
**CONCEPTOS BÁSICOS SOBRE
CODIFICACIÓN Y ALMACENAMIENTO DE
LA INFORMACIÓN**

Javier Portillo Berasaluce
Departamento de Ingeniería de Sistemas y Automática
Escuela de Ingenieros de Bilbao

1.	SISTEMAS DE REPRESENTACIÓN DE LA INFORMACIÓN	1
1.1	LOS SISTEMAS DE NUMERACIÓN	1
1.1.1	<i>El sistema decimal</i>	2
1.1.2	<i>Teorema fundamental de la numeración</i>	2
1.1.3	<i>El sistema binario</i>	2
1.1.4	<i>El sistema octal</i>	4
1.1.5	<i>El sistema hexadecimal</i>	4
1.1.6	<i>Convenios para la representación de las bases</i>	4
1.1.7	<i>Conversiones entre los sistemas de numeración</i>	4
1.1.7.1	$N_{(10)} \rightarrow A_{(B)}$	4
1.1.7.2	$A_{(B)} \rightarrow N_{(10)}$	5
1.1.7.3	$A_{(m)n} \rightarrow C_{(m)}$	6
1.1.7.4	$A_{(m)} \rightarrow C_{(m)^n}$	6
1.1.7.5	Conversiones más habituales	7
1.1.8	<i>Representación de números enteros</i>	8
1.1.8.1	Representación con signo-magnitud	8
1.1.8.2	Representación en complemento a 1	8
1.1.8.3	Representación en complemento a 2	9
1.2	SISTEMAS DE CODIFICACIÓN	9
1.2.1	<i>Códigos binarios numéricos</i>	10
1.2.1.1	<i>Códigos BCD (Binary-Coded Decimal)</i>	10
1.2.2	<i>Códigos alfanuméricos</i>	11
2.	ALMACENAMIENTO DE DATOS	12
2.1	UNIDADES DE INFORMACIÓN	12
2.1.1	<i>BIT</i>	12
2.1.2	<i>BYTE</i>	12
2.1.3	<i>NIBBLE</i>	12
2.1.4	<i>WORD (PALABRA)</i>	12
2.1.5	<i>DOBLE PALABRA</i>	12
2.2	ALMACENAMIENTO DE DATOS EN LAS UNIDADES DE INFORMACIÓN	12
2.2.1	<i>Coma fija sin signo (Binario Puro)</i>	13
2.2.2	<i>Complemento a 2</i>	13
	REFERENCIAS	15
	EJERCICIOS	16

1. SISTEMAS DE REPRESENTACIÓN DE LA INFORMACIÓN

La información, para ser procesada por las máquinas (ordenadores, autómatas programables u otras), debe ser representada mediante símbolos diferentes a los que habitualmente empleamos los humanos, ya que las capacidades de las máquinas difieren de las de los humanos. Los sistemas de representación de la información deben permitir expresar ésta en términos apropiados para que sea entendida y procesada por las máquinas.

Para ser capaces de expresar datos en formatos diferentes a los habituales, vamos a profundizar un poco en los siguientes conceptos:

- *Sistemas de numeración.* Permiten la representación de datos numéricos en función de diferentes reglas.
- *Sistemas de codificación.* Utilizados para expresar la información (números y caracteres) que manejamos habitualmente, en un formato adecuado para su procesamiento a través de máquinas.

1.1 LOS SISTEMAS DE NUMERACIÓN

Podemos definir un sistema de numeración como el conjunto de símbolos y reglas que se utilizan para la representación de cantidades.

La evolución de estas técnicas a lo largo de la historia se puede resumir en estos hitos:

- La primera representación de los números podría ser el uso de ‘palitos’ ó líneas que se dibujaban en la tierra, o en la roca, para mantener la cuenta de algún evento. Evidentemente, este sistema no es adecuado para representar números muy pequeños o muy grandes.
- La aparición de un símbolo que representaba el ‘10’ (año 3.400 A.C. en Egipto, 3.000 A.C. en Mesopotamia) supuso una revolución matemática porque disminuía el número de símbolos necesarios (por ejemplo, para representar el 14 bastaba el símbolo del ‘10’ y 4 unidades).
- El sistema de numeración arábigo (desarrollado en la India sobre el 300 A.C.) es el más utilizado hoy en día
- La introducción del símbolo ‘0’ supuso otro hito porque se empezó a usar para indicar el valor posicional de los dígitos. Esto hizo posible el concepto de agrupamiento, el nuevo símbolo permitía expresar grupos de 10 unidades, 100 unidades, etc.

En definitiva, hoy en día nos regimos por el sistema decimal y con un conjunto básico de 10 símbolos (0 a 9), que repetimos formando diferentes agrupaciones, podemos formar números muy grandes o muy pequeños. Un gran avance comparado con el sistema de ‘palitos’. Sin embargo, en la era digital nos hemos visto en la necesidad de

reducir ese conjunto de 10 símbolos a tan sólo dos (el '0' y el '1') para hacernos entender por las máquinas que utilizan un sistema binario.

