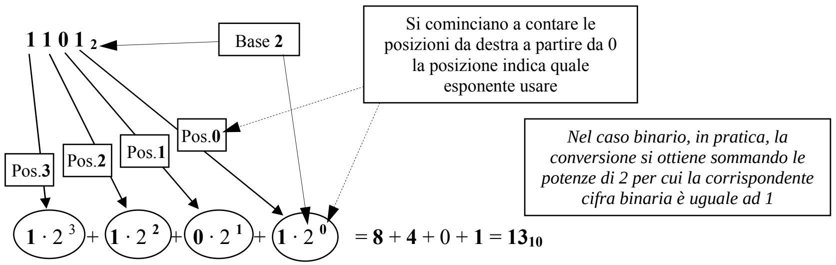
Esercitazione del 03/03/2005 - Soluzioni

1. **Conversione binario → decimale**
   (*Rappresentazione dell'Informazione – Conversione da base n a base 10 , slide 10)*

   a. **$1101_2 \rightarrow ?_{10}$**



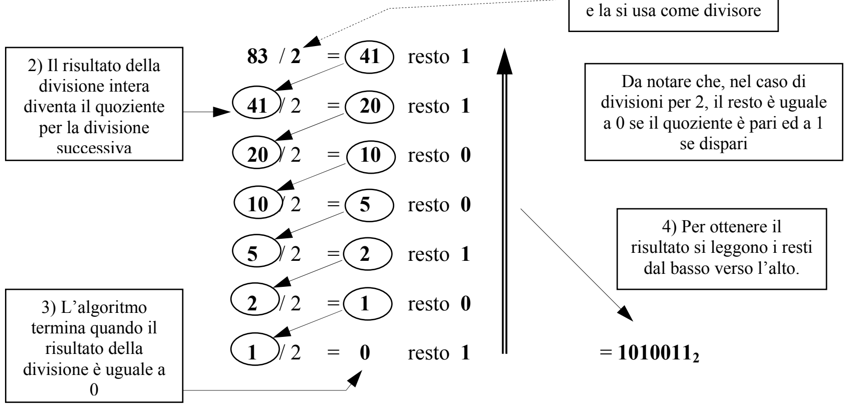
   b. **10101010_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 128 + 32 + 8 + 2 = \mathbf{170_{10}}$

   c. **1000010_2** $= \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 64 + 2 = \mathbf{66_{10}}$

   d. **101100101_2** $= \mathbf{1} \cdot 2^8 + \mathbf{0} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 256 + 64 + 32 + 4 + 1 = \mathbf{357_{10}}$

2. **Conversione decimale → binario**
   (*Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 14)*

   a. $83_{10} \rightarrow ?_2$



   b. $478_{10} \rightarrow ?_2$

478	**/ 2 =**	**239**	resto **0**
239	**/ 2 =**	**119**	resto **1**
119	**/ 2 =**	**59**	resto **1**
59	**/ 2 =**	**29**	resto **1**
29	**/ 2 =**	**14**	resto **1**
14	**/ 2 =**	**7**	resto **0**
7	**/ 2 =**	**3**	resto **1**
3	**/ 2 =**	**1**	resto **1**
1	**/ 2 =**	**0**	resto **1**

   $478_{10} = 111011110_2$

   c. $2471_{10} \rightarrow ?_2$

2471	**/ 2 =**	**1235**	resto **1**
1235	**/ 2 =**	**617**	resto **1**
617	**/ 2 =**	**308**	resto **1**
308	**/ 2 =**	**154**	resto **0**
154	**/ 2 =**	**77**	resto **0**
77	**/ 2 =**	**38**	resto **1**
38	**/ 2 =**	**19**	resto **0**
19	**/ 2 =**	**9**	resto **1**
9	**/ 2 =**	**4**	resto **1**
4	**/ 2 =**	**2**	resto **0**
2	**/ 2 =**	**1**	resto **0**
1	**/ 2 =**	**0**	resto **1**

