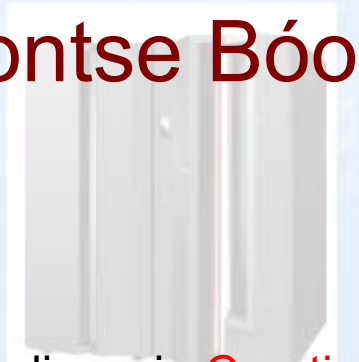
Tipo de datos

Montse Bóo Cepeda

Estructura del curso

1. Evolución y caracterización de los computadores.
2. Arquitectura del MIPS: Introducción.
3. Tipo de datos.
4. El repertorio de instrucciones.
5. Aritmética del computador.
6. El camino de datos.
7. Sección de control.
8. El camino de datos multiciclo.
9. Sección de control multiciclo.
10. Entrada/Salida (I/O).

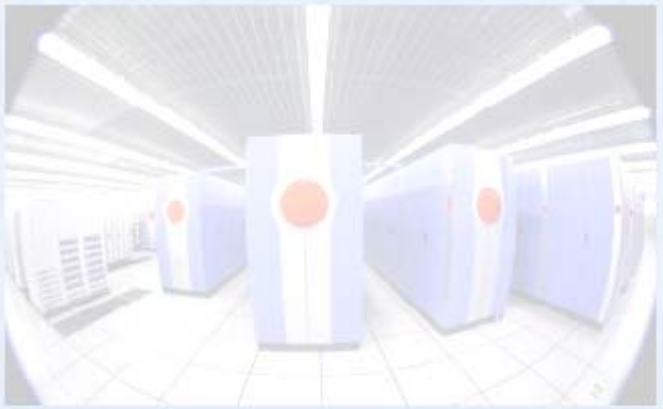
Esquema de contenidos

1. Clasificación de la información
2. Punto fijo
   - Representación de los datos.
3. Punto flotante
   - Representación de los datos.

Representación de los datos

- Factores a tener en cuenta:
  - Tipo de números a representar (entero, real,..)
  - El rango posible de valores.
  - Precisión
  - Coste hardware
- Formato:
  - Punto fijo:
    - Rango limitado de valores
    - Requerimientos hardware simples
  - Punto flotante:
    - Mayor rango de valores
    - Procesamiento más costoso

Punto fijo



Representación de enteros

- Representación de números en cualquier base:
  - Sistema de numeración posicional: cada número viene definido por una cadena de dígitos, donde cada uno de ellos está afectado por un factor de escala.

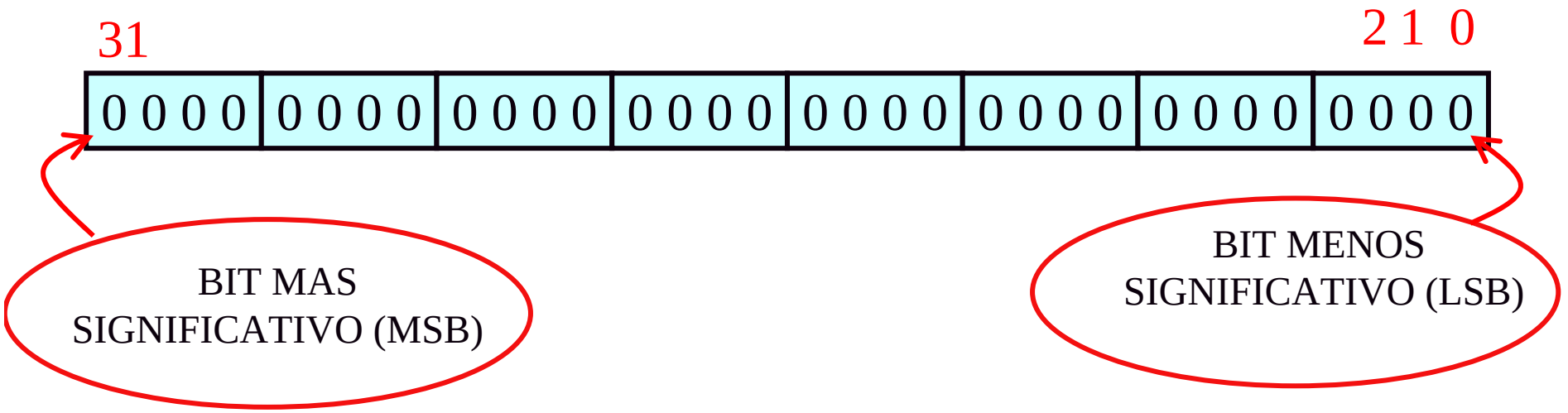
Valor del i-ésimo dígito d:

