

Tipologie di codici

Nel seguito vedremo tipologie di rappresentazioni diverse:

- Senza assumere limitazioni sul numero di bit a disposizione:
per numeri [notazione binaria, ovvero posizionale con base 2]
- Disponendo di un numero di bit limitato:
 - numeri naturali
 - interi relativi [valore assoluto e segno, complemento a due]
 - “reali” [virgola fissa e virgola mobile]
 - valori logici, caratteri alfabetici, testi
 - suoni, immagini e sequenze video
 - codici per la rilevazione e correzione di errori
- Codici di compressione (senza | con perdita)

Rappresentazione dei numeri reali

- Per rappresentare numeri come


- $1/3 = 0.333333333333....$

- $\pi = 3.141592653...$

- $\sqrt{2} = 1.4142135...$

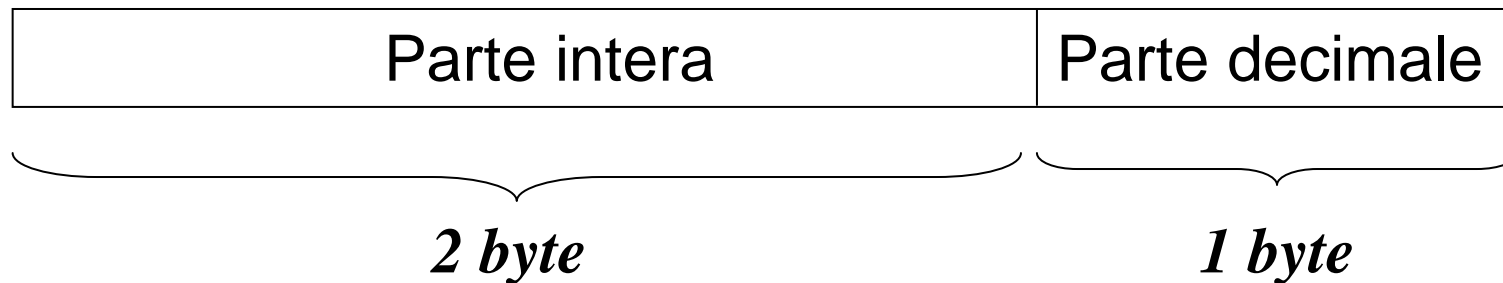
servirebbe un **numero di cifre illimitato**

- Nel calcolatore è possibile usare solo successioni di bit di lunghezza finita
- Necessaria **approssimazione**

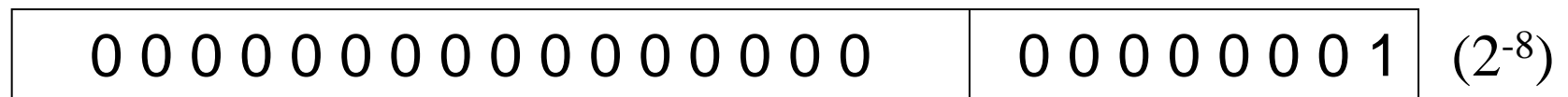
 2 modalità {
Rappresentazione in virgola fissa
Rappresentazione in virgola mobile

Rappresentazione in virgola fissa

Esempio (in base 2): numeri di 3 byte



Il numero più piccolo rappresentabile è:



La “granularità” è fissa e pari a 2^{-8} :

per un qualsiasi numero rappresentabile I , il successivo è $I + 2^{-8}$

Inconveniente della rappresentazione in virgola fissa

- Peso molecola CO₂: 2^{-80} grammi
00000000000000000000.0000000000 ... 0000000000000001
- Peso Ferrari: 600 kg = 2^{19} grammi
10000000000000000000.0000000000 ... 0000000000000000
- Potrei rappresentare tutti i numeri con 20 cifre a sinistra della virgola e 80 cifre a destra, ma in questo modo spreco i bit disponibili!
- Si vuole un sistema di rappresentazione in cui i bit a disposizione siano sfruttati in modo più flessibile, adattando la granularità all'ordine di grandezza del numero rappresentato ed estendendo così ***l'intervallo*** dei numeri esprimibili
→ idea: ***notazione scientifica***

Notazione scientifica

- Un numero in base 10 viene rappresentato come

The diagram illustrates the components of scientific notation: $\pm m \times 10^e$. Arrows point from labels to specific parts: 'segno' points to the sign (\pm), 'mantissa' points to m , 'base' points to 10, and 'esponente' points to e . The m is circled in red, 10 is circled in light blue, and e is circled in blue.

