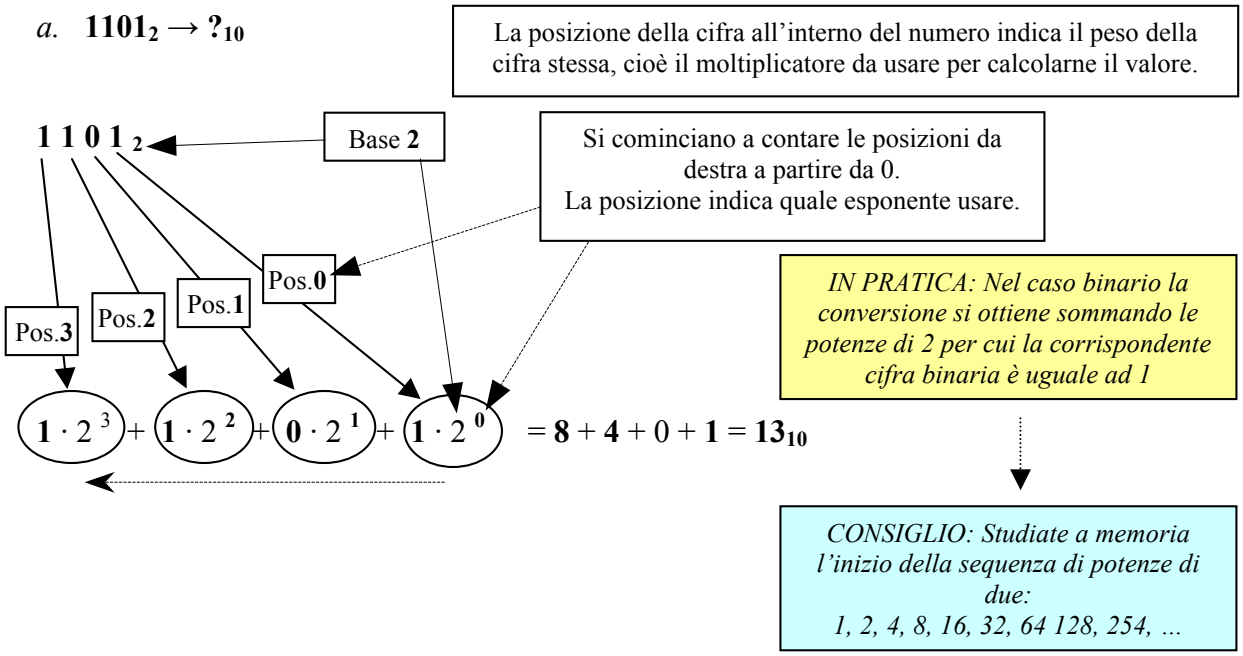
Laboratorio del 21/10/2010- Numeri binari e conversione

1. **Conversione binario → decimale**

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



   b. **10101101_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 128 + 32 + 8 + 4 + 1 = \mathbf{173_{10}}$

   c. **1101001_2** $= \mathbf{1} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 64 + 32 + 8 + 1 = \mathbf{105_{10}}$

   d. **10100101_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 128 + 32 + 4 + 1 = \mathbf{165_{10}}$

