Tema 1 - Representación de números de punto flotante

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?
2. Representación de números enteros
   - Signo y Magnitud
   - Exceso a 2^{n-1}
   - Estándar IEEE 754
     - Precisión simple
     - Precisión doble
     - Casos especiales del IEEE 754

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?
2. Representación de números enteros
   - Signo y Magnitud
   - Exceso a 2^{n-1}
   - Estándar IEEE 754
     - Precisión simple
     - Precisión doble
     - Casos especiales del IEEE 754
3. Propagación de errores
   - Modelos para el desastre
   - Ejercicio 1

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?
2. Representación de números enteros
   - Signo y Magnitud
   - Exceso a 2^{n-1}
   - Estándar IEEE 754
     - Precisión simple
     - Precisión doble
     - Casos especiales del IEEE 754
3. Propagación de errores
   - Modelos para el desastre
   - Ejercicio 1

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
```

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
-2.7755575615628914e-17
```

¿Qué es lo que pasa?

Realiza la siguiente operación en Python:

```
>>> 1 - 0.9 - 0.1
-2.7755575615628914e-17
```

Pregunta

¿Por qué el resultado que esperamos no es cero?

Considera el siguiente código

```
1 suma = 1
2 for i in range(1,1000001):
3     suma = suma+0.0000001
4 print suma
```

¿Cuál es el resultado que nos devuelve el programa?

Considera el siguiente código

```
1 suma = 1
2 for i in range(1,1000001):
3     suma = suma+0.0000001
4 print suma
```

¿Cuál es el resultado que nos devuelve el programa?

1.10000000006

Considera el siguiente código

```
1 suma = 1
2 for i in range(1,1000001):
3     suma = suma+0.0000001
4 print suma
```

¿Cuál es el resultado que nos devuelve el programa?

1.10000000006

Pregunta

¿En dónde está el error?

Para dar respuesta a las preguntas, tendremos que ahondar en la manera en que se representan los números en una computadora.

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?
2. Representación de números enteros
   - Signo y Magnitud
   - Exceso a 2^{n-1}
   - Estándar IEEE 754
     - Precisión simple
     - Precisión doble
     - Casos especiales del IEEE 754
3. Propagación de errores
   - Modelos para el desastre
   - Ejercicio 1

Signo y Magnitud

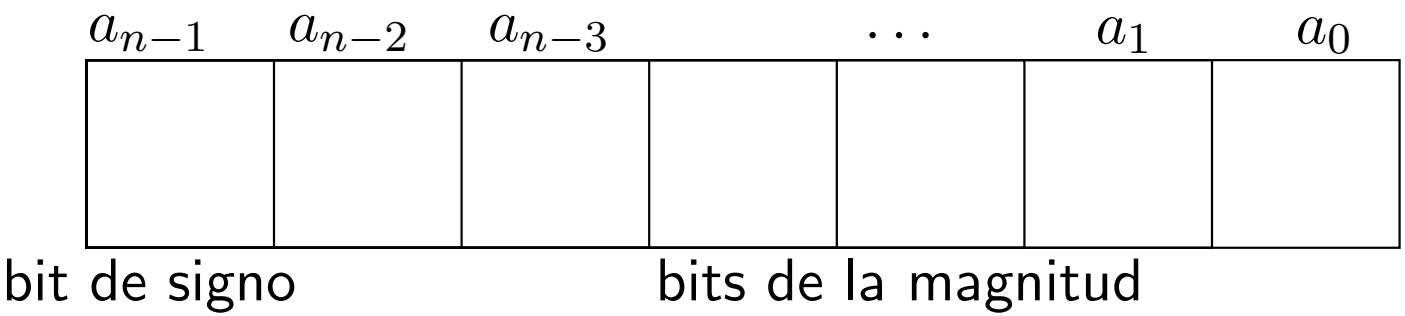
En la representación de un número entero en *Signo y Magnitud*, de los **n** bits de dicha representación, el más significativo representa el signo de la misma representación, por lo que se llama *bit de signo*.

El resto de los bits representan la magnitud.

$$N_{SM} = a_{n-1}a_{n-2}\ldots a_1a_0$$

donde a_{n-1} representa el signo del número y el resto de los bits, la magnitud del mismo:

Representación de Signo y Magnitud



Cuando se quiera representar el signo del número negativo, el bit de signo valdrá $\mathbf{1}$, siendo $\mathbf{0}$ cuando el número es positivo.

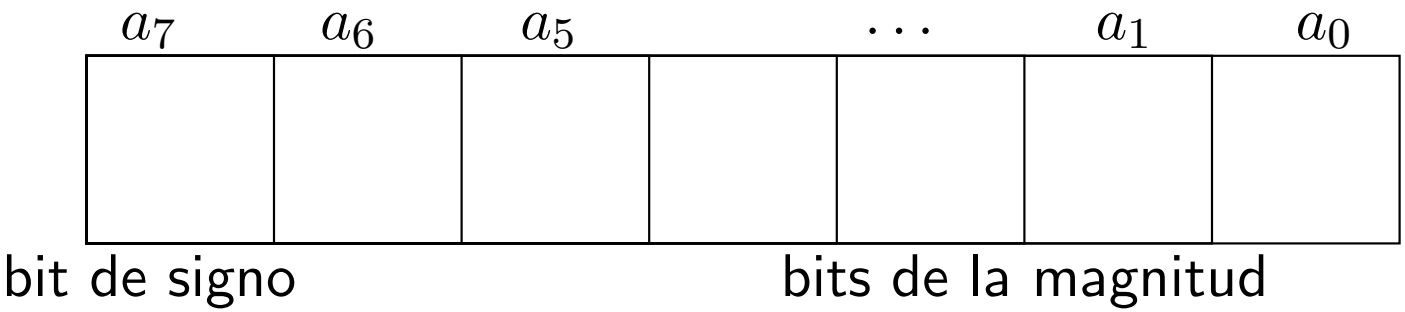
Rango de operación en Signo Magnitud

El rango de representación de este sistema es el siguiente:

