

TEMA 2: REPRESENTACIÓN DE LA INFORMACIÓN

1. Sistemas de numeración y cambio de base
2. Aritmética binaria
3. Sistemas de codificación y representación numéricos
4. Representación de los números enteros
5. Representación de los números reales
6. Detección y corrección de errores

TEMA 2: REPRESENTACIÓN DE LA INFORMACIÓN

Bibliografía:

- ❑ P. M. Anasagasti. Fundamentos de los Computadores.
 - Cap. 2 Representación de la Información.
- ❑ T.L.Floyd. Fundamentos de Sistemas Digitales.
 - Cap. 2: Sistemas de Numeración, Operaciones y Códigos.
- ❑ C.Blanco. Fundamentos de Electrónica Digital.
 - Cap. 1: Sistemas y Códigos Numéricos.
- ❑ J.M^a Angulo y J. García. Sistemas Digitales y Tecnología de Computadores.
 - Cap. 2: Sistemas de Numeración y Códigos.
- ❑ W. Stallings. Organización y Arquitectura de Computadores.
 - Cap. 9: Aritmética del Computador. Representación en Coma Flotante.
 - Cap. 4: Memoria Interna. Memoria Principal Semiconductora. Corrección de errores.

1. Sistemas de Numeración y Cambio de Base

Un sistema de numeración en *base* ***b*** utiliza para representar los números un alfabeto compuesto por ***b*** símbolos, dígitos o cifras.

Ejemplos:

***b* = 10** (*decimal*) {0,1,2,3,4,5,6,7,8,9}

***b* = 8** (*octal*) {0,1,2,3,4,5,6,7}

***b* = 16** (*hexadecimal*) {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

***b* = 2** (*binario*) {0,1}

El número se expresa mediante una secuencia de cifras:

$$N \equiv \dots n_4 n_3 n_2 n_1 n_0 n_{-1} n_{-2} n_{-3} \dots$$

El valor de cada cifra depende de la cifra en si y de la posición que ocupa en la secuencia. Todo número ***N*** sometido a un sistema *posicional* se puede descomponer como un polinomio de potencias de la base:

$$N \equiv \dots + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} \dots$$

Es decir, $N = \sum_i n_i b^i$

Ejemplos:

$$3278,52_{10} = 3 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

$$175,372_8 = 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 + 3 \cdot 8^{-1} + 7 \cdot 8^{-2} + 2 \cdot 8^{-3} = 125,48828125_{10}$$

Conversión decimal - base b

- Para obtener la representación de un número decimal entero N en una base b (con $b < 10$) bastará con que dividamos sucesivamente N entre b . Agrupando los restos junto con el último cociente en orden inverso al obtenido, obtendremos el número buscado.

1. Sistemas de Numeración y Cambio de Base

	<u>Cociente</u>	<u>Resto</u>
363 : 2	181	1
181 : 2	90	1
90 : 2	45	0
45 : 2	22	1
22 : 2	11	0
11 : 2	5	1
5 : 2	2	1
2 : 2	1	0

$$363_{10} = 10110101_2$$

Para obtener la representación de la parte fraccionaria bastará con que multipliquemos sucesivamente N por b . Agrupando las partes enteras en el mismo orden de su obtención obtendremos la representación buscada.

$0.67245 \times 2 = 1.3449$	→	1
$0.34490 \times 2 = 0.6898$	→	0
$0.68980 \times 2 = 1.3796$	→	1
$0.37960 \times 2 = 0.7592$	→	0
$0.75920 \times 2 = 1.5184$	→	1
$0.51840 \times 2 = 1.0368$	→	1

$$0.67245_{10} = 0.1010110..._2$$

¿Cómo actuaríamos si tenemos que convertir un número a una base mayor de 10?

1. Sistemas de Numeración y Cambio de Base

Denominamos Rango de Representación al conjunto de valores representable en una determinada base. Con n cifras en la base b podemos formar b^n combinaciones distintas. $[0..b^n-1]$.

El **sistema binario** es un sistema posicional. Utiliza únicamente dos símbolos: el “0” y el “1”, a los que denominamos bits.

Binario	Decimal
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Ejemplos:

$$\begin{aligned} 110100_2 &= (1 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^2) = 2^5 + 2^4 + 2^2 = 32 + 16 + 4 = 52_{10} \\ 0,10100_2 &= 2^{-1} + 2^{-3} = (1/2) + (1/8) = 0,625_{10} \\ 10100,001_2 &= 2^4 + 2^2 + 2^{-3} = 16 + 4 + (1/8) = 20,125_{10} \end{aligned}$$

2. Aritmética Binaria

Las operaciones básicas que podemos realizar son las mismas que en cualquier otra base: suma, resta, multiplicación y división.