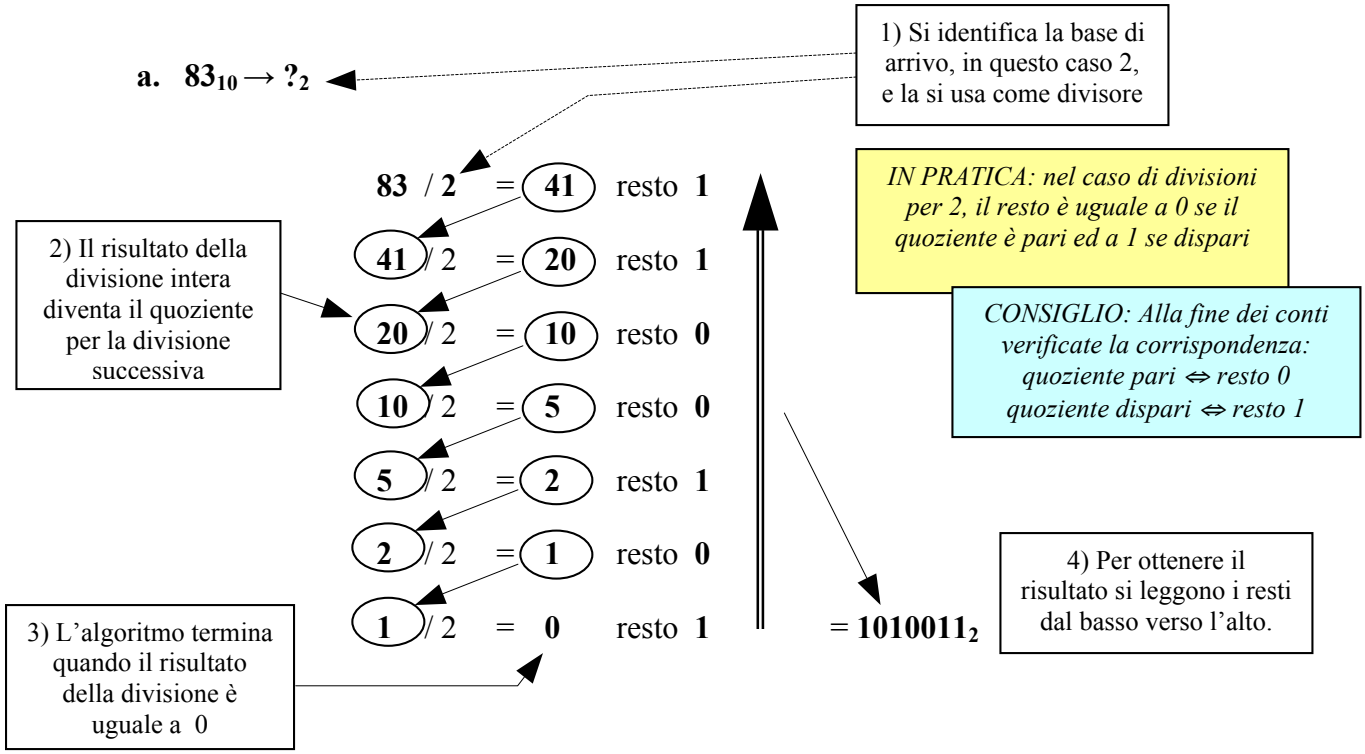
Nota:
n "1" = 111.....1 $= 2^n - 1$
Un "1" seguito da n "0" = 100.....0 $= 2^n$

Nota:
Lo stesso procedimento può essere adottato per passsare da esadecimale (base 16) a decimale.

Massimo Marchi – marchi@dsi.unimi.it

2. **Conversione decimale → binario**

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



b. $125_{10} \rightarrow ?_2$

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**

$125_{10} = 1111101_2$

c. $3184_{10} \rightarrow ?_2$

3184	**/ 2 =**	**1592**	resto	**0**
1592	**/ 2 =**	**796**	resto	**0**
796	**/ 2 =**	**398**	resto	**0**
398	**/ 2 =**	**199**	resto	**0**
199	**/ 2 =**	**99**	resto	**1**
99	**/ 2 =**	**49**	resto	**1**
49	**/ 2 =**	**24**	resto	**1**
24	**/ 2 =**	**12**	resto	**0**
12	**/ 2 =**	**6**	resto	**0**
6	**/ 2 =**	**3**	resto	**0**
3	**/ 2 =**	**1**	resto	**1**
1	**/ 2 =**	**0**	resto	**1**

$3184_{10} = 110001110000_2$

d. $7569_{10} \rightarrow ?_2$

7569	**/ 2 =**	**3784**	resto	**1**
3784	**/ 2 =**	**1892**	resto	**0**
1892	**/ 2 =**	**946**	resto	**0**
946	**/ 2 =**	**473**	resto	**0**
473	**/ 2 =**	**236**	resto	**1**
236	**/ 2 =**	**118**	resto	**0**
118	**/ 2 =**	**59**	resto	**0**
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**