$$\left(-2^{n-1}+1\right)_{10} \leq x \leq \left(2^{n-1}-1\right)_{10}$$

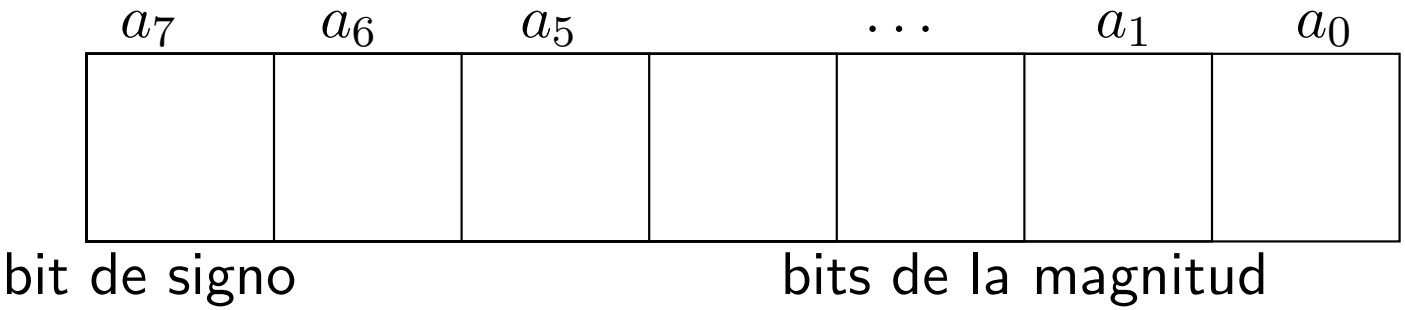
Ejercicio 1

En SM para $\mathbf{n} = 8$, el bit a_7 representa el signo del número y es resto de bits, la magnitud del mismo:



Ejercicio 1

En SM para $\mathbf{n} = 8$, el bit a_7 representa el signo del número y es resto de bits, la magnitud del mismo:



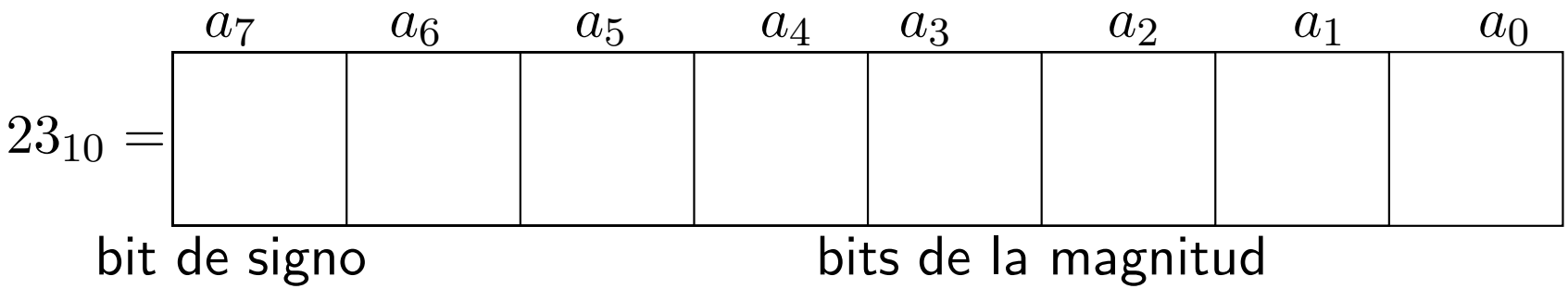
Su rango de representación es:

$$\begin{aligned} \left(-2^{8-1}+1\right)_{10} &\leq x \leq \left(2^{8-1}-1\right)_{10} \\ \left(2^{7}+1\right) &\leq x \leq \left(2^{7}-1\right)_{10} \\ \left(-128+1\right)_{10} &\leq x \leq \left(128-1\right)_{10} \\ -127_{10} &\leq x \leq 127_{10} \end{aligned}$$

Por tanto, se pueden representar $2^8 - 1 = 255$ números enteros.

