2. REPRESENTACIÓN DIGITAL DE LA INFORMACIÓN

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

2. REPRESENTACIÓN DIGITAL DE LA INFORMACIÓN

2.1 Conceptos generales sobre la información.

- Concepto de información y unidad de información.
- Codificación de la Información.

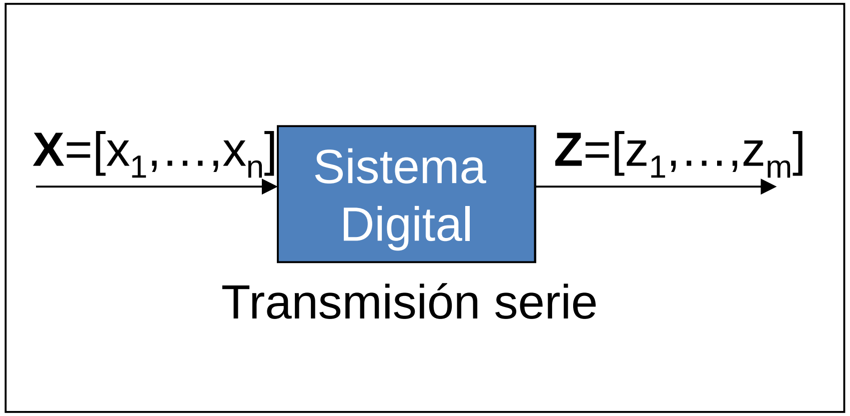
2.2 Sistemas de numeración.

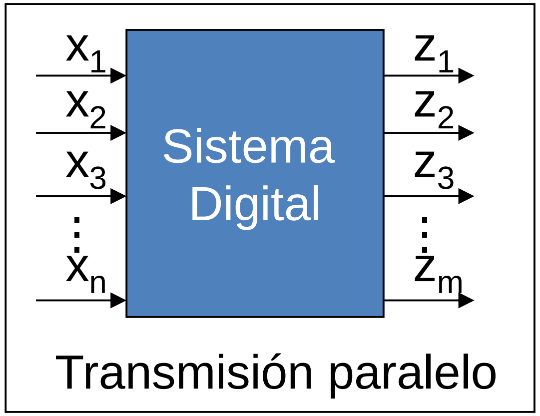
- Sistema de numeración binario.
- Sistema de numeración octal.
- Sistema de numeración hexadecimal.
- Conversión entre sistemas.

2.3 Códigos binarios.

- Código binario natural.
- Códigos decimales codificados en binario: BCD, BCD-Exceso 3.
- Códigos binarios continuos y cíclicos: Gray y Johnson.
- Representación de números con signo.
- Representación de números en coma fija y en coma flotante.
- Códigos alfanuméricos: ASCII.
- Aplicaciones.

- **Concepto de información y unidad de información**
  - En lo sistemas digitales la información se representa y transmite mediante señales de dos niveles, cada una de las cuales se interpreta como 0 y 1 respectivamente





X=[x_1,…,x_n]
donde x_i = 0 ó 1

x_i representa 1 bit de información
X es un elemento de información y contiene n bits de información

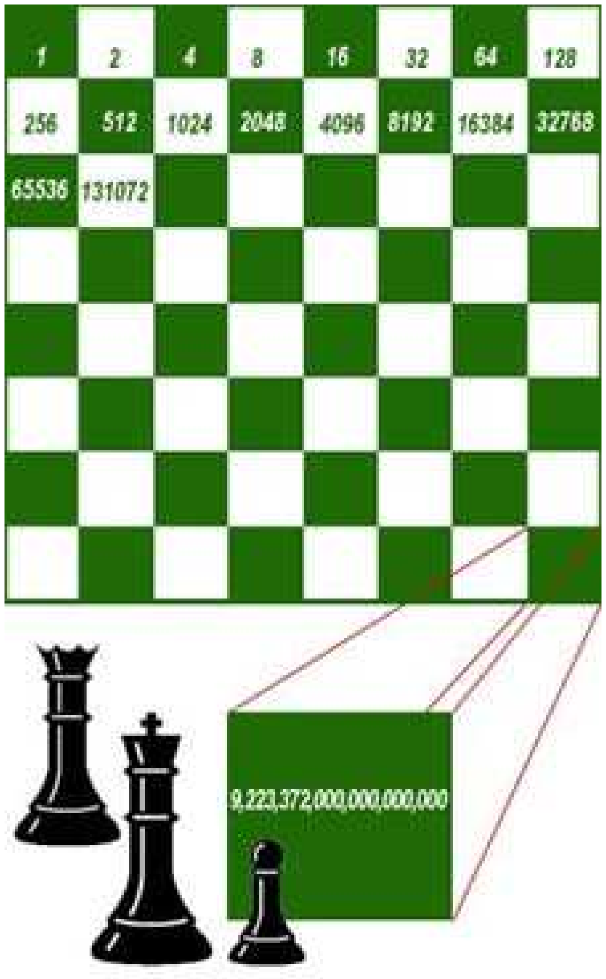
  - **X** puede tomar 2^n valores distintos ⇒ puede representar 2^n elementos distintos de información
  - Ejemplos: n = 1 bit 2^1=2 elementos de información:
    [0] = blanco [1] = negro

    n = 2 bit 2^2=4 elementos de información:
    [00] = norte [01] = sur [10] = este [11] = oeste

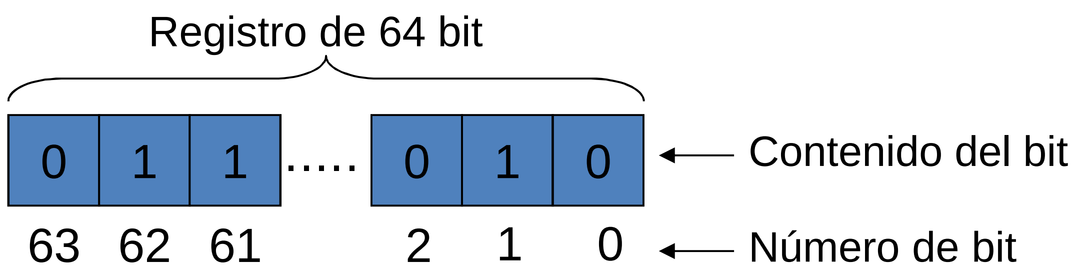
n = 4 bit 2^4=16 elementos
NYBBLE

n= 8 bit 2^8=256 elementos
BYTE

- **Concepto de información y unidad de información**
  - En los sistemas digitales la información se almacena en registros, que son dispositivos o circuitos capaces de almacenar información digital (se verán en el capítulo 6)



  - La leyenda del tablero de ajedrez (http://bit.ly/1pmRk3R)
    - Si se quiere rellenar cada casilla de un tablero de ajedrez con un número de granos de trigo equivalente a elevar 2 al número de la casilla, la casilla 64 equivaldría a más de 20000 años de cosechas mundiales
    - Esa cifra es el número de elementos de información que se pueden representar con 64 bits



