Estructura de Computadores

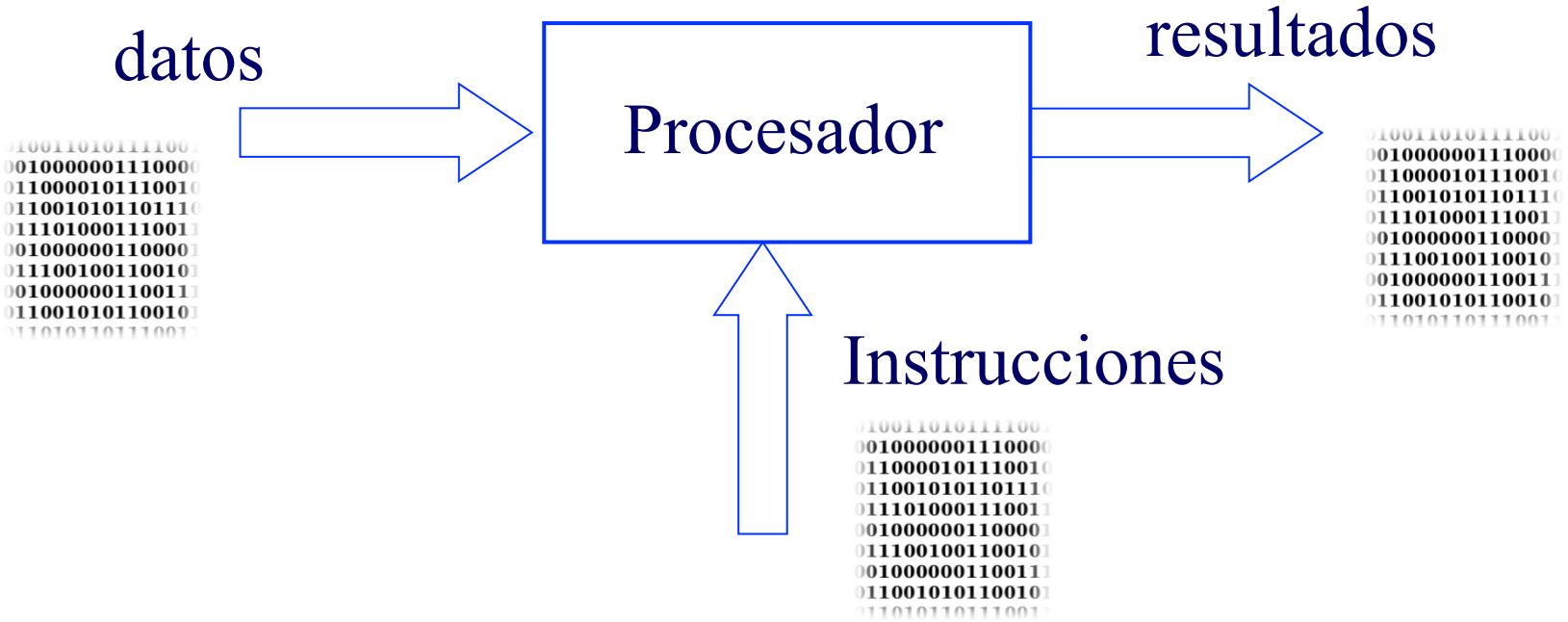
Tema 2. Representación de la información

Departamento de Informática
Grupo de Arquitectura de Computadores, Comunicaciones y Sistemas
UNIVERSIDAD CARLOS III DE MADRID

Contenido

- Repaso del concepto de computador
- Introducción a la representación de la información
  - Tipos de información a representar
  - Sistemas posicionales
- Representaciones
  - Alfanuméricas
  - Numéricas sin signo y con signo
  - Numéricas: coma flotante
    - Estándar IEEE 754

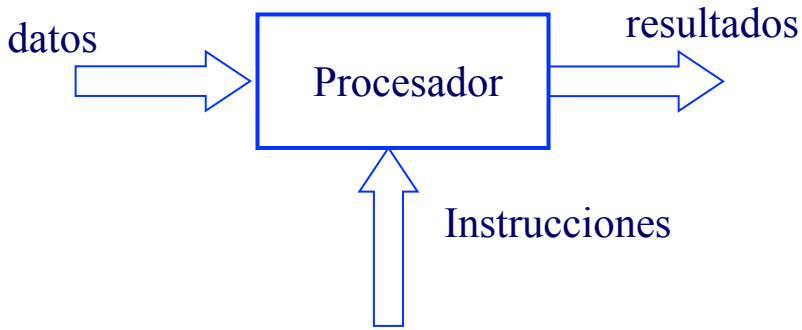
¿Qué es un computador?



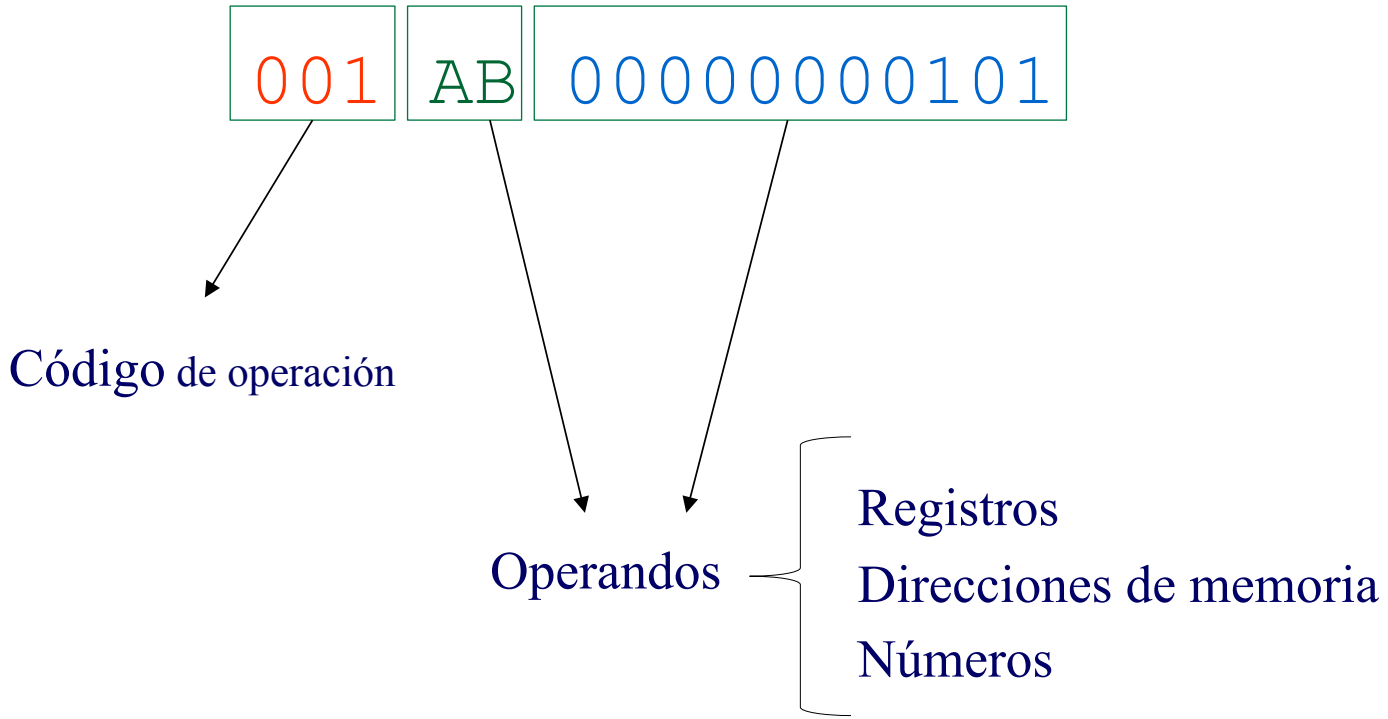
Toda la información de se representa en binario

Tipos de información

- Instrucciones máquina
- Datos
  - Caracteres
  - Números naturales
  - Números enteros (con signo)
  - Números reales



Formato de una instrucción máquina



Sistemas de representación posicionales

- Un número se define por una cadena de dígitos, estando afectado cada uno de ellos por un factor de escala que depende de la posición que ocupa en la cadena.
- Dada una base de numeración b, un número:

$$X = (\cdots x_2\ x_1\ x_0\ ,\ x_{-1}\ x_{-2}\ \cdots)_b$$

Con $0 \le x_i < b$

Su valor decimal es X:

$$V(X) = \sum_{i=-\infty}^{+\infty} b^i \cdot x_i = \cdots b^2 \cdot x_2 + b^1 \cdot x_1 + b^0 \cdot x_0 + b^{-1} \cdot x_{-1} + b^{-2} \cdot x_{-2} \cdots$$

Sistemas de representación posicionales

- Binario

  X = 1 0 1 0 0 1 0 1

  ... 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

- Hexadecimal

  Y = 0x F 1 F A 8 0

  ... 16^5 16^4 16^3 16^2 16^1 16^0

  - De binario a hexadecimal:
    - Agrupar de 4 en 4 bits, de derecha a izquierda
    - Cada 4 bits es el valor del dígito hexadecimal
    - Ej.: 1 0 1 0 0 1 0 1

      0x A 5

Sistemas de representación posicionales

- ¿Cuántos ‘valores’ (códigos) se pueden representar con n bits?
- ¿Cuántos bits se necesitan para representar m ‘valores’?

Sistemas de representación posicionales

- ¿Cuántos 'valores' (códigos) se pueden representar con n bits?
  - **2^n**
  - Ej.: con 8 bits se pueden representar 256 códigos posibles
- ¿Cuántos bits se necesitan para representar m 'valores'?
  - **$Log_2(n)$ por exceso**
  - Ej.: para representar 35 valores se necesitan 6 bits
- Con n bits
  - El valor mínimo representable es 0
  - El valor máximo representable es 2^n-1

Ejemplo

- Representar 342 en binario:

pesos	256	128	64	32	16	8	4	2	1
	?	?	?	?	?	?	?	?	?

Ejemplo

- Representar 342 en binario:

pesos	256	128	64	32	16	8	4	2	1
	1	0	1	0	1	0	1	1	0
	342-256=86		86-64=22		22-16=6		6-4=2		2-2=0

Ejemplo

- Calcular el valor decimal de 23 unos:

$$11111111111111111111111_2$$

Ejemplo

- Calcular el valor decimal de 23 unos:

$$11111111111111111111111_2$$

$$X = 2^{23} - 1$$

Truco:

$$\begin{array}{rll} & 11111111111111111111111_2 & = X \\ + & 00000000000000000000001_2 & = 1 \\ \hline & 10000000000000000000000_2 & = 2^{23} \end{array}$$

$$X = 2^{23} - 1$$

Ejemplo de suma

$$
\begin{array}{cccccc}
 & 1 & 1 & & & \\
 & 1 & 0 & 1 & 0 & \\
+ & 0 & 1 & 1 & 1 & \\
\hline
1 & 0 & 0 & 0 & 1 &
\end{array}
$$

Prefijos

Nombre	Abr	Factor	SI
Kilo	K	2^{10} = 1,024	10^{3} = 1,000
Mega	M	2^{20} = 1,048,576	10^{6} = 1,000,000
Giga	G	2^{30} = 1,073,741,824	10^{9} = 1,000,000,000
Tera	T	2^{40} = 1,099,511,627,776	10^{12} = 1,000,000,000,000
Peta	P	2^{50} = 1,125,899,906,842,624	10^{15} = 1,000,000,000,000,000
Exa	E	2^{60} = 1,152,921,504,606,846,976	10^{18} = 1,000,000,000,000,000,000
Zetta	Z	2^{70} = 1,180,591,620,717,411,303,424	10^{21} = 1,000,000,000,000,000,000,000
Yotta	Y	2^{80} = 1,208,925,819,614,629,174,706,176	10^{24} = 1,000,000,000,000,000,000,000,000

- 1 KB = 1024 bytes, pero en el SI es 1000 bytes
- Los fabricantes de disco duros y en telecomunicaciones emplea el SI.
  - Un disco duro de 30 GB almacena 30 x 10^{9} bytes
  - Una red de 1 Mbit/s transfiere 10^{6} bps.

