

# Números Fraccionarios

95.57/75.03 Organización del computador

---

**Docentes:** Patricio Moreno y Adeodato Simó

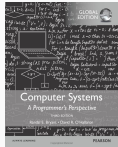
1.<sup>er</sup> cuatrimestre de 2020

Última modificación: Sun Apr 19 09:41:20 2020 -0300

Facultad de Ingeniería (UBA)

# Créditos

Para armar las presentaciones del curso nos basamos en:



R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2015.



D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2018.



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2019.

# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C

# Floating Point Puzzles

- Argumentar si es cierta, o explicar por qué es falsa cada una de las siguientes expresiones de C:

```
1 int x = ...;  
2 float f = ...;  
3 double d = ...;
```

Asumir que no son  
NaN f ni d

1. `x == (int) (float) x;`
2. `x == (int) (double) x;`
3. `f == (float) (double) f;`
4. `d == (double) (float) d;`
5. `f == -(-f);`
6. `2/3 == 2/3.0;`
7. `d < 0  $\Rightarrow$  ((d*2) < 0)`
8. `d > f  $\Rightarrow$  -f > -d`
9. `d * d >= 0`
10. `(d + f) - d == f`

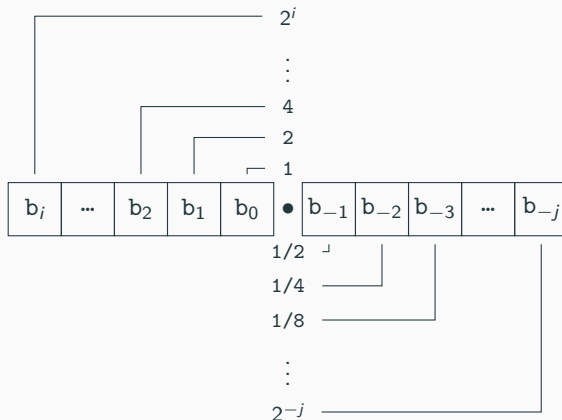
# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C

# Números fraccionarios en binario

¿Qué es  $11000000.11011010_2$ ?



# Ejemplos

■ Representación	Valor
$101.11_2$	$5 \frac{3}{4}$
$10.111_2$	$2 \frac{7}{8}$
$1.0111_2$	$1 \frac{7}{16}$

## ■ Observaciones

- Desplazamiento a derecha: división por 2 (**unsigned**)
- Desplazamiento a izquierda: multiplicación por 2
- Números de la forma:  $0.111111\dots_2$  son casi 1.0
  - $1/2 + 1/4 + 1/8 + \dots + 1/2^i + \dots \rightarrow 1,0$
  - $1 - \varepsilon$
- Números periódicos:  $0.0011[0011]\dots$

# Números representables

- Limitación #1
  - Representa exactamente números de la forma  $x/2^k$  únicamente
    - Otros números racionales tienen representaciones periódicas
      - Valor      Representación
      - 1/3      0.0101010101 [01]...<sub>2</sub>
      - 1/5      0.001100110011 [0011]...<sub>2</sub>
      - 1/10      0.0001100110011 [0011]...<sub>2</sub>
- Limitación #2
  - Sólo una posición del punto binario entre los  $w$  bits
    - Rango limitado (¿valores grandes? ¿valores chicos?)



# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C

## Representación en punto fijo (decimal)

- Representación escalada:
  - 16.162 se puede representar como 16162 con un escalado de  $1/1000$ .
  - 16162 se puede representar como 16.162 con un escalado de 1000.
- Cantidad de decimales fijos
- Se eliminan los decimales
- La aritmética de punto fijo es la de los enteros

## Ejemplo: punto fijo en base 10

### Definiciones

Sea  $x = 4681$ ,  $y = 3511$ ,  $k = 1/10$  el escalado (1 decimal). Entonces:  $x$  representa el número 468,1, e  $y$  representa el número 351,1.

