Grupo ARCOS

uc3m | Universidad Carlos III de Madrid

Tema 2
Coma flotante

Estructura de Computadores

Grado en Ingeniería Informática

Contenidos

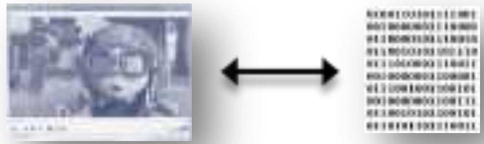
1. Introducción
   1. Objetivo
   2. Motivación
   3. Sistemas posicionales

2. **Representaciones**
   1. Alfanuméricas: caracteres y cadenas
   2. Numéricas: naturales y enteras
   3. **Numéricas: coma fija**
   4. Numéricas: coma flotante: estándar IEEE 754

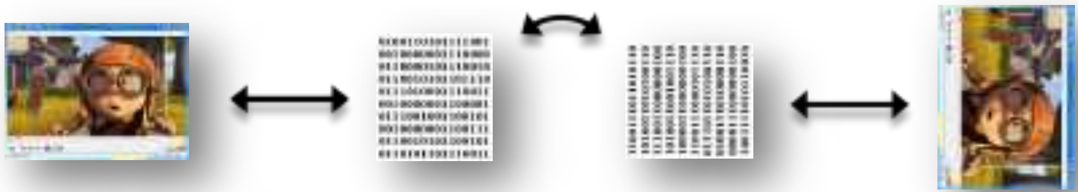
Recordatorio: necesitaremos…

- Conocer posibles representaciones:

- Conocer las características de las mismas:
  - Limitaciones



- Conocer cómo operar con la representación:



Félix García-Carballeira, Alejandro Calderón Mateos

Otras necesidades de representación

- ¿Cómo representar?
  - Números muy grandes: $30.556.926.000_{(10}$
  - Números muy pequeños: $0.0000000000529177_{(10}$
  - Números con decimales: 1,58567

Ejemplo de fallo…

- Explosión del **Ariane 5** (primer viaje)
  - Enviado por ESA en junio de 1996
  - Coste del desarrollo:
    10 años y **7000 millones de dólares**
  - Explotó 40 segundos después de despegar, a 3700 metros de altura.
  - Fallo debido a la pérdida total de la información de altitud:
    - El software del sistema de referencia inercial realizó la conversión de un valor real en coma flotante de 64 bits a un valor entero de 16 bits. El número a almacenar era mayor de 32767 (el mayor entero con signo de 16 bits) y se produjo un fallo de conversión y una excepción.

Coma fija [racionales]

- Se fija la posición de la coma binaria y se utilizan los pesos asociados a las posiciones decimales

- Ejemplo:

$$1001.1010 = 2^{4} + 2^{0} + 2^{-1} + 2^{-3} = 9{,}625$$

Félix García-Carballeira, Alejandro Calderón Mateos

Representación de fracciones con representación binaria en coma fija

- Ejemplo de representación con 6 bits:

$$xx.yyyy$$

$$2^{1} \quad 2^{0} \quad 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4}$$

$10{,}1010_{(2} = 1 \times 2^{1} + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.62510$

Asumiendo esta coma fija, el rango sería:

- 0 a 3.9375 (casi 4)

Potencias negativas

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.251/4	
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	

Contenidos

Notación científica decimal

$$\underbrace{9.12}_{\text{mantisa}}{}_{10} \times 10^{\overbrace{25}^{\text{exponente}}}$$

mantisa **base** **exponente**

- Cada número lleva asociado una mantisa y un exponente
- Notación científica decimal usada: notación normalizada
  - Solo un dígito distinto de 0 a la izquierda del punto
- Se adapta el número al orden de magnitud del valor a representar, trasladando la *coma decimal* mediante el exponente

Notación científica en binario

mantisa **exponente**

$$1.0_{(2} \text{ x } 2^{-1}$$

coma binaria **base**

- Forma normalizada: Un 1 (solo un dígito) a la izquierda de la coma
  - Normalizada: 1.0001 x 2-9
  - No normalizada: $0.0011 \text{ x } 2^{-8}$, $10.0 \text{ x } 2^{-10}$

Félix García-Carballeira, Alejandro Calderón Mateos

Estándar IEEE 754
[racionales]

1 0010...0101 00010...001

signo **exponente** **mantisa**

- Estándar para coma flotante usado en la mayoría de los ordenadores.
- **Características** (salvo casos especiales):
  - Exponente: en exceso con sesgo $2^{\text{num\_bits\_exponente} - 1} - 1$
  - Mantisa: signo-magnitud, normalizada, con bit implícito
- Diferentes **formatos**:
  - **Precisión simple**: 32 bits (signo: 1, exponente: 8 y mantisa: 23)
  - **Doble precisión**: 64 bits (signo: 1, exponente: 11 y mantisa: 52)
  - **Cuádruple precisión**: 128 bits (signo: 1, exponente: 15 y mantisa: 112)

http://speleotrove.com/decimal/

Números normalizados

- En este estándar los números a representar tienen que estar normalizados. Un número normalizado es de la forma:
  - $1{,}bbbbbbb \times 2^{e}$
  - mantisa: 1,bbbbbb (siendo b = 0, 1)
  - 2 es la base del exponente
  - e es el exponente

Normalización y bit implícito

- Normalización
  Para normalizar la mantisa se ajusta el exponente
  para que el bit más significativo de la mantisa sea 1
  - Ejemplo: **1**00100000000000000000000 x 2^3 (ya está normalizado)
  - Ejemplo: ~~000~~1000000000010101 x 2^3 (no lo está)
    1000000000010101000 x 2^0 (ahora sí)

