# Organización de Computadoras

Clase 3

# Temas de Clase

- Representación de números en Punto Flotante

# Números en punto fijo

- Todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria está siempre ubicada en el mismo lugar.
- La diferencia principal entre la representación en el papel y su almacenamiento en la computadora, es que no se guarda coma alguna, se supone que está en un lugar determinado.

Ejemplo:
1)si estamos trabajando con 8 bits, 5 parte entera y 3 fraccionaria

01000,100 se almacena como 01000100 en la memoria

Si hicieramos un programa que lee un binario y lo muestra en la pantalla en formato decimal, al leer 01000100 deberá mostrar 8,5

- Todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria está siempre ubicada en el mismo lugar.
- La diferencia principal entre la representación en el papel y su almacenamiento en la computadora, es que no se guarda coma alguna, se supone que está en un lugar determinado.

Ejemplo:
1)si estamos trabajando con 8 bits, 5 parte entera y 3 fraccionaria

01000,100 se almacena como 01000100 en la memoria

Si hicieramos un programa que lee un binario y lo muestra en la pantalla en formato decimal, al leer 01000100 deberá mostrar 8,5

- Todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria está siempre ubicada en el mismo lugar.
- La ... en ... com ... algu ... dete

2)si en cambio,
estamos trabajando con 8 bits, 4 parte entera y 4 fraccionaria

0100,0100 se almacena como 01000100 en la memoria

Si hicieramos un programa que lee un binario y lo muestra en la pantalla en formato decimal, al leer 01000100 deberá mostrar 4,25

Ejemplo:
1)si estamos trabajando con 8 bits, 5 parte entera y 3 fraccionaria

01000,100 se almacena como 01000100 en la memoria

Si hicieramos un programa que lee un binario y lo muestra en la pantalla en formato decimal, al leer 01000100 deberá mostrar 8,5

- Todos los números exactamente la misma coma fraccionaria está mismo lugar.

La "coma" no está almacenada, no está señalizada de ninguna forma. Sólo está en la cabeza del programador que sabe como hay que interpretar el binario 01000100 que está almacenado en memoria.

- La dif... en com... algu... dete...

2)si en cambio,
estamos trabajando con 8 bits, 4 parte entera y 4 fraccionaria

0100,0100 se almacena como 01000100 en la memoria

Si hicieramos un programa que lee un binario y lo muestra en la pantalla en formato decimal, al leer 01000100 deberá mostrar 4,25

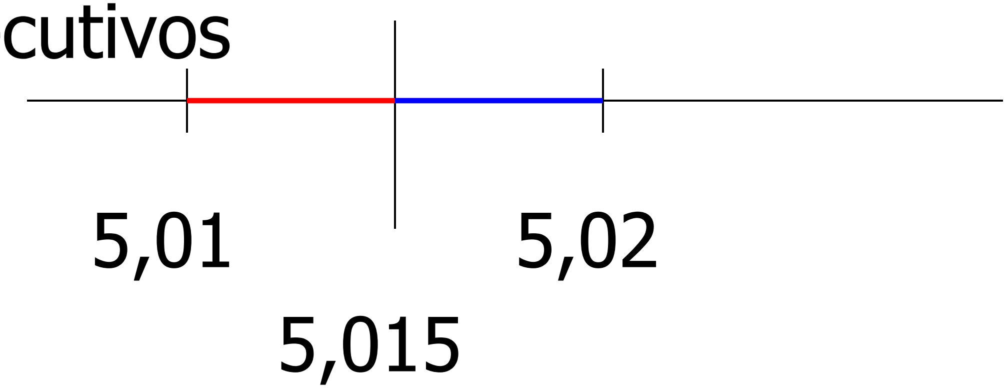
# Rango y Resolución

- Rango: diferencia entre el número mayor y el menor
- Resolución: diferencia entre dos números consecutivos

➢ Observar que al fijar la cantidad de dígitos y la posición del punto decimal, hemos también fijado el RANGO y RESOLUCIÖN de la representación.

