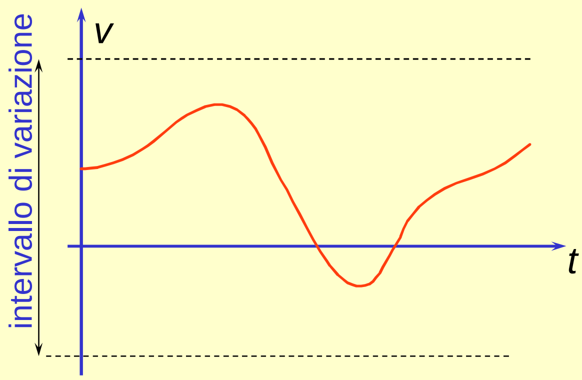
Aritmetica dei Calcolatori Elettronici

Prof. Orazio Mirabella

L'informazione Analogica

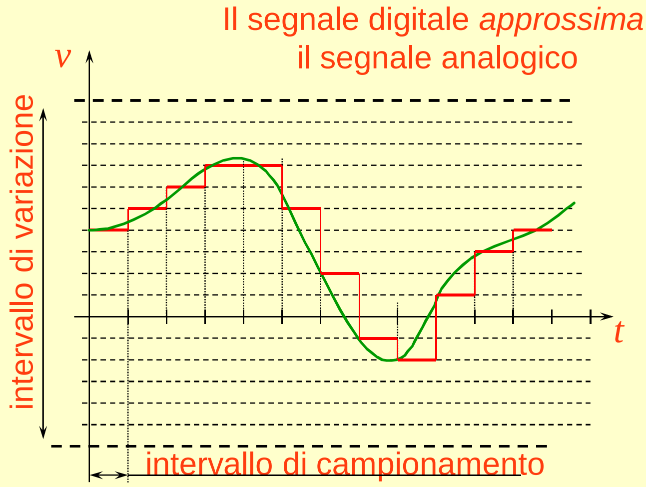
Segnale analogico: variabile continua

assume un numero infinito di valori entro l'intervallo di variazione



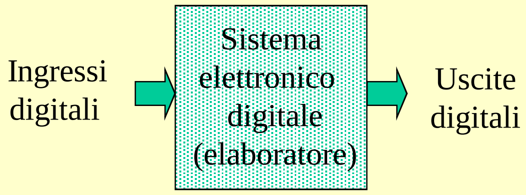
Informazione digitale: variabili discrete

assumono un numero finito di valori entro l'intervallo di variazione



L'*approssimazione* migliora al crescere del numero di valori discreti in cui viene suddiviso l'intervallo di variazione e l'intervallo di campionamento temporale

Elaborazione digitale



La comunicazione fra mondo analogico e digitale richiede:

Teorema del campionamento (Nyquist): $T_s = 1/(2 \times f_{max})$

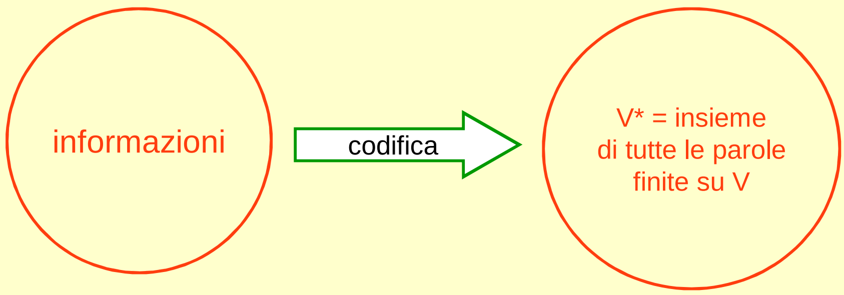
Discretizzazione del tempo e delle ampiezze dei segnali