Ejemplo

- ¿Cuántos bytes tiene un disco duro de 200 GB?
- ¿Cuántos bytes por segundo transmite mi ADSL de 20 Mb?

Solución

- ¿Cuántos bytes tiene un disco duro de 200 GB?
  - 200 GB = 200 * 109bytes = 186.26 Gigabytes

- ¿Cuántos bytes por segundo transmite mi ADSL de 20 Mb?
  - B → Byte
  - b → bit.
  - 20 Mb = 20 * 10^6bits = 20 * 10^6 / 8 bytes = 2.38 Megabytes por segundo

Tamaños privilegiados

- Octeto, carácter o byte
  - Representación de un carácter
  - Típicamente 8 bits
- Palabra
  - Información manipulada en paralelo en el interior del computador
  - Típicamente 32, 64 bits
- Media palabra
- Doble palabra

Representación alfanumérica

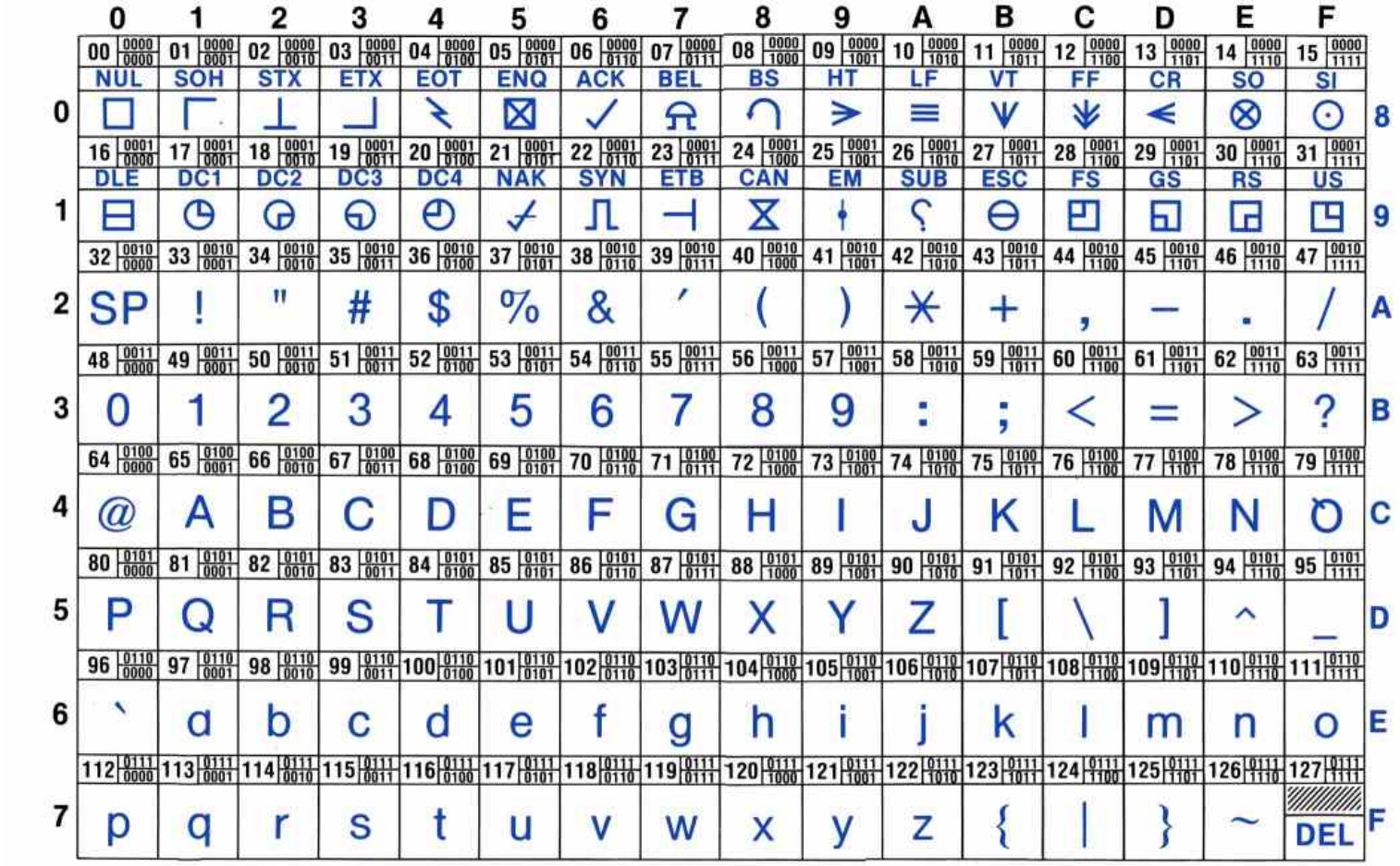
- Cada carácter se codifica con un octeto.
- Para n bits ⇒ 2^n caracteres representables:
  - 6 bits (64 caracteres)
    - 26 letras (A...Z), 10 números (0...9), puntuación(. , ; : ...) y especiales (+ - [...)
    - Ejemplo: ***BCDIC***
  - 7 bits (128 caracteres)
    - añade mayúsculas y minúsculas y caracteres de control de periféricos
    - Ejemplo: ***ASCII***
  - 8 bits (256 caracteres)
    - añade letras acentuadas, ñ, caracteres semigráficos
    - Ejemplo: ***EBCDIC*** y ***ASCII ex(10dido*** (actual)
  - 16 bits (34.168 caracteres)
    - distintos idiomas (chino, árabe,...)
    - Ejemplo: ***UNICODE***

Representación de caracteres

- Sistemas
  - EBCDIC (8 bits)
  - ASCII (8 bits)
  - Unicode (8 bits)

- Correspondencia de un código numérico a cada carácter representado

Ejemplo: tabla ASCII (7 bits)



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	00 0000 0000	01 0000 0001	02 0000 0010	03 0000 0011	04 0000 0100	05 0000 0101	06 0000 0110	07 0000 0111	08 0000 1000	09 0000 1001	10 0000 1010	11 0000 1011	12 0000 1100	13 0000 1101	14 0000 1110	15 0000 1111	
	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	
0	□	⌜	⊥	⌟	↘	⊠	✓	品	↶	➢	≡	⩔	⤋	⮜	⊗	⊙	8
	16 0001 0000	17 0001 0001	18 0001 0010	19 0001 0011	20 0001 0100	21 0001 0101	22 0001 0110	23 0001 0111	24 0001 1000	25 0001 1001	26 0001 1010	27 0001 1011	28 0001 1100	29 0001 1101	30 0001 1110	31 0001 1111	
	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	
1	⊟	◔	◶	◵	◴	✓	⎍	⊣	⧖	⫲	ς	⊖	◱	◱	◱	◳	9
	32 0010 0000	33 0010 0001	34 0010 0010	35 0010 0011	36 0010 0100	37 0010 0101	38 0010 0110	39 0010 0111	40 0010 1000	41 0010 1001	42 0010 1010	43 0010 1011	44 0010 1100	45 0010 1101	46 0010 1110	47 0010 1111	
2	SP	!	"	#	$	%	&	′	(	)	*	+	,	−	.	/	A
	48 0011 0000	49 0011 0001	50 0011 0010	51 0011 0011	52 0011 0100	53 0011 0101	54 0011 0110	55 0011 0111	56 0011 1000	57 0011 1001	58 0011 1010	59 0011 1011	60 0011 1100	61 0011 1101	62 0011 1110	63 0011 1111	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	B
	64 0100 0000	65 0100 0001	66 0100 0010	67 0100 0011	68 0100 0100	69 0100 0101	70 0100 0110	71 0100 0111	72 0100 1000	73 0100 1001	74 0100 1010	75 0100 1011	76 0100 1100	77 0100 1101	78 0100 1110	79 0100 1111	
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	C
	80 0101 0000	81 0101 0001	82 0101 0010	83 0101 0011	84 0101 0100	85 0101 0101	86 0101 0110	87 0101 0111	88 0101 1000	89 0101 1001	90 0101 1010	91 0101 1011	92 0101 1100	93 0101 1101	94 0101 1110	95 0101 1111	
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	D
	96 0110 0000	97 0110 0001	98 0110 0010	99 0110 0011	100 0110 0100	101 0110 0101	102 0110 0110	103 0110 0111	104 0110 1000	105 0110 1001	106 0110 1010	107 0110 1011	108 0110 1100	109 0110 1101	110 0110 1110	111 0110 1111	
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	E
	112 0111 0000	113 0111 0001	114 0111 0010	115 0111 0011	116 0111 0100	117 0111 0101	118 0111 0110	119 0111 0111	120 0111 1000	121 0111 1001	122 0111 1010	123 0111 1011	124 0111 1100	125 0111 1101	126 0111 1110	127 0111 1111	
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	F

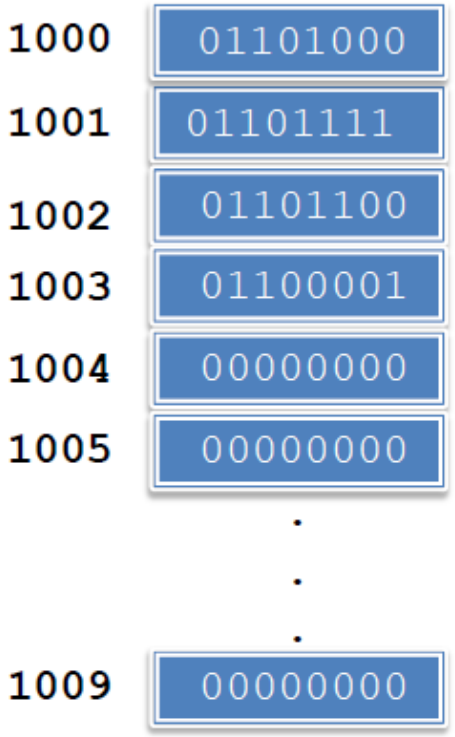
Código ASCII. Propiedades

- Caracteres de ‘0’ a ‘9’ son consecutivos
  - Simplifica comprobación de dígito
  - Simplifica la operación de obtener el valor numérico
  - ¿Por qué?
- Mayúsculas y minúsculas se diferencia en un bit
  - Simplifica conversión de mayúsculas a minúsculas
- Caracteres de control situados en un rango
  - Simplifica su interpretación

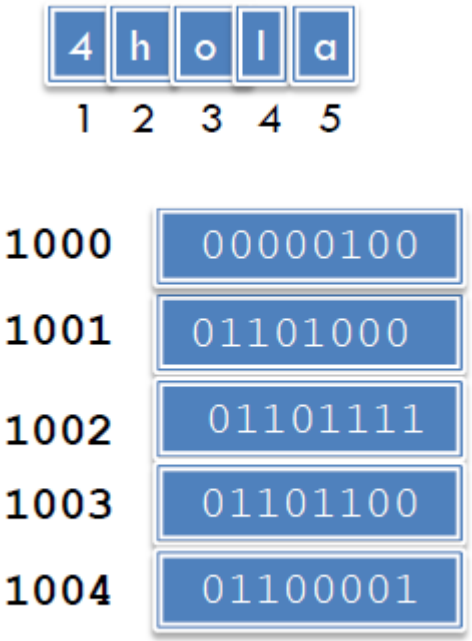
Cadenas de caracteres

- Cadenas de longitud fija
- Cadenas de longitud variable con separador
- Cadenas de longitud variable con longitud en cabecera

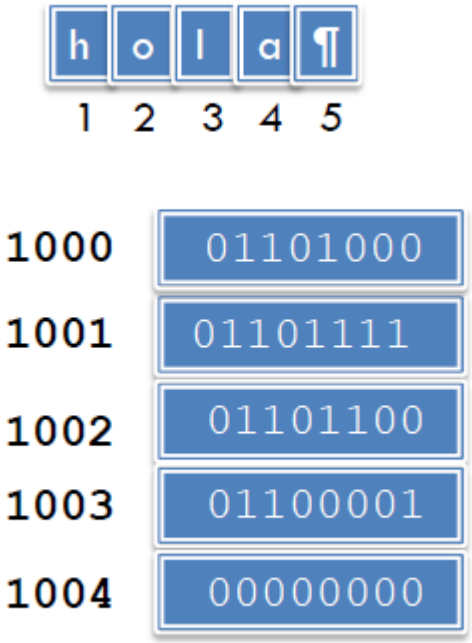
Cadenas de longitud fija



Cadenas con longitud en la cabecera



Cadenas con separador



Representación numérica

- Clasificación de números:
  - Naturales: 0, 1, 2, 3, ...
  - Enteros: ... -3, -2, -1, 0, 1, 2, 3, ...
  - Racionales: fracciones (5/2 = 2,5)
  - Irracionales: $2^{1/2}$, π, e, ...

