

Estructura de Computadores

Tema 4

Aritmética en coma flotante

DISCA

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ Juego de instrucciones de coma flotante
- Operadores de coma flotante
 - ✓ Operador de cambio de signo
 - ✓ Operadores de conversión de tipo
 - ✓ Operador de multiplicación

Bibliografía

D.L. Patterson, J. L. Hennessy: Estructura y diseño de computadores

- Ed. Reverté, 2000: volumen I, capítulo 4
- Ed. Reverté, 2011, traducción de la 4ª edición en inglés: cap. 3

W. Stallings: Organización y Arquitectura de Computadores (7a ed.)

Prentice Hall, capítulo 9

David Goldberg: Computer Arithmetic

- Apéndice H de J. L. Hennessy, D. L. Patterson: Computer Architecture, a Quantitative Approach, 3a edición
 - Disponible en castellano en la 1ª edición en McGraw-Hill
- ✓ David Goldberg: What every computer scientist should know about floating-point arithmetic
- PDF (accesible en muchos sitios de la web)

Índice

- **Introducción**

- ✓ Medida del rendimiento

- **La norma IEEE y su implementación en el MIPS**

Vídeo 1

- ✓ La norma IEEE 754

- ✓ Visión del programador: banco de registros y movimiento de datos

- ✓ Juego de instrucciones de coma flotante

- **Operadores de coma flotante**

- ✓ Operador de cambio de signo
- ✓ Operadores de conversión de tipo
- ✓ Operador de multiplicación

Vídeo 2



Introducción

- **La aritmética de coma flotante**
 - ✓ Sirve para cálculos definidos sobre reales (*float* y *double* en alto nivel)
 - ✓ Puede que no esté directamente soportada por la ALU:
 - no es esencial para el funcionamiento de la CPU
 - puede ser emulada mediante instrucciones de enteros
 - ✓ Evolución
 - Hasta 1985 (aprox) los operadores de CF iban dentro de un chip opcional que se colocaba al lado de la CPU
 - Desde 1985 en adelante, las CPU de propósito general incluyen operadores de CF dentro de su UAL
 - Desde 1990, los adaptadores de vídeo incluyen una GPU (*Graphics Processing Unit*) con un número creciente de operadores de CF

Introducción

- **La medida de prestaciones en coma flotante**

- ✓ El número de operaciones de coma flotante por segundo (FLOPS, prefijos $M=10^6$, $G=10^9$, $T=10^{12}$) es una medida de prestaciones utilizada en dos contextos:
 - ✓ En el diseño de operadores de CF el número máximo de operaciones por segundo viene determinado por el tiempo de operación de cada circuito.
 - ✓ En las comparativas entre computadores o entre aceleradores gráficos: se utiliza el número de operaciones de CF que ejecuta el dispositivo por segundo.
 - **El número de operaciones:** depende del número y características de los operadores incluidos y del uso que se hace de ellos.
 - **Productividad punta** de un computador o de un acelerador gráfico: es suma de las productividades de todos los operadores de CF incluidos.
 - Suele ser imposible de alcanzar en el uso corriente.

Introducción

- **Productividades punta**



Procesador Intel Core2 Duo @2GHz
16 GFLOPS



Procesador Intel Core i7 965 XE
70 GFLOPS



GPU ATI Radeon HD4890
2.4 TFLOPS



K Computer (Japón, junio 2011)
10 PFLOPS

Índice

Video

<http://politube.upv.es/play.php?vid=63612>

- Introducción

- ✓ Medida del rendimiento

- **La norma IEEE y su implementación en el MIPS**

Vídeo 1

- ✓ La norma IEEE 754

- ✓ Visión del programador: banco de registros y movimiento de datos

- ✓ Juego de instrucciones de coma flotante

- Operadores de coma flotante

Siguiente 

- ✓ Operador de cambio de signo

- ✓ Operadores de conversión de tipo

- ✓ Operador de multiplicación

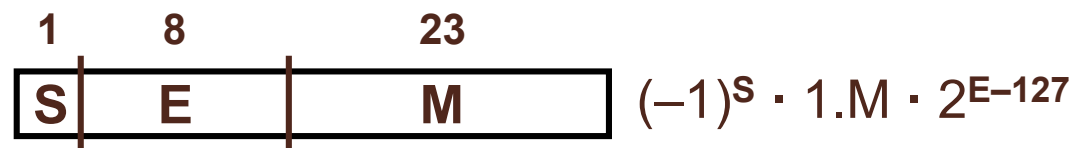
Notas sobre la representación de números

- Representación del conjunto \mathbb{R} (reales positivos y negativos)
 - ✓ El conjunto \mathbb{R} es un conjunto denso: entre dos números reales hay infinitos números reales
 - ✓ La representación del computador es limitada y no siempre es exacta
 - Con 32 bits se pueden obtener 2^{32} palabras diferentes. Por tanto, como máximo se pueden representar 2^{32} valores del conjunto \mathbb{R}
 - ✓ Hay números reales que tienen representación exacta otros que no, como los números con parte decimal periódica



La norma IEEE 754

- Representación del conjunto R (reales positivos y negativos)
 - ✓ ¿Cómo codificar un número real en una palabra de bits?
 - Aplicando un formato arbitrario, como el IEEE 754, estructurado en tres campos de bits para el signo, el exponente y la parte significativa (mantisa)



- ✓ Campo S: signo
 - $S = 0$ Valor positivo
 - $S = 1$ Valor negativo
- ✓ Campo M: mantisa (normalizada 1,M)
- ✓ Campo E : Exponente en exceso a 127

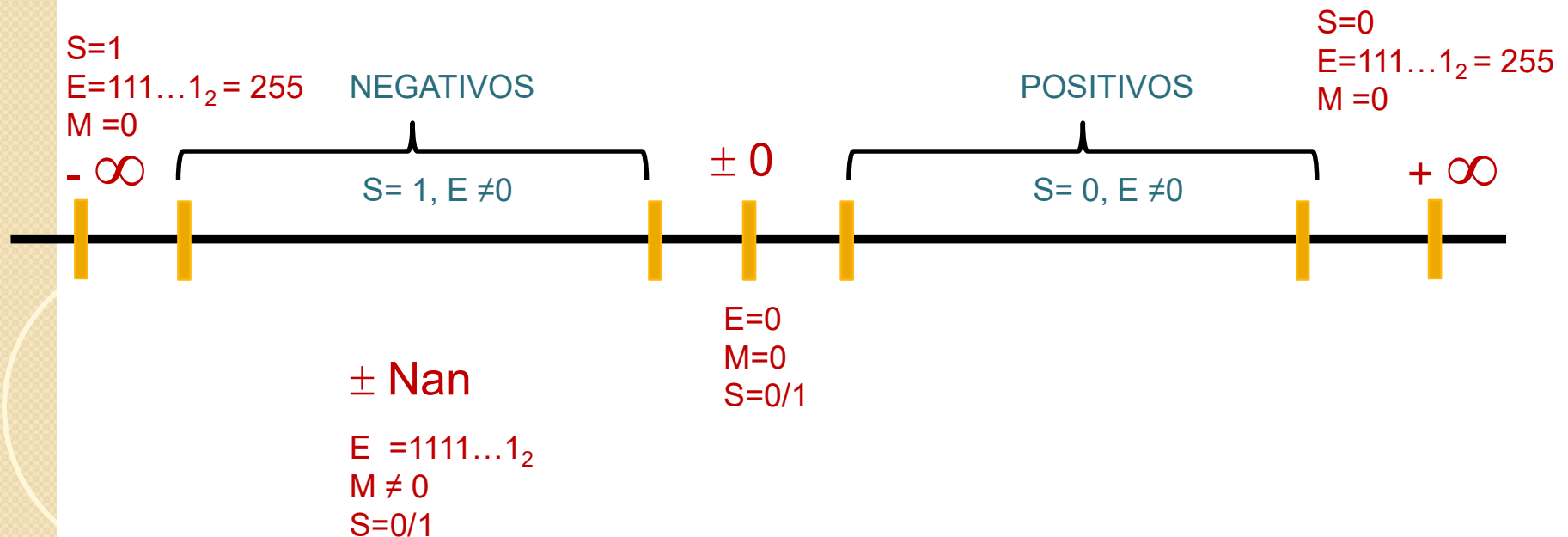
La norma IEEE 754

$$(-1)^S \cdot 1.M \cdot 2^{E-127}$$

- Representación del conjunto R (reales positivos y negativos)
 - ✓ El formato impone más restricciones a la representación:
 - habrá algunas palabras de bits con una significación matemática especial:
 - como el valor infinito, $E=255, M = 0$
 - el cero $E = 0, M = 0$
 - Nan (not a number) $E = 255, M \neq 0$

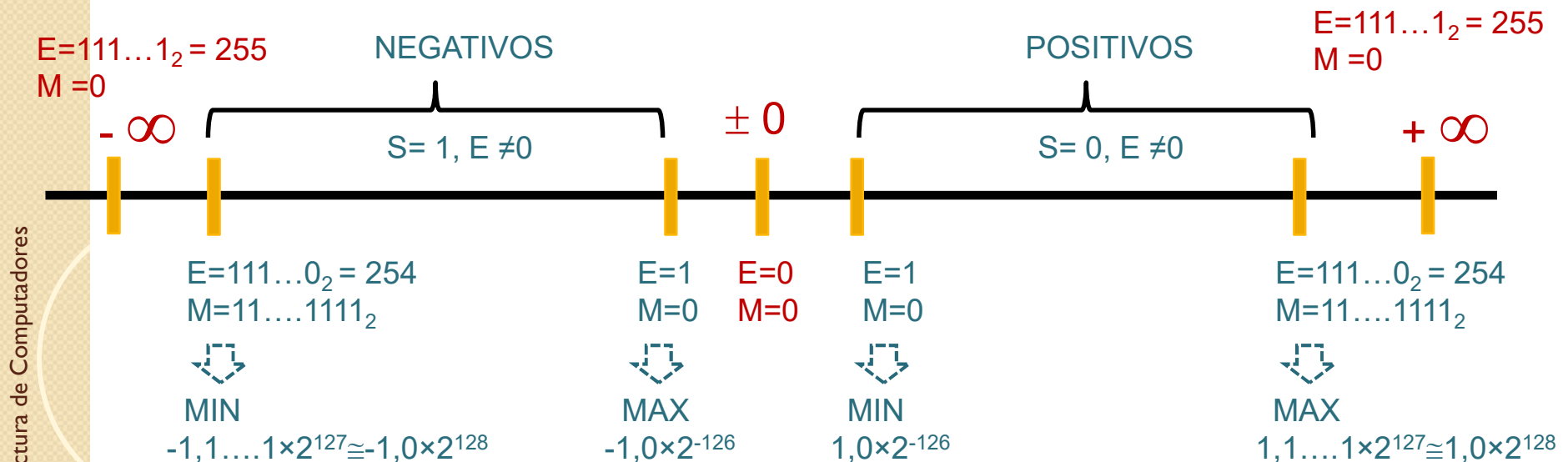
La norma IEEE 754

- Patrón de la representación de los números reales
 - ✓ Ningún valor real tendrá por tanto exponente igual a cero, ni exponente igual a 255 porque están reservados.
 - ✓ La diferencia entre la representación del valor infinito y Nan es la mantisa.



La norma IEEE 754

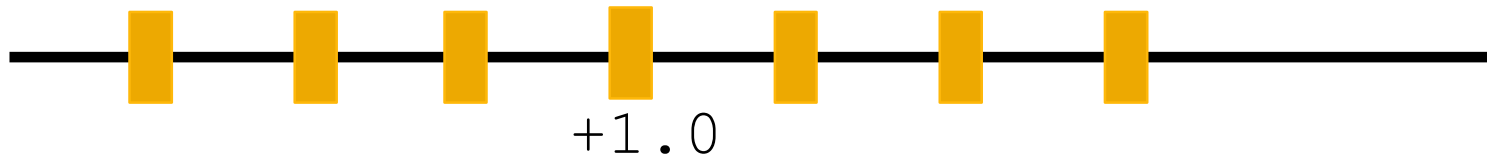
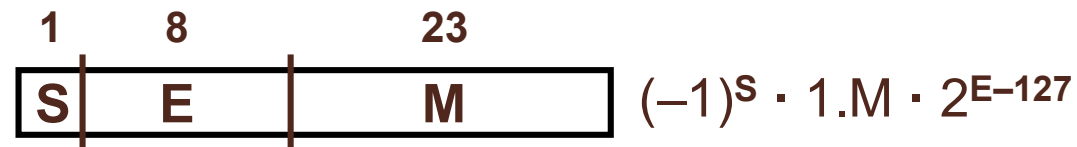
- Patrón de la representación de los números reales
 - ✓ Los rangos de representación de números positivos o negativos se calculan del mismo modo, solo les cambia el signo.
 - ✓ El rango será $] -1.0 \times 2^{128} \dots -1.0 \times 2^{-126}] \cup 0 \cup [1.0 \times 2^{-126} \dots 1.0 \times 2^{128} [$