- Bit implícito
  Una vez normalizado, dado que el bit más significativo es 1, **no** se almacena para dejar espacio para un bit más (aumenta la precisión)
  - Así se puede representar mantisas con un bit más

Estándar IEEE 754 de precisión simple [racionales]

bits	0	1 … 8	9 … 31
	S	E	M

S es el signo (1 bit)
E es el exponente (8 bits)
M es la mantisa (23 bits)

- El valor se calcula con la siguiente expresión (salvo casos especiales):

$$\mathbf{N = (-1)^S \times 2^{E-127} \times 1.M}$$

donde:

S = 0 indica número positivo, S =1 indica número negativo

$0 < E < 255$ (E=0 y E=255 indican casos especiales)

$00000000000000000000000 \leq M \leq 11111111111111111111111$

Estándar IEEE 754 de precisión simple [racionales]

- Existencia de casos especiales:

$(-1)^s * 0.mantisa * 2^{-126}$

Exponente	Mantisa	Valor especial
0 (0000 0000)	0	+/- 0 (según signo)
0 (0000 0000)	No cero	Número NO normalizado
255 (1111 1111)	No cero	NaN (0/0,...)
255 (1111 1111)	0	+/-infinito (según signo)
1-254	Cualquiera	Número normalizado (no especial)

$(-1)^s * 1.mantisa * 2^{exponente-127}$

Félix García-Carballeira, Alejandro Calderón Mateos

Ejemplos (incluyen casos especiales)

S	E	M	N
1	00000000	00000000000000000000000	-0 (Excepción 0) E=0 y M=0.
1	01111111	00000000000000000000000	$-2^0 \times 1.0_2 = -1$
0	10000001	11100000000000000000000	$+2^2 \times 1.111_2 = +2^2 \times (2^0+2^{-1}+2^{-2}+2^{-3}) = +7.5$
0	11111111	00000000000000000000000	∞ (Excepción ∞) E=255 y M=0
0	11111111	10000000000000000000001	NaN (Not a Number, como la raíz cuadrada de un número negativo) E=255 y M≠0.

Ejercicio

a) Calcular el valor correspondiente al número
0 10000011 11000000000000000000000
dado en coma flotante según norma 754 de simple precisión

Félix García-Carballeira, Alejandro Calderón Mateos

Ejercicio (solución)

a) Calcular el valor correspondiente al número
0 10000011 11000000000000000000000
dado en coma flotante según norma 754 de simple precisión

> a) Bit de signo: 0 $\Rightarrow (-1)^0 = +1$
> b) Exponente: $10000011_2 = 131_{10} \Rightarrow E - 127 = 131 - 127 = 4$
> c) Mantisa: 11000000000000000000000 $\Rightarrow 1 \times 2^{-1} + 1 \times 2^{-2} = 0{,}75$
>
> Por tanto el valor decimal del nº es $+1 \times 2^4 \times 1{,}75 = +28$

Ejercicio

b) Expresar según norma IEEE 754 de simple precisión el nº -9

Ejercicio (solución)

b) Expresar según norma IEEE 754 de simple precisión el nº -9

$$-9_{10} = -1001_2 = -1001_2 \times 2^0 = -1{,}001_2 \times 2^3 \text{ (mantisa normalizada)}$$

a) Bit de signo: negativo ⇨ S=1
b) Exponente: 3+127 (exceso) = 130 ⇨ 10000010
c) Mantisa: 1,001 (bit impl.) ⇨ 00100000000000000000000

Por tanto -9 = 1 10000010 00100000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Estándar IEEE 754 de precisión simple [racionales]

- Rango de magnitudes representables (sin considerar el signo):
  - Menor normalizado:
    $2^{1-127} \times 1.00000000000000000000000_2$
  - Mayor normalizado:
    $2^{254-127} \times 1.11111111111111111111111_2$
  - Menor no normalizado:
    $2^{-126} \times 0.00000000000000000000001_2$
  - Mayor no normalizado:
    $2^{-126} \times 0.11111111111111111111111_2$

$$(-1)^s * 0.\text{mantisa} * 2^{-126}$$

Exponente	Mantisa	Valor especial
0	**$\neq$ 0**	**No normalizado**
1-254	**cualquiera**	**normalizado**

$$(-1)^s * 1.\text{mantisa} * 2^{\text{exponente}-127}$$

Félix García-Carballeira, Alejandro Calderón Mateos

Estándar IEEE 754 de precisión simple [racionales]

- Rango de magnitudes representables (sin considerar el signo):
  - Menor normalizado:
    $2^{1-127} \times 1.00000000000000000000000_2 = 2^{-126}$
  - Mayor normalizado:
    $2^{254-127} \times 1.11111111111111111111111_2 = 2^{127} \times (2 - 2^{-23}) = 2^{128} \times (1 - 2^{-24})$
  - Menor no normalizado:
    $2^{-126} \times 0.00000000000000000000001_2 = 2^{-149}$
  - Mayor no normalizado:
    $2^{-126} \times 0.11111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$

Truco:

