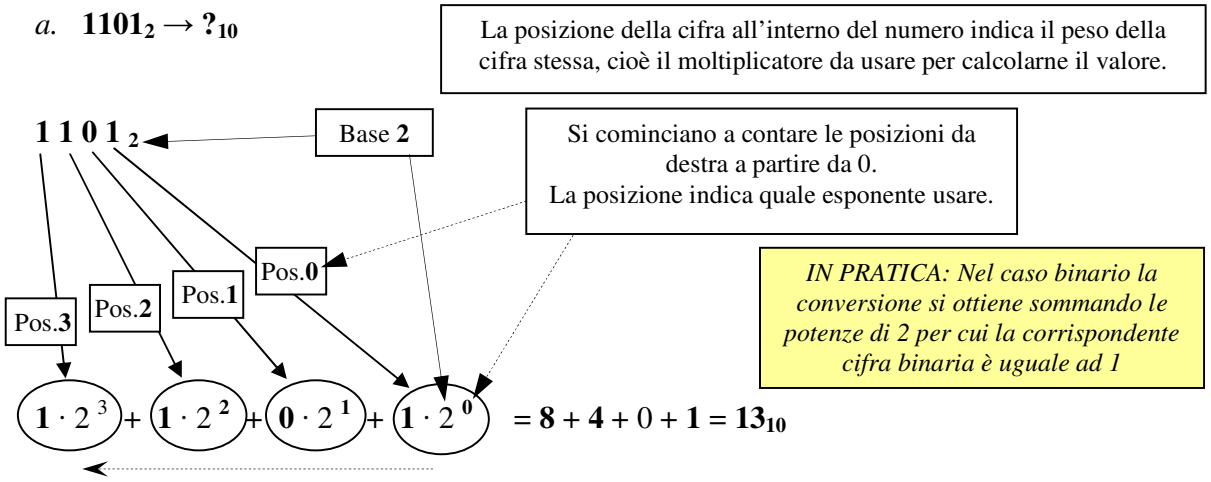
Esercitazione del 08/03/2007 - Soluzioni

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

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



   b. **10110110_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0$
   $= 128 + 32 + 16 + 4 + 2 = \mathbf{182_{10}}$

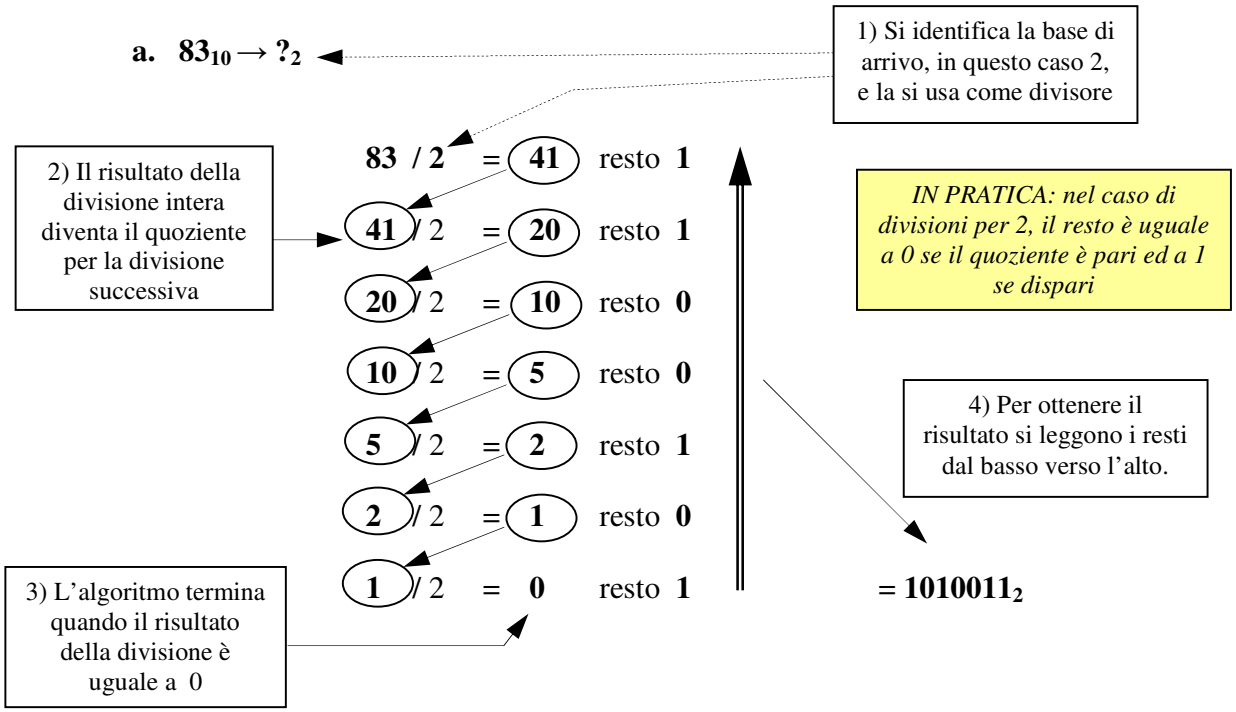
   c. **1111111_2** $= \mathbf{1} \cdot 2^6 + \mathbf{1} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 64 + 32 + 16 + 8 + 4 + 2 + 1 = \mathbf{127_{10}}$