    - De ahí que la capacidad de almacenar información en formato digital sea tan grande

- **Codificación de la información**
  - Toda información que tenga que ser procesada mediante circuitos digitales, debe ser previamente codificada
  - **Código**: conjunto de unidades de información relacionadas de forma sistemática y biunívoca con otro conjunto de signos y símbolos según unas determinadas reglas de traducción fijadas de antemano
  - Los códigos utilizados en sistemas digitales son binarios (combinaciones de 0s y 1s)
  - Con n bits se pueden representar 2^n=K elementos distintos y se pueden crear $2^n!$ códigos
  - **Ejemplo:**

n= 2 bits $K = 2^n = 2^2 = 4$ Número de códigos (asignaciones) = $2^2! = 4! = 24$

N	00	00	00	00	00	00	01	01	01	01	01	01	10	10	10	10	10	10	11	11	11	11	11	11
S	01	01	10	10	11	11	00	00	10	10	11	11	00	00	01	01	11	11	00	00	01	01	10	10
E	10	11	01	11	01	10	10	11	00	11	00	10	01	11	00	11	00	01	01	10	00	10	00	01
O	11	10	11	01	10	01	11	10	11	00	10	00	11	01	11	00	01	00	10	01	10	00	01	00

  - **Tipos de codificación de la información**
    - Lógica: cada uno de los n bits representa verdad (1) o falsedad (0) de una sentencia (cada sentencia es independiente)
    - Simbólica: cada grupo de n bits representa un carácter alfanumérico o símbolo especial (A, B, C, 0, +, …). Por ejemplo el '9' = 0111001 en ASCII
    - Numérica: cada grupo de n bits representa un número binario.

- **Aspectos generales**
  - Casi todos los sistemas de numeración son de tipo polinomial:
    - Un número es una expresión formada por un conjunto de símbolos llamados “dígitos” o “cifras”, cada uno con un valor fijo y diferente
    - El número de símbolos distintos es la “base”
    - El valor numérico que indica una combinación de dígitos depende de:
      - El valor de los símbolos o dígitos y
      - de la posición dentro de la combinación
    - La posición del dígito tiene un valor que aumenta de derecha a izquierda según potencias sucesivas de la base
- **Sistemas de numeración usados por el ser humano:**

Sistema decimal o arábigo　　Base 10　　Dígitos: 0,1,2,3,4,5,6,78,9

$N = 123.125_{(10)} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$

La suma es en potencias de 10 y los coeficientes representan el número en decimal

Otros sistemas:

- En el siguiente link puedes encontrar otros sistemas de numeración que se han empleado a lo largo de la historia: http://bit.ly/1pOHhGu

- El más curioso es el maya, de base 20. Un interesante vídeo: http://bit.ly/1nojjwO

- **Sistemas de numeración usados por las máquinas**
  - Sistema binario   Base 2   Dígitos: 0,1

    $N = 123.125_{(10)} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 11110011.001_{(2)}$

    La suma es en potencias de 2 y los coeficientes representan el número en binario

    El anterior número en binario también se puede representar con esta notación: 0b11110011.001

  - Sistema octal   Base 8   Dígitos: 0,1,2,3,4,5,6,7

    $N = 123.125_{(10)} = 1 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0 + 1 \cdot 8^{-1} = 173.1_{(8)}$

    La suma es en potencias de 8 y los coeficientes representan el número en binario

    El anterior número en octal también se puede representar con esta notación: 0o173.1

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

    $N = 123.125_{(10)} = 7 \cdot 16^1 + \overset{B}{11} \cdot 16^0 + 2 \cdot 16^{-1} = 7B.2_{(16)}$

    La suma es en potencias de 16 y los coeficientes representan el número en binario

    El anterior número en hexadecimal también se puede representar con esta notación: 0x7B.2

Nota: desplazar el punto n lugares a la derecha o a la izquierda significa multiplicar o dividir por 10^n, 2^n, 8^n, o 16^n

Ejemplo: $1111011.001_{(2)} = 123.125_{(10)}$

$11110110.01_{(2)} = 246.25_{(10)}$

- **Tabla relacional de los sistemas decimal, binario, octal y hexadecimal**

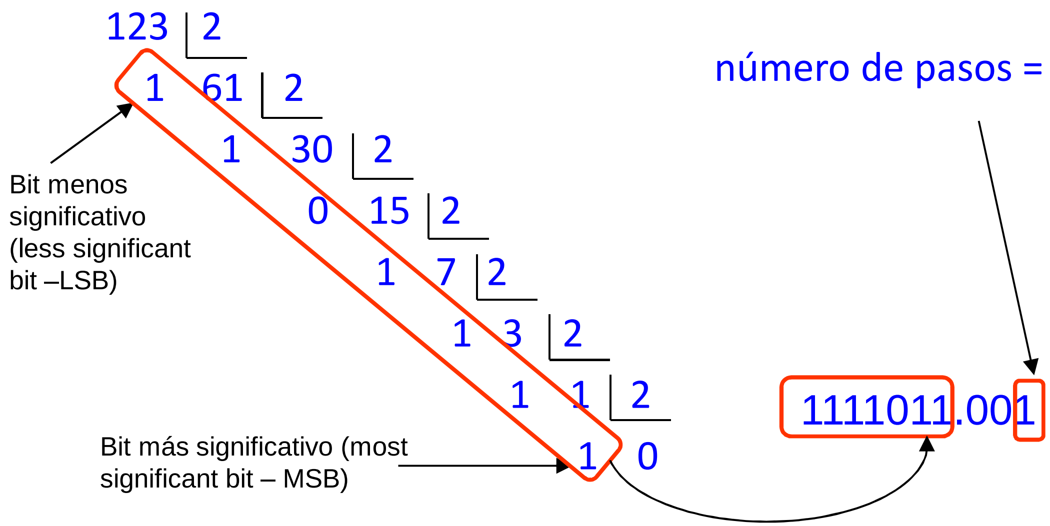
Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12

– Binario es el sistema básico de numeración de los ordenadores
– Octal permite compactar la representación numérica binaria (cada 3 bits se representan mediante un dígito)
– Hexadecimal también compacta (cada 4 bits se representan mediante un dígito)
– La ventaja de hexadecimal es que el número de bits que representa es potencia de 2, y el espacio de memoria de los ordenadores se organiza más eficazmente en binario.
– 2 dígitos hexadecimales son 1 byte, y el byte (8 bit) es la unidad básica en informática.
– Los ordenadores evolucionan hacia bloques de memoria cada vez mayores pero siempre potencia de 2: 16, 32, 64 bit…

- # Conversión entre sistemas

  - ## Conversión decimal – binario

N = 123.125(10)



0.125 x 2 = 0.25 (no llega a la unidad)
0.25 x 2 = 0.5 (no llega a la unidad)
0.5 x 2 = 1 (alcanza la unidad)
1-1=0 (se resta la parte entera y si queda algo se repiten las multiplicaciones por 2)

