Esercitazione di Informatica A

Codifiche numeriche: Conversioni da e per basi 2^n; Virgola fissa, virgola mobile

Stefano Cherubin
<nome>.<cognome>@polimi.it

Esercitazione 4
03 Novembre 2016

Sezione 1

1. Conversioni da e per basi 2^n
   - Cambio di base
   - Conversioni rapide
   - Esercizi
2. Virgola fissa, virgola mobile
   - Virgola fissa
   - Virgola mobile
   - Esercizi

Recall: Conversione da decimale a binario

Algoritmo

Dividere ripetutamente per la base (2). Scrivere i resti della divisione in ordine inverso.

Recall: Conversione da base 2 a decimale

Algoritmo

Moltiplicare il valore di ciascuno dei bit di modulo per 2 elevato alla rispettiva posizione. Sommare il tutto.

0	0	1	0	1	0	1	0
	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$\pm$	modulo						

Conversione da decimale a base b

Algoritmo

Dividere ripetutamente per la base b. Scrivere i resti della divisione in ordine inverso.

Conversione da base b a decimale

Algoritmo

Moltiplicare il valore di ciascuno dei bit di modulo per la base b elevata alla rispettiva posizione. Sommare il tutto.

0	0	2	5	4	7	1	0
	b^6	b^5	b^4	b^3	b^2	b^1	b^0
$\pm$				modulo			

Conversione da oct a binario

Algoritmo

Ogni cifra ottale può essere convertita in un blocco di 3 bit. La rappresentazione in binario si ottiene concatenando i blocchi così ottenuti.

oct	bin	oct	bin
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Conversione da oct a binario

Algoritmo

Ogni cifra ottale può essere convertita in un blocco di 3 bit. La rappresentazione in binario si ottiene concatenando i blocchi così ottenuti.

oct	bin	oct	bin
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

5	1	4
101	001	100

$$514_8 = 101001100_2$$

Conversione da binario a oct

Algoritmo

Raggruppare i bit in blocchi da 3 partendo dal bit meno significativo (LSB). In caso di necessità estendere in segno per raggiungere un numero di bit multiplo di 3. Convertire ciascun blocco da binario a ottale e concatenare i risultati.

Conversione da binario a oct

Algoritmo

Raggruppare i bit in blocchi da 3 partendo dal bit meno significativo (LSB). In caso di necessità estendere in segno per raggiungere un numero di bit multiplo di 3. Convertire ciascun blocco da binario a ottale e concatenare i risultati.

	0	1	1	0	1

0	0	1	1	0	1
	1			5	

$$01101_2 = 15_8$$

Conversione da hex a binario

Algoritmo

Ogni cifra esadecimale può essere convertita in un blocco di 4 bit. La rappresentazione in binario si ottiene concatenando i blocchi così ottenuti.

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

Conversione da hex a binario

Algoritmo

Ogni cifra esadecimale può essere convertita in un blocco di 4 bit. La rappresentazione in binario si ottiene concatenando i blocchi così ottenuti.

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

D	5
1101	0101

$$D5_{16} = 11010101_2$$

Conversione da binario a hex

Algoritmo

Raggruppare i bit in blocchi da 4 partendo dal bit meno significativo (LSB). In caso di necessità estendere in segno per raggiungere un numero di bit multiplo di 4. Convertire ciascun blocco da binario a esadecimale e concatenare i risultati.

Conversione da binario a hex

Algoritmo

Raggruppare i bit in blocchi da 4 partendo dal bit meno significativo (LSB). In caso di necessità estendere in segno per raggiungere un numero di bit multiplo di 4. Convertire ciascun blocco da binario a esadecimale e concatenare i risultati.

	1	0	1	1	0	1	1

0	1	0	1	1	0	1	1
5				B			

$$1011011_2 = 5B_{16}$$

Conversione a base n

Convertire il seguente numero binario in base 12

- 1101001

Conversione a base n

Convertire il seguente numero binario in base 12

- 1101001

Per semplicità è opportuno portare il numero binario in base 10 e poi convertire a base 12

$$\begin{array}{ccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

$$2^0 + 2^3 + 2^5 + 2^6 = 1 + 8 + 32 + 64 = 105$$

Conversione a base n

Convertire il seguente numero binario in base 12

- 1101001

Per semplicità è opportuno portare il numero binario in base 10 e poi convertire a base 12

$$\begin{array}{ccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

$$2^0 + 2^3 + 2^5 + 2^6 = 1 + 8 + 32 + 64 = 105$$

$$\begin{array}{c|cc} 105 & 12 & \\ \hline 8 & 9 & \uparrow \\ 0 & 8 & \uparrow \end{array}$$

$$1101001_2 = 105_{10} = 89_{12}$$

Aritmetica in base 16: $1F + A3$

Eseguire la seguente operazione in base 16:

$$1F + A3$$

Aritmetica in base 16: $1F + A3$

Eseguire la seguente operazione in base 16:

$$1F + A3$$

$$\begin{array}{cc}
1 & \mathrm{F} & + \\
\mathrm{A} & 3 & = \\
\hline
\mathrm{C} & 2 &
\end{array}$$

Aritmetica in base 16...

