Grupo ARCOS

uc3m | Universidad Carlos III de Madrid

Tema 2: Representación de la información (2/2)

Estructura de Computadores

Grado en Ingeniería Informática
Grado en Matemática aplicada y Computación
Doble Grado en Ingeniería Informática y Administración de Empresas

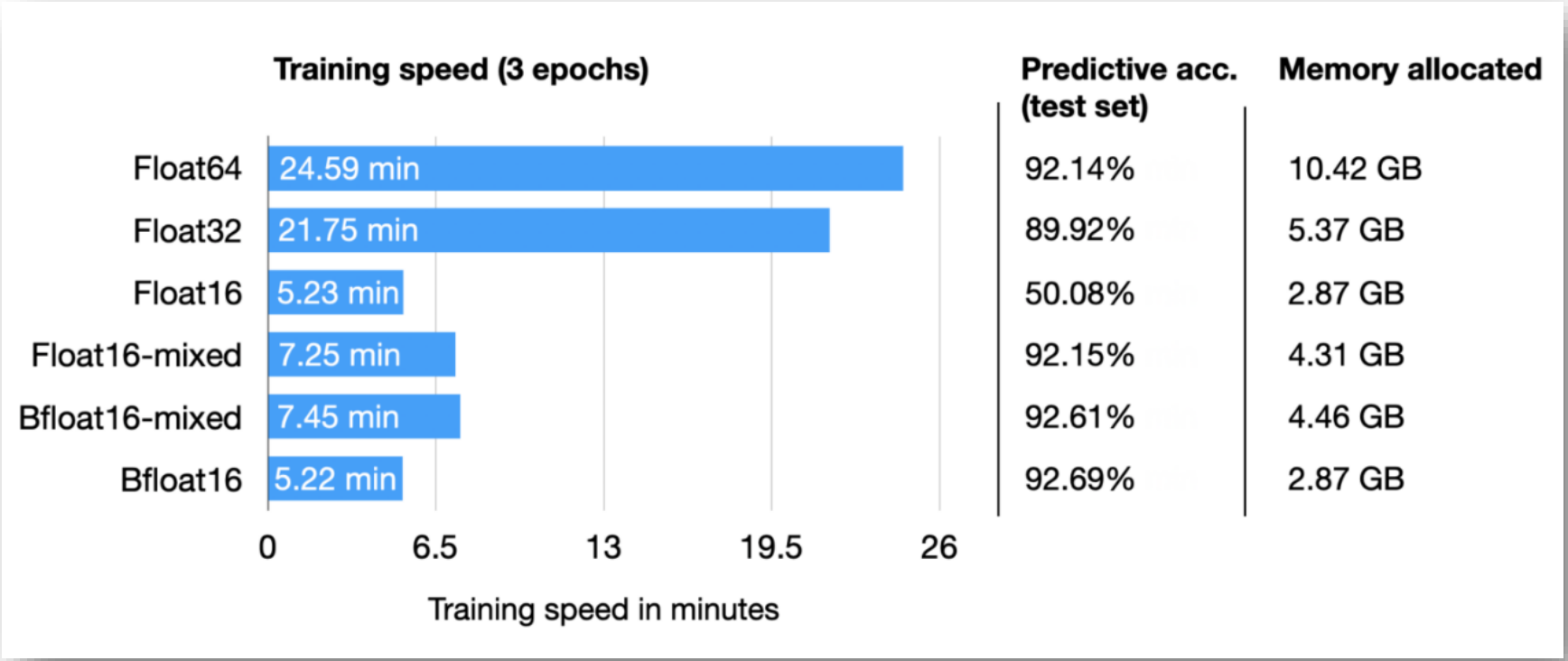
Contenidos

1. Introducción
   1. Motivación y objetivos
   2. Sistemas posicionales (convertir, rango y complemento)
2. Representaciones
   1. Alfanuméricas
      1. Caracteres
      2. Cadenas de caracteres
   2. Numéricas
      1. Naturales y enteras
      2. **Coma fija**
      3. **Coma flotante (estándar IEEE 754)**

Otras necesidades de representación

- ¿Cómo representar?
  - Números muy grandes: $30.556.926.000_{(10}$
  - Números muy pequeños: $0{,}000000000529177_{(10}$
  - Números con decimales: 1,58567

Impacto en IA: cuantización



https://lightning.ai/pages/community/tutorial/accelerating-large-language-models-with-mixed-precision-techniques/

Importancia de la representación
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.

http://www.rvs.uni-bielefeld.de/lectures/TechInf/TI2/download/ariane_pics.html

Contenidos

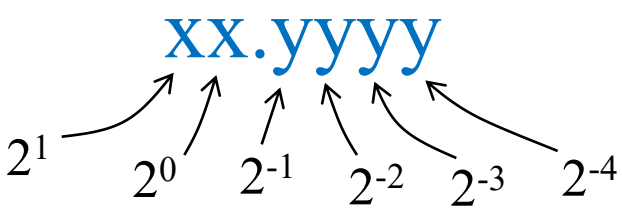
1. Introducción
   1. Motivación y objetivos
   2. Sistemas posicionales (convertir, rango y complemento)
2. Representaciones
   1. Alfanuméricas
      1. Caracteres
      2. Cadenas de caracteres
   2. Numéricas
      1. Naturales y enteras
      2. **Coma fija**
      3. Coma flotante (estándar IEEE 754)

Coma fija [racionales]

Potencias negativas

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
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	
11	…	

- Se fija la posición de la coma binaria y se utilizan los pesos asociados a las posiciones decimales
- Ejemplo de representación con 6 bits:

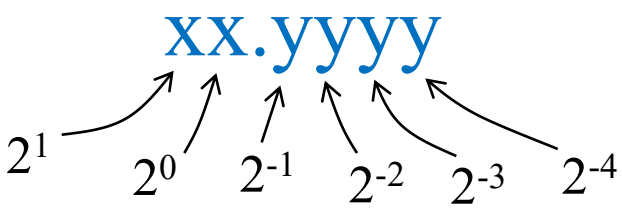


Coma fija [racionales]

Potencias negativas

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
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	
11	…	

- Se fija la posición de la coma binaria y se utilizan los pesos asociados a las posiciones decimales
- Ejemplo de representación con 6 bits:

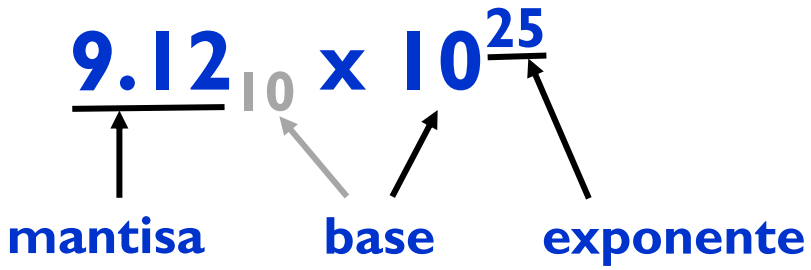


- Ejemplo de número:
  $10.1010_{(2} = 2^{1} + 2^{-1} + 2^{-3} = 2{,}62510$
- El rango sería en este caso:
  [0 a 3.9375 (casi 4)]

Contenidos

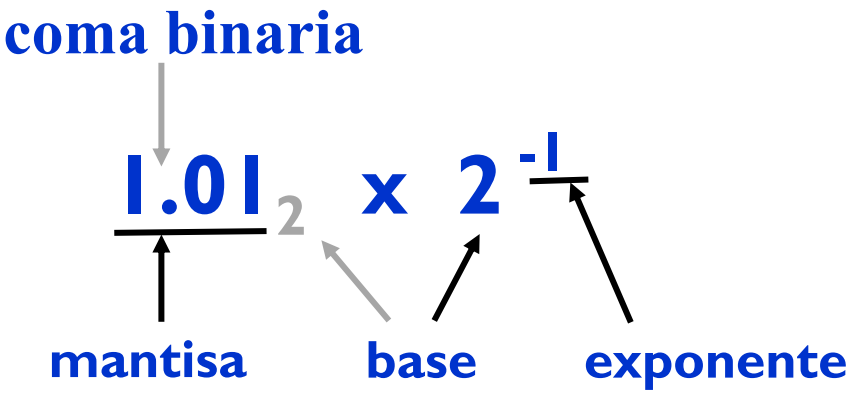
1. Introducción
   1. Motivación y objetivos
   2. Sistemas posicionales (convertir, rango y complemento)
2. Representaciones
   1. Alfanuméricas
      1. Caracteres
      2. Cadenas de caracteres
   2. **Numéricas**
      1. Naturales y enteras
      2. Coma fija
      3. **Coma flotante (estándar IEEE 754)**

Notación científica decimal



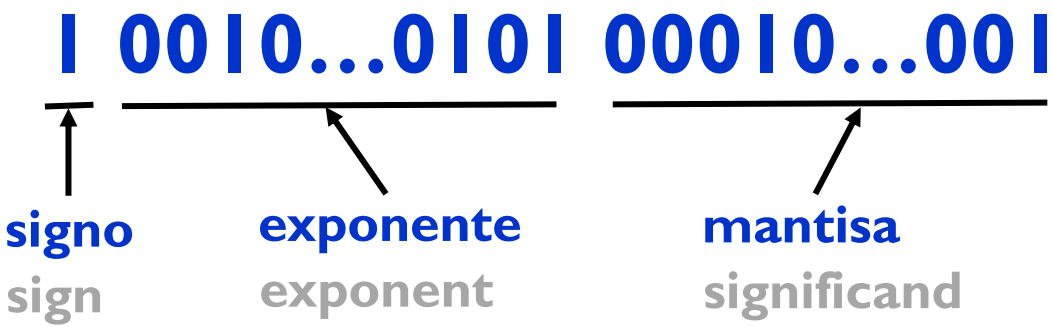
- Cada número lleva asociado una mantisa y un exponente
- Se adapta el número al orden de magnitud del valor a representar, trasladando la *coma decimal* mediante el exponente
- Notación científica decimal usada: **forma normalizada**
  - Solo un dígito distinto de 0 a la izquierda del punto

