Módulo 06

Norma IEEE-754 (Pt. 1)

Organización de Computadoras
Depto. Cs. e Ing. de la Comp.
Universidad Nacional del Sur

Copyright

- Copyright © **2011-2024** A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU** Free Documentation License, Versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página **http://www.gnu.org/copyleft/fdl.html**

- La versión transparente de este documento puede ser obtenida de la siguiente dirección:

http://cs.uns.edu.ar/~ags/teaching

Contenidos

- Teoría de redondeo
- Redondeos óptimos y simétricos
- Norma **IEEE-754**
- Formatos contemplados
- Concepto de denormal
- Implementación de las operaciones básicas
- Redondeos en la norma
- Consideraciones al implementar en **HW**

Precisión de la mantisa

- En ocasiones el resultado de una operación **FLP** puede exceder los **p** dígitos de precisión reservados para la mantisa
  - Por ejemplo, los **p + 1** dígitos que resultan de sumar cualesquiera dos números normalizados de **p** dígitos
  - También, al multiplicar dos mantisas, normalizadas o no, con **p** dígitos de precisión se obtiene un resultado de **2p** dígitos
- ¿Qué debemos hacer con estos dígitos adicionales? ¿Descartarlos?

Truncado

- Para el ejemplo de la suma, el overflow virtual se puede compensar corriendo la mantisa a la derecha una posición
  - Esto usualmente tiene como efecto colateral eliminar el último bit del resultado (el bit menos significativo), de manera independiente de su valor
- El mecanismo descripto se denomina truncado
  - Por ejemplo, para **X = 0.130581** en el marco de una mantisa fraccionaria con base **b = 10**, el truncado a **4** dígitos de precisión resulta **0.1305**

Redondeo

- Para el ejemplo de la multiplicación, una forma razonable de descartar los **p** dígitos menos significativos de los **2p** dígitos del resultado consiste en analizar el bit más significativo pero entre los que serán descartados:
  - Si ese bit es **1**, entonces sumamos **1 LSB** al resultado preliminar formado con los **p** dígitos del resultado que han de ser conservados
  - Caso contrario, si ese bit es **0**, entonces el resultado preliminar es el definitivo

Redondeo

- Este mecanismo que tiene en cuenta el valor de los dígitos descartados se denomina redondeo
  - Por caso, para **X = 0.12345678** con las mismas condiciones que antes, el redondeo a **2** dígitos de precisión resulta **0.12** (el resultado coincide con el obtenido al truncar), pero en cambio el redondeo a **5** dígitos resulta **0.12346**
  - De forma análoga, para **Y = 0.110101** en el marco ahora de una base **b = 2**, el redondeo a **2** dígitos resulta **0.11**, pero en cambio el redondeo a **5** dígitos resulta **0.11011**

Redondeo

- Cuando el bit más significativo descartado es **1**, el resultado está en la mitad o pasada la mitad. Caso contrario al ser **0**, el resultado está necesariamente por debajo de la mitad
  - En otras palabras, el redondeo siempre conduce al número máquina de **p** dígitos de precisión más próximo al resultado obtenido de **2p** dígitos
- En consecuencia, el máximo error que se puede cometer está acotado por la expresión:

$$\mathbf{Err = b^{-p} / 2 = \tfrac{1}{2}\ LSB}$$

Redondeo

- Cabe señalar que independientemente de la base adoptada, sabemos que los dígitos serán representados eventualmente en base **2**
- Por esta razón, el umbral se puede resolver inspeccionando sólo el bit más significativo entre los dígitos a ser descartados, más allá de la base adoptada
  - Esto se debe a que ese bit será necesariamente un **1** al representar valores por encima de **b / 2** (porque la base adoptada debe ser potencia de **2**)

Máximo error

- Para analizar el máximo error que se puede cometer bajo cada esquema de redondeo hace falta detectar el escenario en el que peor se desempeña ese redondeo:
  - Para el truncado, el peor caso se verifica cuando el resultado se aproxima a un número máquina pero sin llegar a éste; en ese caso, la magnitud descartada representará a lo sumo **1 LSB**
  - Para el redondeo, la situación mejora, ya que el peor casos se verifica cuando el resultado equidista de dos números máquina, descartando en ese caso **½ LSB**

