Introducción al estudio de los circuitos lógicos y sistemas numéricos.

Sistemas numéricos

Un sistema numérico son un conjunto de símbolos y reglas que se utilizan para representar datos numéricos o cantidades. Se caracterizan por su base que indican el número de símbolos distinto que utiliza y además es el coeficiente que determina cual es el valor de cada símbolo dependiendo de la posición que ocupe. Estas cantidades se caracterizan por tener dígitos enteros y fraccionarios.

Si a_j indica cualquier dígito de la cifra, b la base del sistema de numeración y además de esto la cantidad de dígitos enteros y fraccionarios son n y k respectivamente, entonces el número representado en cualquier base se puede expresar de la siguiente forma:

$$N_b = [a_{n-1}.a_{n-2}.a_{n-3}.....a_3.a_2.a_1.a_0.a_{-1}.a_{-2}.a_{-3}.....a_{-k}]_b$$

Donde: **j** = {n-1, n-2,.....2, 1, 0,-1, -2, ..., -k} y **n + k** indica la cantidad de dígitos de la cifra.

Por ejemplo, el número 31221, 32_4 en base cuatro tiene **n=5** y **k=2** con la parte entera: $\mathbf{a_{n-1}=a_4}=3$; $\mathbf{a_3}=1$; $\mathbf{a_2}=2$; $\mathbf{a_1}=2$; $\mathbf{a_0}=1$ y parte fraccionaria $\mathbf{a_{-1}}=3$; $\mathbf{a_{-2}}=2$

SISTEMA DECIMAL.

Este es el sistema que manejamos cotidianamente, está formado por diez símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} por lo tanto la base del sistema es diez (10).

SISTEMA BINARIO.

Es el sistema que utiliza internamente el hardware de las computadoras actuales, se basa en la representación de cantidades utilizando los dígitos 1 y 0. Por tanto su base es 2 (número de dígitos del sistema). Cada dígito de un número en este sistema se denomina bit (contracción de **bi**nary digi**t**). Se puede utilizar con nombre propio determinados conjuntos de dígitos en binario. Cuatro bits se denominan **cuaterno** (ejemplo: 1001), ocho bits **octeto** o **byte** (ejemplo: 10010110), al conjunto de 1024 bytes se le llama **Kilobyte** o simplemente **K,** 1024 Kilobytes forman un **megabyte** y 1024 megabytes se denominan **Gigabytes.**

SISTEMA OCTAL.

El sistema numérico octal utiliza ocho símbolos o dígitos para representar cantidades y cifras numéricas. Los dígitos son: {0, 1, 2, 3, 4, 5, 6, 7}; la base de éste es ocho (8) y es un sistema que se puede convertir directamente en binario como se verá más adelante.

SISTEMA HEXADECIMAL.

El sistema numérico hexadecimal utiliza dieciséis dígitos y letras para representar cantidades y cifras numéricas. Los símbolos son: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}; la base del sistema es dieciséis (16). También se puede convertir directamente en binario como se verá más adelante. En la tabla 1.1 se muestran los primeros veintiuno números decimales con su respectiva equivalencia binaria, octal y hexadecimal.

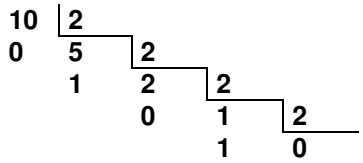
DECIMAL	BINARIO	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Tabla 1.1. Equivalencia entre sistemas de los primeros veintiuno números decimales.

CONVERSIÓN ENTRE LOS SISTEMAS NUMÉRICOS

CONVERSIÓN DECIMAL-BINARIO: Los métodos mas conocidos son:

1. **Divisiones sucesivas entre 2:** Consiste en dividir sucesivamente el número decimal y los cocientes que se van obteniendo entre 2, hasta que una de las divisiones se haga 0. La unión de todos los restos obtenidos escritos en orden inverso, nos proporcionan el número inicial expresado en el sistema binario. **Ej.:**



$10_{(10)}=1010_{(2)}$

2. Multiplicación sucesiva por 2: Se utiliza para convertir una fracción decimal a binario, consiste en multiplicar dicha fracción por 2, obteniendo en la parte entera del resultado el primero de los dígitos binarios de la fracción binaria que buscamos. A continuación repetimos el mismo proceso con la parte fraccionaria del resultado anterior, obteniendo en la parte entera del nuevo resultado el segundo de los dígitos buscados. Iteramos sucesivamente de esta forma, hasta que desaparezca la parte fraccionaria o hasta que tengamos los suficientes dígitos binarios que nos permitan no sobrepasar un determinado error.

Ejemplo:

Convertir la fracción decimal 0.0828125 en fracciones binarias

0.828125 x 2 = **1**.656250
0.656250 x 2 = **1**.31250
0.31250 x 2 = **0**.6250
0.6250 x 2 = **1**.250
0.250 x 2 = **0**.50
0.50 x 2 = **1**.0

$0.828125_{10} \rightarrow 0.110101_2$

3. Métodos de las restas sucesivas de las potencias de 2: Consiste en tomar el numero a convertir y buscar la potencia de 2 mas grande que se pueda restar de dicho numero, tomando como nuevo numero para seguir el proceso el resultado de la resta. Se repiten las mismas operaciones hasta que el número resultante en una de las restas es 0 o inferior al error que deseamos cometer en la conversión. El numero binario resultante será un uno (1) en las posiciones correspondientes a las potencias restadas y un cero (0) en las que no se han podido restar. Ej.
Convertir el número decimal 1994 a binario.

Posición	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valor	1024	512	256	128	64	32	16	8	4	2	1
Digito	1	1	1	1	1	0	0	1	0	1	0

1994	-	1024	=	970
970	-	512	=	458
458	-	256	=	202
202	-	128	=	74
74	-	64	=	10
10	-	8	=	2

Resp: $1994_{10} \rightarrow 11111001010_2$

CONVERSIÓN DE BINARIO A DECIMAL: El método consiste en reescribir él número binario en posición vertical de tal forma que la parte de la derecha quede en la zona superior y la parte izquierda quede en la zona inferior. Se repetirá el siguiente proceso para cada uno de los dígitos comenzados por el inferior: Se coloca en orden descendente la potencia de 2 desde el cero hasta n, donde el mismo el tamaño del número binario, el siguiente ejemplo ilustra de la siguiente manera. Utilizando el teorema fundamental de la numeración tenemos que 1001.1es igual a:

$$1*2^3+0*2^2+0*2^1+1*2^0+1*2^{-1}=9.5_{(10)}$$

CONVERSIÓN DECIMAL – OCTAL: Consiste en dividir un número y sus sucesivos cocientes obtenidos por ocho hasta llegar a una división cuyo cociente sea 0. El numero Octal buscado es el compuesto por todos los restos obtenidos escritos en orden inverso a su obtención. **Ej.:**

1992	**8**		
39	**249**	**8**	
72	**09**	**31**	**8**
0	**1**	**7**	**3**

$1000_{(10)}=3710_{(8)}$

CONVERSIÓN DE UNA FRACCIÓN DECIMAL A UNA OCTAL: Se toma la fracción decimal y se multiplica por 8, obteniendo en la parte entera del resultado el primer dígito de la fracción octal resultante y se repite el proceso con la parte decimal del resultado para obtener el segundo dígito y sucesivos. El proceso termina cuando desaparece la parte fraccionaria del resultado o dicha parte fraccionaria es inferior al error máximo que deseamos obtener. **Ej. :**

0.140625*8=1.125
0.125*8=1.0
$0.140625_{(10)}=0.11_{(8)}$

CONVERSIÓN OCTAL A DECIMAL: Existen varios métodos siendo el más generalizado el indicado por el TFN (Teorema fundamental de la numeración) que hace la conversión de forma directa por medio de la formula. **Ej. :** utilizando el teorema fundamental de la numeración tenemos que 4701 es igual a:

$$4*8^3+7*8^2+0*8^1+1*8^0=2497_{(10)}$$

Conversión decimal – hexadecimal: Se divide el numero decimal y los cocientes sucesivos por 16 hasta obtener un cociente igual a 0. El número hexadecimal buscado será compuesto por todos los restos obtenidos en orden inverso a su obtención. **Ej.:**

1000	**16**	
40	**62**	**16**
8	**14**	**3**

$1000_{(10)}$=3E8$_{(16)}$

CONVERSIÓN DE UNA FRACCIÓN DECIMAL A HEXADECIMAL: a la fracción decimal se multiplica por 16, obteniendo en la parte entera del resultado el primer dígito de la fracción hexadecimal buscada, y se repite el proceso con la parte fraccionaria de este resultado. El proceso se acaba cuando la parte fraccionaria desaparece o hemos obtenido un número de dígitos que nos permita no sobrepasar el máximo error que deseemos obtener. **Ej.:** Pasar a hexadecimal la fracción decimal 0.06640625

0.06640625*16=1.0625
0.0625*16 = 1.0
Luego $0.06640625_{(10)}=0.11_{(16)}$

CONVERSIÓN HEXADECIMAL- DECIMAL: el método más utilizado es el TFN que nos da el resultado por la aplicación directa de la formula. **Ej. :** utilizando el teorema fundamental de la numeración tenemos que $2*16^2+C*16^1+A*16^0=714_{(10)}$ 2CA es igual a:

CONVERSIÓN DE HEXADECIMAL-BINARIO: para convertir un número hexadecimal a binario, se sustituye cada dígito hexadecimal por su representación binaria según la siguiente tabla.

Dígito Hexadecimal	Dígito Binarios
0	**0000**
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**0101**
6	**0110**
7	**0111**
8	**1000**
9	**1001**
A	**1010**
B	**1011**
C	**1100**
D	**1101**
E	**1110**
F	**1111**

Ej.: pasar el número 2BC a binario

2	**B**	**C**
0010	**1011**	**1100**

Finalmente él número hexadecimal en binario es igual a: 001010111100

CONVERSIÓN DE OCTAL A BINARIO: para convertir un numero octal a binario se sustituye cada dígito octal en por sus correspondientes tres dígitos binarios según la siguiente tabla.

Dígito Octal	Dígito Binario
0	**000**
1	**001**
2	**010**
3	**011**

4	**100**
5	**101**
6	**110**
7	**111**

Ej.: Convertir el número octal 1274 en binario.

1	**2**	**7**	**4**
001	**010**	**111**	**100**

Por lo tanto el número octal en binario es igual a: 001010111100

OPERACIONES ARITMÉTICAS DE LOS DISTINTOS SISTEMAS.

Al igual que en el sistema decimal, también en otros sistemas de numeración, se pueden realizar operaciones aritméticas, tales como: suma, resta, multiplicación y división tomando como referencia la base del sistema dado.

SUMA BINARIA, OCTAL Y HEXADECIMAL.