Notación científica en binario



- En binario la base es 2
- Forma normalizada: un 1 a la izq. de la coma (solo un dígito)
  - Normalizada: 1.0001×2^{-9}
  - No normalizadas: 0.0011×2^{-8}, 10.0×2^{-10}

Estándar IEEE 754-2008 [racionales]



- Estándar para coma flotante usado en la mayoría de los ordenadores.
- 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)
  - **Octuple precisión**: 256 bits (signo: 1, exponente: 19 y mantisa: 237)
- **Características** (salvo casos especiales):
  - Exponente: en exceso con sesgo $2^{num\_bits\_exponente - 1} - 1$
  - Mantisa: signo-magnitud, normalizada, con bit implícito

http://speleotrove.com/decimal/

Estándar IEEE 754-2008 [racionales]

- Normalización

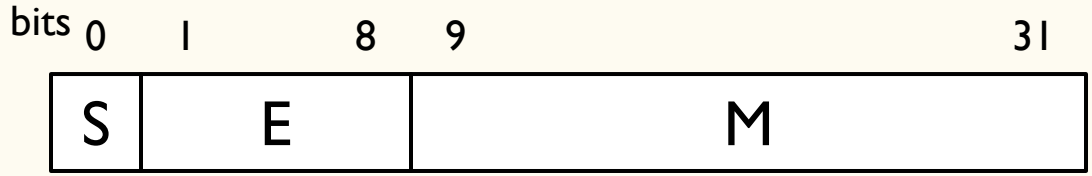
  Para normalizar la mantisa se ajusta el exponente para que el bit más significativo de la mantisa sea 1
  - Ejemplo: 1001000000000000000000 x 2^3 (ya está normalizado)
  - Ejemplo: ~~000~~10000000010101 x 2^3 (no lo está)
    10000000010101000 x 2^0 (ahora sí)
- Bit implícito

  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

- **Características** (salvo casos especiales):
  - Exponente: en exceso con sesgo $2^{num\_bits\_exponente - 1} - 1$
  - Mantisa: signo-magnitud, normalizada, con bit implícito

http://speleotrove.com/decimal/

Estándar IEEE 754 de precisión simple [racionales]



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):

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

donde:

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

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

00000000000000000000000 ≤ M ≤ 11111111111111111111111

Estándar IEEE 754 de precisión simple [racionales]

- Existencia de casos especiales:

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

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)

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

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*) 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

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

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

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	≠ 0	No normalizado
1-254	cualquiera	normalizado

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

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:
    $2^{-126} \times 0.11111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$

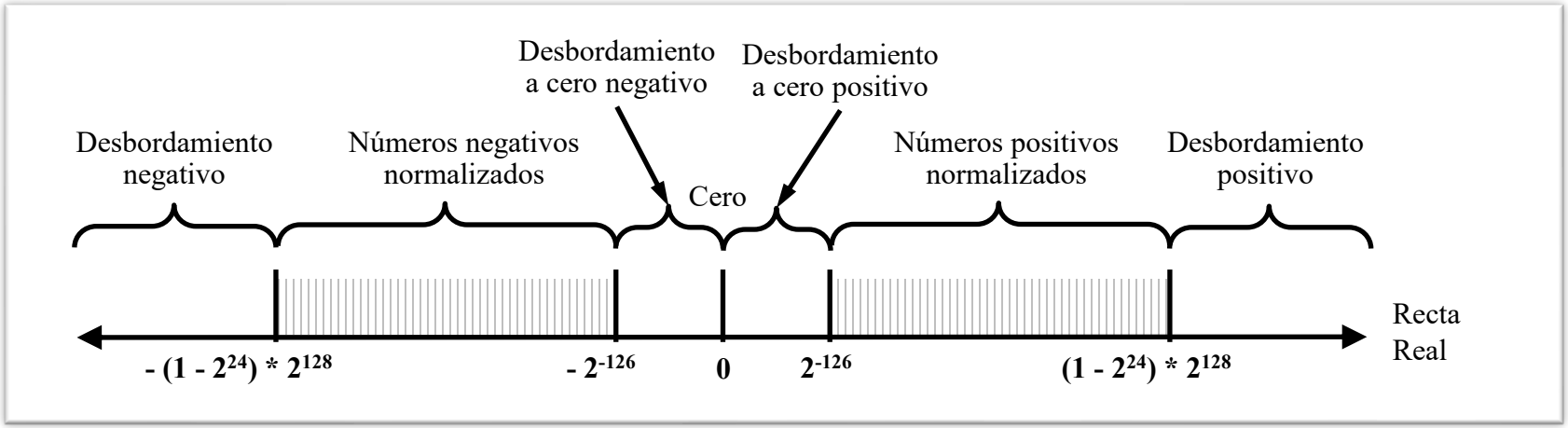
Truco:

$$
\begin{array}{rll}
 & 1.11111111111111111111_2 & = X \\
+ & 0.00000000000000000001_2 & = 2^{-23} \\
\hline
 & 10.00000000000000000000_2 & = 2
\end{array}
$$

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

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:
    $2^{-126} \times 0.11111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$



www.math.unipd.it/~frossi/cap9-2.pdf

Ejercicio

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

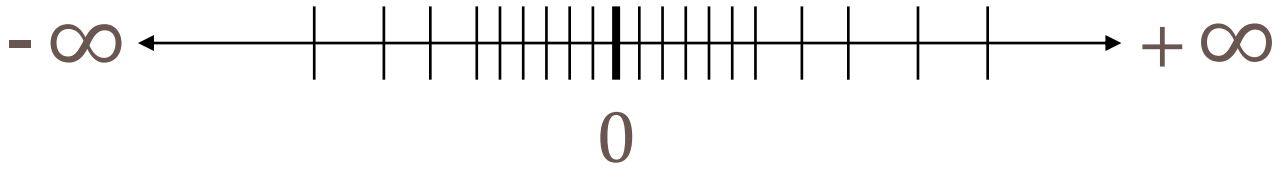
- ¿Cuántos números de *floats* (coma flotante de simple precisión) hay entre el 2 y el 3 (no incluido)?