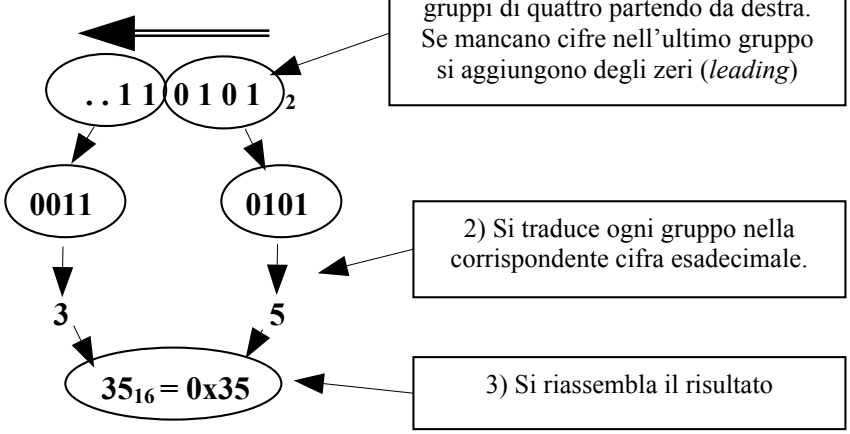
$7569_{10} = 1110110010001_2$

Nota:
Lo stesso procedimento può essere adottato per passsare da decimale a esadecimale (base 16).

Massimo Marchi – marchi@dsi.unimi.it

3. **Conversione binario → esadecimale**

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



CONSIGLIO:Poiché le possibili combinazioni (16) sono poche, conviene imparare a memoria una tabella lookup per la conversione

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**

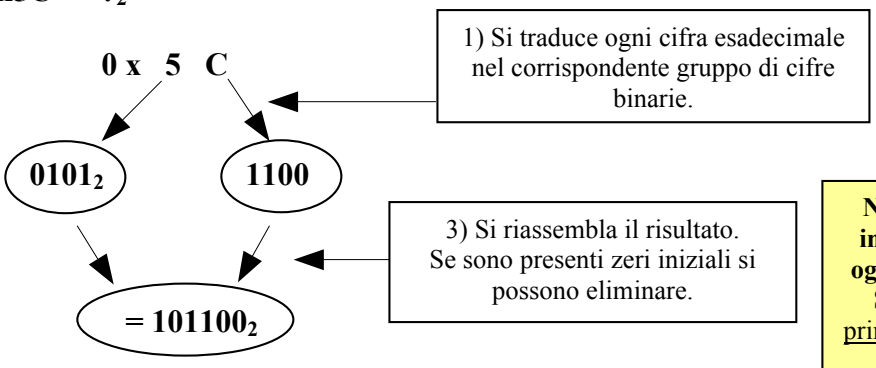
b. $\mathbf{101100_2} = 00101_2 \mid 1100_2 = 0x2 \mid 0xC =$ **0x2C**

c. $\mathbf{111101001010_2} = 1111_2 \mid 0100_2 \mid 1010_2 = 0xF \mid 0x4 \mid 0xA =$ **0xF4A**

d. $\mathbf{10110000001_2} = 0101_2 \mid 1000_2 \mid 0001_2 = 0x5 \mid 0x8 \mid 0x1 =$ **0x581**

4. **Conversione esadecimale → binario**

a. $\mathbf{0x5C \rightarrow ?_2}$



Non dimenticate gli zero iniziali quando traducete ogni singolo blocco di 4 bit
Solo gli zero iniziali del primo blocco vanno eliminati

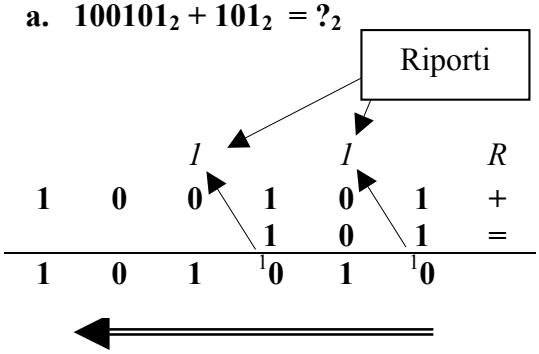
b. **0xBD4** = 0xB | 0xD | 0x4 = $1011_2 \mid 1101_2 \mid 0100_2$ = $\mathbf{110111010100_2}$

c. **0x159** = 0x1 | 0x5 | 0x9 = ~~000~~$1_2 \mid 0101_2 \mid 1001_2$ = $\mathbf{101011001_2}$

d. **0xB062** = 0xB | 0x0 | 0x6 | 0x2 = $1011_2 \mid 0000_2 \mid 0110_2 \mid 0010_2$ = $\mathbf{1011000001100010_2}$