En general, para realizar la suma se procede de la misma forma como se hace en el sistema decimal. Por ejemplo, si $a_{n-1}a_{n-2}.....a_2a_1a_0,a_{-1}a_{-2}....a_{-k}$ es un número dado en una base ***b*** y $h_{n-1}h_{n-2}.....h_2h_1h_0,h_{-1}h_{-2}....h_{-k}$ es otro dado en la misma base entonces la suma se debe realizar de la siguiente forma:

$$\begin{array}{cccccccccc} a_{n-1} & a_{n-2} & ..... & a_1 & a_0, & a_{-1} & .... & a_{-k} & + \\ h_{n-1} & h_{n-2} & ..... & h_1 & h_0, & h_{-1} & .... & h_{-k} & \end{array}$$

$$(a_{n-1}+h_{n-1}+c_{n-2})(a_{n-2}+h_{n-2}+c_{n-3}).....(a_1+h_1+c_0)(a_0+h_0+c_{-1}),(a_{-1}+h_{-1}+c_{-2}).....(a_{-k}+h_{-k})$$

Los dígitos $m_j=(a_j+h_j+c_{j-1})$ pertenecientes al resultado se forman sumando los dígitos de cada columna de los cosumandos, más el acarreo c_{j-1} que viene de la columna anterior. Cada unidad de acarreo tiene el mismo valor de la base del sistema, por ejemplo, en la suma binaria es dos, en octal ocho y en hexadecimal dieciséis. Por ejemplo, llevar 2 en hexadecimal significa que el acarreo es el doble de la base y vale exactamente 32; de este mismo modo, en binario equivale a 4 veces y 16 en octal. Los acarreos aparecen cuando las semisumas de las columnas superan la base del sistema numérico.

SUMA BINARIA: Las operaciones de suma binaria se realizan de la siguiente forma:

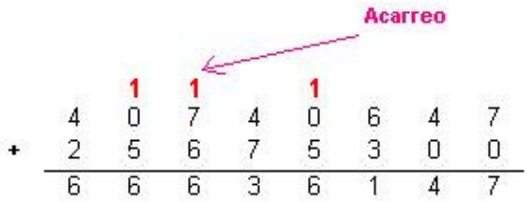
0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	Llevo 1

Ejemplo: Dado los números binarios: W=111110001_2; T=1101110101_2; Obtener W+T

0	1	1	1	1	1	0	0	0	0	1
0	1	1	0	1	1	1	0	1	0	1
1	1	1	0	1	0	1	0	1	1	0

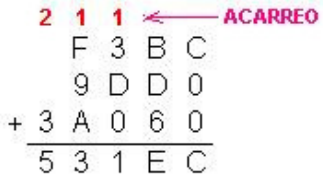
SUMA OCTAL: Se debe restar o dividir la semisuma de cada columna, cuando la misma exceda la base del sistema, y colocar en la columna inmediata del lado izquierdo, el valor del acarreo tantas veces se haya superado la base del sistema. De esta misma forma cada unidad que se acarree equivale a ocho unidades de la columna anterior.

Ejemplo: Dado los números binarios: A. 40740647 y B. 25675300, Obtener A+B



SUMA HEXADECIMAL: Se debe restar o dividir la semisuma de cada columna, cuando la misma exceda la base del sistema, y colocar en la columna inmediata del lado izquierdo, el valor del acarreo tantas veces se haya superado la base del sistema. Cada unidad que se acarree equivale a dieciséis unidades de la columna anterior.

Ejemplo: Dado los números binarios:

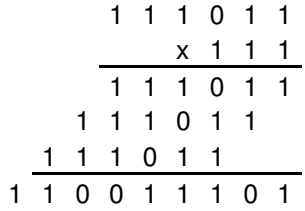


MULTIPLICACIÓN BINARIA, OCTAL Y HEXADECIMAL.

La operación aritmética de multiplicar se realiza del mismo modo que en el sistema numérico decimal.

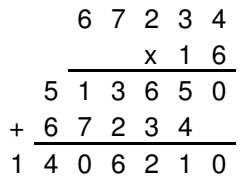
MULTIPLICACIÓN BINARIA:

Ej: Multiplicar A. 111011_2 y B. 111_2



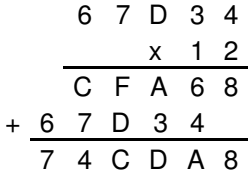
MULTIPLICACIÓN OCTAL:

Ej: Multiplicar A. 67234_8 y B. 16_8



MULTIPLICACIÓN HEXADECIMAL:

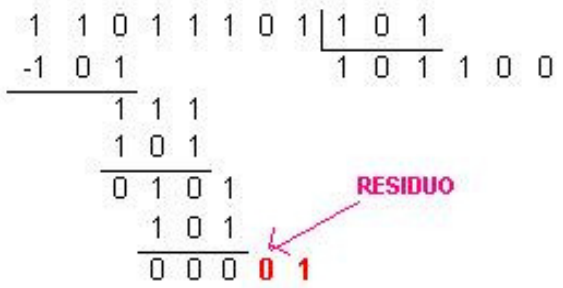
Ej: Multiplicar A. $67D34_{16}$ y B. 12_{16}



DIVISIÓN BINARIA, OCTAL Y HEXADECIMAL.

La operación aritmética de dividir se realiza del mismo modo que en el sistema numérico decimal.

DIVISIÓN BINARIA:



DIVISIÓN OCTAL Y HEXADECIMAL: La división se efectúa del mismo modo que en el sistema decimal y se realiza directamente en la misma base del sistema octal o hexadecimal. Sin embargo, también se puede obtener previamente la conversión en binario y proceder, como en el caso anterior, a realizarla en binario; y después el resultado transformarlo de nuevo al sistema numérico original.

COMPLEMENTO DE UN NÚMERO CON RESPECTO A LA BASE DEL SISTEMA.

Las representaciones de los números en los distintos sistemas son hechas por convenciones y acuerdos. La finalidad de esto es buscar formas sencillas de manejar universalmente operaciones y representaciones numéricas, representar números fraccionarios, números negativos, etc. El complemento de un número sirve para normalizar y reglamentar las operaciones aritméticas con signo, de forma que puedan ser procesadas por los circuitos internos de una calculadora o computadora.

El complemento a la base de un número se define por la siguiente fórmula:

$N_b^C = b^n - N_b$ **(Ec.1.3)** donde N_b^C es el número complementado a la base b del sistema, ***n*** la cantidad de dígitos y N_b es el número dado.

Ejemplo: Hallar el complemento a diez del número 897324_{10}
Solución: El número esta dado en el sistema decimal y la cantidad de dígitos es seis

$$N_{10}^C = 10^6 - 897324_{10} = 102676_{10}$$

Ejemplo: Hallar el complemento a dieciséis del número $A9EFC21_{16}$
Solución: El número está dado en el sistema hexadecimal y la cantidad de dígitos es siete.

$$N_{16}^C = 16^7 - A9EFC21_{16} = 10000000_{16} - A9EFC21_{16} = 56103DF_{16}$$

Ejemplo: Hallar el complemento a ocho del número 60472_8
Solución: El número está dado en el sistema octal y la cantidad de dígitos es cinco.

$$N_8^C = 8^5 - 60472_8 = 100000_8 - 60472_8 = 17306_8$$

Ejemplo: Hallar el complemento a dos del número 100111011101_2
Solución: El número está dado en el sistema binario y la cantidad de dígitos es doce.

$$N_2^C = 2^{12} - 100111011101_2 = 1000000000000_2 - 100111011101_2 = 011000100011_2$$

COMPLEMENTO DISMINUIDO EN UNO A LA BASE DEL SISTEMA.

Existe otra forma de hallar el complemento a la base del sistema, ésta es, obteniendo el complemento disminuido a uno y luego sumando uno. Para obtener esta fórmula se procede con un artificio en la **Ec.1.3** de la siguiente forma:

$N_b^C = (b^n - N_b) + 1 - 1 = [(b^n - 1) - N_b] + 1$ **(Ec.1.3.1)**. El valor $N_b^{C-1} = (b^n - 1) - N_b$ **(Ec.1.4)**

Se conoce como el complemento de la base disminuido a uno. También se le denomina ***complemento a uno*** del sistema numérico correspondiente y por lo tanto, para hallar el complemento a la base solamente se le debe sumar uno a la **(Ec.1.4)**.

COMPLEMENTO DISMINUIDO A UNO DEL SISTEMA BINARIO, OCTAL Y HEXADECIMAL.

El complemento disminuido a uno se obtiene aplicando la **Ec.1.4** en cualquiera de los sistemas numéricos. La expresión ***(b^n-1)*** se debe usar como minuendo en el tope de la potencia ***b^n***

menos uno, lo que significa tener una cifra compuesta por los dígitos más significativos y de mayor valor del sistema numérico. Por ejemplo, para hallar el minuendo de 56437_{8}, en el sistema octal, se procede de la siguiente forma:
n=5; entonces 8^5 -1=100000_{8} -1=77777_{8}. Ahora, para hallar el complemento disminuido a uno se resta el número dado: $N_b^{C-1} = 77777_8 - 56437_8 = 21340_8$.

Ejemplo: Hallar el complemento disminuido a uno de los siguientes números:
a) $24BCA0F7_{16}$; b) 10011101101_{2}; c) 1265730_{8}

Sol. (a): $N_{16}^{C-1} = (16^8 - 1) - 24BCA0F7_{16} = FFFFFFFF_{16} - 24BCA0F7_{16} = DB435F08_{16}$

Sol. (b): $N_2^{C-1} = (2^{11} - 1) - 10011101101_2 = 11111111111_2 - 10011101101_2 = 01100010010_2$

Sol. (c): $N_8^{C-1} = (8^7 - 1) - 1265730_8 = 7777777_8 - 1265730_8 = 6512047_8$

En cualquier sistema de numeración el complemento disminuido a uno se puede hallar con la fórmula resultante de la Ec.1, Ec.2 y Ec.3 de la siguiente forma:

$[(b^n - 1) - N_b] = [(b-1)(b-1)....(b-1)(b-1) - (a_{n-1})(a_{n-2})....(a_1)(a_0)]$ Donde cada ***(b-1)*** corresponde al dígito de mayor peso en el sistema de numeración de base ***b.*** Los a_j son los ***n*** dígitos del número que se va complementar, con ***j=0,1,....,n-2,n-1.*** El complemento disminuido a uno se halla, en forma directa, de la siguiente manera:

$$N_b^{C-1} = [(b-1) - a_{n-1}][(b-1) - a_{n-2}].....[(b-1) - a_2][(b-1) - a_1][(b-1) - a_0] \quad \textbf{(Ec.1.4.1).}$$

Ejemplo: Hallar el complemento disminuido a uno de los siguientes números:
a) $FCBC40_{16}$; b) 101011011_{2}

Solución (a): $N_{16}^{C-1} = FFFFFF_{16} - FCBC40_{16} = 0343BF_{16}$

Solución (b): $N_2^{C-1} = 111111111_2 - 101011011_2 = 010100100_2$

COMPLEMENTO A UNO.

Es un caso particular del complemento disminuido a uno de la base binaria, tiene muchas aplicaciones en los circuitos digitales y sistemas de computación. Sirven para representar tablas numéricas de cantidades positivas y negativas, invertir los estados de los bits que conforman el dato binario y es utilizado como paso previo para hallar el complemento a dos. De la **Ec.1.4** se puede determinar que el complemento a uno se obtiene invirtiendo el estado o nivel de los bits que conforman la cifra.