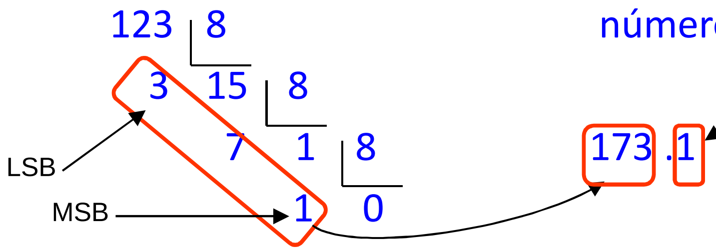
  - ## Conversión binario – decimal

$N = 1111011.001_{(2)} = 1·2^6 + 1·2^5 + 1·2^4 + 1·2^3 + 0·2^2 + 1·2^1 + 1·2^0 + 0·2^{-1} + 0·2^{-2} + 1·2^{-3} = 123.125_{(10)}$

  - ## Conversión decimal – octal

$N = 123.125_{(10)}$

número de pasos = 1 → 0.125 x 8 = 1.0 (alcanza la unidad)
1-1=0 (se resta la parte entera y si queda algo se repiten las multiplicaciones por 8)



- **Conversión entre sistemas**

  – **Conversión octal – decimal**

$N = 173.1_{(8)} = 1{\cdot}8^{2} + 7{\cdot}8^{1} + 3{\cdot}8^{0} + 1{\cdot}8^{-1} = 123.125_{(10)}$

  – **Conversión decimal – hexadecimal**

$N = 123.125_{(10)}$

$$\begin{array}{l|l} 123 & 16 \\ \hline \boxed{11} & 7 \;|\; \underline{16} \\ & \boxed{7} \quad 0 \end{array}$$

LSB

MSB

número de pasos = 1 → 0.125 x 16 = 2.0 (alcanza la unidad)

2-2=0 (se resta la parte entera y si queda algo

se repiten las multiplicaciones por 16)

$\boxed{7B}.\boxed{2}$

→ 11 en hexadecimal es B

  – **Conversión hexadecimal – decimal**

$N = 7B.2_{(16)} = 7{\cdot}16^{1} + 11{\cdot}16^{0} + 2{\cdot}16^{-1} = 123.125_{(10)}$

  – **Conversión binario – octal**

$\underbrace{0\,0\,1}_{1}\;\underbrace{1\,1\,1}_{7}\;\underbrace{0\,1\,1}_{3}\;.\;\underbrace{0\,0\,1}_{1}{}_{(2)} = 173.1_{(8)}$

Conversión binario – hexadecimal

$\underbrace{0\,1\,1\,1}_{7}\;\underbrace{1\,0\,1\,1}_{B}\;.\;\underbrace{0\,0\,1\,0}_{2}{}_{(2)} = 7B.2_{(16)}$

  – **Conversión octal – binario**

$173.1_{(8)} = \underbrace{0\,0\,1}_{1}\;\underbrace{1\,1\,1}_{7}\;\underbrace{0\,1\,1}_{3}\;.\;\underbrace{0\,0\,1}_{1}{}_{(2)} = 1111011.011_{(2)}$

Conversión hexadecimal – binario

$7B.2_{(16)} = \underbrace{0\,1\,1\,1}_{7}\;\underbrace{1\,0\,1\,1}_{B}\;.\;\underbrace{0\,0\,1\,0}_{2}{}_{(2)} = 1111011.011_{(2)}$

  – **Conversión octal – hexadecimal:** $173.1_{(8)} = \overbrace{0\,0\,1}^{1}\;\overbrace{1\,1\,1}^{7}\;\overbrace{0\,1\,1}^{3}\;.\;\overbrace{0\,0\,1}^{1}\,0_{(2)} = 7B.2_{(16)}$

$0\;\underbrace{0\,1\,1\,1}_{7}\;\underbrace{1\,0\,1\,1}_{B}\;.\;\underbrace{0\,0\,1\,0}_{2}$

- **Código binario natural**
  - El sistema binario se llama código binario natural

Decimal	Binario natural				
	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
0					0
1					1
2				1	0
3				1	1
4			1	0	0
5			1	0	1
6			1	1	0
7			1	1	1
8		1	0	0	0
9		1	0	0	1
10		1	0	1	0
11		1	0	1	1
12		1	1	0	0
13		1	1	0	1
14		1	1	1	0
15		1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0

  - Es un código ponderado: a cada posición o cifra binaria se le asigna un peso. El número decimal equivalente se obtiene sumando los pesos de las posiciones que poseen el valor 1
  - Se utiliza en la realización de operaciones aritméticas (suma, resta, producto binario)
  - Además del código binario natural se utilizan otros códigos binarios: BCD, BCD-Exceso 3, Gray, Johnson, etc.
  - **Conversores de código:** circuitos combinacionales que pasan de un código a otro

- **Código decimales codificados en binario (binary coded decimal – BCD)**
  - El ser humano está acostumbrado al sistema de numeración decimal
  - Código BCD: consiste en codificar en binario y por separado cada una de las cifras del número decimal (cada cifra ocupará 4 bits):

Número decimal	Binario natural	BCD	BCD Aiken (2)	BCD Aiken (5)	BCD Exceso 3
Pesos	**8 4 2 1**	**Decenas Unidades** **8 4 2 1 8 4 2 1**	**Decenas Unidades** **2 4 2 1 2 4 2 1**	**Decenas Unidades** **5 4 2 1 5 4 2 1**	**Decenas Unidades**
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	0 1 0 1	1 0 1 1	1 0 0 0	1 0 0 0
6	0 1 1 0	0 1 1 0	1 1 0 0	1 0 0 1	1 0 0 1
7	0 1 1 1	0 1 1 1	1 1 0 1	1 0 1 0	1 0 1 0
8	1 0 0 0	1 0 0 0	1 1 1 0	1 0 1 1	1 0 1 1
9	1 0 0 1	1 0 0 1	1 1 1 1	1 1 0 0	1 1 0 0
10	1 0 1 0	0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0	0 1 0 0 0 0 0 0

  - Los códigos BCD ocupan más bits que el binario natural
  - Todo los anteriores códigos son ponderados menos el BCD Exceso 3
  - Ejemplo: $132_{(10)}$: en binario natural: 10000100 y en BCD natural: $\underbrace{0001}_{1}\ \underbrace{0011}_{3}\ \underbrace{0010}_{2}$