$d \times Base^i$

- Ejemplos:
  - $(176)_{diez} = 1 \times 10^2 + 7 \times 10^1 + 6 \times 10^0 = 1 \times 100 + 7 \times 10 + 6$
  - $(1011)_{dos} = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = (11)_{diez}$

Representación de enteros : Sistema binario

- Base: 2
- Dígitos: 0 y 1 (denominados bits)
- Rango de valores representable con n bits: $[0, 2^n-1]$
- Ejemplo: Representación palabra 32 bits:



Representación de enteros : Sistema hexadecimal

- Base: 16
- Dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

HEXADECIMAL	BINARIO
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

HEXADECIMAL	BINARIO
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Números con signo: Signo magnitud

Signo y magnitud

- MSB representa el signo del número, el resto de los bits representan el valor absoluto en binario natural
- Ejemplo (palabras de 8 bits):

0 1 1 1	1 1 1 1	$= (127)_{diez}$
1 1 1 1	1 1 1 1	$= (-127)_{diez}$

Rango $[-(2^{n-1}-1),(2^{n-1}-1)]$

- Hay dos formas de representar el 0:

0 0 0 0	0 0 0 0	$= (0)_{diez}$
1 0 0 0	0 0 0 0	$= (-0)_{diez}$

- En algunas de las primeras computadoras binarias (IBM 7090)

Números con signo: Complemento a uno

Complemento a uno

- Los positivos se representan por su valor absoluto en binario natural, los negativos se representan por el complemento a 1
- Ejemplo (palabras de 8 bits):

0 0 0 0	0 0 1 0	$= (2)_{diez}$

Rango $[-(2^{n-1}-1),(2^{n-1}-1)]$

1 1 1 1	1 1 0 1	$= (-2)_{diez}$

- Hay dos formas de representar el 0:

0 0 0 0	0 0 0 0	$= (0)_{diez}$

1 1 1 1	1 1 1 1	$= (-0)_{diez}$

- Común en las computadoras antiguas (PDP-1 y UNIVAC 1100/2200)

Números con signo: Complemento a dos

Complemento a dos

- Los positivos se representan por su valor absoluto en binario natural y los negativos por su complemento a dos.
- Ejemplo (palabras de 8 bits):

0 0 0 0	0 0 1 0	$= (2)_{diez}$

Rango $[-2^{n-1},(2^{n-1}-1)]$

1 1 1 1	1 1 10	$= (-2)_{diez}$

- El 0 tiene una representación única

0 0 0 0	0 0 0 0	$= (0)_{diez}$

- Ventajas:
  - Comodidad para operar: la suma y resta no necesita realizar ajustes para obtener el resultado de la operación
  - el cero tiene una única representación

Números con signo: Complemento a dos

- Conversión de complemento a dos a decimal:
- Basta con tener en cuenta el papel del bit de signo. Para 32 bits:
  - $d_{31} \times \mathbf{-2^{31}} + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$
- Ejemplo

