

# INTRODUZIONE

Il sistema che utilizza il pc per elaborare le istruzioni è il **SISTEMA BINARIO**, esso è composto da due sole cifre e sono: **0 e 1**.

Utilizzando il sistema binario e avendo  $n$  cifre a disposizione possiamo rappresentare  $2^n$  numeri, questo per via del calcolo combinatorio:

$$2 \cdot 2 \cdots 2_n$$

## POTENZE DI DUE

Le potenze di due utili per le conversioni da binario a decimale sono:

• $2^0 = 1$	• $2^8 = 256$
• $2^1 = 2$	• $2^9 = 512$
• $2^2 = 4$	• $2^{10} = 1024$
• $2^3 = 8$	• $2^{11} = 2048$
• $2^4 = 16$	• $2^{12} = 4096$
• $2^5 = 32$	• $2^{13} = 8192$
• $2^6 = 64$	• $2^{14} = 16384$
• $2^7 = 128$	• $2^{15} = 32768$

## CONVERSIONE NUMERICA

### CONVERSIONE DA BINARIO A DECIMALE

Un esempio di conversione da binario a decimale può essere:

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 \ 1 \\ \hline 2 \end{array} = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 19_{10}$$

### CONVERSIONE DA DECIMALE A BINARIO

Per la conversione da binario a decimale ci sono 2 metodi:

#### METODO 1:

- Trova la più grande potenza di 2 più adatta;
- sottrai;
- se non esce 0 ripeti.

#### ESEMPIO:

$47_{10}$	1	0	1	1	1	1
11	1	0	1	1	1	1

$$47 - 32 = 15 \quad 1$$

$$15 - 8 = 7 \quad 1$$

$$7 - 4 = 3 \quad 1$$

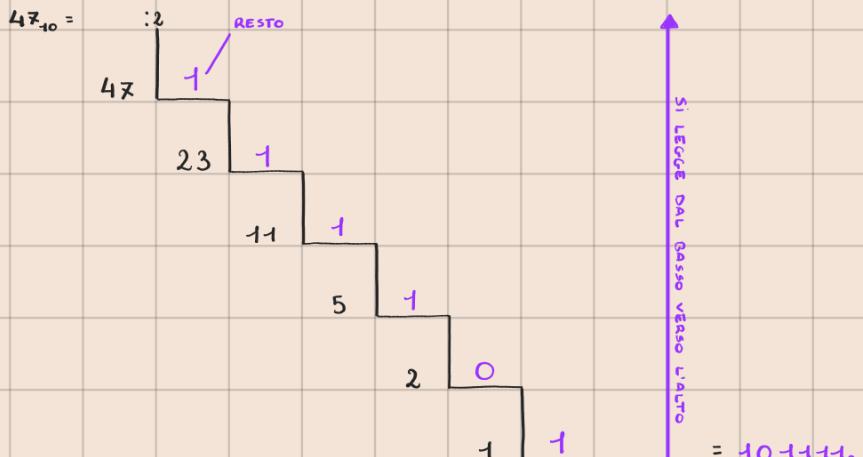
↓

$$3-2=1 \quad 1$$

## • METODO 2:

- dividi per 2;
- metti il resto a sinistra.

## ESEMPIO:



## DA X A Y

Quando invece abbiamo una base  $x$  che vogliamo convertirla ad una base  $y$  prima convertiamo da  $x$  a decimale e poi da decimale a  $y$ .

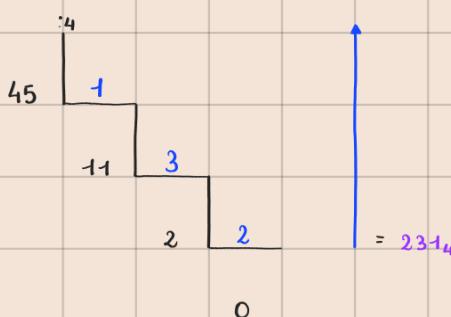
## ESEMPIO:

Abbiamo  $25_{20}$  e vogliamo convertirlo in base 4:

- convertiamo prima in base 10 quindi:

$$25_{20} = 5 \cdot 20^0 + 2 \cdot 20^1 = 45_{10}$$

- convertiamo 45 in base 4 e per fare questo basterà dividere 45 finché non esce 0 e ad ogni divisione metteremo da parte i resti.