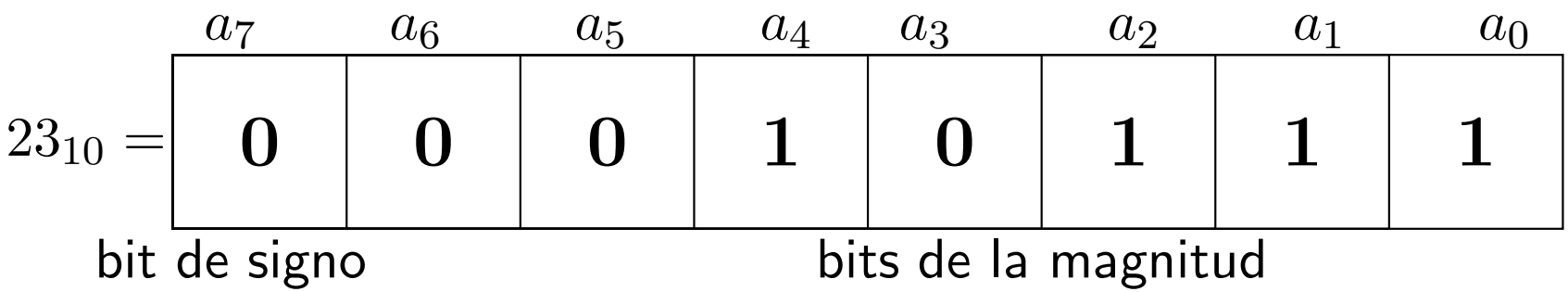
Ejercicio 2

Expresa en SM, para $\mathbf{n = 8}$, el número 23_{10}



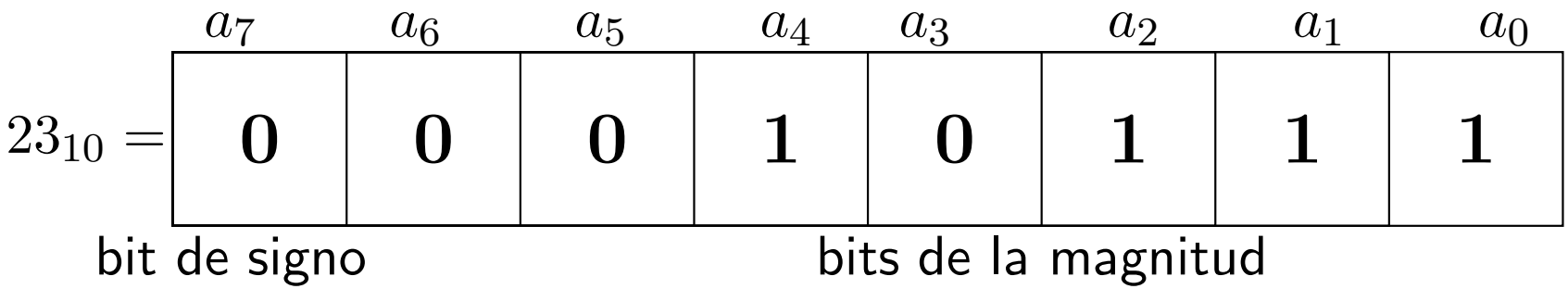
Ejercicio 2

Expresa en SM, para $\mathbf{n = 8}$, el número 23_{10}



Ejercicio 2

Expresa en SM, para $\mathbf{n = 8}$, el número 23_{10}

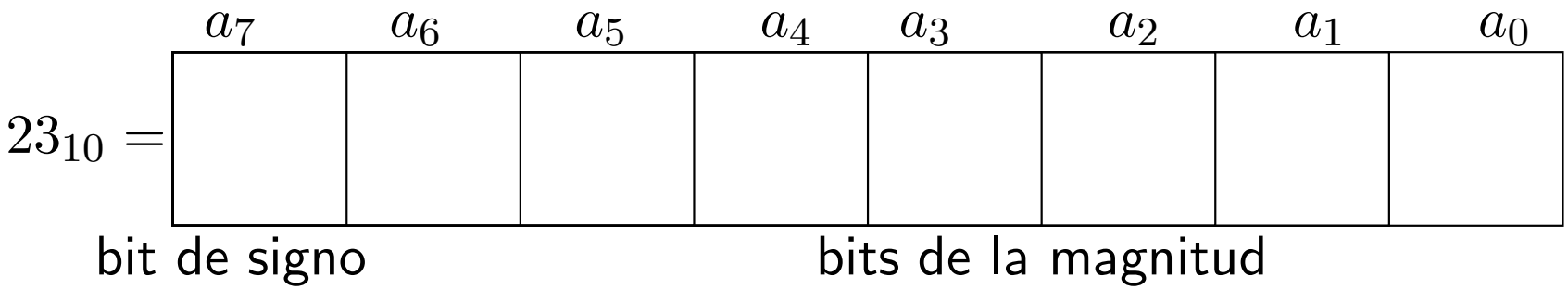


De tal manera que

$$\mathbf{23_{10} = 00010111_{SM}}$$

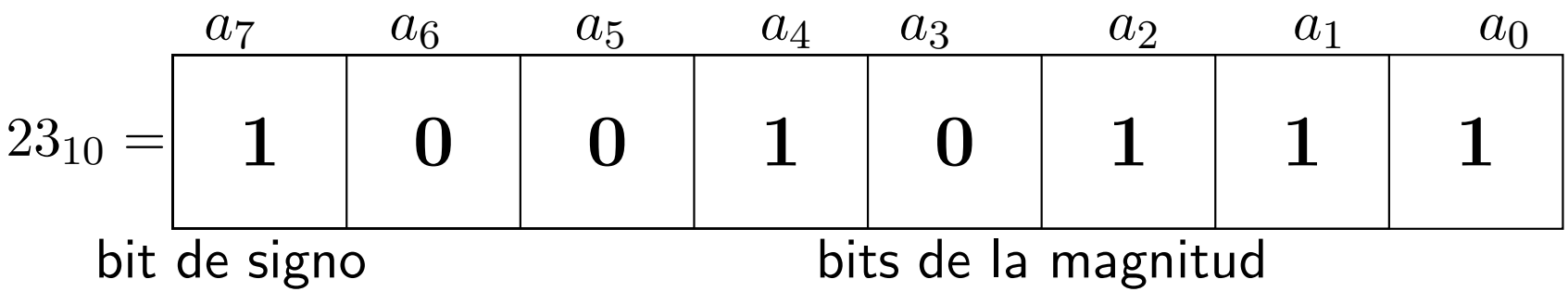
Ejercicio 3

Expresa en SM para $\mathbf{n = 8}$ el número -23_{10}



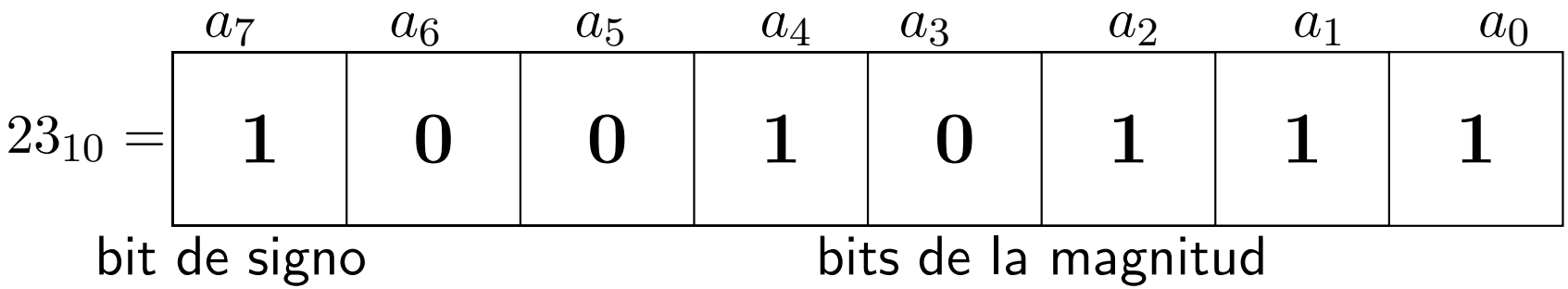
Ejercicio 3

Expresa en SM para $\mathbf{n = 8}$ el número -23_{10}



Ejercicio 3

Expresa en SM para $\mathbf{n = 8}$ el número -23_{10}



Por tanto:

$$\mathbf{-23_{10} = 1001011_{SM}}$$

¿Qué es el Exceso a 2^{n-1}?