# Error en punto fijo (1)

- El máximo error cometido en una representación puede considerarse como la mitad de la diferencia (resolución) entre dos números consecutivos




- 5,01 ≤ Nº< 5,015 se representa por 5,01
- 5,015 ≤ Nº < 5,02 se representa por 5,02

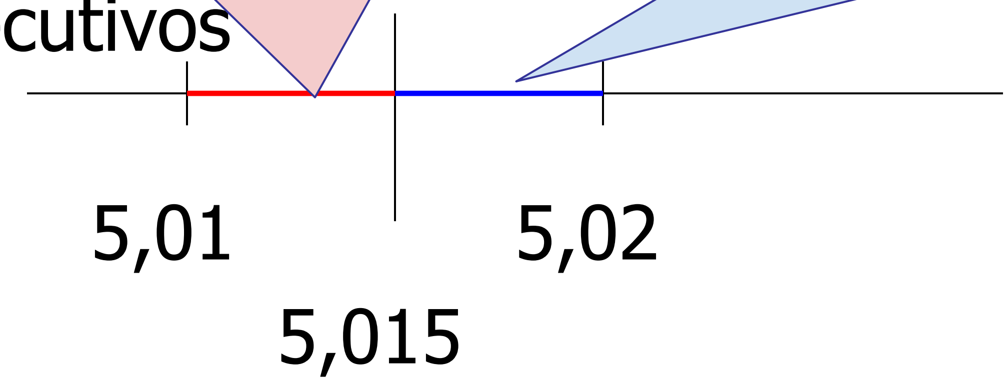
# Error en punto fijo (1)

- El máximo error cometido en una representación

consecutivos

Cualquier número que pertenezca a este intervalo se representa con el 5,01=>cometemos error

Cualquier número que pertenezca a este intervalo se representa con el 5,02=>cometemos error




- 5,01 ≤ Nº< 5,015 se representa por 5,01
- 5,015 ≤ Nº < 5,02 se representa por 5,02

# Error en punto fijo (2)

- En cualquiera de los dos casos el Error Absoluto máximo resulta ser:

  $EA\ max = 5{,}015 - 5{,}01 = 0{,}005$ ó

  $(5{,}02 - 5{,}01)/2 = 0{,}005$

  - Que corresponden a los Nº marcados en rojo ó azul.

# Números en punto flotante

- En punto fijo (ej. Ca2), es posible representar un rango de enteros positivos y negativos centrados en 0.
- Suponiendo un número con componente fraccionaria, en este formato de punto fijo también se pueden representar números.
- Limitaciones: “números muy grandes y números muy pequeños”.

# Números en punto flotante (2)

- Un número decimal “muy grande”:

  976.000.000.000.000

  se puede representar como:

  $9{,}76 \times 10^{14}$

- Un número decimal “muy pequeño”:

  0,00000000000000976

  $9{,}76 \times 10^{-14}$

# Números en punto flotante (3)

- Lo que hemos hecho es desplazar en forma dinámica la coma decimal a una posición conveniente y usar el exponente de base 10 para mantener la “pista” de la coma.
- Esto permite tener un rango de números desde “muy pequeños” a “muy grandes” y pueden ser representados con pocos dígitos.

# Números en punto flotante (4)

Veamos este mismo enfoque con números binarios:

- Un número se puede representar de la forma:

$$\pm M \times B^{\pm E}$$

- Este número se puede almacenar en una palabra binaria con dos campos:
  - Mantisa M
  - Exponente E

# Números en punto flotante (5)

- La base B es implícita y no necesita almacenarse ya que es la misma para todos los números. Debemos almacenar M y E.
- Se necesitan menos bits para almacenar M y E, que para almacenar el "número completo" en la base correspondiente.

