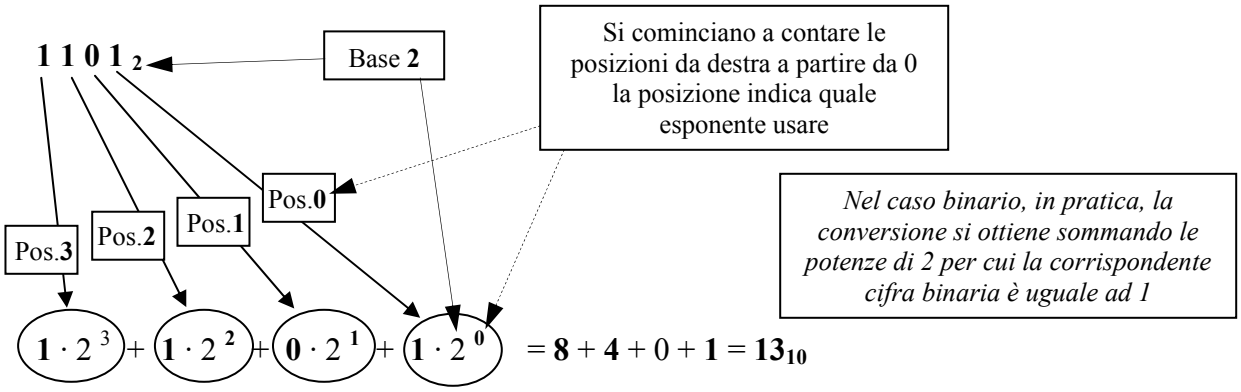
Esercitazione del 09/03/2006 - Soluzioni

1. **Conversione binario → decimale**
   (*Rappresentazione dell'Informazione – Conversione in e da un numero binario , slide 10)*

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



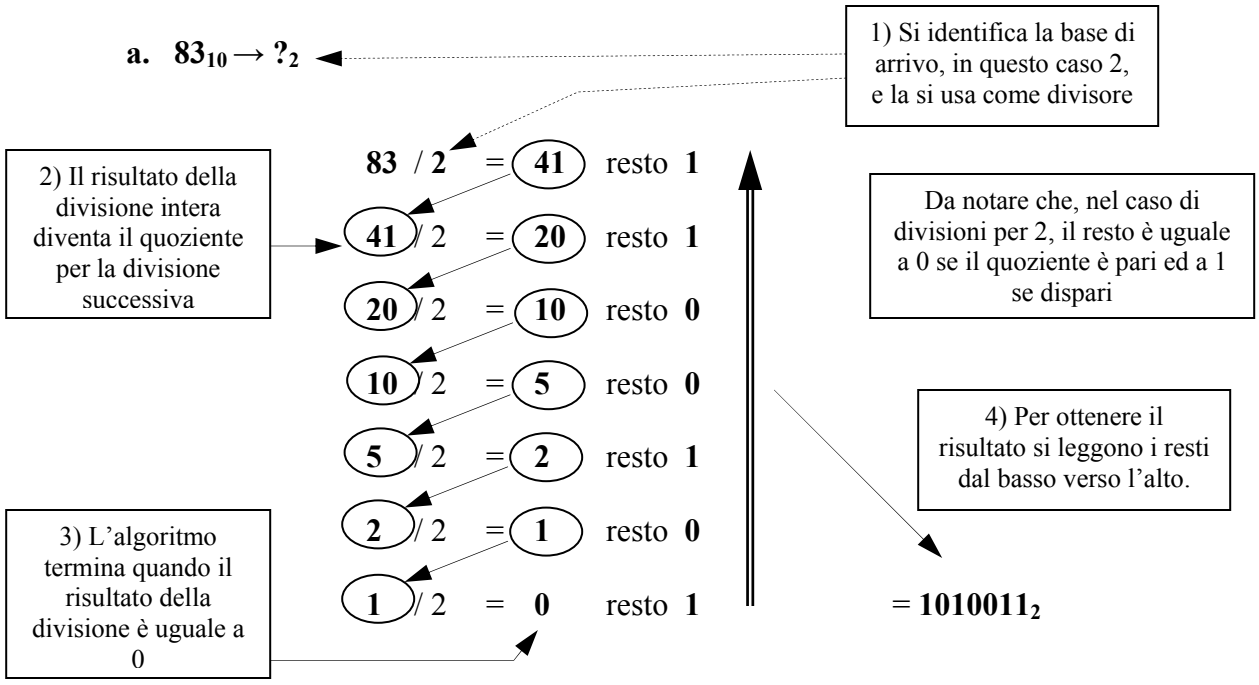
   b. **11100110_2** $= \mathbf{1} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 128 + 64 + 32 + 4 + 2 = \mathbf{230_{10}}$