  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{dos}$

  $= 1x\text{-}2^{31} + 1x2^{30} + 1x2^{29} + \ldots + 1x2^2 + 0x2^1 + 0x2^0$

  $= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0$

  $= -2{,}147{,}483{,}648_{diez} + 2{,}147{,}483{,}644_{diez}$

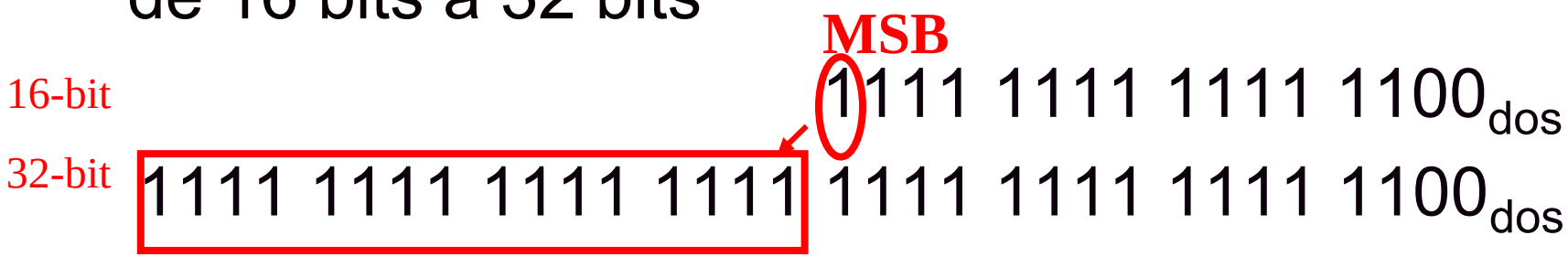
  $= -4_{diez}$

Números con signo: Complemento a dos

$0000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0000_{dos} =$	0_{diez}
$0000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0001_{dos} =$	1_{diez}
$0000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0010_{dos} =$	2_{diez}
. . .	
$0111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1101_{dos} =$	$2{,}147{,}483{,}645_{diez}$
$0111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1110_{dos} =$	$2{,}147{,}483{,}646_{diez}$
$0111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1111_{dos} =$	$2{,}147{,}483{,}647_{diez}$
$1000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0000_{dos} =$	$-2{,}147{,}483{,}648_{diez}$
$1000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0001_{dos} =$	$-2{,}147{,}483{,}647_{diez}$
$1000\ ...\ 0000\ \ 0000\ \ 0000\ \ 0010_{dos} =$	$-2{,}147{,}483{,}646_{diez}$
. . .	
$1111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1101_{dos} =$	-3_{diez}
$1111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1110_{dos} =$	-2_{diez}
$1111\ ...\ 1111\ \ 1111\ \ 1111\ \ 1111_{dos} =$	-1_{diez}

Números con signo: Complemento a dos

- Negación de un número binario en complemento a dos: invertir todos los bits y sumar una unidad (hacer su C2).
- Representación de un número de N bits con un número mayor de bits: EXTENSION DE SIGNO. Ejemplo, extensión del número -4_{diez} de 16 bits a 32 bits



Representación de enteros

BINARIO	SIN SIGNO	C-1	C-2
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	-7	-8
1001	9	-6	-7
1010	10	-5	-6
1011	11	-4	-5
1100	12	-3	-4
1101	13	-2	-3
1110	14	-1	-2
1111	15	-0	-1

Representación de reales en punto fijo

- Se implementa aplicando el sistema posicional también a las potencias negativas de la base

Valor del i-ésimo dígito d:

$d \times Base^{i}$

- Ejemplos:
  - $(12.5)_{diez} = 1 \times 10^{1} + 2 \times 10^{0} + 5 \times 10^{-1} = 10 + 2 + 0.5$
  - $(101.11)_{dos} = 1 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} + 1 \times 2^{-1} + 1 \times 2^{-2} = 4 + 0 + 1 + 0.5 + 0.25 = (5.75)_{diez}$

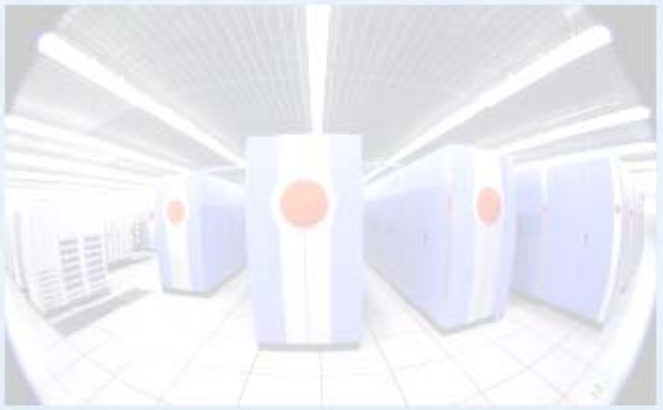
Codificación de texto

- ASCII
  - Antiguo, 7 bits → 128 caracteres
  - Extendido por IBM a 8 bits → 256 caracteres

ASCII

(nul)	0	(dle)	16	(sp)	32	0	48	@	64	P	80	`	96	p	112
(soh)	1	(dc1)	17	!	33	1	49	A	65	Q	81	a	97	q	113
(stx)	2	(dc2)	18	"	34	2	50	B	66	R	82	b	98	r	114
(etx)	3	(dc3)	19	#	35	3	51	C	67	S	83	c	99	s	115
(eot)	4	(dc4)	20	$	36	4	52	D	68	T	84	d	100	t	116
(enq)	5	(nak)	21	%	37	5	53	E	69	U	85	e	101	u	117
(ack)	6	(syn)	22	&	38	6	54	F	70	V	86	f	102	v	118
(bel)	7	(etb)	23	'	39	7	55	G	71	W	87	g	103	w	119
(bs)	8	(can)	24	(	40	8	56	H	72	X	88	h	104	x	120
(ht)	9	(em)	25	)	41	9	57	I	73	Y	89	i	105	y	121
(nl)	10	(sub)	26	*	42	:	58	J	74	Z	90	j	106	z	122
(vt)	11	(esc)	27	+	43	;	59	K	75	[	91	k	107	{	123
(np)	12	(fs)	28	,	44	<	60	L	76	\	92	l	108	|	124
(cr)	13	(gs)	29	-	45	=	61	M	77	]	93	m	109	}	125
(so)	14	(rs)	30	.	46	>	62	N	78	^	94	n	110	~	126
(si)	15	(us)	31	/	47	?	63	O	79	_	95	o	111	(del)	127

Punto flotante (PF)



Rango de valores

- Qué se puede representar con n bits:
  - Enteros sin signo: 0 a $2^n - 1$
  - Enteros con signo: $-2^{(n-1)}$ a $2^{(n-1)} - 1$
- Y los otros números?
  - Números demasiado grandes?