Eseguire la seguente operazione in base 16:

$$CC + FF$$

Aritmetica in base 16...

Eseguire la seguente operazione in base 16:

$$CC + FF$$

$$\begin{array}{ccc} & C & C \\ & F & F \\ \hline 1 & C & B \end{array} \begin{array}{c} + \\ = \\ \end{array}$$

...per chi ha dimenticato a casa la laurea in matematica

$$x_{16} + y_{16} = x_{10} + y_{10}$$

...per chi ha dimenticato a casa la laurea in matematica

$$x_{16} + y_{16} = x_{10} + y_{10}$$

$CC_{16} = C \cdot 16^0 + C \cdot 16^1 = 12 \cdot 1 + 12 \cdot 16 = 204_{10}$
$FF_{16} = F \cdot 16^0 + F \cdot 16^1 = 15 \cdot 1 + 15 \cdot 16 = 255_{10}$

...per chi ha dimenticato a casa la laurea in matematica

$$x_{16} + y_{16} = x_{10} + y_{10}$$

$CC_{16} = C \cdot 16^0 + C \cdot 16^1 = 12 \cdot 1 + 12 \cdot 16 = 204_{10}$
$FF_{16} = F \cdot 16^0 + F \cdot 16^1 = 15 \cdot 1 + 15 \cdot 16 = 255_{10}$

$$204 + 255 = 459$$

$$x_{16} + y_{16} = x_{10} + y_{10}$$

$$CC_{16} = C \cdot 16^0 + C \cdot 16^1 = 12 \cdot 1 + 12 \cdot 16 = 204_{10}$$
$$FF_{16} = F \cdot 16^0 + F \cdot 16^1 = 15 \cdot 1 + 15 \cdot 16 = 255_{10}$$

$$204 + 255 = 459$$

$$\begin{array}{r|c}
459 & 16 \\ \hline
28 & \text{B} \quad \uparrow \\
1 & \text{C} \quad \uparrow \\
0 & 1 \quad \uparrow
\end{array}$$

$$CC_{16} + FF_{16} = 204_{10} + 255_{10} = 459_{10} = 1CB_{16}$$

Sezione 2

1. Conversioni da e per basi 2^n
   - Cambio di base
   - Conversioni rapide
   - Esercizi
2. Virgola fissa, virgola mobile
   - Virgola fissa
   - Virgola mobile
   - Esercizi

Virgola: not this one

Figura: Virgola
http://nonciclopedia.wikia.com/wiki/File:Virgola_impiccato.jpeg

Notazione fixed point

Si assegnano un numero fisso di bit per la parte intera e un numero fisso di bit per la parte decimale.

$$106.85$$

Notazione fixed point

Si assegnano un numero fisso di bit per la parte intera e un numero fisso di bit per la parte decimale.

$$106.85$$

- Solitamente si sceglie la dimensione della parte intera in modo da non causare overflow / underflow
- È possibile che venga rappresentata solo una approssimazione del numero decimale equivalente (non dispongo di infinite cifre per la parte frazionaria)
  - L'errore di approssimazione è fisso in valore assoluto.

Conversione da fixed point a decimale

Algoritmo

Moltiplicare il valore di ciascuno dei bit di modulo per 2 elevato alla rispettiva posizione, relativamente alla virgola. Sommare il tutto.

Conversione da fixed point a decimale

Algoritmo

Moltiplicare il valore di ciascuno dei bit di modulo per 2 elevato alla rispettiva posizione, relativamente alla virgola. Sommare il tutto.

0	1	0	1	0	0	1	1
	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
$\pm$	int				fraz		

$$2^3 + 2^1 + 2^{-2} + 2^{-3} = 8 + 2 + 1/4 + 1/8 = 10.375$$

Conversione da decimale a binario (fixed point)

Algoritmo

Si procede con l'algoritmo a divisioni successive per la parte intera. I bit relativi alla componente decimale si ottengono con un simile algoritmo per moltiplicazioni successive in cui le cifre da considerare sono date dalle unità intere generate come risultato delle moltiplicazioni.

