

# Modulo: Approfondimenti sui Sistemi Aritmetici di un computer: tipo reale [P2\_03]

## Unità didattica: Errori di roundoff

[3-AT]

**Titolo:** Accuratezza statica e dinamica del S. A. Standard  
IEEE 754

Argomenti trattati:

- ✓ Misure di errore (errore assoluto ed errore relativo) e loro significato
- ✓ Errore di roundoff di rappresentazione (accuratezza statica)
- ✓ Errore di roundoff delle operazioni aritmetiche floating-point (accuratezza dinamica)
- ✓ Ottimalità del S.A. Standard: massima accuratezza statica e massima accuratezza dinamica

Prerequisiti richiesti: Rappresentazione dei numeri floating-point

# **PRECISIONE FINITA DEL SISTEMA ARITMETICO FLOATING-POINT**



**Errore di roundoff**

**statico**

**dinamico**

rappresentazione  
in memoria

risultato di opera-  
zioni aritmetiche

# Misure di errore

Se  $x$  indica il valore esatto ed  $\tilde{x}$  una sua approssimazione ( $\tilde{x} = fl(x)$ ), per misurare l'accuratezza di  $\tilde{x}$  rispetto ad  $x$  si usano:

*Errore assoluto*  $E_A(\tilde{x}) = |x - \tilde{x}|$

*Errore relativo*  $E_R(\tilde{x}) = \left| \frac{x - \tilde{x}}{x} \right|$  per  $x \neq 0$

