

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
(non usata nei calcolatori)

Rappresentazione in virgola mobile

Formato

In entrambi i casi, una rappresentazione è 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

Esempio (n=32)

S	E	M
1	8	23

Vedremo poi il significato...

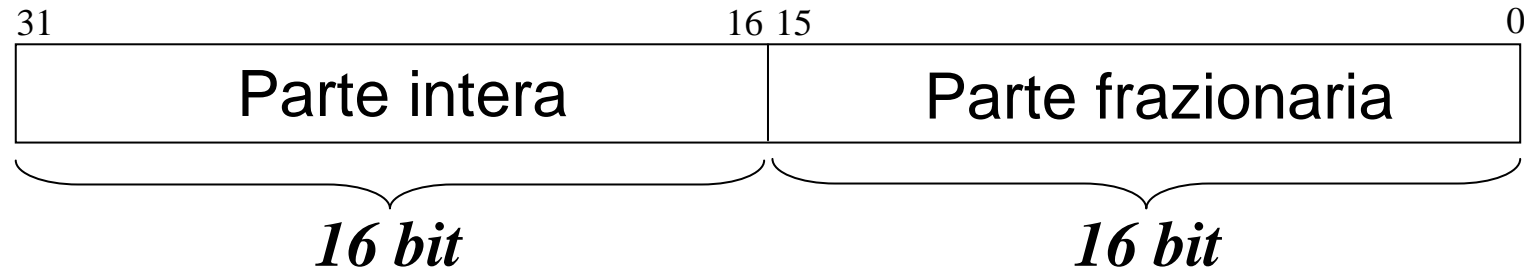
Rappresentazione in virgola fissa

- 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

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:

0000000000000011010100000000000000

|

Intervallo di rappresentazione

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

1111111111111111.1111111111111111
└──────────┘ └──────────┘
16 bit *16 bit*

Intervallo di rappresentazione

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

1111111111111111.1111111111111111

 16 bit 16 bit

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

- Parte frazionaria:

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

Intervallo di rappresentazione

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

1111111111111111.1111111111111111

 16 bit 16 bit

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

- Parte frazionaria:

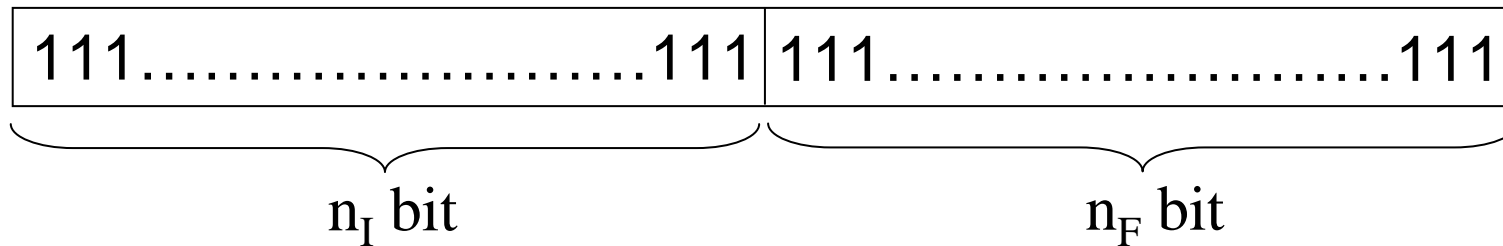
$$\frac{0.1111111111111111 + 0.0000000000000001}{1.0000000000000000} = \leftarrow 2^{-16}$$

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

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

Intervallo di rappresentazione

- max numero rappresentabile N_{MAX} :



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

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

se n_F grande pari a circa 1

➡ $N_{MAX} = (2^{n_I}-1) + (1 - 2^{-n_F})$
se n_F grande pari a circa 2^{n_I}

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?

$$\begin{array}{r} 00000000000001101|0100000000000000 \\ 0.0000000000000001 \\ \hline 00000000000001101|0100000000000000\mathbf{1} \end{array} +$$

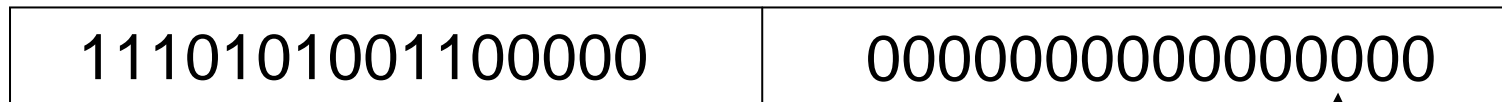
In generale

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

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

Inconveniente della rappresentazione in virgola fissa

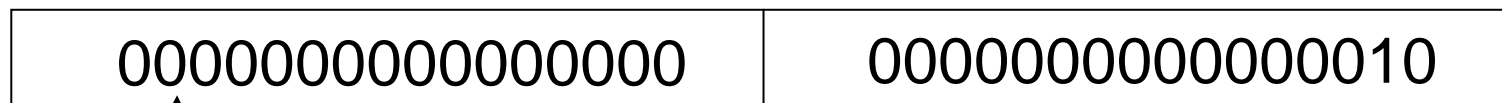
- Con la rappresentazione precedente, vogliamo rappresentare un numero “grande”, ad esempio 60.000



a cosa mi servono questi?