La norma IEEE 754

- **Representación en Simple Precisión (32 bits)**

Símbolos: S es el signo, M la magnitud de la mantisa, E el exponente



- **Ejemplo: Codificar el valor real 1.0_{10}**

$$1.0_{10} = 1.0_2 \times 2^0 \quad S = 0$$

$$M = 0$$

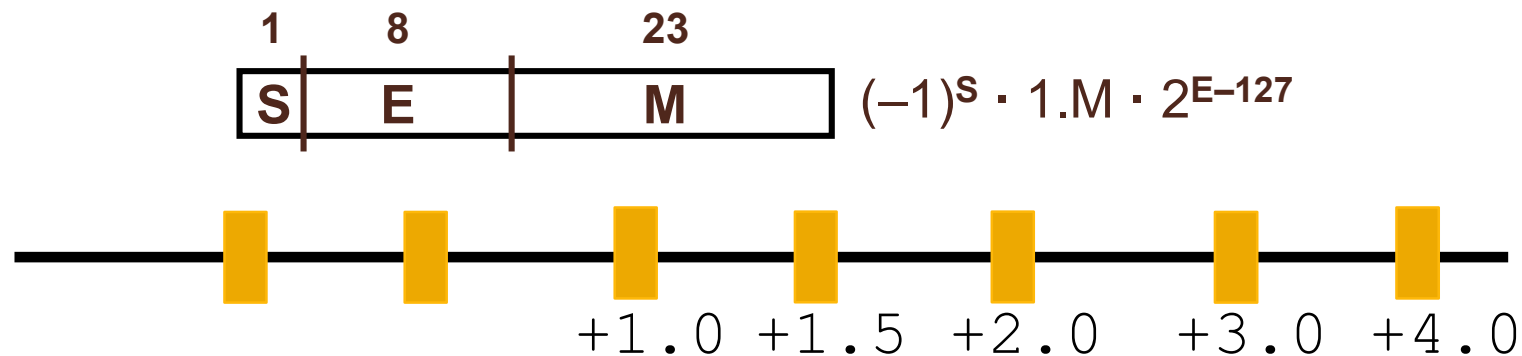
$$E = 0 + 127 = 01111111_2$$

$$0 \mid 01111111 \mid 0000000000000000 \dots 00 = 0x3F800000$$

La norma IEEE 754

- **Representación en Simple Precisión (32 bits)**

Símbolos: S es el signo, M la magnitud de la mantisa, E el exponente



- **Ejemplo: Codificar el valor real 1.5_{10}**

$$1.5_{10} = 1.1_2 \times 2^0 \quad S = 0$$

$$M = 1000 \dots 0$$

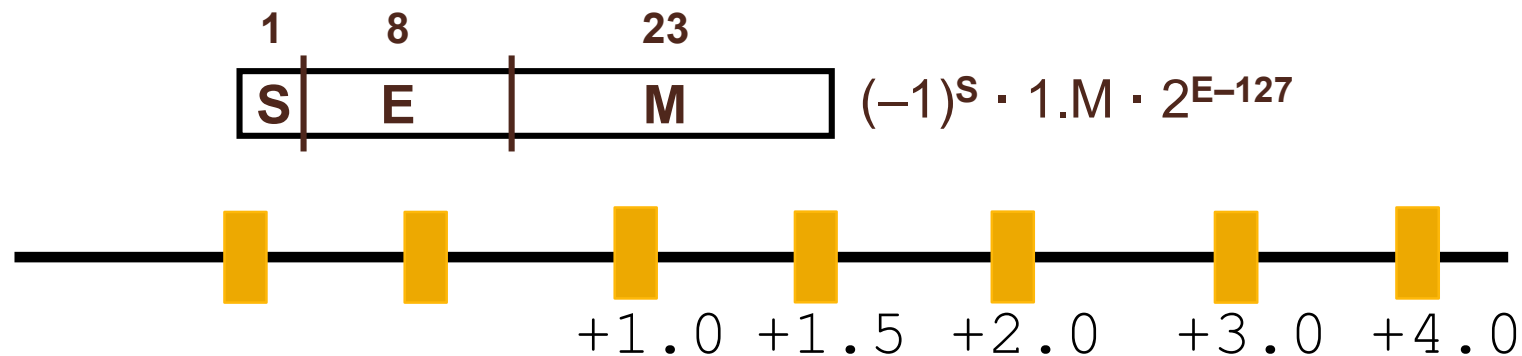
$$E = 0 + 127 = 01111111_2$$

$$\boxed{0 \mid 01111111 \mid 10000000000000 \dots 00} = 0x3FC00000$$

La norma IEEE 754

- **Representación en Simple Precisión (32 bits)**

Símbolos: S es el signo, M la magnitud de la mantisa, E el exponente



- **Ejemplo: Codificar el valor real 2.0_{10}**

$$2.0_{10} = 10.0_2 \times 2^0 = 1.0_2 \times 2^1 \quad S = 0$$

$$M = 0$$

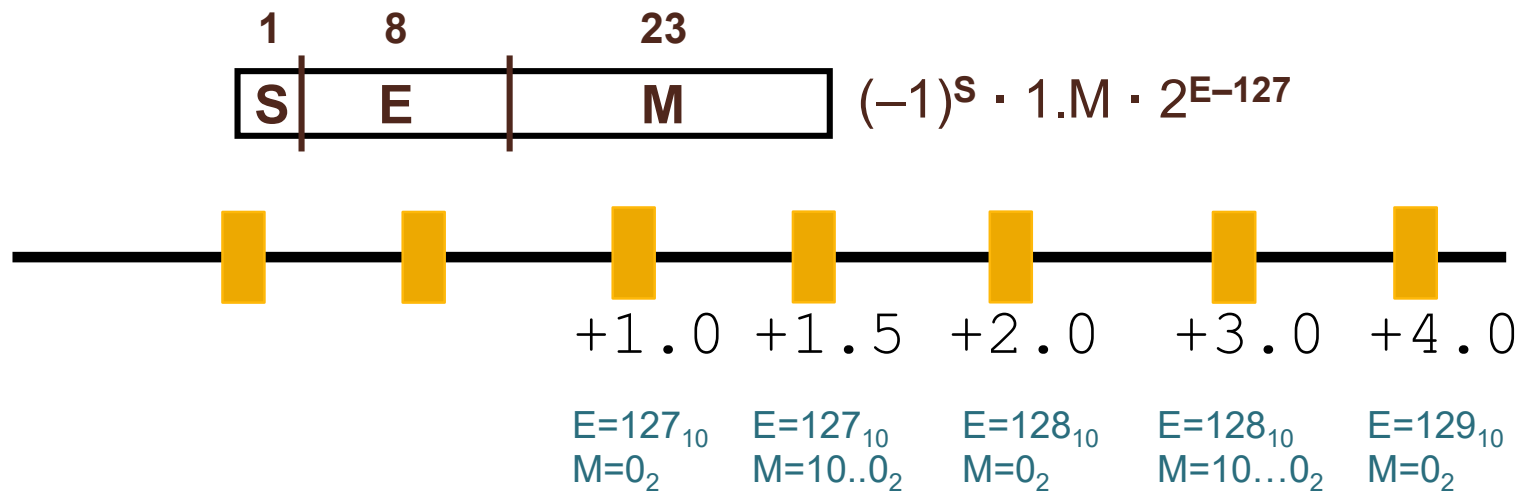
$$E = 1 + 127 = 10000000_2$$

$$\boxed{0 \mid 10000000 \mid 000000000000 \dots 00} = 0x40000000$$

La norma IEEE 754

- **Representación en Simple Precisión (32 bits)**

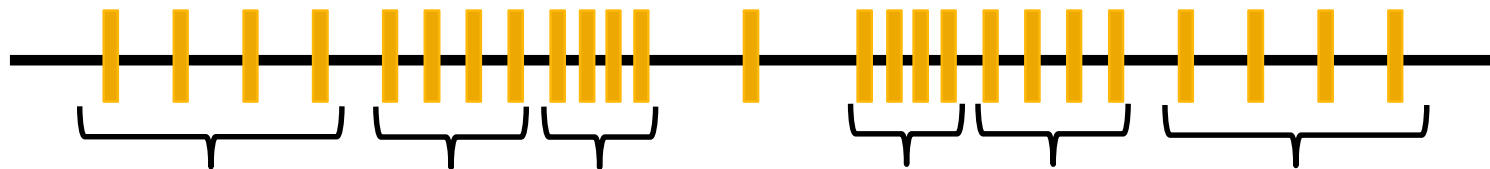
Símbolos: S es el signo, M la magnitud de la mantisa, E el exponente



La norma IEEE 754

• Conclusiones:

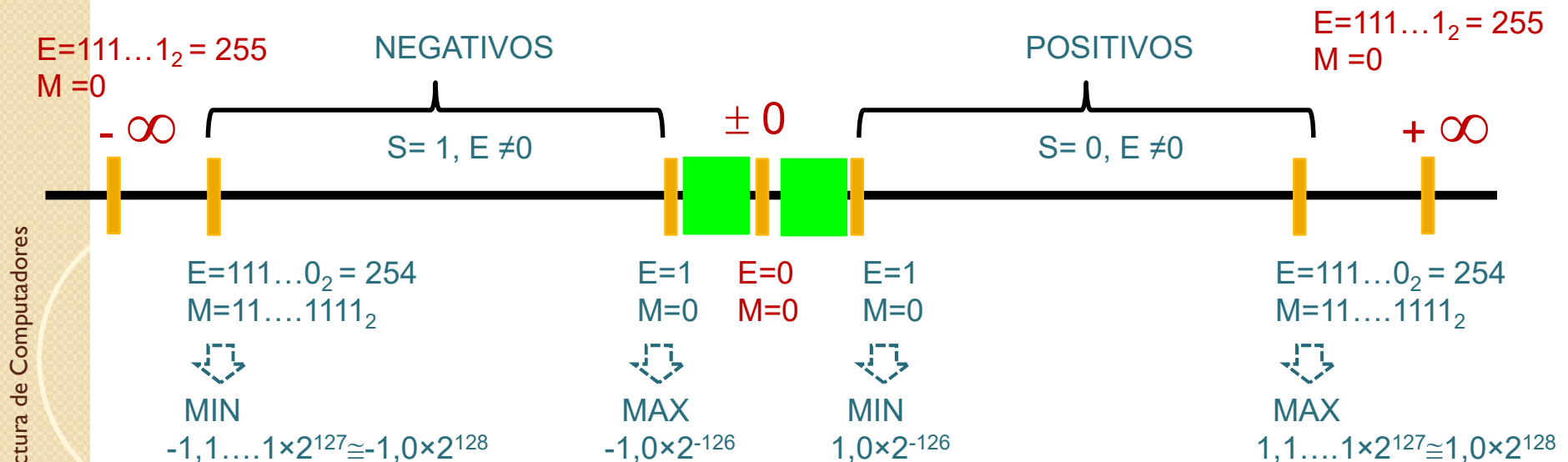
- ✓ Cuanto más grande es el exponente más distancia hay entre dos números representados consecutivos (la densidad de representación disminuye)
- ✓ Para un mismo valor de exponente los números representados están separados por la misma distancia
- ✓ Además existe baja densidad de representación cerca del cero.



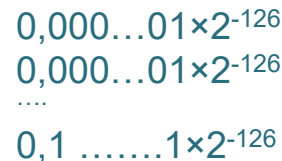
Cada grupo de valores tiene el mismo exponente y diferentes mantisas

La norma IEEE 754

- Los valores cercanos al cero
 - ✓ El formato IEEE 754 reserva un subconjunto de palabras de bits para representar números reales cerca del cero y que se interpretan de forma diferente del resto de valores (valores desnormalizados)
 - Exponente = 0, y mantisa $\neq 0$
 - ✓ No todas las unidades de coma flotante soportan este subconjunto de valores



Estructura de Computadores



La norma IEEE 754

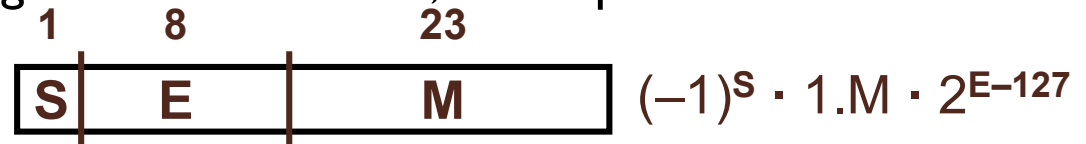
- Representación del conjunto \mathbb{R} (reales positivos y negativos)
 - ✓ En el computador interesa aumentar
 - La cantidad de números representados (densidad)
 - El rango de la representación
 - ✓ Estos dos aspectos dependen de los campos de la parte significativa (mantisa) y del exponente del formato
 - Exponente : influye sobre el rango
 - Mantisa: influye sobre la densidad
 - ✓ La norma contempla dos representaciones:
 - Simple precisión (float): 32 bits
 - Doble precisión (double): 64 bits

La norma IEEE 754

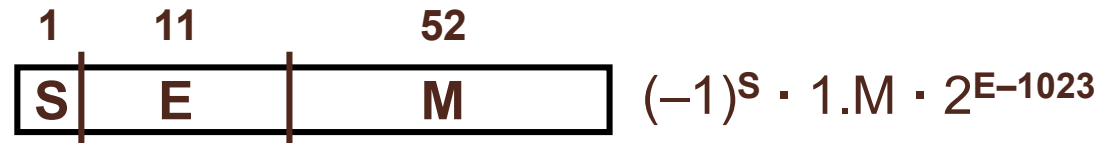
• Representación: **Todos los formatos**

Símbolos: S es el signo, M la magnitud de la mantisa, E el exponente

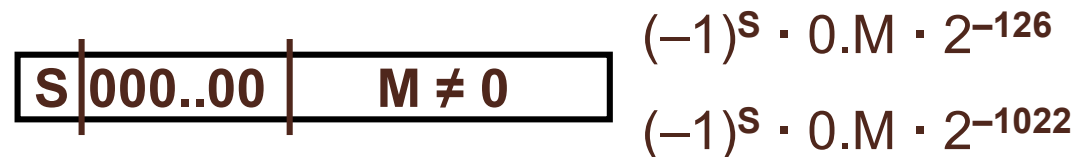
- Simple precisión (SP)



- Doble precisión (DP)



- ✓ Valores subnormales



- (SP y DP)

- ✓ Valores especiales (SP y DP)



La norma IEEE 754

- **Alcance de la norma**

- ✓ La norma IEEE 754 (y su ampliación a la aritmética de punto fijo, IEEE 854) especifican:
 - codificación: cómo representar los números en diversos formatos (precisiones simple, doble y extendida, SP DP EP) y el tratamiento de casos particulares: NaN (Not a Number), $\pm\infty$ (infinity), 0 (zero)
 - unos modos de funcionamiento (p. ej, el método de redondeo aplicable durante los cálculos)
 - un conjunto de operaciones que se pueden implementar en el hardware o en forma de bibliotecas
 - el soporte que ha de dar el sistema de excepciones de los procesadores (para que se puedan diseñar buenas bibliotecas de cálculo numérico)

La norma IEEE 754

- **Los valores especiales**

- ✓ Son manipulados por las operaciones junto con los reales corrientes

- ✓ Cero e infinito:

- se entienden como límites matemáticos; por eso

- $+\infty + +\infty = +\infty; -\infty + -\infty = -\infty; \text{etc.}$

- $+\infty \times \text{positivo} = +\infty; +\infty \times \text{negativo} = -\infty; \text{etc.}$

- $\text{positivo} / +0 = +\infty; \text{positivo} / -0 = -\infty; \text{etc.}$

- atención a las comparaciones: $+0$ y -0 son iguales

- ✓ *Not a Number*:

La norma IEEE 754

- **Los valores especiales**

- ✓ Son manipulados por las operaciones junto con los reales corrientes
- ✓ Cero e infinito
- ✓ *Not a Number*:
 - propagación: cualquier operación donde un operando es NaN dará como resultado NaN
 - generación: NaN es el resultado de $(+\infty) + (-\infty)$, $\pm 0 \times \pm \infty$, $\pm 0 / \pm 0$, $\pm \infty / \pm \infty$ y otras
 - una comparación ($=$, $<$, \geq , etc) entre NaN y otro número resulta siempre falsa

La norma IEEE 754

- La norma y los lenguajes de programación
 - ✓ Los valores especiales permiten tratar los incidentes del cálculo
 - ✓ El desbordamiento aritmético produce un resultado representable

```
float x = +0.0f;
float y = 1/x;
float z = Float.NEGATIVE_INFINITY;
float t = 1/z;
float u = x*z;
System.out.println("x = " + x);
System.out.println("1/x = " + y);
System.out.println("z = " + z);
System.out.println("1/z = " + 1/z);
System.out.println("x * z = " + u);
```

Java

```
x = 0.0
1/x = Infinity
z = -Infinity
1/z = -0.0
x * z = NaN
```


La norma IEEE 754

- **El redondeo**

- ✓ Situación frecuente: una operación genera una mantisa M de longitud más larga (p bits) que la prevista en el formato (m bits)
 - los m primeros bits de la mantisa M se llaman *retenidos*
- ✓ Posibilidades:
 - M es representable de forma *exacta* en el formato: los $p-m$ bits no retenidos son 0 y se pueden eliminar: **010000** \rightarrow **0100**
 - M se encuentra entre dos valores representables M_- y M_+ ($M_- < M < M_+$) y hay que *redondear*: escoger uno de ellos como representación *inexacta* de M
- ✓ La norma admite cuatro *modos de redondeo*:

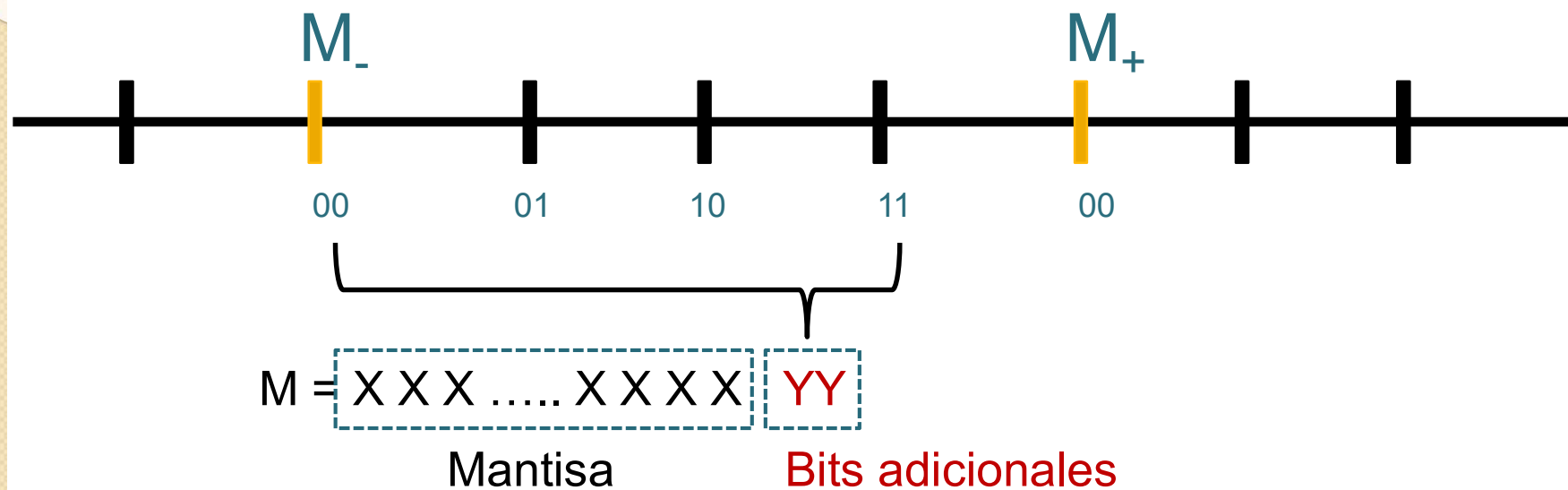
La norma IEEE 754

- **El redondeo**

- ✓ Situación frecuente: una operación genera una mantisa M de longitud más larga (p bits) que la prevista en el formato (m bits)
 - los m primeros bits de la mantisa M se llaman *retenidos*
- ✓ Posibilidades
- ✓ La norma admite cuatro *modos de redondeo*:
 - Hacia $+\infty$
 - Hacia $-\infty$
 - Hacia 0
 - Escoger el más próximo de los dos (este es el modo por omisión)

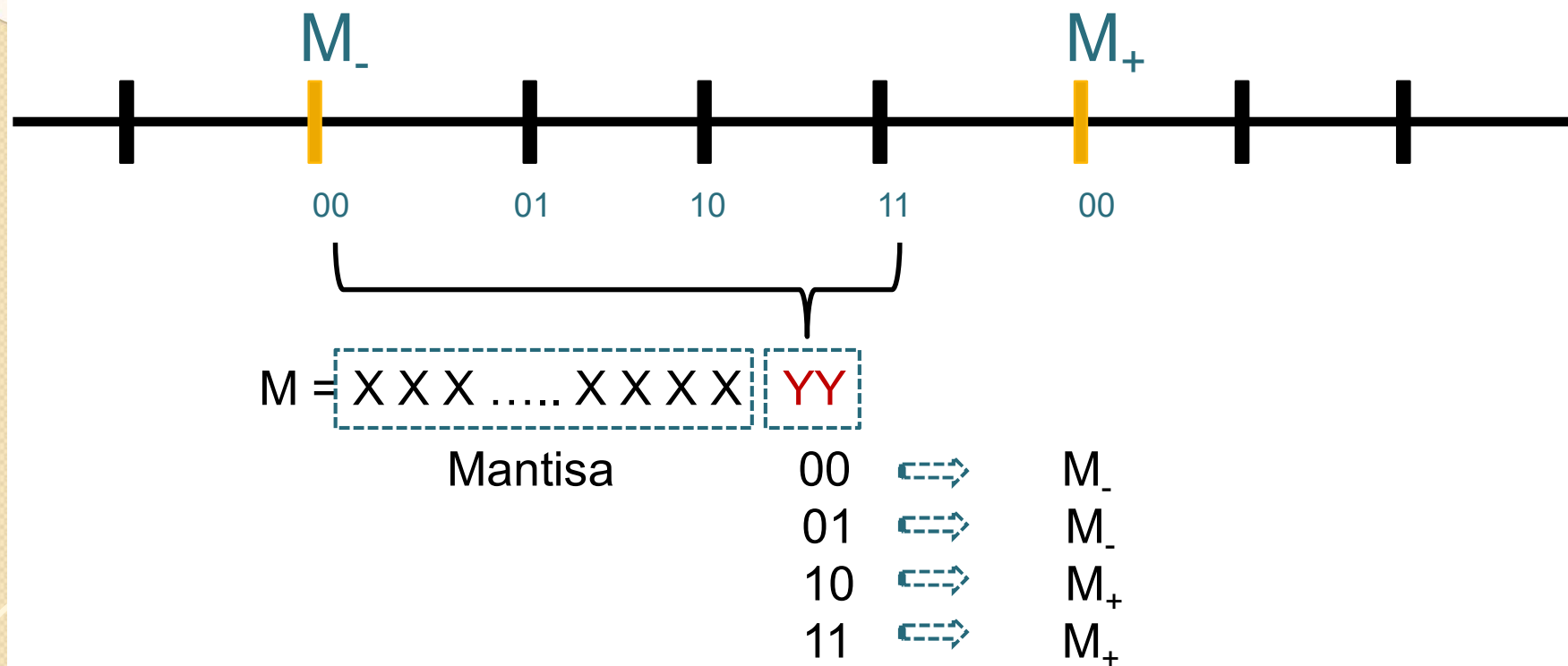
La norma IEEE 754

- **El redondeo**



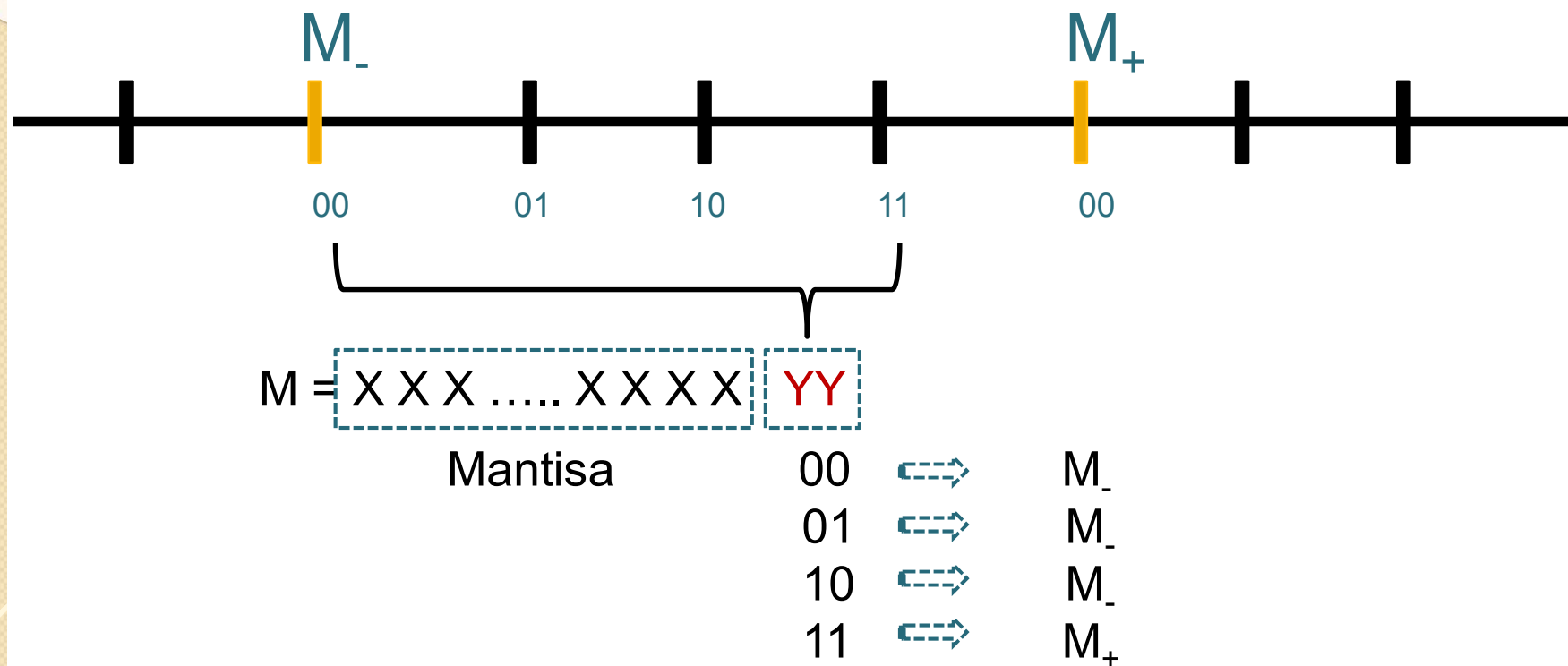
La norma IEEE 754

- El redondeo: hacia $+\infty$



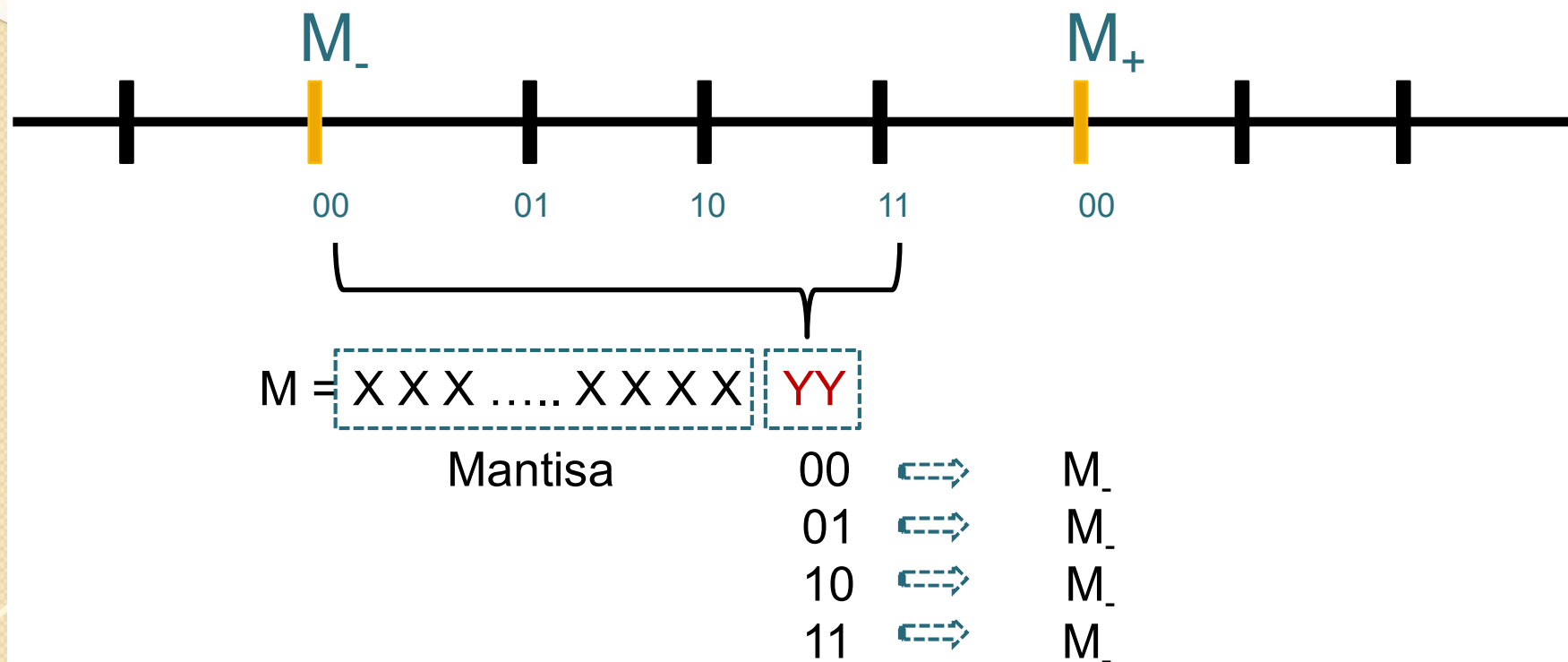
La norma IEEE 754

- **El redondeo: hacia $-\infty$**



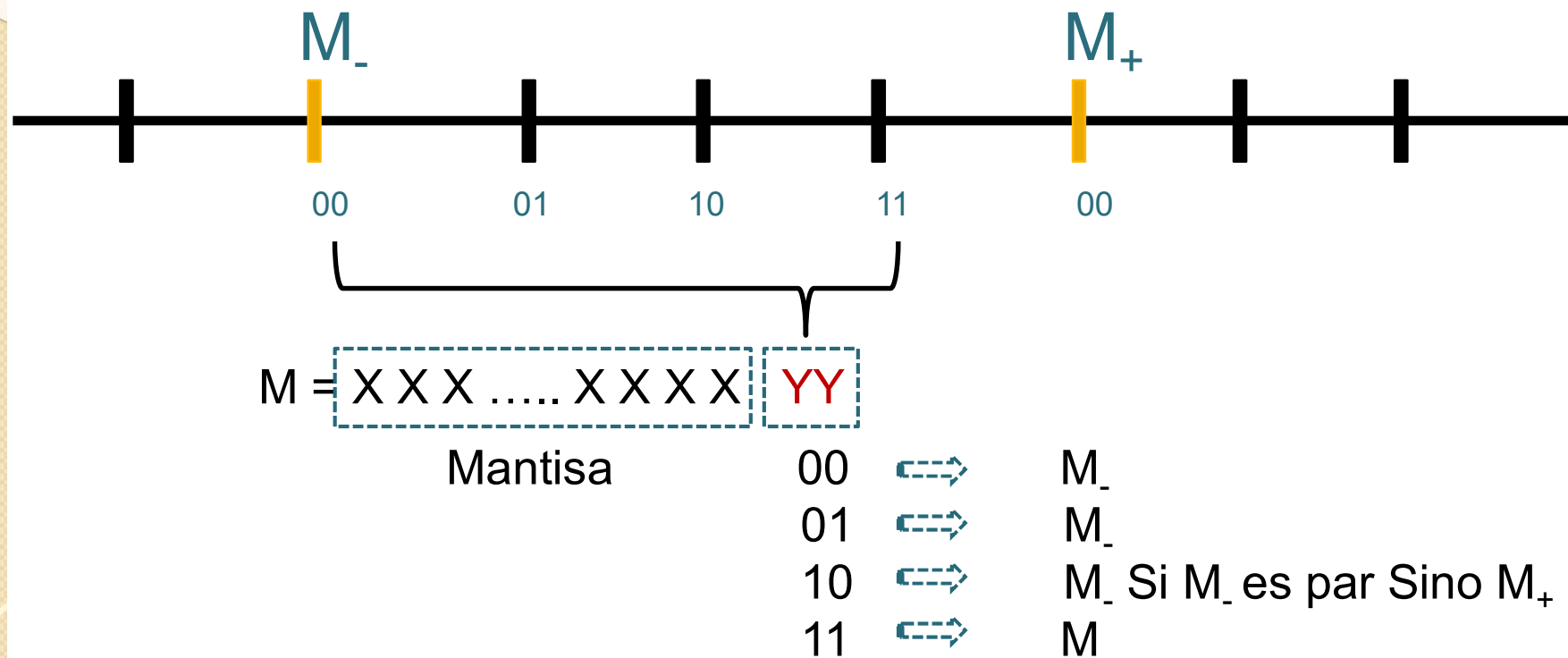
La norma IEEE 754

- El redondeo: hacia 0



La norma IEEE 754

- El redondeo: hacia el par más próximo

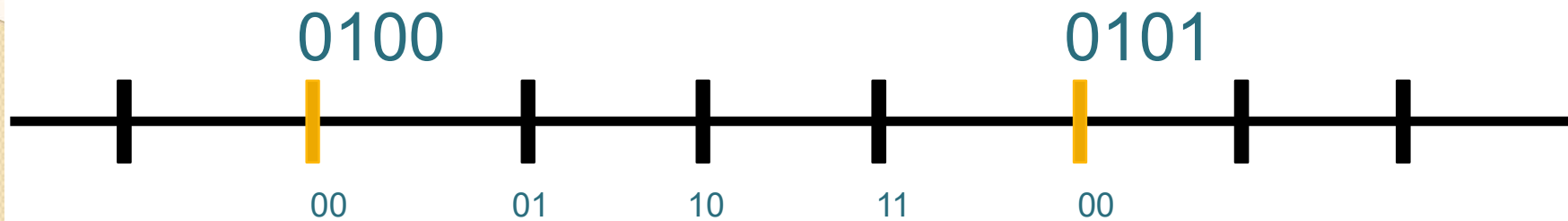


- ✓ La variante por omisión es “tie to even”: en caso de que M equidiste de M₋ y M₊ hay que escoger la mantisa representable par (o sea, la que acabe en 0)

