

quindi 0 x x x x x sarà positivo, mentre 1 x x x x x sarà negativo

sappiamo che 3 con 6 bit è 0 0 0 0 1 1, ma come rappresentiamo -3?

per rappresentare -3, usiamo il complemento a 2 della rappresentazione di 3

0 0 0 0 1 1 = 3, prendiamo il complemento di 3
1 1 1 1 0 0 = 63 -3
e sommiamo 1
1 1 1 1 0 1 = 63 – 3 + 1 = 61 = 64 - 3

-3 è rappresentato da 61 e notiamo che 61- 64 = -3

se rifacciamo la stessa operazione su -3
0 0 0 0 1 0 + 1 = 0 0 0 0 1 1 = 3 infatti $2^6 - (2^6 - 3) = 3$

la rappresentazione in complemento a 2 ci permette di calcolare la differenza usando la somma:

$x – y = x + (-y)$ e $-y$ è il complemento a 2 di y

- calcoliamo $3 + (- 3) = 000011 + 111101$

e se vogliamo calcolare «al volo» la rappresentazione in complemento a 2 di -15 ?

$x – 64 = -15 \Rightarrow x = 49$

la cosa funziona per 0 che non è né positivo né negativo:
0 0 0 0 0 0 => 1 1 1 1 1 1 + 1 = 0 0 0 0 0 0

il più grande positivo che possiamo rappresentare è :
31 = 0 1 1 1 1 1 e - 31 si ottiene calcolando il complemento
cioè 1 0 0 0 0 0 e sommando 1 => 1 0 0 0 0 1 = 33 = 64 -31

si osservi che -31 non è il più piccolo negativo visto che abbiamo
1 0 0 0 0 0 = -32

mentre +32 non è rappresentabile
con 6 bit si rappresentano -32….0…..31, ovviamente 64 valori

rappresentazione in floating point:

con n bit a disposizione, la rappresentazione floating point viene fatta riservando

- il bit più a sinistra per il segno,
- k bit per l'esponente
- m bit per la mantissa (cioè la parte decimale)

Ovviamente $1+m+k=n$

nella mantissa si rappresenta solo la parte decimale perché la parte intera è sempre 1 che infatti viene sottointesa

Se i k bit per l’esponente contengono un numero x in $[0..2^k – 1]$, allora l’esponente da applicare è x – bias, dove bias = $(2^{k-1} -1)$ in questo modo si rappresentano sia esponenti positivi che negativi

quindi se s è il segno, w il contenuto della mantissa ed esp = x-bias, il valore rappresentato è

$(-1)^s \; 1.w * 2^{esp}$

nel caso dei float (precisione semplice): 32 bit

1 per il segno,
8 per l'esponente
23 per la mantissa

bias = $2^7 - 1 = 127$ quindi gli esponenti vanno da -127 a +128
però i valori -127 e +128 sono riservati
-127 per valori più piccoli del minimo rappresentabile
+128 per valori numerici non rappresentabili
quindi restano a disposizione gli esponenti -126..+127

esercizio:

se consideriamo 4 bit e esponente, allora il bias = 2^3 -1 = -7 e gli esponenti vanno da -7 a 8 e quelli utili da -6 a 7

Esempio 1. Supponiamo che n=9 e che si voglia dedicare 4 bit per la mantissa e per l'esponente

Immaginiamo di voler rappresentare 9, la sua rappresentazione binaria è 1 0 0 1, quindi se lo vogliamo rappresentare in floating point con parte intera 1, avremo
1,001 e quindi per recuperare il valore 9 dovremo mettere l'esponente a 10 (osserva, 10 -7=3) infatti $1{,}001 * 2^{(10-7)} =$ 1001=9 in base 10

Esempio 2. Sempre con n=9 e mantissa ed esponente di 4 bit, vediamo come rappresentare 0,34

per trovare il corrispondente valore binario, basta moltiplicare 0,34 per 2 e controllare se otteniamo un valore di almeno 1,x oppure 0,x nel primo caso il bit più a sinistra della rappresentazione che cerchiamo è 1, nel secondo caso è 0 e per calcolare il resto del numero binario, dobbiamo continuare con x finché non otteniamo 0 oppure otteniamo il numero di bit a disposizione, nel nostro esempio 4 (dimensione della mantissa)

ovviamente se questo processo non arriva a x=0, la rappresentazione binaria sarà solo approssimata facciamo quanto appena detto:

,34*2 = ,68 ➔0

,68*2=1,36 ➔1

,36*2=,72 ➔0

,72*2=1,44 ➔1

,44*2 = ,88 ➔0

,88*2 =1,76 ➔1

la rappresentazione di 0,34 calcolata finora è ,010101

se spostiamo la virgola per mettere a 1 la parte intera, otteniamo 1,0101 e quindi abbiamo esaurito i 4 bit a disposizione per la mantissa. Per spostare la virgola di 2 posizioni verso sinistra, l'esponente deve essere -2 e quindi 5 visto che 5-7=-2 e quindi con 4 bit avremo 0101 si osservi che $1{,}0101 * 2^{-2}$ non è una rappresentazione binaria esatta di 0,34.

L'errore è dovuto alla finitezza della mantissa.

a che valore reale corrisponde il numero floating point $1{,}0101 * 2^{-2}$?