Codifica binaria:
- numeri razionali -

Ingegneria Meccanica e dei Materiali

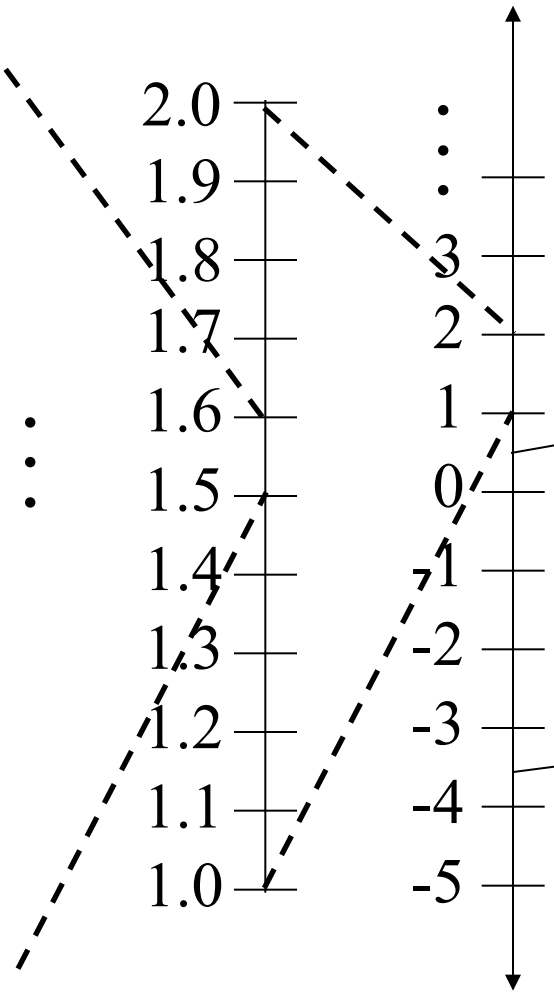
Università degli Studi di Brescia

Prof. Massimiliano Giacomin

Tipologie di codici

- Per la rappresentazione di:
  - caratteri alfabetici e testi
  - valori logici
  - numeri naturali
  - numeri interi relativi [val assoluto e segno, complemento a due]
  - numeri "reali" [virgola fissa e virgola mobile]
  - suoni, immagini e sequenze video
- Codici per la rilevazione e correzione di errori
- Codici di compressione (senza | con perdita)

Codifica di numeri "reali": il problema



n bit: 2^n combinazioni

QUALI?

00000000

00000001

00000010

⋮

11111110

11111111

Modalità di rappresentazione

2 modalità
- Rappresentazione in virgola fissa (non usata nei calcolatori)
- Rappresentazione in virgola mobile

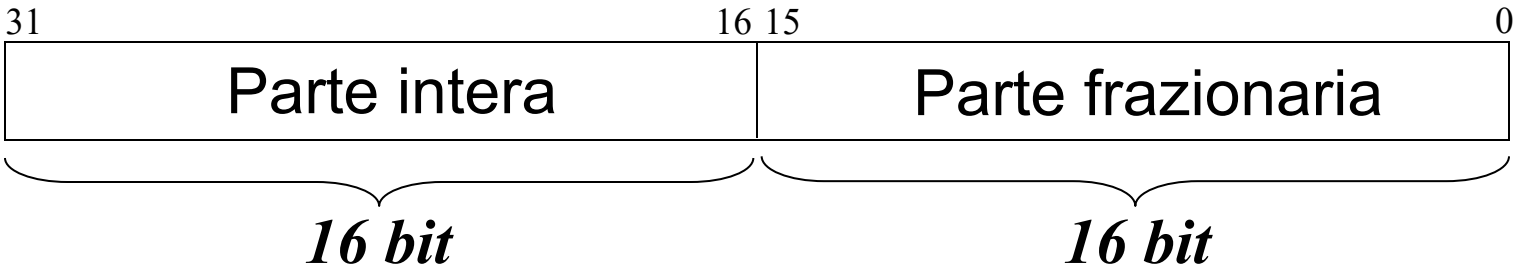
In entrambi i casi rappresentazione definita mediante un ***formato***

