

Tema 1: Sistemas de numeración y códigos binarios

1.1: Introducción.

1.2: Sistema binario.

1.2.1: Aritmética binaria.

1.3: Sistema hexadecimal.

1.4: Representación y aritmética de números con signo.

1.5: Códigos binarios, alfanuméricos y detectores/correctores de errores.

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 otra 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.

_ Un número es una representación mediante símbolos de una cantidad dada. En general, una misma cantidad se representará mediante un número distinto en función del sistema de numeración que se considere. Así, por ejemplo:

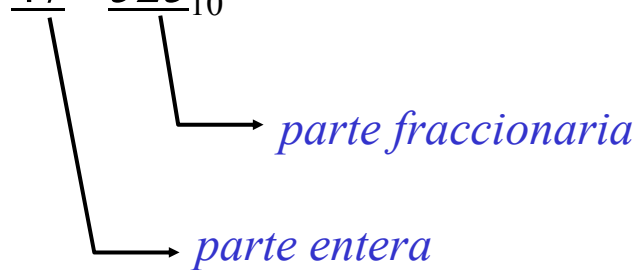
$$20_{10} = 14_{16} = 24_8 = 110_4 = 202_3 = 10100_2$$

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:

- Una *cantidad* se representa mediante una sucesión ordenada de símbolos (dígitos o cifras) situados a izquierda y derecha de una coma (o punto) de referencia.

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



parte fraccionaria

parte entera

- En un número, cada dígito tiene asociado un *peso* cuyo valor depende de la *base* (b) 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:

$$\begin{array}{cccccccccccc}
 \text{Pesos:} & b^{n-1} & b^{n-2} & & & b^1 & b^0 & \text{coma} & b^{-1} & b^{-2} & & & b^{-(m-1)} & b^{-m} \\
 & \downarrow & \downarrow & & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & & \downarrow & \downarrow \\
 \text{Número (base = } b\text{):} & a_{n-1} & a_{n-2} & \cdot & \cdot & \cdot & a_1 & a_0 & , & a_{-1} & a_{-2} & \cdot & \cdot & \cdot & a_{-(m-1)} & a_{-m} \\
 & \uparrow & \uparrow & & & & \uparrow & \uparrow & & \uparrow & \uparrow & & & & \uparrow & \uparrow \\
 \text{Posiciones:} & n-1 & n-2 & & & & 1 & 0 & & -1 & -2 & & & & -(m-1) & -m \\
 & \underbrace{\hspace{10em}} & & & & & \underbrace{\hspace{10em}} & & & \underbrace{\hspace{10em}} & & & & & \underbrace{\hspace{10em}} & \\
 & \text{parte entera} & & & & & \text{parte fraccionaria} & & & & & & & & &
 \end{array}$$

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 (<i>base</i> = 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 (<i>base</i> = 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

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

<i>Binario</i>	<i>Ternario</i>	<i>Cuaternario</i>	<i>Octal</i>	<i>Decimal</i>	<i>Hexadecimal</i>
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 :

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

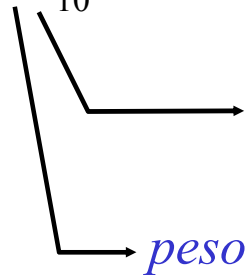
está determinada por el siguiente *polinomio equivalente*:

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

si las operaciones indicadas en el polinomio equivalente se realizan en un sistema de numeración de base b' , el resultado que se obtiene es un número que representa en el sistema de numeración de base b' la misma cantidad que el número N_b .

Ejemplos:

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

 *peso* $10^0 = 1$ (*unidades*)

peso $10^1 = 10$ (*decenas*)

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

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

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

Ejemplo:

$$\begin{aligned}
 \overset{8}{1}\overset{2}{0}\overset{1}{1}1_2 &= 1_2 \cdot (10_2)^{11_2} + 0_2 \cdot (10_2)^{10_2} + 1_2 \cdot (10_2)^{1_2} + 1_2 \cdot (10_2)^{0_2} = \\
 &= 1_{10} \cdot (2_{10})^{3_{10}} + 0_{10} \cdot (2_{10})^{2_{10}} + 1_{10} \cdot (2_{10})^{1_{10}} + 1_{10} \cdot (2_{10})^{0_{10}} = \\
 &= 1_{10} \cdot 8_{10} + 0_{10} \cdot 4_{10} + 1_{10} \cdot 2_{10} + 1_{10} \cdot 1_{10} = 8 + 0 + 2 + 1 = 11_{10}
 \end{aligned}$$

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: $8 + 2 + 1 = 11$).

(más ejemplos)

$$\begin{aligned} 2F'4_{16} &= 2_{16} \cdot (10_{16})^{1_{16}} + F_{16} \cdot (10_{16})^{0_{16}} + 4_{16} \cdot (10_{16})^{-1_{16}} = \\ &= 10_2 \cdot (10000_2)^{1_2} + 1111_2 \cdot (10000_2)^{0_2} + 100_2 \cdot (10000_2)^{-1_2} = 101111'01_2 \end{aligned}$$

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

→ *Nota:* lo que nos resulta útil es representar todas las cantidades que aparecen en el polinomio equivalente en base 10. De esta forma, las operaciones indicadas en el polinomio se pueden realizar en base 10 y el resultado obtenido será la representación equivalente del número de partida en 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:

$$b^n - 1$$

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

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

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 n de dígitos que se necesita para representar en *binario* una cantidad N_{10} cumple lo siguiente:

$$n = \text{valor entero mayor o igual que } \left\lceil \frac{\log_{10} N_{10}}{\log_{10} 2} \right\rceil$$

Ejemplo: $N = 325_{10}$

$$n = \text{entero mayor o igual que } \left\lceil \frac{\log_{10} 325}{\log_{10} 2} = 8.3442 \right\rceil$$

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

- 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: $1111111111111111_2 = 177777_8 = 65535_{10} = \text{FFFF}_{16}$

(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.

Ejemplo: $100001111111010'00001100_2 = 87FA'0C_{16}$

Sistema binario (o binario natural):

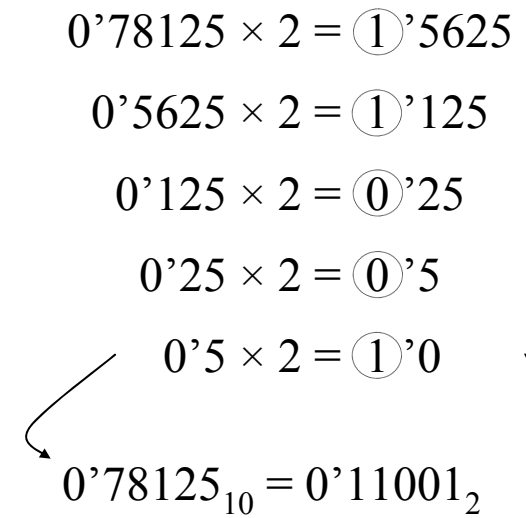
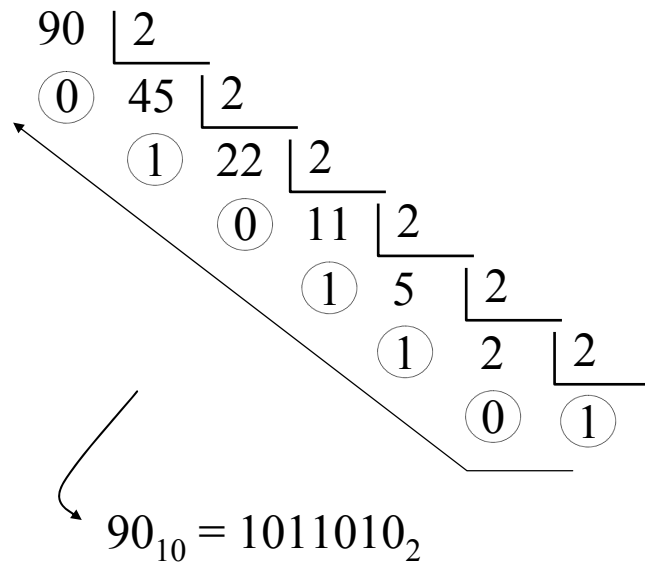
- Introducido por *Leibniz* en el siglo XVII
- Su base es 2 y por lo tanto en este sistema de numeración sólo se utilizan combinaciones de dos símbolos distintos (1 y 0) para representar cualquier cantidad.
- Es el sistema de numeración en el que operan todos los *sistemas digitales*. Los motivos de que esto sea así son tanto *tecnológicos* como *económicos*.