Suma binaria

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ y llevo } 1$$

$$1 + 1 + 1 = 1 \text{ y llevo } 1$$

Ejemplos:

$$\begin{array}{r} 1110101 \\ + 1110110 \\ \hline 11101011 \end{array}$$

$$\begin{array}{r} 10110110 \\ + 10011011 \\ \hline 101010001 \end{array}$$

$$\begin{array}{r} 101101101 \\ + 010100011 \\ \hline 1000010000 \end{array}$$

Multiplicación binaria:

$$0 \times 0 = 0$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 1 = 1$$

Ejemplos:

$$\begin{array}{r} 1101010 \\ \times 11 \\ \hline 1101010 \\ 1101010 \\ \hline 100111110 \end{array}$$

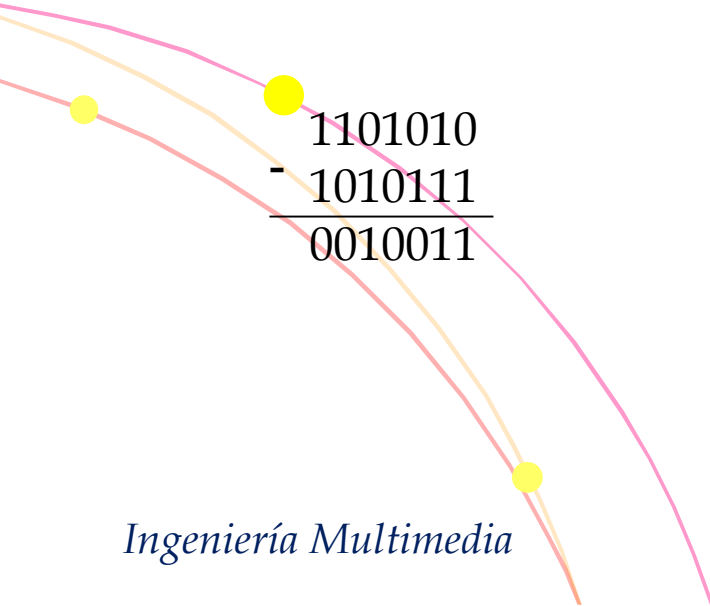
$$\begin{array}{r} 1010011 \\ \times 101 \\ \hline 1010011 \\ 1010011 \\ \hline 110011111 \end{array}$$

$$\begin{array}{r} 1011011 \\ \times 1011 \\ \hline 1011011 \\ 1011011 \\ 1011011 \\ \hline 1111101001 \end{array}$$

Resta binaria:

$$\begin{array}{l} 0 - 0 = 0 \\ 1 - 0 = 1 \\ 1 - 1 = 0 \\ 0 - 1 = 1 \text{ y llevo } 1 \end{array}$$

Ejemplos:


$$\begin{array}{r} 1101010 \\ - 1010111 \\ \hline 0010011 \end{array}$$

$$\begin{array}{r} 10110110 \\ - 01111010 \\ \hline 00111100 \end{array}$$

$$\begin{array}{r} 1011011000 \\ - 0101110011 \\ \hline 0101100101 \end{array}$$

División binaria:

$$0 \div 0 = \text{Operación no definida}$$

$$0 \div 1 = 0$$

$$1 \div 0 = \text{Operación no definida}$$

$$1 \div 1 = 1$$
Ejemplos:

$$\begin{array}{r} 1101,011 \\ - 101 \\ \hline \end{array}$$

$$\begin{array}{r} 0011\ 0 \\ - 101 \\ \hline \end{array}$$

$$\begin{array}{r} 00\ 111 \\ - 101 \\ \hline \end{array}$$

$$\begin{array}{r} 10 \\ \hline \end{array}$$

$$\begin{array}{r} \overline{)101} \\ 10,101 \end{array}$$

$$\begin{array}{r} 1011011101 \\ - 1110 \\ \hline \end{array}$$

$$\begin{array}{r} 010001 \\ - 1110 \\ \hline \end{array}$$

$$\begin{array}{r} 0001111 \\ - 1110 \\ \hline \end{array}$$

$$\begin{array}{r} 000101 \end{array}$$

$$\begin{array}{r} \overline{)1110} \\ 110100 \end{array}$$

3. Sistemas de Codificación y Representación Numérica

Sistema Octal:

Puesto que necesitamos 8 símbolos diferentes se utilizan del 0 al 7. Su utilidad radica en la fácil conversión a sistema binario y viceversa.

Conversión decimal a octal.

	Cociente	Resto
$2014 : 8 =$	251	6
$251 : 8 =$	31	3
$31 : 8 =$	3	7

$$2014_{10} = 3736_8$$

La correspondencia con el sistema binario es inmediata. Puesto que $8 = 2^3$, una cifra en octal corresponde a 3 binarias:

$$10001101100.1101_2 = \underbrace{010}_{2}\underbrace{001}_1\underbrace{110}_5\underbrace{1100}_4.\underbrace{110}_6\underbrace{100}_4 = 2154.64_8$$

$$537.24_8 = \underbrace{101}_5\underbrace{011}_3\underbrace{111}_7.\underbrace{010}_2\underbrace{100}_4 = 101011111.0101_2$$

Sistema Hexadecimal:

Como ahora vamos a necesitar 16 símbolos diferentes utilizaremos los números del 0 al 9 junto con las 6 primeras letras, es decir de la 'A' a la 'F'.