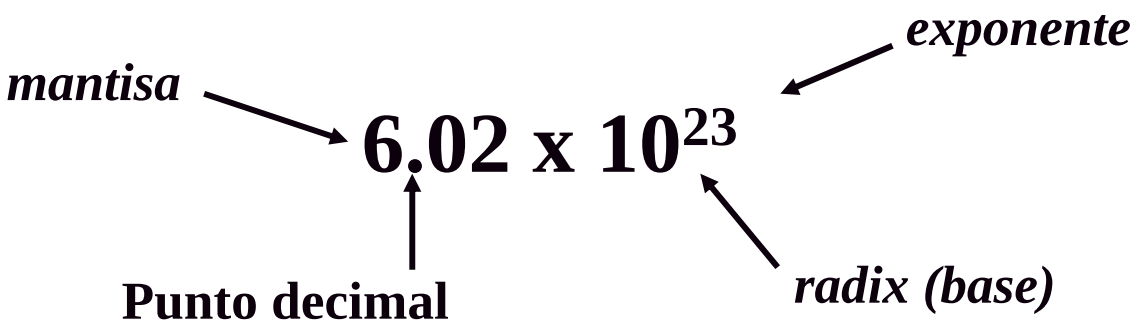
    $3{,}155{,}760{,}000_{10}$ ($3.15576_{10} \times 10^9$)
  - Números demasiado pequeños ?

    0.00000001_{10} ($1.0_{10} \times 10^{-8}$)
  - Si tenemos n bits, cuántos destinamos a la parte entera y a la parte decimal?

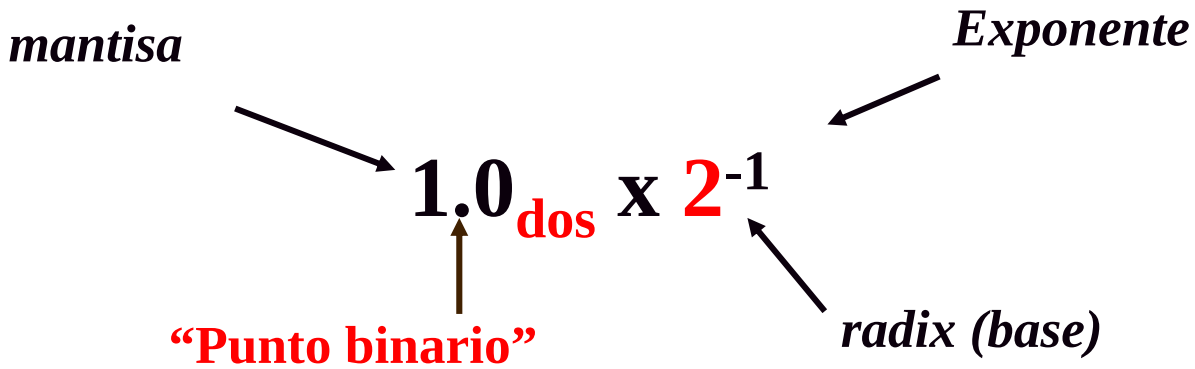
    xxxxxxxxxxxxxxxxx.yyyyyyyyyyyyyy?

Representación en punto flotante

Notación científica (decimal)

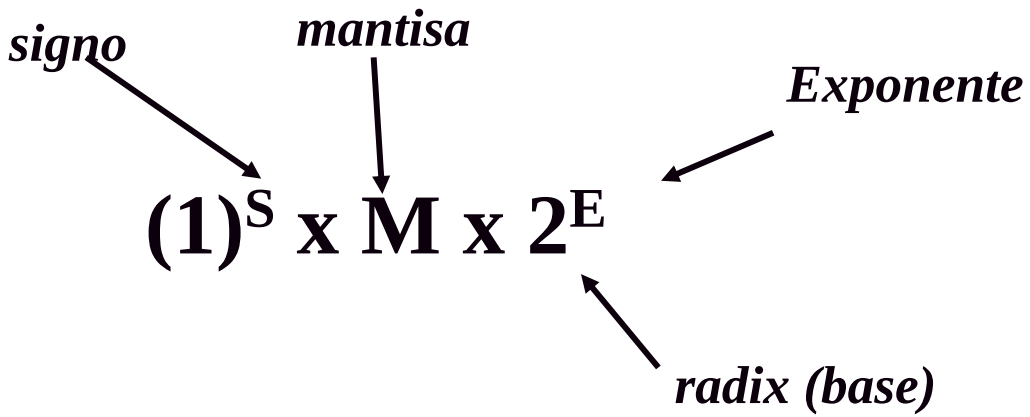


Forma binaria



Representación en punto flotante

Forma binaria



Representación basada en:

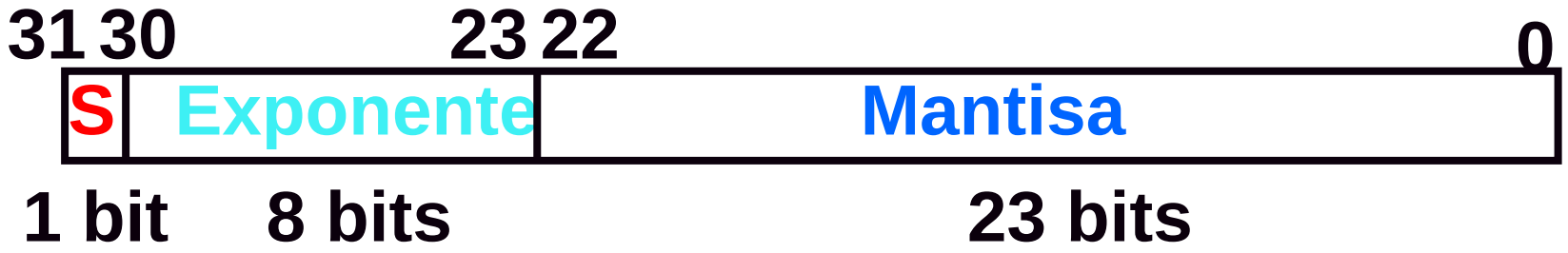
- SIGNO
- MANTISA
- EXPONENTE

Representación en punto flotante

Forma binaria normalizada:

$$S\ 1.xxxxxxxxxx_{dos} * 2^{yyyy_{dos}}$$

Con 32 bits:



- Signo = S (0 positivo, 1 negativo)
- Mantisa en rango [1,2) = 1.xxxxxxxxxx
- Exponente: Positivo si el valor absoluto del número es mayor o igual que 1

Representación en punto flotante

Ejemplos exponente positivo (el valor absoluto del número es mayor o igual que 1)

- $(4)_{diez} = 1.0 \times 2^{2}$
- $(-8)_{diez} = -1.0 \times 2^{3}$

Ejemplos exponente negativo (el valor absoluto del número es menor que 1)

- $(0.5)_{diez} = 1.0 \times 2^{-1}$
- $(-0.25)_{diez} = -1.0 \times 2^{-2}$

Rango de valores que se pueden representar:

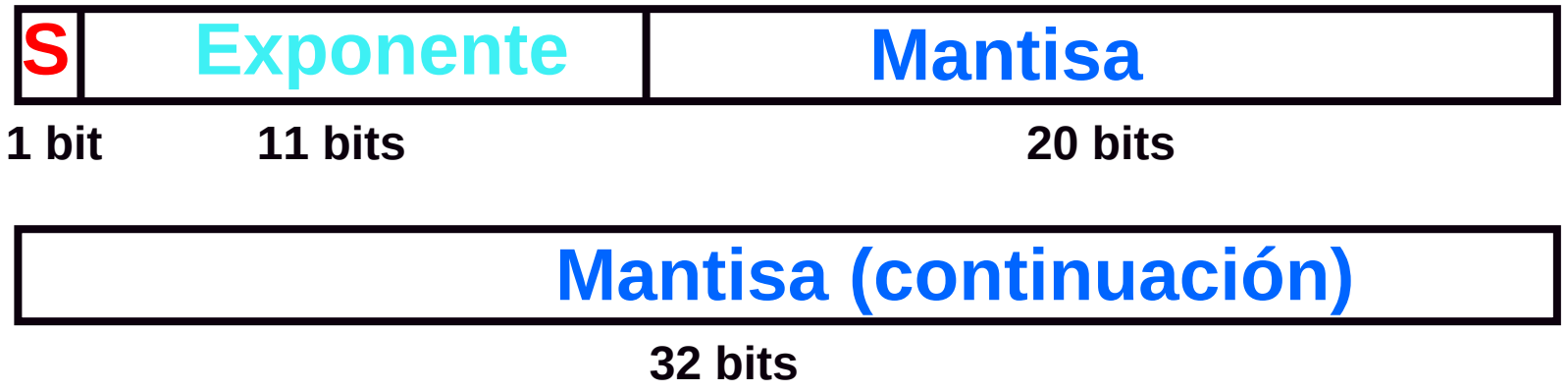
de 2.0×10^{-38} a 2.0×10^{38}

Representación en punto flotante

Overflow: El número es demasiado grande para almacenar el exponente positivo en 8 bits

Underflow: La fracción es demasiado pequeña para almacenar el exponente negativo en 8 bits.