### suma/resta

$$s = x + y = \frac{4681}{10} + \frac{3511}{10} = \frac{8192}{10} = 819,2.$$

$$r = x - y = \frac{4681}{10} - \frac{3511}{10} = \frac{1170}{10} = 117,0.$$

### multiplicación

$$m = x \cdot y = \frac{4681}{10} \cdot \frac{3511}{10} = \frac{16434991}{100} = 164349,91. \text{ Hay que ajustar a 1 decimal (dividir por 10). } m = \frac{1643499,1}{10}.$$

### división

$$d = x/y = \frac{4681}{10} / \frac{3511}{10} = 1,3332\dots \text{ Hay que ajustar a 1 decimal (multiplicar por 10). } d = \frac{13,332}{10}.$$

## Representación en punto fijo (binario)

- Análoga a la versión decimal
- Cambia la base ( $b = 2$ ) y el escalado ( $k = 2^i$ )
- Puede ser signada o no signada
- Aritmética de enteros
- Agregamos el tamaño de la palabra de la máquina ( $w$ )
- Separamos los  $w$  bits en la parte entera y la parte fraccional
  - $Qd$ :  $d$  bits en la parte fraccionaria, ¿ $w$ ?
  - $Qi, d$ :  $i$  bits en la parte entera,  $d$  en la fraccionaria,  $w = i + d + 1$
  - $fxi, w$ :  $i$  bits en la parte entera,  $d = w - i - 1$
  - $fixed\langle w, d \rangle$ :  $d$  bits en la parte fraccionaria,  $i = w - d - 1$

### Ejemplo: Q5,2

f = 010010.10

?	16	8	4	2	1		0.5	0.25
0	1	0	0	1	0	.	1	0
								suma 18.5

## Ejemplo: punto fijo en base 2

### Representación: Q6.2

a = 01001010 = 18.50 (01001010 / 4)

b = 01010111 = 21.75 (01010111 / 4)

### suma (ej. unsigned)

```

    01001010 (74)
+   01010111 (87)
-----
    10100001 (161)
-----
    101000.01 (40.25)
  
```

### multiplicación

```

           01001010 (74)
      *      01010111 (87)
      -----
    0001100100100110 (6438)
      -----
    000110010010.0110 (402.375)
  
```

### resta (ej. con signo)

```

    01001010 (74)
-   01010111 (87)
-----
    11110011 (-13)
-----
    111100.11 (-3.25)
  
```

### división

```

    01001010 (74)
  /   01010111 (87)
  -----
    00000000 (0)
  -----
    000000.00 (0.00)
  
```

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50 \text{ (} 01001010 / 4 \text{)}$

$b = 01010111 = 21.75 \text{ (} 01010111 / 4 \text{)}$

## multiplicación

```

      01001010 (74)
    * 01010111 (87)
    -----
    0001100100100110 (6438)
    000110010010.0110 (402.375)
  
```

*Puede hacer overflow*

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

- Requiere el doble de bits
- Tiene el doble de precisión

## multiplicación

```

    0000000001001010 (74)
  * 0000000001010111 (87)
  -----
    0001100100100110 (6438)
    0001100100100111 (6438+1)
  -----
    0000011001001001 (6438+1 >> 2)
    00000110010010.01 (402.375)
  
```

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

a = 01001010 = 18.50 (01001010 / 4)

b = 01010111 = 21.75 (01010111 / 4)

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50$  ( $01001010 / 4$ )

$b = 01010111 = 21.75$  ( $01010111 / 4$ )

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

	0000000001001010	(74)
*	0000000001010111	(87)
<hr/>		
	0001100100100110	(6438)
<hr/>		
	0001100100100111	(6438+1)
<hr/>		
	0000011001001001	(6438+1 >> 2)
<hr/>		
	010010.01	(18.25)



# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50$  ( $01001010 / 4$ )

$b = 01010111 = 21.75$  ( $01010111 / 4$ )

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

0000000001001010	(74)	promoción a <code>uint16_t</code>
* 0000000001010111	(87)	
<hr/>		
0001100100100110	(6438)	
<hr/>		
0001100100100111	(6438+1)	
<hr/>		
0000011001001001	(6438+1 >> 2)	
<hr/>		
010010.01	(18.25)	