Codifica di informazioni discrete

V = vocabolario = insieme finito di simboli = {a,b,c,…}

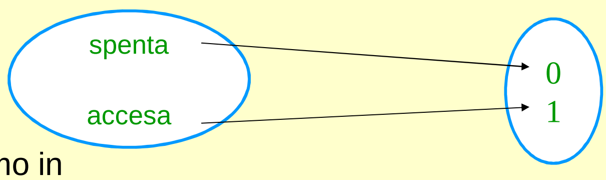
parola = sequenza finita di simboli in V = $\alpha_1 \dots \alpha_n$

esempio: “aabcedc” è una parola di n = 7 simboli

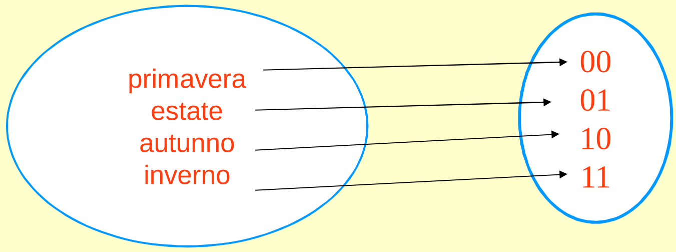


Codifica binaria

La lampadina è



Siamo in



La codifica dei numeri

Rappresentazioni dal passato

La rappresentazione *unaria* è sicuramente la codifica più semplice dei numeri

La barretta I rappresenta il numero 1

La sequenza IIIIII denota il numero 6 e così via

La dimensione della rappresentazione cresce in modo lineare con il numero da rappresentare: Impraticabile per gestire numeri ‘grandi’

Per minimizzare la dimensione della rappresentazione i romani hanno introdotto una codifica basata sui multipli di 5 - Alfabeto: I V X L C D M

I numeri si ottengono come combinazioni di tali simboli.

19 rappresentato da X IX

1281 rappresentato da M CC LXXX I

50.000 rappresentato da una sequenza di 50 M

Rappresentazione decimale

- La numerazione decimale utilizza una *codifica posizionale basata sul numero 10* e sull’alfabeto di simboli 0 1 2 ... 9
- I numeri si leggono da sinistra a destra e sono associati a potenze di 10 (mille, diecimila, ecc)
- Es. la sequenza `312’ rappresenta il numero $3 \times 10^{2} + 1 \times 10^{1} + 2 \times 10^{0}$
- La notazione posizionale può essere utilizzata in qualsiasi altra *base*

Numerazione in base B

Fissata una qualsiasi base $B > 1$ la sequenza

$c_n \, c_{n-1} \ldots c_1 \, c_0$ dove ciascun $c_k < B$

rappresenta il numero

$$r = c_0 \times B^0 + c_1 \times B^1 + \ldots + c_{n-1} \times B^{n-1} + c_n \times B^n$$

Basi comunemente usate:

Base decimale **B = 10**: alfabeto 0,1,2,3,4,5,6,7,8,9

Base binaria **B=2**: alfabeto 0,1

Base ottale **B=8**: alfabeto 0,1,2,3,4,5,6,7

Base esadecimale **B=16**: alfabeto 0,1,…,9,A,B,C,D,E,F
dove A vale 10, B vale 11,…, F vale 15

Sistema di numerazione decimale

Cifre (digits): $C_i \in$ *0, 1, 2, …9.*

Numero$_{10}$ *:* $$N_{10} = \sum_{k=0}^{k=n-1} C_i \times 10^k$$

Alla *posizione* della cifra nel numero è associato un *peso decimale*

Esempio (numero intero):

$$3459_{10} = 3\times10^3 + 4\times10^2 + 5\times10^1 + 9\times10^0$$

migliaia centinaia decine unità

Esempio (numero frazionario decimale):

$$34.59_{10} = 3\times10^1 + 4\times10^0 + 5\times10^{-1} + 9\times10^{-2}$$

decine unità decimi centesimi

Sistema di numerazione binario

- Se B=2 gli unici simboli dell'alfabeto sono 0 & 1
- la sequenza $c_n\, c_{n-1}\, c_{n-2} \cdots c_1\, c_0$

Dove ciascun $c_k < 2$

rappresenta il numero $c_n \times 2^n + c_{n-1} \times 2^{n-1} + \ldots c_1 \times 2^1 + c_0 \times 2^0$

si possono anche rappresentare numeri frazionari

- la sequenza $C_n\, C_{n-1}\, C_{n-2} \cdots C_1\, C_0,\, C_a\, C_b \cdots C_k$

rappresenta il numero

$c_n \times 2^n + c_{n-1} \times 2^{n-1} + \ldots c_1 \times 2^1 + c_0 \times 2^0 + c_a \times 2^{-1} + c_b \times 2^{-2} + \ldots c_k \times 2^{-k}$