   d. **10000001_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
   $= 128 + 1 = \mathbf{129_{10}}$

Massimo Marchi – marchi@dsi.unimi.it

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

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



b. $237_{10} \rightarrow ?_2$

237 / 2 = 118 resto **1**
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**

$237_{10} = 11101101_2$

c. $3172_{10} \rightarrow ?_2$

3172 / 2 = 1586 resto **0**
1586 / 2 = 793 resto **0**
793 / 2 = 396 resto **1**
396 / 2 = 198 resto **0**
198 / 2 = 99 resto **0**
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**

$3172_{10} = 110001100100_2$

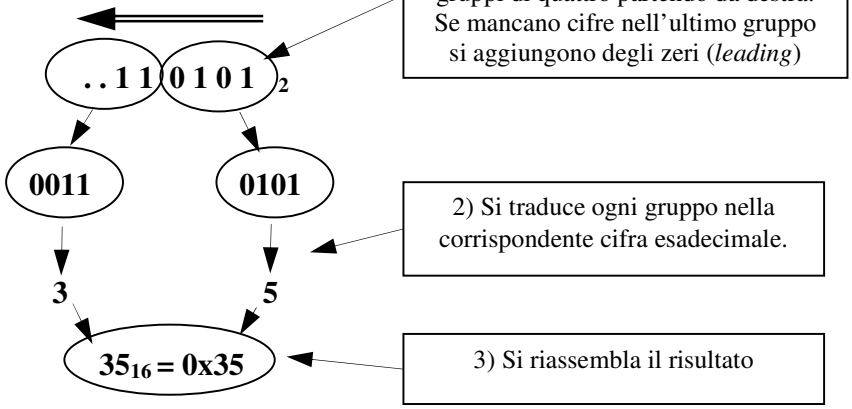
d. $8873_{10} \rightarrow ?_2$

8873 / 2 = 4436 resto **1**
4436 / 2 = 2218 resto **0**
2218 / 2 = 1109 resto **0**
1109 / 2 = 554 resto **1**
554 / 2 = 277 resto **0**
277 / 2 = 138 resto **1**
138 / 2 = 69 resto **0**
69 / 2 = 34 resto **1**
34 / 2 = 17 resto **0**
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**

$8873_{10} = 10001010101001_2$

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

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



Poiché le possibili combinazioni (16) sono poche, conviene usare (i.e. 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**

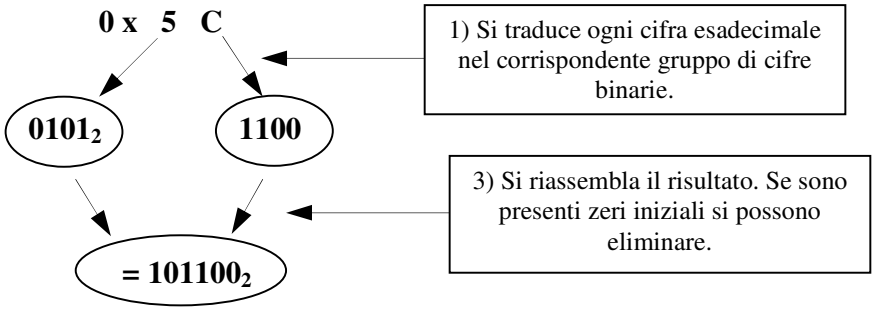
b. $\mathbf{10011_2}$ = 000_2 | 0011_2 = 0x1 | 0x3 = **0x13**

