Sistemas Numéricos y Códigos Binarios

Marcelo Guarini
Departamento de Ingeniería Eléctrica,

5 de Abril, 2005

1 Sistemas Numéricos en Cualquier Base

En el sistema decimal, cualquier número puede representarse como una suma ponderada de potencias de 10, donde 10 es la base del sistema decimal. Ejemplo:

$$1492 = 1 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

Evidentemente, los números con parte fraccionaria también pueden ser representados de esta forma. Por ejemplo:

$$1492.1012 = 1 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 + 1 \times 10^{-1} + 0 \times 10^{-2} + 1 \times 10^{-3} + 2 \times 10^{-4}$$

De esta misma forma, cualquier número en cualquier base b, con n dígitos en la parte entera y m dígitos en la parte decimal, puede escribirse

$$a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_1 b^1 + a_0 b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots a_{-m}b^{b-m} \tag{1}$$

Por ejemplo supongamos el número binario (base 2) 1101,011

$$1101.011 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}. \tag{2}$$

Para encontrar el valor de este número binario (1101,011) en el sistema decimal basta con resolver la expresión de la derecha en la ecuación 2, utilizando aritmética de base 10, es decir

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 13.375$$

Conclusión : para pasar un número en base b a base decimal se aplica la expresión (1), operando en aritmética decimal.

Es posible pasar un número que está en base b_1 a base b_2 distinta de la decimal, también utilizando la expresión (1). Sin emargo esta tarea es muy compleja, ya que las operaciones de multiplicación y suma deben efectuarse en aritmética de base b_2.

Veamos ahora como pasar un número que esta en base decimal a otra base distinta b. Consideremos primero un número entero, por ejemplo: 147, y supongamos que debemos pasarlo a base 2.
Como vimos una forma posible es aplicar (1) operando en aritmética de base 2. Sin embargo esto último resulta muy complejo debido a que, como dijimos, estamos acostumbrados a operar en base decimal.

1.1 Regla de las restas sucesivas

Una alternativa posible es restar sucesivamente las potencias de 2, partiendo por la más alta que cabe en el número a convertir, de la siguiente forma:

$$\begin{aligned} 147 - 2^7 = 19 &\Longrightarrow 1 \\ 19 - 2^6 < 0 &\Longrightarrow 0 \\ 19 - 2^5 < 0 &\Longrightarrow 0 \\ 19 - 2^4 = 3 &\Longrightarrow 1 \\ 3 - 2^3 < 0 &\Longrightarrow 0 \\ 3 - 2^2 < 0 &\Longrightarrow 0 \\ 3 - 2^1 = 1 &\Longrightarrow 1 \\ 1 - 2^0 = 0 &\Longrightarrow 1 \end{aligned}$$

Como se aprecia, este método es una forma de aplicación inversa de la ecuación (1). El número binario correspondiente es el de la columna de la derecha, desde el dígito más significativo hasta el menos significativo (10010011).

Este método de restas sucesivas es evidentemente aplicable además a números con parte fraccionaria.

1.2 Regla de las divisiones sucesivas

Si se lo analiza en detalle, el método que veremos ahora también es una forma inversa de aplicar la expresión (1). Sin embargo en esta ocasión el primer dígito obtenido corresponderá al menos significattivo. Contrariamente al método de las restas sucesivas, este método no es aplicable a números con parte fraccionaria.

Consideremos el mismo ejemlo anterior, pasar a base 2 el número $(147)_{10}$. El procedimiento consiste en dividir repetidamente por dos generando sólo la parte entera de la división. El resto corresponderá al dígito buscado.

