

CODIFICA DEI NUMERI REALI IEEE 754 – FLOATING POINT (A VIRGOLA MOBILE)

Il modo in cui i numeri reali (con la virgola) vengono codificati nella memoria è stabilito da uno standard chiamato **IEEE 754**. IEEE (Institute of Electronic and Electric Engineer) è un istituto americano che si occupa di standardizzazione, ossia di creare delle regole condivise, nel mondo dell'elettronica e dell'informatica. Il nome ufficiale di questo standard è **IEEE 754 Standard for Binary Floating-Point Arithmetic**, ma generalmente viene indicato semplicemente con il nome **IEEE 754** oppure **floating point** (in italiano: "virgola mobile").

Prima di spiegare cosa prevede questo standard è opportuno ricordare il concetto di rappresentazione dei numeri in "notazione scientifica normalizzata". Cosa che ora faremo.

La notazione scientifica

In un numero con la virgola, in base 10 come in qualsiasi altra base di un sistema di numerazione posizionale (binario, esadecimale, ottale..), le cifre a destra della virgola apportano al numero un contributo pari al valore della cifra stessa moltiplicato per la base elevata alla "posizione" della cifra, dove però le posizioni a destra della virgole sono considerate negative (-1,-2,...),

Esempio:

$$\begin{array}{c} 2,23 \\ \downarrow \quad \quad \quad \searrow \quad \quad \quad \rightarrow \\ \boxed{2 \times 10^0 = 2} \quad + \quad \boxed{2 \times 10^{-1} = 0,2} \quad + \quad \boxed{3 \times 10^{-2} = 0,03} \quad = \quad \boxed{2,23} \end{array}$$

La stessa cosa vale in binario

$$\begin{array}{c} 1,01 \\ \downarrow \quad \quad \quad \searrow \quad \quad \quad \rightarrow \\ \boxed{1 \times 2^0 = 1} \quad + \quad \boxed{0 \times 2^{-1} = 0 \times \frac{1}{2} = 0} \quad + \quad \boxed{1 \times 2^{-2} = 1 \times \frac{1}{4} = 0,25} \quad = \quad \boxed{1,25_{(10)}} \end{array}$$

La notazione scientifica è un modo per rappresentare "in maniera comoda" numeri molto grandi o numeri molto piccoli utilizzando le potenze della base del sistema di numerazione, quindi nel sistema decimale le potenze di 10.

Ad esempio, la massa della Terra, espressa in kg è questa:

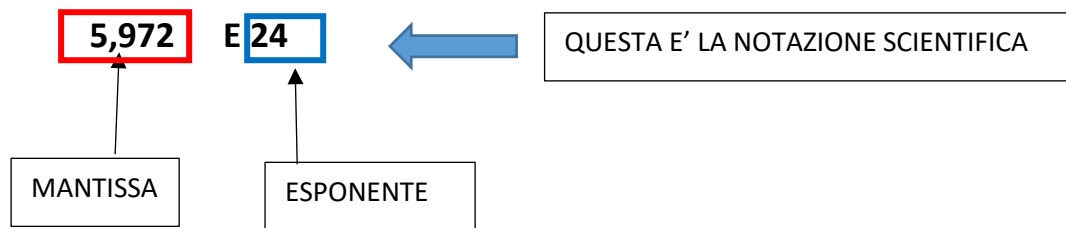
5972000000000000000000000000 kg (un po' scomodo...)

Lo stesso numero si può esprimere in notazione scientifica nel seguente modo:

5,972 E24 kg

dove "E24" significa "per 10 elevato 24".

La prima parte del numero (5,972) si chiama mantissa, la seconda (24) si chiama esponente.



Lo stesso numero si potrebbe esprimere in diversi modi moltiplicando e dividendo per la stessa potenza di 10:

59,72 E 23 oppure 0,5972 E 25 (vedi?...la virgola è mobile!)

Quindi la notazione scientifica è quella che esprime un numero come:

MANTISSA X 10^{ESPONENTE}

Notazione scientifica normalizzata

La notazione scientifica si dice normalizzata quando ha la parte intera costituita da **una sola cifra diversa da 0**. Ossia la parte intera è un numero compreso fra 1 e 9.