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50$  ( $01001010 / 4$ )

$b = 01010111 = 21.75$  ( $01010111 / 4$ )

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

0000000001001010	(74)	promoción a <code>uint16_t</code>
* 0000000001010111	(87)	promoción a <code>uint16_t</code>
<hr/>		
0001100100100110	(6438)	
<hr/>		
0001100100100111	(6438+1)	
<hr/>		
0000011001001001	(6438+1 >> 2)	
<hr/>		
010010.01	(18.25)	

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50 \text{ (} 01001010 / 4 \text{)}$

$b = 01010111 = 21.75 \text{ (} 01010111 / 4 \text{)}$

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

0000000001001010	(74)	promoción a <code>uint16_t</code>
* 0000000001010111	(87)	promoción a <code>uint16_t</code>
<hr/>		
0001100100100110	(6438)	resultado en <code>uint16_t</code>
<hr/>		
0001100100100111	(6438+1)	
<hr/>		
0000011001001001	(6438+1 >> 2)	
<hr/>		
010010.01	(18.25)	

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50$  ( $01001010 / 4$ )

$b = 01010111 = 21.75$  ( $01010111 / 4$ )

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

0000000001001010	(74)	promoción a <code>uint16_t</code>
* 0000000001010111	(87)	promoción a <code>uint16_t</code>
<hr/>		
0001100100100110	(6438)	resultado en <code>uint16_t</code>
<hr/>		
0001100100100111	(6438+1)	ajustado + 1
<hr/>		
0000011001001001	(6438+1 >> 2)	
<hr/>		
010010.01	(18.25)	

# Multiplicación en punto fijo: detalles

## Representación: Q6.2

$a = 01001010 = 18.50$  ( $01001010 / 4$ )

$b = 01010111 = 21.75$  ( $01010111 / 4$ )

- Si se puede: promover enteros
- Ajustar resultado (+1)
- Desplazar (escalar correctamente)
- Truncar (o redondear)

## multiplicación

0000000001001010	(74)	promoción a <code>uint16_t</code>
* 0000000001010111	(87)	promoción a <code>uint16_t</code>
<hr/>		
0001100100100110	(6438)	resultado en <code>uint16_t</code>
<hr/>		
0001100100100111	(6438+1)	ajustado + 1
<hr/>		
0000011001001001	(6438+1 >> 2)	corregir escalado y truncar
<hr/>		
010010.01	(18.25)	

# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C

# Punto Flotante (IEEE<sup>1</sup>)

## Estándar IEEE 754

- Establecido en 1985
  - IEEE 754-1985 - IEEE Standard for Binary Floating-Point Arithmetic
  - IEEE 754-2008 - IEEE Standard for Floating-Point Arithmetic
  - última revisión: P754/D2.45, Jan 2019 - IEEE *Draft* Standard for Floating-Point Arithmetic
- Soporte general
  - Hardware
  - Software
- Desarrollado *mirando* las necesidades numéricas
  - Formas estándar de redondeo, *overflow*, *underflow*
  - Difícil obtener realizaciones veloces en hardware
    - Análisis numérico vs diseño de hardware

---

<sup>1</sup>IEEE: Instituto de Ingenieros Eléctricos y Electrónicos

# Representación

## Expresión numérica

$$f = (-1)^s M 2^E$$

- **s**: bit de signo
- **M**: significando (normalmente un valor en el intervalo  $[1.0, 2.0)$ )
- **E**: exponente, pesa el valor por una potencia de 2

## Codificación (almacenamiento)

- El msb **s** es el bit de signo **s**
- exp codifica **E**, pero no es E
- frac codifica **M**, pero no es M