# Números en punto flotante (6)

- M y E están representados en alguno de los sistemas en punto fijo que ya conocíamos como BSS, BCS, Ca2, Ca1, Exceso.

| exponente | mantisa |
|---|---|

La figura muestra un formato típico

# Ejemplo

- Supongamos el siguiente formato en punto flotante

$$\text{Mantisa} \begin{cases} \text{BSS} \\ \text{4 bits} \\ \text{entera} \end{cases} \qquad \text{Exponente} \begin{cases} \text{BSS} \\ \text{4 bits} \\ \text{entero} \end{cases}$$

Determinar el rango y resolución

# Ejemplo

- S ... siguier ... punto fl...

mantisa máxima 1111=15
mantisa mínima 0000=0

exp. máximo 1111=15
exp. mínimo 0000=0

Mantisa: BSS, 4 bits, entera

Exponente: BSS, 4 bits, entero

Determinar el rango y resolución

# Ejemplo 1

- Máximo = $1111 \times 2^{1111} = 15 \times 2^{15}$
- Mínimo = 0
- Rango = $[\,0,..,15\text{x}2^{15}]=[\,0,..,491520]$
- Resolución en el extremo superior

  $R = (15 - 14)\text{x}2^{15} = 1 \times 2^{15}$
- Resolución en el extremo inferior

  $R = (1 - 0)\text{x}2^{0} = 1$

# Ejemplo 2

Consideremos enteros de 8 bits y en BSS

Calcular el rango y resolución:

- Rango = [ 0,..,255 ]
- Resolución en el extremo superior

  R = 255 – 254 = 1

- Resolución en el extremo inferior

  R = 1 – 0 = 1

# Comparación

Si comparamos ambos ejemplos vemos:

- el rango en punto flotante es mayor
- la cantidad de combinaciones binarias distintas es la misma en ambos sistemas $2^8 = 256$
- en punto flotante la resolución no es constante a lo largo del intervalo, como lo es en el segundo ejemplo.

# Conclusión

- En el sistema de punto flotante el rango es mayor. Podemos representar números más grandes ó más pequeños que en un sistema de punto fijo (para igual cantidad de bits), pero pagamos el precio que los Nºs no están igualmente espaciados, como en punto fijo.

# Mantisa y exponente en Ca2

- Ejemplo: supongamos el siguiente formato en punto flotante

$$\text{Mantisa}\begin{cases}\text{Ca2}\\ \text{4 bits}\\ \text{entera}\end{cases}\qquad \text{Exponente}\begin{cases}\text{Ca2}\\ \text{4 bits}\\ \text{entero}\end{cases}$$

Determinar el rango y resolución

# Mantisa y exponente en Ca2

- Máximo = $0111 \times 2^{0111} = +7 \times 2^{+7}$
- Mínimo = $1000 \times 2^{0111} = -8 \times 2^{+7}$
- Rango = $[-8 \times 2^{+7},\ldots,+7 \times 2^{+7}]$
- Resolución en el extremo superior

  $R = (7 - 6) \times 2^{7} = 1 \times 2^{7}$
- Resolución en el origen

  $R = (1 \times 2^{-8} - 0) = 1 \times 2^{-8}$

# Mantisa fraccionaria

- Ejemplo: supongamos el siguiente formato en punto flotante

Mantisa
- BCS (ó MyS)
- 23 bits
- fraccionaria
- 1 bit signo

Exponente
- Ca2
- 8 bits
- entero

Determinar el rango y resolución

- 1 bit de signo y 23 bits para codificar el módulo o magnitud de la mantisa que es fraccionaria
- mantisa fraccionaria significa que los 23 bits "están después de la coma"

El módulo más grande será 0,1111....1 (23 unos)
El módulo más chico será 0,0000.....0 (23 ceros)

- E … formato en punto flotante

Mantisa: BCS (ó MyS), 23 bits fraccionaria, 1 bit signo

Exponente: Ca2, 8 bits entero

Determinar el rango y resolución

- 1 bit de signo y 23 bits para codificar el módulo o magnitud de la mantisa que es fraccionaria
- mantisa fraccionaria significa que los 23 bits "están después de la coma"

El módulo más grande será 0,1111....1 (23 unos)
El módulo más chico será 0,0000.....0 (23 ceros)