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

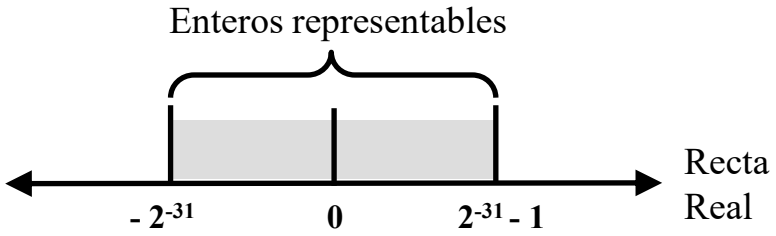
Números representables

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

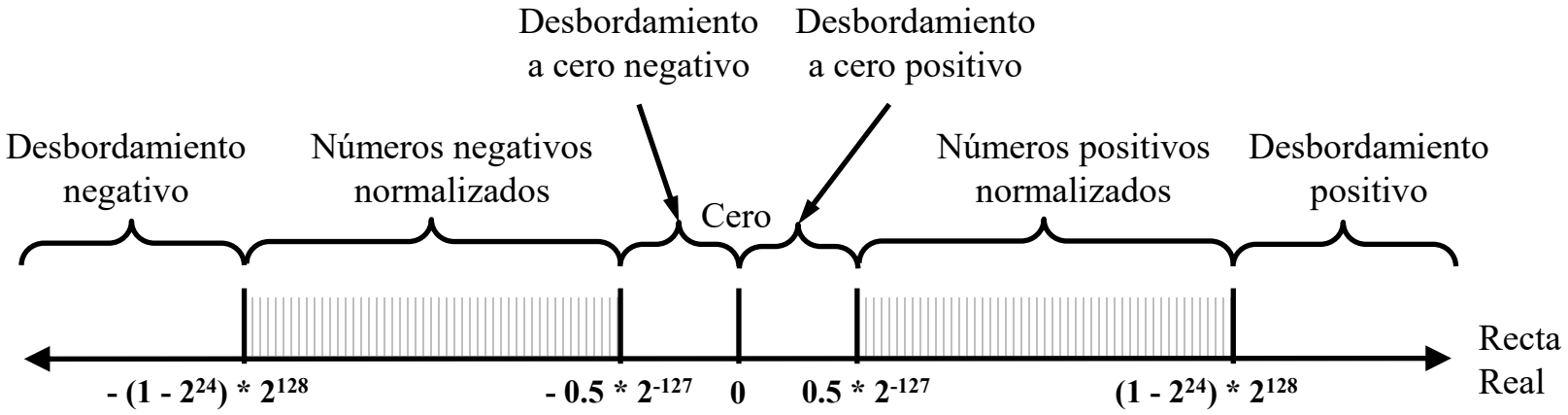


https://www.hobbyconsolas.com/reportajes/toy-story-espacio-disney-infinity-58154

Números representables



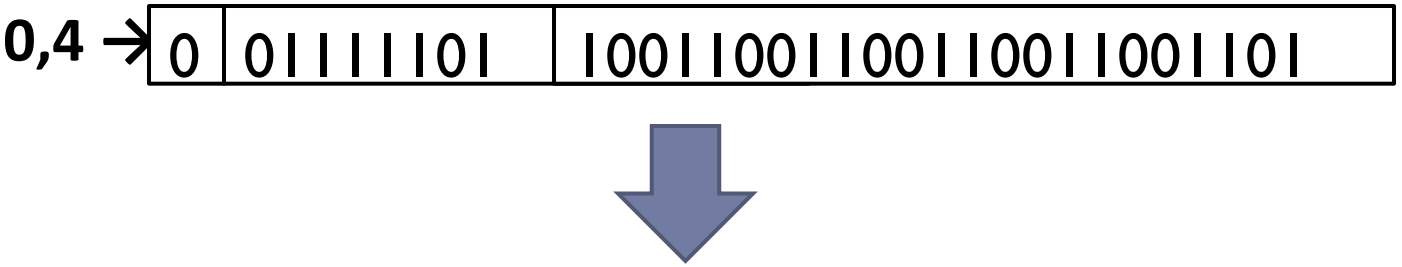
(a) Enteros en complemento a dos

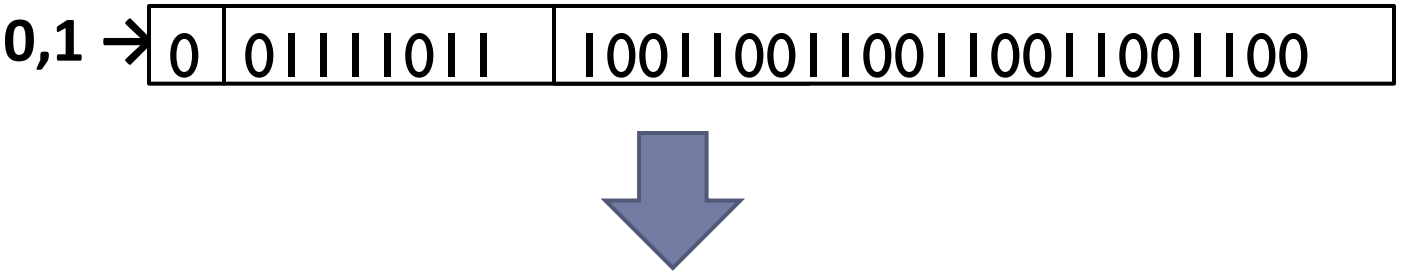


(b) Números en coma flotante

www.math.unipd.it/~frossi/cap9-2.pdf

Ejemplo 1
imprecisión





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) ;
}
```

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) ;
}
```

```
$ 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