Es. la sequenza **1011** in base 2 denota il numero

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{(10)}$$

mentre la sequenza **1011,11** denota il numero

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 11,75_{(10)}$$

Sistema di numerazione binario

Le Cifre 0, 1 sono chiamate Binary digit =Bit

In generale un numero binario non intero può essere espresso come:

$$N_2 = \sum_{k=0}^{k=n-1} C_i \times 2^k + \sum_{K=-1}^{k=-m} C_i \times 2^k$$

Alla *posizione* della cifra nel numero è associato un *peso binario*

Esempio (numero intero):

$$10111_2 = 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 \left(= 23_{10}\right)$$

Esempio (numero frazionario):

$$10.111_2 = 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3} \left(= 2.875_{10}\right)$$

Il sistema binario è particolarmente vantaggioso per semplificare la realizzazione dei circuiti elettronici di elaborazione numerica e per interfacciarsi con calcolatori e microprocessori

Sistema di numerazione ottale

Cifre $C_i \in$ *0, 1,…, 7.*

Numero: $$N_8 = \sum_{k=0}^{k=n-1} C_i \times 8^k$$

esempio: $$161_8 = 1\times 8^2 + 6\times 8^1 + 1\times 8^0 \left(=113_{10}\right)$$

Sistema di numerazione esadecimale

Cifre $C_i \in$ *0, 1,…, 9, A, B, C, D, E, F.*

Numero: $$N_{16} = \sum_{k=0}^{k=n-1} C_i \times 16^k$$

esempio:

$$F15A_{16} = 15\times 16^3 + 1\times 16^2 + 5\times 16^1 + 10\times 16^0 \left(= 61786_{10}\right)$$

Conversione fra sistemi di numerazione

Regola generale:
conversione del numero reale *a* (maggiore di 1) in base *b* =10
nel corrispondente numero reale *c* in base *d*.
c si ottiene dividendo *a* per *d*.
I resti ad ogni passo della divisione rappresentano le cifre di c.

Esempio: convertire 27_{10} nel suo equivalente in base *2*
in questo caso: c=?; a=27(decimale >1); b=10; d=2

27:2=13 resto 1 (bit meno significativo: peso 2^0)
13:2=6 resto 1 (peso 2^1)
6:2=3 resto 0 (peso 2^2)
3:2=1 resto 1 (peso 2^3)
1:2=0 resto 1 (bit più significativo: peso 2^4)

per cui: $27_{10}=11011_2 (=1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)$

Conversione Decimale / Binario

ESEMPIO: $18751_{10} = 100100100111111_{2}$

18751	
9375	1
4687	1
2343	1
1171	1
585	1
292	1
146	0
73	0
36	1
18	0
9	0
4	1
2	0
1	0
0	1

CONTROPROVA:

$1\text{x}2^{14} + 1\text{x}2^{11} + 1\text{x}2^{8} + 1\text{x}2^{5} + 1\text{x}2^{4} + 1\text{x}2^{3} + 1\text{x}\,2^{2} +$

$+ 1\text{x}2 + 1 =$

$= 16384 + 2048 + 256 + 32 + 16 + 8 + 4 + 2 + 1 =$

$= 18751$

Conversione da Binario a Decimale

$1101_2 \longrightarrow ?_{10}$

$11100110_2 \longrightarrow ?_{10}$

$1010100_2 \longrightarrow ?_{10}$

$111000100_2 \longrightarrow ?_{10}$

Conversione da Binario a Decimale

$1101_2 \rightarrow ?_{10}$ (13)
$11100110_2 \rightarrow ?_{10}$ (230)
$1010100_2 \rightarrow ?_{10}$ (84)
$111000100_2 \rightarrow ?_{10}$ (452)

Conversione da decimale a binario

$83_{10} \rightarrow ?_2$
$330_{10} \rightarrow ?_2$
$2291_{10} \rightarrow ?_2$
$9902_{10} \rightarrow ?_2$

Conversione da decimale a binario

$83_{10} \rightarrow ?_2$	(1010011_2)
$330_{10} \rightarrow ?_2$	(101001010_2)
$2291_{10} \rightarrow ?_2$	(100011110011_2)
$9902_{10} \rightarrow ?_2$	(10011010101110_2)

Conversione Decimale / Binario parte frazionaria

Conversione del numero reale *a* (minore di 1) in base *b*=10
nel corrispondente numero reale *c* in base *d*.
C si ottiene moltiplicando *a* per *d*. Le parti intere ad ogni passo della moltiplicazione rappresentano le cifre di c.

