Rappresentazione dei numeri reali

Conversione da decimale ad altra base

Per effettuare la conversione di un numero frazionario $0,c_1c_2c_3$… da decimale ad altra base si usa il metodo delle moltiplicazioni successive, simile a quello delle divisioni successive visto per la conversione degli interi.

Per convertire un numero da decimale a base b:

- moltiplicare il numero per b;
- togliere la parte intera, che è la prima cifra;
- moltiplicare quello che rimane per b;
- …

Conversione da decimale ad altra base

Per esempio, i numeri decimale 0,6875 si converte in binario come segue:

•0,6875×2 =	**1**,375	la parte intera è uno; sottratta, lascia 0,375
•0,375×2 =	**0**,75	la parte intera è zero;
•0,75×2 =	**1**,5	la parte intera è uno; si toglie, lasciando 0,5
•0,5×2 =	**1**	la parte intera è uno; togliendola, rimane 0

Le cifre binarie sono le parti intere in ordine (non in ordine opposto come nella conversione della parte non frazionaria). In questo caso: 0,1011. Quando il valore che rimane è zero, la procedura termina

Conversione da decimale ad altra base

Non è però sempre detto che la procedura termini. Per esempio, la conversione di 1/5 ovvero $(0,2)_{10}$ produce:

0,2×2=**0**,4

0,4×2=**0**,8

0,8×2=**1**,6   parte intera 1

0,6×2=**1**,2   parte intera 1

0,2×2=0,4

…

Si è tornati a 0,2, per cui si ripete tutto uguale, tornando ancora a 0,2. Il numero binario che si ottiene è quindi 0,0011 0011 0011…, dove le cifre 0011 si ripetono all'infinito

Conversione da decimale ad altra base

Questo avviene perché 10 è divisibile per il numero primo 5 e la rappresentazione in base dieci di 1/5 è finita: 1/5=**$2/10=2*10^{-1}$**. D'altra parte, 2 non è divisibile per il numero primo 5, per cui la rappresentazione di 1/5 in base due non è finita: non c'è modo di ottenere 1/5 come somma delle potenze del due 1/2, 1/4, 1/8, ecc.

Questa proprietà vale in generale: se la base di arrivo è divisibile per tutti i numeri primi di quella di partenza allora un numero frazionario con cifre finite si converte sempre in uno con cifre finite. Nel caso contrario, questo può o meno avvenire a seconda dei casi: 1,375 si converte in un numero binario con cifre finite, mentre 0,2 no.

Nota: Un numero frazionario con infinite cifre in base 10 può averne un numero finito in un'altra

$1/3 = (0.3333333....)_{10}$ mentre $1/3 = (3/9)=3*9^{-1}= (0.3)_9$

Rappresentazione di numeri reali

- Con un numero finito di cifre è solo possibile rappresentare un numero razionale *che approssima con un certo errore* il numero reale dato
- Vengono usate due notazioni:

A) <u>Notazione in virgola fissa</u>

Dedica parte delle cifre alla parte intera e le altre alla parte frazionaria

± XXX .YY

B) <u>Notazione in virgola mobile</u>

Dedica alcune cifre a rappresentare un esponente della base che indica l’ordine di grandezza del numero rappresentato

Notazione in virgola mobile

- A parità di cifre estende l'intervallo di numeri rappresentati, rispetto alla notazione in *virgola fissa*
- Numeri reali rappresentati da una coppia di numeri **<m,e>**

  m : mantissa normalizzata tra due potenze successive della base

  $$b^{i-1} \leq |m| < b^{i}$$

  esempio in base 10

  $3.7\ 10^{6}$

  – **e** : esponente intero con segno

  $$n = m \cdot b^{e}$$

  esempio in base 2

  $1.01\ 2^{6}$

- Sia **m** che **e** hanno un numero prefissato di cifre

  Intervalli limitati ed errori di arrotondamento

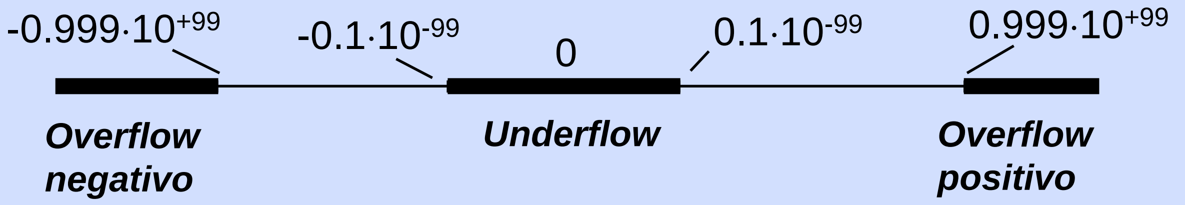
Esempio in base 10

- Numerali a **5** cifre ± **.XXX** ± **EE**
- *Mantissa* : **3** cifre con segno

$$0.1 \leq |m| < 1$$

- *Esponente*: 2 cifre con segno

$$-99 \leq e \leq +99$$



Con le stesse **5** cifre in notazione a virgola fissa ± **XXX .YY** :

