Tema 1: Sistemas de numeración y códigos binarios

Diferencia 1 *entre una persona que ha estudiado Ingeniería y una que no: "Un ingeniero es una persona que puede hacer lo que haga cualquier persona que no ha estudiado Ingeniería, pero más barato"*

Si A es el éxito en la vida, entonces A = X + Y + Z. El trabajo es X, Y es la suerte y Z es mantener la boca cerrada.

Albert Einstein

En cierta ocasión, un alumno le preguntó a un profesor por qué no daba consejos sobre como estudiar la asignatura. La respuesta del profesor fue: "los alumnos inteligentes no los necesitan y los demás no los escuchan".

Introducción a los sistemas de numeración (*conceptos básicos* y *propiedades*)

_ Un *sistema de numeración* no es más que un método para representar cantidades de forma simbólica, siguiendo unas determinadas reglas y convenios.

_ La *representación* de una cantidad en un sistema de numeración se denomina de forma genérica como *número* y dicha representación (número) es única en cada sistema de numeración. En general, una misma cantidad se representará mediante un número distinto en función del sistema de numeración que se considere. Así, por ejemplo, los siguientes números representan la misma cantidad (cada número corresponde a la representación en un sistema de numeración distinto):

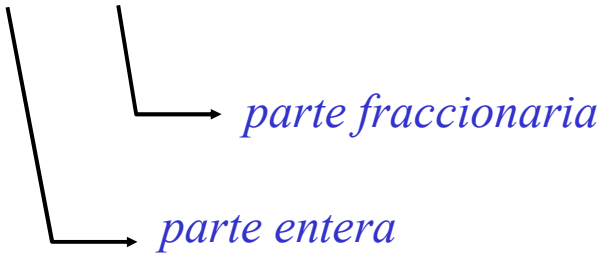
$$20_{10} \equiv 14_{16} \equiv 24_{8} \equiv 110_{4} \equiv 202_{3} \equiv 10100_{2}$$

Nota: un *sistema de numeración* se utiliza para representar de forma simbólica el *módulo*, la *magnitud* o el *valor absoluto* de cantidades (la forma de indicar si una cantidad es positiva o negativa es una cuestión aparte)

_ Hoy en día se utilizan *sistemas de numeración posicionales*, los cuales se caracterizan por lo siguiente:

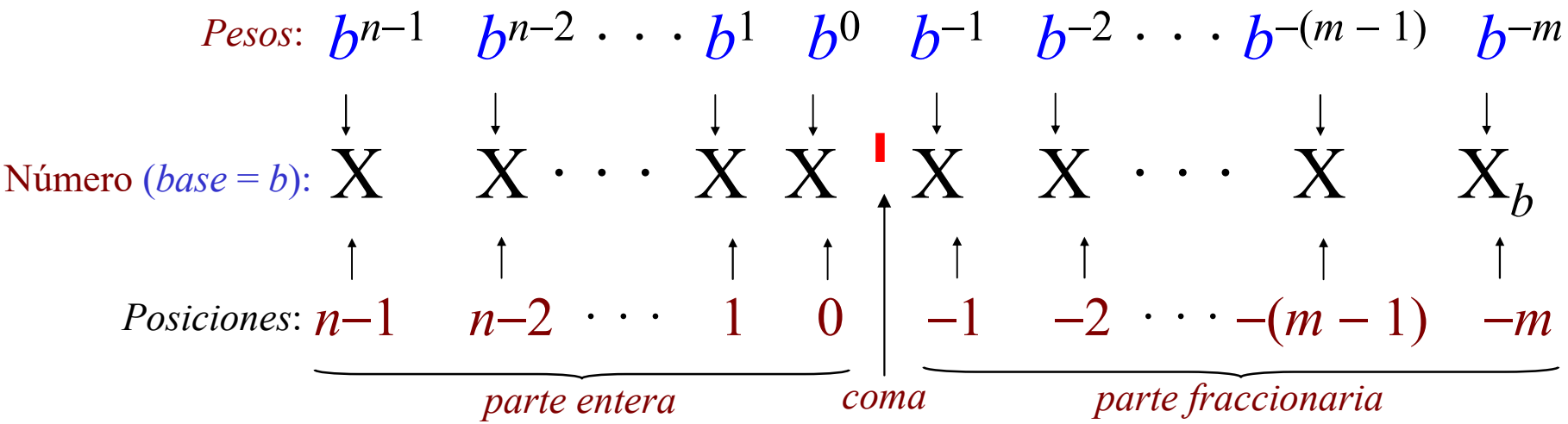
- Un *número* está formado por un conjunto de *símbolos, dígitos* o *cifras* situados a la izquierda y a la derecha de una coma (o punto) de referencia. Los dígitos situados a la izquierda de la coma representan la parte entera de la cantidad, mientras que los dígitos situados a la derecha de la coma representan la parte fraccionaria de la cantidad.

Ejemplo: $47\,'\,525_{10}$

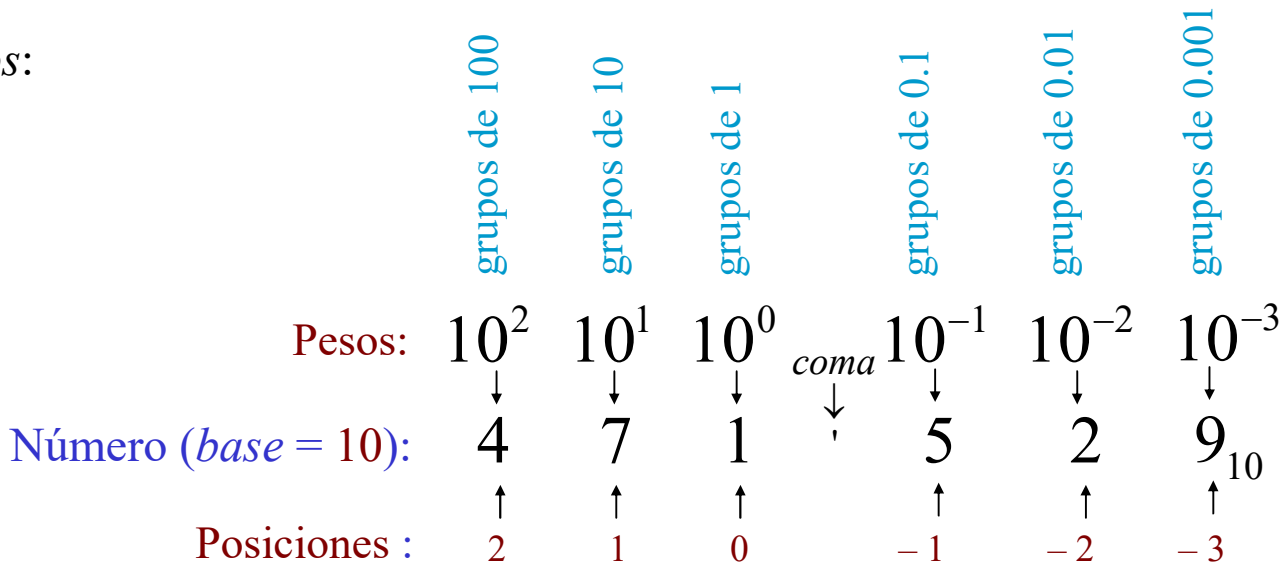


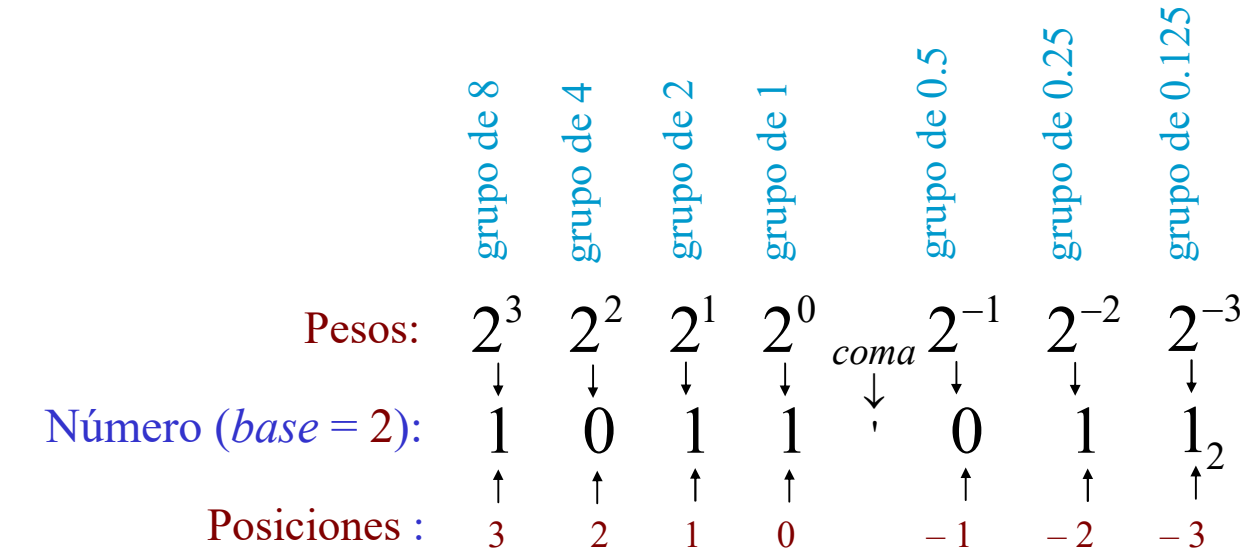
Nota: *en esta asignatura, a los dígitos* (1 y 0) *de un número binario también los vamos a denominar bits, siendo* bit *la contracción de las palabras* binary digit (*dígito binario*)

▪ En un número, cada dígito tiene asociado un *peso* cuyo valor depende de la *base* (*b*) del número y de la *posición* que ocupe el dígito en el número. Si consideramos un número con *n* dígitos en la parte *entera* y *m* dígitos en la parte *fraccionaria*, las *posiciones* y los *pesos* asociados a cada uno de los dígitos guardan la siguiente relación:



Ejemplos:


grupos de 100
grupos de 10
grupos de 1
grupos de 0.1
grupos de 0.01
grupos de 0.001
Pesos:
10^2 10^1 10^0 *coma* 10^{-1} 10^{-2} 10^{-3}
Número (*base* = 10):
4 7 1 5 2 9_{10}
Posiciones :
2 1 0 – 1 – 2 – 3


grupo de 8
grupo de 4
grupo de 2
grupo de 1
grupo de 0.5
grupo de 0.25
grupo de 0.125
Pesos:
2^3 2^2 2^1 2^0 *coma* 2^{-1} 2^{-2} 2^{-3}
Número (*base* = 2):
1 0 1 1 0 1 1_2
Posiciones :
3 2 1 0 – 1 – 2 – 3

▪ En un *sistema de numeración* de *base b* (*b*>1) se pueden utilizar hasta *b* *símbolos* distintos para representar cualquier *cantidad*.

La *base* de un sistema de numeración también se denomina *raíz* o *módulo*.

Tabla *T1*

Sistema de numeración	***Símbolos utilizados***
binario (b=2)	0, 1
ternario (b=3)	0, 1, 2
cuaternario (b=4)	0, 1, 2, 3
octal (b=8)	0, 1, 2, 3, 4, 5, 6, 7
decimal, arábigo-hindú (b=10)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexadecimal (b=16)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Binario	*Ternario*	*Cuaternario*	*Octal*	*Decimal*	*Hexadecimal*
0	0	0	0	0	0
1	1	1	1	1	1
10	2	2	2	2	2
11	10	3	3	3	3
100	11	10	4	4	4
101	12	11	5	5	5
110	20	12	6	6	6
111	21	13	7	7	7
1000	22	20	10	8	8
1001	100	21	11	9	9
1010	101	22	12	10	A
1011	102	23	13	11	B
1100	110	30	14	12	C
1101	111	31	15	13	D
1110	112	32	16	14	E
1111	120	33	17	15	F
10000	121	100	20	16	10
10001	122	101	21	17	11
10010	200	102	22	18	12
10011	201	103	23	19	13
10100	202	110	24	20	14

En todos los sistemas de numeración posicionales se cuenta de forma 'análoga'.

▪ La *cantidad* representada por un *número* es igual a la suma ponderada de todos los dígitos que lo forman. Cumpliéndose que la cantidad que representa un número N, en un sistema de numeración de base b:

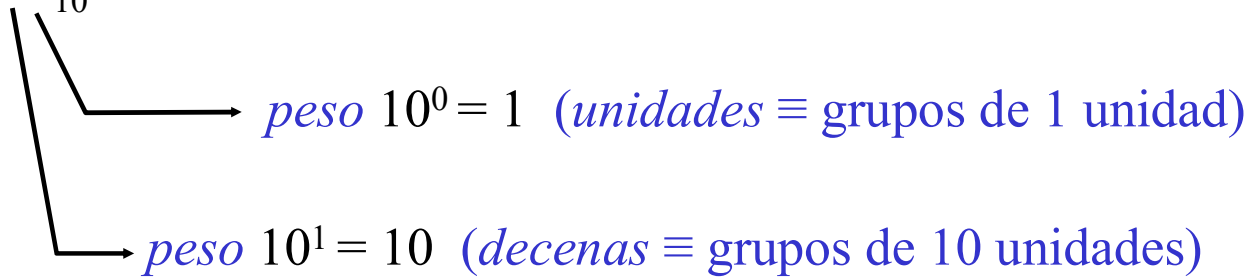
$$N_b : \overset{MSB}{\downarrow}\ \underbrace{a_{n-1}\ a_{n-2}\ \cdots\ a_1\ a_0}_{parte\ entera}\ \overset{coma}{\downarrow}\ ,\ \underbrace{a_{-1}\ a_{-2}\ \cdots\ a_{-(m-1)}\ \overset{LSB}{\downarrow}\ a_{-m}}_{parte\ fraccionaria}$$

está determinada por el siguiente *polinomio equivalente*:

$$N_b = \sum_{i=-m}^{i=n-1} a_i b^i = \underbrace{a_{n-1}b^{n-1} + \cdots + a_1 b^1 + a_0 b^0}_{parte\ entera} + \underbrace{a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots + a_{-m}b^{-m}}_{parte\ \ fraccionaria}$$

Ejemplos:

$87_{10} = 8 \cdot 10^1 + 7 \cdot 10^0 = 8 \cdot 10 + 7 = 80 + 7$



$2405'607_{10} = 2 \cdot 10^3 + 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 + 6 \cdot 10^{-1} + 0 \cdot 10^{-2} + 7 \cdot 10^{-3}$

$1011'01_2 = 1_2 \cdot (10_2)^{11_2} + 0_2 \cdot (10_2)^{10_2} + 1_2 \cdot (10_2)^{1_2} + 1_2 \cdot (10_2)^{0_2} + 0_2 \cdot (10_2)^{-1_2} + 1_2 \cdot (10_2)^{-10_2}$

$8C'A_{16} = 8_{16} \cdot (10_{16})^{1_{16}} + C_{16} \cdot (10_{16})^{0_{16}} + A_{16} \cdot (10_{16})^{-1_{16}}$

▪ Si las operaciones indicadas en el *polinomio equivalente* de un número N_b se realizan en un sistema de base b_1 ($b_1 \neq b$) , entones lo que se obtiene es la representación equivalente de dicho número en el sistema de base b_1.