Conversión decimal a hexadecimal.

	Cociente	Resto
$4373 : 16 =$	273	5
$273 : 16 =$	17	1
$17 : 16 =$	1	1
$4373_{10} = 1115_{16}$		

La correspondencia con el sistema binario es inmediata. Puesto que $16 = 2^4$, una cifra en hexadecimal corresponde a 4 binarias:

$$10010111011111.1011101 = \underbrace{0010}_2 \underbrace{0101}_5 \underbrace{1101}_D \underbrace{1111}_F . \underbrace{1011}_B \underbrace{1010}_A = 25DF.BA_{16}$$

$$592.D8_{16} = 10110010010.11011_2$$

3. Sistemas de Codificación y Representación Numérica

Hexadecimal	Decimal	Binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Código Gray:

Es un código no ponderado, continuo (los números decimales consecutivos se codifican con combinaciones adyacentes, es decir que solo difieren en un bit) y cíclico (la última combinación es adyacente a la primera). También se denomina reflejado debido a la forma de obtenerse.

2 bits	3 bits	4 bits	Decimal
0 0	0 0 0	0 0 0 0	0
0 1	0 0 1	0 0 0 1	1
1 1	0 1 1	0 0 1 1	2
1 0	0 1 0	0 0 1 0	3
	1 1 0	0 1 1 0	4
	1 1 1	0 1 1 1	5
	1 0 1	0 1 0 1	6
	1 0 0	0 1 0 0	7
		1 1 0 0	8
		1 1 0 1	9
		1 1 1 1	10
		1 1 1 0	11
		1 0 1 0	12
		1 0 1 1	13
		1 0 0 1	14
		1 0 0 0	15

Conversión Binario-Gray:

El primer bit coincide siempre. A partir de éste, sumamos el bit binario que queremos obtener con el de su izquierda.

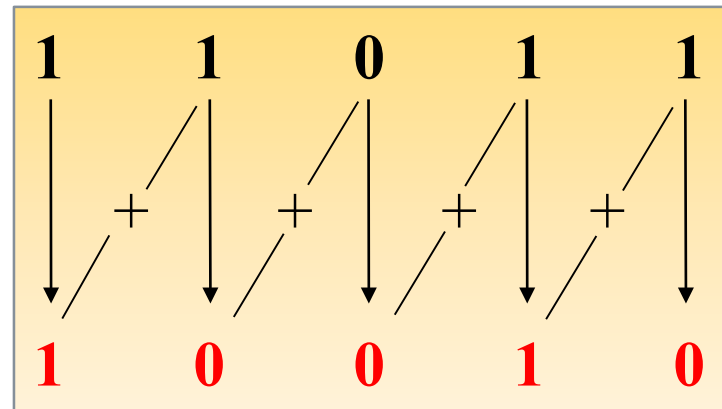
Ejemplo:

1	0	1	1	0	→	Código Binario Original
↓						
1					→	El Primer bit coincide
1	+	0	1	1	0	→ Sumamos el 1 ^{er} y 2 ^o bit
		↓				
1		1				→ Obtenemos el 2 ^o bit en Gray
1	0	+	1	1	0	→ Sumamos el 2 ^o y 3 ^{er} bit
			↓			
1	1		1			→ Obtenemos el 3 ^{er} bit en Gray
1	0	1	+	1	0	→ Sumamos el 3 ^{er} y 4 ^o bit
				↓		
1	1	1		0		→ Obtenemos el 4 ^o bit en Gray
1	0	1	1	+	0	→ Sumamos el 4 ^o y 5 ^o bit
				↓		
1	1	1	0		1	→ Obtenemos el 5 ^o bit en Gray
1	1	1	0	1		→ Código Gray equivalente

Conversión Gray-Binario:

El primer bit siempre coincide. A partir de éste, a cada bit del código binario generado se le suma el bit en código Gray de la siguiente posición adyacente, descartando los acarreos.

Ejemplo:



4. Números enteros

Para poder expresar números positivos y negativos, se hace necesaria la representación del signo. Existen diferentes métodos de representación de los números con signo.

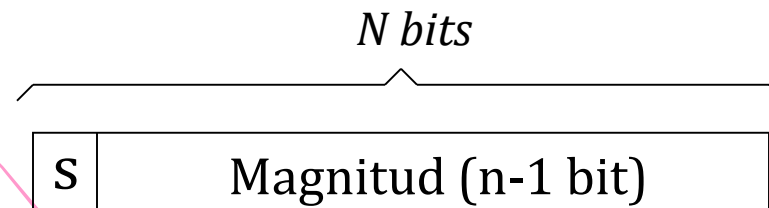
Signo y Magnitud

El signo se representa en el bit más a la izquierda del dato. [Bit (n-1)], según el criterio:

0 : Número Positivo

1 : Número Negativo

En el resto de los bits se representa el valor del número en binario natural. [Bits (n-2)..0]



Signo y Magnitud

Ejemplos:

$$01101_2 = +13_{10}$$

$$11101_2 = -13_{10}$$

Rango de Representación:

$$-(2^{n-1} - 1) \leq x \leq (2^{n-1} - 1)$$

Existe duplicidad en la representación del cero.