# Quali informazioni danno gli errori $E_A$ ed $E_R$ sul grado di approssimazione?

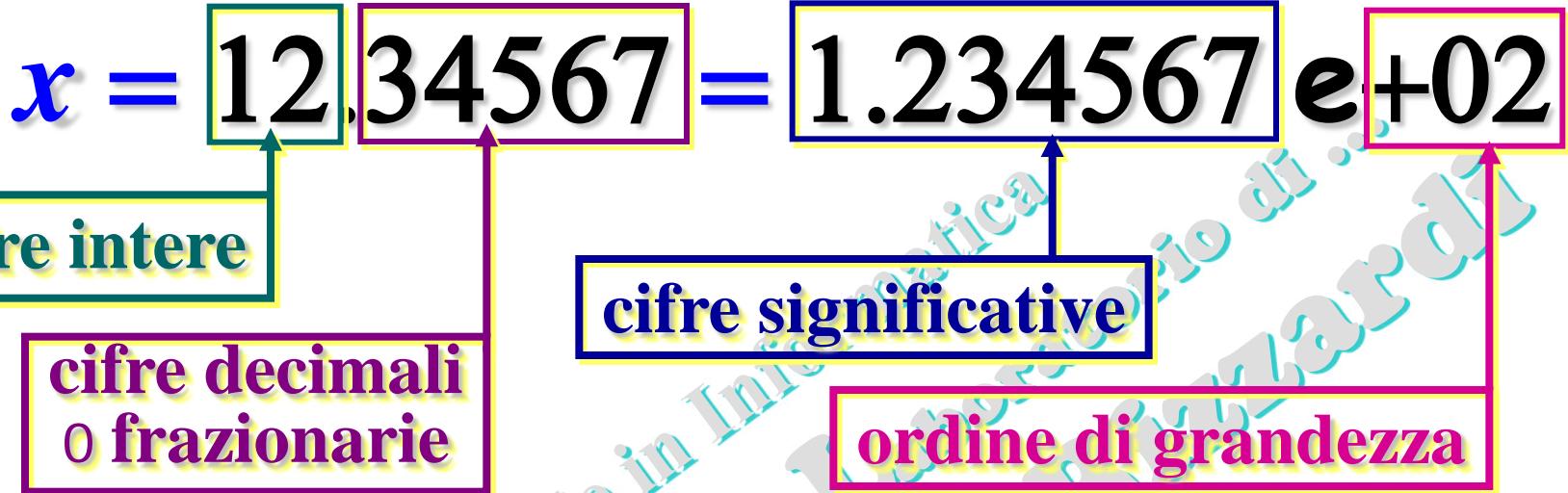
*Errore assoluto  $E_A$*

*cifre decimali corrette*

*Errore relativo  $E_R$*

*cifre significative corrette*

**... cosa significa ???**



### Proprietà 1

Se  $\tilde{x}$  è un'approssimazione di  $x$  corretta a  $p$  cifre decimali, allora si ha  $|x - \tilde{x}| < 10^{-p}$

### Proprietà 2

Se  $\tilde{x}$  è un'approssimazione di  $x$  corretta a  $p$  cifre significative, allora si ha

$$\frac{|\tilde{x} - x|}{|x|} < 10^{-p+1}$$

Anche se le due proprietà precedenti valgono per una sola implicazione ( $\Rightarrow$ ), nella pratica si suppone l'equivalenza ( $\iff$ ), nel senso che dall'ordine di grandezza degli errori si hanno informazioni sulle cifre (decimali o significative) corrette di un'approssimazione rispetto al corrispondente valore esatto.

# Esempio: il seguente programma

```
#include <stdio.h>
#include <math.h> // per pow() e atan() [arcotangente]

void main()
{double double_x, rel_err, ass_err; float float_x; char i;
 for (i=-4; i<5; i=i+2)
 {double_x=4*atan(1.0)*pow(10,i);
  float_x=(float) double_x;

  ass_err=fabs(double_x-float_x);
  rel_err=ass_err/fabs(double_x);

  printf("double = %22.16e\n",double_x);
  printf("single = %22.16e\n",float_x);
  printf("errore assoluto = %8.3e\n",ass_err);
  printf("errore relativo = %8.3e\n",rel_err);
  puts("\n\n");
 }
```

calcola gli *errori di rappresentazione* del tipo **float** ...

<b>double</b> (valore esatto!)	<b>float</b>	<b>double</b> (valore esatto!)
0. <u>00031415926</u> 535...	<u>3.1415926</u> 059...e-004	<u>3.1415926</u> 535...e-004
0. <u>03141592</u> 6535...	<u>3.141592</u> 8155...e-002	<u>3.141592</u> 6535...e-002
3. <u>141592</u> 6535...	<u>3.141592</u> 7410...e-000	<u>3.141592</u> 6535...e-000
314. <u>1592</u> 6535...	<u>3.141592</u> 7124...e+002	<u>3.141592</u> 6535...e+002
31415. <u>92</u> 6535...	<u>3.141592</u> 5781...e+004	<u>3.141592</u> 6535...e+004

P2\_03\_03.8

Tipo Reale Floating-Point

(prof. M. Rizzardi)

Errore ass.	Errore rel.
4.76...e-012	1.51...e-008
1.61...e-009	5.15...e-008
8.74...e-008	2.78...e-008
5.88...e-006	1.87...e-008
7.54...e-004	2.40...e-008

cifre significative corrette

# Errore di rappresentazione (tipo **float**)

Errore ass.	Errore rel.
4.76...e-012	1.51...e-008
1.61...e-009	5.15...e-008
8.74...e-008	2.78...e-008
5.88...e-006	1.87...e-008
7.54...e-004	2.40...e-008

Perché l'errore relativo ha sempre lo stesso ordine di grandezza?

Perché dipende dalle cifre significative della mantissa  
(nel tipo float del C la precisione è di 24 bit)!

Perché l'ordine di grandezza è  $10^{-8}$ ?

