TEMA 2

REPRESENTACIÓN BINARIA

ÍNDICE

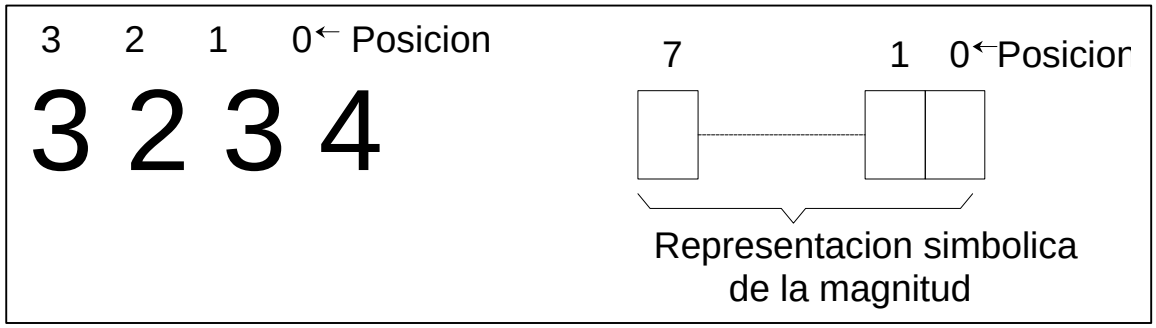
2. REPRESENTACIÓN POSICIONAL DE MAGNITUDES

La representación simbólica de magnitudes usando los signos numéricos (dígitos) habituales (0..9), necesitan de varios de estos dígitos, situados en distintas posiciones, si las magnitudes que representan son mayores o iguales a diez.

Por ejemplo, la magnitud representada por el símbolo 3234, expresa una cantidad igual a TRES mil, más DOS cientos, mas TRES decenas mas 4 unidades.

$$3234 = 3\times 1000 + 2\times 100 + 3\times 10 + 4\times 1$$

Los diez dígitos usados, desde el 0 … al 9 representan por sí solos 10 magnitudes diferentes. Cualquier magnitud mayor debe usar una combinación posicional de los anteriores, en la que, a cada dígito que se añade a la izquierda se le asigna un peso de valor igual a una potencia de diez. El exponente, n, de dicha potencia de diez, depende de la posición relativa de los números.



El dígito con menor peso asociado se denomina el **dígito menos significativo**. Por el contrario, el dígito con mayor peso asociado se denomina el **dígito más significativo**.

$$3234 = 3\times 10^3 + 2\times 10^2 + 4\times 10^1 + 1\times 10^0$$

Al vector de dimensión k, formado por los pesos de los k dígitos de un número, se denomina peso del número. Para el ejemplo anterior, el peso del número 3234 es (1000,100,10,1).

La base de representación numérica, B, utilizada hasta ahora, es la base diez (B=10). El valor de B se relaciona con el número de dígitos, que por sí solos, representan cantidades: 0,1,2,3,4,5,6,7,8,9.

$$3234 = 3\times B^3 + 2\times B^2 + 4\times B^1 + 1\times B^0 \text{ donde } B=10$$

Cada uno de los diez dígitos o signos anteriores representan una cantidad que varía desde 0 hasta 9, o sea, desde 0,.., hasta B-1.

Sea a_i un dígito que representa una magnitud comprendida entre 0 y B-1, (expresado como $a_i \in [0,..,B-1]$)entonces a_i es un dígito en base B. El conjunto de los dígitos a_i que representan a las magnitudes comprendidas entre [0,..,B-1] constituyen una **base de numeración**. El símbolo formado por la unión de dígitos a_i, representa la magnitud del número, donde el subíndice i del dígito representa la posición que ocupa este en el símbolo.

$$a_0 x B^0 + a_1 x B^1 + \ldots + a_{N-1} x B^{N-1} = \sum_{i=0}^{N-1} a_i x B^i$$

Una magnitud puede tener múltiples representaciones, dependiendo de la base de numeración.

Ejemplos:

76 unidades = 114 en octal = 114_8

$114_8 = 1\times 8^2 + 1\times 8^1 + 4\times 8^0 = 64_{10} + 8_{10} + 4_{10} = 76_{10}$

$1101_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = 13_{10} = 13$ unidades

$1101_4 = 1\times 4^3 + 1\times 4^2 + 0\times 4^1 + 1\times 4^0 = 81_{10} = 81$ unidades

$1101_{16} = 1\times 16^3 + 1\times 16^2 + 0\times 16^1 + 1\times 16^0 = 4353_{10} = 4353$ unidades

Existen infinitas bases de numeración, tantas como posibles valores de B. Sólo unas pocas son de nuestro interés: la base binaria, en la que B=2, la base octal, B=8, la base decimal, B=10 y la base hexadecimal, B=16.

	Base 2	Base 8	Base 10	Base hexadecimal
0	0	0	0	0
1	1	1	1	1
2		2	2	2
3		3	3	3
4		4	4	4
5		5	5	5
6		6	6	6
7		7	7	7
8			8	8
9			9	9
10				A
11				B
12				C
13				D
14				E
15				F

Los dígitos de la base binaria, llamados **bits**, son 2, el 0 y el 1.