En Exceso a 2^{n-1} tenemos que si se dispone de **n** bits para representar a un número entero (**N**) positivo o negativo, dicho número se representa como $N + 2^{n-1}$, por lo que:

$$N_{Ex} = N + 2^{n-1}$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$9503_{10} = (9503_{10} + (2^{16-1})_{10})$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$\begin{aligned} 9503_{10} =& (9503_{10} + (2^{16-1})_{10}) \\ =& (9503_{10} + (2^{15})_{10}) \end{aligned}$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$\begin{aligned} 9503_{10} =& (9503_{10} + (2^{16-1})_{10}) \\ =& (9503_{10} + (2^{15})_{10}) \\ =& (9503_{10} + 32768_{10}) \end{aligned}$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$\begin{aligned} 9503_{10} =& (9503_{10} + (2^{16-1})_{10}) \\ =& (9503_{10} + (2^{15})_{10}) \\ =& (9503_{10} + 32768_{10}) \\ =& (42271_{10}) \end{aligned}$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$\begin{aligned} 9503_{10} =& (9503_{10} + (2^{16-1})_{10}) \\ =& (9503_{10} + (2^{15})_{10}) \\ =& (9503_{10} + 32768_{10}) \\ =& (42271_{10}) \\ =& (1010010100011111_2) \end{aligned}$$

Ejemplo

En Exceso a 2^{n-1} para $n = 16$, el número positivo 9503_{10} se presenta como:

$$\begin{aligned} 9503_{10} =& (9503_{10} + (2^{16-1})_{10}) \\ =& (9503_{10} + (2^{15})_{10}) \\ =& (9503_{10} + 32768_{10}) \\ =& (42271_{10}) \\ =& (1010010100011111_2) \\ =& 1010010100011111_{\text{Exp a } 32768} \end{aligned}$$

El Estándar IEE 754

El estándar IEEE 754 establece dos formatos básicos para representar a los números reales en la computadora:

- de precisión simple.
- de precisión doble.

Precisión simple

Cuando se representa un número real en precisión simple, se usan 32 bits (4 bytes) de la siguiente manera:

1. 1 bit para el signo (**s**) de número.
2. 8 bits para el exponente (**exp**)
3. 23 bits para la mantisa (**m**) que se distribuyen de la siguiente forma:

El exponente se suele representar en notación *Exceso a* 2^{n-1}, mientras que para la mantisa, normalmente se usa el Signo y Magnitud.

La mantisa se suele normalizar colocando el punto decimal a la derecha del bit más significativo.

Ejemplo

Para escribir el número

$$101110.01010111010000111110000111110001001 1_2$$

en el estándar IEEE 754 con precisión simple, exponente en Exceso a 2^{n-1} y mantisa en Signo y Magnitud, lo primero que que hay que hacer es normalizarlo:

Ejemplo

Para escribir el número

$$101110.01010111010000111110000111110010011_2$$

en el estándar IEEE 754 con precisión simple, exponente en Exceso a 2^{n-1} y mantisa en Signo y Magnitud, lo primero que que hay que hacer es normalizarlo:

$$1.01110010101110100001111100001111100100112 \times 2^5$$

El exponente en Exceso a $2^{n-1}-1$ será:

$$\begin{aligned} 5_{10}+(2^{8-1}-1)_{10} &= 5_{10}+(27-1)_{10}= \\ &= 5_{10}+(128-1)_{10}= \\ &= 132_{10}= \\ &= 10000100_{\text{Exp. a } 127} \end{aligned}$$

Tomamos de la mantisa los 23 bits más significativos:

$$1.01111001010111000000111$$

El resto de los bits no se puede representar ya que no caben en la mantisa.

Cuando la mantisa se normaliza dejando el punto decimal a la derecha del bit más significativo, dicho bit siempre vale **1**, por lo que se puede prescindir de él y tomar agregar en su lugar otro bit, por lo que aumenta la precisión del número representado:

Tomamos de la mantisa los 23 bits más significativos:

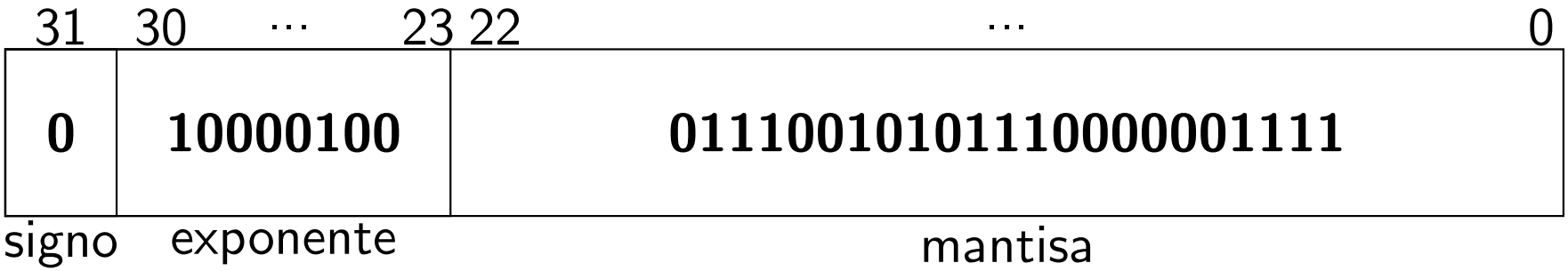
$$1.01111001010111000000111$$

El resto de los bits no se puede representar ya que no caben en la mantisa.