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

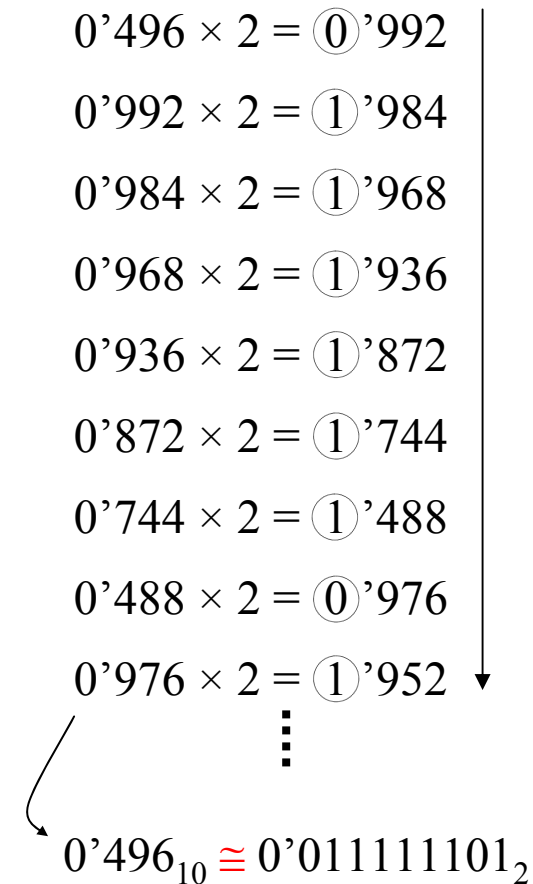
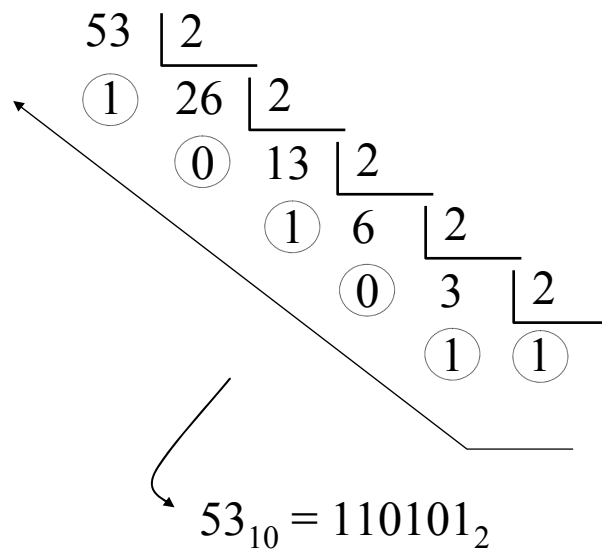
Método 1 (teórico): para determinar la representación de la *parte entera* se realizan divisiones por 2 de forma sucesiva, hasta obtener un cociente inferior a 2. La solución está formada por los restos de las divisiones, leídos en orden inverso al que se generan (ver ejemplo).

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).

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



Ejemplo 2: $53'496 \cong 110101'011111101_2 \rightarrow$ se obtiene una representación en binario 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:

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

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

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

- Conversión de *binario a decimal*: se utiliza el *polinomio característico* \equiv 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

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

$$\begin{aligned} 101011'11_2 &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = \\ &= 32 + 0 + 8 + 0 + 2 + 1 + 0.5 + 0.25 = 43.75_{10} \end{aligned}$$

32 16 8 4 2 1 0.5 0.25

$$1 \ 0 \ 1 \ 0 \ 1 \ 1' \ 1 \ 1_2 \rightarrow 32+8+2+1+0,5+0,25 = 43.75$$

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)

$$\begin{array}{r}
 \begin{array}{c} \text{d} \swarrow \text{acarreo} \\ \text{x x x } \boxed{a}_2 \\ + \text{x x x } \boxed{b}_2 \\ \hline \boxed{c} \end{array}
 \quad
 \begin{array}{c} 0 \\ \text{x x x } \boxed{0}_2 \\ + \text{x x x } \boxed{0}_2 \\ \hline \boxed{0} \end{array}
 \quad
 \begin{array}{c} 0 \\ \text{x x x } \boxed{1}_2 \\ + \text{x x x } \boxed{0}_2 \\ \hline \boxed{1} \end{array}
 \quad
 \begin{array}{c} 0 \\ \text{x x x } \boxed{0}_2 \\ + \text{x x x } \boxed{1}_2 \\ \hline \boxed{1} \end{array}
 \quad
 \begin{array}{c} 1 \\ \text{x x x } \boxed{1}_2 \\ + \text{x x x } \boxed{1}_2 \\ \hline \boxed{0} \end{array}
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ 1 \ 0 \ 1 \ 1 \ 0 \ 1_2 \equiv 11'25_{10} \\ + \ 1 \ 1 \ 0 \ 1 \ 1 \ 1_2 \equiv 6'875_{10} \\ \hline 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1_2 \equiv 18'125_{10} \end{array}
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 1 \\ 1 \ 1 \ 1 \ 0 \ 1_2 \equiv 7'25_{10} \\ + \ 1 \ 0 \ 0 \ 1 \ 0 \ 1_2 \equiv 9'25_{10} \\ \hline 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0_2 \equiv 16'50_{10} \end{array}
 \end{array}$$

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

$$\begin{array}{r} \text{X X X} \text{ a}_2 \\ - \text{X X X} \text{ b}_2 \\ \hline \text{d} \text{ c} \end{array}$$
 acarreo

$$\begin{array}{r} \text{X X X } 0_2 \\ - \text{X X X } 0_2 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{X X X } 1_2 \\ - \text{X X X } 0_2 \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{xxx} \overline{)0} \\ - \text{xxx} \overline{)1} \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{X X X } \boxed{1}_2 \\ - \text{X X X } \boxed{1}_2 \\ \hline \end{array}$$

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

$$\begin{array}{r} 1011'01_2 \equiv 11'25_{10} \\ - \quad \begin{array}{ccccccc} & 1 & 1 & 0 & 1 & 1 & 1 \\ & _ & _ & _ & _ & _ & _ \\ 1 & 0 & 0 & 1 & 1 & 1 & \end{array} \equiv 6'875_{10} \\ \hline 0100'011_2 \equiv 4'375_{10} \end{array}$$

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

$$\begin{array}{r} \alpha \alpha \alpha \boxed{a}_2 \\ \times \quad \alpha \alpha \boxed{b}_2 \\ \hline \quad \quad \quad \boxed{c} \end{array}$$

$$\begin{array}{r} \alpha \alpha \alpha \boxed{0}_2 \\ \times \quad \alpha \alpha \boxed{0}_2 \\ \hline \quad \quad \quad \boxed{0} \end{array}$$

$$\begin{array}{r} \alpha \alpha \alpha \boxed{1}_2 \\ \times \quad \alpha \alpha \boxed{0}_2 \\ \hline \quad \quad \quad \boxed{0} \end{array}$$

$$\begin{array}{r} \alpha \alpha \alpha \boxed{0}_2 \\ \times \quad \alpha \alpha \boxed{1}_2 \\ \hline \quad \quad \quad \boxed{0} \end{array}$$

$$\begin{array}{r} \alpha \alpha \alpha \boxed{1}_2 \\ \times \quad \alpha \alpha \boxed{1}_2 \\ \hline \quad \quad \quad \boxed{1} \end{array}$$

$$\begin{array}{r} 10110_2 \equiv 22_{10} \\ \times \quad 101_2 \equiv 5_{10} \\ \hline \quad 10110 \\ \quad 00000 \\ 10110 \\ \hline 1101110 \equiv 110_{10} \end{array}$$

$$\begin{array}{r} 10111'011_2 \\ \times \quad 100_2 \\ \hline 00000000 \\ 00000000 \\ 10111011 \\ \hline 1011101'100_2 \end{array}$$