En este capítulo nos familiarizaremos con diferentes sistemas de numeración y deberemos ser capaces de pasar números de un sistema a otro. Vamos a utilizar *Sistemas Posicionales* de numeración (decimal, binario, etc). En los Sistemas Posicionales un número viene dado por una cadena de dígitos, estando afectado cada uno de los dígitos por un factor de escala (o peso) que depende de la posición que ocupa el dígito dentro de la cadena dada. En estos sistemas existe un elemento característico que define el sistema y se denomina BASE, siendo ésta el número de símbolos que se utilizan para dicha representación.

Un ejemplo de sistema 'no posicional' es el sistema de numeración romano.

1.1.1 El sistema decimal

El sistema decimal utiliza diez dígitos o símbolos (del 0 al 9) con un valor absoluto y una posición relativa. Cuando hemos utilizado todos los símbolos (para valores mayores a 9) tenemos que usar varios dígitos para poder representar cantidades mayores. Cada cifra situada a la izquierda de otra vale 10 veces más que ésta.

1.1.2 Teorema fundamental de la numeración

Todos los sistemas de numeración posicionales toman como referencia el punto decimal y tienen una base de numeración que, de forma implícita, interviene en la cantidad que con una determinada representación se quiere referenciar.

El teorema fundamental de la numeración relaciona una cantidad expresada por $A_n A_{n-1} \dots A_1 A_0 A_{-1} \dots A_{-p}$ en cualquier sistema de numeración (B) con la misma cantidad expresada en el sistema decimal. El cálculo se realiza con el **polinomio equivalente**

$$N = A_n B^n + A_{n-1} B^{n-1} + \dots + A_0 B^0 + \dots + A_{-p} B^{-p} = \sum_{i=0}^n A_i B^i + \sum_{i=1}^p A_{-i} B^{-i} = \sum_{i=-p}^n A_i B^i$$

- B: Base del sistema de numeración.
Número máximo de símbolos que puede tener el sistema.
- A_i: Coeficiente.
Es un símbolo del sistema y cumple que $0 \leq A_i < B-1$
- N+1: Número de dígitos enteros
- P: Número de dígitos fraccionarios.

Ejemplo: $312.2_{(4)} = 3 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0 + 2 \cdot 4^{-1} = 3 \cdot 16 + 4 + 2 + 0.5 = 54.5_{(10)}$

1.1.3 El sistema binario

El sistema binario, o sistema de numeración en base dos, fue introducido por Leibniz en el siglo XVII, siendo el más adecuado para usar en las máquinas electrónicas, debido a que utilizan esencialmente sistemas de dos estados estables: encendido y apagado. En el sistema binario los datos se representan en un sistema que sólo admite dos estados: 0 y 1.

El término BIT nace como contracción de los vocablos ingleses "BInary digiT" (dígito binario). Por tanto, se puede definir el BIT como "la unidad mínima de información comprensible por el ordenador".

La numeración binaria es muy útil para ser empleada por las máquinas; sin embargo es bastante incómoda para los usuarios por necesitar muchas cifras para representar un número, por este motivo utilizaremos sistemas de numeración mas condensados: el octal y el hexadecimal.

Con P bits podemos representar 2^P números (desde el 0 hasta el 2^P-1). Esto nos permite saber el número de bits que necesitamos para representar el número decimal N:

$$2^P - 1 \geq N \rightarrow P \geq \log_2(N + 1)$$

Si queremos representar el número 7 necesitamos 3 bits:

$$P \geq \log_2(7 + 1)$$

$$P \geq 3$$

Si queremos representar el número 8 necesitamos 4 bits:

$$P \geq \log_2(8 + 1)$$

$$P \geq 3.17 \rightarrow P = 4$$

(primer número entero que cumple)

Por lo tanto, aplicando el Teorema Fundamental de la numeración al sistema binario, vemos que el peso de cada bit es determinado por su posición (el bit de más a la derecha tiene un peso de $2^0=1$, el siguiente $2^1=2$, el siguiente $2^2=4$, y así sucesivamente) y las conversiones de binario a decimal se realizan teniendo en cuenta los pesos:

$$1100110_2 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^2 + 1 \times 2^1 = 102_{10}$$

La suma binaria es semejante a la suma decimal, con la diferencia de que se mantienen solo dos dígitos (0 y 1), de tal forma que cuando el resultado excede de los símbolos utilizados se agrega el exceso (denominado acarreo) a la suma parcial siguiente hacia la izquierda.

Tabla del 0	Tabla del 1
0+0=0	1+0=1
0+1=1	1+1=10 (0 con acarreo1)

Repasemos la aritmética de la suma para analizar el proceso con detalle.

