Diego Emilio Dominguez Tableros

4.1 Determina la representación de punto flotante en formato IEEE simple para los siguientes números:

$$2, 30, 31, 32, 33, \frac{23}{4}, \left(\frac{23}{4}\right) \times 2^{100}, \left(\frac{23}{4}\right) \times 2^{-100}, \left(\frac{23}{4}\right) \times 2^{-135}$$

Truncando la mantisa como en el ejemplo de $\frac{1}{10}$, has lo mismo para los números $\frac{1}{5}, \frac{1024}{5}$ y $\frac{1}{10} \times 2^{-140}$. Usa la expansión binaria de $\frac{1}{10}$ para evitar conversiones de decimales a binarios.

Recordemos que la representación de un número de punto flotante esta dado por un **signo** (un bit, 0 si es $+$, 1 si es $-$), un **exponente** (si el exponente es E entonces se almacena la representación binaria de $E + 127$ en el campo) y una **mantisa** (compuesta por 23 bits de la representación del número, obviando el *bit escondido*).

- $2 = (1)_2 = (1)_2 \times 2^1$

0	1000 0000	0000 0000 0000 0000 0000 000

- $30 = (11110)_2 = (1.111)_2 \times 2^4$

0	1000 0011	1110 0000 0000 0000 0000 000

- $31 = (11111)_2 = (1.1111)_2 \times 2^4$

0	1000 0011	1111 0000 0000 0000 0000 000

- $32 = (100000)_2 = (1)_2 \times 2^5$

0	1000 0100	0000 0000 0000 0000 0000 000

- $33 = (100001)_2 = (1.00001)_2 \times 2^5$

0	1000 0100	0000 1000 0000 0000 0000 000

- $\frac{23}{4} = 5.75 = (101.11)_2 = (1.0111)_2 \times 2^2$

0	1000 0001	0111 0000 0000 0000 0000 000

- $\frac{23}{4} \times 2^{100} = (101.11)_2 \times 2^{100} = (1.0111)_2 \times 2^{102}$

0	1110 0101	0111 0000 0000 0000 0000 000

- $\frac{23}{4} \times 2^{-100} = (101.11)_2 \times 2^{-100} = (1.0111)_2 \times 2^{-98}$

0	0001 1101	0111 0000 0000 0000 0000 000

- $\frac{23}{4} \times 2^{135} = (101.11) \times 2^{-135} = (1.0111)_2 \times 2^{-133} = (0.0000010111)_2 \times 2^{-127}$

0	0000 0000	0000 0101 1100 0000 0000 000

Usando $\frac{1}{10} = (0.0001100110011\cdots)_2 = \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \frac{1}{256} + \frac{1}{512} + \frac{0}{1024} + \cdots$

- $\frac{1}{5} = \left(\frac{1}{10}\right) \times 2 = (0.001100110011\cdots)_2 = (1.10011001100\cdots)_2 \times 2^{-3}$

0	0111 1100	1001 1001 1001 1001 1001 100

- $\frac{1024}{5} = \left(\frac{1}{10}\right) \times 2^{11} = (11001100.1100110011\cdots)_2 = (1.100110011\cdots)_2 \times 2^{7}$

0	1000 0110	1001 1001 1001 1001 1001 100

- $\frac{1}{10} \times 2^{-140} = (0.000110011\cdots)_2 \times 2^{-140} = (0.00000000000000011001\cdots) \times 2^{-127}$

0	0000 0000	0000 0000 0000 0000 1100 110

4.2 ¿Cuál es la separación entre 2 y el primer número de punto flotante en formato IEEE simple más grande que 2? ¿Cuál es la separación entre 1024 y el primer número de punto flotante en formato IEEE simple más grande que 1024?