Valor de un número cualquiera

$$\left. \begin{array}{l} \text{si } x_{n-1} = 0, \quad x = \sum_{i=0}^{n-2} x_i 2^i \\ \text{si } x_{n-1} = 1, \quad x = -\sum_{i=0}^{n-2} x_i 2^i \end{array} \right\} x = (1 - 2 \cdot x_{n-1}) \sum_{i=0}^{n-2} x_i 2^i$$

Complemento Restringido a la Base

La representación de un número negativo $(-A)$ la obtenemos a partir de la expresión:

$$-A \rightarrow B^n - 1 - A$$

donde B es la base en que estamos trabajando y n el número de dígitos de que disponemos

Ejemplos:

Base 10 (Complemento a 9)

$$n=2 \quad -26_{10} \rightarrow 10^2 - 1 - 26 = 73 \quad (-26_{10} = 73_{C9})$$

$$n=3 \quad -26_{10} \rightarrow 10^3 - 1 - 26 = 973 \quad (-26_{10} = 973_{C9})$$

$$n=4 \quad -1384_{10} \rightarrow 10^4 - 1 - 1384 = 8615 \quad (-1384_{10} = 8615_{C9})$$

Si particularizamos para la **Base 2**, hablaremos del **Complemento a 1**. La representación de los números negativos la obtendremos a partir de la expresión:

$$(-A)_{C1} \rightarrow 2^n - 1 - A$$

Ejemplo:

Representemos con 6 bits ($N = 6$) el número -18_{10} .

$$-18 \rightarrow 2^6 - 1 - 18 = 64 - 1 - 18 = 45$$

$$-18_{10} = 101101_{C1}$$

Como características citaremos:

- Los valores positivos tienen la misma representación que en SM.
- Los números positivos empiezan por cero
- Los números negativos empiezan por uno
- Presenta doble representación del 0.
- El complemento se hace y se deshace de la misma forma
- El Rango de Representación es: $[-2^{n-1} + 1, 2^{n-1} - 1]$

Complemento a la Base

La representación de un número negativo $(-A)$ la obtenemos a partir de la expresión:

$$-A \rightarrow B^n - A$$

donde B es la base en que estamos trabajando y n el número de dígitos de que disponemos

Ejemplos:

Base 10 (Complemento a 10)

$n=3$	$-63_{10} \rightarrow 10^3 - 63 = 937$	$(-26_{10} = 937_{C10})$
$n=3$	$-16_{10} \rightarrow 10^3 - 16 = 984$	$(-16_{10} = 984_{C10})$
$n=4$	$-16_{10} \rightarrow 10^4 - 16 = 9984$	$(-1384_{10} = 9984_{C10})$

Si particularizamos para la **Base 2**, hablaremos del **Complemento a 2**. La representación de los números negativos la obtendremos a partir de la expresión:

$$(-A)_{C2} \rightarrow 2^n - A$$

Ejemplo:

Representemos con 6 bits ($N = 6$) el número -21_{10} .

$$-21 \rightarrow 2^6 - 21 = 64 - 21 = 43$$

$$-21_{10} = 101011_{C2}$$

Como características citaremos:

- Los valores positivos tienen la misma representación que en SM y en C1.
- Todos los números positivos empiezan por cero
- Los números negativos empiezan por uno
- Tiene una representación única del 0.
- El complemento a 2 se hace y se deshace de la misma forma
- El Rango de Representación es: $[-2^{n-1}, 2^{n-1} - 1]$

4. Representación de los Números Enteros

Para obtener la representación de un número negativo en Complemento a 2 se intercambian ceros por unos y unos por ceros a la representación del número positivo y sumaremos 1.

Ejemplo: Representación del -21_{10} con 6 bits.

$$21_{10} = 10101_2 \rightarrow +21 = 010101_{C2}$$

$$-21_{10} \rightarrow 101010 + 1 = 101011_{C2}$$

Sumas y Restas en Complemento a 2. Ejemplo.

$$(1011101_2) - (110010_2)$$

$$+1011101_2 = 01011101_{C2}$$

$$-110010_2 = 11001110_{C2}$$

El acarreo final **NO**
forma parte del
resultado

$$\begin{array}{r}
 01011101_{C2} \\
 + \\
 11001110_{C2} \\
 \hline
 (1)00101011_{C2}
 \end{array}$$

Representación sesgada

La representación se obtiene sumando un sesgo o cantidad al valor del número.

El sesgo suele ser: 2^{n-1} , en cuyo caso el rango de representación será $[-2^{n-1}, 2^{n-1} - 1]$.