Suma decimal: $2334 + 192 = 2526$
Suma binaria: $1010 + 011 = 1101$ ($10 + 3 = 13$)

1.1.4 El sistema octal

Este sistema de numeración es un sistema posicional que utiliza 8 símbolos (del 0 al 7) para representar las cantidades. La aritmética de este sistema es similar a la de los sistemas decimal y binario.

1.1.5 El sistema hexadecimal

Este sistema de numeración es un sistema posicional que utiliza 16 símbolos (del 0 al 9 y A, B, C, D, E y F) para representar las cantidades. Los símbolos A, B, C, D, E y F tienen el siguiente valor absoluto:

Símbolo	A	B	C	D	E	F
Valor absoluto	10	11	12	13	14	15

La aritmética de este sistema es similar a la de los sistemas decimal y binario.

1.1.6 Convenios para la representación de las bases

Para identificar en qué sistema numérico está representado un dato se suele usar bien una letra minúscula como sufijo (d, b, h, o), o la base en subíndice (a veces cerrada con un paréntesis). Si no tiene ningún identificativo se sobrentiende que es decimal.

Para los números hexadecimales, los compiladores suelen emplear el prefijo '0x'. Además, suelen exigir que haya un '0' delante del número hexadecimal en el caso de que el primer dígito sea una letra (ejemplo 0x0FA34).

$56_{10} = 56_{(10)} = 56d = 56$
 $101001_2 = 101001_{(2)} = 101001b$
 $56ED_{16} = 56ED_{(16)} = 56EDh = 0x56ED$
 $56_8 = 56_{(8)} = 56o$

1.1.7 Conversiones entre los sistemas de numeración

1.1.7.1 $N_{(10)} \rightarrow A_{(B)}$

Para convertir un número N expresado en el sistema decimal al sistema en base B utilizaremos el método de división-multiplicación. Se convierten por separado las partes entera y fraccionaria del número:

- **Parte entera.** Se realiza la división entera tomando la parte entera de N como dividendo y la base B como divisor. Cogiendo como dividendo el cociente de la operación anterior se repite el proceso hasta que el resto obtenido sea menor que la base B. El último cociente y todos los restos forman la parte entera del número N expresado en la base B.
- **Parte fraccionaria.** La parte fraccionaria del número N se multiplica por B. La parte entera del resultado será la siguiente cifra de la representación de N en la base B. Con la parte fraccionaria se repite el proceso hasta que la parte fraccionaria sea 0 ó hasta que aparezca repetida alguna parte fraccionaria. En este último caso el número expresado en la base B será periódico.

Ejemplo: Convertir el número $453,140625_{(10)}$ (expresado en base 10 o decimal) a base 8.

Parte entera		Parte fraccionaria		
453	8	$0,140625 \times 8$	= 1,125	--> 1
53	56	$0,125 \times 8$	= 1,0	--> 1
5	0			
	7			

Por lo tanto $453,140625_{(10)} = 705,11_{(8)}$

También podríamos utilizar el método de las restas sucesivas, que consiste en buscar la potencia de B más grande que se pueda restar del número a convertir, tomando como nuevo número el resultado de la resta.

Ejemplo: Convertir el número $540,25_{(10)}$ a binario

potencia	1024	512	256	128	64	32	16	8	4	2	1	0,5	0,25
posición	10	9	8	7	6	5	4	3	2	1	0	-1	-2

540,25	-	512 (2^9)	=	284,25
284,25	-	256 (2^8)	=	28,25
28,25	-	16 (2^4)	=	12,25
12,25	-	8 (2^3)	=	4,25
4,25	-	4 (2^2)	=	0,25
0,25	-	0,25 (2^{-2})	=	0

posición	10	9	8	7	6	5	4	3	2	1	0	-1	-2
dígito	0	1	1	0	0	0	1	1	1	0	0	0	1

Por lo tanto $540,25_{(10)} = 1100011100,01$

1.1.7.2 $A_{(B)} \rightarrow N_{(10)}$

Para convertir un número A expresado en cualquier base B al sistema decimal utilizaremos el polinomio equivalente.

Ejemplo: Convertir el número 705,11₈ (expresado en base 8 o octal) a base 10:

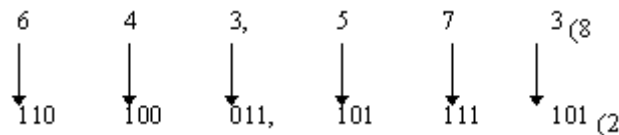
$$7 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 + 1 \cdot 8^{-1} + 1 \cdot 8^{-2} = 7 \cdot 64 + 0 + 5 + 0,125 + 0,015625 = 453,1406255$$

Por lo tanto, $705,11_8 = 453,1406255_{10}$

1.1.7.3 $A_{(m^n)} \rightarrow C_{(m)}$

Para convertir un número representado en base m^n a base n (bases relacionadas por potencias), se convertirán independientemente cada uno de los dígitos a base n utilizando siempre n bits.

