Range di rappresentazione

Nel complemento a 2 può capitare che in una somma fra 2 valori in codice binario, il riporto della somma vada una cifra in più a sinistra, facendo uscir fuori un risultato diverso dall'effettiva somma dei 2 numeri (Tale problema può presentarsi esclusivamente quando i valori hanno lo stesso segno) :

Esempio

  1 0 1 0 +
  1 0 0 0 =
----

1 0 0 0 0

 il numero evidenziato a causa dell'overflow non viene contato, quindi il risultato di 1010+1000 non può essere rappresentato con 4 bit, per questo si adotta l'estensione dei bit.

Si aggiungono bit significativi sulla sinistra quanti ne servono per rientrare nel valore, se il numero è positivo si aggiungono tutti 0, se negativo si aggiungono tutti 1.

L'operazione di prima diventa :

1 1 0 1 0 +

1 1 0 0 0 =
----

1 0 0 0 0

Prodotto di due numeri binari

La moltiplicazione fra 2 numeri binari si esegue in colonna come le moltiplicazioni alle elementari.

```
   0101 *
   0111 =
-----------------
   0101
  0101
 0101
0000
----------------
100011
```

Le frazioni in codice binario

Come possiamo rappresentare il numero 3,5?

3,5 = 7/2

Si shifta verso destra di una posizione, quindi diventa 011,1

I valori dopo la virgola continuano con la numerazione 2^-1. ESEMPIO :

quindi i numeri binari dopo la virgola partono da 0,5 e per ogni posizione verso destra tale valore viene dimezzato.

Binario	decimale
0,1	0,5
0,01	0,25
0,001	0,125
0,0001	0,0625
0,11	0,75

Come si rappresenta il numero 3,4 ?

La parte intera è 3, la parte decimale è 0,4.

3 in binario vale 11. Ma come rappresentiamo 0,4?

3,4 = 11,011 questi valore binario equivale a 0,375. Si avvicina al 0,4 ma non è preciso, più cifre aggiungiamo dopo la virgola più potremmo rappresentare con precisione il valore.

Come convertire da decimale a binario un numero con la virgola

3,75 in binario : si rappresenta prima la parte intera cioè 3.

11 = 3

Rimane da rappresentare 0,75. Si esegue quindi un procedimento ricorsivo. Si moltiplica il valore dopo la virgola per 2. Del risultato, il valore a sinistra della virgola viene aggiunto come valore decimale nel nostro numero binario. Il valore a destra della virgola subirà poi lo stesso procedimento ricorsivo finché non uscirà un risultato con il valore a destra della virgola uguale a 0.

Si vede l'esempio

11 , 0,75 * 2 = 1,5

11 , 1 0,5 * 2 = 1 , 0 ← è 0, quindi finisce qui l'operazione.

11 , 11

3,75 = 11,11

Adesso prendiamo come esempio 3,4 ed eseguiamo lo stesso procedimento. 3 = 11

11,   0,4 * 2 = 0,8

11,0   0,8 * 2 = 1,6

11,01   0,6 * 2 = 1,2

11,011   0,2 * 2 = 0,4

11,0110   0,4 * 2 = 0,8 Siamo ritornati allo stesso valore di partenza. Ciò ci fa capire che tale operazione andrebbe avanti all'infinito in modo periodico. Possiamo dunque dire che il periodo è 0110, più cifre aggiungeremo più ci avvicineremo con precisione a 3,4, senza però mai uguagliarlo. $3{,}4 = 11{,}\overline{0110}$

Ovviamente in un sistema digitale non possiamo rappresentare un numero periodico, dato che abbiamo a disposizione un numero finito di bit.

Esistono 2 modi per rappresentare i numeri con la virgola :

Numeri a virgola fissa (fixed point numbers)

Nei numeri a virgola fissa ci si mette d'accordo su quanti bit rappresenteranno la parte a destra della virgola. Facciamo finta questi siano 4, in più abbiamo 4 bit per rappresentare i numeri a sinistra della virgola.

6,75 = 01101100 perché 0110 , 1100 = 6 , 75

Si possono rappresentare anche i numeri negativi con il complemento a 2.

Rappresentiamo -7,5 con 4 cifre frazionarie e 8 bit totali in complemento a 2 :

prima di tutto prendiamo 7,5 ed invertiamolo :

0 1 1 1 1 0 0 0 = 1 0 0 0 0 1 1 1

Aggiungiamo +1 : 1 0 0 0 0 1 1 1 + 0 0 0 0 0 0 0 0 1 = 1 0 0 0 1 0 0 0

Il risultato è chiaro : 1 0 0 0 1 0 0 0

-8   +0,5   = -7,5

Numeri a virgola mobile (floating point numbers)

Prima di capire come funzionano i numeri binari a virgola mobile dobbiamo capire come funziona l'annotazione scientifica dei numeri.

Il numero 273 in annotazione scientifica viene scritto = $2{,}73 \ * \ 10^2$

Lo standard è $\pm M * B^E$ dove

M = mantissa (essa misura la precisione e deve essere maggiore o uguale ad 1)

B = base (nella numerazione decimale è 10, in quella binaria è 2)

E = esponente

Per questa rappresentazione in numero binario, si utilizza uno standard chiamato IEEE 754 dove ai numeri vengono riservati tali bit :

1 bit per il segno S (esso è il più significativo a sinistra, se 0 positivo, se 1 negativo)

8 bit per l'esponente E (al valore dell'esponente viene sottratto poi -127)

23 bit per la mantissa M

Per un totale di 32 bit, si utilizza questa formula :

$$S(\pm)\ 1, M * 2^{E-127}$$

1 bit 23 bit 8 bit

In un numero binario con standard IEEE 754 i 32 bit per rappresentare segno, mantissa ed esponente vengono disposti in questo modo :

1 _ 10010010 _ 01011010010110101011010

Segno esponente mantissa

il bit a sinistra per il segno, i primi 23 per la mantissa e gli altri 8 per l'esponente.

Conversione da decimale a binario (IEEE 754)

Osserviamo in 3 step come si converte un numero decimale a binario con standard IEEE 754. Convertiamo il numero 228

1. Convertiamo il numero in un binario normale : 228 -> 1100100
2. Scriviamo 1100100 in annotazione scientifica : $1{,}11001 * 2^7$
3. Riempiamo i campi dello standard IEEE 754 osservando l'annotazione scientifica scritta nel passaggio 2.
   Ricordiamo la formula $S(\pm)\ 1{,}M * 2^{E-127}$ e riempiamo i campi.
   segno = 0 perché il numero è positivo
   esponente = 7 + 127 (si somma 127) = 134 -> 10000110
   mantissa = 11001 (la parte a sinistra della mantissa, dato che nella formula è già previsto che il numero sia frazionario e maggiore di 1.) Gli altri campi della mantissa che non sono riempiti andranno riempiti con degli zero.

Come risultato otteniamo il numero binario a 32 bit :

0 10000110 11010000000000000000000

segno esponente mantissa

Questi 32 bit equivalgono ad 8 nibble, quindi raggruppandoli in gruppi da 4 possiamo rappresentare un numero esadecimale da 8 posizioni :

0x43640000