- Conjuntos infinitos y espacio de representación finito
  - Imposible representar todos

Problemas de la representación de números en el computador

- Cualquier conjunto numérico es infinito
- Números irracionales no son representables por requerir infinitos dígitos
- Espacio material de representación finito
- Una secuencia de n bits permite representar 2^n **códigos distintos**

Características de la representación usada

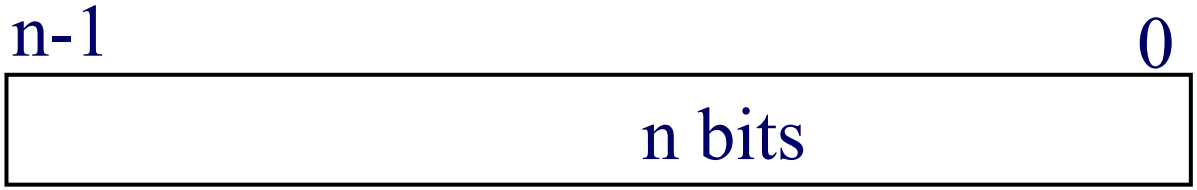
- Rango de representación:
  - Intervalo entre el menor y mayor no representable
- Precisión:
  - No todos los números son representables de forma exacta
- Resolución de representación:
  - Diferencia entre un nº representable y el inmediato siguiente
  - Resolución = máximo error cometido en la representación
  - La resolución puede ser:
    - Constante a lo largo de todo el rango
    - Variable a lo largo del rango (coma flotante)

Sistemas de representación binarios más usados

- Nº naturales (sin signo)
  - Coma fija sin signo o binario puro
- Números enteros (con signo):
  - Signo magnitud
  - Complemento a uno 1
  - Complemento a dos
  - Representación en exceso o sesgada
- Nº reales
  - Coma flotante: Estándar IEEE 754

Coma fija sin signo o binario puro

- Sistema posicional con base 2 y sin parte fraccionaria.



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i$$

- Rango de representación: $[0, 2^n - 1]$
- Resolución: 1 unidad

Representación de números con signo

- Signo-magnitud
- Complemento a uno
- Complemento a dos
- Representación en exceso

Signo magnitud

- Se reserva un bit (S) para el signo (0 ⇒ +; 1 ⇒ -)

$$\text{Si } x_{n-1} = 0 \quad V(X) = \sum_{i=0}^{n-2} 2^i \cdot x_i$$

$$\text{Si } x_{n-1} = 1 \quad V(X) = -\sum_{i=0}^{n-2} 2^i \cdot x_i$$

$$\Rightarrow V(X) = (1 - 2 \cdot x_{n-1}) \cdot \sum_{i=0}^{n-2} 2^i \cdot x_i$$

Rango de representación: $[-2^{n-1}+1,\ 2^{n-1}-1]$
Resolución: 1 unidad

Ejemplo

- Si n = 6 bits
- El número 7 se presenta como: 00111
  - El primer bit indica el signo
- El número -7 se representa como: 10111
  - El primer bit indica el signo

Ejemplo

- ¿Se puede representar el número 745 en binario usando 10 bits con representación en signo-magnitud?

Ejemplo

- ¿Se puede representar el número 745 en binario usando 10 bits con representación en signo-magnitud?

- Solución:
  - Con 10 bits el rango de representación en signo-magnitud es: $[-2^9+1,\ldots,-0,+0,\ldots.2^9-1] \Rightarrow [-511, 511]$
    Por tanto, no se puede representar el 745

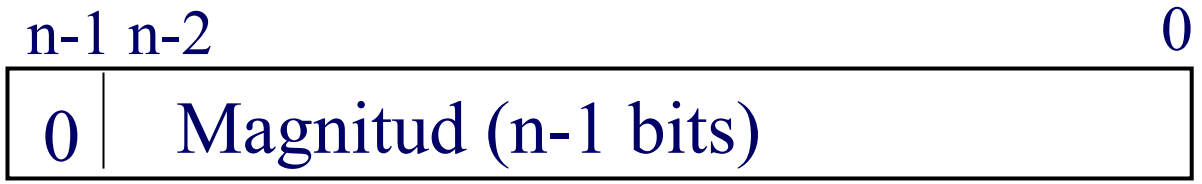
Problemas de la representación en signo-magnitud

- Doble representación del 0:
  - Con n = 5 bits:
    - 00000 representa el 0
    - 10000 representa el 0

- Circuitos diferentes para sumas y resta

Complemento a uno

- **Número positivo**:
  - Se representa en binario puro con n-1 bits

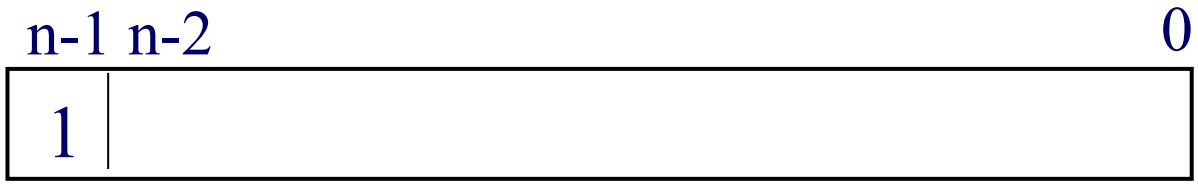


$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i = \sum_{i=0}^{n-2} 2^i \cdot x_i$$

- Rango de representación: $[0, 2^{n-1} - 1]$
- Resolución: 1 unidad

Complemento a uno

- **Número negativo**:
  - El número $X < 0$ se representa como $2^n - X - 1$
    - Se complementa: cambian los 0´s por 1´s y los 1´s por 0´s
    - El resultado es un número que tiene un 1 en el bit superior
      - Este bit no es un bit de signo, forma parte del valor del número

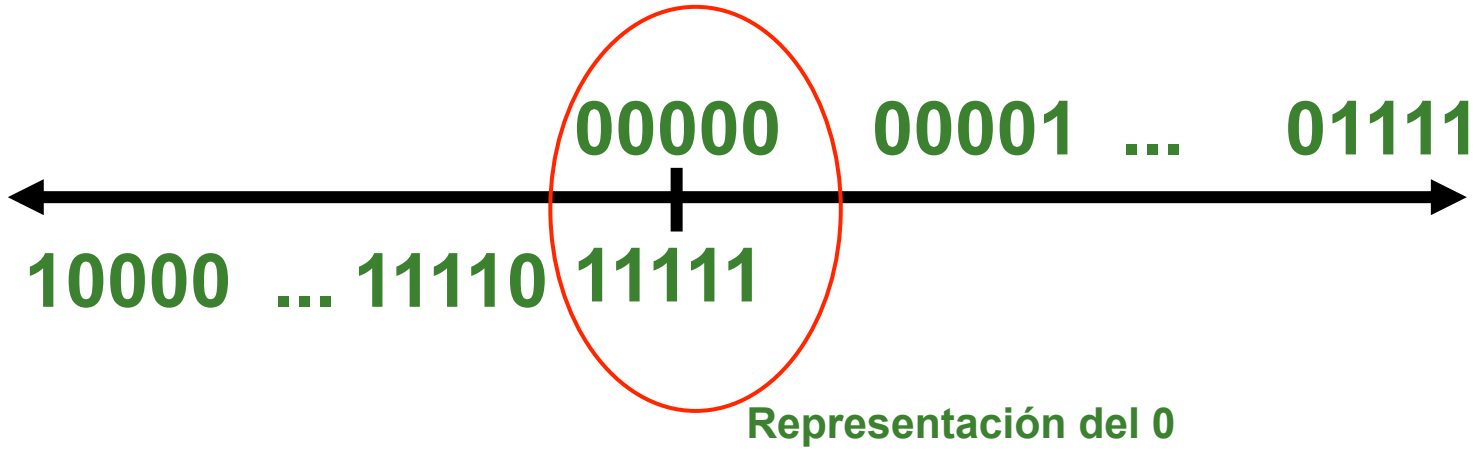


$$V(X) = -2^n + \sum_{i=0}^{n-1} 2^i \cdot x_i - 1$$

- Rango de representación: $[-2^{n-1}+1, -0]$
- Resolución: 1 unidad

Complemento a uno

- Los números positivos tienen 0s a la izquierda y los negativos 1s



Ejemplo

- Para n = 5 bits
- ¿Cómo se representa X = 5?
  - Como es positivo, en binario puro
    - 00101
- ¿Cómo se representa X = -5?
  - Como es negativo, se complementa el valor 5 (00101)
    - 11010
- ¿Cuál es el valor de 00111 en complemento a 2?
  - Como es positivo, su valor es directamente 7
- ¿Cuál es el valor de 11000 en complemento a 2?
  - Como es negativo, se complementa y se obtiene 00111 (7)
    - El valor es -7

Sumas y restas

- Para n = 5 bits
- Sea X = 5
  - En complemento a uno = 00101
- Sea Y = 7
  - En complemento a uno = 00111
- ¿X + Y?

X	=	00101
Y	=	00111+
X+Y	=	01100

- El valor de 01100 en complemento a uno es 12

Sumas y restas

- Para n = 5 bits
- Sea X = -5
  - En complemento a uno = complemento de 00101: 11010
- Sea Y = -7
  - En complemento a uno = complemento de 00111: 11000
- ¿X + Y?

  -X = 11010

  -Y = 11000+

  -(X+Y) = 110010 Se produce un acarreo, se suma y se desprecia

  1

  10011
- El valor de 10011 en complemento a uno es el valor negativo de su complemento -01100 = - 12

¿Porqué se desprecia el acarreo y se suma al resultado?