Ejemplo: De base 8 ($=2^3$) a base 2. Convertir el número 643,573₈ a base 2 (binario).

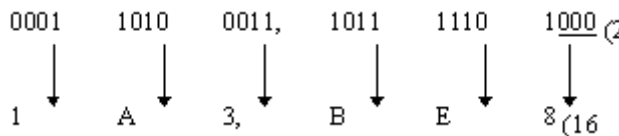


Por lo tanto, $643,573_8 = 110\ 100\ 011, 101\ 111\ 101_2$

1.1.7.4 $A_{(m)} \rightarrow C_{(m^n)}$

Para convertir un número representado en base m a base m^n (bases relacionadas por potencias), se agrupan los bits de n en n hacia la derecha y hacia la izquierda de la coma y se calcula independientemente el polinomio equivalente de cada uno de los grupos.

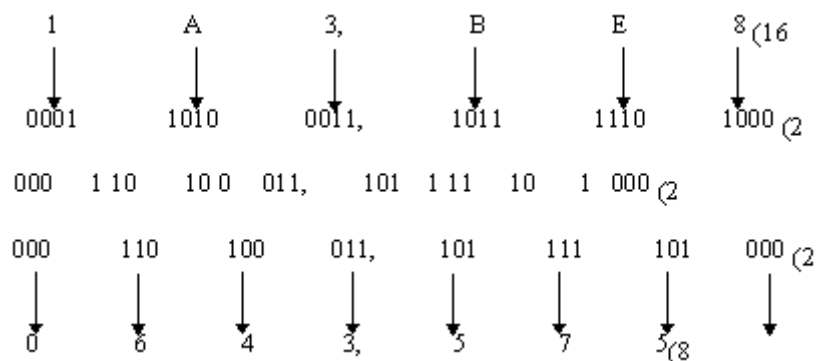
Ejemplo: Convertir el número 000110100011,101111101₂ (binario) a base 16 (hexadecimal)



Observar que hemos añadido 000 a la derecha del último 1 de la parte fraccionaria para tener cuatro bits.

Por lo tanto, $000110100011,101111101_2 = 1A3,BE8_{16}$

Ejemplo: Convertir el número 1A3,BE8₁₆ (hexadecimal) a base 8 (octal).



Por lo tanto, $1A3,BE8_{(16)} = 643,575_{(8)}$

1.1.7.5 Conversiones más habituales

Los sistemas de numeración que más emplearemos serán el binario y el hexadecimal (aparte del decimal) y debemos realizar las conversiones entre estas bases con agilidad. Como ya se ha comentado, las máquinas usan una base binaria pero con el uso de la base hexadecimal se compacta la información porque 4 dígitos binarios se corresponden con uno hexadecimal.

BINARIO → HEXADECIMAL

Ejemplo: $1010000101000101b = A145h$

- De derecha a izquierda de la cadena numérica, se van cogiendo cadenas de 4 dígitos binarios, y se transforman a su correspondiente dígito hexadecimal. En el ejemplo, $0101_2 = (0 \cdot 8) + (1 \cdot 4) + (0 \cdot 2) + (1 \cdot 1) = 5_{(10)} = 5_{(16)}$
- Siguiendo el proceso con el resto de la cadena,

$$0100_2 = (0 \cdot 8) + (1 \cdot 4) + (0 \cdot 2) + (0 \cdot 1) = 4_{(10)} = 4_{(16)}$$

$$0001_2 = (0 \cdot 8) + (0 \cdot 4) + (0 \cdot 2) + (1 \cdot 1) = 1_{(10)} = 1_{(16)}$$

$$1010_2 = (1 \cdot 8) + (0 \cdot 4) + (1 \cdot 2) + (0 \cdot 1) = 10_{(10)} = A_{(16)}$$
- El número resultante en base 16 es A145h, que es mucho más fácil de recordar y almacenar (en cuanto a código fuente se refiere) que el correspondiente en base 2.
- En caso de que el número en binario tenga menos de 4 dígitos, se rellenan las posiciones que faltan hacia la izquierda con ceros. Es decir, si tenemos el número $100101b$, al pasarlo a base hexadecimal, tenemos 5 unidades ($0101b$), y para el dígito de las decenas tenemos que encontrar el correspondiente hexadecimal a la cadena $10b$, que es lo mismo que $0010b$, o sea 2 en hexadecimal. Tenemos entonces que $100101b = 25h$.

HEXADECIMAL → BINARIO

Ejemplo: F28Ah = 1111001010001010b

Basta con escribir con 4 dígitos binarios el número correspondiente a cada dígito hexadecimal. Notar que siempre deben utilizarse 4 dígitos binarios aunque haya que añadir ceros a la izquierda.

$$F_{16} = 15_{10} = 1111_2$$

$$2_{16} = 2_{10} = 0010_2$$

$$8_{16} = 8_{10} = 1000_2$$

$$A_{16} = 10_{10} = 1010_2$$

1.1.8 Representación de números enteros

Los números con signo se pueden representar de varias maneras.