   c. **1010100_2** $= \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 64 + 16 + 4 = \mathbf{84_{10}}$

   d. **111000100_2** $= \mathbf{1} \cdot 2^8 + \mathbf{1} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 256 + 128 + 64 + 4 = \mathbf{452_{10}}$

Massimo Marchi – marchi@dsi.unimi.it

2. **Conversione decimale → binario**
(*Rappresentazione dell'Informazione – Conversione in e da un numero binario , slide 12)*



b. $330_{10} \rightarrow ?_2$

330	**/ 2 =**	**165**	resto	**0**
165	**/ 2 =**	**82**	resto	**1**
82	**/ 2 =**	**41**	resto	**0**
41	**/ 2 =**	**20**	resto	**1**
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**

$\mathbf{330_{10} = 101001010_2}$

c. $2291_{10} \rightarrow ?_2$

2291	**/ 2 =**	**1235**	resto	**1**
1145	**/ 2 =**	**572**	resto	**1**
572	**/ 2 =**	**286**	resto	**0**
286	**/ 2 =**	**143**	resto	**0**
143	**/ 2 =**	**71**	resto	**1**
71	**/ 2 =**	**35**	resto	**1**
35	**/ 2 =**	**17**	resto	**1**
17	**/ 2 =**	**8**	resto	**1**
8	**/ 2 =**	**4**	resto	**0**
4	**/ 2 =**	**2**	resto	**0**
2	**/ 2 =**	**1**	resto	**0**
1	**/ 2 =**	**0**	resto	**1**

$\mathbf{2291_{10} = 100011110011_2}$

d. $9902_{10} \rightarrow ?_2$

9902	**/ 2 =**	**4951**	resto	**0**
4951	**/ 2 =**	**2475**	resto	**1**
2475	**/ 2 =**	**1237**	resto	**1**
1237	**/ 2 =**	**618**	resto	**1**
618	**/ 2 =**	**309**	resto	**0**
309	**/ 2 =**	**154**	resto	**1**
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**

$\mathbf{9902_{10} = 10001010101110_2}$

3. **Conversione binario → esadecimale**
(*Rappresentazione dell'Informazione – Conversione in e da un numero binario , slide 18)*