Nell' esempio che abbiamo fatto:

5,972 E 24 è in notazione scientifica **normalizzata**

59,72 E 23 è in notazione scientifica ma **non normalizzata**

0,5972 E 25 è in notazione scientifica ma **non normalizzata**

Altro esempio:

la massa dell'elettrone è

0,000000000000000000000000000091 kg

che in notazione scientifica normalizzata si scrive 9,1 E -31

Notazione scientifica normalizzata in binario

Anche i numeri binari si possono esprimere in notazione scientifica normalizzata, ad esempio il numero 1011,001

in notazione scientifica normalizzata diventa:

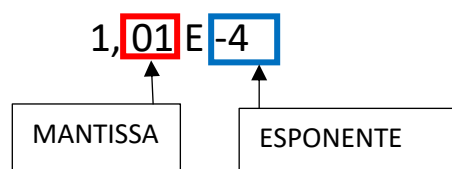
1,011001 E 3 dove ora "E" significa "per 2 elevato"

Altri esempi:

0,000101 diventa 1,01 E -4

1101101 diventa 1,101101 E6

In binario però l'unica cifra che può esserci a sinistra della virgola è "1" (essendo l'unica cifra diversa da 0), per questo motivo, essendo ovvio che la parte a sinistra della virgola è sempre 1, con il termine **MANTISSA** si indica, in binario, **solamente la parte del numero che sta alla destra della virgola**.

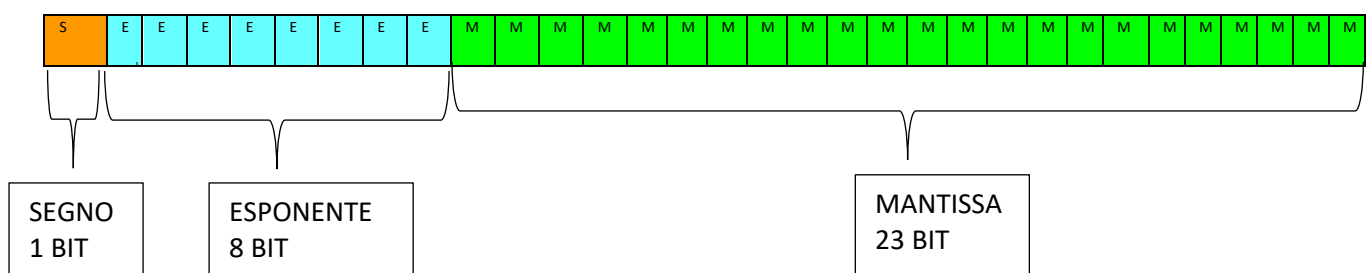


Dopo aver spiegato i concetti di notazione scientifica, notazione scientifica normalizzata, mantissa ed esponente, illustriamo lo standard IEEE 754 per la codifica dei numeri reali. Tale standard si divide in due possibili standard: floating point a precisione singola e floating point a precisione doppia.

IEEE 754 Floating point a precisione singola

Lo standard **IEEE 754 A PRECISIONE SINGOLA** (indicato con la parola SINGLE) prevede la memorizzazione di un numero con la virgola in binario esprimendolo in notazione scientifica normalizzata e **UTILIZZANDO 32 BIT** nel seguente modo:

- UN BIT per il segno del numero (vale 0 se il numero è positivo, 1 se negativo)
- 8 BIT per l'ESPONENTE (codificato in un modo particolare)
- 23 BIT per la MANTISSA



Come indicato sopra, l'esponente è codificato in un modo particolare. Questa modalità è indicata con il

termine di BIAS 127 (o anche “**ECCESSO 127**”) che ora spieghiamo.

Come abbiamo visto negli esempi, l’esponente della notazione scientifica normalizzata può assumere in alcuni casi valori positivi e in alcuni casi valori negativi. Con gli 8 bit a disposizione è quindi necessario poter rappresentare sia numeri positivi che negativi. (si potrebbe usare la codifica in CA2 ma si è preferito usare la tecnica BIAS perché essa consente di costruire ALU in grado di eseguire calcoli in floating point più velocemente).