Cuando la mantisa se normaliza dejando el punto decimal a la derecha del bit más significativo, dicho bit siempre vale **1**, por lo que se puede prescindir de él y tomar agregar en su lugar otro bit, por lo que aumenta la precisión del número representado:

$$01111001010111000001111$$

Al bit omitido se le llama *bit implícito*. Por otra lado, el bit del signo vale **0**, ya que es un número posi-
tivo, por tanto, el número se puede representar como:



Precisión doble

En precisión doble, para escribir un número real, se utilizan 64 bits (8 bytes):

1. 1 bit para el signo (**s**) del número.
2. 11 bits para el exponente (**exp**)
3. 52 bits para la mantisa (**m**).

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$\begin{array}{r|l} 19 & 2 \\ \hline \end{array}$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$\begin{array}{r|l} 19 & 2 \\ \hline 9 & \end{array}$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

19	2
1	9

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$\begin{array}{rr|l} & 19 & 2 \\ \hline a_0 = 1 \longleftarrow & \mathbf{1} & 9 \end{array}$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$\begin{array}{rr|l} 19 & 2 & \\ \hline a_0 = 1 \longleftarrow \mathbf{1} & 9 & \\ \hline \end{array}$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rr|l}
 & 19 & 2 \\
\hline
a_0 = 1 \longleftarrow & \mathbf{1} & 9 \;\; 2 \\
\hline
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rr|l}
 & 19 & 2 \\
\hline
a_0 = 1 \longleftarrow & \mathbf{1} & 9 \quad 2 \\
\hline
 & & 4
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rcl}
 & 19 & \big| \; 2 \\
\cline{2-3}
a_0 = 1 \longleftarrow & \mathbf{1} \quad 9 & \big| \; 2 \\
\cline{2-3}
 & \mathbf{1} \quad 4 &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \longleftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \longleftarrow & \mathbf{1} & 4 &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & \\
\hline
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \longleftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \longleftarrow & \mathbf{1} & 4 & 2 \\
\hline
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
 & & 2 &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
 & \mathbf{0} & 2 &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 & \\
\hline
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

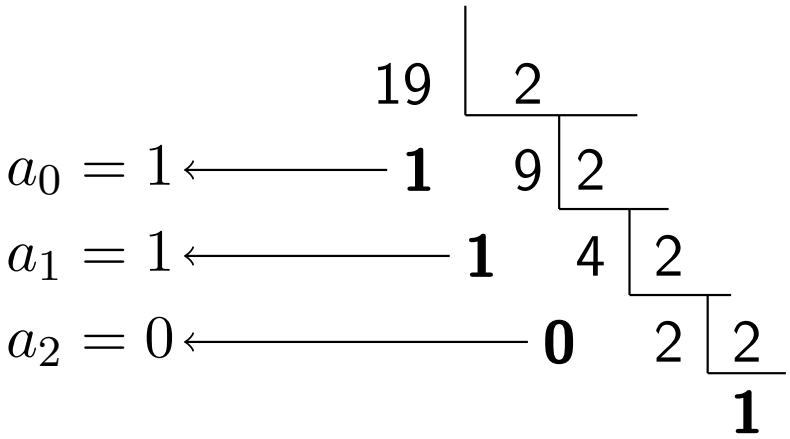
1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 & 2 \\
\hline
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:



Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 & 2 \\
\hline
 & \mathbf{0} & \mathbf{1} &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 & 2 \\
\hline
a_3 = 0 \leftarrow & \mathbf{0} & \mathbf{1} &
\end{array}
$$

Ejemplo

Se quiere expresar el número 19.5625_{10} en el estándar IEEE 754 con precisión doble, por lo que tendremos que realizar:

1. Cambiar 19.5625_{10} a base 2, iniciando con la parte entera:

$$
\begin{array}{rrr|l}
 & 19 & & 2 \\
\hline
a_0 = 1 \leftarrow & \mathbf{1} & 9 & 2 \\
\hline
a_1 = 1 \leftarrow & \mathbf{1} & 4 & 2 \\
\hline
a_2 = 0 \leftarrow & \mathbf{0} & 2 & 2 \\
\hline
a_3 = 0 \leftarrow & \mathbf{0} & \mathbf{1} & \rightarrow a_4 = 1
\end{array}
$$

2. Ahora la parte fraccionaria

$$0.5625 \times 2 = \mathbf{1}.125$$

2. Ahora la parte fraccionaria

$$0.5625 \times 2 = \mathbf{1}.125 \longrightarrow a_{.1} = 1$$

2. Ahora la parte fraccionaria

$$0.5625 \times 2 = \mathbf{1}.125 \longrightarrow a_{.1} = 1$$
$$0.125 \times 2 = \mathbf{0}.25$$

2. Ahora la parte fraccionaria

$$0.5625 \times 2 = \mathbf{1}.125 \longrightarrow a_{.1} = 1$$
$$0.125 \times 2 = \mathbf{0}.25 \longrightarrow a_{.2} = 0$$

2. Ahora la parte fraccionaria

$$
\begin{aligned}
0.5625 \times 2 &= \mathbf{1}.125 \longrightarrow a_{.1} = 1 \\
0.125 \times 2 &= \mathbf{0}.25 \longrightarrow a_{.2} = 0 \\
0.25 \times 2 &= \mathbf{0}.5
\end{aligned}
$$

2. Ahora la parte fraccionaria

$$
\begin{aligned}
0.5625 \times 2 &= \mathbf{1}.125 \longrightarrow a_{.1} = 1 \\
0.125 \times 2 &= \mathbf{0}.25 \longrightarrow a_{.2} = 0 \\
0.25 \times 2 &= \mathbf{0}.5 \longrightarrow a_{.3} = 0
\end{aligned}
$$

2. Ahora la parte fraccionaria

$$
\begin{aligned}
0.5625 \times 2 &= \mathbf{1}.125 \longrightarrow a_{.1} = 1 \\
0.125 \times 2 &= \mathbf{0}.25 \longrightarrow a_{.2} = 0 \\
0.25 \times 2 &= \mathbf{0}.5 \longrightarrow a_{.3} = 0 \\
0.5 \times 2 &= \mathbf{1}
\end{aligned}
$$

