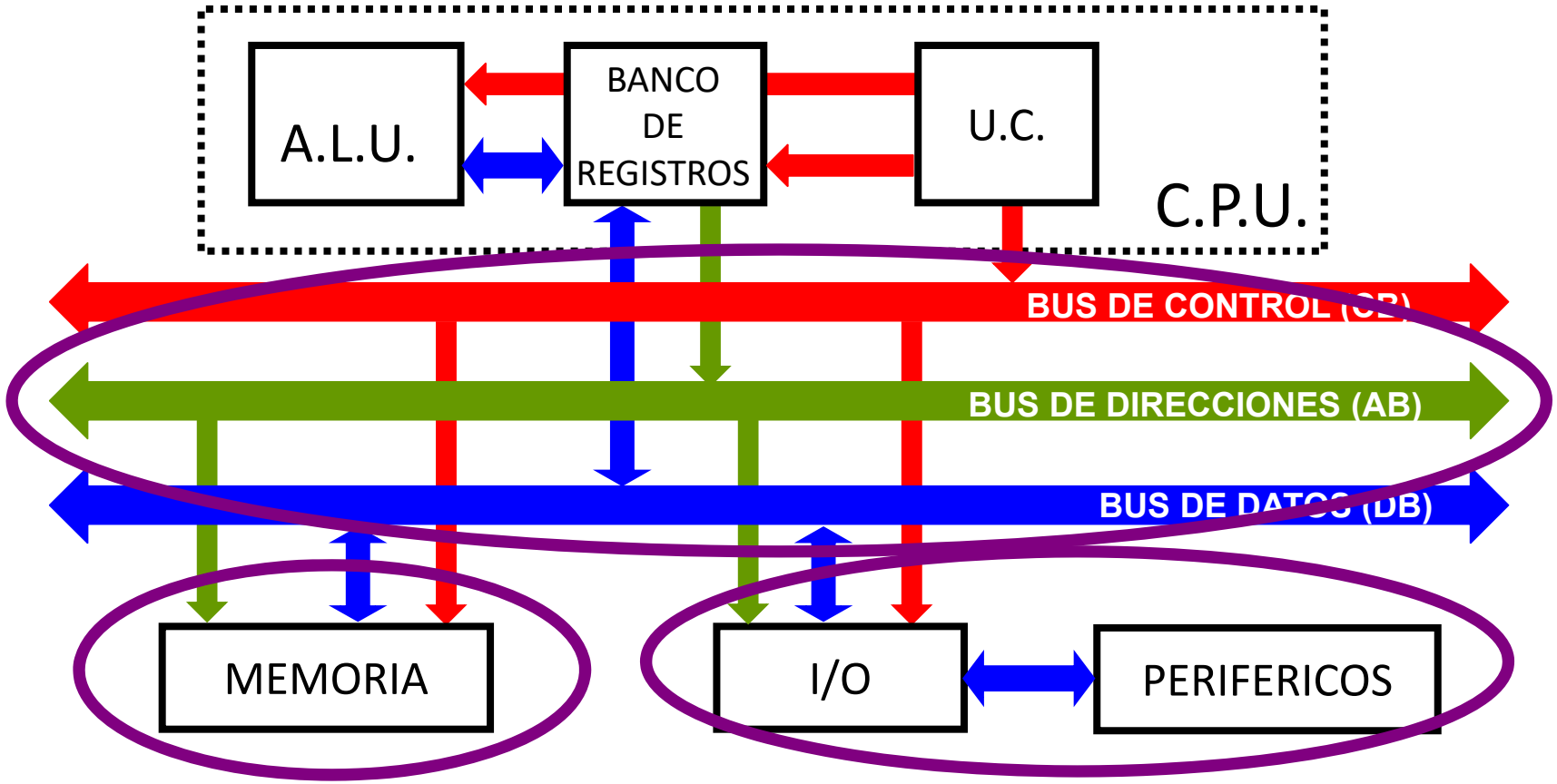
# Estructura de Computadores

## *Unidad 6. Mapas de memoria. Operaciones con Reales*

Escuela Politécnica Superior - UAM

# Índice

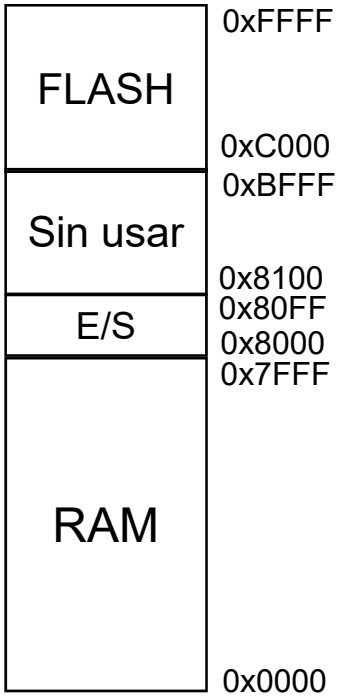
# Arquitectura clásica Von NEUMANN



# Interfaz entre los dispositivos de E/S y el procesador

**¿Cómo se accede desde el micro a la información de los dispositivos de E/S?**

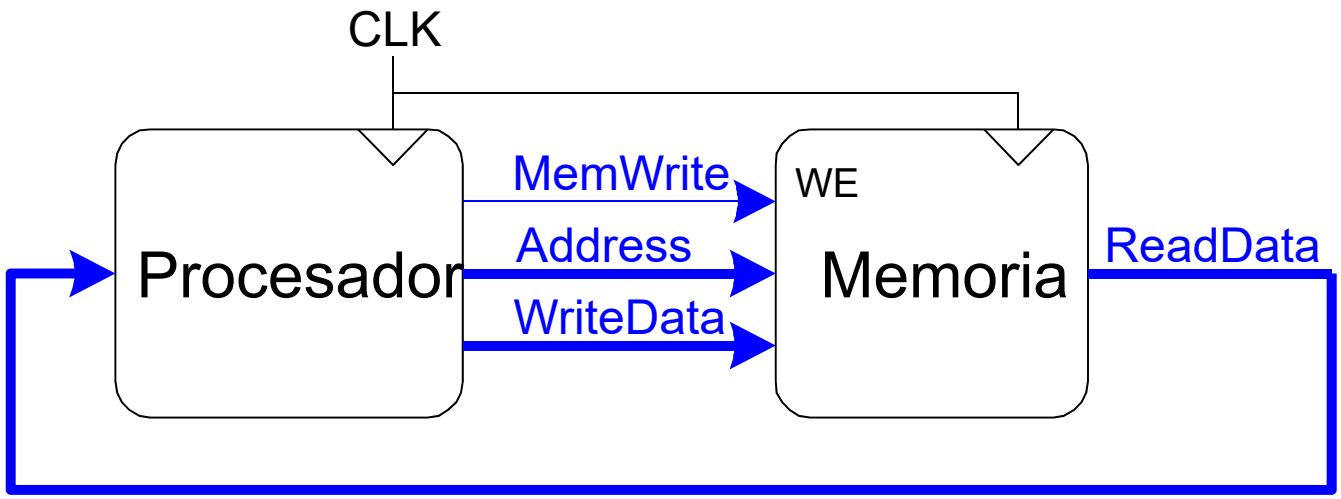
- ✓ El micro sólo accede al exterior por el bus, así que para él sólo hay direcciones distintas
- ✓ Los dispositivos de E/S se mapean sobre el espacio de direcciones: mapa de direcciones
- ✓ Al leer ciertas direcciones se consigue la información de E/S, y al escribir sobre otras se envía información a E/S




Mapa de memoria
(64 kbytes)