Visto che le possibili configurazioni che si possono ottenere con 8 bit sono 256 si è deciso di utilizzare i “primi” numeri binari, da 1 a 126, per gli esponenti negativi, il 127 per rappresentare l’esponente = 0 e la seconda metà di configurazioni (da 128 a 254) per gli esponenti positivi. La configurazione esponente=00000000 e esponente=11111111 sono invece utilizzate per “casi particolari”, che vedremo in seguito.

In pratica quindi:

- BIT dell’esponente=01111111 (sarebbe 127_{10}) indica l’esponente 0
- BIT dell’esponente=01111110 (sarebbe 126_{10}) indica l’esponente -1
- BIT dell’esponente=01111101 (sarebbe 125_{10}) indica l’esponente -2
-
- BIT dell’esponente=10000000 (sarebbe 128_{10}) indica l’esponente 1
- BIT dell’esponente=10000001 (sarebbe 129_{10}) indica l’esponente 2
- BIT dell’esponente=10000010 (sarebbe 130_{10}) indica l’esponente 3
-

Conversione da “floatig point single precision” a decimale

Come è possibile dunque capire quale è il numero decimale con la virgola memorizzato in floating point a precisione singola (quindi con 32 bit) in una cella di memoria?

Si può fare con la seguente procedura (algoritmo):

PROCEDURA (ALGORITMO)	Esempio con 0 10000001 111000000000000000000000
Leggo il BIT in posizione 0 partendo da sinistra (MSB) e determino il segno del numero se BIT(0)=0 → positivo se BIT(0)=1 → negativo	BIT (0)= 0 → segno +
Calcolo il valore dell’esponente della notazione scientifica normalizzata in base 2: converto in decimale il numero intero memorizzato nei BIT dalla posizione 1 alla posizione 8 (da sinistra) e sottraggo 127	Bit(1 – 8): 10000001 → $129_{(10)}$ $129 - 127 = 2$ Quindi esponente=2
Scrivo il numero binario in notazione scientifica normalizzata: 1,Mantissa E Esponente (la Mantissa è formata dai 23 Bit dalla posizione 9 alla posizione alla posizione 31 da sinistra)	1,111000000000000000000000 E 2
Converto in decimale i Il numero binario espresso in notazione scientifica normalizzata	$1,111000000000000000000000 E 2 =$ $111,1_{(2)} \rightarrow 7,5_{(10)}$ Il numero decimale memorizzato è 7,5

Esercizio: determinare il valore decimale del seguente numero memorizzato in floating point a precisione singola 10111110001000000000000000000000

Soluzione:

1 01111100 010000000000000000000000

s e m

- Segno: BIT(0)=1 → segno –
- esponente: 01111100 → 124
 $124 - 127 = -3$
-
- numero in notazione scientifica normalizzata: 1,01 E -3
- Conversione: $1,01 \text{ E } -3 = 0,00101 = 2^{-3} + 2^{-5} = 1/2^3 + 1/2^5 = 1/8 + 1/32 = 0,15625$
- Numero decimale memorizzato: **-0,15625**

ATTENZIONE: RICORDARSI DI AGGIUNGERE IL SEGNO ALLA FINE!

Per verificare se l'esercizio è stato svolto correttamente si può utilizzare il seguente convertitore FP (Floating Point): <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

Conversione da decimale a floatig point single precision

Come è possibile ottenere la rappresentazione in floating point a precisione singola (quindi con 32 bit) di un numero decimale?

Si può fare con la seguente procedura:

PROCEDURA (ALGORITMO)	Esempio con il numero 4,25
Determino il bit del segno: numero positivo $\rightarrow$ Bit(0) = 0 numero negativo $\rightarrow$ Bit(0) = 1	BIT(0) = 0
Converto in binario la parte intera	$4_{(10)} \rightarrow 100_{(2)}$
Converto in binario la parte decimale e la aggiungo alla parte intera	$4,25_{(10)} \rightarrow 100,01_{(2)}$
Esprimo il numero binario ottenuto in notazione scientifica normalizzata	$100,01 = 1,0001 \text{ E } 2$
Determino la MANTISSA prendendo la parte a destra della virgola del numero in notazione scientifica normalizzata. Se necessario aggiungo degli 0 a destra fino ad arrivare a 23 Bit	$1,0001 \text{ E } 2 \rightarrow$ Mantissa: 00010000000000000000000
Sommo 127 all'esponente della notazione scientifica normalizzata e converto il numero così ottenuto in binario. Il numero binario ottenuto è l'esponente della rappresentazione in FP	$2 + 127 = 129$ $129_{(10)} \rightarrow 10000001_{(2)}$ Esponente: 10000001
Scrivo in sequenza i bit del segno, dell'esponente e della mantissa.	Il numero in FP è: 01000000100010000000000000000000

Esercizio: rappresentare in Floating point a precisione singola il numero decimale -120,125

Soluzione:

- numero negativo quindi Bit(0) = **1**
- $120_{(10)} \rightarrow 1111000_{(2)}$
- $0,125_{(10)} \rightarrow 0,001_{(2)}$
- Il numero 120,125 in base 10 diventa in binario 1111000,001
- Il numero binario in notazione scientifica diventa $1,111000001 \text{ E } 6$
- Mantissa: **11100000100000000000000**
- Sommo 127 al valore dell'esponente: $6+127=133$

- Converto in binario il valore ottenuto: $133_{(10)} \rightarrow 10000101_{(2)}$
- Esponente: **10000101**
- Il numero in FP è: **11000010111100000100000000000000**

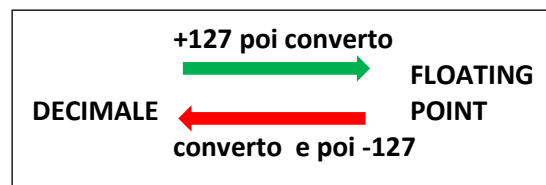
Esercizio: rappresentare in FP a precisione singola i seguenti numeri decimali (verificare la correttezza del risultato con il convertitore IEEE 754 online)

-198 57,125

Esercizio: determinare i numeri decimali rappresentati nella seguente codifica in floating point single precision (verificare la correttezza del risultato con il convertitore IEEE 754 online):

01000010001001000100000000000000 , 11000001101001000000000000000000

Può essere utile ricordarsi questo schema per il calcolo **dell'esponente** nella codifica floating point single:



Configurazioni speciali IEEE 754

Lo standard IEEE 754 prevede delle configurazioni speciali per la rappresentazione di valori particolari, con questo si intende che tali configurazioni speciali rappresentano dei valori predefiniti e per essi non si applicano gli algoritmi di conversione visti precedentemente:

Le configurazioni speciali sono identificate dal valore assunto dai bit riservati all'esponente

1. BIT dell'esponente tutti a 0: 00000000. In questo caso ci sono due possibilità:
 - a. se tutti i bit della MANTISSA SONO 0 rappresenta il valore decimale 0
 $000000000000000000000000_{(FP)} \rightarrow 0_{(10)}$
 - b. se vi sono bit della MANTISSA DIVERSI DA 0 il valore rappresentato è detto NON NORMALIZZATO (o SUBNORMALE) (spiegato dopo)
2. BIT dell'esponente tutti a 1: 11111111. Anche in questo caso ci sono due sottocasi:
 - a. se tutti i bit della MANTISSA SONO A 0 rappresenta il valore + infinito o – infinito (a seconda del valore del segno).

01111111100000000000000000000000_(FP) → +inf
 11111111100000000000000000000000_(FP) → -inf

Queste configurazioni vengono utilizzate, per esempio per indicare il risultato di una divisione per 0.

- b. Se vi sono bit della MANTISSA DIVERSI DA 0, la configurazione rappresenta il valore speciale NaN (Not a Number), utilizzata per indicare il risultato di un'operazione impossibile (ad esempio la radice quadrata di un numero negativo).

Estremi dell'intervallo di rappresentazione

Grazie all'utilizzo della notazione scientifica, la codifica in FP consente di rappresentare numeri molto grandi e numeri molto piccoli, ma anche tale rappresentazione, poiché utilizza un numero di bit finito (32) ha dei limiti.