# precisione = numero di cifre <sub>$\beta$</sub> significative rappresentate

rappresentazione (binaria) fl.p.  
a bit implicito su  $t$  bit per la  
mantissa

$$\text{precisione (binaria)} = t + 1$$

Nell'Aritmetica Standard l'**epsilon macchina**  $\epsilon_{\text{mach}} = 2^{-t}$

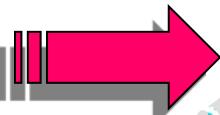
è tale che 
$$\frac{|x - \text{fl}(x)|}{|x|} < \frac{\epsilon_{\text{mach}}}{2}$$

## A quale precisione binaria corrisponde?

$$-\log_2 \left( \frac{|x - \text{fl}(x)|}{|x|} \right) \geq -\log_2 \left( \frac{\epsilon_{\text{mach}}}{2} \right) = t + 1$$

# precisione = numero di cifre <sub>$\beta$</sub> significative rappresentate

rappresentazione (binaria) fl.p.  
a bit implicito su  $t$  bit per la  
mantissa



$$\text{precisione (binaria)} = t + 1$$

E per la **precisione decimale equivalente?**

$$-\log_{10} \left( \frac{|x - fl(x)|}{|x|} \right) \geq -\log_{10} \left( \frac{\varepsilon_{\text{mach}}}{2} \right) \quad (\varepsilon_{\text{mach}} = 2^{-t})$$

$\beta=2$ , bit implicito

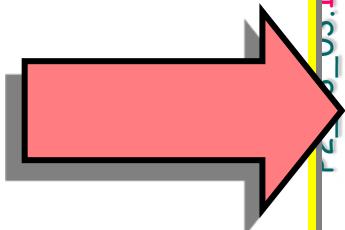
**precisione decimale equivalente**

float	$t=23$ bit	s.p.	$-\log_{10}(\dots) \approx 6.9 \approx 7 \div 8$ cifre decimali
double	$t=52$ bit	d.p.	$-\log_{10}(\dots) \approx 15.9 \approx 16 \div 17$ cifre decimali

perciò l'errore relativo nel tipo float è sempre dell'ordine di  $10^{-8}$

**... a che serve?**  
È inutile visualizzare  
più cifre della preci-  
sione decimale equi-  
valente

```
#include <stdio.h>
#include <stdlib.h>
void main()
{float a; double b;
 b=0.6666666666666666; a=(float)b;
 printf("a = %...          b = %...\n",a,b);
}
```



formato	s.p.	d.p.
%f	0.666667	0.666667
%e	6.666667e-001	6.666667e-001
%8.3f	0.667	0.667
%8.3e	6.667e-001	6.667e-001
%21.15f	0.66666686534882	0.66666666666667
%21.15e	6.666666865348816e-001	6.66666666666666e-001



$$\frac{|x - fl(x)|}{|x|}$$

# accuratezza statica

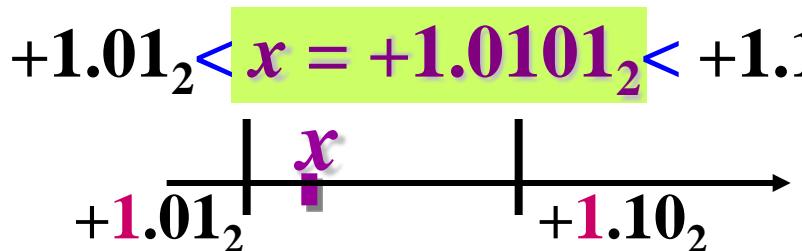
## Errore di roundoff di rappresentazione

È l'errore introdotto nel passare da un numero reale  $x$  a precisione infinita al suo rappresentante floating-point in memoria  $fl(x)$

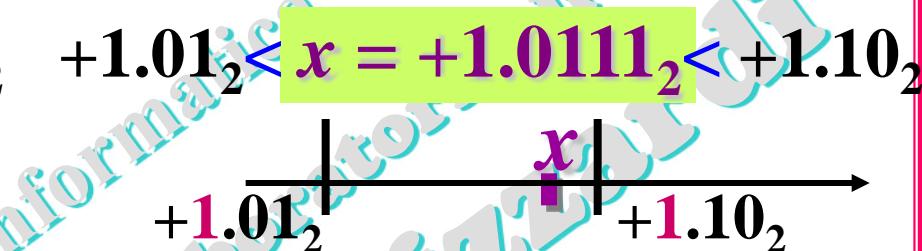
Dipende dalla precisione del Sistema Aritmetico Floating-point e dallo schema di *rounding* utilizzato