Teoría de redondeo

- Hasta aquí el análisis naturalmente sólo contempló valores absolutos; una descripción algebraica más rigurosa debe distinguir números positivos y negativos, a los efectos de estandarizar los métodos existentes
- Recordemos que la función de redondeo se define formalmente como el mapeo $\rho : \mathbb{R} \rightarrow \mathbb{M}$ tal que para todo $a, b \in \mathbb{R}$ se verifica que:

$$\rho(a) \leq \rho(b), \text{ toda vez que } a \leq b$$

Redondeo óptimo

- Una función de redondeo ρ se denomina óptima sí, y sólo sí, para todo $a \in \mathbb{M}$ se verifica que $\rho(a) = a$
- Como corolario de esta definición, sabemos que un redondeo óptimo asegura que para todo $a \in \mathbb{R}$, con m_1, m_2 dos números consecutivos en $\mathbb{M}$ tales que $m_1 < a < m_2$, se verificará necesariamente que $\rho(a) = m_1$ ó $\rho(a) = m_2$

Redondeo óptimo

- Demostración:
  - Asumamos que $\rho(a) = m'$, con $m' \notin [m_1, m_2]$
  - Como $m_1 < a < m_2$, por definición de ρ obtenemos que $\rho(m_1) < \rho(a) < \rho(m_2)$
  - Como ρ es óptimo, $\rho(m_1) = m_1$ y $\rho(m_2) = m_2$, entonces $m_1 < m' < m_2$, absurdo, que provino de asumir que existía tal $m' \notin [m_1, m_2]$
  - Luego, $m' \in [m_1, m_2]$, lo que implica que necesariamente $m' = m_1$ ó $m' = m_2$

Terminología

- Diremos que un redondeo ρ es dirigido hacia arriba sí, y sólo sí, para todo $a \in \mathbb{R}$ se verifica que $\rho(a) \geq a$
- De manera análoga, diremos que un redondeo es dirigido hacia abajo sí, y sólo sí, para todo $a \in \mathbb{R}$ se verifica que $\rho(a) \leq a$
- Finalmente, un redondeo se dice simétrico sí, y sólo sí, para todo $a \in \mathbb{R}$ se verifica que $-\rho(-a) = \rho(a)$

Redondeos óptimos

- El redondeo óptimo dirigido hacia abajo $\nabla : \mathbb{R} \rightarrow \mathbb{M}$ se define como:

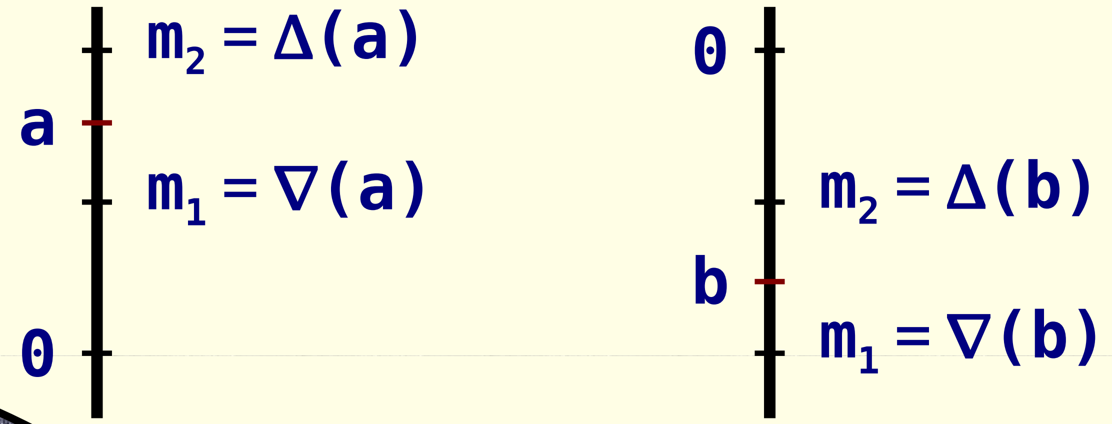
$$\nabla(\mathbf{a}) = \mathbf{máx}(\{\mathbf{m} \in \mathbb{M} \mid \mathbf{m} \leq \mathbf{a}\})$$