Ejemplo: Hallar el complemento a uno de los siguientes números binarios:
a) $1100010101011110 10_{2}$; b) 101011010101_{2}

Solución (a): $N_2^{C-1} = 0011101010100001 01_2$

Solución (b): $N_2^{C-1} = 010100101010_2$

COMPLEMENTO A DOS.

Es un caso particular del complemento a la base del sistema binario, tiene muchas aplicaciones en los circuitos digitales y sistemas de computación. Sirven para representar tablas numéricas de cantidades positivas y negativas, invertir los estados de los bits que conforman el dato binario y realizar operaciones aritméticas con signo en el sistema binario. Con la **Ec.1.3** se puede determinar el complemento a dos de un número binario; no obstante, con la misma ecuación se puede hallar un método directo para obtener también el complemento a dos. Este método consiste en ir seleccionando y colocando de derecha a izquierda los dígitos binarios hasta conseguir el primer bit en uno, de allí en adelante se cambian de estado todos los bits restantes.

El otro método para hallar el complemento a dos consiste en obtener el complemento a uno de la cifra y luego sumarle uno; esto último está reflejado en la **(Ec.1.3.1).**

Ejemplo: Hallar el complemento a dos de los siguientes números binarios:
a) 101100101010111_{2}; b) 10001101000100_{2}; c) 10111001110000_{2}
Aplicando el método con la **(Ec.2.1)**;

Solución (a): $N_2^C = 010011010101000_2 + 1 = 010011010101001_2$

Solución (b): $N_2^C = 01110010111011_2 + 1 = 01110010111100_2$

Solución (c): $N_2^C = 01000110001111_2 + 1 = 01000110010000_2$

REPRESENTACIÓN NUMÉRICA EN COMPLEMENTO A DOS.

En el sistema binario, la forma más utilizada para representar los números enteros con signo es la de complemento a dos. Los circuitos microprocesadores poseen internamente unidades de procesamiento aritmético que trabajan bajo éste formato, el cual puede estar constituido por n bits múltiplos de la potencia de base dos. Por ejemplo, para representar los números positivos y negativos se definen datos con tamaño estándar: ocho bits, 16 bits, 32 bits, etc.

En este formato, el bit más significativo (MSB) del dato se utiliza para indicar el signo y los bits restantes representan la magnitud del número. En la figura 1.2 se puede apreciar la representación del formato utilizado para 16 bits, donde el más significativo (B15) indica que el signo es negativo si vale uno o positivo si vale cero. Las cantidades positivas se encuentran en binario normal mientras que los números negativos están en complemento a dos, esto significa que estos últimos, se deben complementar para poder hallar su verdadero valor.

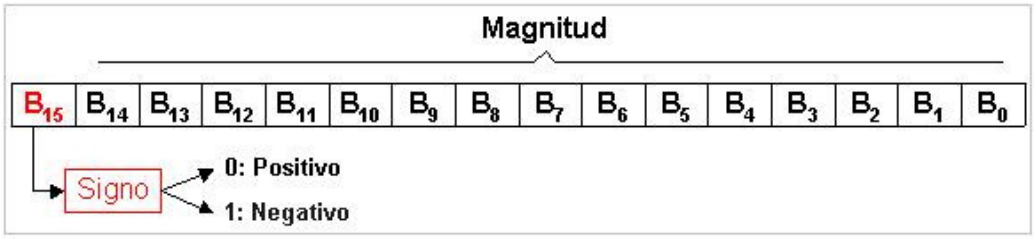


Figura 1.2. Formato de 16 bits para representación numérica con signo.

Número entero	Formato de 16 bits
32767	0111111111111111
.	.
.	.
+5	0000000000000101
+4	0000000000000100
+3	0000000000000011
+2	0000000000000010
+1	0000000000000001
0	**0000000000000000**
-1	1111111111111111
-2	1111111111111110
-3	1111111111111101
-4	1111111111111100
-5	1111111111111011
.	.
.	.
-32767	1000000000000001

POSITIVOS — BINARIO NORMAL

NEGATIVOS — COMPLEMENTO A DOS

Tabla 1.2. Representación de números enteros con 16 bits.

El complemento de un número, en éste formato, es igual que cambiar el signo del mismo. Por otra parte, el complemento del complemento da como resultado el mismo número. $N_2^C(N_2^C(X)) = X$

Ejemplo: Determinar el valor de los siguientes números dados en representación con signo de 16 bits (Formato de 16 bits):

a) 1100101010111000_2; b) $7FA8_{16}$; c) 1111110000011100_2;

d) 176102_8; e) $FA8_{16}$;

Solución (a): El bit 15 del dato vale uno; esto significa que el número es negativo y está dado en complemento a dos. Primero se debe complementar el dato para hallar su verdadero valor en binario y después se transforma a decimal.

$$N_2^C = 0011010101001000_2 = -14640_{10}$$

Solución (b): Se debe transformar hexadecimal a binario y completar con ceros a la izquierda en caso de que el dato no tenga los 16 bits completos. Luego se hace la transformación a decimal.

$$7FA8_{16} = 0111111110101000_2 = +32680_{10}$$

Solución (c): El bit 15 del dato vale uno; esto significa que el número es negativo y está dado en complemento a dos. Primero se debe complementar el dato para hallar su verdadero valor en binario y después se transforma a decimal.

$$N_2^C = 0000001111100100_2 = -996_{10}$$

Solución (d): Se debe transformar octal a binario y completar con ceros a la izquierda en caso de que el dato no tenga los 16 bits completos. Luego se hace la transformación a decimal.

$$176102_8 = 1111110001000010_2$$

$$N_2^C = 0000001110111110_2 = -958_{10}$$

Solución (e): Se debe transformar hexadecimal a binario y completar con ceros a la izquierda en caso de que el dato no tenga los 16 bits completos. Luego se hace la transformación a decimal.

$$FA8_{16} = 111110101000_2 = 0000111110101000_2 = +4008_{10}$$

OPERACIONES ARITMÉTICAS EN COMPLEMENTO A DOS.

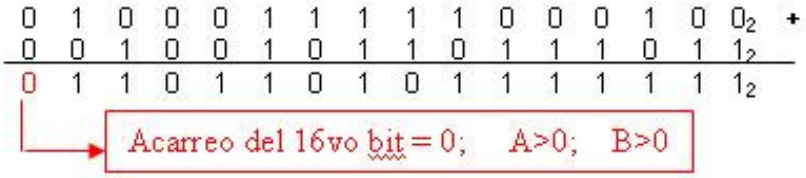
La suma y resta son las operaciones básicas realizadas por los microprocesadores, cualquiera otra operación, es consecuencia **recursiva** de éstas. A continuación se describen estas dos operaciones aritméticas, realizadas con números binarios en complemento a dos utilizando formato de signo y magnitud de 16 bits.

SUMA EN COMPLEMENTO A DOS.

Son cuatro casos que se presentan al sumar dos datos en formato con signo de complemento a dos:

I) SUMA DE DOS NÚMEROS POSITIVOS. El resultado debe ser positivo, y el bit más significativo de la suma, siempre dará cero.

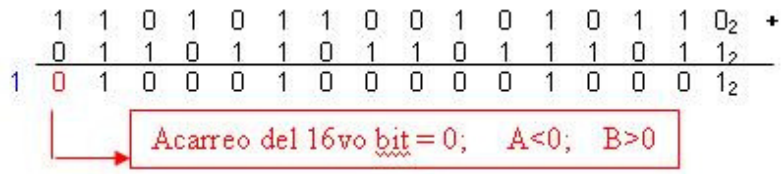
Ejemplo: A = 100011111000100_2; B = 10010110111011_2.



Antes de realizar la suma binaria se debe tener la precaución de sumar en decimal los números. De esta manera se puede chequear el resultado de la suma para tener la certeza de que no exceda el valor $+32767_{10}$ y por lo tanto no sobrepasar el formato de 16 bits (Esto se conoce como OVERFLOW). También el 16vo bit en uno señala el sobreflujo de la operación.

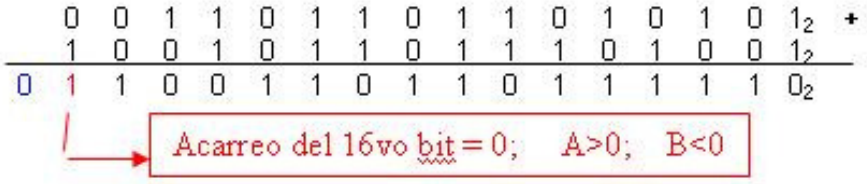
II) SUMA DE UNO NEGATIVO Y OTRO POSITIVO. El resultado debe poseer el signo del que tenga mayor valor absoluto. En este caso el resultado es positivo y el 16vo bit vale cero.

Ejemplo: A = 1101011001010110_2; B = 110110110111011_2

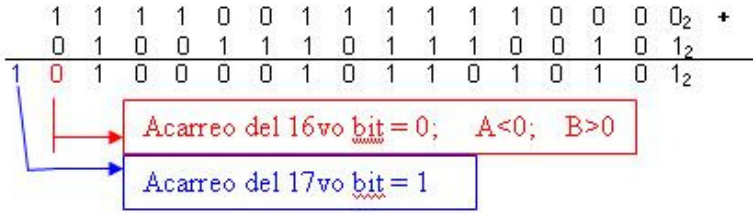


III) SUMA DE UNO POSITIVO Y OTRO NEGATIVO. El resultado debe poseer el signo del que tenga mayor valor absoluto. En este caso el resultado es negativo y el 16vo bit vale cero; del mismo modo no se debe tomar en cuenta el acarreo del 17vo bit.

Ejemplo: A = 11011011010101_2; B = 1001011011101001_2



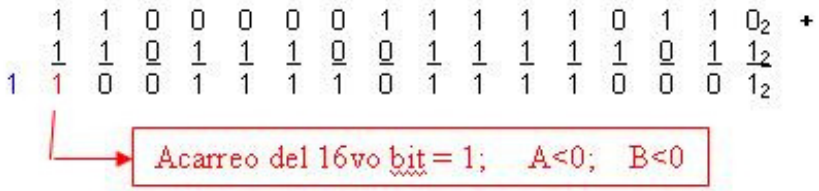
A = 1111001111110000_2; B = 100111011100101_2



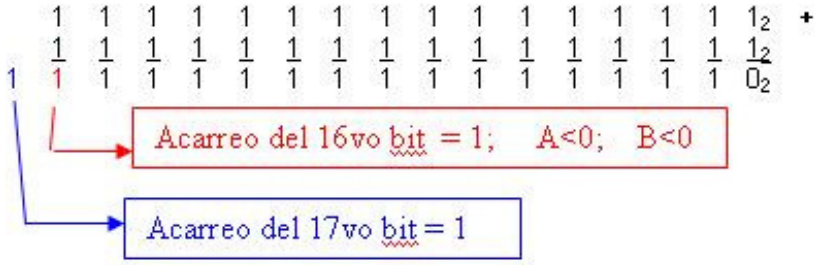
Con dos números de distintos signos se dan los casos de acarreo en el 17vo bit. Si éste acarreo es cero significa que el resultado es negativo y se debe complementar para hallar su verdadero valor de la otra forma, si el acarreo es uno, entonces el signo del resultado es mayor o igual a cero y se encuentra en verdadero valor.