Ejemplo:

$$\overset{8\ \ 2\ 1}{1011_2} = 1_2 \cdot (10_2)^{11_2} + 0_2 \cdot (10_2)^{10_2} + 1_2 \cdot (10_2)^{1_2} + 1_2 \cdot (10_2)^{0_2} =$$

$$= 1_{10} \cdot (2_{10})^{3_{10}} + 0_{10} \cdot (2_{10})^{2_{10}} + 1_{10} \cdot (2_{10})^{1_{10}} + 1_{10} \cdot (2_{10})^{0_{10}} =$$

$$= 1_{10} \cdot 8_{10} + 0_{10} \cdot 4_{10} + 1_{10} \cdot 2_{10} + 1_{10} \cdot 1_{10} = 8 + 0 + 2 + 1 = 11_{10}$$

Propiedad: en el caso particular de cantidades representadas en binario, la representación equivalente en base 10 se puede obtener sumando los pesos en base 10 asociados a los dígitos que valen 1 en el número en binario (ver ejemplo anterior: 8 + 2 + 1 = 11).

$$2F'4_{16} = 2_{10} \cdot (16_{10})^{1_{10}} + 15_{10} \cdot (16_{10})^{0_{10}} + 4_{10} \cdot (16_{10})^{-1_{10}} = 47'25_{10}$$

$$75'32_8 = 7 \cdot (10_8)^{1_8} + 5 \cdot (10_8)^{0_8} + 3 \cdot (10_8)^{-1_8} + 2 \cdot (10_8)^{-2_8} =$$

$$= 7 \cdot 8^1 + 5 \cdot 8^0 + 3 \cdot 8^{-1} + 2 \cdot 8^{-2} = 61'40625_{10}$$

las cantidades que aparecen en el polinomio se han pasado a base 10

▪ Del *polinomio equivalente* de un número se deduce que la mayor cantidad que se puede representar en un sistema de numeración de base b, utilizando n dígitos, es igual a:

$$b^n - 1$$

De acuerdo con el resultado anterior, con n dígitos *binarios* se pueden representar valores entre 0 y $2^n - 1$, cumpliéndose que:

_ con $n = 3$ se pueden representar valores entre 0 y $7 = 2^3 - 1 = 8 - 1$

_ con $n = 4$ se pueden representar valores entre 0 y $15 = 2^4 - 1$

_ con $n = 5$ se pueden representar valores entre 0 y $31 = 2^5 - 1$

Nota: la suma S_n de n términos consecutivos de una progresión geométrica de razón b es igual a:

$$S_n = b^{n-1} + b^{n-2} + \cdots + b^2 + b + 1 \overset{(a_1=1)}{=} a_1 \frac{(b^n - 1)}{b - 1}$$

Nota: el mayor valor que se puede representar con 1 dígito en un sistema de numeración de base b es igual a $b - 1$

▪ El número mínimo (n) de dígitos que se necesitan para representar en *binario* una cantidad N_{10} cumple que: $2^n - 1 \geq N_{10}$. De esta condición se deduce que n es el menor valor entero que cumple lo siguiente:

$$n \geq \log_2(N_{10}+1) = \left\{ \frac{\log_{10}(N_{10}+1)}{\log_{10} 2} \right\}$$

Ejemplo: $N = 325_{10} \Rightarrow n = \textit{entero mayor o igual que} \left\{ \frac{\log_{10} 326}{\log_{10} 2} = 8.3487 \right\}$

de acuerdo con el resultado anterior, el número mínimo de dígitos que se necesita para representar 325_{10} en binario es $n = 9$ $\quad (325_{10} = 101000101_2)$

Nota: $111111111_2 \equiv 511_{10}$

- Cuanto mayor sea la base *b* de un sistema de numeración, *en general*, menor será el número *n* de dígitos que se necesitan para representar una cantidad dada en dicho sistema.

Ejemplo: $1111111111111111_2 = 177777_8 = 65535_{10} = FFFF_{16}$

(16) (6) (5) (4)

- Cuanto menor es la base *b* de un sistema de numeración, más simples son las reglas que rigen las operaciones aritméticas en dicho sistema.

- Los sistemas de numeración más utilizados son:

Binario ($b = 2$): lo utilizan los *sistemas digitales*.

Decimal ($b = 10$): lo utilizamos las *personas*

Octal ($b = 8$) y *Hexadecimal* ($b = 16$): los utilizamos las *personas* para representar cantidades cuya representación en el sistema binario requiere el uso de muchos dígitos. La utilización de estos sistemas de numeración se debe a la facilidad de conversión entre el sistema binario y estos sistemas y a que su base es mayor que la del sistema binario.

Ejemplo:

$100100011101010101101001'0101001010101011_2 = 91D569'52AB_{16}$

para una persona es mucho más fácil leer y escribir (sin equivocarse) el número en hexadecimal que el número en binario.

Sistema binario (o binario natural):

- Introducido por *Leibniz* en el siglo XVII

- Es el sistema de numeración posicional de base 2. Lo que hace que en este sistema de numeración sólo se utilicen combinaciones de dos símbolos distintos (1 y 0) para representar cualquier cantidad.

- Es el sistema de numeración en el que operan todos los *sistemas digitales*. Los motivos de que esto sea así son tanto *tecnológicos* como *económicos**

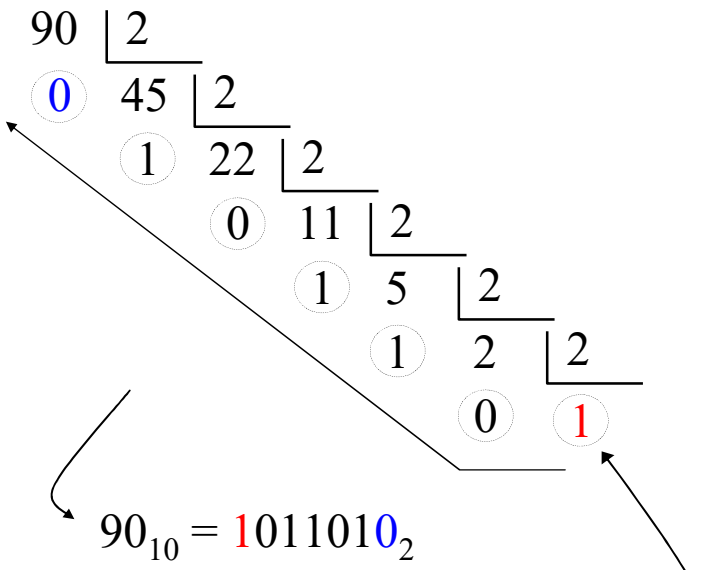
*Nota: tal como se mencionó en la primera clase, los sistemas digitales operan con señales binarias continuas en el tiempo, no con números. Lo que ocurre es que si al valor más alto de las señales binarias le asignamos como nombre el símbolo 1 y al nivel más bajo de tensión le asignamos como nombre el símbolo 0, el valor que tiene un grupo de señales binarias en un instante de tiempo dado se puede interpretar como un número binario. De ahí que coloquialmente se diga que los sistemas digitales operan con números binarios.

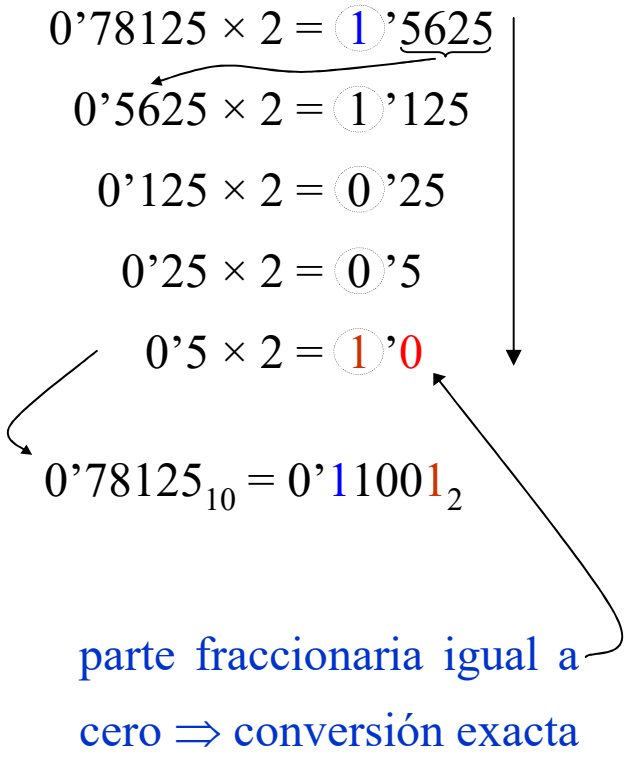
▪ Conversión de *decimal* a *binario*: la conversión de la parte entera y de la parte fraccionaria se realizan por separado.

Método 1 (teórico): para determinar la representación de la *parte entera* se realizan divisiones por 2 de forma sucesiva, hasta obtener un cociente inferior a 2. La solución está formada por los restos de las divisiones, leídos en orden inverso al que se generan y el último cociente obtenido (ver ejemplo en la siguiente diapositiva).

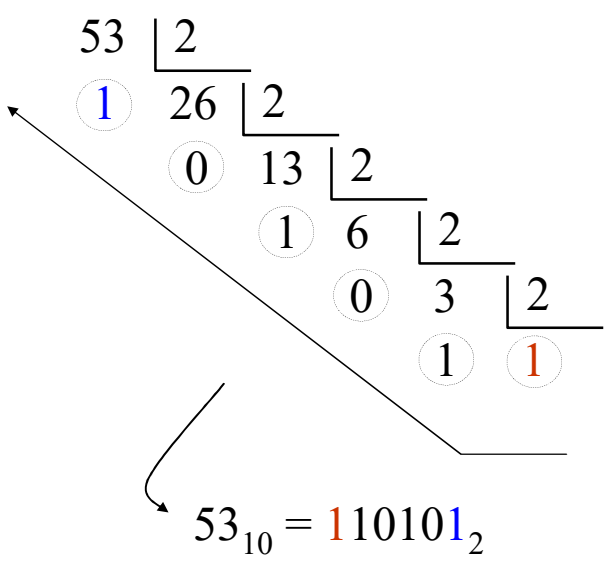
Para determinar la representación de la *parte fraccionaria* se realizan multiplicaciones por 2 de forma sucesiva hasta obtener un resultado cuya parte fraccionaria sea igual a cero (conversión exacta) o hasta determinar el número de dígitos que se desee de la solución (conversión aproximada). La representación en binario de la parte fraccionaria está formada por los valores enteros de los resultados de las distintas multiplicaciones leídos en el orden en el que se obtienen (ver ejemplo en la siguiente diapositiva).

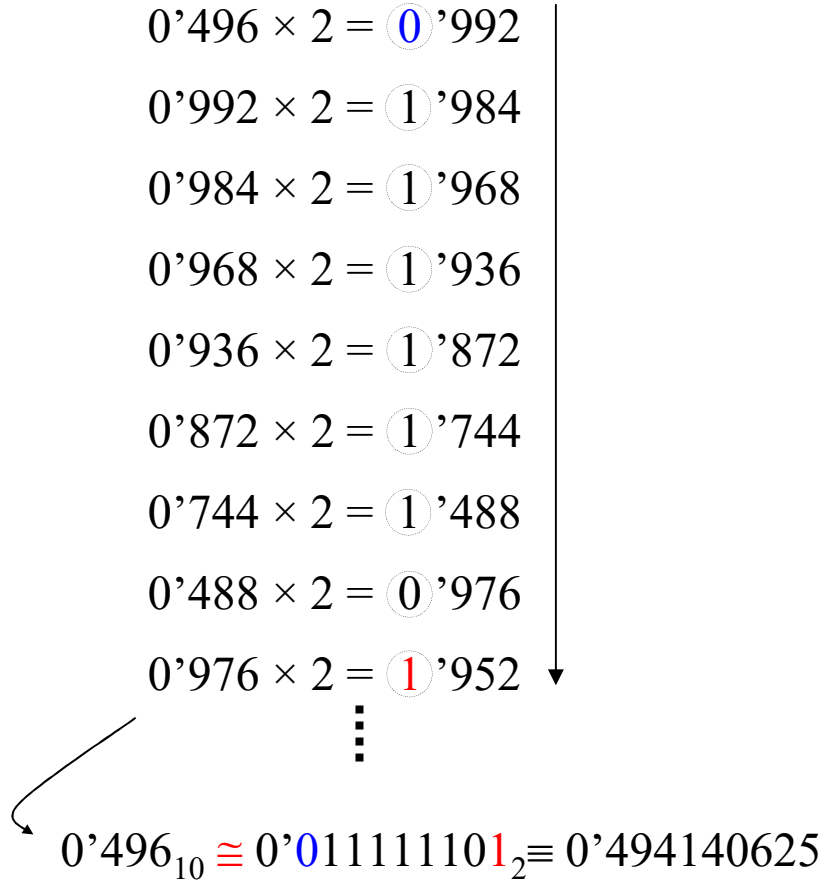
Ejemplo 1: $90'78125_{10} = 1011010'11001_2 \rightarrow$ se determina una representación exacta





Ejemplo 2: 53’496 ≅ 110101’011111101$_2$ = 53’494140625 → a continuación se obtiene una representación en binario de 53’496 con 9 dígitos de precisión en la parte fraccionaria.





Método 2 (método rápido para cantidades pequeñas): hay que determinar las potencias enteras de 2 que hay que sumar para obtener la cantidad indicada en base 10.

Ejemplos:

$$\begin{array}{rcl} & {\scriptstyle 16\ \ 8\ \ 4\ \ 2\ \ 1} & \rightarrow \textit{pesos} \text{ (potencias enteras de 2)} \\ 27 = & 1\ \ 1\ \ 0\ \ 1\ \ 1_2 & (27 = 16+8+2+1) \end{array}$$

$$\begin{array}{rcl} & {\scriptstyle 64\ \ 32\ \ 16\ \ 8\ \ 4\ \ 2\ \ 1} & \rightarrow \textit{pesos} \text{ (potencias enteras de 2)} \\ 109 = & 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1_2 & (109 = 64+32+8+4+1) \end{array}$$

$$\begin{array}{rcl} & {\scriptstyle 32\ \ 16\ \ 8\ \ 4\ \ 2\ \ 1\ \ 0.5\ \ 0.25\ \ 0.125} & \rightarrow \textit{pesos} \text{ (potencias enteras de 2)} \\ 50'625 = & 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ '\ 1\ \ 0\ \ 1_2 & (50'625 = 32+16+2+0'5+0'125) \end{array}$$

▪ Conversión de *binario* a *decimal*: se utiliza el *polinomio característico* ≡ equivale a sumar los pesos de los dígitos que valen 1 en el número en binario

Ejemplos:

$$1011_2 = 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 8+0+2+1 = 11_{10}$$

8 4 2 1

$1\ 0\ 1\ 1_2 \rightarrow 8+2+1 = 11$

$$101011'11_2 = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} =$$

$$= 32 + 0 + 8 + 0 + 2 + 1 + 0.5 + 0.25 = 43.75_{10}$$

32 16 8 4 2 1 0.5 0.25

$1\ 0\ 1\ 0\ 1\ 1\text{‘}\ 1\ 1_2 \rightarrow 32+8+2+1+0{,}5+0{,}25 = 43.75$

Aritmética binaria: los fundamentos de la aritmética en cualquier sistema de numeración de base *b* se basan en el conocimiento de las reglas que rigen las operaciones de *suma* y de *multiplicación* de 2 dígitos.