5. Somme binarie

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



La somma binaria si esegue esattamente come quella decimale classica. Si allineano a destra i numeri da sommare quindi si procede sommando ogni coppia di bit e l'eventuale riporto, da destra verso sinistra. Gli eventuali riporti si sommano sulla coppia di bit successiva a sinista

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

$100101_2 + 101_2 = 101010_2$

CONSIGLIO: verificate sempre i risultati traducendo i numeri binari in decimale e rifacendo i calcoli in questa base

b. $1011101_2 + 11001100_2 = 100101001_2$

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

c. $10011_2 + 110111001_2 = 111101010_2$

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

d. $111100110_2 + 110101001_2 = 1110001111_2$

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

Massimo Marchi – marchi@dsi.unimi.it

6. Sottrazioni binarie (in complemento a due)

a. $1001_2 - 110_2 = ?_2$

COMPLEMENTO A DUE

Completamento

$1001_2 - 110_2$

$\textcircled{0}1001_2 - \textcircled{0}\,\textcircled{0}110_2$

Bit di segno

$\textcircled{1}1001_2$

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 indicata, 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**	+

$\rightarrow$ **-110_2** *in formato* CA2

$01001_2 + 11010_2$

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

Dato un numero in CA2 si può tornare alla notazione *Modulo&Segno calcolando il CA2 del CA2,* e cioè invertendo e sommando 1.

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

$\rightarrow$ **$+11_2$**

Il riporto oltre il bit di segno viene scartato

Può capitare che il risultato sia troppo grande, in modulo, per essere rappresentato dal numero di bit disponibili. In questo caso si dice che si è verificato un errore di *OVERFLOW*.

Ex: Calcolare 3+3 usando 2 bit + segno
$3 + 3 = \mathbf{0}11_2 + \mathbf{0}11_2 = \mathbf{1}10_2 =$ **-2 in CA2 !!**

Ex: Calcolare -2-3 usando 2 bit + segno
$-2 - 3 = \mathbf{1}10_2 + \mathbf{1}01_2 = \mathbf{0}01_2 = +1$!!

La condizione di overflow si verifica controllando la coerenza tra segno dei termini sommati ed il risultato.

Ex: **"+" + "+" = "-" incoerente! overlfow!**
"-" + "-" = "+" incoerente! overlfow!
Se i segni sono diversi, l'overflow non si verifica mai.

Risultato: $\mathbf{1001_2 - 110_2 = +11_2}$

*Nota: per sottrarre 1 in binario potete cercare il primo **1** a destra, porlo a **0** e quindi porre a **1** tutti gli **0** a sinistra dell'**1** trovato.*
Ex: $11000_2 - 1_2 = 10\mathbf{111}_2$

Massimo Marchi – marchi@dsi.unimi.it

b. $101_2 - 1011_2 = ?_2$

Uso quattro cifre più il bit di segno:

$$\mathbf{101_2 - 1011_2 = 0\ 0101_2 - 0\ 1011_2 = 00101_2 + (\ 10100_2 + 1\ )}$$

Calcolo il CA2 di $\mathbf{-1001_2}$:

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

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

S	*1*		*1*		*R*	
0	**0**	**1**	**0**	**1**	+	
1	**0**	**1**	**0**	**1**	=	
1	**1**	^{1}**0**	**1**	^{1}**0**		→ **-110_2** in CA2
						(00101+1=00110 → -110_2)

Risultato: $\mathbf{101_2 - 1011_2 = -110_2}$

c. $10001_2 - 1111_2 = ?_2$

Uso cinque cifre più il bit di segno:

$$\mathbf{10001_2 - 1111_2 = 0\ 10001_2 - 0\ 01111_2 = 010001_2 + (\ 110000_2 + 1\ )}$$