- -X se representa como $2^n - X - 1$
- -Y se representa como $2^n - Y - 1$
- $-(X + Y)$ se representa como $2^n - (X+Y) - 1$

- Cuando sumamos directamente $-X - Y$ se obtiene

$$-X = 2^n - X - 1$$
$$-Y = \underline{2^n - Y - 1}$$
$$-(X+Y) = 2^n + 2^n - (X + Y) - 2$$

Se corrige el resultado despreciando el acarreo

2^n (bit de acarreo) y se suma al resultado

$$\Rightarrow 2^n - (X + Y) - 1$$

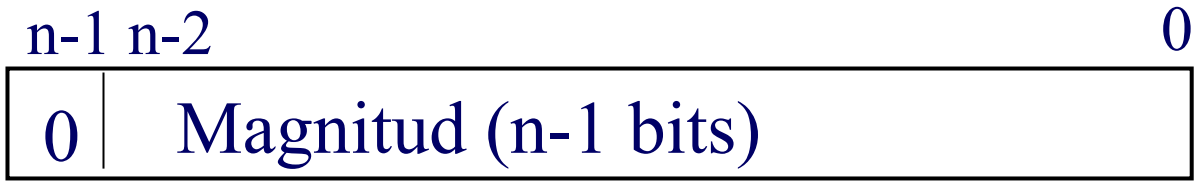
Problemas del complemento a uno

- Doble representación del 0
- Con n = 5 bits
  - 00000 representa el 0
  - 11111 representa el 0

- Rango de representación para positivos: $[0,\ -2^{n-1}-1]$
- Rango de representación para negativos: $[-(2^{n-1}-1),\ 0]$

Complemento a dos

- **Número positivo**:
  - Se representa en binario puro con n-1 bits

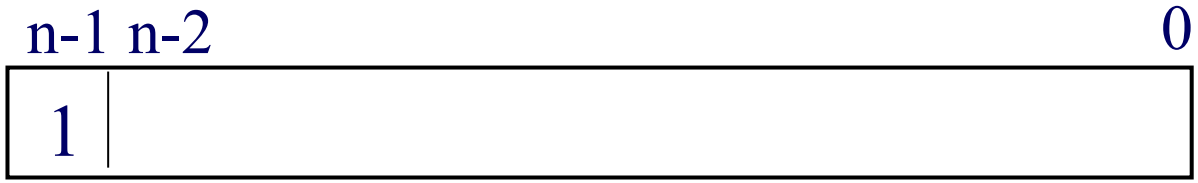


$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i = \sum_{i=0}^{n-2} 2^i \cdot x_i$$

- Rango de representación: $[0, 2^{n-1} - 1]$
- Resolución: 1 unidad

Complemento a dos

- **Número negativo**:
  - se complementa a la base. El número X< 0 se representa como $2^n - X$
  - El bit superior es 1: No es un bit de signo, forma parte del valor del número



$$V(X) = -2^n + \sum_{i=0}^{n-1} 2^i \cdot y_i$$

- Rango de representación: $[-2^{n-1}, -1]$
- Resolución: 1 unidad

Complemento a dos

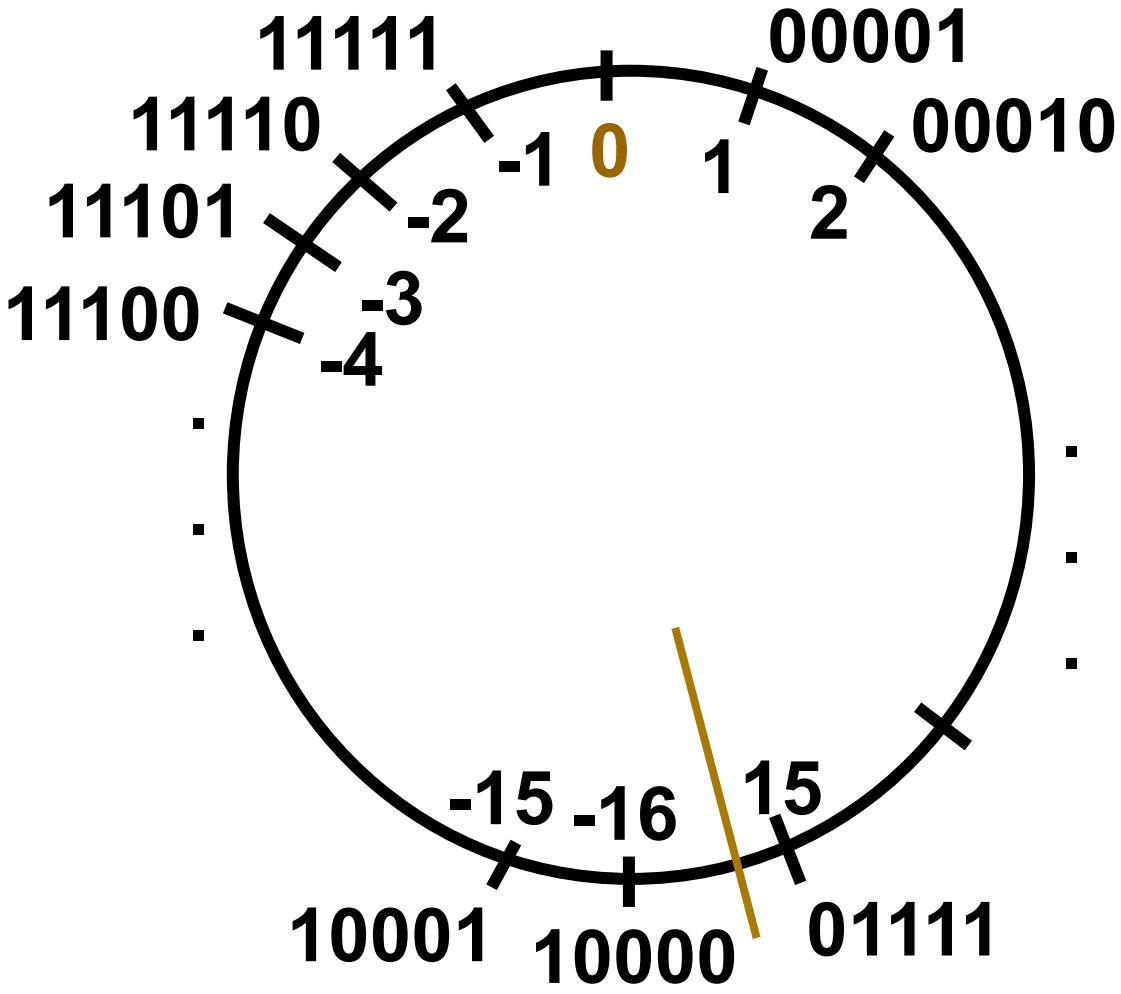
- Truco: Si X > 0, C a 2 de X = X

  Si X < 0, C a 2 de -X = C a 1 de X + 1

  - Ejemplo: Para n=5 ⇒ 00011 = + 3
  - Ejemplo: Para n=5 el -3 = 11101
    - 11101 ⇒ Para ob(10er su valor en C a 2, es el valor negativo de C a 1(11101) + 1 = 00010 + 1 = 00011
      - Es decir -3

- Rango de representación: $[-2^{n-1}, 2^{n-1}-1]$
- Resolución: 1 unidad
- El 0 tiene una única representación (No ∃ -0)
- Rango asimétrico

Complemento a dos



- 2^{N-1} no negativos
- 2^{N-1} negativos
- Un cero

Complemento a dos para 32 bits

$0000 \ldots 0000\ 0000\ 0000\ 0000_{dos} = 0_{(10}$

$0000 \ldots 0000\ 0000\ 0000\ 0001_{dos} = 1_{(10}$

$0000 \ldots 0000\ 0000\ 0000\ 0010_{dos} = 2_{(10}$

. . .

$0111 \ldots 1111\ 1111\ 1111\ 1101_{dos} = 2{,}147{,}483{,}645_{(10}$

$0111 \ldots 1111\ 1111\ 1111\ 1110_{dos} = 2{,}147{,}483{,}646_{(10}$

$0111 \ldots 1111\ 1111\ 1111\ 1111_{dos} = 2{,}147{,}483{,}647_{(10}$

$1000 \ldots 0000\ 0000\ 0000\ 0000_{dos} = -2{,}147{,}483{,}648_{(10}$

$1000 \ldots 0000\ 0000\ 0000\ 0001_{dos} = -2{,}147{,}483{,}647_{(10}$

$1000 \ldots 0000\ 0000\ 0000\ 0010_{dos} = -2{,}147{,}483{,}646_{(10}$

. . .

$1111 \ldots 1111\ 1111\ 1111\ 1101_{dos} = -3_{(10}$

$1111 \ldots 1111\ 1111\ 1111\ 1110_{dos} = -2_{(10}$

$1111 \ldots 1111\ 1111\ 1111\ 1111_{dos} = -1_{(10}$

Sumas y restas

- Para n = 5 bits
- Sea X = 5
  - En complemento a dos= 00101
- Sea Y = 7
  - En complemento a uno = 00111
- ¿X + Y?

  X = 00101

  Y = 00111+

  X+Y = 01100
- El valor de 01100 en complemento a uno es 12

Sumas y restas

- Para n = 5 bits
- Sea X = -5
  - En complemento a dos= complemento de 00101: 11010 + 1 = 11011
- Sea Y = -7
  - En complemento a uno = complemento de 00111: 11000 +1 = 11001
- ¿X + Y?

  -X = 11011

  -Y = 11001+

  -(X+Y) = 110100 Se produce un acarreo: se desprecia

- El valor de 10100. Su valor en complemento a dos = el valor negativo de su complemento a dos = complemento a uno: 01011 + 1 = 01100 = >- 12

Sumas y restas

- Para n = 5 bits
- Sea X = 8
  - En complemento a dos= 01000
- Sea Y = 9
  - En complemento a uno = 01001
- ¿X + Y?

  X = 01000

  Y = 01001+

  X+Y = 10001
- Se obtiene un negativo ⇒ desbordamiento

Sumas y restas

- Para n = 5 bits
- Sea X = -8
  - En complemento a dos= complemento de 01000: 10111 + 1 = 11000
- Sea Y = -9
  - En complemento a uno = complemento de 01001: 10110 +1 = 10111
- ¿X + Y?

  -X = 11000

  -Y = 10111+

  -(X+Y) = 101111 Se produce un acarreo: se desprecia

- El valor 01111, como es positivo ⇒ desbordamiento

Desbordamientos en complemento a dos

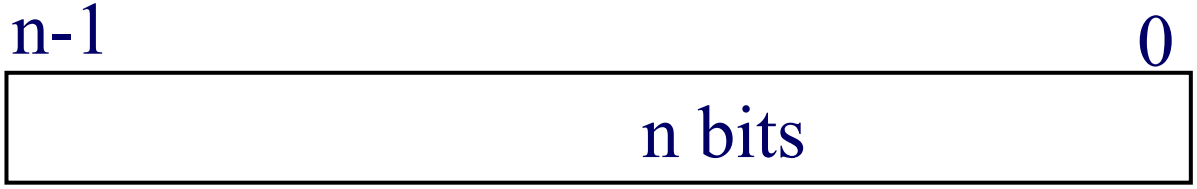
- Suma de dos negativos ⇒ positivo
- Suma de dos positivos ⇒ negativo

Extensión de signo en complemento a dos

- ¿Cómo pasar de n bits a m bits, siendo n < m?
- Ejemplo:
  - n = 4, m = 8
  - Si X = 0110 con 4 bits ⇒ X = 00000110 con 8 bits
  - Si X = 1011 con 4 bits ⇒ X = 11111011 con 8 bits

Representación en exceso (2^{n-1}-1)

- Con n bits, se suma 2^{n-1}-1 al valor a representar



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i - (2^{n-1} - 1)$$

- Rango de representación: $[-2^{n-1}+1, 2^n-1]$
- Resolución: 1 unidad
- No existe ambigüedad con el 0

Ejemplo comparativo (3 bits)

Decimal	Binario Puro	Signo magnitud	Complemento a uno	Complemento a dos	Exceso 3
+7	111	N.D.	N.D.	N.D.	N.D.
+6	110	N.D.	N.D.	N.D.	N.D.
+5	101	N.D.	N.D.	N.D.	N.D.
+4	100	N.D.	N.D.	N.D.	111
+3	011	011	011	011	110
+2	010	010	010	010	101
+1	001	001	001	001	100
+0	000	000	000	000	011
-0	N.D.	100	111	N.D.	N.D.
-1	N.D.	101	110	111	010
-2	N.D.	110	101	110	001
-3	N.D.	111	100	101	000
-4	N.D.	N.D.	N.D.	100	N.D.
-5	N.D.	N.D.	N.D.	N.D.	N.D.
-6	N.D.	N.D.	N.D.	N.D.	N.D.
-7	N.D.	N.D.	N.D.	N.D.	N.D.