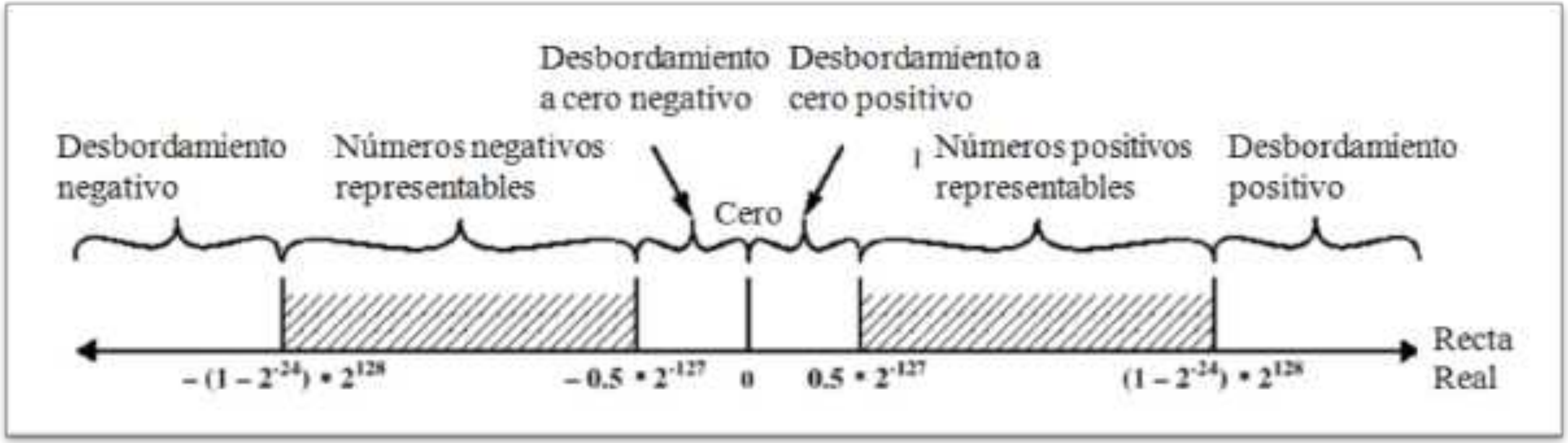
$$
\begin{array}{rl}
1.11111111111111111111111_2 & = X \\
+ \quad 0.00000000000000000000001_2 & = 2^{-23} \\
\hline
10.00000000000000000000000_2 & = 2
\end{array}
$$

$$X = 2 - 2^{-23}$$

Félix García-Carballeira, Alejandro Calderón Mateos

Estándar IEEE 754 de precisión simple [racionales]

- Rango de magnitudes representables (sin considerar el signo):
  - Menor normalizado:
    $2^{1-127} \times 1.00000000000000000000000_2 = 2^{-126} = 2^{-127} \times 0.5$
  - Mayor normalizado:
    $2^{254-127} \times 1.11111111111111111111111_2 = 2^{127} \times (2 - 2^{-23}) = 2^{128} \times (1 - 2^{-24})$
  - Menor no normalizado:
    $2^{-126} \times 0.00000000000000000000001_2 = 2^{-149}$
  - Mayor no normalizado:
    $2^{-126} \times 0.11111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$



Ejercicio

- ¿Cuántos números de *floats* (coma flotante de simple precisión) hay entre el 1 y el 2 (no incluido)?

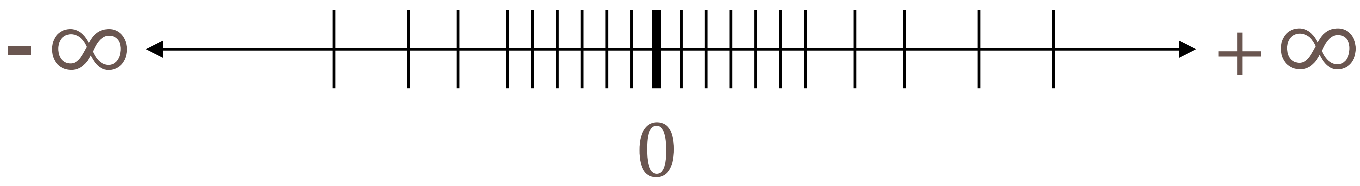
- ¿Cuántos números de *floats* (coma flotante de simple precisión) hay entre el 2 y el 3 (no incluido)?

Ejercicio (solución)

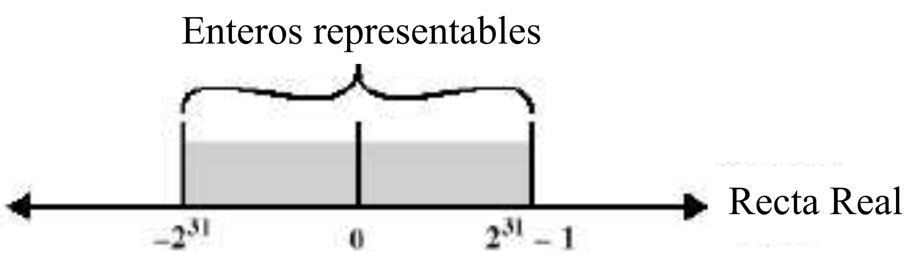
- ¿Cuántos números de *floats* (coma flotante de simple precisión) hay entre el 1 y el 2 (no incluido)?
  - 1 = 1,00000000000000000000000 x 2^0
  - 2 = 1,00000000000000000000000 x 2^1
  - Entre 1 y 2 hay 2^{23} números
- ¿Cuántos números de *floats* (coma flotante de simple precisión) hay entre el 2 y el 3 (no incluido)?
  - 2 = 1,00000000000000000000000 x 2^1
  - 3 = 1,10000000000000000000000 x 2^1
  - Entre 2 y 3 hay 2^{22} números