Sabemos que $2 = (1.0)_2 \times 2^1$ y está representado por esquemáticamente como

0	1000 0000	0000 0000 0000 0000 0000 000

Entonces, el siguiente número más grande que 2 corresponde a sumar 1 en el último bit de la mantisa (el correspondiente a 2^{-22}), es decir el número $(10.00000000000000000000001)_2$ representado como

0	1000 0000	0000 0000 0000 0000 0000 001

Este número, en representación decimal es 2.00000023842, por lo tanto, la distancia entre el 2 y este número es

$$2^{-22} = 0.00000023842$$

Por otro lado, sabemos que $1024 = (10000000000)_2 = (1.0)_2 \times 2^{10}$ y está representado por esquemáticamente como

0	1000 1010	0000 0000 0000 0000 0000 000

Entonces, el siguiente número más grande que 2 corresponde a sumar 1 en el último bit de la mantisa (el correspondiente a 2^{-13}), es decir el número $(10000000000.0000000000001)_2$ representado como

0	1000 1010	0000 0000 0000 0000 0000 001

Este número, en representación decimal es 1024.0012207, por lo tanto, la distancia entre el 1024 y este número es

$$2^{-13} = 0.0012207$$

Diego Emilio Dominguez Tableros

5.1 ¿Cuáles son las representaciones, usando el formato binario simple IEEE, del valor redondeado de 1/10 usando cada uno de los cuatro modos de redondeo? ¿Cuáles son para los números $1 + 2^{-25}$ y 2^{130}?

Recordemos que $\frac{1}{10} = (0.0001100110011\cdots)_2$, o en forma normalizada:

$$(1.1001\ 1001\ 1001\ 1001\ 1001\ 100\mathbf{1}\ 1100\cdots)_2 \times 2^{-4}$$

(se pone en negrita el bit correspondiente a 2^{-24}, el que se va a redondear). Ciertamente este número es positivo, por lo que *round down* y *round towards zero* son idénticos. Para *round up* se elige sumar 1 en el bit 24 y por último en el *round to nearest*, como hay más bits hacia adelante con valores no-ceros se tiene el mismo efecto que el *round up*.

Dicho esto, las representaciones son

Round down	0	0111 1011	1001 1001 1001 1001 1001 100
Round towards zero	0	0111 1011	1001 1001 1001 1001 1001 100
Round up	0	0111 1011	1001 1001 1001 1001 1001 101
Round to nearest	0	0111 1011	1001 1001 1001 1001 1001 100

Para $1 + 2^{-25} = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{0}\ 1)_2 \times 2^0$ tenemos el bit 25 (después del punto binario) encendido, por lo tanto, el bit 24 (con el que redondeamos, puesto en negritas) es 0. Los redondedos *down* y *towards zero* son idénticos y equivalen a 1. El redondeo *nearest*, al tener como cero el bit 24, es igual a los dos redondeos anteriores. Finalmente el *up* es distinto pues suma 1 en el bit 23. Las representaciones de cada uno son:

Round down	0	1000 0000	0000 0000 0000 0000 0000 000
Round towards zero	0	1000 0000	0000 0000 0000 0000 0000 000
Round up	0	1000 0000	0000 0000 0000 0000 0000 001
Round to nearest	0	1000 0000	0000 0000 0000 0000 0000 000

Finalmente, para 2^{130}, dado que es mayor que N_{max}, por definición tenemos que el redondeo *down* es el mismo N_{max}, al igual que el *towards zero* pues es una cantidad positiva. El redondeo *up* es igual a ∞. En el caso del redondeo *to nearest* se aplica un caso especial dado por

$$round(2^{130}) = \begin{cases} N_{max} \text{ si } 2^{130} < N_{max} + \dfrac{ulp(N_{max})}{2} \\ \infty \text{ en otro caso} \end{cases}$$

Se utilizó una calculadora para verificar la desigualdad, recordando que $ulp(N_{max})$ es el número de punto flotante inmediatamente mayor que N_{max} y que N_{max} es el número de punto flotante con todos los bits de la mantisa encendidos y elevado al exponente más grande, 127. Esta comprobación puede verse en la Fig. 1. La conclusión es que no se cumple la desigualdad, entonces el redondeo *up* resulta ser infinito.

Dicho esto, las representaciones de los redondeos son:

Round down	0	1111 1110	1111 1111 1111 1111 1111 111
Round towards zero	0	1111 1110	1111 1111 1111 1111 1111 111
Round up	0	1111 1111	0000 0000 0000 0000 0000 000
Round to nearest	0	1111 1111	0000 0000 0000 0000 0000 000