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

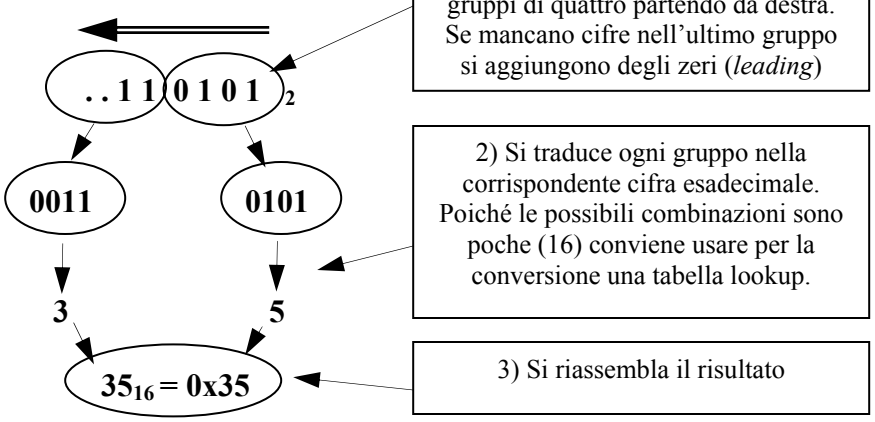


Tabella lookup

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**

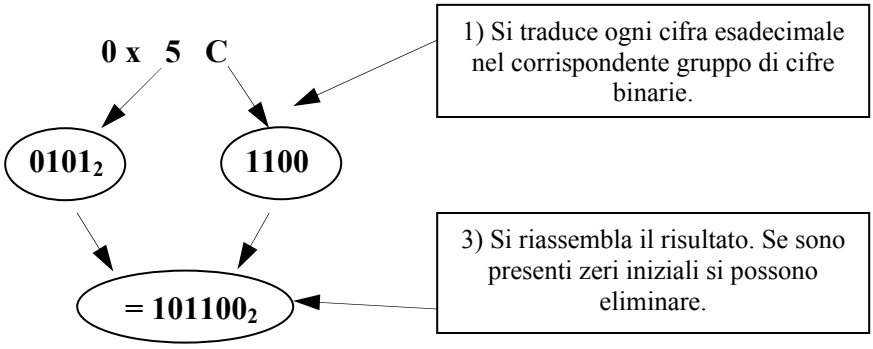
b. $\mathbf{101011_2} = 00 10_2 \mid 1011_2 = 0x2 \mid 0xB =$ **0x2B**

c. $\mathbf{100111100000_2} = 1001_2 \mid 1110_2 \mid 0000_2 = 0x9 \mid 0xE \mid 0x0 =$ **0x9E0**

d. $\mathbf{11110100010_2} = 0111_2 \mid 1010_2 \mid 0010_2 = 0x7 \mid 0xA \mid 0x2 =$ **0x7A2**

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

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

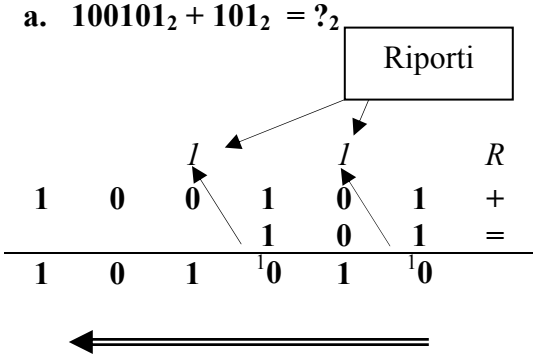


b. **0xC17** = 0xC | 0x1 | 0x7 = $1100_2 \mid 0001_2 \mid 0111_2 =$ **110000010111_2**
c. **0x141** = 0x1 | 0x4 | 0x1 = $0001_2 \mid 0100_2 \mid 0001_2 =$ **101000001_2**
d. **0xAB0C** = 0xA | 0xB | 0x0 | 0xC = $1010_2 \mid 1011_2 \mid 0000_2 \mid 1100_2 =$ **1010101100001100_2**

5. Somme binarie

(*Rappresentazione dell'Informazione – Operazioni elementari su numeri binari ... , slide 20)*

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



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**

$100101_2 + 101_2 = 101010_2$

b. $11100011_2 + 1101101_2 = 101001101_2$

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

c. $101_2 + 101110101_2 = 101111010_2$

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

d. $100100110_2 + 101110101_2 = 1010011011_2$

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

6. Sottrazioni binarie (in complemento a due)

(*Rappresentazione dell'Informazione – Operazioni elementari su numeri binari ... , slide 21-23)*

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

COMPLEMENTO A DUE

Completamento

1) Estendo le cifre alla rappresenta-zione scelta se necessario

$1001_2 - 110_2$

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

Bit di segno

$\textcircled{1}1001_2$

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

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

→ **-110_2** *in formato* 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: **$1001_2 - 110_2 = +11_2$**

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.

Dato un numero in CA2 si può tornare alla notazione *Modulo&Segno* sottraendo uno e quindi invertendo bit a bit.