La norma IEEE 754

- El redondeo hacia el más próximo (sesgado al par)

✓ Ejemplo:



M	se elige	M resultante
010000	(exacta)	0100
010001	M ₋ (más próxima)	0100
010010	M ₋ (par)	0100
010011	M ₊ (más próxima)	0101
010100	(exacta)	0101
010101	M ₋ (más próxima)	0101
010110	M ₊ (par)	0110
010111	M ₊ (más próxima)	0110
011000	(exacta)	0110

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ Juego de instrucciones de coma flotante
- Operadores de coma flotante
 - ✓ Operador de cambio de signo
 - ✓ Operadores de conversión de tipo
 - ✓ Operador de multiplicación

La coma flotante en el MIPS

- **Visión del programador**

Enteros: Procesador

Bytes / Half / Words

Registros: \$0, \$1,

\$0 = CERO

Reales: Coprocesador 1

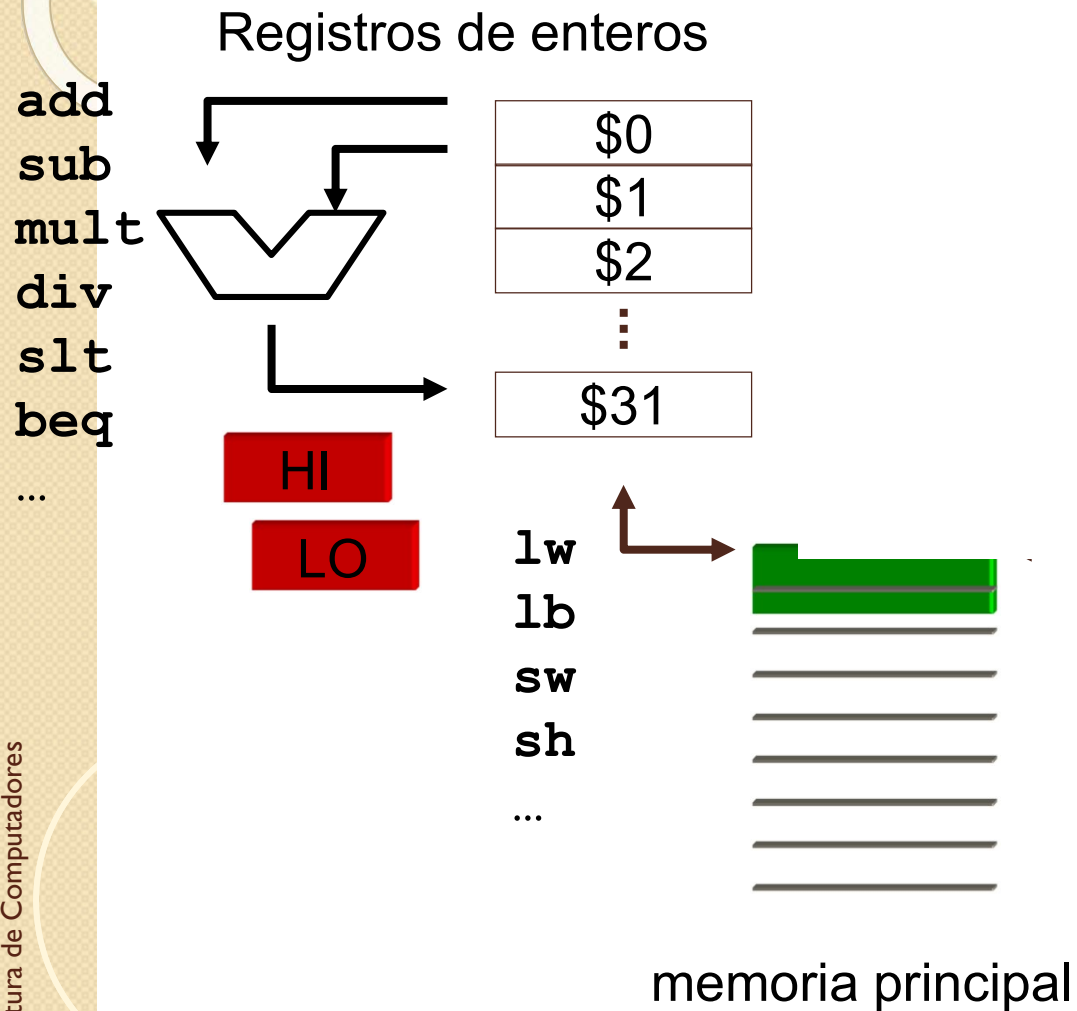
Word: simple precisión IEEE 754
Doble Word: doble precisión IEEE 754

Registros: \$f0, \$f1,

\$f0 ≠ CERO

La coma flotante en el MIPS

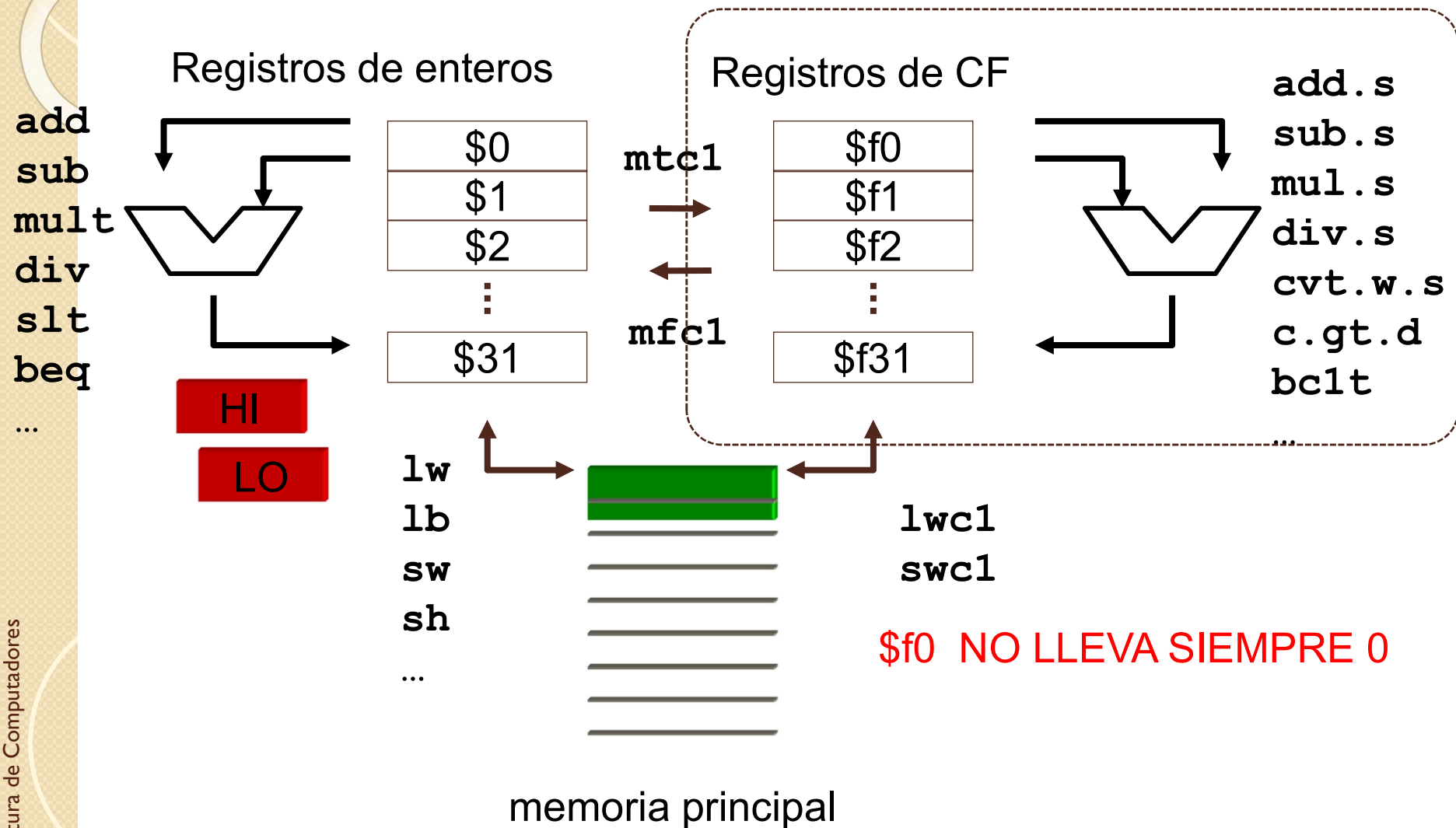
- **Visión del programador**



La coma flotante en el MIPS

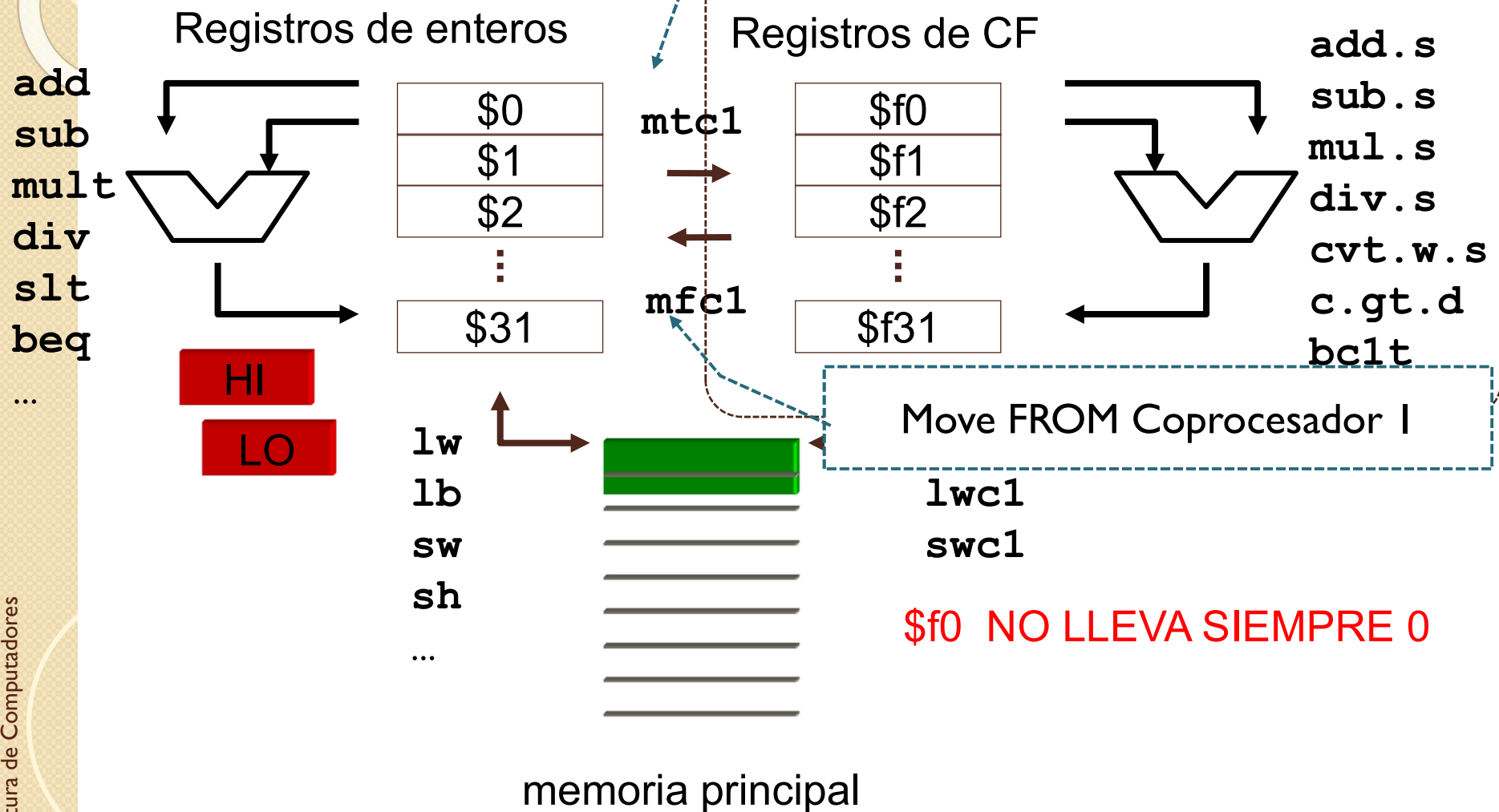
- Visión del programador

Coprocesador 1



La coma flotante en el MIPS

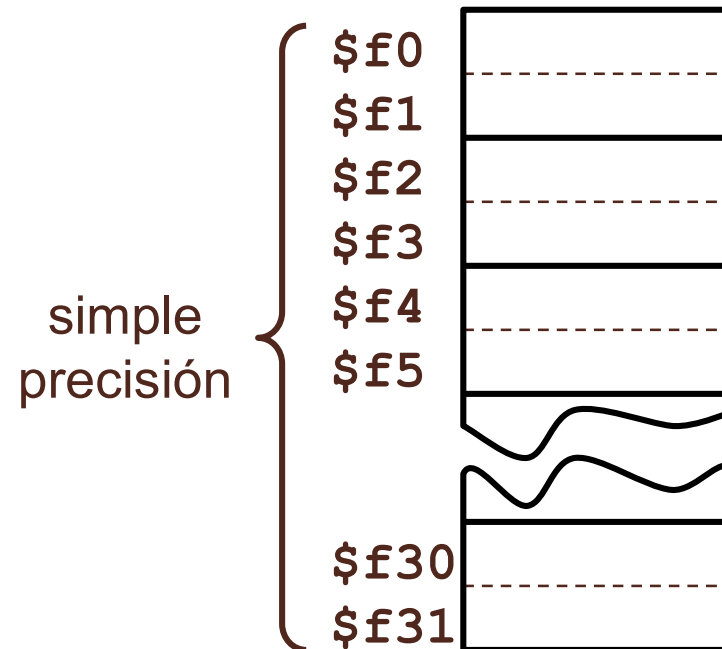
- Visión del programador



La coma flotante en el MIPS

• El banco de registros

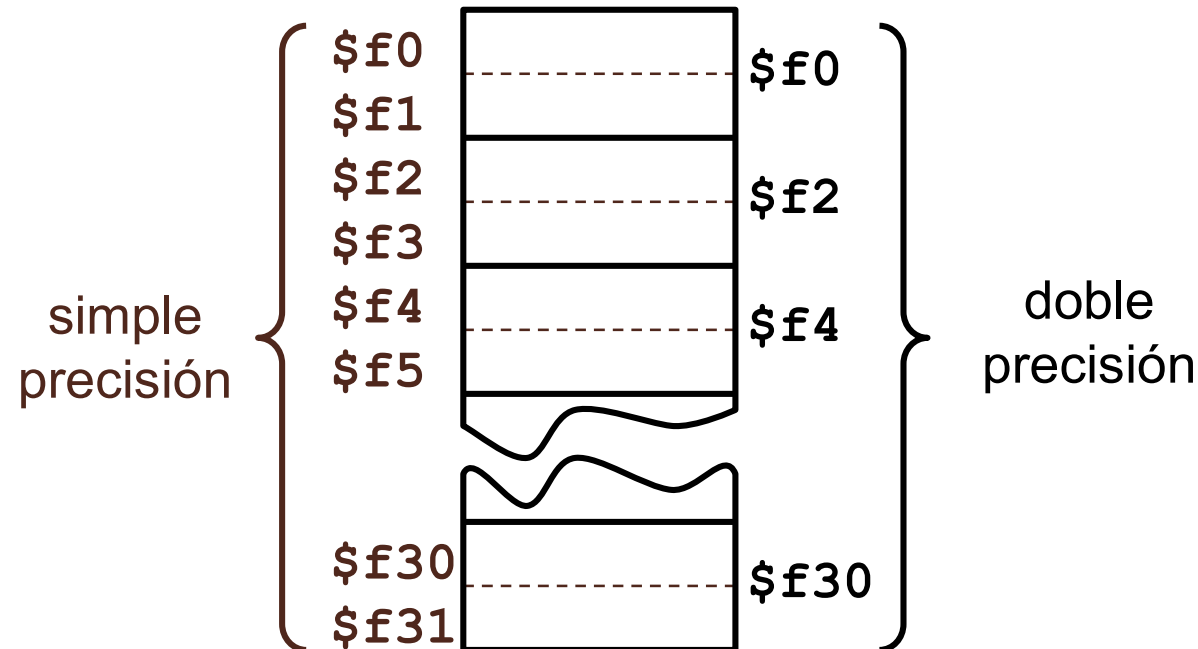
- ✓ Hay 32 registros de 32 bits, $\$f0$, $\$f1$, \dots , $\$f31$ para tipo float
 - Se suelen utilizar los números pares $\$f0$, $\$f2$, \dots , $\$f30$