IV) SUMA DE DOS NÚMEROS NEGATIVOS. El resultado debe ser negativo, por lo tanto el bit más significativo de la suma siempre dará uno.

A = 1100000111110110_2; B = 1101110011111011_2



A = 1111111111111111_2; B = 1111111111111111_2



Antes de realizar la suma binaria se debe tener la precaución de sumar en decimal los números. De esta manera se puede chequear el resultado de la suma para tener la certeza de que no exceda el valor -32767_{10} y por lo tanto no sobrepasar el formato de 16 bits (Esto se conoce como OVERFLOW). También el 16vo y/o 17vo bits en cero señalan el sobreflujo de la operación.

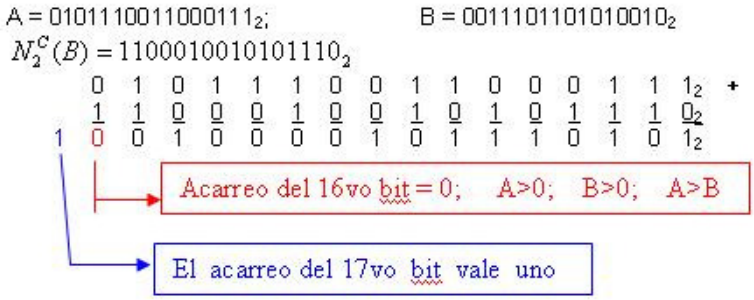
RESTA EN COMPLEMENTO A DOS.

La resta en complemento a dos resuelve el problema de esta operación con los signos. Por ejemplo, el sustraendo negativo y minuendo positivo produce un resultado positivo; la resta de dos números A y B negativos puede dar resultados positivos o negativos. Para realizarla se procede con la fórmula definida de la siguiente forma:

$A - B = A + N_2^{C-1}(B) + 1 = A + N_2^C(B)$ **(Ec.1.5)**; La diferencia de dos números, **A menos B** es equivalente a la suma de **A** más el complemento a dos de **B**.

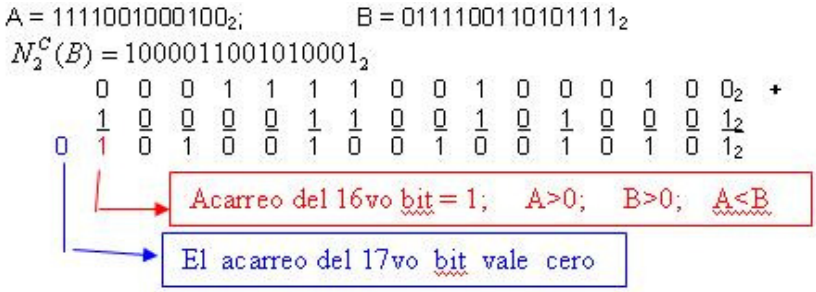
I) Resta de dos números positivos. El resultado puede presentar varias formas que se determinan con los siguientes casos:

(A mayor o igual que B):



De está manera, el resultado queda en forma binaria normal y es igual a valor del 17vo bit no se toma en cuenta para el resultado. En decimal A=23751_{10} y B=15186_{10}; entonces A-B=8565_{10} = 0010000101110101_2

(A menor que B):



De está manera, el resultado es negativo y queda en forma de complemento a dos, el acarreo del 17vo bit no se toma en cuenta. Sin embargo, para saber el verdadero valor, el resultado se debe complementar a dos. Este es un número binario negativo de 16 bits, lo cual tiene un valor de: $N_2^C(N_2^C(B)) = 0101101101101011_2$. En decimal la operación se efectúa: A = 7748_{10} y B = 31151_{10} entonces el resultado es A-B = -23403_{10}.

II) RESTA DE DOS NÚMEROS NEGATIVOS Y DE DISTINTO SIGNO. El resultado puede presentar varias formas que se determinan aplicando los mismos casos de la suma en formato de 16 bits.

Tabla 1. 3. Resumen de las operaciones suma y resta binaria con los datos A y B, utilizando el formato de 16 bits.

Operación	*Acarreo 17vo bit*	*Acarreo 16vo bit*	*Resultado*	*Observaciones*
A+B A>0; B>0	0	0	Positivo en binario normal	Chequear para no exceder el formato de 16 bits.
A+B A>0; B<0 **(**)**	0	1	Negativo en complemento a dos	Complementar los 16 bits para obtener el verdadero valor.
A+B A<0; B>0 **(**)**	1	0	Positivo en binario normal	El 17vo bit no se toma en cuenta para el resultado.
A+B A<0; B<0	1	1	Negativo en complemento a	Complementar los 16 bits para obtener el verdadero

			dos	valor, Chequear para no exceder el formato de 16 bits y el 17vo bit no se toma en cuenta.
A-B A>0; B>0 A>=B	1	0	Positivo en binario normal	El 17vo bit no se toma en cuenta para el resultado.
A-B A>0; B>0 A<B	0	1	Negativo en complemento a dos	Complementar los 16 bits para obtener el verdadero valor.
A-B A>0; B<0	0	0	Positivo en binario normal	Chequear para no exceder el formato de 16 bits.
A-B A<0; B>0	1	1	Negativo en complemento a dos	Complementar los 16 bits para obtener el verdadero valor, Chequear para no exceder el formato de 16 bits y el 17vo bit no se toma en cuenta.
A-B A<0; B<0 **(**)**	0	1	Negativo en complemento a dos o positivo normal	Complementar los 16 bits para obtener el verdadero valor o dejarlo igual. Todo depende de la magnitud de A y B.
(**) Se producen resultados negativos o positivos dependiendo del mayor entre A y B.				

REPRESENTACIÓN NUMÉRICA EN COMA FIJA Y COMA FLOTANTE.

Estas representaciones son utilizadas por las computadoras para procesar cálculos numéricos con formatos grandes. Consiste en una cadena de bits que guardan relación con la notación científica, y pueden representar números enteros y números reales tanto negativos como positivos. Los formatos más conocidos son la coma fija y la coma flotante, también denominados punto fijo y punto flotante respectivamente. Antes de comenzar a describir estos formatos se debe entender el funcionamiento de un caso especial de complemento a dos el cual se denomina representación con exceso o sesgada.

REPRESENTACIÓN CON EXCESO O SESGADA.

Son representaciones para números con signo que eliminan el centrado de la representación básica en complemento a dos. Por ejemplo para indicar números decimales desde un valor numérico $\mathbf{-P_{10}}$ hasta $\mathbf{+P_{10}}$ es necesario desplazar el equivalente binario $(\mathbf{-P_{10}})_2$ sumando $\mathbf{P_2}$ unidades positivas. Esta cantidad se conoce como exceso o sesgo. Las representaciones con exceso se utilizan, con frecuencia, para representar los exponentes de los números con coma flotante. En la tabla 1.4 se pueden observar las representaciones desde $\mathbf{-8_{10}}$ hasta $+\mathbf{8_{10}}$ en complemento a dos y en código con exceso donde $\mathbf{P_2} = \mathbf{1000_2}$. En complemento a dos $\mathbf{-8_{10}}$ es igual a $\mathbf{1000_2}$. Sin embargo, la representación del mismo número negativo en código desplazado con exceso **8** es de $\mathbf{0000_2}$; es de hacer notar que solamente ocurre un cambio en el bit más significativo (MSB: Most Significative Bit) del código con exceso. Por lo tanto, la representación de cualquier código con exceso **-P,** para indicar números negativos, se forma sumando el valor de **P** a cada palabra o número del código.

COMPARACIÓN DE CÓDIGOS EN COMPLEMENTO A DOS Y EXCESO 8.

DECIMAL	*COMPLEMENTO A DOS*	*EXCESO 8*
+7	**0111**	**1111**
+6	**0110**	**1110**
+5	**0101**	**1101**
+4	**0100**	**1100**
+3	**0011**	**1011**

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

REPRESENTACIÓN NUMÉRICA EN COMA FIJA.

Los números fraccionarios y con signo se pueden representar mediante la coma fija; ejemplo de esto se puede apreciar en la tabla 1.2 y la figura 1.3(a) donde se tiene la representación de números enteros con signo en formato de 16 bits. No obstante, existe otra representación para coma fija, la cual consiste en fijar la posición de la coma después del bit de signo; ver figura 1.3(b) respectivamente. Los restantes bits deben indicar la magnitud fraccionaria.

Figura 1.3 (a). Representación entera de coma fija.

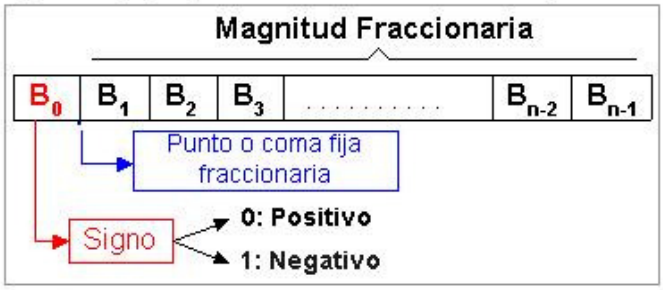


Figura 1.3 (b). Representación fraccionaria de coma fija.

REPRESENTACIÓN NUMÉRICA EN COMA FLOTANTE.

Los números representados en coma flotante tienen la misma forma que la notación científica. La representación tiene la siguiente forma

$N = Mxb^E$ **(Ec.1.6);** donde ***M*** es la mantisa o significado y se representa en coma fija, este valor indica la cantidad de dígitos significativos que tiene el número ***N*** de coma flotante. El valor ***E*** es el exponente o característica, también de coma fija; está dado en formato de complemento a dos con exceso y ***b*** es la base del sistema. En forma general, de la **Ec.1.1** se puede obtener la representación con signo de coma fija y está dada por: $N = \pm(a_{n-1}a_{n-2}.....a_0, a_{-1}a_{-2}.....a_{-k})_b$, ahora sustituyendo por el formato de coma fija, dada en la figura 1.3(b), se obtiene la forma de coma flotante

$N = \pm(0, a_{n-1}a_{n-2}.....a_{-k})xb^n$

$M = \pm(0, a_{n-1}a_{n-2}a_{n-3}.....a_{-k})$ **(Ec.1.7).** La fórmula general queda del siguiente modo;

$N = (-1)^{bs}\, x(0, a_{n-1}a_{n-2}.....a_{-k})_b\, xb^{E'+2^{(e-1)}}$ **(Ec.1.8)**

donde ***bs*** es el bit de signo, ***e*** es el número de bits del exponente con $E = E' + 2^{(e+1)}$; esto es equivalente a escribir ***E*** con formato de exceso en base dos de la siguiente manera; $E' = (c_{e-1}c_{e-2}.....c_0)_2$, por lo tanto, $E = (c_{e-1}c_{e-2}.....c_0)_2 + 2^{e-1}$

Existen varias formas de representar los formatos de coma flotante; sin embargo, los que más se utilizan son los siguientes:

- $N = Mxb^E$
- $N = (M \div b)xb^{E+1}$
- $N = (Mxb)xb^{E-1}$

En las figuras 1.4(a) y 1.4(b) se definen los formatos en coma flotante para datos numéricos reales cortos y largos utilizados en los computadores.