$$
\begin{array}{rcrcll}
147 & : & 2 & = & 73 & \\
1 & & & & \longrightarrow & 1 \\
 & & & & & \\
73 & : & 2 & = & 36 & \\
1 & & & & \longrightarrow & 1 \\
 & & & & & \\
36 & : & 2 & = & 18 & \\
0 & & & & \longrightarrow & 0 \\
 & & & & & \\
18 & : & 2 & = & 9 & \\
0 & & & & \longrightarrow & 0 \\
 & & & & & \\
9 & : & 2 & = & 4 & \\
1 & & & & \longrightarrow & 1 \\
 & & & & & \\
4 & : & 2 & = & 2 & \\
0 & & & & \longrightarrow & 0 \\
 & & & & & \\
2 & : & 2 & = & 1 & \\
0 & & & & \longrightarrow & 0 \\
 & & & & & \\
1 & : & 2 & = & 0 & \\
1 & & & & \longrightarrow & 1
\end{array}
$$

La columna de la derecha contiene el número buscado, sin embargo, como dijimos, en esta ocasión el primer dígito obtenido es el menos significativo y el último el más significativo.

El método de las divisiones sucesivas no sólo es valido para pasar un número decimal a base 2. Dividiendo por la base correspondiente b es posible pasar a cualquier base, operando en arittmética decimal.

1.3 Regla de las multiplicacione sucesivas

Como se mencionó, la regla de las divisiones sucesivas no es aplicable a la parte fraccionaria de un número. Supongamos que debemos pasar el número $(147,6875)_{10}$ a base 2. En este caso la parte entera $(147)_{10}$ se pasa utilizando la regla de la divisiones sucesivas. Para la parte fraccionaria se procede de la siguiente forma:

$$
\begin{array}{cccc}
 & & 0.6875 & \times \quad 2 \\
1 & \longleftarrow & 1.3750 & \\
 & & & \\
 & & 0.375 & \times \quad 2 \\
0 & \longleftarrow & 0.750 & \\
 & & & \\
 & & 0.75 & \times \quad 2 \\
1 & \longleftarrow & 1.50 & \\
 & & & \\
 & & 0.5 & \times \quad 2 \\
1 & \longleftarrow & 1.0 &
\end{array}
$$

En esta regla, el dígito más significativo es el que se obtiene primero, y el menos significativo el último. Como se ve en el ejemplo, el proceso termina cuando la parte fracionaria se hace 0. **Es importante notar que en algunos casos este proceso puede ser ∞ ya que una parte fraccionaria con número finito de digitos en una base b_1 puede no tener representación exacta en otra base b_2.**

1.4 Conversion entre bases que son potencias entre ellas

Existe un método muy simple para convertir números entre bases que son potencias unas de otra, por ejemplo entre base 2, 4, 8, 16, etc. Para estos casos, la expresión (1) puede factorizarse en forma exacta, permitiendo una conversión dígito a díigito. Veamos el caso para convertir entre base 2 y base 8. Consideremos un número octal de tres dígitos enteros, en este caso la expresión (1) se factoriza de la siguiente forma:

$$a_8 \times 2^8 + a_7 \times 2^7 + a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0 =$$

$$(2^3)^2(a_8 \times 2^2 + a_7 \times 2^1 + a_6 \times 2^0) + (2^3)^1(a_5 \times 2^2 + a_4 \times 2^1 + a_3 \times 2^0) + (2^3)^0(a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0) =$$

$$8^2(a_8 \times 2^2 + a_7 \times 2^1 + a_6 \times 2^0) + 8^1(a_5 \times 2^2 + a_4 \times 2^1 + a_3 \times 2^0) + 8^0(a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0)$$

De la factorización se aprecia que cada uno de los grupos de tres dígitos binarios, $(a_8a_7a_6)$, $(a_5a_4a_3)$, $(a_2a_1a_0)$, corresponde a cada uno de los dígitos del número en base octal. De la misma forma, es posible factorizar la expresión (1) considerando grupos de a cuatro dígitos binarios. En este caso la factorización corresponde al sistema numérico de base 16 o sistema hexadecimal. Vemos que en ambos casos, todas las combinaciones de dígitos binarios, tres en el caso octal y cuatro en el caso hexadecimal, corresponden a cada uno de los dígitos del sistema considerado, ni una más y ni una menos. La tabla 1 ilustra claramente lo que hemos establecido.