2. Ahora la parte fraccionaria

$$
\begin{aligned}
0.5625 \times 2 &= \mathbf{1}.125 \longrightarrow a_{.1} = 1 \\
0.125 \times 2 &= \mathbf{0}.25 \longrightarrow a_{.2} = 0 \\
0.25 \times 2 &= \mathbf{0}.5 \longrightarrow a_{.3} = 0 \\
0.5 \times 2 &= \mathbf{1} \longrightarrow a_{.4} = 1
\end{aligned}
$$

De modo que el número queda expresado como:

$$19.5625_{10} = 10011.1001_2$$

De modo que el número queda expresado como:

$$19.5625_{10} = 10011.1001_2$$

3. Se normaliza el número en binario que se obtuvo, dejando el punto decimal a la derecha del bit más significativo:

$$10011.1001_2 = 1.00111001 \times 2^4$$

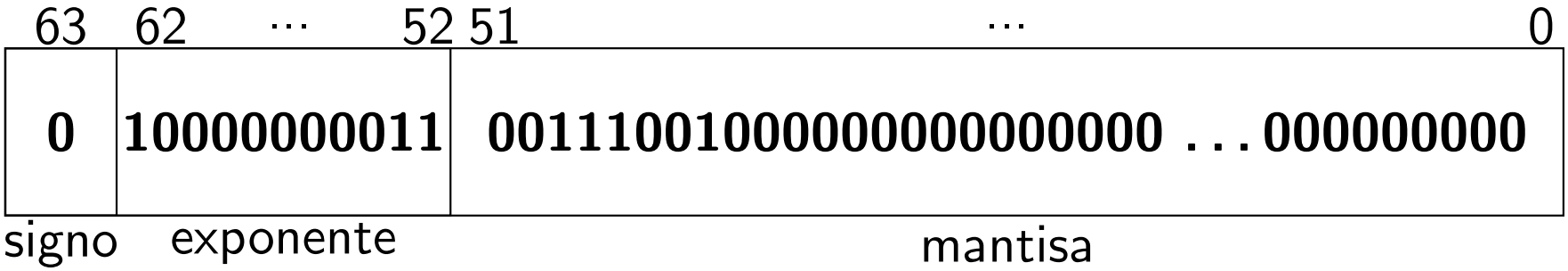
4. Se escribe el exponente en Exceso a $2^{n-1}-1$

$$\begin{aligned} 4_{10}+(2^{11-1}-1)_{10} &= 4_{10}+(2^{10}-1)_{10} \\ &= 4_{10}+(1024-1)_{10} \\ &= 1027_{10} \\ &= 10000000011_{\text{Exp. a } 1023} \end{aligned}$$

5. Se deja la mantisa utilizando el bit implícito: se eligen los ocho bits que están a la derecha de la coma $(\mathbf{00111001})$ y el resto de la mantisa se rellena con ceros:

$$00111001000000\ldots000000$$

El número expresado en el estándar IEEE 754 con precisión doble es:



Casos especiales del IEEE 754

Tanto en los tipos de datos de precisión simple como de precisión doble, existen algunos casos especiales que dependen de los valores del signo, del exponente y de la mantisa.

Signo (s)	Exponente (exp)	Mantisa (m)	Significado
Positivo (0)	Todos unos $(111\ldots11)$	Todos ceros $(000\ldots00)$	Más infinito $(+\infty)$
Negativo (1)	Todos unos $(111\ldots11)$	Todos ceros $(000\ldots00)$	Menos infinito $(-\infty)$
0 ó 1	Todos unos $(111\ldots11)$	Distinta de todo ceros	No es un número (Not a Number, **NaN**)
0 ó 1	Todos ceros $(000\ldots00)$	Todos ceros $(000\ldots00)$	Representa al cero (0)
0 ó 1	Todos ceros $(000\ldots00)$	Distinta de todo ceros	Número muy pequeño cercano al cero

¿Y si la representación del número binario es periódica?

Hagamos la conversión de 0.4 a binario:

$$
\begin{aligned}
0.4 \times 2 &= 0.8 && \rightarrow a_{.1} = 0 \\
0.8 \times 2 &= 1.60 && \rightarrow a_{.2} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.3} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.4} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.5} = 0 \\
0.8 \times 2 &= 1.60 && \rightarrow a_{.6} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.7} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.8} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.9} = 0
\end{aligned}
$$

$$
\begin{aligned}
0.8 \times 2 &= 1.60 && \rightarrow a_{.10} = 1 \\
0.6 \times 2 &= 1.20 && \rightarrow a_{.11} = 1 \\
0.2 \times 2 &= 0.4 && \rightarrow a_{.12} = 0 \\
0.4 \times 2 &= 0.8 && \rightarrow a_{.13} = 0
\end{aligned}
$$

Como vemos, al expresar el valor de 0.4 a binario, tenemos un número periódico, en donde 0011 se repite indefinidamente. En el ejemplo se llegó hasta 0.01100011001100 que es igual a 0.39990234375.

Cuanto más se repita el cálculo, más nos acercamos a 0.40, pero como tenemos un número binario periódico, no importa la cantidad de dígitos fraccionarios, siempre se acercará a 0.40, por lo que se redondea luego de cierto número de dígitos.

Veamos la conversión a decimal:

$$\begin{aligned}0.01100011001100 = &(0 \times 2^{0}) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) + \\ &+ (0 \times 2^{-4}) + (0 \times 2^{-5}) + (1 \times 2^{-6}) + (1 \times 2^{-7}) + \\ &+ (0 \times 2^{-8}) + (0 \times 2^{-9}) + (1 \times 2^{-10}) + (1 \times 2^{-11}) + \\ &+ (0 \times 2^{-12}) + (0 \times 2^{-13}) = \\ &= 0.39990234375\end{aligned}$$

Que se puede redondear a 0.4

Contenido

1. Representación de números de punto flotante
   - Qué es lo que pasa?
2. Representación de números enteros
   - Signo y Magnitud
   - Exceso a 2^{n-1}
   - Estándar IEEE 754
     - Precisión simple
     - Precisión doble
     - Casos especiales del IEEE 754
3. Propagación de errores
   - Modelos para el desastre
   - Ejercicio 1

Modelos para el desastre

Un cálculo que utiliza números que se almacenan de manera aproximada en la computadora, puede devolver una solución aproximada.