1.1.8.1 Representación con signo-magnitud

Es la misma representación que la utilizada para los números positivos excepto por la inclusión de un bit (bit de signo) que indica si el número es positivo (0) o negativo (1). El rango de representación con n bits es $[-(2^{n-1}-1), 2^{n-1}-1]$

Ejemplo utilizando 3 bits:

3	2	1	0	-0	-1	-2	-3
0 1 1	0 1 0	0 0 1	0 0 0	1 0 0	1 0 1	1 1 0	1 1 1

Inconvenientes:

- Tenemos dos representaciones del 0: 000...000 y 100...000
- Para realizar operaciones debemos hacer consideraciones de signo

1.1.8.2 Representación en complemento a 1

Los números positivos se representan como en el caso de signo-magnitud y los negativos como el complemento a 1 del entero positivo de la misma magnitud.

El complemento a 1 de un número se obtiene sustituyendo los 0's por 1's y viceversa. El rango de representación con n bits es $[-(2^{n-1}-1), 2^{n-1}-1]$

Propiedad: $C1(C1(A)) = A$

Observación: En los números positivos el primer bit es un 0 y en los negativos es un 1.

Ejemplos:

- utilizando 3 bits

3	2	1	0	-0	-1	-2	-3
0 1 1	0 1 0	0 0 1	0 0 0	1 1 1	1 1 0	1 0 1	1 0 0

- utilizando 8 bits: 18 = 00010010 y -18 = 11101101

Inconvenientes:

- Continuamos teniendo dos representaciones del 0: 000...000 y 111...11
- Para realizar operaciones debemos hacer consideraciones de signo

1.1.8.3 Representación en complemento a2

Los números positivos se representan como en el caso de signo-magnitud y los negativos como el complemento a2 del entero positivo de la misma magnitud.

El complemento a2 de un número se obtiene sumando 1 a su complemento a 1
El rango de representación con n bits es $[-2^{n-1}, 2^{n-1}-1]$

Propiedad: $C2(A) = 2^n - N$

Observación: En los números positivos el bit más significativo es un 0 y en los negativos es un 1.

Ejemplos:

- utilizando 3 bits

3	2	1	0	-1	-2	-3	-4
0 1 1	0 1 0	0 0 1	0 0 0	1 1 1	1 1 0	1 0 1	1 0 0

- utilizando 8 bits

18 = 00010010 y -18 = $C1(-18) + 1 = 11101101 + 1 = 11101110$

1.2 Sistemas de codificación

La información representada en un alfabeto de entrada debe ser transformada (codificada) a un alfabeto de salida que entienda la máquina. La codificación debe realizarse de manera que a cada elemento del primer alfabeto le corresponda un elemento distinto del segundo.

INFORMACION ==> SISTEMA DE CODIFICACION ==> INFORMACION CODIFICADA
(Alfabeto de entrada) (Alfabeto de salida)

Todo sistema de codificación lleva consigo un código que se define como la ley de correspondencia biunívoca entre los datos que se van a representar y su codificación. Algunas de las características de un código son el conjunto de caracteres y el número de bits que se utilizan para codificar un carácter (longitud del código). El número máximo del conjunto de caracteres que podemos representar es 2^{longitud} .

1.2.1 Códigos binarios numéricos

1.2.1.1 Códigos BCD (Binary-Coded Decimal)

Son aquellos que codifican directamente cada dígito decimal en un código binario.

- Ponderados: En este caso los bits tienen un peso en función de la posición que ocupan.
 - * BCD natural: Cada dígito se codifica con el código binario de 4 bits. En realidad los pesos son 8,4,2,1
 - * BCD Aiken: Se utiliza un código de pesos 2,4,2,1
- No ponderados: Los bits no tienen peso.
 - * BCD exceso 3: Cada dígito N se codifica como el correspondiente a N+3 en binario natural.

Decimal	Pesos	BDC Natural				BCD Aiken				BCD exceso 3			
		8	4	2	1	2	4	2	1				
0		0	0	0	0	0	0	0	0	0	0	1	1
1		0	0	0	1	0	0	0	1	0	1	0	0
2		0	0	1	0	0	0	1	0	0	1	0	1
3		0	0	1	1	0	0	1	1	0	1	1	0
4		0	1	0	0	0	1	0	0	0	1	1	1
5		0	1	0	1	1	0	1	1	1	0	0	0
6		0	1	1	0	1	1	0	0	1	0	0	1
7		0	1	1	1	1	1	0	1	1	0	1	0
8		1	0	0	0	1	1	1	0	1	0	1	1
9		1	0	0	1	1	1	1	1	1	1	0	0

Vamos a emplear el BCD natural. Consiste en emplear cuatro bits para codificar los dígitos del 0 al 9, desperdiciando las seis combinaciones que van de la 1010 a la 1111 y ocupando, por tanto, más espacio que el código binario natural. La ventaja es la simplicidad de conversión a/de base 10, que resulta inmediata. Los números BCD pueden almacenarse desempaquetados, en cuyo caso cada byte contiene un dígito BCD; o empaquetados, almacenando dos dígitos por byte (para construir los números que van del 00 al 99). La notación BCD ocupa cuatro bits -un nibble- por cifra, de forma que en el formato desempaquetado el nibble superior siempre es 0.

