ARQUITECTURA DE LAS COMPUTADORAS
PRACTICA

SISTEMAS NUMÉRICOS

INTRODUCCIÓN TEÓRICA:

Definimos **Sistema de Numeración** como al conjunto de reglas que permiten, con una cantidad finita de símbolos, representar un número natural cualquiera.

Los números pueden representarse en diversos sistemas de numeración, que se diferencian por su base. La BASE de un sistema numérico es el número de símbolos distintos que tiene el sistema para la representación de las cantidades en el mismo. El mayor de los símbolos es menor en una unidad a la base del sistema.. (Ejs.: base del sistema decimal: 10, símbolo mayor: 9; base del binario: 2, símbolo mayor: 1; base del octal: 8, símbolo mayor: 7; etc.). El menor de los símbolos es siempre el "0". Los símbolos utilizados se denominan CARACTERES.

Los caracteres presentan dos tipos de valores:

- VALOR INTRINSECO o ABSOLUTO: Es el referido a la unidad.
- VALOR RELATIVO: Es de acuerdo a la posición que ocupe en la configuración de un número.

Es posible representar un número según una expresión genérica como la siguiente:

$$N_r = a_n \cdot r^n + \ldots + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + \ldots + a_{-m} \cdot r^{-m}$$

donde: r = es la base del sistema

a_i= son los caracteres, que pueden tomar valores entre 0 y (r - 1)

Esta expresión es una **suma de Valores Relativos**.

Vemos que:

VALOR RELATIVO = VALOR ABSOLUTO x PESO

donde: PESO es la base del sistema elevada a un exponente que indica la posición del caracter dentro del número. Para la parte entera, el exponente es positivo y su valor absoluto es uno menos que la verdadera posición del caracter dentro del número. Para la parte fraccionaria, el exponente es negativo, y su valor absoluto indica la verdadera posición del caracter.

EJEMPLOS:

1) SISTEMA DECIMAL:

a) SIMBOLOS: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

b) BASE: 10

c) Cada caracter se denomina DIGITO.

d) Dado el número: $N_{10} = 1534,27_{10}$, podemos desarrollarlo, según la expresión vista:

$N_{10} = 1 . 10^{3} + 5 . 10^{2} + 3 . 10^{1} + 4 . 10^{0} + 2 . 10^{-1} + 7 . 10^{-2}$

2) SISTEMA BINARIO:

a) SIMBOLOS: 0, 1

b) BASE: 2

c) Cada caracter se denomina BIT.

d) Dado el número: $N_{2} = 1001,11_{2}$, se desarrollará:

$N_{2} = 1 . 2^{3} + 0 . 2^{2} + 0 . 2^{1} + 1 . 2^{0} + 1 . 2^{-1} + 1 . 2^{-2}$

3) SISTEMA OCTAL:

a) SIMBOLOS: 0, 1, 2, 3, 4, 5, 6, 7.

b) BASE: 8

c) Cada símbolo se denomina caracter octal.

d) Dado: $N_{8} = 432,76_{8}$, será:

$N_{8} = 4 . 8^{2} + 3 . 8^{1} + 2 . 8^{0} + 7 . 8^{-1} + 6 . 8^{-2}$

4) SISTEMA HEXADECIMAL:

a) SIMBOLOS: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

b) BASE: 16

c) Cada símbolo se llama caracter hexadecimal.

d) Si tenemos: $N_{16} = 2B41,C5_{16}$, su desarrollo será:

$N_{16} = 2 . 16^{3} + B . 16^{2} + 4 . 16^{1} + 1 . 16^{0} + C . 16^{-1} + 5 . 16^{-2}$

TABLA :

SIST. DECIMAL	SIST. BINARIO	SIST. OCTAL	SIST. 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

∴ LA BASE DE UN SISTEMA, EN SU PROPIO SISTEMA, SE ESCRIBE “10”.

CONVERSION DE SISTEMAS:

I) Para pasar un número entero del sistema decimal a otro sistema numérico, debe dividirse el número por la base nueva sucesivamente, hasta que el cociente sea menor que la base. Luego se ordenan los restos de izquierda a derecha, comenzando por el últimoresultado, para poder obtener la nueva expresión del número.

Ejemplo: $87_{10} = N_8$

$87_{10} = 127_8$

87 / 8

7/ 10 / 8

2/ 1

II) Para pasar un número fraccionario decimal a otro sistema numérico, debe multiplicarse por la base del nuevo sistema, tantas veces, como caracteres querramos tener después de la coma.

Ejemplo: $0{,}67_{10} = N_{16}$

0,67

x 16

10, 72 0,72

x 16

11, 52 0,52

x 16

8, 32

➔ $N_{16} = 0, AB8\ldots_{16}$

III) En el caso que el número tenga parte entera y fraccionaria, debe trabajarse cada parte por separado, y luego juntar los resultados.

Ejemplo: $97, 012_{10} = N_8$

97 / 8

1 / 12 / 8

4/ 1

0,012

x 8

0, 096

x 8

0, 768

x 8

6, 144

➔ $N_8 = 141, 006\ldots_8$

IV) Para pasar un número de un sistema numérico cualquiera a decimal, debe hacerse la suma de los valores relativos de los caracteres del mismo.

Ejemplos:

1) $101_{2} = 1 . 2^{2} + 0 . 2^{1} + 1 . 2^{0} = 4 + 0 + 1 = 5_{10}$

1) $11001,1_{2} = 1. 2^{4} + 1 . 2^{3} + 0 . 2^{2} + 0 . 2^{1} + 1 . 2^{0} + 1 . 2^{-1} = 16 + 8 + 1 + 0,5 = 25,5_{10}$

2) $327 , 1_{8} = 3. 8^{2} + 2 . 8^{1} + 7 . 8^{0} + 1 . 8^{-1} = 215,125_{10}$

3) $BA , 81_{16} = B . 16^{1} + A . 16^{0} + 8 . 16^{-1} + 1 . 16^{-2} = 11 . 16 + 10 . 1 + 8 . 16^{-1} + 1 . 16^{-2} =$

$= 186 , 503\ldots_{10}$

V) Los sistemas numéricos cuya base es de la forma 2^{n} , ofrecen una simple conversión desde el binario. Un ejemplo sería el sistema numérico octal ($2^{3}=8$), y otro, el hexadecial ($2^{4} = 16$).

