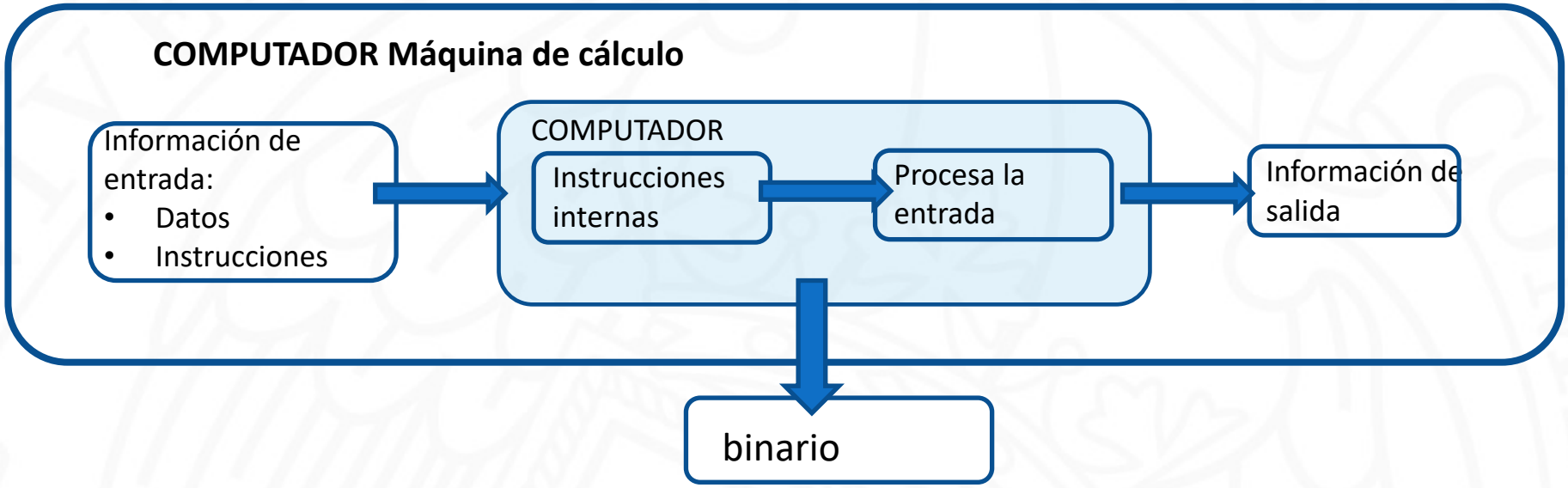
# Tema 2

## Aritmética del computador

María Guijarro Mata-García

2021-22 / 1ºD




¿Cómo se guardan los datos?
¿Cómo se opera con ellos?

**Números enteros:**

Representación exacta

0 = 0
1 = 1
2 = 10
3 = 11
4 = 100
…

Límite de números representados: Arquitectura del ordenador.

¿Tiene suficientes espacios para guardar todos los dígitos en un registro de memoria?

**Números no enteros:**

Podrán tener representación exacta o no.

Depende de:

- El número representado.
- Arquitectura del ordenador.

¿Tiene representación finita?

¿Tiene suficientes espacios para guardar todos los dígitos en un registro de memoria?

Se trata de representar la mayor cantidad posible de números, con el menor espacio entre ellos. La representación nunca va a ser continua.

**Números máquina:** Los podemos representar de manera exacta (cantidad finita).

Pero no es lo mismo representar:

1 2 3 4 5 6 7 8 9 10

Que:

0.5 1 1.5 2 2.5 3 3.5 … 6 6.5 7 7.5 8 8.5 9 9.5 10

# Arquitectura del ordenador

Actualmente tenemos sistemas de **32 bits** y **64 bits** de longitud de palabra.

Cada bit es un espacio de memoria donde guardar 0 ó 1.

**32 bits** → Máximo de 32 espacios a rellenar con 0's o 1's para representar un número

**64 bits** → Máximo de 64 espacios a rellenar con 0's o 1's para representar un número

Esto determina qué números se pueden representar de manera exacta. → LA MAYORÍA NO

# ¿Cómo los representamos?

**Podemos representar $2^{16}$ números enteros**

Veamos un **ejemplo** para **números enteros positivos**: sistema de **16 bits**.