- il numero di bit *n* a disposizione
- i ***campi*** in cui sono suddivisi i bit:
  - quanti
  - in che ordine
  - quanto è lungo ciascun campo
  - cosa rappresenta ciascun campo

Rappresentazione in virgola fissa

- Abbiamo *n* bit a disposizione (e vogliamo rappresentare numeri reali):
  - un campo del formato rappresenta la parte intera
  - un campo del formato rappresenta la parte frazionaria

Esempio n = 32 bit



- Ad esempio 13.25 sarebbe rappresentato così:

00000000000011010100000000000000

Intervallo di rappresentazione

Nell'esempio precedente, qual è il valore massimo rappresentabile?

$$\underbrace{1111111111111111}_{\textit{16 bit}}.\underbrace{1111111111111111}_{\textit{16 bit}}$$

- Parte intera: $2^{16} - 1 = 65535$
- Parte frazionaria:

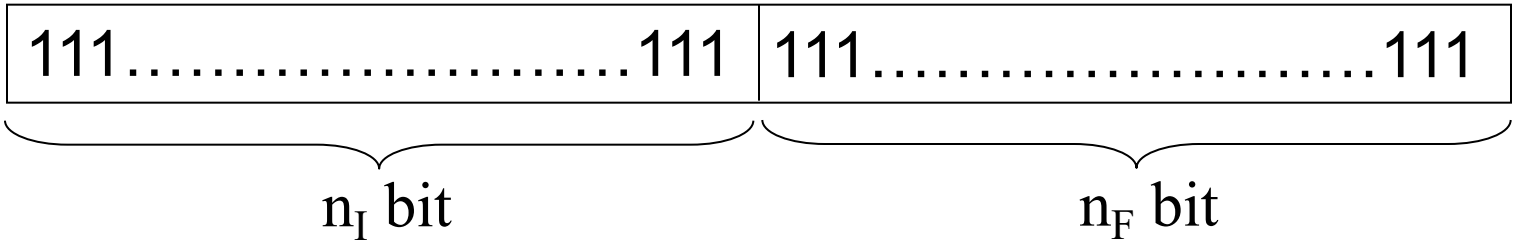
$$\begin{array}{l} 0.1111111111111111 + \\ \underline{0.0000000000000001} = \longleftarrow 2^{-16} \\ 1.0000000000000000 \end{array}$$

$$\Rightarrow \quad 1 - 2^{-16} = 0.9999847421 \quad \text{circa } 1$$

$$\Rightarrow \quad N_{MAX} = 65535.9999847421 \quad \text{circa} \quad 65536$$

Intervallo di rappresentazione in generale

- max numero rappresentabile N_{MAX}:



Max parte intera: $2^{nI}-1$

Max parte frazionaria: $1 - 2^{-nF}$

se n_F grande pari a circa 1

$N_{MAX} = (2^{nI}-1) + (1 - 2^{-nF})$

se n_F grande pari a circa 2^{nI}

Granularità

- Qual è la "precisione" di questa rappresentazione?
- ***Granularità***: differenza tra un numero e il successivo rappresentabile: quanto più è piccola, tanto più precisa è la rappresentazione!

Nell'esempio precedente

Dato un numero, rappresentato, qual è il successivo?

$$\begin{array}{r} 0000000000001101|0100000000000000 \quad + \\ 0.0000000000000001 \\ \hline 0000000000001101|0100000000000001 \end{array}$$