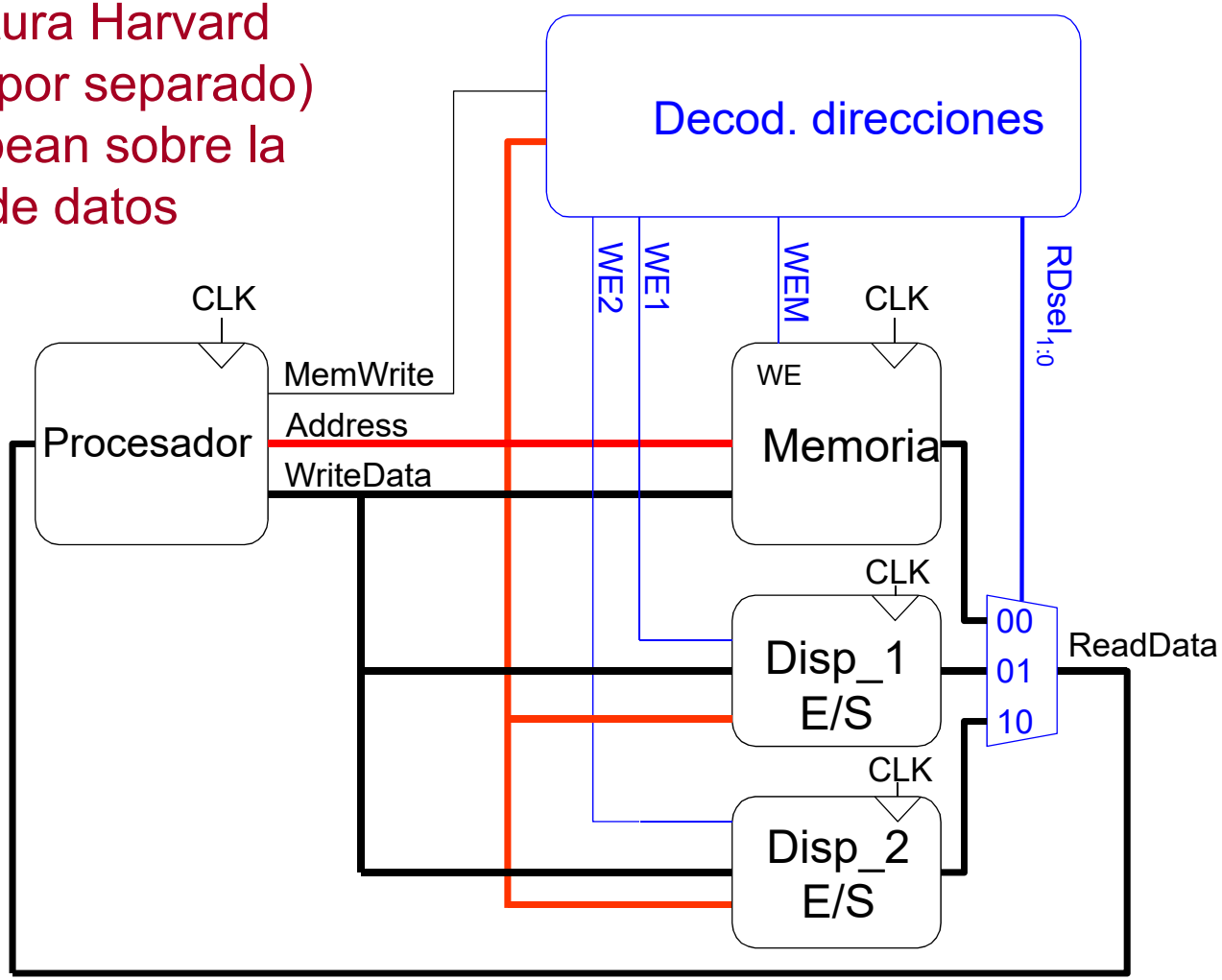
# Esquema de acceso a memoria: sin E/S

Suponiendo arquitectura Von Neumann con un solo bus (código y datos en la misma memoria lógica, aunque sean chips distintos)



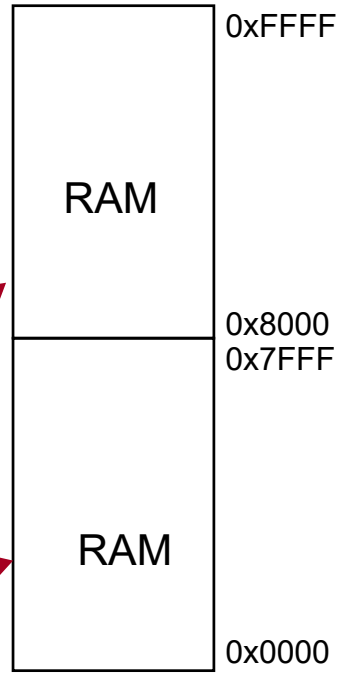
# Esquema de acceso a memoria: con E/S (decodificador direcciones)

Si es arquitectura Harvard (código y datos por separado) las E/S se mapean sobre la memoria de datos



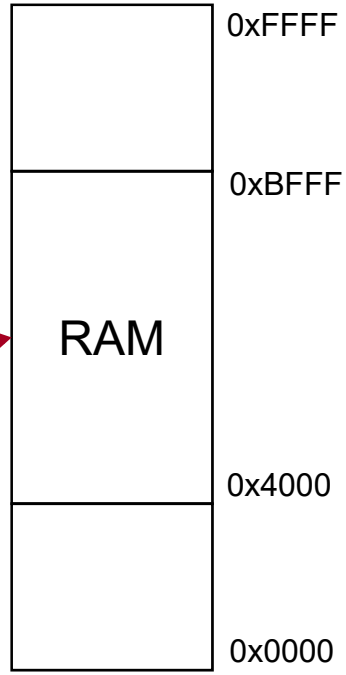
# Bloques alineados y no alineados

- Cada conjunto de direcciones que representa a un mismo dispositivo se denomina bloque. Normalmente el tamaño de cada bloque en bytes es una potencia de dos.
- Un **bloque está alineado** si todos los bits no implicados en su direccionamiento interno están fijos. La decodificación entonces está minimizada.
  - ✓ Ejemplo: Microprocesador byte-direccionable de 64 kBytes de espacio de memoria. Necesita 16 bits para la direccionar.