Tabla 1: Combinaciones binarias correspondientes al sistema numérico octal y hexadecimal.

Binario		Octal	Binario		Hexadecimal
000	$\longrightarrow$	0	0000	$\longrightarrow$	0
001	$\longrightarrow$	1	0001	$\longrightarrow$	1
010	$\longrightarrow$	2	0010	$\longrightarrow$	2
011	$\longrightarrow$	3	0011	$\longrightarrow$	3
100	$\longrightarrow$	4	0100	$\longrightarrow$	4
101	$\longrightarrow$	5	0101	$\longrightarrow$	5
110	$\longrightarrow$	6	0110	$\longrightarrow$	6
111	$\longrightarrow$	7	0111	$\longrightarrow$	7
			1000	$\longrightarrow$	8
			1001	$\longrightarrow$	9
			1010	$\longrightarrow$	A
			1011	$\longrightarrow$	B
			1100	$\longrightarrow$	C
			1101	$\longrightarrow$	D
			1110	$\longrightarrow$	E
			1111	$\longrightarrow$	F

Veamos algunos ejemplos típicos de conversión entre base binaria, octal y hexadecimal.

1. $(1101011101,0111011)_2 \longrightarrow (N)_8$

$$\begin{array}{ccccccccccc} (& \underbrace{001} & \underbrace{101} & \underbrace{011} & \underbrace{101} & , & \underbrace{011} & \underbrace{101} & \underbrace{100} &)_2 \\ & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \\ (& 1 & 5 & 3 & 5 & , & 3 & 5 & 4 &)_8 \end{array}$$

2. $(\text{E4A,5C})_{16} \longrightarrow (N)_2$

$$\begin{array}{ccccccccc} \text{E} & & \text{4} & & \text{A} & , & \text{5} & & \text{C} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \overbrace{1110} & & \overbrace{0100} & & \overbrace{1010} & , & \overbrace{0101} & & \overbrace{1100} \end{array}$$

3. $(\text{B7,D})_{16} \longrightarrow (N)_8$

$$\begin{array}{ccccccccc} (& \text{B} & & \text{7} & , & \text{D} &)_{16} & & \\ & \downarrow & & \downarrow & & \downarrow & & & \\ (& \overbrace{1011} & & \overbrace{0111} & , & \overbrace{1101} &)_2 & = & (\underbrace{010} \quad \underbrace{110} \quad \underbrace{111} \quad , \quad \underbrace{110} \quad \underbrace{100} \quad)_2 \\ & & & & & & & & \downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow \quad \downarrow \\ & & & & & & & & (\quad 2 \quad 6 \quad 7 \quad , \quad 6 \quad 4 \quad)_8 \end{array}$$

Es importante notar que los grupos de 3 o 4 dígitos binarios deben hacerse de derecha a izquierda para la parte entera y de izquierda a derecha para la parte fraccionaria. Para completar los grupos se agregan ceros a la izquierda para la parte entera y a la derecha para la parte fraccionaria.

2 Representación Binaria de los Dígitos Decimales

Debido a la dificultad para convertir en forma simple un número binario en su correspondiente decimal, un grán número de máquinas dígitales simples operan directamente utilizando dígitos decimales codificados en binario. Esto permite mostrar los resultados directamente al usuario en el sistema decimal, a través de un circuito combinacional muy simple y de muy bajo costo. Entre las máquinas que utilizan el sistema de dígitos decimales codificados en binario estan las calculadoras. Un grán número de microprocesadores y microcontroladores proveen instrucciones de máquina para operar tanto en binario como utilizando dígitos decimales codificados en binario.

2.1 Código BCD

El código BCD (Binary Coded Decimal), es la forma más natural de representar los dígitos decimales en binario. Corresponde justamente al valor binario de cada uno de los dígitos decimales como muestra la tabla 2.

Tabla 2: Código BCD.