Nota: *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

$$\begin{array}{r}
 1101'1001_2 \quad \overline{)101_2} \\
 \underline{101} \\
 00111 \\
 \underline{101} \\
 1000 \\
 \underline{101} \\
 111 \\
 \underline{101} \\
 101 \\
 \underline{101} \\
 0
 \end{array}$$

$$\begin{array}{r}
 1001'01_2 \quad \overline{)11_2} \\
 \underline{11} \\
 011 \\
 \underline{11} \\
 0011 \\
 \underline{11} \\
 0
 \end{array}$$

$$\begin{array}{r}
 101101'00_2 \quad \overline{)100_2} \\
 \underline{100} \\
 110 \\
 \underline{100} \\
 101 \\
 \underline{100} \\
 100 \\
 \underline{100} \\
 0
 \end{array}$$

Nota: *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: $1101010011010101010011110101_2$

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, *en general*, requerirá 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: $b = 8 = 2^3$ (apenas se utiliza hoy en día)

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

Nota: en esta asignatura a los dígitos de un número binario también los vamos a denominar bits.

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.
- 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: $1010111100110100'110110111001_2 = \text{AF34'DB9}_{16}$

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

Ejemplo: $4A_{16} = 0x4A = 4AH = 4Ah$

- 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.

$$\text{Ejemplo: } \underbrace{11}_{3} \underbrace{1000}_{8} \underbrace{1111}_{F} \underbrace{0111}_{7} \underbrace{0000}_{0} \underbrace{0101}_{5} \underbrace{1110}_{E} \underbrace{1010}_{A} \underbrace{0001}_{1} \underbrace{11}_D_2 = 38F70'5EA1D_{16}$$

- 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)

$$\text{Ejemplo: } 9F60E'D4CA_{16} = \underbrace{1001}_9 \underbrace{1111}_F \underbrace{0110}_6 \underbrace{0000}_0 \underbrace{1110}_E \underbrace{1101}_D \underbrace{0100}_4 \underbrace{1100}_C \underbrace{1010}_A_2$$

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

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

- 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 10 no lo es!!!

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

$$\begin{array}{r} 65_{10} \overline{) 16_{10}} \\ \underline{64} \\ 1 \end{array}$$

(4)

(1)

$65_{10} = 41_{16}$

$$\begin{array}{l} 0'28125 \times 16 = (4)'5 \\ 0'5 \times 16 = (8)'0 \end{array}$$

conversion exacta

$0'28125_{10} = 0'48_{16}$

Ejemplo 2: $2619'324_{10} \cong A3B'52F1A9_{16} \rightarrow$ se obtiene una representación en hexadecimal con 9 dígitos de precisión en la parte fraccionaria.

$$\begin{array}{r}
 2619_{10} \begin{array}{l} \overline{16} \\ 101 \\ \underline{96} \\ 59 \\ \underline{48} \\ \textcircled{11} \end{array} \begin{array}{l} \overline{16_{10}} \\ 163 \\ \underline{16} \\ 03 \\ \underline{0} \\ \textcircled{3} \end{array} \begin{array}{l} \overline{16_{10}} \\ 10 \\ \underline{\quad} \\ \textcircled{10} \end{array} \\
 \text{B} \qquad \qquad \qquad \text{A}
 \end{array}$$

$2619_{10} = A3B_{16}$

$$\begin{array}{l}
 0'324 \times 16 = \textcircled{5}'184 \\
 0'184 \times 16 = \textcircled{2}'944 \\
 0'944 \times 16 = \textcircled{15}'104 \\
 0'104 \times 16 = \textcircled{1}'664 \\
 0'664 \times 16 = \textcircled{10}'624 \\
 0'624 \times 16 = \textcircled{9}'984 \\
 \vdots
 \end{array}$$

$0'324_{10} \cong 0'52F1A9_{16} \rightarrow$ con 6 decimales de precisión en la parte fraccionaria.

Formatos binarios de representación de números 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.

$$\left. \begin{array}{l} 1 \text{ lógico} \rightarrow 5\text{v.} \equiv \text{nivel alto de tensión} \\ 0 \text{ lógico} \rightarrow 0\text{v.} \equiv \text{nivel bajo de tensión} \end{array} \right\} \equiv \text{lógica positiva}$$

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

$$\begin{array}{l} \text{Formatos binarios} \\ \text{de representación de} \\ \text{cantidades con signo} \end{array} \left\{ \begin{array}{l} \text{Formatos de coma fija} \left\{ \begin{array}{l} \text{Código signo-magnitud (csm)} \\ \text{Código complemento a 1 (cca1)} \\ \text{Código complemento a 2 (cca2)} \end{array} \right. \\ \text{Formatos de coma flotante} \end{array} \right.$$

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:



cumpléndose que:

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

Formatos de coma fija (**implícita**):

Código Signo-Magnitud:

- el módulo de una cantidad se representa en binario (natural).
- el cero tiene dos representaciones distintas.
- con n dígitos, el rango de representación es: $-(2^{n-1} - 1)$ hasta $(2^{n-1} - 1)$

**CONVENIO
SIGNO-MAGNITUD
(csm) n=4**

+ 7	^{BS} 0 1 1 1
+ 6	0 1 1 0
+ 5	0 1 0 1
+ 4	0 1 0 0
+ 3	0 0 1 1
+ 2	0 0 1 0
+ 1	0 0 0 1
+ 0	0 0 0 0
− 0	1 0 0 0
− 1	1 0 0 1
− 2	1 0 1 0
− 3	1 0 1 1
− 4	1 1 0 0
− 5	1 1 0 1
− 6	1 1 1 0
− 7	1 1 1 1
− 8	- - - -

ANEXO: Aritmética complementaria

- Las siguientes definiciones son válidas para números 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: $\underbrace{x x \cdots x x'}_{\alpha \text{ dígitos(bits)}} \underbrace{x x \cdots x x_2}_{\beta \text{ dígitos(bits)}}$

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

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

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

1º método: Aplicando la definición anterior

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

$$cal(N) = 2^4 - 2^{-2} - N = 16_{10} - 0.25_{10} - 14.25_{10} = 1.5_{10} = \textcolor{red}{000}1.1\textcolor{red}{0}$$

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

$$cal(N) = 2^4 - 2^0 - N = 16_{10} - 1_{10} - 11_{10} = 4_{10} = \textcolor{red}{0}100$$

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

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

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

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

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

Propiedades del cal :

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

$$\text{es decir, si } M = cal(N) \Rightarrow N = cal(M)$$

$$ii) \quad N_2 + cal(N_2) = 2^\alpha - 2^{-\beta} = 11 \cdots 1^{\alpha} 1' 11 \cdots 1^{\beta} \quad (\alpha + \beta \text{ dígitos})$$

- Complemento a 2 de un número binario de $n+k$ bits: $\underbrace{x x x \cdots x x'}_{\alpha \text{ dígitos(bits)}} \underbrace{x x x \cdots x x_2}_{\beta \text{ dígitos(bits)}}$

Dado un número binario N (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 de la siguiente forma:

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

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

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

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

1º método: Aplicando la definición anterior

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

$$ca2(N_2) = 2^4 - N_2 = 16_{10} - 14.25_{10} = 1.75_{10} = \textcolor{red}{000}1.11_2$$

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

$$ca2(N_2) = 2^4 - N_2 = 16_{10} - 11_{10} = 5_{10} = \textcolor{red}{0}101_2$$

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

$$ca2(N_2) = 2^3 - 0 = 8_{10} = \textcolor{red}{\cancel{1}}000_2 \rightarrow ca2(000) = 000_2$$

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

2º método: cálculo del $ca2$ en función del $ca1$

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

Nota : para $\beta = 0$ se cumple que $ca2(N_2) = 2^\alpha - N_2 = ca1(N_2) + 1$

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


$$ca1(N) = 0001.10$$

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

3° método: para obtener el $ca2$ de un número 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:

$$\begin{array}{cccccccc}
 N & = & 1 & 0 & 1 & 1 & 0 & . & 0 & 1 & 0 & 0_2 \\
 & & | & | & | & | & | & & | & | & | & \\
 ca2(N) & = & 0 & 1 & 0 & 0 & 1 & . & 1 & 1 & 0 & 0_2
 \end{array}$$


 se ha escrito el primer 1

$$\begin{array}{ccccccc}
 N & = & 0 & 0 & 0 & . & 0 & 0_2 \\
 & & | & | & | & & | & \\
 ca2(N) & = & 0 & 0 & 0 & . & 0 & 0_2
 \end{array}$$

$$\begin{array}{cccccccccccccccccccc}
 N = & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & . & 0 & 1 & 1 & 1 & 0 & 0 & 1_2 \\
 & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | \\
 ca2(N) = & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & . & 1 & 0 & 0 & 0 & 1 & 1 & 1_2
 \end{array}$$


 se ha escrito el primer 1

Propiedades del $ca2$:

i) $M_2 = 2^\alpha - N_2 \Rightarrow N_2 = 2^\alpha - M_2 = ca2(N_{ca2})$

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

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

Fin Anexo: Aritmética complementaria

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

Código complemento a 1 (*cca1*)

Propiedades:

- Es un código binario para representar cantidades con signo.
- El bit de signo se sitúa en la posición del MSB \equiv *most significant bit*.
- Las cantidades positivas tienen la misma representación que en el *csm*.
- 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).

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 !!!

Código complemento a 2 (*cca2*)

Propiedades:

- Es un código binario para representar cantidades con signo.
- El bit de signo se sitúa en la posición del MSB.
- Las cantidades positivas tienen la misma representación que en el *csm* y que en el *cca1*
- 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:

$$(-N)_{cca2} = ca2(N_{cca2})$$

tanto si N_{cca2} representa una cantidad positiva
como si representa una cantidad negativa.

Ejemplos:

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

$$\overset{+7_{10}}{\text{III}} \quad \overset{BS}{ca2(00111)} = 1\ 1001_{cca2} \equiv -7_{10}$$

$$\overset{-7_{10}}{\text{III}} \quad \overset{BS}{ca2(11001)} = 0\ 0111_{cca2} \equiv +7_{10} \quad (***)$$

→ *Calcular el ca2 de un número representado en el cca2 equivale a cambiarlo de signo.*


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

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

↪ **El cero tiene una única representación.**

- Calcular el complemento a 2 de un número equivale a cambiarlo de signo:

$$N_{cca2}^- = ca2(N_{cca2}^+) = 2^n - N_{cca2}^+ \quad \Rightarrow \quad N_{cca2}^+ = 2^n - N_{cca2}^- = ca2(N_{cca2}^-)$$


 por definición

del resultado anterior se deduce que *calcular el ca2 de una cantidad representada en el cca2, tanto si es positiva como si es negativa, equivale a cambiarla de signo*. Y que por lo tanto calcular dos veces seguidas el ca2 de una cantidad equivale a no hacer nada.

Ejemplo:

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

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

- Se cumple que:

$$N_{cca2} + (-N)_{cca2} = 100 \cdot \overset{n+k)}{\cdot \cdot \cdot} 0 \quad 1 + (n + k) \text{ ceros}$$

- El rango de cantidades enteras que se pueden representar con n dígitos (incluido el bit de signo) es **

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

** Nota: de acuerdo con la definición del código *cca2*, 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 bits. El cual, matemáticamente corresponde a la representación en el *cca2* 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 *cca2*).

	CONVENIO SIGNO MAGNITUD (<i>csm</i>) $n = 4$	CONVENIO COMPLEMENTO A UNO (<i>cca1</i>) $n = 4$	CONVENIO COMPLEMENTO A DOS (<i>cca2</i>) $n = 4$
	B.S.	B.S.	B.S.
+ 7	0 1 1 1	0 1 1 1	0 1 1 1
+ 6	0 1 1 0	0 1 1 0	0 1 1 0
+ 5	0 1 0 1	0 1 0 1	0 1 0 1
+ 4	0 1 0 0	0 1 0 0	0 1 0 0
+ 3	0 0 1 1	0 0 1 1	0 0 1 1
+ 2	0 0 1 0	0 0 1 0	0 0 1 0
+ 1	0 0 0 1	0 0 0 1	0 0 0 1
+ 0	0 0 0 0	0 0 0 0	0 0 0 0
- 0	1 0 0 0	1 1 1 1	0 0 0 0
- 1	1 0 0 1	1 1 1 0	1 1 1 1
- 2	1 0 1 0	1 1 0 1	1 1 1 0
- 3	1 0 1 1	1 1 0 0	1 1 0 1
- 4	1 1 0 0	1 0 1 1	1 1 0 0
- 5	1 1 0 1	1 0 1 0	1 0 1 1
- 6	1 1 1 0	1 0 0 1	1 0 1 0
- 7	1 1 1 1	1 0 0 0	1 0 0 1
- 8	- - - -	- - - -	1 0 0 0 → excepción con 4 bits

Nota: con 4 dígitos se pueden escribir 16 números distintos. En el caso del *cca2*, 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. 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*, *cca1* y *cca2*? ¿Se podrá representar el -8 en el *cca2* con 5 bits? ¿y el -16?

Aritmética en el *cca2*:

_ Las operaciones con números que no son ni muy grandes ni muy pequeños (próximos a cero) se suelen realizar en el *cca2*.

_ Sólo vamos a considerar las operaciones de suma/resta

_ La ventaja del *cca2* reside en que las operaciones de suma/resta se realizan utilizando un circuito sumador.

Adición y sustracción en el *cca2*:

- 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 el análisis realizado en las siguientes diapositivas se supone que los números A y B con los que se opera son números *positivos*, representados en el *cca2*.
- Los resultados se obtienen representados en el *cca2*.
- 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: $M = A_{cca2} + B_{cca2} = \overset{BS}{0} A_2 + \overset{BS}{0} B_2 = \overset{BS}{0} (A_2 + B_2)$

- La suma de dos o más números positivos representados en el *cca2* proporciona el resultado correcto, representado en el *cca2*, 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:

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

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

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

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

$$A_{cca2} = \overset{BS}{0110}_{cca2} \equiv +6_{10} \quad y \quad B_{cca2} = \overset{BS}{0100}_{cca2} \equiv +4_{10}$$

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

$$\overset{BS}{1} \ 0 \ 1 \ 0_{cca2} \equiv +10_{10} ? \quad \rightarrow \text{desbordamiento: sumando dos números positivos se obtiene como resultado un número negativo.}$$