Ejemplos de base hexadecimal

$8E_{16} = 8\times16^1 + E\times16^0 = 8\times16^1 + 14\times16^0 = 142_{10}$

$1BC_{16} = 1\times16^2 + B\times16^1 + B\times16^0 = 1\times16^2 + 11\times16^1 + 12\times16^0 = 444_{10}$

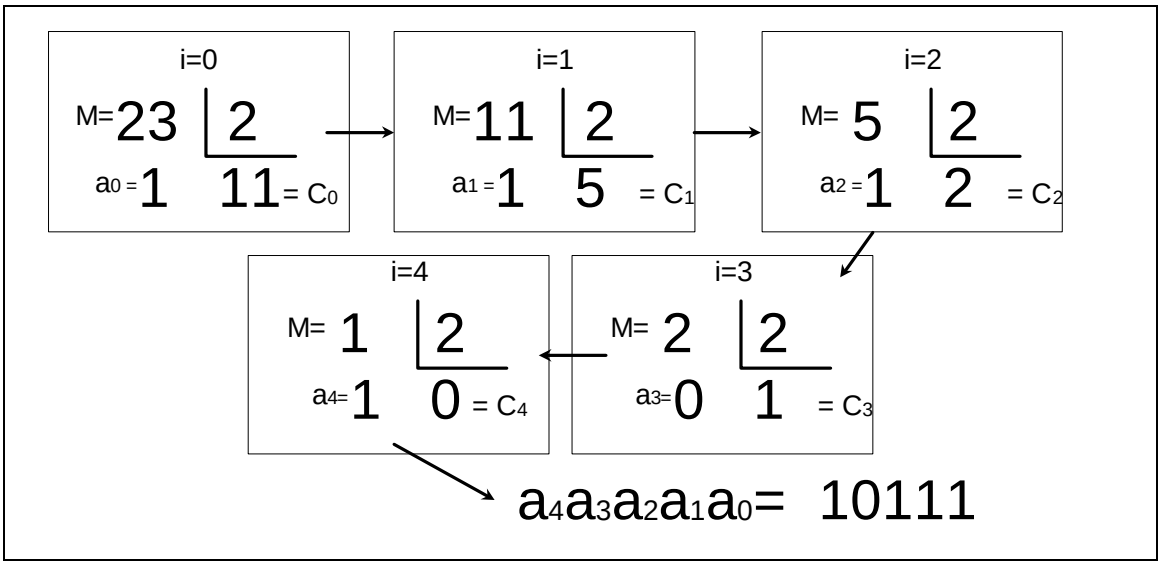
$1A0F_{16} = 1\times16^3 + A\times16^2 + 0\times16^1 + F\times16^0 = 1\times16^3 + 10\times16^2 + 0\times16^1 + 15\times16^0 = 6671_{10}$

2.1Transformaciones entre sistemas de representación (cambio de base)

2.1.1. Cambio de base de decimal a binario

Dado un número M en base 10, se desea encontrar el equivalente en base 2. Para ello se debe seguir el algoritmo que se presenta a continuación.

1. Sea el entero i = 0
2. Se divide el número M entre 2.
3. La división del punto 2 genera un resto que llamaremos a_i y un cociente C_i
4. Si el cociente C_i es distinto de cero, se hace M= C_i , se incrementa i y se repite desde el punto 2.
5. Si el cociente C_i es igual a cero, el proceso finaliza. El número en base 2 esta formado por el conjunto de los bits a_i donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el primer resto que se obtuvo (para i=0, a_0) es el bit menos significativo y, el último, el más significativo.



2.1.2. Cambio de base de binario a decimal.

Se aplica la siguiente expresión.

$$M = \sum_{i=0}^{N-1} a_i \times 2^i$$

2.1.3 Cambio de base de decimal a base p

1. Sea el entero i = 0
2. Se divide el número M entre p.
3. La división del punto 2 genera un resto que llamaremos a_i y un cociente C_i
4. Si p>10 y la magnitud del resto es a_i >=10, este debe convertirse al correspondiente dígito en base p)
5. Si el cociente C_i es distinto de cero, se hace M= C_i , se incrementa i y se repite desde el punto 2.
6. Si el cociente C_i es igual a cero, el proceso finaliza. El número en base p esta formado por el conjunto de los bits a_i donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el primer resto que se obtuvo (para i=0, a_0) es el bit menos significativo y, el último, el más significativo.

2.1.4 Cambio de base p a decimal

Se aplica la siguiente expresión.

$$M = \sum_{i=0}^{N-1} a_i \, x \, p^i$$

2.1.5 Cambio de representación de un número en base p a otro en base q

1. Se transforma previamente M a base 10, usando las técnicas descritas en el apartado 2.1.4. Llamemos R al número resultado de dicha transformación

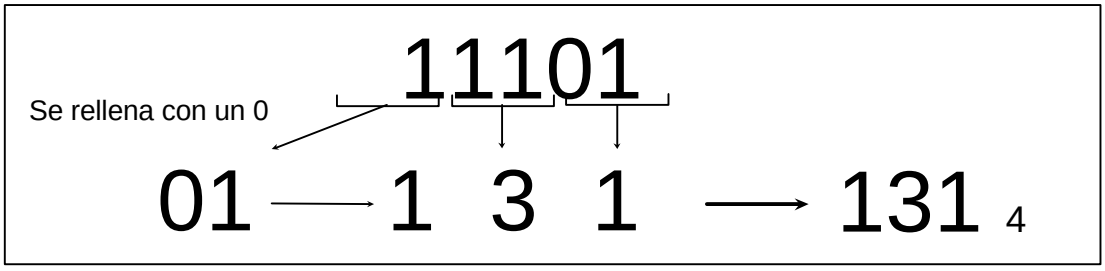
2. Se transforma R, expresado en base 10, a base q, usando las técnicas descritas en el apartado 2.1.3

2.1.6 Casos especiales de cambio de base.

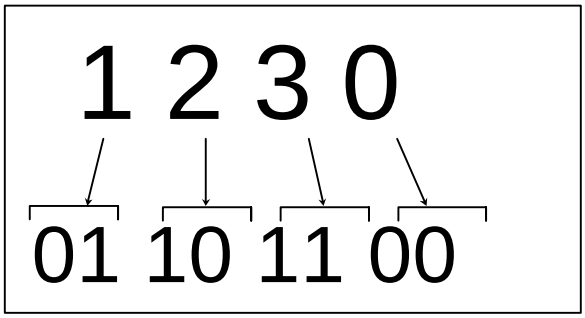
Si la base de partida, p, del número cuyo cambio de base se desea, se relaciona con la base, q, de representación final mediante algunas de las siguientes expresiones **$p = q^n$ o $p^n = q$**, donde n es un número entero mayor que 1, entonces se pueden aplicar técnicas de **compresión o expansión** de dígitos, respectivamente.