```
#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
```

Ejemplo

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
  - $1,111111011010110110011010101 \times 2^{26}$
  - S = 0 (positivo)
  - e = 26 $\rightarrow$ E = 26 + 127 = 153
  - M = 11111011010110110011010 (se pierden los 3 últimos bits)
- El número realmente almacenado es
  - $1,11111011010110110011010 \times 2^{26}$ =
  - $11111011010110110011010 \times 2^3 = 133000400$

Ejemplo
Conversión float → int → float

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

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

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

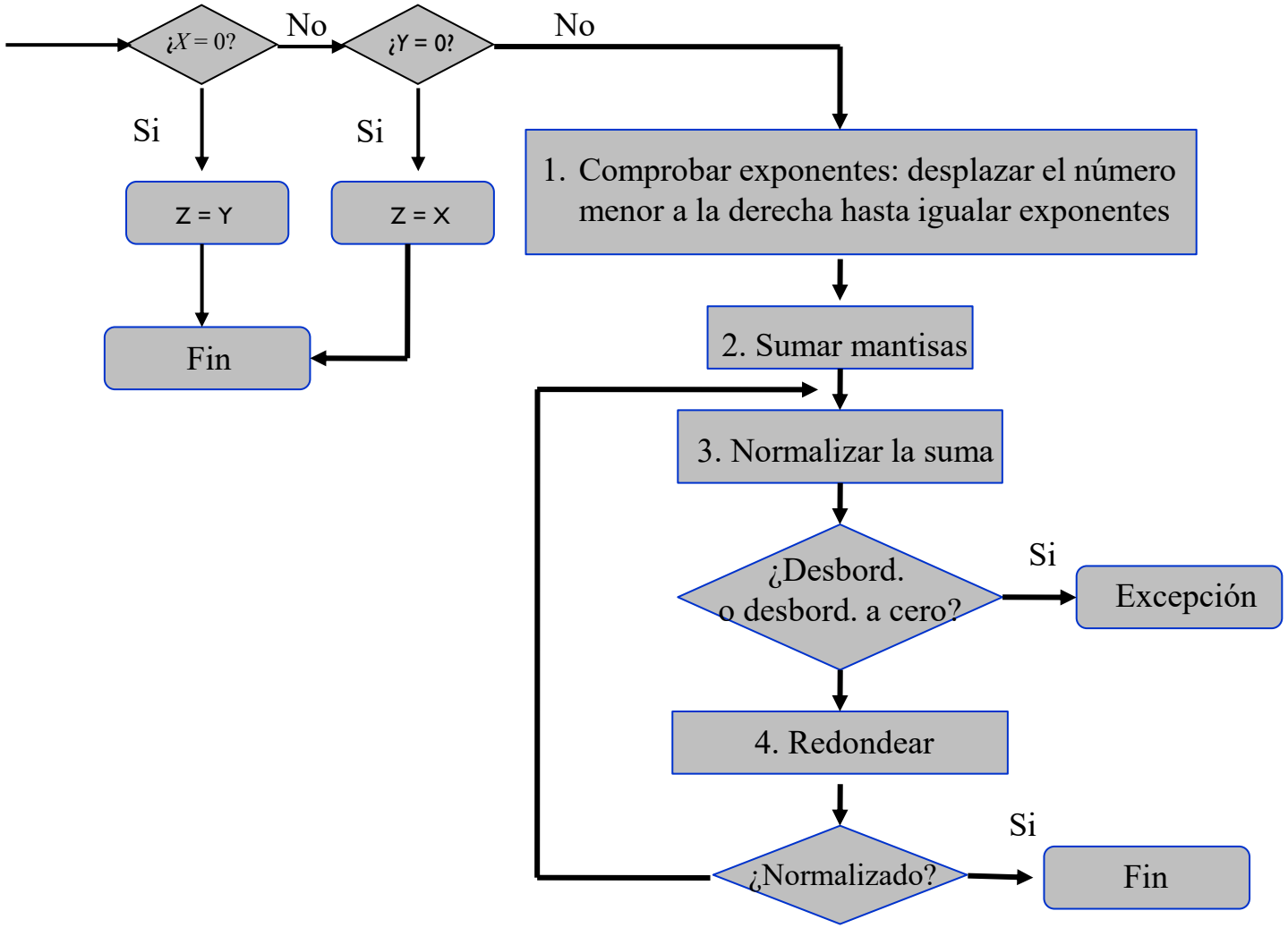
- Se utilizan dígitos de guarda para mejorar la precisión: internamente se usan dígitos adicionales para operar.
- Ejemplo: $2{,}65 \times 10^0 + 2.34 \times 10^2$

	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	$2{,}36 \times 10^2$	$2{,}37 \times 10^2$