Número más grande:

$$1*2^{15} + 1*2^{14} + 1*2^{13} + \ldots + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 2^{16}-1 = 65535$$

| posición | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| valor | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Número más pequeño:

$$0*2^{15} + 0*2^{14} + 0*2^{13} + \ldots + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 0$$

| posición | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| valor | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Rango de representación: **n bits → [0, $2^n$-1]**

# ¿Cómo los representamos?

**Podemos representar $2^{16}$ números enteros**

Veamos un **ejemplo** para **números enteros positivos**: sistema de **16 bits**.

¿Y si queremos también números negativos y no enteros?

→ Tendremos que organizar nuestros 16 bits para representar el mayor número posible

Siempre teniendo en cuenta que:

Los nº racionales periódicos y los irracionales no pueden representarse de forma exacta con un número finito de decimales → siempre representación aproximada.

Los nº racionales no periódicos pueden representarse o no de manera exacta dependiendo del número que se trate, el tamaño de los registros del ordenador y el sistema empleado para la representación.

# ¿Cómo los representamos?

Veamos un **ejemplo** para **números enteros positivos**: sistema de **16 bits**.

Los dos sistemas de representación de números no enteros más conocidos son:

- **Representación en PUNTO FIJO.**
- **Representación en PUNTO FLOTANTE.**

# ¿Cómo los representamos?

OJO: Es un ejemplo arbitrario

Veamos un **ejemplo** para **punto fijo** en sistema de **16 bits**.

Usado en los primeros ordenadores.

1. Un campo de bit para el signo del número: 1→-; 0→ +
2. Un campo de bits para la parte entera del número.
3. Un campo de bits para la parte decimal del número.

**1 Bit para el signo**

**7 Bits para parte entera**

**8 Bits para parte decimal**

Ejemplo:

$-10111.0011101101_2$:

| signo | Parte entera | | | | | | | Parte decimal | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| - | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ | $2^{-7}$ | $2^{-8}$ |

Problema: me sobran bits en la parte entera y tengo que truncar 2 decimales en la decimal

He representado -23.23046875 y nuestro número inicial era -23.2314453125

# ¿Cómo los representamos?

OJO: Es un ejemplo arbitrario

Veamos un **ejemplo** para **punto flotante** en sistema de **16 bits**.

El **punto** decimal de la mantisa **no separa la parte entera de la parte decimal**.

Representamos el número con un solo dígito en la parte entera (como formato científico).

Por ejemplo:

El número -1101.00101 se representa $-1.10100101 \times 2^3$

Utilizamos el valor del exponente para recuperar la parte entera del número.

# ¿Cómo los representamos?

OJO: Es un ejemplo arbitrario

Veamos un **ejemplo** para **punto flotante** en sistema de **16 bits**.

Se expresa el número en función de cuatro componentes:

- **Signo:** indica el signo del número (0= positivo, 1=negativo)
- **Mantisa:** contiene la magnitud del número (en binario puro)
- **Exponente:** contiene el valor de la potencia de la base (siempre positivo)
- **Base:** queda implícita y es común a todos los números (habitual base 2)

Ejemplo:

$-1101.00101 \rightarrow -1.10100101 \times 2^{3}$

| signo | Mantisa | | | | | | | | | | Exponente | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| - | $2^{0}$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ | $2^{-7}$ | $2^{-8}$ | $2^{-9}$ | $2^{-10}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ |

# ¿Cómo los representamos?

Necesitamos un formato estándar para que todos los ordenadores lo hagan igual.

Ejemplo:

$$-1101.0010111 \rightarrow -1.1010010111 \times 2^{3}$$

No es lo mismo representarlo con 11 bits en la mantisa…

| signo | Mantisa | | | | | | | | | | Exponente | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| - | $2^{0}$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ | $2^{-7}$ | $2^{-8}$ | $2^{-9}$ | $2^{-10}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ |

… que con 10 bits en la mantisa: perdemos el último dígito

| signo | Mantisa | | | | | | | | | Exponente | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| - | $2^{0}$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ | $2^{-7}$ | $2^{-8}$ | $2^{-9}$ | $2^{4}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ |

# ¿Cómo los representamos?

Necesitamos un formato estándar para que todos los ordenadores lo hagan igual.

El proceso de estandarización fue bastante lento: El estándar no fue introducido hasta 1985.