c. $\mathbf{110010010000_2}$ = 1100_2 | 1001_2 | 0000_2 = 0xC | 0x9 | 0x0 = **0xC90**

d. $\mathbf{11011011011_2}$ = 0110_2 | 1101_2 | 1011_2 = 0x6 | 0xD | 0xB = **0x6DB**

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

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



b. **0xB23** = 0xB | 0x2 | 0x3 = 1011_2 | 0010_2 | 0011_2 = $\mathbf{101100100011_2}$

c. **0x223** = 0x2 | 0x4 | 0x1 = ~~00~~10_2 | 0010_2 | 0011_2 = $\mathbf{1000100011_2}$

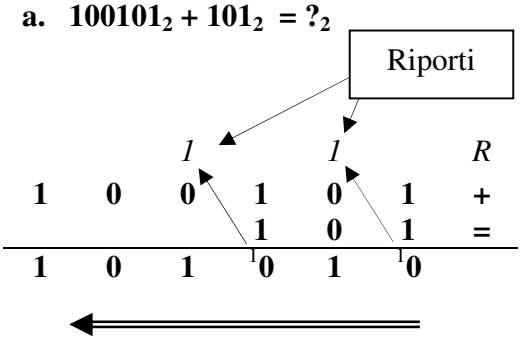
d. **0x104D** = 0x1 | 0x0 | 0x4 | 0xB = ~~000~~1_2 | 0000_2 | 0100_2 | 1101_2 = $\mathbf{1000001001101_2}$

Massimo Marchi – marchi@dsi.unimi.it

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 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 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. $1111111_2 + 10101000_2 = 100100111_2$

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

c. $1010_2 + 101010111_2 = 101100001_2$

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

d. $110110100_2 + 101010101_2 = 1010011011_2$

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

Massimo Marchi – marchi@dsi.unimi.it

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 rappresentazione scelta se necessario

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.

$1001_2 - 110_2$

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

Bit di segno

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

$\textcircled{1}1001_2$

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* sottraendo 1 e quindi invertendo bit a bit.

*IN PRATICA: 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: $11000_2 - 1_2 = 10\mathbf{111}_2$

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

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

Massimo Marchi – marchi@dsi.unimi.it

b. $11_2 - 1100_2 = ?_2$

Uso quattro cifre più il bit di segno:

$11_2 - 1100_2 = 0\ 0011_2 - 0\ 1100_2 = 00011_2 + (\ 10011_2 + 1\ )$

Calcolo il CA2 di **-1100_2**:

S		*1*	*1*		*R*	
1	**0**	**0**	**1**	**1**	**+**	
0	**0**	**0**	**0**	**1**	**=**	
1	**0**	**1**	^{1}**0**	^{1}**0**		→ **-1100_2** in CA2

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

S					*R*	
0	**0**	**0**	**1**	**1**	**+**	
1	**0**	**1**	**0**	**0**	**=**	
1	**0**	**1**	**1**	**1**		→ **-1001_2** in CA2
						(10111-1=10110 → 01001)

Risultato: **$11_2 - 1100_2 = -1001_2$**

c. $11001_2 - 1001_2 = ?_2$

Uso cinque cifre più il bit di segno:

$11001_2 - 1001_2 = 0\ 11001_2 - 0\ 01001_2 = 011001_2 + (\ 110110_2 + 1\ )$

Calcolo il CA2 di **-1001_2** :

S						*R*	
1	**1**	**0**	**1**	**1**	**0**	**+**	
0	**0**	**0**	**0**	**0**	**1**	**=**	
1	**1**	**0**	**1**	**1**	**1**		→ **-1001_2** in CA2