   $2471_{10} = 100110100111_2$

   d. $14123_{10} \rightarrow ?_2$

14123	**/ 2 =**	**7061**	resto **1**
7061	**/ 2 =**	**3530**	resto **1**
3530	**/ 2 =**	**1765**	resto **0**
1765	**/ 2 =**	**882**	resto **1**
882	**/ 2 =**	**441**	resto **0**
441	**/ 2 =**	**220**	resto **1**
220	**/ 2 =**	**110**	resto **0**
110	**/ 2 =**	**55**	resto **0**
55	**/ 2 =**	**22**	resto **1**
27	**/ 2 =**	**13**	resto **1**
13	**/ 2 =**	**6**	resto **1**
6	**/ 2 =**	**3**	resto **0**
3	**/ 2 =**	**1**	resto **1**
1	**/ 2 =**	**0**	resto **1**

   $2471_{10} = 11011100101011_2$

3. **Conversione binario → esadecimale**
(*Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 18)*

a. $110101_2 \rightarrow ?_{16}$

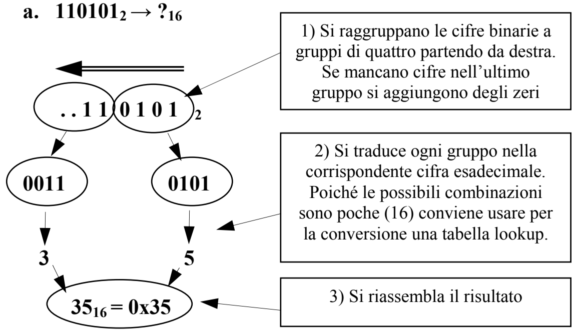


Tabella lookup

0000 ↔ **0**	1000 ↔ **8**
0001 ↔ **1**	1001 ↔ **9**
0010 ↔ **2**	1010 ↔ **A**
0011 ↔ **3**	1011 ↔ **B**
0100 ↔ **4**	1100 ↔ **C**
0101 ↔ **5**	1101 ↔ **D**
0110 ↔ **6**	1110 ↔ **E**
0111 ↔ **7**	1111 ↔ **F**

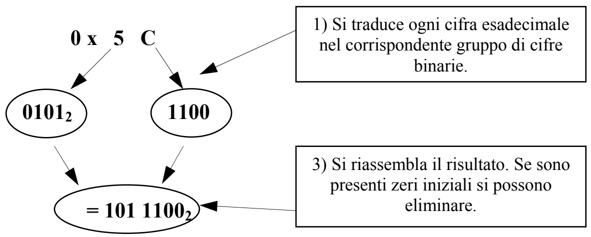
b. $\mathbf{10010011_2}$ = 1001_2 | 0011_2 = 0x9 | 0x3 = **0x93**

c. $\mathbf{110111000000_2}$ = 1101_2 | 1100_2 | 0000_2 = 0xD | 0xC | 0x0 = **0xDC0**

d. $\mathbf{1101011101_2}$ = 0011_2 | 0101_2 | 1101_2 = 0x3 | 0x5 | 0xD = **0x35D**

4. **Conversione esadecimale → binario**
(*Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 17)*

a. $0x5C \rightarrow ?_2$



b. 0xF03 = 0xF | 0x0 | 0x3 = 1111_2 | 0000_2 | 0011_2 = $\mathbf{111100000011_2}$
c. 0x16C = 0x1 | 0x3 | 0xC = 0001_2 | 0110_2 | 1100_2 = $\mathbf{101101100_2}$
d. 0x85A1 = 0x8 | 0x5 | 0xA | 0x1 = 1000_2 | 0101_2 | 1010_2 | 0001_2 = $\mathbf{1000010110100001_2}$

5. **Somme binarie**
(*Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 20)*

a. $100101_2 + 101_2 = ?_2$

Riporti

		1		*1*		*R*
1	**0**	**0**	**1**	**0**	**1**	+
			1	**0**	**1**	=
1	**0**	**1**	**0**	**1**	**0**	

$100101_2 + 101_2 = 101010_2$

La somma binaria si esegue esattamente come quella decimale classica. Si allineano a destra i numeri binari da sommare quindi si procede a sommare ogni coppia di bit corrispondente e l'eventuale riporto, da destra verso sinistra.