Binario		Decimal
0000	$\longrightarrow$	0
0001	$\longrightarrow$	1
0010	$\longrightarrow$	2
0011	$\longrightarrow$	3
0100	$\longrightarrow$	4
0101	$\longrightarrow$	5
0110	$\longrightarrow$	6
0111	$\longrightarrow$	7
1000	$\longrightarrow$	8
1001	$\longrightarrow$	9

Aplicando la expresión (1) a los números binarios de la primera lista de la tabla 2, permite obtener los dígitos decimales de la segunda lista. Al hacer esto, cáda dígito binario de la primera columna es multiplicado por $2^3 = 8$, cada dígito de la segunda columna por $2^2 = 4$, cada dígito de la tercera columna por $2^1 = 2$, y cada dígito de la cuarta columna por $2^0 = 1$. Por este motivo, al código BCD también se le ha llamado código 8421. Al factor formado por la base elevada a la potencia que corresponde a cada columna se le llama **peso o ponderador**. Por este motivo, a este tipo de códigos se les conoce como **códigos en base a pesos o códigos ponderados**, en ingles **weighted codes**.

Para escribir un número decimal en código BCD simplemente se escribe cada dígito expresado en binario según la tabla 2 en el mismo orden en que aparececn en la representación decimal. Por ejemplo

$$529 \longrightarrow 0101\ 0010\ 1001$$

De la misma forma se escriben los números con parte fraccionaria, por ejemplo

$$4673,51 \longrightarrow 0100\ 0110\ 0111\ 0011\ ,\ 0101\ 0001$$

2.2 Otros códigos ponderados

Aunque como mencionamos el código de este tipo más natural es el código BCD u 8421, existen otros códigos ponderados. Los pesos en este caso no tienen porqué corresponder a la base elevada a la potencia correspondiente a la posición del dígito binario. Algunos ejemplos de estos códigos se muestran en la tabla 3.

Tabla 3: Ejemplos de códigos en base a pesos para representar los dígitos decimales.

6421	→	Decimal	4321	→	Decimal	5211	→	Decimal
0000	→	0	0000	→	0	0000	→	0
0001	→	1	0001	→	1	0001	→	1
0010	→	2	0010	→	2	0010	→	1
0011	→	3	0011	→	3	0011	→	2
0100	→	4	0100	→	3	0100	→	2
0101	→	5	0101	→	4	0101	→	3
0110	→	6	1000	→	4	0110	→	3
1000	→	6	0110	→	5	0111	→	4
0111	→	7	1001	→	5	1000	→	5
1001	→	7	0111	→	6	1001	→	6
1010	→	8	1010	→	6	1010	→	6
1011	→	9	1011	→	7	1011	→	7
			1100	→	7	1100	→	7
			1101	→	8	1101	→	8
			1110	→	9	1110	→	8
						1111	→	9

Como se puede apreciar en la tabla 3, no todos los dígitos decimales tienen una sola representación. Esta es una desventaja que presentan los códigos ponderados, distintos del BCD u 8421. Algunos de los códigos ponderados, como el 5211 de la tabla **??**, tienen la propiedad de ser autocomplementarios. Esto significa que el complemento correspondiente al 0 decimal es el 9, el complemento del 1 decimal es el 8, el complemento del 2 decimal es el 7, etc. De la misma forma, el complemento del 9 es el 0, el complemento del 8 es el 1, etc.

También es posible diseñar códigos con combinaciones de pesos positivos y negativos, por ejemplo el código 642-1. Sin embargo, estos no son utilizados.

3 Otros Códigos Binarios

Se han desarrollado una serie de códigos binarios para aplicaciones específicas. Estos códigos aprovechan características de las aplicación para las cuales se construyen con el objeto de obtener el mejor desempeño posible.

3.1 Código exceso de 3

Los códigos con excesos se obtienen sumando una cantidad determinada (el exceso), a cada palabra del código original. De esta forma, el código exceso de tres se obtiene sumando tres a cada palabra del código BCD de la siguiente forma

Tabla 4: Código Exceso de Tres.