La coma flotante en el MIPS

• El banco de registros

- ✓ Hay 32 registros de 32 bits, $\$f0, \$f1, \dots, \$f31$ para tipo float
 - Se suelen utilizar los números pares $\$f0, \$f2, \dots, \$f30$
- ✓ **Emparejables** para formar 16 registros de 64 bits para tipo **double**
 - Si $\$f0$ “contiene” un double: $\$f0$ tiene la parte baja y $\$f1$ la parte alta ($\$f1 || \$f0$)



La coma flotante en el MIPS R2000

- Convenio de uso de los registros

Nombre del registro	Utilización
\$f0	Retorno de función (parte real)
\$f2	Retorno de función (parte imaginaria)
\$f4, \$f6, \$f8, \$f10	Registros temporales
\$f12, \$f14	Paso de parámetros a funciones
\$f16, \$f18	Registros temporales
\$f20, \$f22, \$f24, \$f26, \$f28, \$f30	Registros a preservar entre llamadas

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
- Operadores de coma flotante
 - ✓ Operador de cambio de signo
 - ✓ Operadores de conversión de tipo
 - ✓ Operador de multiplicación

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ Juego de instrucciones de coma flotante
 - Intercambio con la memoria y registros
 - Pseudoinstrucciones de carga
 - Intercambios entre registros
 - Conversión de formatos
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

lectura $\$ft \leftarrow Mem[X+\$rs]$

escritura $Mem[X+\$rs] \leftarrow \ft

transferencia $\$fs \leftarrow \rt

transferencia $\$rt \leftarrow \fs

instrucción

lwc1 $\$ft, X(\$rs)$

swc1 $\$ft, X(\$rs)$

mtc1 $\$rt, \fs

mfc1 $\$rt, \fs

fs y **ft**: registros de coma flotante
rs y **rt**: son registros de enteros

mtc1 = MOVE TO coprocesador 1 = escribo en c1

mfc1 = MOVE FROM coprocesador 1 = leo de c1

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

instrucción

lectura $\$ft \leftarrow Mem[X+\$rs]$

lwc1 $\$ft, X(\$rs)$

escritura $Mem[X+\$rs] \leftarrow \ft

swc1 $\$ft, X(\$rs)$

transferencia $\$fs \leftarrow \rt

mtc1 $\$rt, \fs

transferencia $\$rt \leftarrow \fs

mfc1 $\$rt, \fs

fs y **ft**: registros de coma flotante
rs y **rt**: son registros de enteros

```
x:      .data
        .float 3.14
y:      .double 0.1
        .text
la $t0,x          # f0 <- x
lwc1 $f0,0($t0)
```

Leer un FLOAT

Otro modo: **lwc1** $\$f0, x$

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

instrucción

lectura $\$ft \leftarrow Mem[X+\$rs]$

lwc1 $\$ft, X(\$rs)$

escritura $Mem[X+\$rs] \leftarrow \ft

swc1 $\$ft, X(\$rs)$

transferencia $\$fs \leftarrow \rt

mtc1 $\$rt, \fs

transferencia $\$rt \leftarrow \fs

mfc1 $\$rt, \fs

fs y **ft**: registros de coma flotante
rs y **rt**: son registros de enteros

```
.data
x:    .float 3.14
y:    .double 0.1
.text
la $t0,x          # f0 <- x
lwc1 $f0,0($t0)
la $t0,y          # f2 <- y
lwc1 $f2,0($t0)
lwc1 $f3,4($t0)
```

Leer un DOUBLE

Otro modo: **lwc1** $\$f2, y$
lwc1 $\$f3, y+4$

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

instrucción

lectura $\$ft \leftarrow Mem[X+\$rs]$

lwc1 $\$ft, X(\$rs)$

escritura $Mem[X+\$rs] \leftarrow \ft

swc1 $\$ft, X(\$rs)$

transferencia $\$fs \leftarrow \rt

mtc1 $\$rt, \fs

transferencia $\$rt \leftarrow \fs

mfc1 $\$rt, \fs

fs y **ft**: registros de coma flotante
rs y **rt**: son registros de enteros

```
.data
x:    .float 3.14
y:    .double 0.1
.text
la $t0,x          # f0 <- x
lwc1 $f0,0($t0)
la $t0,y          # f2 <- y
lwc1 $f2,0($t0)
lwc1 $f3,4($t0)
mtc1 $0,$f4       # f4 <- 0.0
```

Inicializar un registro del coprocesador:

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

instrucción

lectura $\$ft \leftarrow Mem[X+\$rs]$

lwc1 $\$ft, X(\$rs)$

escritura $Mem[X+\$rs] \leftarrow \ft

swc1 $\$ft, X(\$rs)$

transferencia $\$fs \leftarrow \rt

mtc1 $\$rt, \fs

transferencia $\$rt \leftarrow \fs

mfc1 $\$rt, \fs

fs y ft: registros de
coma flotante
rs y rt: son registros
de enteros

```
x:      .data
y:      .float 3.14
        .double 0.1
        .text
la $t0,x          # f0 <- x
lwc1 $f0,0($t0)
la $t0,y          # f2 <- y
lwc1 $f2,0($t0)
lwc1 $f3,4($t0)
mtc1 $0,$f4       # f4 <- 0.0
```

Las instrucciones de CF no admiten operandos inmediatos. Hay que ubicar las constantes en la memoria o construirlas en los registros de enteros

La coma flotante en el MIPS

- Intercambio con la memoria y los registros de enteros

operación

instrucción

lectura $\$ft \leftarrow Mem[X+\$rs]$

lwc1 $\$ft, X(\$rs)$

escritura $Mem[X+\$rs] \leftarrow \ft

swc1 $\$ft, X(\$rs)$

transferencia $\$fs \leftarrow \rt

mtc1 $\$rt, \fs

transferencia $\$rt \leftarrow \fs

mfc1 $\$rt, \fs

fs y ft: registros de
coma flotante
rs y rt: son registros
de enteros

```
.data
x:    .float 3.14
y:    .double 0.1
.text
la $t0,x          # f0 <- x
lwc1 $f0,0($t0)
la $t0,y          # f2 <- y
lwc1 $f2,0($t0)
lwc1 $f3,4($t0)
mtc1 $0,$f4       # f4 <- 0.0
```

Por eso se han introducido
las pseudoinstrucciones
que inicializan los registros
del coprocesador.

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
 - Intercambio con la memoria y registros
 - **Pseudoinstrucciones de carga**
 - Intercambios entre registros
 - Conversión de formatos
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación


La coma flotante en el MIPS

Pseudoinstrucciones de carga de constantes

Operación	pseudoinstrucción	
$\$ft \leftarrow cte$	<code>li.s \$ft,cte</code>	ft: registro en coma flotante par por convenio para float por necesidad para double
$\$ft+1 \$ft \leftarrow cte$	<code>li.d \$ft,cte</code>	

El ensamblador codifica cte en formato IEEE 754

<code>x:</code>	<code>.float 3.14</code>		
<code>y:</code>	<code>.double 0.1</code>		
	<code>.text</code>		<code>.text</code>
	<code>la \$t0,x</code>	<code># f0 <- x</code>	
	<code>lwc1 \$f0,0(\$t0)</code>		<code>li.s \$f0, 3.14</code>
	<code>la \$t0,y</code>	<code># f2 <- y</code>	
	<code>lwc1 \$f2,0(\$t0)</code>		<code>li.d \$f2, 0.1</code>
	<code>lwc1 \$f3,4(\$t0)</code>		
	<code>mtc1 \$0,\$f4</code>	<code># f4 <- 0.0</code>	<code>li.s \$f4,0.0</code>



Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
 - Intercambio con la memoria y registros
 - Pseudoinstrucciones de carga
 - Intercambios entre registros
 - Conversión de formatos
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación

La coma flotante en el MIPS

- **Intercambio entre registros del coprocesador**

operación

instrucción

$\$fd \leftarrow \fs

`mov.s $fd,$fs`

$\$fd \leftarrow \fs

`mov.d $fd,$fs`

$\$fd+1 \leftarrow \$fs+1$

fs y ft: registros de
coma flotante

La coma flotante en el MIPS

- Intercambio entre registros del coprocesador

operación

instrucción

$\$fd \leftarrow \fs

`mov.s $fd,$fs`

$\$fd \leftarrow \fs

`mov.d $fd,$fs`

$\$fd+1 \leftarrow \$fs+1$

fs y ft: registros de coma flotante

```
x:    .float 3.14
y:    .double 0.1
```

Mover un DOUBLE

```
.text
la $t0,x          # $f0 <- x
lwc1 $f0,0($t0)
la $t0,y          # $f2 <- y
lwc1 $f2,0($t0)
lwc1 $f3,4($t0)
mov.d $f0,$f2     # $f1<-$f3 y $f0 <-$f2
```

La coma flotante en el MIPS

- Intercambio entre registros del coprocesador

operación

instrucción

$\$fd \leftarrow \fs

`mov.s $fd,$fs`

$\$fd \leftarrow \fs

`mov.d $fd,$fs`

$\$fd+1 \leftarrow \$fs+1$

fs y ft: registros de coma flotante

```
x:      .float 3.14
y:      .double 0.1
```

```
.text
```

```
la $t0,x          # f0 <- x
```

```
lwc1 $f0,0($t0)
```

```
la $t0,y          # f2 <- y
```

```
lwc1 $f2,0($t0)
```

```
lwc1 $f3,4($t0)
```

```
mov.d $f0,$f2
```

¿Cómo sería mover un FLOAT?

```
# $f1, $f0 <- $f3,$f2
```

```
# $f4 <- $f0
```


Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
 - Intercambio con la memoria y registros
 - Pseudoinstrucciones de carga
 - Intercambios entre registros
 - **Conversión de formatos**
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación

La coma flotante en el MIPS

- **Conversión de formatos**

- ✓ Los registros de CF pueden contener:

<i>símbolo</i>	<i>tipo</i>
S	números en coma flotante en SP
D	(por parejas) números en coma flotante en DP
W	números enteros de 32 bits

- ✓ La instrucción `cvt._._ fd, fs` hace las conversiones posibles entre los tres tipos

La coma flotante en el MIPS

- **Conversión de formatos**

- ✓ Los registros de CF pueden contener:

<i>símbolo</i>	<i>tipo</i>
S	números en coma flotante en SP
D	(por parejas) números en coma flotante en DP
W	números enteros de 32 bits

- ✓ La instrucción `cvt. _ . _ fd, fs` hace las conversiones posibles entre los tres tipos

Ejemplo: hace la conversión del entero contenido en \$f7
a CF en doble precisión contenido en \$f5||\$f4

`cvt.d.w $f4,$f7 # (double) $f5||$f4 = (entero) $f7`

La coma flotante en el MIPS

- **Conversión de formatos**

- ✓ Los registros de CF pueden contener:

<i>símbolo</i>	<i>tipo</i>
S	números en coma flotante en SP
D	(por parejas) números en coma flotante en DP
W	números enteros de 32 bits

- ✓ La instrucción `cvt. __. __ fd, fs` hace las conversiones posibles entre los tres tipos
 - Ejemplo: `cvt.d.w $f4, $f7` hace la conversión del entero contenido en `$f7` a CF en doble precisión contenido en `$f5||$f4`
- ✓ En combinación con las instrucciones de transferencia con el banco de registros de enteros, se puede hacer aritmética con variables de tipos diversos

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
 - Intercambio con la memoria y registros
 - Pseudoinstrucciones de carga
 - Intercambios entre registros
 - Conversión de formatos
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación

La coma flotante en el MIPS

• Instrucciones aritméticas básicas

- ✓ Hay dos versiones de cada operación: S (simple precisión) y D (doble precisión)
 - Ejemplo: `add.s $f0,$f1,$f2; add.d $f2,$f4,$f6`

operación	instrucción
suma	<code>add._ fd,fs,ft</code>
resta	<code>sub._ fd,fs,ft</code>
multiplicación	<code>mul._ fd,fs,ft</code>
división	<code>div._ fd,fs,ft</code>
cambio de signo	<code>neg._ fd,fs</code>
valor absoluto	<code>abs._ fd,fs</code>

La coma flotante en el MIPS

Chuleterio de instrucciones

Tabla de instrucciones Flotantes:

rs es un registro del banco de enteros, ft del banco de flotantes

Tipo = s (float), d (double)

	MEMORIA ↔ REGISTRO FLOTANTE	REG. ENTERO ↔ REG. FLOTANTE	REG.FLOTANTE ↔ REG. FLOTANTE						
CARGA	$lwc1\ \$ft, X(\$rs) \quad \# \quad \$ft \leftarrow Mem(\$rs+X)$ $swc1\ \$ft, X(\$rs) \quad \# \quad \$ft \rightarrow Mem(\$rs+X)$	$mtc1\ \$rs, \$ft \quad \# \quad \$ft \leftarrow \rs $mfc1\ \$rs, \$ft \quad \# \quad \$ft \rightarrow \rs	$mov.tipo\ \$fd, \$fs \quad \# \quad \$fd \leftarrow \fs						
CONVERSION	$cvt.dest.font\ \$fd, \$fs \quad \# \quad \$fd \leftarrow \fs Dest = s,d,w Font = s,d,w								
ARITMETICAS	$add.tipo\ \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft + \fs $sub.tipo\ \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft - \fs	$mul.tipo\ \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft * \fs $div.tipo\ \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft / \fs	$neg.tipo\ \$fd, \$fs \quad \# \quad \$fd = - \fs $abs.tipo\ \$fd, \$fs \quad \# \quad \$fd = \$fs $						
COMPARACION	$c.comp.tipo\ \$fd, \$fs \quad \# \quad \$fd\ comp\ \fs comp = <table><tr><td>$\\$fd > \\fs gt</td><td>$\\$fd = \\fs eq</td><td>$\\$fd \leq \\fs lt</td></tr><tr><td>$\\$fd < \\fs le</td><td>$\\$fd \neq \\fs neg</td><td>$\\$fd \geq \\fs ge</td></tr></table>			$\$fd > \fs gt	$\$fd = \fs eq	$\$fd \leq \fs lt	$\$fd < \fs le	$\$fd \neq \fs neg	$\$fd \geq \fs ge
$\$fd > \fs gt	$\$fd = \fs eq	$\$fd \leq \fs lt							
$\$fd < \fs le	$\$fd \neq \fs neg	$\$fd \geq \fs ge							
SALTO	$bc1t\ etiqueta \quad \# \text{ Salta si es TRUE}$ $bc1f\ etiqueta \quad \# \text{ Salta si es FALSE}$								