  - ✓ 3 Bloques: RAM de 32 kB, Flash de 16 kB y E/S 256 direcciones.
  - ✓ El bloque RAM de 32 kB usa 15 bits para su direccionamiento interno: estará alineado si el bit que sobra está fijo:
    - ✓ 1XXX XXXX XXXX XXXX ($8000_{16}$ a $FFFF_{16}$): $A_{15} = 1$
    - ✓ 0XXX XXXX XXXX XXXX ($0000_{16}$ a $7FFF_{16}$): $A_{15} = 0$



# Bloques alineados y no alineados

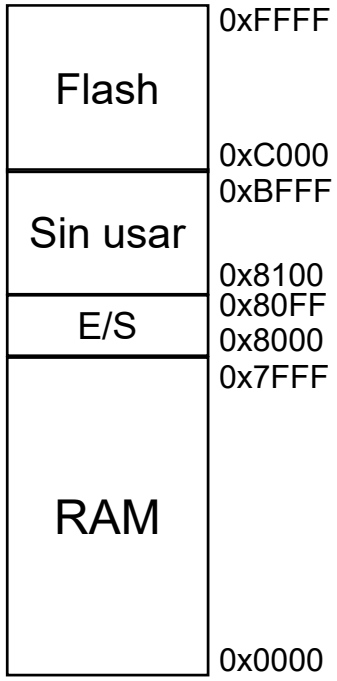
- Si el bloque **no está alineado**, la decodificación se complica:
  - ✓ ($4000_{16}$ a $BFFF_{16}$): $A_{15}A_{14}$=01 or $A_{15}A_{14}$=10
  - ✓ En realidad utiliza dos espacios de 16 kB (14 bits) yuxtapuestos:
  - ✓ 01XX XXXX XXXX XXXX ($4000_{16}$ a $7FFF_{16}$)
  - ✓ 10XX XXXX XXXX XXXX ($8000_{16}$ a $BFFF_{16}$)




- El circuito decodificador de direcciones (combinacional), genera las señales de entrada de habilitación (*enable*) para cada bloque (CE, *Chip Enable* y/o WE, *Write Enable*) y de control del multiplexor que elige entre los datos de salida de cada bloque.