Esempio: convertire 0.375_{10} nel suo equivalente in base 2

in questo caso: c=?; a=0.375(decimale <1); b=10; d=2
0.375×2 = 0.750 parte intera 0; parte frazionaria 0.750; (peso 2^{-1})
0.750×2 = 1.500 parte intera 1; parte frazionaria 0.500; (peso 2^{-2})
0.500×2 = 1.000 parte intera 1; parte frazionaria 0.000; (peso 2^{-3})
per cui: 0.375_{10}=0.011_2 (=0+0´ 2^{-1} + 1´ 2^{-2} + 1´ 2^{-3})

Per convertire un numero frazionario, basta procedere separatamente alla conversione della parte intera e della parte frazionaria

Analogamente per le conversioni da decimale in altri sistemi numerici

Esempio : convertire $207{,}874_{10}$ nel suo equivalente in base 2

Parte intera

207:2=103 resto 1 (peso 2^0)
103:2=51 resto 1 (peso 2^1)
51:2=25 resto 1 (peso 2^2)
25:2=12 resto 1 (peso 2^3)
12:2= 6 resto 0 (peso 2^4)
6:2= 3 resto 0 (peso 2^5)
3:2= 1 resto 1 (peso 2^6)
1:2= 0 resto 1 (peso 2^7)

Parte frazionaria

0.874x2=0.748 parte intera 1 (peso 2^{-1})
0.748x2=0.496 parte intera 1 (peso 2^{-2})
0.496x2=0.992 parte intera 0 (peso 2^{-3})
0.992x2=0,984 parte intera 1 (peso 2^{-4})
0.984x2=0.968 parte intera 1 (peso 2^{-5})
0.968x2=0.936 parte intera 1 (peso 2^{-6})
0.936x2=0.872 parte intera 1 (peso 2^{-7})
0.872x2=0.744 parte intera 1 (peso 2^{-8})

... troncamento alla ottava cifra dopo la virgola

per cui: $207{,}874_{10}=11001111{,}11011111_2$

Esempio : convertire $72{,}87_{10}$ nel suo equivalente in base 2

Parte intera

Esempio : convertire $72{,}87_{10}$ nel suo equivalente in base 2

Parte intera

72:2=36	resto 0 (peso 2^0)
36:2=18	resto 0 (peso 2^1)
18:2= 9	resto 0 (peso 2^2)
9:2= 4	resto 1 (peso 2^3)
4:2= 2	resto 0 (peso 2^4)
2:2= 1	resto 0 (peso 2^5)
1:2= 0	resto 1 (peso 2^6)

per cui: $72_{10}=1001000_2$

Esempio : convertire $72{,}87_{10}$ nel suo equivalente in base 2

Parte intera

72:2=36	resto 0 (peso 2^0)
36:2=18	resto 0 (peso 2^1)
18:2= 9	resto 0 (peso 2^2)
9:2= 4	resto 1 (peso 2^3)
4:2= 2	resto 0 (peso 2^4)
2:2= 1	resto 0 (peso 2^5)
1:2= 0	resto 1 (peso 2^6)

Parte frazionaria

0.87x2=0.74 parte intera 1 (peso 2^{-1})
0.74x2=0.48 parte intera 1 (peso 2^{-2})
0.48x2=0.96 parte intera 0 (peso 2^{-3})
0.96x2=0,92 parte intera 1 (peso 2^{-4})
0.92x2=0.84 parte intera 1 (peso 2^{-5})
0.84x2=0.68 parte intera 1 (peso 2^{-6})
0.68x2=0.36 parte intera 1 (peso 2^{-7})
0.36x2=0.72 parte intera 0 (peso 2^{-8})

... troncamento alla ottava cifra dopo la virgola

per cui: $72{,}87_{10}=1001000{,}11011110_2$

Convertire un numero Binario in Ottale/Esadecimale

Dato un numero binario, per convertirlo in ottale basta raggruppare le cifre a tre a tre, partendo da destra e calcolare il valore della singola terna.
Es: 11001010 → *11 001 010* → *3 1 2*

Dato un numero binario, per convertirlo in esadecimale basta raggruppare le cifre a quattro a quattro, partendo da destra e calcolare il valore della singola quaterna (nibble).
Es: 11001010 → *1100 1010* → *C A*