Ejemplos:

a) Cambio de base del número 11101_2 a base 4.



b) Cambio de base del número 1230_4 hacia su correspondiente en base 2.



2.1.6.1 Cambio de base binaria a base octal

Implica compresión en grupos de 3 bits.

Ejemplos:

$1001111011_2 = 001\ \ 001\ 111\ \ 011_2 = 1173_8$
$11111010000000_2 = \ \ 011\ \ 111\ \ 010\ \ 000\ \ 000_2 = 37200_8$

2.1.6.2 Cambio de base binaria a base hexadecimal

Implica compresión en grupos de 4 bits

Ejemplos:

$1001111011_2 = 0010\ 0111\ 1011_2 = \$27B$
$11111010000000_2 = 0011\ 1110\ 1000\ 0000_2 = \$3E80$

2.1.6.3 Cambio de base octal a binario

Implica expansión en grupos de 3 bits.

Ejemplos:

$7512_8 = 111\ 101\ 001\ 010_2 = 111101001010_2$
$2506_8 = 010\ 101\ 000\ 110_2 = 10101000110_2$

2.1.6.4 Cambio de base hexadecimal a binario

Implica expansión en grupos de 4 bits.

Ejemplos:

$\$F10A0 = 1111\ 0001\ 0000\ 1010\ 0000_2 = 11110001000010100000_2$
$\$2506 = 0010\ 0101\ 0000\ 0110_2 = 0010010100000110_2$

2.2 Representación de magnitudes con parte fraccionaria

Los dígitos pertenecientes a la parte fraccionaria de un número en base B, tienen unos pesos asociados iguales a B^{-i} donde i representa la posición que ocupa el "dígito fraccionario", siendo i=1 para el dígito más significativo de la parte fraccionaria, o sea, el que está más pegado a la coma, i=2, el dígito situado a su derecha, y así sucesivamente.

Para distinguir los dígitos de la parte fraccionaria de los de la parte entera, usaremos la expresión a_{-i}.

Ejemplos:

$23{,}456_{10} = 2\times10^1 + 3\times10^0 + 4\times10^{-1} + 5\times10^{-2} + 6\times10^{-3}$.

$10{,}101_2 = 1\times2^1 + 0\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

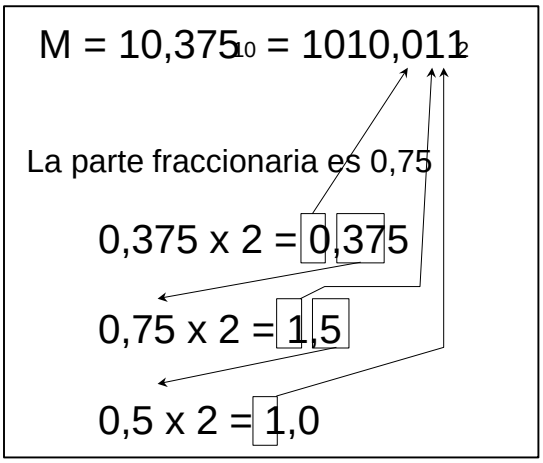
Un número en base B con p bits en su parte entera y q bits en su parte fraccionaria expresa una magnitud igual a la cantidad.

$$M = \sum_{i=0}^{p-1} a_i \times B^i \quad + \quad \sum_{i=1}^{q-1} a_{-i} \times B^{-i}$$

2.2.1. Cambio de base de decimal a binario

1. Sea el entero i = 1
2. Sea E la parte entera de M y F, la parte fraccionaria de M.
3. De M se retira la parte entera y se convierte a binario aplicando los métodos del apartado 2
4. Se multiplica la parte fraccionaria F por 2.
5. El resultado del punto 4 genera un número con una parte entera, que llamaremos a_{-i} y una fraccionaria, C_{-i}
6. Si C_{-i} es distinto de cero, se hace $F = C_i$, se incrementa i y se repite el punto 4.
7. Si el cociente C_{-i} es igual a cero, el proceso finaliza. El número en base 2 esta formado por el conjunto de los bits a_{-i} donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, la primera parte entera que se obtuvo (para i=1, a_{-1}) es el bit más significativo y, el último, el menos significativo.

La siguiente ilustración muestra el procedimiento para la conversión del número decimal 10,375 en binario.



Ejemplos:

$4{,}23_{10} = 100{,}00111...$
$18{,}0625_{10} = 10010{,}00010...$
$1{,}3_{10} = 1{,}0\ 1001\ 1001\ 1001\ ...$

2.2.2. Cambio de base de binario a decimal.

Se aplica la siguiente expresión.

$$M = \sum_{i=0}^{p-1} a_i \times 2^i \quad + \quad \sum_{i=1}^{q-1} a_{-i} \times 2^{-i}$$

2.2.3 Cambio de base de decimal a base B

1. Sea el entero i = 1
2. Sea E la parte entera de M y F, la parte fraccionaria de M.
3. De M se retira la parte entera y se convierte a base B aplicando los métodos del apartado 2
4. Se multiplica la parte fraccionaria F por B.
5. El resultado del punto 4 genera un número con una parte entera, que llamaremos a_{-i} y una fraccionaria, C_{-i}
6. Si C_{-i} es distinto de cero, se hace $F = C_i$, se incrementa i y se repite el punto 4.
7. Si el cociente C_{-i} es igual a cero, el proceso finaliza. El número en base 2 esta formado por el conjunto de los bits a_{-i} donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, la primera parte entera que se obtuvo (para i=1, a_{-1}) es el bit más significativo y, el último, el menos significativo.