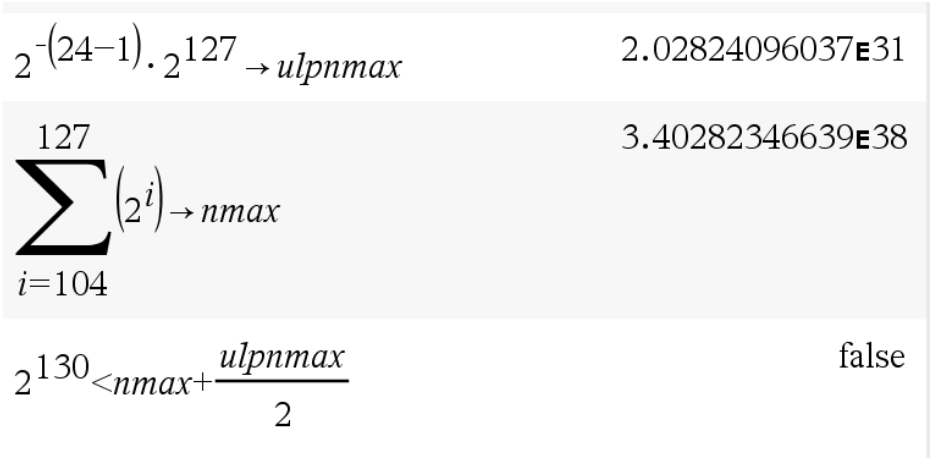

Fig. 1. Comprobación de desigualdad para caso especial de redondeo al mas cercano. Los decimales mostrados son aproximados, obtenidos en un sistema de precisión de 12 decimales.

5.4 ¿Cuál es el abserr(1/10), usando el formato simple IEEE, para cada uno de los cuatro modos de redondeo?

Como 1/10 es positivo, los redondeos *down* y *towards zero* resultan ser iguales. Utilizando las representaciones dadas en el problema **5.1** tenemos que

$$abserr_{D\&TZ}\left(\frac{1}{10}\right) = \left|\frac{1}{10} - round_{D\&TZ}\left(\frac{1}{10}\right)\right| \approx 1.49012 \times 10^{-9}$$

Ahora, para los redondeos *up* y *nearest*, usando las representaciones de sus redondeos idénticos del problema **5.1**, tenemos que

$$abserr_{U\&N}\left(\frac{1}{10}\right) = \left|\frac{1}{10} - round_{U\&N}\left(\frac{1}{10}\right)\right| \approx 5.960464 \times 10^{-9}$$

5.5 Suponga que $x > N_{max}$. ¿Cuál es abserr(x), para cada uno de los cuatro modos de redondeo? Mira cuidadosamente la definición de round(x).

Analogamente, tomando las conclusiones del problema **5.1** podemos decir que los redondeos *down* y *towards zero* resultan ser iguales, por lo que

$$abserr_{D\&TZ}(x) = |x - round_{D\&TZ}(x)| = |x - N_{max}| = N_{max} - x$$

De igual forma, los redondeos *nearest* y *up* son iguales y son iguales a infinito, por lo que

$$abserr_{U\&N}(x) = |x - round_{U\&N}(x)| = |x - \infty|$$

Donde infinito tiene como representación binaria la proporcionada en el ejercicio **5.1**.

5.9 ¿El resultado establecido en el teorema de error de redondeo se sigue manteniendo para $0 < |x| < N_{min}$? Si no, de un x para el cual la conclusión es falsa.

Observemos que si $0 < |x| < N_{min}$ entonces x es un número subnormal. Consideremos entonces $x = \frac{1}{10} \times 2^{-142}$ que claramente es un número subnormal que tiene representación binaria $(0.0001100110011\cdots)_2 \times 2^{-142}$, o bien, ajustando el exponente a -127 (el menor en el formato simple) tenemos que

$$\frac{1}{10} \times 2^{142} = (0.0000\ 0000\ 0000\ 0000\ 0011\ 001\mathbf{1}\ 0011\cdots)_2 \times 2^{-127}$$

Se marca en negrita el bit 24, que es igual a un 1.

Apliquemos redondeo *to nearest*. La conclusión del teorema nos dice que la magnitud del error relativo entre el número original y el redondeado es estrictamente menor que $\varepsilon/2$ con ε el épsilon del sistema simple, es decir $\varepsilon = 2^{-23}$.