Ejemplos:

$$n = 8 \Rightarrow \text{Sesgo} = 2^{8-1} = 128_{10} = 1000\ 0000_2$$

$$\begin{array}{rcl}
 11010_2 & = & 10011010_S \\
 -11010_2 & = & 01100110_S \\
 0_2 & = & 10000000_S
 \end{array}$$

$$n = 4 \Rightarrow \text{Sesgo} = 2^{4-1} = 8_{10} = 1000_2$$

$$\begin{array}{rcl}
 1_2 & = & 1001_S \\
 -1_2 & = & 0111_S
 \end{array}$$

5. Números reales

Representación en coma fija

El formato de coma fija reserva una cantidad de bits para la representación de la parte entera y otra para la parte fraccionaria. El número de bits de cada una de las partes es siempre el mismo independientemente del valor que se desee representar.

Ejemplo:

Disponemos de un formato en coma fija compuesto por 5 bits para la parte entera y 3 para la parte fraccionaria.

$$11,25 = 01011.010$$

$$16,5 = 10000.100$$

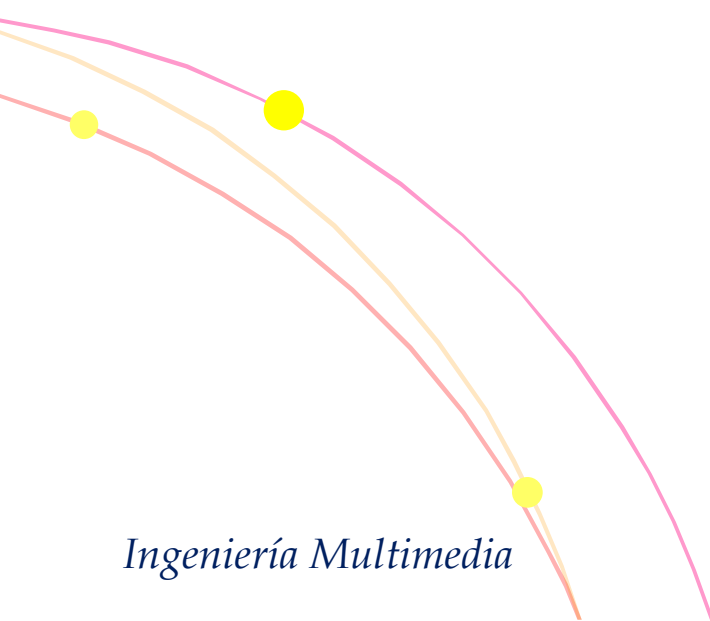
$$0,125 = 00000.001$$

También se suele incluir un bit para el signo.

Rango de Representación.

Si para la representación de x disponemos de 1 bit de signo, n bits para la parte entera y p para la parte fraccionaria, su rango será:

$$-[(2^n - 2^{-p}), 2^{-p}] \leq x \leq [2^{-p}, (2^n - 2^{-p})]$$



Representación en coma flotante.

El formato de coma flotante utiliza el siguiente formato para representar cualquier número:

$$N = \pm M \cdot B^E$$

Donde:

N \equiv Valor numérico

M \equiv Mantisa

B \equiv Base

E \equiv Exponente

Ejemplo:

$$\begin{aligned} 1.234535 \cdot 10^3 &= 1234.535 \cdot 10^0 = 0.1234535 \cdot 10^4 = \\ &= 123453.5 \cdot 10^{-2} = 0.0001234535 \cdot 10^7 \end{aligned}$$

Estándar IEEE754

$$N = (-1)^s M \cdot 2^E$$

Representación **S E M**

$$N = nS + nE + nM$$

Siendo nS la cantidad de bits para el signo, nE la cantidad de bits para el exponente y nM la cantidad de bits para la mantisa.

Estándar IEEE754 (simple precisión)

Consta de 32 bits, distribuidos de la siguiente forma:

Campo de signo : 1 bit

Campo del exponente: 8 bits con representación sesgada, siendo el sesgo:

$$S = 2^{nE-1} - 1$$

Ejemplos:

Puesto que tenemos $nE = 8 \Rightarrow S = 2^{nE-1} - 1 = 127 = 0111\ 1111$

(E)	E+S	(e)
0	$127+0=127$	0111 1111
+2	$127+2=129$	1000 0001
+127	$127+127=254$	1111 1110
-1	$127-1=126$	0111 1110
-126	$127-126=1$	0000 0001

Campo de mantisa : 23 bits, con formato normalizado: $1 \leq M < 2$

La mantisa siempre tendrá la forma $M = [1.m]$ donde m es el valor que se almacena en el formato.