Decimal	Binario		Ex. de Tres
0	0000	$\longrightarrow$	0011
1	0001	$\longrightarrow$	0100
2	0010	$\longrightarrow$	0101
3	0011	$\longrightarrow$	0110
4	0100	$\longrightarrow$	0111
5	0101	$\longrightarrow$	1000
6	0110	$\longrightarrow$	1001
7	0111	$\longrightarrow$	1010
8	1000	$\longrightarrow$	1011
9	1001	$\longrightarrow$	1100

Como se aprecia en la tabla 4 el código exceso de tres tiene la característica de ser autocomplementario.

El uso de códigos con exceso es particularmente atractivo para representar los exponentes en números de punto flotante, de tal forma que el valor mínimo del exponente quede representado sólo por ceros.

3.2 Códigos Cíclicos

En un gran número de aplicaciones prácicas es muy deseable utilizar códigos en los que todas palabras sucesivas difieran sólo en un bit. Este tipo de códigos se conoce como códigos cíclicos. Un código cíclico particularmente importante es el *cógigo Gray*. La característica que hace que este código sea útil es la simplicidad del procedimiento para convertir de un número binario a su correspondiente en código Gray y viseversa.

Sea $g_n, \cdots, g_1, g_0$ una palabra en código Gray de $n+1$ bits, y sea $b_n, \cdots, b_1, b_0$, su correspondiente número binario. Los subíndices 0 y n corresponden al dígito menos significativo y meas significativo respectivamente. El $i-esimo$ dígito g_i se puede obtener del correspondiente binario, de la siguiente forma

$$g_i = b_i \oplus b_{i+1} \qquad 0 \leq i \leq n-1$$
$$g_n = b_n$$

Donde la operación $\oplus$, corresponde al *Or-Exclusivo* definido como

Tabla 5: Función *Or-Exclusivo*.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Para convertir de código Gray a código binario se comienza con el dígito de más a la izquierda, dígito más significativo, y se procede hasta el dígito menos significativo, haciendo $b_i = g_i$ si el número de unos que precede a g_i es par, y haciendo $b_i = \bar{g}_i$ si el número de unos que precede a g_i es impar. En notación matemática

$$b_n = g_n$$
$$b_i = g_n \oplus g_{n-1} \oplus \cdots \oplus g_i$$

El código Gray es miembro de una clase llamada *códigos reflejados*. El término *reflejado* se utiliza para designar a los códigos que tienen la propiedad de que el código de n bits puede generarse por reflexión del código de $(n-1)$ bit como se ilustra en la tabla 6.

4 Representación de Números con Signo

El signo de los números que utilizan las máquinas digitales se especifica mediante un bit llamado *bit de signo*. Generalmente se ubica en primer lugar, a la izquierda de los dígitos correspondientes a la magnitúd del número. Por convención los números positivos se especifican con un bit de signo igual a cero y los negativos con un bit igual a uno. Aunque la magnitud de los números positivos corresponde al código binario posicional estándar, existen diversos métodos para representar la magnitud de los números negativos.

4.1 Números signo y magnitud

El método más simple para representar números con signo, corresponde al llamado *signo y magnitud*. Simplemente, consiste en la magnitud del número, utilizando el sistema binario posicional estándar,

Tabla 6: Generación del código Gray por reflexión.

00	0	00	0	000
01	0	01	0	001
11	0	11	0	011
10	0	10	0	010
	1	10	0	110
	1	11	0	111
	1	01	0	101
	1	00	0	100
			1	100
			1	101
			1	111
			1	110
			1	010
			1	011
			1	001
			1	000

agregando, como dijimos, el signo a la izquierda del bit más significativo. Aunque este código es muy simple, para su uso práctico en máquinas digitales requiere de circuitos y algoritmos digitales complejos, que elevan tanto el costo en términos de componentes, como de tiempo de cálculo. En la tabla 7 se muestra la representación signo y magnitud para los primeros 16 números positivos (incluyendo el 0) y para los primeros 16 negativos, frente a otras representaciones más convenientes del punto de vista de las aplicaciones. En forma general un número en signo y magnitud se representa de la siguiente forma

$$+a_{n-1}\cdots a_0, a_{-1}\cdots a{-}m = 0\ a_{n-1}\cdots a_0, a_{-1}\cdots a{-}m$$
$$-a_{n-1}\cdots a_0, a_{-1}\cdots a{-}m = 1\ a_{n-1}\cdots a_0, a_{-1}\cdots a{-}m$$

