Representación de números en binario

Héctor Antonio Villa Martínez
Programa de Ciencias de la Computación
Universidad de Sonora

El sistema binario es el más utilizado en sistemas digitales porque es más sencillo diseñar circuitos digitales para manejar números binarios que para cualquier otra base. Por ese motivo, este reporte presenta los principales métodos para representar números en binario. Los métodos se muestran dependiendo del tipo del número: números enteros sin signo (Sección 1), números enteros con signo (Sección 2) y números reales (Sección 3).

1. Enteros sin signo

A los números enteros positivos, incluyendo el cero, se les conoce como enteros sin signo porque, al contrario de los números negativos, se pueden escribir sin signo. Los enteros sin signo se pueden representar en binario convirtiéndolos a base 2 (ver [2]). Alternativamente, se puede diseñar un código binario para representar cada dígito del número. El Cuadro 1 presenta los códigos binarios más conocidos para dígitos decimales.

Usando un código binario un entero sin signo se representa reemplazando cada dígito por su equivalente binario. Así, el número 126.47_{10} se representa como 000100100110.01000111 en BCD (binary-coded decimal o decimal codificado en binario):

```
  1    2    6   .   4    7
0001 0010 0110  . 0100 0111
```

Los códigos BCD y 6-3-1-1 son códigos ponderados de 4 dígitos binarios (bits) porque cada bit tiene asignado un peso. En el caso de BCD los pesos

Cuadro 1: Principales códigos binarios para dígitos decimales (basado en Roth [3, p. 19]).

Dígito decimal	**Código BCD**	**Código 6-3-1-1**	**Código exceso 3**	**Código 2 de 5**	**Código Gray**
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

son 8, 4, 2 y 1. Para el código 6-3-1-1, como su nombre lo indica, los pesos son 6, 3, 1 y 1. De esta forma, dado el código binario 0111 es posible saber que representa 7 en BCD ($0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7$), y 5 en código 6-3-1-1 ($0 \times 6 + 1 \times 3 + 1 \times 1 + 1 \times 1 = 5$).

El código exceso 3 se obtiene a partir de BCD sumándole 3 a cada código. El código 2 de 5 tiene la propiedad de que en cada código solo 2 de los 5 bits son 1. Por su parte, en el código Gray cada código consecutivo difiere en un solo bit. Estos dos últimos códigos son no ponderados. Es decir, el valor decimal de un dígito codificado en código 2 de 5 o en código Gray no puede obtenerse a partir de una fórmula como en el caso de los códigos ponderados.

Las combinaciones de dígitos binarios no presentes en el Cuadro 1 se consideran códigos no válidos. Con 4 bits se pueden generar hasta 16 (2^4) códigos binarios. Sin embargo, como solo hay 10 dígitos decimales en cada código binario de 4 bits hay 6 combinaciones que no se usan. En BCD, por ejemplo, los códigos 1010, 1011, 1100, 1101, 1110 y 1111 no son válidos. En general, los códigos no válidos se pueden usar para simplificar el diseño de convertidores de código como se verá mas adelante.

Cuadro 2: Signo y magnitud con 4 bits.

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

2. Enteros con signo

El uso del signo permite representar números enteros negativos y positivos. Los tres métodos que se estudian en ésta sección tienen en común que los enteros positivos se representan directamente por su valor en binario y que reservan el primer bit para indicar el signo del número, 0 para los positivos y 1 para los negativos. Las diferencias estriban en la forma de representar los enteros negativos, en la facilidad que ofrecen para la realización de las operaciones aritméticas básicas y en el rango de enteros que se pueden representar. Los tres métodos que se presentan son: signo y magnitud (Subsección 2.1), complemento a uno (Subsección 2.2) y complemento a dos (Subsección 2.3).

2.1. Signo y magnitud

En el método de signo y magnitud se reserva un bit para el signo y el resto para la magnitud. Usando este método con n bits se pueden representar números enteros en el rango entre -2^{n-1} y $+2^{n-1}$. Como se muestra en el Cuadro 2, para números de cuatro bits el rango es entre -7 y $+7$. Note que hay dos representaciones para el cero, una conocida como el cero negativo (-0) y la otra como el cero positivo ($+0$). La única desventaja, por llamarla de algún modo, de ésta situación es que se desperdicia un número binario que podría usarse para representar algún otro número decimal.

El método de signo y magnitud es sencillo de comprender. No obstante, algunas operaciones aritméticas, como la suma y la resta, son relativamente más complicadas que en los métodos que se verán a continuación.

Cuadro 3: Complemento a uno con 4 bits.