- L'intervallo scende **[-999.99,+999.99]**
- Ma si hanno **5** cifre significative invece di **3**

Standard IEEE 754 (1985)

- Formato non proprietario cioè indipendente dall'architettura
- Semplice precisione a 32 bit: (float del C)

1	8	23
SEGNO	ESPONENTE	MANTISSA

errore relativo max $=2^{-23} \sim 10^{-6.9}$

- Doppia precisione a 64 bit: (double del C)

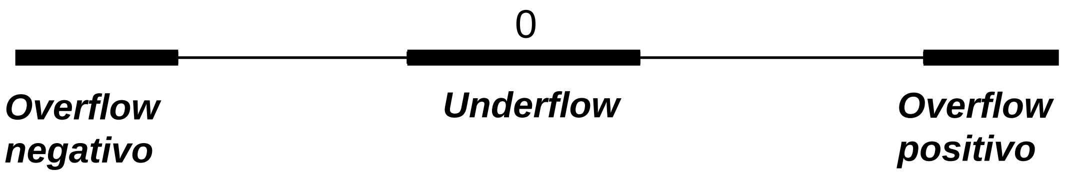
1	11	52
SEGNO	ESPONENTE	MANTISSA

errore relativo max $= 2^{-52} \sim 10^{-15.6}$

- Notazioni in modulo e segno
- Alcune configurazioni dell'esponente sono riservate

Standard IEEE 754 (1985)

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	–126 to +127	–1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}



IEEE 754 a 32 bit

1	8	23
SEGNO	ESPONENTE	MANTISSA

- **ESPONENTE**
  - Rappresentato in *eccesso 127*
  - L'intervallo rappresentabile è **[-127, +128]**
  - Le due configurazioni estreme non si usano, quindi:

$$-126 \leq e \leq 127$$

- **MANTISSA**
  - *Normalizzata* (1.xxxxxxxxxxxxxxxxxxxxxxx)
  - *Denormalizzata* (0.xxxxxxxxxxxxxxxxxxxxxxx)
  - Se ne rappresenta <u>*solo*</u> *la parte frazionaria*

Due rappresentazioni, *a seconda del valore dell'esponente*:

A) Numeri normalizzati $e \neq 00000000 = (-127)_{10}$

B) Numeri denormalizzati $e = 00000000$

Numeri normalizzati

- Un numerale si intende in questa rappresentazione quando:

$$\overset{-127}{e \neq 00000000}, \quad \overset{+128}{e \neq 11111111} \quad -126 \leq e \leq 127$$

- La mantissa è normalizzata tra **1** e **2**: **00000001** **11111110**

$$1 \leq m < 2$$

- Quindi è sempre nella forma:

1.XXXXXXXXXXXXXXXXXXXXXXX

- I **23** bit rappresentano la sola parte frazionaria
- Gli intervalli di numeri rappresentati sono pertanto:

$$(-2^{128},-2^{-126}] \quad [2^{-126},2^{128})$$

- Gli estremi sono esclusi perché il massimo valore assoluto di **m** è molto vicino a 2 ma comunque inferiore
- L'intervallo $(-2^{-126}, 2^{-126})$ è *intervallo di underflow*

Numeri denormalizzati

- Un numerale si intende in questa rappresentazione quando:

  e = 00000000   -127

- L'esponente assume il valore <u>*convenzionale*</u> **e=-126**
- La mantissa è compresa tra **0** e **1**:

  $$0 < m < 1$$

- Quindi è sempre nella forma:

  0.XXXXXXXXXXXXXXXXXXXXXXX

- I **23** bit rappresentano la sola parte frazionaria
- La più piccola mantissa vale $\mathbf{2^{-23}}$ **(0.00000000000000000000001)**
- Gli intervalli rappresentati sono:

  $$(-2^{-126},-2^{-149}] \quad [2^{-149},2^{-126})$$

Più piccola è la mantissa minore è il numero di cifre significative

Altre configurazioni

- Lo Standard IEEE 754 attribuisce valori convenzionali a particolari configurazioni di **e** ed **m**

  A) **e** ed **m** tutti **0** rappresentano il valore **0** (*altrimenti non rappresentabile*)