Veamos primero, una forma de pasar números enteros del sistema numérico binario al decimal, según el método enseñado, pero ordenándolos en una tabla:

----	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
----	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
							1	1	0	$= 6_{10}$
					1	0	0	0	0	$= 16_{10}$
				1	1	0	0	1	0	$= 50_{10}$

Ahora, utilicemos esta misma tabla, para pasar números enteros del sistema numérico decimal, al binario, llenando los casilleros desde el que tiene mayor peso al menor, según los necesitemos, comenzando con el que tiene el peso igual o menor al número que estamos convirtiendo (o sea, pasando de un sistema numérico a otro):

----	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
----	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
							1	0	1	$= 5_{10}$
				1	0	0	1	1	0	$= 38_{10}$
		1	0	0	0	0	1	0	1	$= 133_{10}$
	1	0	0	0	1	0	0	1	1	$= 275_{10}$

Ahora aprovecharemos esta forma de conversión, de la siguiente manera:

a) Para pasar del binario al sistema cuya base es potencia de "2", o sea, " $\mathbf{2^n}$ " deben agruparse tantos bits como sea el exponente "n". La agrupación se hace desde el bit de menor peso hacia el de mayor peso, para la parte entera, y desde el de mayor peso hacia el de menor peso, en la parte fraccionaria. Luego se escribe el caracter equivalente (según el nuevo sistema) de cada grupito.

Ejemplos:

I) Para pasar al sistema octal, como $\mathbf{8 = 2^3}$, se agruparán de a tres bits. Como, al trabajar en binario con tres bits, podemos escribir del "0" al "7" decimal, que equivalen del "0" al "7" octal, usaremos las tres columnas de menor peso (las tres de la derecha), de la tabla recién vista.

1) $011010_2 = 011\ \ 010_{\ 2} = 3\ 2_{\ 8}$

Verificación: $011010_2 = 1 . 2^4 + 1 . 2^3 + 0 + 1 . 2^1 + 0 = 16 + 8 + 0 + 2 + 0 = 26_{\ 10}$

26 /_8_

2 / 3 ➔ $32_{\ 8}$

2) $1101_{\ 2} = 001101_{\ 2} = 001\ \ 101_{\ 2} = 15_{\ 8}$

3) $10,11_{\ 2} = 010,110_{\ 2} = 2,6_{\ 8}$

En este ejercicio, vemos que el número tiene parte entera y parte fraccionaria, y estamos usando la tabla que era para convertir números enteros! Veamos si el resultado es válido:

Verificación: $10,11_{\ 2} = 1 .2^1 + 0 + 1 . 2^{-1} + 1 . 2^{-2} = 2 + 0,5 + 0,25 = 2,75_{\ 10}$

2 / 8 0,75

2/ 0 x 8

6,00 ➔ Rta.: $02,6_{\ 8}$

II) Lo mismo sucede cuando queremos pasar del sistema numérico binario al hexadecimal. Como la base **16** es $\mathbf{2^4}$, agruparemos de a cuatro bits. Utilizaremos las cuatro columnas de menor peso de la tabla, ya que al trabajar en binario con cuatro bits, podremos escribir del "0" al "15" decimal que equivalen, del "0" al "15" hexadecimal, teniendo en cuenta que a partir del "10" debemos trabajar con las letras.

1) $111101,101_{\ 2} = \mathbf{00}11\ 1101\ ,\ 1010_{\ 2} = 3D,A_{\ 16}$

Verificación:

$111101{,}101_{\,2} = 1.\,2^5 + 1.\,2^4 + 1\,.\,2^3 + 1.\,2^2 + 0.\,2^1 + 1.\,2^0 + 1.\,2^{-1} + 0.\,2^{-2} + 1.\,2^{-3} =$

$= 32 + 16 + 8 + 4 + 0 + 1 + 0{,}5 + 0 + 0{,}125 = 61{,}625_{\,10}$

61 / 16	0,625
13/ **3**	x 16
	3750
	625
	10,000 ➔ Rta.: $3D{,}A_{\,16}$

2) $1011111{,}111_{\,2} = \mathbf{0}101\ 1111\ ,\ 111\mathbf{0}_{\,2} = 5F{,}E_{\,16}$

b) Para pasar del sistema numérico cuya base es potencia de "2" al binario, debe escribirse cada caracter de ese sistema con tantos bits como sea el exponente "n".

Ejemplos:

1) $64{,}2_{\,8} = 110\ 100\ ,\ 010_{\,2}$

Verificación: $64{,}2_{\,8} = 6\,.\,8^1 + 4\,.\,8^0 + 2\,.\,8^{-1} = 48 + 4 + 0{,}25 = 52{,}25_{\,10}$

52 / 2	0,25	0,5
0 / 26 / 2	x 2	x 2
0/ 13 / 2	**0**,5	**1**,00
1/ 6 / 2		
0/ 3 / 2		
1/ 1	∴ $52{,}25_{\,10} = 110100{,}01_{\,2}$	

2) $750{,}1_{\,8} = 111\ 101\ 000\ ,\ 001_{\,2}$

3) $2C4{,}B1_{\,16} = 0010\ 1100\ 0100\ ,\ 1011\ 0001_{\,2}$

4) $970{,}F_{\,16} = 1001\ 0111\ 0000\ ,\ 1111_{\,2}$

Ejercicios: Realizar las siguientes conversiones:

1) $249_{16} = N_2$, N_8

Rta.: Como: $2_{16} = 0010_2$; $4_{16} = 0100_2$; $9_{16} = 1001_2$

$\Rightarrow$ $N_2 = 001001001001_2$ y $001/001/001/001_2 = 1111_8 = N_8$

2) $467_{10} = N_{16}$ Rta.: $1D3_{16}$

3) $57,3_{10} = N_2$ Rta.: $111001,01001_2$

4) $895,38_{10} = N_8$ Rta.: $1577,302_8$

5) $01001110,11_2 = N_{10}$ Rta.: $78,75_{10}$

6) $1234_8 = N_{10}$ Rta.: 668_{10}

7) $EFD,34_{16} = N_{10}$ Rta.: $3837,2031_{16}$

8) $F78_{16} = N_8$ Rta.: 7570_8

9) $111011111,10001_2 = N_{16}$ Rta.: $1DF,88_{16}$

10) $5260,0216_8 = N_{16}$ Rta.: $AB0,08E_{16}$

11) $11,0B9_{16} = N_2$ Rta.: $10001,000010111001_2$

12) $1010011,100010_2 = N_8$ Rta.: $123,42_8$

13) $42,53_8 = N_2$ Rta.: $100010,101011_2$

SUMAS EN DIFERENTES BASES:

Se siguen las mismas reglas que en el sistema decimal. Se comienza la suma con la columna de menor peso y se sigue con las siguientes columnas, hasta terminar con la de mayor peso. En el caso de existir acarreos, se pasan ordenadamente a las columnas de mayor peso contiguas, y se sigue sumando.