# Precisión

## Precisión simple: 32 bits



## Precisión doble: 64 bits



## Precisión extendida: 80 bits (Intel)



# Valores: normalizados

$$v = (-1)^s M 2^E$$

- Cuando  $\text{exp} \neq 000\dots 0$  y  $\text{exp} \neq 111\dots 1$
- Exponente codificado por exceso:  $E = \text{Exp} - \text{bias}$ 
  - Exp: valor **unsigned** del campo exp
  - $\text{bias}$ :  $2^{k-1} - 1$ , donde  $k$  es el número de bits del campo exp
    - Simple: 127 (Exp: 1, ..., 254; **E**: -126, ..., 127)
    - Doble: 1023 (Exp: 1, ..., 2046; **E**: -1022, ..., 1023)
- Significando codificado con parte entera implícitamente 1:  $M = 1.sss\dots s_2$ 
  - $sss\dots s$ : bits del campo frac
  - Mínimo cuando  $\text{frac} = 00\dots 0$  (**M** = 1.0)
  - Máximo cuando  $\text{frac} = 11\dots 1$  (**M** =  $2.0 - \varepsilon$ )

## Ejemplo de codificación normalizada

$$v = (-1)^s M 2^E$$

$$E = \text{Exp} - \text{bias}$$

Valor: float  $F = 49374.0;$

$$\begin{aligned} \blacksquare 49374,0_{10} &= 1100000011011110_2 \\ &= 1.100000011011110_2 \times 2^{15} \end{aligned}$$

### Significando

$$M = 1.100000011011110_2$$

$$\text{frac} = 10000001101111000000000_2$$

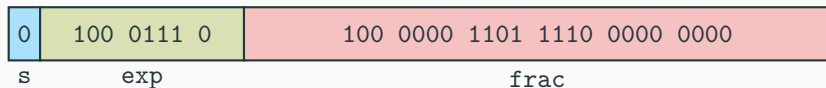
### Exponente

$$E = 15$$

$$\text{bias} = 127$$

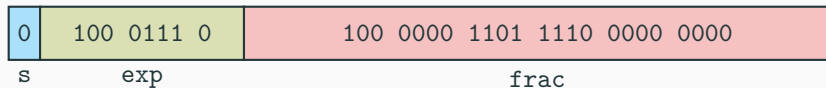
$$\text{exp} = 142 = 10001110$$

### Resultado



# Ejemplo de codificación normalizada

## Representación en punto flotante



<b>Hexa:</b>	4	7	4	0	D	E	0	0
<b>Binario:</b>	0100	0111	0100	0000	1101	1110	0000	0000
<b>142:</b>	100	0111	0					
<b>49374:</b>			1100	0000	1101	1110		

## Valores: de(s)normalizados

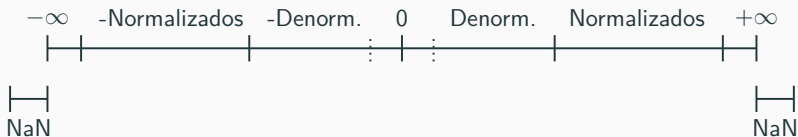
$$v = (-1)^s M 2^E$$
$$E = 1 - bias$$

- Cuando  $exp = 000\dots 0$
- Exponente:  $E = 1 - bias$  (en vez de  $0 - bias$ )
  - Simple: -126
  - Doble: -1022
- Significando codificado con parte entera implícitamente 0:  $M = 0.sss\dots s_2$ 
  - $sss\dots s$ : bits del campo *frac*
- Casos
  - $exp = 000\dots 0, frac = 000\dots 0$ 
    - Representa el valor cero (0)
    - Distintos valores para +0 y -0
  - $exp = 000\dots 0, frac \neq 000\dots 0$ 
    - Valores más cercanos a 0.0
    - Equiespaciados