Código binarios continuos y cíclicos

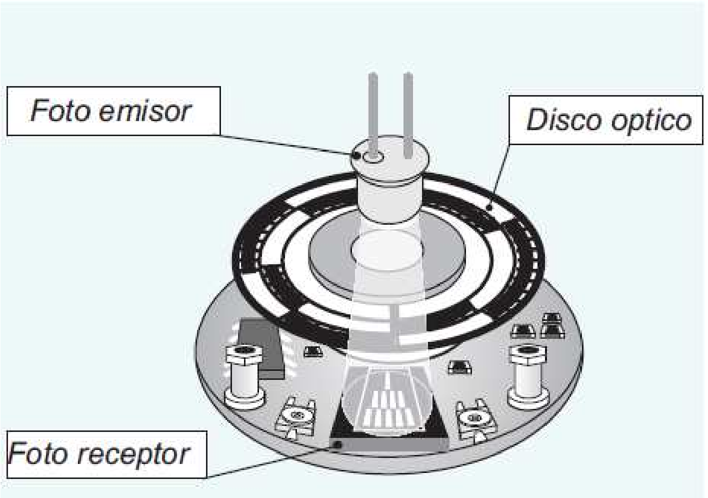
- **Código binario continuo:** si las combinaciones correspondientes a los números decimales consecutivos son adyacentes (difieren solamente en un bit)
- **Código cíclico:** código continuo en que la última combinación es adyacente a la primera
- Los dos códigos más importantes son: Gray y Johnson
- **Código Gray:** es un código continuo y cíclico que se llama también "reflejado":

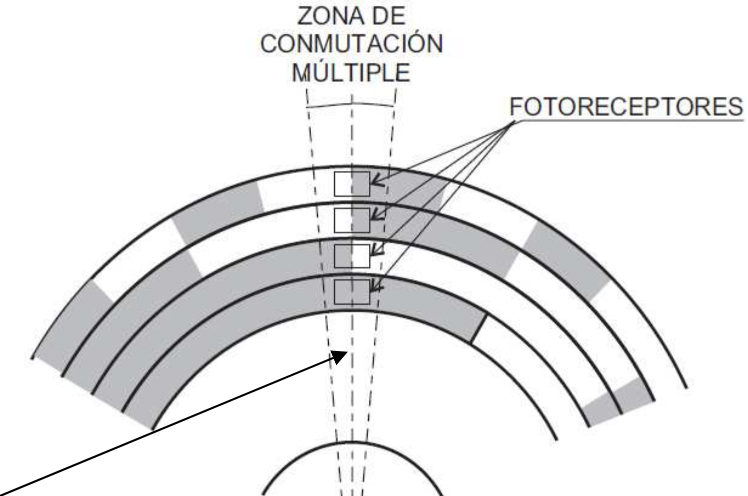
Número decimal	Gray (1 bit)	Gray (2 bits)	Gray (3 bits)	Gray (4 bits)	Binario natural
0	0	00	000	0000	0000
1	1	01 ESPEJO	001	0001	0001
2		11	011	0011	0010
3		10	010 ESPEJO	0010	0011
4			110	0110	0100
5			111	0111	0101
6			101	0101	0110
7			100	0100 ESPEJO	0111
8				1100	1000
9				1101	1001
10				1111	1010
11				1110	1011
12				1010	1100
13				1011	1101
14				1001	1110
15				1000	1111

- Formación de un código Gray de n bits: se parte de n-1 bits y se repiten simétricamente sus combinaciones añadiéndose un 0 en las 2^{n-1} primeras combinaciones y un 1 en las 2^{n-1} siguientes

- ## Aplicación del código Gray
  - Resulta muy útil para codificación de entradas o estados de un sistema cuando estos evolucionan siguiendo una secuencia fija. Al diferenciarse en un solo bit, el paso de una combinación a la siguiente no produce errores debidos a diferencias en el tiempo en la transición o propagación de cada bit
  - **Ejemplo: un encoder absoluto** (http://bit.ly/TdesFx)

    Se trata de un sistema que detecta la posición de un disco en el que hay zonas transparentes y opacas que permiten o no el paso de la luz hacia unos fotoreceptores. La señal recibida por estos será '1' o '0'. Con cuatro fotoreceptores se podrán detectar 2^4 posiciones diferentes





Con código binario natural, el paso del $\boxed{3 \text{ al } 4}$ puede provocar estados intermedios no deseados:

011 ⟶ 000 ⟶ 100

011 ⟶ 111 ⟶ 100

Con código Gray se evitan los estados intermedios:

$\underbrace{010}_{3} \longrightarrow \underbrace{110}_{4}$ (sólo cambia un bit)

• Conversión de código Gray a natural y viceversa

Paso de natural a Gray:

$g_i(\text{gray}) = b_i \oplus b_{i+1}$ (binario)

Ejemplo:

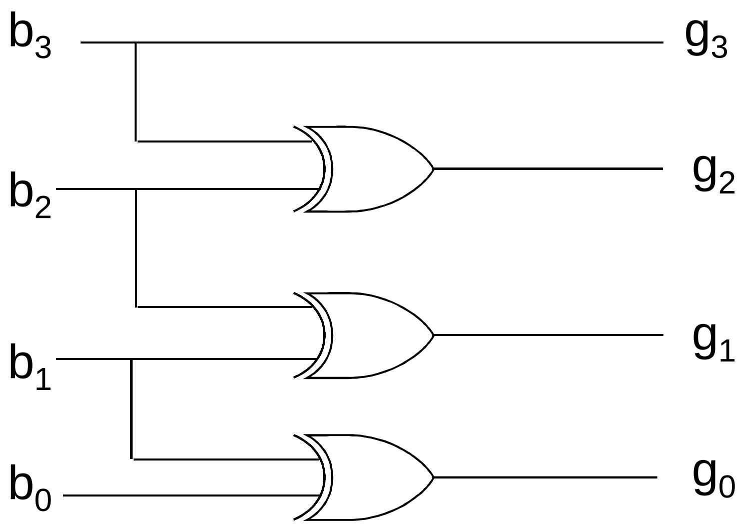
$$\begin{array}{c} b_3\ b_2\ b_1\ b_0 \\ 4 = 0\ 1\ 0\ 0 \end{array} \xrightarrow{g_i = b_i \oplus b_{i+1}} \begin{array}{c} g_3\ g_2\ g_1\ g_0 \\ 0\ 1\ 1\ 0 = 4 \end{array}$$

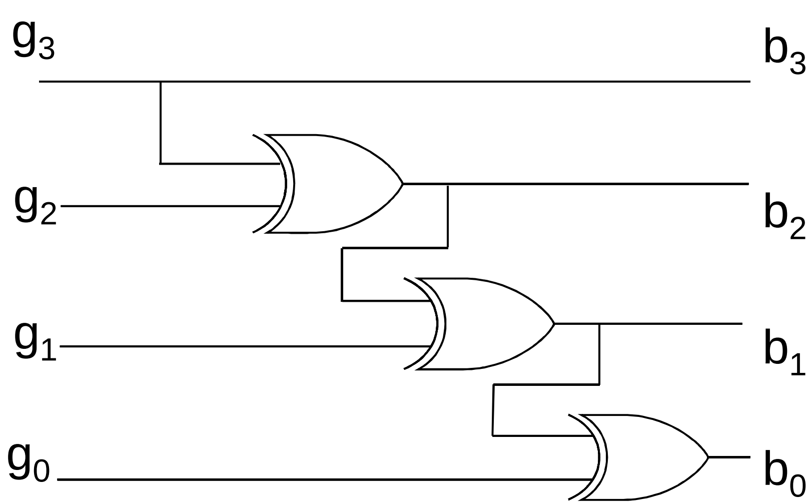
Paso de Gray a natural

bi (binario) = $g_i \oplus g_{i+1} \oplus \ldots \oplus g_{n-1}$ (gray)