Números representables

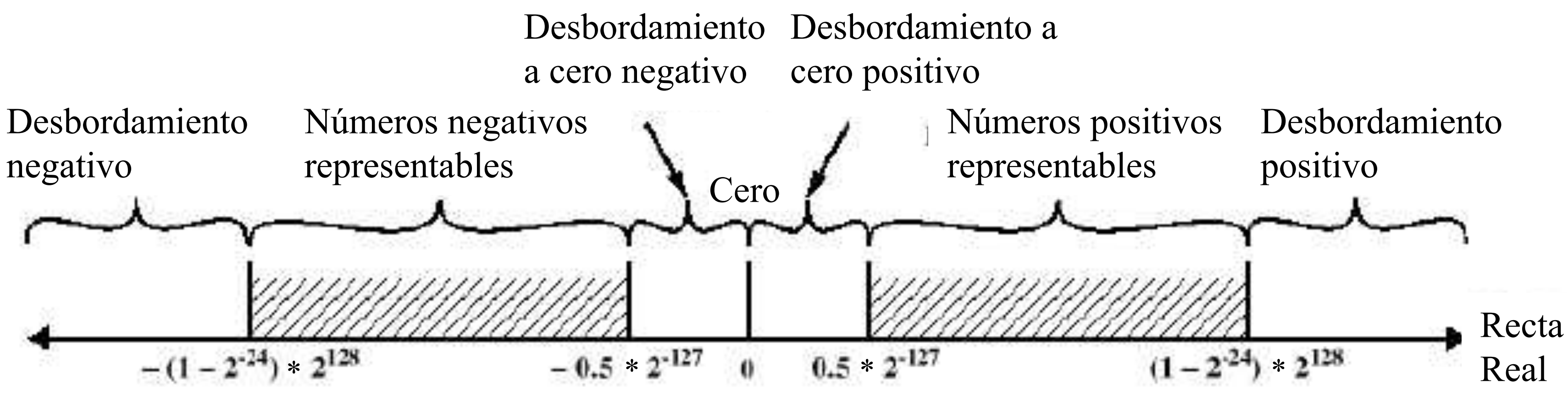
- Resolución variable:
  Más denso cerca de cero, menos hacia el infinito



Números representables



(a) Enteros en complemento a dos



(b) Números en coma flotante

Ejemplo 1
imprecisión

0,4 →

0	01111101	10011001100110011001101

↓

3.9999998 x 10^{-1}

0,1 →

0	01111011	10011001100110011001100

↓

9.9999994 x 10^{-2}

Ejemplo 2
imprecisión

- ¿Cómo realiza C una división?

t2.c

```
#include <stdio.h>

int main ( )
{
   float a ;

   a = 3.0/7.0 ;
   if (a == 3.0/7.0)
          printf("Igual\n") ;
   else printf("No Igual\n") ;
   return (0) ;
}
```


Félix García-Carballeira, Alejandro Calderón Mateos

Ejemplo 2
imprecisión

- ¿Cómo realiza C una división?

t2.c

```c
#include <stdio.h>

int main ( )
{
   float a ;

   a = 3.0/7.0 ;
   if (a == 3.0/7.0)
          printf("Igual\n") ;
   else printf("No Igual\n") ;
   return (0) ;
}
```

```
$ gcc -o t2 t2.c
$ ./t2
No Igual
```

Ejemplo 2
imprecisión

- ¿Cómo realiza C una división?

t2.c

```
#include <stdio.h>

int main ( )
{
   float a ;

   a = 3.0/7.0 ;
   if (a == 3.0/7.0)
          printf("Igual\n") ;
   else printf("No Igual\n") ;
   return (0) ;
}
```

float

double

```
$ gcc -o t2 t2.c
$ ./t2
No Igual
```

Ejemplo 3
imprecisión

- La propiedad asociativa no siempre se cumple

¿ a + (b + c) = (a + b) + c ?

t1.c

```
#include <stdio.h>

int main ( )
{
      float x, y, z ;

      x = 10e30;   y = -10e30;   z = 1;
      printf("(x+y)+z = %f\n",(x+y)+z) ;
      printf("x+(y+z) = %f\n",x+(y+z)) ;

      return (0) ;
}
```

Ejemplo 3
imprecisión

- La propiedad asociativa no siempre se cumple

¿ a + (b + c) = (a + b) + c ?

t1.c

```c
#include <stdio.h>

int main ( )
{
      float x, y, z ;

      x = 10e30;   y = -10e30;   z = 1;
      printf("(x+y)+z = %f\n",(x+y)+z) ;
      printf("x+(y+z) = %f\n",x+(y+z)) ;

      return (0) ;
}
```

```
$ gcc -o t1 t1.c
$ ./t1
(x+y)+z = 1.000000
x+(y+z) = 0.000000
```

Redondeo

- El redondeo elimina cifras menos significativas de un número para obtener un valor aproximado.