- Esempio: 159 300 000 = $+1.593 \times 10^8$ dove $m=1.593$ ed $e=8$
- Esempio: -0.00001593 = -1.593×10^{-5} dove $m=1.593$ ed $e=-5$

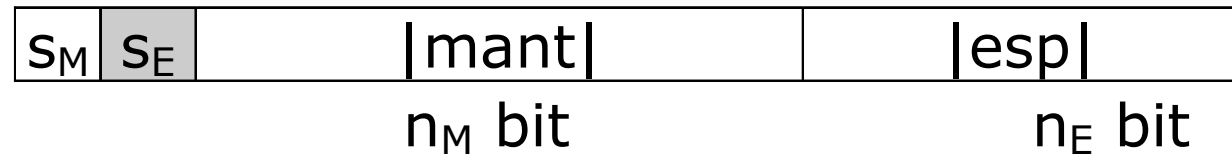
Rappresentazione in virgola mobile (floating point)

$$N = \pm \text{mant} * 2^{\text{esp}}$$

FORMATO

Basta memorizzare
segno, *mantissa* ed *esponente*

ESEMPIO DI
FORMATO



Forma normalizzata della mantissa:

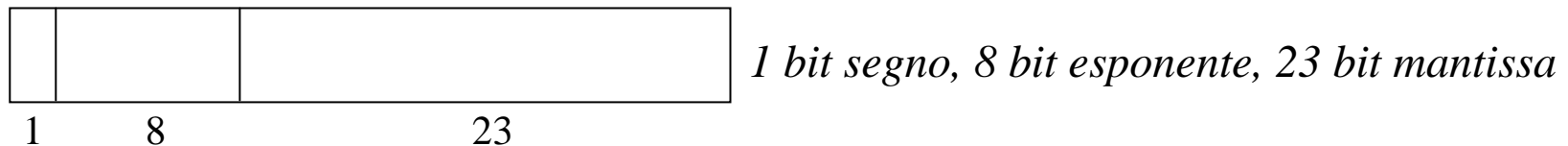
Stesso numero 0.1_2 : $1 * 2^{-1}$
 $10 * 2^{-2}$

...

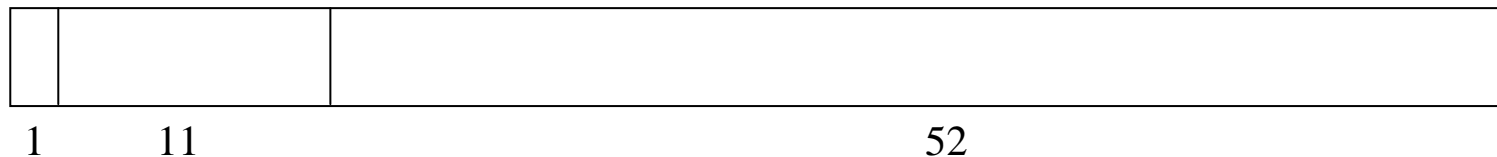
⇒ si stabilisce una forma univoca di rappresentazione

Standard IEEE 754

- Necessità di uniformare la precisione del calcolo (calcolatori di produttori diversi con strutture differenti)
- Lo standard IEEE* 754 stabilisce la lunghezza di mantissa ed esponente
- 32 bit per i numeri in precisione singola



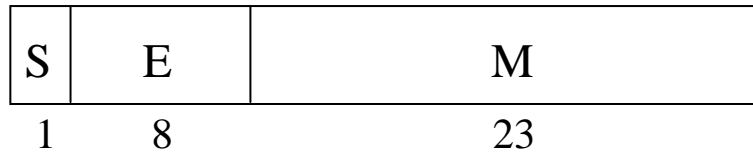
- 64 bit per i numeri in precisione doppia *1 bit segno, 11 bit esponente, 52 bit mantissa*