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ **Juego de instrucciones de coma flotante**
 - Intercambio con la memoria y registros
 - Pseudoinstrucciones de carga
 - Intercambios entre registros
 - Conversión de formatos
 - Instrucciones aritméticas básicas
 - Instrucciones de comparación

La coma flotante en el MIPS

• La comparación

- ✓ Las instrucciones de comparación escriben un bit implícito **FPc** que codifica cierto=1 y falso=0
- ✓ Este bit se encuentra en un registro de control del coprocesador y puede ser consultado por las instrucciones de salto
- ✓ Para cada tipo de datos, hay un conjunto de comparaciones codificables
- ✓ Las más importantes: (**c. __.s fd, fs** o **c. __.d fd, fs**)

fd>fs	fd=fs	fd<fs
gt	eq	lt
le	neq	ge
fd≤fs	fd≠fs	fd≥fs

La coma flotante en el MIPS

Control de flujo y aritmética de coma flotante

- ✓ Hay dos instrucciones de bifurcación asociadas al bit FPc

`bc1t eti` si (FPc == 1) bifurcar a *eti*

`bc1f eti` si (FPc == 0) bifurcar a *eti*

- ✓ Combinadas con las instrucciones de comparación, permiten bifurcar con condiciones aritméticas complejas
- ✓ Cada condición permite dos implementaciones
 - Ejemplo en simple precisión: si ($\$f0 > \$f2$) bifurcar a *eti*

```
; mirar si $f0>$f2
      c.gt.s $f0,$f2
; saltar si afirmativo
      bc1t eti
```

```
; mirar si $f0<=$f2
      c.le.s $f0,$f2
; saltar si negativo
      bc1f eti
```

La coma flotante en el MIPS

Chuletarío de instrucciones

Tabla de instrucciones Flotantes:

rs es un registro del banco de enteros, ft del banco de flotantes

Tipo = s (float), d (double)

	MEMORIA ↔ REGISTRO FLOTANTE	REG. ENTERO ↔ REG. FLOTANTE	REG.FLOTANTE ↔ REG. FLOTANTE												
CARGA	$\text{lwc1 } \$ft, X(\$rs) \quad \# \quad \$ft \leftarrow \text{Mem}(\$rs+X)$ $\text{swc1 } \$ft, X(\$rs) \quad \# \quad \$ft \rightarrow \text{Mem}(\$rs+X)$	$\text{mtc1 } \$rs, \$ft \quad \# \quad \$ft \leftarrow \rs $\text{mfc1 } \$rs, \$ft \quad \# \quad \$ft \rightarrow \rs	$\text{mov.tipo } \$fd, \$fs \quad \# \quad \$fd \leftarrow \fs												
CONVERSION	$\text{cvt.dest.font } \$fd, \$fs \quad \# \quad \$fd \leftarrow \fs Dest = s,d,w Font = s,d,w														
ARITMETICAS	$\text{add.tipo } \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft + \fs $\text{sub.tipo } \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft - \fs	$\text{mul.tipo } \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft * \fs $\text{div.tipo } \$fd, \$ft, \$fs \quad \# \quad \$fd = \$ft / \fs	$\text{neg.tipo } \$fd, \$fs \quad \# \quad \$fd = - \fs $\text{abs.tipo } \$fd, \$fs \quad \# \quad \$fd = \$fs $												
COMPARACION	$\text{c.comp.tipo } \$fd, \$fs \quad \# \quad \$fd \text{ comp } \fs comp =	<table><tr><td>$\\$fd > \\fs</td><td>$\\$fd = \\fs</td><td>$\\$fd \leq \\fs</td></tr><tr><td>gt</td><td>eq</td><td>lt</td></tr><tr><td>le</td><td>neg</td><td>ge</td></tr><tr><td>$\\$fd < \\fs</td><td>$\\$fd \neq \\fs</td><td>$\\$fd \geq \\fs</td></tr></table>		$\$fd > \fs	$\$fd = \fs	$\$fd \leq \fs	gt	eq	lt	le	neg	ge	$\$fd < \fs	$\$fd \neq \fs	$\$fd \geq \fs
$\$fd > \fs	$\$fd = \fs	$\$fd \leq \fs													
gt	eq	lt													
le	neg	ge													
$\$fd < \fs	$\$fd \neq \fs	$\$fd \geq \fs													
SALTO	$\text{bc1t etiqueta} \quad \# \text{ Salta si es TRUE}$ $\text{bc1f etiqueta} \quad \# \text{ Salta si es FALSE}$														

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ Juego de instrucciones de coma flotante

HACER Todos los ejercicios pendientes.

El boletín Tema 4 Problemas de aritmética real.pdf

Índice

- Introducción
 - ✓ Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - ✓ La norma IEEE 754
 - ✓ Visión del programador: banco de registros y movimiento de datos
 - ✓ Juego de instrucciones de coma flotante
- **Operadores de coma flotante**
 - ✓ Operador de cambio de signo
 - ✓ Operadores de conversión de tipo
 - ✓ Operador de multiplicación

Video 2

Operadores de coma flotante

Operadores

- ✓ Toman como entrada uno o dos operandos en un formato de CF dado
- ✓ Su resultado es un valor en CF codificado según la norma
 - excepto los operandos de comparación
- ✓ Su diseño es complejo porque, además de realizar la operación definida, se han de ocupar de ciertos detalles:
 - Han de suministrar el resultado correctamente **normalizado** según la precisión con la que trabajan
 - Han de gestionar los **valores especiales** definidos en la norma
 - Si procede, han de redondear el resultado según el **modo** programado
 - Han de señalar las **excepciones** previstas por la norma
- ✓ Estudiaremos la estructura básica de algunos operadores y veremos, en casos seleccionados, cómo resuelven los detalles

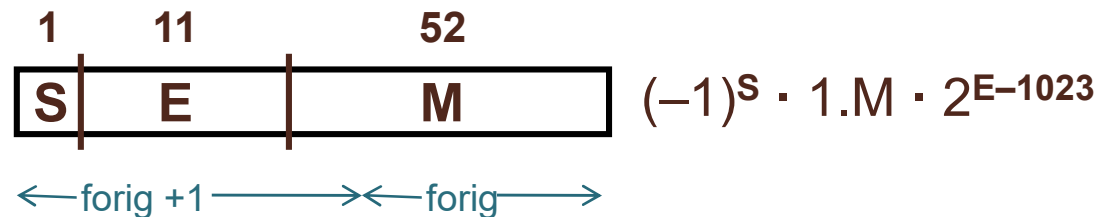
Operadores de coma flotante

■ Ejemplos de operadores

- NEG.S y NEG.D (cambio de signo)
 - Estructura
- CVT.D.S (conversión de simple a doble precisión)
 - Estructura básica
 - Detalle: tratamiento de los valores especiales
- CVT.S.D (conversión de doble a simple precisión)
 - Estructura básica
 - Detalle: el redondeo
- CVT.D.W (conversión de entero a CF doble precisión)
 - Estructura básica
 - Detalle: la normalización
- MULT.S y MULT.D (multiplicación)
 - Estructura básica
 - Detalle: la renormalización

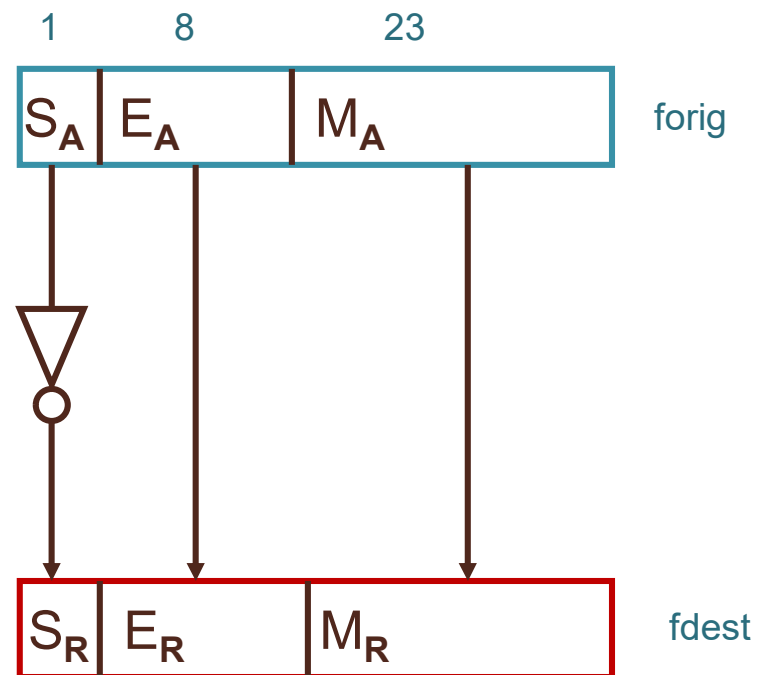
neg.s fdest, forig # fdest = -forig

- Descripción de la operación:
 - ✓ Cambia el signo del valor almacenado en forig
 - ✓ Hay dos versiones : neg.s y neg.d
 - ✓ En neg.s el valor es de simple precisión (float, 32bits)
 - ✓ En neg.d el valor está almacenado en dos registros, forig y forig+1, y es de doble precisión (64 bits)
 - El dato vendrá expresado en formato IEEE 754 y se distribuye en los dos registros como muestra la figura:



neg.s fdest, forig # fdest = -forig

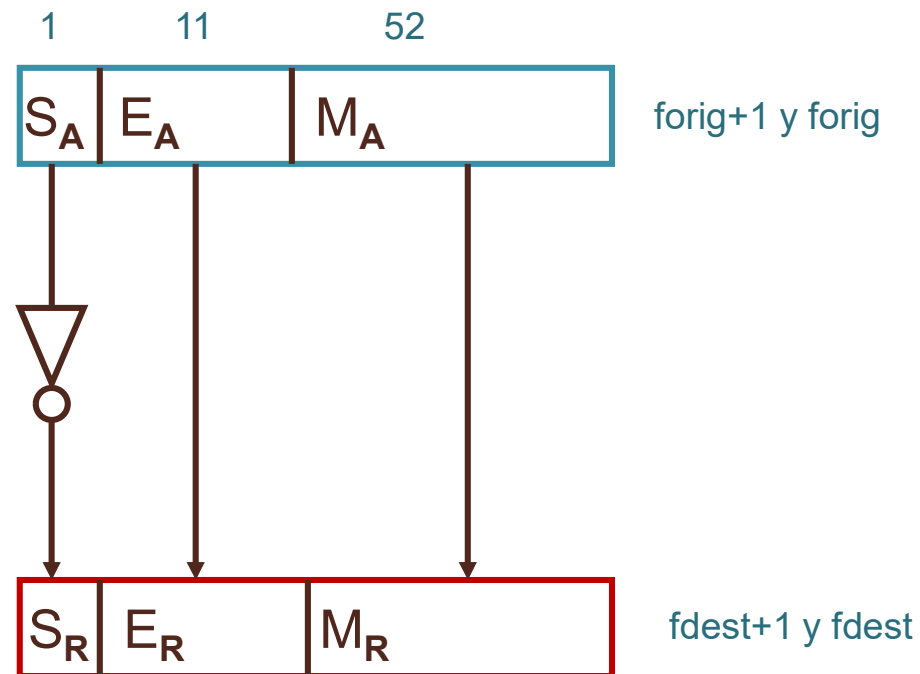
- Operaciones elementales necesarias:
 - ✓ Cambiar el signo: $S_R = \text{not } S_A$
 - ✓ Copiar el exponente: $E_R = E_A$
 - ✓ Copiar la mantisa: $M_R = M_A$



neg.d fdest, forig # fdest = -forig

- Operaciones elementales necesarias:

- ✓ Cambiar el signo: $S_R = \text{not } S_A$
- ✓ Copiar el exponente: $E_R = E_A$
- ✓ Copiar la mantisa: $M_R = M_A$



Emulación del cambio de signo, **neg.s**

```
float x = 1.0;
```

```
x = -x;
```

```
x:      .float 1.0
```

```
lw      $t0, x           # $t0 <- x
lui     $t1, 0x8000       # $t1 <- 0x80000000
xor     $t0, $t0, $t1     # $t0 <- -1.0
sw      $t0, x           # x <- $t0
```

Emulación del cambio de signo, **neg.s**

```
float x = 1.0;
```

```
x = -x;
```

$0x8000 = 10000 \dots 0000_2$

```
x: .float 1.
```

```
lw    $t0, x           # $t0 <- x
lui    $t1, 0x80000000  # $t1 <- 0x80000000
xor    $t0, $t0, $t1    # $t0 <- -1.0
sw     $t0, x           # x <- $t0
```

1 \oplus x = $\neg x$ # Invierte el valor entrante

0 \oplus x = x # deja el valor entrante inalterado

Emulación del cambio de signo, **neg.s**

```
float x = 1.0;
```

```
x = -x;
```

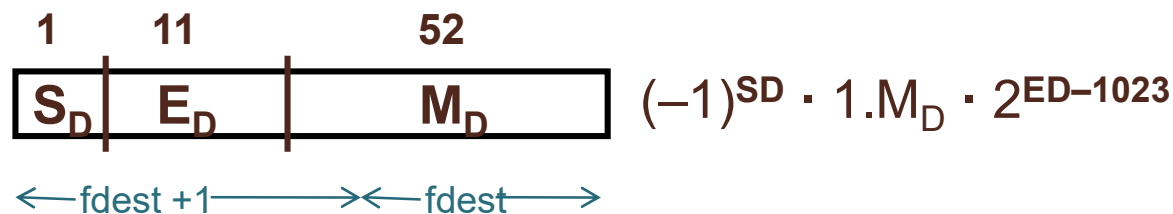
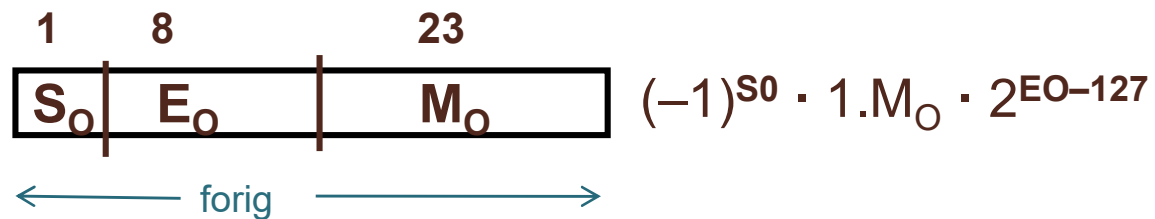
```
x:      .float 1.0
```

```
lw      $t0, x           # $t0 <- x
lui     $t1, 0x8000       # $t1 <- 0x80000000
xor     $t0, $t0, $t1     # $t0 <- -1.0
sw      $t0, x           # x <- $t0
```

¿Qué modificaciones introducirías si fuera
double x = 1.0 ?