# Esempio: schemi IEEE St. 754

$x \notin F(\beta=2, t=2 \text{ (bit隐式), } E_{\min}=0, E_{\max}=3)$



$$fl(x) = \begin{cases} 1.01 \\ 1.01 \\ 1.10 \\ 1.01 \end{cases}$$



$$fl(x) = \begin{cases} 1.01 \\ 1.10 \\ 1.10 \\ 1.01 \end{cases}$$

L'errore è sempre il minimo per RN

$$|x - fl(x)| = \begin{cases} 0.0001 \\ 0.0001 \\ 0.0011 \\ 0.0001 \end{cases}$$

$$|x - fl(x)| = \begin{cases} 0.0011 \\ 0.0001 \\ 0.0001 \\ 0.0011 \end{cases}$$

$$|x - fl(x)| = \begin{cases} 0.0011 \\ 0.0001 \\ 0.0001 \\ 0.0011 \end{cases}$$

Proprietà matematica:  $\forall a \in \mathbb{R}, \forall \varepsilon > 0 \rightarrow a + \varepsilon > a$

Nel Sistema Aritmetico Floating-point non vale!

$$\forall a \in F(\beta, t, E_{\min}, E_{\max}), \exists \varepsilon > 0 : a \oplus \varepsilon = a$$

↑addizione floating-point

Fissato un numero floating-point  $a$ : qual è il più piccolo numero floating-point che dà contributo nell'addizione floating-point con  $a$ , cioè

$$ulp(a) = \min\{0 < \varepsilon \in F(\beta, t, E_{\min}, E_{\max}) : a \oplus \varepsilon > a\}$$

**u.l.p.** = Unit in the Last Place

# Approssimazione di *ulp*(1)

```
#include <stdio.h>
void main()
{ float eps, epsp1; int n;
  eps=1; n=0;
  epsp1=eps+1;
  while (epsp1>1)
  { eps=eps/2; n++;
    epsp1=eps+1;
  }
  eps=2.0f*eps; n--;
  printf("num. divisioni per 2=%d\nepsilon=%e \n",n,eps);
}
```

genera successione  
 $\{2^{-n}\}$

calcola  $\{1+2^{-n}\}$

num. divisioni per 2=23  
 $\epsilon$ =1.192093e-007

*u.l.p.(1) = Epsilon macchina  $\epsilon_{mach}$*

$$\epsilon_{mach} = \min\{0 < \epsilon \in F(\beta, t, E_{min}, E_{max}) : 1 \oplus \epsilon > 1\}$$

dove  $\oplus$  sta per *addizione floating-point*

bit隐式 + round to nearest →  $\epsilon_{mach} = 2^{-t}$

# Esempio: *Epsilon macchina* $\epsilon_{mach}$ in singola precisione

mostra\_float\_double.c

<b>FLT_EPSILON</b>	34000000	02_03_0017
<b>1.19...e-7</b>	0 011 0100 0 000 0000 0000 0000 0000 0000	
<b>+1</b>	3f800000 0 011 1111 1 000 0000 0000 0000 0000 0000	.Point
<b>1+FLT_EPS</b>	3f800001 0 011 1111 1 000 0000 0000 0000 0000 0001	Reale F
<b>1+.6FLT_EPS</b>	3f800001 0 011 1111 1 000 0000 0000 0000 0000 0001	
<b>1+.5FLT_EPS</b>	3f800000 0 011 1111 1 000 0000 0000 0000 0000 0000	M. Riz