A continuación se muestran varios ejemplos de la *suma*, *resta*, *multiplicación* y *división* de números binarios.

- Suma binaria: la suma de dos números binarios se basa en la suma de dos dígitos (binarios)

acarreo

$$\begin{array}{r} {\scriptstyle d}\\ x\,x\,x\,a_2\\ +\,x\,x\,x\,b_2\\ \hline c \end{array} \qquad \begin{array}{r} {\scriptstyle 0}\\ x\,x\,x\,0_2\\ +\,x\,x\,x\,0_2\\ \hline 0 \end{array} \qquad \begin{array}{r} {\scriptstyle 0}\\ x\,x\,x\,1_2\\ +\,x\,x\,x\,0_2\\ \hline 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 0}\\ x\,x\,x\,0_2\\ +\,x\,x\,x\,1_2\\ \hline 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\\ x\,x\,x\,1_2\\ +\,x\,x\,x\,1_2\\ \hline 0 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1}\\ 1\,0\,1\,1'0\,1_2 \equiv 11'25_{10}\\ +\quad 1\,1\,0'1\,1\,1_2 \equiv 6'875_{10}\\ \hline 1\,0\,0\,1\,0'0\,0\,1_2 \equiv 18'125_{10} \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\quad\ 1}\\ 1\,1\,1'0\,1_2 \equiv 7'25_{10}\\ +\,1\,0\,0\,1'0\,1_2 \equiv 9'25_{10}\\ \hline 1\,0\,0\,0\,0'1\,0_2 \equiv 16'50_{10} \end{array}$$

- Resta binaria: la resta de dos números binarios se basa en la resta de dos dígitos (binarios)

$$\begin{array}{r} x\,x\,x\,a_2 \\ -\,x\,x\,x\,b_2 \\ \hline \quad\quad c \end{array}$$

$a - b = c$ con acarreo d (se suma al sustraendo)

$$\begin{array}{r} x\,x\,x\,0_2 \\ -\,x\,x\,x\,0_2 \\ \hline \quad\quad 0 \end{array}$$

$$\begin{array}{r} x\,x\,x\,1_2 \\ -\,x\,x\,x\,0_2 \\ \hline \quad\quad 1 \end{array}$$

$$\begin{array}{r} x\,x\,x\,0_2 \\ -\,x\,x\,x\,1_2 \\ \hline \quad\quad 1 \end{array}$$

$$\begin{array}{r} x\,x\,x\,1_2 \\ -\,x\,x\,x\,1_2 \\ \hline \quad\quad 0 \end{array}$$

$$\begin{array}{r} 1\,0\,0\,1'0\,1_2 \equiv 9'25_{10} \\ -\;\overset{1\;1}{1}\,1\,1'0\,1_2 \equiv 7'25_{10} \\ \hline 0\,0\,1\,0'0\,0_2 \equiv 2'00_{10} \end{array}$$

$$\begin{array}{r} 1\,0\,1\,1'0\,1_2 \equiv 11'25_{10} \\ -\;\overset{1\;0\;0\;1\;1\;1}{1\;1\;0'1\;1\;1}\,1_2 \equiv 6'875_{10} \\ \hline 0\,1\,0\,0'0\,1\,1_2 \equiv 4'375_{10} \end{array}$$

Curiosidad: más adelante se verá que los sistemas digitales determinan el resultado de una resta efectuando una suma… "no realizan las restas de forma análoga a como lo hacemos las personas"

▪ Multiplicación binaria: la multiplicación de dos números binarios se basa en la multiplicación de dos dígitos (binarios)

$$\begin{array}{r} \alpha\,\alpha\,\alpha\,a_2 \\ \times\quad \alpha\,\alpha\,b_2 \\ \hline c \end{array} \qquad \begin{array}{r} \alpha\,\alpha\,\alpha\,0_2 \\ \times\quad \alpha\,\alpha\,0_2 \\ \hline 0 \end{array} \qquad \begin{array}{r} \alpha\,\alpha\,\alpha\,1_2 \\ \times\quad \alpha\,\alpha\,0_2 \\ \hline 0 \end{array} \qquad \begin{array}{r} \alpha\,\alpha\,\alpha\,0_2 \\ \times\quad \alpha\,\alpha\,1_2 \\ \hline 0 \end{array} \qquad \begin{array}{r} \alpha\,\alpha\,\alpha\,1_2 \\ \times\quad \alpha\,\alpha\,1_2 \\ \hline 1 \end{array}$$

$$\begin{array}{rl} 1\,0\,1\,1\,0_2 & \equiv 22_{10} \\ \times\quad 1\,0\,1_2 & \equiv 5_{10} \\ \hline {\scriptstyle 1}\;1\,0\,1\,1\,0 & \\ 0\,0\,0\,0\,0 & \\ 1\,0\,1\,1\,0 & \\ \hline 1\,1\,0\,1\,1\,1\,0 & \equiv 110_{10} \end{array} \qquad \begin{array}{r} 1\,0\,1\,1\,1'0\,1\,1_2 \\ \times\qquad\quad 1\,0\,0_2 \\ \hline 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,1\,1\,1\,0\,1\,1 \\ \hline 1\,0\,1\,1\,1\,0\,1'1\,0\,0_2 \end{array}$$

Nota para programadores de hardware: *multiplicar* un número binario N_2 por un número que es una potencia entera de 2 (2^{α}), equivale a desplazar α posiciones hacia la *derecha* la coma del número N_2.

- División binaria

$$
\begin{array}{l|l}
1\,1\,0\,1'1\,0\,0\,1\,1_2 & 1\,0\,1_2 \\
\cline{2-2}
1\,0\,1 & 1\,0'1\,0\,1\,1\,1_2 \\
\cline{1-1}
0\,0\,1\,1\,1 & \\
\quad 1\,0\,1 & \\
\cline{1-1}
\quad\; 1\,0\,0\,0 & \\
\quad\;\; 1\,0\,1 & \\
\cline{1-1}
\quad\;\;\; 1\,1\,1 & \\
\quad\;\; 1\,0\,1 & \\
\cline{1-1}
\quad\;\;\;\; 1\,0\,1 & \\
\quad\;\;\;\; 1\,0\,1 & \\
\cline{1-1}
\quad\;\;\;\;\;\; 0 &
\end{array}
$$

$$
\begin{array}{l|l}
1\,0\,0\,1'0\,1\,1_2 & 1\,1_2 \\
\cline{2-2}
1\,1 & 1\,1'0\,0\,1_2 \\
\cline{1-1}
0\,1\,1 & \\
\quad 1\,1 & \\
\cline{1-1}
\quad\; 0\,0\,1\,1 & \\
\quad\;\; 1\,1 & \\
\cline{1-1}
\quad\;\;\; 0 &
\end{array}
$$

$$
\begin{array}{l|l}
1\,0\,1\,1\,0\,1'0\,0_2 & 1\,0\,0_2 \\
\cline{2-2}
1\,0\,0 & 1\,0\,1\,1'0\,1_2 \\
\cline{1-1}
\quad 1\,1\,0 & \\
\quad 1\,0\,0 & \\
\cline{1-1}
\quad\;\; 1\,0\,1 & \\
\quad\;\; 1\,0\,0 & \\
\cline{1-1}
\quad\;\;\;\; 1\,0\,0 & \\
\quad\;\;\;\; 1\,0\,0 & \\
\cline{1-1}
\quad\;\;\;\;\;\; 0 &
\end{array}
$$

Nota para programadores de hardware: *dividir* un número binario N_2 por un número que es una potencia entera de 2 (2^{α}), equivale a desplazar α posiciones hacia la *izquierda* la coma del número N_2

Inconveniente del sistema binario: El hecho de que la base del *sistema binario* tenga un valor tan pequeño hace que, *en general*, la representación de una cantidad en este sistema requiera muchos dígitos. A las personas nos resulta difícil leer y escribir números en binario que tengan *muchos* dígitos, sin equivocarnos.

Ejemplo: $1101010011010101010011110101001011010100011111_2$

La *solución* que se encontró a este problema consiste en utilizar sistemas de numeración caracterizados por lo siguiente:

i) Que tengan una base mucho mayor que la del sistema binario. De este modo la representación de una cantidad requerirá, *en general*, muchos menos dígitos que en el sistema binario.

ii) Que la base de dichos sistemas de numeración sea una potencia entera de la base del sistema binario ($base = 2^{\alpha}$, $\alpha \in \mathbb{N}^*$). Se puede demostrar que el proceso de conversión entre los sistemas de numeración que cumplen esta propiedad y el sistema binario es muy sencillo.

De los sistemas de numeración que cumplen las condiciones anteriores, en su momento se eligieron los sistemas:

Octal: $b = 8 = 2^3$ (ya no se utiliza hoy en día)

Hexadecimal: $b = 16 = 2^4$ (utilizado en programación de hardware)

Nota: *en esta asignatura a los dígitos* 1 y 0 *de un número binario también los vamos a denominar bits, siendo* bit *la contracción de las palabras* binary digit (*dígito binario*)

Sistema hexadecimal:

- Es el sistema de numeración de base $b = 16$ y, por lo tanto, se pueden utilizar hasta 16 símbolos distintos para representar cualquier cantidad:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, *F*

- *En general*, el número de dígitos que tiene la representación de una cantidad en el sistema hexadecimal es 4 veces menor que el número de dígitos que tiene la representación de la misma cantidad en el sistema binario.

  Ejemplo: $1010111100110100'110110111001_2 = AF34'DB9_{16}$

- En la literatura técnica se pueden ver diferentes formas de indicar que una cantidad está representada en el sistema hexadecimal.

  Ejemplo: $4A_{16}$ = 0x4A = 4AH = 4Ah = x“4A”

▪ Conversión de *binario* a *hexadecimal*: en el número binario se hacen grupos de 4 dígitos a partir de la coma (o punto) y se sustituye cada grupo de 4 dígitos por el símbolo (valor) hexadecimal equivalente.

Ejemplo: $\underbrace{11}_{3}\underbrace{1000}_{8}\underbrace{1111}_{F}\underbrace{0111}_{7}\underbrace{0000}_{0}$'$\underbrace{0101}_{5}\underbrace{1110}_{E}\underbrace{1010}_{A}\underbrace{0001}_{1}\underbrace{11}_{C}{}_2 = 38F70'5EA1C_{16}$

▪ Conversión de *hexadecimal* a *binario*: cada dígito del número hexadecimal se sustituye por el número binario de 4 dígitos que representa el mismo valor (cantidad)

Ejemplo: $9F60E'D4CA_{16} = \underbrace{1001}_{9}\underbrace{1111}_{F}\underbrace{0110}_{6}\underbrace{0000}_{0}\underbrace{1110}_{E}$'$\underbrace{1101}_{D}\underbrace{0100}_{4}\underbrace{1100}_{C}\underbrace{1010}_{A}{}_2$

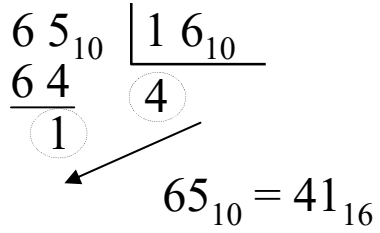
▪ Conversión de *hexadecimal* a *decimal*: se utiliza el polinomio equivalente.

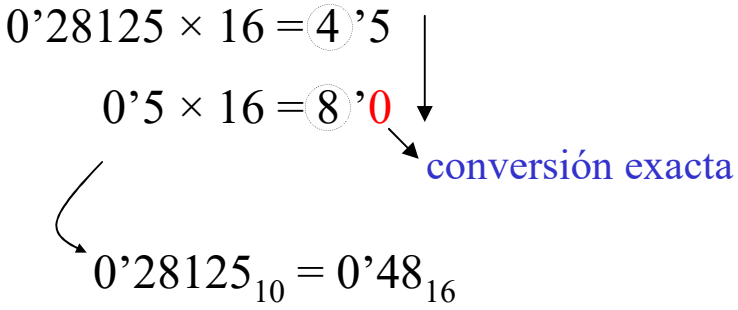
Ejemplo: $F4E0'3C9A_{16} = 15\cdot16^3 + 4\cdot16^2 + 14\cdot16^1 + 0\cdot16^0 + 3\cdot16^{-1} + 12\cdot16^{-2} + 9\cdot16^{-3} + 10\cdot16^{-4} \cong 62688'236724853_{10}$

▪ Conversión de *decimal* a *hexadecimal*: la conversión de la parte entera y de la parte fraccionaria se realizan por separado. Para convertir la parte entera se realizan divisiones sucesivas por 16 y para convertir la parte fraccionaria se realizan multiplicaciones sucesivas por 16.

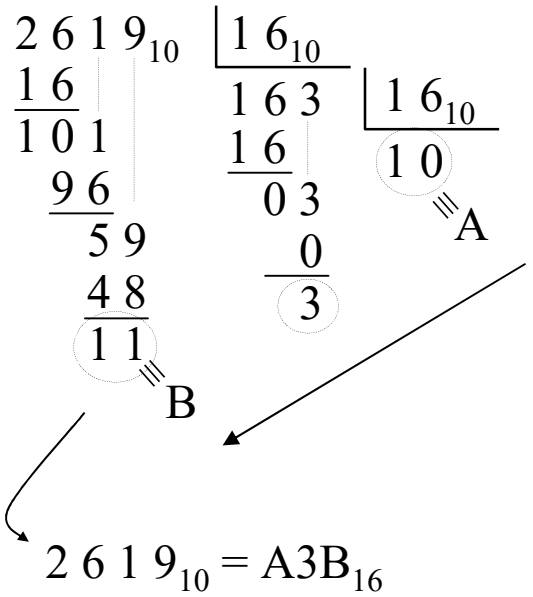
Nota: 16 es una potencia entera de 2, pero no es una potencia entera de 10 !!!