# Mapeo completo/incompleto

- 3 Bloques: RAM de 32 kB (15 bits), Flash de 16 kB (14 bits) y E/S 256 direcciones (8 bits).
  - ✓ RAM: ($0000_{16}$ a $7FFF_{16}$) => 0XXX XXXX XXXX XXXX
  - ✓ Flash: ($C000_{16}$ a $FFFF_{16}$) => 11XX XXXX XXXX XXXX
  - ✓ E/S: ($8000_{16}$ a $80FF_{16}$) => 1000 0000 XXXX XXXX
  - ✓ Sin usar (libre): ($8100_{16}$ a $BFFF_{16}$)




- Decodificación:
  - ✓ RAM: $A_{15} = 0$
  - ✓ Flash: $A_{15}A_{14} = 11$
  - ✓ E/S, mapeo completo: $A_{15}A_{14}A_{13}A_{12}\ A_{11}A_{10}A_9A_8 = 1000\ 0000$
  - ✓ E/S, mapeo incompleto: $A_{15}A_{14} = 10$ (con esto ya se distingue)

**¿Y si el micro pone una dirección libre?** ($0x9000_{16}$)

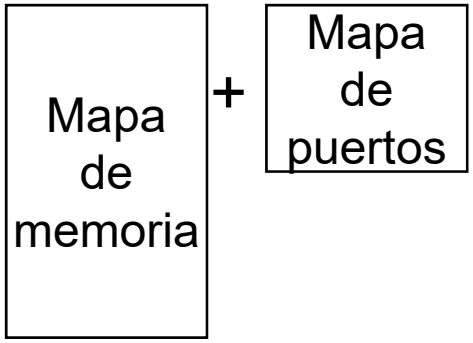
- ✓ En mapeo completo no habilitará nada
- ✓ En mapeo incompleto creerá que es otro dispositivo (en este caso E/S)

# Mapeo en memoria vs Mapeo en puertos
## (*Memory-mapped vs port-mapped*)

P

- La mayoría de las arquitecturas mapean E/S en el único espacio de direcciones (mapa de memoria).
  - ✓ Para leer de E/S se usa **load**
  - ✓ Para escribir en E/S se usa **store**


- Otras (p.ej. IA-32, conocida como x86) usan otro espacio de direcciones (mapa de puertos).
  - ✓ Hay un pin adicional que indica si la dirección es de memoria o de puertos (M/$\overline{IO}$)
  - ✓ **load** y **store** usan la memoria (M/$\overline{IO}$ = '1')
  - ✓ Otras instrucciones (p.ej. **in** y **out**) usan los puertos (M/$\overline{IO}$ = '0')



# Índice

# Sistemas de numeración

- Tipos de números representables de forma binaria:
  - **Números positivos**
    - Binario sin signo (*unsigned binary*)
  - **Números negativos**
    - Complemento a 2 (*two's complement*)
    - Signo/magnitud (*sign/magnitude*)
  - **Números fraccionarios**
    - Coma fija (*fixed-point*)
    - Coma flotante (*floating-point*). IEEE-754:
      - Precisión simple (32 bits)
      - Doble precisión (64 bits)

# Números en coma fija

- Representación en coma fija de 6,75 con 4 bits para la parte entera y 4 para la parte fraccionaria:

01101100

0 1 1 0.1 1 0 0

8 4 2 1 0,5 0,25 0,125 0,0625

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$$

- La coma no se representa, sino que va implícita.
- El número de bits para cada parte se tiene que acordar en cada ocasión entre los que generan el número y lo leen.

# Signo en coma fija

- Como los enteros, tienen dos formas de representación:
  - Signo/magnitud
  - Complemento a 2
- Representar $-7{,}5_{10}$ usando 8-bits, 4 para la parte entera y 4 para la parte fraccionaria.
  - Signo/magnitud: 1111.1000
  - Complemento a 2:

| | |
|---|---|
| 1. +7.5: | 0111.1000 |
| 2. Invertir bits: | 1000.0111 |
| 3. Sumar 1: | + 1 |
| | 1000.1000 |

# Números en coma flotante

- La coma se pone a la derecha del 1 más significativo.
- Similar a notación científica en decimal.
- Por ejemplo, $273_{10}$ en notación científica:

$$\mathbf{273 = 2{,}73 \times 10^2}$$

- En general, un número se representa como:

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

donde,

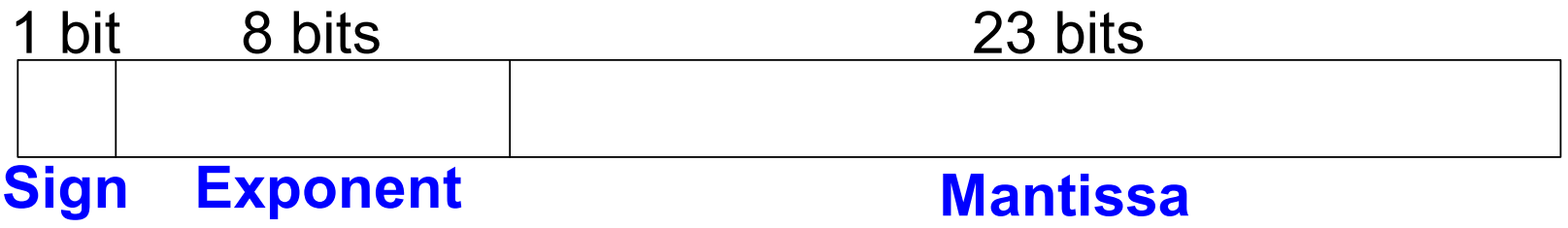
  - **M** = mantissa **B** = base **E** = exponent
  - En el ejemplo, M = 2,73; B = 10; E = 2

# Coma flotante IEEE-754, precisión

- Precisión simple (*single-precision*):
  - 32-bits en total
  - 1 bit de signo, 8 de exponente y 23 de mantisa
  - sesgo = 127 (sesgado, **positiva**)
- Precisión doble (*double-precision*):
  - 64-bits en total
  - 1 bit de signo, 11 de exponente y 52 de mantisa
  - sesgo = 1023 (sesgado, **positiva**)

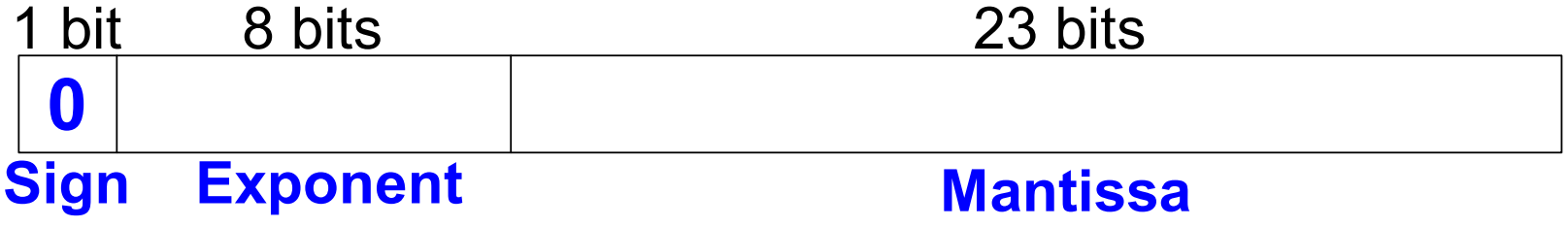
**Nota:** El exponente se define en binario sin signo. Por tanto, la forma de crear exponentes negativos es creando un sesgo igual a $2^{(exp-1)}-1$. Por ejemplo, en *single-precision*, el sesgo es $2^{(8-1)}-1 = 127$. Por tanto, un exponente de "00000000"(0) realmente representa el valor -127. Y un exponente de "11111111"(255) representa el valor +128.

# Números en coma flotante (IEEE-754)




**Ejemplo:** representar $228_{10}$ en CF en el estándar IEEE-754 (32bits)

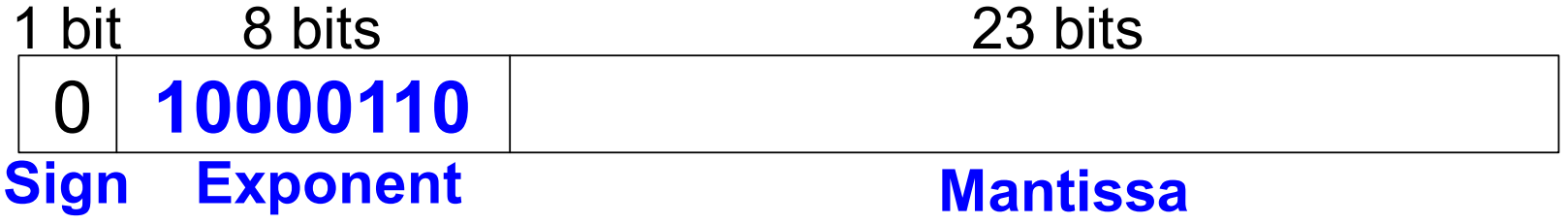
- Convertir el número decimal a binario:
  - $228_{10} = 11100100_2 = 1{,}11001 \times 2^7$
- Rellenar cada parte del número (s):
  - Bit de signo positivo (0)



# Números en coma flotante (IEEE-754)

**Ejemplo:** representar $228_{10}$ en CF en el estándar IEEE-754 (32bits)

- Convertir el número decimal a binario:
  - $228_{10} = 11100100_2 = 1{,}11001 \times 2^7$
- Rellenar cada parte del número (e):
  - Exponente con sesgo = sesgo + exponente
  - Sesgo => 127 = $01111111_2$
  - El exponente, 7, se almacena como: 127+7 = 134 = $10000110_2$



# Números en coma flotante (IEEE-754)

**Ejemplo:** representar $228_{10}$ en CF en el estándar IEEE-754 (32bits)

- Convertir el número decimal a binario:
  - $228_{10} = 11100100_2 = 1{,}11001 \times 2^7$
- Rellenar cada parte del número (m):
  - El primer bit de la mantisa siempre es 1:
  - Así que guardar ese 1, llamado el 1 implícito, es redundante.
  - Por tanto, ese primer 1 no se almacena y hay más sitio para la parte fraccionaria.