Ejemplo

- Dado el valor 110110 (6 bits)
- ¿Cuál es su valor?
- En binario puro = $2^5 + 2^4 + 2^2 + 2^1 = 52_{(10}$
- En signo-magnitud = $- (2^4 + 2^2 + 2^1) = -21_{(10}$
- En complemento a uno
  - Se complementa ⇒ 001001 = $9_{(10}$
  - Su valor es $-9_{(10}$
- En complemento a dos
  - Se complementa ⇒ 001001 = 9 y se suma 1 = 001010 = 10
  - Su valor es $-10_{(10}$
- En exceso ($2^{6-1} -1 = 31$)
  - Valor de $110110_{(2} = 52_{(10}$
  - Valor almacenado = $52 – 31 = 21_{(10}$

Ejemplo

- Indique la representación de los siguientes números, razonando brevemente su respuesta:

1. -17 en signo magnitud con 6 bits
2. +16 en complemento a dos con 5 bits
3. -16 en complemento a dos con 5 bits
4. +15 en complemento a uno con 6 bits

Solución

1. 110001
2. Con 5 bits no es representable en C2:
   $[-2^{5-1} , 2^{5-1}-1] = [-16 , 15]$
3. 10000
4. 001111

Ejemplo

- Usando 5 bits para representarlo, haga las siguientes sumas en complemento a uno:
  a) 4 +12
  b) 4 -12
  c) –4 -12

Solución

- Usando 5 bits en complemento a uno:

a) 4 +12

00100
01100

10000 ⇒ -15 ⇒ overflow

Solución

- Usando 5 bits en complemento a uno:

b) 4 - 12

```
  00100
  10011
  -----
  10111 ⇒ -8
```

Solución

- Usando 5 bits en complemento a uno:

  c) -4 - 12

$$
\begin{array}{r}
11011 \\
10011 \\
\hline
101110
\end{array}
\Rightarrow \text{necesita 6 bits} \Rightarrow \text{overflow}
$$

Repaso

- Con N bits se pueden representar:
  - 2^N códigos distintos
  - Enteros sin signo:

    0 a $2^N - 1$

    para N=32, $2^N - 1 = 4.294.967.295$

  - Enteros con signo en complemento a dos

    $-2^{(N-1)}$ a $2^{(N-1)} - 1$

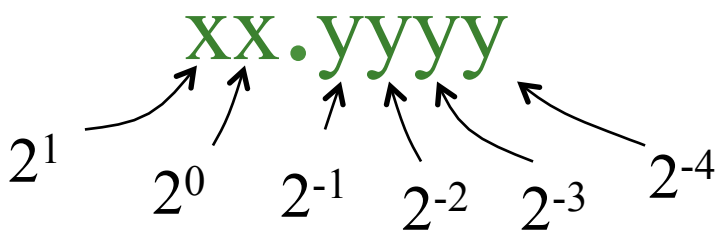
    para N=32, $2^{(N-1)} = 2.147.483.648$

Otras necesidades de representación

- ¿Cómo representar?
  - Números muy grandes: $30.556.926.000_{(10}$
  - Números muy pequeños: $0.00000000000529177_{(10}$
  - Números con decimales: 1,58567

Representación de fracciones con representación binaria en coma fija

- Ejemplo de representación con 6 bits:



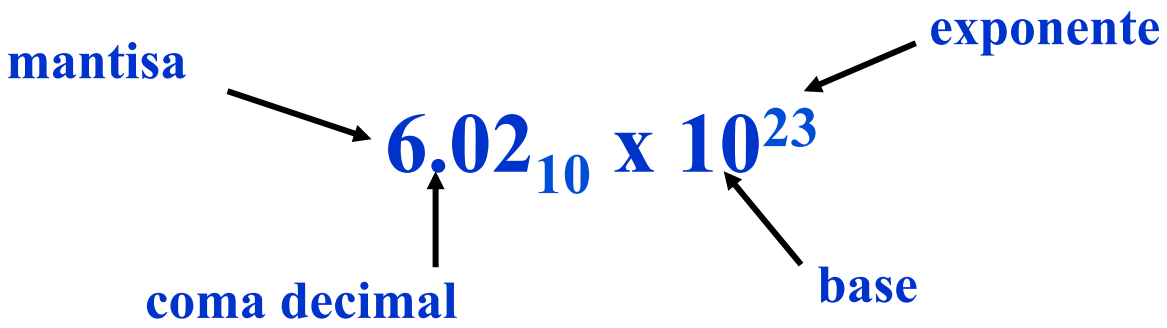
- $10{,}1010_{(2} = 1x2^1 + 1x2^{-1} + 1x2^{-3} = 2.62510$
- Asumiendo esta coma fija el rango sería:
  - 0 a 3.9375 (casi 4)

Potencias negativas

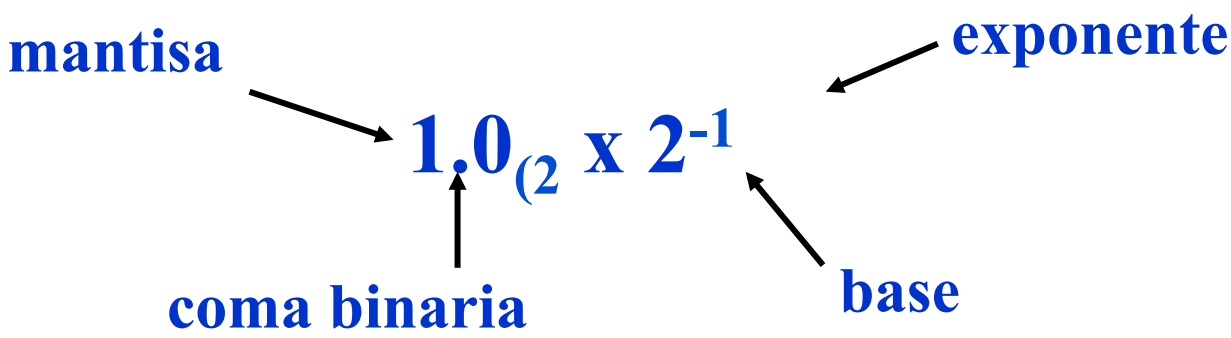
i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.251/4	
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	

Representación en coma flotante

- Cada número lleva asociado un exponente
- Permite adaptar número al orden de magnitud del valor a representar, trasladando la *coma decimal* —mediante un exponente
- Notación científica decimal → notación normalizada, solo un dígito distinto de 0 a la izquierda del punto



Notación científica en binario



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

Estándar IEEE 754

- Estándar para el almacenamiento en coma flotante utilizado por la mayoría de los ordenadores.

Bit de signo	Exponente sesgado	Parte significativa o mantisa

- **Características** (salvo casos especiales):
  - Exponente: en exceso, con sesgo 2^{n-1} -1 (siendo n el nº de bits del Exponente
  - Mantisa: signo-magnitud, normalizada con bit implícito de la forma
    M = 1,xx…

Estándar IEEE 754

Formato	Bits	Base	Mantisa	Exponente	Exceso a
Simple	32	2	23 bits	8 bits	127
Doble	64	2	52 bits	11 bits	1023
Cuádruple	128	2	112 bits	15 bits	16383

Además existen formatos con base 10 de 32, 64 y 128 bits

`http://speleotrove.com/decimal/`

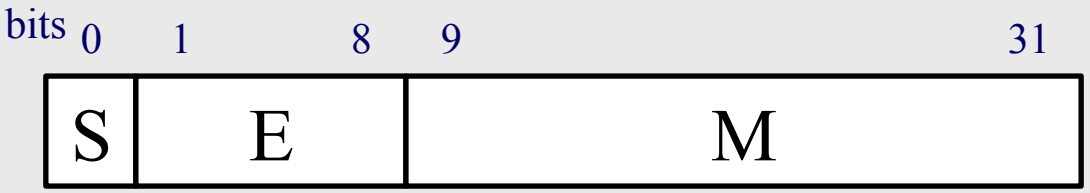
Números normalizados

- 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

- Normalización: Es preciso normalizar la mantisa, es decir, el exponente se ajusta para que el bit más significativo de la mantisa sea 1
  - Ejemplo: **1,111001** x 2^3 (ya está normalizado)
  - Ejemplo: 1111,101 x 2^3 no está normalizado, se desplaza la ,
    - 1111,101 x 2^3 = 1,111101 x 2^6
    - 1,111101 x 2^6 si está normalizado

Estándar IEEE 754 de precisión simple



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 excepciones)

  00000000000000000000000 ≤ M ≤ 11111111111111111111111

- Bit implícito: Una vez normalizado, el bit más significativo es 1, **no** se almacena en M para dejar espacio para un bit más (aumenta la precisión)

Ejemplo

- Representar 7,5 y 1,5 usando el formato IEEE 754

Solución

$$7{,}5 = 111{,}1 \times 2^0 = 1{,}111 \times 2^2$$

$$1{,}5 = 1{,}1 \times 2^0$$

Solución

$$7{,}5 = 111{,}1 \times 2^0 = 1{,}111 \times 2^2$$

Signo = 0 (positivo)
Exponente = 2 -> exponente a almacenar = 2 + 127= 129 = 10000001
Mantisa = 1,111 -> mantisa a almacenar = 1110000 … 0000

$$1{,}5 = 1{,}1 \times 2^0$$

Signo = 0 (positivo)
Exponente = 0 -> exponente a almacenar = 0 + 127= 127 = 01111111
Mantisa = 1,1 -> mantisa a almacenar = 1000000 … 0000

Solución

$7{,}5 = 111{,}1 \times 2^0 = 1{,}111 \times 2^2$

7,5 →

0	10000001	11100000000000000000000

$1{,}5 = 1{,}1 \times 2^0$

1,5 →

0	01111111	10000000000000000000000

Estándar IEEE 754 de precisión simple

- Existencia de casos especiales:

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

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, sqrt(-4), ….)
255 (1111 1111)	0	+/-infinito (según signo)
1-254	Cualquiera	Valor normal (no especial)

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

Estándar IEEE 754 de precisión simple

- Ejemplos:

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

a) Bit de signo: 0 ⇒ número positivo
b) Exponente: $10000011_2 = 131_{10} \Rightarrow E - 127 = 131 - 127 = 4$
c) Mantisa almacenada: 11000000000000000000000
d) Mantisa real : 1,1100 $\Rightarrow 1 + 1 \times 2^{-1} + 1 \times 2^{-2} = 1{,}75$

Por tanto el valor decimal del nº es $1{,}75 \times 2^4 = 28$

Rango en estándar IEEE 754 de precisión simple

- Rango de magnitudes representables sin considerar el signo:
  - Menor número normalizado:

$$1.00000000000000000000000_2 \times 2^{1-127} = 2^{-126}$$

  - Mayor número normalizado

$$1.11111111111111111111111_2 \times 2^{254-127} = (2 - 2^{-23}) \times 2^{127}$$

Truco:

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

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

Rango en estándar IEEE 754 de precisión simple

- Rango de magnitudes representables sin considerar el signo:
  - Menor número no normalizado:

$$0.00000000000000000000001_2 \times 2^{-126} = 2^{-149}$$

  - Mayor número no normalizado

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

Truco:

$$\begin{array}{rl} & 0.11111111111111111111111_2 = X \\ + & 0.00000000000000000000001_2 = 2^{-23} \\ \hline & 1.00000000000000000000000_2 = 1 \end{array}$$

$$X = 1 - 2^{-23}$$

¿Cuántos números no normalizados distintos de cero se pueden representar?