Ejemplos:

1)	110111_{2}	2)	6799_{16}	3)	11002_{8}
	$+ 100101_{2}$		$+ B761_{16}$		$+ 7150_{8}$
	101111_{2}		$A41_{16}$		4750_{8}
	010011_{2}		$1293B_{16}$		25122_{8}
	10011110_{2}				

Ejercicios:

1) $1011111_{2} + 111_{2} + 10011_{2} = N_{2}$ Rta.: 1111001_{2}

2) $107D_{16} + F9AB_{16} = N_{16}$ Rta.: $10A28_{16}$

3) $352_{8} + 173_{8} = N_{8}$ Rta.: 545_{8}

4) $1000000_{2} + 24_{8} + 71_{16} = N_{16}$ Rta.: $C5_{16}$

5) $AE6_{16} + CF2_{16} = N_{8}$ Rta.: 13730_{8}

6) $7765_{8} + 110101100_{2} = N_{16}$ Rta.: $11A1_{16}$

7) $C18E_{16} + 43D6_{16} + 200F_{16} = N_{16}$ Rta.: 12573_{16}

REPRESENTACION DE NUMEROS BINARIOS:

Un número binario puede representarse con signo o sin él.

A) NÚMEROS SIN SIGNO:

Una configuración binaria de "n" bits puede representarse como suma de valores relativos, según lo vimos al comienzo del capítulo.

Trabajando con esa expresión, llegamos a que el alcance de la representación sin signo es:

$$\mathbf{0 \leq N \leq 2^n - 1}$$

B) NÚMEROS CON SIGNO:

Por convención, se reserva el bit más pesado para el signo:

"0", si el número es positivo y "1", si es negativo. Existen tres modos de representación:

I) SIGNO Y MÓDULO.

II) COMPLEMENTO A LA BASE MENOS UNO (o Complemento a 1: C_1).

III) COMPLEMENTO A LA BASE (o Complemento a 2: C_2)

I) SIGNO Y MÓDULO: El bit más significativo representa el signo, y los (n-1) restantes, el valor absoluto del mismo.

Su rango de representación es:

$$\mathbf{-(2^{n-1} - 1) \leq N_{SM} \leq (2^{n-1} - 1)}$$

O sea, que tenemos 2^n configuraciones binarias, y dos de las cuales representan al cero.

Ejemplos: $+ 13_{10} = 00001101_2$ $+ 0_{10} = 00000000_2$

$- 13_{10} = 10001101_2$ $- 0_{10} = 10000000_2$

En estas representaciones, al realizar una suma o una resta, debemos proceder de la siguiente manera: comparar valores absolutos y con ellos determinar el signo luego de la operación, o sumar y luego ver si el resultado es negativo o positivo.

II) COMPLEMENTO A LA BASE MENOS UNO:

Es un **convenio** para **representar** la **inversa de un número** (positivo o negativo).

Si la operación la está efectuando un sistema digital, debemos considerar su capacidad, es decir, el número de bits con que trabaja el mismo.

Matemáticamente se expresa con la siguiente fórmula:

$$-N = b^n - 1 - |N|$$

En esta forma de representación, los números positivos son iguales que en el caso anterior. Los negativos son codificados como complementos a la base menos uno.

El alcance de esta representación es igual que en Signo y Módulo:

$$-(2^{n-1}) + 1 \leq N_{C1} \leq (2^{n-1}) - 1$$

Para escribir un número negativo debe escribirse el positivo, y ver **cuánto le falta para la base menos uno** (en binario, cuánto le falta para "1") a cada uno de los bits que forman el número positivo. Tenemos, nuevamente, una doble representación del cero.

Ejemplos: $+ 13_{10} = 00001101_2$ $+ 0_{10} = 00000000_2$

$- 13_{10} = 11110010_2$ $- 0_{10} = 11111111_2$

Como se ve, para obtener el complemento a 1, basta con invertir todos los bits del número dado.

En cualquier sistema numérico, si se suma el número N al negativo (-N) se obtiene "cero".

Ejemplo:

	13_{10}	00001101_2
	$+ \underline{(-13)_{10}}$	$+ \underline{11110010_2}$
	00	11111111_2 (que es el cero "negativo")

III) COMPLEMENTO A LA BASE:

Al igual que el caso anterior, el **convenio de complemento a a la base** $^{\text{número de dígitos}}$ **(C a b ^{n})** es un convenio para representar la **inversa de un número** (positivo o negativo).

Si se hablase de "complemento a la base" a "secas", debería aclararse con cuántos caracteres se está trabajando.

Si la operación la está efectuando un sistema digital, debemos considerar su capacidad, es decir, el número de bits con que trabaja el mismo.

Matemáticamente se expresa con la siguiente fórmula:

$$- N = b^n - | N |$$

Los números positivos son iguales que en los dos casos anteriores. Los negativos son codificados como complemento a la base.

El alcance es el siguiente:

$$- (2^{n-1}) \leq N_{C2} \leq (2^{n-1}) - 1$$

Para obtener un número negativo, debe escribirse el número positivo y ver **cuánto le falta para llegar a la base**. En esta representación no existe el cero "negativo", debido a esto, hay una configuración binaria adicional disponible que aumenta el alcance negativo en una unidad, respecto al caso de Complemento a la base menos 1.

Ejemplos: $+ 13_{10} = 00001101_2$ $0_{10} = 00000000_2$

$- 13_{10} = 11110011_2$

Existen dos formas sencillas para obtener el complemento a la base:

Una consiste en obtener el complemento a la base menos uno (invirtiendo todos los bits en el caso del binario) y sumarle un uno, en la posición menos significativa.

La segunda, es tomar el número positivo y se va buscando desde el bit menos significativo al de mayor peso, un "1"; se copian todos los bits tal cual son, incluyendo a este primer 1, y luego se invierten los bits restantes.

Como es lógico, si sumo al número N, su complemento a la base, el resultado debe ser "0".

Ejemplo:

13_{10}	00001101_2	
$+ \underline{(-13)_{10}}$	$+ \underline{11110011_2}$	
00_{10}	$1\ 00000000_2$	(el noveno bit se pierde, y queda el 0)

OPERACIONES EN SISTEMAS NUMÉRICOS:

La ecuación de la resta es:	Minuendo - Sustraendo = Resultado
O, lo que es lo mismo:	Minuendo + (-Sustraendo) = Resultado
Podríamos escribirla así:	Minuendo + Complemento del sustraendo = Resultado

Esta última ecuación será la que usaremos para realizar las restas, o sea, las transformaremos en sumas.