# Valores especiales

- **Condición:**  $\text{exp} = 111\dots 1$
- Caso:  $\text{exp} = 111\dots 1$ ,  $\text{frac} = 000\dots 0$ 
  - Representa el valor  $\infty$  (infinito)
  - Operaciones que dan *overflow*
  - Hay positivo y negativo
  - Ej.:  $1,0/0,0 = -1,0/-0,0 = +\infty$ ,  $1,0/-0,0 = -\infty$
- Caso:  $\text{exp} = 111\dots 1$ ,  $\text{frac} \neq 000\dots 0$ 
  - NaN: Not-A-Number
  - Representa casos donde no se puede determinar un valor numérico
  - Ej.:  $\text{sqrt}(-1)$ ,  $\infty - \infty$ ,  $\infty \times 0$

# Visualización



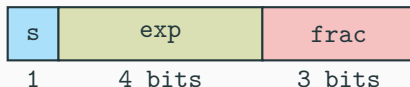
# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C



## Ejemplo con un float pequeño



### Representación en punto flotante de 8 bits

- el bit de signo es el msb
- el exponente es de 4 bits:  $bias = 7$
- la parte fraccionaria: 3 bits

### Misma idea que el formato IEEE

- normalizados, denormalizados
- representación del 0, NaN, infinito

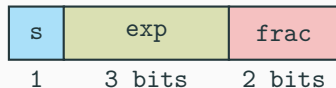
# Rango Dinámico (positivos)

	s	exp	frac	E	Valor	
Números denormalizados	0	0000	000	-6	0	cero
	0	0000	001	-6	$1/8 \cdot 2^{-6} = 1/512$	denormalizado menor
	0	0000	010	-6	$2/8 \cdot 2^{-6} = 2/512$	
	:					
	0	0000	110	-6	$6/8 \cdot 2^{-6} = 6/512$	denormalizado mayor
	0	0000	111	-6	$7/8 \cdot 2^{-6} = 7/512$	
Números Normalizados	0	0001	000	-6	$8/8 \cdot 2^{-6} = 8/512$	normalizado menor
	0	0001	001	-6	$9/8 \cdot 2^{-6} = 9/512$	
	:					
	0	0110	110	-1	$14/8 \cdot 2^{-1} = 14/16$	más cercano a 1 (menor) 1 más cercano a 1 (mayor)
	0	0110	111	-1	$15/8 \cdot 2^{-1} = 15/16$	
	0	0111	000	0	$8/8 \cdot 2^0 = 1$	
	0	0111	001	0	$9/8 \cdot 2^0 = 9/8$	
	0	0111	010	0	$10/8 \cdot 2^0 = 10/8$	
	:					
	0	1110	110	7	$14/8 \cdot 2^7 = 224$	normalizado mayor
	0	1110	111	7	$15/8 \cdot 2^7 = 240$	
	0	1111	000	--	$\infty$	infinito

# Distribución de valores: mini float

## float tipo IEEE de 6 bits

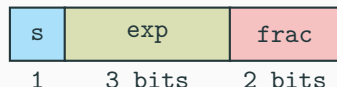
- fracción de 2 bits
- exponente de 3 bits
- $bias = 2^{3-1} - 1 = 3$



# Distribución de valores (zoom): mini float

## float tipo IEEE de 6 bits

- fracción de 2 bits
- exponente de 3 bits
- $bias = 2^{3-1} - 1 = 3$



# Propiedades de la codificación según el IEEE

**El cero de punto flotante (FP) es el mismo que con los enteros**

- Todos los bits en 0

**Casi sirve la comparación de enteros**

- Comparar primero los bits de signo
- Tener en cuenta que FP tiene 2 ceros ( $-0$  y  $+0$ )
- NaN es problemático
  - Es mayor que cualquier otro número
  - ¿Qué debe dar la comparación?
- Caso contrario, está bien
  - Denormalizados vs Normalizados
  - Normalizados vs Infinito

## Rangos (positivos)

{simple, doble}			
Descripción	exp	frac	Valor numérico
Cero	00...00	00...00	0,0
Denormalizado menor	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
simple $\approx 1,4 \times 10^{-45}$			
doble $\approx 4,9 \times 10^{-324}$			
Denormalizado mayor	00...00	11...11	$(1,0 - \varepsilon) \times 2^{-\{126,1022\}}$
simple $\approx 1,18 \times 10^{-38}$			
doble $\approx 2,2 \times 10^{-308}$			
Normalizado menor	00...01	00...00	$1,0 \times 2^{\{126,1022\}}$
Uno (1)	01...11	00...00	1,0
Normalizado mayor	11...10	11...11	$(2,0 - \varepsilon) \times 2^{\{127,1023\}}$

# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C

# Operaciones con punto flotante

- $u +_f v = \text{Round}(u + v)$
- $u \times_f v = \text{Round}(u \times v)$

## Idea básica

- Primero obtener el resultado **exacto**
- Ajustarlo a la precisión disponible
  - Puede haber overflow si el exponente es grande
  - **Redondear** para que entre en `frac`



# Redondeo

- Hay 4 modos de redondeo
  - Hacia 0,  $-\infty$ ,  $+\infty$ , el más cercano (*nearest*)

## Ejemplos

Hacia	\$1,40	\$1,60	\$1,50	\$2,50	−\$1,50
0	\$1,00	\$1,00	\$1,00	\$2,00	−\$1,00
abajo ( $-\infty$ )	\$1,00	\$1,00	\$1,00	\$2,00	−\$2,00
arriba ( $+\infty$ )	\$2,00	\$2,00	\$2,00	\$3,00	−\$1,00
el más cercano	\$1,00	\$2,00	\$2,00	\$2,00	−\$2,00

El redondeo por omisión es al (par) más cercano, llamado  
**Round-to-Even**

# Round-to-Even

- Es el modo por omisión
  - No es simple configurar otro modo
  - Todos los demás están estadísticamente sesgados

## Ejemplo: redoneando a 2 decimales

- Cuando se está justo en el medio entre 2 valores
  - Redondear para que el dígito menos significativo sea par

- Ejemplo

7.8949999	7.89	(Por debajo de la mitad)
7.8950001	7.90	(Por arriba de la mitad)
7.8950000	7.90	(Justo en la mitad—redondear hacia arriba)
7.8850000	7.88	(Justo en la mitad—redondear hacia abajo)

# Redondeo en binario

- Números fraccionarios
  - “Par” cuando el bit menos significativo es 0
  - “En la mitad” cuando los bits a la derecha de la posición de redondeo son:  $100...0_2$

## Ejemplo

- Redondear al  $1/4$  más cercano (2 bits después del punto)

Valor	Binario <sub>2</sub>	Redondeado <sub>2</sub>	Acción	Final
$2 \frac{3}{32}$	$10.000\textcolor{red}{11}_2$	$10.00_2$	( $<1/2$ –abajo)	2
$2 \frac{3}{16}$	$10.00\textcolor{red}{11}0_2$	$10.01_2$	( $>1/2$ –arriba)	$2 \frac{1}{4}$
$2 \frac{7}{8}$	$10.11\textcolor{red}{100}_2$	$11.00_2$	( $1/2$ –arriba)	3
$2 \frac{5}{8}$	$10.10\textcolor{red}{100}_2$	$10.10_2$	( $1/2$ –abajo)	$2 \frac{1}{2}$

# Multiplicación

Operación:  $(-1)^{s_1} M_1 2^{E_1} \times (-1)^{s_2} M_2 2^{E_2}$

**Resultado Exacto:**  $(-1)^s M 2^E$

- signo  $s$   $s_1 \wedge s_2$
- significando  $M$   $M_1 \times M_2$
- exponente  $E$   $E_1 + E_2$

## Ajuste

- Si  $M \geq 2$ , desplazar  $M$  a la derecha, incrementar  $E$
- Si  $E$  fuera de rango, *overflow*
- Redondear  $M$  para concuerde con la precisión de `frac`