Calcolo il CA2 di $\mathbf{-1001_2}$:

S						*R*	
1	**1**	**0**	**0**	**0**	**0**	+	
0	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**0**	**0**	**1**		→ **-1111_2** in CA2

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

1	*1*				*1*		*R*	
	0	**1**	**0**	**0**	**0**	**1**	+	
	1	**1**	**0**	**0**	**0**	**1**	=	
~~1~~	^{1}**0**	^{1}**0**	**0**	**0**	**1**	^{1}**0**		→ **$+10_2$**

Risultato: $\mathbf{10001_2 - 1111_2 = +10_2}$

d. $\mathbf{-111_2 - 101010_2 = ?_2}$ **(***Eseguire i calcoli a 8 bit)*

Uso sette cifre più il bit di segno:

$\mathbf{-111_2 - 101010_2 = - 0\ 0000111_2 - 0\ 0101010_2 = (1\ 1111000_2 + 1_2) + (\ 11010101_2 + 1_2\ )}$

S								*R*	
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	+	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**1**	**1**	**1**	**0**	**0**	**1**		→ **-111_2** in CA2

S						*1*		*R*	
1	**1**	**0**	**1**	**0**	**1**	**0**	**1**	+	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**1**	**0**	**1**	**1**	^{1}**0**		→ **-101010_2** in CA2

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

1	*1*	*1*	*1*					*R*	
	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	+
	1	**1**	**0**	**1**	**0**	**1**	**1**	**0**	=
~~1~~	^{1}**1**	^{1}**1**	^{1}**0**	^{1}**0**	**1**	**1**	**1**	**1**	→ **-110001_2** in CA2
							(00110000+1=→ -110001)		

Risultato: $\mathbf{-111_2 - 101010_2 = -110001_2}$

I segni sono uguali ("-"), l'overflow è possibile; verifico la condizione:

"-" + "-" = "-" → segni concordi, nessun overflow

7. Conversione in floating point secondo lo standard IEEE 754

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

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: $\boldsymbol{s = 1}$

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: $\mathbf{1{,}010011_2 \rightarrow m = 0100110000\ 0000000000\ 000}$

Polarizzo l'esponente: $\mathbf{4_{10} + 127_{10} = 131_{10} = 128_{10} + 2_{10} + 1_{10} = 10000011_2}$

Esponente a 8 bits: $\mathbf{10000011_2 \rightarrow e = 10000011}$

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

Massimo Marchi – marchi@dsi.unimi.it

b. $+9{,}3125_{10}$ = <s,e,m>?

Numero positivo: ***s* = 0**

Converto 9_{10} in base 2: **$9_{10} = 1001_2$**

9 / **2** = **4** resto **1**
4 / **2** = **2** resto **0**
2 / **2** = **1** resto **0**
1 / **2** = **0** resto **1**

Converto $,3125_{10}$ in base 2: **$0{,}3125_{10} = 0{,}0101_2$**

0,3125 * 2 = **0,625** parte intera **0**
0,625 * 2 = **1,25** parte intera **1**
0,25 * 2 = **0,5** parte intera **0**
0,5 * 2 = **1,0** parte intera **1**
#########

Unisco i risultati: **$9{,}3125_{10} = 1001{,}0101_2$**

Normalizzo: **$1{,}0010101_2 \ (10_2)^3$** *(shift a sx di 3 posizioni ⇒ exp= +3)*

Mantissa a 23 bit: **$1{,}0010101_2 \rightarrow m$ = 00101 01000 00000 00000 000**

Polarizzo l'esponente: **$+3_{10} + 127_{10} = 130_{10} = 10000010_2$**

In questo caso la matissa è più corta di 23 bit richiesti. Ne segue che si aggiungono 0 a destra fino a completare.

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**

In questo caso l'esponente è già a 8 bit, altrimenti si aggiungono 0 a sinistra

Esponente a 8 bits: **$10000010_2 \rightarrow e$ = 10000010**

$+9{,}3125_{10}$ = <0, 10000010, 00101010000000000000000>

Massimo Marchi – marchi@dsi.unimi.it

c. $-0{,}125_{10}$ = <s,e,m>?