∗ OPERACION MANUAL:

El procedimiento de trabajo es el siguiente: dados dos números, representados en alguno de los convenios, debemos:

1) Igualar la cantidad de caracteres de ambos números, agregándole ceros al que tiene menor cantidad de caracteres.

2) Representar el minuendo (M) y la inversa del sustraendo (-S) en el complemento elegido.

3) Sumar al minuendo la inversa del sustraendo, es decir: **M + (-S)**.

4) Contar los caracteres del resultado. Si el mismo tiene uno más que los operandos, indica que el resultado es **positivo** y ese caracter se desprecia en Complemento a la base y se suma al resultado parcial en Complemento a la base menos uno, para obtener el resultado final. Si el resultado tiene igual cantidad de caracteres que los operandos, indica que el resultado es **negativo** y el resultado parcial debe complementarse para obtener el resultado final, utilizando el mismo convenio con que se realizó la operación.

5) Representar el resultado en el convenio de símbolo y módulo. En este convenio, se representa al número en valor absoluto y se le antepone el signo "+" o "-", según corresponda.

Ejemplos:

Realizar las siguientes restas como sumas:

1) Dados dos números decimales, representados en módulo, resolver su resta mediante una suma y dar el resultado en símbolo y módulo:

I) $57 - 759 = N_{10}$

Rta.: ♦ Igualamos la cantidad de dígitos de cada número: $\mathbf{0}57 - 759 = N_{10}$

♦ Ahora realizo la operación: $057 + (-759) = N_{10}$

a) representando ambos números en C a 10^3 y realizando la suma:

057 759 057 298

+ (-759) C a 10^3 + **241** + **241** C a 10^3 **702**

1000 298 1000

=> N_{10} = - 702 (en símbolo y módulo)

b) representando ambos números en C a $(10^3 – 1)$ y realizando la suma:

057 759 057 297

+ (-759) C a $(10^3 –1)$ + **240** + **240** C a $(10^3 –1)$ + **702**

999 297 999

⇨ N_{10} = - 702 (en símbolo y módulo)

II) $21_{10} - 16_{10} = N_{10}$

a) 21 16 21

+ (-16) C a 10^2 + **84** + **84**

100 1 05 ➔ N_{10} = + 05 (en símbolo y módulo)

b) 21 16 21

+ 16 C a $(10^2 - 1)$ +**83** + **83**

99 1 04

+ 1

05 ➔ N_{10} = + 05 (en símbolo y módulo)

2) Dados dos números binarios, representados en módulo, realizar su resta como una suma y representar el resultado en símbolo y módulo:

$110 - 10111 = x_{(2)}$

Rta.: Según lo visto en el ejercicio anterior, nos queda:

00110 + (-10111) = $x_{(2)}$

a) 00110 00110

- 10111 en C a 2^5 + **01001**

01111 ➔ = - 10001 (en símbolo y módulo)

b) 00110 00110

- 10111 en C a $(2^5 - 1)$ + **01000**

01110 ➔ = - 10001 (en símbolo y módulo)

Ejercicios: Dados los siguientes números en módulo y símbolo, resolver la resta como suma y expresar el resultado en símbolo y módulo:

a) $110110_2 - 11101_2 = N_2$ Rta.: $011001_2 = +11001_2$

b) $873_{16} - 1532_8 = N_{16}$ Rta.: $+519_{16}$

c) $31_8 - 50_8 = N_8$ Rta.: -17_8

d) $00111_2 - 01101_2 = N_2$ Rta.: -00110_2

e) $5601_8 - 111011101_2 = N_{16}$ Rta.: $+9A4_{16}$

f) $47_{16} - 6E_{16} = N_{16}$ Rta.: -27_{16}

g) $177_8 - 01111011_2 - N_{16} = 2A_{16}$ Rta.: -026_{16}

h) $-3F_{16} + 157_8 = 00110001_2 + N_8$ Rta.: -001_8

i) $010011_2 - 10101_2 = N_{8,2}$ Rta.: $-02_8 = -000010_2$

j) $1011_{2} - 00101_{2} = N_{16}$ Rta.: 06_{16}

k) $65_{8} - 01000_{2} = N_{2}$ Rta.: 101101_{2}

l) $A7_{16} - 54_{8} = N_{8,16}$ Rta.: $173_{8} = 7B_{16}$

ll) $AF_{16} - 13B_{16} = N_{8}$ Rta.: -0214_{8}

m) $375_{8} - 457_{8} = N_{8}$ Rta.: -062_{8}

n) $AF_{16} - 3B_{16} = N_{16}$ Rta.: 74_{16}

ñ) $011011_{2} - 01110011_{2} = N_{2}$ Rta.: -01011000_{2}

o) $317_{8} - 52_{10} = N_{16}$ (No hacerlo en decimal) Rta.: $+09B_{16}$

∗ OPERACIÓN AUTOMÁTICA:

La capacidad de la ALU es fija, por lo tanto, los números se representan todos con esa cantidad de caracteres. Cuando los números no son representables con esa cantidad, se produce el **overflow**.

Existe **overflow** o desborde o rebasamiento, cuando al trabajar con un sistema de **n** bits, se suman dos números de igual signo, representables en **n** bits, y el resultado es de signo contrario. Se haría entonces una interpretación errónea del resultado. Esto ocurre cuando el número de bits del resultado excede la capacidad de almacenamiento del registro del sistema, destinado a recibirlo.

El overflow se manifiesta, entonces, cuando al sumar dos números positivos, el resultado da negativo, o, al sumar dos negativos, el resultado da positivo. Estos "errores" se corregirían, si se pudiesen realizar las mismas operaciones con un bit más (esto puede realizarse en "operación manual").