El redondeo *to nearest* hace que nuestro $\frac{1}{10} \times 2^{142}$ se almacene como el número:

$$round\left(\frac{1}{10} \times 2^{142}\right) = (0.0000\ 0000\ 0000\ 0000\ 0011\ 010)_2 \times 2^{-127}$$

Que ciertamente es distinto al original. El error relativo entre $\frac{1}{10} \times 2^{142}$ y su redondeo es

$$|\delta| = \frac{\left|\frac{1}{10} \times 2^{142} - round\left(\frac{1}{10} \times 2^{142}\right)\right|}{\left|\frac{1}{10} \times 2^{142}\right|} \approx 0.15625$$

De lo cual podemos concluir que la relación $|\delta| < \frac{\varepsilon}{2} \approx 5.9605 \times 10^{-8}$ es falsa, por lo tanto, **el teorema no se verifica para este valor**. Se muestra en la Fig. 2 el resultado a través del uso de una calculadora como comprobación.

$\left(\frac{1}{2^{19}}+\frac{1}{2^{20}}+\frac{1}{2^{22}}\right)\cdot 2^{-127} \to r$	1.82168800362E-44
$\frac{1}{10}\cdot 2^{-142} \to x$	1.79366203434E-44
$2^{-23} \to eps$	0.000000119209
$\left\|\frac{r-x}{x}\right\| < \frac{eps}{2}$	false

Fig. 2. Comprobación de desigualdad para excepción en el teorema del error del redondeo. Los decimales mostrados son aproximados, obtenidos en un sistema de precisión de 12 decimales.

6.1 Muestra que la regla IEEE del redondeado aritmético correcto inmediatamente garantiza que las respuestas a las Preguntas 6.1 a la 6.3 deben ser que sí. Muestra además que no se necesita redondeado, es decir, que el resultado exacto es un número de punto flotante a excepción de un caso específico, ¿Cuál es ese caso?

Demostremos que las preguntas son correctas aplicando las propiedades del correcto redondeo aritmético bajo el IEEE.

- ¿ $\mathbf{1} \otimes \boldsymbol{x} = \boldsymbol{x}$?
  $x \otimes 1 = round(x \times 1) = round(x) = x$

- ¿ $\boldsymbol{x} \oslash \boldsymbol{x} = \mathbf{1}$?
  $x \oslash x = round(x/x) = round(1) = 1$

- ¿ $\mathbf{0.5} \otimes \boldsymbol{x} = \boldsymbol{x} \oslash \mathbf{2}$?
  $0.5 \otimes x = round(0.5 \times x) = round(0.5 \times x) = 0.5 \times x$
  $x \oslash 2 = round(x/2) = round(0.5 \times 2) = 0.5 \times x$

6.2 Suponga que el formato de destino es IEEE simple y el modo de redondeo es redondear al más cercano. ¿Qué son $64 \oplus 2^{20}$, $64 \oplus 2^{-20}$, $32 \oplus 2^{-20}$, $16 \oplus 2^{-20}$, $8 \oplus 2^{-20}$? Da tus respuestas en binario, no en decimal. ¿Cuáles son los resultado si el modo de redondeo es hacia arriba?

- $64 \oplus 2^{20} = round(64 + 2^{20})$

Veamos que $64 = (1)_2 \times 2^6 = (1000000)_2$ y $2^{20} = (1000000000000000)_2$, por tanto

$64 + 2^{20} = (10000000000001000000)_2 = (1.0000\ 0000\ 0000\ 0100\ 0000\ 000)_2 \times 2^{20}$
entonces $round(64 + 2^{20}) = (1.0000\ 0000\ 0000\ 0100\ 0000\ 000)_2 \times 2^{20} = 64 + 2^{20}$

Por lo tanto, concluimos $64 \oplus 2^{20} = 64 + 2^{20}$.

- $64 \oplus 2^{-20} = round(64 + 2^{-20})$

Veamos que $64 = (1)_2 \times 2^6 = (1000000)_2$ y $2^{-20} = (0.0000\ 0000\ 0000\ 0000\ 0001)_2$, por tanto

$64 + 2^{-20} = (1000000.0000\ 0000\ 0000\ 0000\ 0001)_2$
$= (1.0000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{0}\ 01)_2 \times 2^6$
entonces $round(64 + 2^{-20}) = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000)_2 \times 2^6 = 64$

Por lo tanto, concluimos $64 \oplus 2^{-20} = 64$.