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

1	*1*	*1*	*1*	*1*	*1*		*R*	
	0	**1**	**1**	**0**	**0**	**1**	**+**	
	1	**1**	**0**	**1**	**1**	**1**	**=**	
~~1~~	^{1}**0**	^{1}**1**	^{1}**0**	^{1}**0**	^{1}**0**	^{1}**0**		→ **$+10000_2$**

Risultato: **$11001_2 - 1001_2 = +10000_2$**

Massimo Marchi – marchi@dsi.unimi.it

d. $\mathbf{101_2 - 101111_2 = ?_2}$ (*Eseguire i calcoli a 8 bit)*

Uso sette cifre più il bit di segno:

$\mathbf{101_2 - 101111_2 = 0\ 0000101_2 - 0\ 0101111_2 = 00000101_2 + (\ 11010000_2 + 1_2\ )}$

S								*R*	
1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**+**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**=**	
1	**1**	**0**	**1**	**0**	**0**	**0**	**1**		→ $\mathbf{-101111_2}$ in CA2

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

S						*1*		*R*	
0	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**+**	
1	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**=**	
1	**1**	**0**	**1**	**0**	**1**	**1**	^{1}**0**		→ $\mathbf{-101010_2}$ in CA2
						(11010110-1=11010101→-101010)			

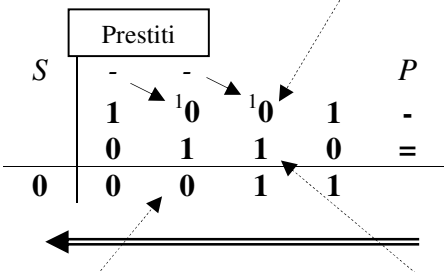
Risultato: $\mathbf{101_2 - 101111_2 = -101010_2}$

Massimo Marchi – marchi@dsi.unimi.it

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



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

$\mathbf{1001_2 - 110_2}$

Estendo le cifre se necessario

$\mathbf{01001_2 - 00110_2}$

Calcolo il complemento a uno del secondo termine invertendo i bit

$\mathbf{01001_2 + 11001_2}$

La sottrazione in **complemento a uno** si esegue sommando al primo termine il complemento a uno del secondo termine.
Se il risultato è di segno concorde, 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			*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

Massimo Marchi – marchi@dsi.unimi.it

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

Modulo e Segno: il primo termine è più piccolo quindi inverto i termini, il risultato sarà di segno opposto a quello del primo termine, 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.

$\mathbf{101_2 - 1011_2 = 00101_2 - 01011_2 = 00111_2 + 10100_2}$

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 di segno discorde rispetto al primo termine quindi non occorre correggerlo.

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

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

$\mathbf{10011_2 - 1111_2 = 010011_2 - 01111_2 = 010011_2 + 110000_2}$

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 concorde quindi occorre correggerlo.

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

Massimo Marchi – marchi@dsi.unimi.it

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

Modulo e Segno: il primo termine è più piccolo quindi inverto i termini, il risultato sarà di segno opposto del primo termine, 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	*1*					*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.

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

Modulo e Segno: il primo termine è negativo come il secondo. In questo caso si esegue la somma tra i moduli e si aggiusta il segno.

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

$-1001_2 - 10011_2 = -001001_2 - 010011_2 = 110110_2 + 101100_2$

1	*1*	*1*	*1*				*R*	
	1	**1**	**0**	**1**	**1**	**0**	**+**	
	1	**0**	**1**	**1**	**0**	**0**	**=**	
~~**1**~~	^{1}**1**	^{1}**0**	^{1}**0**	^{1}**0**	**1**	**0**		→ **-11101_2** in CA1

Il risultato è di segno concorde quindi occorre correggerlo sommando 1.