Ejemplos de overflow: (suponemos trabajar con 8 bits):

1) 01101011_{2}
$+ \underline{01001001_{2}}$
10110100_{2}

2) 10000001_{2}
$+\underline{11110000_{2}}$
$1/01110001_{2}$

3) 0110
$+ \underline{0100}$
1010

4) 1100
$+ \underline{1011}$
1/0111

Ejemplos de restas realizadas como sumas, en forma automática:

Realizar las siguientes operaciones "automáticas" encadenadas, mostrando los resultados parciales y el resultado final que queda almacenado. Suponer que estamos trabajando con un sistema digital con una capacidad de n = 8 bits y que trabaja con el convenio de:

♦ Complemento a dos (C a 2):

a) 21 - 9 +2 – 12 = 2

Rta.: 21 = 00010101 9 = 00001001 - 9 = 11110111
2 = 00000010 12 = 00001100 - 12 = 11110100

I) 00010101 II) 00001100 III) 00001110
+11110111 + 00000010 +11110100
1/00001100 00001110 1/00000010

b) 17 - 44 + 11 = - 16

Rta.: 17 = 00010001 44 = 00101100 - 44 = 11010100
11 = 00001011

I) 00010001 II) 11100101
+ 11010100 + 00001011
11100101 11110000 (= - 00010000 $_2$ = -16 $_{10}$)

c) - 96 + 38 - 3 +21 = - 40

Rta.: 96 = 01100000 - 96 = 10100000 38 = 00100110
3 = 00000011 - 3 = 11111101 21 = 00010101

I) 10100000 II) 11000110 III) 11000011
+ 00100110 + 11111101 +00010101
11000110 1/11000011 11011000 (= - 00101000$_2$ = - 40$_{10}$)

♦ Complemento a uno (C a 1):

a) 34 + 23- 56 – 8 = - 7

Rta.: 34 = 00100010 23 = 00010111 56 = 00111000
- 56 = 11000111 8 = 00001000 - 8 = 11110111

I) 00100010
+ 00010111
00111001

II) 00111001
+ 11000111
1/00000000
+ 1
00000001

III) 00000001
+ 11110111
11111000
(= - 000111_2 = - 7_{10})

b) 64 - 71 + 25 = 18

Rta.: 64 = 01000000 71 = 01000111 - 71 = 10111000
25 = 00011001

I) 01000000
+ 10111000
11111000

II) 11111000
+ 00011001
1/00010001
+ 1
00010010 (= 18_{10})

c) - 87 + 29 – 2 = - 60

Rta.: 87 = 01010111 - 87 = 10101000 29 = 00011101
2 = 00000010 - 2 = 11111101

I) 10101000
+ 00011101
11000101
(= - 00111010_2 = - 58_{10})

II) 11000101
+ 11111101
1/11000010
+ 1
11000011 (= - 00111100_2 = - 60_{10})

ARITMÉTICA DE PUNTO FIJO Y PUNTO FLOTANTE:

La representación de un número en **punto fijo** consiste en escribir el mismo colocando la coma en el lugar que le corresponde, separando la parte entera de la fraccionaria. Esto hace que esté muy limitada la capacidad de almacenamiento, ya que se deben prever cifras enteras y varias fraccionarias. Normalmente, los datos de punto (coma) fijo son números enteros con signo.

La mayoría de las calculadoras y computadoras escriben los números en **punto flotante** o **notación científica**. Esta es una representación que permite localizar automáticamente la coma en el curso del cálculo.

Un número N, en punto o coma flotante, se representa de la siguiente manera:

$$\mathbf{N = M.\ B^{E}}$$

Donde: **M** es la mantisa, **B** es la base de la representación y **E** es el exponente.

La Mantisa representa todos los bits del número sin coma o punto decimal.

Como las computadoras disponen de un número fijo "n" para escribir el número binario, el mismo se almacena de la siguiente manera:

n (M)	n (E)	= n

Donde: "$n_{(M)}$" son los bits que conforman la mantisa y "$n_{(E)}$" son los del exponente.

Ejemplos:

1) Encontrar la representación en punto flotante:

- en el sistema binario:

a) $27{,}5_{10} = 011011{,}1_{2} = 0{,}110111 \times 2^{(5_{10})} = .110111 \times 2^{(5_{10})}$
$= .110111 \times 2^{101} = 0{,}110111 \times 2^{101}$

b) $.00001101_{2} = .1101 \times 2^{-(4_{10})} = .1101 \times 2^{-100} = 0{,}1101 \times 2^{-100}$

- sin cambiar de base:

a) $342{,}16_{10} = 0{,}34216 \times 10^{3}$

b) $0{,}056_{7} = 0{,}56 \times 7^{-1}$

2) Encontrar la representación en punto fijo:

- en el sistema binario:

a) $.110101 \times 2^{1011} = .110101 \times 2^{(11_{10})} = 11010100000_2$

b) $-0,5 \times 10^{0} = -0,5_{10} = -0,1_2$

- sin cambiar de base:

a) $-0,458 \times 9^{-3} = -0,000458_9$

b) $0,22 \times 7^{4} = 2200_7$

Ejercicios:

1) Encontrar la representación en punto flotante, sin cambiar de base:

a) $10110,1101_2$ R: $.10110110 \times 2^{101} = 0,10110110 \times 2^{101}$

b) $.00000010110_2$ R: $.10110 \times 2^{(-110)} = 0,10110 \times 2^{(-110)}$

c) 1000000000000_2 R: $.1 \times 2^{1101} = 0,1 \times 2^{1101}$

d) $36,7_8$ R: $0,367 \times 8^{2}$

e) $0,000A8C_{16}$ R: $0,A8C \times 16^{-3}$

2) Encontrar la representación en punto fijo de los siguientes números en punto flotante, sin cambiar de base:

a) $.101101 \times 2^{(-1011)}$ R: $.00000000000101101_2$

b) $.101101 \times 2^{110}$ R: 101101_2

c) $.11111 \times 2^{1010}$ R: 1111100000_2

d) $0,321 \times 4^{-2}$ R: $0,00321_4$

e) $0,43 \times 5^{3}$ R: 430_5

PRECISIÓN SIMPLE Y DOBLE:

En el caso de trabajar con **punto flotante**, en **precisión simple**, se utiliza una palabra lógica para almacenar la mantisa y el exponente ($n_{(M)}$ y $n_{(E)}$).

En **precisión doble**, se almacena la mantisa en una palabra lógica y el exponente en otra palabra (pueden existir modificaciones según el micro).

En el supuesto caso de tener que trabajar en **punto fijo**, en **precisión simple**, se almacena el número en una sola palabra lógica.

En **precisión doble**, se almacenaría en dos palabras lógicas, pudiendo así, almacenar números más grandes que en el caso anterior.

SISTEMAS DE CODIFICACIÓN:

"Codificar" significa poner en correspondencia biunívoca dos conjuntos de símbolos.

Podemos considerar al sistema binario como un código, ya que con dos símbolos: 0 y 1, representamos al sistema decimal.

Existen códigos numéricos y códigos alfanuméricos.

I) CÓDIGOS NUMÉRICOS:

a) CÓDIGO **BCD**:

BCD significa: Decimal Codificado en Binario.