- Tipos de redondeo:
  - Redondeo hacia + ∞
    - Redondeo "hacia arriba": 2.001 → 3, -2.001 → -2
  - Redondeo hacia - ∞
    - Redondea "hacia abajo": 1.999 → 1, -1.999 → -2
  - Truncar
    - Descarta los últimos bits: 1.299 → 1.2
  - Redondeo al más cercano
    - 2.4 → 2, 2.6 → 3, -1.4 → -1

Redondeo

- El redondeo supone ir perdiendo precisión.
- El redondeo ocurre:
  - Al pasar a una representación con menos representables:
    - Ej.: Un valor de doble a simple precisión
    - Ej.: Un valor en coma flotante a entero
  - Al realizar operaciones aritméticas:
    - Ej.: Después de sumar dos números en coma flotante (al usar dígitos de guarda)

Dígitos de guarda

- Se utilizan dígitos de guarda para mejorar la precisión: internamente se usan dígitos adicionales para operar.
- Ejemplo: $2{,}65 \times 10^0 + 2.34 \times 10^2$

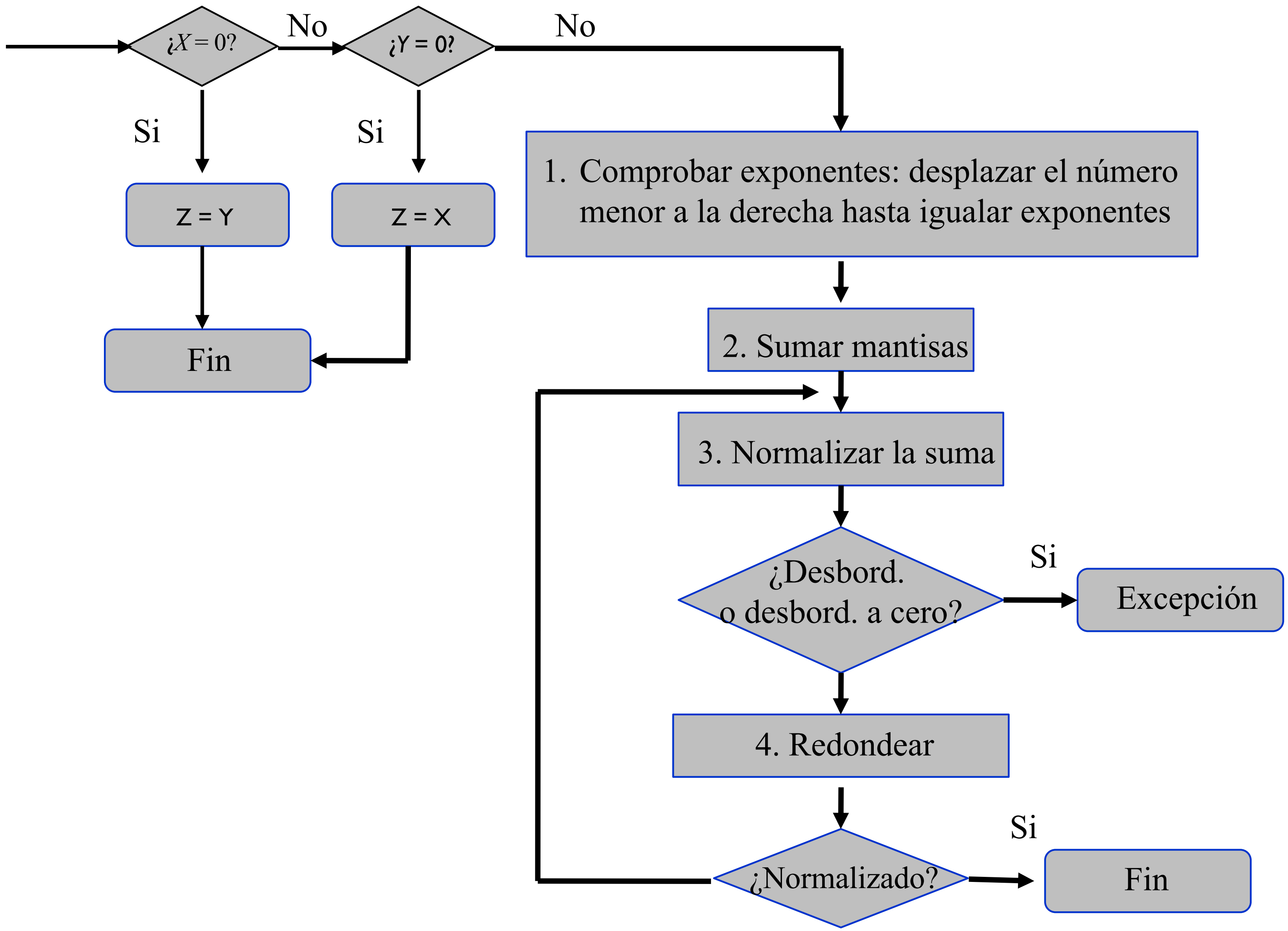
	SIN dígitos de guarda	CON dígitos de guarda
1.- igualar exponentes	$0{,}02 \times 10^2$ $+\ 2{,}34 \times 10^2$	$0{,}0265 \times 10^2$ $+\ 2{,}3400 \times 10^2$
2.- sumar	$2{,}36 \times 10^2$	$2{,}3665 \times 10^2$
3.- redondear	$2{,}36 \times 10^2$	$2{,}37 \times 10^2$