**Estándar actual:** **IEEE 754** (Institute of Electrical and Electronic Enginers).

Se pensó en dos tamaños de registro:

- 32 bits → precisión simple.
- 64 bits → precisión doble.

# El estándar IEEE 754

Formato de punto flotante de 32 bits de longitud de palabra
**(Simple precisión)**

- **1 bit para signo:** 0 → positivo; 1→ negativo
- **23 bits para la mantisa**
- **8 bits para el exponente [-127, 128]**

Formato de punto flotante de 64 bits de longitud de palabra
**(Doble precisión)**

- **1 bit para signo:** 0 → positivo; 1→ negativo
- **52 bits para la mantisa**
- **11 bits para el exponente [-1023, 1024]**

# El estándar IEEE 754

Exponente:

Con 8 bits podemos representar $2^8 = 256$ → 0….  255

00000000  11111111

Cómo conseguir exponentes negativos? **Exponente en exceso 127**

0 ----------127 128----------255

exp. Negativos  exp. Positivos

-127---------- 0----------128

# El estándar IEEE 754

Formato de punto flotante de 32 bits de longitud de palabra **(Simple precisión)**

| Exponente representado | Aplico exceso 127 | Decimal | Bits de exponente |
|---|---|---|---|
| -127 | -127+127=0 | 0 | 00000000 |
| -125 | -125+127=2 | 2 | 00000010 |
| 0 | -127+127=0 | 127 | 01111111 |
| 128 | 128+127=255 | 255 | 11111111 |

**Valores restringidos de exponente:**

Exponente = 128 → Estándar evita desbordamiento, no se considera número (valor depende de mantisa):

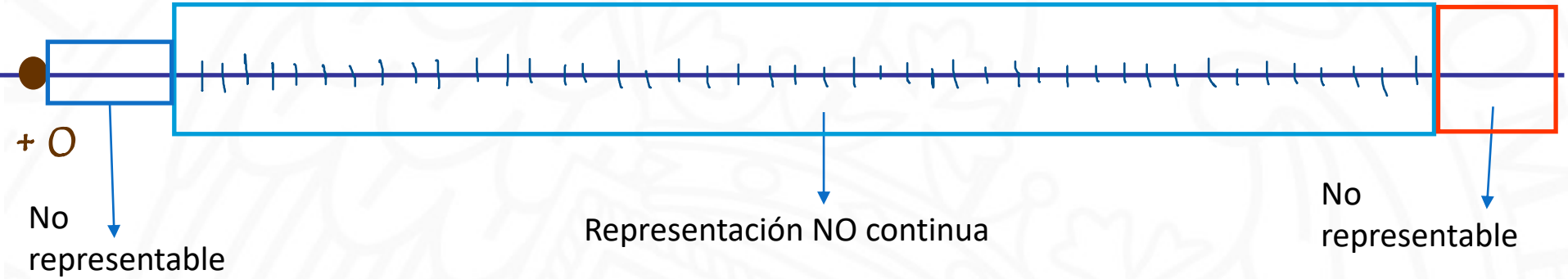
- Bit de mantisa son todo 0's → registro representa ∞
- Bit de mantisa ≠ 0 → NaN (Not a Number) (sin significado matemático).

Exponente = -127 reservado para números muy pequeños: **números desnormalizados** (la mantisa no lleva el 1 entero implícito y el exponente es -126)

Queda por tanto para el exponente: [-126, 127]

# El estándar IEEE 754

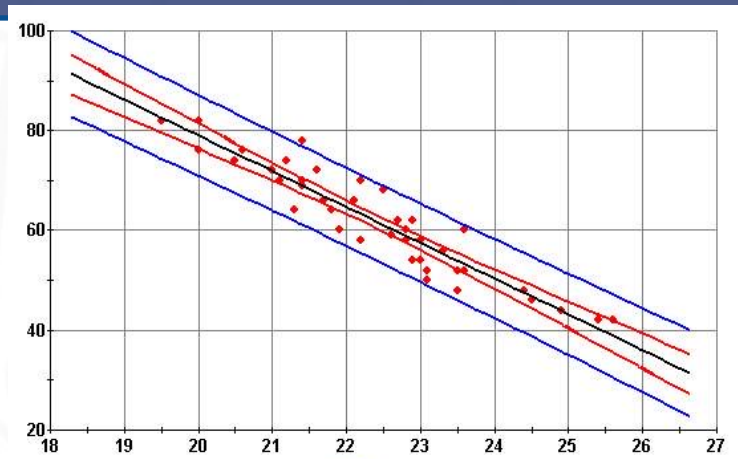
Se trata de representar la mayor cantidad posible de números, con el menor espacio entre ellos.
De forma que todos los ordenadores lo hagan igual.