- El redondeo óptimo dirigido hacia arriba $\Delta : \mathbb{R} \rightarrow \mathbb{M}$ se define como:

$$\Delta(\mathbf{a}) = \mathbf{mín}(\{\mathbf{m} \in \mathbb{M} \mid \mathbf{m} \geq \mathbf{a}\})$$

Redondeos óptimos

- En cierta forma, el redondeo óptimo hacia arriba redondea el resultado alejándose del cero en los positivos y acercándose en los negativos
- El redondeo óptimo hacia abajo se comporta manera complementaria, acercándose al cero en los positivos y alejándose en los negativos



Redondeos simétricos

- A partir de estos dos redondeos óptimos es posible definir tres redondeos simétricos que presentan un comportamiento interesante:
  - <u>Truncado:</u> redondea positivos y negativos siempre hacia el cero
  - <u>Aumentación:</u> redondea positivos y negativos siempre alejándose del cero
  - <u>Proximidad:</u> redondea siempre hacia el número máquina más próximo, eligiendo el de mayor magnitud en el caso límite, al equidistar entre dos números máquina consecutivos

Redondeos simétricos

- Truncado: ($\mathbf{T} : \mathbb{R} \to \mathbb{M}$)

$$\mathbf{T(a)} = \begin{cases} \nabla(\mathbf{a}) \text{ cuando } \mathbf{a} \geq 0 \\ \Delta(\mathbf{a}) \text{ cuando } \mathbf{a} < 0 \end{cases}$$

- Aumentación: ($\mathbf{A} : \mathbb{R} \to \mathbb{M}$)

$$\mathbf{A(a)} = \begin{cases} \Delta(\mathbf{a}) \text{ cuando } \mathbf{a} \geq 0 \\ \nabla(\mathbf{a}) \text{ cuando } \mathbf{a} < 0 \end{cases}$$

Redondeos simétricos

- Proximidad biased: ($\mathbf{P} : \mathbb{R} \rightarrow \mathbb{M}$)

$$\mathbf{P(a)} = \begin{cases} \nabla(\mathbf{a}) & \text{cuando } \nabla(\mathbf{a}) \leq \mathbf{a} < \mathbf{umbral} \\ \Delta(\mathbf{a}) & \text{cuando } \mathbf{umbral} < \mathbf{a} \leq \Delta(\mathbf{a}) \\ \nabla(\mathbf{a}) & \text{cuando } \mathbf{a} = \mathbf{umbral} \text{ y } \mathbf{a} < \mathbf{0} \\ \Delta(\mathbf{a}) & \text{cuando } \mathbf{a} = \mathbf{umbral} \text{ y } \mathbf{a} \geq \mathbf{0} \end{cases}$$

$$\text{con } \mathbf{umbral} = \frac{\nabla(\mathbf{a}) + \Delta(\mathbf{a})}{\mathbf{2}}$$

Redondeos simétricos

- La mayoría de las implementaciones en hardware hacen uso de **T** o de **P**
- El diseñador del sistema de punto flotante debe tomar decisiones que afectan tanto la velocidad de cómputo como a la exactitud
  - Se puede demostrar que la mejor precisión se logra a través del uso de bases pequeñas y mecanismos de redondeo sofisticados
  - Por contraposición, la velocidad de cómputo se incrementa usando grandes bases y mecanismos de redondeo simples como **T**

Biased vs. unbiased

- **P** presenta un comportamiento sesgado (biased) porque cuando el número a redondear equidista se prefiere aumentar su magnitud
  - Es decir, existe un sesgo en la manera de resolver el caso límite entre dos números máquina
- La norma **IEEE-754** implementa un redondeo por proximidad unbiased, prefiriendo en el caso límite los resultados pares
  - Esto disminuye el error acumulado producto de sucesivos redondeos

Biased vs. unbiased

- Proximidad unbiased: ($\mathbf{P} : \mathbb{R} \rightarrow \mathbb{M}$)