cvt.d.s fdest, forig # (fdest+1 y fdest) ← forig

- Descripción de la operación:
 - ✓ El valor almacenado en forig está codificado en simple precisión (32 b)
 - ✓ Se debe traducir a formato de doble precisión (64 b)



cvt.d.s fdest, forig # (fdest+1 y fdest) \leftarrow forig

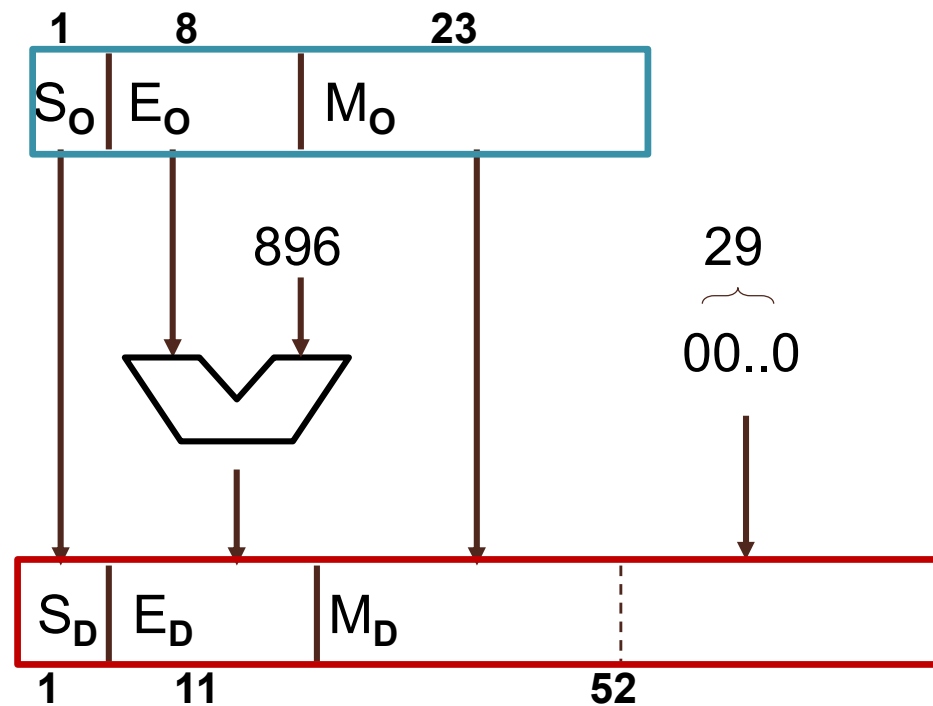
- **Operaciones elementales necesarias:**

- ✓ El signo no cambia: $S_D = S_O$
- ✓ Exponente: hay que cambiar de exceso 127 a exceso 1023
 - $E_D = (E_O - 127) + 1023 = E_O + 896$
- ✓ Mantisa: hay que añadir $52 - 23 = 29$ ceros a la derecha
 - $M_D = M_O || 00 \dots 0$

$\text{cvt.d.s fdest, forig} \# (\text{fdest} + 1 \text{ y fdest}) \leftarrow \text{forig}$

- **El operador básico**

- ✓ No trata los valores especiales



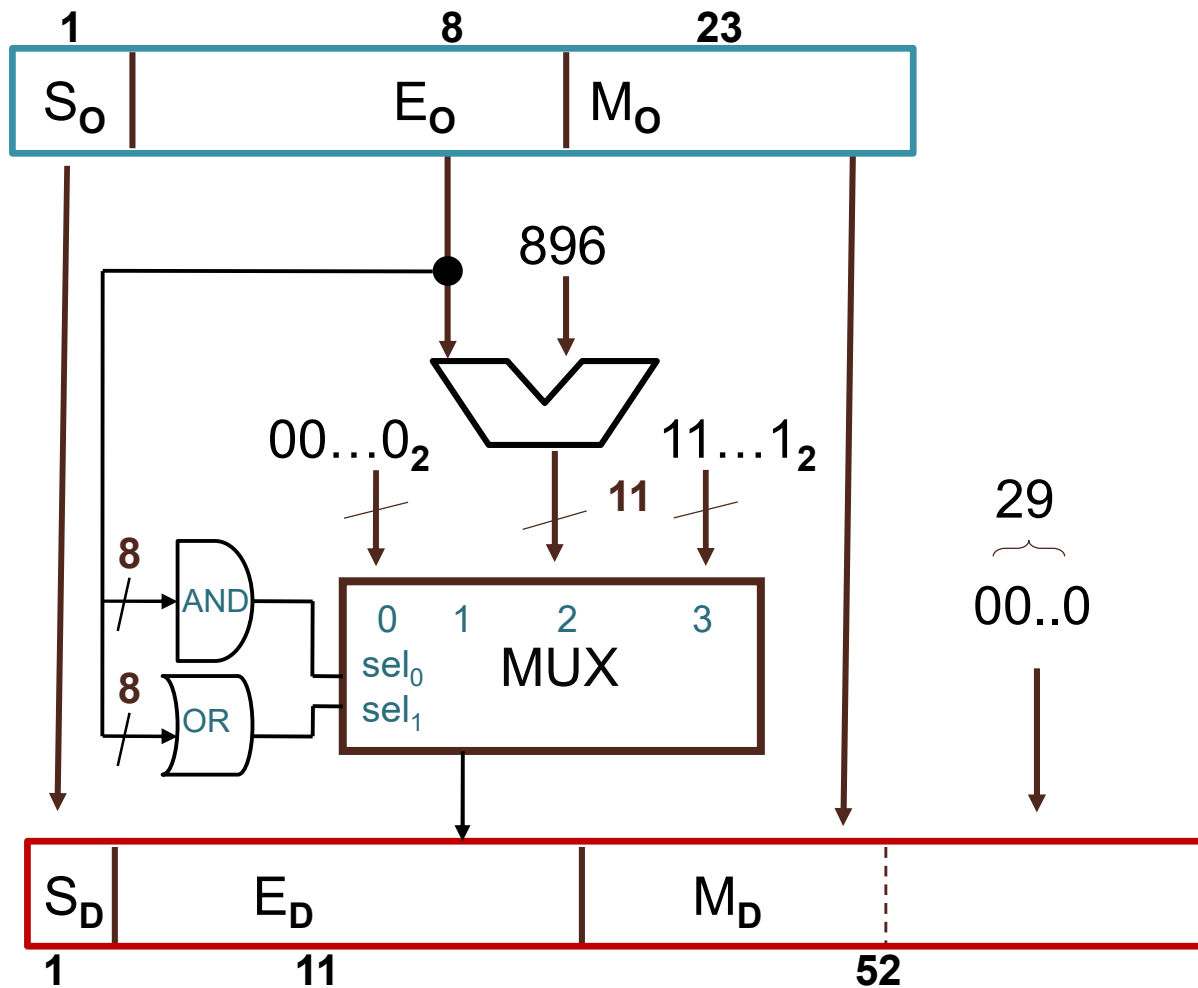
$\text{cvt.d.s fdest, forig} \# (\text{fdest} + 1 \text{ y fdest}) \leftarrow \text{forig}$

- **Valores especiales: Exponente cero o todo unos**

	E_o	S_D	E_D	M_D
zero y subnorma:	00000000_2	S_o	000000000000_2	$M_o \parallel 00\dots 0$
$\pm\infty$ y NaN:	11111111_2	S_o	111111111111_2	$M_o \parallel 00\dots 0$
Valores corrientes:	(otros valores)	S_o	$E_o + 896$	$M_o \parallel 00\dots 0$

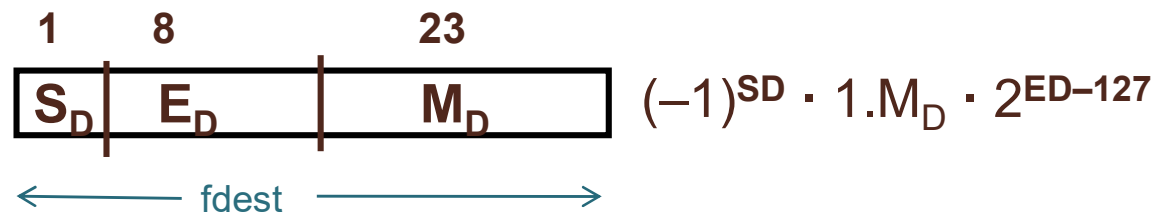
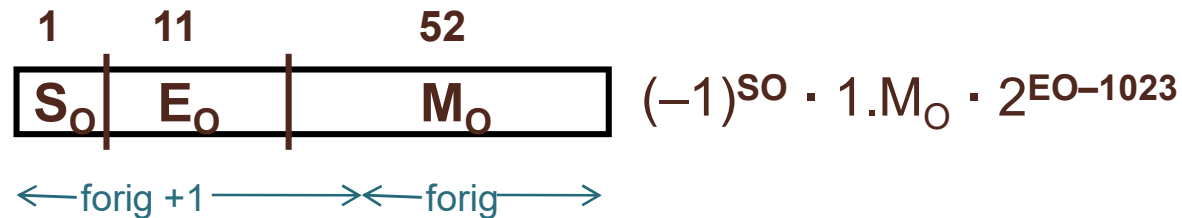
$\text{cvt.d.s } f_{\text{dest}}, \text{forig} \# (f_{\text{dest}} + 1 \text{ y } f_{\text{dest}}) \leftarrow \text{forig}$

- El operador básico, con valores especiales



cvt.s.d fdest, forig # fdest \leftarrow (forig+1 y orig)

- Descripción de la operación:
 - ✓ El valor almacenado en forig está codificado en doble precisión (64 b), abarca dos registros.
 - ✓ Se debe traducir a formato de simple precisión (32 b)



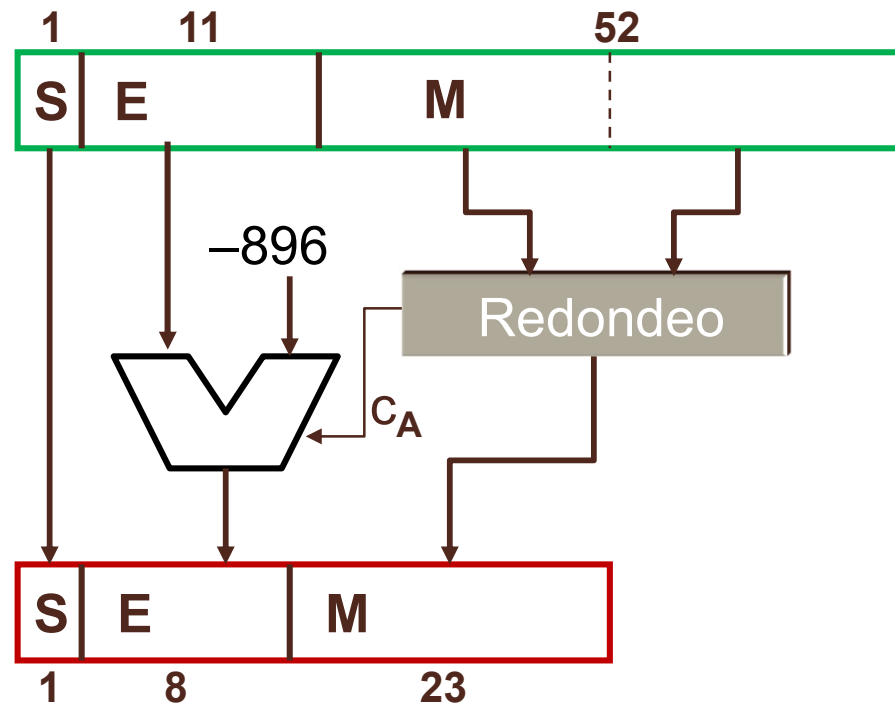
cvt.s.d fdest, forig # fdest \leftarrow (forig+1 y orig)

- **Operaciones elementales necesarias:**

- ✓ El signo no cambia: $S_D = S_O$
- ✓ Exponente: hay que cambiar de exceso 1023 a exceso 127
 - $E_D = (E_O - 1023) + 127 = E_O - 896$
- ✓ Exponente hay que adaptarlo de 11 a 8 bits.
 - Podría haber desbordamiento, que no vamos a considerar
- ✓ Mantisa: hay que adaptar de 52 a 23 bits
 - Habrá que redondear según dicta la norma IEEE 754

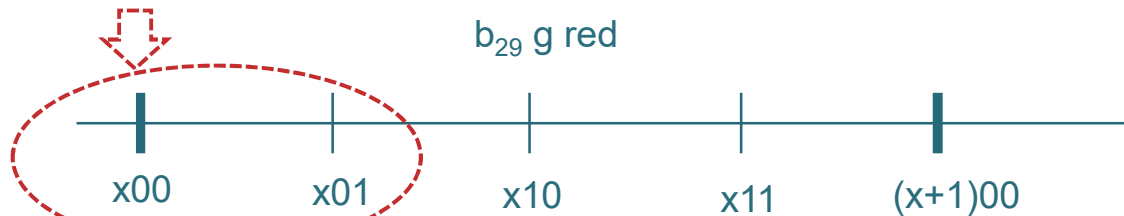
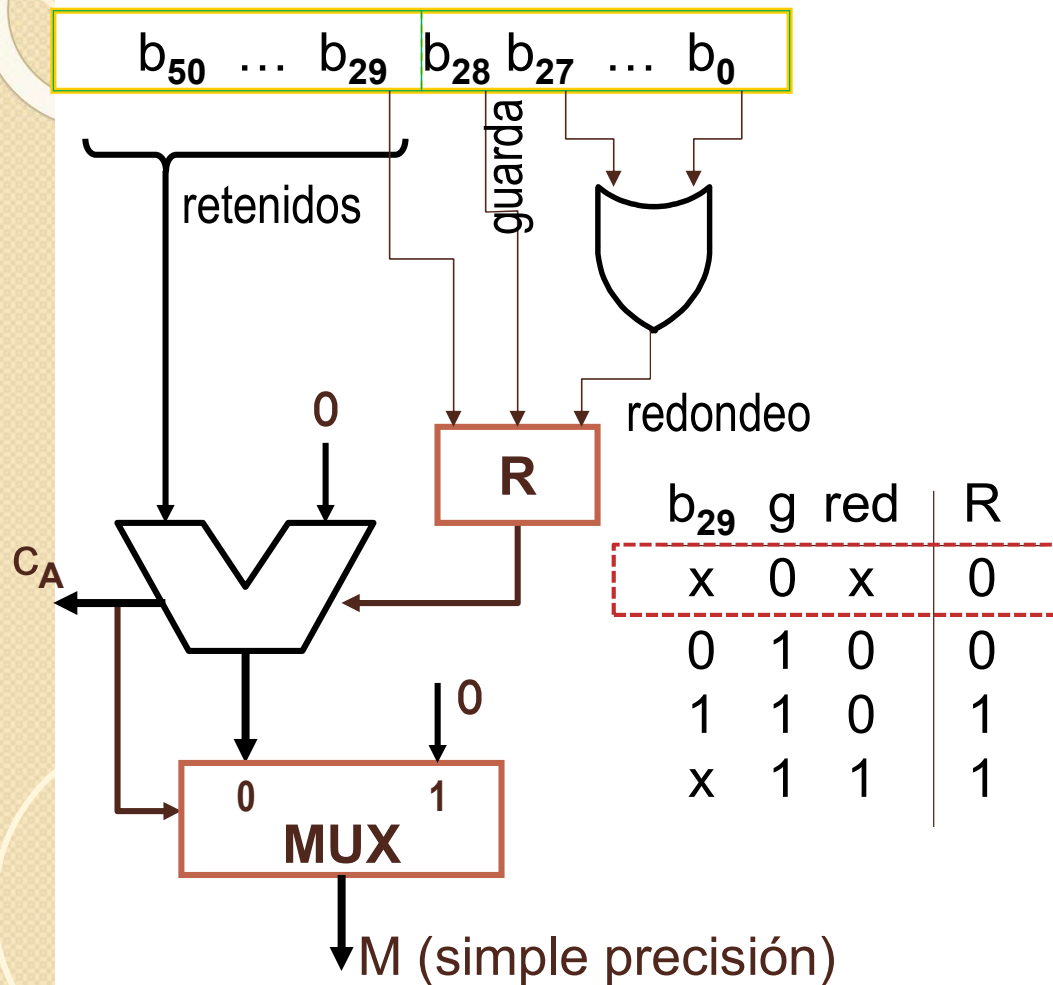
cvt.s.d fdest, forig # fdest \leftarrow (forig+1 y orig)