Es el código más simple: describe en binario, los diez caracteres decimales.

Se codifica cada dígito con cuatro bits.

La correspondencia es la siguiente:

0 - 0000	5 - 0101
1 - 0001	6 - 0110
2 - 0010	7 - 0111
3 - 0011	8 - 1000
4 - 0100	9 - 1001

Ejemplo: $12_{10} = 00010010_{BCD}$

b) CÓDIGO DE **GRAY**:

Este código se denomina también "Código Binario Reflejo" o "Código Espejo".

La correspondencia con el sistema decimal es la siguiente (supongamos trabajar con 8 bits):

0 - 00000000	5 - 00000111	10 - 00001111
1 - 00000001	6 - 00000101	11 - 00001110
2 - 00000011	7 - 00000100	12 – 00001010
3 - 00000010	8 - 00001100	etc…
4 - 00000110	9 - 00001101	

Como se puede observar, entre un número en Gray y el siguiente en la tabla, existe **adyacencia** (se produce un solo cambio de bit al pasar de un número al siguiente consecutivo, o sea, dos valores sucesivos difieren solamente en uno de sus bits).

♦ CONVERSIÓN DE BINARIO A GRAY:

Explicaremos dos métodos:

1er. método)

Se suma al número en binario, el mismo número corrido un lugar a la derecha, perdiendo el menos significativo. La suma se hace sin acarreo.

Ejemplo: $12_{10} = 1100_2$

$$\begin{array}{r} 1\ 1\ 0\ 0 \\ +\ \underline{1\ 1\ 0\ /\ 0} \\ 1\ 0\ 1\ 0_{GRAY} = 12_{10} \end{array}$$

2do. método)

Comenzando con el MSB del número binario, se compara cada par de bits sucesivos:

- Si son iguales, se coloca un "0" en el número en Gray.
- Si son distintos, se coloca un "1" en la palabra en código Gray.

Para comenzar, se compara el primer bit con "0".

Nota: Esta acción es igual a sumar en "módulo 2" a cada par de bits. Esto significa, sumar como una OR-EXCLUSIVA.

Ejemplo: $12_{10} = 1\ 1\ 0\ 0_2 = 1\ 0\ 1\ 0_{GRAY}$

♦ CONVERSION DE GRAY A BINARIO: Son los métodos inversos de los recién explicados:

1er método)

Se toma el resultado de la "cuenta", que es el número en Gray, y se va averiguando desde el bit más significativo al de menor peso, cuáles fueron los bits de los sumandos, teniendo en cuenta la ausencia de acarreo y el desplazamiento de los bits de los mismos.

Por ejemplo:

$$\begin{array}{r} A\ B\ C\ D \\ +\ \underline{A\ B\ C\ /\ D} \\ 1\ 0\ 1\ 0_{GRAY} \end{array}$$

Luego: $A + 0 = 1 \rightarrow A = 1$

$B + A = 0 \rightarrow B + 1 = 0 \rightarrow B = 1$

C + B = 1 ➔ C + 1 = 1 ➔ C = 0

D + C = 0 ➔ D + 0 = 0 ➔ D = 0

O sea: $A\ B\ C\ D = 1\ 1\ 0\ 0_{\ 2} = 12_{\ 10}$

2do. método)

Reglas:

$2_{\ 1}$) El MSB de Gray es igual al MSB del binario, y todos los bits hacia la derecha del binario siguen siendo iguales al número en Gray, hasta el primer 1, inclusive.

$2_{\ 2}$) Ahora se usa el bit del código Gray, como control:

- Si es "1", cambiar el carácter binario que precede, para obtener el bit actual.
- Si es "0", se repite el carácter binario.

Repetir este último punto, para convertir todos los bits del número en código Gray.

Nota: El bit que se cambia o se repite, es sólo en la palabra binaria. El código Gray, actúa como control.

Ejemplo: $1010_{\ GRAY} = 1100_{\ 2} = 12_{\ 10}$

Ejercicios:

1) $011010111101_{\ 2} = N_{\ GRAY}$ R: $010111100011_{\ GRAY}$

2) $111010101110_{\ 2} = N_{\ GRAY}$ R: $100111111001_{\ GRAY}$

3) $011010111101_{\ 2} = N_{\ GRAY}$ R: $010111100011_{\ GRAY}$

4) $1001110101101_{\ GRAY} = N_{\ 2}$ R: $1110100110110_{\ 2}$

5) $10101010101_{\ GRAY} = N_{\ 2}$ R: $11001100110_{\ 2}$

6) $01110111011110_{\ GRAY} = N_{\ 2}$ R: $01011010010100_{\ 2}$

7) $689_{\ 10} = N_{\ NBCD} = N_{\ BCD}$ R: $011010001001_{\ BCD}$

8) $164_{\ 8} = N_{\ NBCD} = N_{\ BCD}$ R: $000100010110_{\ BCD}$

9) $01001001_{\ BCD} = N_{\ 8}$ R: $61_{\ 8}$

10) $10010_{2} = N_{BCD}$ R: 00011000_{BCD}

11) $37_{8} + 10001101000_{BCD} = N_{16}$ R: $1F3_{16}$

12) $FA_{16} = N_{GRAY}$ R: 10000111_{GRAY}

13) $11011_{GRAY} = N_{NBCD}$ R: 00011000_{NBCD}

14) $1010111011_{2} - 7A2_{16} = N_{BCD}$ R: -0001001001010101_{BCD}

15) $111001_{BCD} = N_{GRAY}$ R: 0110100_{GRAY}

16) $11001_{BCD} + 1011_{2} = N_{2}$ R: 011110_{2}

17) $1A_{16} + 1011_{2} = N_{BCD}$ R: 00110111_{BCD}

18) $10111_{2} - 101001_{BCD} = N_{16}$ R: -6_{16}

19) $1010010_{BCD} + 1011_{GRAY} = N_{BCD}$ R: 01100101_{BCD}

20) $70_{8} - 10011_{GRAY} = N_{GRAY}$ R: 010110_{GRAY}

c) CÓDIGOS DETECTORES Y AUTOCORRECTORES DE ERRORES:

El cambio de un bit en el almacenamiento o la manipulación de datos, origina resultados erróneos. Para detectar e incluso corregir estas alteraciones, se usan códigos que agregan "bits redundantes". Estos códigos reciben el nombre de "Códigos de Paridad".

Los bits redundantes se denominan "bits de paridad". Los mismos se agregan al dato en el momento de su envío y son corroborados en el momento de su recepción por algún componente de la computadora.