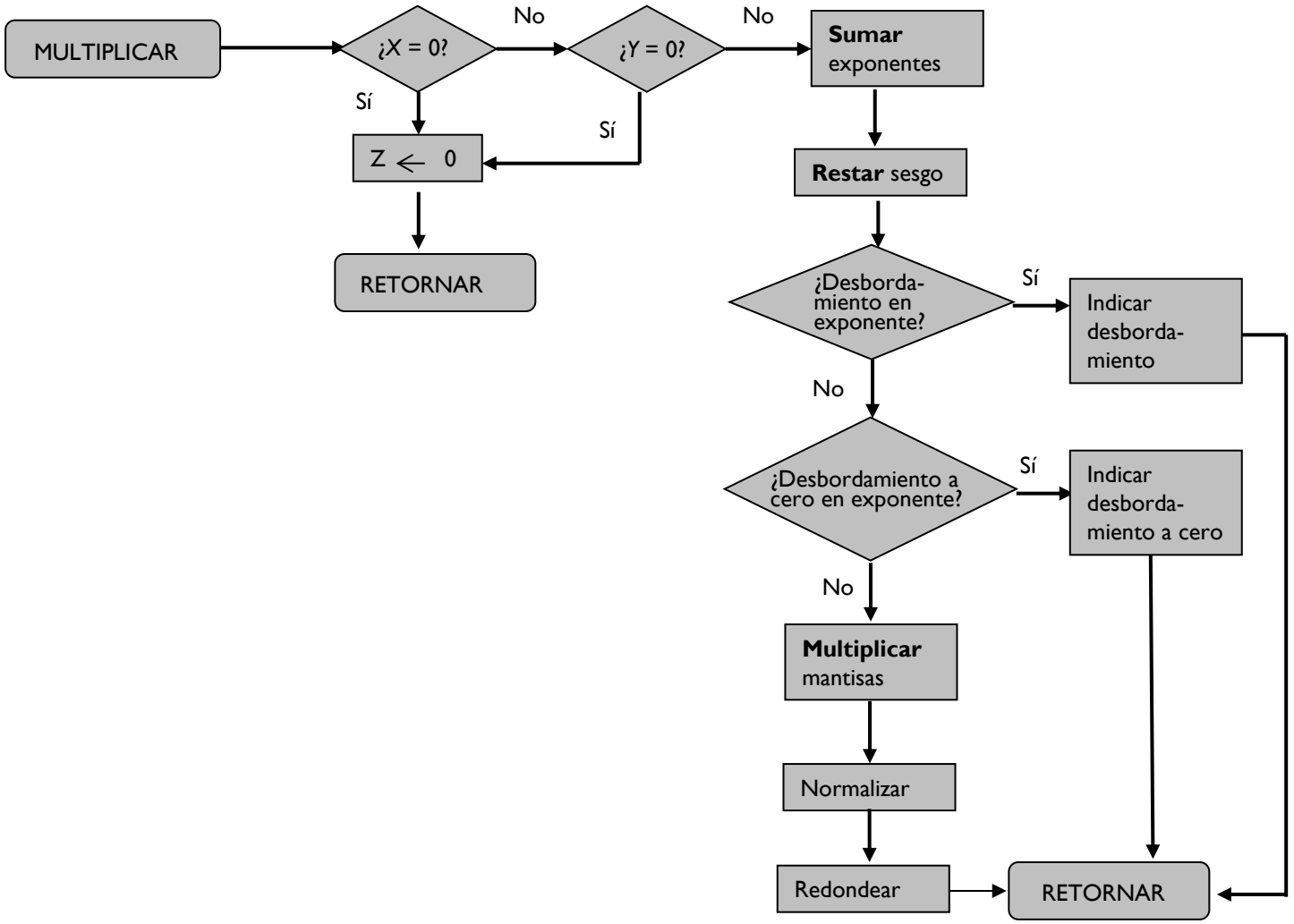
Operaciones en coma flotante

- 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.

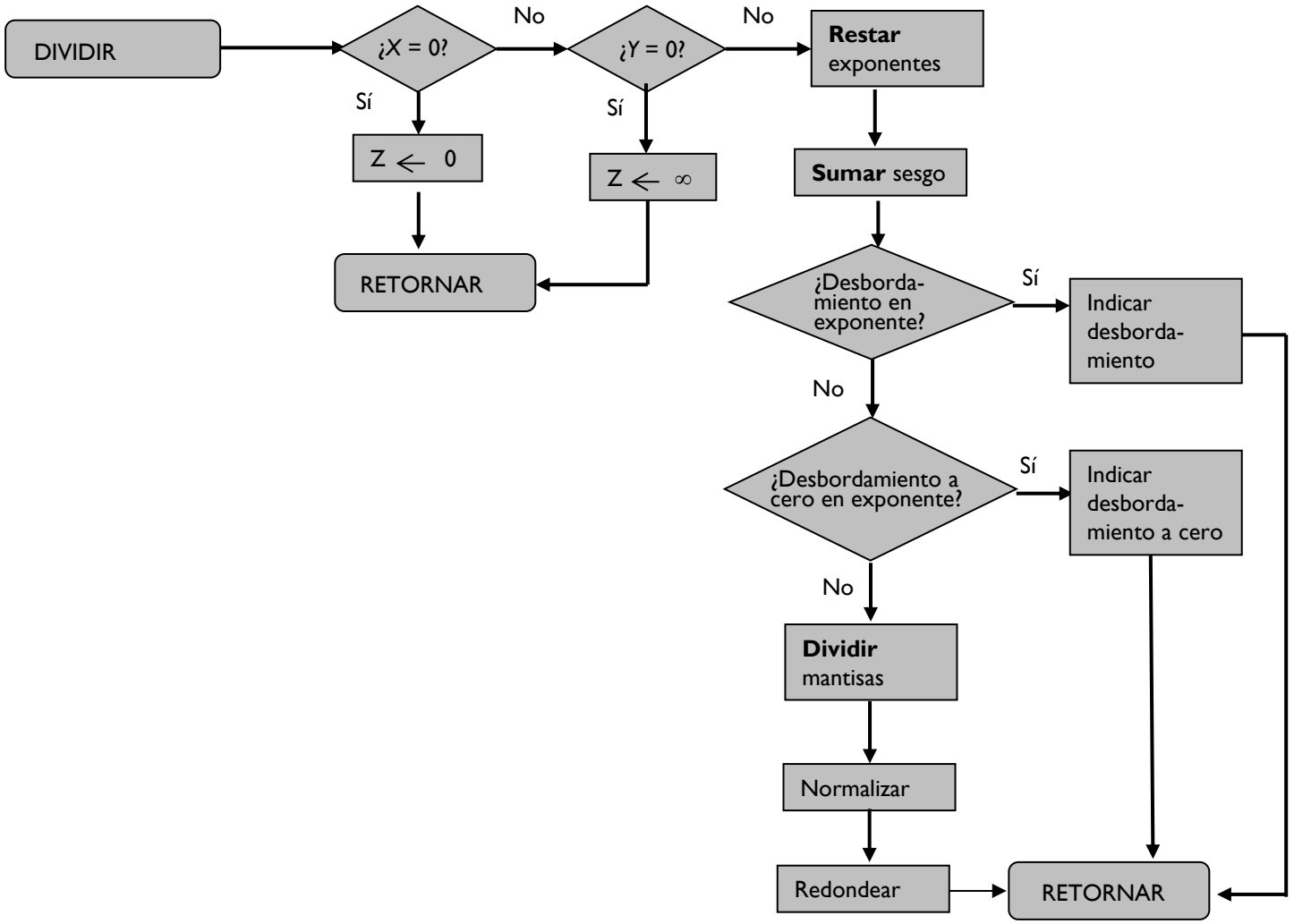
Suma y resta: Z=X+Y y Z=X-Y



Multiplicación: Z=X*Y



División: Z=X/Y



Ejercicio

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

Solución (sin IEEE 754)

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

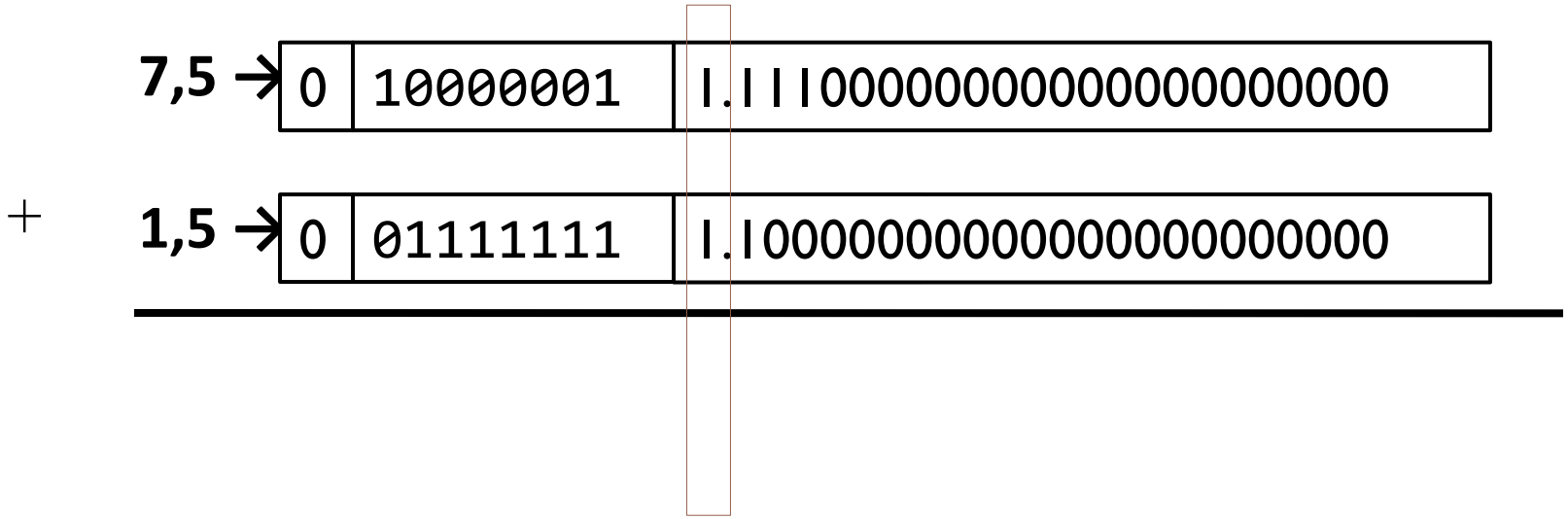
- Representación de los números

7,5 → 0 10000001 11100000000000000000

1,5 → 0 01111111 10000000000000000000

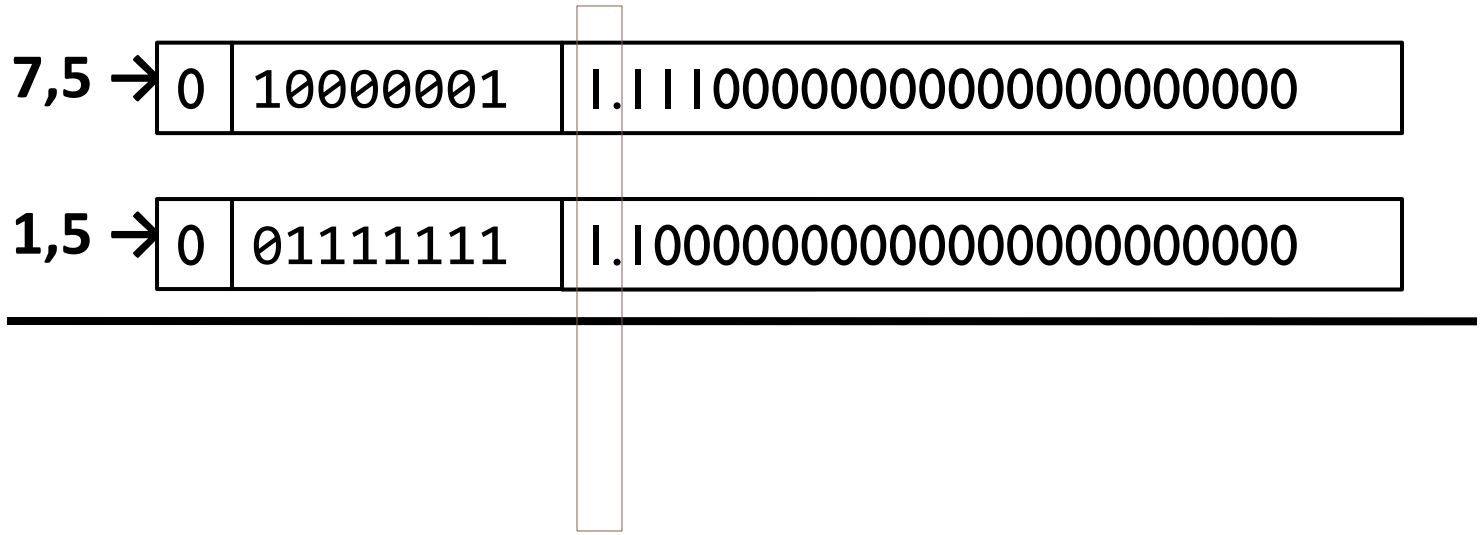
Solución

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



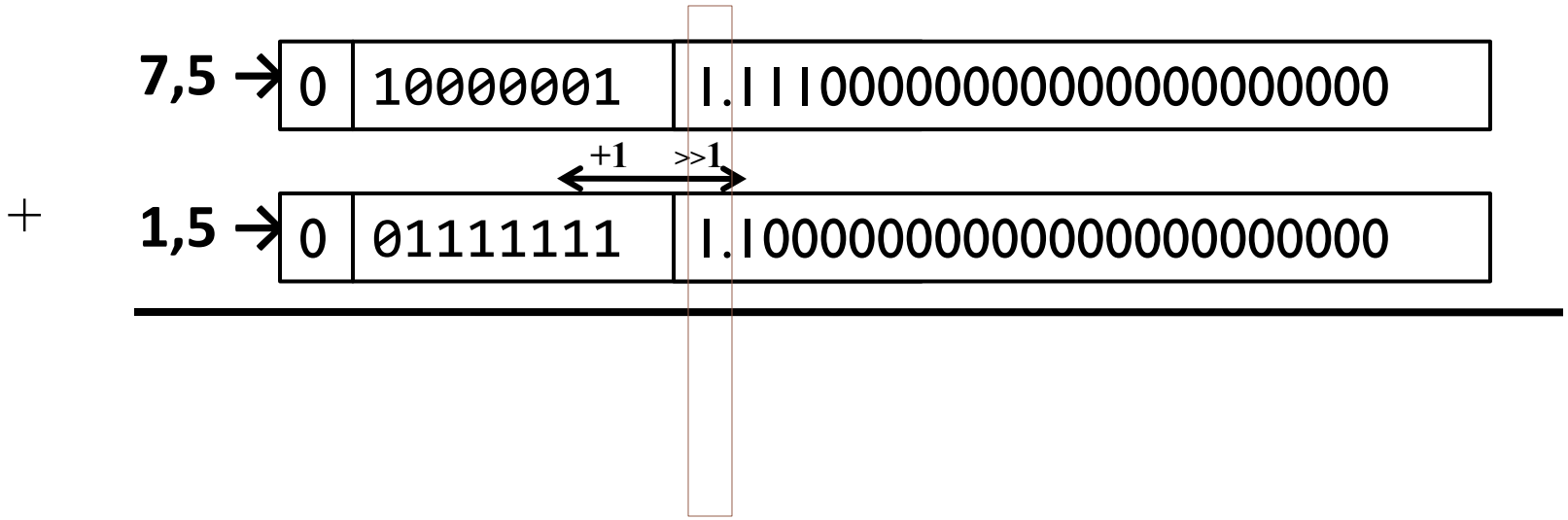
Solución

- Igualar exponentes



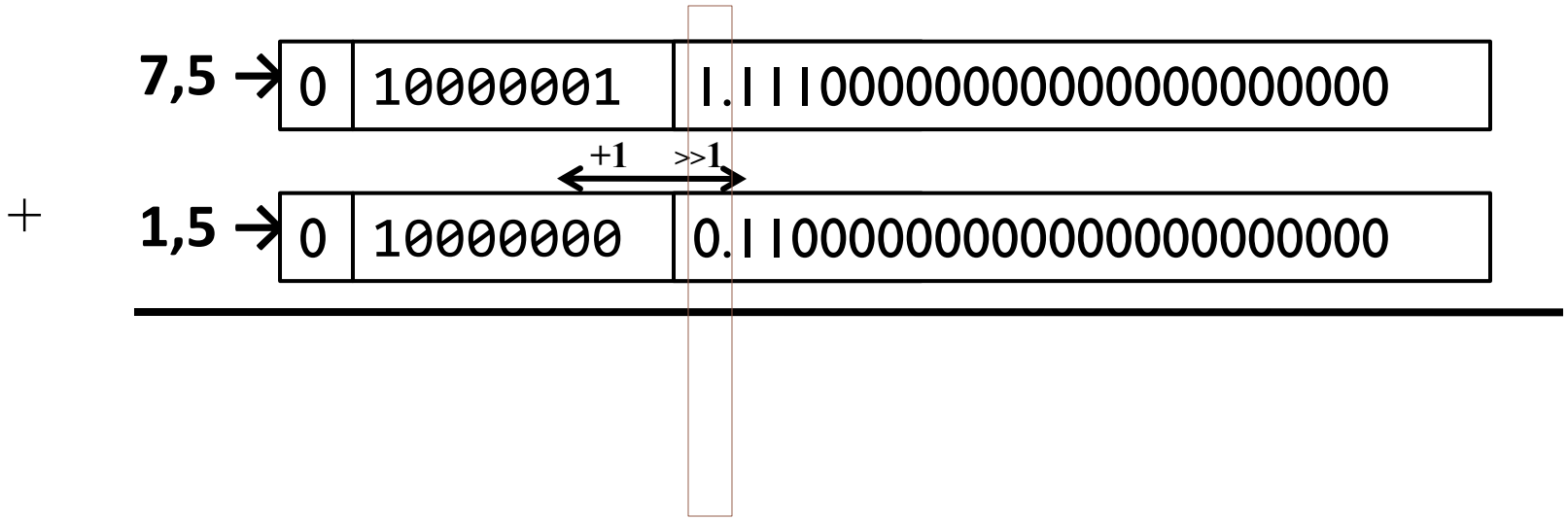
Solución

- Igualar exponentes



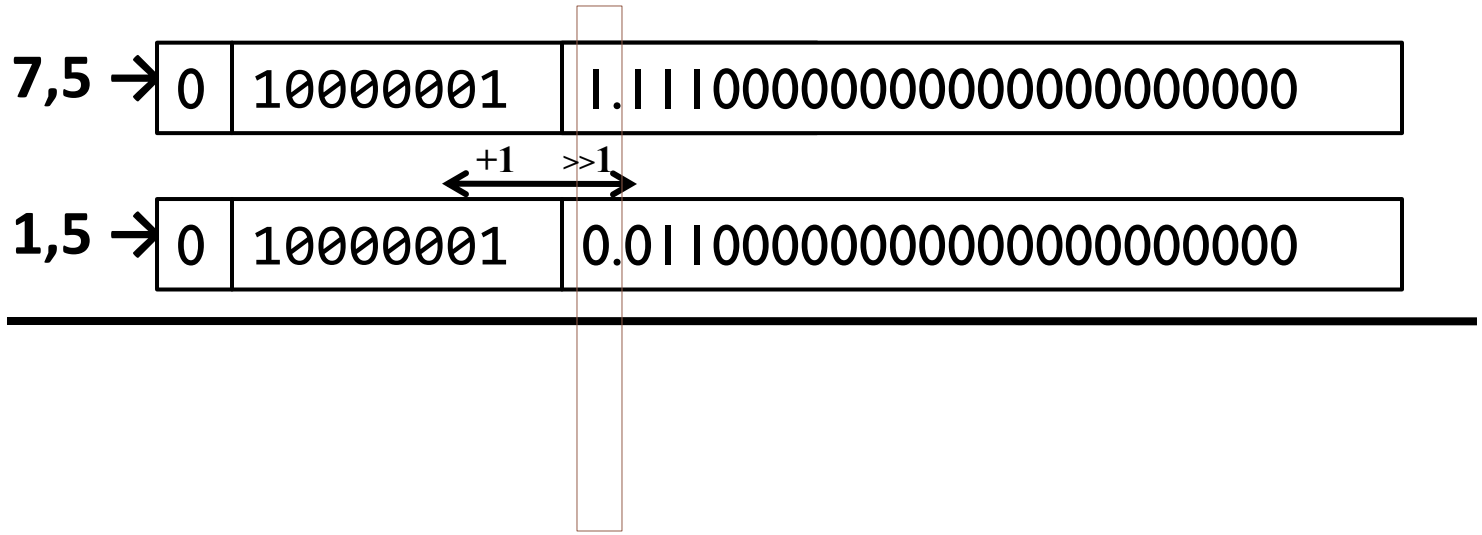
Solución

- Igualar exponentes