## Implementación

- Lo más difícil es multiplicar los significandos

# Suma

Operación:  $(-1)^{s_1} M_1 2^{E_1} + (-1)^{s_2} M_2 2^{E_2}$

**Resultado Exacto:**  $(-1)^s M 2^E$

- signo **s**, significando **M**:
  - Resultado de alinear y sumar
- exponente **E**:  $E_1$

$$\begin{array}{r}
 \begin{array}{|c|} \hline (-1)^{s_1} M_1 \\ \hline \end{array} \xleftarrow{E_1 - E_2} \\
 + \qquad \begin{array}{|c|} \hline (-1)^{s_2} M_2 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|} \hline (-1)^s M \\ \hline \end{array}
 \end{array}$$

## Ajuste

- Si  $M \geq 2$ , desplazar M a la derecha, incrementar E
- Si  $M < 1$ , desplazar M a la izquierda, decrementar E
- Si E fuera de rango, *overflow*
- Redondear M para concuerde con la precisión de frac

# Propiedades matemáticas de la suma

## ■ Comparando con un grupo abeliano

- ¿Cerrado bajo la suma? Sí
- ¿Es conmutativa? Sí
- ¿Es asociativa? No
  - Puede ocurrir un *overflow*; es inexacto (redondea)
  - Ej.:  $(3.14+1e10) - 1e10 = 0.0$  ,  $3.14 + (1e10-1e10) = 3.14$
- ¿Es 0 la identidad aditiva? Sí
- ¿Todo elemento tiene un inverso? Casi
  - Excepto los infinitos y NaNs

## ■ Monotonicidad

- $a \geq b \ \& \ c \geq 0 \Rightarrow a + c \geq b + c?$  Casi
  - Salvo por infinitos y/o NaNs

# Propiedades matemáticas de la multiplicación

## ■ Comparando con un anillo conmutativo

- ¿Cerrado bajo la multiplicación? Sí
- ¿Es conmutativa la multiplicación? Sí
- ¿Es asociativa la multiplicación? No
  - Puede ocurrir un *overflow*; es inexacto (redondea)
  - Ej.:  $(1e20 * 1e20) * 1e-20 = \text{inf}$  ,  $1e20 * (1e20 * 1e-20) = 1e20$
- ¿Es 1 la identidad multiplicativa? Sí
- ¿Es distributiva sobre la suma? No
  - Puede ocurrir un *overflow*; es inexacto (redondea)
  - Ej.:  $1e20 * (1e20 - 1e20) = 0.0$  ,  $1e20 * 1e20 - 1e20 * 1e20 = \text{NaN}$

## ■ Monotonicidad

- $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c?$  Casi
  - Salvo por infinitos y/o NaNs

# Tabla de contenidos

---

1. Números fraccionarios en binario
2. Representación en punto fijo
3. Representación en punto flotante
4. Ejemplos y propiedades
5. Redondeo, suma y multiplicación
6. Punto Flotante en C



# Punto Flotante en C

- C garantiza 2 tipos
  - `float`: precisión simple
  - `double`: precisión doble
- Conversiones/*casting*
  - *casting* entre `int`, `float`, y `double` cambia la representación binaria
  - `double/float` → `int`
    - Trunca la parte fraccionaria
    - Como redondear hacia 0
    - No está definido cuando el valor está fuera de rango o es NaN: generalmente asigna TMin.
  - `int` → `double`
    - Conversión exacta, en tanto el `int` tenga menos de 53 bits
  - `int` → `float`
    - En general va a redondear (salvo enteros de 16 bits o menos)

# Floating Point Puzzles

- Argumentar si es cierta, o explicar por qué es falsa cada una de las siguientes expresiones de C:

```
1 int x = ...;  
2 float f = ...;  
3 double d = ...;
```

Asumir que no son  
NaN f ni d

1. `x == (int) (float) x;`
2. `x == (int) (double) x;`
3. `f == (float) (double) f;`
4. `d == (double) (float) d;`
5. `f == -(-f);`
6. `2/3 == 2/3.0;`
7. `d < 0  $\Rightarrow$  ((d*2) < 0)`
8. `d > f  $\Rightarrow$  -f > -d`
9. `d * d >= 0`
10. `(d + f) - d == f`

# Licencia del estilo de beamer

---

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

`github.com/matze/mtheme`