In base 2, occorre ricordarsi che:
1 + 1 = 0 riporto 1
1 + 1 + 1= 1 riporto 1

b. $11010111_2 + 1001011_2 = 100100010_2$

	1		*1*	*1*	*1*	*1*	*1*		*R*
	1	**1**	**0**	**1**	**0**	**1**	**1**	**1**	+
		1	**0**	**0**	**1**	**0**	**1**	**1**	=
1	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	

c. $1101_2 + 101100101_2 = 101110010_2$

				1	*1*		*1*		*R*
					1	**1**	**0**	**1**	+
1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	=
1	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	

d. $100101110_2 + 111000011_2 = 1011110001_2$

1					*1*	*1*	*1*			*R*
	1	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	+
	1	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	=
1	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	

6. Sottrazioni binarie

(Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 21-23)

a. $1001_2 – 110_2 = ?_2$

COMPLEMENTO A DUE

Completamento

$1001_2 – 110_2$

$0\,1001_2 \; – \; 0\,0110_2$

$1\,1001_2$

Bit di segno

1) Estendo le cifre alla rappresentazione scelta se necessario

2) Calcolo il complemento a due del secondo termine invertendo i bit e sommando uno

La sottrazione in **complemento a due** si esegue sommando al primo termine il complemento a due del secondo termine. Il **complemento a due** si calcola invertendo le cifre bit a bit e quindi sommando 1. I calcoli si eseguono sul numero di bit della rappresentazione richiesta, o, se non è data nessuna dimensione, sul numero di cifre del più grande dei due. Se uno dei due termini risulta più corto allora occorre estendere il segno fino alla lunghezza necessaria.

S			*1*		*R*	
1	**1**	**0**	**0**	**1**	+	
0	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**1**	¹**0**	+	→ **-110_2** in CA2

$01001_2 + 11010_2$

3) Sommo il primo termine con il complemento a due del secondo.

1	*1*					*R*	
	0	**1**	**0**	**0**	**1**	+	
	1	**1**	**0**	**1**	**0**	=	
~~1~~	¹**0**	¹**0**	**0**	**1**	**1**	+	→ **$+11_2$**

Il riporto oltre il bit di segno viene scartato

Risultato: **$1001_2 – 110_2 = +11_2$**

b. $111_2 – 1101_2 = ?_2$

Uso quattro cifre più il bit di segno:

$111_2 – 1101_2 = 0\,0111_2 – 0\,1101_2 = 00111_2 + (10010_2 +1) = -110_2$

Calcolo il CA2 di **$–1101_2$**:

S					*R*	
1	**0**	**0**	**1**	**0**	+	
0	**0**	**0**	**0**	**1**	=	
1	**0**	**0**	**1**	**1**		→ **-1101_2** in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

S	*1*	*1*	*1*		*R*	
0	**0**	**1**	**1**	**1**	+	
1	**0**	**0**	**1**	**1**	=	
1	**1**	¹**0**	¹**1**	¹**0**		→ **-110_2** in CA2

Risultato: **$111_2 – 1101_2 = -110_2$**

c. $11101_2 – 1011_2 = ?_2$

Uso cinque cifre più il bit di segno:

$11101_2 – 1011_2 = 0\,11101_2 – 0\,01011_2 = 011101_2 + (110100_2 + 1)$

Calcolo il CA2 di **$–1011_2$**:

S						*R*	
1	**1**	**0**	**1**	**0**	**0**	+	
0	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**1**	**0**	**1**		→ **-1011_2** in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

1	*1*	*1*	*1*		*1*		*R*	
	0	**1**	**1**	**1**	**0**	**1**	+	
	1	**1**	**0**	**1**	**0**	**1**	=	
~~1~~	¹**0**	¹**1**	¹**0**	¹**0**	**1**	¹**0**		→ **$+10010_2$**

Risultato: **$11101_2 – 1011_2 = +10010_2$**

d. $\mathbf{1011_2 - 10101_2 = ?_2}$ (*Eseguire i calcoli a 8 bit, segno compreso)*

Uso sette cifre più il bit di segno:

$\mathbf{1011_2 - 10101_2 = 0\ 0001011_2 - 0\ 0010101_2 = 00001011_2 + (\ 11101010_2 + 1_2\ )}$