Ejemplo:

Mientras que el número 133 en base decimal se almacena en un sólo byte utilizando el sistema de representación Binario Puro (los 8 bits '10000101'), necesitará 3 bytes para ser representado en BCD. La representación en BCD sería la cadena de bytes para representar los dígitos '1', '3' y '3'. Es decir, 0001, 0011 y 0011. El número 23849 necesitaría solamente 2 bytes (la palabra '101110100101001') para ser representado en Binario Puro, mientras que en BCD necesitaría 5 bytes, uno para cada dígito decimal.

1.2.2 Códigos alfanuméricos

Los códigos alfanuméricos son utilizados por los ordenadores para guardar y transmitir información. Muchas veces la información que queremos codificar no es numérica. Por lo tanto, una computadora necesita almacenar diversos tipos de caracteres.

- Caracteres alfabéticos:
 - * letras mayúsculas: De la A a la Z
 - * letras minúsculas: De la a a la z
- Cifras decimales: Del 0 al 9
- Caracteres especiales:
 - * Caracteres: punto (.), coma (,), punto y coma (;), asterisco (*), etc...
 - * Ordenes de control:
 - ◆ NUL Null (caracter nulo)
 - ◆ CR Carriage return (retorno de carro)
 - ◆ ACK Acknowledge (reconocimiento de transmisión)
 - ◆ FF Form feed (avance de página)
 - ◆ LF Line feed (avance de línea)

El código ASCII es el más utilizado hoy en día. Consta de 127 símbolos y, por lo tanto, se necesitan 7 bits para codificarlos. Debido a que los ordenadores manipulan los bits de 8 en 8 (byte) el código ASCII se ha extendido a 8 bits (ASCII extendido) pudiéndose codificar símbolos adicionales.

El código A.S.C.I.I. (American Standard Code for Information Interchange) es un convenio adoptado para asignar a cada carácter un valor numérico; su origen está en los comienzos de la Informática tomando como muestra algunos códigos de la transmisión de información de radioteletipo.

Los dígitos numéricos (códigos ASCII 30h al 39h) difieren de sus respectivos valores sólo en el nibble (4 bits) de alto orden, restando 30h de un código numérico ASCII dado se obtiene el equivalente numérico.

2. ALMACENAMIENTO DE DATOS

2.1 Unidades de información

2.1.1 BIT

Toda la memoria del ordenador se compone de dispositivos electrónicos que pueden adoptar únicamente dos estados, que representamos matemáticamente por 0 y 1. Cualquiera de estas unidades de información se denomina *BIT*, contracción de «binary digit» en inglés.

2.1.2 BYTE

Cada grupo de 8 bits se conoce como *byte* u octeto. Es la unidad de almacenamiento en memoria, la cual está constituida por un elevado número de posiciones que almacenan bytes. La cantidad de memoria de que dispone un sistema se mide en Kilobytes (1 KB = 1024 bytes), en Megabytes (1 MB = 1024 Kb), Gigabytes (1 GB = 1024 Mb), Terabytes (1 TB = 1024 Gb) o Petabytes (1 PB = 1024 Tb).

Los bits en un byte se numeran de derecha a izquierda y de 0 a 7, correspondiendo con los exponentes de las potencias de 2 que reflejan el valor de cada posición. Un byte nos permite, por tanto, representar 256 estados (de 0 a 255) según la combinación de bits que tomemos.

2.1.3 NIBBLE

Cada grupo de cuatro bits de un byte constituye un *nibble*, de forma que los dos nibbles de un byte se llaman nibble superior (el compuesto por los bits 4 a 7) e inferior (el compuesto por los bits 0 a 3). El nibble tiene gran utilidad debido a que cada uno almacena un dígito hexadecimal.

2.1.4 WORD (PALABRA)

Un word o palabra es un grupo de 8 bits, es decir, dos bytes.

2.1.5 DOBLE PALABRA

La doble palabra está compuesta por 32 bits (2 palabras ó 4 bytes).

2.2 Almacenamiento de datos en las unidades de información

Cuando se trabaja con números naturales (enteros positivos, incluido el cero), utilizaremos el sistema de representación binario puro. Si trabajamos con números enteros en general (positivos y negativos), se utiliza el complemento a 2. Además de estos dos sistemas de representación, veremos el uso del sistema BCD, usado por la rapidez que ofrece para codificar y decodificar rápidamente información decimal (con la que estamos familiarizados) en una base binaria a la que no estamos tan acostumbrados.