  B) **m** tutti **0** ed **e** tutti **1** rappresentano l'*overflow*

  C) **m** $\neq$ **0** ed **e** tutti **1** indicano la situazione *Not A Number (NAN)*, cioè un valore indefinito (ad es. il risultato di una divisione per **0**)

Queste convezioni sono una caratteristica peculiare della notazione IEEE 754; non valgono, se non esplicitamente definite, per altre notazioni

(soprattutto nei compiti d' esame)

IEEE 754: estremi degli intervalli

- Modulo *più grande normalizzato* **$\sim 2^{128}$** :

X 11111110 11111111111111111111111

± 2^{127} ~ 2

- *Modulo più piccolo normalizzato* **2^{-126}** :

X 00000001 00000000000000000000000

± 2^{-126} 1

- *Modulo più grande denormalizzato* **$\sim 2^{-126}$** :

X 00000000 11111111111111111111111

± 2^{-126} $(0.11\ldots)_2 \sim 1$

- *Modulo più piccolo denormalizzato* **2^{-149}** :

X 00000000 00000000000000000000001

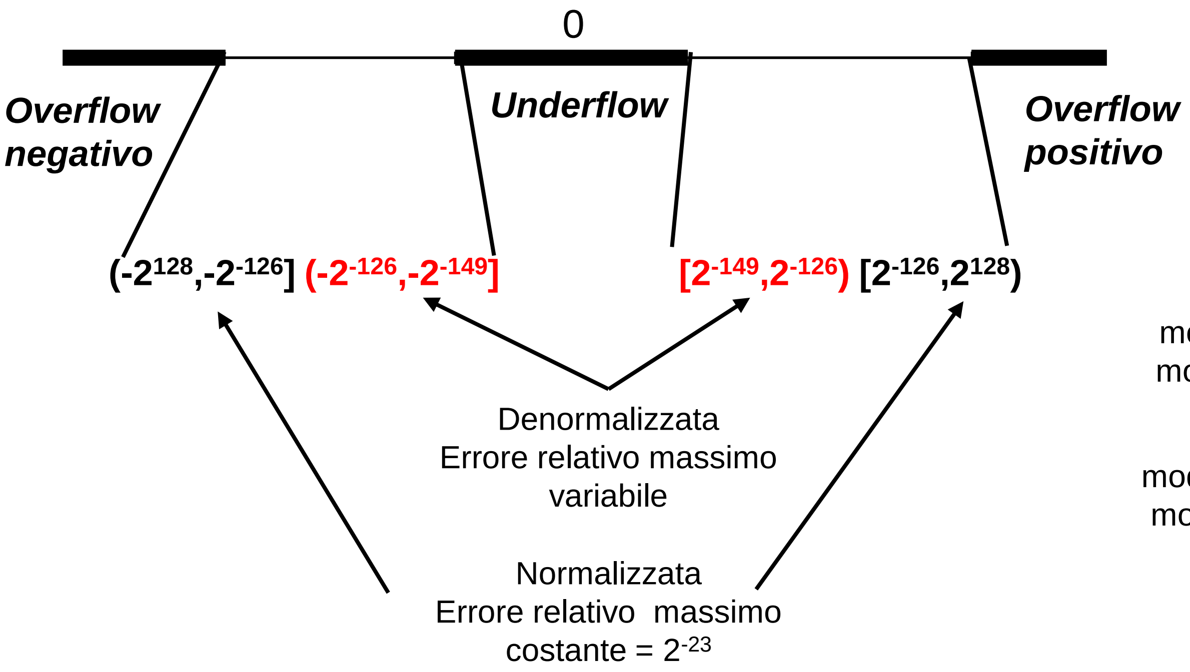
± 2^{-126} $(0.00\ldots1)_2 = 2^{-23}$

Esponente
eccesso
127
(aumenta il modulo massimo)

[-127, +128]
0000000, 1111111
usati
[-126, +127]
0000001,1111110

Ma 0000000 significa, convenzionalmente, e=-126 e mantissa denormalizzata ...

Quindi



modulo più grande normalizzato: 2^{128}
modulo più piccolo normalizzato: 2^{-126}

modulo più grande denormalizzato: $\sim 2^{-126}$
modulo più piccolo denormalizzato: 2^{-149}

Idea dello standard

- Rappresentazione numeri reali in virgola mobile

•esempio: $+\ 1.101\ 2^{2}$ 110100 52 in base 10

•segno (0 positivo 1 negativo), esponente in eccesso e mantissa in binario puro

cercando di estendere gli intervalli rappresentati e minimizzare gli errori

•il tutto avendo a disposizione un numero finito di bit...