S						*R*	
1	**0**	**0**	**0**	**1**	**0**	**+**	
0	**0**	**0**	**0**	**0**	**1**	**=**	
1	**0**	**0**	**0**	**1**	**1**		→ **-11100_2** in CA1

7. Conversione in floating point secondo lo standard IEEE 754

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

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

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: $s = 1$

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

$-20{,}75_{10} = \langle 1, 10000011, 01001100000000000000000 \rangle$

Massimo Marchi – marchi@dsi.unimi.it

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

Numero positivo: $s = 0$
Converto 0_{10} in base 2: $0_{10} = 0_2$
Converto $0{,}125_{10}$ in base 2: $0{,}125_{10} = 0{,}001_2$

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: $0{,}125_{10} = 0{,}001_2$
Normalizzo: $0{,}001_2 = 1{,}0_2 \cdot (10_2)^{-3}$ *(shift a sinistra di 3 posizioni: exp= -3)*
Mantissa a 23 bit: $1{,}0_2 \rightarrow m = 00000000000000000000000$
Polarizzo l'esponente: $-3_{10} + 127_{10} = 124_{10} = 1111100_2$

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**

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

$+0{,}125_{10}$ = <0, 01111100, 00000000000000000000000>

c. -5_{10} = <s,m,e>?

Numero negativo: $s = 1$
Converto 5_{10} in base 2: $5_{10} = 101_2$

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

Converto $,0_{10}$ in base 2: $0,0_{10} = 0,0_2$
Unisco i risultati: $5,0_{10} = 101,0_2$
Normalizzo: $101,0_2 = 1,01_2 \cdot (10_2)^2$ *(shift a destra di 2 posizioni: exp= +2)*
Mantissa a 23 bit: $1,01_2 \rightarrow m = 0100000000\ 0000000000\ 000$
Polarizzo l'esponente: $2_{10} + 127_{10} = 129_{10} = 10000001_2$

129	**/ 2 =**	**64**	resto	**1**
64	**/ 2 =**	**32**	resto	**0**
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: $10000001_2 \rightarrow e = 10000001$

-5_{10} <1, 10000001, 01000000000000000000>

Massimo Marchi – marchi@dsi.unimi.it

d. $-0{,}3_{10}$ = <s,m,e>?

Numero negativo: $s = 1$
Converto 0_{10} in base 2: $0_{10} = 0_2$
Converto $0{,}3_{10}$ in base 2: $0{,}3_{10} = 0{,}1\ \overline{0110}_2$

0,3 * 2 = 0,6 parte intera **0**
0,6 * 2 = 1,2 parte intera **1**
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 * …
########

Da qui in poi si ripetono ciclicamente le cifre **1001**

Unisco i risultati: $0{,}3_{10} = 0{,}0\ \overline{1001}_2$
Normalizzo: $0{,}0\ \overline{1001}_2 = 0{,}0\ 1001\ \overline{1001}_2 = 1{,}001\ \overline{1001}_2 \cdot (10_2)^{-2}$
Mantissa a 23 bit: ~~1~~, 001 1001 1001 1001 1001 1001 ~~1001~~
$\rightarrow m = 00110011001100110011001$
Polarizzo l'esponente: $-2_{10} + 127_{10} = 1111111_2$

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

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: $1111101_2 \rightarrow e = 01111101$

$-0{,}3_{10}$ = <1, 01111101, 00110011001100110011001 >

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.

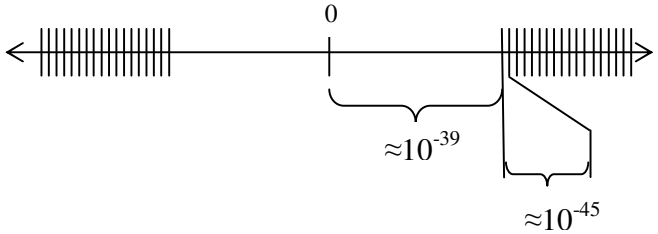
Il più piccolo numero positivo rappresentabile in formato IEEE 754 standard a precisione singola è (*si ricordi che l'esponente uguale a **0** è usato per indicare valori speciali come lo **zero***):

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

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}.



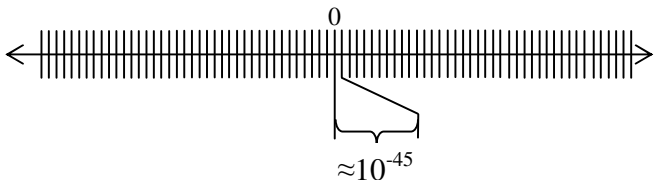
Il più piccolo numero positivo rappresentabile in formato IEEE 754 denormalizzato a precisione singola è ($0.f \times 2^{-126}$):

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

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