- $32 \oplus 2^{-20} = round(32 + 2^{-20})$

Veamos que $32 = (1)_2 \times 2^5 = (100000)_2$ y $2^{-20} = (0.0000\ 0000\ 0000\ 0000\ 0001)_2$, por tanto

$32 + 2^{-20} = (100000.0000\ 0000\ 0000\ 0000\ 0001)_2$
$= (1.0000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{0}\ 1)_2 \times 2^5$
entonces $round(32 + 2^{-20}) = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000)_2 \times 2^5 = 32$

Por lo tanto, concluimos $32 \oplus 2^{-20} = 32$.

- $16 \oplus 2^{-20} = round(32 + 2^{-20})$

Veamos que $16 = (1)_2 \times 2^4 = (10000)_2$ y $2^{-20} = (0.0000\ 0000\ 0000\ 0000\ 0001)_2$, por tanto

$16 + 2^{-20} = (10000.0000\ 0000\ 0000\ 0000\ 0001)_2$
$= (1.0000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{1})_2 \times 2^5$
entonces $round(32 + 2^{-20}) = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000)_2 \times 2^5 = 16$

Observemos que se aplicó redondeo al **bit 24** con valor 1, pero como todos los siguientes bits son cero (de hecho, no hay), el redondeo al más cercano es el redondeo hacia abajo, obteniendo el mismo número. Por lo tanto, concluimos $16 \oplus 2^{-20} = 16$.

- $8 \oplus 2^{-20} = round(32 + 2^{-20})$

Veamos que $8 = (1)_2 \times 2^3 = (1000)_2$ y $2^{-20} = (0.0000\ 0000\ 0000\ 0000\ 0001)_2$, por tanto

$$32 + 2^{-20} = (1000.0000\ 0000\ 0000\ 0000\ 0001)_2$$
$$= (1.0000\ 0000\ 0000\ 0000\ 0000\ 001\mathbf{0})_2 \times 2^3$$

entonces $round(32 + 2^{-20}) = (1.0000\ 0000\ 0000\ 0000\ 0000\ 001)_2 \times 2^3 = 8 + 2^{-20}$

Aquí caben los 23 bits perfectamente. Por lo tanto, concluimos $8 \oplus 2^{-20} = 8 + 2^{-20}$.

6.4 ¿Cuál es el mayor número de punto flotante x para el cual $1 \oplus x$ es exactamente 1, asumiendo que el formato de destino es IEEE simple y el modo de redondeo es al más cercano? ¿Cuál es si el formato de destino es IEEE doble?

Recordemos que la definición de la épsilon del sistema es la distancia entre 1 y el siguiente mayor número de punto flotante. En el caso del sistema simple tenemos $\varepsilon = 2^{-(24-1)} = 2^{-23}$. Esto tiene mucho sentido por que

$$1 = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000)_2$$
$$2^{-23} = (0.0000\ 0000\ 0000\ 0000\ 0000\ 001)_2$$
$$1 + 2^{-23} = (1.0000\ 0000\ 0000\ 0000\ 0000\ 001)_2$$

Que no necesita ser redondeado pues podemos almacenar los 23 bits. Ciertamente el número $1 + \varepsilon = 1 + 2^{-23} \neq 1$.

Bien, consideremos entonces el número $\frac{\varepsilon}{2} = \frac{2^{-23}}{2} = 2^{-24}$ y veamos que en este caso

$$1 = (1.0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2$$
$$2^{-24} = (0.0000\ 0000\ 0000\ 0000\ 0000\ 0001)_2$$
$$1 + 2^{-23} = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000\mathbf{1})_2$$

Pero por el redondeo al más cercano (tomando el bit 24 igual a 1, señalado en negritas) que equivale a redondear hacia abajo obtendremos como resultado

$$round(1 + 2^{-23}) = (1.0000\ 0000\ 0000\ 0000\ 0000\ 000)_2 = (1)_2 = 1$$

Obteniendo que $1 + \frac{\varepsilon}{2} = 1 + 2^{-24} = 1$. Y este resulta ser el número más grande que cumple con esta condición pues cualquier número menor tiene más bits a la izquierda, provocando el mismo resultado y cualquier número mayor tiene mas bits a la derecha, mismos que no sufren el efecto del redondeado y no verifican la igualdad.