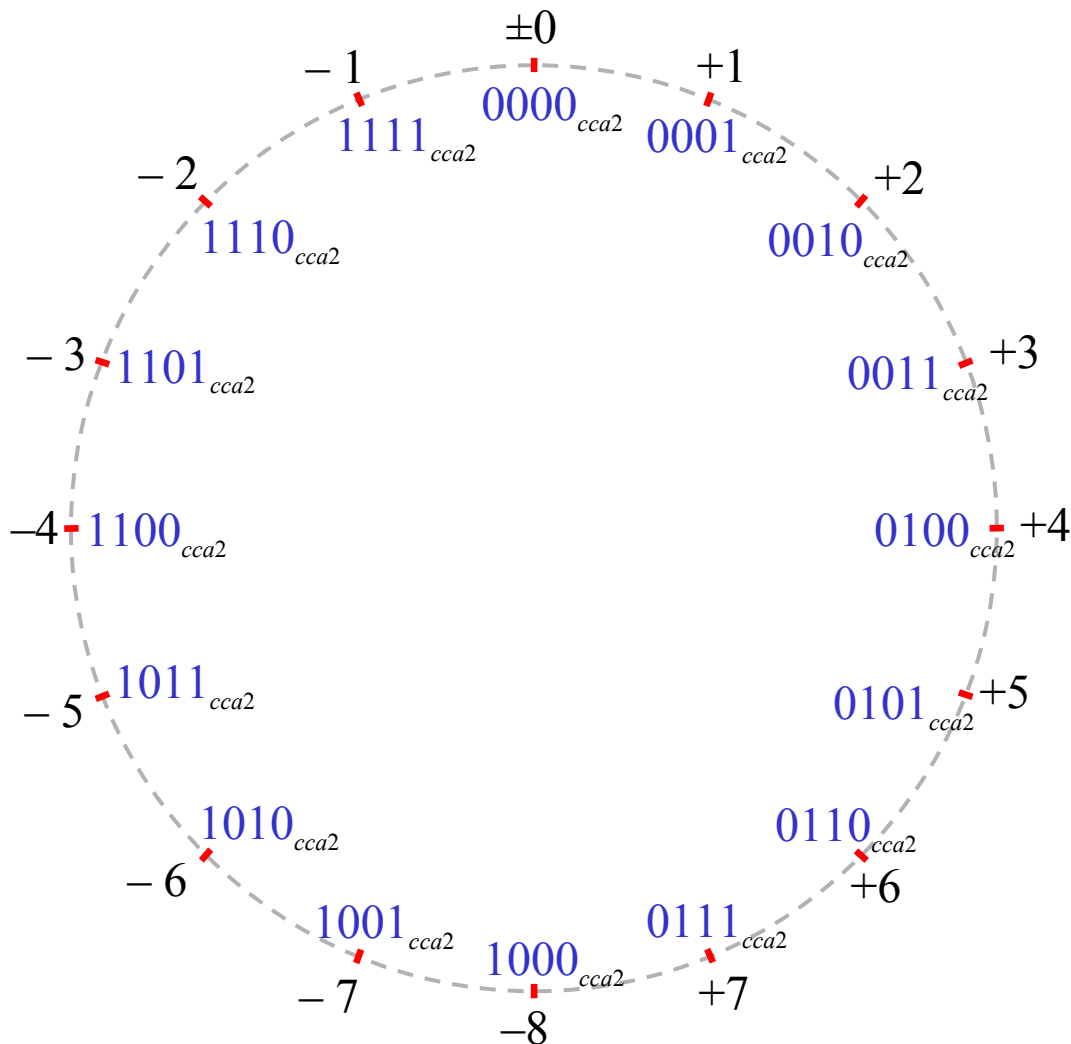
$$\begin{array}{r} \overset{BS}{0} \ \overset{1}{0} \ 1 \ 1 \ 0_{cca2} \equiv +6_{10} \\ + 0 \ 0 \ 1 \ 0 \ 0_{cca2} \equiv +4_{10} \\ \hline \end{array}$$

$$\overset{BS}{0} \ 1 \ 0 \ 1 \ 0_{cca2} \equiv +10_{10}$$

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

	<i>cca2</i>	valor <i>cca2</i>
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: $0110_{cca2} + 0011_{cca2} = 1001_{cca2}$
 $\begin{array}{ccc} \text{|||} & & \text{|||} \\ +6 & +3 & \text{¿+9 ó -7?} \end{array}$



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

Si se sustituye el signo menos por una operación de *ca2*, se puede calcular el valor de $A_{cca2} - B_{cca2}$ realizando una suma. Ya que la operación $A_{cca2} - B_{cca2}$ se convierte en una suma de dos números binarios:

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

una resta que se convierte en una suma de 2 números binarios

Nota:

$$\left. \begin{array}{l} A_{cca2} \text{ es un número binario} \\ -B_{cca2} = ca2(B_{cca2}) \text{ es un número binario} \end{array} \right\} \rightarrow A_{cca2} + ca2(B_{cca2}) \text{ es una suma de números binarios}$$

i) H: $A \geq B$

$$\begin{aligned}
 M_{cca2} &= A_{cca2} - B_{cca2} \Rightarrow A_{cca2} + (-B_{cca2})_{cca2} = A_{cca2} + ca2(B_{cca2}) = \\
 &= A_{cca2} + 2^n - B_{cca2} = 2^n + \underbrace{0 A_2 - 0 B_2}_{\text{número positivo}} = 2^n + \underbrace{M_{cca2}}
 \end{aligned}$$

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

Ejemplo:

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

$$A_{cca2} = \overset{BS}{0} 1 1 0_{cca2} \equiv +6_{10} \quad y \quad B_{cca2} = \overset{BS}{0} 1 0 0_{cca2} \equiv +4_{10}$$

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

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

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

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

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

el resultado obtenido coincide directamente con el valor deseado.

Ejemplo:

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

$$A_{cca2} = \overset{BS}{0\ 1\ 0\ 1}_{cca2} \equiv +5_{10} \quad y \quad B_{cca2} = \overset{BS}{0\ 1\ 1\ 1}_{cca2} \equiv +7_{10}$$

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

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

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

$$\begin{aligned}
 M_{cca2} = -A_{cca2} - B_{cca2} &\Rightarrow ca2(A_{cca2}) + ca2(B_{cca2}) = 2^n - A_{cca2} + 2^n - B_{cca2} = \\
 &= 2^n + 2^n - \underbrace{\left(\overset{BS}{0} A_2 + \overset{BS}{0} B_2 \right)}_{\text{número positivo en } cca2} = 2^n + \underbrace{ca2(A_{cca2} + B_{cca2})}_{\text{número negativo en } cca2} = \underbrace{2^n + M_{cca2}}
 \end{aligned}$$

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:

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

$$A_{cca2} = \overset{BS}{0\ 0\ 1\ 1}_{cca2} \equiv +3_{10} \quad y \quad B_{cca2} = \overset{BS}{0\ 1\ 0\ 0}_{cca2} \equiv +4_{10}$$

$$ca2(A_{cca2}) = \overset{BS}{1\ 1\ 0\ 1}_{cca2} \equiv -3_{10} \quad y \quad ca2(B_{cca2}) = \overset{BS}{1\ 1\ 0\ 0}_{cca2} \equiv -4_{10}$$

$$\begin{array}{rcl} ca2(A_{cca2}) \equiv & \overset{BS}{1} & \\ & 1 & 1 \quad 0 \quad 1_{cca2} \equiv -3_{10} \\ ca2(B_{cca2}) \equiv & + & 1 \quad 1 \quad 0 \quad 0_{cca2} \equiv -4_{10} \\ & \hline & \cancel{1} & 1 \quad 0 \quad 0 \quad 1_{cca2} \end{array}$$

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

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

$$ca2(A_{cca2}) \equiv \begin{array}{c} \boxed{BS} \\ 1 \end{array} 1\ 0\ 0_{cca2} \equiv -4_{10}$$

$$ca2(B_{cca2}) \equiv + \begin{array}{c} 1 \\ 1 \end{array} 0\ 1\ 1_{cca2} \equiv -5_{10}$$

$$\cancel{\begin{array}{c} 1 \\ 0 \end{array}} 1\ 1\ 1_{cca2}$$

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

$$\begin{array}{c} BS \\ 1 \\ 1 \end{array} 1\ 1\ 1\ 0\ 0_{cca2} \equiv -4_{10}$$

$$+ 1\ 1\ 0\ 1\ 1_{cca2} \equiv -5_{10}$$

$$\cancel{1} 1\ 0\ 1\ 1\ 1_{cca2}$$

Resumen: de los casos analizados en las diapositivas anteriores se deduce lo siguiente

- Todas las operaciones de suma/resta de cantidades representadas en el *cca2* se pueden realizar efectuando sumas.
- Hay que sustituir los signos menos (–) por operaciones de *ca2*
- Los sumandos y el resultado **siempre** deben tener el **mismo** número de dígitos
- El bit de signo del resultado **siempre** debe estar en la misma columna que el bit de signo de los sumandos.
- Se desprecia cualquier acarreo que se produzca al sumar los bits de signo.