# ESEMPIO 7b: *Epsilon macchina* $\varepsilon_{mach}$ in doppia precisione

**DBL\_EPSILON**

**2.22...e-16**

3cb00000 00000000

0	011 1100 1011	0000 0000 0000... 0000 0000
---	---------------	-----------------------------

+1

3ff00000 00000000

0	011 1111 1111	0000 0000 0000... 0000 0000
---	---------------	-----------------------------

**1+DBL\_EPS**

3ff00000 00000001

0	011 1111 1111	0000 0000 0000... 0000 0001
---	---------------	-----------------------------

**1+DBL\_EPS/2**

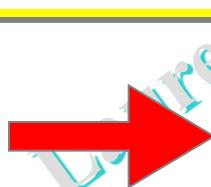
3ff00000 00000000

0	011 1111 1111	0000 0000 0000... 0000 0000
---	---------------	-----------------------------

# Se si elimina la variabile di appoggio `epspl` dal programma ...

```
#include <stdio.h>
void main()
{ float eps; int n;
  eps=1; n=0;
  while (eps+1>1)
    eps=eps/2; n++;
  eps=2.0f*eps; n--;
printf("num. divisioni per 2=%d\nepsilon=%e \n",n,eps);
}
```

epsilon macchina



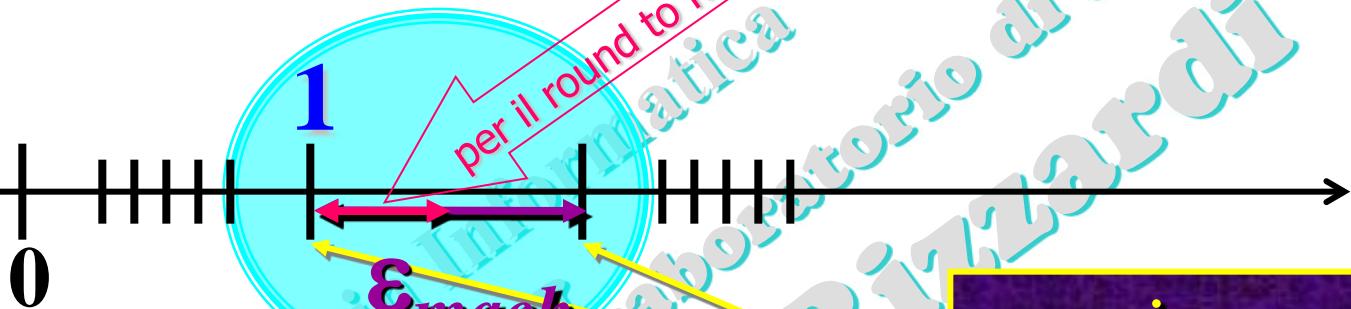
num. divisioni per 2=63  
epsilon=1.084202e-019

???

Senza la variabile di appoggio `epspl`, il ciclo `while` viene eseguito nei registri dell'unità aritmetica che hanno un campo mantissa maggiore di quello della memoria:

`precisione_registro = precisione_formato IEEE std Extended`  
(cioè del tipo `long double` del C).

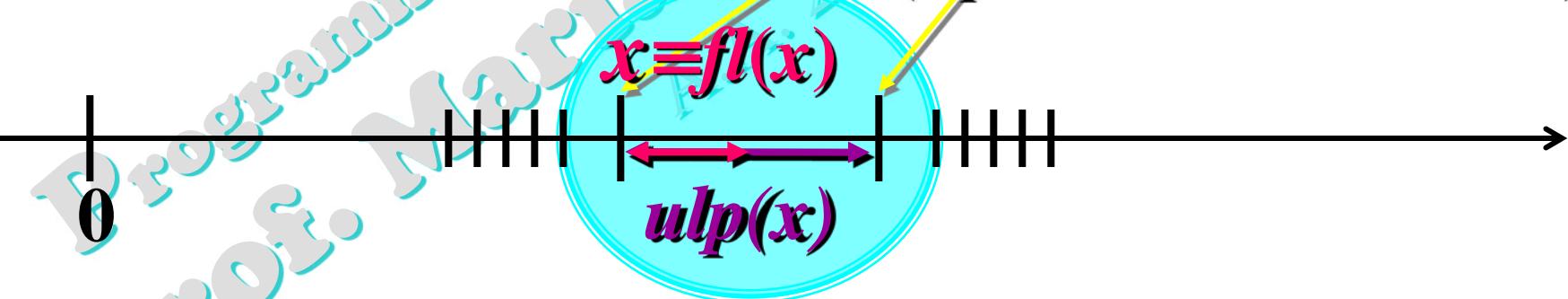
# Interpretazione dell'epsilon macchina $\varepsilon_{mach}$