S								*R*	
1	**1**	**1**	**0**	**1**	**0**	**1**	**0**	+	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**1**	**0**	**1**	**0**	**1**	**1**		→ $\mathbf{-10101_2}$ in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

S			*1*		*1*	*1*		*R*	
0	**0**	**0**	**0**	**1**	**0**	**1**	**1**	+	
1	**1**	**1**	**0**	**1**	**0**	**1**	**1**	=	
1	**1**	**1**	**1**	^{1}**0**	**1**	^{1}**1**	^{1}**0**		→ $\mathbf{-1010_2}$ in CA2

Risultato: $\mathbf{1011_2 - 10101_2 = -1010_2}$

Altri metodi per eseguire la sottrazione binaria (facoltativo)

MODULO e SEGNO

prendo ricorsivamente in prestito una *decina binaria* dalle cifre a sinistra in modo da eseguire la sottrazione $\mathbf{0_2 - 1_2}$

La sottrazione in **modulo e segno** si esegue in maniera simile alla sottrazione per i numeri naturali. Si esegue la sottrazione bit a bit da destra verso sinistra, tenendo conto di eventuali prestiti dalle cifre a sinistra quando necessario.
Il metodo non funziona se il risultato della sottrazione è negativo, cioè quando il primo termine è più piccolo del secondo. In questo caso occorre invertire i termini prima della sottrazione e aggiustare il segno alla fine del calcolo.

S					*P*
Prestiti	-	-			
	1	^{1}**0**	^{1}**0**	**1**	**-**
	0	**1**	**1**	**0**	**=**
0	**0**	**0**	**1**	**1**	

Nota: qui il risultato è 0 perchè una unità del prestito dalla cifra a sinistra è stata passata alla cifra a destra.

In base 2, occorre ricordarsi che:
0 - 1 = ^{1}**0 -1 = 1** *con un prestito dalla cifra a sinistra*.
Se la cifra a sinistra è 0 allora occorre estendere il prestito verso sinistra fino a trovare un 1.

COMPLEMENTO A UNO

$\mathbf{1001_2 - 110_2}$

$\mathbf{01001_2 - 00110_2}$

$\mathbf{01001_2 + 11001_2}$

Estendo le cifre se necessario

Calcolo il complemento a uno del secondo termine invertendo i bit

La sottrazione in **complemento a uno** si esegue sommando al primo termine il complemento a uno del secondo termine. **Se il risultato è positivo, occorre correggerlo sommando $\mathbf{1_2}$.**
I calcoli si eseguono sul numero di bit della rappresentazione richiesta, o, se non è data nessuna dimensione, sul numero di cifre del più grande dei due. Se uno dei due termini risulta più corto allora occorre estendere il segno fino alla lunghezza necessaria.

S			*1*		*R*
0	**1**	**0**	**0**	**1**	+
1	**1**	**0**	**0**	**1**	=
0	**0**	**0**	**1**	^{1}**0**	+
				1	
0	**0**	**0**	**1**	**1**	

Sommo i due termini. Il risultato è positivo quindi lo correggo sommando uno

e. $111_2 – 1101_2 = ?_2$

Modulo e Segno: il primo termine è più piccolo quindi inverto i termini, il risultato sarà negativo.

S						
	-	-			*P*	
	1	^{1}1	^{1}0	1	-	
	0	1	1	1	=	
1	0	1	1	0		→ **-110_2** in M&S

Complemento a UNO: uso quattro cifre più il bit di segno.

$111_2 – 1101_2 = 00111_2 – 01101_2 = 00111_2 + 10010_2$

S	*1*	*1*			*R*	
0	0	1	1	1	+	
1	0	0	1	0	=	
1	1	^{1}0	^{1}0	1		→ **-110_2** in CA1

Il risultato è negativo quindi non occorre correggerlo.

f. $11101_2 – 1011_2 = ?_2$

Modulo e Segno: il primo termine è più grande quindi i termini così ordinati, il risultato sarà positivo.

S							
	-		-			*P*	
	1	1	1	^{1}0	1	-	
	0	1	0	1	1	=	
0	1	0	0	1	0		→ **$+10010_2$**

Complemento a UNO: uso cinque cifre più il bit di segno.