## VALORI BINARIO E INTERVALLO

Numeri decimali a  $n$  cifre

- Quanti valori?  $10^n$

- Range?  $[0, 10^n - 1]$

ESEMPIO CON UN NUMERO DECIMALE A 3 CIFRE:

$$\cdot 10^3 = 1000 \text{ possibili valori}$$

• Range:  $[0, 999]$

Numeri binari a n cifre:

• Quanti valori?  $2^n$

• Range?  $[0, 2^n - 1]$

ESEMPIO CON UN NUMERO BINARIO A 3 CIFRE:

•  $2^3 = 8$  possibili valori

• Range:  $[0, 7] = [000_2, 111_2]$

## NUMERI ESADECIMALI

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Le caratteristiche principali dei numeri esadecimali sono 2:

1) HA COME BASE 16

2) ABBREVIAZIONE BINARIA

### CONVERSIONE DA ESADECIMALE A BINARIO

Proviamo per esempio a convertire 4AF (anche scritto 0x4AF) in binario:

0100 1010 1111  
4 A F

### CONVERSIONE DA ESADECIMALE A DECIMALE

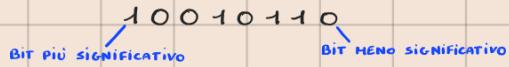
Per la conversione da esadecimale a decimale invece proviamo a convertire sempre 4AF<sub>16</sub> in decimale:

$$4AF = 4 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 = 1199_{10}$$

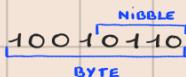
# BITS, BYTES, NIBBLE...

Un **BIT** è una singola unità di un numero binario ad esempio 0 o 1, invece un **BYTE** sono 8 bit ed un **NIBBLE** invece sono 4 bit (quindi 2 nibble formano un byte).

## ESEMPIO DI BITS:



## ESEMPIO DI BYTES E NIBBLE:



## ESEMPIO DI BYTE:



# GRANDI POTENZE DI 2

Esempi di grandi potenze di 2 sono:

$$2^{10} \Rightarrow 1 \text{ Kilo} \approx 1000(1024)$$

•  $2^{20} \Rightarrow 1 \text{ mega} \stackrel{?}{=} 1 \text{ milione (1048576)}$

$\bullet 2^{30} \Rightarrow 1 \text{ pipa} \equiv 1 \text{ miliardo} (10^9)$

# OPERAZIONI CON I NUMERI BINARI

## • ADDIZIONE

Per eseguire la somma tra 2 o più numeri binari la prima cosa è incolumnarli e una volta incolumnati si esegue **LA SOMMA DEI NUMERI BINARI** ricordando che:

$$0+0=0 \quad 0+1=1$$

$$1+0=1 \quad 1+1=0 \text{ CON RIPORTO DI } +$$

### ESEMPIO:

$$\begin{array}{r} & 1 \\ 1 & 0 & 1 & 0 & 0 & + \\ \hline & 1 & 1 & 1 & = \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}$$

### **SOTTRAZIONE**

Per eseguire correttamente la **DIFERENZA TRA DUE NUMERI BINARI**, bisogna innanzitutto incollarli così come visto nell'addizione e poi ricordare che:

$$0+0+0 = 1+1+1$$

$1 = 8 = 1$   $8 - 1 = 1$  dopo essersi prestato di Paula (colonna a sinistra)

ESEMPIO:

$$\begin{array}{r} \text{* } 0 0 1 0 - \\ 1 1 0 1 = \\ \hline 0 0 1 0 1 \end{array}$$

### •MOLTIPLICAZIONE

Per eseguire la **MOLTIPLICAZIONE TRA DUE NUMERI BINARI** bisogna immanzitutto tener presente che:

$$\begin{array}{l} 0 \cdot 1 = 0 \quad 0 \cdot 0 = 0 \\ 1 \cdot 0 = 0 \quad 1 \cdot 1 = 1 \end{array}$$

ESEMPIO:

$$\begin{array}{r} 1 1 0 \times \\ 1 0 = \\ \hline 0 0 0 + \Rightarrow 1 1 0 \cdot 0 \\ 1 1 0 \Rightarrow \Rightarrow 1 1 0 \cdot 1 \\ 1 1 0 0 \end{array}$$