numeri  
floating-point  
consecutivi

Epsilon macchina relativo ad  $x$  ( $ulp(x)$ )

( $ulp$  = Unit in the Last Place)



$$x = s \times (l \cdot m) \times 2^e \rightarrow ulp(x) = \varepsilon_{mach} \cdot 2^e$$

Nel S.A. Std., se  $x$  è un numero reale rappresentabile e  $fl(x)$  indica il corrispondente numero floating point;

$fl(x)$  risultato di massima accuratezza



$$fl(x) = x(1 + \delta),$$

$$|\delta| \leq \frac{1}{2} \epsilon_{mach}$$

dove  $\delta = \frac{|x - fl(x)|}{|x|}$ ,  $\delta$  errore relativo

Mediante il bit隐式 e lo schema del round to nearest, il **S.A. IEEE Std 754** assicura la massima accuratezza nella rappresentazione in memoria dei numeri reali (**massima accuratezza statica**).

Come assicurare nelle operazioni aritmetiche risultati di massima accuratezza (**massima accuratezza dinamica**) ?

# accuratezza dinamica

Si dimostra che nei registri dell'unità aritmetica per assicurare risultati di massima accuratezza (**massima accuratezza dinamica**) sono sufficienti:

- 1 bit per rappresentare il bit implicito ( $\ell$ )
- 2 guard-bit (g) oltre la mantissa
- 1 sticky-bit \*

\* sticky bit = bit che vale 1 se per esso transita almeno un bit=1



In tal modo con *registri lunghi* ( $t+3+1$  sticky) *bit* si ottengono gli stessi risultati aritmetici che si otterrebbero con registri di lunghezza ...“infinita” (economia di costi).

# Esempio 1: uso dei guard bit

cancellazione in s.p.  $x-y$

$$x=1 \quad 3f800000$$

0	011 1111 1	000 0000 0000 0000 0000 0000
---	------------	------------------------------

$$y=1-2^{-24} \quad 3f7fffff$$

0	011 1111 0	111 1111 1111 1111 1111 1111
---	------------	------------------------------

nei registri, dopo l'allineamento degli operandi ...

guard bit  
sticky bit

$x$	0	011 1111 1	1	000 0000 0000 0000 0000 0000	0	0	0
$y$	0	011 1111 1	0	111 1111 1111 1111 1111 1111	1	0	0
$x-y$	0	011 1111 1	0	000 0000 0000 0000 0000 0000	1	0	0
$x-y$	0	011 0011 1	1	000 0000 0000 0000 0000 0000	0	0	0

↔  
normalizzazione

# Esempio 2: uso dello sticky bit

## cancellazione in s.p. $x-y$

$x=1$	$3f800000$	
0	011 1111 1	000 0000 0000 0000 0000 0000
$y=(1+2^{-23})2^{-25}$		$33000001$
0	011 0011 0	000 0000 0000 0000 0000 0001

sticky bit  
guard bit

nei registri, dopo l'allineamento degli operandi ...