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 codificado en el *cca2*. Y siempre que se declara una variable de tipo *unsigned* su valor se guarda codificado 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 *cca2* el resultado que se obtiene está representado en el *cca2*.

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

Errores típicos en las pruebas individuales:

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

Respuesta 1

$$A_{cca2} = \overset{BS}{0\ 1\ 1\ 0}_{cca2} \equiv +6_{10}$$

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

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

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

¿seguro?

Respuesta 2

$$A_{cca2} = \overset{BS}{0\ 1\ 1\ 0}_{cca2} \equiv +6_{10}$$

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

$$\overset{BS}{0\ 1\ 1\ 0}_{cca2} \equiv +6_{10}$$

$$+ \overset{BS}{1\ 1\ 1}_{cca2} \equiv -1_{10}$$

$$\overset{BS}{0\ 1\ 0\ 1}_{cca2} \equiv +5_{10}$$

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

Formatos de coma flotante (o de punto flotante)

- Se utilizan para representar números muy grandes o muy pequeños (números próximos a cero)

Ejemplos:

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

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

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

Velocidad de la luz: $v = 2.99792 \times 10^8 \text{ m/s}$

- 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* \equiv número en coma fija (el número de dígitos determina la *precisión* de los cálculos)

_ *e*: *exponente* \equiv número en coma fija (el número de dígitos está directamente relacionado con el rango de valores que puede manejar el sistema)

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

Ejemplo:

$$\begin{aligned} 336_{10} &= 10.5 \times 2^5 = 1010.1_2 \times (10_2)^{0101_2} = \\ &= 5.25 \times 2^6 = 101.01_2 \times (10_2)^{0110_2} = \\ &= 2.625 \times 2^7 = 10.101_2 \times (10_2)^{0111_2} = \\ &= 1.3125 \times 2^8 = \underbrace{1.0101}_2 \times (10_2)^{1000_2} \end{aligned}$$

↪ *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 números en coma flotante (norma IEEE 754).

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



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

siendo:

E^* \equiv 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*

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

$$0_{10} \equiv \begin{array}{|c|c|c|} \hline 0 & 00000000 & 000 \dots\dots\dots 0 \\ \hline \end{array}$$

$$\infty \equiv \begin{array}{|c|c|c|} \hline BS & 11111111 & 000 \dots\dots\dots 0 \\ \hline \end{array}$$

Ejemplo 1:

$$2.25_{10} = 10.01_2 = 1.\underbrace{001}_2 \times (10_2)^{1_2}$$

$$BS = 0$$

$$M = \underbrace{001}_{\text{parte fraccionaria de la mantisa normalizada}} 0000 0000 0000 0000 0000 \equiv \text{parte fraccionaria de la mantisa normalizada}$$

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

$$2.25_{10} \equiv \boxed{0 \mid 10000000 \mid 00100000000000000000000000000000}$$

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

Ejemplo 2:

$$-347.75_{10} = -101011011.11_2 = -1.0101101111_2 \times (10_2)^{1000_2}$$

$$BS = 1$$

$$M = 0101101111000 \overset{23)}{\dots} 0$$

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

$$-347.75_{10} \equiv \boxed{1 \mid 10000111 \mid 010110111100000000000000}$$

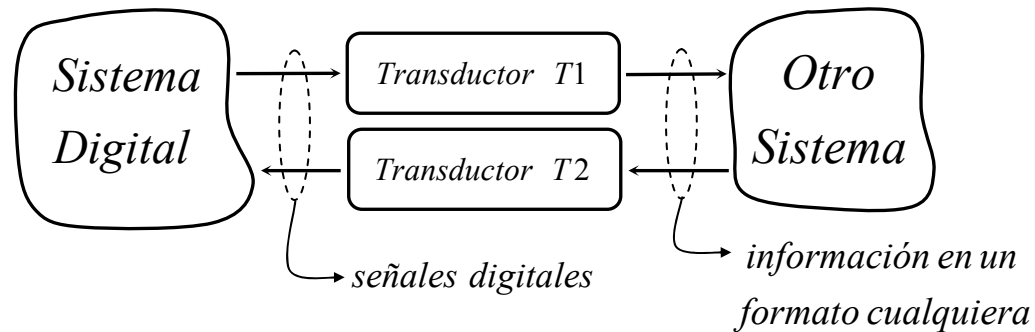
Resultado: 11000011101011011110000000000000

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 expresar 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 $0_s \equiv$ *códigos binarios*

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

Def.: Se denomina decodificación 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*.

Ejemplo:

Alfabeto fuente	Palabras código	Alfabeto código
0	^{8 4 2 1} 0000	0
1	0001	1
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	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 se le asigna un peso que depende de la posición que ocupa 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.

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.

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.

Def.: Se dice que un código es *continuo* si sus palabras código consecutivas son adyacentes.

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 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*, *cca1* y *cca2*

- _ 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*: los más utilizados son los códigos *BCD* (*Binary Coded Decimal*). En estos códigos (*BCD*) a cada símbolo del sistema decimal le corresponde una la palabra código dada.

	$BCD_{natural}$	$BCD_{exceso\ 3}$	BCD_{Aiken}
0	^{8 4 2 1} 0000	0011	^{2 4 2 1} 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

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

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

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

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

- *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 clase)

Conversión de *Gray* a *binario*:

$$\left(G_{n-1} \cdots G_2 G_1 G_0 \right)_{Gray} \rightarrow \left(B_{n-1} \cdots B_2 B_1 B_0 \right)_2 \quad \left/ \begin{array}{l} B_{n-1} = G_{n-1} \\ B_i = B_{i+1} \oplus G_i \quad 0 \leq i \leq n-2 \end{array} \right.$$

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

$$\begin{array}{cccccc}
G_{n-1} & \cdots & G_{n-1} & G_{n-1} & G_{n-1} & G_{n-1} \\
\cdots & & & & & \\
& & G_3 & G_3 & G_3 & G_3 \\
& & & G_2 & G_2 & G_2 \\
& & & & G_1 & G_1 \\
\oplus & & & & & G_0
\end{array}$$

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

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

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

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

_ Conversión de *binario* a *Gray*:

$$(B_{n-1} \cdots B_2 B_1 B_0)_2 \rightarrow (G_{n-1} \cdots G_2 G_1 G_0)_{Gray} \left/ \begin{array}{l} G_{n-1} = B_{n-1} \\ G_i = B_{i+1} \oplus B_i \quad 0 \leq i \leq n-2 \end{array} \right.$$