- E[…] formato en punto flotante

Mantisa $\left\{ \begin{array}{l} \text{BCS (ó MyS)} \\ \text{23 bits} \\ \text{fraccionaria} \\ \text{1 bit signo} \end{array} \right.$ Exponente $\left\{ \begin{array}{l} \text{Ca2} \\ \text{8 bits} \\ \text{entero} \end{array} \right.$

Ca2 8 bits

Exponente máximo +127 (números grandes, alejados del 0)
Exponente mínimo -128 (números pequeños, cercanos al 0)

De[…]

# Mantisa fraccionaria

- Máximo positivo

  $0 \;\; 0{,}111..111 \times 2^{01111111} = +(1-2^{-23}).2^{+127}$

- Mínimo positivo ($\neq 0$)

  $0 \;\; 0{,}000..001 \times 2^{10000000} = +(2^{-23}).2^{-128}$

- Máximo negativo ($\neq 0$)

  $1 \;\; 0{,}000..001 \times 2^{10000000} = -\,(2^{-23}).2^{-128}$

- Mínimo negativo

  $1 \;\; 0{,}111..111 \times 2^{01111111} = -(1-2^{-23}).2^{+127}$

# Mantisa fraccionaria

- Máximo positivo

$0 \quad 0{,}111..111 \times 2^{01111111} = +(1-2^{-23}).2^{+127}$

- Mínimo positivo ($\neq 0$)

$0 \quad 0{,}000..001 \times 2^{10000000} = +(2^{-23}$ … $28$

- Máximo negativo ($\neq 0$)

$1 \quad 0{,}0$

- Míni

$1 \quad 0{,}1$

0,0000.....001 un 1 en la posición 23 es $2^{-23}$
observar que eso es lo que le falta para llegar a 1,0 al
0,1111.....111 (23 unos)
es decir
0,1111.....111+ 0,0000...001 = 1
por lo tanto:
- 0,1111.....111 = 1 $-2^{-23}$

# Formato final

- El formato anterior se puede representar

| 0 | 1          8 | 9              31 |
|---|---|---|
| S | Exponente | Mantisa |

- El mínimo negativo es

| | | |
|---|---|---|
| 1 | 01111111 | 1111……………………………………11 |

# Normalización

Veamos el siguiente ejemplo:

$$40x10^{0} = 4x10^{1} = 0,4x10^{2} = 400x10^{-1}$$

- Existen distintos valores de mantisa y exponente para representar un mismo número.
- Lo mismo sucede en base 2.
- Con el objetivo de tener un único par de valores de mantisa y exponente para un número, se introduce la *normalización*.

# Normalización

- Con el objetivo anterior, las mantisas fraccionarias se definen de la forma:

  0,1dddddd.....ddd

  - donde d es un dígito binario que vale 0 ó 1.

- Todas las mantisas empiezan con 0,1...

# Normalización

- Con el objetivo anterior, las mantisas fraccionarias se definen de la forma:

  0,1dddddd.....ddd

  - donde d es un dígito binario que vale 0 ó 1.

Observar que no tendremos el 0,
ya que 0,0000...00 no es una mantisa válida

- Todas las mantisas empiezan con 0,1...