Decimal	Complemento a uno	Decimal	Complemento a uno
-7	1000	$+0$	0000
-6	1001	$+1$	0001
-5	1010	$+2$	0010
-4	1011	$+3$	0011
-3	1100	$+4$	0100
-2	1101	$+5$	0101
-1	1110	$+6$	0110
-0	1111	$+7$	0111

2.2. Complemento a uno

En el método de complemento a uno, como se dijo arriba, los enteros positivos se representan directamente en binario. Los números negativos, en cambio, se representan mediante su complemento a uno, el cual se encuentra convirtiendo el valor absoluto del número negativo a binario e intercambiando los unos y ceros del resultado. Como ejemplo, considere el problema de representar el número decimal -12 con 8 bits. Primero se convierte su valor absoluto, es decir $+12$, a binario y se escribe con 8 bits, 00001100. Segundo, se intercambian los unos y ceros, en este caso el resultado es 11110011 que es la representación de -12 en complemento a uno.

Con n bits se pueden representar enteros en el rango entre -2^{n-1} y $+2^{n-1}$. Como se muestra en el Cuadro 3, con 4 bits el rango es de -7 a $+7$. Note que al igual que en el método de signo y magnitud hay dos representaciones para el cero. Sin embargo, el método de complemento a uno tiene la ventaja sobre el de signo y magnitud de que las operaciones de suma y resta son más fáciles de implementar.

La suma de dos enteros con signo se realiza considerando la representación binaria de los enteros como si fueran números positivos, aplicando la suma binaria común (ver [2]) y revisando si se produjo un acarreo final. Si no hubo acarreo final el resultado de la suma binaria es el resultado correcto. En cualquier otro caso, el acarreo final se le añade a la suma binaria para obtener el resultado correcto de la suma. También existe la posibilidad de que ocurra un *overflow*, en cuyo caso el resultado de la suma binaria es incorrecto porque está fuera del rango.

Como ejemplos, considere las sumas $5+6$, $5+(-6)$, $-5+6$ y $-5+(-6)$, asumiendo que los números se representan en binario con 4 bits.

$$\begin{array}{rll} & 0101 & (+5) \\ + & 0110 & (+6) \\ \hline & 1011 & (-4) \quad \text{Resultado incorrecto} \end{array}$$

El resultado es incorrecto porque el resultado de la suma $5+6$ es 11 que no se puede representar con 4 bits (ver Cuadro 3).

$$\begin{array}{rll} & 0101 & (+5) \\ + & 1001 & (-6) \\ \hline & 1110 & (-1) \quad \text{Resultado correcto} \end{array}$$

No hubo acarreo final y el resultado es correcto.

$$\begin{array}{rll} & 1010 & (-5) \\ + & 0110 & (+6) \\ \hline & 10000 & \\ + & 1 & \text{Acarreo final} \\ \hline & 0001 & (+1) \quad \text{Resultado correcto} \end{array}$$

Hubo un acarreo final, el cual se suma al resultado de la suma binaria para obtener el resultado correcto.

$$\begin{array}{rll} & 1010 & (-5) \\ + & 1001 & (-6) \\ \hline & 10011 & \\ + & 1 & \text{Acarreo final} \\ \hline & 0100 & (+4) \quad \text{Resultado incorrecto} \end{array}$$

En este caso también hubo un acarreo final, sin embargo el resultado final es incorrecto porque -11 no se puede representar con 4 bits.

De los ejemplos anteriores se desprende que una forma de detectar si ocurrió un overflow es cuando al sumar dos números positivos el resultado es negativo, o cuando al sumar dos enteros negativos el resultado es positivo.

La resta de dos enteros con signo es trivial. Si se desea calcular la resta $A-B$, la operación se escribe como $A+(-B)$, se encuentra el complemento a uno de B y se hace la suma como se ilustró arriba.

2.3. Complemento a dos

El método de complemento a dos es muy similar al de complemento a uno. Los enteros positivos se representan directamente en binario y los negativos por su complemento a dos, el cuál se encuentra obteniendo primero el complemento a uno y luego sumándole 1 al resultado. Otra opción es aplicar

Cuadro 4: Complemento a dos con 4 bits.

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

el siguiente procedimiento. El primer paso es convertir el valor absoluto del entero negativo a binario. El segundo es recorrer de derecha a izquierda el binario, pasando al resultado final los ceros y el primer uno. Y tercero, se intercambian los ceros y unos del resto del binario. El resultado es el complemento a dos del entero negativo.