- Valore più grande positivo rappresentabile in FP (estremo superiore):
 - segno positivo: BIT(0)=0
 - esponente con tutti 1 tranne LSB (se tutti i bit fossero a 1 rappresenterebbe inf.)
11111110
 - Mantissa con tutti i bit a 1
11111111111111111111111111111111

L'estremo superiore positivo FP che corrisponde a circa **3,4 E38**₍₁₀₎ (la verifica applicando l'algoritmo di conversione non è difficile ma un po' lunga... meglio usare il convertitore online)

01111111011111111111111111111111_(FP) → **3,4 E38**₍₁₀₎

- Valore più piccolo rappresentabile in FP (estremo inferiore):
L'estremo inferiore rappresentabile è l'opposto dell'estremo superiore (cambia il primo bit)

11111111011111111111111111111111_(FP) → **-3,4 E38**₍₁₀₎

Nel caso in cui il risultato di una operazione determinasse un risultato superiore dell'estremo superiore o inferiore all'estremo inferiore si verificherebbe, anche in questo caso un **Overflow**.

- Valore più piccolo positivo rappresentabile in FP (minimo positivo):
 - segno positivo: BIT(0)=0
 - esponente con tutti 0 tranne l'ultimo BIT (altrimenti sarebbe una *configurazione speciale*) 00000001
 - Mantissa con tutti i bit a 0
000000000000000000000000

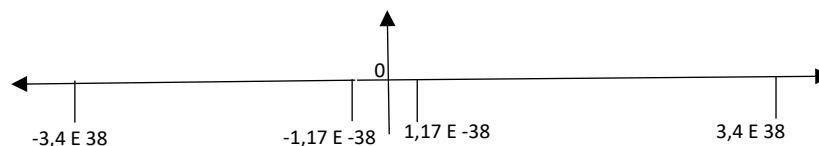
Il minimo positivo corrisponde dunque a:

$$00000000100000000000000000000000_{(FP)} \rightarrow 1,17 \text{ E } -38_{(10)}$$

- Valore più grande negativo rappresentabile in FP (massimo negativo):

Il massimo negativo rappresentabile è l'opposto del minimo positivo (cambia il primo bit)

$$10000000100000000000000000000000_{(FP)} \rightarrow -1,17 \text{ E } -38_{(10)}$$



La presenza del minimo positivo e del massimo negativo, ossia dei valori più vicini allo 0 ma diversi da 0, porta ad un nuovo problema, il problema dell'**underflow**, ossia l'impossibilità di rappresentare numeri troppo piccoli in valore assoluto. Infatti è impossibile rappresentare numeri positivi compresi fra 0 e $1,17 \text{ E } -38_{(10)}$ oppure numeri negativi compresi fra 0 e $-1,17 \text{ E } -38_{(10)}$.

Rappresentazione NON NORMALIZZATA O SUBNORMALE

Il problema dell'**underflow** non è risolvibile completamente. Per ridurre il "gap" di valori fra lo 0 e il più piccolo valore rappresentabile, lo standard prevede di rappresentare i numeri molto piccoli in forma NON NORMALIZZATA O SUBNORMALE.

Se:

- gli 8 bit dell'esponente della rappresentazione FP sono tutti a 0: 00000000
- e vi sono nella MANTISSA dei Bit diversi da 0

Per convenzione il numero viene considerato rappresentato in forma NON NORMALIZZATA, quindi il IL BIT a sinistra della virgola nella rappresentazione in notazione scientifica viene considerato **0 anziché 1**, inoltre **l'esponente della notazione scientifica binaria viene posto convenzionalmente a -126**.

Il numero in valore assoluto più piccolo rappresentabile in floating point single diventa quindi
segno: 0

esponente: 00000000

mantissa: 000000000000000000000001

e il suo valore decimale si ottiene da:

$$0,000000000000000000000001 \text{ E } -126 \rightarrow 1 \times 2^{-23} 2^{-126}_{(10)} = 2^{-149}_{(10)} = \mathbf{1,4 \times E-45}_{(10)}$$

Esercizio: quale è il valore decimale rappresentato nella seguente configurazione di bin in floating point single: 00000000000110000000000000000000 ?