$11101_2 – 1011_2 = 011101_2 – 001011_2 = 011101_2 + 110100_2$

1	*1*	*1*	*1*				*R*
	0	1	1	1	0	1	+
	1	1	0	1	0	0	=
~~1~~	^{1}0	^{1}1	^{1}0	^{1}0	0	1	

Il risultato è positivo quindi occorre correggerlo.

	S				*1*		*R*	
	0	1	0	0	0	1	+	
	0	0	0	0	0	1	=	
~~1~~	0	1	0	0	1	^{1}0		→ **$+10010_2$**

g. $1011_2 – 10101_2 = ?_2$

Modulo e Segno: il primo termine è più piccolo quindi inverto i termini, il risultato sarà negativo.

S							
	-		-			*P*	
	1	^{1}0	1	^{1}0	1	-	
	0	1	0	1	1	=	
1	0	1	0	1	0		→ **-1010_2** in M&S

Complemento a UNO: uso cinque cifre più il bit di segno.

$1011_2 – 10101_2 = 001011_2 – 010101_2 = 001011_2 + 101010_2$

S	*1*		*1*			*R*	
0	0	1	0	1	1	+	
1	0	1	0	1	0	=	
1	1	^{1}0	1	^{1}0	1		→ **-1010_2** in CA1

Il risultato è negativo quindi non occorre correggerlo.

7. Conversione in floating point secondo lo standard IEEE 754

(Rappresentazione dell'Informazione – Conversione da base 10 a base n , slide 29, 34-38)

a. $-20{,}75_{10}$ = <*s*,*e*,*m*>?

Per convertire in floating point occorre:
- calcolare il segno
- convertire la parte intera in binario (ex: con il metodo delle divisioni per 2 successive)
- convertire la parte frazionaria in binario (ex: con il metodo delle moltiplicazioni per 2 successive)
- unire i due risultati e normalizzare.
- calcolare la mantissa secondo la precisione voluta (occorre ricordarsi di scartare il primo 1 della normalizzazione)
- calcolare l'esponente della normalizzazione polarizzato e convertirlo in binario secondo la precisione voluta

Numero negativo: **s = 1**
Converto 40_{10} in base 2: $\mathbf{20_{10} = 10100_2}$

20	/ **2** =	**10**	resto	**0**
10	/ **2** =	**5**	resto	**0**
5	/ **2** =	**2**	resto	**1**
2	/ **2** =	**1**	resto	**0**
1	/ **2** =	**0**	resto	**1**

Converto $0{,}5_{10}$ in base 2: $\mathbf{0{,}75_{10} = 0{,}11_2}$

La conversione della parte frazionaria **per moltiplicazioni 2** successive si esegue nel seguente modo:
- si moltiplica per 2 la parte frazionaria. La parte intera del risultato costituisce il primo bit della parte frazionaria espressa in binario.
- Si ripete il passo precedente sulla parte frazionaria del risultato. La parte intera del risultato costituirà adesso il secondo bit della parte frazionaria espressa in binario.
- Si ripete il procedimento ricavando i successivi bit fino a che la parte frazionaria risulta uguale a zero (tutti i bit successivi saranno a zero) oppure si è raggiunta la precisione voluta (es. si sono ricavati già i 23 bit necessari per una mantissa in precisione singola).

0,75 *** 2** = **1,5** parte intera **1**
0,5 *** 2** = **1,0** parte intera **1**
#######

Leggo i bit dall'alto verso il basso. **Il bit più vicino alla virgola è il primo bit trovato.**

La parte frazionaria è uguale a zero. Il metodo termina.