Como ejemplo del procedimiento anterior, considere obtener el complemento a dos de -12 usando 8 bits. Primero se convierte su valor absoluto, es decir $+12$, a binario y se escribe con 8 bits, 00001100. Segundo, de derecha a izquierda se pasan los ceros hasta el primer uno (100 en el ejemplo). Finalmente, se intercambian los unos y ceros del resto del binario, obteniendo 11110100 que es la representación de -12 en complemento a dos.

Con n bits se pueden representar enteros en el rango entre -2^n y $+2^{n-1}$. Como se muestra en el Cuadro 4, con 4 bits el rango es de -8 a $+7$. Note que a diferencia de los métodos de signo y magnitud y complemento a uno, en complemento a dos solo hay una representación para el cero. Otra ventaja del método de complemento a dos es que las operaciones de suma y resta son todavía más fáciles de implementar.

La suma de dos enteros con signo se realiza aplicando la suma binaria (ver [2]) directamente a la representación de los números, descartando cualquier acarreo final. También existe la posibilidad de overflow si el resultado de la suma no se puede representar con el número de bits dados.

Como ejemplos, considere las sumas $5+6$, $5+(-6)$, $-5+6$ y $-5+(-6)$, asumiendo que los números se representan en binario con 4 bits.

$$\begin{array}{rll} & 0101 & (+5) \\ + & 0110 & (+6) \\ \hline & 1011 & (-5) \quad \text{Resultado incorrecto} \end{array}$$

El resultado es incorrecto porque el resultado de la suma $5 + 6$ es 11 que no se puede representar con 4 bits (ver Cuadro 4).

$$\begin{array}{rll} & 0101 & (+5) \\ + & 1010 & (-6) \\ \hline & 1111 & (-1) \quad \text{Resultado correcto} \end{array}$$

No hubo acarreo final y el resultado es correcto.

$$\begin{array}{rll} & 1011 & (-5) \\ + & 0110 & (+6) \\ \hline & 10001 & \quad\quad\quad\quad\quad\quad\quad\quad \text{Acarreo final} \\ & 0001 & (+1) \quad \text{Resultado correcto} \end{array}$$

Hubo acarreo final, éste se descarta para obtener el resultado correcto.

$$\begin{array}{rll} & 1011 & (-5) \\ + & 1010 & (-6) \\ \hline & 10101 & \quad\quad\quad\quad\quad\quad\quad\quad \text{Acarreo final} \\ & 0101 & (+5) \quad \text{Resultado incorrecto} \end{array}$$

Hubo acarreo final, pero el resultado es incorrecto porque -11 no se puede representar con 4 bits.

Al igual que en el método de complemento a uno, se puede detectar si ocurrió un overflow cuando al sumar dos números positivos el resultado es negativo, o cuando al sumar dos enteros negativos el resultado es positivo.

La resta de dos enteros con signo, $A - B$, se calcula escribiendo la resta como $A + (-B)$, encontrando el complemento a dos de B y sumando como se ilustró arriba.

3. Reales

Dependiendo de la posición del punto decimal se pueden considerar dos clases de números reales: de punto fijo y de punto flotante. En punto fijo, la posición del punto decimal dentro del número real está fija. De este modo, los reales de punto fijo tienen un número predeterminado de dígitos en la parte fraccionaria y, por lo general, en la parte entera. Un ejemplo de reales de punto fijo son los números de cantidades monetarias, si se asume que la unidad mínima de operación es el centavo. Por otro lado, en punto flotante el punto decimal puede estar en cualquier parte del número real. Esta sección presenta dos métodos para representar números reales en binario. El primero

es para punto fijo (Subsección 3.1) y el segundo es para punto flotante (Subsección 3.2).

3.1. Punto fijo

Para representar un número real de punto fijo en binario, se reserva un espacio de memoria determinado para almacenar por separado la parte entera y la parte fraccionaria. Si n y m son el espacio en bits de la parte entera y de la parte fraccionaria, respectivamente, entonces el rango de valores que se pueden representar para reales sin signo es entre 0 y $2^n - 1/2^m$. Por otro lado, para reales con signo, asumiendo que la parte entera se guarda en complemento a uno, el rango es entre $-2^{n-1} + 1/2^m$ y $2^{n-1} - 1/2^m$.

Como ejemplo, supongamos que $n = m = 8$. Para reales sin signo el rango es entre 0 (00000000.00000000) y 255.99609375 (11111111.11111111), mientras que para reales con signo el rango es entre -127.99609375 (10000000.11111111) y 127.99609375 (01111111.11111111).