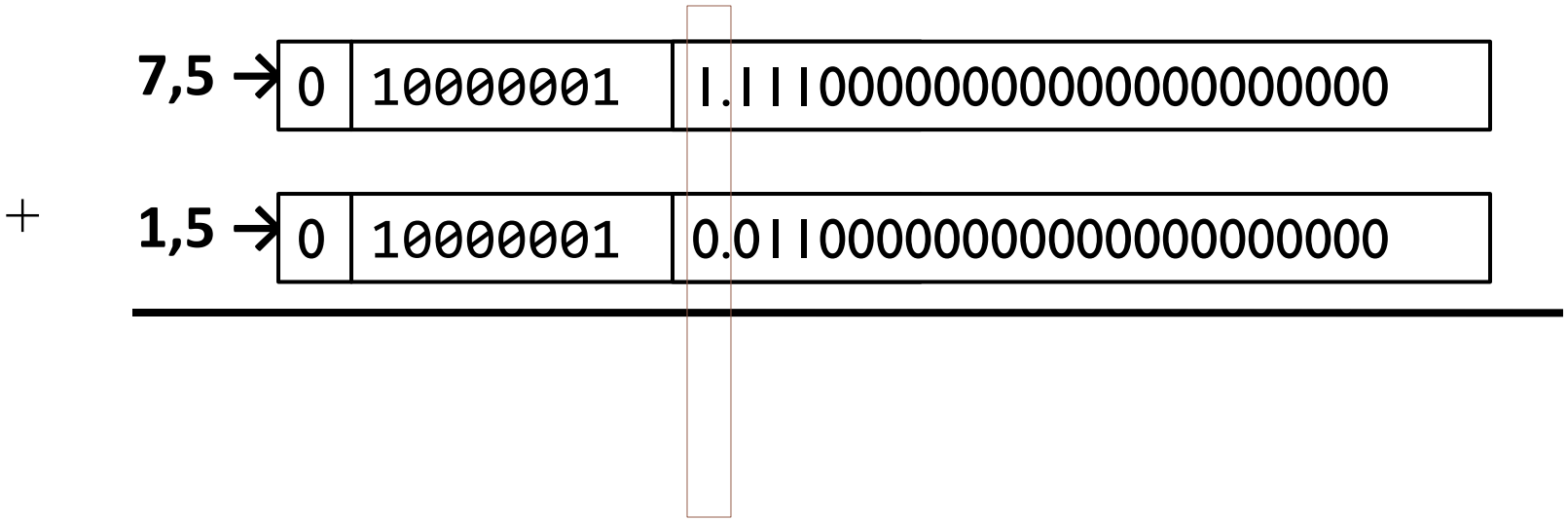
Solución

- Igualar exponentes



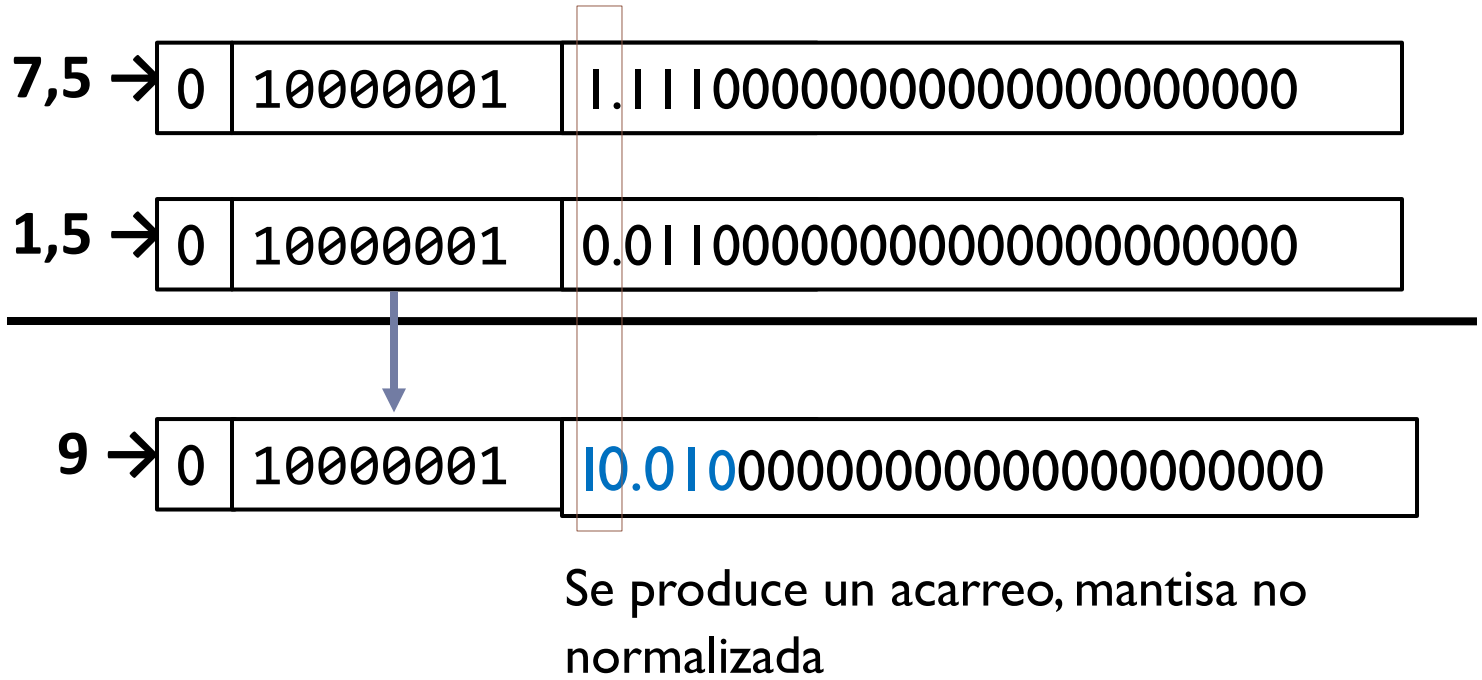
Solución

- Sumar mantisas



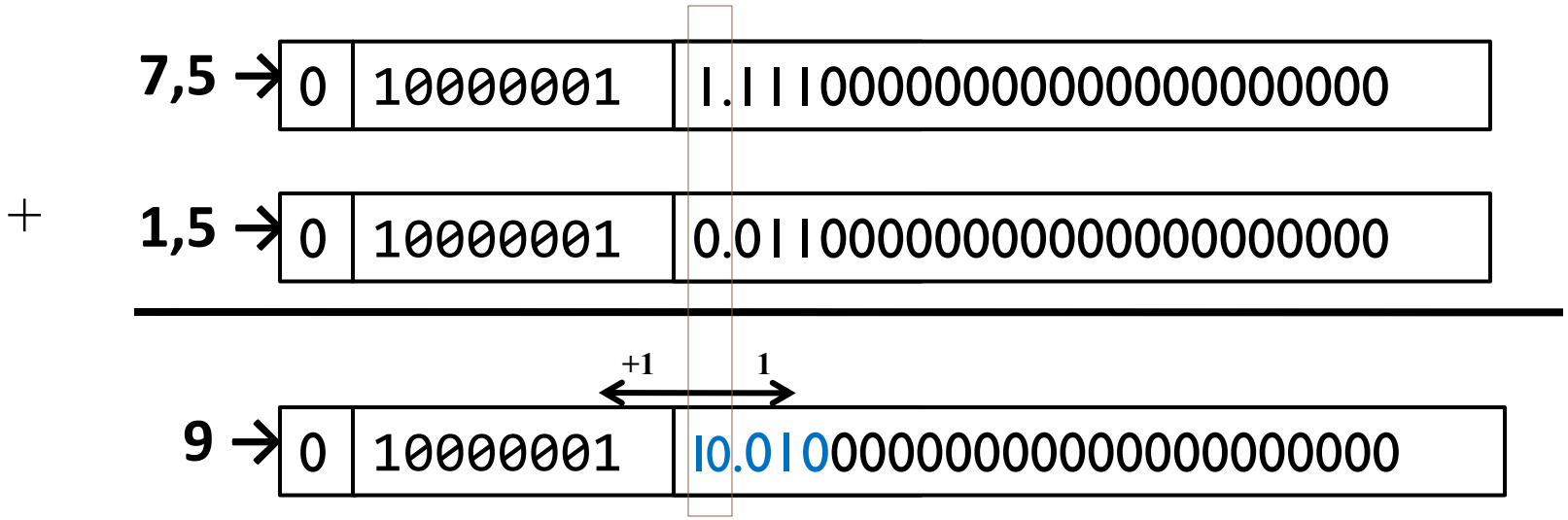
Solución

- Normalizar el resultado

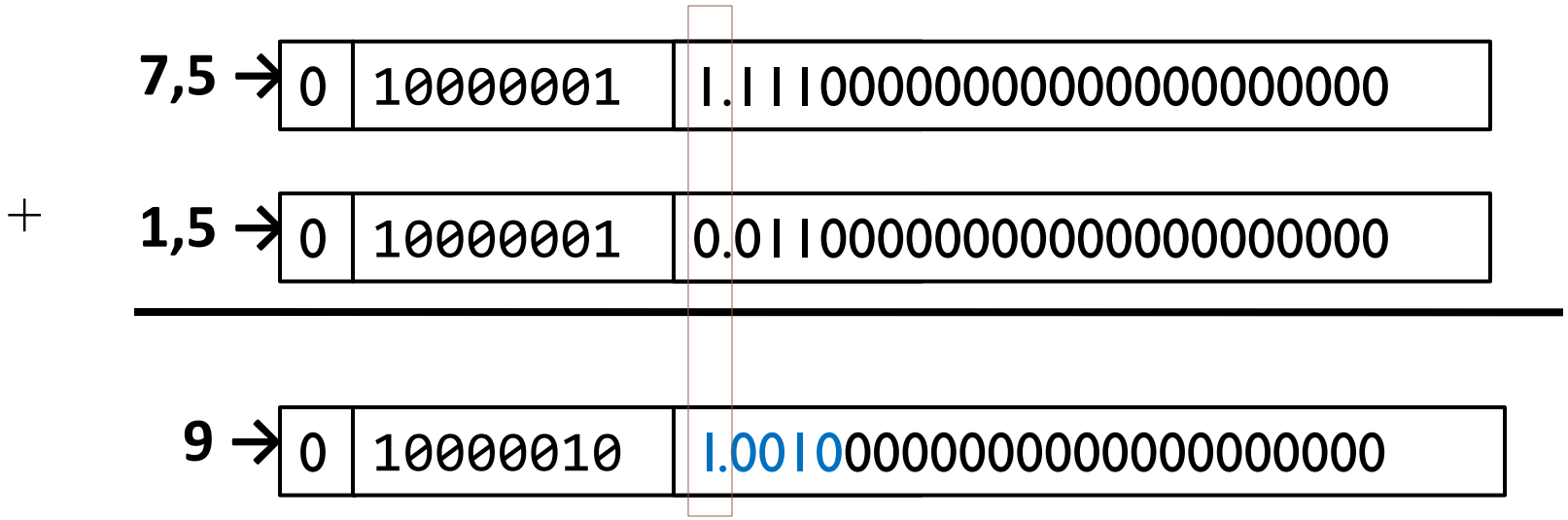


Solución

- Normalizar el resultado



Solución


7,5 →
0
10000001
1.111000000000000000000
+
1,5 →
0
10000001
0.011000000000000000000
9 →
0
10000010
1.001000000000000000000

Solución

- Se almacena el resultado eliminando el bit implícito

9 → `0 10000010 00100000000000000000`

Ejercicio

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

Solución

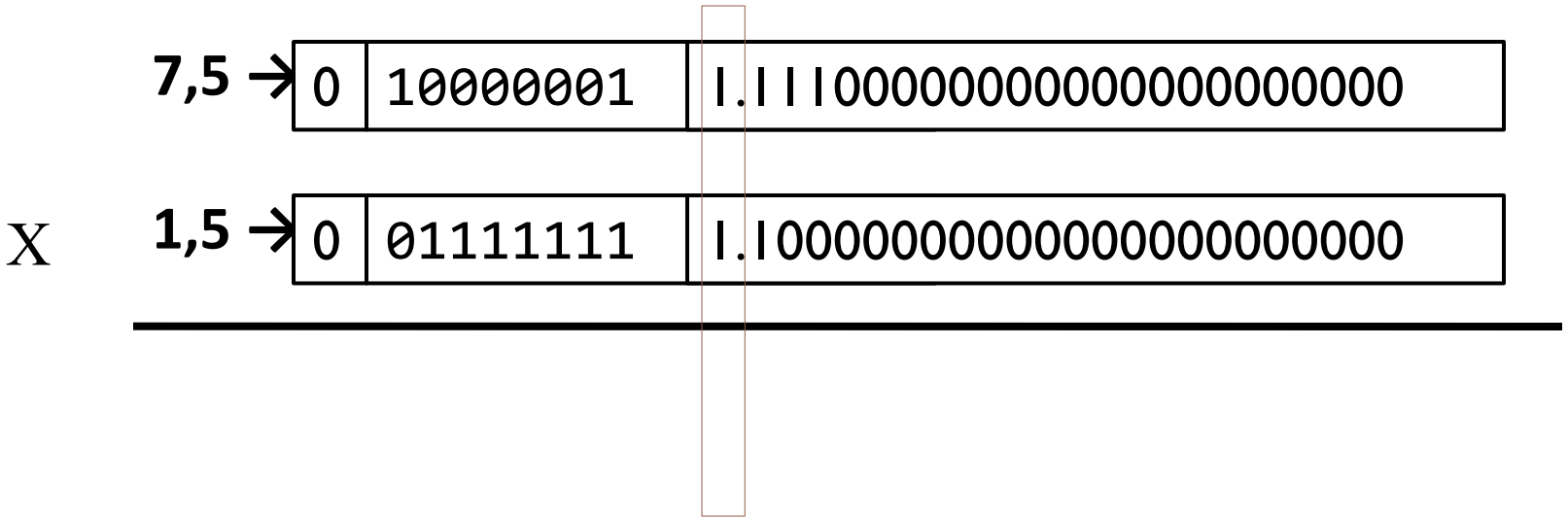
- Representación de los números

9 → `0 10000010 00100000000000000000000`

7,5 → `0 10000001 11100000000000000000000`

Solución

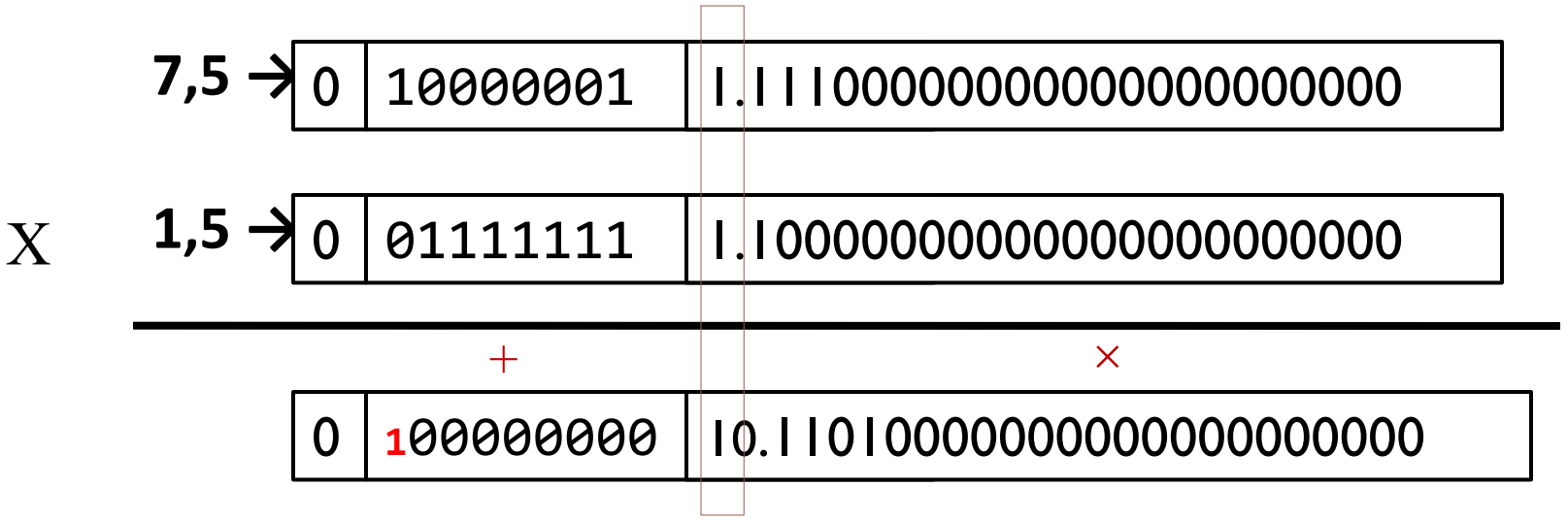
- Se separan exponentes y mantisas y se añade bit implícito



Se añade el bit implícito para operar