Ejemplos de normalización:

$$\begin{aligned} N1 &= 1001.1100110 \cdot 2^{-5} &= 1.0011100110 \cdot 2^{-2} \\ N2 &= 0.000001101101 \cdot 2^{34} &= 1.101101 \cdot 2^{28} \end{aligned}$$

Ejercicio:

Supongamos un formato de las mismas características que el IEEE754, pero dotado solo de 16 bits, de los cuales 1 es para el signo y 8 para el exponente. Identifiquemos el número:

$$N = 1001 \ 1111 \ 0001 \ 1101$$

$$s = 1 \Rightarrow N < 0 \qquad e = 0011 \ 1110 \Rightarrow E = -65$$

$$m = 001 \ 1101 \Rightarrow M = 1.0011101_2 = -1.2265625_{10}$$

$$N = -1.2265625 \cdot 2^{-65} = -3,32440346980633 \cdot 10^{-20}$$

Situaciones especiales:

Si el en el campo del exponente encontramos $e = 0$, la mantisa está **desnormalizada**. El sesgo pasa a ser $2^{ne-1}-2$. Y el exponente del número será:

$$E = e - S = -2^{ne-1} + 2$$

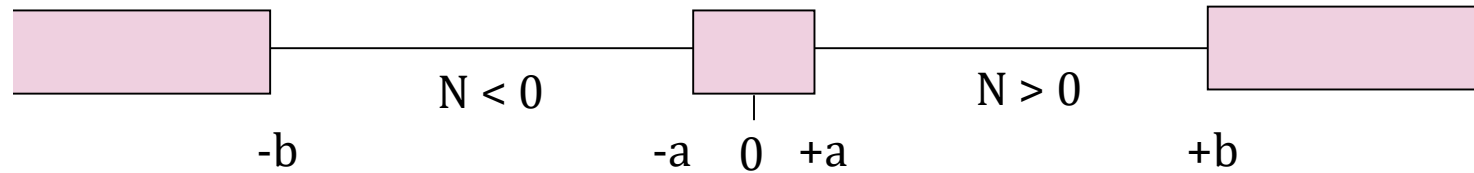
- Si $(e = 0) \wedge (m = 0) \Rightarrow N = 0$
- Si $(e = 11...1) \wedge (m = 0) \Rightarrow N = \infty$
- Si $(e = 11...1) \wedge (m \neq 0) \Rightarrow N = \text{NaN}$

Redondeo:

El redondeo de un número lo podemos realizar por exceso, por defecto, al más cercano o al par. El formato IEEE754 emplea el redondeo al par.

Ejemplos:

Resultado	Acción	Mantisa
1.01101 10	Sumar 1	1.01110
1.01100 10	Truncar	1.01100
1.01100 11	Sumar 1	1.01101
1.01100 01	Truncar	1.01100
1.01100 00	Truncar	1.01100

Rango de Representación:*Números Normalizados:*

$$|b| = M_{\max} 2^{E_{\max}} \begin{cases} M_{\max} = 2 - 2^{-nm} \\ E_{\max} = 2^{ne-1} - 1 \end{cases} \quad |a| = M_{\min} 2^{E_{\min}} \begin{cases} M_{\min} = 1 \\ E_{\min} = -(2^{ne-1} - 2) \end{cases}$$

Números Desnormalizados:

$$|a'| = M'_{\min} 2^{E'_{\min}} \begin{cases} M'_{\min} = 2^{-nm} \\ E'_{\min} = -(2^{ne-1} - 2) \end{cases}$$

6. Detección y Corrección de Errores.

Introducción. Definición de error y tipos de errores.

Cuando trabajamos con ordenadores hemos de tener en cuenta que la información circula entre diferentes máquinas (por ejemplo, a través de una red) o bien por el interior de una ellas (entre sus diferentes componentes) y se almacena en determinados dispositivos.

La aparición de ruidos en las comunicaciones o la existencia de defectos en la superficie de un disco son, entre otras causas, fuentes de la aparición de cambios accidentales en la información que se está manejando.

Denominamos error a la alteración de la información desde que se emite (o se almacena) hasta que se recibe (o se recupera).

Esta alteración consistirá en el cambio de valor de algunos bits.

Tipos de errores:

Dependiendo de cómo aparecen hablaremos de errores ***aislados*** o de ***ráfagas*** de errores.

En un error aislado tendremos un bit erróneo rodeado de otros correctos. Pueden ser simples o múltiples.

Información Inicial:

0	1	1	0	1	0	1	0	1	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Error Aislado Simple:

0	1	1	0	1	0	1	1	1	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Errores Aislados Múltiples:

0	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ráfaga de Errores:

0	1	1	1	0	1	0	1	1	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Tipos de códigos para el tratamiento de errores

El principal objetivo es salvaguardar la información ante posibles errores.

Para ello, se añade una información adicional a cada dato.

Dependiendo de las necesidades podemos definir códigos que son capaces de detectar y/o corregir errores.

En un código **detector** cada carácter o bloque de información se establece conforme a unas reglas específicas de construcción, de forma que las desviaciones de estas reglas indican la presencia de un error. Si además este error puede repararse de forma automática estaremos ante un código **corrector**.

- Los códigos detectores se suelen utilizar cuando la corrección es muy costosa y es posible la retransmisión de la información.

Si no es posible o conveniente esta retransmisión utilizaremos un código corrector.