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

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

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

Ejemplo:

$$12_{16} = 10010_2 = B_4 B_3 B_2 B_1 B_0 \rightarrow n = 5$$

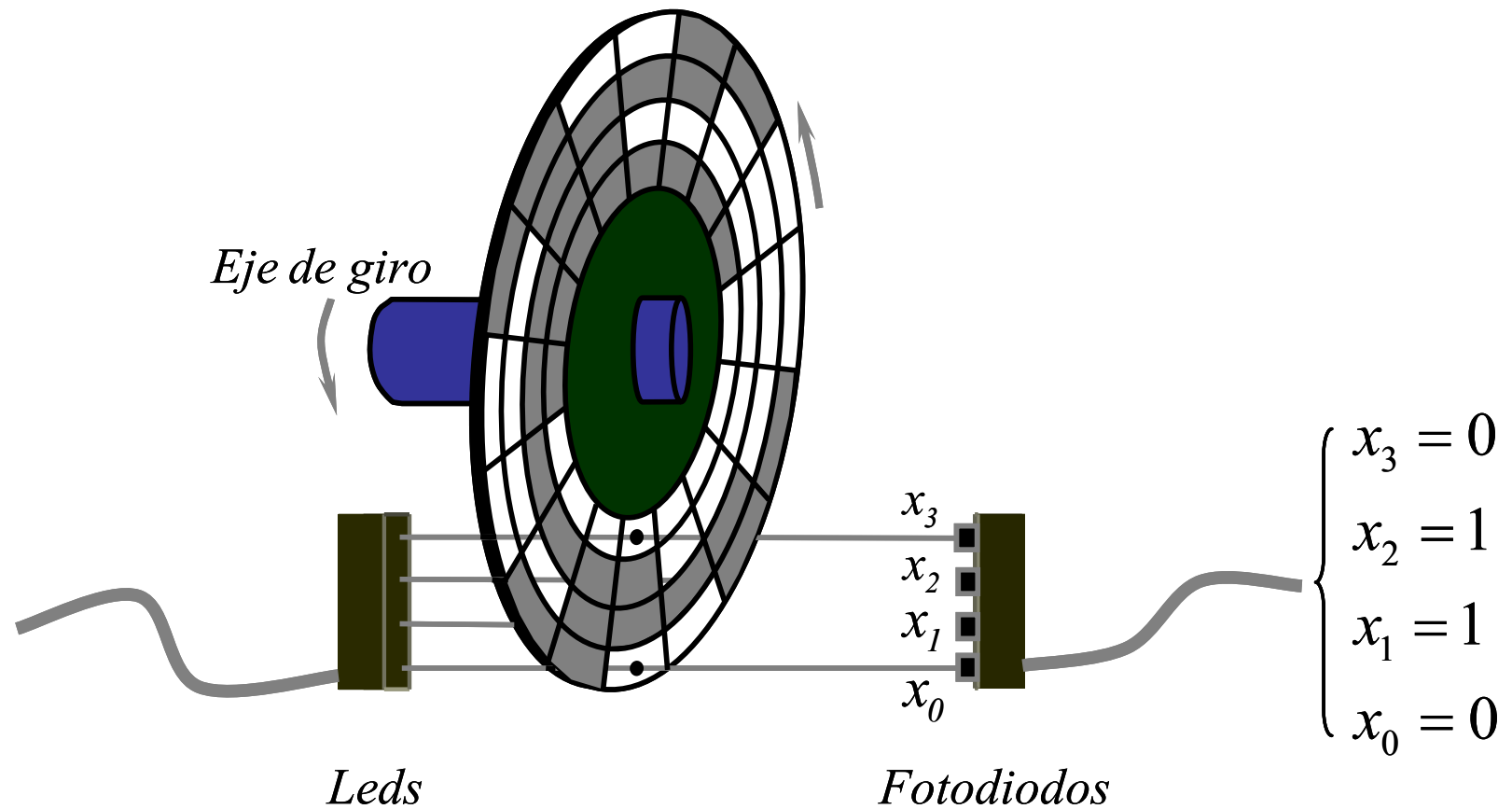
$$\left. \begin{array}{l} G_4 = B_4 = 1 \\ G_3 = B_4 \oplus B_3 = 1 \oplus 0 = 1 \\ G_2 = B_3 \oplus B_2 = 0 \oplus 0 = 0 \\ G_1 = B_2 \oplus B_1 = 0 \oplus 1 = 1 \\ G_0 = B_1 \oplus B_0 = 1 \oplus 0 = 1 \end{array} \right\} \Rightarrow (G_4 G_3 G_2 G_1 G_0)_{\text{Gray}} = \mathbf{11011}_{\text{Gray}}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

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 \ 0 \\ \oplus \quad 1 \ 0 \ 0 \ 1 \\ \hline \mathbf{1 \ 1 \ 0 \ 1 \ 1} \end{array}$$

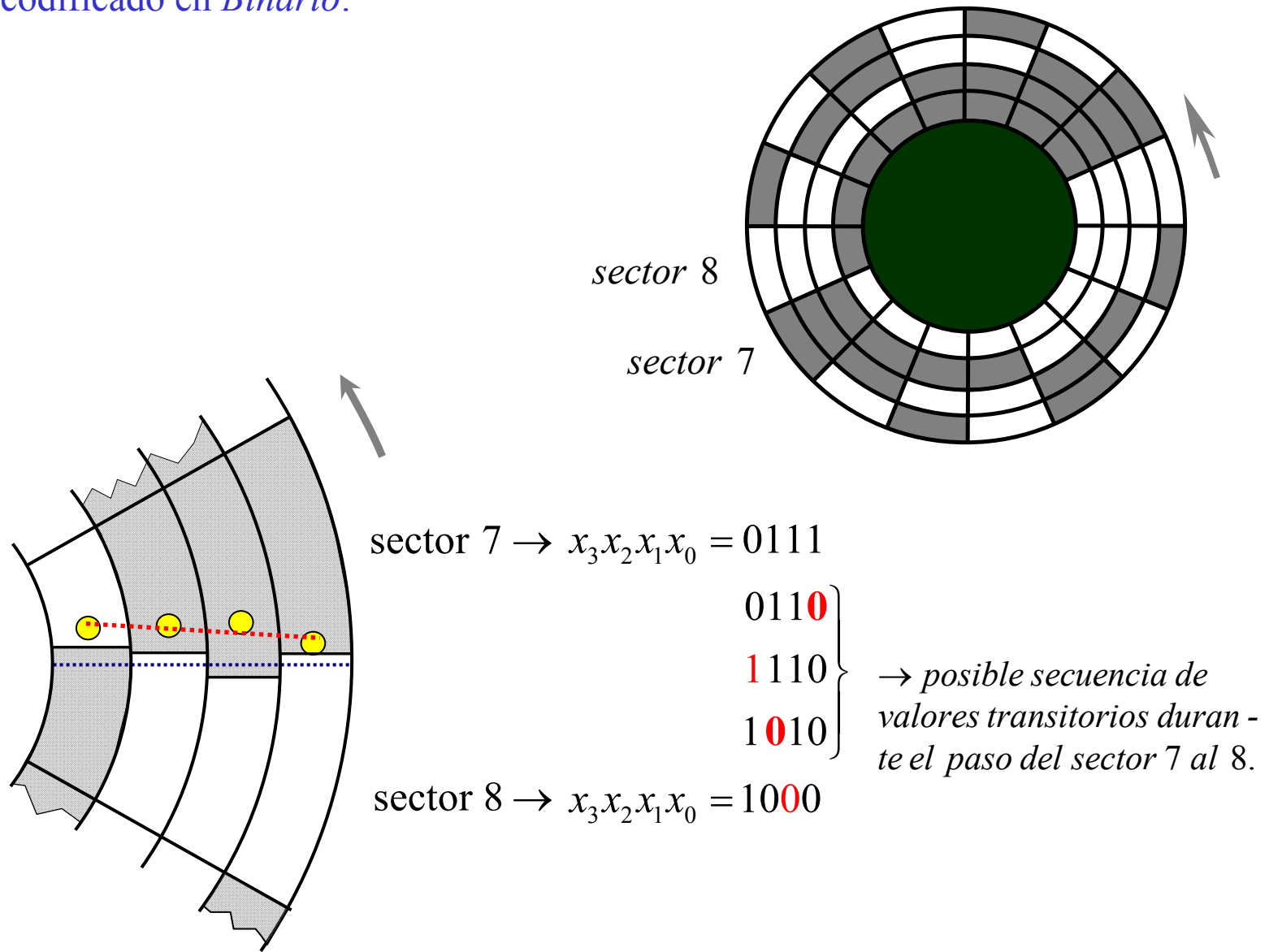
Propiedades de los códigos *Gray*: son *continuos*, *cíclicos*, *no ponderados* y su *distancia* es igual a 1.