Esempio giocattolo : 8 bit (small float ☺)

- 1 bit per il segno (0 + , 1 -)
- 3 bit per l'esponente in eccesso 2^2
- 4 bit per la mantissa (normalizzata, ovvero 1.XXXX)

X XXX XXXX

- intervallo dell'esponente ? [-4,+3]
- numero positivo più piccolo ? numero positivo più grande ?
- numero negativo più piccolo ? numero negativo più grande ?

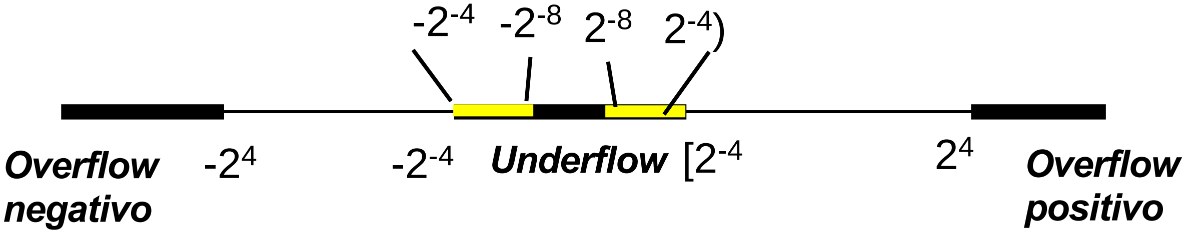
Esempio giocattolo : 8 bit (small float ☺)

- intervallo dell'esponente [-4, 3]
- numero positivo più piccolo $1.0000\ 2^{-4}$ 2^{-4} 0 000 0000
- numero positivo più grande $1.1111\ 2^{3} \sim 2^{4}$ 0 111 1111
- numero negativo più piccolo $-1.1111\ 2^{3} \sim -2^{4}$ 1 111 1111
- numero negativo più grande $-1.0000\ 2^{-4}$ -2^{-4} 1 000 0000

ridurre l'overflow richiede di aumentare il numero di bit
come faccio, invece, a ridurre l'intervallo di underflow?

Esempio giocattolo : 8 bit (small float ☺)

- denormalizzando la mantissa 0.XXXX
- e bloccando l'esponente al suo valore minimo -4
- numero positivo più piccolo $0.0001\ 2^{-4}\ \ 2^{-4} = 2^{-8}$ 0 000 0001
- numero positivo più grande $0.1111 \sim 1\ 2^{-4} \sim 2^{-4}$ 0 000 1111
- numero negativo più grande $-0.0001\ 2^{-4}\ \ 2^{-4} = -2^{-8}$ 1 000 0001
- numero negativo più piccolo $-0.1111 \sim 1\ 2^{-4} \sim -2^{-4}$ 1 000 1111



Errore relativo

- 0.0000000001 0.001
- 1000000000
- 0.0000000000000000000000001
- 0.0000000000000000000000011
- 1.xxxxxxxxxxxxxxxxxxxxxxxx m bit 2^{-m}
- 0.0001 111111111111
- 1.0011 2^{-4}

Ex 1: virgola mobile

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 8 bit per l'esponente, in eccesso 128
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2

1. Calcolare gli estremi degli intervalli rappresentati, i numerali corrispondenti, e l'ordine di grandezza decimale

Esempio 1: virgola mobile

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)   00000000 1111111
  - 8 bit per l'esponente, in eccesso 128 → [-128,+127]
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2

1. Calcolare gli estremi degli intervalli rappresentati, i numerali corrispondenti, e l’ordine di grandezza decimale.

Modulo + grande = 1.1111111 $*2^{127} \sim \mathbf{2^{128}}$

Modulo + piccolo = 1.0000000 $*2^{-128} = \mathbf{2^{-128}}$

Intervalli $(-2^{128},-2^{-128}]$ $[2^{-128}, 2^{128})$ $[10^{-38.4}, 10^{38.4}]$

(1 11111111 1111111, 1 00000000 0000000] [0 00000000 0000000, 0 11111111 1111111)

Ex 1: virgola mobile

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 8 bit per l'esponente, in eccesso 128
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2

1.