Ejemplo

$$\begin{array}{c} g_3\ g_2\ g_1\ g_0 \\ 4 = 0\ 1\ 1\ 0 \end{array} \xrightarrow{b_i = g_i \oplus g_{i+1} \oplus \ldots \oplus g_{n-1}} \begin{array}{c} b_3\ b_2\ b_1\ b_0 \\ 0\ 1\ 0\ 0 = 4 \end{array}$$





• Código Johnson

– Es un código continuo y cíclico que se llama también "progresivo":

Número decimal	Johnson (3 bits)	Johnson (4 bits)	Johnson (5 bits)
0	000	0000	00000
1	001	0001	00001
2	011	0011	00011
3	111	0111	00111
4	110	1111	01111
5	100	1110	11111
6		1100	11110
7		1000	11100
8			11000
9			10000

– Se forma añadiendo 1s y después 0s progresivamente

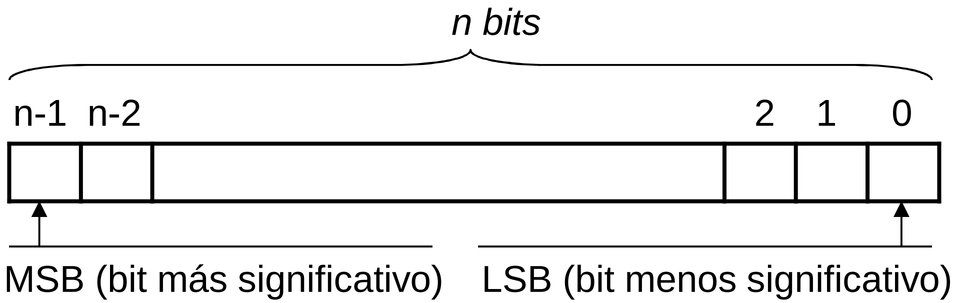
– Con n bits se pueden codificar 2n cantidades diferentes

– Para codificar un número N en código en código Johnson se necesita un número n de bits tal que: $2(n\text{-}1) \leq N < 2n$

– **Inconveniente:** dada su poca capacidad de codificación (binario natural y Gray codifican 2^n cantidades diferentes), no se utiliza en sistemas digitales complejos por implicar una mayor complejidad de los mismos

– **Ventaja:** gran sencillez de diseño de sistemas de conteo bastados en este código

- **Representación de números con y sin signo**
  - Números sin signo:



n bits ⇨ 2^n números diferentes (0 al 2^n-1)

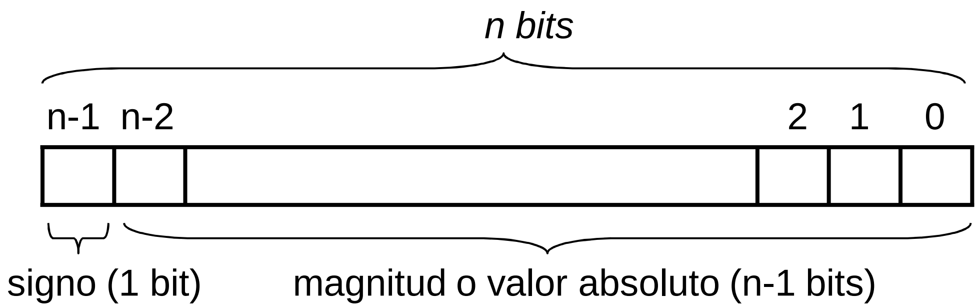
Ejemplo: *n*=8 2^8 = 256 números:

$00000000_{(2)} = 0_{(10)}$

$11111111_{(2)} = 255_{(10)}$

  - Números con signo
    - Convenio signo-magnitud

n bits ⇨ 2^n números diferentes (-(2^{n-1}-1) al +2^{n-1}-1)



Ejemplo: *n*=8 2^8 = 256 números:

$1\ 1111111_{(2)} = -127_{(10)}$

$1\ 0000000_{(2)} = -0_{(10)}$

$0\ 0000000_{(2)} = +0_{(10)}$

$0\ 1111111_{(2)} = +127_{(10)}$

    - La ventaja de este convenio su simplicidad:
      - Bit de signo: si vale 0 el número es positivo y si vale 1 es negativo
      - Magnitud: su valor binario pasado a decimal es la magnitud
    - La desventaja es que requiere de circuitos sumadores y restadores si se suman números, cosa no necesaria con los convenios complemento A1 y A2

- **Representación de números con y sin signo**
  - Convenio complemento A 1 (CA1): se define como el complemento A 1 de un número N (sin signo y representado por n bits) como:

    $CA1(N)=(2^n-1)-N$
  - El complemento A1 se obtiene también de otras dos maneras:

    a) Restando el número de tantos 1s como bits (n) tiene el número:

    $$\begin{array}{rcl} 1111 & = & (2^n-1) \\ -\ 0010 & = & 2 \\ 1101 & = & CA1(2) \end{array}$$

    b) Cambiando 1s y 0s en el número

    $$\boxed{\begin{array}{l} 2 \quad\quad\; = 0010 \\ CA1(2) = 1101 \end{array}} \longrightarrow \text{la más sencilla}$$
  - Para representar números positivos y negativos

    a) número positivo: convenio signo magnitud

    +2 = 00010

    b) número negativo: bit de signo (1) y CA1 de la magnitud

    -2 = 11101 = CA1(+2)

- **Representación de números con y sin signo**
  - Convenio complemento A 2 (CA2): se define como el complemento A 2 de un número N (sin signo y representado por n bits) como:

    $CA2(N)=2^n-N$
  - El complemento A2 se obtiene también de otras dos maneras:

    a) Restando el número de un 1 seguido de tantos 0s como bits (n) tiene el número:

    $$\begin{array}{rcl} 10000 & = & 2^n \\ -\ 0010 & = & 2 \\ 1110 & = & CA2(2) \end{array}$$

    b) Calculando el CA1 y sumando un 1

    $$\boxed{\begin{array}{rl} 2 & = 0010 \\ CA1(2) & = 1101 \\ & \underline{\quad + 1} \\ & 1110 = CA2(2) \end{array}}$$

    la más sencilla
  - Para representar números positivos y negativos

    a) número positivo: convenio signo magnitud

    +2 = 00010

    b) número negativo: bit de signo (1) y CA2 de la magnitud

    -2 = 11110 = CA2(+2)

