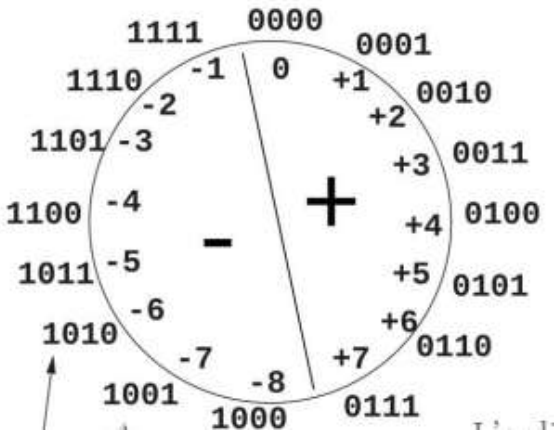
Lezione 3 Rappresentazione Dati 2

venerdì 29 settembre 2023 12:47

Riepilogando il complemento a 2 :



Da notare che con questa rappresentazione posso rappresentare sia i numeri positivi che i numeri negativi, ma da notare che il numero 8 (1000) non ha la controparte positiva : i positivi da 0 a 7 , mentre i negativi da 8 a 15 . **Infatti il valore massimo è 2 ^n-1** , il quale "-1" è dovuto alla rappresentazione dello "0" che leva il posto al "+8". Ricordiamo le operazioni di somma e sottrazione le quali vengono effettuate normalmente, ignorando il riporto più significativo. Ma attenzione al risultato : può capitare che il risultato superi il numero di cifre consentite, portando cosi' il sistema in una condizione di **overflow**:

1. Se sommo due numeri opposti il risultato è corretto
2. Se sommo due numeri positivi il risultato eccede la rappresentazione
3. Se sommo due numeri negativi il risultato eccede la rappresentazione

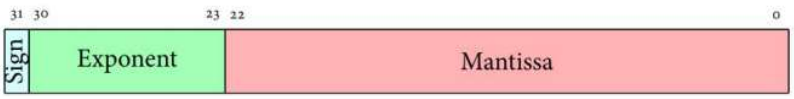
Per accorgermi se sono in questa condizione, devo vedere gli ultimi 2 riporti : se sono concordi va bene, altrimenti ho avuto un'inversione di segno durante l'operazione.
Similmente a questa divisione vi è un'altra rappresentazione : Rappresentazione in eccesso , ovvero in questa rappresentazione si sceglie opportuno valore di "0" : ovvero il valore k che è uguale a 2^n-1 , dividendo cosi' l'intervallo di numerazione in 2 sotto intervalli. In questa rappresentazione , chiamata anche offset binary, è possibile traslare lo "0" in avanti di k valori, avendo cosi la rappresentazione dello "0" unica e con cifra più significativa="1". Da notare che si mantiene ordinamento numero/numerale .

$$x' = x + k \qquad x = x' - k$$

Con intervallo di rappresentazione pari a : [-2^n-1,2^n-1-1].
Ampliamo ora il nostro intervallo di rappresentazione : usiamo la rappresentazione in virgola mobile : ovvero possiamo rappresentare un'intervallo maggiore a parità di numero di bit. La gestione di questa rappresentazione, viene gestita da un'apposita circuiteria molto complessa e laboriosa : FPU (floating ponit unit) .I dettaglio : **questa rappresentazione permette di spostare la virgola .**

$$12{,}345 = \underbrace{1{,}2345}_{\text{mantissa}} \times \underbrace{10^{1}}_{\text{base}} \} \text{ esponente}$$

Questa rappresentazione segue lo standard IEEE754 a 32 bit:



Usando esponente come eccesso e =E + 127.
Da notare che il segno ha un solo bit (0 -> positivo, 1->negativo), l'esponente ne ha 8 mentre la mantissa i 23 restanti. La mantissa è rappresentata come (1.x). Quindi un numero in questo formato viene scritto cosi':

$$(-1)^{s} \cdot 2^{e-127} \cdot 1.m$$

Vediamo un esempio : (-53,1) base 10 :

1. Convertire il numero in binario
   a. 53 -> 110101
   b. 0.1->$0\overline{0011}$
   c. -110101,00011001100110011
2. Sposto la virgola
   a. -1,10101000110011001100110011*(2^5)
3. Estrapolo i valori:
   a. s=1
   b. Esponente: 5+127=132
   c. Mantissa= 10101000110011001100110011
4. Rappresentazione dell'esponente
   a. 132 = 10000100
   b. Se minore di 8 aggiungo "0" a sx
5. Codifica finale
   a.