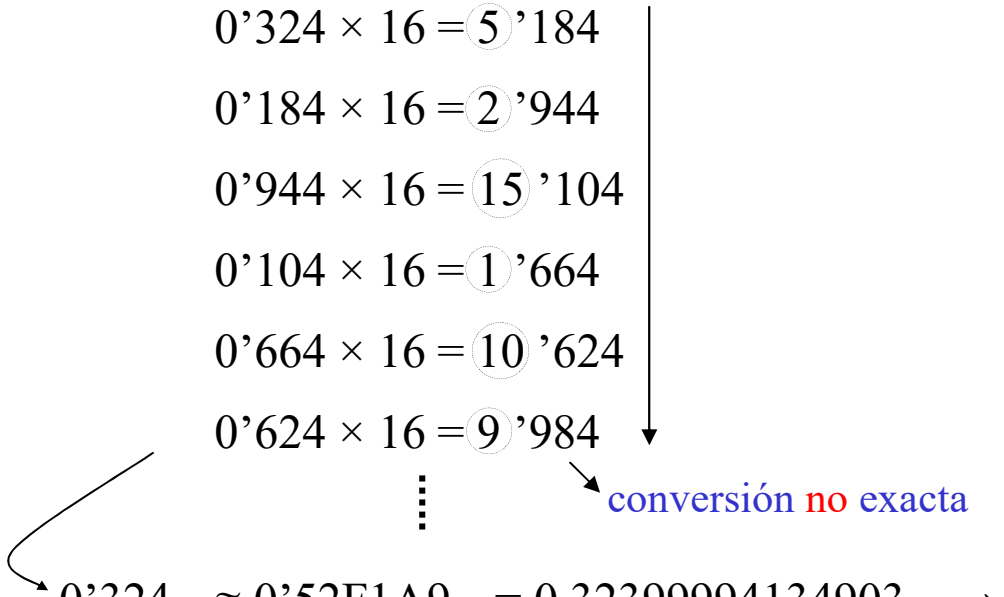
Ejemplo 1: $65'28125_{10} = 41'48_{16}$





Ejemplo 2: $2619'324_{10} \cong A3B'52F1A9_{16}$ → se obtiene una representación en hexadecimal con 6 dígitos de precisión en la parte fraccionaria.





Formatos binarios de representación de números reales con signo:

Antes de ver códigos, vamos a ver cómo se representan cantidades enteras *positivas* y *negativas*, teniendo en cuenta que los sistemas digitales sólo trabajan con dos niveles de tensión.

1 lógico → 5v. ≡ nivel alto de tensión
0 lógico → 0v. ≡ nivel bajo de tensión
} ≡ *lógica positiva*

Los formatos (códigos) de representación de cantidades con signo se pueden clasificar de la siguiente manera:

- *Formatos binarios de representación de cantidades con signo*
  - *Formatos de coma fija*
    - *Código signo-magnitud* (*csm*)
    - *Código complemento a* 1 (*cca*1)
    - *Código complemento a* 2 (*cca*2)
  - *Formatos de coma flotante*

La diferencia entre los distintos formatos (códigos) de representación reside en cómo representan la *magnitud* (*módulo* o *valor absoluto*) de las cantidades.

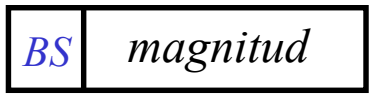
Para representar una cantidad y su signo, todos los formatos utilizan la misma estructura:

$B.S.$	*magnitud*

cumpliéndose que:

$$\left.\begin{array}{l} B.S. = 0 \equiv \text{número } \textit{positivo} \\ B.S. = 1 \equiv \text{número } \textit{negativo} \end{array}\right\} \rightarrow \text{aplicable a } \textit{todos} \text{ los formatos}$$

Formatos de coma fija (implícita):



Código Signo-Magnitud (*csm*):

- Es un *código binario* para representar *cantidades o magnitudes con signo*.
- El *bit de signo* está en la posición del dígito más significativo (MSB). El resto de los bits representan el *módulo* o *valor absoluto* de la cantidad.
- El *módulo* o *magnitud* de una cantidad se representa en *binario natural*. El valor del *bit de signo* (*BS*) de las cantidades *positivas* es 0 y el de las cantidades *negativas* es 1.
- El cero tiene dos representaciones distintas.
- El rango de valores que se pueden representar con *n* dígitos, incluido el bit de signo, es el siguiente:

$$-\left(2^{n-1}-1\right) \text{ hasta } \left(2^{n-1}-1\right)$$

	csm (4 *bits*)
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
– 0	1000
– 1	1001
– 2	1010
– 3	1011
– 4	1100
– 5	1101
– 6	1110
– 7	1111
–8	- - - -

ANEXO: Aritmética complementaria

- Las siguientes definiciones son válidas para números binarios sin signo
- El resultado siempre debe tener el mismo número de bits que el dato de partida ****
- Complemento a 1 de un número binario de $\alpha + \beta$ bits: $\underbrace{x\,x\,x\,\cdots\,x\,x}_{\alpha\ d\acute{i}gitos\ (bits)}\,'\,\underbrace{x\,x\,\cdots\,x\,x}_{\beta\ d\acute{i}gitos\ (bits)}{}_2$

Dado un número binario N_2 (sin signo) de α dígitos en la parte entera y β dígitos en la parte fraccionaria, se define el complemento a 1 ($ca1$) de N_2 de la siguiente forma:

$$ca1(N_2) = 2^{\alpha} - 2^{-\beta} - N_2$$

Nota 1: para $\beta = 0$ se cumple que $ca1(N_2) = 2^{\alpha} - 1 - N_2$

Nota 2: *si tienes una expresión en la que aparecen cantidades representadas en diferentes sistemas de numeración, para poder realizar los cálculos primero tienes que representar todas las cantidades en el mismo sistema de numeración (uno en el que te resulte fácil realizar los cálculos)*

Nota 3: *En una ALU (Arithmetic-Logic Unit) de n bits tanto los operandos como el resultado se representan siempre con n dígitos.*

Métodos de cálculo del complemento a 1 *de un número binario*:

1º método: Aplicando la definición anterior

$N = 1110.01_2 = 14.25_{10} \qquad (\alpha = 4\ y\ \beta = 2)$

$ca1(N) = 2^4 - 2^{-2} - N = 16_{10} - 0.25_{10} - 14.25_{10} = 1.5_{10} = 0001.10$

$N = 1011_2 = 11_{10} \qquad (\alpha = 4\ y\ \beta = 0)$

$ca1(N) = 2^4 - 2^{-0} - N = 16_{10} - 1_{10} - 11_{10} = 4_{10} = 0100$

$N = 000_2 = 0_{10} \qquad (\alpha = 3\ y\ \beta = 0)$

$ca1(N) = 2^3 - 2^{-0} - N = 8_{10} - 1_{10} - 0_{10} = 7_{10} = 111$

2º método: cambiando los 1_s por 0_s y viceversa

$$N = 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ .\ 1\ \ 0\ \ 1\ \ 1_2$$

$$ca1(N) = 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ .\ 0\ \ 1\ \ 0\ \ 0_2$$

Propiedades del $ca1$:

i) $M_2 = 2^{\alpha} - 2^{-\beta} - N_2 \quad \Rightarrow \quad N_2 = 2^{\alpha} - 2^{-\beta} - M_2 = ca1(M_2)$

es decir, si $M = ca1(N) \ \Rightarrow \ N = ca1(M)$

ii) $N_2 + ca1(N_2) = 2^{\alpha} - 2^{-\beta} = 11\overset{\alpha}{\cdots}1'11\overset{\beta}{\cdots}1$ ($\alpha + \beta$ *dígitos*)

- Complemento a 2 de un número binario de $\alpha + \beta$ bits: $\underbrace{x\,x\,x\cdots x\,x}_{\alpha \text{ dígitos (bits)}}\,'\,\underbrace{x\,x\cdots x\,x}_{\beta \text{ dígitos (bits)}}{}_2$

Dado un número binario N_2 (sin signo) de α dígitos en la parte entera y β dígitos en la parte fraccionaria, se define el complemento a 2 ($ca2$) de N_2 de la siguiente forma:

$$ca2\left(N_2\right) = 2^{\alpha} - N_2 = M_2$$

Recordatorio: el resultado (M_2) siempre debe tener el mismo número de dígitos que el número (dato) de partida (N_2).

Nota: *si tienes una expresión en la que aparecen cantidades representadas en diferentes sistemas de numeración, para poder realizar los cálculos primero tienes que representar todas las cantidades en el mismo sistema de numeración (uno en el que te resulte fácil realizar los cálculos)*

Métodos de cálculo del complemento a 2 de un número binario:

1º *método*: Aplicando la definición anterior

$$N = 1110.01_2 = 14.25_{10} \qquad (\alpha = 4)$$

$$ca2(N_2) = 2^4 - N_2 = 16_{10} - 14.25_{10} = 1.75_{10} = 0001.11_2$$

$$N = 1011_2 = 11_{10} \qquad (\alpha = 4)$$

$$ca2(N_2) = 2^4 - N_2 = 16_{10} - 11_{10} = 5_{10} = 0101_2$$

$$N = 000_2 = 0_{10} \qquad (\alpha = 3)$$

$$ca2(N_2) = 2^3 - 0 = 8_{10} = 1000_2 \quad \rightarrow \quad ca2(000) = 000_2$$

el resultado siempre debe tener el mismo número de dígitos que el número de partida.

2º *método*: cálculo del *ca*2 en función del *ca*1 (útil para diseño)

$$ca2(N_2) = 2^{\alpha} - N_2 = 2^{\alpha} - N_2 + 2^{-\beta} - 2^{-\beta} = \left[2^{\alpha} - 2^{-\beta} - N_2\right] + 2^{-\beta} = ca1(N_2) + 2^{-\beta}$$

Nota: del resultado anterior se deduce que, en el caso de que N_2 sea un número entero ($\beta = 0$), se cumple lo siguiente:

$$ca2(N_2) = 2^{\alpha} - N_2 = ca1(N_2) + 1$$

Ejemplo: $N = 14.25_{10} = 1110.01_2$

$$ca1(N) = 0001.10$$

$$ca2(N) = ca1(N) + 2^{-2} = 0001.10_2 + 0.01_2 = 0001.11$$

3º *método*: para obtener el *ca2* de un numero *N* se escriben los mismos dígitos que tiene el número *N*, empezando por el dígito menos significativo, hasta escribir el primer 1. A partir de dicho dígito, los demás dígitos del resultado se obtienen cambiando los 1s que aparezcan en el número *N* por 0s y viceversa.

Ejemplos:

$N = 1\ 0\ 1\ 1\ 0.\ 0\ 1\ 0\ 0_2$

$ca2(N) = 0\ 1\ 0\ 0\ 1.\ 1\ 1\ 0\ 0_2$

se ha escrito el primer 1

$N = 0\ 0\ 0.\ 0\ 0_2$

$ca2(N) = 0\ 0\ 0.\ 0\ 0_2$

$$N_2 = 1\,0\,1\,1\,0\,1\,1\,1\,0\,0\,1\,0.\,0\,1\,1\,1\,0\,0\,1_2$$

$$ca2(N_2) = 0\,1\,0\,0\,1\,0\,0\,0\,1\,1\,0\,1.\,1\,0\,0\,0\,1\,1\,1_2$$

se ha escrito el primer 1

Propiedades del *ca2*:

i) $M_2 = 2^\alpha - N_2 \quad \Rightarrow \quad N_2 = 2^\alpha - M_2 = ca2(M_2)$

es decir, si $M_2 = ca2(N_2) \;\Rightarrow\; N_2 = ca2(M_2)$

ii) $N_2 + ca2(N_2) = 2^\alpha = 1\overbrace{0\cdots0}^{\alpha}{}'\overbrace{0\cdots0}^{\beta}$

si se limita la representación de $N_2 + ca2(N_2)$ a $\alpha+\beta$ bits, entonces se cumple que $N_2 + ca2(N_2) = \underbrace{0\cdots0}_{\alpha}{}'\underbrace{0\cdots0}_{\beta}$

Fin Anexo: Aritmética complementaria

(continuación de los formatos de representación de cantidades con signo)

Código complemento a 1 (*cca*1)

Propiedades:

- Es un *código binario* para representar *cantidades con signo*.
- El *bit de signo* se sitúa en la posición del dígito más significativo (MSB). El resto de los bits representan el *módulo* o *valor absoluto* de la cantidad.
- Las *cantidades positivas* tienen la misma representación que en el *código signo-magnitud*
- La representación de una *cantidad negativa* se obtiene como el *complemento a* 1 de la representación de la misma cantidad *positiva* (bit de signo incluido). De hecho, se cumple que calcular el *ca*1 de una cantidad representada en el *código complemento a* 1 (*cca*1) proporciona la representación en el *cca*1 de la misma cantidad cambiada de signo

$$A^{-}_{cca1} = ca1\left(A^{+}_{cca1}\right) = 2^{\alpha} - 2^{-\beta} - A^{+}_{cca1} \quad \Rightarrow \quad A^{+}_{cca1} = 2^{\alpha} - 2^{-\beta} - A^{-}_{cca1} = ca1\left(A^{-}_{cca1}\right)$$

Nomenclatura: A^{+}_{cca1} es la representación de una magnitud positiva A en el *cca*1.

A^{-}_{cca1} es la representación de una magnitud negativa A en el *cca*1.

Ejemplos:

$$+7_{10} \equiv \overset{BS}{0}\,111_{cca1} \equiv \overset{BS}{0}\,111_{csm}$$

$$-7_{10} \equiv ca1(\overset{BS}{0}\,111) = \overset{BS}{1}\,000_{cca1}$$

El *BS* se trata como si fuese un dígito perteneciente al módulo

$$+0_{10} \equiv \overset{BS}{0}\,0000_{cca1}$$

$$-0_{10} \equiv ca1(\overset{BS}{0}\,0000) = \overset{BS}{1}\,1111_{cca1}$$

El cero tiene dos representaciones distintas para un número de dígitos dado !!!.

· El rango de cantidades enteras que se pueden representar con *n* dígitos, incluido el bit de signo, es:

$$-\left(2^{n-1}-1\right) \text{ hasta } \left(2^{n-1}-1\right)$$

· Se cumple que: $N_{cca1}+\left(-N\right)_{cca1}=111\overset{n)}{\cdots}1\neq 0$

↑ representa el – 0 en el *cca*1

· En la práctica este convenio da lugar a circuitos más complicados que los que requiere el *cca*2.

	csm (4 *bits*)	*cca*1(4 *bits*)
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
– 0	1000	1111
– 1	1001	1110
– 2	1010	1101
– 3	1011	1100
– 4	1100	1011
– 5	1101	1010
– 6	1110	1001
– 7	1111	1000
–8	- - - -	- - - -

Código complemento a 2 (*cca*2)

Propiedades:

· Es un *código binario* para representar *cantidades con signo*.

· El *bit de signo* se sitúa en la posición del dígito más significativo (MSB). El resto de los bits representan el *módulo* o *valor absoluto* de la cantidad

· Las *cantidades positivas* tienen la misma representación en el *código complemento a* 2 (*cca*2) que en el *csm* y que en el *cca*1

· La representación de una *cantidad negativa* se obtiene como el *complemento a* 2 de la representación de la misma cantidad *positiva* (bit de signo incluido). De hecho, se cumple que el cálculo del *ca*2 de una cantidad representada en el *código complemento a* 2 (*cca*2) proporciona la representación en el *cca*2 de la misma cantidad cambiada de signo.

$$A^-_{cca\,2} = ca2\left(A^+_{cca\,2}\right) = 2^n - A^+_{cca\,2} \quad \Rightarrow \quad A^+_{cca\,2} = 2^n - A^-_{cca\,2} = ca2\left(A^-_{cca\,2}\right)$$

Nomenclatura: $A^+_{cca\,2}$ es la representación de una magnitud positiva A en el *cca*2.