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

Exponente	Mantisa	Valor especial
0 (0000 0000)	No cero	Número no normalizado

¿Cuántos números no normalizados distintos de cero se pueden representar?

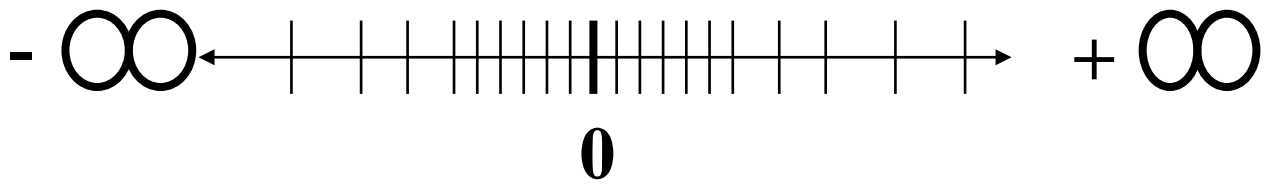
$(s) \times 0.\text{mantisa} \times 2^{-126}$

Exponente	Mantisa	Valor especial
0 (0000 0000)	No cero	Número no normalizado

- Solución:
  - 23 bits para la mantisa (distinta de cero)

$$2^{23} - 1$$

Representación discreta



- Disminuye la densidad hacia infinito

Ejercicio

- Sea `f(1,2)` = número de floats entre 1 y 2
- Sea `f(2,3)` = número de floats entre 2 y 3

- ¿Cuál es mayor `f(1,2)` o `f(2,3)`?

Ejercicio

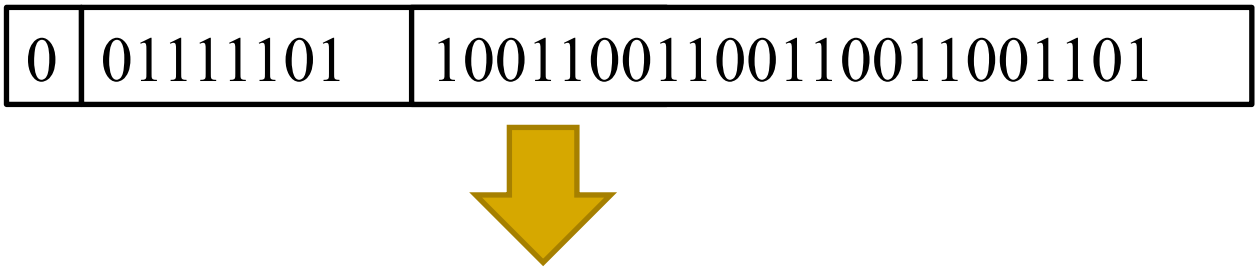
- Sea `f(1,2)` = número de floats entre 1 y 2
- Sea `f(2,3)` = número de floats entre 2 y 3

- ¿Cuál es mayor `f(1,2)` o `f(2,3)`?

- Solución:
  - $1 = 1{,}0 \times 2^0$
  - $2 = 1{,}0 \times 2^1$
  - $3 = 1{,}1 \times 2^1$
  - Entre el 1 y el 2 hay 2^{23} números
  - Entre el 2 y el 3 hay 2^{22} números

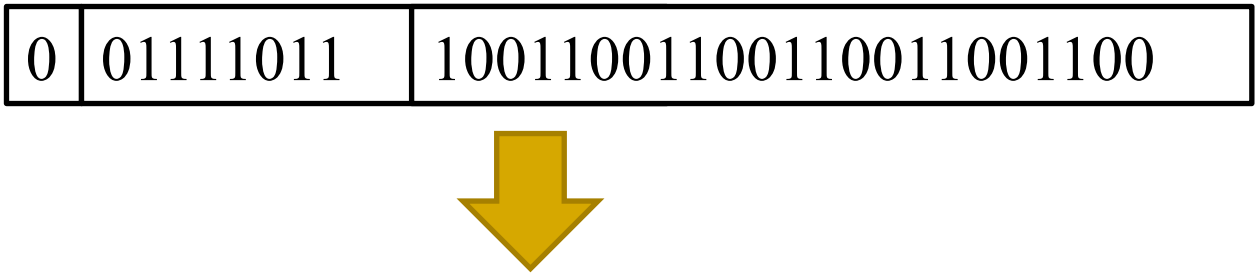
Curiosidad

0,4 →



3.9999998 e-1

0,1 →



9.9999994 e-2

Ejemplo

- Indique el valor binario en IEEE754 y el valor decimal del siguiente número hexadecimal representado en IEEE754 de 32 bits: 3FE00000

Solución

- El valor binario:

  3 F E 0 0 0 0 0

  0011 1111 1110 0000 0000 0000 0000 0000

- El valor decimal:

  0011 1111 1110 0000 0000 0000 0000 0000

  - Signo: 0
  - Exponente: 01111111 $\Rightarrow$ 127-127 = 0
  - Mantisa: **1.**11000000000000000000000 $\Rightarrow$ 1+0,5+0,25 = 1,75

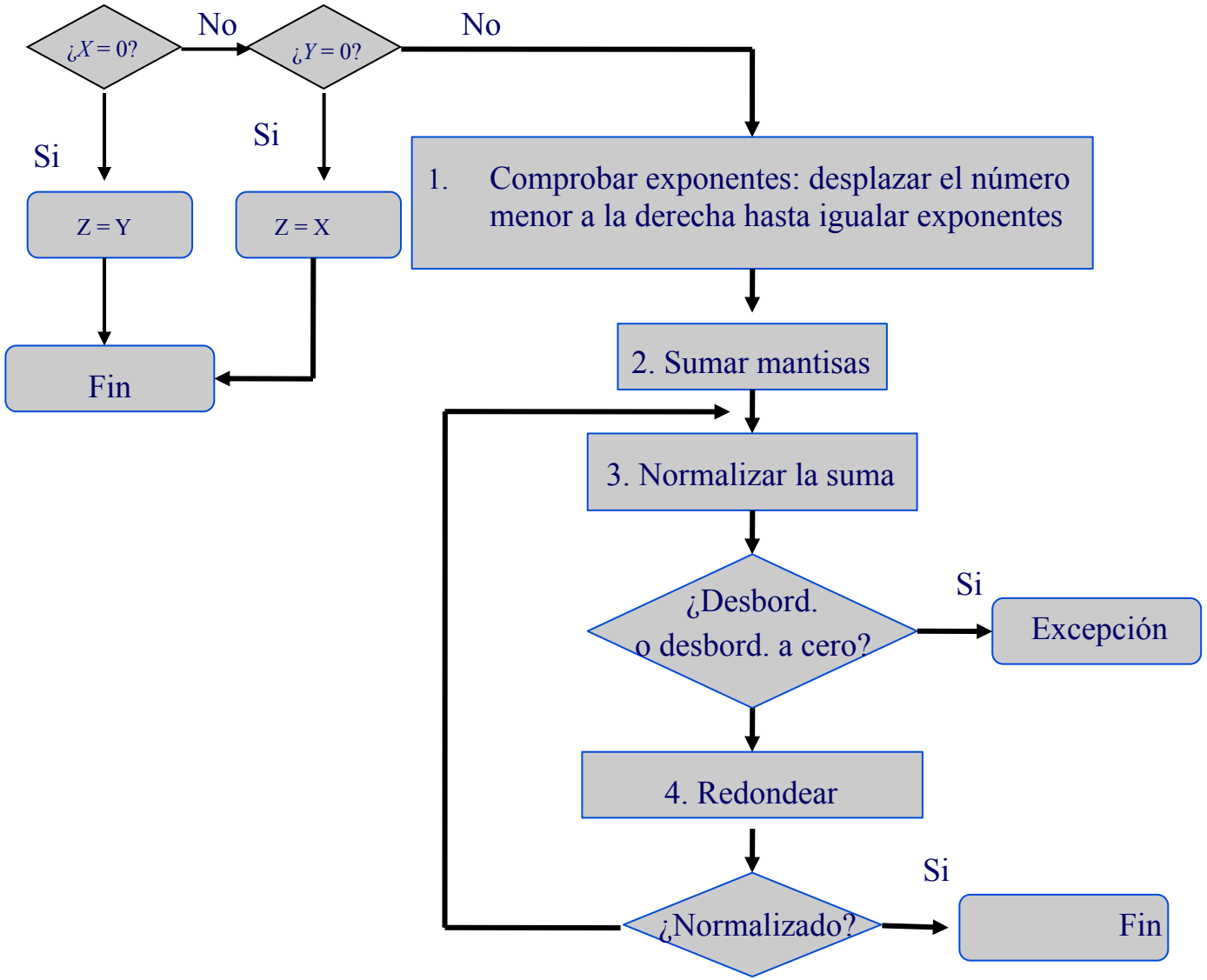
  Por tanto, el valor es: $+1 \times 1{,}75 \times 2^0 = 1{,}75$

Operaciones con el estándar IEEE 754

- Sumas y restas:
  1) Comprobar valores cero.
  2) Ajuste de mantisas (ajuste de exponentes).
  3) Sumar o restar las mantisas.
  4) Normalizar el resultado.

Suma y resta

$Z = X \times Y$



Operaciones con el estándar IEEE 754

- Ejemplos de sumas:

$N_1 = 0 \;\; 10000001 \;\; 11100000000000000000000 = +7.5$

$N_2 = 0 \;\; 01111111 \;\; 10000000000000000000000 = +1.5$

Operaciones con el estándar IEEE 754

- Se restan los exponentes:

  E_1 = 10000001 ⇒ exponente real = 129 -127 = 2

  E_2 = 01111111 - ⇒ exponente real = 127

  00000010 = $2_{(10}$

- Luego se desplaza la mantisa del número de exponente menor ($1.M_2$) dos lugares a la izquierda (E_1-E_2) para igualar los exponentes, incluyendo el bit implícito:

- $1.M_2$=1.10000000000000000000000

  ⇒ 0.01100000000000000000000

Operaciones con el estándar IEEE 754

- Se suman las mantisas $1.M_1$, y $1.M_2$ desplazada:

  M_1 = 1.11100000000000000000000

  M_2 = 0.01100000000000000000000 +

  M_S = 10.01000000000000000000000

- El resultado es 10.01×2^2
- Se normaliza el resultado ajustando el exponente: $1.001 \text{ x} \times 2^3 = 9_{(10}$

- El resultado final es:

  Resultado = 0 10000010 00100000000000000000000

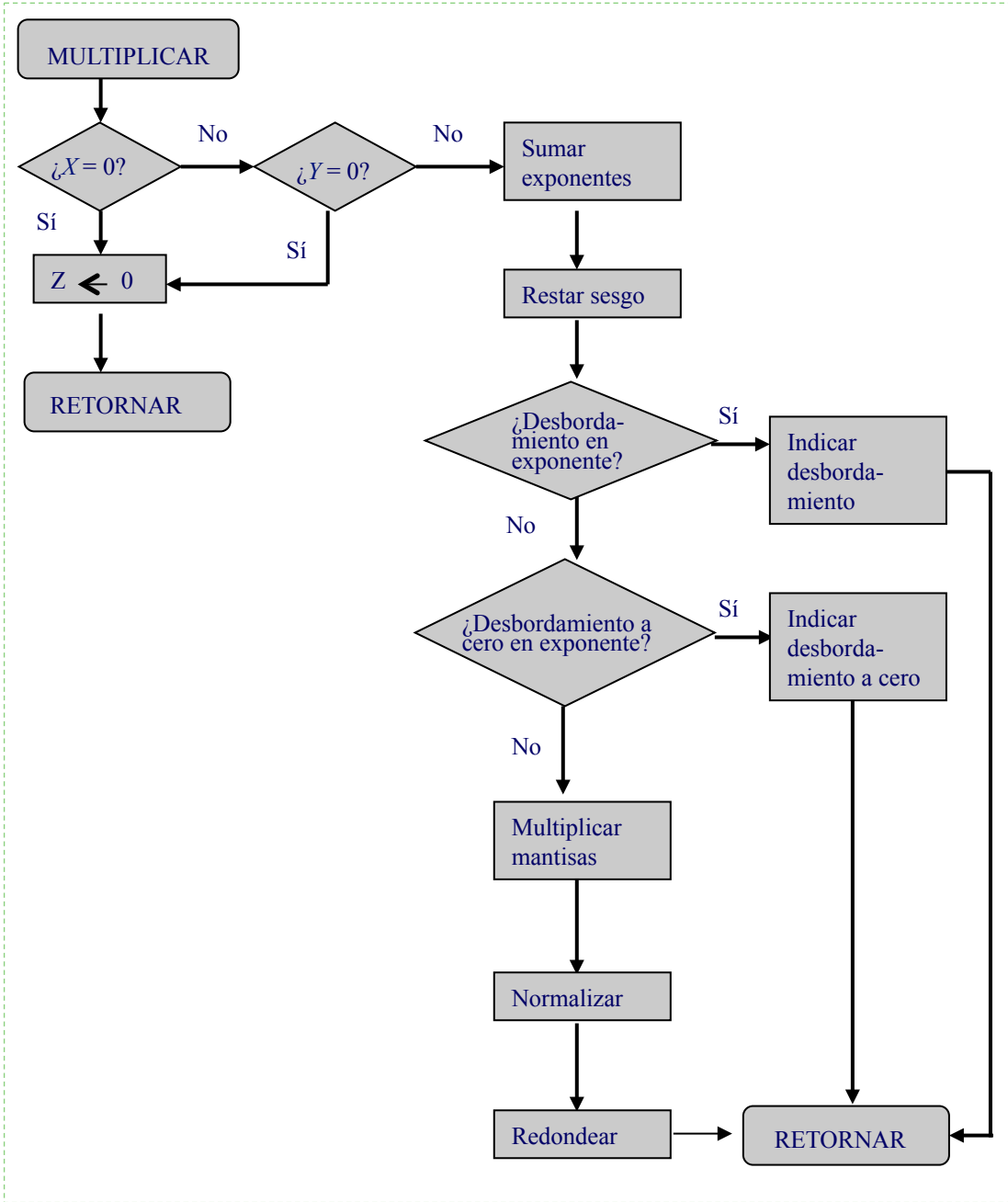
  - $0 \Rightarrow +$
  - $e = 3 \Rightarrow E = 127 + 3 =$ 10000010
  - Mantisa real = 1.00100000000000000000000, la mantisa a almacenar es 00100000000000000000000

Operaciones con el estándar IEEE 754

- Multiplicación y división:
  1. Comprobar valores cero.
  2. Sumar (restar) exponentes.
  3. Multiplicar (dividir) mantisas (teniendo en cuenta el signo).
  4. Normalizar.
  5. Redondear.

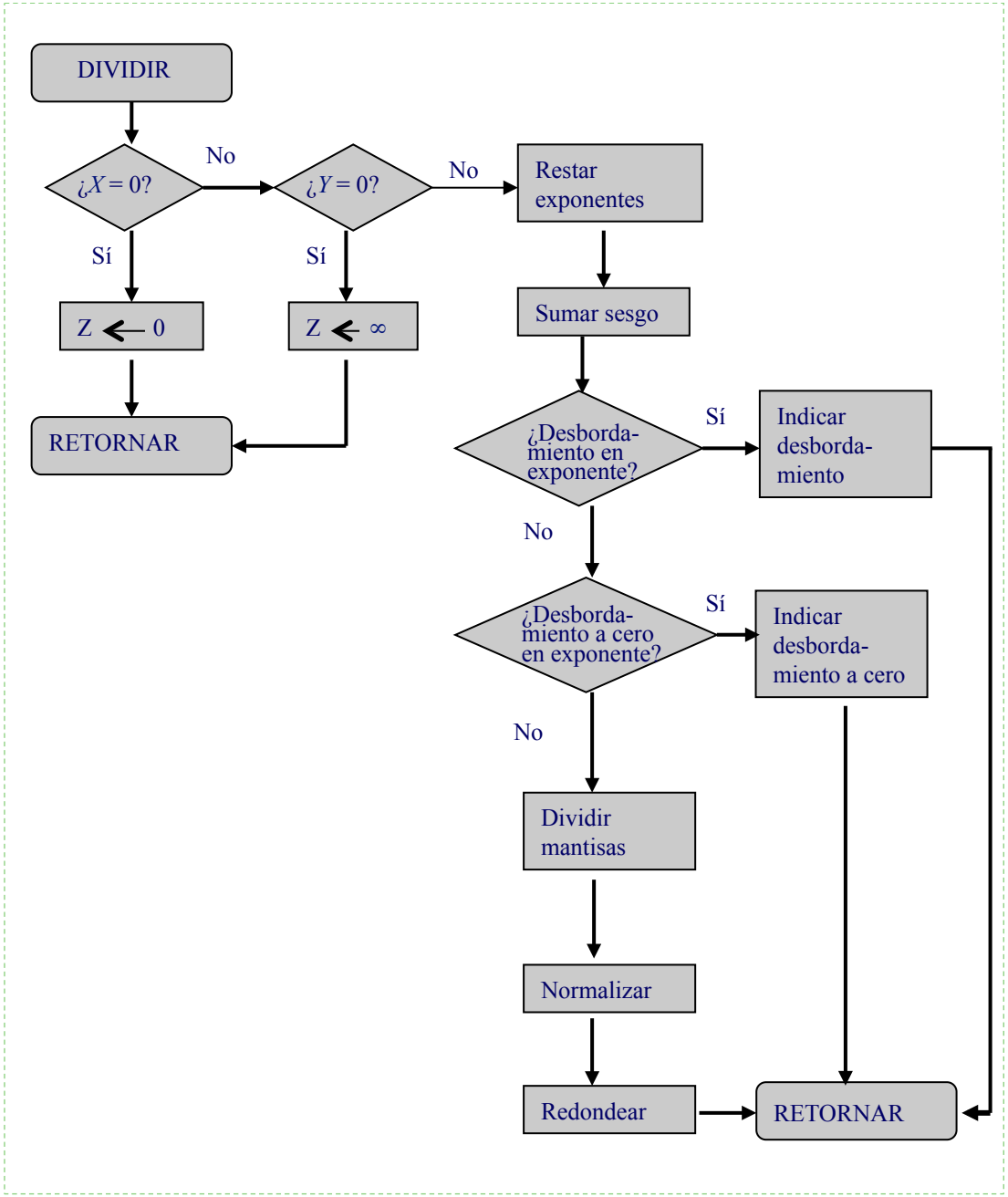
Multiplicación en coma flotante

$Z = X \times Y$



División en coma flotante

$Z = X / Y$



Ejemplo

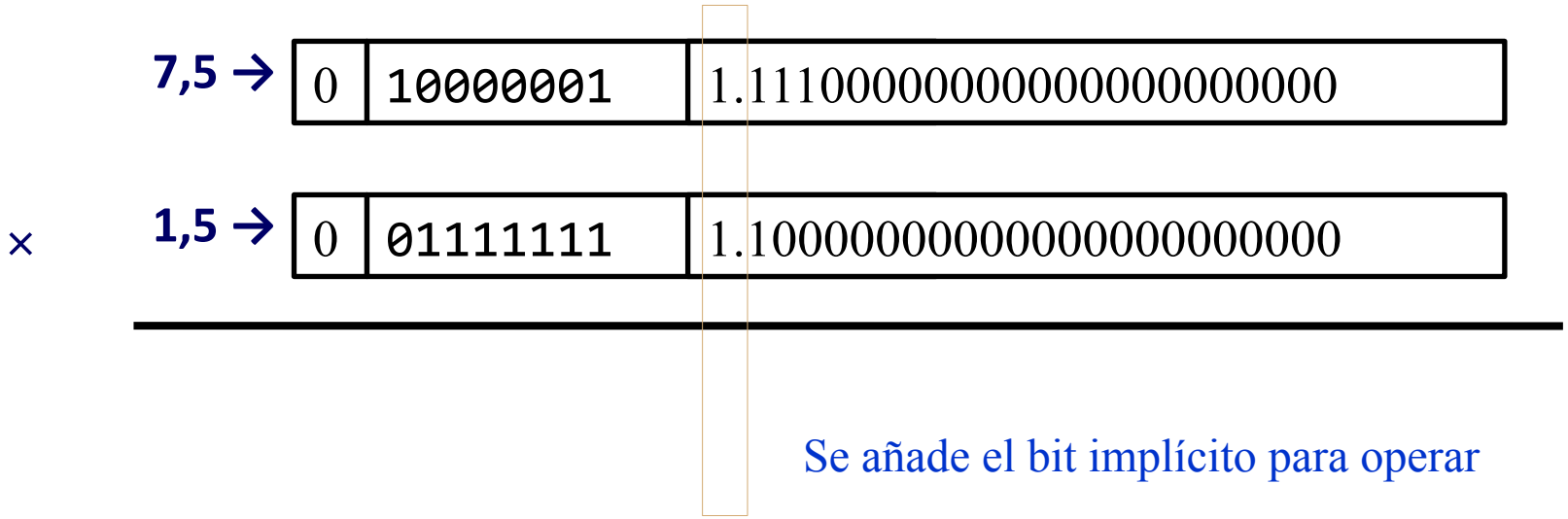
- Usando el formato IEEE 754, multiplicar 7,5 y 1,5 paso a paso

Solución (1)

$$
\begin{aligned}
7{,}5 \times 1{,}5 &= (1{,}111_2 \times 2^2) \times (1{,}1_2 \times 2^0) \\
&= (1{,}111_2 \times 1{,}1_2) \times 2^{(2+0)} \\
&= (10{,}1101_2) \times 2^2 \\
&= (1{,}01101_2) \times 2^3 \\
&= 11{,}25
\end{aligned}
$$