Para reducir las posibilidades de overflow/underflow: Uso de 64 bits: (valores entre 2.0×10^{-308} a 2.0×10^{308})



Representación en punto flotante

Forma binaria normalizada:

$$S\ 1.xxxxxxxxxx_{dos} * 2^{yyyy_{dos}}$$

Precisión simple: con 32 bits

Precisión doble: con 64 bits

El estándar IEEE 754

Define formatos estándar para números en punto flotante

Hay 2 versiones

- Simple precisión → 32 bits
- Doble precisión → 64 bits

Se utiliza en la mayor parte de los computadores modernos

El estándar IEEE 754

Precisión simple:

- 32 bits
- 1 bit para el signo
  - Si es positivo el bit de signo es ‘0’
  - Si es negativo el bit de signo es ‘1’
- 8 bits para el exponente
- 23 bits para la mantisa

El estándar IEEE 754

Precisión simple: LA MANTISA

- Siempre está en el intervalo [1, 2)
- Por tanto siempre es de la forma **1.xxxx…**
- Así pues, el **1** se sobreentiende y no es necesario almacenarlo
- La mantisa tiene en realidad **24** bits

1.← 23 bits de mantisa →

El estándar IEEE 754

Precisión simple: El exponente

- Es un número de 8 bits
- Toma valores entre **-126 y 127**
- NO se almacena en Complemento a 2
- En lugar de ello se codifica en exceso a **127**
  - 4 → 131 -2 → 125 -100 → 27
- Hay sólo 254 exponentes posibles…
- … y 2 exponentes reservados
  - Exp. -127 → 0
  - Exp. 128 → 255

$$(-1)^{S} * (1 + \text{Mantisa}) * 2^{(\text{Exponente} - \text{Exceso})}$$

El estándar IEEE 754

Precisión simple: Valores especiales

Exponente	Mantisa	Valor
0	0	CERO
0	!=0	No Normalizado
255	0	Infinito
255	!=0	NaN (Not a Number)

El estándar IEEE 754

Precisión simple: Números no normalizados

- Caso especial
- Para números muy próximos a 0
- Ya no se asume que la mantisa sea 1.xxx
- Exponente = 0
- Mantisa ≠0

Ejemplo:

0 00000000 00011....
Tiene el valor: 0.00011... x 2^{-126}

$$(-1)^S * (\text{Mantisa}) * 2^{(1 - \text{Exceso})}$$

El estándar IEEE 754

Doble precisión:

- 64 bits
- 1 bit para el signo
- 11 bits para el exponente
- 52 bits para la mantisa

El estándar IEEE 754

Doble precisión: El exponente

- Es un número de 11 bits
- Toma valores entre **-1022 y 1023**
- NO se almacena en Complemento a 2
- En lugar de ello se codifica en exceso a **1023**

  4 → 1027 -2 → 1021 -1000 → 23
- Hay sólo 2046 exponentes posibles…
- … y 2 exponentes reservados
  - Exp. -1023 → 0
  - Exp. 1024 → 2047