2.2.1 Coma fija sin signo (Binario Puro)

Tomemos una cadena de 3 bits o dígitos de longitud ($N=3$). En este caso podremos representar todos los enteros positivos en el rango $[0..(2^3-1)]$. Es decir, todos los números enteros comprendidos entre 0 y 7. Veamos ahora este sistema de representación con los 3 tamaños de datos básicos manejados en programación.

- **Tamaño Byte** (8 bits). La longitud de esta unidad de información es de 8 bits, es decir, $N=8$. Por tanto en este tipo de datos (y con el sistema de representación Binario Puro), vamos a poder representar enteros comprendidos en el Rango $[0..(2^8-1)] = [0..255]$. Esto quiere decir que en un registro o posición de memoria de tamaño byte (8 bits), vamos a poder tener 256 valores diferentes.
- **Tamaño Word ó palabra** (16 bits). La longitud de esta unidad de información es de 16 bits. Por tanto, con el sistema de representación Binario Puro, vamos a poder representar enteros comprendidos en el Rango $[0..(2^{16}-1)] = [0..65535]$. Esto quiere decir que en un registro o posición de memoria de tamaño palabra (16 bits), vamos a poder tener 65536 valores diferentes. O sea, que el mayor número que se podrá representar es el 65535; y el menor número representable será el 0.
- **Tamaño DWord, Double Word, ó Doble palabra** (32 bits). La longitud de esta unidad de información es de 32 bits. Por tanto, con el sistema de representación Binario Puro, vamos a poder representar enteros comprendidos en el Rango $[0..(2^{32}-1)] = [0..4294967295]$.

2.2.2 Complemento a 2

El sistema de representación Complemento a 2, es el usado para poder realizar sumas y restas con números enteros sin tener que hacer comprobaciones del signo de los operandos.

Supongamos que estamos trabajando con un dato tamaño Byte (8 bits), luego N (longitud de la cadena numérica) es igual a 8. Si estuviéramos trabajando sólo con números enteros positivos (binario puro), el registro ó posición de memoria admitiría 256 valores positivos. Pero estamos utilizando el sistema de Complemento a 2, porque vamos a trabajar con enteros en general, positivos y negativos. Luego esos 256 valores han de dividirse en 2 grupos: uno para los positivos, y otro para los negativos.

Para los números positivos se reservan los códigos que tengan el bit 7 (bit más significativo, ya que estamos trabajando con datos de 8 bits) con valor 0. Es decir, los códigos 00000000 hasta 01111111. El resto de códigos posibles se utilizan para representar los números negativos. Es decir, los códigos 10000000 hasta 11111111.

Es fácil determinar si un número es positivo o negativo. Si es positivo, su bit más significativo valdrá 0. Si es negativo, su bit más significativo valdrá 1. Hemos visto los códigos reservados para cada grupo de números, los positivos y los negativos. Veremos a continuación las diferencias en la representación entre los positivos y los negativos.

Los números positivos se representan en binario puro, como hemos visto en el apartado anterior. Es decir, si queremos representar el número 37 en un registro de tipo byte, quedaría de la forma 00100101. Como podemos observar, al utilizar números positivos en Complemento a 2, estamos utilizando la representación en Binario Puro. Hay que tener en cuenta el mayor número positivo representable en este tipo de dato (8 bits) y con este sistema de representación. Es decir, no se pueden representar números positivos más allá del 01111111b, por tanto, el mayor número positivo representable es el 127 para este tamaño de dato. Todo lo expuesto hasta ahora (y a continuación) para el tipo de dato byte, es extensivo para el resto de tipos de datos (palabra y doble palabra), teniendo en cuenta su diferente longitud (N), obviamente.

En general, el rango de representación de los números positivos en el sistema de complemento a 2 es: $[0..(2^{N-1}-1)]$. En el caso de dato de tipo byte: $[0..(2^{8-1}-1)] = [0..127]$. Para el tipo de dato word: $[0..(2^{16-1}-1)] = [0..32767]$.

Con los números negativos es cuando sí se utiliza el Complemento a 2. A estas alturas, podremos apreciar que para representar un número negativo en complemento a 2, no introducimos ese número en el registro ó posición de memoria, sino su complemento con respecto a 2^N . De esta manera, todos los números negativos tendrán su bit más significativo con valor 1, indicando su condición de número negativo. Para obtener el rango de los números negativos, tenemos que calcular el mínimo y el máximo representable. Tomemos el tamaño de dato byte, para concretar: El más pequeño número negativo es -1, y su complemento es 11111111. Como en los números negativos, el bit más significativo (bit 7 en este caso) debe ser 1, el máximo negativo representable será el máximo al que se le pueda hacer el complemento sin que el bit más significativo sea 0, ya que entonces sería positivo. Llegamos entonces a la conclusión de que tal máximo es -128, con el complemento 10000000b. En general, el rango de representación de los números negativos en el sistema de complemento a 2 es: $[-2^{N-1}..-1]$. En el caso de dato de tipo byte $[-2^{8-1}..-1] = [-128..-1]$

El rango completo de representación (negativos y positivos) en el sistema de Complemento a 2 es: $[-2^{N-1}..(2^{N-1}-1)]$. Siendo N, la longitud del dato.