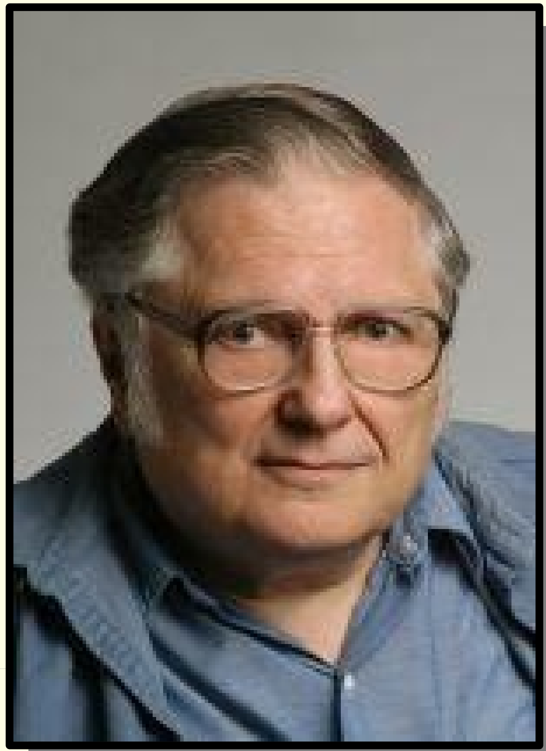
$$
\mathbf{P(a)} = \begin{cases}
\nabla\mathbf{(a)} \text{ cuando } \nabla\mathbf{(a)} \leq \mathbf{a} < \mathbf{umbral} \\
\Delta\mathbf{(a)} \text{ cuando } \mathbf{umbral} < \mathbf{a} \leq \Delta\mathbf{(a)} \\
\Delta\mathbf{(a)} \text{ cuando } \mathbf{a} = \mathbf{umbral},\ \mathbf{a} < \mathbf{0} \text{ y } \mathbf{p_0} = \mathbf{0} \\
\nabla\mathbf{(a)} \text{ cuando } \mathbf{a} = \mathbf{umbral} \text{ y } \mathbf{a} < \mathbf{0} \text{ y } \mathbf{p_0} = \mathbf{1} \\
\nabla\mathbf{(a)} \text{ cuando } \mathbf{a} = \mathbf{umbral} \text{ y } \mathbf{a} \geq \mathbf{0} \text{ y } \mathbf{p_0} = \mathbf{0} \\
\Delta\mathbf{(a)} \text{ cuando } \mathbf{a} = \mathbf{umbral} \text{ y } \mathbf{a} \geq \mathbf{0} \text{ y } \mathbf{p_0} = \mathbf{1}
\end{cases}
$$

$\mathbf{p_0}$ denota al bit menos significativo entre los conservados

IEEE-754

- La norma **IEEE-754** de 1985 y su revisión del 2019 son el estándar de facto a la hora de implementar en hardware una representación de punto flotante
- Fue escrito en su mayor parte por el profesor W. M. Kahan

Turing Award
1989



Formatos de representación

- La norma **IEEE-754** contempla en principio dos formatos para los números **FLP**:
  - Precisión simple (**32 bits**):

0	1 … 8	9 … 31
sg	exp (8)	mantisa (23)

  - Precisión doble (**64 bits**):

0	1 … 11	12 … 63
sg	exp (11)	mantisa (52)

Formatos de representación

- Características:
  - En todos los casos se hace uso de una base **b = 2**
  - La mantisa es representada usando la notación **SM**
  - La norma sanciona una operación normalizada, por lo que la mantisa contará con un bit adicional de precisión (el denominado hidden bit)
  - El exponente es representada usando la notación exceso, que para el caso del formato precisión simple corresponde a **exceso-127**

Formatos de representación

- En síntesis, los bits almacenados por caso en el formato precisión simple se deben interpretar de la siguiente manera:

$$(-1)^{sg} \times \underbrace{1.}_{\text{hidden bit}}\text{mantisa} \times 2^{(exp-127)}$$