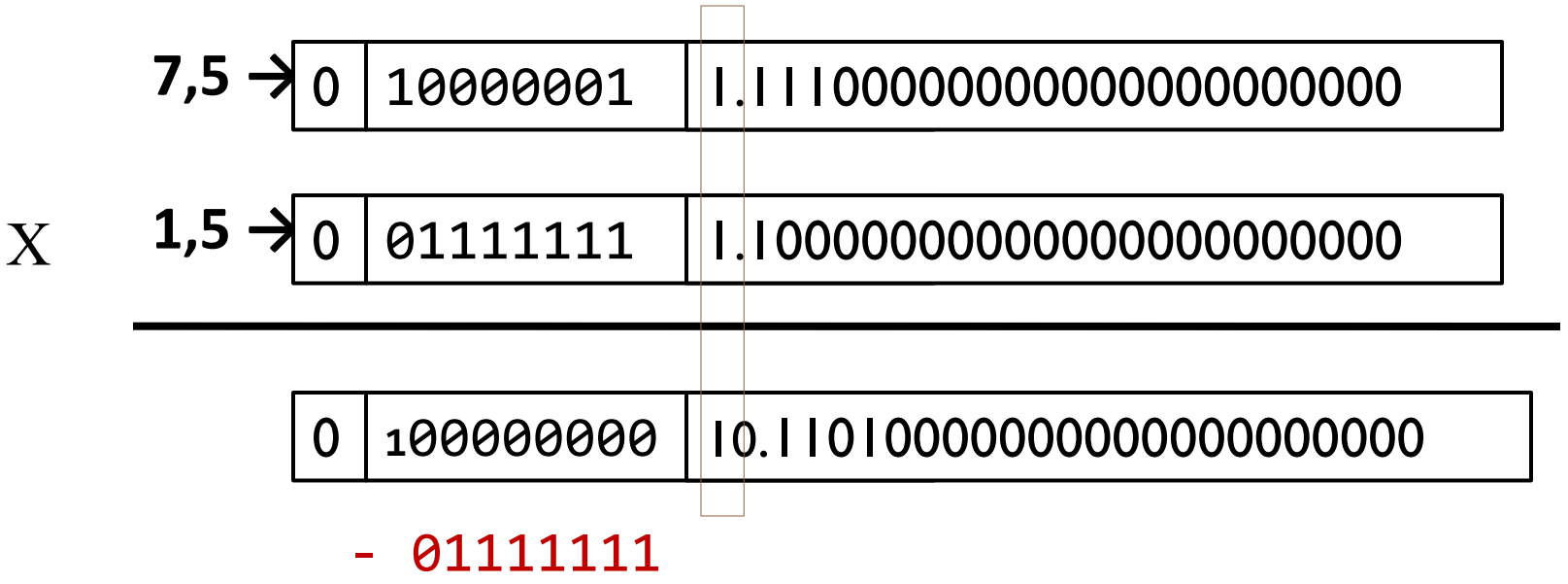
Solución

- Multiplicar: sumar exponentes y multiplicar mantisas



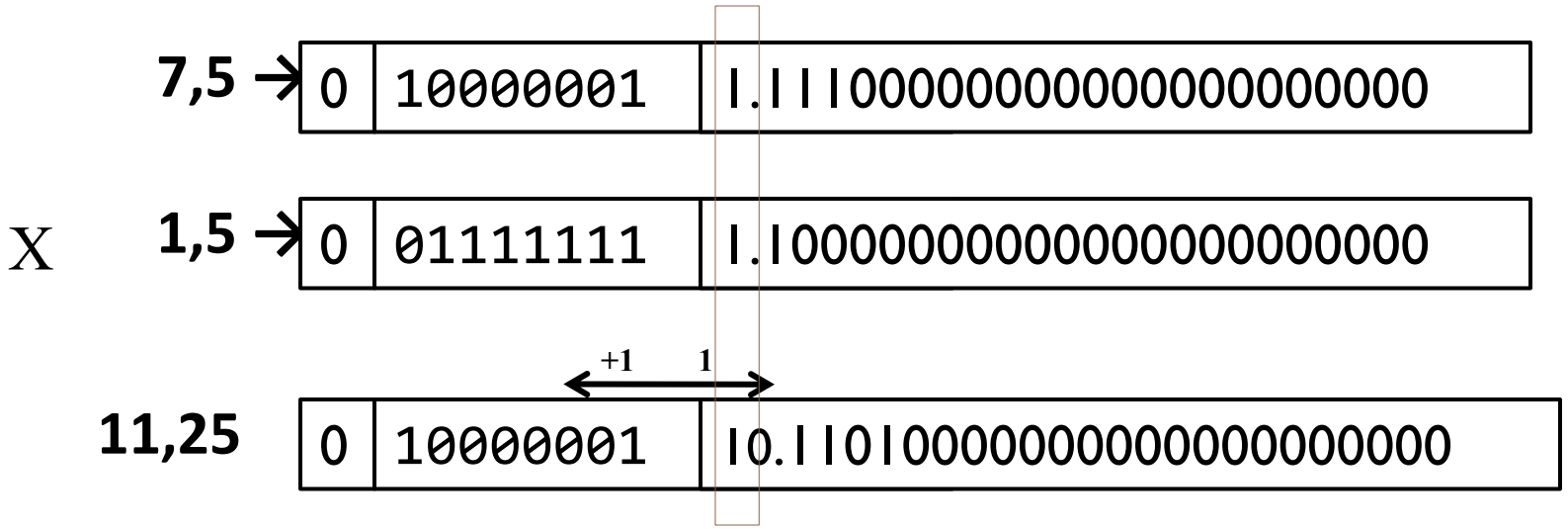
Solución

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



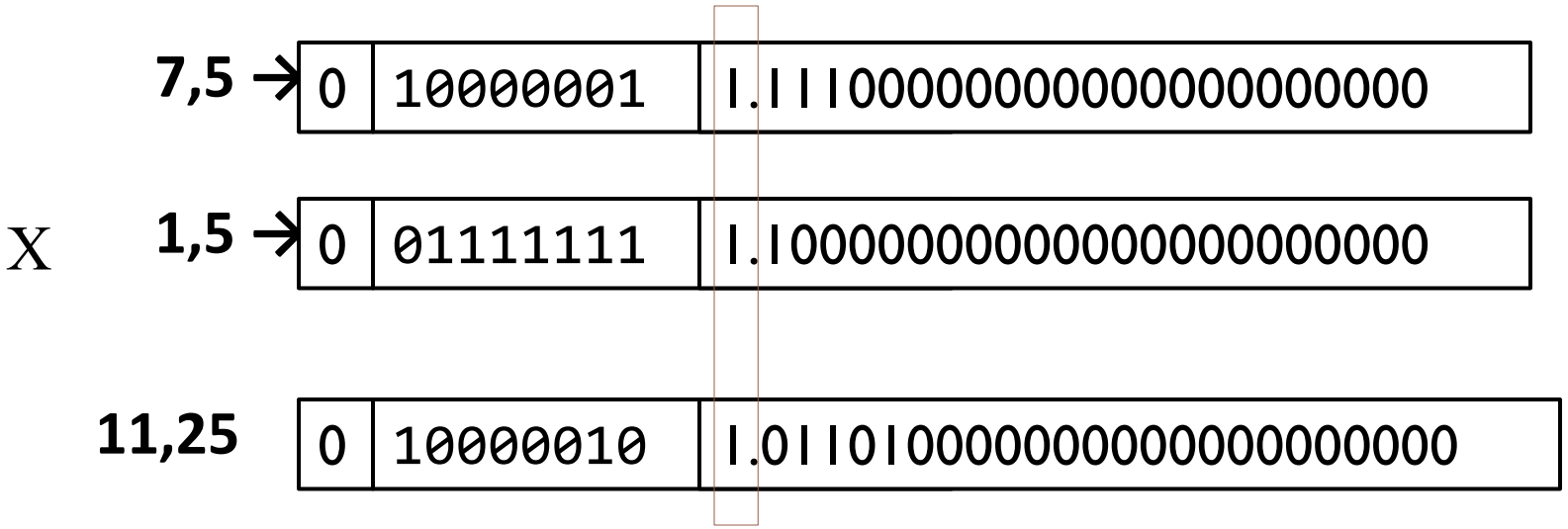
Solución

- Multiplicar: normalizar el resultado



Solución

- Resultado normalizado…



Solución

- Se almacena el resultado eliminando el bit implícito

11,25 `0 10000010 01101000000000000000000`

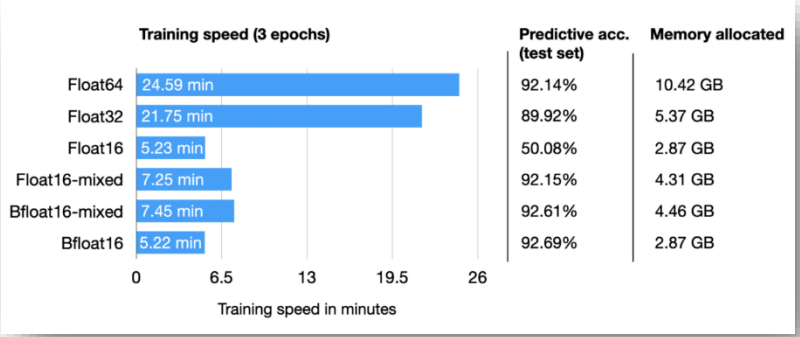
Evolución de IEEE 754

- 1985 – IEEE 754
- 2008 – IEEE 754-2008 (754+854)
- 2019 – IEEE 754-2019 (aclaraciones y arreglos)
- 2029 – IEEE 754-2029 (previsión inicial, en desarrollo)

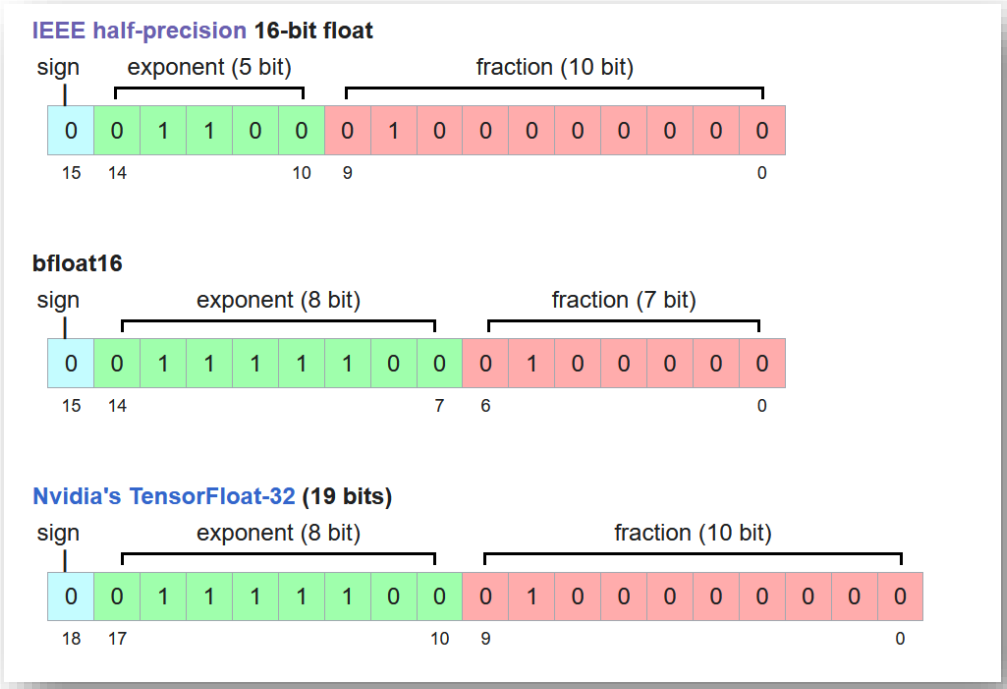
Name	Common name	Base	Significand bits/digits	Exp. bits	Decimal E max	Exponent bias[14]	E min	E max
binary16	Half precision	2	11	5	4.51	24−1 = 15	−14	+15
binary32	Single precision	2	24	8	38.23	27−1 = 127	−126	+127
binary64	Double precision	2	53	11	307.95	210−1 = 1023	−1022	+1023
binary128	Quadruple precision	2	113	15	4931.77	214−1 = 16383	−16382	+16383
binary256	Octuple precision	2	237	19	78913.2	218−1 = 262143	−262142	+262143
decimal32		10	7	7.58	96	101	−95	+96
decimal64		10	16	9.58	384	398	−383	+384
decimal128		10	34	13.58	6144	6176	−6143	+6144

Evolución para IA

- 2008 – IEEE 754-2008 (*binary16*)
- …
- 2019 – bfloat16 (*brain floating point*)



https://lightning.ai/pages/community/tutorial/accelerating-large-language-models-with-mixed-precision-techniques/



https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

Grupo ARCOS

uc3m | Universidad Carlos III de Madrid

Tema 2: Representación de la información (2/2)

Estructura de Computadores

Grado en Ingeniería Informática
Grado en Matemática aplicada y Computación
Doble Grado en Ingeniería Informática y Administración de Empresas