Figura 1.4(a). Declaración de datos cortos en coma flotante.

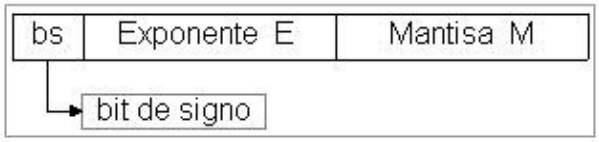
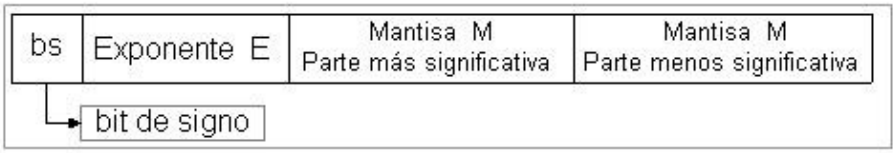


Figura 1.4(b). Declaración de datos largos en coma flotante.



La tabla 1.5 muestra un resumen de los formatos de precisión sencilla y doble (corto y largo) respectivamente; usados en los sistemas de computación.

FORMATO	*TOTAL DE BITS*	*BITS DE LA MANTISA*	*BITS DEL EXPONENTE*	*EXCESO DEL EXPONENTE*
Estándar IEEE 754-1985				
Precisión sencilla	32	24	8	128
Doble Precisión	64	53	11	1024
IBM 360				
Precisión sencilla	32	24	7	64
Doble precisión	64	56	7	64
DEC VAX 11/780				
Formato F	32	24	8	128
Formato D	64	56	8	128
Formato G	64	53	11	1024

Tabla 1.5. Formatos comunes para números representados en coma flotante.

Ejemplo: Escribir en formato de coma flotante los números: a) $11011101{,}1101_2$
b) $0{,}0000111010101_2$

Solución (a): Se debe llevar a la forma $N = Mxb^E$; primero hay que hallar la mantisa con la Ec.1.7 y luego el exponente **E** con exceso;
M=+$(0{,}1101110111010)_2$
E=+8_{10}=+$(1000)_2$; si el bit de signo es positivo entonces E'=0100_2. En este caso hay que sumarle al exponente un exceso de 16_{10}; E= 01000_2+10000_2 = 11000_2

La solución final queda de la siguiente forma:

bs	Exponente E	Mantisa M
0	11000	1101110111010

Solución (b): Se debe llevar a la forma $N = Mxb^E$; primero hay que hallar la mantisa con la Ec.1.7 y luego el exponente **E** con exceso;
M=+$(0,111010101)_2$
E=-4_{10}=-$(100)_2$; si el bit de signo es negativo entonces E'=1100_2. En este caso hay que sumarle al exponente un exceso de 8_{10}; E= 1100_2+1000_2 = 0100_2
La solución final queda de la siguiente forma:

bs	Exponente E	Mantisa M
0	0100	111010101

CÓDIGOS DE NUMERACIÓN, ALFANUMÉRICOS Y DE ERRORES.

Los códigos en los sistemas digitales se clasifican en tres tipos: códigos numéricos, códigos alfanuméricos y códigos detectores y correctores de errores. El objetivo de los códigos es simplificar la comunicación entre los distintos circuitos digitales, normalizar el funcionamiento de los mismos y detectar posibles fallas de datos para su posterior corrección.

1.5.1 Códigos numéricos.

Los más utilizados, en circuitos digitales combinacionales son el código BCD, Exceso 3, Aiken o 2421, 5421, Biquinario, Dos de Cinco. Existen otros códigos de tipo secuencial cíclicos, dos de ellos es son código Jhonson y el código Gray. En la tabla 1.6 se describen algunos de ellos con sus respectivos equivalentes decimales.

Decimal	*BCD*	*Exceso 3*	*2421*	*5421*	*Biquinario*	*Dos de cinco*	*Gray*
0	0000	0011	0000	0000	0100001	00011	0000
1	0001	0100	0001	0001	0100010	00101	0001
2	0010	0101	0010	0010	0100100	01001	0011
3	0011	0110	0011	0011	0101000	10001	0010
4	0100	0111	0100	0100	0110000	00110	0110
5	0101	1000	1011	1000	1000001	01010	0111
6	0110	1001	1100	1001	1000010	10010	0101
7	0111	1010	1101	1010	1000100	01100	0100
8	1000	1011	1110	1011	1001000	10100	1100
9	1001	1100	1111	1100	1010000	11000	1101
10	0001 0000	0100 0011	0001 0000	0001 0000	0100010 0100001	00101 00011	1111
11	0001 0001	0100 0100	0001 0001	0001 0001	0100010 0100010	00101 00101	1110
12	0001 0010	0100 0101	0001 0010	0001 0010	0100010 0100100	00101 01001	1010
13	0001 0011	0100 0110	0001 0011	0001 0011	0100010 0101000	00101 10001	1011
14	0001 0100	0100 0111	0001 0100	0001 0100	0100010 0110000	00101 00110	1001
15	0001 0101	0100 1000	0001 1011	0001 1000	0100010 1000001	00101 01010	1000

Tabla 1.6. Equivalencia desde cero hasta quince de algunos códigos numéricos más utilizados.

Código BCD.

(Binario Codificado en Decimal): La conversión con el sistema decimal se realiza directamente, en grupos de cuatro bits por cada dígito decimal con ponderación 8421. Este código tiene aplicación en visualizadores (displays) hechos con diodos led o LCD, los cuales poseen previamente convertidores que transforman el grupo de cuatro bits BCD en otro especial, llamado 7 segmentos.

Por ejemplo, para transformar el número decimal 78905_{10} en código BCD se toman los equivalentes en grupos de cuatro bits cada uno; ver tabla 1.6:

7	8	9	0	5_{10}
0111	1000	1001	0000	0101_{BCD}

Resp: 78905_{10} = 0111 1000 1001 0000 0101_{BCD}

Para realizar la equivalencia del BCD con el sistema binario se debe tomar la precaución de realizar primero la transformación decimal y posteriormente la conversión al BCD.

Ejemplo: Transformar en BCD los siguientes números:
a) 1011101111111_2; b) $5F3C,B_{16}$

Solución (a): $1011101111111_2 = 6015_{10} = 0110\ 0000\ 0001\ 0101_{BCD}$

Solución (b): $5F3C,B_{16} = 24380,6875_{10} = 0010\ 0100\ 0011\ 1000\ 0000,\ 0110\ 1000\ 0111\ 0101_{BCD}$

SUMA EN BCD.

La suma en BCD puede dar como resultado un número no perteneciente al código. Por ejemplo, al sumar los números BCD 1000 + 0001 el resultado es 1001, este número también pertenece al código; sin embargo, cuando se suman 0111 +1000 el resultado es 1111, este número no pertenece al código BCD y su valor equivalente es quince unidades.

Cuando suceden estos casos es necesario sumar un factor de corrección que depende del rango donde se encuentre el resultado de la suma. La tabla 1.7 muestra los valores del factor de corrección con su respectivo rango. Para el rango binario desde diez (1010_2) hasta diecinueve (10011_2) el factor de corrección es seis 0110_2; este factor se duplica en forma proporcional del mismo modo que aumenta la decena en el resultado. Por lo que se debe aplicar la fórmula $Fc = n_2.(0110)_2$ donde ***n*** es igual al valor binario de la decena del resultado. El factor de corrección se debe aplicar siempre y cuando el resultado de la suma sea mayor o igual a diez. Del mismo modo, la suma debe realizarse en binario.

Factor de corrección Binario $(Fc)_2$	***Valor decimal de rango***	***Valor decimal de la decena (n)***	***Factor de corrección Decimal $(Fc)_{10}$***
0110	(10 ~ 19)	1	6
1100	(20 ~ 29)	2	12
10010	(30 ~ 39)	3	18
11000	(40 ~ 49)	4	24
.	.	.	.
.	.	.	.
.	.	.	.
$Fc = n_2x(0110)_2$	(***n***0 ~ ***n***9)	***n***	***Fc = nx6***

Tabla 1.7. Factores de corrección para la suma BCD.

Ejemplo. Dado los números **p, q, r** en código BCD, sumar: **a) p + q + r**, **b) q + r**, **c) p + q** y obtener el resultado también en BCD.
p = $1000\ 0110\ 0010\ 0000\ 1001_{BCD}$; **q** = $0100\ 1001\ 1001\ 0011\ 0111_{BCD}$
r = $0111\ 1001\ 1000\ 0110\ 0010\ 0011\ 1001_{BCD}$

Solución (a): Los resultados que superen el **1001** hay que sumarle el factor de corrección según la tabla 1.7 y llevar el acarreo correspondiente.

Acarreo	**1**	**10**	**10**	**1**		**10**		
P =			1000	0110	0010	0000	1001	+
Q =			0100	1001	1001	0011	0111	
R =	0111	1001	1000	0110	0010	0011	1001	
	1000	1011	10110	10110	1101	1000	11001	+
Fc =	0000	0110	1100	1100	0110	0000	1100	
Resultado =	1000	**1**0001	**10**0010	**10**0010	**1**0011	1000	**10**0101	

Respuesta (a): p+q+r = $1000\ 0001\ 0010\ 0010\ 0011\ 1000\ 0101_{BCD} = 8122385_{10}$

Solución (b): Los resultados que superen el **1001** hay que sumarle el factor de corrección según la tabla 1.7 y llevar el acarreo correspondiente.

1	1	1	1		1		
		0100	1001	1001	0011	0111	+
0111	1001	1000	0110	0010	0011	1001	
1000	1010	1101	10000	1011	0111	10000	+
0000	0110	0110	0110	0110	0000	0110	
1000	10000	10011	10110	10001	0111	10110	

Respuesta (b): $q+r = 1000\ 0000\ 0011\ 0110\ 0001\ 0111\ 0110_{BCD} = 8036176_{10}$

Solución (c): Los resultados que superen el **1001** hay que sumarle el factor de corrección según la tabla 1.7 y llevar el acarreo correspondiente.

1	1	1			1		
	1000	0110	0010	0000	1001	+	
	0100	1001	1001	0011	0111		
	1101	10000	1011	0100	10000	+	
	0110	0110	0110	0000	0110		
0001	10011	10110	10001	0100	10110		

Respuesta (c): $p+q = 0001\ 0011\ 0110\ 0001\ 0100\ 0110_{BCD} = 136146_{10}$

CÓDIGO EXCESO 3.

Es un código igual al BCD, sin embargo se deben añadir tres unidades a este para transformarlo en exceso 3.