Vedremo solo il formato a precisione singola

*Institute of Electrical and Electronic Engineering

IEEE 754 singola precisione



Segno S: 0 segno + 1 segno -

Normalizzazione e mantissa

$$N = \underset{\substack{\downarrow \\ \text{hidden bit}}}{\cancel{1}.xxxxxxx} * 2^{esp}$$

23 bit (M)

Rappresentazione dell'esponente

$$E = esp + 127$$

Rappresentazione in 8 bit a eccesso 127,
con le configurazioni 00000000 e 11111111 non ammesse
($-126 \leq esp \leq 127$)

Esempio: num = -3.5

Segno: 1

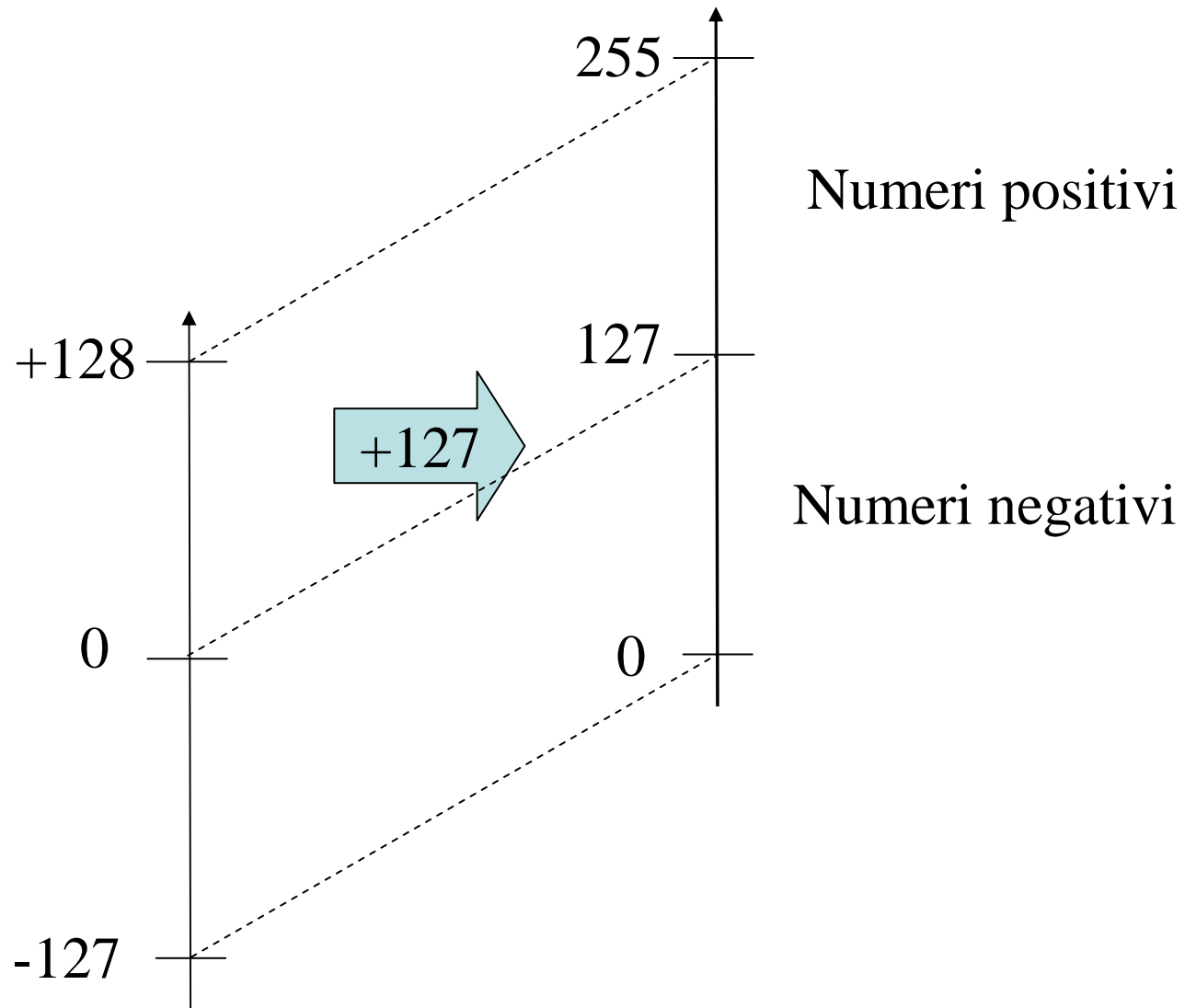
$$\begin{aligned} num &= 11.1 \\ &= 1.11 * 2^1 \\ M &= 1100...0 \end{aligned}$$