### •DIVISIONE

La **DIVISIONE TRA NUMERI BINARI** avviene in questo modo:

1) Si prende nel dividendo un gruppo di cifre tali da avere un numero maggiore o uguale al divisore.

$$\begin{array}{r} \overline{1 0 0 1 0} \quad | 1 0 \\ \hline \end{array}$$

2) Poiché un numero non può iniziare con 0 la prima cifra sarà necessariamente 1, riporteremo quindi 1 come prima cifra del risultato;

$$\begin{array}{r} \overline{1 0 0 1 0} \quad | 1 0 \\ \hline 1 \\ \hline \end{array}$$

3) Si moltiplica il divisore per 1;

$$\begin{array}{r} \overline{1 0 0 1 0} \quad | 1 0 \\ 1 0 \quad | \\ \hline \end{array}$$

4) Si prosegue come fatto nei punti precedenti:

$$\begin{array}{r} \overline{10010} \\ -10 \\ \hline = 0 \\ \overline{-0} \\ = 1 \\ \overline{10} \\ = = \end{array}$$

## OVERFLOW

Poiché i sistemi digitali operano con un numero fisso di Bit, è necessario considerare quei casi in cui, durante un'addizione o una moltiplicazione, il numero di Bit sia insufficiente a poter rappresentare il risultato.

Questo problema prende il nome di **OVERFLOW** (ossia strabordare)

ESEMPIO:

$$\begin{array}{r} 101 \times \\ 111 = \\ \hline 101 + \\ 101 \ll + \\ \hline 100011 \end{array}$$

In questo caso i due moltiplicandi utilizzano entrambi 3 bit ma il loro prodotto ne richiede almeno 6. In questi casi i bit in eccesso vengono **SCARICATI** generando un risultato sbagliato perché togliendo l'eccesso verrebbe  $101 \cdot 111 = 011$ , dunque  $5 \cdot 7 = 3$ .

## SHIFT DI BIT A DESTRA O A SINISTRA

Un operatore binario aggiuntivo rispetto alla normale aritmetica decimale è l'operazione di **SHIFT** (o spostamento).

Lo shift è un operatore che prevede lo spostamento a **DESTRA O SINISTRA** di una certa quantità di caselle di tutti i bit del numero binario sul quale viene applicato:

$$000101_2 \ll 2 = 010100_2$$

$$01000_2 \gg 2 = 000110_2$$

## NUMERI BINARI NEGATIVI

Fino ad ora, abbiamo parlato di numeri binari positivi. Per comprendere ulteriori operazioni aritmetiche è necessario introdurre il **SEGNO** nei numeri binari.

Esistono 2 diversi metodi di rappresentare i numeri negativi e sono:

- Sign/Magnitude

- Complemento a 2

## • NUMERI IN SIGN/MAGNITUDE

Nel primo metodo di rappresentazione la differenza è poca:

- il Bit più significativo rappresenta il **SEGN**o;

- mentre tutti gli altri rappresentano il suo valore (o **MAGNITUD**e).

Nel caso in cui il Bit del segno assume valore **0** allora il numero sarà positivo mentre se è **1** allora il numero sarà negativo:

$$\begin{array}{c} \textcircled{0} \quad 10111_2 = +23 \\ \text{SEGN POSITIVO} \qquad \text{MAGNITUD} \end{array}$$

$$\begin{array}{c} \textcircled{1} \quad 01011_2 = -11 \\ \text{SEGN POSITIVO} \qquad \text{MAGNITUD} \end{array}$$

Quindi dati  $m$  bit nei 2 metodi di rappresentazione abbiamo i seguenti intervalli di valori disponibili:

$$\begin{array}{ll} [0, 2^m - 1] & [-(2^{m-1} - 1), 2^{m-1} - 1] \\ \text{SENZA SEGN} & \text{SIGN/MAGNITUD} \end{array}$$

Nonostante ciò, anche questo formato ha delle problematiche:

1) LE ADDIZIONI NON FUNZIONANO.

2) ABBINNO 2 MODI PER RAPPRESENTARE LO 0 (1000 E 0000, CORRISPONDENTI A -0 E +0)

ESEMPIO:

$$\begin{array}{r} 1110 + \\ 0110 = \\ 10100 \end{array}$$

In questa addizione per esempio c'è un errore perché anche se scattassimo l'overflow avremmo comunque  $-6 + 6 = +4$ .