Análisis

- Analicemos la semántica otorgada por la norma a las combinaciones de los distintos valores posibles para los campos mantisa y exponente:

	exp = 0	**0 < exp < 255**	**exp = 255**
mantisa = 0	**0**	potencias de **2**	±∞
mantisa ≠ 0	no normalizado (denormal)	números comunes	**NaN** (not a number)

Análisis

Real Number and NaN Encodings For 32-Bit Floating-Point Format

S	E	F			S	E	F
1	0	0	–0	+0	0	0	0
1	0	0.XXX[2]	–Denormalized Finite	+Denormalized Finite	0	0	0.XXX[2]
1	1...254	Any Value	–Normalized Finite	+Normalized Finite	0	1...254	Any Value
1	255	0	–∞	+∞	0	255	0
X[1]	255	1.0XX[2]	–SNaN	+SNaN	X[1]	255	1.0XX[2]
X[1]	255	1.1XX	–QNaN	+QNaN	X[1]	255	1.1XX

Análisis

- En síntesis, los formatos de la normal precisión simple y precisión doble brindan el siguiente rango de representación:

	denormalizado	normalizado	rango decimal
precisión simple	$\pm[2^{-149}, (1-2^{-23})\times2^{-126}]$	$\pm[2^{-126}, (2-2^{-23})\times2^{127}]$	$\pm[\sim10^{-44.85}, \sim10^{38.53}]$
precisión doble	$\pm[2^{-1074}, (1-2^{-52})\times2^{-1022}]$	$\pm[2^{-1022}, (2-2^{-52})\times2^{1023}]$	$\pm[\sim10^{-323.3}, \sim10^{308.3}]$

Normalización en la norma

- La norma adopta una mantisa peculiar, con exactamente un dígito a la izquierda del punto
- La norma también estipula que opera sólo con mantisas normalizadas, por lo que debemos adecuar su definición a este contexto:
  - Diremos que una mantisa está normalizada cuando cuenta con exactamente un **1** a la izquierda de punto
  - Por caso, la mantisa **101.110** no está normalizada, pero la mantisa **1.01110** si lo está

Conversión hacia la norma

- El procedimiento para convertir un número en notación **FXP** a la norma es bastante directo, abarca los siguientes pasos:
  - Convertir a binario la parte entera usando cualquiera de los métodos vistos
  - Convertir a binario la parte decimal, usando nuevamente cualquiera de los métodos vistos
  - Normalizar el resultado obtenido anteriormente, ajustando el exponente de manera acorde
  - Finalmente, expresar la mantisa y el exponente usando alguno de los formatos de la norma

Ejemplo

- Convertir a la norma el valor $(10.666015625)_{10}$
  - Parte entera: $(10)_{10} = (1010)_2$
  - Parte fraccionaria: $(.666015625)_{10} = (.101010101)_2$
  - Combinando los resultados anteriores resulta: $1010.101010101 \times 2^0$
  - Normalizando, nos queda: $1.010101010101 \times 2^3$
  - Exponente en exceso: $3 + 127 = 130 = 1000010$

0	1 … 8	9 … 31
0	10000010	01010101010100000000000

el hidden bit no se almacena

Ejemplo

- Convertir a la norma el valor $\mathbf{(-0.171875)_{10}}$
  - Parte entera: $\mathbf{(0)_{10} = (0)_2}$
  - Parte fraccionaria: $\mathbf{(.171875)_{10} = (.001011)_2}$
  - Combinando los resultados anteriores resulta: $\mathbf{0.001011 \times 2^{0}}$
  - Normalizando, nos queda: $\mathbf{1.011 \times 2^{-3}}$
  - Exponente en exceso: $\mathbf{-3 + 127 = 124 = 01111100}$

0	1 … 8	9 … 31
1	01111100	01100000000000000000000

Conversión entre FXP y FLP

- Consideremos el siguiente fragmento código **C**:

```
int i; float f;
f = (float) i;
i = (int) f;
```

- ¿Son redundantes las conversiones explícitas de tipos usadas en este fragmento de código?
  - ¿Pierdo precisión al convertir un **int** en un **float**?
  - ¿Y al convertir un **float** en un **int**?