$$\begin{aligned} E &= 1 + 127 = 128 \\ &= 10000000 \end{aligned}$$



1|10000000|110000000000000000000000

Nota sulla rappresentazione dell'esponente



Esercizio (Appello del 23 set 2003)

Definire il concetto di polarizzazione dell'esponente nella codifica IEEE 754 dei numeri in virgola mobile (singola precisione). Se il campo esponente di una codifica contiene il numero 00111011 qual è il valore decimale dell'esponente? [2]

Soluzione

$E = \text{esp} + 127$ valori rappresentabili da -127 a $+128$
(NB: gli estremi non sono utilizzati propriamente)

Nel caso specifico: $E = 59$
quindi $\text{esp} = 59 - 127 = -68$

Esercizio (Appello del 7 gen 2003)

Ricavare il valore decimale del seguente numero in virgola mobile rappresentato secondo lo standard IEEE 754 a 32 bit: [3]

0 10000000 100000000000000000000000000000

Soluzione

Segno: +

Esponente: $E = 2^7 = 128$ $\text{esp} = 128 - 127 = 1$

Mantissa: $\text{mant} = 1.1$

$$\Rightarrow N = 1.1_2 * 2^1 = 11_2 = 3_{10}$$

Esercizio (Appello del 5 feb 2001)

Utilizzando la rappresentazione standard IEEE per numeri floating point su 32 bit, si determini il valore decimale della sequenza di bit corrispondente a 3F400000 in base 16. [2]

Soluzione

Notazione binaria:

0011 1111 0100 0000 0000 0000 0000 0000

segno: +

esponente: $E = 01111110_2 = 126$

$esp = 126 - 127 = -1$

$\Rightarrow N = 1.1 * 2^{-1} = 0.11_2 = 2^{-1} + 2^{-2} = 0.5 + 0.25 = 0.75$

Esercizio

Rappresentare il numero decimale -4.5 secondo lo standard in virgola mobile IEEE 754 a 32 bit.

Soluzione

Segno: 1

Rapp. binaria: $4.5_{10} = 100.1_2$

Forma normalizzata: $N = 1.001 * 2^2$

Esponente: $esp = 2 \Rightarrow E = 2 + 127 = 129_{10} =$
 10000001

\Rightarrow IEEE754: 1 10000001 0010.....0

PASSAGGIO IMPORTANTE: normalizzazione

ES. 1

Rappresentazione binaria:

$$\begin{aligned} 1001.01001 &= \\ 1.00101001 * 2^3 &\quad (\text{forma normalizzata}) \end{aligned}$$

ES. 2

Rappresentazione binaria:

$$\begin{aligned} 0.001011 &= \\ 1.011 * 2^{-3} &\quad (\text{forma normalizzata}) \end{aligned}$$

PER ESERCITARSI...

<http://www.h-schmidt.net/FloatApplet/IEEE754.html>

Convertitore (on line) tra:

- IEEE754
- Binario
- Decimale
- Esadecimale

E' sufficiente inserire un valore in una delle precedenti rappresentazioni per ottenere i valori delle altre tre.

(vedi anche il sito del corso...)