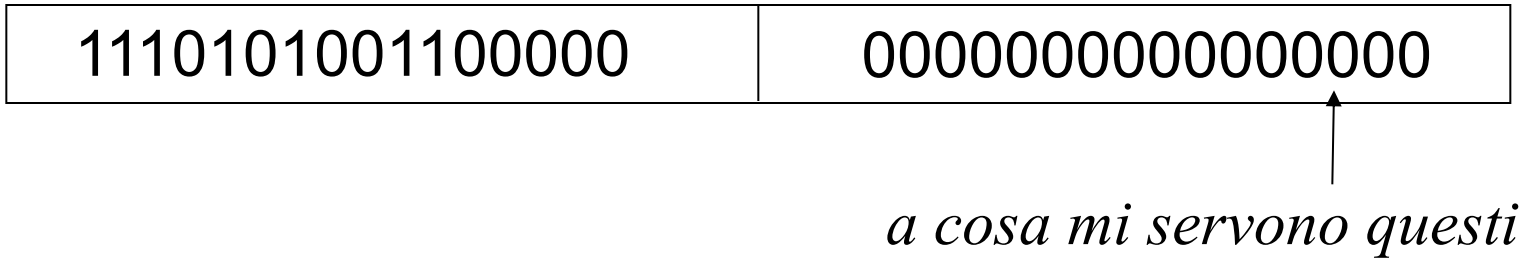
In generale

La granularità è fissa e pari a 2^{-nF} :

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

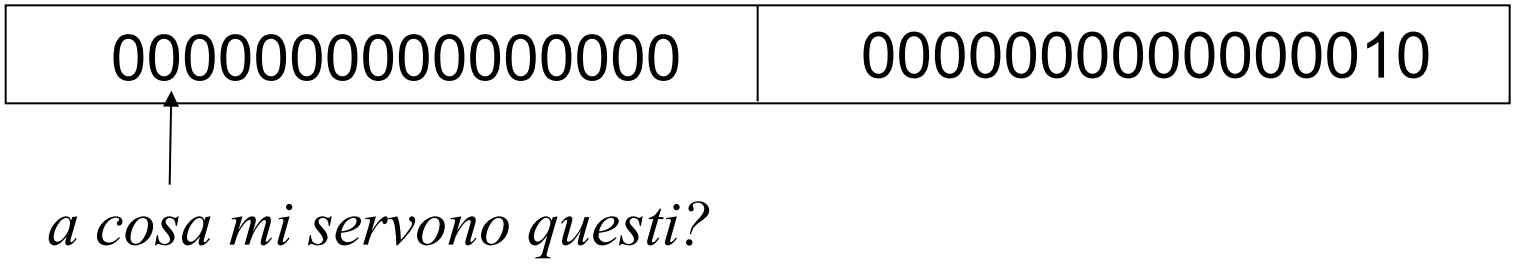
Inconveniente della rappresentazione in virgola fissa

- Con la rappresentazione precedente, vogliamo rappresentare un numero "grande", ad esempio 60˙000



⇒ Molto meglio un campo più grande per la parte intera!

- Vogliamo rappresentare un numero "piccolo", ad esempio 2^{-15}



⇒ Molto meglio un campo più grande per la parte frazionaria!

Per numeri grandi, inutile grande precisione (piccola granularità)

⇨ Si vuole un sistema di rappresentazione in cui la granularità dipende dal numero rappresentato:

si estende così ***l'intervallo*** dei numeri rappresentabili

parte intera	parte frazionaria

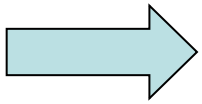
Numeri piccoli *Numeri grandi*

Rappresentazione in virgola mobile (floating point)

IDEA DI BASE

Nelle calcolatrici scientifiche si usa la notazione esponenziale

1.345 EXP 08	rappresenta $1.345*10^8$



IDEA: rappresentare un numero come

$$N = mant * 2^{esp}$$

(l'esponente 2 è più comodo con numeri binari)

QUINDI

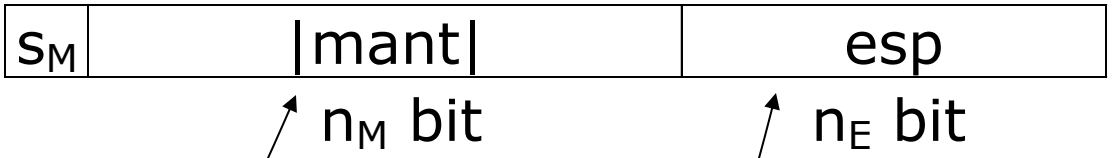
Dato un numero da rappresentare N:

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

FORMATO

Si memorizzano
segno, ***mantissa*** ed ***esponente***

Esempio di formato:



Esempi

$1011010111_2 * 2^{29}$

$0.11011011_2 * 2^{-36}$

QUINDI

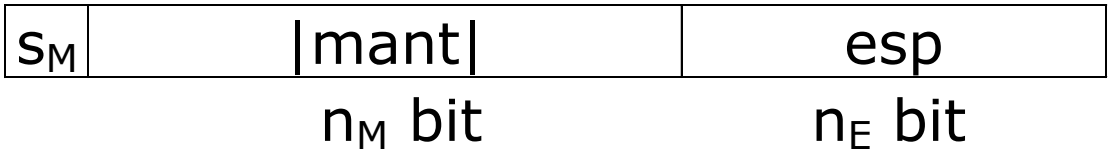
Dato un numero da rappresentare N:

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

FORMATO

Si memorizzano
segno, ***mantissa*** ed ***esponente***

Esempio di formato:



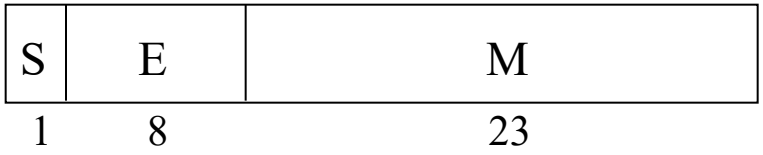
Problema

In questo modo la rappresentazione non è univoca:

$$0.1_2 = 1_2 * 2^{-1}$$
$$= 10_2 * 2^{-2}$$
$$= 0.1_2 * 2^{0}$$

$\Rightarrow$ si stabilisce la "forma normalizzata" della mantissa

Standard IEEE 754 in singola precisione (32 bit)



Segno S: 0 segno + 1 segno -

Normalizzazione e mantissa

$N = \cancel{1}.xxxxxxxxx * 2^{esp}$

23 bit (M)