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 0 | 10000110 | **11001000000000000000000** |
| **Sign** | **Exponent** | **Mantissa** |

- En hexadecimal: 0x43640000

# Números en coma flotante (IEEE-754)

**Ejemplo**: representar $-58{,}25_{10}$ en el estándar IEEE 754 de 32-bits.

- Primero, convertir de decimal a binario y notación científica:
  - $58{,}25_{10} = 111010{,}01_2 = 1{,}1101001 \times 2^5$
- Después, hallar los tres campos del número:
  - Bit de signo: 1 (negativo)
  - Exponente sesgado en 8 bits: $(127 + 5) = 132 = 10000100_2$
  - 23 bits de mantisa: 110 1001 0000 0000 0000 0000

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 1 | 100 0010 0 | 110 1001 0000 0000 0000 0000 |
| **Sign** | **Exponent** | **Fraction** |

- En hexadecimal: 0xC2690000

# Coma flotante IEEE-754, casos especiales

P

- El estándar IEEE 754 incluye casos especiales para números díficiles de representar. Por ejemplo, el 0 no tiene un bit 1 para el 1 implícito.

| Número | Signo | Exponente | Mantisa |
|---|---|---|---|
| 0 | X | 00000000 | 00000000000000000000000 |
| ∞ | 0 | 11111111 | 00000000000000000000000 |
| - ∞ | 1 | 11111111 | 00000000000000000000000 |
| NaN | X | 11111111 | Distinto de cero |

NaN (*Not a Number*) se usa para números que no existen, como por ejemplo √-1 o log(-5).

# Suma en coma flotante

1. Extraer bits de exponentes y mantisas
2. Añadir el 1 por la izquierda en las mantisas
3. Comparar exponentes
4. Desplazar la mantisa menor (si hace falta)
5. Sumar mantisas
6. Normalizar la mantisa y ajustar exponente (si hace falta)
7. Redondear resultado
8. Componer el resultado con signo, exponente y mantisa

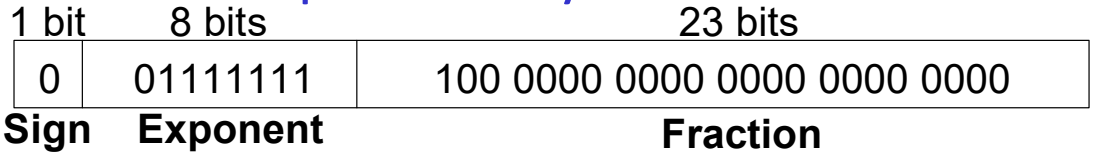
# Suma en coma flotante: ejemplo

Ejemplo. Sumar los siguientes números en coma flotante:

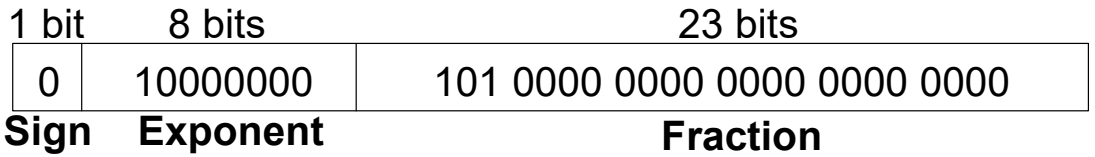
N1 = 0x3FC00000 y N2 = 0x40500000

1. Extraer bits de exponentes y mantisas

N1:




N2:




Para N1: S = 0, E = 127, F = .1

Para N2: S = 0, E = 128, F = .101

2. Añadir el 1 por la izquierda en las mantisas

N1: 1.1

N2: 1.101

# Suma en coma flotante: ejemplo

3. Comparar exponentes

   127 – 128 = -1, así que se desplaza N1 a la derecha 1 posición

4. Desplazar la mantisa menor (si hace falta)

   Desplazar la mantisa de N1: 1.1 >> 1 = 0.11 $(\times 2^1)$

5. Sumar mantisas

$$\begin{array}{r} 0.11 \times 2^1 \\ \underline{+1.101 \times 2^1} \\ 10.011 \times 2^1 \end{array}$$

6. Normalizar la mantisa y ajustar exponente (si hace falta)

   $10.011 \times 2^1 = 1.0011 \times 2^2$

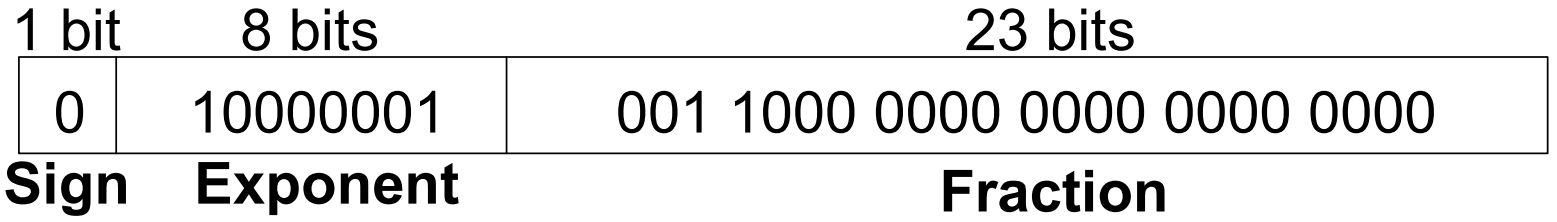
# Suma en coma flotante: ejemplo

7. Redondear resultado

   No hace falta (cabe en 23 bits)

8. Componer el resultado con signo, exponente y mantisa

   S = 0, E = 2 + 127 = 129 = $10000001_2$, F = 001100..