Para demostrar este hecho, consideremos que hay un valor de incertidumbre, sea x_c el valor representado en la computadora del valor exacto x, tal que:

$$x_c \simeq x(1 + \epsilon_x)$$

Donde ϵ_x es el error relativo de x_c, el cual se espera que sea similar en magnitud al épsilon de la máquina ϵ_m.

Si usamos ésta notación para una diferencia entre dos valores, tenemos que:

$$a = b - c \ \rightarrow a_c \simeq b_c - c_c \simeq b(1+\epsilon_b) - c(1+\epsilon_c)$$

$$\rightarrow \frac{a_c}{a} \simeq 1 + \epsilon_b \frac{b}{a} - \frac{c}{a}\epsilon_c$$

Vemos que el error resultante en a es un promedio de los errores en b y c, y no hay seguridad en que los dos términos se cancelen.

El error en a_c se incrementa cuando se restan dos valores cercanos $(b \simeq c)$, dado que cuando se restan las cifras más significativas de ambos números, el error de las cifras menos significativas toma la forma de:

$$\frac{a_c}{a} = 1 + \epsilon_a \simeq 1 + \frac{b}{a}(\epsilon_b - \epsilon_c) \simeq 1 + \frac{b}{a} max(|\epsilon_b|, |\epsilon_c|)$$

Ejercicio 1

Aprendimos en la secundaria a resolver la ecuación homogénea de segundo grado:

$$ax^2 + bx + c = 0$$

que tiene una solución analítica que se puede escribir como

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad x_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

Revisando la expresión anterior vemos que la cancelación de la diferencia (y por tanto, un incremento en el error) aumenta cuando $b^2 >> 4ac$ debido a que la raíz cuadrada y el siguiente término están muy próximas a cancelarse.

Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.

Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.
2. Revisa cómo los errores obtenidos en los cálculos, aumentan conforme hay una cancelación de la diferencia de términos y su relación con la precisión de la máquina. Prueba con los siguientes valores $a = 1$, $b = 1$, $c = 10^{-n}, n = 1, 2, 3, \ldots$

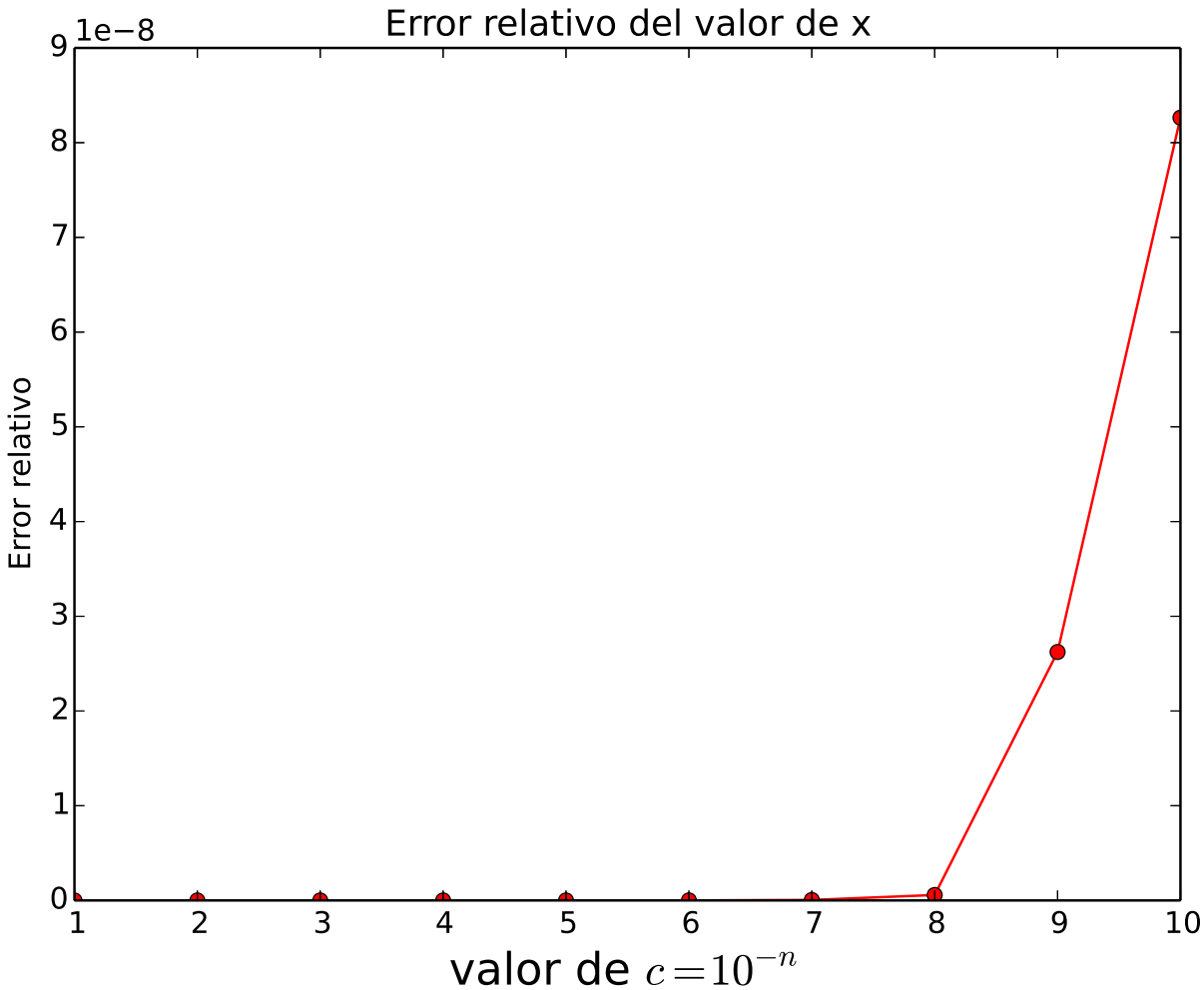
Manos al código

1. Escribe un programa que calcule las cuatro soluciones para valores arbitrarios de a, b y c.
2. Revisa cómo los errores obtenidos en los cálculos, aumentan conforme hay una cancelación de la diferencia de términos y su relación con la precisión de la máquina. Prueba con los siguientes valores $a = 1$, $b = 1$, $c = 10^{-n}, n = 1, 2, 3, \ldots$
3. Cómo mejorarías el programa para obtener la mayor precisión en tu respuesta?