2.2.4 Cambio de base B a decimal

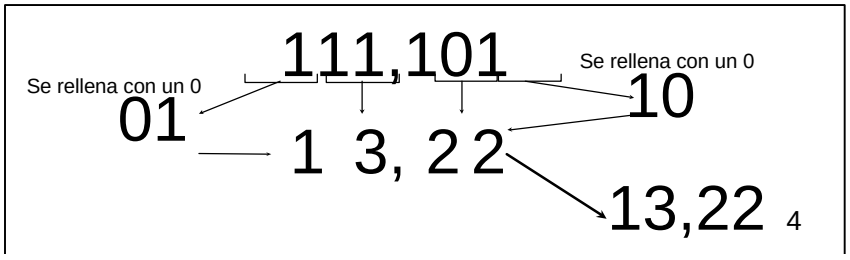
Se aplica la siguiente expresión.

$$M = \sum_{i=0}^{p-1} a_i \times B^i \quad + \quad \sum_{i=1}^{q-1} a_{-i} \times B^{-i}$$

2.2.5 Casos especiales de cambio de base

Aquí se incluyen aquellos en los que las bases destino y fuente de conversión de base están relacionadas de forma que, una, es igual a la potencia entera de la otra.

Ejemplos



$75{,}12_8 = 111\ \ 101,\ \ 001\ \ 010_2 = \ 111101{,}00101_2$
$250{,}6_8 = 010\ \ 101\ \ 000,\ \ 110_2 = \ 10101000{,}11_2$
$FA{,}0C_{16} = 1111\ \ 1010,\ \ 0000\ \ 1100_2 = \ 11111010{,}000011_2$
$2{,}A98_{16} = 0010,\ \ 1010\ \ 1001\ \ 1000_2 = \ 10{,}101010011_2$
$E7{,}0C_{16} = 32\ 13,\ 00\ 30_4 = \ 3214{,}003_4$
$10{,}100111110_2 = 0010,\ \ 1001\ 1111_2 = \ 2{,}9F_{16}$
$10{,}100111110_2 = 010,\ \ 100\ 111\ 110_2 = \ 2{,}476_8$

3. Códigos binarios

Un código es una colección de símbolos y reglas que permite formular e identificar cierta información. Cuando dicha colección simbólica está formada por agrupaciones de bits, el código se denomina binario.

Ejemplos de códigos binarios para los números decimales
a)

Número decimal	Código binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

b)

Número decimal	Código binario
0	0110000
1	0110001
2	0110010
3	0110011
4	0110100
5	0110101
6	0110110
7	0110111
8	0111000
9	0111001

Existe infinitas representaciones o códigos de un mismo conjunto de elementos. En efecto, si a cada elemento del conjunto le asignamos una hilera arbitraria de unos y ceros de modo que de forma unívoca cada hilera identifique un único elemento del conjunto, se habrá conseguido generar un código binario de dicho conjunto.

En cualquier caso, es interesante conocer el mínimo número de bits necesarios para poder codificar un conjunto de elementos.

En general, un código binario de p bits es capaz de codificar un conjunto de 2^p elementos. De forma inversa, para un conjunto de N elementos, el mínimo número p de bits necesarios para codificar dichos elementos debe cumplir la siguiente relación

$$2^{p-1} < N \leq 2^p$$

Dada un número N de elementos, los p bits mínimos necesarios para la codificación se pueden obtener mediante

$$p = \mathrm{Re}[\log_2 N]$$

3.1 Códigos binarios para la representación de los dígitos decimales

3.1.1 Código BCD (Binary Code for Decimal digits)

Es un código de 4 bits, utilizado para la codificación de los diez dígitos decimales. Cada grupo de cuatro bits del código utiliza la representación posicional binaria.

Número decimal	Código BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ejemplos:

$1234_{10} = (0001\ \ 0010\ \ 0011\ \ 0100)_{BCD}$

$709_{10} = (0111\ \ 0000\ \ 1001)_{BCD}$

3.1.2. Código exceso-3 (Excess-3)

Número decimal	Código Excess-3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Ejemplos:

$1234_{10} = (0100\ \ 0101\ \ 0110\ \ 0111)_{excess\text{-}3}$

$709_{10} = (1010\ \ 0011\ \ 1100)_{excess\text{-}3}$

3.1.3. Código 2-de-5

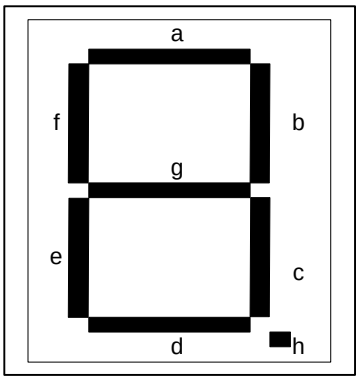
Número decimal	Código 2-de-5
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

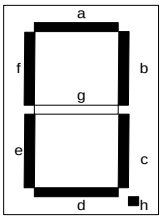
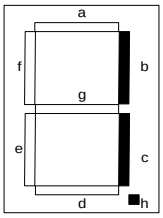
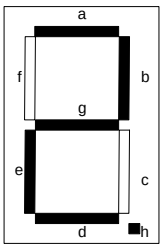
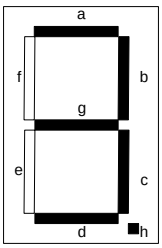
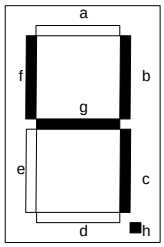
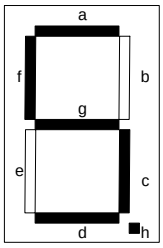
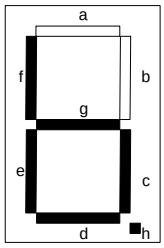
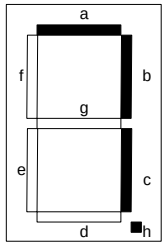
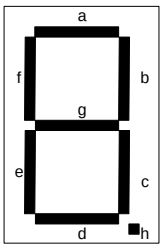
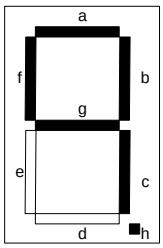
Ejemplos:

$1234_{10} = (00101\ \ 00110\ \ 01001\ \ 01010)_{2\text{-}de\text{-}5}$
$709_{10} = (10010\ \ 00000\ \ 11000)_{2\text{-}de\text{-}5}$

3.1.4 Códigos de siete segmentos

Es un código no pesado de 7 bits, utilizado para la representación de los números decimales en displays o pantallas de siete segmentos



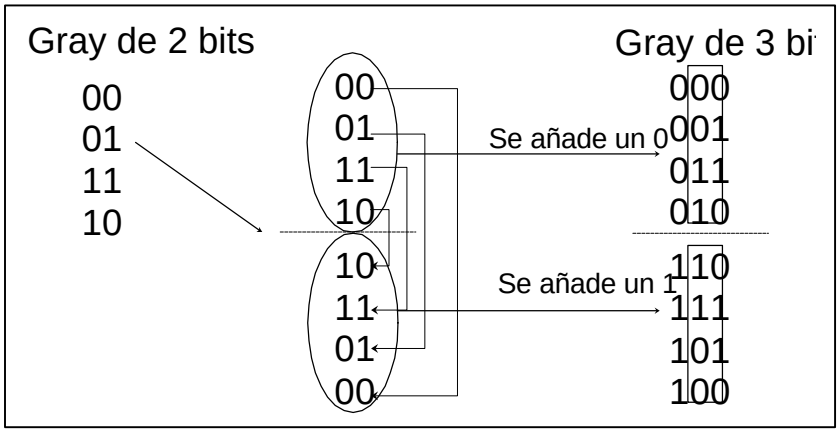
Código BCD	Código 7-segmentos **a b c d e f g**		Código BCD	Código 7-segmentos **a b c d e f g**	
0 0 0 0	1 1 1 1 1 1 0		0 0 0 1	0 1 1 0 0 0 0	
0 0 1 0	1 1 0 1 1 0 1		0 0 1 1	1 1 1 1 0 0 1	
0 1 0 0	0 1 1 0 0 1 1		0 1 0 1	1 0 1 1 0 1 1	
0 1 1 0	0 0 1 1 1 1 1		0 1 1 1	1 1 1 0 0 0 0	
1 0 0 0	1 1 1 1 1 1 1		1 0 0 1	1 1 1 0 0 1 1	

3.2 Código Gray

Tiene la propiedad de que sólo existe un bit diferente entre dos elementos consecutivos del código.

Número	Código Gray de 3 bits
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Si se conoce el código Gray de n bits, es fácil obtener el código Gray correspondiente a n+1 bits.



	Gray de 1 bit	Gray de 2 bits	Gray de 3 bits	Gray de 4 bits
0	0	0**0**	0**00**	0**000**
1	1	0**1**	0**01**	0**001**
2		1**1**	0**11**	0**011**
3		1**0**	0**10**	0**010**
4			1**10**	0**110**
5			1**11**	0**111**
6			1**01**	0**101**
7			1**00**	0**100**
8				**1100**
9				**1101**
10				**1111**
11				**1110**
12				**1010**
13				**1011**
14				**1001**
15				**1000**

3.3 Códigos alfanuméricos

Son aquellos que representan tanto letras, como números y demás signos de puntuación. Para codificar un total de más de 64 símbolos gráficos (26 letras minúsculas, 26 letras mayúsculas, 10 números y demás signos de puntuación como interrogantes, admiraciones, comas, puntos, etc) son necesarios siete bits como mínimo.

Uno de los más usados es el ASCII (American Standard Code for Information Interchange) que puede ser de siete u ocho bits.

		$C_6C_5C_4$							
		000	001	010	011	100	101	110	111
	0000	NUL	DEL	SP	0	@	P	'	p
	0001	SOH	DC1	!	1	A	Q	a	q
	0010	STX	DC2	"	2	B	R	b	r
	0011	ETX	DC3	#	3	C	S	c	s
	0100	EOT	DC4	$	4	D	T	d	t
	0101	ENQ	NAK	%	5	E	U	e	u
	0110	ACK	SYN	&	6	F	V	f	v
$C_3C_2C_1C$	0111	BEL	ETB	'	7	G	W	g	w
	1000	BS	CAN	(	8	H	X	h	x
	1001	HT	EM	)	9	I	Y	i	y
	1010	LF	SUB	*	;	J	Z	j	z
	1011	VT	ESC	+	:	K	[	k	{
	1100	FF	FS	´	<	L	\	l	|
	1101	CR	GS	-	=	M	]	m	}
	1110	S0	RS	.	>	N	^	n	~
	1111	S1	US	/	?	O	-	o	DEL

3.4 Códigos detectores de errores

Los códigos detectores de errores se construyen a partir de códigos predeterminados a los que se les añade cierta información redundante.

El método más simple de codificación (y por eso poco eficiente -50%-) es el basado en el bit de paridad.

Existen dos tipos de bit de paridad: bit de paridad par o bit de paridad impar.