2. Rappresentare in tale notazione il numero *n* rappresentato in complemento a 2 dai tre byte FF5AB9.

Esempio 1: virgola mobile

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 8 bit per l'esponente, in eccesso 128
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2

2. **Rappresentare in tale notazione il numero *n* rappresentato in complemento a 2 dai tre byte F F 5 A B 9**

- Ovvero 1111 1111 0101 1010 1011 1001
- Negativo, di modulo 0000 0000 1010 0101 0100 0111
- n= $-1.0100101010000111\ 2^{15} \rightarrow 1.0100101\ \cancel{0100\ 0111}\ 2^{15}$
- **1 10001111 0100101**

Addizioni in virgola mobile

Per addizionare e sottrarre occorre portare i numeri allo stesso esponente (quello + grande) e scalare le mantisse

ESEMPIO: $n_3 = n_1 + n_2$ eccesso 128

n_1 : 0 10011010 00010111011100101100111

n_2 : 0 10101011 11001100111000111000100

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$: → e_1 deve diventare 43=26+17

occorre scalare m_1 1.00010111011100101100111 *di 17 posti*

⇩

m'_1 : 0.00000000000000001000101 +

m_2 : 1.11001100111000111000100

m_3 : 1.11001100111001000001001

n_3 : 0 10101011 11001100111000111000100

Notare che l'addendo più piccolo perde cifre significative

Moltiplicazioni fra interi

- L'operazione viene effettuata sui numerali, come in decimale
- Il numerale che si ottiene rappresenta il risultato
- La tabellina delle moltiplicazioni però è molto più semplice che nel caso decimale **2 x 2** invece di **10 x 10**

	0	1
0	0	0
1	0	1

Come in decimale, si incolonnano e si sommano i prodotti parziali scalandoli opportunamente

Moltiplicazioni fra interi: esempio

$5 \cdot 11 = 55$

$$
\begin{array}{rr}
(11)_{10} & 1011 \;\; \times \\
(5)_{10} & 101 \;\; = \\
\hline
 & 1011 \\
 & 0000\!\! \\
 & 1011 \\
\hline
(55)_{10} & 110111
\end{array}
$$

Notare che, in base alla tabellina, ciascun prodotto parziale è pari a zero oppure al moltiplicando

Moltiplicazioni in virgola fissa

Si opera come in decimale, tenendo conto del numero di cifre frazionarie e riposizionando il punto frazionario

2.75 1.25 = 3.4375

```
(2.75)10            10.11  x
(1.25)10             1.01  =
                  ----------
                     1011
                    0000
                   1011
                  ----------
(3.4375) 10        11.0111
```

Moltiplicare o dividere per 2^n *equivale a spostare il punto di* n *posti a destra o a sinistra*

Moltiplicazioni in virgola mobile

Si moltiplicano le mantisse e si sommano algebricamente gli esponenti e, se necessario, si scala la mantissa per normalizzarla e si riaggiusta l'esponente

ESEMPIO: $n_3 = n_1 \times n_2$

n_1 : 0 10011010 10010111011100101100111 1

n_2 : 1 10101011 10000000000000000000000

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$

- $e_3 = e_1 + e_2 = (69)_{10}$ = 11000100
- $n_1 \times n_2$ = 1$\underline{0.0110001100101011101101}$01
- *si scala la mantissa di un posto*
- *si aumenta di 1 l'esponente*

n_3 : 1 11000110 00110001100101011101101

Errore assoluto e relativo

- Rappresentando un numero reale **n** in una notazione floating point si può commettere un errore di approssimazione
- In realtà viene rappresentato un numero razionale **n’** con un numero limitato di cifre significative

ERRORE ASSOLUTO: $e_A = n - n'$

ERRORE RELATIVO: $e_R = e_A / n = (n - n')/n$

- L’ordine di grandezza dell’**errore assoluto** dipende dal *numero di cifre significative* e dall’*ordine di grandezza del numero*
- L’ordine di grandezza dell’**errore relativo** dipende **solo** dal *numero* **m** *di cifre significative della mantissa*

Errore relativo massimo

Se la mantissa è normalizzata l'errore relativo è sempre inferiore a un'unità sull'ultima cifra rappresenta

- Con **k** cifre frazionarie e mantissa normalizzata tra **1** e **2** si hanno sempre **k + 1** cifre significative 1.xxxxxxxxx
- L'errore relativo **massimo** è dell'ordine di $\mathbf{2^{-k}}$: $\mathbf{e_R < 2^{-k}}$

ESEMPIO

Con 10 cifre frazionarie l'errore massimo è ordine $\mathbf{2^{-k}}$

Su un numero **n** *di ordine* $\mathbf{2^m}$ *l'errore assoluto è dato da* $\mathbf{e_A = e_R \cdot n}$ *e quindi è ordine* $\mathbf{2^{-10}\ 2^m = 2^{m-10}}$

Nelle notazioni non normalizzate il numero di cifre significative, e quindi l'errore relativo massimo, non è costante