L'insieme dei numeri in virgola mobile non coincide con \mathbb{R}

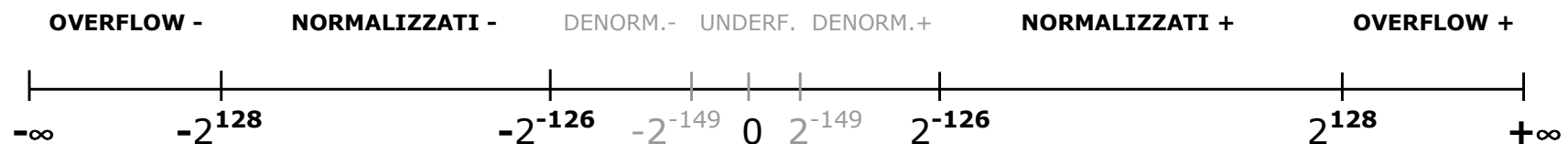
- L'insieme dei numeri reali è ***denso e illimitato***
 - L'insieme dei numeri in virgola mobile ***non è denso*** ed esistono un numero reale ***massimo*** ed uno ***minimo*** esprimibili
- ⇒ L'aritmetica “reale” del calcolatore è diversa da quella classica

Esempio: IEEE 754 con singola precisione

MinExp = -126 MaxExp = +127 [estremi per scopi speciali]

Numero più grande normalizzato: $\pm 1.11...1_2 * 2^{127} \approx 2 * 2^{127} = 2^{128}$

Numero più piccolo normalizzato: $\pm 1.00...0_2 * 2^{-126} = 2^{-126}$



NB: nel caso della virgola fissa, l'intervallo dei valori rappresentabili sarebbe molto più limitato. Es:

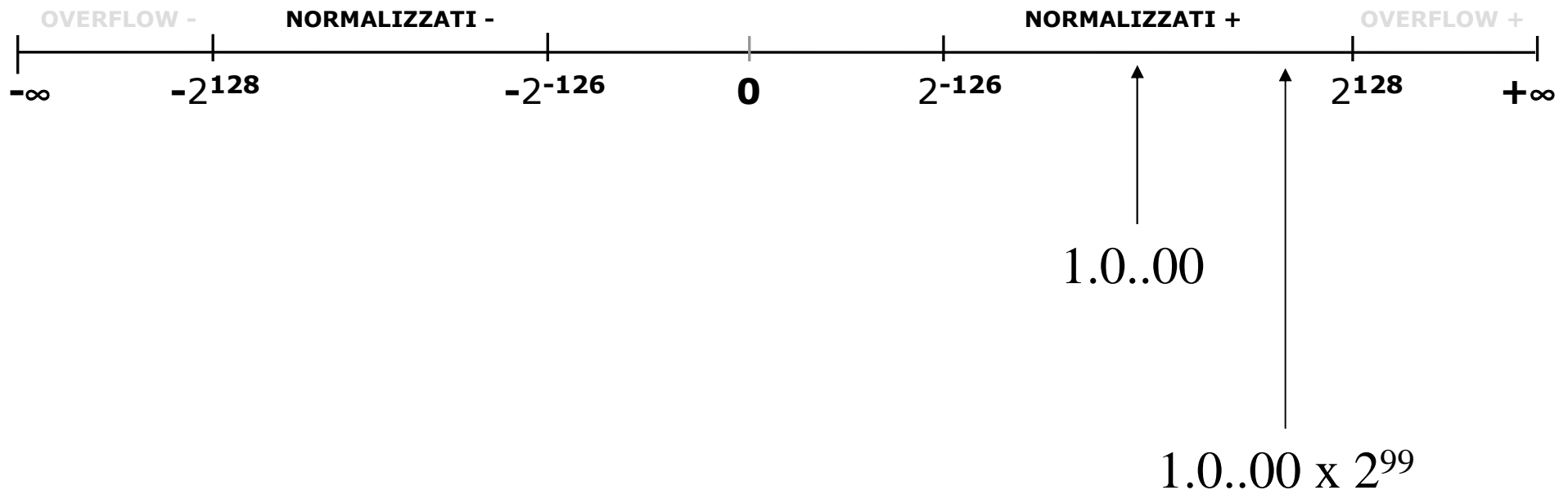
- anche se dedicassimo tutti i 32 bit alla parte intera, max 2^{32}
- anche se dedicassimo tutti i 32 bit alla parte fraz, min 2^{-32}