CÓDIGO AIKEN O 2421.

La ponderación de este código es diferente al BCD, para hallar su peso se debe tomar también grupos de cuatro bits, considerando los valores 2421, por dígito decimal.

Este código se conoce como autocomplementado a uno porque sus diez valores, en la tabla 1.6; se pueden formar, complementando, a partir de los primeros cinco dígitos.

CÓDIGO 5421.

La ponderación de este código es diferente al BCD, para hallar su peso se debe tomar también grupos de cuatro bits, considerando los valores 5421, por dígito decimal. Este código se forma repitiendo los cinco primeros valores de la tabla 1.6, de modo tal, que cambia solo el bit más significativo de cero a uno.

CÓDIGO BIQUINARIO.

Necesita siete bits para formarse; siempre hay dos bits en nivel alto (uno) y los restantes cinco deben estar en nivel bajo (cero). El primer bit del código, en uno, se usa para indicar si el dígito se encuentra comprendido entre 5 y 9; el segundo bit del código, en uno, señala que se encuentra en el rango de 0 a 4. La desventaja de este código es la cantidad de bits que se deben utilizar para transmitir información, siete por cada dígito. Sin embargo, tiene la ventaja de poder realizar fáciles algoritmos para el chequeo de errores de transmisión; solamente se debe detectar que hayan dos bits, en nivel uno, por cada dato. Uno de estos se debe encontrar entre los primeros dos bits y el otro en los cinco restantes que forman el dígito.

CÓDIGO DOS DE CINCO.

Este código es similar al Biquinario, pero requiere de cinco bits para el correcto funcionamiento. Dos bits deben estar en nivel alto y los otros tres en cero.

CÓDIGO GRAY.

Este código cíclico no posee una relación directa con la ponderación de los dígitos del sistema decimal. Se forma cambiando el bit menos significativo de manera continua y consecutiva. Solamente cambia un bit, y éste, debe ser el menos significativo; de manera que no se repita con alguna combinación anterior. También se puede formar obteniendo las primeras ocho combinaciones con tres bits y luego, desde la 8va combinación hay que repetir simétricamente los valores, cambiando solamente el bit más significativo de cero a uno. Por ejemplo, la 8va posición es **0100** y a continuación viene la 9na **1100**; del mismo modo, la 7ma **0101** es simétrica con la 11va **1101**. El código Gray tiene aplicaciones en contactos de escobillas de motores, sistemas donde solo se necesite cambiar un bit de estado cíclicamente.

La ventaja del código Gray radica en que la probabilidad de ocurrir menos errores y problemas de transición aumenta a medida que cambian mas bits de estado simultáneamente. El cambio consecutivo del código BCD desde **0111** a **1000** puede producir transiciones intermedias que originan el **1111** antes de estabilizarse en **1000**. Sin embargo, el código Gray pasará desde **0111** a **0101** cambiando solamente un bit y por lo tanto, con menos posibilidad de cometer errores.

CÓDIGOS ALFANUMÉRICOS.

Estos códigos son interpretados por el computador como caracteres e indistintamente pueden representar símbolos numéricos, símbolos de control y letras. Las computadoras se comunican mediante estos códigos y los más utilizados son el código ASCII y el UNICODE.

CÓDIGO ASCII.

ASCII: American Standard Code Interchange Information. Cada caracter alfanumérico esta formado por una cadena de siete bits. Este código representa 128 símbolos diferentes entre dígitos, letras e instrucciones de control del computador. La tabla 1.xx muestra los símbolos con su respectivo valor hexadecimal. Por ejemplo, para codificar la palabra UNEXPO se procede de la siguiente forma:
1010101 1001110 1000101 1011000 1010000 1001111

U	N	E	X	P	O
55H	4EH	45H	58H	50H	4FH

Tabla 1.8. Código ASCII.

		$B_6B_5B_4$							
	BIN	000	001	010	011	100	101	110	111
$B_3B_2B_1B_0$	HEX	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

UNICODE.

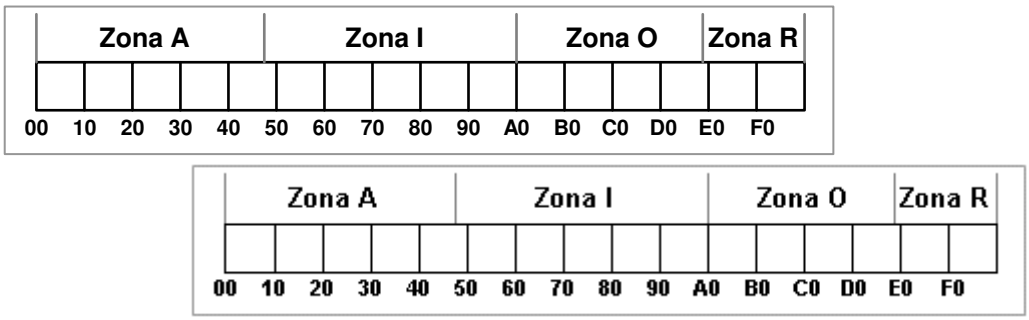
Es un código universal actualizado de propósito general, sirve para representar todos los símbolos utilizados en los alfabetos internacionales. Es una nueva norma de códigos alfanuméricos de 16 bits. Los símbolos se representan con cuatro dígitos hexadecimales como se muestra en la tabla 1.9. El código ASCII es un subconjunto de éste y está representado desde 0000_{16} hasta $007F_{16}$. En la figura 1.4 se observa la distribución del código en cuatro zonas que van desde 0000_{16} hasta $FFFF_{16}$. La zona A comprende los códigos para alfabetos, sílabas, y símbolos. En la zona I están los códigos ideográficos como lo son los alfabetos Chinos y Japoneses. La zona O no es utilizada actualmente, sin embargo, está reservada para futuros ideogramas.

La zona R es de uso restringido. Se subdivide en Área de uso privado, Área de compatibilidad y Códigos especiales. FFFE y FFFF no son códigos de carácter y se excluyen específicamente del UNICODE. El Área de uso privado está a disposición de quienes necesiten caracteres especiales para sus programas de aplicación; por ejemplo, los iconos empleados en los menús podrían especificarse por medio de códigos de carácter en esta área. La zona de compatibilidad tiene caracteres correlacionados con otras áreas del espacio global de código. La transmisión serial de un carácter UNICODE se realiza con dos bytes **(byte 0 y byte 1).** Primero se envía la palabra de control FFFE o FEFF indicando cual de los dos bytes es el más significativo; Por ejemplo, al enviar los símbolos FFFE, 4100, 4E00, 4700, 4500, 4C00 indica que se debe cambiar el orden de los bytes,

esto es: 0041, 004E, 0047, 0045, 004C que se codifica como 'ANGEL' en la tabla 1.9. Sin embargo, en caso de haber enviado la palabra de control FEFF indicaba que el orden de los bytes era el mismo. Lo que no correspondía con los códigos ASCII del UNICODE.

Estos ordenamientos en los bytes del UNICODE guardan relación con los formatos de datos para comunicación de computadoras Litle-Endian o Big-Endian.

Figura 1.4. Distribución del código UNICODE.



Ejemplo 1.24. Indicar si es posible decodificar las siguientes palabras dadas en UNICODE.

a) FFFE, 4300, A200, 6400, 6900, 6700, 6F00

b) FEFF, 0055, 004E, 0045, 0058, 0050, 004F

Solución (a): El orden de los bytes debe ser invertido; 0043, 00A2, 0064, 0069, 0067, 006F que corresponde con la palabra '**Código**'.

Solución (b): El orden de los bytes es el correcto 0055, 004E, 0045, 0058, 0050, 004F que corresponde con la palabra '**UNEXPO**'.

Tabla 1.9. Primeros 256 Símbolos UNICODE.

HEX	**000**	**001**	**002**	**003**	**004**	**005**	**006**	**007**	**008**	**009**	**00A**	**00B**	**00C**	**00D**	**00E**	**00F**
0	CTL	CTL	SP	0	@	P	`	P	Ç	É	á	_	+	ð	Ó	-
1	CTL	CTL	!	1	A	Q	a	Q	ü	æ	í	_	-	Ð	ß	±
2	CTL	CTL	"	2	B	R	b	R	é	Æ	ó	_	-	Ê	Ô	_
3	CTL	CTL	#	3	C	S	c	S	â	ô	ú	¦	+	Ë	Ò	¾
4	CTL	CTL	$	4	D	T	d	T	ä	ö	ñ	¦	-	È	õ	¶
5	CTL	CTL	%	5	E	U	e	U	à	ò	Ñ	Á	+	I	Õ	§
6	CTL	CTL	&	6	F	V	f	V	å	û	ª	Â	Ã	Í	µ	÷
7	CTL	CTL	'	7	G	W	g	W	ç	ù	º	À	Ã	Î	Þ	¸
8	CTL	CTL	(	8	H	X	h	X	ê	ÿ	¿	©	+	Ï	Þ	°
9	CTL	CTL	)	9	I	Y	i	Y	ë	Ö	®	¦	+	+	Ú	¨
A	CTL	CTL	*	:	J	Z	j	Z	è	Ü	¬	¦	-	+	Û	·
B	CTL	CTL	+	;	K	[	k	{	ï	ø	½	+	-	_	Ù	¹
C	CTL	CTL	,	<	L	\	l	|	î	£	¼	+	¦	_	Ý	³
D	CTL	CTL	-	=	M	]	m	}	ì	Ø	¡	¢	-	¦	Ý	²
E	CTL	CTL	.	>	N	^	n	~	Ä	×	«	¥	+	Ì	¯	_
F	CTL	CTL	/	?	O	_	o	CTL	Å	ƒ	»	+	CTL	_	´	SP

CÓDIGOS DETECTORES Y CORRECTORES DE ERRORES.

La transmisión y recepción de datos binarios, desde un dispositivo a otro, están propensas a errores, campos magnéticos, interferencias y ruidos eléctricos pueden ocasionar este problema. El costo agregado que ocasiona añadir circuitos detectores y correctores de error se ve compensado con el avance de la tecnología en el área de las telecomunicaciones. Los sistemas de comunicación digital son la tecnología de punta en el ámbito mundial y, específicamente, las redes de computadoras; ejemplo de esto son las redes locales, Internet, etc.

Los sistemas deben detectar y/o corregir errores de comunicación en el menor tiempo posible de manera que puedan mantener el intercambio de información digital en línea y en tiempo real. La

tarea no parece sencilla; sin embargo, los diseñadores de sistemas digitales deben considerar el costo de estos circuitos adicionales, a la hora de implementar el circuito.
De hecho, es necesario agregar más bits al dato que se desea transmitir con la finalidad de chequear, en el receptor, los posibles errores durante el proceso de comunicación.

El método para realizar esto; va desde solicitar que reenvíen el dato, el bloque o hasta la información completa. También hay métodos más seguros que implementan sistemas redundantes de tres o más circuitos de comunicación idénticos que operan en paralelo y por lo tanto disminuyen considerablemente el índice de errores.