Numero negativo: ***s* = 1**
Converto 5_{10} in base 2: $\mathbf{0_{10} = 0_2}$
Converto $\cdot{,}125_{10}$ in base 2: $\mathbf{0{,}125_{10} = 0{,}001_2}$

0,125 * 2 = 0,25 parte intera **0**
0,25 * 2 = 0,5 parte intera **0**
0,5 * 2 = 1,0 parte intera **1**
#########

Unisco i risultati: $\mathbf{0{,}125_{10} = 0{,}001_2}$
Normalizzo: $\mathbf{0{,}001_2 = 1{,}0_2 \cdot (10_2)^{-3}}$ *(shift a destra di 3 posizioni: exp= -3)*
Mantissa a 23 bit: $\mathbf{\cancel{1},0_2 \rightarrow m = 00000\ 00000\ 00000\ 00000\ 000}$
Polarizzo l'esponente: $\mathbf{-3_{10} + 127_{10} = 124_{10} = 1111100_2}$

124 / 2 = 62 resto **0**
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**

In questo caso l'esponente è più corto di 8 bit, aggiungo uno 0 a <u>sinistra</u>

Esponente a 8 bits: $\mathbf{1111100_2 \rightarrow e = 01111100}$

$-0{,}125_{10}$ <1, 01111100, 00000000000000000000>

Massimo Marchi – marchi@dsi.unimi.it

d. $0{,}1_{10}$ = <s,e,m>?

Numero positivo: $s = 0$
Converto 1_{10} in base 2: $0_{10} = 0_2$
Converto $\cdot,1_{10}$ in base 2: $0{,}1_{10} = 0{,}0\ \overline{0011}_2$

0,1 * 2 = 0,2 parte intera **0**
0,2 * 2 = 0,4 parte intera **0**
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 * …
########

Da qui in poi si ripetono ciclicamente le cifre **0011**

Unisco i risultati: $0{,}1_{10} = 0{,}0\ \overline{0011}_2$
Normalizzo: $0{,}0\ \overline{0011}_2 = 0{,}0\ 0011\ \overline{0011}_2 = 1{,}1\overline{0011}_2 \cdot (10_2)^{-4}$
Mantissa a 23 bit: ~~1~~, 100 1100 1100 1100 1100 1100 ~~110011~~
$\rightarrow m = 00110011001100110011001$
Polarizzo l'esponente: $-4_{10} + 127_{10} = 123_{10} = 1111011_2$

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

123 / 2 = 61 resto **1**
61 / 2 = 30 resto **1**
30 / 2 = 15 resto **0**
15 / 2 = 7 resto **1**
7 / 2 = 3 resto **1**
3 / 2 = 1 resto **1**
1 / 2 = 0 resto **1**

Nota: per numeri la cui mantissa reale supera i 23 bit, l'ultimo bit (il 23esimo) andrebbe arrotondato considerando anche i bit 24 e 25. Per lo scopo del corso è sufficiente troncare la mantissa dopo il bit 23.

Esponente a 8 bits: $1111011_2 \rightarrow e = 01111011$

$0{,}1_{10}$ = <0, 01111011, 00110011001100110011001100 >

Alcune configurazioni con significato speciale (singola precisione):

0	<0,00000000,00000000000000000000000>
+/-∞	<[*segno*],11111111,00000000000000000000000>
NaN	<[*segno*],11111111, [*mantissa ≠ 0*]>
N.ro denormalizzato	<[*segno*],00000000, [*mantissa denormalizzata≠0*]> *(vedi di seguito)*

Alcuni casi notevoli (singola precisione):

1	**$1{,}0 \cdot (10_2)^0$**	<0,01111111,00000000000000000000000> *(E=0+127)*
MaxFloat	**$1{,}11..1 \times 2^{+127}$**	<0,11111110,11111111111111111111111> *(E=+127+127)*
MinFloat	**$1{,}0 \times 2^{-126}$**	<0,00000001,00000000000000000000000> *(E=-126+127)*
MinFloatDeN	**$0{,}0..01 \times 2^{-126}$**	<0,00000000,00000000000000000000001>