REFERENCIAS

- Electronic logic systems, Almaini, Prentice Hall, 1986
- Electrónica digital moderna, J. M. Angulo, Paraninfo, 1991
- Estructura y tecnología de computadores, Carlos Cerrada Somolinos y Vicente Feliú Batlé, I, Universidad Nacional de Educación a Distancia, 1993
- Introducción al Diseño lógico digital, J. Hayes, Addison-Wesley Iberoamericana, 1996
- Electrónica Digital, Sanz y Torres, Mira y col., 1993
- Circuitos digitales y microprocesadores, H. Taub, McGraw-Hill, 1983
- Principios digitales, R. L. Tokheim, McGraw-Hill, 1995.

EJERCICIOS

1. Expresar los siguientes números en base binaria
66h
FFFFh
FFFEh
4FD3₁₆)
9
236d
0x42AB
546
32
8
4
12
7631₈)
2. Expresar los siguientes números en base hexadecimal
101111b
13
15
16d
0100111010110b
17
18d
11
111111111000001b
537_o
10001110110₂)
3. ¿Cuál es el número de bits mínimo necesario para representar el número 2345 en binario?
4. ¿Número de bits mínimo necesario para representar el número -2345 en binario?
5. ¿Cuál es el mayor número natural que se puede representar con 11 bits? ¿Y el mayor número entero?
6. ¿Cuántos valores diferentes pueden representarse con números binarios de 5 bits?
7. ¿Cuál es el mayor número binario que se puede representar con 2 bytes? ¿Y empleando codificación BCD natural?
8. Expresar los siguientes números en código binario puro, en BCD y en ASCII
12, 99, 34, 3, 123

1. Expresar los siguientes números en base binaria y decimal

66h	= 1100110b	= 102
FFFFh	= 111111111111111b	= 65535
FFFEh	= 111111111111110b	= 65534
4FD3 ₁₆)	= 100111111010011b	= 20435
9	= 1001b	= 9
236d	= 11101100b	= 236
0x42AB	= 100001010101011b	= 17067
546	= 1000100010b	= 546
32	= 100000b	= 32
8	= 1000b	= 8
4	= 100b	= 4
12	= 1100b	= 12
7631 ₈)	= 111110011001b	= 3993

2. Expresar los siguientes números en base hexadecimal y decimal

101111b	= 2Fh	= 47
13	= 0Dh	= 13
15	= 0Fh	= 15
16d	= 10h	= 16
0100111010110b	= 9D6h	= 2518
17	= 11h	= 17
18d	= 12h	= 18
11	= 0Bh	= 11
1111111111000001b	= 0FFC1h	= 65473
537 _o	= 15Fh	= 351
10001110110 ₂)	= 476h	= 1142

3. ¿Cuál es el número de bits mínimo necesario para representar el número 2345 en binario?

Con P bits represento 2^P números, desde el 0 hasta el $2^P - 1$, para representar N, $2^P - 1 \geq N$

$$P \geq \log_2(2345 + 1) = 11.19$$

12 bits

4. ¿Número de bits mínimo necesario para representar el número -2345 en binario?

Con P bits represento 2^P números: el cero, $(2^{P-1} - 1)$ positivos y 2^{P-1} negativos, para poder representar el número N:

$$2^{P-1} \geq N$$

$$P \geq \log_2 N + 1$$

$$P \geq \log_2(2345) + 1 = 12.19$$

13 bits

5. ¿Cuál es el mayor número natural que se puede representar con 11 bits? ¿Y el mayor número entero?

$$2^{11} - 1 = 2047 \text{ (es decir, 2048 números contando el 0)}$$

$$2^{11-1} - 1 = 1023 \text{ (es decir, 1023 positivos, el cero y 1024 negativos)}$$

6. ¿Cuántos valores diferentes pueden representarse con números binarios de 5 bits?

$$2^5 = 32$$

7. ¿Cuál es el mayor número binario que se puede representar con 2 bytes? ¿Y empleando codificación BCD natural?

$$2^{16} - 1 = 65535$$

$$99$$

8. Expresar los siguientes números en código binario puro, en BCD y en ASCII

12	= 1100b	= 0001 0010b	= 31h, 32h → 00110001b, 00110010b
99	= 1100011 b	= 1001 1001b	= 39h, 39h → 00111001b, 00111001b
34	= 100010 b	= 0011 0100b	= 33h, 34h → 00110011b, 00110100b
3	= 11b	= 0000 0011b	= 30h, 33h → 00110000b, 00110011b
123	= 1111011	= 0001 0010 0011b	= 31h, 32h, 33h → 00110001b, 00110010b 00110011b