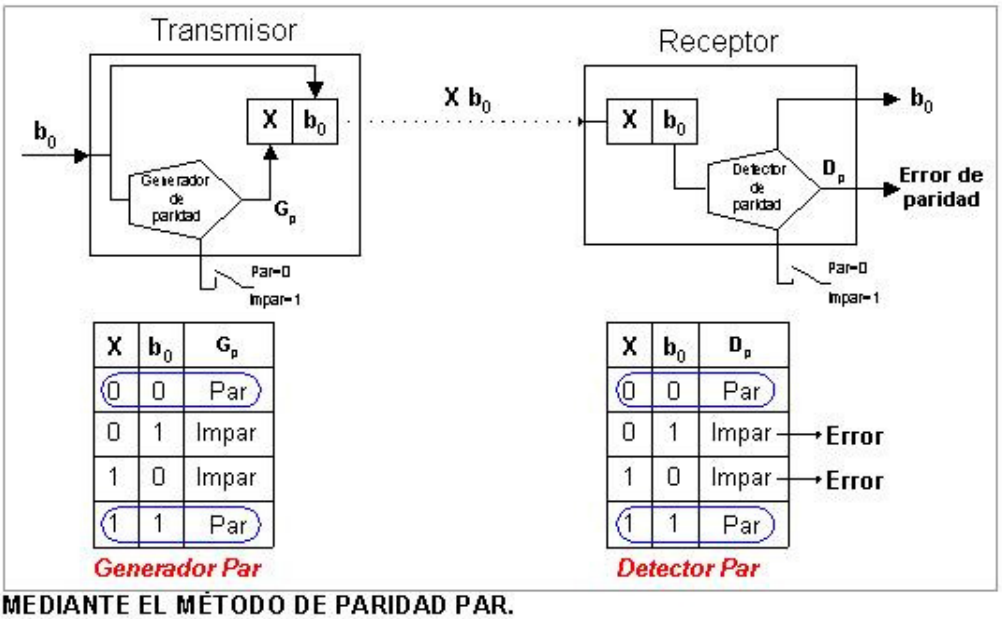
En esta sección se analizaran los métodos de detección de errores por paridad y detección y/o corrección mediante el código Hamming.

DISTANCIA Y PESO DE LOS DATOS BINARIOS.

Para chequear un bit de dato, en el receptor, es necesario agregar al sistema de comunicación, por lo menos, otro bit. De esta manera, el código queda formado por dos bits; uno para dato y el otro para chequeo y control. De esta misma forma, se debe establecer un patrón de comunicación (protocolo de comunicación). Por ejemplo, establecer que el bit de control se genere de la siguiente forma: sea el más significativo y además, la suma de los dos bits sea siempre par.
Esto se ilustra en la figura 1.5; aquí se puede ver los cuatro cambios posibles de los bits **X** y $\mathbf{b_0}$. El bit $\mathbf{b_0}$ tiene dos valores posibles 0 y 1; para enviar un cero se debe agregar en el generador de paridad $\mathbf{G_P}$ otro cero para mantener la paridad par. Si, por el contrario, el $\mathbf{b_0}$ es uno entonces hay que generar en $\mathbf{G_P}$ un uno para mantener el protocolo de paridad par sin errores.
El circuito receptor de información detecta la paridad de los dos bits **(X $\mathbf{b_0}$)**, chequea las combinaciones posibles; activando la señal de error cuando es recibida la combinación **(0 1)** o **(1 0)**. Este ejemplo se puede extender para datos que tengan **n** bits de información ya que, basta un bit adicional, para generar y chequear errores de paridad. Para entender mejor esta última afirmación, se definen a continuación, los términos distancia y peso en los datos binarios.

La distancia máxima entre dos datos binarios, de igual longitud, es equivalente al número de bits que cambian de estado. Por ejemplo, la distancia entre los datos D_1=10010111 y D'_1=10110001 es tres. La distancia se puede definir también como el número de bits diferentes entre dos palabras.

Figura 1.6. Sistema de transmisión y recepción de un bit con generación y detección de error



MEDIANTE EL MÉTODO DE PARIDAD PAR.

Otro ejemplo para tomar en cuenta es el caso donde la palabra transmitida y recibida difieren en dos bits; esto es, transmitida A=1100101 y recibida A'=1101100. La distancia es dos; sin embargo, aunque la palabra cambie, la paridad se mantiene y por lo tanto no habrá señalización de error. Al comparar, este caso, con el cambio entre D_1 y D'_1 se observa que si hay señalización de error porque la paridad no se mantiene.

El número de bits en nivel uno de (D_1 - D'_1) no son iguales. Por el contrario, en el caso (A - A') se observa el mismo número de bits en uno. Este número de bits en nivel alto, de un dato binario, es lo que se conoce como el peso de la palabra o peso del dato binario. Por ejemplo, D_1 tiene un peso de 5 y D'_1 tiene un peso de 4; del mismo modo, A y A' pesan respectivamente 4.

DETECCIÓN DE ERROR USANDO EL MÉTODO DE PARIDAD.

El sistema de chequeo de error por paridad es muy utilizado en las comunicaciones seriales de datos. El método consiste en establecer un tipo de paridad (par o impar) en el sistema de comunicación y generar en el transmisor, un bit adicional de modo que el peso del dato corresponda con la paridad (par o impar) establecida. Por lo general, este bit se agrega en la posición más significativa del dato.

Ejemplo: En los datos a, b, y c generar el bit de paridad par e impar en la posición más significativa (MSB).

a) 1010; b) 1110101; c) 00001

Solución par: El bit, hay que generarlo en el MSB de forma que el peso sea par;

a) 01010; b) 11110101; c) 100001

Solución impar: El bit, hay que generarlo en el MSB de forma que el peso sea impar;

a) 11010; b) 01110101; c) 000001

Ejemplo: Un sistema de comunicación ha recibido los siguientes caracteres ASCII: I) 01000001; II) 10111000; III) 11111110; y se desea saber si hay error. El protocolo de paridad es par. Indicar, en caso de ser correcto, el carácter enviado.

Solución (I): El peso de este dato es par (dos), por lo tanto, es correcto y corresponde al carácter ASCII 41H = 'A'.

Solución (II): El peso de este dato es par (cuatro), por lo tanto, es correcto y corresponde al carácter ASCII 38H = '8'.

Solución (III): El peso de este dato es impar (siete), por lo tanto, hay error de transmisión. En estos casos no es posible reconstruir el dato.

DETECCIÓN Y CORRECCIÓN DE ERRORES MEDIANTE EL CÓDIGO HAMMING.

El método de paridad con un solo bit es eficiente en la detección de errores cuando hay confiabilidad en el sistema de comunicación. De hecho, el peso del dato queda determinado con ***m=n+1*** bits, donde ***n*** es el número de bits que contiene la información. Este método solamente puede detectar errores de dos datos que difieran en un bit; osea, tengan distancia uno y que cambie, por error del sistema, solamente un bit. Sin embargo, no los corrige y a lo sumo, puede señalizar error y/o solicitar que vuelvan a enviar el byte, dato, palabra, o bloque de información que presentó el problema de comunicación.

De la misma forma, si hay cambios de distancias pares (2,4, 6,...), el método no detectará error. Sin embargo, en las distancias impares señaliza los errores. Ejemplo de esto se puede ver comparando, en el punto anterior, los casos (D_1 - D'_1) y (A - A').

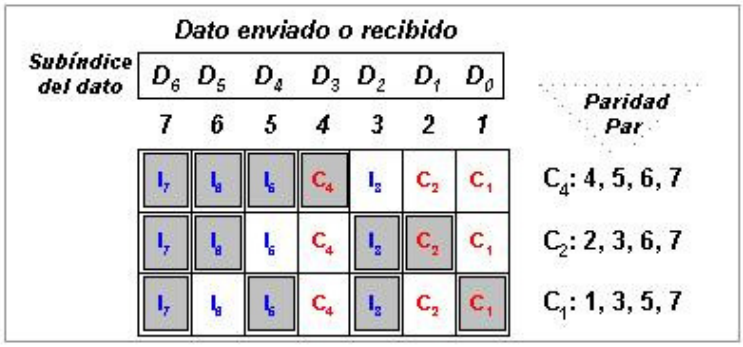
En 1950 R.W. Hamming introdujo un método para detectar y corregir errores de datos en los sistemas de comunicación donde las distancias pueden ser mayores a la unidad. Este código trabaja con una distancia mínima de tres y puede detectar errores con cambios de 1 o 2 bits y corregir, cambios de un solo bit.

Los bits necesarios para el código Hamming se dividen en dos grupos; m bits de información y k bits de chequeo o paridad, por lo que, el tamaño del dato a transmitir debe ser ***n=m+k*** bits. Éste debe cumplir con la siguiente ecuación:

$2^k \geq m + k + 1$ **(Ec.1.9)**.

La paridad del código puede ser par o impar, sin embargo, toda la información relacionada está dada en paridad par. Por lo tanto, los ejemplos se realizaran tomando como referencia codificación Hamming de paridad par con el número de bits ***n*** igual a siete. En la figura 1.7 se observa la distribución de paridades para los bits de chequeo con formato de siete bits de dato. De esta forma, al aplicar la Ec.1.9 se determina que **m=4** y **k=3**, por lo tanto la información que se puede transmitir va desde 0000_2 hasta 1111_2; éstos están distribuidos, en la figura 1.7 como $\mathbf{I_7}$, $\mathbf{I_6}$, $\mathbf{I_5}$, $\mathbf{I_3}$ y deben mezclarse con los de chequeo $\mathbf{C_4}$, $\mathbf{C_2}$, $\mathbf{C_1}$. Estos últimos ocupan las posiciones de la potencia en base 2 indicada por los subíndices dos, uno y cero respectivamente.

Figura 1.7. Formación del código Hamming de 7 bits.



El código se forma entrelazando los bits de información (q3 q2 q1 q0) con los bits de control (h_2 h_1 h_0) de forma que los subíndices de h correspondan con la posición decimal del código formado. Los bits (q_3 q_2 q_1 q_0) de información se hacen corresponder, en la figura 1.7, con los bits (I_7 I_6 I_5 I_3) respectivamente; la finalidad es ubicarlos en la posición decimal del código. Del mismo modo, (h_2 h_1 h_0) es equivalente con las posiciones según en subíndice $h_2=C_2^2=C_4$; $h_1=C_2^1=C_2$; $h_0=C_2^0=C_1$. Finalmente el código de siete bits queda formado de la siguiente manera:

q_3	q_2	q_1	h_2	q_0	h_1	h_0
I_7	I_6	I_5	C_4	I_3	C_2	C_1
D_6	D_5	D_4	D_3	D_2	D_1	D_0

Al enviar el dato de siete bits, este es recibido como un paquete formado por (D_6 D_5 D_4 D_3 D_2 D_1 D_0) donde no se reconoce quien es información y/o quien es control. Sin embargo, con el método se realizan tres grupos de detección y corrección formado por cuatro bits cada uno, los cuales siempre deben tener paridad par. Estos grupos están resaltados de gris en la figura 1.7 y forman tres cuartetos agrupados de la siguiente forma: (I_7 I_5 I_3 C_1); (I_7 I_6 I_3 C_2); (I_7 I_6 I_5 C_4). Ellos sirven tanto para generar, detectar y corregir datos con distancia uno y dos respectivamente.