Denominamos *Distancia de Hamming* entre dos vectores al número de bits que toman valores diferentes.

Ejemplo:

$$c1 = 0101 \text{ y } c2 = 1100$$

Tienen un distancia de Hamming de 2.

Distancia de un código es la mínima distancia Hamming entre todas las posibles combinaciones distintas que forman el código.

Ejemplo:

código: {100, 111, 011}

$$\min\{d(100, 111), d(100, 011), d(111, 011)\} = \min\{2, 3, 1\} = 1$$

La *Propiedad Detectora* de un código nos dice que si la distancia de un código es d , entonces podemos detectar errores que afecten a $d-1$ bit.

Ejemplo:

$$C = \{001, 010, 100\} \quad \text{Distancia de Hamming} = 2$$

- Un error aislado siempre se detecta
 - Un error en 001 \Rightarrow 101, 011, 000, $\notin C$
- Dos errores aislados no se detectan
 - Dos errores en 001 \Rightarrow 111, 010, 100. Los dos últimos pertenecen a C

La *Propiedad Correctora* de un código establece que si la distancia de un código es d , entonces es posible detectar y corregir los errores que afecten a t bits según la siguiente expresión:

$$d = 2 \cdot t + 1$$

Es decir, podemos llegar a corregir $t = \frac{d-1}{2}$ errores.

Ejemplo:

$$C = \{0000000000, 0000011111, 1111100000, 1111111111\}$$

La Distancia de Hamming es 5.

Por tanto:

- Se pueden detectar $d-1 = 5-1 = 4$ errores
- Se pueden corregir $(d-1)/2 = 4/2 = 2$ errores

Códigos para el tratamiento de errores.

Existen diversos métodos para la detección de la existencia de errores, entre los que podemos citar:

- Detección de paridad
 - Simple
 - Cruzada (horizontal y vertical)
- Códigos de redundancia cíclica.

Detección de paridad simple

Este método añade a cada dato un bit denominado de paridad. Dependiendo del bit añadido, hablaremos de *paridad par* o de *paridad impar*. Permite detectar errores de un bit.

- Bit de paridad par:
 - Si el nº total de “1” del dato es par: Bit de paridad = 0
 - Si el nº total de “1” del dato es impar: Bit de paridad = 1
- Bit de paridad impar:
 - Si el nº total de “1” del dato es par: Bit de paridad = 1
 - Si el nº total de “1” del dato es impar: Bit de paridad = 0

Ejemplo:

'H'	1001000	0
'O'	1001111	1
'L'	1001100	1
'A'	1000001	0

← Bit de paridad par

Detección de paridad cruzada

Este método agrupa palabras de la información como bloques de tamaño fijo dando lugar a una matriz de m filas y k columnas. Añade un bit de paridad simple para cada una de las filas (paridad horizontal, LRC) y para cada una de las columnas (paridad vertical, VRC) de esta matriz. Adicionalmente tendremos un bit de paridad (par) cruzada que reflejará la paridad de los LCR y VRC.

Ejemplo: Para enviar la palabra “HOLA” en ASCII (7 bits) construiremos una matriz de 7×4 .

'H':	1001000	0	← LRC
'O':	1001111	1	
'L':	1001100	1	
'A':	1000001	0	
VRC →	0001010	0	← Paridad Cruzada

Por tanto, la información enviada será: 90 , 9F, 99, 82, 14 (hexadecimal)

A través de VRC podemos detectar fácilmente los errores de ráfaga que podrían quedar enmascarados en el LRC.

En cualquier caso, siempre seremos capaces de detectar que existe error si se ha producido el cambio en 3 bits como máximo.

Si tuviéramos la certeza de que el error se ha producido en un bit único seríamos capaces de corregirlo.

Por ejemplo, la aparición de 4 errores consecutivos queda reflejada en VRC.

'H':	0110000	0
'O':	1001111	1
'L':	1001100	1
'A':	1000001	0
	1110010	0

Si se trata de 4 errores aislados que ocupan posiciones idénticas en dos palabras diferentes pasarían inadvertidos.

Los códigos de redundancia cíclica (CRC), también conocidos como códigos polinomiales constituyen el método de detección de errores más empleado en comunicaciones.