Conversione da decimale a binario (fixed point)

Algoritmo

Si procede con l'algoritmo a divisioni successive per la parte intera. I bit relativi alla componente decimale si ottengono con un simile algoritmo per moltiplicazioni successive in cui le cifre da considerare sono date dalle unità intere generate come risultato delle moltiplicazioni.

$$12.375$$

$$\begin{array}{c|cc} 12 & 2 & \\ \hline 6 & 0 & \uparrow \\ 3 & 0 & \uparrow \\ 1 & 1 & \uparrow \\ 0 & 1 & \uparrow \end{array} \qquad \begin{array}{cc|c} & .375 & 2 \\ \hline 0 & .75 & \downarrow \\ 1 & .5 & \downarrow \\ 1 & .0 & \downarrow \end{array}$$

$$12.375_{10} = 1100.011$$

Notazione floating point

Si ricorre alla notazione scientifica e si assegnano un numero fisso di bit per la mantissa e un numero fisso di bit per l'esponente.

$$1.0537E^{+14}$$

Notazione floating point

Si ricorre alla notazione scientifica e si assegnano un numero fisso di bit per la mantissa e un numero fisso di bit per l'esponente.

$$1.0537E^{+14}$$

$$1.\underbrace{0537}_{mantissa}\ E^{\overbrace{exp}^{+14}}$$

Notazione floating point

Si ricorre alla notazione scientifica e si assegnano un numero fisso di bit per la mantissa e un numero fisso di bit per l'esponente.

$$1.0537E^{+14}$$

$$1.\underbrace{0537}_{mantissa}\ E^{\overbrace{exp}^{+14}}$$

- È possibile che venga rappresentata solo una approssimazione del numero decimale equivalente (non ho infiniti decimali)
  - L'errore di approssimazione non è fisso in valore assoluto ma aumenta al crescere del numero rappresentato.
- La notazione scientifica per i numeri in binario è simile: $1.mantissa \cdot 2^{exp}$

float single precision (32 bit)

Standard IEEE 754

0	10100101	10011000000000000000000
±	exponent	fraction

- 1 bit per il segno
- 8 bit per l’esponente
- 23 bit per la mantissa

float single precision (32 bit)

Standard IEEE 754

0	10100101	10011000000000000000000
$\pm$	exponent	fraction

- 1 bit per il segno
- 8 bit per l'esponente
- 23 bit per la mantissa

Attenzione

L'esponente viene memorizzato come numero senza segno, sfasato di +127. Occorre tenerlo presente quando si ricostruisce il valore decimale.

float single precision (32 bit)

Standard IEEE 754

0	10100101	10011000000000000000000
$\pm$	exponent	fraction

- 1 bit per il segno
- 8 bit per l'esponente
- 23 bit per la mantissa

Attenzione

L'esponente viene memorizzato come numero senza segno, sfasato di +127. Occorre tenerlo presente quando si ricostruisce il valore decimale.

Mantissa: $2^{-1} + 2^{-4} + 2^{-5} = 0.5 + 0.0625 + 0.03125 = 0.59375$
Esponente+127: $2^0 + 2^2 + 2^5 + 2^7 = 1 + 4 + 32 + 128 = 165$
Esponente: $165 - 127 = 38$

$$+1.59375 \cdot 2^{38}$$

Operazioni in virgola fissa

Esegui le seguenti operazioni in binario senza segno, virgola fissa, rispettando come vincoli di rappresentazione 4 bit per la parte intera, 4 bit per la parte frazionaria

- $4.5 + 1.75$
- $3.875 + 2.125$

Operazioni in virgola fissa: $4.5 + 1.75$

Operazioni in virgola fissa: $4.5 + 1.75$

Convertiamo gli operandi in binario

- $4.5_{10} = 100.1_2$
- $1.75_{10} = 1.11_2$

Operazioni in virgola fissa: $4.5 + 1.75$

Convertiamo gli operandi in binario

- $4.5_{10} = 100.1_2$
- $1.75_{10} = 1.11_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0100.1000
- 0001.1100

Operazioni in virgola fissa: $4.5 + 1.75$

Convertiamo gli operandi in binario

- $4.5_{10} = 100.1_2$
- $1.75_{10} = 1.11_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0100.1000
- 0001.1100

Eseguiamo la somma

$$
\begin{array}{cccccccc c}
0 & 1 & 0 & 0 & .1 & 0 & 0 & 0 & + \\
0 & 0 & 0 & 1 & .1 & 1 & 0 & 0 & = \\
\hline
0 & 1 & 1 & 0 & .0 & 1 & 0 & 0 &
\end{array}
$$