1	1000100	1010 1001 0011 0011 0011 0011

Quindi è possibile cambiare la rappresentazione della parola, quindi di una stessa parola si può cambiare il contesto:

Quindi la parola precedente diventa in hex : C2546666

Questi tipi di numeri di dicono normalizzati. In antitesi a questi ci sono i denormalizzati, i quali seguono la stessa logica, ma sono quelli che rappresentano numeri vicini allo 0, ommettendo cosi la parte "0" del numero .

Da notare che in questa notazione la rappresentazione dello "0" è doppia, cosi' come "infinito" (attenzione ad overflow) e cosi come i NAN (not a number) . In dettaglio :

e	m	Tipo di valore
[1,254]	qualsiasi	$(-1)^s \cdot 2^{e-127} \cdot 1.m$ (numeri normalizzati)
0	$\neq 0$	$(-1)^s \cdot 2^{-126} \cdot 0.m$ (numeri denormalizzati)
0	0	$(-1)^s \cdot 0$ (zero con segno)
255	0	$(-1)^s \cdot \infty$ (infinito con segno)
255	$\neq 0$	NaN

Per quanto riguarda la gestione dei NAN si hanno varie casistiche, sollevando delle eccezioni, ovvero delle condizioni di errore non previste che devono essere trattate opportunamente :

1. Operazione invalida
   a. Operazione non corretta
2. Overflow
   a. Numero eccede la rappresentazione
3. Divisione per zero:
   a. Divido per zero
4. Underflow:
   a. Overflow per risultati troppo piccoli
5. inesatto:
   a. Errore di arrotondamento

Vediamo ora i valori tipici della mantissa valori (normalizzati):

1. Esponente massimo : -126
   a. Mantissa minima : 2^-126
2. Esponente minimo : 127
   a. Mantissa massima : 2^127*(2-2^23)

Vediamo ora i valori tipici della mantissa valori (denormalizzati):

1. Esponente : 126
   a. Mantissa minima : 2^-126*2^-23=0 2^-149
   b. Mantissa massima : 2^-126*(1-2^-23)

Vediamo ora come sono disposti i numeri negli intervalli , ma usando la virgola mobile , quindi approssimando un numero, può capitare che vi sia un errore di approssimazione . Questi errori sono dovuti sia alla modalità con cui sono sparsi ,sia al sistema di numerazione usato.

Da notare che la maggior parte dei numeri , o meglio della densità, sono compresi tra -1 e 1 e tra 65563 e 131072.Come vedere l'errore commesso nell'approssimazione ??? Secondo due modi :errore assoluto ed errore relativo. Il primo dice quanto mi sono discostato da informazione originale, mentre il secondo dice se errore commesso è piccolo o grande .

$$\varepsilon_A = x - x' \qquad \varepsilon_R = \frac{x - x'}{x}$$

Mentre per vedere il numero delle cifre affette da errore si usa il logaritmo in base :

$$-\log_{10} \varepsilon_R$$

Vediamo un esempio :

Consideriamo $x = 3.5648722189$ e supponiamo di volerlo rappresentare utilizzando soltanto quattro cifre decimali:

$$x \approx \bar{x} = 3.5648$$

Con questa approssimazione, otteniamo un errore relativo pari a:

$$\varepsilon_R = \frac{x - \bar{x}}{x} = \frac{3.5648722189 - 3.5648}{3.5648722189} = 0.000020258$$

Il numero di cifre affidabili di $\bar{x}$ è pari a:

$$-\log_{10}(0.000020258) = 4.69$$