Operaciones en coma flotante

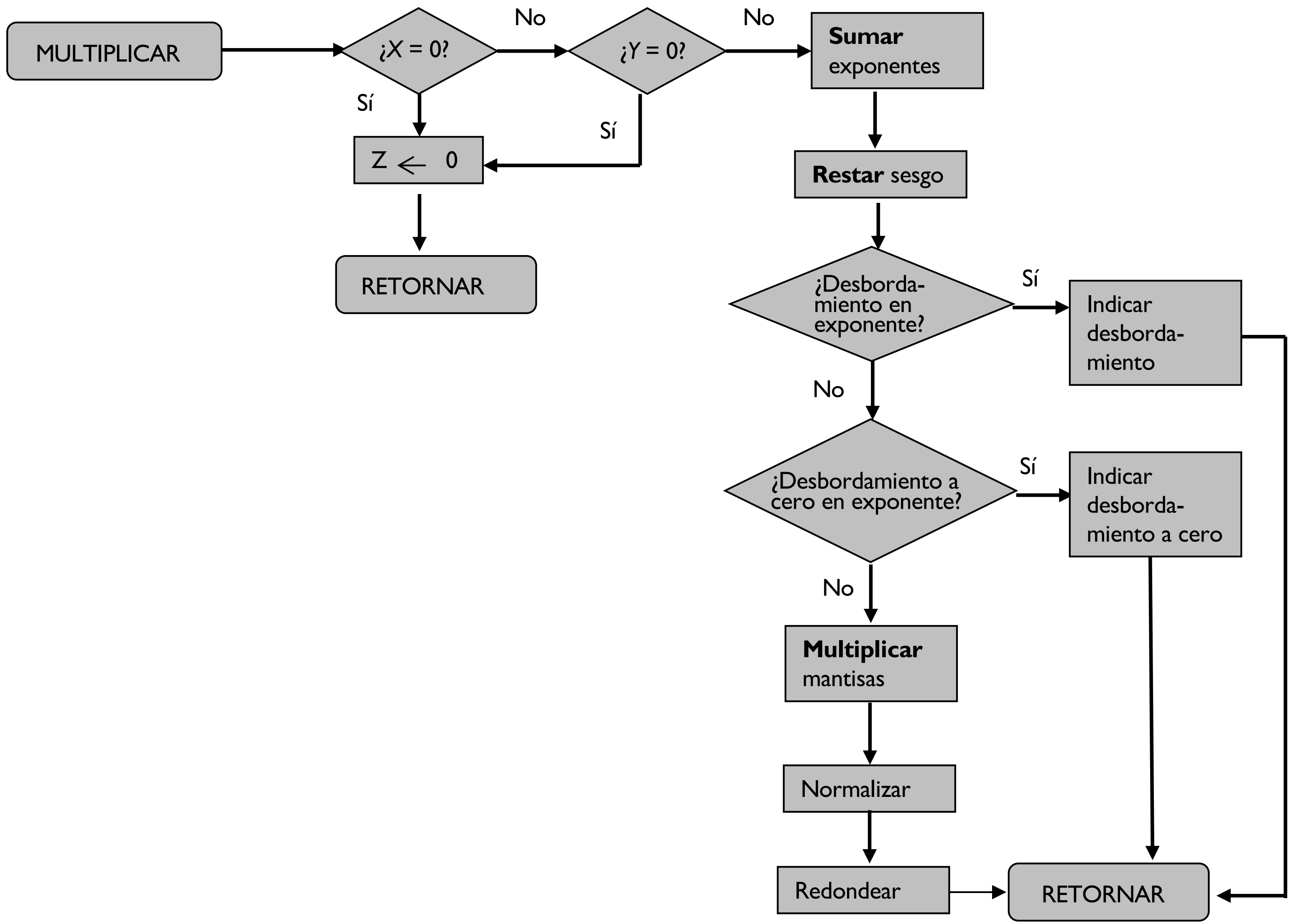
- Sumar
- Restar
  1. Comprobar valores cero.
  2. Igualar exponentes (desplazar número menor a la derecha).
  3. Sumar/restar las mantisas.
  4. Normalizar el resultado.

- Multiplicar
- Dividir
  1. Comprobar valores cero.
  2. Sumar/restar exponentes.
  3. Multiplicar/dividir mantisas (teniendo en cuenta el signo).
  4. Normalizar el resultado.
  5. Redondear el resultado.