$x$	0	011 1111 1	1	000 0000 0000 0000 0000 0000	0	0	0
$y$	0	011 1111 1	0	000 0000 0000 0000 0000 0000	0	1	1
$x-y$	0	011 1111 1	0	111 1111 1111 1111 1111 1111	1	0	1
$x-y$	0	011 1111 0	1	111 1111 1111 1111 1111 1111	0	1	0

← = normalizzazione

Senza lo sticky-bit, nemmeno un registro a lunghezza doppia avrebbe fornito un risultato di massima accuratezza!

# Esercizi

1

Scrivere delle function C per calcolare rispettivamente l'epsilon macchina della singola, della doppia precisione e della precisione long double, visualizzando ad ogni passo i singoli bit.

Progetto:  
Esempio: singola precisione

n= 0	eps+1 = 0	10000000	00000000000000000000000000000000
n= 1	eps+1 = 0	01111111	10000000000000000000000000000000
n= 2	eps+1 = 0	01111111	01000000000000000000000000000000
n= 3	eps+1 = 0	01111111	00100000000000000000000000000000
n= 4	eps+1 = 0	01111111	00010000000000000000000000000000
n= 5	eps+1 = 0	01111111	00001000000000000000000000000000
n= 6	eps+1 = 0	01111111	00000100000000000000000000000000
n= 7	eps+1 = 0	01111111	00000010000000000000000000000000
n= 8	eps+1 = 0	01111111	00000001000000000000000000000000
n= 9	eps+1 = 0	01111111	00000000100000000000000000000000
n=10	eps+1 = 0	01111111	00000000010000000000000000000000
n=11	eps+1 = 0	01111111	00000000001000000000000000000000
n=12	eps+1 = 0	01111111	00000000000100000000000000000000
n=13	eps+1 = 0	01111111	00000000000010000000000000000000
n=14	eps+1 = 0	01111111	00000000000001000000000000000000
n=15	eps+1 = 0	01111111	00000000000000100000000000000000
n=16	eps+1 = 0	01111111	00000000000000010000000000000000
n=17	eps+1 = 0	01111111	00000000000000001000000000000000
n=18	eps+1 = 0	01111111	00000000000000000100000000000000
n=19	eps+1 = 0	01111111	00000000000000000010000000000000
n=20	eps+1 = 0	01111111	00000000000000000001000000000000
n=21	eps+1 = 0	01111111	00000000000000000000100000000000
n=22	eps+1 = 0	01111111	00000000000000000000010000000000
n=23	eps+1 = 0	01111111	00000000000000000000001000000000

2

Scrivere una function C per calcolare dalla definizione l'ULP(x) dove x è il parametro reale float di input. [liv. 3]

Generando in modo random i bit<sup>(\*)</sup> di un numero reale  $x$  (double  $x$ ), determinare i bit della corrispondente variabile  $\text{flx}$  (float  $\text{flx}$ ;  $\text{flx}=(\text{float}) x$ ). Se il numero  $x$  è rappresentabile nel tipo *float* calcolarne l'errore assoluto  $E_A$  e relativo  $E_R$  (considerando come esatto il double  $x$  e come approssimante il float  $\text{flx}$ ) dalle formule

$$E_A(\text{flx}) = |x - \text{flx}| \quad E_R(\text{flx}) = \frac{|x - \text{flx}|}{|x|}$$

[liv. 2]

(\*) La funzione C **rand()**, in **stdlib.h**, restituisce un intero pseudorandom minore o uguale di **RAND\_MAX** ( $= 32767_{10} = 7fff_{16} = 0111\ 1111\ 1111\ 1111_2$  - massimo intero positivo su 16 bit).

Come generare a caso i 64 bit di una variabile double?

- Generando i singoli bit random:
  - come resto della divisione per 2 [... come?];
  - associando 0 agli interi  $< 16384$  e 1 agli interi  $> 16383$ ;
- Generando 4 sequenze random di 16 bit ( $4 \times 16 = 64$ ) [... come?]

vedere: Uso di **rand()** in Materiale di supporto