Ejemplo: Supóngase el código binario de 4 bits formado por todos los números comprendidos entre 0 y 15. La siguiente tabla muestra el código detector de error por el método del bit de paridad resultante del primero, tanto para paridad par como impar

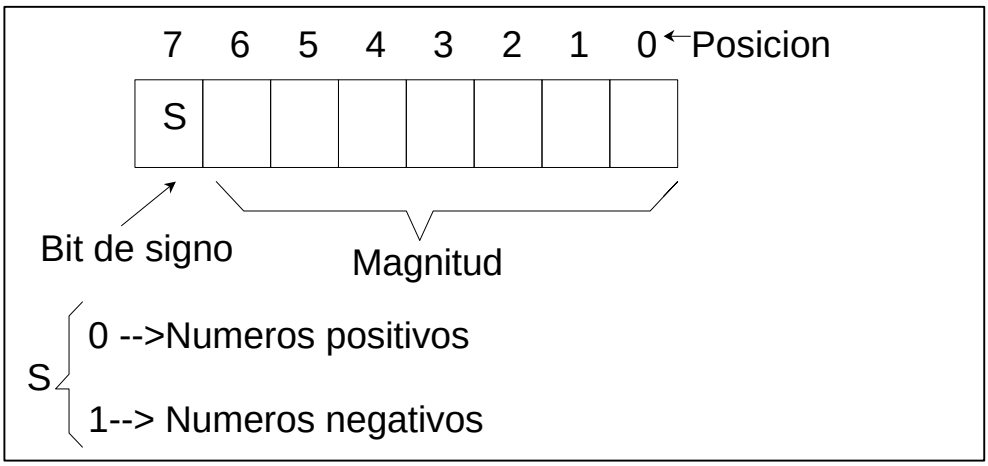
	Bit de paridad par	Bit de paridad Impar
0	**0**0000	**1**0000
1	**1**0001	**0**0001
2	**1**0010	**0**0010
3	**0**0011	**1**0011
4	**1**0100	**0**0100
5	**0**0101	**1**0101
6	**0**0110	**1**0110
7	**1**0111	**0**0111
8	**1**1000	**0**1000
9	**0**1001	**1**1001
10	**0**1010	**1**1010
11	**1**1011	**0**1011
12	**0**1100	**1**1100
13	**1**1101	**0**1101
14	**1**1110	**0**1110
15	**0**1111	**1**1111

4. Representación de números con signo

- Notación signo-magnitud (S-M)
- Notación en Complemento a 1(Ca1)
- Notación en Complemento a 2(Ca2)

4.1 Notación signo-magnitud

Es la más "humana" de las representaciones de números con signo, puesto que, al conjunto de los bits que representa la magnitud del número se antepone (en la posición más significativa) un bit, denominado bit de signo, que toma el valor 0 para números positivos y el 1, para los negativos.



Ejemplo: +4 usando 4 bits para la magnitud = 00100
-4 usando 4 bits para la magnitud = 10100

	Código S-M		Código S-M
+0	0000	-0	1000
+1	0001	-1	1001
+2	0010	-2	1010
+3	0011	-3	1011
+4	0100	-4	1100
+5	0101	-5	1101
+6	0110	-6	1110
+7	0111	-7	1111

En general, se puede afirmar que si se utilizan **n** bits para representar un **número A con signo en notación S-M**, el rango de valores posibles para A está comprendido entre:

$$-2^{n-1} \leq A \leq 2^{n-1}$$

El operador unario complemento

Se define el operador complemento a N de un número M expresado en base N, y se representa como CaN(M_N) como la transformación que genera el siguiente resultado

$$CaN\ (M_N) = N^p - M_N$$

Donde p es el número de dígitos de la parte entera de M_N.

Se define el operador complemento a N-1 de un número M expresado en base N, y se representa como CaN-1(M_N) como la transformación que genera el siguiente resultado

$$CaN\text{-}1\ (M_N) = N^p - N^{-q} - M_N$$

Donde p es el número de dígitos de la parte entera de M_N y q el número de dígitos de la parte fraccionaria.

Propiedades de operadores complemento

1a.- $CaN(CaN(M_N)) = M_N$.
1b.- $CaN\text{-}1(CaN\text{-}1(M_N)) = M_N$.

El complemento a N del complemento a N de un número M, es el propio número M. (Idém para el complemento a N-1)

2. $CaN\text{-}1(M_N) = CaN(M_N) - N^{-q}$ ó $CaN(M_N) = CaN\text{-}1(M_N) + N^{-q}$ donde q es el número de dígitos de la parte fraccionaria del número N.

Demostración:

Como $CaN\ (M_N) = N^p - M_N$

Entonces $CaN\text{-}1\ (M_N) = N^p - N^{-q} - M_N = (N^p - M_N) - N^{-q} = CaN(M_N) - N^{-q}$

4.2 Notación en Complemento a 1

Los números positivos en notación Ca1 se expresan igual que en SM. En cambio, los números negativos se obtienen a partir de aplicar el operador Ca1 al número expresado como si fuera positivo.

Ejemplo: +4 usando 4 bits para la magnitud = 00100 en Ca1
-4 usando 4 bits para la magnitud = Ca1(00100) = 11011

	Código Ca1		Código Ca1
+0	0000	-0	1111
+1	0001	-1	1110
+2	0010	-2	1101
+3	0011	-3	1100
+4	0100	-4	1011
+5	0101	-5	1010
+6	0110	-6	1001
+7	0111	-7	1000

En general, se puede afirmar que si se utilizan **n** bits para representar un **número A con signo en notación Ca1,** el rango de valores posibles para A está comprendido entre:

$$-2^{n-1} \leq A \leq 2^{n-1}$$

4.3 Notación en Complemento a 2

Los números positivos en notación Ca2 se expresan igual que en SM y en Ca1En cambio, los números negativos se obtienen a partir de aplicar el operador Ca2 al número expresado como si fuera positivo.

Ejemplo: +4 usando 4 bits para la magnitud = 00100 en Ca1
-4 usando 4 bits para la magnitud = Ca2(00100) = 11100

	Código Ca2		Código Ca2
+0	0000	-1	1111
+1	0001	-2	1110
+2	0010	-3	1101
+3	0011	-4	1100
+4	0100	-5	1011
+5	0101	-6	1010
+6	0110	-7	1001
+7	0111	-8	1000

En general, se puede afirmar que si se utilizan **n** bits para representar un **número A con signo en notación Ca1,** el rango de valores posibles para A está comprendido entre:

$$-2^{n} \leq \quad A \leq \quad 2^{n-1}$$

4.4 Comparación entre las distintas notaciones numéricas

Todas ellas requieren de un bit de signo situado en la posición más significativa, y con idéntico significado: un 0 para los números positivos, y un 1 para los negativos.