hidden bit

Rappresentazione dell'esponente

$E = esp + 127$

Esempio: N= -3.5

Segno: 1

$N = 11.1$

$= 1.11 * 2^1$

$M = 1100\ldots0$

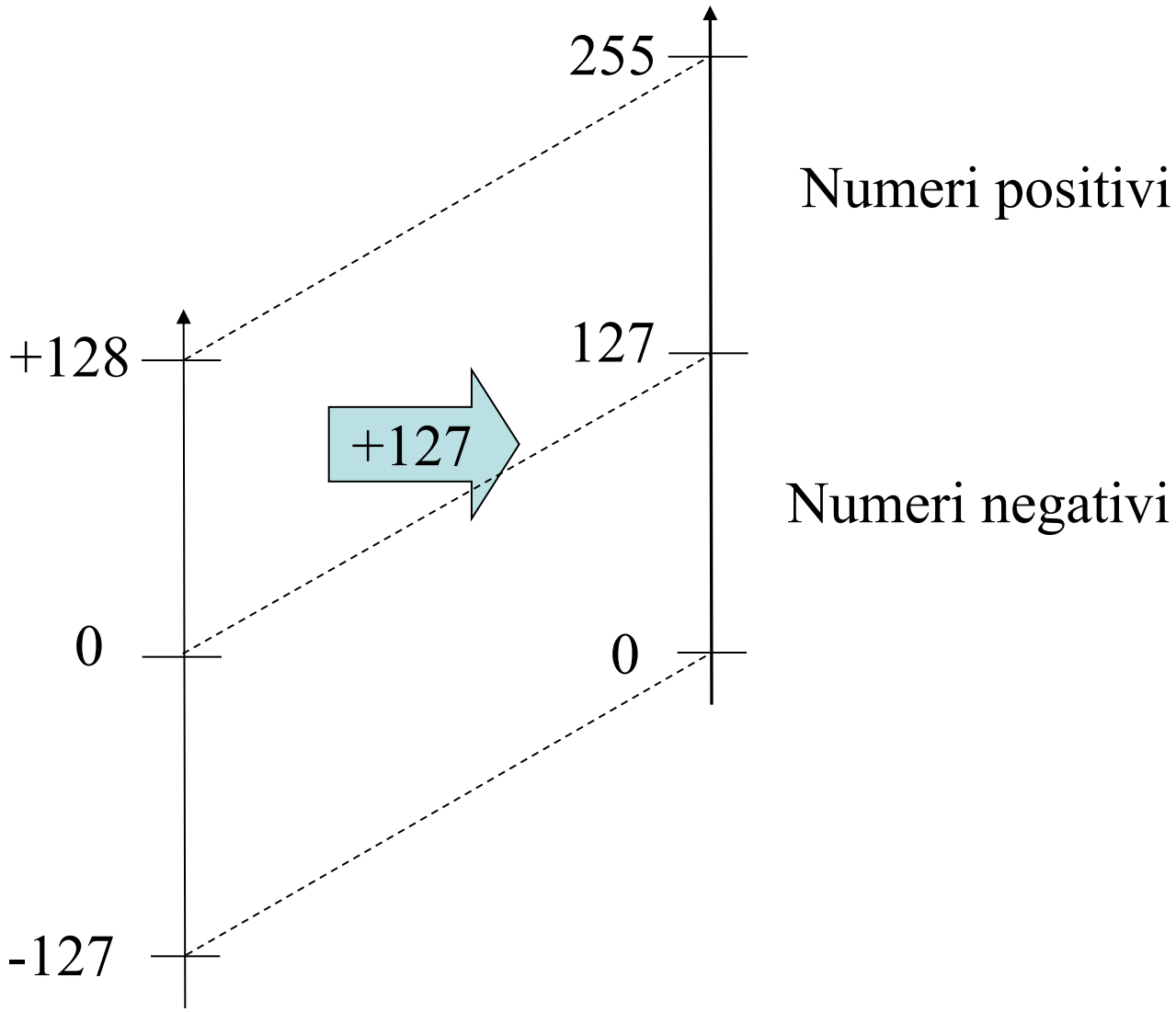
$E = 1 + 127 = 128$

$= 10000000$

1 10000000 11000000000000000000000

*Institute of Electrical and Electronic Engineering

Nota sulla rappresentazione dell'esponente



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 10000000000000000000000

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 10000000000000000000000

Soluzione

Segno: +

Esponente: $E = 2^7 = 128$ esp = 128-127 = 1

Mantissa: mant = 1.1

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

Esercizio

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

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

Esercizio

Rappresentare il numero decimale 0.25 secondo lo standard in virgola mobile IEEE 754 a 32 bit.

Esercizio

Rappresentare il numero decimale 0.25 secondo lo standard in virgola mobile IEEE 754 a 32 bit.

Soluzione

Segno: 0

Rapp. binaria: $0.25_{10} = 0.01_2$

Forma normalizzata: $N = 1.0 * 2^{-2}$

Esponente: esp = -2 $\Rightarrow$ E = -2+127 = 125_{10} = 01111101

$\Rightarrow$ IEEE754: 0 01111101 0000……0

PASSAGGIO IMPORTANTE: normalizzazione

ES. 1

Rappresentazione binaria:

1001.01001 =

$1.00101001 * 2^3$ (forma normalizzata)

ES. 2

Rappresentazione binaria:

0.001011 =

$1.011 * 2^{-3}$ (forma normalizzata)

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…)

Limiti di rappresentazione per la IEEE-754 a 32 bit

Nota "tecnica" sull'esponente

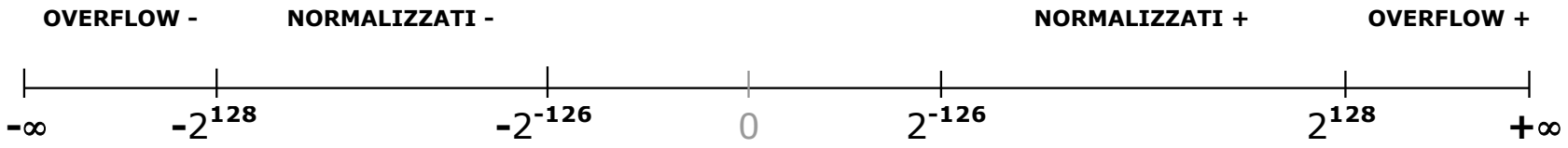
- Estremo superiore 1…1 (255) indica +∞, -∞ o situazioni particolari a seconda di mantissa e segno
- Estremo inferiore 0…0 (0) indica un numero "denormalizzato"