Operazioni in virgola fissa: $4.5 + 1.75$

Convertiamo gli operandi in binario

- $4.5_{10} = 100.1_2$
- $1.75_{10} = 1.11_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0100.1000
- 0001.1100

Eseguiamo la somma

$$
\begin{array}{cccccccc c}
0 & 1 & 0 & 0 & .1 & 0 & 0 & 0 & + \\
0 & 0 & 0 & 1 & .1 & 1 & 0 & 0 & = \\
\hline
0 & 1 & 1 & 0 & .0 & 1 & 0 & 0 &
\end{array}
$$

$$
\begin{aligned}
100.1 + 1.11 &= 110.01 \\
4.5 + 1.75 &= 6.25
\end{aligned}
$$

Operazioni in virgola fissa: $3.875 + 2.125$

Operazioni in virgola fissa: $3.875 + 2.125$

Convertiamo gli operandi in binario

- $3.875_{10} = 11.111_2$
- $2.125_{10} = 10.001_2$

Operazioni in virgola fissa: $3.875 + 2.125$

Convertiamo gli operandi in binario

- $3.875_{10} = 11.111_2$
- $2.125_{10} = 10.001_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0011.1110
- 0010.0010

Operazioni in virgola fissa: $3.875 + 2.125$

Convertiamo gli operandi in binario

- $3.875_{10} = 11.111_2$
- $2.125_{10} = 10.001_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0011.1110
- 0010.0010

Eseguiamo la somma

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

Operazioni in virgola fissa: $3.875 + 2.125$

Convertiamo gli operandi in binario

- $3.875_{10} = 11.111_2$
- $2.125_{10} = 10.001_2$

Fissiamo la rappresentazione a 4+4 bit senza segno

- 0011.1110
- 0010.0010

Eseguiamo la somma

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

$$\begin{aligned} 11.111 + 10.001 &= 110 \\ 3.875 + 2.125 &= 6 \end{aligned}$$

Operazioni in virgola mobile

Eseguire le seguenti operazioni in virgola mobile (float single precision - standard 32 bit)

- $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$
- $1.5 \cdot 2^4 - 2 \cdot 2^5$

Operazioni in virgola mobile: $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$

Operazioni in virgola mobile: $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$

- $1.875_{10} = 1.111_2$
- $6.25_{10} = 110.01_2$
  - $6.25 \cdot 2^{23} = 1.5625 \cdot 2^{25}$

Operazioni in virgola mobile: $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$

- $1.875_{10} = 1.111_2$
- $6.25_{10} = 110.01_2$
  - $6.25 \cdot 2^{23} = 1.5625 \cdot 2^{25}$

$1.875 \cdot 2^5$	0	10000100	11100000000000000000000
$6.25 \cdot 2^{23}$	0	10011000	10010000000000000000000
	$\pm$	exp	mantissa

Operazioni in virgola mobile: $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$

- $1.875_{10} = 1.111_2$
- $6.25_{10} = 110.01_2$
  - $6.25 \cdot 2^{23} = 1.5625 \cdot 2^{25}$

$$\begin{array}{cccc}
1.875 \cdot 2^5 & 0 & 10000100 & 11100000000000000000000 \\
6.25 \cdot 2^{23} & 0 & 10011000 & 10010000000000000000000 \\
 & \pm & \text{exp} & \text{mantissa}
\end{array}$$

Portare tutte le mantisse al maggiore degli esponenti in gioco (25) tramite opportuni bit shift

$$\begin{array}{ll}
0. & 00000000000000000001111 \quad + \\
1. & 10010000000000000000000 \quad = \\
\hline
1. & 10010000000000000001111
\end{array}$$

Operazioni in virgola mobile: $1.875 \cdot 2^5 + 6.25 \cdot 2^{23}$

- $1.875_{10} = 1.111_2$
- $6.25_{10} = 110.01_2$
  - $6.25 \cdot 2^{23} = 1.5625 \cdot 2^{25}$

$1.875 \cdot 2^5$	0	10000100	11100000000000000000000
$6.25 \cdot 2^{23}$	0	10011000	10010000000000000000000
	$\pm$	exp	mantissa

Portare tutte le mantisse al maggiore degli esponenti in gioco (25) tramite opportuni bit shift

$$
\begin{array}{ll l}
0. & 00000000000000000001111 & + \\
1. & 10010000000000000000000 & = \\
\hline
1. & 10010000000000000001111 &
\end{array}
$$