## • NUMERI IN COMPLEMENTO A 2

Per risolvere questo problema è stata introdotta il sistema binario basato sul **COMPLEMENTO A 2**, che va ad aggiungere un ulteriore significato al Bit più significativo.

In questo sistema infatti assume il suo valore effettivo ma a differenza del sistema senza segno, il valore assoluto sarà **NEGATIVO**.

Dati i bit, il Bit più significativo assumerà il valore  $-(2^{n-1})$ , mentre tutti gli altri bit manderanno il valore positivo:

$$10111_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + (-1 \cdot 2^0) = -9_{10}$$

Per poter calcolare il complemento a 2 di un numero decimale negativo, invece, è prima necessario calcolare il **COMPLEMENTO A 1** del suo corrispettivo numero binario positivo.

Con complemento a 1 si intende semplicemente invertire tutti i bit, trasformando quindi tutti i suoi 0 in 1 e viceversa:

$$-25_{10} \Rightarrow +25_{10} = 011001_2 \Rightarrow 100110_2$$

Una volta calcolato il prossimo passo sarà semplicemente sommare +1 al risultato del complemento a 2:

$$100110 + 1 = 100111_2$$

Per confermare che la conversione sia avvenuta correttamente, possiamo calcolare il valore del complemento a 2 quindi:

$$\begin{array}{r} 543^2+0 \\ 100111_2 = -1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + (-1 \cdot 2^5) = -25_{10} \end{array}$$

L'utilità del complemento a 2 è evidente, perché grazie al modo in cui viene rappresentato **È POSSIBILE SVOLGERE OPERAZIONI TRA NUMERI** di segno opposto senza problematiche.

VIENE ANCHE RISOLTO IL PROBLEMA DEI 2 ZERI: 0000 corrisponderà a 0 e 1000 a -8.

$$+6 = 0110_2$$

$$-6 = 1010_2$$

Infatti:

$$\begin{array}{r} 1010 + \\ 0110 = \\ \hline 10000 \end{array}$$

Scartando l'overflow otteniamo  $+6 - 6 = 0$ , quindi il risultato è corretto.

Il range di valori negativi rappresentabili in complemento a 2 sarà:

$$\boxed{[-(2^{m-1}), 2^{m-1} - 1]}$$

COMPLEMENTO A 2

## INCREMENTARE IL NUMERO DI BIT

In alcuni casi nel complemento a 2 può essere necessario incrementare il numero di bit utilizzati, per far questo avremmo bisogno di tenere conto del

**SEGNONE** del numero rappresentato:

ESEMPIO:

Il numero  $010011_2$  può essere allungato aggiungendo la quantità di 0 desiderata:

$$01011_2 \Rightarrow \textcolor{violet}{0000}1011_2$$

questo è possibile perché è un numero **POSITIVO**, se invece avessimo un numero **NEGATIVO** dobbiamo aggiungerne 1 e no 0.

ESEMPIO:

Il numero  $110011_2$  può essere allungato aggiungendo la quantità di 1 desiderata:

$$10011_2 \Rightarrow \textcolor{violet}{111}110011_2$$

## LO SHIFT COME MOLTIPLICAZIONE O DIVISIONE

Lo spostamento di bit nel caso del complemento a 2 può essere utilizzato come **MOLTIPLICAZIONE E DIVISIONE**, ma solo se il moltiplicatore o divisore sia una potenza

$$B \cdot 2^m = B \ll m$$

$$B / 2^m = B \gg m$$

ESEMPI:

$$\bullet 00001111_2 = 00100 \Rightarrow 1 \cdot 2^4 = 16 \text{ (com 4 mi muovo di 2 posti)}$$

$$\bullet 11101111_2 = 11010 \Rightarrow -3 \cdot 2^4 = -48 \text{ (com 2 mi muovo di 1 posto)}$$

$$\bullet 01000000_2 = 00010 \Rightarrow 8 \cdot 2^4 = 8$$

$$\bullet 10000000_2 = 11000 \Rightarrow -16 \cdot 2^4 = -16$$

## NUMERI BINARI RAZIONALI

Come per il sistema decimale, anche nel sistema binario è possibile rappresentare i numeri razionali.