Limiti dei numeri "normalizzati"

MinExp = -126 MaxExp = +127

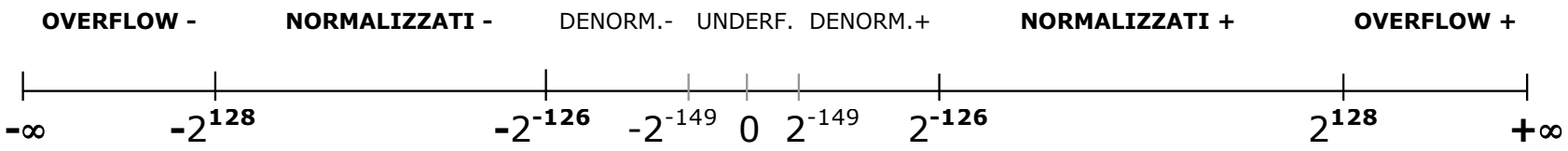
Numero più grande: $\pm\ 1.11\ldots1_2 * 2^{127} \approx 2*2^{127} = 2^{128}$

Numero più piccolo: $\pm\ 1.00\ldots0_2 * 2^{-126} = 2^{-126}$



Numeri denormalizzati

$E=0 \Rightarrow N=0.M*2^{-126}$



Confronto con virgola fissa

L'intervallo dei valori rappresentabili è molto più limitato. Es:

- anche se dedicassimo tutti i 32 bit alla parte intera, max $2^{32}-1$
- anche se dedicassimo tutti i 32 bit alla parte fraz, min 2^{-32}

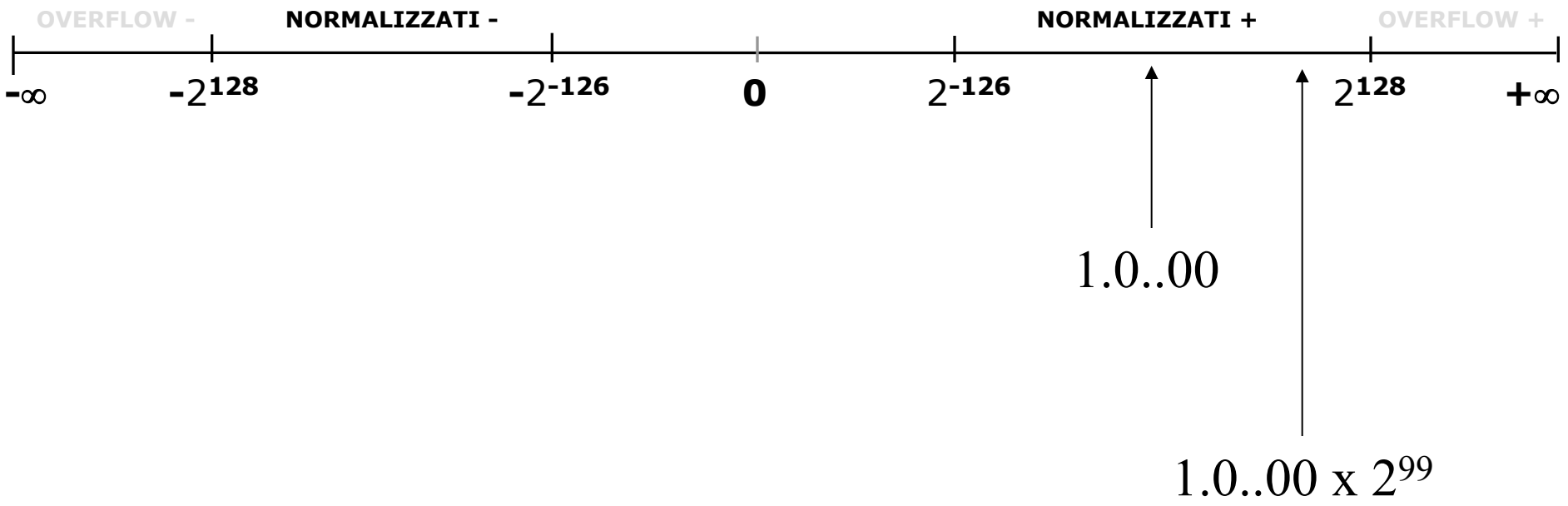
Precisione (granularità)

- I due intervalli di numeri positivi e negativi esprimibili non formano insiemi continui e i numeri ***non sono uniformemente distribuiti*** (sono più radi per valori elevati dell'esponente e più fitti per valori piccoli dell'esponente - si infittiscono nei pressi dello zero)

⇨ La precisione è "concentrata" dove ce n'è bisogno!