¿Qué resultados debemos esperar?

Al momento de sentarse a escribir el código, hay que contemplar o estimar, qué resultados son los que deberíamos de esperar, con el ejercicio, para que valor de $c = 10^{-n}$, obtenemos un error relativo, y pues al variar el valor de $n = 1, \ldots, 9, 10$ podremos obtener una idea del comportamiento del error, si graficamos los valores del error relativo contra n.

Gráfica del error relativo y el valor de c

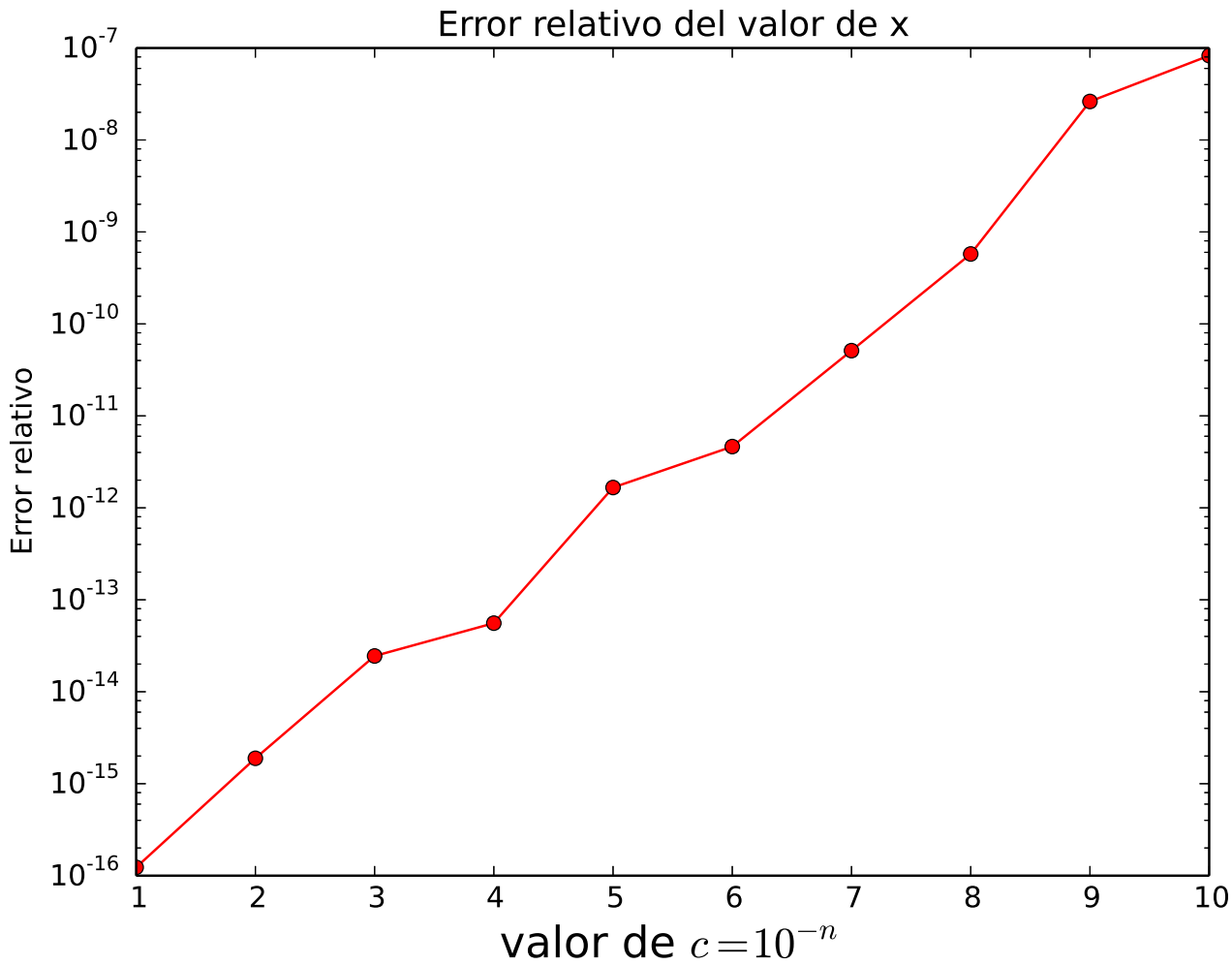


¿Qué estamos viendo?

En la gráfica anterior, vemos una línea muy pegada al eje x, pero hay que considerar que el valor es muy pequeño, mientras que para $c = 10^{-9}$, ya se eleva más el punto y se hace más notorio en la gráfica.

Podríamos pensar que el error relativo es casi el mismo para valores menores de $n = 9$, pero hay que recordar que el ojímetro no funciona bien y menos en física computacional, por lo que ahora hacemos un cambio de escala en el eje y, para ello, usamos la función `semilogy()` de la librería `matplotlib`, que nos cambia la escala a una de tipo semilogarítmico.

Gráfica del error relativo y el valor de c, ajustando el eje y



Tenemos una mejor idea de lo que ocurre

- Vemos que existe una "tendencia" lineal (adviertiendo que uno de los ejes está en escala logarítmica)
- El error relativo va aumentando conforme se incrementa el valor de n.
- Hay algunas variaciones con respecto al valor de error, es decir, la dispersión es notaria, aunque no hemos dicho que debe de ser estrictamente un comportamiento lineal.
- ¿Qué podemos hacer para evitar las operaciones entre valores muy grandes $(b = 1)$ con valores muy pequeños $(4ac)$?