Esistono 2 tipi di rappresentazione utilizzabili:

• VIRGOLA FISSA (FIXED-POINT NUMBERS)

• VIRGOLA MOBILE (FLOATING-POINT NUMBERS)

### • FIXED-POINT NUMBERS

Questo sistema è molto simile alla classica rappresentazione dei numeri razionali nel sistema decimale.

Il modo in cui i numeri razionali vengono rappresentati corrisponde alla seguente somma:

$$134.56_{10} = 1 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}$$

In binario invece sarà:

$$110.11_{2} = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 6.75_{10}$$

Per poter convertire un numero decimale razionale in un numero binario razionale, è necessario scomporre il processo in 2 fasi:

1) conversione in binario della **PARTE INTEGRALE** del numero decimale;

2) conversione in binario della **PARTE FRAZIONARIA** del numero decimale.

La prima fase siamo già in grado di svilupparla tramite le metodologie imparate fino ad ora.

Per la seconda invece bisognerà usare un nuovo metodo:

1) moltiplicare la parte razionale per 2:

a) SE il risultato ottenuto è **UGUALE AD 1**, passa al punto 2;

b) SE È **MAGGIOR DI 1** allora sarà necessario sottrarre 1 al risultato e poi ripeti il punto 1 utilizzando l'attuale risultato come parte frazionaria;

c) SE IL risultato è **MINORE DI 1** ripeti il punto 1 utilizzando l'attuale risultato parziale come parte frazionaria.

2) una volta che la parte frazionaria non è più presente nel risultato parziale sarà necessario scrivere partendo dall'alto tutte le **PARTI INTEGRE**

DEI RISULTATI PARZIALI OTENUTI.

ESEMPIO:

Proviamo usando il numero  $17.625_{10}$ :

• la parte intera sarà  $-17_{10} = 10001_2$

• per la parte razionale dovremmo fare:

$$0.625 \cdot 2 = -1.25 (-1.25 - 1 = 0.25)$$

$$0.25 \cdot 2 = 0.5$$

$$0.5 \cdot 2 = -1$$

Una volta raggiunto l'1, consideriamo partendo dall'alto tutte le parti intere dei risultati, quindi:

$$-101_2$$

• alla fine uniremo la parte intera binaria a quella razionale:

$$-17.625_{10} = -10001 \cdot 101_2$$

## FLOATING-POINT NUMBERS

I numeri a virgola mobile sono l'equivalente binario della notazione scientifica dei numeri decimali.

I numeri in notazione scientifica vengono rappresentati nella seguente forma:



Lavorando con definite quantità di bit, solitamente i numeri a virgola mobile vengono rappresentati utilizzando 32 bit dove:

• 1° BIT RAPPRESENTA IL SEGNO

• I SUCCESSIVI 8 BIT RAPPRESENTANO L'ESPOENTE

• I RESTANTI 23 BIT



Per convertire un numero decimale nella sua forma a virgola mobile, è necessario eseguire questi passaggi:

1) convertire il numero in un binario fixed-point unsigned, visto che il segno sarà rappresentato dal 1° bit;

2) riscrivere il numero in "notazione scientifica binaria" portando la virgola fino ad una sola unità;

3) riscrivere il segno, la mantissa e l'esponente nei loro campi equivalenti della rappresentazione floating-point.

ESEMPPIO:

Usando  $-17.625_{10}$ , che è stato convertito in fixed-point nell'esempio precedente, convertiamolo in floating-point.

$$-17.625_{10} = -10001 \cdot 101_2$$

$$\frac{10001 \cdot 101_2}{2^8} = -1.0001101_2 \cdot 2^8$$

Successivamente convertire i 3 campi necessari in binario per poi riscrivere nei campi rispettivi:

**RISERVA:** O perché il numero è positivo;

**RISPOSTA:**  $-100_2$  che sarebbe 4 rappresentato in binario;

**MANTISSA:**  $1000\text{-}110\text{-}1$



Guardando attentamente il secondo passaggio possiamo notare come qualsiasi numero venga rappresentato in notazione scientifica binaria abbia un '-' come unità a cui viene attribuita la virgola:

$$+17.625_{10} = -1000\text{-}1.101_2 = 1.000\text{-}1101_2 \cdot 2^4$$

$$-58.25_{10} = 1110\text{-}10.01_2 = 1.110\text{-}1001_2 \cdot 2^5$$

$$+228_{10} = 11100\text{-}100_2 = 1.1100\text{-}100_2 \cdot 2^7$$

Per convenzione, quindi, nel momento in cui andiamo a riempire i rispettivi campi del floating-point possiamo direttamente ignorare l'-' prima della virgola, andando a guadagnare un bit aggiuntivo da poter utilizzare per rappresentare le cifre dopo la virgola:

$$17.625_{10} = -1000\text{-}1.101_2 = 1.000\text{-}1101_2 \cdot 2^4$$

$0 \quad 00000100 \quad 1000110100000000000000 \Rightarrow$  SENZA IGNORARE L'-' DAVANTI LA VIRGOLA

$0 \quad 00000100 \quad 0001101000000000000000 \Rightarrow$  IGNORANDO L'-' DAVANTI LA VIRGOLA

Questo è un passaggio estremamente utile, nel caso in cui dovessimo rappresentare in float un numero in notazione scientifica binaria con 23 bit dopo la virgola, saremmo costretti a dover togliere il bit meno significativo, poiché non possiamo inserire 24 bit nello spazio di 23 predisposto dalla mantissa.

## LO STANDARD IEEE 754

Dovendo abbiamo guadagnato un bit aggiuntivo da poter usare per la precisione della mantissa, resta solo da scoprire **COME RAPPRESENTARE I NUMERI IN NOTAZIONE SCIENTIFICA BINARIA CON ESPONENTE NEGATIVO**.

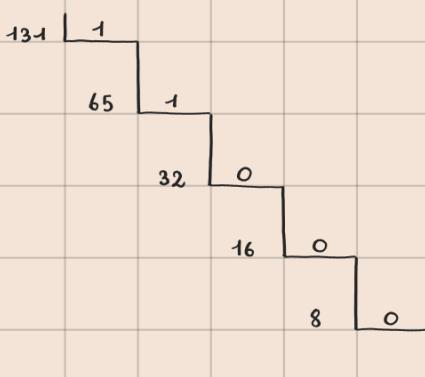
Lo standard IEEE754 propone una soluzione semplice: aggiungiamo un **BIAS DI -127 ALL'ESPONENTE**.

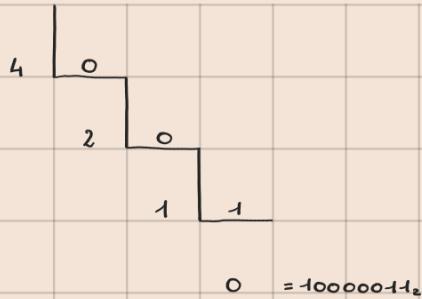
$$17.625_{10} = -1000\text{-}1.101_2 = 1.000\text{-}1101_2 \cdot 2^4$$

Una volta portato il numero in questa forma, aggiungiamo -127 al suo esponente, per poi convertirlo nel formato floating-point:

$$1.000\text{-}1101_2 \cdot 2^{(4+127)} = 1.000\text{-}1101_2 \cdot 2^{131}$$

Convertito -131 in binario





quindi:

0	10000011	00011010000000000000
		13-1

Grazie al bias -127 possiamo anche scrivere i numeri con un esponente negativo fino a -127. Tuttavia, perdiamo anche l'uso di una parte di esponenti positivi, poiché il valore massimo rappresentabile in 8 bit è 255.

## CASI SPECIALI, TIPI DI PRECISIONE E RAPPRESENTAZIONE IN ESADECIMALE

Nel caso dei numeri a virgola mobile, sono stati stabiliti dei casi speciali per poter rappresentare dei valori particolari e non esprimibili numericamente:

NUMERO EQUIVALENTE	SEGNONE	ESPOENTE	MANTISSA
0	1 o 0	00000000	00000000000000000000000000000000
$\infty$	0	11111111	00000000000000000000000000000000
$-\infty$	1	11111111	00000000000000000000000000000000
NaN	1 o 0	00000000	diverso da 0
Num. Decimali	1 o 0	00000000	$(-1)^s \cdot 2^{-126} \cdot 0.m$ con $m \neq 0$

Inoltre esistono vari tipi di precisione possibili per i numeri float in base al numero di bit utilizzati per rappresentarli:

TIPOLOGIA	BIT TOT.	SEGNONE	ESPOENTE	MANTISSA	BIAS
Half-precision	16 bit	1 bit	5 bit	10 bit	15
Single-precision	32 bit	1 bit	8 bit	23 bit	127
Double-precision	64 bit	1 bit	11 bit	52 bit	1023

L'utilità di avere più tipi di precisione è per prevenire la possibilità di un overflow o un underflow (ossia quando il numero è troppo piccolo per essere rappresentato) poiché altimenti sarebbe necessario arrotondare il valore, perdendo una parte di precisione.