Esempio

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



➡ La precisione è "concentrata" dove ce n'è bisogno!

MA NON E' TUTTO ORO QUELLO CHE LUCCICA…

- L'insieme dei numeri in virgola mobile non coincide con R (l'insieme dei numeri reali):
  - l'insieme dei numeri reali è ***denso e illimitato***
  - l'insieme dei numeri in virgola mobile ***non è denso*** ed è limitato tra un numero reale ***massimo*** ed uno ***minimo*** esprimibili

    ⇒ Approssimazioni inevitabili

    ⇒ Aritmetica "reale" del calcolatore diversa da quella classica

UNA STORIA DI GUERRA (A)

- Durante la Guerra del Golfo nel 1991, un missile Patriot non riuscì ad intercettare un missile Scud iracheno: morirono 28 americani

UNA STORIA DI GUERRA (B)

- In uno studio successivo, si scoprì il motivo:
  - il Patriot incrementava un contatore ogni 0.1 sec
  - il tempo complessivo veniva ottenuto moltiplicando per 0.1
  - problema: 0.1 non è rappresentabile esattamente in virgola mobile

$$0.1_{10} = 0.0\overline{0011}$$

  - l'errore con la precisione adottata: 0.000000095367431640625

    …che dopo 100 ore portava ad una sottostima del tempo di 0.34 s
  - e in 0.34 sec lo Scud percorre circa 500 m…!

Operazioni sui numeri "reali": problemi

- Le operazioni possono dare luogo a ***errori***
- Esempio: divisione per numeri molto piccoli

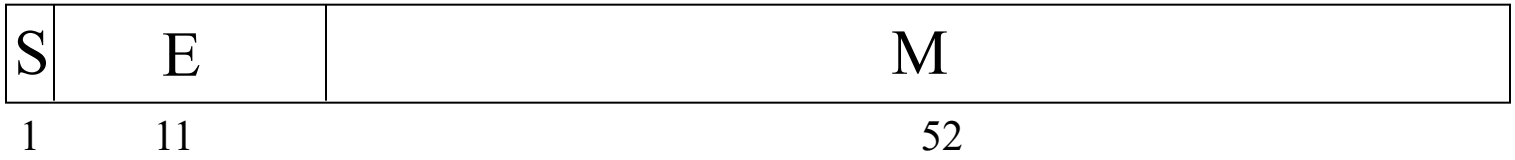
  → il risultato può cadere nell'intervallo di ***overflow*** (+ o -)
- Esempio: divisione per numeri molto grandi

  → può dar luogo al fenomeno della ***cancellazione*** (risultato = 0)

IEEE 754 in doppia/quadrupla precisione

- Doppia precisione

64 bit



$$N = (-1)^S * 1.M * 2^{E-1023}$$

- Quadrupla precisione

128 bit

$$N = (-1)^S * 1.M * 2^{E-16383}$$

NUMERI IN VIRGOLA MOBILE

NEI LINGUAGGI

DI PROGRAMMAZIONE

Linguaggio macchina (cenno)

- Istruzioni specifiche per i calcoli in virgola mobile
- Spesso nei processori a 32 bit sono presenti istruzioni per il calcolo in doppia precisione (64 bit): si utilizzano due registri per operando!

Esempio

```
add.s  $f1, $f3, $f4     # $f1=$f3+$f4 (singola precisione-32 bit)

add.f  $f0, $f2, $f4     # [$f0-$f1]=[$f2-$f3]+[$f4-$f5]
                         # (doppia precisione-64bit)
```

Linguaggio C

Sono disponibili i tipi *float*, *double*, *long double*

(cfr LINGUAGGIO C – Tipi predefiniti)