0	10011000	10010000000000000001111
$\pm$	exp	mantissa

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (1/2)

- $1.5_{10} = 1.1_2$
- $2_{10} = 10_2$
  - $-2 \cdot 2^5 = -1 \cdot 2^6$

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (1/2)

- $1.5_{10} = 1.1_2$
- $2_{10} = 10_2$
  - $-2 \cdot 2^5 = -1 \cdot 2^6$

$1.5 \cdot 2^4$	0	10000011	10000000000000000000000
$2 \cdot 2^5$	1	10000101	00000000000000000000000
	$\pm$	exp	mantissa

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (1/2)

- $1.5_{10} = 1.1_2$
- $2_{10} = 10_2$
  - $-2 \cdot 2^5 = -1 \cdot 2^6$

$1.5 \cdot 2^4$	0	10000011	10000000000000000000000
$2 \cdot 2^5$	1	10000101	00000000000000000000000
	$\pm$	exp	mantissa

Si portano tutte le mantisse al maggiore degli esponenti in gioco (6); si calcola il complemento a 2 delle mantisse negative e si esegue la somma.

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (1/2)

- $1.5_{10} = 1.1_2$
- $2_{10} = 10_2$
    - $-2 \cdot 2^5 = -1 \cdot 2^6$

$1.5 \cdot 2^4$	0	10000011	10000000000000000000000
$2 \cdot 2^5$	1	10000101	00000000000000000000000
	$\pm$	exp	mantissa

Si portano tutte le mantisse al maggiore degli esponenti in gioco (6); si calcola il complemento a 2 delle mantisse negative e si esegue la somma.

$$\begin{array}{ll l}
0. & 01100000000000000000000 & + \\
1. & 00000000000000000000000 & = \\
\hline
1. & 01100000000000000000000 &
\end{array}$$

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (2/2)

Si calcola il risultato in notazione complemento a 2 e poi si converte in modulo e segno.[1] Per tornare in notazione modulo e segno:

[1]approfondimenti sulle operazioni in virgola mobile
`http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html`

[2]A priori si sa che il risultato sarà negativo perché l'operando negativo ha esponente maggiore

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (2/2)

Si calcola il risultato in notazione complemento a 2 e poi si converte in modulo e segno.[1] Per tornare in notazione modulo e segno:

1. si esegue un complemento a 2 (per ottenere il modulo)
2. si normalizza in modo da ottenere la forma 1.*mantissa*
3. si imposta il bit di segno[2]

---

[1] approfondimenti sulle operazioni in virgola mobile
`http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html`

[2] A priori si sa che il risultato sarà negativo perché l'operando negativo ha esponente maggiore

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (2/2)

Si calcola il risultato in notazione complemento a 2 e poi si converte in modulo e segno.[1] Per tornare in notazione modulo e segno:

1. si esegue un complemento a 2 (per ottenere il modulo)
2. si normalizza in modo da ottenere la forma 1.*mantissa*
3. si imposta il bit di segno[2]

$$\begin{aligned} Res_{C2} &= 0.10100000000000000000000 \cdot 2^6 \\ Res_{norm} &= 1.01000000000000000000000 \cdot 2^5 \end{aligned}$$

---

[1] approfondimenti sulle operazioni in virgola mobile
`http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html`

[2] A priori si sa che il risultato sarà negativo perché l'operando negativo ha esponente maggiore

Operazioni in virgola mobile: $1.5 \cdot 2^4 - 2 \cdot 2^5$ (2/2)

Si calcola il risultato in notazione complemento a 2 e poi si converte in modulo e segno.[1] Per tornare in notazione modulo e segno:

1. si esegue un complemento a 2 (per ottenere il modulo)
2. si normalizza in modo da ottenere la forma 1.*mantissa*
3. si imposta il bit di segno[2]

$$
\begin{aligned}
Res_{C2} &= 0.10100000000000000000000 \cdot 2^6 \\
Res_{norm} &= 1.01000000000000000000000 \cdot 2^5
\end{aligned}
$$

1	10000100	01000000000000000000000
±	exp	mantissa

[1] approfondimenti sulle operazioni in virgola mobile
`http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html`

[2] A priori si sa che il risultato sarà negativo perché l'operando negativo ha esponente maggiore

Queste slides contengono elementi tratti da materiale di Gerardo Pelosi redatto per il corso di Fondamenti di Informatica per Ingegneria dell'Automazione a.a. 2014/15.

Grazie per l'attenzione!

[3] http://people.freebsd.org/~phk/