Por ejemplo, para enviar el dato de información (1100) codificado en Hamming se deben agregar tres bits de control de manera que los cuartetos tengan paridad par:
Primero hay que hacer corresponder los bits de información; (1100)=(I_7 I_6 I_5 I_3), después se organizan los cuartetos de forma que la paridad sea par:

Los bits de control generados son: (C_4 C_2 C_1) = (001); en consecuencia el dato a enviar es (D_6 D_5 D_4 D_3 D_2 D_1 D_0) = (I_7 I_6 I_5 C_4 I_3 C_2 C_1) = (1100001). De la misma forma se procede a obtener la codificación de los bits en código Hamming. En la tabla 1.10 están representados los 4 bits de información y los tres bits de chequeo del código Hamming de 7 bits. También se puede observar que la mínima distancia, entre dos datos consecutivos, es tres.

Decimal	Información	Control	Dato codificado
	$I_7I_6I_5I_3$	$C_4C_2C_1$	$I_7I_6I_5C_4I_3C_2C_1$
0	0000	000	0000000
1	0001	011	0000111
2	0010	101	0011001
3	0011	110	0011110
4	0100	110	0101010
5	0101	101	0101101
6	0110	011	0110011
7	0111	000	0110100
8	1000	111	1001011
9	1001	100	1001100

10	1010	010	1010010
11	1011	001	1010101
12	1100	001	1100001
13	1101	010	1100110
14	1110	100	1111000
15	1111	111	1111111

Tabla 1.10. Código Hamming de 7 bits.

También se pueden corregir errores de datos con distancia uno de la siguiente forma:
Ejemplo 1.27. Se han recibido los datos **a, b, c, d** codificados en Hamming de 7 bits con paridad par, y es necesario detectar y corregir los bits con errores.
a) 1100100; b) 1110101; c) 1010101; d) 1110111
Solución (a): Para mantener la paridad par en el grupo 2,3,6,7 debe cambiarse el bit de la posición 2 (C_2). El dato corresponde a 1101.

I_7	I_6	I_5	C_4	I_3	C_2	C_1
1	1	0	0	1	0	0

I_7	I_6	I_5	C_4
1	1	0	0
I_7	I_6	I_3	C_2
1	1	1	0
I_7	I_5	I_3	C_1
1	0	1	0

Error en C_2; este bit debe ser 1

Solución (b): Para mantener la paridad par en los grupos 2,3,6,7 y 4,5,6,7 se debe cambiar el bit de la posición 6 (I_6) para obtener la paridad correcta. El dato es: 1011.

I_7	I_6	I_5	C_4	I_3	C_2	C_1
1	1	1	0	1	0	1

I_7	I_6	I_5	C_4
1	1	1	0
I_7	I_6	I_3	C_2
1	1	1	0
I_7	I_5	I_3	C_1
1	1	1	1

Error en I_6; este bit debe ser 0

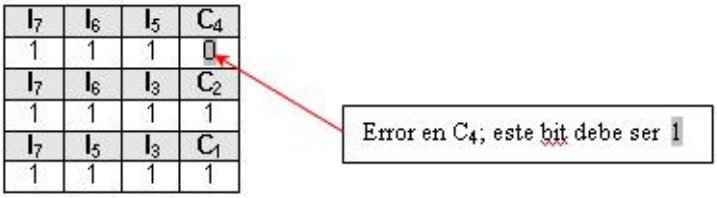
Solución (c): En este caso, no hay error en el dato enviado.

I_7	I_6	I_5	C_4	I_3	C_2	C_1
1	0	1	0	1	0	1

I_7	I_6	I_5	C_4
1	0	1	0
I_7	I_6	I_3	C_2
1	0	1	0
I_7	I_5	I_3	C_1
1	1	1	1

Solución (d): Para mantener la paridad par en los grupos 4,5,6,7 se debe cambiar el bit de la posición 4 (C_4) para obtener la paridad correcta. El dato es: 1111.

I_7	I_6	I_5	C_4	I_3	C_2	C_1
1	1	1	0	1	1	1



Los casos **a** y **d** pueden ser aceptados como errores dobles o simple. Sin embargo, al asumir algún cambio en los bits de chequeo implica descartar errores dobles en los bits de información. Debido a esto, en el ejemplo 1.27(a) pueden ser considerado los cambios de los bits $\mathbf{I_7}$ e $\mathbf{I_5}$. De esta misma forma, en el ejemplo 1.27(d), los cambios pueden ocurrir en los bits $\mathbf{I_7}$ e $\mathbf{I_3}$. Los cambios dobles (distancia dos) no pueden ser corregidos con el código Hamming de 7 bits, sin embargo, para resolver esto es necesario el código Hamming de 8 bits.

Ejercicios propuestos

1. Transformar al sistema binario, octal y hexadecimal los siguientes números decimales:
   - $8879{,}482_{10}$
   - $11011{,}01_{10}$
   - $4456{,}2_{10}$
   - $6824{,}81_{10}$
   - $2467{,}42_{10}$
   - $28079{,}83_{10}$
   - 4095_{10}
   - $65468{,}932_{10}$
   - $1000{,}55_{10}$
   - $699{,}2_{10}$
   - $2047{,}33_{10}$
   - $789{,}19_{10}$

2. Transformar al sistema decimal los siguientes números:
   - $5A79{,}C8_{16}$
   - $1111011{,}011_{2}$
   - $77425{,}26_{8}$
   - $10101110{,}11_{2}$
   - $11001{,}1101_{8}$
   - $ABCD{,}7F_{16}$
   - $6724{,}61_{8}$
   - $2467{,}423_{16}$
   - $5A79{,}C8_{16}$
   - $13444{,}27_{8}$
   - $3FFFF_{16}$
   - $28079{,}7_{8}$
   - $10010101{,}1_{2}$
   - $1111000{,}001_{2}$
   - $62666{,}03_{8}$
   - $443221{,}77_{8}$
   - $ABCD{,}7F_{16}$
   - $4ED{,}6F2_{16}$
   - $4ED{,}6F2_{16}$
   - $10000{,}01_{8}$
   - $1111000{,}001_{2}$
   - $9988{,}62_{16}$
   - $111111{,}11_{2}$
   - $222457{,}3_{8}$

3. Construir una secuencia numérica, desde cero hasta sesenta, equivalente con el sistema decimal. Se deben tomar grupos de seis símbolos que correspondan con los siguientes: □, ☽, ⦿, ⭘; los valores posicionales son continuos y se incrementan de uno en uno. El equivalente decimal es el siguiente:
   - □ Cero unidades.
   - ☽ Una unidad.
   - ⦿ Dos unidades.
   - ⭘ Tres unidades.

4. Transformar al sistema requerido los siguientes números:
   - $3FFCD{,}4AB2_{16}$ → Octal
   - $1237650{,}771_{8}$ → Hexadecimal
   - $334156{,}2_{8}$ → Hexadecimal
   - 64202513_{8} → Hexadecimal
   - $10001{,}101_{16}$ → Octal
   - $ABCD6{,}2_{16}$ → Octal

5. Dado los siguientes números:
   a) 10111011101_{2} b) $6FAB{,}8_{16}$ c) $100100000111001010000110_{BCD}$ d) $58FF3D_{16}$
   e) 1111011010101011_{2} f) $5432{,}76_{8}$ g) $11000011001110000110_{Exc3}$ h) $7964{,}9_{10}$

   Hallar las sumas:
   I) a+b en octal II) c+e+f en hexadecimal III) c+d en binario
   IV) f+g+h en BCD V) b+e+a+f en octal VI) f+b+c en binario

6. Dado los siguientes números:
   a) $FA0B_{16}$ b) 1101101101_{2} c) 43375_{8}
   d) $7FFF_{16}$ e) -9863_{10} f) 1111000010101000_{2}

Realizar las siguientes operaciones aritméticas utilizando el formato de números con signo de 16 bits:

I) a - c **II)** b + a **III)** d - b
IV) e + c **V)** f - e **VI)** b + e + d

7 Un sistema de comunicación envía datos de 9 bits. En cada uno, se codifican dos dígitos BCD más un bit de paridad que es generado en la posición más significativa y con paridad par. Se pide detectar los errores que puedan ocurrir en los códigos BCD recibidos, e indicar si son de paridad y/o de código.

a) 101111001	b) 110011100	c) 111110001
d) 010000100	e) 010101011	f) 100000111

8. Los siguientes caracteres UNICODE son enviados en binario con paridad impar en el MSB. Detectar, por el método de paridad, si hay errores de comunicación, y de no ser así, indicar el símbolo correspondiente.

a) 101111110	b) 110100101	c) 101101110
d) 110101100	e) 001000001	f) 00100000

9. Dado los números:

a) $10011000011100000100_{BCD}$ b) 789463_{10}

c) $1100100010100011001111001_{Exc3}$ d) $01000111001110011000011100000_{BCD}$

Realizar las siguientes sumas en BCD.

I) a + c + d **II)** c + b **III)** a + b + c + d

10. Detectar y corregir los errores de los siguientes datos, dados en exceso 3, y codificados en Hamming de 7 bits con paridad par.

a) 1100001	b) 1000110	c) 0101100
d) 1111111	e) 0001110	f) 0000001

BIBLIOGRAFÍA.

- CUESTA, Luís M. PADILLA G, Antonio. REMIRO D, Fernando. (1993). Electrónica digital. Madrid: McGraw Hill. S/f. p.445.
- GAJSKI, Daniel D. (1997). Principios de diseño digital. Madrid: Prentice Hall Iberia. S/f. p.488. "Principles of digital design". Traducido por: Alberto Prieto Espinosa.
- LLORIS, Antonio. PRIETO, Alberto. (1996). Diseño lógico. Madrid: McGraw Hill. S/f. p.403.
- MANO, Morris. KIME, Charles. (1998). Fundamentos de diseño lógico y computadoras. México: Prentice Hall. Primera edición en español. P.604. "Logic and computer design fundamentals". Traducido por: Teresa Sanz Falcón.
- NEAMEN A, Donald. (1999). Análisis y diseño de circuitos electrónicos. Tomo II. México: McGraw Hill. S/f. p.1176. "Electronic circuit analysis and design". Traducido por: Felipe Castro Pérez.
- NELSON, V. NAGLE, H. CARROLL, B. IRWIN, J. (1996). Análisis y diseño de circuitos lógicos digitales. México: Prentice Hall. Primera edición. p.842. "Digital logic circuit analysis and design". Traducido por: Oscar A. Palmas V.
- TOCCI, Ronald. (1995). Sistemas digitales principios y aplicaciones. México: Prentice Hall. Quinta edición. p.823. "Digital systems principles and applications". Traducido por: Edmundo G. Urbina M.
- WARKELY, John F. (1997). Diseño digital principios y prácticas. México: Prentice Hall. S/f. p.743. "Digital design principles and practices". Traducido por: Gutiérrez R. Raymundo H.