# Normalización

- Ejemplo: formato en punto flotante

Mantisa
- BCS
- 23 bits
- fraccionaria
- 1 bit signo
- Normalizada

Exponente
- Exceso
- 8 bits
- entero

Determinar el rango y resolución

# Normalización

- Máximo positivo

  $0 \quad 0{,}111..111 \times 2^{11111111} = +(1-2^{-23}).2^{+127}$

- Mínimo positivo

  $0 \quad 0{,}100..000 \times 2^{00000000} = +(0{,}5).2^{-128}$

- Máximo negativo

  $1 \quad 0{,}100..000 \times 2^{00000000} = -(0{,}5).2^{-128}$

- Mínimo negativo

  $1 \quad 0{,}111..111 \times 2^{11111111} = -(1-2^{-23}).2^{+127}$

# Normalización

- Máximo positivo

  0 $0{,}111..111 \times 2^{11111111} = +(1-2^{-23}).2^{+127}$

- Mínimo positivo

  0 $0{,}100..000 \times 2^{00000000} = +(0{,}5).2^{-128}$

Observar que no tendremos el 0, ya que 0,0000...00 no es una mantisa válida

- Máximo negativo

  1 $0{,}100..000 \times 2^{00000000} = -(0{,}5).2^{-128}$

- Mínimo negativo

  1 $0{,}111..111 \times 2^{11111111} = -(1-2^{-23}).2^{+127}$

# Normalización

- El formato anterior se puede representar

| 0 | 1    8 | 9    31 |
|---|---|---|
| S | Exponente | Mantisa |

- El máximo negativo es

| | | |
|---|---|---|
| 1 | 00000000 | 1000.........................................00 |

# Bit implícito

- Como todos los números comienzan con 0,1 ¿es necesario almacenar el 1?
  - siempre está presente !!!
- Si no lo almaceno, puedo “adicionar” un bit más a la mantisa. El bit no almacenado se conoce como *bit implícito.*

| | | |
|---|---|---|
| 1 | 00000000 | 1000.................................000 |

# Bit implícito

- Como todos los números comienzan con 0,1 ¿es necesario almacenar el 1?
  - siempre está presente !!!
- Si no lo almaceno, puedo “adicionar” un bit más a la mantisa. El bit no al conoce como *bit implícito.*

Agregamos este bit, ya que tenemos 23 bits

Este bit NO lo guardamos

| | | |
|---|---|---|
| 1 | 00000000 | 1000.................................000 |

# Bit implícito

- Como todo ... nienzan con 0,1 ¿es necesa ... ?
  - siempre e ...
- Si no lo alı ... dicionar" un bit más a la m ... *bit impli ... .*

Es decir que ahora el módulo de la mantisa tiene 24 bits.
Si antes teníamos:
0,1111......11= $1 -2^{-23}$
Ahora tenemos:
0,1111......111= $1 -2^{-24}$

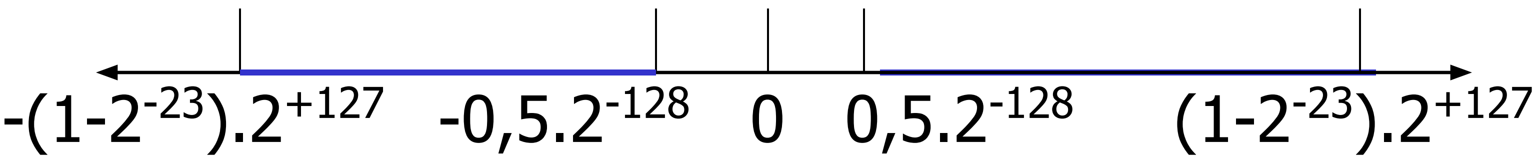
Agregamos este bit, ya que tenemos 23 bits

Este bit NO lo guardamos

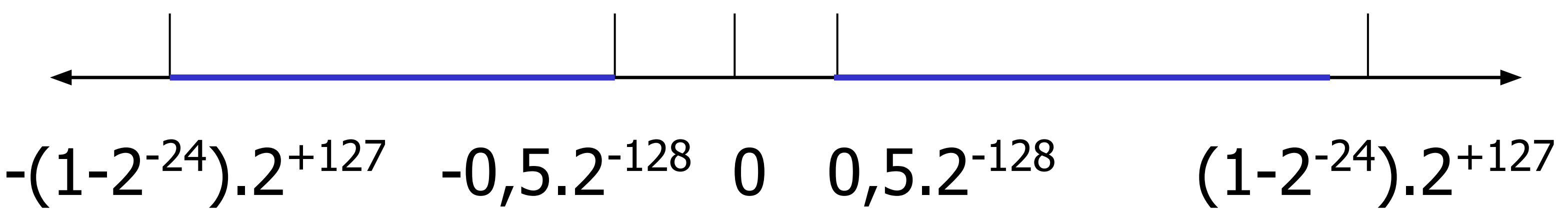
| | | |
|---|---|---|
| 1 | 00000000 | 1000.................................000 |