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

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



a cosa mi servono questi?

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

Altro esempio

- Peso molecola CO₂: 2^{-80} grammi

000000000000000000000000.0000000000 ... 000000000000000001

- Peso Ferrari: 600 kg = 2^{19} grammi

1000000000000000000000000.0000000000 ... 000000000000000000

- 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 la granularità dipende dal numero rappresentato:

si estende così *l'intervallo* dei numeri rappresentabili

Rappresentazione in virgola mobile (floating point)

parte intera	parte frazionaria
--------------	-------------------



Numeri piccoli

Numeri grandi

- IDEA: un campo per rappresentare la “posizione della virgola”!

“posizione”	...
-------------	-----

PERO’...

- Se devo rappresentare un numero grande potrei non avere la virgola, anzi potrei avere molti zeri alla fine...

`1011010111000.000000`

- Se devo rappresentare un numero piccolo: potrei non avere parte intera, anzi molti zeri subito dopo la virgola

[illegible]

IDEA: rappresentare un numero come

$$N = \text{mant} * 2^{\text{exp}}$$

Esempi precedenti:

$1011010111 \cdot 2^{29}$
 $0.11011011 \cdot 2^{-36}$

nel campo “esponente”

QUINDI

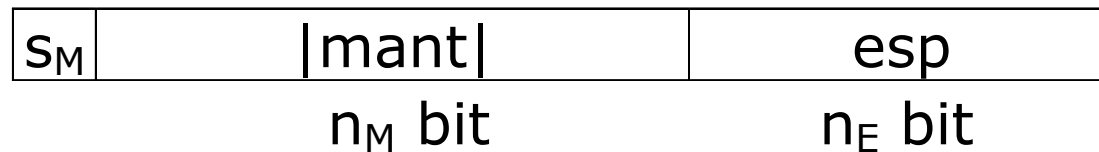
Dato un numero da rappresentare N:

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

FORMATO

Si memorizzano
segno, *mantissa* ed *esponente*

Esempio di formato:



QUINDI

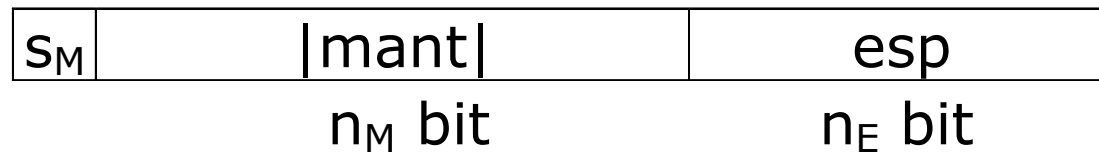
Dato un numero da rappresentare N:

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

↓ ↓
FORMATO

Si memorizzano
segno, *mantissa* ed *esponente*

Esempio di formato:



Problema

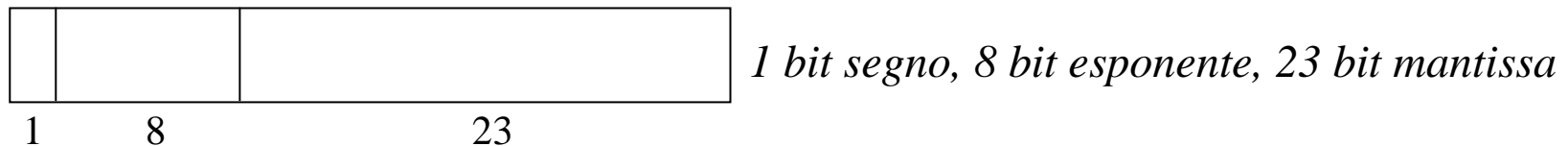
In questo modo la rappresentazione non è univoca:

$$\begin{aligned} 0.1_2 &= 1 * 2^{-1} \\ &= 10 * 2^{-2} \\ &= 0.1 * 2^0 \end{aligned}$$

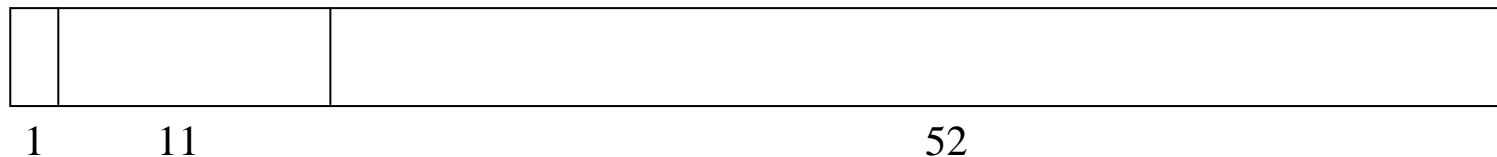
⇒ si stabilisce la “forma normalizzata” della mantissa