$$(-1)^{S} * (1 + \text{Mantisa}) * 2^{(\text{Exponente} - \text{Exceso})}$$

Algunos ejemplos

`0 00000000000 0000000000000000000000000000000000000000000000000000`
0

`0 01111111111 0000000000000000000000000000000000000000000000000000`
1

`0 10000000000 1000000000000000000000000000000000000000000000000000`
3

`0 11111111111 0000000000000000000000000000000000000000000000000000`
+infinito

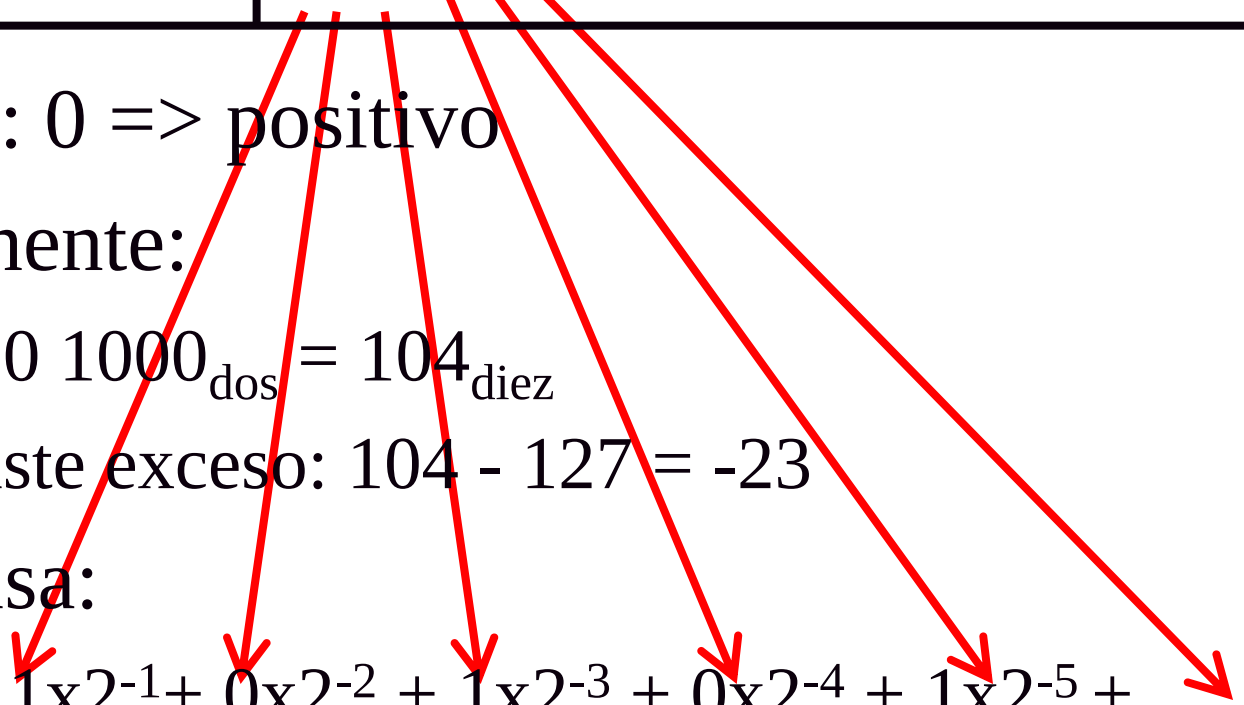
`1 11111111111 0000000000000000000000000000000000000000000000000000`
-infinito

`0 11111111111 0000011110000000000000000000000000000000000000000000`
NaN

`0 00000000000 0001011100000010101011100100110100011111101101011110`
No normalizado

Conversión de binario PF a decimal

0	0110 1000	101 0101 0100 0011 0100 0010



- Signo: 0 => positivo
- Exponente:
  - $0110\ 1000_{dos} = 104_{diez}$
  - Ajuste exceso: 104 - 127 = -23
- Mantisa:
  - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \ldots$