# Recta numérica

- Sin bit implícito




- Con bit implícito



# ¿Cómo se escribe un Nº en punto flotante normalizado?

1. Se escribe el Nº en el sistema propuesto para la mantisa.
2. Se desplaza la coma y se cambia el exponente hasta obtener la forma normalizada.
3. Se convierte el exponente al sistema propuesto para él.

# ¿Cómo......? (2)

Ej. - 13,5 . Formato anterior

- 1) 1  1101,100..0=1  1101,100..0x$2^0$
- 2) 1  0,110110..0 x $2^4$
- 3) 4 en Ca2=00000100

  4 en Exceso=10000100
- Finalmente

# ¿Cómo...... ? (3)

- Sin bit implícito

| | | |
|---|---|---|
| 1 | 10000100 | 1101100000.................00 |

- Con bit implícito

| | | |
|---|---|---|
| 1 | 10000100 | 101100000...................00 |

# Resolución – Error absoluto

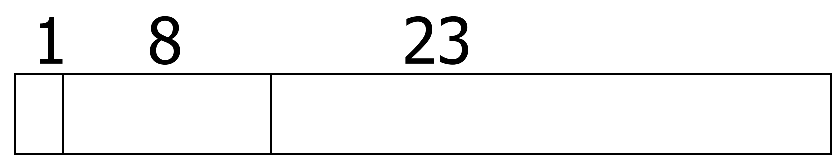
- Resolución: es la diferencia entre dos representaciones sucesivas, y varía a lo largo del rango, no es constante como en el caso de punto fijo.
- Error Absoluto: es la diferencia entre el valor representado y el valor a representar

# Error absoluto y relativo

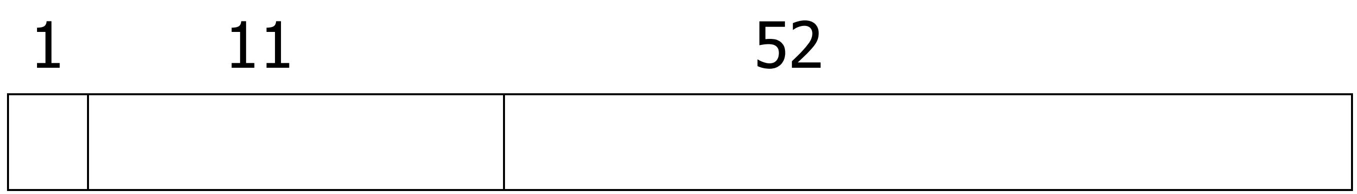
- Error Absoluto máximo ≤ Resolución/2
- Error Relativo = EA/Número a representar

# Estándar IEEE 754

- Simple precisión




- Doble precisión



# Estándar IEEE 754

- Mantisa: fraccionaria normalizada, con la coma después del primer bit que es siempre uno (1,) en M y S.
- Exponente: representado en exceso

$$2^{n-1} - 1$$

# Estándar IEEE 754

- Mantisa: fraccionaria normalizada, con la coma después del primer bit que es siempre u

En simple precisión:
exponentes 8 bits=> exceso 127
(observar que no es exceso 128!!)