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



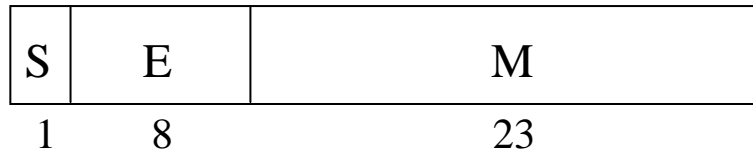
- 64 bit per i numeri in precisione doppia *1 bit segno, 11 bit esponente, 52 bit mantissa*



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

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

23 bit (M)

Rappresentazione dell'esponente

$$E = \text{esp} + 127$$

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

Esempio: num = -3.5

Segno: 1

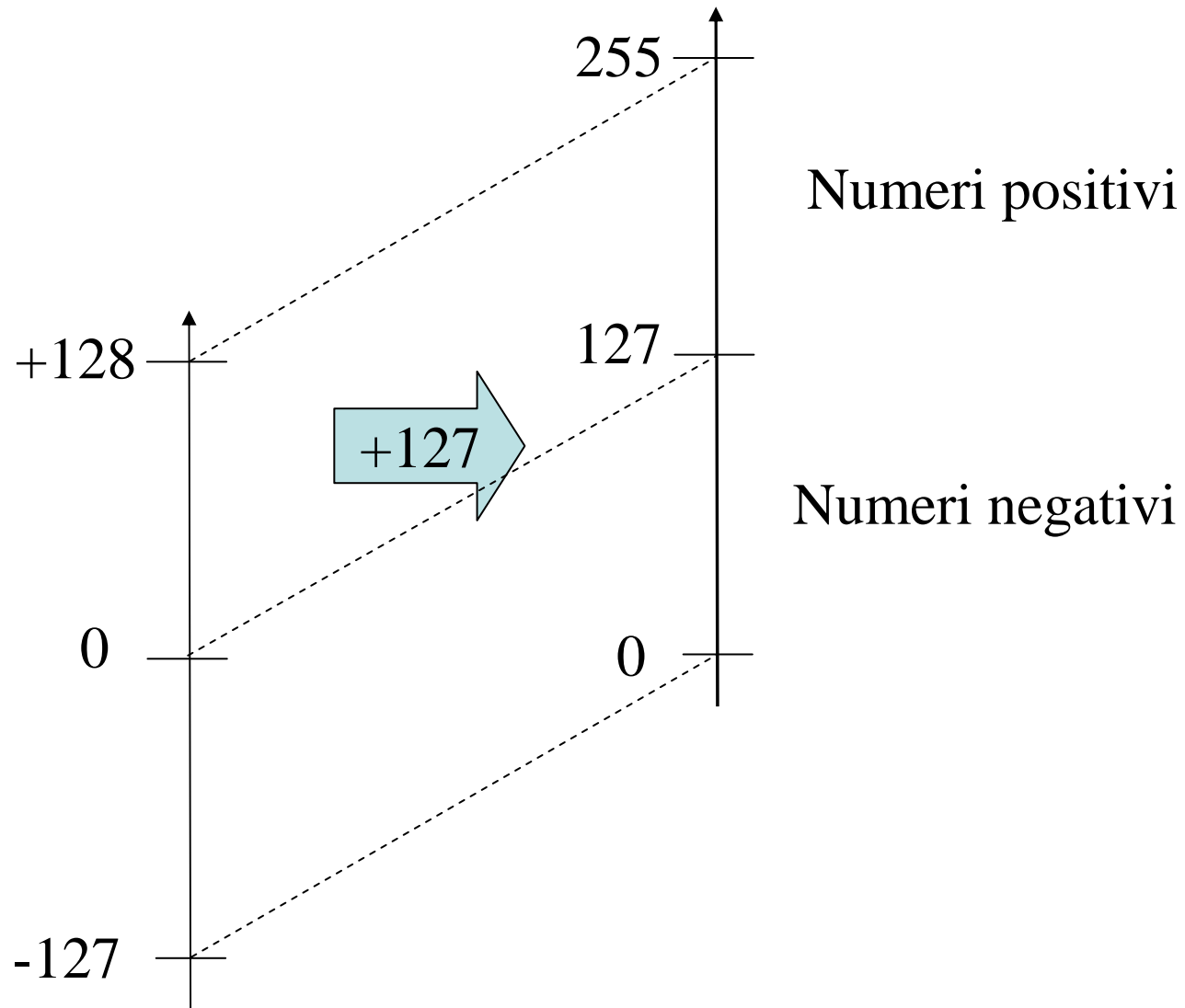
$$\begin{aligned} \text{num} &= 11.1 \\ &= 1.11 * 2^1 \\ M &= 1100...0 \end{aligned}$$

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



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 1000000000000000000000000000

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: $E = 01111110_2 = 126$

$$\text{esp} = 126 - 127 = -1$$

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

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...)