$A^-_{cca\,2}$ es la representación de una magnitud negativa A en el *cca*2.

Ejemplo:

$$A = +7_{10} \overset{BS}{\equiv} 0\,0111_{cca2}$$

$$ca2(A) = ca2(\overset{+7_{10}}{\overset{|||}{00111}}) = \overset{BS}{1\,1001_{cca2}} \equiv -7_{10} = -A$$

$$ca2(-A) = ca2(\overset{-7_{10}}{\overset{|||}{11001}}) = \overset{BS}{0\,0111_{cca2}} \equiv +7_{10} = A$$

De la propiedad anterior se deduce que calcular dos veces seguidas el *ca2* de una cantidad equivale a no hacer nada.

Ejemplo:

$$ca2(+9) = ca2(\overset{BS}{0\,1001}) = \overset{BS}{1\,0111_{cca2}} \equiv -9_{10}$$

$$ca2(-9) = ca2(\overset{BS}{1\,0111}) = \overset{BS}{0\,1001_{cca2}} \equiv +9_{10}$$

· El cero tiene una representación única en el *cca2*:

$$+0_{10} \equiv \overset{BS}{0\ 0000}_{cca2}$$

$$-0_{10} \equiv ca2(\overset{BS}{0\ 0000}) = \overset{BS}{0\ 0000}_{cca2}$$

· Se cumple que:

$$N_{cca2} + (-N)_{cca2} = 100 \cdot \overset{(\alpha+\beta)}{\cdots} \cdot 0$$ → un 1 seguido de $\alpha+\beta$ ceros, siendo N un número con α dígitos en la parte entera y β dígitos en la parte fraccionaria. Si el resultado de la suma anterior de limita a $\alpha+\beta$ dígitos, entonces lo que se obtiene son $\alpha+\beta$ ceros.

· El rango de cantidades enteras que se pueden representar con n dígitos, incluido el bit de signo, es:****

$$-\left(2^{n-1}\right) \text{ hasta } +\left(2^{n-1}-1\right)$$

**** Nota: de acuerdo con la definición del código *cca*2, con n bits se pueden representar las cantidades comprendidas entre $-(2^{n-1}-1)$ y $+2^{n-1}-1$. Pero debido a que en este código el +0 y el –0 tienen una misma representación, queda sin utilizar uno de los 2^n números que se pueden escribir en binario con n dígitos. El cual, matemáticamente corresponde a la representación en el *cca*2 de -2^{n-1}, siendo éste el motivo por el que a dicha combinación se le asigna la representación de -2^{n-1} (a pesar de que no se ajusta a la definición del *cca*2).

	csm (4 *bits*)	*cca*1(4 *bits*)	*cca*2(4 *bits*)
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
– 0	1000	1111	0000
– 1	1001	1110	1111
– 2	1010	1101	1110
– 3	1011	1100	1101
– 4	1100	1011	1100
– 5	1101	1010	1011
– 6	1110	1001	1010
– 7	1111	1000	1001
–8	- - - -	- - - -	1000

excepción con $n = 4$ bits

Nota: con 4 dígitos se pueden escribir 16 números distintos. En el caso del *cca*2, de acuerdo con la definición de dicho código, con 4 dígitos se pueden representar cantidades enteras comprendidas entre +7 y –7, con una misma representación para el +0 y el –0. Lo que hace que quede sin utilizar un número de los 16 que se pueden escribir con 4 dígitos binarios. Dicho número es el 1000, el cual no cumple la definición del código, pero matemáticamente se cumple que representa el valor –8.

Pregunta: con 5 bits, incluido el bit de signo, ¿qué rangos de valores se pueden representar en los códigos *csm*, *cca*1 y *cca*2?. ¿Se puede representar el –8 en el *cca*2 con 5 bits? ¿y el –16?

NO

Código *offset binary* (aún se utiliza en algunos sistemas)

Propiedades:

· Es un *código binario* para representar *cantidades con signo*

· No sigue el formato $bs = 1 \Leftrightarrow$ negativo y $bs = 0 \Leftrightarrow$ positivo

· La representación de una cantidad N_{10} utilizando n bits se puede determinar como

$$Offset\ binary = 2^{n-1} + N_{10}$$

Así, por ejemplo, la representación de –5 utilizando $n = 4$ bits se obtiene como:

$$2^{4-1} + (-5) = 8 - 5 = 3 = 0011_{offset\ binary}$$

y la representación de +6 utilizando $n = 4$ bits se obtiene como:

$$2^{4-1} + 6 = 8 + 6 = 14 = 1110_{offset\ binary}$$

· *Método de cálculo rápido*: la representación de una cantidad en el código *offset binary* se puede obtener a partir de la representación en el *cca*2 invirtiendo el bit de signo

NO

	csm (4 *bits*)	*cca*1(4 *bits*)	*cca*2(4 *bits*)	*Offset binary*
+7	0111	0111	0111	1111
+6	0110	0110	0110	1110
+5	0101	0101	0101	1101
+4	0100	0100	0100	1100
+3	0011	0011	0011	1011
+2	0010	0010	0010	1010
+1	0001	0001	0001	1001
+0	0000	0000	0000	1000
– 0	1000	1111	0000	1000
– 1	1001	1110	1111	0111
– 2	1010	1101	1110	0110
– 3	1011	1100	1101	0101
– 4	1100	1011	1100	0100
– 5	1101	1010	1011	0011
– 6	1110	1001	1010	0010
– 7	1111	1000	1001	0001
–8	- - - -	- - - -	1000	0000

$\overline{a}_4\ a_3\ a_2\ a_1\ a_0$ *offset binary*

$a_4\ a_3\ a_2\ a_1\ a_0$ *cca2*

excepción con $n = 4$ bits

Aritmética en el *cca*2:

_ En los sistemas digitales, las operaciones con números enteros o con números que no son ni muy grandes ni muy pequeños (próximos a cero) se suelen realizar en el *cca*2.

Curiosidad: en los PCs, los valores de las variables de tipo *signed int* se guardan representados en el *cca*2. Mientras que los valores de las variables de tipo *unsigned int* se guardan representados en binario.

_ Sólo vamos a considerar las operaciones de suma/resta

_ La ventaja del *cca*2 reside en que las operaciones de suma/resta se realizan utilizando un circuito sumador (como el que se utiliza para sumar números binarios).

Nota: las ALU (*Arithmetic Logic Units*) y los circuitos sumadores de coma fija están diseñados para operar con cantidades sin signo, representadas en el sistema binario. En las siguientes páginas se analizan los resultados que proporciona un sumador o una ALU cuando opera con (suma) cantidades codificadas en el código complemento a 2 (*cca*2). Las ALUs y los circuitos sumadores está diseñados para que el resultado y los operandos tengan el mismo número de dígitos (bits).

Adición y sustracción en el *cca*2:

· Cuando se opera con números positivos y negativos el hablar de sumas y de restas deja de tener sentido.

· Con el fin de simplificar el análisis y sin que ello implique la menor pérdida de generalidad, en las siguientes diapositivas se supone que los números *A* y *B* con los que se opera son números enteros, *positivos*, representados en el *cca*2.

· Los resultados se obtienen representados en el *cca*2.

· Los operandos deben tener siempre el mismo número de dígitos. Y el resultado debe tener siempre el mismo número de dígitos que los operandos.

1º caso: $M = A_{cca2} + B_{cca2}$

$$M = A_{cca2} + B_{cca2} \overset{BS}{=} 0\ A_2 + \overset{BS}{0}\ B_2 \overset{BS}{=} 0\,(A_2 + B_2)$$

· La suma de dos o más números positivos representados en el *cca*2 proporciona el resultado correcto, representado en el *cca*2, excepto cuando se produce un *desbordamiento* (*overflow*).

Nota: se produce un *desbordamiento* cuando no se puede representar el resultado con el mismo número de dígitos que se ha utilizado para representar los sumandos. Los desbordamientos se detectan analizando los bits de signo de los sumandos y del resultado. En este caso, "*sumando dos números positivos no puede dar como resultado un número negativo*". Este problema se resuelve aumentando el número de dígitos que se utilizan para representar los sumandos y como consecuencia de ello, del resultado.

Ejemplos:

$M = A_{cca2} + B_{cca2}$, *siendo*:

$$A_{cca2} = \overset{BS}{0\,100}_{cca2} \equiv +4_{10} \quad y \quad B_{cca2} = \overset{BS}{0\,010}_{cca2} \equiv +2_{10}$$

$$\begin{array}{r} \overset{BS}{0\ 1\ 0\ 0}_{cca2} \equiv +4_{10} \\ +\ 0\ 0\ 1\ 0_{cca2} \equiv +2_{10} \\ \hline \overset{BS}{0\ 1\ 1\ 0}_{cca2} \equiv +6_{10} \end{array}$$

(*correcto*)

$M = A_{cca2} + B_{cca2}$, *siendo*:

$A_{cca2} = \overset{BS}{0}\,110_{cca2} \equiv +6_{10}$ *y* $B_{cca2} = \overset{BS}{0}\,100_{cca2} \equiv +4_{10}$

$$\begin{array}{r} \overset{BS}{\overset{1}{0}\ 1\ 1\ 0}_{cca2} \equiv +6_{10} \\ +\ 0\ 1\ 0\ 0_{cca2} \equiv +4_{10} \\ \hline \overset{BS}{1\ 0\ 1\ 0}_{cca2} \equiv \text{¿}+10_{10}\text{?} \end{array}$$

→ *desbordamiento*: *sumando dos números positivos se obtiene como resultado un número negativo.*

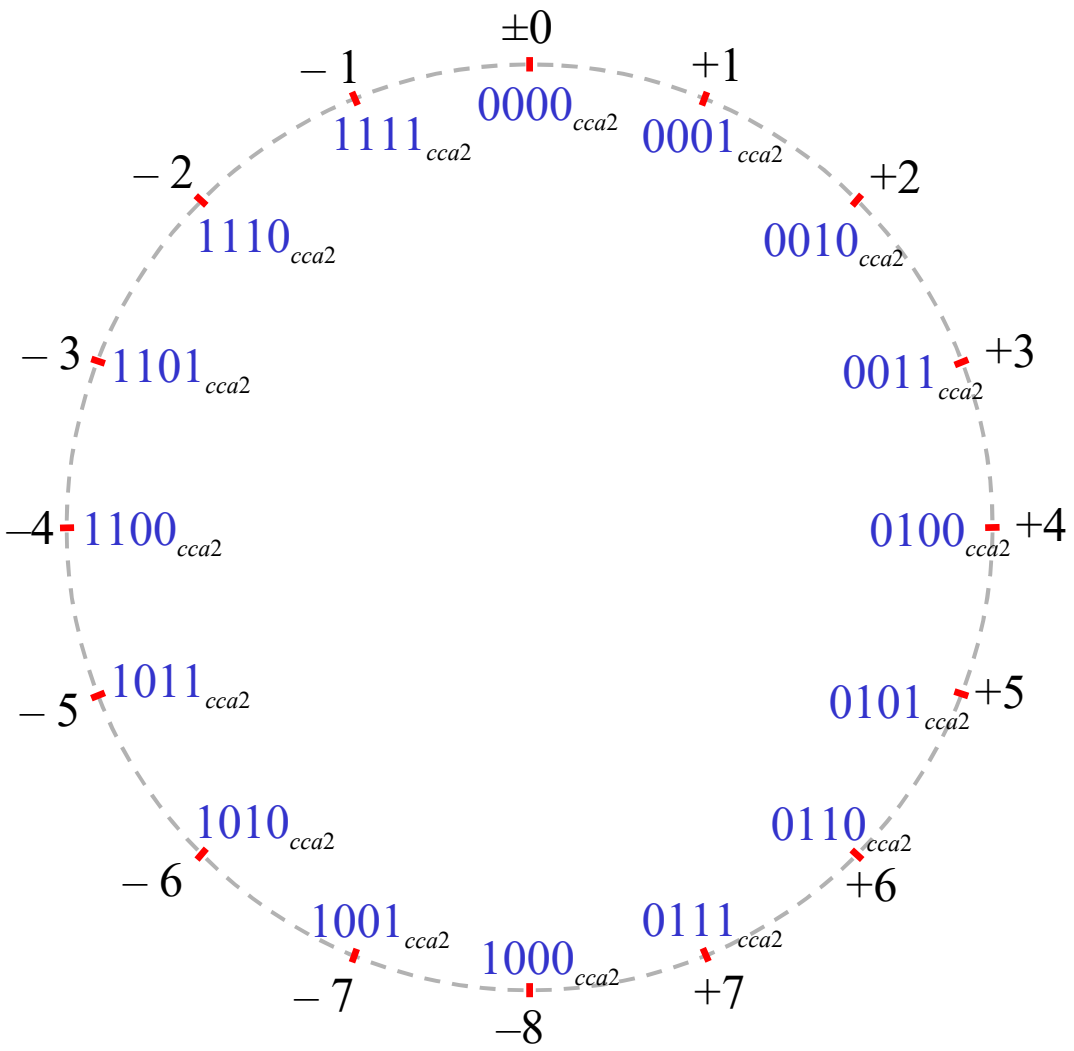
$$\begin{array}{r} \overset{BS}{0}\ \overset{1}{0}\ 1\ 1\ 0_{cca2} \equiv +6_{10} \\ +\ 0\ 0\ 1\ 0\ 0_{cca2} \equiv +4_{10} \\ \hline \overset{BS}{0\ 1\ 0\ 1\ 0}_{cca2} \equiv +10_{10} \end{array}$$

Nota para programadores: *se puede producir el desbordamiento de una variable de tipo signed* (y también de una de tipo *unsigned*!!!.)

	cca2	*valor cca2*
0	0000	±0
1	0001	+1
2	0010	+2
3	0011	+3
4	0100	+4
5	0101	+5
6	0110	+6
7	0111	+7
8	1000	–8
9	1001	–7
10	1010	–6
11	1011	–5
12	1100	–4
13	1101	–3
14	1110	–2
15	1111	–1

Ejemplo: 0110_{cca2} + 0011_{cca2} = 1001_{cca2}

+6 + 3 ¿+9 ó – 7?



2º caso: $M_{cca2} = A_{cca2} - B_{cca2}$ $\qquad A_{cca2} \overset{BS}{=} 0\ A_2\ ,\ B_{cca2} \overset{BS}{=} 0\ B_2$

Si se sustituye el signo menos por una operación de *ca*2, se puede calcular el valor de $A_{cca2} - B_{cca2}$ realizando una suma de dos números binarios:

$$A_{cca2} - B_{cca2} = A_{cca2} + C_{cca2} \quad \text{siendo} \quad C_{cca2} = -B_{cca2} = ca2\left(B_{cca2}\right)$$

es decir, $A_{cca2} - B_{cca2} = A_{cca2} + ca2\left(B_{cca2}\right)$

una resta que se convierte en una suma de números binarios

Nota:

A_{cca2} es un número binario

$-B_{cca2}=ca2(B_{cca2})$ es un número binario

$\Rightarrow A_{cca2} + ca2(B_{cca2})$ es una suma de números binarios

i) **H**: $A \geq B$

$$M_{cca2} = A_{cca2} - B_{cca2} \Rightarrow A_{cca2} + (-B_{cca2})_{cca2} = A_{cca2} + ca2(B_{cca2}) =$$

$$= A_{cca2} + 2^n - B_{cca2} = 2^n + \overbrace{0^{BS} A_2 - 0^{BS} B_2}^{\text{número positivo}} = \underbrace{2^n + M_{cca2}}$$

el resultado obtenido coincide con el valor deseado (M_{cca2}) si se desprecia el dígito n+1 procedente del término 2^n.