Per sottrarre uno 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: $11000_2 - 1_2 = 10\mathbf{111}_2$

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!**
Ex: **"-" + "-" = "+" incoerente!**

Massimo Marchi – marchi@dsi.unimi.it

b. $\mathbf{101_2 - 1011_2 = ?_2}$

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 $\mathbf{-1101_2}$:

S					*R*	
1	**0**	**1**	**0**	**0**	+	
0	**0**	**0**	**0**	**1**	=	
1	**0**	**1**	**0**	**1**		→ $\mathbf{-1011_2}$ 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**	$^{1}\mathbf{0}$	**1**	$^{1}\mathbf{0}$		→ $\mathbf{-110_2}$ in CA2

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

c. $\mathbf{10011_2 - 1111_2 = ?_2}$

Uso cinque cifre più il bit di segno:

$\mathbf{10011_2 - 1111_2 = 0\ 10011_2 - 0\ 01111_2 = 010011_2 + (\ 110000_2 + 1\ )}$

Calcolo il CA2 di $\mathbf{-1011_2}$:

S						*R*	
1	**1**	**0**	**0**	**0**	**0**	+	
0	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**0**	**0**	**1**		→ $\mathbf{-1111_2}$ in CA2

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

1	*1*			*1*	*1*		*R*	
	0	**1**	**0**	**0**	**1**	**1**	+	
	1	**1**	**0**	**0**	**0**	**1**	=	
~~1~~	$^{1}\mathbf{0}$	$^{1}\mathbf{0}$	**0**	**1**	$^{1}\mathbf{0}$	$^{1}\mathbf{0}$		→ $\mathbf{+000100_2}$

Risultato: $\mathbf{10011_2 - 1111_2 = +100_2}$

Massimo Marchi – marchi@dsi.unimi.it

d. $\mathbf{1001_2 - 10111_2 = ?_2}$ **(***Eseguire i calcoli a 8 bit, segno compreso)*

Uso sette cifre più il bit di segno:

$\mathbf{1001_2 - 10111_2 = 0\ 0001001_2 - 0\ 0010111_2 = 00001001_2 + (\ 11101000_2 + 1_2\ )}$

S								*R*	
1	**1**	**1**	**0**	**1**	**0**	**0**	**0**	+	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**		→ **– 10111$_2$** in CA2

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

S			*1*			*1*		*R*	
0	**0**	**0**	**0**	**1**	**0**	**0**	**1**	+	
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**	=	
1	**1**	**1**	**1**	^{1}**0**	**0**	**1**	^{1}**0**		→ **-1110$_2$** in CA2

Risultato: $\mathbf{1001_2 - 10111_2 = -1110_2}$

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 $\mathbf{0}_2$ $\mathbf{-1}_2$

Prestiti

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

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.

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 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			*I*		*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

Massimo Marchi – marchi@dsi.unimi.it

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

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**	**=**	
1	**0**	**1**	**1**	**0**		→ **-110_2** in M&S

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

$101_2 - 1011_2 = 00101_2 - 01011_2 = 00111_2 + 10100_2$

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

Il risultato è negativo quindi non occorre correggerlo.

f. $10011_2 - 1111_2 = ?_2$

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

S	-	-				*P*	
	1	^{1}**0**	^{1}**0**	**1**	**1**	**-**	
	0	**1**	**1**	**1**	**1**	**=**	
0	**0**	**0**	**1**	**0**	**0**		→ **$+100_2$**

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

$10011_2 - 1111_2 = 010011_2 - 01111_2 = 010011_2 + 110000_2$

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

Il risultato è positivo quindi occorre correggerlo.

	S			*1*	*1*		*R*	
	0	**0**	**0**	**0**	**1**	**1**	+	
	0	**0**	**0**	**0**	**0**	**1**	=	
1	**0**	**0**	**0**	**1**	^{1}**0**	^{1}**0**		→ **$+100_2$**

Massimo Marchi – marchi@dsi.unimi.it

g. $1001_2 - 10111_2 = ?_2$

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

S	-					*P*	
	1	^{1}**0**	**1**	**1**	**1**	**-**	
	0	**1**	**0**	**0**	**1**	=	
1	**0**	**1**	**1**	**1**	**0**		→ **-1110_2** in M&S

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

$1001_2 - 10111_2 = 001001_2 - 010111_2 = 001001_2 + 101000_2$

S	*I*					*R*	
0	**0**	**1**	**0**	**0**	**1**	+	
1	**0**	**1**	**0**	**0**	**0**	=	
1	**1**	^{1}**0**	**0**	**0**	**1**		→ **-1110_2** in CA1

Il risultato è negativo quindi non occorre correggerlo.