    $=1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$

    $= 1.0 + 0.666115$
- Representa: $1.666115*2^{-23} \sim 1.986*10^{-7}$

Conversión de decimal a binario PF

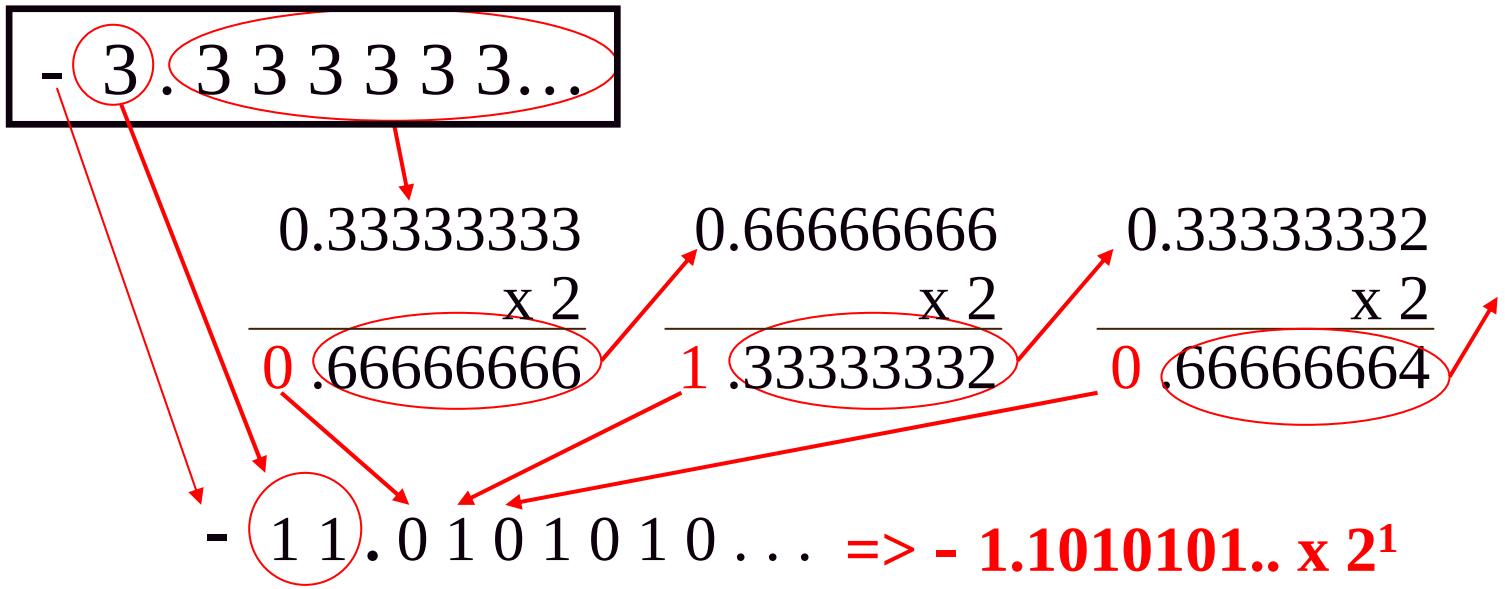
- **Caso sencillo:** El denominador es una potencia de 2 (2, 4, 8, 16, etc)
- **Ejemplo: Representación binaria PF de -0.75**
  - $(-0.75)_{diez} = (-3/4)_{diez} = (-3/ 2^2)_{diez}$
  - $-11_{dos}/100_{dos} = -0.11_{dos}$
  - Normalizado es = $-1.1_{dos} \times 2^{-1}$
  - $(-1)^S \times (1 + \text{Mantisa}) \times 2^{(\text{Exponente}-127)}$
  - $(-1)^1 \times (1 + .100\ 0000 \ldots 0000) \times 2^{(126-127)}$

1	0111 1110	100 0000 0000 0000 0000 0000

Conversión de decimal a binario PF

- **Caso complejo:** El denominador no es una potencia de 2 (2, 4, 8, 16, etc) → Pérdida de precisión en la representación
- **Ejemplo: Representación binaria PF de -3.33333...**

Conversión de decimal a binario PF



1. Mantisa: 101 0101 0101 0101 0101 0101
2. Signo: negativo => 1
3. Exponente: 1+ 127 = 128_{diez} = $1000\ 0000_{dos}$

1	1000 0000	101 0101 0101 0101 0101 0101

Redondeo

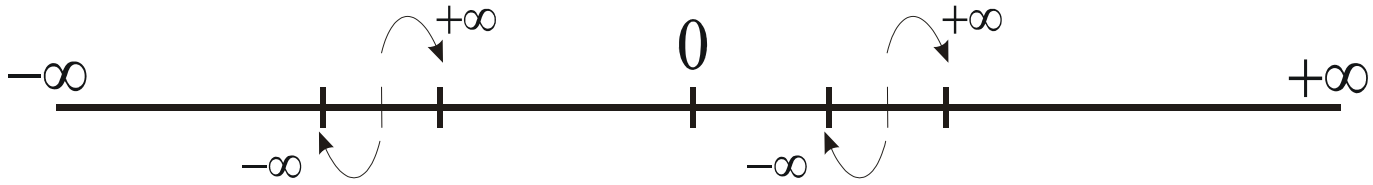
- El problema del redondeo aparece
  - Al convertir un número decimal a punto flotante
  - Al realizar una operación en punto flotante
- En muchos casos tenemos bits extra que exceden la longitud de la mantisa
- ¿Qué hacemos con ellos?

1.00010100010010101000110 x
1.10010000111000000111101=
1.10110000101001101011111*11010011000000110101110*

- Mantisa = 10110000101001101011111
- Sobra = 11010011000000110101110

Redondeo

- El estándar IEEE 754 define 4 modos de redondeo
  1. Truncamiento
     - El truncamiento es trivial, consiste en ignorar los bits adicionales
  2. Redondeo al número par más próximo
     - Es el más complejo
  3. Redondeo a +infinito (siempre hacia arriba)
  4. Redondeo a –infinito (siempre hacia abajo)



- El redondeo complica la implementación. Es posible que tras redondear haya que volver a normalizar la mantisa y sea necesario un segundo redondeo.