Los números positivos se representan de forma idéntica en las tres notaciones, sólo cambia para los negativos.

La notación SM y Ca1 tienen dos codificaciones distintas para un mismo número (+0 y -0), situación esta que no ocurre en Ca2.

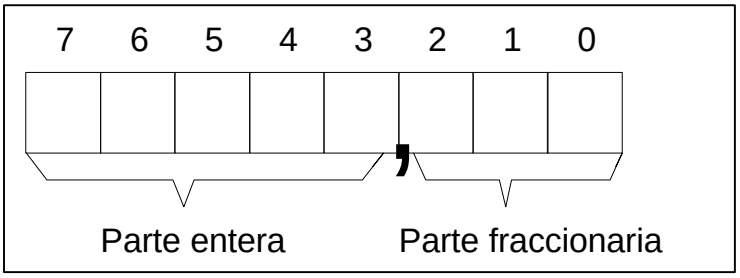
5. Representación binaria de números reales

5.1 Representación en coma fija

Los números binarios, enteros y fraccionarios, son almacenados en las máquinas digitales en unos elementos denominados "registros".

Estos registros tienen una capacidad finita de almacenamiento. Un registro de n bits, almacena un total de n bits, como parece evidente.

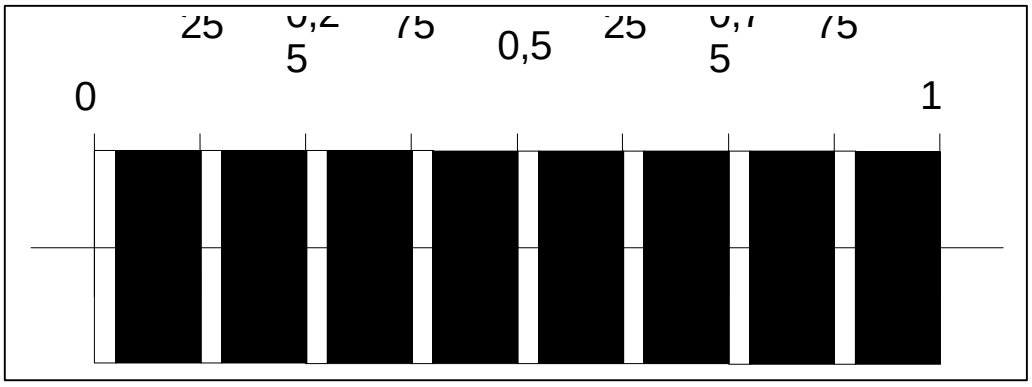
Para almacenar números reales, una porción del registro se debe destinar a la parte entera, y el resto, a la fraccionaria.



Ejemplo: ejemplo de almacenamiento del número 5.5_{10} que en binario es 101.1_2 ,

En resumen, el número de bits de la parte fraccionaria siempre es fijo lo que limita las magnitudes fraccionarias y enteras representadas. Para el caso que nos ocupa, tres bits para la parte fraccionaria, sólo unas pocas fracciones pueden ser almacenadas con exactitud.

.



En cualquier caso, el número de celdas reservadas para almacenar la parte fraccionaria es finito, mientras que la cantidad de números fraccionarios es infinita. Esto puede generar errores de precisión.

- El truncado de un número elimina, de este, aquellos bits de la parte fraccionaria que no se pueden almacenar en el registro.
- El redondeo de un número A, tiende a almacenar dicho número como si fuera otro número B, lo más cercano posible a A, que si sea representable con exactitud en el registro.

5.1 Representación en coma flotante

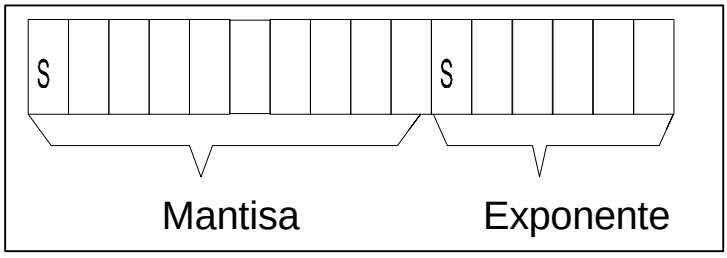
Esta representación busca el almacenamiento del número real en forma de exponencial. Cualquier número real en base b puede ser expresado en la forma siguiente

$$N = m \times b^{e}$$

Ejemplos:

a) el número $2{,}45_{10}$ puede expresarse como $245x10^{-2}$, siendo la mantisa, m=245, y el exponente, e=-2.
b) El número binario 100.11, se representa, exponencialmente, como $10011x2^{-10}$. La mantisa, ahora, es m=10011, y el exponente, e=-10 (-2 en decimal).

Cualquier número real binario que se almacene con el formato exponencial sólo precisa guardar la mantisa y el exponente, la base se da por conocida.



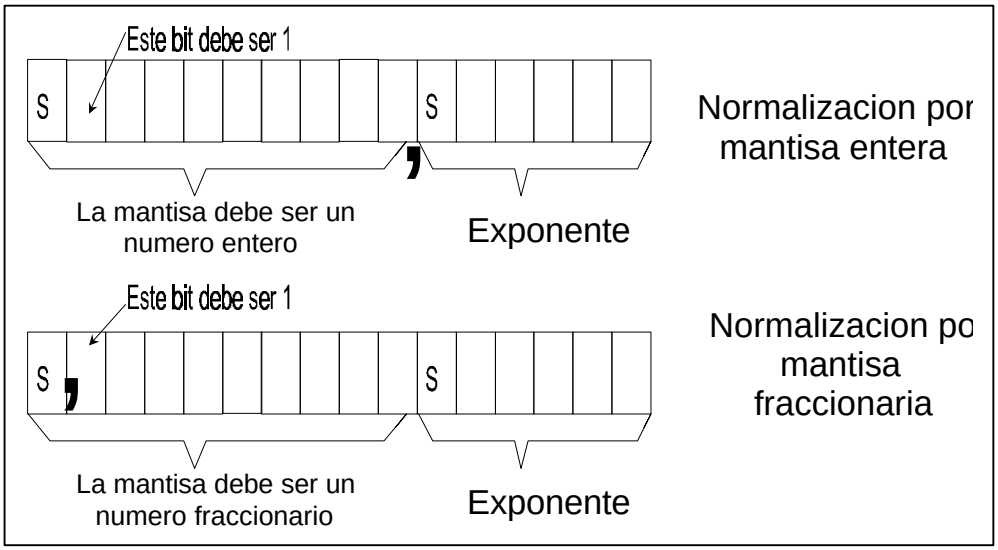
La notación numérica para representar mantisas y exponentes con signo será la signo-magnitud.