• Representación de números con y sin signo

Cuadro resumen de los diferentes tipos de numeración con signo para números de 4 bits incluyendo el de signo

Decimal	Signo y magnitud	Complemento a 1	Complemento a 2
+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	N/A
0	N/A	N/A	0000
−0	1000	1111	N/A
−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

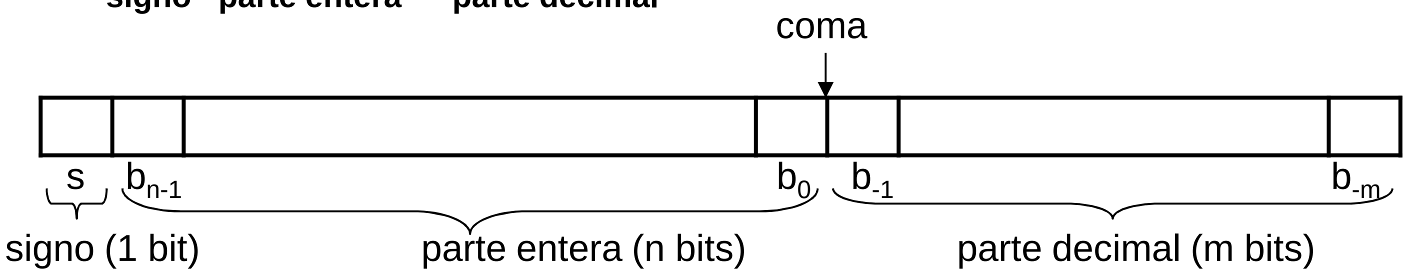
• Representación de números racionales

Un número racional tiene una parte entera y otra decimal o fraccionaria

a) Representación en coma fija

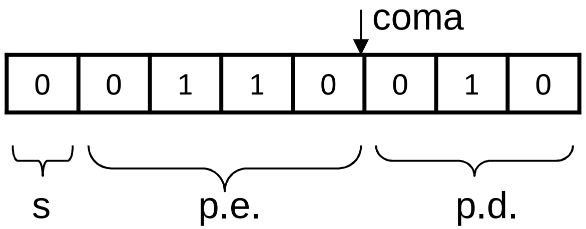
– Los números se representan con un número fijo de cifras decimales:

$$N(2) = \underbrace{(\pm)}_{\textbf{signo}}\ \underbrace{b_{n-1} \cdots b_0}_{\textbf{parte entera}} \cdot \underbrace{b_{-1} \cdots b_{-m}}_{\textbf{parte decimal}}$$



– **Ejemplo:**

Considerando un registro de 8 bits (1 byte), donde 1 bit es de signo, 4 de parte entera y 3 de parte decimal, el número $6.25_{(10)} = 110.01_{(2)}$ se representa como:



– **Para practicar:** representa el -6.5, el 8.75 y el -11.2 en un registro de 16 bit con 1 bit de signo 8 de parte entera y 7 de parte decimal. Después comprueba que se ha realizado bien la conversión obteniendo el número decimal a partir del número en coma fija

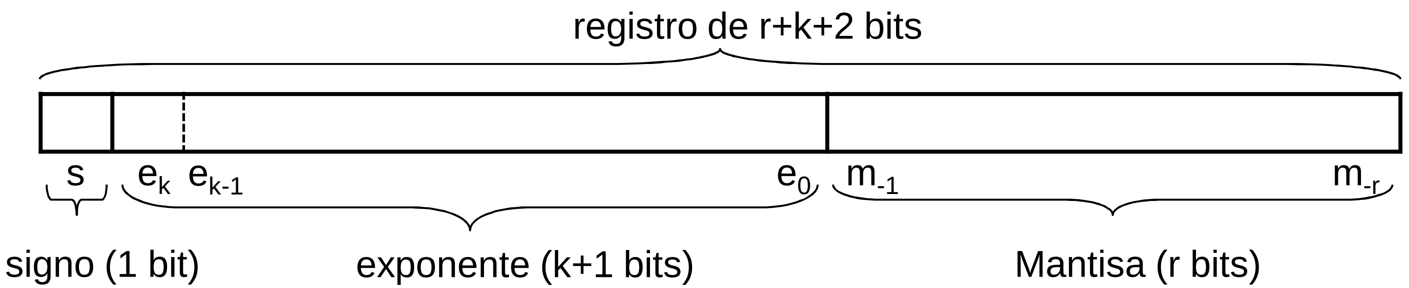
• Representación de números racionales

b) Representación en coma flotante (notación científica)

– Se utiliza para representar números grandes o pequeños

– Los números se representan en forma exponencial (base del exponente 2):

$$N(2) = \underbrace{(\pm)}_{\textbf{signo}}\ \underbrace{m_{-1}\cdot\cdot\cdot m_{-r}}_{\textbf{mantisa}} \cdot 2^{\underbrace{\pm e_{k-1}\cdot\cdot\cdot e_0}_{\textbf{exponente}}}$$



– Se siguen diversos estándares. Veremos el **Estándar ANSI IEEE 754 precisión simple:**

- Signo: 0 para números positivos y 1 para negativos

- Exponente: 8 bits con exceso 127 (al exponente de la forma exponencial se le suma 127)

- Mantisa: la mantisa de la forma exponencial se representa como el 1 seguido de la coma y de 23 cifras. Estas 23 cifras son las que se representan en el campo mantisa del registro

Ejemplo: para representar el 6.25 en ANSI IEEE 754 precisión simple:

$6.25_{(10)} = 110.01_{(2)} = 1.1001_{(2)}\ \text{x}2^2$ → Exponente: $2+127=129_{(10)} = 10000001_{(2)}$

Mantisa: $10010000000000000000000_{(2)}$

s	exponente					mantisa					
0	1	0	......	0	1	1	0	0	1	0	....	0

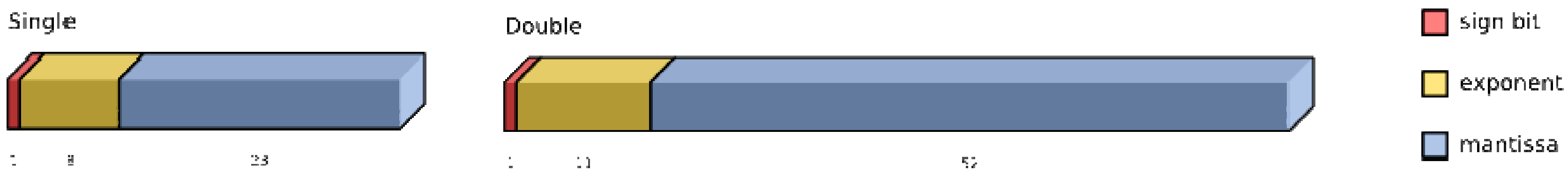
• Representación de números racionales

Más ejemplos de representación en coma flotante ANSI 754 single

Representación en precisión simple (IEEE Standard) S Exponente Mantisa	Signo	Exponente (en exceso a -127)	Valor binario (representación con coma)	Valor decimal
0 10000000 00000000000000000000000	+	128-127=**1**	+1.0	+2
1 01111111 11000000000000000000000	-	127-127=**0**	-1.11	-1.75
0 10000001 10100000000000000000000	+	129-127=**2**	+110.1	
1 01111111 11100000000000000000000	-	127-127=**0**		
1 10000010 01000100000000000000000	+			