4.2 Representación complementaria

Los números complementarios representan la base de la aritmética complementaria. Esta permite realizar operaciones matemáticas con números positivos y negativos. En la representación complementaria los números positivos se representan igual que en la representación signo y magnitud, sin embargo, los números negativos se representan por el complemento del número positivo correspondiente. En general existen dos tipos de representación complementaria, el complemento a la base y el complemento disminuido a la base. A continuación revisaremos ambas representaciones aplicadas al sistema binario o de base dos.

Tabla 7: Algunos códigos para representar números con signo.

Decimal	Signo y Magnitud	Complemento de Dos	Complemento de Uno
+15	01111	01111	01111
+14	01110	01110	01110
+13	01101	01101	01101
+12	01100	01100	01100
+11	01011	01011	01011
+10	01010	01010	01010
+9	01001	01001	01001
+8	01000	01000	01000
+7	00111	00111	00111
+6	00110	00110	00110
+5	00101	00101	00101
+4	00100	00100	00100
+3	00011	00011	00011
+2	00010	00010	00010
+1	00001	00001	00001
0	00000	00000	00000
	10000	—	11111
-1	10001	11111	11110
-2	10010	11110	11101
-3	10011	11101	11100
-4	10100	11100	11011
-5	10101	11011	11010
-6	10110	11010	11001
-7	10111	11001	11000
-8	11000	11000	10111
-9	11001	10111	10110
-10	11010	10110	10101
-11	11011	10101	10100
-12	11100	10100	10011
-13	11101	10011	10010
-14	11110	10010	10001
-15	11111	10001	10000
-16	—	10000	—

4.2.1 Complemento a la base 2

El complemento de un número binario a la base dos o más simple, complemento de dos de un número binario, está dado por

$$2^n - (N)_2 \tag{3}$$

donde n es el número de bits del número binario $(N)_2$. El complemento de dos de un número binario es la forma más común de representar números negativos. El motivo es que no se requiere *hardware* digital especial para operar números con signo. En efecto, un sumador binario común permite, utilizando este código, sumar y restar. La tabla 7 muestra los números posibles de representar en complemento de dos si utilizamos 5 dígitos binarios, incluyendo el bit de signo. Veamos como obtener el complemento de dos de un número.

Determinar el complemento de dos del número $(0110100010)_2$

Aplicando la ecuación (3) se tiene

$$\begin{aligned} 2^{10} - (0110100010)_2 &= (10000000000)_2 - (0110100010)_2 \\ &= (1001011110)_{Comp-2} \end{aligned}$$

Una forma más simple, que corresponde a restar el número $(N)_2$ a 2^n, donde n es el número de dígitos incluyendo el bit de signo, es complementar el número y sumar uno al resultado, de la siguiente forma:

$$\begin{array}{rcl} (N)_2 & = & (0110100010)_2 \\ (\bar{N})_2 & = & (1001011101)_2 \\ & + & (0000000001)_2 \\ \hline & & (1001011110)_{Comp-2} \end{array}$$

Otra forma aun más rápida consiste en inspeccionar el número de derecha a izquierda (desde menos significativo a más significativo), copiando el número tal como está hasta el primer uno inclusive. Desde allí, se complementan tódos los dígitos.

$$\text{se complementa} \rightarrow \quad \overbrace{01101000}^{10010111}\overbrace{10}^{10} \quad \leftarrow \text{se deja igual}$$

Estos tres procedimientos funcionan exactamente igual para pasar un número negativo a su correspondiente positivo.