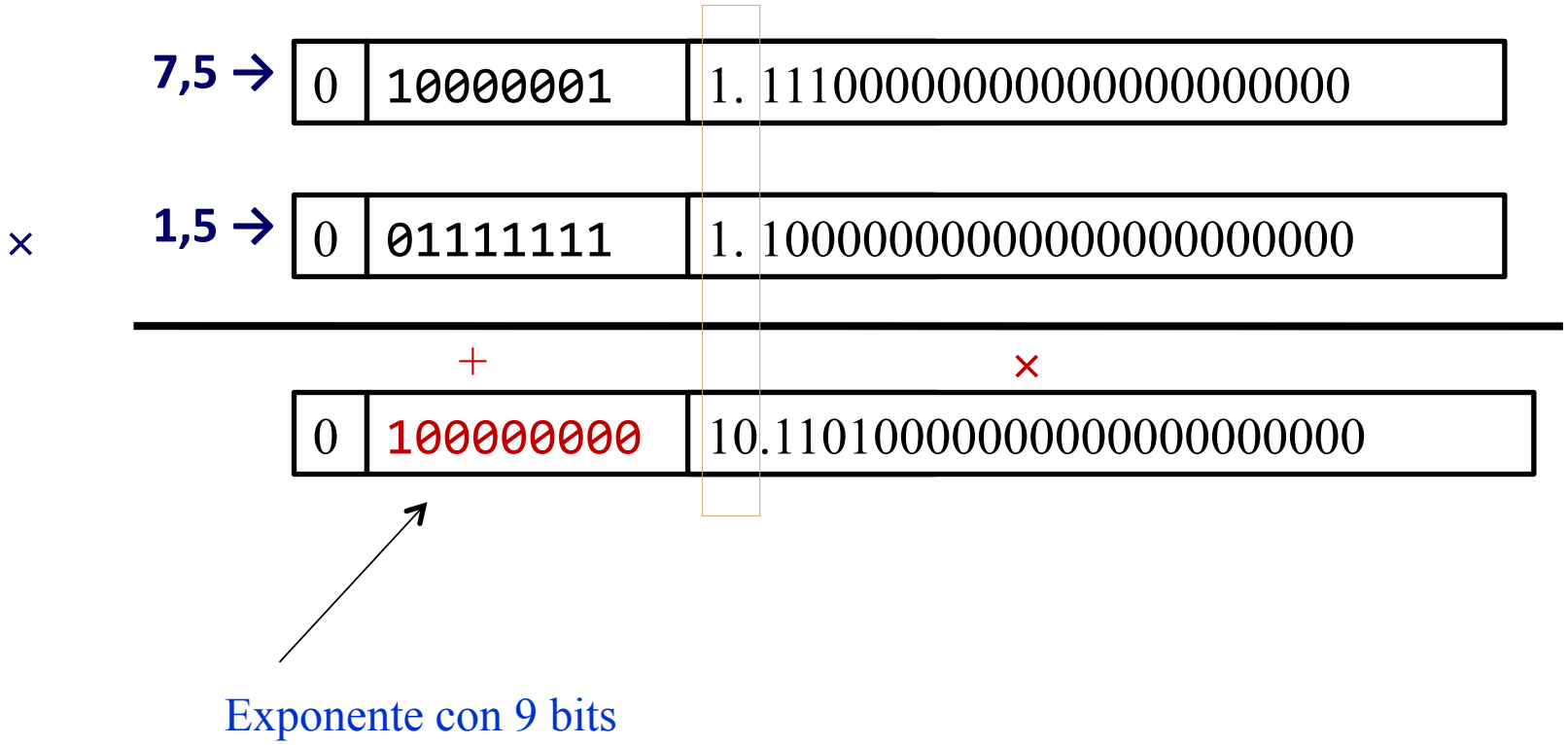
Solución (2)

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



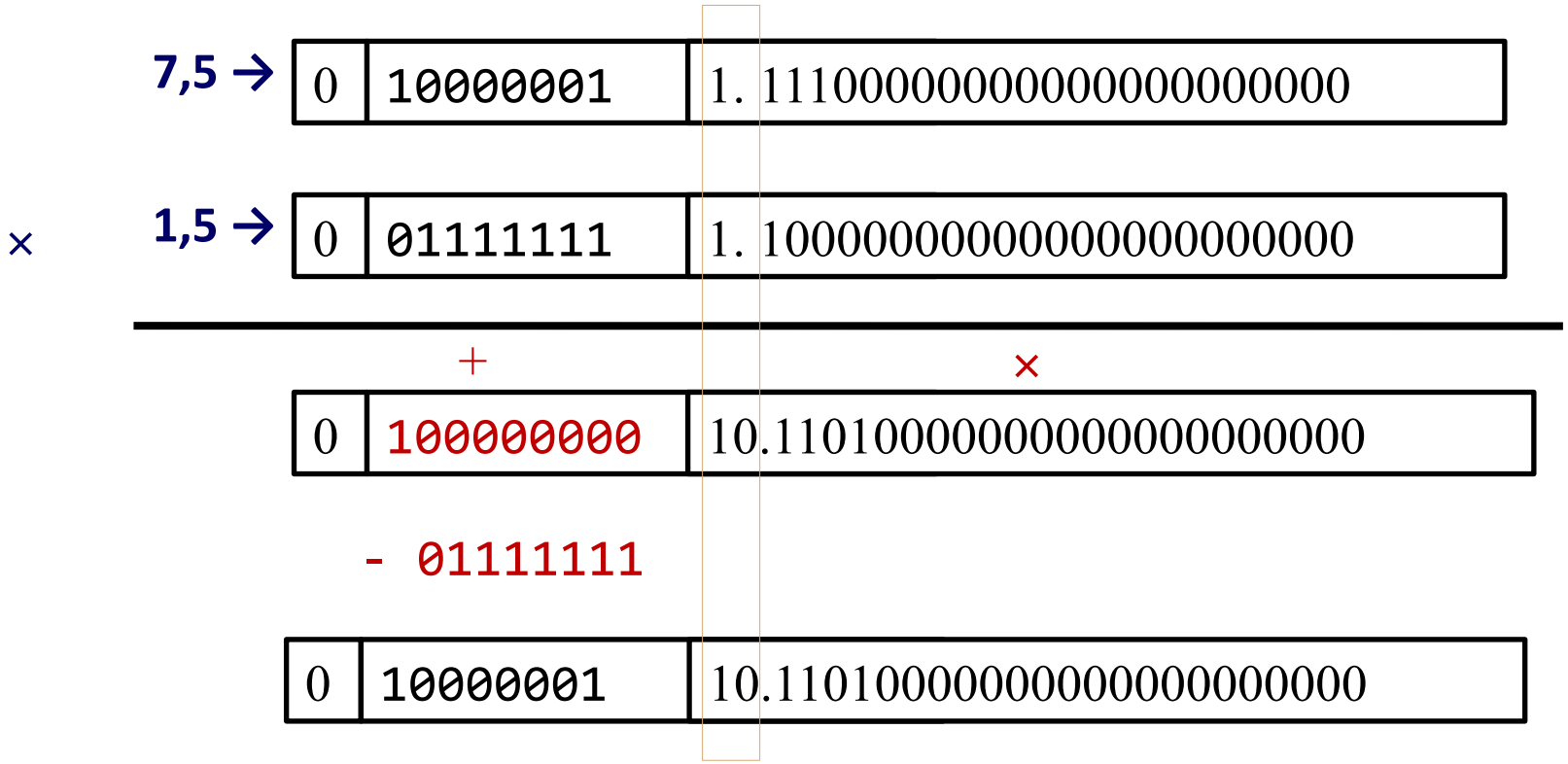
Solución (3)

- Se suman los exponentes y se multiplican las mantisas



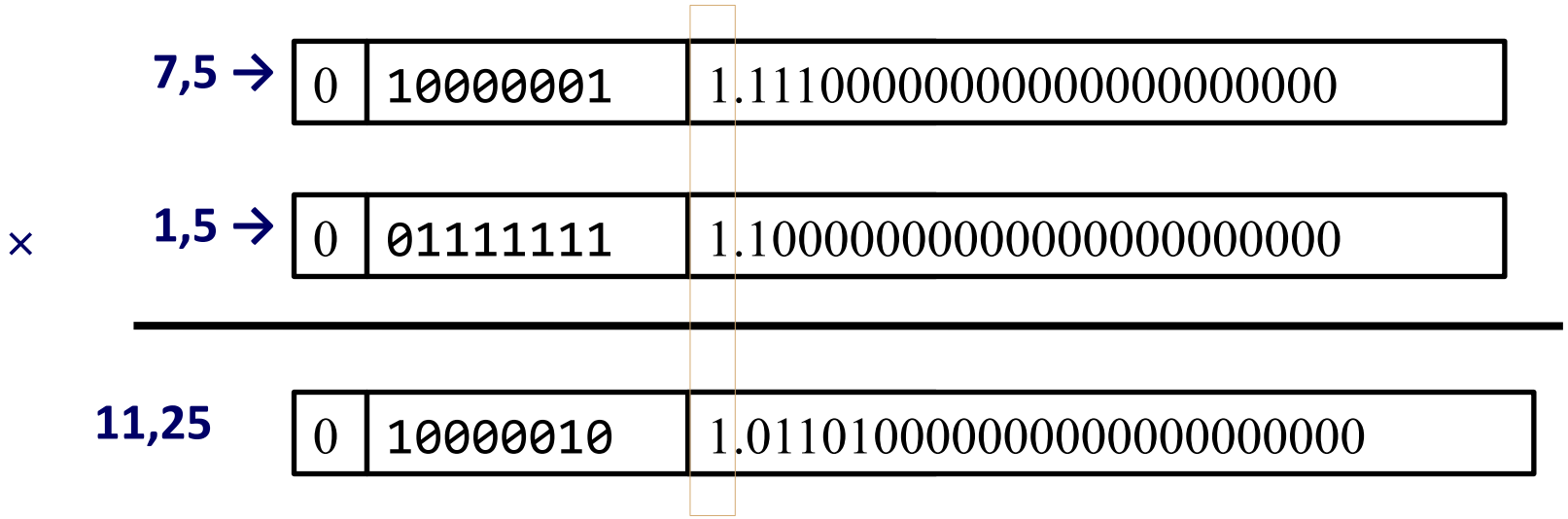
Solución (4)

- Se resta el sesgo al exponente (127)



Solución (5)

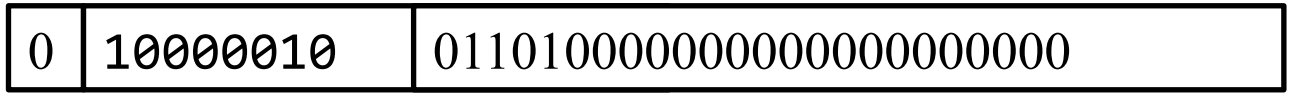
- Normalizar el resultado



Solución (6)

- Se elimina el bit implícito

11,25



Redondeo

- El hardware de coma flotante añade dos bits extra (bits de guarda) de precisión antes de operar
- Después de operar hay que eliminarlos: redondeando
- El redondeo también ocurre cuando se convierte:
  - Un valor de doble a simple precisión
  - Un valor en coma flotante a entero

Dígitos de guarda. Justificación

- Sumar 2,56 x 10^0 y 2.34x 10^2 suponiendo que solo tenemos tres dígitos decimales significativos:

$$\begin{array}{r} 2{,}56 \times 10^0 \\ +\ 2{,}34 \times 10^2 \\ \hline \end{array}$$

Se ajusta el exponente

$$\begin{array}{r} 0{,}02 \times 10^2 \\ +\ 2{,}34 \times 10^2 \\ \hline 2{,}36 \times 10^2 \end{array}$$

Dígitos de guarda. Justificación

- Sumar 2,56 x 10^0 y 2.34x 10^2 suponiendo que solo tenemos tres dígitos decimales significativos pero se utilizan dos dígitos de guarda

$$\begin{array}{l} 2{,}56 \times 10^0 \\ +\ 2{,}34 \times 10^2 \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{l} 2{,}5600 \times 10^0 \\ +\ 2{,}3400 \times 10^2 \\ \hline \end{array}$$

Se añaden los dígitos de guarda

Dígitos de guarda. Justificación

- Sumar 2,56 x 10^0 y 2.34x 10^2 suponiendo que solo tenemos tres dígitos decimales significativos pero se utilizan dos dígitos de guarda

$$\begin{array}{r} 2{,}5600 \times 10^0 \\ +\ 2{,}3400 \times 10^2 \\ \hline \end{array}$$

Se ajusta el exponente

$$\begin{array}{r} 0{,}0256 \times 10^2 \\ +\ 2{,}3400 \times 10^2 \\ \hline 2{,}3656 \times 10^2 \end{array}$$

Se redondea

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

Modos de rendondeo en IEEE 754

- 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
- Redondeo al más cercano
  - 2.4 → 2,
  - 2.6 → 3,
  - 2.5 → 2,
  - 3.5 → 4

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:
  - 11111101101011011001101010 1 (27 bits)
- $11111101101011011001101010 1 \times 2^0$
- Se normaliza
  - $1, 11111011010110110011010101 \times 2^{26}$
  - S = 0 (positivo)
  - e = 26 → E = 26 + 127 = 153
  - M = 11111011010110110011010 (se pierden los 3 últimos bits)
- El número realmente almacenado es
  - $1, 11111011010110110011010 \times 2^{26} =$
  - $11111101101011011001101010 \times 2^3 = 133000400$

Conversión float → int → float

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

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