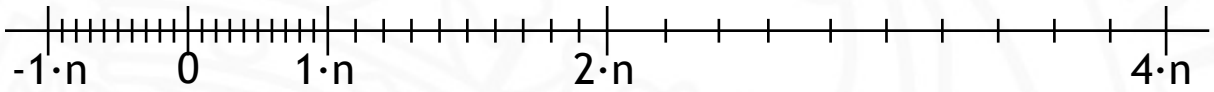


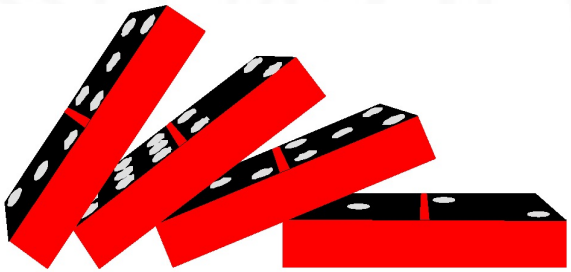

No todos los números se pueden representar → ERRORES

# Tipos de errores

1. Datos de entrada
   - Experimental
   - Cálculos previos
2. Representación de los números :
   - Redondeo
   - Desbordamiento
3. Cálculos:
   - Acumulación de errores de redondeo
   - Anulación catastrófica
   - Desbordamiento
4. Algoritmo :
   - Discretización / Truncamiento
   - Estabilidad del algoritmo



$$\frac{dx}{dt} \approx \frac{x_n - x_{n-1}}{T}$$

# Tipos de errores

Representación de los números :