$n = 1$	$n = 2$	$n = 3$	$n = 4$	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

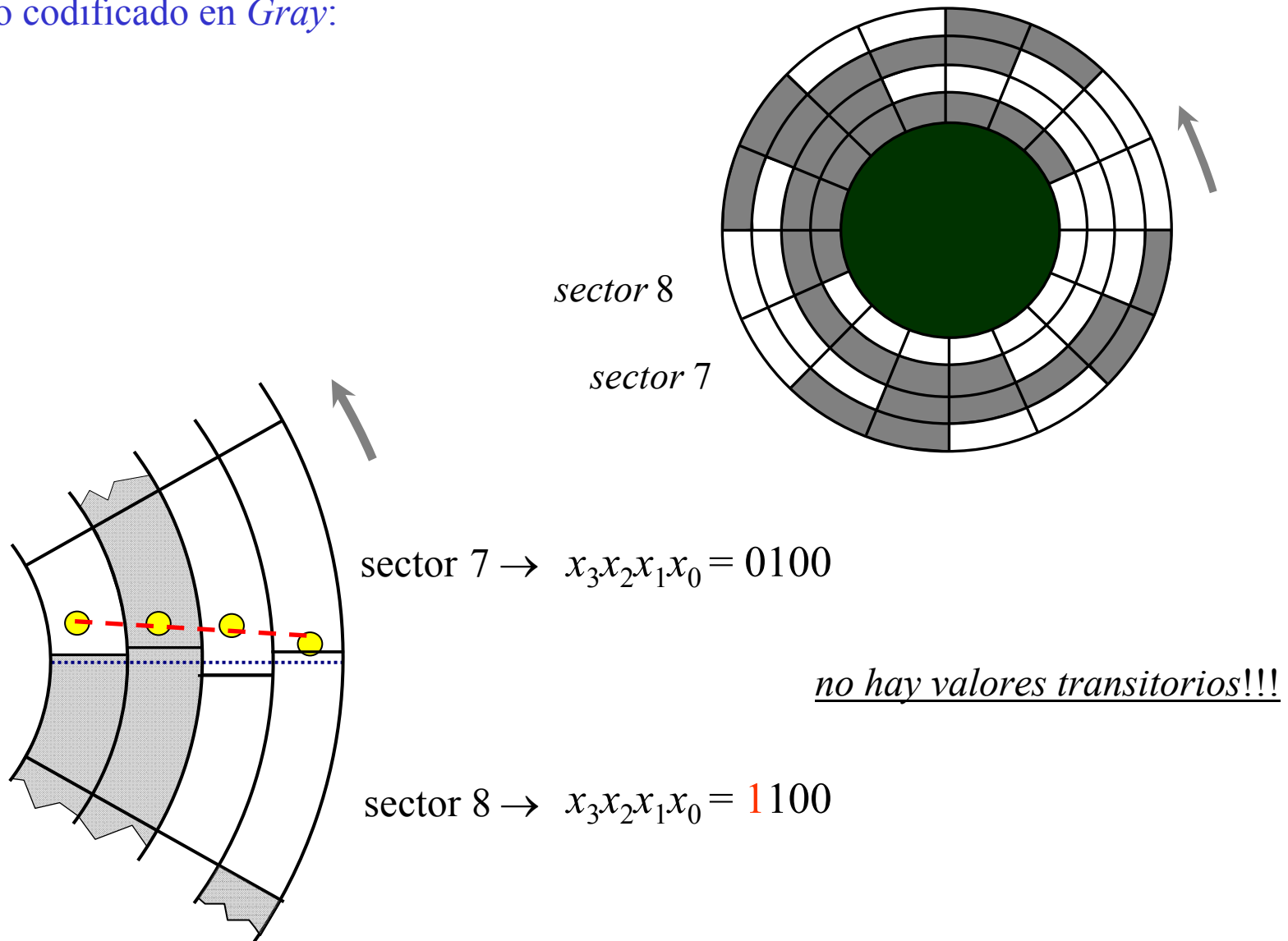
Los *códigos de Gray* tienen muchas aplicaciones, entre ellas cabe destacar la codificación de la posición de un elemento móvil.



Disco codificado en *Binario*:



Disco codificado en *Gray*:



- 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. Es *continuo*, no es *cíclico* y su *distancia* es igual a 2.

	<i>Johnson</i>	<i>Johnson</i>	<i>Anillo</i>	<i>Anillo</i>
	(3dígitos)	(4dígitos)	(3dígitos)	(4dígitos)
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		

Códigos alfanuméricos: el más utilizado es el código *ASCII* \equiv *American Standard Code for Information Interchange*.

La versión estándar 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 transmission
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	i
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	
<i>en ASCII:</i>	0100010	1000100	1001001	1000111	1001001	1010100	
	A	L	“				
	1000001	1001100	0100010				<i>(en los circuitos)</i>

	“	D	I	G	I	T	A	L	”	
<i>en decimal*:</i>	34	68	73	71	73	84	65	76	34	} <i>(las personas)***</i>
	“	D	I	G	I	T	A	L	”	
<i>en hexadecimal*:</i>	22	44	49	47	49	54	41	4C	22	

Ejemplo 2: HOLA

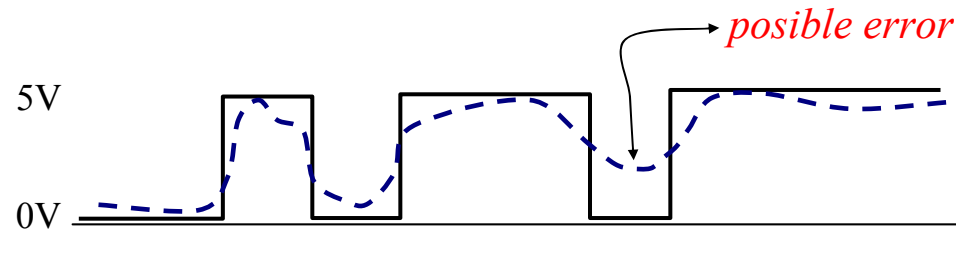
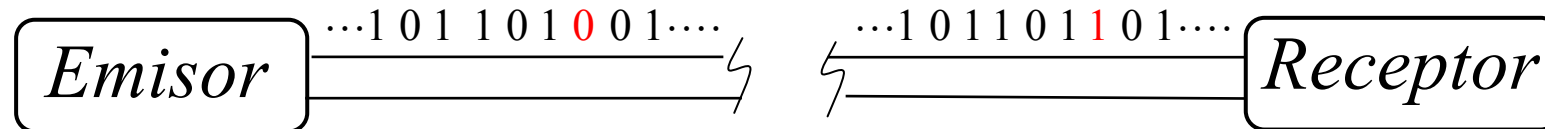
en ASCII: 1001000 1001111 1001100 1000001

Además del *código ASCII estándar* existe un *código ASCII extendido* formado por 128 caracteres 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 de distancia unidad, añadiéndole a sus palabras código un bit denominado *bit de paridad*.

H: El uso de estos códigos está basado en la hipótesis de que al enviar una *palabra*, como mucho al *receptor* llega 1 bit con un valor 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: $\underbrace{01001100}_{\text{dato a enviar}} \rightarrow \overset{BP}{1} \underbrace{01001100}$

▪ *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: $\underbrace{01001100}_{\text{dato a enviar}} \rightarrow \overset{BP}{0} \underbrace{01001100}$

_ 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*