Pueden ser códigos de **paridad par** o de **paridad impar**. El bit que se agrega hará que el total de "1" en un dato sea par o impar, según la paridad elegida. Cuando se recibe un número, se chequea la paridad (par o impar, según se haya elegido), y se toma como correcto si pasa el test.

Al código BCD, se le puede agregar un bit de paridad (par o impar), a la derecha.

El Código Autocorrector de Hamming permite detectar y corregir errores producidos en una trnasmisión, agregando bits de paridad.

Ejercicios:

1) Decir “FALSO” o “VERDADERO” en las siguientes palabras en código BCD con paridad:

a) 1001 0	paridad impar	(R: FALSO)
b) 1010 0	paridad par	(R: FALSO, pues no es BCD)
c) 1000 0	paridad par	(R: FALSO)
d) 0111 0	paridad impar	(R: VERDADERO)
e) 1011 0	paridad impar	(R: FALSO, pues no es BCD)

2) Escribir en BCD con la paridad indicada:

a) 961_{10} con bit de paridad par, por dígito.	(R: 100100110000011)
b) 5798_{10} con bit de paridad impar, por dígito.	(R: 01011011101001110000)
c) 620_{10} con bit de paridad impar por dígito.	(R: 011010010000001)
d) 6058_{10} con bit de paridad par por dígito.	(R: 01100000000101010001)

3) Decir FALSO o VERDADERO en las siguientes palabras con paridad:

a) 11110001 1	paridad par	(R: VERDADERO)
b) 00000111 1	paridad impar	(R: FALSO)
c) 110110010	paridad par	(R: FALSO)
d) 111000111	paridad impar	(R: FALSO)
e) 011111101	paridad impar	(R: VERDADERO)

4) Colocar el bit necesario para obtener paridad par en las siguientes palabras:

a) 01110100	(R: 0)
b) 11100110	(R: 1)
c) 11111111	(R: 0)

5) Colocar el bit necesario para obtener paridad impar en las siguientes palabras:

a) 01101100	(R: 1)
b) 10001001	(R: 0)
c) 00001001	(R: 1)

II) CÓDIGOS ALFANUMÉRICOS:

Representan caracteres alfabéticos, numéricos y caracteres especiales (>, <, {, }, +, %, =, $, -, ?, etc.). Es usual que los códigos alfanuméricos sufran pequeñas alteraciones, al ser utilizados en países con distintos alfabetos.

CÓDIGO **ASCII**:

ASCII: American Standard Code for Information Interchange o Código estandard americano para el intercambio de información.

Este código se ha adoptado a nivel internacional.

Su uso primordial es facilitar el intercambio de información entre sistemas de procesamiento de datos.

Inicialmente, se desarrolló con 6 bits, permitiendo 2^6 (64) configuraciones distintas: 10 caracteres numéricos decimales, 26 caracteres alfabéticos y 28 símbolos especiales.

Posteriormente, se aumentó a 7 bits, obteniéndose 2^7 (128) configuraciones diferentes, que incluían los mismos caracteres alfanuméricos de la primera versión, más algunas expresiones sintácticas y símbolos que permitían realizar instrucciones operativas de unidades periféricas. No contemplaba ni caracteres especiales ni caracteres específicos de otras lenguas que no fuese la lengua inglesa.

Actualmente utilizan 8 bits, o sea, $2^8 = 256$ configuraciones distintas.

La siguiente tabla se refiere al código ASCII normalizado, donde se codifican 128 caracteres. El octavo bit vale cero, y se trata de un bit de "reserva", que también puede valer uno, según sea usado para verificación de paridad, selección de caracteres sobre fondo blanco o negro, caracteres gráficos especiales (incluídas las vocales con acentos), etc.

De esta forma, cualquier carácter de un teclado tiene asignado un código en un formato de 8 bits, constituyendo un byte de información.

Resulta entonces equivalente hablar de letras, caracteres, bytes u octetos.

En el renglón superior, están los tres bits más pesados de la palabra en código ASCII, y en la columna de la izquierda, los cuatro bits más livianos.

Ejemplos: $A = 41_{16} = 01000001_2$ SP (espacio) $= 20_{16} = 00100000_2$

	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000 0	**NUL** 0	**DLE** 16	**SP** 32	**0** 48	**@** 64	**P** 80	**`** 96	**p** 112
0001 1	**SOH** 1	**DC1** 17	**!** 33	**1** 49	**A** 65	**Q** 81	**a** 97	**q** 113
0010 2	**STX** 2	**DC2** 18	**“** 34	**2** 50	**B** 66	**R** 82	**b** 98	**r** 114
0011 3	**ETX** 3	**DC3** 19	**#** 35	**3** 51	**C** 67	**S** 83	**c** 99	**s** 115
0100 4	**EOT** 4	**DC4** 20	**$** 36	**4** 52	**D** 68	**T** 84	**d** 100	**t** 116
0101 5	**ENQ** 5	**NAK** 21	**%** 37	**5** 53	**E** 69	**U** 85	**e** 101	**u** 117
0110 6	**ACK** 6	**SYN** 22	**&** 38	**6** 54	**F** 70	**V** 86	**f** 102	**v** 118
0111 7	**BEL** 7	**ETB** 23	**’** 39	**7** 55	**G** 71	**W** 87	**g** 103	**w** 119
1000 8	**BS** 8	**CAN** 24	**(** 40	**8** 56	**H** 72	**X** 88	**h** 104	**x** 120
1001 9	**HT** 9	**EM** 25	**)** 41	**9** 57	**I** 73	**Y** 89	**i** 105	**y** 121
1010 A	**LF** 10	**SUB** 26	*** 42	**:** 58	**J** 74	**Z** 90	**j** 106	**z** 122
1011 B	**VT** 11	**ESC** 27	**+** 43	**;** 59	**K** 75	**[** 91	**k** 107	**{** 123
1100 C	**FF** 12	**FS** 28	**´** 44	**<** 60	**L** 76	**** 92	**l** 108	**|** 124
1101 D	**CR** 13	**GS** 29	**-** 45	**=** 61	**M** 77	**]** 93	**m** 109	**}** 125
1110 E	**SO** 14	**RS** 30	**.** 46	**>** 62	**N** 78	**^** 94	**n** 110	**~** 126
1111 F	**SI** 15	**US** 31	**/** 47	**?** 63	**O** 79	**_** 95	**o** 111	**DEL** 127

Caracteres de control ASCII:

NUL Nulo
SOH Comienzo de cabeza
STX Comienzo de texto
ETX Final de texto
EOT Final de transmisión
ENQ Petición, consulta
ACK Acuse de recibo
BEL Pitido
BS Retroceso de un espacio
HT Tabulación horizontal
LF Saltar a línea siguiente
VT Tabulación vertical
FF Alimentación de hoja
CR Retorno de carro
SO Fuera de código
SI Dentro de código

DLE Escape del enlace de datos. Carácter de control que cambia el significado del carácter que se da a continuación
DCi Control del dispositivo *i*
NAK Acuse de recibo negativo
SYN Sincronización
ETB Final de bloque de transmisión
CAN Anulación
EM Fin de soporte (de cinta, etc)
SUB Sustituir
ESC Escape
FS Separador de archivo
GS Separador de grupo
RS Separador de registro
US Separador de sub-registro (campo)
DEL Borrar, suprimir

CÓDIGO **EBCDIC**:

EBCDIC: Extended Binary Coded Decimal Information Code o código BCD extendido.