Para hacer más ejemplos, convertidor online entre binarios y formatos de precisión simple, doble y otros: **www.binaryconvert.com**

Comparativa de ANSI 754 single con double:



Tipo de precisión	Signo (1 bit)	Exponente	Mantisa	Menor Valor	Mayor Valor
Single	0=positivo 1= negativo	8 bits exceso a 127	23+1	1.5×10^{-45}	$3.4 \times 10^{+38}$
Doble	0=positivo 1= negativo	11 bits exceso a 1023	52+1	5.0×10^{-324}	$1.7 \times 10^{+308}$

Ejemplo de programa en PASCAL

```
PROGRAM Test;
VAR
   x : REAL;          { variable name is x, type is real
   i : INTEGER:       { variable name is i, type is integer
   c : CHAR;          { variable name is c, type is character
   s : STRING;        { variable name is s, type is string
BEGIN
    x := -34.55;      { valid real number assigned to variable x }
    x := -3.9E-3;     { valid real number assigned to variable x }
    WRITELN(x);       { x contains the value -3.9E-3 }
    i := 10;          { valid integer number assigned to variable i }
    i := i * i;       { valid (!) - i will be 100 now }
    i := 9933;        { valid integer number assigned to variable i }
    i := -99999;      { invalid integer - too small }
    i := 999.44;      { invalid assignment - types do not match }
    c := '1';         { valid character assigned to variable c }
    c := 1;           { invalid assignment - types do not match }
    c := 'Bert';      { invalid assignment - types do not match }
    c := 'd';         { valid character assigned to variable c }
    WRITELN(c);       { c contains the value 'd' }
    d := 'c';         { unknown variable - the variable d is not declared }
    WRITELN(s);       { invalid reference - s has undefined value }
END.
```

El INTEGER es un registro de 16 bit con un bit de signo y convenio signo magnitud

El REAL es en notación decimal como la coma fija y en científica como la coma flotante

El BOOLEAN es un bit que puede valer true of false (1 o 0)

El STRING es un array de registros de 8 bit sin signo

Tipos de variables en Pascal

INTEGER	A positive or negative integer between a smallest (negative) and a largest number. In general the smallest and largest number possible depends on the machine; for IBM PC and Turbo Pascal they are: **smallest Integer:** -32766 **largest Integer:** 32767
REAL	Can contain a real number in scientific or decimal notation. There is a limit on the size and accuracy of the real number that will be covered later. Valid real numbers are, for example: **Decimal Notation:** 1.234 or -34.5507 **Scientific Notation:** 5.0E-3 or -7.443E3
CHAR	Any key on the keyboard is considered a valid character. Characters are usually enclosed in single quotes. For example: '1' is a character, while 1 is an integer.
BOOLEAN	We will deal with boolean variables later
STRING	A string is a collection of up to 255 characters enclosed in single quotes. For example: 'Bert' is a string of 4 characters. More details about strings will follow later.

Códigos binarios

Ejemplo de programa en C

```
#include
int i = 0;
int j = 2 + 2;
int k = 2 * (3 << 8) / 3;
int m = (int)(&i + 2);
int p = sizeof(float) * 2;
int q = sizeof(p);
float r = (float)(2 * 3);
main () {
  printf ("i = %i\n", i);
  printf ("j = %i\n", j);
  printf ("k = %i\n", k);
  printf ("m = %i\n", m);
  printf ("p = %i\n", p);
  printf ("q = %i\n", q);
  printf ("r = %f\n", r);
  for (r = 0.0; r < 1.0; r += 0.1) {
    double s = sin(r);
    printf ("The sine of %f is %f\n", r, s);
  }
}
```

El char es un registro para números enteros de 8 bit sin signo o con signo

El short e int son registros para números enteros de 16 bit o 32 bit sin signo o con signo

El float y el double son registros para números en coma flotante ANSI 754 single o double

El bool es un bit que puede valer true of false (1 o 0)

Tipos de variables en lenguaje C

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1 bit	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 *or* 4 bytes	1 wide character

- **Códigos alfanuméricos: ASCII**
  - ASCII es el código alfanumérico más empleado
  - Originariamente cada carácter o símbolo se representaba mediante 7 bits, lo que servía para abarcar el alfabeto latino, pero se amplió a 8 bits para incluir un mayor abanico de símbolos
  - A continuación se muestra la tabla para 7 bits, que es internacional

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

  - Para incluir los símbolos del alfabeto chino se usa un mayor número de bits por carácter: http://bit.ly/1lGQb5z

- **Códigos detectores de error**
  - En el manejo y transmisión de la información se pueden producir errores (cambio de bits). Esto obliga a la creación de códigos capaces de detectar o corregir errores
  - Condición necesaria para que un código binario de n bits sea detector de errores:
    - no utilizar todas las combinaciones posibles (2^n), ya que una combinación del código se transformará en otra que también pertenece a él
  - La condición anterior es necesaria pero no suficiente:
    - Ejemplo: El código BCD
      - Utiliza para cada cifra decimal 4 bits (esto supone utilizar 10 de las 16 combinaciones posibles)
      - 2(10)=0010(2) $\xrightarrow{\text{cambiando el último bit}}$ 0011(2)=3 ⇨ Cambiando un bit se ha pasado de detectar el 2 a detectar el 3, con lo que se creerá que hay un 3 cuando debería haber un 2
  - Condición necesaria y suficiente: **que la distancia mínima del código sea superior a la unidad**
    - Distancia mínima de un código: la menor distancia entre dos combinaciones cualesquiera pertenecientes al mismo
    - Distancia entre dos combinaciones de un código: número de bits de una de ellas que hay que modificar para obtener la otra

BCD: 2 = 0 0 1 $\boxed{0}$ / 3 = 0 0 1 $\boxed{1}$ → d=1   3 = 0 $\boxed{0}$ $\boxed{1}$ 1 / 5 = 0 $\boxed{1}$ $\boxed{0}$ 1 → d=2 ⇨ d_{min} BCD = 1 ⇨ No es código detector de error

  - **En general el número de bits erróneos que un código puede detectar es:** $\boxed{\text{nº bits erróneo} = d_{min} - 1}$

Ejemplos de códigos detectores de error

a) Códigos de paridad

– Se obtienen partiendo de un código de distancia mínima 1 al que se le añade un bit llamado de paridad

– **Código de paridad impar:** se añade al código de partida un bit de forma que el número de unos de cada combinación resultante sea impar:

Número decimal	BCD Exceso 3
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

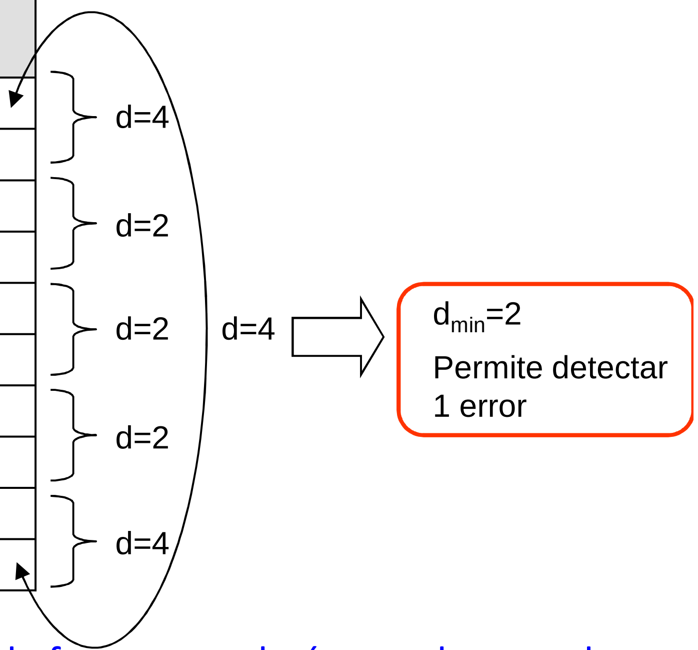
+

Bit paridad impar
1
0
1
1
0
0
1
1
0
1

=

d=2
d=2
d=2
d=2

BCD Exceso 3
0 0 1 1 1
0 1 0 0 0
0 1 0 1 1
0 1 1 0 1
0 1 1 1 0
1 0 0 0 0
1 0 0 1 1
1 0 1 0 1
1 0 1 1 0
1 1 0 0 1



– **Código de paridad par:** se añade al código de partida un bit de forma que el número de unos de cada combinación resultante sea par

– **Conclusión:** la detección consiste en comprobar si el número de 1s de cada combinación es par (paridad par) o impar (paridad impar). Existen circuitos generadores y detectores de bit de paridad

Ejemplos de códigos detectores de error

b) Códigos de peso constante

– Todas las combinaciones tienen el mismo número de 1s

Número decimal	Código 2 entre 5	Código biquinario
Pesos		5 0 4 3 2 1 0
0	0 1 1 0 0	0 1 0 0 0 0 1
1	1 1 0 0 0	0 1 0 0 0 1 0
2	1 0 1 0 0	0 1 0 0 1 0 0
3	1 0 0 1 0	0 1 0 1 0 0 0
4	0 1 0 1 0	0 1 1 0 0 0 0
5	0 0 1 1 0	1 0 0 0 0 0 1
6	1 0 0 0 1	1 0 0 0 0 1 0
7	0 1 0 0 1	1 0 0 0 1 0 0
8	0 0 1 0 1	1 0 0 1 0 0 0
9	0 0 0 1 1	1 0 1 0 0 0 0

Bi Quinario

– **Código biquinario:** es un código ponderado de 7 bits dividido en dos partes:
  - Bi (2 bits): indica si el número está por encima o por debajo de 5
  - Quinario (5 bits): el 1 va desplazándose progresivamente hacia bits de mayor peso al aumentar el valor

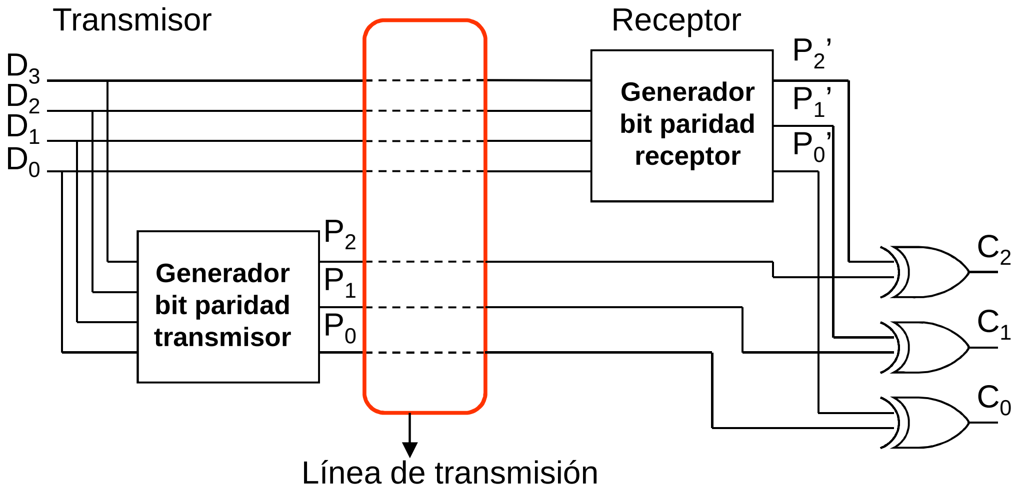
– Tanto el biquinario como el 2 entre 5 son códigos con distancia mínima 2, luego permiten detectar 1 error. Se plantea como ejercicio comprobar que efectivamente tienen d_{min}=2

- ## Códigos correctores de error
  - Para que un código pueda corregir los bits erróneos su distancia mínima debe ser mayor que 2.
  - La ecuación para calcular el número de errores que puede detectar un código es:

$\boxed{d_{min}=2\cdot(\text{nº de errores a corregir})+1}$ ⇒ *para detectar 2 errores dmin=5, para 3 errores dmin=7, etc.*

  - **Ejemplo: Código Hamming**

Número decimal	D3	D2	D1	P2	D0	P1	P0
	b7	b6	b5	b4	b3	b2	b1
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0



$$C_0 = P_0 \oplus P_0' = P_0 \oplus D_0 \oplus D_1 \oplus D_3$$
$$C_1 = P_1 \oplus P_1' = P_1 \oplus D_0 \oplus D_2 \oplus D_3$$
$$C_2 = P_2 \oplus P_2' = P_2 \oplus D_1 \oplus D_2 \oplus D_3$$

$$P_0 = D_0 \oplus D_1 \oplus D_3$$
$$P_1 = D_0 \oplus D_2 \oplus D_3$$
$$P_2 = D_1 \oplus D_2 \oplus D_3$$

P_i bit de paridad
D_i bit de datos

$\boxed{d_{min} = 3}$

Ejemplo (el código de C_2, C_1, C_0 indica el número del bit erróneo):

Número decimal	D3	D2	D1	P2	D0	P1	P0
	b7	b6	b5	b4	b3	b2	b1
Transmitido	1	1	1	1	1	1	1
Recibido	1	1	1	1	1	1	0

$$C_2 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$C_1 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$C_0 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

b_1 erróneo