Capacità di rappresentazione dei numeri in virgola mobile

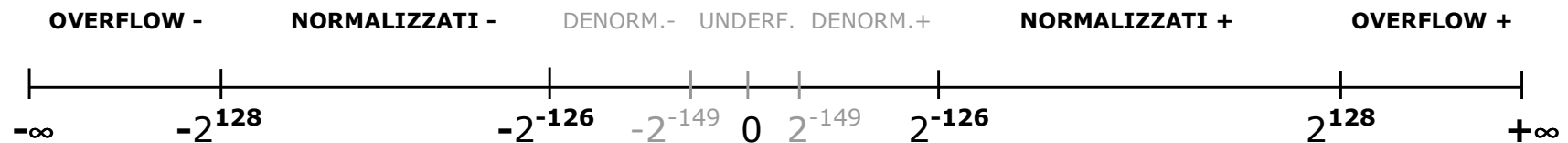
- L'insieme dei numeri in virgola mobile non coincide con \mathbb{R} (l'insieme dei numeri reali):
 - l'insieme dei numeri reali è *denso e illimitato*
 - l'insieme dei numeri in virgola mobile *non è denso*
ed è limitato tra un numero reale *massimo* ed uno *minimo* esprimibili
- \Rightarrow L'aritmetica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

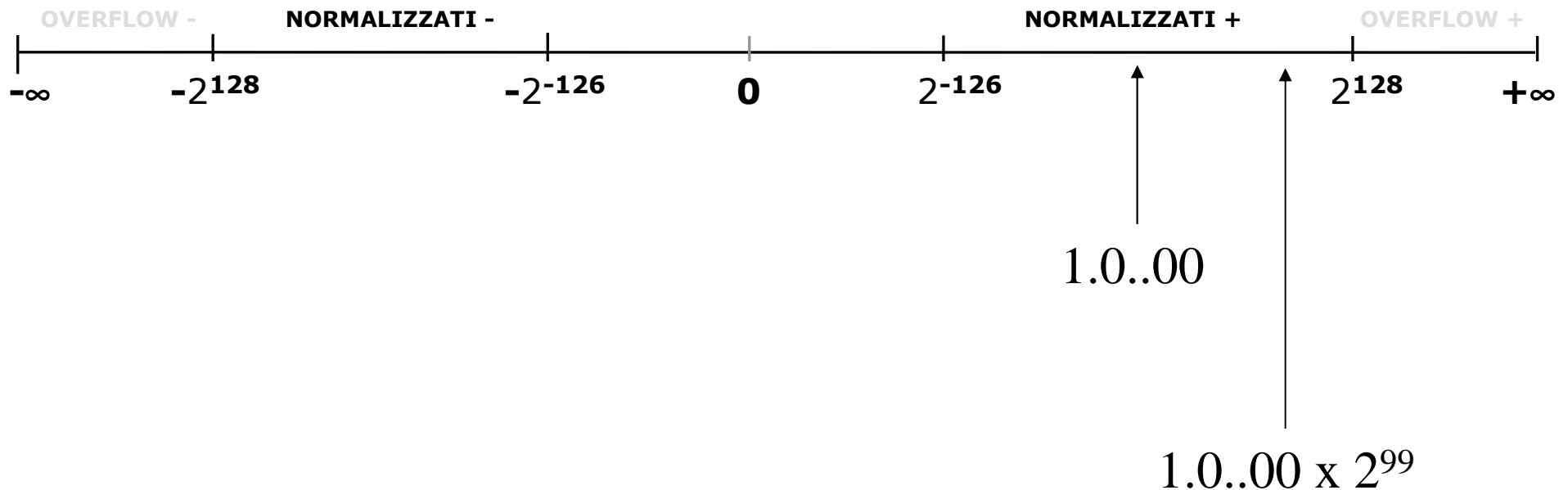
- anche se dedicassimo tutti i 32 bit alla parte intera, max 2^{32}
- anche se dedicassimo tutti i 32 bit alla parte fraz, min 2^{-32}

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
→ 4.261.412.865 numeri
- 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)

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

Fenomeno della cancellazione: esempio

Consideriamo la sottrazione

$$1.11100\dots0 \times 2^{-126} -$$

$$1.11000\dots0 \times 2^{-126} =$$

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

➔ 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
(vedere libro per gli interessati)