Ejemplo:

$$M = A_{cca2} - B_{cca2} \Rightarrow A_{cca2} + ca2\left(B_{cca2}\right), \quad siendo:$$

$$A_{cca2} = \overset{BS}{0}\,110_{cca2} \equiv +6_{10} \quad y \quad B_{cca2} = \overset{BS}{0}\,100_{cca2} \equiv +4_{10}$$

$$\begin{array}{r} \overset{BS}{\overset{1}{0}}\ 1\ 1\ 0_{cca2} \equiv +6_{10} \\ +\ 1\ 1\ 0\ 0_{cca2} \equiv -4_{10} \\ \hline \cancel{1}\ 0\ 0\ 1\ 0_{cca2} \equiv +2_{10} \end{array}$$

Pregunta: ¿En el segundo caso (resta de dos números positivos) se puede producir un desbordamiento?. ¿Por qué?

ii) **H**: $A < B$ $\quad M_{cca2} = A_{cca2} - B_{cca2} < 0$ $\quad A_{cca2} \overset{BS}{=} 0\ A_2\ ,\ B_{cca2} \overset{BS}{=} 0\ B_2$

$$M_{cca2} = A_{cca2} - B_{cca2} \Rightarrow A_{cca2} + (-B_{cca2})_{cca2} = A_{cca2} + ca2(B_{cca2}) = A_{cca2} + 2^n - B_{cca2} =$$

$$= 2^n - \underbrace{[\overset{BS}{0}B_2 - \overset{BS}{0}A_2]}_{\text{número positivo en cca2}} = \underbrace{ca2(B_{cca2} - A_{cca2})}_{\text{número negativo en cca2}} = [-(B_{cca2} - A_{cca2})]_{cca2} = M_{cca2}$$

el resultado obtenido coincide directamente con el valor deseado.

Ejemplo:

$$M = A_{cca2} - B_{cca2} \Rightarrow A_{cca2} + ca2\left(B_{cca2}\right), \quad siendo:$$

$$A_{cca2} = \overset{BS}{0}\,101_{cca2} \equiv +5_{10} \quad y \quad B_{cca2} = \overset{BS}{0}\,111_{cca2} \equiv +7_{10}$$

$$\begin{array}{r} \overset{BS}{0}\ 1\ \overset{1}{0}\ 1_{cca2} \equiv +5_{10} \\ +\ 1\ 0\ 0\ 1_{cca2} \equiv -7_{10} \\ \hline 1\ 1\ 1\ 0_{cca2} \equiv -2_{10} \end{array}$$

3º caso: $M_{cca2} = -A_{cca2} - B_{cca2}$ $\qquad A_{cca2} \overset{BS}{=} 0\ A_2\ ,\ B_{cca2} \overset{BS}{=} 0\ B_2$

Si se sustituyen los signos '–' por operaciones de *ca2*, se puede calcular el valor de $-A_{cca2} - B_{cca2}$ realizando una suma.

$$M_{cca2} = -A_{cca2} - B_{cca2} \Rightarrow (-A_{cca2}) + (-B_{cca2}) = ca2(A_{cca2}) + ca2(B_{cca2}) =$$

$$= 2^n - A_{cca2} + 2^n - B_{cca2} = 2^n + 2^n - \underbrace{[0A_2 + 0B_2]}_{\text{número positivo en cca2}} = 2^n + \underbrace{ca2(A_{cca2} + B_{cca2})}_{\text{número negativo en cca2}} =$$

$$= 2^n + M_{cca2}$$

el resultado obtenido es el buscado (M_{cca2}) si se desprecia el dígito n+1 que aparece en el resultado debido al término 2^n.

Nota: en este caso también se puede producir un desbordamiento!!! (ver comentarios 1º caso)

Ejemplos:

$$M = -A_{cca2} - B_{cca2} = \left[-(A_{cca2})\right] + \left[-(B_{cca2})\right] \Rightarrow ca2\left(A_{cca2}\right) + ca2\left(B_{cca2}\right),\ \textit{siendo}:$$

$$A_{cca2} \overset{BS}{=} 0\,011_{cca2} \equiv +3_{10} \quad y \quad B_{cca2} \overset{BS}{=} 0\,100_{cca2} \equiv +4_{10}$$

$$ca2\left(A_{cca2}\right) \overset{BS}{=} 1\,101_{cca2} \equiv -3_{10} \quad y \quad ca2\left(B_{cca2}\right) \overset{BS}{=} 1\,100_{cca2} \equiv -4_{10}$$

$$\begin{array}{rl} ca2(A_{cca2}) \equiv & \overset{BS}{1}\ 1\ 1\ 0\ 1_{cca2} \equiv -3_{10} \\ ca2(B_{cca2}) \equiv & +\ 1\ 1\ 0\ 0_{cca2} \equiv -4_{10} \\ \hline & \not{1}\ 1\ 0\ 0\ 1_{cca2} \equiv -7_{10} \end{array}$$

$M = -A_{cca2} - B_{cca2}$, *siendo*:

$$A_{cca2} = \overset{BS}{0}100_{cca2} \equiv +4_{10} \quad y \quad B_{cca2} = \overset{BS}{0}101_{cca2} \equiv +5_{10}$$

$$\begin{array}{rl} ca2(A_{cca2}) \equiv & \overset{BS}{1}\ 1\ 0\ 0_{cca2} \equiv -4_{10} \\ ca2(B_{cca2}) \equiv & +\ 1\ 0\ 1\ 1_{cca2} \equiv -5_{10} \\ \hline & \not{1}\ 0\ 1\ 1\ 1_{cca2} \end{array}$$

→ *desbordamiento*: *sumando dos números negativos se obtiene como resultado un número positivo.*

$$\begin{array}{r} \overset{BS}{\overset{1}{}} \\ 1\ 1\ 1\ 0\ 0_{cca2} \equiv -4_{10} \\ +\ 1\ 1\ 0\ 1\ 1_{cca2} \equiv -5_{10} \\ \hline \not{1}\ 1\ 0\ 1\ 1\ 1_{cca2} \end{array}$$

Resumen de los casos analizados en las diapositivas anteriores:

▪ Todas las operaciones de suma/resta de cantidades representadas en el *cca*2 se pueden realizar efectuando sumas, considerando todos los dígitos como si fuesen dígitos binarios. Para ello sólo hay que sustituir los signos menos (–) que haya por operaciones de *ca*2 y despreciar cualquier acarreo que se produzca al sumar los bits de signo.

Notas:

_ Los sumandos y el resultado *siempre* deben de tener el *mismo* número de dígitos

_ El bit de signo del resultado *siempre* debe de estar en la misma columna que el bit de signo de los sumandos.

Curiosidad para programadores: siempre que en el *lenguaje de programación C* se declara una variable de tipo *signed*, el valor de dicha variable se guarda representado en el *cca*2. Y siempre que se declara una variable de tipo *unsigned* su valor se guarda representado en binario (sistema de numeración de base $b = 2$)

- Al sumar dos números positivos o dos números negativos se puede producir un *desbordamiento* (*overflow*). El cual se detecta analizando los bit de signo de los operandos (sumandos) y del resultado [la suma de dos números *positivos* no puede dar como resultado un número *negativo* y la suma de dos números *negativos* no puede dar como resultado un número *positivo*]. La causa de que se produzca un *desbordamiento* es debido a que el resultado no se puede representar con el mismo número de dígitos que se ha utilizado para representar los sumandos. Cuando ocurre este problema se resuelve aumentando el número de dígitos que se utiliza para representar los sumandos y, como consecuencia de ello, del resultado.

- Al realizar una operación de suma/resta de dos o más cantidades codificadas en el *cca*2 el resultado que se obtiene está representado en el *cca*2.

Ejercicio: Calcular $M = A_{cca2} - B_{cca2}$, siendo $A_{cca2} = +3_{10}$ y $B_{cca2} = -7_{10}$

Errores típicos en los exámenes:

Ejercicio: Calcula $M = A_{cca2} + B_{cca2}$, siendo $A_{cca2} = +6_{10}$ y $B_{cca2} = -1_{10}$

Respuesta 1

$$A_{cca2} \overset{BS}{=} 0110_{cca2} \equiv +6_{10}$$

$$B_{cca2} \overset{BS}{=} 111_{cca2} \equiv -1_{10}$$

$$\begin{array}{r} \overset{BS}{} \\ 0\ 1\ 1\ 0_{cca2} \equiv +6_{10} \\ +\ \ 1\ 1\ 1_{cca2} \equiv -1_{10} \\ \hline 1\ 1\ 0\ 1_{cca2} \end{array}$$

¿el resultado corresponde a $+5_{10}$?

¿seguro?

Respuesta 2

$$A_{cca2} \overset{BS}{=} 0110_{cca2} \equiv +6_{10}$$

$$B_{cca2} \overset{BS}{=} 111_{cca2} \equiv -1_{10}$$

$$\begin{array}{r} \overset{BS}{0\ 1\ 1\ 0}_{cca2} \equiv +6_{10} \\ + \quad \overset{BS}{1\ 1\ 1}_{cca2} \equiv -1_{10} \\ \hline \overset{BS}{0\ 1\ 0\ 1}_{cca2} \equiv +5_{10} \end{array}$$

Si se efectúa la suma no se obtiene el valor indicado!!!

Nota: Tanto en las ALUs como en los circuitos sumadores el resultado siempre tiene el mismo número de dígitos (bits) que los operandos.

Formatos de coma flotante (o de punto flotante)

Propiedades:

▪ Se utilizan para representar números muy grandes o muy pequeños (números próximos a cero) así como números con parte fraccionaria no nula (números reales)

Ejemplos:

Masa de un electrón: $m_e = 9.11 \times 10^{-31}$ kg.

Carga de un electrón: $Q_e = 1.6 \times 10^{-19}$ Coulombs.

Permeabilidad del vacío: $\mu_0 = 4\ \pi \times 10^{-7}$ N/A^2

Permitividad del vacío o constante dieléctrica: $\varepsilon_0 = 8{,}85418 \cdot 10^{-12}$ F/m.

Velocidad de la luz: $v = 2.99792 \times 10^8$ m/sg

▪ La notación que utilizan los *sistemas digitales*, equivalente a la *notación científica* que utilizamos las *personas* [$N = \pm M \times 10^E$], se denomina *notación en coma flotante* y tiene el siguiente formato:

$$N = (BS)\ m \times 2^e$$

siendo:

_ *BS*: *bit de signo* del número.

_ *m*: *mantisa* ≡ número en coma fija (el número de dígitos con el que se representa influye fundamentalmente en la *precisión* de los cálculos y del resultado)

_ *e*: *exponente* ≡ número en coma fija (el número de dígitos con el que se representa influye fundamentalmente en el rango de valores que se puede representar)

▪ La representación de una cantidad en la notación en *coma flotante* (o *punto flotante*) no es única.

Ejemplo:

$$336_{10} = 10.5 \times 2^5 = 1010.1_2 \times (10_2)^{0101_2} =$$

$$= 5.25 \times 2^6 = 101.01_2 \times (10_2)^{0110_2} =$$

$$= 2.625 \times 2^7 = 10.101_2 \times (10_2)^{0111_2} =$$

$$= 1.3125 \times 2^8 = \underbrace{1.0101}_{2} \times (10_2)^{1000_2}$$

mantisa normalizada: *su parte entera es igual a* 1

En 1985 el *Instituto de Ingenieros Eléctricos y Electrónicos* (IEEE) de Estados Unidos publicó una norma con la que estableció un formato estándar para representar cantidades utilizando la *notación en coma flotante* (norma IEEE 754).

$$N = (-1)^{BS} \times m \times 2^{e'} = (-1)^{BS} \times \underbrace{1.M}_{\text{mantisa normalizada}} \times 2^{e}$$

La versión de 32 dígitos (*formato de precisión simple*) tiene la siguiente estructura:

siendo:

E^* ≡ exponente correspondiente a la mantisa *normalizada*, representado en código "*exceso a* 127"

$$E^* = e + 127 \in (0, 255)_{10} \equiv (00000000_2,\ 11111111_2) \Rightarrow$$

$$\Rightarrow\ e = E^* - 127 \in (-127, +128)_{10}$$

valor a guardar

valor del exponente correspondiente a la *mantisa normalizada*

Nota: la codificación del exponente en código exceso a 127 permite representar tanto números muy grandes como muy pequeños sin tener que indicar el signo del exponente (se representan exponentes desde −127 a +128 utilizando un número entero de 8 bits sin signo)

M: parte *fraccionaria* de la *mantisa normalizada* (es un número entero sin signo).

No se guarda la parte *entera* de la *mantisa* porque siempre va a valer 1.

Excepciones

$0_{10} \equiv$

0	00000000	000 ····· 0

$\infty \equiv$

BS	11111111	000 ····· 0

Otros formatos: formato de *media precisión* (16 bits = 1+5+10), de *precisión simple* (32 bits = 1+8+23), de *precisión doble* (64 bits = 1+11+52), de *precisión doble ampliada* (≥ 80 bits) y de *cuádruple precisión* (128 bits).