Unisco i risultati: $\mathbf{20{,}75_{10} = 10100{,}11_2}$
Normalizzo: $\mathbf{10100{,}11_2 = 1{,}010011_2 \cdot (10_2)^4}$
Mantissa a 23 bits: $\mathbf{\cancel{1},010011_2 \rightarrow \mathit{m} = 0100110000\ 0000000000\ 000}$
Polarizzo l'esponente: $\mathbf{4_{10} + 127_{10} = 131_{10} = 128_{10} + 2_{10} + 1_{10} = 10000011_2}$
Esponente a 8 bits: $\mathbf{10000011_2 \rightarrow \mathit{e} = 10000011}$

$\mathbf{-20{,}75_{10}}$ **= <1, 10000011, 01001100000000000000000>**

b. $\mathbf{-0{,}25_{10}}$ **= <s,m,e>?**

Numero negativo: **s = 1**
Converto 0_{10} in base 2: $\mathbf{0_{10} = 0_2}$
Converto $0{,}75_{10}$ in base 2: $\mathbf{0{,}25_{10} = 0{,}01_2}$

0,25 *** 2** = **0,5** parte intera **0**
0,5 *** 2** = **1,0** parte intera **1**
#######

Unisco i risultati: $\mathbf{0{,}25_{10} = 0{,}01_2}$
Normalizzo: $\mathbf{0{,}01_2 = 1{,}0_2 \cdot (10_2)^{-2}}$
Mantissa a 23 bit: $\mathbf{\cancel{1},0_2 \rightarrow \mathit{m} = 0000000000\ 0000000000\ 000}$
Polarizzo l'esponente: $\mathbf{-2_{10} + 127_{10} = 125_{10} = 1111101_2}$

125	/ **2** =	**62**	resto	**1**
62	/ **2** =	**31**	resto	**0**
31	/ **2** =	**15**	resto	**1**
15	/ **2** =	**7**	resto	**1**
7	/ **2** =	**3**	resto	**1**
3	/ **2** =	**1**	resto	**1**
1	/ **2** =	**0**	resto	**1**

Esponente a 8 bits: $\mathbf{1111101_2 \rightarrow \mathit{e} = 01111101}$

$\mathbf{-0{,}25_{10}}$ **= <1, 01111101, 00000000000000000000000>**

c. $+10_{10} = <s,m,e>?$

Numero positivo: $\boldsymbol{s = 0}$
Converto 10_{10} in base 2: $\mathbf{10_{10} = 1010_2}$

10	/ **2** =	**5**	resto	**0**
5	/ **2** =	**2**	resto	**1**
2	/ **2** =	**1**	resto	**0**
1	/ **2** =	**0**	resto	**1**

Converto $0{,}0_{10}$ in base 2: $\mathbf{0{,}0_{10} = 0{,}0_2}$
Unisco i risultati: $\mathbf{10_{10} = 1010{,}0_2}$
Normalizzo: $\mathbf{1010{,}0_2 = 1{,}01_2 \cdot (10_2)^3}$
Mantissa a 23 bit: $\mathbf{\cancel{1}{,}01_2 \rightarrow \boldsymbol{m} = 0100000000\ 0000000000\ 000}$
Polarizzo l'esponente: $\mathbf{3_{10} + 127_{10} = 130_{10} = 10000010_2}$

130	/ **2** =	**65**	resto	**0**
65	/ **2** =	**32**	resto	**1**
32	/ **2** =	**16**	resto	**0**
16	/ **2** =	**8**	resto	**0**
8	/ **2** =	**4**	resto	**0**
4	/ **2** =	**2**	resto	**0**
2	/ **2** =	**1**	resto	0
1	/ **2** =	**0**	resto	**1**

Esponente a 8 bits: $\mathbf{10000010_2 \rightarrow \boldsymbol{e} = 10000010}$

$+10_{10}$ <0, 10000010, 01000000000000000000000>

d. $-1{,}7_{10} = <s,m,e>?$

Numero negativo: $\boldsymbol{s = 1}$
Converto 0_{10} in base 2: $\mathbf{1_{10} = 1_2}$
Converto $0{,}7_{10}$ in base 2: $\mathbf{0{,}25_{10} = 0{,}1\ \overline{0110}_2}$

0,7	* **2**	=	**1,4**	parte intera	**1**
0,4	* **2**	=	**0,8**	parte intera	**0**
0,8	* **2**	=	**1,6**	parte intera	**1**
0,6	* **2**	=	**1,2**	parte intera	**1**
0,2	* **2**	=	**0,4**	parte intera	**0**
#######					