3.2. Standard IEEE 754-2008

El standard para aritmética de punto flotante de la IEEE (Institute of Electrical and Electronic Engineers - Instituto de ingenieros eléctricos y electrónicos), conocido como IEEE 754-2008, define tres formatos para representar números binarios de punto flotante *normalizados*. Un número normalizado consta de una mantisa, que tiene un solo dígito distinto de cero en la parte entera, y de un exponente aplicado a la base del número. Por ejemplo, el binario 11101.11 se normaliza a 1.110111×2^4 recorriendo el punto decimal a la izquierda en 4 posiciones. Para este ejemplo 1.110111 es la mantisa y 4 es el exponente.

Los tres formatos difieren entre sí en la cantidad de bits utilizados para representar el número binario: 32 bits para precisión sencilla, 64 bits para precisión doble y 128 bits para precisión *quad*. Por ejemplo, en precisión sencilla los 32 bits se dividen así: un bit para el signo, 8 bits para el exponente y 23 bits para la mantisa. El bit de signo es 0 si el número es positivo o 1 si es negativo. El exponente se guarda en exceso 127, es decir, al valor del exponente se le suma 127. Por último, solo se almacena la parte fraccionaria de la mantisa porque en base 2 la parte entera de un número normalizado siempre es 1.

La representación de un número de punto flotante en el standard IEEE 754-2008 se encuentra siguiendo esta serie de pasos.

1. Guardar el bit de signo.
2. Convertir el valor absoluto del número a binario.
3. Normalizar el número binario.
4. Guardar el exponente.
5. Guardar la parte fraccionaria de la mantisa justificado a la izquierda y rellenando de ceros a la derecha si es necesario.

Como ejemplos, se mostrará la representación de dos números decimales, uno positivo y el otro negativo.

Ejemplo 1: representar 17.15 en el formato de precisión sencilla binario del standard IEEE 754-2008.

1. El bit de signo es 0.
2. 17.15_{10} en binario es $10001.00\overline{1001}$.
3. El binario normalizado es $1.00010\overline{01001} \times 2^4$.
4. El exponente, 4, en exceso 127 es 131, que convertido a binario es 10000011.
5. La parte fraccionaria de la mantisa, expandida a 23 bits, es 00010010011001100110011.
6. En conclusión, usando precisión sencilla 17.15 se representa en binario como 01000001100010010011001100110011.

Ejemplo 2: representar -118.675 en el formato de precisión sencilla binario del standard IEEE 754-2008.

1. El bit de signo es 1.
2. 118.675_{10} en binario es 1110110.101.
3. El binario normalizado es 1.110110101×2^6.

4. El exponente en exceso 127 es 133, que convertido a binario es 10000101.

5. La parte fraccionaria de la mantisa, rellenando con ceros a la derecha, es 11011010100000000000000.

6. En conclusión, usando precisión sencilla -118.675 se representa en binario como 11000010111011010100000000000000.

Las principales características de la representación binaria dictada por el standard IEEE 754-2008 son:

- Dos ceros.
  - Cero positivo (+0): signo=0, exponente=0, mantisa=0
  - Cero negativo (−0): signo=1, exponente=0, mantisa=0
- Dos infinitos.
  - Infinito positivo ($+\infty$): signo=0, exponente=255, mantisa=0
  - Infinito negativo ($-\infty$): signo=1, exponente=255, mantisa=0
- Dos formas de representar que el resultado no es un número (NaN - not a number).
  - NaN positivo (+NaN): signo=0, exponente=255, mantisa > 0
  - NaN negativo (−NaN): signo=1, exponente=255, mantisa > 0
- Números más grandes: $\pm(1 - 2^{-24}) \times 2^{128} \approx \pm 3.4028235 \times 10^{38}$
- Números más pequeños.
  - Normalizados: $\pm 2^{-126} \approx \pm 1.175494351 \times 10^{-38}$
  - Desnormalizados: $\pm 2^{-149} \approx \pm 1.4012985 \times 10^{-45}$

Se puede obtener información más detallada sobre conceptos de números reales y el standard IEEE 754-2008 consultando las referencias [1] y [4].

Referencias

[1] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[2] Héctor Antonio Villa Martínez. Sistemas de numeración y aritmética binaria. Technical report, Universidad de Sonora, Hermosillo, Sonora, 2008.

[3] Charles H. Roth Jr. *Fundamentos de diseño lógico*. Thomson, México, DF, quinta edition, 2005.

[4] Standard IEEE 754-2008. *http://en.wikipedia.org/wiki/IEEE_754-2008*, Accesado 17-12-2008.