7. Conversione in floating point secondo lo standard IEEE 754

(*Rappresentazione* dell'Informazione – i numeri decimali, slide 27-32, Codifica IEEE754.., *34-38)*

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

> 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: $\mathbf{10000011_2 \rightarrow e = 10000011}$

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

Massimo Marchi – marchi@dsi.unimi.it

b. $-0{,}25_{10}$ = <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: $\mathbf{1{,}0_2 \rightarrow m = 0000000000\ 0000000000\ 000}$
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: $\mathbf{1111101_2 \rightarrow e = 01111101}$

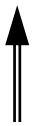
$-0{,}25_{10}$ = <1, 01111101, 00000000000000000000000>

Massimo Marchi – marchi@dsi.unimi.it

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

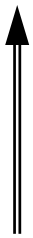
Numero positivo: $\boldsymbol{s = 0}$
Converto 10_{10} in base 2: $\mathbf{1}0_{10} = \mathbf{1010}_2$

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: $\mathbf{\cancel{1},01_2 \rightarrow m = 0100000000\ 0000000000\ 000}$
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: $\mathbf{10000010_2 \rightarrow e = 10000010}$

$+10_{10}$ <0, 10000010, 01000000000000000000>

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

Numero negativo: $s = 1$

Converto 0_{10} in base 2: $1_{10} = 1_2$

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

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: $1{,}7_{10} = 1{,}1\ \overline{0110}_2$

Normalizzo: $1{,}1\ 0110_2 = 1{,}1\ \overline{0110}_2 \cdot (10_2)^0$

Mantissa a 23 bit: ~~1~~, 1 0110 0110 0110 0110 0110 01~~10~~ ~~$\overline{0110}$~~

$\rightarrow$ m = 1011001100 1100110011 001

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

Polarizzo l'esponente: $0_{10} + 127_{10} = 1111111_2$

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: $1111101_2 \rightarrow e$ = 01111111

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

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	**$= 1{,}11..1 \times 2^{+127}$**	<0,11111110,00000000000000000000000> *(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>

Massimo Marchi – marchi@dsi.unimi.it

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

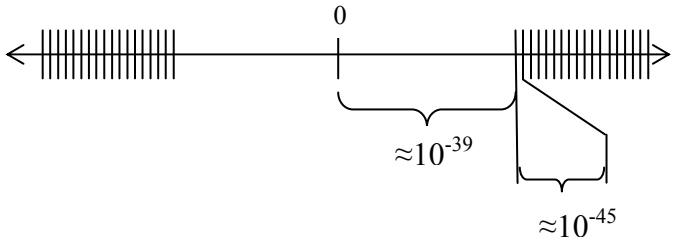
Il più piccolo numero positivo rappresentabile in formato IEEE 754 standard a precisione singola è:

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

Il più piccolo numero positivo successivo:

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

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}.



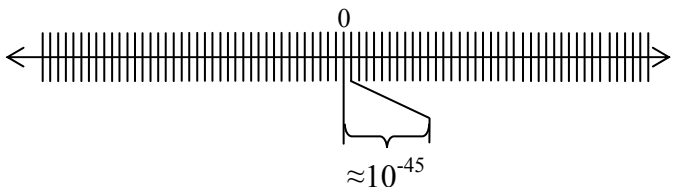
Il più piccolo numero positivo rappresentabile in formato IEEE 754 denormalizzato a precisione singola è ($0{,}f \times 2^{-126}$):

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

Il più piccolo numero positivo successivo:

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

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.

Per saperne di più: ***http://stevehollasch.com/cgindex/coding/ieeefloat.html***

Massimo Marchi – marchi@dsi.unimi.it