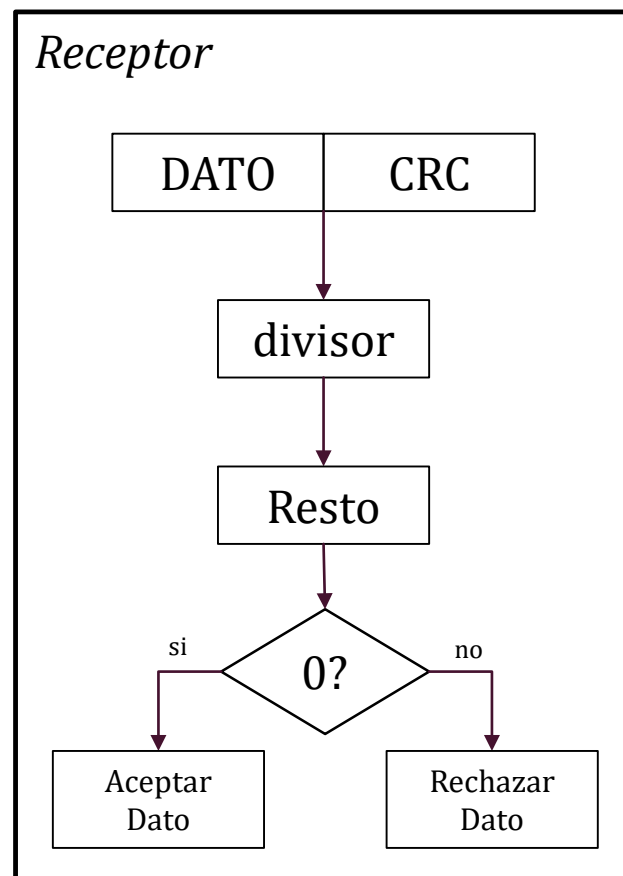
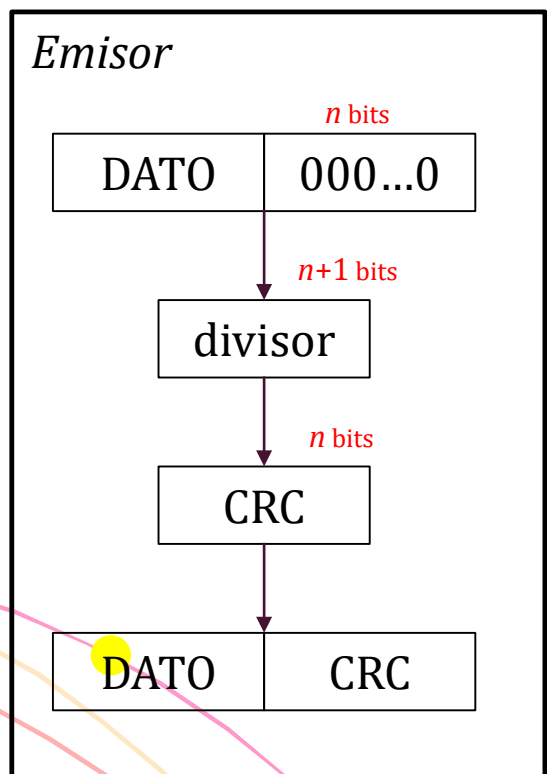
Se utiliza con esquemas de transmisión orientados a tramas (o bloques).

Permiten sustanciales mejoras en fiabilidad respecto a los métodos anteriores, siendo a la vez una técnica de fácil implementación.

En este método, una secuencia de bits redundantes denominados CRC se añade al final de los datos a transmitir de forma que la totalidad de la información sea divisible de forma exacta por un determinado número binario.

En el destino, la información de entrada se vuelve a dividir por el mismo número. Si la operación no obtiene resto, los datos se asumen como correctos. Y en caso contrario se indica que los datos están dañados y que es necesaria su retransmisión.

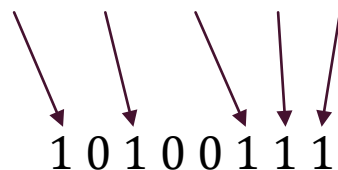
Los bits redundantes que se añaden como CRC se derivan de la división de los datos entre un divisor predeterminado. El resto constituye el CRC.



El divisor se obtiene a partir de los denominados polinomios generadores.

Por ejemplo, el polinomio $x^7 + x^5 + x^2 + x + 1$, proporciona el divisor:

1 0 1 0 0 1 1 1



El polinomio generador debe ser tal que no sea divisible por x y sí lo sea por $x + 1$.

Habitualmente se utilizan los estándares internacionales:

- CRC-12: $x^{12} + x^{11} + x^3 + x^2 + 1$
- CRC-16: $x^{16} + x^{15} + x^2 + 1$
- CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$
 - 100% errores simples
 - 100% errores dobles
 - 100% errores en un número impar de bits
 - 100% errores en ráfagas menores o iguales 16 bits
 - 99.997% errores de ráfagas de 17 bits
 - 99.998% de errores en ráfagas de 18 o más bits

Ejemplo:

Tomando como polinomio generador: $x^3 + 1$, obtén su CRC e indica que información que se enviará si el dato original es 10011011.

El polinomio generador nos proporciona un divisor compuesto por 4 bits: 1001

Por tanto, añadiremos a la información original 3 ceros por la derecha: 10011011000

A continuación dividiremos en módulo 2. Es decir, mediante aritmética sin acarreos.

10011011000	1001
1001	10001010
<u>0000</u>	
1011	
<u>1001</u>	
001000	
<u>1001</u>	
00010	

La información que se envía será: 10011011010

A la recepción se efectúa nuevamente la división:

10011011010		1001
1001		10001010
<u>0000</u>		
1011		
<u>1001</u>		
001001		
<u>1001</u>		
00000		