Un inconveniente, a priori, de este tipo de representación, es la multitud de representaciones distintas para un mismo número.

Ejemplo: Supóngase que se desea representar el número binario +100,1 en notación coma flotante con 8 bits para la mantisa y 4 para el exponente.

$100,1 = 1001x2^{-01}$ (o equivalentemente $0001001x2^{1001}$)
$100,1 = 10010x2^{-10}$ (o $0010010x2^{1010}$),
$100,1 = 00100x2^{-11}$(o $0100100x2^{-1011}$)
$100,1 = 0,0001001x2^{110}$ (o $0100100x2^{0110}$).

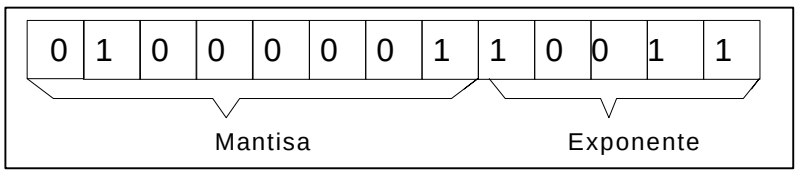
Existen infinitas formas de representación de un único número real. Para no crear confusión en el modo de interpretación de los números en coma flotante almacenados se utilizan normalizaciones. Estas buscan que el dígito más significativo de la mantisa, el siguiente al bit de signo, sea distinto de cero. Existen dos tipos: **normalización por mantisa entera** y **normalización por mantisa fraccionari**a.



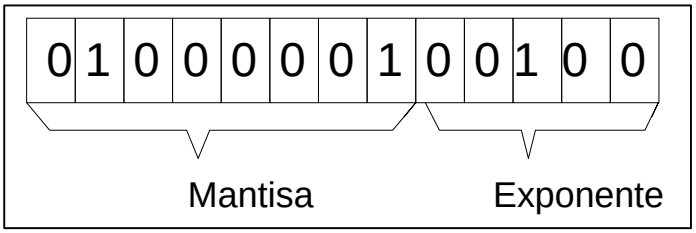
Ejemplo:

Se dispone de 8 bits para la mantisa y 5 para el exponente. Represente el número binario +1000,001 usando a) la normalización por mantisa entera y b) la normalización por mantisa fraccionaria.

a)

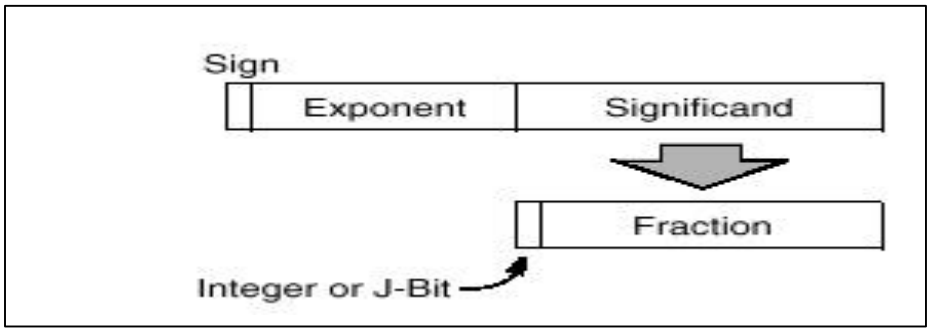


b)



5.1.1 Formato IEEE 754

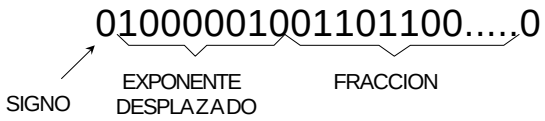
Es el estándar de representación de números en coma flotante.



El número real representado depende del exponente y el significando, de la siguiente forma

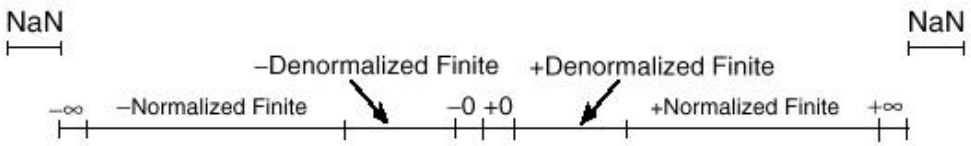
$$(-1)^{S} \times 2^{E-127} \times 1.F$$

Ejemplo: El real corto



Representa el número

$$(-1)^{0} \times 2^{130-127} \times 1.01101..$$



Real Number and NaN Encodings For 32-Bit Floating-Point Format

S	E	F			S	E	F
1	0	0	–0	+0	0	0	0
1	0	$0.XXX^2$	–Denormalized Finite	+Denormalized Finite	0	0	$0.XXX^2$
1	1...254	Any Value	–Normalized Finite	+Normalized Finite	0	1...254	Any Value
1	255	0	–∞	+∞	0	255	0
X^1	255	$1.0XX^2$	–SNaN	+SNaN	X^1	255	$1.0XX^2$
X^1	255	1.1XX	–QNaN	+QNaN	X^1	255	1.1XX

NOTES:
1. Sign bit ignored.
2. Fractions must be non-zero.