ES. 11111111 → *1111 1111* → *F F*

ES 11100100101010 → *11 1001 0010 1010* → *3 9 2 A*

Conversione da Binario a Esadecimale

$$110101_2 \rightarrow ?_{16}$$
$$101011_2 \rightarrow ?_{16}$$
$$100111100000_2 \rightarrow ?_{16}$$
$$11110100010_2 \rightarrow ?_{16}$$

(35_{16})
$(2B_{16})$
$(9E0_{16})$
$(7A2_{16})$

Conversione da Esadecimale a Binario

$0x5C \rightarrow ?_2$
$0xC17 \rightarrow ?_2$
$0x141 \rightarrow ?_2$
$0xAB0C \rightarrow ?_2$

Conversione da Esadecimale a Binario

$0x5C \rightarrow ?_2$ (1011100_2)
$0xC17 \rightarrow ?_2$ (110000010111_2)
$0x141 \rightarrow ?_2$ (101000001_2)
$0xAB0C \rightarrow ?_2$ (1010101100001100_2)

Operazioni sui numeri Binari

Cifre $C_i \in$ 0, 1. Binary digit=Bit=informazione elementare

Numero$_2$: $$N_2 = \sum_{k=0}^{k=n-1} C_i \times 2^k$$ sequenza ordinata di bits

I numeri binari sono convenzionalmente suddivisi in gruppi di 4 bits

Con 4 bits (nibble) si conta da 0 ad 1111 (da 0 a 15 in decimale)

Con 8 bits (byte) si conta da 0 ad 11111111 (da 0 a 255 in decimale)

Con 16 bits (word) si conta da 0 a 1111111111111111
(da 0 a 65535 in decimale)

$N = 2^n - 1$ max. numero decimale rappresentabile con n bit

E' possibile sviluppare una matematica binaria e definire regole di calcolo in binario in analogia al sistema decimale

Operazioni sui numeri binari

- La codifica in binario dei numeri naturali permette di utilizzare operazioni `bit per bit' per costruire operazioni su sequenze quali la somma
- Operazione di somma su un bit
  - → 0 + 0 = 0 0 + 1 = 1 1 + 0 = 1
- Se si lavora su un solo bit 1 + 1 genera un errore di *overflow* (2 non è rappresentabile su un bit)
- Su più bit allora 1+1 = 0 e genera un riporto di 1
- Si utilizza la somma bit per bit propagando il riporto (come nei decimali)

Binario	Decimale
0 1 1 0 1 +	**13 +**
0 1 0 0 1 =	**9 =**
1 0 1 1 0	**22**

Si applicano regole formalmente analoghe al sistema decimale

a) somma di due numeri binari:

Esempio: $101_2 + 111_2 = ?$

```
 1 1       riporto
  1 0 1+
  1 1 1=
 -------
1 1 0 0
```

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ con riporto 1

b) sottrazione di due numeri binari:

Esempio: $1001_2 - 111_2 = ?$

```
 1 1       prestito
1 0 0 1 –
0 1 1 1=
--------
0 0 1 0
```

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$ con prestito 1

Somma di due numeri binari

$100101_2 + 101_2 = ?_2$

$11100011_2 + 1101101_2 = ?_2$

$101_2 + 101110101_2 = ?2$

$100100110_2 + 101110101_2 = ?_2$

Somma di due numeri binari

$100101_2 + 101_2 = ?_2$ (${101010}_2$

$11100011_2 + 1101101_2 = ?_2$ (${101010000}_2$

$101_2 + 101110101_2 = ?2$ (${101111010}_2$

$100100110_2 + 101110101_2 = ?_2$ (${1010011011}_2$

Sottrazione di due numeri binari

11101 – 11001

11011101 – 1110101

11000001 – 11011

100000000 – 11

Sottrazione di due numeri binari

11101 – 11001	[R. 100]
11011101 – 1110101	[R. 1101000]
11000001 – 11011	[R. 10100110]
100000000 – 11	[R. 11111101]

c) moltiplicazione di due numeri binari:

Esempio: $101_2 \times 10_2 = ?$

$$\begin{array}{r} 1\,0\,1 \times \\ 1\,0 = \\ \hline 0\,0\,0 \\ 1\,0\,1 \\ \hline 1\,0\,1\,0 \end{array}$$

$$\boxed{\begin{array}{l} 0 \times 0 = 0 \\ 1 \times 0 = 0 \\ 0 \times 1 = 0 \\ 1 \times 1 = 1 \end{array}}$$

d) La divisione di due numeri binari non e' immediata e si basa sul seguente procedimento:

• si pone il divisore sotto il dividendo in modo che le cifre piu' significative coincidano;

• si confronta il divisore con la porzione del dividendo equivalente;

• se questa porzione e' maggiore o uguale al dividendo, si scrive un 1 nel quoziente e il divisore e' sottratto alla porzione del dividendo, altrimenti si pone uno zero nel quoziente;

• si sposta il divisore di una posizione verso destra e si ripete la procedura finche' la cifra meno significativa del divisore e' allineata con la cifra meno significativa del dividendo.

Esempio: $1001_2 : 11_2 = ?$

A	*Dividendo*	*1 0 0 1*	0 *1 1* *Quoziente*
	Divisore	*1 1*	
B	*Dividendo*	*1 0 0 1*	
	Divisore	*1 1*	
	Sottrazione	*0 1*	
C	*Dividendo*	*0 1 1*	
	Divisore	*1 1*	
	Sottrazione	*0*	*Resto*

La moltiplicazione (divisione) è riconducibile ad una semplice operazione di scorrimento o shift a sinistra (destra) di m posizioni del moltiplicando (dividendo) nel caso in cui il valore del moltiplicatore (divisore) coincida con una potenza intera m della base (2).

Esempi:

$101_2 \times 10_2 = 1010_2$ scorrimento a sinistra di una posizione

$1110_2 \times 100_2 = 111000_2$ scorrimento a sinistra di due posizioni

$101010_2 : 10_2 = 10101_2$ scorrimento a destra di una posizione

$1011_2 : 1000_2 = 1.011_2$ scorrimento a destra di tre posizioni

Moltiplicazione di due numeri Binari

$$
\begin{array}{rl}
10011001_2 \times & (153_{10}) \\
1011_2 = & (11_{10}) \\
\hline
10011001_2 + & \\
10011001_2 + & \\
00000000_2 + & \\
10011001_2 = & \\
\hline
11010010011_2 & (1683_{10})
\end{array}
$$

Numeri razionali

- Uno dei limiti nella precisione di calcolo dei computer è che è disponibile un numero limitato di bit per rappresentare I numeri (multipli di 8 bit).
- Con un numero finito di cifre è possibile rappresentare solo un numero razionale che approssima con un certo errore un numero reale dato.
- Vengono usate due notazioni:
  - ➔Virgola fissa: si fissa il numero di cifre della parte intera e di quella decimale
  - ➔Virgola mobile: si basa su una rappresentazione esponenziale

Virgola fissa

- Fissiamo quante cifre intere e quante decimali vogliamo rappresentare ed utilizziamo. La posizione della virgola è fissata una volta per tutte.
  - ➔xxxx,yyy!
- Ad esempio: se la cifra piu' a destra rappresenta ½ ($=2^{-1}$):
  - ➔10001 rappresenterà 8.5 = 8 + ½
  - ➔cioe' va letto come: 1000.1
- E' possibile rappresentare solo valori divisibili per potenze negative di 2 (gli altri valori verranno approssimati).
- Non sono rappresentati bene I numeri frazionari troppo grandi o troppo piccoli.

Virgola mobile

- Se vogliamo rappresentare sia numeri molto piccoli che numeri molto grandi occorre utilizzare una rappresentazione in cui la posizione della virgola decimale varia a seconda del numero
- Si usa una rappresentazione del tipo:
  - ➔Valore= $2^{+/-\ \text{Esponente}}$+Mantissa
  - ➔La mantissa viene normalizzata per ottenere una rappresentazione unica (varia tra 1 e ½).
- Cioe' fissata la base dobbiamo memorizzare su K bit le informazioni su: Segno Esponente Mantissa

Standard IEEE

- *Precisione singola* su 32 bit
  - ➔1 bit di segno
  - ➔8 di esponente (da -126 a +127)
  - ➔23 di mantissa
  - ➔Si possono rappresentare valori fino a 2 elevato a (-150)
- *Precisione doppia* su 64 bit
  - ➔1 bit di segno
  - ➔11 di esponente (da -1022 a +1023)
  - ➔52 di mantissa
  - ➔Si possono rappresentare valori fino a 2 elevato a (-1075)