- Exponente: representado en exceso

$$2^{n-1} - 1$$

# Estándar IEEE 754

| Precisión | Simple | Doble |
|---|---|---|
| Bits en signo | 1 | 1 |
| Bits en exponente | 8 | 11 |
| Bits en fracción | 23 | 52 |
| Total de bits | 32 | 64 |
| Exponente en exceso | 127 | 1023 |
| Rango de exponente | -126 a +127 | −1022 a +1023 |
| Rango de números | $2^{-126}$ a $\sim 2^{128}$ | $2^{-1022}$ a $\sim 2^{1024}$ |

# Estándar

Observar que en exceso 127, el rango debería ser: -127 a +128
Sin embargo, dice -126 a +127

Esto es porque las combinaciones binarias correspondientes a -127 (00000000) y +128 (11111111) están reservadas para indicar casos especiales.

| | | |
|---|---|---|
| Precisión | | |
| Bits en signo | | |
| Bits en exponent | | |
| Bits en fracción | 23 | 2 |
| Total de bits | 32 | 64 |
| Exponente en exceso | 127 | 1023 |
| Rango de exponente | -126 a +127 | −1022 a +1023 |
| Rango de números | $2^{-126}$ a $\sim 2^{128}$ | $2^{-1022}$ a $\sim 2^{1024}$ |

# Ejemplo 1 en simple precisión

¿Qué valor representa el hexadecimal 3F800000?

0011 1111 1000 0000 0000 0000 0000 0000

01111111=127 en exceso 127 representa 0

00000000000000000000000=0

$+\ 1{,}0 \times 2^{0} = 1$

# Ejemplo 2 en simple precisión

¿Qué valor representa el hexadecimal C0066666?

1100 0000 0000 0110 0110 0110 0110 0110

10000000=128 en exceso 127 representa 1

00001100110011001100110=0,05

$-\ 1{,}05 \times 2^{1} = -2{,}1$

# Ejemplo 3 en simple precisión

Numero máximo positivo

0 11111110 1111.....1

11111110 = 254 en exceso 127 es +127

1111.....1= 0,1111.....1= $1-2^{-23}$

$+(1.0+1-2^{-23})x2^{+127} = (2- 2^{-23})x2^{+127}$

# Ejemplo 4 en simple precisión

Número mínimo positivo

0 00000001 0000.....0

00000001 = 1 en exceso 127 es -126

0000.....0= 0,000.....00

$+(1.0)\times 2^{-126} = 2^{-126}$

# Estándar IEEE 754

Casos especiales:

- $E = 255/2047$, $M \neq 0 \Rightarrow$ NaN -Not a Number-
- $E = 255/2047$, $M = 0 \Rightarrow$ Infinito
- $E = 0$, $M = 0 \Rightarrow$ Cero
- $E = 0$, $M \neq 0 \Rightarrow$ Denormalizado
  - $\pm$ 0,mantisa_s-p $2^{-126}$
  - $\pm$ 0,mantisa_d-p $2^{-1022}$

# Estándar IEEE 754

Casos esp

- E = 255
  Number-
- E = 255
- E = 0, M
- E = 0, M ≠ 0 ⇒ Denormalizado
  - ± 0,mantisa_s-p $2^{-126}$
  - ± 0,mantisa_d-p $2^{-1022}$

**Denormalizado simple precisión**

Máximo número positivo denormalizado
0 00000000 111......1 = 0,111....11 x $2^{-126}$=
=$(1 - 2^{-23})$x $2^{-126}$

Mínimo número positivo denormalizado
0 00000000 0000.....1 = 0,000....01 x $2^{-126}$=
= $2^{-23}$x $2^{-126}$= $2^{-149}$

# Operaciones aritméticas en pf

Sumar y restar

- Comprobar valores cero.
- Ajuste de mantisas (ajuste de exponentes).
- Sumar o restar las mantisas.
- Normalizar el resultado.

# Operaciones aritméticas… (2)

Multiplicar y dividir

- Comprobar valores cero.
- Sumar y restar exponentes.
- Multiplicar y dividir mantisas
  - tener en cuenta el signo
- Normalizar.
- Redondear.

Todos los resultados intermedios deben doblar su longitud al almacenarse

# mayor información …

- Punto flotante
  - Apunte 2 de Cátedra
  - PFI-PFO. Software en Descargas del sitio de cátedra
- Capítulo 8: Aritmética del computador (8.4., 8.5.)
  - Stallings, W., 5º Ed.

- Link de interés
  - https://babbage.cs.qc.cuny.edu/IEEE-754/