4.2.2 Complemento disminuido a la base 2

El complemento disminuido a la base dos, llamado complemento de uno, está dado por la expresión

$$2^n - (N)_2 - 1 \tag{4}$$

donde n corresponde al número de bits de $(N)_2$. Para determinar el complemento de 1 de un número binario determinado, se puede aplicar directamente la ecuación (4). Sin embargo resulta mucho más fácil complementar el número positivo correspondiente, ya que esta operación es lo mismo que aplicar (4).

Como ejemplo, determinemos el complemento de 1 de $(N)_2 = (0110100010)_2$.

$$\begin{aligned} (N)_2 &= (0110100010)_2 \\ (\bar{N})_2 &= (1001011101)_{Comp-1} \end{aligned}$$

Por motivos que veremos en la próxima sección, el complemento de 1 no se utiliza en máquinas digitales. Aunque las operaciones aritméticas requieren *hardware* más simple que el requerido por el código signo y magnitud, los algoritmos son aún más simples en la representación de complemento de 2, haciendo que el tiempo de cálculo, utilizando esta última representación, sea menor.

5 Aritmética Binaria

A continuación revisaremos como se realiza la operación de suma binaria de números de signo positivo y negativo utilizando los códigos de complemento de uno y de complemento de dos. La mejor forma de enfocar esta revisión es a través de ejemplos, los que además nos permitirán destacar algunas características especiales para corroborar la pertinancia de los resultados.

5.1 Suma en complemento de uno

Sumar en complemento de uno los números $(19)_{10}$ y $(7)_{10}$.

$$\begin{aligned} (19)_{10} &= (010011)_{Comp-1} \\ (7)_{10} &= (0111)_{Comp-1} \end{aligned}$$

Lo primero que debemos hacer es que los dígitos de signo coincidan. En complemento de uno esto se logra haciendo propagando el bit de signo tantos lugares como sea necesario y luego se efectúa la suma. En nuestro ejemplo

$$\begin{array}{rr} (19)_{10} & (010011)_{Comp-1} \\ + \quad (7)_{10} & + \quad (000111)_{Comp-1} \\ \hline (26)_{10} & (011010)_{Comp-1} \end{array}$$

El siguiente ejemplo nos ilustra que puede pasar cuando la cantidad de dígitos escogida no permite representar correctamente el resultado. Tratemos de sumar los números $(19)_{10}$ y $(17)_{10}$.

$$
\begin{array}{lrcl}
 & (19)_{10} & = & (010011)_{Comp-1} \\
+ & (17)_{10} & = + & (010001)_{Comp-1} \\
\hline
 & (36)_{10} & \neq & (100100)_{Comp-1} \quad \leftarrow \text{Error}
\end{array}
$$

Aunque la suma binaria es correcta, la interpretación en complemento de uno no lo es, ya que la suma de dos números positivos da como resultado un número negativo. Esto ocurre en el ejemplo debido a que con 6 dígitos binarios, incluído el bit de signo, podemos representar como máximo el número $(+31)_{10}$. Para solucionar el problema extendemos el bit de signo un lugar hacia la izquierda. Con esto podremos representar hasta el número $(+63)_{10}$. Entonces

$$
\begin{array}{lrcl}
 & (19)_{10} & = & (0010011)_{Comp-1} \\
+ & (17)_{10} & = + & (0010001)_{Comp-1} \\
\hline
 & (36)_{10} & = & (0100100)_{Comp-1} \quad \leftarrow \text{Correcto}
\end{array}
$$

Hasta ahora sólo hemos sumado números positivos, que es exactamente lo mismo que en binario puro excepto por el bit de signo, que debemos cuidar que esté en la posición adecuada y por la cantidad de bits, para poder representar el resultado correcto. Veamos ahora la suma de un número positivo y uno negativo. Sumar en complemento de uno los números $(+23)_{10}$ y $(-12)_{10}$

$$
\begin{array}{lrcl}
 & (12)_{10} & = & (01100)_{Comp-1} \\
 & (-12)_{10} & = & (10011)_{Comp-1} \\
 & & & \\
 & (23)_{10} & = & (010111)_{Comp-1} \\
\text{extendiendo el bit de signo de -12} & (-12)_{10} & = + & (110011)_{Comp-1} \\
\hline
 & (11)_{10} & \neq & (1001010)_{Comp-1} \quad \leftarrow \text{Error}
\end{array}
$$