I problemi che nascono dalla natura digitale del calcolatore

- I numeri nelle regioni $(-\infty, -2^{128})$, $(-2^{126}, 0)$, $(0, 2^{126})$, $(2^{128}, +\infty)$ non possono essere rappresentati
- Determiniamo quanti numeri possono essere rappresentati:
 - la mantissa varia da $m = 1.0..0$ a $1.1..1$ (2^{23} numeri)
 - l'esponente varia da $e = -126$ a $+127$ (254 ordini di grandezza)
 - in totale: $2^{23} \times 254$ numeri positivi, $2^{23} \times 254$ numeri negativi, e lo zero $\rightarrow 4.261.412.865$ numeri
- I due intervalli di numeri positivi e negativi esprimibili non formano insiemi continui: i numeri ***non sono cioè uniformemente distribuiti*** (più radi per valori elevati dell'esponente e più fitti per valori piccoli dell'esponente - si infittiscono nei pressi dello zero)

Esempio

la separazione fra $1.0..00 \times 2^{99}$ e $1.0..01 \times 2^{99}$ (ovvero, $2^{-23+99} = 2^{76}$) è molto maggiore di quella fra $1.0..00 \times 2^0$ e $1.0..01 \times 2^0$ (ovvero, 2^{-23})



➡ La precisione è “concentrata” dove ce n’è bisogno!

Operazioni sui numeri “reali”

- **Addizione e sottrazione:**
 - si trasformano (con eventuale perdita di precisione) gli addendi in una rappresentazione con uguale esponente (il maggiore – ovvero trasformo il minore esponente nel maggiore dividendo la mantissa e perdendo così le cifre “meno significative”)
 - si sommano (sottraggono) le mantisse
 - si normalizza se necessario
- **Moltiplicazione:**
 - si sommano gli esponenti
 - si moltiplicano le mantisse
 - si normalizza se necessario
- **Divisione:**
 - si sottraggono gli esponenti
 - si dividono le mantisse
 - si normalizza se necessario

Esempio

Somma tra i seguenti due numeri rappresentati IEEE 754:

$$0 \underbrace{01111011}_{123 \text{ } (-127 = -4)} 000...111 \quad \text{e} \quad 0 \underbrace{01111101}_{125 \text{ } (-127 = -2)} 000...011$$

Porto l'esponente del primo operando a -2:

$$1.000...111 \times 2^{-4} = 0.010...001 \times 2^{-2}$$

$$\begin{array}{rcl} 0.010...001 & + & [\times 2^{-2}] \\ \hline 1.000...011 & = & [\times 2^{-2}] \\ \hline 1.010...100 & & [\times 2^{-2}] \end{array} \quad \text{NB: già normalizzato!}$$

Risulta quindi

$$0 \ 01111101 \ 010...100$$

NB: le cose sono leggermente più complicate, ma l'idea è questa...

Problemi con le operazioni

- Addizioni e sottrazioni possono dare luogo a **errori**
- Esempio: sottrazione fra due numeri quasi uguali
→ può dar luogo al fenomeno della **cancellazione** (risultato = 0)
- Divisione per numeri molto piccoli
→ il risultato può cadere nell'intervallo di **overflow** (+ o -)

Fenomeno della cancellazione: esempio

Se ho la sottrazione

$$\begin{array}{r} 1.11100\dots0 \times 2^{-126} - \\ 1.11000\dots0 \times 2^{-126} = \end{array}$$

$$0.00100\dots0 \times 2^{-126}$$

→ che normalizzato diventerebbe

1.0×2^{-129} e viene dunque approssimato con 0
poiché il minimo esponente esprimibile è -126

NB: in realtà, è possibile esprimere numeri “denormalizzati”
prossimi allo 0, fino a 2^{-149} , ma noi non ce ne occupiamo...