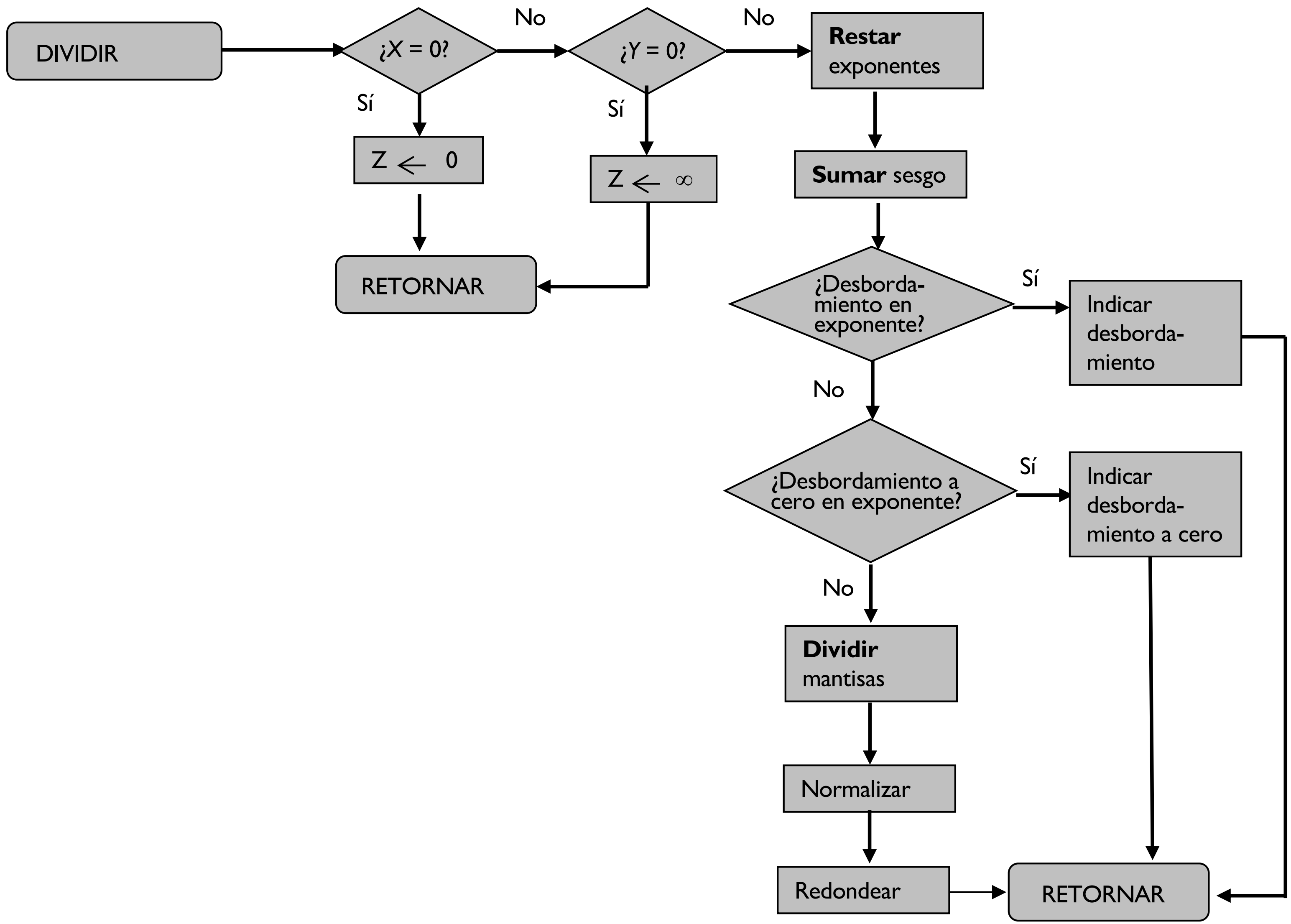
Suma y resta: Z=X+Y y Z=X-Y



Multiplicación: Z=X*Y



División: Z=X/Y



Ejercicio

- Usando el formato IEEE 754, sumar 7,5 y 1,5 paso a paso

Solución

1) $7{,}5 + 1{,}5 =$
2) $1{,}111 * 2^2 + 1{,}1 * 2^0 =$
3) $1{,}111 * 2^2 + 0{,}011 * 2^2 =$
4) $10{,}010 * 2^2 =$
5) $1{,}0010 * 2^3$

Pasar a binario

Igualar exponentes

Sumar

Ajustar exponentes

Solución

- Representación de los números

7,5 → 0 10000001 11100000000000000000000

\+ **1,5 →** 0 01111111 10000000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Se separa exponentes y mantisas y se añade el bit implícito

7,5 →	0	10000001	1.1110000000000000000000
+ **1,5** →	0	01111111	1.1000000000000000000000

Solución

- Igualar exponentes

7,5 →	0	10000001	1.11100000000000000000000
+ 1,5 →	0	01111111	1.10000000000000000000000

Solución

- Igualar exponentes

7,5 →	0	10000001	1.11100000000000000000000
		+1	1
+ **1,5** →	0	01111111	1.10000000000000000000000

Solución

- Igualar exponentes

7,5 →	0	10000001	1.11100000000000000000000
		+1	1
+ **1,5** →	0	10000000	0.110000000000000000000000

Solución

- Igualar exponentes

7,5 →	0	10000001	1.11100000000000000000000
		+1	1
+ 1,5 →	0	10000001	0.01100000000000000000000

Solución

- Sumar mantisas

7,5 →	0	10000001	1.11100000000000000000000
+ 1,5 →	0	10000001	0.01100000000000000000000

Solución

- Normalizar el resultado

	Signo	Exponente	Mantisa
7,5 →	0	10000001	1.11100000000000000000000
+ **1,5** →	0	10000001	0.01100000000000000000000
9 →	0	10000001	10.01000000000000000000000

Se produce un acarreo, mantisa no normalizada

Solución

- Normalizar el resultado

7,5 →	0	10000001	1.11100000000000000000000
+ 1,5 →	0	10000001	0.01100000000000000000000
9 →	0	10000001	10.01000000000000000000000

+1 ⟵⟶ 1

Solución

7,5 →	0	10000001	1.11100000000000000000000
+ **1,5** →	0	10000001	0.01100000000000000000000
9 →	0	10000010	1.00100000000000000000000

Solución

- Se almacena el resultado eliminando el bit implícito

9 → 0 10000010 00100000000000000000000

Ejercicio

- Usando el formato IEEE 754, calcular 9 – 7.5 paso a paso

Solución

- Representación de los números

9 → 0 10000010 00100000000000000000000

- 7,5→ 1 10000001 11100000000000000000000

Solución

- Se separan exponentes y mantisas y se añade bit implícito

9 →	0	10000010	1.00100000000000000000000
- 7,5 →	1	10000001	1. 11100000000000000000000

Se añade el bit implícito para operar

Solución

- Igualar exponentes

9 →	0	10000010	1.00100000000000000000000
- **7,5→**	1	10000001	1. 11100000000000000000000