Da qui in poi si ripetono ciclicamente le cifre **0110**

Unisco i risultati: $\mathbf{1{,}7_{10} = 1{,}1\ \overline{0110}_2}$
Normalizzo: $\mathbf{1{,}1\ 0110_2 = 1{,}1\ \overline{0110}_2 \cdot (10_2)^0}$
Mantissa a 23 bit: $\mathbf{\cancel{1}{,}\ 1\ 0110\ 0110\ 0110\ 0110\ 0110\ 01\cancel{10}\ \cancel{\overline{0110}}}$
$\mathbf{\rightarrow \boldsymbol{m} = 1011001100\ 1100110011\ 001}$
Polarizzo l'esponente: $\mathbf{0_{10} + 127_{10} = 1111111_2}$

Tronco la rappresentazione periodica della mantissa alla lunghezza fissa di 23 bit

127	/ **2** =	**63**	resto	**1**
63	/ **2** =	**31**	resto	**1**
31	/ **2** =	**15**	resto	**1**
15	/ **2** =	**7**	resto	**1**
7	/ **2** =	**3**	resto	**1**
3	/ **2** =	**1**	resto	**1**
1	/ **2** =	**0**	resto	**1**

Esponente a 8 bits: $\mathbf{1111101_2 \rightarrow \boldsymbol{e} = 01111111}$

$-1{,}7_{10}$ = <1, 01111111, 10110011001100110011001 >

Massimo Marchi – marchi@dsi.unimi.it

Problema dell'approssimazione di numeri piccoli: il formato IEEE 754 denormalizzato.

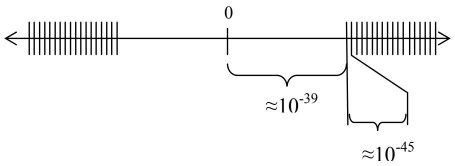
Il più piccolo numero positivo rappresentabile in formato IEEE 754 standard a precisione singola è:

$<0,00000001,00000000000000000000000> = 1,0_2 \cdot 2^{-126} \approx 1,17 * 10^{-38}$

Il più piccolo numero positivo successivo:

$$\begin{aligned}<0,00000001,00000000000000000000001> &= 1,00000000000000000000001_2 \cdot (2_{10})^{-126} \\ &= 2^{-126} + 2^{-149} \\ &\approx 1,17 \cdot 10^{-38} + 1,40 \cdot 10^{-45}\end{aligned}$$

Ciò significa che usando il formato IEEE 754 standard, in prossimità dello zero avremo un cambiamento nella linearità della quantizzazione: da **0** al più piccolo successivo ci sarà un salto dell'ordine di 10^{-38} mentre per passi successivi i salti saranno dell'ordine di 10^{-45}.



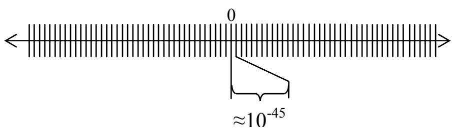
Il più piccolo numero positivo rappresentabile in formato IEEE 754 denormalizzato a precisione singola è:

$$\begin{aligned}<0,00000000,00000000000000000000001> &= 0,00000000000000000000001_2 \cdot 2^{-126} \\ &= 2^{-149} \approx 1,40 * 10^{-45}\end{aligned}$$

Il più piccolo numero positivo successivo:

$$\begin{aligned}<0,00000000,00000000000000000000010> &= 0,00000000000000000000010_2 \cdot 2^{-126} \\ &= 2 * 2^{-149} \\ &\approx 2,80 \cdot 10^{-45}\end{aligned}$$

Usando il formato IEEE 754 denormalizzato, in prossimità dello zero la quantizzazione risulta lineare: da **0** al più piccolo successivo ci sarà un salto dell'ordine di 10^{-45} così come per passi successivi. In aggiunta il salto tra 0 ed il primo positivo successivo risulta minore, e quindi migliore in termini di precisione ottenibile nei calcoli, rispetto al formato IEEE 754 standard.



Lo svantaggio maggiore del formato denormalizzato rispetto al formato standard è che richiede per l'esecuzione dei calcoli aritmetici di algoritmi più complicati.

Massimo Marchi – marchi@dsi.unimi.it