Este código fue creado por IBM, y es, después del ASCII, el más utilizado. Surge como una ampliación del código BCD.

En las transmisiones de datos es necesario utilizar un gran número de caracteres de control para la manipulación de los mensajes y realización de otras funciones. De ahí que el código BCD se extendiera a una representación utilizando 8 bits, dando origen al código EBCDIC.

Deja varias de las 256 configuraciones, sin ningún significado.

Al requerir 8 bits por carácter, no tiene posibilidad de reservar un bit para usos especiales.

En el renglón superior de la siguiente tabla, correspondiente al código EBCDIC, están los cuatro bits más pesados de la palabra, y en la columna de la izquierda, los cuatro bits más livianos.

	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	**NUL** 0	**DLE** 16	**DS** 32	 48	**SP** 64	**&** 80	- 96	 112	 128	 144	 160	 176	**{** 192	**}** 208	\ 224	**0** 240
0001 1	**SOH** 1	**DCI** 17	**SOS** 33	 49	 65	/ 81	 97	 113	**a** 129	**j** 145	 161	 177	**A** 193	**J** 209	 225	**1** 241
0010 2	**STX** 2	**DC2** 18	**FS** 34	**SYN** 50	 66	 82	 98	 114	**b** 130	**k** 146	**s** 162	 178	**B** 194	**K** 210	**S** 226	**2** 242
0011 3	**ETX** 3	**TM** 19	 35	 51	 67	 83	 99	 115	**c** 131	**l** 147	**t** 163	 179	**C** 195	**L** 211	**T** 227	**3** 243
0100 4	**PF** 4	**RES** 20	**BYP** 36	**PN** 52	 68	 84	 100	 116	**d** 132	**m** 148	**u** 164	 180	**D** 196	**M** 212	**U** 228	**4** 244
0101 5	**HT** 5	**NL** 21	**LF** 37	**RS** 53	 69	 85	 101	 117	**e** 133	**n** 149	**v** 165	 181	**E** 197	**N** 213	**V** 229	**5** 245
0110 6	**LC** 6	**BS** 22	**ETB** 38	**UC** 54	 70	 86	 102	 118	**f** 134	**o** 150	**w** 166	 182	**F** 198	**O** 214	**W** 230	**6** 246
0111 7	**DEL** 7	**IL** 23	**ESC** 39	**EOT** 55	 71	 87	 103	 119	**g** 135	**p** 151	**x** 167	 183	**G** 199	**P** 215	**X** 231	**7** 247
1000 8	 8	**CAN** 24	 40	 56	 72	 88	 104	 120	**h** 136	**q** 152	**y** 168	 184	**H** 200	**Q** 216	**Y** 232	**8** 248
1001 9	**RLF** 9	**EM** 25	 41	 57	 73	 89	 105	\ 121	**i** 137	**r** 153	**z** 169	 185	**I** 201	**R** 217	**Z** 233	**9** 249
1010 A	**SMM** 10	**CC** 26	**SM** 42	 58	**cent** 74	**!** 90	| 106	**:** 122	 138	 154	 170	 186	 202	 218	 234	 250
1011 B	**VT** 11	**CU1** 27	**CU2** 43	**CU3** 59	**.** 75	**$** 91	**,** 107	**#** 123	 139	 155	 171	 187	 203	 219	 235	 251
1100 C	**FF** 12	**IFS** 28	 44	**DC4** 60	< 76	* 92	**%** 108	**@** 124	 140	 156	 172	 188	 204	 220	 236	 252
1101 D	**CR** 13	**IGS** 29	**ENQ** 45	**NAK** 61	**(** 77	**)** 93	_ 109	**'** 125	 141	 157	 173	 189	 205	 221	 237	 253
1110 E	**SO** 14	**IRS** 30	**ACK** 46	 62	+ 78	**;** 94	>** 110	= 126	 142	 158	 174	 190	 206	 222	 238	 254
1111 F	**SI** 15	**IUS** 31	**BEL** 47	**SUB** 63	| 79	¬ 95	**?** 111	**"** 127	 143	 159	 175	 191	 207	 223	 239	 255

Caracteres de control EBCDIC:

NUL	Nulo
SOH	Comienzo de cabeza
SOT	Comienzo de texto
EOT	Final de texot
PF	Perforadora desconectada
HT	Tabulación horizontal
LC	Minúscula
DEL	Eliminar, borrar
RLF	Alimentación de linea invertida
SMM	Comienzo mensaje manual
VT	Tabulación vertical
FF	Alimentación de hoja
CR	Retorno de carro
SO	Fuera de código
SI	Dentro de código
DLE	Escape del enlace de datos
TM	Marca de cinta
RES	Restaurar
NL	Pasar a línea siguiente
BS	Retroceso de un espacio
IL	*sin función*
CAN	Cancelar
EM	Final de soporte
CC	Control del cursor
SP	Espacio en blanco
IFS	Separador para intercambio de archivos
IGS	Separador para intercambio de grupos
IRS	Separador para intercambio de registros
IUS	Separador para intercambio de unidad
DS	Selección de dígito
SOS	Comienzo de significado
FS	Separador de campo
BYP	Desviar
LF	Alimentación de línea
ETB	Final de bloque de transmisión
ESC	Escape
SM	Fijar modo
ENQ	Solicitud, petición
ACK	Acuse de recibo
BEL	Pitido
SYN	Sincronzación
PN	Perforadora conectada
RS	Detener lectora
UC	Mayúsculas
EOT	Fin de transmisión
NACK	Acuse de recibo negativo
SUB	Sustituir
DC*i*	Control dispositivo *i*
CU*i*	Control usuario *i*