Solución

- Igualar exponentes

9 →	0	10000010	1.00100000000000000000000
		+1	1
- 7,5 →	1	10000001	1. 11100000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Igualar exponentes

	Signo	Exponente	Mantisa
9 →	0	10000010	1.00100000000000000000000
- 7,5→	1	10000010	0.11110000000000000000000

+1 1

Solución

- Resta

9 →	0	10000010	1.00100000000000000000000
- 7,5→	1	10000010	0.11110000000000000000000
1,5→	0	10000010	0.00110000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Normalizar el resultado

9 →	0	10000010	1.00100000000000000000000
- 7,5 →	1	10000010	0.11110000000000000000000
1,5 →	0	10000010	0.00110000000000000000000

Solución

- Normalizar el resultado

9 →	0	10000010	1.00100000000000000000000
- 7,5→	1	10000010	0.11110000000000000000000

-1 1

1,5→	0	10000001	0.01100000000000000000000

Solución

- Normalizar el resultado

9 →	0	10000010	1.00100000000000000000000
- 7,5 →	1	10000010	0.11110000000000000000000
		-1	1
1,5 →	0	10000000	0.11000000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Normalizar el resultado

	S	Exponente	Mantisa
9 →	0	10000010	1.00100000000000000000000
- 7,5 →	1	10000010	0.11110000000000000000000
1,5 →	0	01111111	1.10000000000000000000000

mantisa ya normalizada

Solución

- Se almacena el resultado definitivo eliminando el bit implícito

1,5→ 0 01111111 10000000000000000000000

Ejercicio

- Usando el formato IEEE 754, multiplicar 7,5 y 1,5 paso a paso

Solución

- Representación de los números

9 → 0 10000010 00100000000000000000000

- 7,5→ 1 10000001 11100000000000000000000

Solución

- Se separan exponentes y mantisas y se añade bit implícito

	7,5 →	0	10000001	1.11100000000000000000000
X	**1,5 →**	0	01111111	1.1000000000000000000000000

Se añade el bit implícito para operar

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Multiplicar: sumar exponentes y multiplicar mantisas

	7,5 →	0	10000001	1.11100000000000000000000
X	1,5 →	0	01111111	1.10000000000000000000000
			+	×
		0	100000000	10.1101000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Multiplicar: quitar el sesgo al exponente (hay dos)

7,5 →	0	10000001	1.11100000000000000000000
X **1,5** →	0	01111111	1.10000000000000000000000
	0	100000000	10.110100000000000000000000
		- 01111111	

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Multiplicar: normalizar el resultado

7,5 →	0	10000001	1.11100000000000000000000
X **1,5** →	0	01111111	1.10000000000000000000000
		+1 ←	→ 1
11,25	0	10000001	10.11010000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Resultado normalizado…

	7,5 →	0	10000001	1.11100000000000000000000
X	1,5 →	0	01111111	1.10000000000000000000000
	11,25	0	10000010	1.011010000000000000000000

Félix García-Carballeira, Alejandro Calderón Mateos

Solución

- Se almacena el resultado eliminando el bit implícito

11,25 0 10000010 01101000000000000000000

Evolución de IEEE 754

- 1985 – IEEE 754
- 2008 – IEEE 754-2008 (754+854)
- 2011 – ISO/IEC/IEEE 60559:2011 (754-2008)

Name	Common name	Base	Digits	E min	E max	Notes	Decimal digits	Decimal E max
binary16	Half precision	2	10+1	−14	+15	storage, not basic	3.31	4.51
binary32	Single precision	2	23+1	−126	+127		7.22	38.23
binary64	Double precision	2	52+1	−1022	+1023		15.95	307.95
binary128	Quadruple precision	2	112+1	−16382	+16383		34.02	4931.77
decimal32		10	7	−95	+96	storage, not basic	7	96
decimal64		10	16	−383	+384		16	384
decimal128		10	34	−6143	+6144		34	6144

http://en.wikipedia.org/wiki/IEEE_floating_point

Asociatividad

- La coma flotante no es asociativa
  - $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, y $z = 1.0$

  - $x + (y + z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0)$
    $= -1.5 \times 10^{38} + (1.5 \times 10^{38}) = 0.0$
  - $(x + y) + z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0$
    $= (0.0) + 1.0 = 1.0$

- Las operaciones coma flotante no son asociativas
  - Los resultados son aproximados
  - 1.5×10^{38} es mucho más grande que 1.0
  - $1.5 \times 10^{38} + 1.0$ en la representación en coma flotante sigue siendo 1.5×10^{38}

Conversión int → float → int

```
if (i == (int)((float) i)) {
      printf("true");
}
```

- No siempre es cierto
- Muchos valores enteros grandes no tienen una representación exacta en coma flotante
- ¿Qué ocurre con `double`?

Ejemplo

- El número 133000405 en binario es:
  - 111111011010110110011010101 (27 bits)
- $111111011010110110011010101 \times 2^0$
- Se normaliza
  - $1,11111011010110110011010101 \times 2^{26}$
  - S = 0 (positivo)
  - $e = 26 \rightarrow \quad E = 26 + 127 = 153$
  - M = 11111011010110110011010 (se pierden los 3 últimos bits)
- El número realmente almacenado es
  - $1,11111011010110110011010 \times 2^{26} =$
  - $111111011010110110011010 \times 2^3 = 133000400$

Conversión float → int → float

```
if (f == (float)((int) f)) {
 printf(“true”);
}
```

- No siempre es cierto
- Los números con decimales no tienen representación entera