Come detto, con questa rappresentazione si riduce il minimo valore assoluto rappresentabile ma non viene risolto il problema dell'**underflow**.

Problema dell'errore inerente

Oltre ai problemi di overflow e underflow, esiste un ulteriore problema nella rappresentazione in floating point dei numeri reali, il problema chiamato dell'**errore inerente**. Con questo nome si **indica l'impossibilità di rappresentare alcuni numeri decimali utilizzando un numero finito di bit**.

Il problema deriva dal fatto che, mentre i numeri reali sono un insieme di numeri **continuo** (ossia, dati due qualsiasi numeri è sempre possibile individuare un numero compreso fra essi), l'insieme dei numeri rappresentabili in FP è **discreto** (ossia si possono rappresentare solo un **numero finito di numeri**, quindi **alcuni numeri non si possono rappresentare!**).

La presenza di questo errore **ineliminabile** nella rappresentazione dei numeri reali con un numero finito di bit costringe i calcolatori ad effettuare delle approssimazioni nella rappresentazione di alcuni numeri.

Esempio:

dato il seguente numero reale rappresentato in FP single

0 10000011 000000000000000000000000

Il numero immediatamente successivo rappresentabile è il seguente:

0 10000011 000000000000000000000001

Quindi non è possibile rappresentare in FP numeri compresi fra questi due.

Convertendoli in numeri decimale:

$$0 \text{ 10000011 000000000000000000000000}_{(FP)} \rightarrow 16_{(10)}$$

$$0 \text{ 10000011 000000000000000000000001}_{(FP)} \rightarrow 16,0000019073486328125_{(10)}$$

Non è possibile rappresentare in FP i numeri decimali compresi fra 16 e 16,0000019073486328125

Il calcolatore dovrà inevitabilmente fare delle approssimazioni, nel caso in cui il risultato di un'operazione fosse

16,0000005 verrebbe approssimato a 16 (errore inerente=0,0000005)

se invece il risultato di una'operazione fosse

16,0000015 verrebbe approssimato a 16,0000019073486328125 (errore inerente=0,0000004073486328125)

Riassumendo si può dire che l'insieme dei numeri rappresentabili in Floating Point è:

- limitato nei suoi valori massimo positivo e minimo negativo, e questo genera il problema dell'**overflow**
- limitato nel più piccolo valore **assoluto** rappresentabile, e questo genera il problema dell'**underflow**
- discreto, e questo genera il problema dell'**errore inerente**.

IEEE 754 Floating point a precisione doppia

La codifica in Floating point che abbiamo visto utilizza **32 bit** ed è indicata con il termine: rappresentazione "in virgola mobile a precisione singola" (o **Single Floating Point**) e prevede, come detto: 1 bit per il segno, 8 bit per l'esponente e 23 bit per la mantissa.

Per aumentare il range di valori rappresentabili è stata definita una rappresentazione in floating point che utilizza **64 bit**. Questa codifica prevede l'utilizzo di **1 bit per il segno, 11 bit per l'esponente e 52 bit per la mantissa**. Tale codifica è detta "rappresentazione in virgola mobile a precisione doppia" (o **Double Floating Point**).

Le moderne CPU hanno al loro interno dei componenti hardware in grado di svolgere calcoli con la rappresentazione dei numeri in floating point (IEEE 754). Tali componenti sono indicati con il termine di **FPU (Floating Point Unit)** e consentono di eseguire calcoli complessi con numeri reali molto più velocemente rispetto alla velocità che impiegherebbe la ALU a svolgere gli stessi calcoli utilizzando le operazioni aritmetiche per numeri interi.

Esercizi: (verificare la correttezza dei risultati con il convertitore online)

1. Quale numero in base 10 è rappresentato con la seguente configurazione in floating point?

0	10000000	100100100000000000000000
S	e	m

2. Quale numero in base 10 è rappresentato con la seguente configurazione in floating point?

1	01111101	001100000000000000000000
S	e	m

3. Rappresentare in FP il numero 12,0140625
4. Rappresentare in FP il numero -6,3125
5. Rappresentare in FP il numero $1_{(10)}$