Ejemplo 1:

$$2.25_{10} = 10.01_2 = 1.001_2 \times (10_2)^{1_2} \qquad (-1)^{BS} \times 1.M \times 2^e$$

$$BS = 0$$

$$M = \overbrace{001}\ 0000\ 0000\ 0000\ 0000\ 0000$$ ≡ parte fraccionaria de la mantisa normalizada

$$e = 1 \quad \rightarrow \quad E^* = e + 127 = 128_{10} = 10000000_2$$

	BS	*E**	*M*
$2.25_{10} \equiv$	0	10000000	00100000000000000000000

Resultado: 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Ejemplo 2:

$$-347.75_{10} = -101011011.11_2 = -1.01011011112 \times (10_2)^{1000_2}$$

con la mantisa normalizada

$$BS = 1$$

$$M = 01011011110000\overset{23)}{\cdots}0$$

$$e = 1000 \quad \rightarrow \quad E^* = e + 127 = 135_{10} = 10000111_2$$

	BS	*E**	*M*
$-347.75_{10} \equiv$	1	10000111	01011011110000000000000

Resultado: 11000011101011011110000000000000

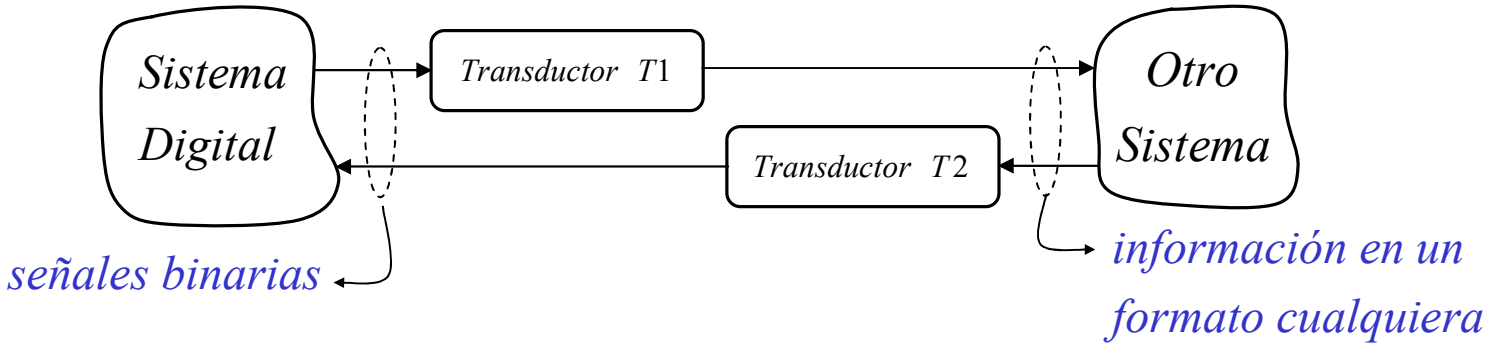
Códigos binarios:

- Conceptos generales (definiciones)
- Códigos numéricos:
  - Binario natural
  - Códigos BCD (natural, exceso 3, Aiken)
  - Códigos de Gray
  - Código Johnson y Anillo

- Códigos alfanuméricos (ASCII)
- Códigos detectores/correctores de errores (de paridad)

Conceptos generales:

Def.: El establecimiento de una correspondencia *sistemática* y *biunívoca* entre dos formas distintas de representar información constituye un *código*.



En esta asignatura sólo se estudian códigos que relacionan una forma cualquiera de representar información con otra que sólo utiliza 1_s y 0_s para representar información ≡ *códigos binarios*

Def.: Se denomina codificación binaria al proceso de asignarle un grupo de bits a una información dada.

Ejemplo: al color *verde* le corresponde 10_2

al color *ámbar* le corresponde 00_2

al color *rojo* le corresponde 11_2

Def.: Se denomina decodificación binaria al proceso en el que se determina el significado (información) que le corresponde a un grupo de bits dado.

Def.: Algunos códigos binarios asignan a cada uno de los símbolos F_i de un *alfabeto fuente* dado una combinación de símbolos (1_s y 0_s) de un *alfabeto código*. Un ejemplo considerando como *alfabeto fuente* los símbolos del *sistema decimal* y como *alfabeto código* los símbolos del *sistema binario* es el siguiente:

Alfabeto fuente	**Palabras código** (8 4 2 1)	**Alfabeto código**
0	0000	0
1	0001	1
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	

Ejemplo de uso del código anterior: $56'419_{10} \equiv 0101\ 0110'0100\ 0001\ 1001$

A cada combinación de símbolos (bits) del alfabeto código correspondiente a un símbolo F_i del alfabeto fuente se le denomina *palabra código* o *palabra de código* (*code word*).

Def.: Un código ponderado se caracteriza porque a cada dígito de una palabra código se le asigna un peso cuyo valor depende de la posición que ocupa el dígito en la palabra código. El valor asociado a una palabra código es igual a la suma de los pesos asociados a los dígitos que valen 1.

Ejemplo: $\begin{matrix} 2\ 4\ 2\ 1 \equiv pesos \\ 1\ 1\ 0\ 0 \equiv 4 + 2 = 6 \end{matrix}$

Def.: Se define la distancia entre dos palabras código como el número de dígitos que deben ser cambiados en una de ellas para obtener la otra.

Ejemplo: $\left.\begin{matrix} 1\ 1\ 0\ 1 \\ 1\ 0\ 1\ 0 \end{matrix}\right\} \rightarrow$ la distancia es 3

Def.: Se define la distancia de un código binario como la menor de las distancias entre sus palabras código.

Def.: Se dice que dos grupos de bits (palabras código incluidas) son *adyacentes* si se puede obtener uno a partir del otro cambiando un solo dígito en uno de ellos (≡ si su distancia es igual a 1).

Def.: Se dice que un código es *continuo* si todas sus palabras código consecutivas son adyacentes (si la distancia entre palabras consecutivas es igual a 1).

Def.: Se dice que un código continuo es *cíclico* si son adyacentes la primera y la última palabra del código.

Def.: Se dice que un código (numérico) es *autocomplementario* si la palabra código asignada al símbolo decimal $N \in [0, 9]$ es igual al *complemento a* 1 de la palabra código asignada al símbolo decimal $9-N$.

Def.: Se dice que un grupo de bits tiene *paridad par* si el número de bits que están a 1 es un número par. Análogamente, si el número de 1_s es impar, se dice que tiene *paridad impar*.

Códigos numéricos**:** se utilizan para representar cantidades con el fin de operar con ellas y/o guardarlas para ser utilizadas (operar con ellas) en otro momento.

▪ *Binario natural*: es el código binario más simple. La codificación de un dato en este código consiste en representar su valor (módulo) en el sistema binario.

▪ *Formatos de representación de números con signo*

_ en *coma fija*: *csm, cca*1 y *cca*2

_ en *coma flotante*: formato de *media precisión* (16 bits = 1+5+10), de *precisión simple* (32 bits = 1+8+23), de *precisión doble* (64 bits = 1+11+52), de *precisión doble ampliada* (≥ 80 bits) y de *cuádruple precisión* (128 bits).

▪ *Códigos para representar específicamente números decimales sin signo*: los más utilizados son los códigos *BCD* (*Binary Coded Decimal*). En estos códigos (*BCD*) a cada símbolo del sistema decimal le corresponde una palabra código dada.

	$BCD_{natural}$	$BCD_{exceso\ 3}$	BCD_{Aiken}
	8 4 2 1		2 4 2 1
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	0100
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

	$BCD_{natural}$ 8-4-2-1	$BCD_{exceso\ 3}$	BCD_{Aiken} 2-4-2-1
Ponderado	*si*	*no*	*si*
Autocomplementario	*no*	*si*	*si*
Continuo	*no*	*no*	*no*
Cíclico	*no*	*no*	*no*
Distancia	1	1	1

Ejemplos: $754'309_{10} = 0111\ 0101\ 0100\ '\ 0011\ 0000\ 1001_{BCD\ NATURAL}$

$754'309_{10} = 1010\ 1000\ 0111\ '\ 0110\ 0011\ 1100_{BCD\ EXCESO\ 3}$

$754'309_{10} = 1101\ 1011\ 0100\ '\ 0011\ 0000\ 1111_{BCD\ AIKEN}$

Nota: los códigos *BCD* sólo codifican la magnitud, el módulo o el valor absoluto de cantidades decimales. No se pueden utilizar para codificar el signo de las cantidades (en el caso de que lo tengan)

▪ *Códigos de Gray*:

Nota: Los códigos de Gray, conocidos también como binario reflejado, fueron inventados por Elisha Gray en 1878 *y reinventados posteriormente por Frank Gray en* 1949

_ Generación por reflexión: (ver ejemplo explicado en la clase de teoría)

_ Conversión de *Gray* a *binario*:

$$\left(G_{n-1}\cdots G_2G_1G_0\right)_{Gray}\rightarrow\left(B_{n-1}\cdots B_2B_1B_0\right)_2 \quad \Bigg/ \quad \begin{array}{l} B_{n-1}=G_{n-1} \\ \\ B_i=B_{i+1}\oplus G_i \quad 0\le i\le n-2 \end{array}$$

Una forma *equivalente* de indicar el método de conversión anterior es la siguiente:

$$\begin{array}{cccccc} G_{n-1} & \cdots & G_{n-1} & G_{n-1} & G_{n-1} & G_{n-1} \\ & & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ & & G_3 & G_3 & G_3 & G_3 \\ & & & G_2 & G_2 & G_2 \\ & & & & G_1 & G_1 \\ \oplus & & & & & G_0 \\ \hline B_{n-1} & \cdots & B_3 & B_2 & B_1 & B_0 \end{array}$$

Nota: la suma *o-exclusiva* $(\oplus)$ es equivalente a una suma *aritmética* en la que no se tienen en cuenta los acarreos (en el caso de que los haya).

Ejemplo: $1101_{Gray} = \left(G_3 G_2 G_1 G_0\right)_{Gray} \rightarrow n = 4$

$$\left.\begin{aligned} B_3 &= G_3 = 1 \\ B_2 &= B_3 \oplus G_2 = 1 \oplus 1 = 0 \\ B_1 &= B_2 \oplus G_1 = 0 \oplus 0 = 0 \\ B_0 &= B_1 \oplus G_0 = 0 \oplus 1 = 1 \end{aligned}\right\} \Rightarrow \left(B_3 B_2 B_1 B_0\right)_2 = 1001_2 = 9_{10} \equiv 1101_{Gray}$$

$$\begin{array}{cccc} 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 \\ & & 0 & 0 \\ \oplus & & & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}$$

_ Conversión de *binario* a *Gray*:

$$\left(B_{n-1}\cdots B_2B_1B_0\right)_2 \rightarrow \left(G_{n-1}\cdots G_2G_1G_0\right)_{Gray} \quad \Big/ \quad \begin{array}{l} G_{n-1} = B_{n-1} \\ \\ G_i = B_{i+1} \oplus B_i \quad 0 \le i \le n-2 \end{array}$$

Una forma *equivalente* de indicar el método de conversión anterior es la siguiente:

$$\begin{array}{cccccc} & B_{n-1} & B_{n-2} & \cdots & B_2 & B_1 & B_0 \\ \oplus & & B_{n-1} & \cdots & B_3 & B_2 & B_1 \\ \hline & G_{n-1} & G_{n-2} & \cdots & G_2 & G_1 & G_0 \end{array}$$

Nota: la suma *o-exclusiva* ($\oplus$) es equivalente a una suma *aritmética* en la que no se tienen en cuenta los acarreos.

Ejemplo:

$$12_{16} = 10010_2 = B_4B_3B_2B_1B_0 \quad \rightarrow \; n = 5$$

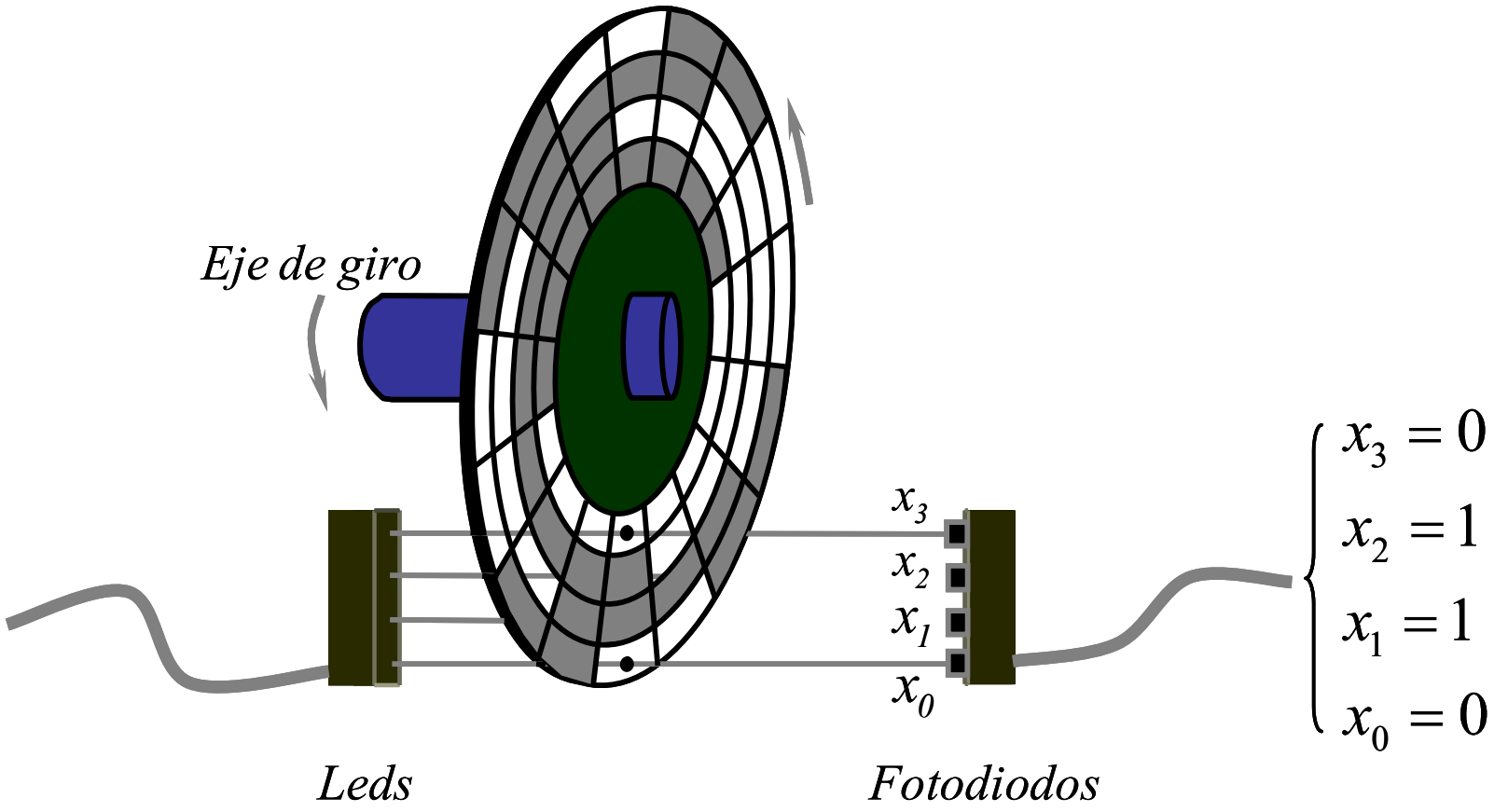
$$\left.\begin{aligned} G_4 &= B_4 = 1 \\ G_3 &= B_4 \oplus B_3 = 1 \oplus 0 = 1 \\ G_2 &= B_3 \oplus B_2 = 0 \oplus 0 = 0 \\ G_1 &= B_2 \oplus B_1 = 0 \oplus 1 = 1 \\ G_0 &= B_1 \oplus B_0 = 1 \oplus 0 = 1 \end{aligned}\right\} \Rightarrow \left(G_4G_3G_2G_1G_0\right)_{Gray} = 11011_{\text{Gray}} \equiv 10010_2 \equiv 12H$$

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

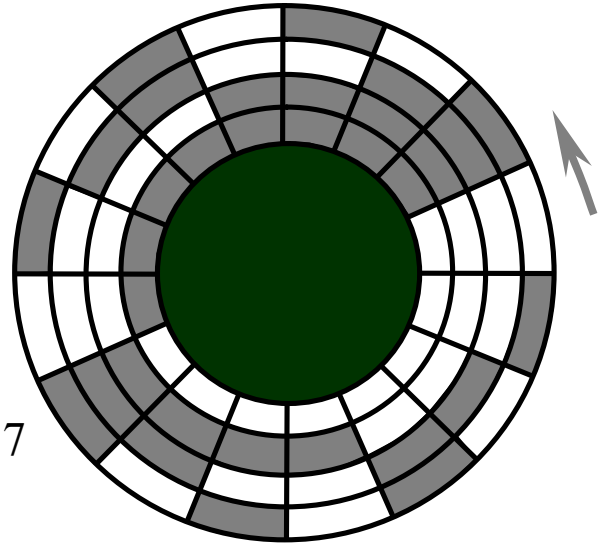
Propiedades de los códigos *Gray*: son *continuos*, *cíclicos*, *no ponderados* y su *distancia* es igual a 1.