Nuevamente, aunque el resultado de la suma binaria es correcto, en complemento de uno se obtuvo un resultado muy distinto de $(11)_{10}$. Este ejemplo ilustra la principal excepción que ocurre en complemento de uno. Para solucionar el problema, **cada vez que el *carry* es igual a uno, desde el bit de signo hacia afuera, este se elimina, pero a su vez se debe sumar uno al resultado**. En nuestro ejemplo,

$$
\begin{array}{r}
(001010)_{Comp-1} \\
+(000001)_{Comp-1} \\
\hline
(001011)_{Comp-1}
\end{array}
$$

De esta forma, el resultado es el correcto. La suma de números negativos es similar, la misma regla aplica cuando se produce un *carry* desde el bit de signo hacia afuera. Sólo hay que cuidar, de la misma forma que para dos positivos, que la suma de dos negativos genere un resultado que pueda ser representado por el número de bits que estamos utilizando. Si no lo es, es necesario extender el bit de signo hacia la izquierda, en este caso igual a 1, tantas veces como sea requerido.

5.2 Suma en complemento de dos

Para la suma en complemento de dos se procede en forma similar a la suma en complemento de uno. Sin embargo, tiene la ventaja de que cuando se produce un *carry* igual a uno, desde el bit de signo hacia afuera, este se elimina, sin requerir la suma de un 1 al resultado. Esto hace que el código complemento de dos sea hoy el más utilizado en las máquinas digitales, ya que permite simplificar el algoritmo de suma, permitiento tiempos de cálculo más cortos. De la misma forma que para el complemento de uno, hay que cuidar que los resultados, cuando se suma dos númerros positivos o cuando se suma dos números negativos, estén dentro de la representación que permite el número de dígitos que estamos utilizando.

Veamos algunos ejemplos.

Sumar en complemento de dos los números $(19)_{10}$ y $(7)_{10}$.

$$\begin{array}{rcl} (19)_{10} & = & (010011)_{Comp-2} \\ (7)_{10} & = & + \quad (000111)_{Comp-2} \\ \hline (26)_{10} & = & (011010)_{Comp-2} \end{array}$$

Note que el procedimiento y el resultado es idéntico a la operación en complemento de uno, incluso la forma en que debimos extender el bit de signo para el número $(7)_{10}$ es idéntica.

Sumemos ahora los números $(-19)_{10}$ y $(-17)_{10}$. En este caso se debe recordar alguna de las formas para obtener el número negativo a partir del correspondiente positivo.

$$\begin{array}{rcl} (-19)_{10} & = & (101101)_{Comp-2} \\ + \quad (-17)_{10} & = & + \quad (101111)_{Comp-2} \\ \hline (-36)_{10} & \neq & (1011100)_{Comp-2} \quad \leftarrow \text{Error} \end{array}$$

Eliminando el *carry*, en color rojo, que se produce desde el bit de signo hacia afuera, vemos que el resultado no es correcto, ya que la suma de dos números negativos nos generó un resultado positivo. Como ya sabemos, el problema ocurre porque no es posible representar el número $(-36)_{10}$ con 6 dígitos binarios incluyendo el bit de signo. Entonces debemos extender el bit de signo en ambos números un lugar a la izquierda.

$$\begin{array}{rcl} (-19)_{10} & = & (1101101)_{Comp-2} \\ + \quad (-17)_{10} & = & + \quad (1101111)_{Comp-2} \\ \hline (-36)_{10} & = & (11011100)_{Comp-2} \quad \leftarrow \text{Correcto} \end{array}$$

Eliminando el *carry*, en color rojo, se produce ahora el resultado correcto.