En hexadecimal: 0x40980000

0x3FC00000 + 0x40500000 = 0x40980000

# Multiplicación en coma flotante

1. Extraer bits de exponentes y mantisas
2. Añadir el 1 por la izquierda en las mantisas
3. Sumar exponentes sin sesgo
4. Multiplicar mantisas
5. Normalizar la mantisa y ajustar exponente (si hace falta)
6. Redondear resultado
7. Renormalizar la mantisa si hace falta tras el redondeo
8. Componer el resultado con signo, exponente y mantisa

# Multiplicación en coma flotante: ejemplo

Ejemplo. Multiplicar los siguientes números en coma flotante:

N1 = 0x3FC00000 y N2 = 0xC0500000

1. Extraer bits de exponentes y mantisas

N1:

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 0 | 01111111 | 100 0000 0000 0000 0000 0000 |
| **Sign** | **Exponent** | **Fraction** |

N2:

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 1 | 10000000 | 101 0000 0000 0000 0000 0000 |
| **Sign** | **Exponent** | **Fraction** |

Para N1: S = 0, E = 127, F = .1

Para N2: S = 1, E = 128, F = .101

2. Añadir el 1 por la izquierda en las mantisas

N1: 1.1

N2: 1.101

# Multiplicación en coma flotante: ejemplo

3. Sumar exponentes sin sesgo

   127 => 0; 128 => 1; 1 + 0 = 1 (será 128 con sesgo)

4. Multiplicar mantisas

$$\begin{array}{r} 1.101 \\ \times\ 1.1 \\ \hline 0.1101 \\ +\ 1.101 \\ \hline 10.0111 \end{array}$$

5. Normalizar la mantisa y ajustar exponente (si hace falta)

   $10.0111 \times 2^1 = 1.00111 \times 2^2$

6. Redondear resultado

   No hace falta (cabe en 23 bits)

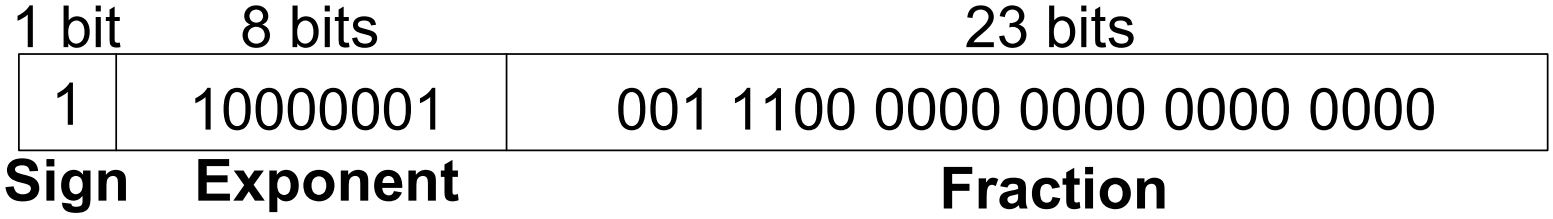
# Multiplicación en coma flotante: ejemplo

7. Renormalizar la mantisa si hace falta tras el redondeo

   No hace falta (sigue siendo 1,M)

8. Componer el resultado con signo, exponente y mantisa

   $S = S_{N1} \oplus S_{N\,2} = 1$; $E = 2+127 = 129 = 10000001_2$; $F = 0011100..$




En hexadecimal: 0xC09C0000

0x3FC00000 x 0xC0500000 = 0xC09C0000

# Ayuda para IEEE-754

- Conversor IEEE-754 a decimal:
  - http://babbage.cs.qc.cuny.edu/IEEE-754.old/32bit.html
- Conversor decimal a IEEE-754
  - http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html
- Calculadora para números en formato IEEE-754
  - http://www.hctlab.com/ieee754/

# *Unidad 6. Mapas de memoria. Operaciones con Reales*

Escuela Politécnica Superior - UAM

# Problemas U6

**6.2.** El mapa de memoria de un sistema controlado por microprocesador es el indicado en la tabla adjunta. Se pide realizar la lógica del decodificador de direcciones. Como memoria RAM se utilizan pastillas de 2 kBytes con entradas de control R/W y Selector de Chip (CSx). Las ROM son de 4 kBytes y una única línea de control CSx. Los puertos de E/S son tratados a todos los efectos como memorias RAM, se supone que para el acceso a los circuitos de E/S existe un único bit de selección para todo el bloque. Se pide crear el mapa de direcciones para lo siguientes casos:

**a)** Utilizando mapeo completo: **a1)** suponiendo los CSx activos en alto ('1') y **a2)** suponiendo los CSx activos en bajo ('0')