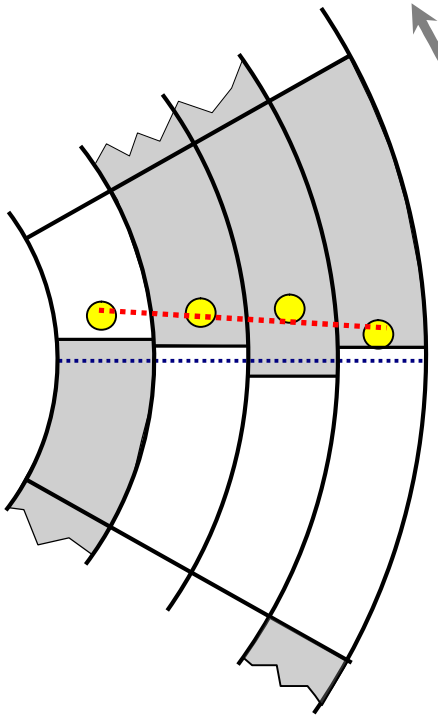
$n=1$	$n=2$	$n=3$	$n=4$	decimal
0	**0** 0	**0** 0 0	**0** 0 0 0	0
1	**0** 1	**0** 0 1	**0** 0 0 1	1
	1 1	**0** 1 1	**0** 0 1 1	2
	1 0	**0** 1 0	**0** 0 1 0	3
		1 1 0	**0** 1 1 0	4
		1 1 1	**0** 1 1 1	5
		1 0 1	**0** 1 0 1	6
		1 0 0	**0** 1 0 0	7
			1 1 0 0	8
			1 1 0 1	9
			1 1 1 1	10
			1 1 1 0	11
			1 0 1 0	12
			1 0 1 1	13
			1 0 0 1	14
			1 0 0 0	15

Los *códigos de Gray* tienen muchas aplicaciones, entre ellas cabe destacar la codificación de la posición de un elemento móvil.



Disco codificado en *Binario*:





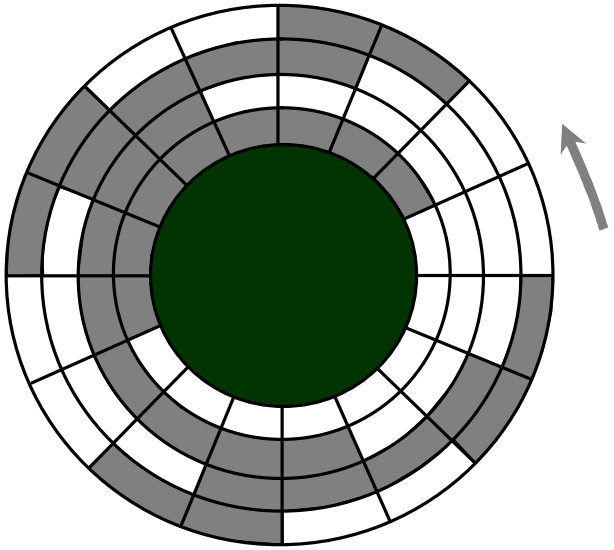
sector 7 $\rightarrow$ $x_3x_2x_1x_0 = 0111$

0110
1110
1010

$\rightarrow$ *posible secuencia de valores transitorios duran - te el paso del sector 7 al 8.*

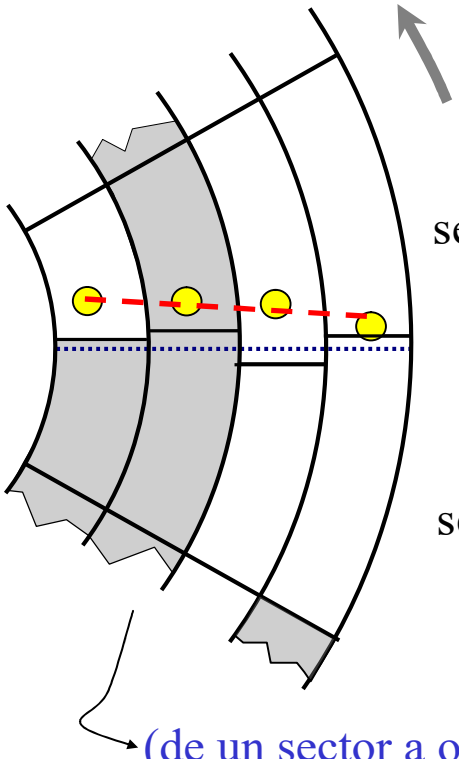
sector 8 $\rightarrow$ $x_3x_2x_1x_0 = 1000$

Disco codificado en *Gray*:



sector 8

sector 7



sector 7 → $x_3x_2x_1x_0 = 0100$

no hay valores transitorios!!!

sector 8 → $x_3x_2x_1x_0 = 1100$

(de un sector a otro sólo cambia 1 digito)

▪ Código *Johnson* o *Möbius*: es muy fácil de generar y de decodificar. Es *continuo*, *cíclico* y su *distancia* es igual a 1.

▪ Código en *Anillo*: se genera con un circuito denominado *contador en anillo*. Es muy fácil de generar y la decodificación es automática. No es *continuo*, no es *cíclico* y su *distancia* es igual a 2.

	Johnson (3*dígitos*)	*Johnson* (4*dígitos*)	*Anillo* (3*dígitos*)	*Anillo* (4*dígitos*)
0	000	0000	001	0001
1	001	0001	010	0010
2	011	0011	100	0100
3	111	0111		1000
4	110	1111		
5	100	1110		
6		1100		
7		1000		

NO

▪ Código *One-hot*: es muy fácil de generar y de decodificar. No es *continuo* y su *distancia* es igual a 2. Se utiliza para codificar los estados internos de un sistema secuencial. Este código tiene la ventaja de que el circuito combinacional que determina el siguiente estado interno del sistema así como el circuito combinacional que genera las salidas del sistema resultan ser más sencillos que con otras codificaciones, lo que permite utilizar una señal de reloj con una mayor frecuencia (sistemas más rápidos!!!). Tiene el inconveniente de utilizar muchos *flip-flops* (uno por cada estado interno del sistema).

▪ Código *Two-hot*: es parecido al código *one-hot*, con la ventaja de que con *n flip-flops* se pueden codificar hasta $n(n-1)/2$ estados internos.

▪ Código *Almost one-hot*: es como el código *one-hot*, con un valor inicial extra en el que todos los bits son cero.

NO

Decimal	*Binario*	*One-hot*	*Two-hot*	*Almost one-hot*
0	000	00000001	00011	0000000
1	001	00000010	00101	0000001
2	010	00000100	01001	0000010
3	011	00001000	10001	0000100
4	100	00010000	00110	0001000
5	101	00100000	01010	0010000
6	110	01000000	10010	0100000
7	111	10000000	01100	1000000

Códigos alfanuméricos: se utilizan para representar números y caracteres de todo tipo. El más utilizado es el código *ASCII* ≡ *American Standard Code for Information Interchange*. La versión normalizada de este código tiene 128 palabras código de 7 dígitos, las cuales se pueden dividir en dos grupos:

· Las primeras 32 palabras código corresponden a *comandos de control* que se utilizan en la comunicación entre sistemas digitales.

· Las demás palabras código (96) corresponden a diversos tipos de *caracteres*, entre los que se incluyen:

_ *las letras del alfabeto* (mayúsculas y minúsculas).

_ *los diez símbolos del sistema decimal*.

_ *signos de puntuación*.

_ *otros símbolos* utilizados habitualmente como: >, <, =, { }, [ ], ( )

Palabra código	Número decimal	Comando de control
0000000	0	Nul: null
0000001	1	SOH: start of heading
0000010	2	STX: start of text
0000011	3	ETX: end of text
0000100	4	EOT: end of transmision
0000101	5	ENQ: enquiry
0000110	6	ACK: acknowledge
0000111	7	BEL: bell
0001000	8	BS: backspace
0001001	9	HT: horizontal tab
0001010	10	LF: line feed, new line
0001011	11	VT: vertical tab
0001100	12	FF: form feed
0001101	13	CR: carriage return
0001110	14	SO: shift out
0001111	15	SI: shift in

Palabra código	Número decimal	Carácter
0100000	32	space
0100001	33	¡
0100010	34	“
0100011	35	#
0100100	36	$
0100101	37	%
0100110	38	&
0100111	39	‘
0101000	40	(
0101001	41	)
0101010	42	*
0101011	43	+
0101100	44	,
0101101	45	-
0101110	46	.
0101111	47	/
0110000	48	0
0110001	49	1
0110010	50	2
0110011	51	3

Palabra código	Número decimal	Carácter
0111010	58	:
0111011	59	;
0111100	60	<
0111101	61	=
0111110	62	>
0111111	63	?
1000000	64	@
1000001	65	A
1000010	66	B
1000011	67	C
1000100	68	D
1000101	69	E
1000110	70	F
1000111	71	G
1001000	72	H
1001001	73	I
1001010	74	J
1001011	75	K
1001100	76	L
1001101	77	M
1001110	78	N
1001111	79	O

Palabra código	Número decimal	Carácter
1010000	80	P
1010001	81	Q
1010010	82	R
1010011	83	S
1010100	84	T
1010101	85	U
1010110	86	V
1010111	87	W
1011000	88	X
1011001	89	Y
1011010	90	Z

Ejemplo: “DIGITAL”

	“	D	I	G	I	T
en ASCII:	0100010	1000100	1001001	1000111	1001001	1010100

A	L	“	
1000001	1001100	0100010	(*en los circuitos*)

	“	D	I	G	I	T	A	L	”
*en decimal**:	34	68	73	71	73	84	65	76	34
*en hexadecimal**:	22	44	49	47	49	54	41	4C	22

} (*las personas*)***

Ejemplo 2: HOLA

en ASCII: 1001000 1001111 1001100 1000001

en base 10: 72 79 76 65

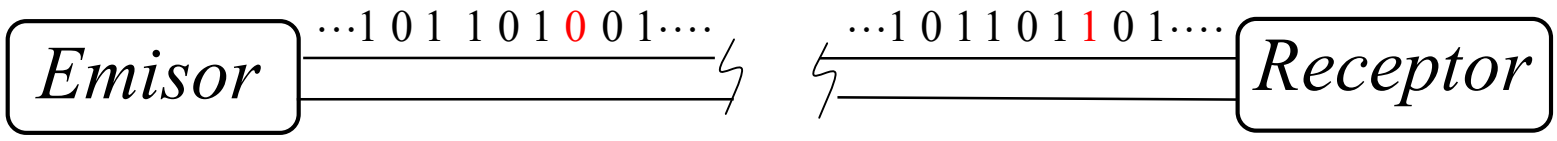
en hexadecimal: 48 4F 4C 41

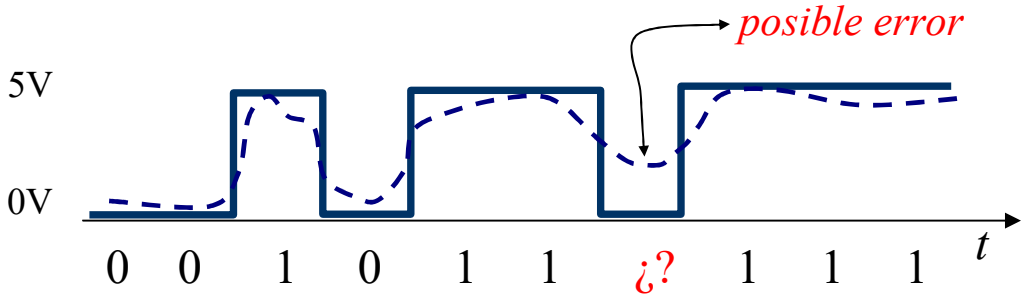
Además del *código ASCII estándar* existe un *código ASCII extendido* formado por 128 palabras código que se utilizan para codificar:

- *caracteres alfabéticos no ingleses*: ñ, ç, etc.
- *símbolos matemáticos*.
- *símbolos para gráficos*.
- etc.

El código *ASCII extendido* fue utilizado inicialmente por *IBM* en sus ordenadores personales (PCs). Hoy en día es tan popular que se ha convertido en un estándar de facto*.

Códigos detectores/correctores de errores





Estos códigos se caracterizan porque añaden información redundante a la información a transmitir. De modo que el *receptor,* al analizar la información que recibe, puede determinar si se ha producido o no un error en la transmisión de algún *bit*. En el caso de los códigos *correctores de errores* el *receptor* puede determinar además el *bit* en el que se ha producido el error.

▪ Códigos de paridad par/impar: se generan a partir de cualquier código, añadiéndole a sus palabras código un bit denominado *bit de paridad*.

H: El uso de estos códigos se basa en la hipótesis de que las palabras que llegan al receptor tienen como mucho 1 bit erróneo.

_ *Códigos de paridad par*: el *bit de paridad* (*BP*) toma el valor adecuado para que el número de bits que están a 1 sea un número *par*.

Ejemplo: $\underbrace{01001100}$ → *dato a enviar* : $\overset{BP}{1}\ \underbrace{01001100}$

_ *Códigos de paridad impar*: el *bit de paridad* toma el valor adecuado para que el número de bits que están a 1 sea un número *impar*.

Ejemplo: $\underbrace{01001100}$ → *dato a enviar* : $\overset{BP}{0}\ \underbrace{01001100}$

- Códigos de *Hamming*: permiten detectar y corregir errores

- Códigos *CRC* (*códigos polinómicos* o de *redundancia cíclica*): se basan en añadir r bits al mensaje de k bits, de forma tal que el polinomio resultante, $T(x)$, correspondiente a los $k + r$ bits, sea divisible por un polinomio $G(x)$. El receptor verifica si el polinomio $T(x)$ es divisible o no (división módulo 2) por el polinomio generador $G(x)$. En el caso de que no lo sea (resto distinto de cero), significa que ha habido un error en la transmisión.

- Códigos con *Checksum*

▪ Nombres asociados a grupos de bits:

_ Bit: grupo de 1 digito binario

_ Crumb, Tydbit, or Tayste: grupo de 2 bits

_ Nibble or Nybble: grupo de 4 bits

_ Nickle: grupo de 5 bits

_ Byte: grupo de 8 bits

_ Deckle: grupo de 10 bits

_ Playte: grupo de 16 bits

_ Dynner: grupo de 32 bits

_ Word: depende del sistema considerado. En un procesador de 16 bits el término *word* indica 16 bits, mientras que en un procesador de 32 bits el término *word* significa 32 bits. Los términos *playte* y *dynner* se refieren a grupos de 16 y de 32 bits respectivamente, con independencia del contexto en el que se utilicen.