Por lo tanto, **el mayor número de punto flotante que verifica $1 \oplus x = 1$ en este sistema es**

$$\boldsymbol{\frac{\varepsilon}{2} = 2^{-24}}$$

Para el formato doble podemos dar un argumento completamente análogo solo que tomando en cuenta que la épsilon en este sistema es $\varepsilon = 2^{-52}$.

Por lo tanto, **el mayor número de punto flotante que verifica $1 \oplus x = 1$ en este sistema es**

$$\boldsymbol{\frac{\varepsilon}{2} = 2^{-53}}$$

6.10 Considere la operación $x + y$, donde $x = 1.0$ y $y = (1.0 \cdots 0)_2 \times 2^{-24}$, es decir y tiene 22 ceros a la derecha del punto binario y un 1 al final. ¿Cuál es el resultado redondeado correcto asumiendo que se esta usando redondeo al más cercano? ¿Cuál es el computado si se usa solo un digito de guardia? ¿Cuál es el computado si se usan dos dígitos de guardia? ¿Cuál es si se usan dos de guardia y un bit *sticky*?

Se representaran todos los bits necesarios a la izquierda de una barra (que simboliza los 23 bits de la mantisa) para identificar correctamente cada caso. Todos los números son binarios.

- **Resultado redondeado correcto**

  $1 = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 0\ 0000\ 0000\ 0000\ 0000\ 0000\ 000$
  $y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 1\ 0000\ 0000\ 0000\ 0000\ 0000\ 001$
  $1 + y = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid \mathbf{1}\ 0000\ 0000\ 0000\ 0000\ 0000\ 001$
  $round(1 + y) = 1.0000\ 0000\ 0000\ 0000\ 0000\ 001$

En este caso el redondeo al más cercano se usó con el caso en que se tiene más bits diferentes de cero más allá del bit 24 (marcado en negrita), en este caso se suma 1 al bit 24, obteniendo el resultado mostrado.

Conclusión: usando el redondeado correcto se obtiene $(\mathbf{1.0000\ 0000\ 0000\ 0000\ 0000\ 001})_\mathbf{2}$

- **Usando un dígito de guardia**

  $1 = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 0$
  $y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 1$
  $1 + y = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid \mathbf{1}$
  $round(1 + y) = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000$

En este caso el redondeo al más cercano se usó con el caso en que no se tienen más bits diferentes de cero más allá del bit 24 (marcado en negrita), en este caso se trunca la expansión como un redondeo hacia abajo, obteniendo el resultado mostrado.

Conclusión: usando un digito de guardia se obtiene $(\mathbf{1.0000\ 0000\ 0000\ 0000\ 0000\ 000})_\mathbf{2}$

- **Usando dos dígitos de guardia**

  $1 = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 00$
  $y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 10$
  $1 + y = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid \mathbf{1}0$
  $round(1 + y) = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000$

En este caso el redondeo al más cercano se usó con el caso en que no se tienen más bits diferentes de cero más allá del bit 24 (marcado en negrita), en este caso se trunca la expansión como un redondeo hacia abajo, obteniendo el resultado mostrado.

Conclusión: usando un digito de guardia se obtiene $(\mathbf{1.0000\ 0000\ 0000\ 0000\ 0000\ 000})_\mathbf{2}$

- **Usando dos dígitos de guardia y un *sticky***

$$1 = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 00\ 0$$
$$y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid 10\ 1$$
$$1 + y = 1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \mid \mathbf{1}0\ 1$$
$$round(1 + y) = 1.0000\ 0000\ 0000\ 0000\ 0000\ 001$$

En este caso el redondeo al más cercano se usó con el caso en que se tiene más bits diferentes de cero más allá del bit 24 (marcado en negrita), en este caso se suma 1 al bit 24, obteniendo el resultado mostrado.

Conclusión: usando un digito de guardia se obtiene $(\mathbf{1.0000\ 0000\ 0000\ 0000\ 0000\ 001})_{\mathbf{2}}$

Como conclusión general podemos decir que el uso del bit sticky proporciona una forma de ahorrarnos indefinidos dígitos de guardia como en el primer caso y seguir obteniendo el número correcto (el del primer caso). Esto es sorprendente pues **con este bit sticky ya no es necesario el uso de muchos dígitos de guardia**, mismos que no aseguran obtener el resultado apropiado como fueron los casos dos y tres que, a pesar de usar dígitos de guardia, se obtuvo un resultado erróneo al ser distinto del verdadero mostrado en el primer caso.