Un altro modo per rappresentare dei numeri float è tramite l'uso dei numeri decimali.

Poiché 32 bit corrispondono 8 gruppi da 4 bit, possiamo riscrivere ogni numero float come **8 NUMERI ESADECIMALI**.

ESEMPIO:

$$17.625_{10} = 0.1000\underset{4}{\underline{0}}.0001\underset{4}{\underline{0}}0000\underset{4}{\underline{1}}0000\underset{4}{\underline{0}}0000\underset{4}{\underline{0}}0000\underset{4}{\underline{0}}0000_{16} \text{ float} \Rightarrow 0x44800000$$

# OPERAZIONI FRA NUMERI FLOATING-POINT

Per sommare due numeri in rappresentazione floating-point il processo necessario è molto semplice:

- 1) rappresentare nuovamente i 2 numeri in forma di **NOTAZIONE SCIENTIFICA BINARIA**;
- 2) portare entrambi i numeri allo stesso esponente;
- 3) sommare le due mantisse, normalizzandole se necessario;
- 4) aggiustare l'esponente e riscrivere in forma floating-point, arrotondando in caso di overflow o underflow.

ESEMPIO:

Somma i due numeri float 0x3FC00000 e 0x40500000

1)

$0x3FC00000 = \underbrace{0\ 0\ 11\ 1111\ 100\ 0000\ 0000000000000000}_{\text{SEGNONE}} \underbrace{000000000000000000000000}_{\text{MANTISSA}}$  float = -1.1

SEGNONE = 0    ESPONENTE = 127    MANTISSA = .1

Successivamente anteponi 1 quindi 1.1

$0x40500000 = \underbrace{0\ 10000000\ 1010000000000000000000}_{\text{SEGNONE}} \underbrace{000000000000000000000000}_{\text{MANTISSA}}$  float =  $-1 \cdot 10^1 \cdot 2^4$

SEGNONE = 0    ESPONENTE = 128    MANTISSA = -10.1

Successivamente anteponi 1 quindi -1.10.1

2)

$127 - 128 = -1$ , quindi si sposta di -1 la virgola

$$-1.101 = 0.11$$

3)

$$0.110 +$$

$$\underline{1.101} =$$

$$10.011$$

4)

$$10.011 \cdot 2^4 \Rightarrow 1.0011 \cdot 2^5$$

0	10000001	001100000000000000000000
---	----------	--------------------------

che in esadecimale sarà: 0x40980000

Per moltiplicare due numeri float, invece, il processo necessario è leggermente diverso:

- 1) rappresentare nuovamente i due numeri in forma di notazione scientifica binaria
- 2) sommare i due esponenti
- 3) moltiplicare le due mantisse, normalizzandole se necessario
- 4) aggiustare l'esponente e riscrivere in forma floating point, avviolandolo in caso di overflow o underflow

ESEMPIO:

Calcola il prodotto dei due numeri float 0x3FC00000 e 0x40500000

1)

$$0x3FC00000 = 0.011111100000000000000000 = 1.1 \cdot 2^0$$

2)

$$0x40500000 = 0.100000001010000000000000 = 1.101 \cdot 2^1$$

3)

$$1.101 \cdot 2^1 \cdot 1.1 \cdot 2^0 = 10.0111 \cdot 2^1$$

$$\begin{array}{r} 1.101 \times \\ 1.1 \\ \hline 1.101 + \\ 1.101 \swarrow = \\ 10.0111 \end{array}$$

4)

$$10.0111 \cdot 2^1 \rightarrow 1.00111 \cdot 2^2 = \underbrace{0}_{4} \underbrace{100}_{0} \underbrace{0000}_{9} \underbrace{1001}_{c} \underbrace{1100}_{c} \underbrace{0000}_{0} \underbrace{0000}_{0} \underbrace{0000}_{0} \underbrace{0000}_{0}$$