**b)** Utilizando mapeo incompleto: **b1)** suponiendo los CSx activos en alto ('1') y **b2)** suponiendo los CSx activos en bajo ('0')

| **Dirección (hexa)** | | **Tipo** | **Tamaño** | **Bit de selección** | |
|---|---|---|---|---|---|
| **Inicial** | **Final** | | | (1) | (2) |
| 0000 | 07FF | RAM Sistema | 2 kBytes | CS1 | /CS1 |
| 0800 | 0FFF | RAM Usuario | 2 kBytes | CS2 | /CS2 |
| Zona no utilizada | | | | | |
| B000 | BFFF | Puertos de E/S | 4 kBytes | CS5 | /CS5 |
| Zona no utilizada | | | | | |
| E000 | EFFF | ROM Tablas | 4 kBytes | CS3 | /CS3 |
| F000 | FFFF | ROM Programa | 4 kBytes | CS4 | /CS4 |

# Problemas U6

**6.4.** Un sistema con un bus de direcciones de 20 bits, dispone de un mapa de memoria de cinco elementos tal y como indica la tabla adjunta. Se pide:
a) Completar la tabla con los valores adecuados. Se conoce que el dispositivo MEM4 ocupa las posiciones de memoria consecutivas a MEM1 y que MEM5 puede ocupar cualquier bloque alineado.
(**Nota:** Se considerará válida cualquiera de las zonas elegidas)
b) Calcular las ecuaciones de selección mínimas para los cinco elementos.

| Pastilla | Bit de selección | Capacidad | Dirección (hexa) | |
|---|---|---|---|---|
| | | | Inicial | Final |
| **MEM1** | **/CS1** | | $20000_{16}$ | $3FFFF_{16}$ |
| **MEM2** | **/CS2** | 64 kBytes | $90000_{16}$ | |
| **MEM3** | **/CS3** | 32 kBytes | | $E7FFF_{16}$ |
| **MEM4** | **/CS4** | 256 kBytes | | |
| **MEM5** | **/CS5** | 128 kBytes | | |

# Problemas U6

**6.8.** Se desea diseñar un sistema empotrado basado en un procesador MIPS, con una memoria ROM que contiene el firmware, un bloque RAM, y varios bloques destinados a dispositivos periféricos. Se conoce que el procesador es capaz de manejar un bus de direcciones de 32 bits. Se ofrece su mapa de direcciones incompleto.

| Pastilla | Bit de selección | Capacidad | Dirección (hexa) | |
|---|---|---|---|---|
| | | | Inicial | Final |
| **ROM** | CS1 | | 0x00000000 | 0x3FFFFFFF |
| **RAM** | /CS2 | 1 Gbyte | | |
| **Periférico 1** | /CS3 | 256 Mbytes | 0x40000000 | |
| **Periférico 2** | CS4 | 128 Mbytes | | 0xC7FFFFFF |

a. Rellene el mapa de memoria de la tabla anterior. El bloque de RAM debe tener 1 GB de capacidad, pero puede ir en cualquier lugar del mapa mientras que el bloque esté alineado.
b. Indique las ecuaciones de CS1 y /CS3**,** utilizando el mapeo completo de direcciones.
c. Indique las ecuaciones más reducidas posibles (mapeo incompleto) de /CS2, /CS3 y CS4, entendiendo que deben ser ecuaciones válidas para seleccionar el mapa completo.

**Nota:** Para referirse a los bits de direcciones utilice la notación $A_0 \ldots A_{31}$ donde $A_0$ es el bit menos significativo y $A_{31}$ el bit más significativo

# Problemas U6

**6.11.** Dado el número escrito en decimal N = - 954,625, se pide, escribir el número en formato hexadecimal, suponiendo que:

a. Es un número de 16 bits con representación signo-magnitud y en coma fija, con 4 bits para la parte fraccionaria.
b. Es un número de 16 bits con representación en complemento a 2 y en coma fija, con 4 bits para la parte fraccionaria.
c. Es un número de 32 bits escrito con el estándar IEEE-754.

**6.14.** Dados dos números reales X e Y, ambos escritos en hexadecimal, el primero X = 0x635A está escrito en un formato en complemento a 2 con notación en coma fija con 4 bits para la parte fraccionaria. El segundo Y = 0xC2B8A000, en un formato de coma flotante según el estándar IEEE-754. Se pide:

Realice la conversión del número Y al mismo sistema de representación para números reales en binario utilizado para el número X.

Realice la operación de Suma = X+Y, y muestre el resultado en hexadecimal para 16 bits y con representación en complemento a 2 con notación en coma fija con 4 bits para la parte fraccionaria. Señale, justificando breve y necesariamente la respuesta, si el resultado es correcto o incorrecto.

Realice la operación de Resta = X-Y, y muestre el resultado en hexadecimal para 16 bits y con representación en complemento a 2 con notación en coma fija con 4 bits para la parte fraccionaria. Señale, justificando breve y necesariamente la respuesta, si el resultado es correcto o incorrecto

**6.12.** Se dan dos números A y B escritos en hexadecimal, ambos números son fraccionarios y positivos. Sabiendo que uno de ellos está representado en el estándar IEEE-754 de coma flotante y el otro en coma fija sin signo, se pide colocar la coma en donde se precise para que se cumpla que A = B.

**A:** $81CA8000_{16}$ **B:** $4501CA80_{16}$