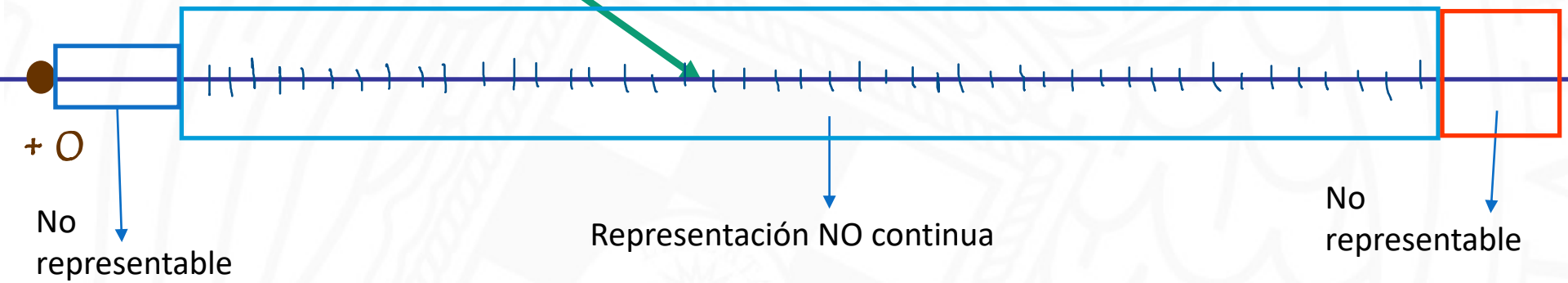
Errores de redondeo

Si el tamaño de la mantisa es mayor que el representable

Número no representable

Hay que aproximar

Al número máquina más cercano



# Tipos de errores

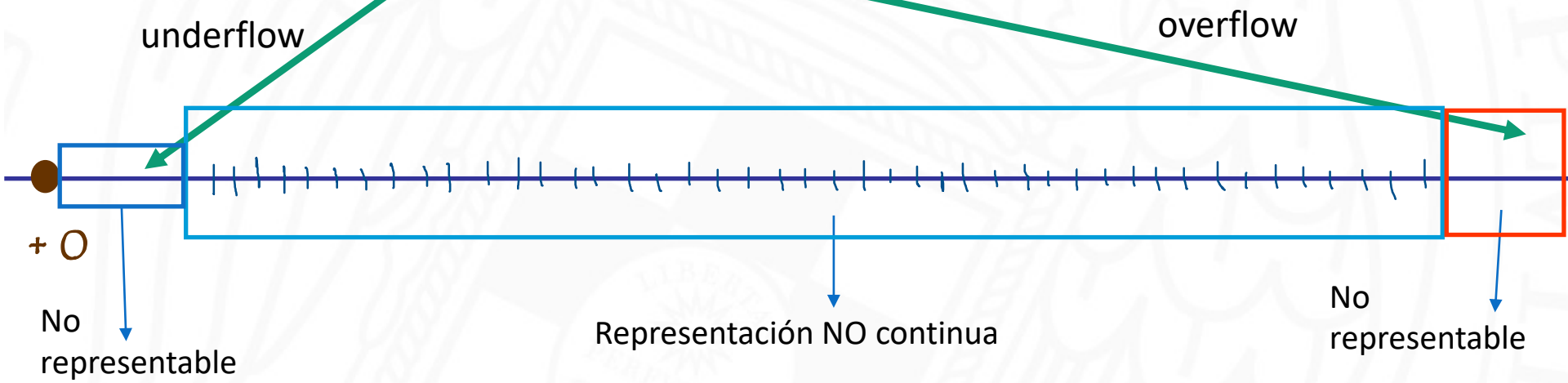
Representación de los números :

Errores de desbordamiento

Si el exponente es demasiado grande o pequeño

Número no representable

Underflow → 0
Overflow → $\pm\infty$



**Operaciones aritméticas:** Acumulación de errores de redondeo

Ejemplo: Representación en base 10 en punto flotante: mantisa de cuatro dígitos y exponente de dos dígitos

Tenemos dos números: 99.99 y 0.161 y los sumamos.

$99.99 = 9.999 \times 10^{1}$
$0.161 = 1.61 \times 10^{-1}$

Pasos que sigue el ordenador para realizar esta suma:

*1. Alineamiento*: representar el número más pequeño usando el exponente del mayor.
$1.610 \times 10^{-1} \rightarrow 0.0161 \times 10^{1} \rightarrow 0.016 \times 10^{1}$

2. *Operación:* sumar:
$9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$ ¿Hay desbordamiento?

# Tipos de errores

Operaciones aritméticas: Acumulación de errores de redondeo

3. *Normalización:* si hay desbordamiento de la mantisa→ volver a normalizarla

$10.015 \times 10^1 \rightarrow 1.0015 \times 10^2$

4. *Redondeo*: redondeo porque no caben todos los dígitos en la mantisa (sólo 4 dígitos)

$1.0015 \times 10^2 \rightarrow 1.002 \times 10^2$

5. *Renormalización*: a veces es necesario volver a normalizar la mantisa después del redondeo.

## Operaciones aritméticas: Anulación catastrófica

Ocurre cuando en una operación aritmética (ej. la sustracción) se van cancelando los dígitos no afectados por el redondeo y quedan los que sí están afectados. Hay que evitar este tipo de operaciones.

Ejemplo: Operación a realizar $b^2-4\cdot a\cdot c$ en base 10 y mantisa de 5 dígitos. b = $3.3357\text{x}10^0$, a = $1.2200\text{x}10^0$, c = $2.2800\text{x}10^0$

Resultado exacto: $b^2-4\cdot a\cdot c = 4.944\text{x}10^{-4}$

$b^2 = 11.126894490000002 \approx 1.112689\text{x}10^1$

$4\cdot a\cdot c = 11.126399999999999 \approx 1.112640\text{x}10^1$

$b^2-4\cdot a\cdot c = 4.900\text{x}10^{-4}$ → error 1%

El resultado de la sustracción solo contiene los dígitos sometidos a redondeo. Se usan 2 bits de guarda: La sustracción no comete error.

# Tipos de errores

## Operaciones aritméticas: Desbordamiento

Al realizar operaciones aritméticas es posible llegar a resultados no representables: overflow (+ o -) o bien underflow (+ o -).

Una vez producido el desbordamiento el resto de las operaciones quedan invalidadas.

La solución a este tipo de problemas exige modificar la forma en que se realizan los cálculos.