Denormals

- Existen combinación de valores para la mantisa y el exponente a los que se les reservó una interpretación especial
- Los números cuyo exponente es cero pero no su mantisa, se denominan denormals
  - Los denormals admiten mantisas sin normalizar, por lo que permiten representar valores más pequeños que al utilizar mantisas normalizadas
  - Nótese que los denormals, al no estar normalizados, no cuentan con el hidden bit

Ejemplo

- Número positivo normalizado más pequeño:

0	1 … 8	9 … 31
0	00000001	00000000000000000000000

$$(-1)^{0} \times 1.00000000000000000000000 \times 2^{-126}$$

- Próximo positivo todavía más pequeño (se trata de un denormal):

0	1 … 8	9 … 31
0	00000000	11111111111111111111111

$$(-1)^{0} \times 0.11111111111111111111111 \times 2^{-126}$$

Normalizados vs. denormals

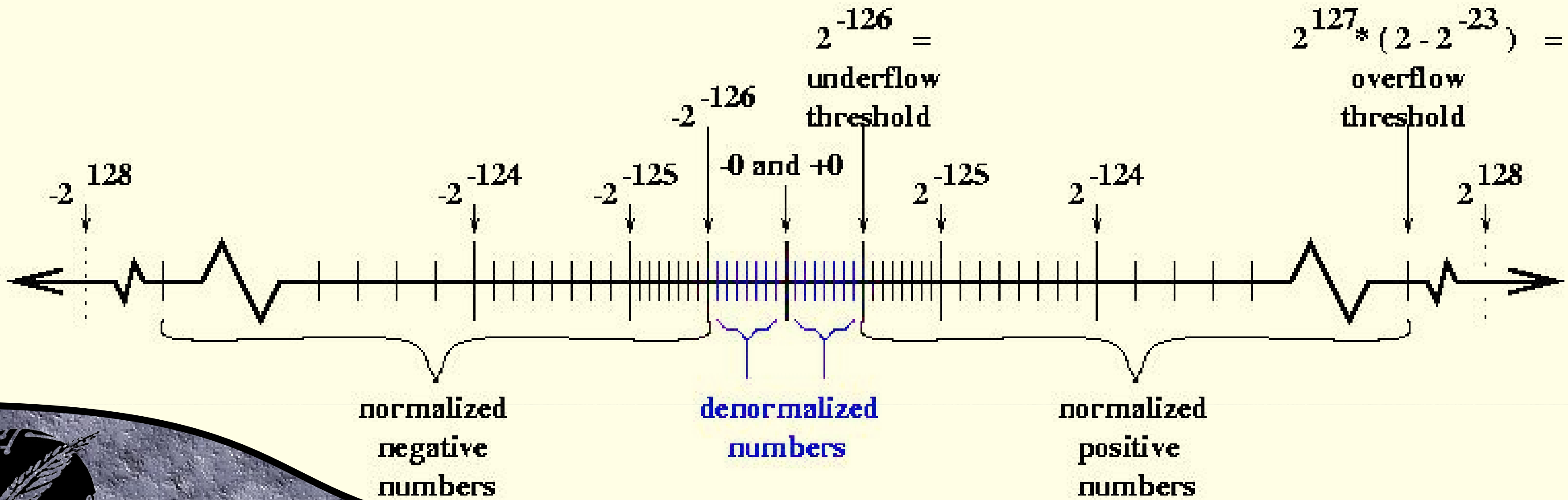
- El exponente asociado a un denormal es **-126**, más allá de que el valor codificado es cero
  - Es decir, los denormals no codifican su exponente en exceso, ya que en exceso, **0** representaría **-127**
- ¿A qué se debe esta aparente inconsistencia?
  - La idea es permitir representar números próximos a los números ordinarios, por eso el exponente representado es el mismo (esto es, **-126**)
  - Naturalmente, el exponente almacenado difiere, para poder distinguir normalizados de no normalizados

Ejemplo

- El número positivo más pequeño de todos (se trata de un denormal):

0	1 … 8	9 … 31
0	00000000	00000000000000000000001

$$(-1)^0 \times 0.00000000000000000000001 \times 2^{-126} = 2^{-149}$$



¿Preguntas?