- **El operador básico:**



El redondeo

- Circuito para el redondeo al más próximo



Si $R=0$, truncar:

$$\begin{array}{r}
 1.1001101 \\
 + \quad \quad \quad 0 \\
 \hline
 1.10011
 \end{array}$$

Si $R=1$, incrementar:

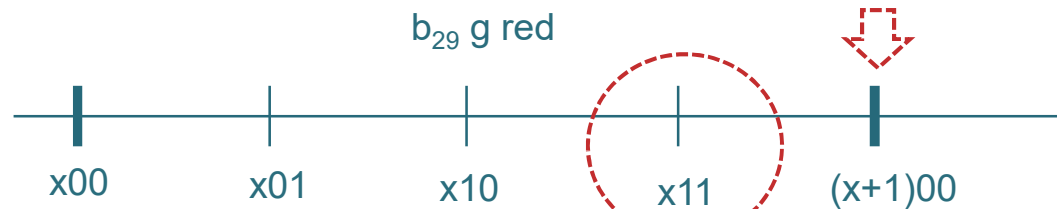
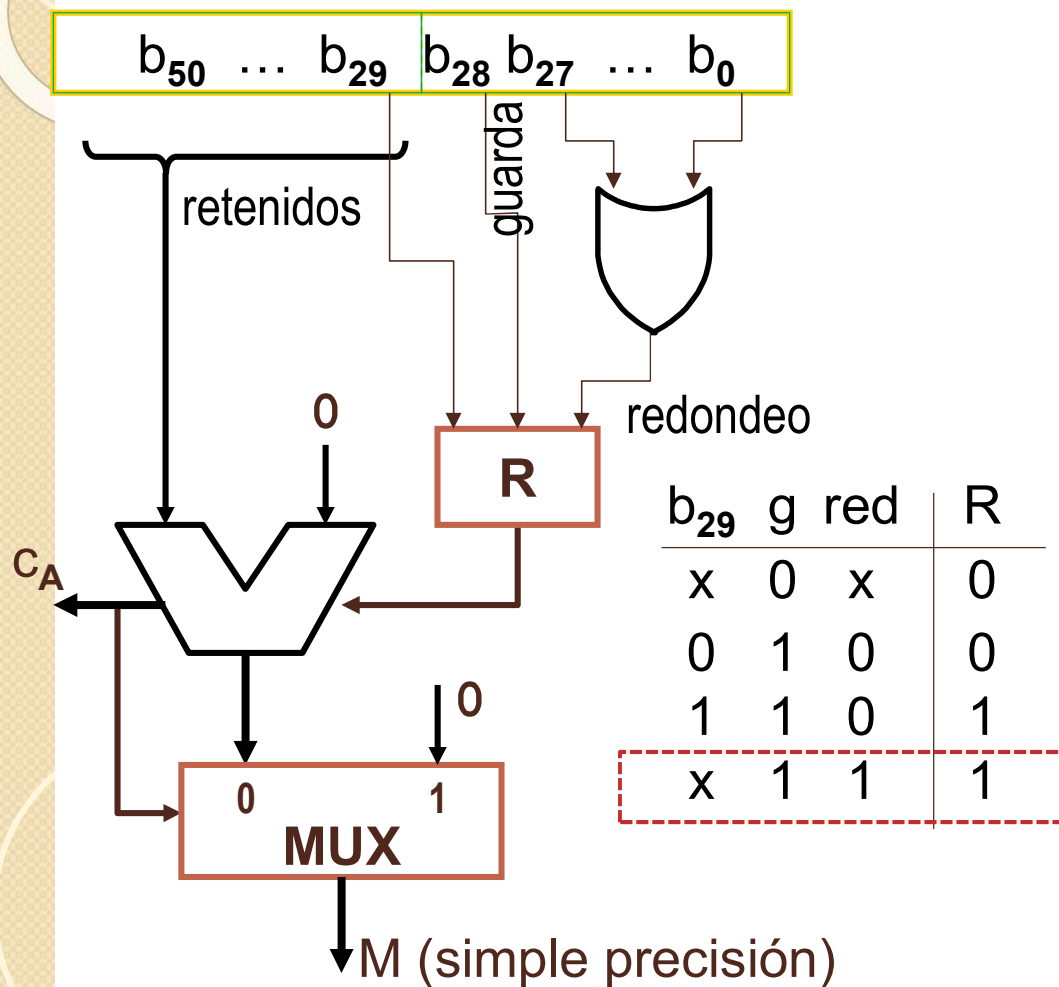
$$\begin{array}{r}
 1.1001111 \\
 + \quad \quad \quad 1 \\
 \hline
 1.10100
 \end{array}$$

Si $c_A=1$, corregir

$$\begin{array}{r}
 1.1111111 \\
 + \quad \quad \quad 1 \\
 \hline
 10.00000 \\
 + 1.00000 \\
 \hline
 11.00000
 \end{array}$$

El redondeo

- Circuito para el redondeo al más próximo



Si $R=0$, truncar:

$$\begin{array}{r} 1.1001101 \\ + 0 \\ \hline 1.10011 \end{array}$$

Si $R=1$, incrementar:

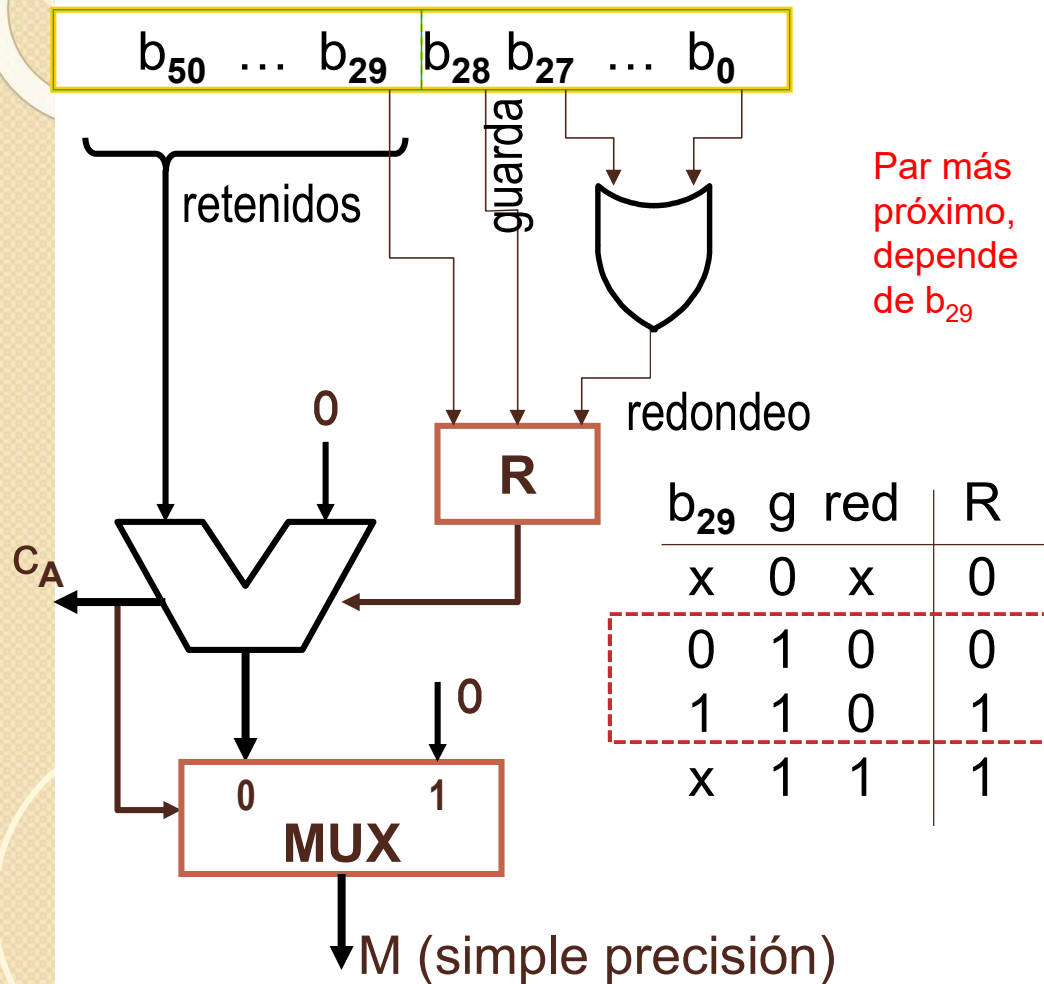
$$\begin{array}{r} 1.1001111 \\ + 1 \\ \hline 1.10100 \end{array}$$

Si $c_A=1$, corregir

$$\begin{array}{r} 1.1111111 \\ + 1 \\ \hline 10.00000 \\ + 1.00000 \\ \hline 11.00000 \end{array}$$

El redondeo

- Circuito para el redondeo al más próximo



Si $R=0$, truncar:

$$\begin{array}{r} 1.1001101 \\ + \\ \hline 1.10011 \end{array}$$

Si $R=1$, incrementar:

$$\begin{array}{r} 1.1001111 \\ + \\ \hline 1.10100 \end{array}$$

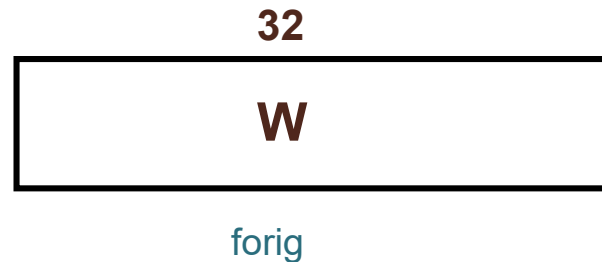
Si $c_A=1$, corregir

$$\begin{array}{r} 1.1111111 \\ + \\ \hline 10.00000 \\ + 1.00000 \\ \hline 11.00000 \end{array}$$

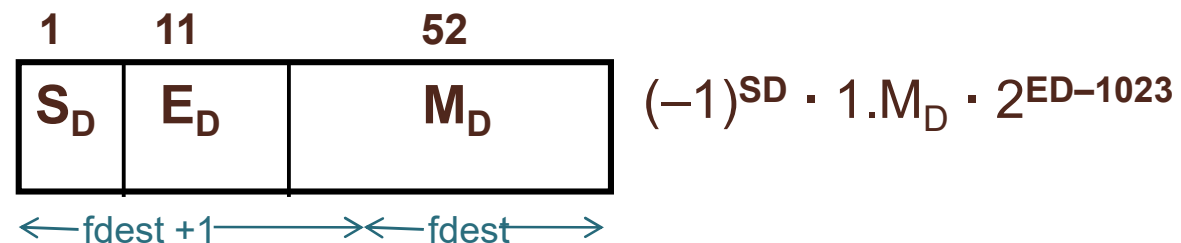
cvt.d.w fdest, forig # (fdest+1 y fdest) ← forig

- Descripción de la operación:

- ✓ El valor almacenado en forig es un número entero (32 b)



- ✓ Se debe traducir a formato de doble precisión (64 b)



cvt.d.w fdest, forig # (fdest+1 y fdest) ←forig

• Filosofía del operador

- ✓ Si W es positivo se puede escribir como $+0.W \times 2^{32}$
- ✓ Si es negativo, se reescribe como $-0.(-W) \times 2^{32}$
- ✓ La mantisa W comenzará por una serie de Z ceros ($0 \leq Z \leq 32$)



- ✓ Habrá que normalizar la mantisa desplazándola $Z+1$ posiciones hacia la izquierda y restar $Z+1$ al exponente

$$W = 1, X \dots X \cdot 2^{32-(Z+1)}$$

cvt.d.w fdest, forig # (fdest+1 y fdest) ←forig

- **Operaciones a realizar:**

- ✓ SD = Signo(W)

- El signo es el bit 31, ya que el dato viene expresado en Ca2

- ✓ MD

- Calcular $|W|$

- Si $W < 0$, hay que hacer $W = -W$

- Desplazar $Z + 1$ bits a la izquierda y completar con ceros

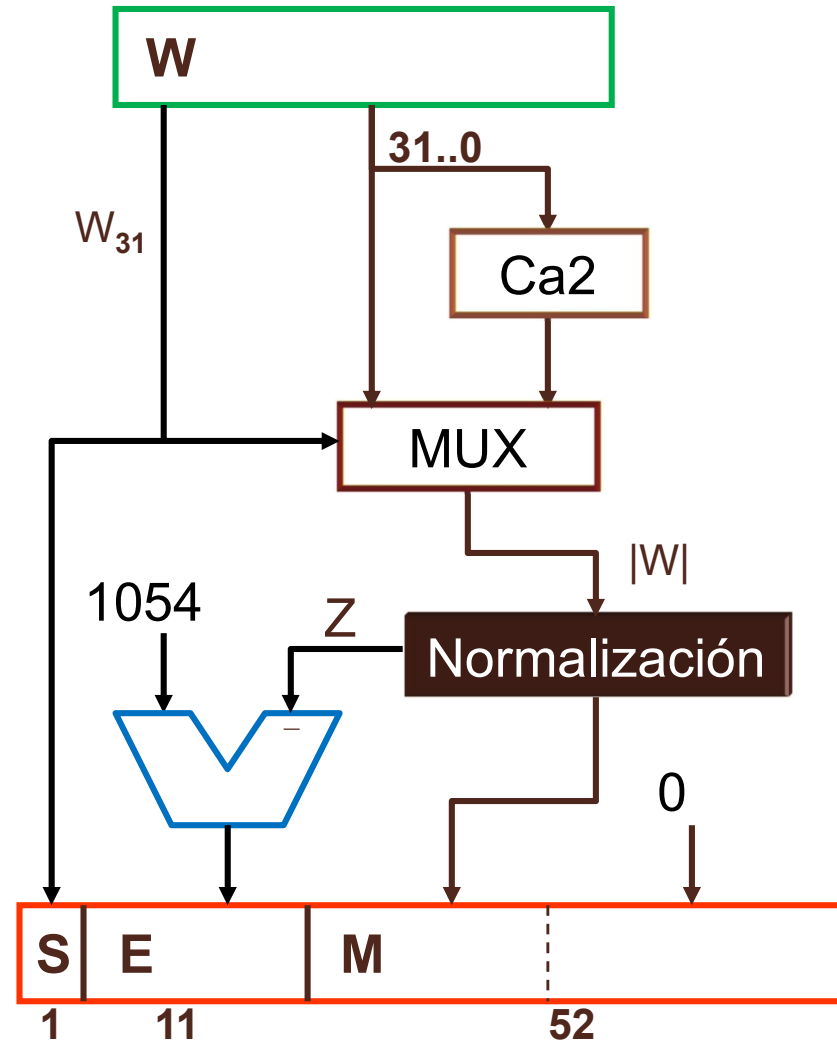
- $|W| \ll Z + 1$ (operador \ll es un desplazador)

- ✓ ED, hay que normalizarlo en doble precisión:

- $1023 + 32 - Z - 1 = 1054 - Z$

$\text{cvt.d.w fdest, forig} \quad \# \quad (\text{fdest} + 1 \text{ y fdest}) \leftarrow \text{forig}$

- El circuito básico:

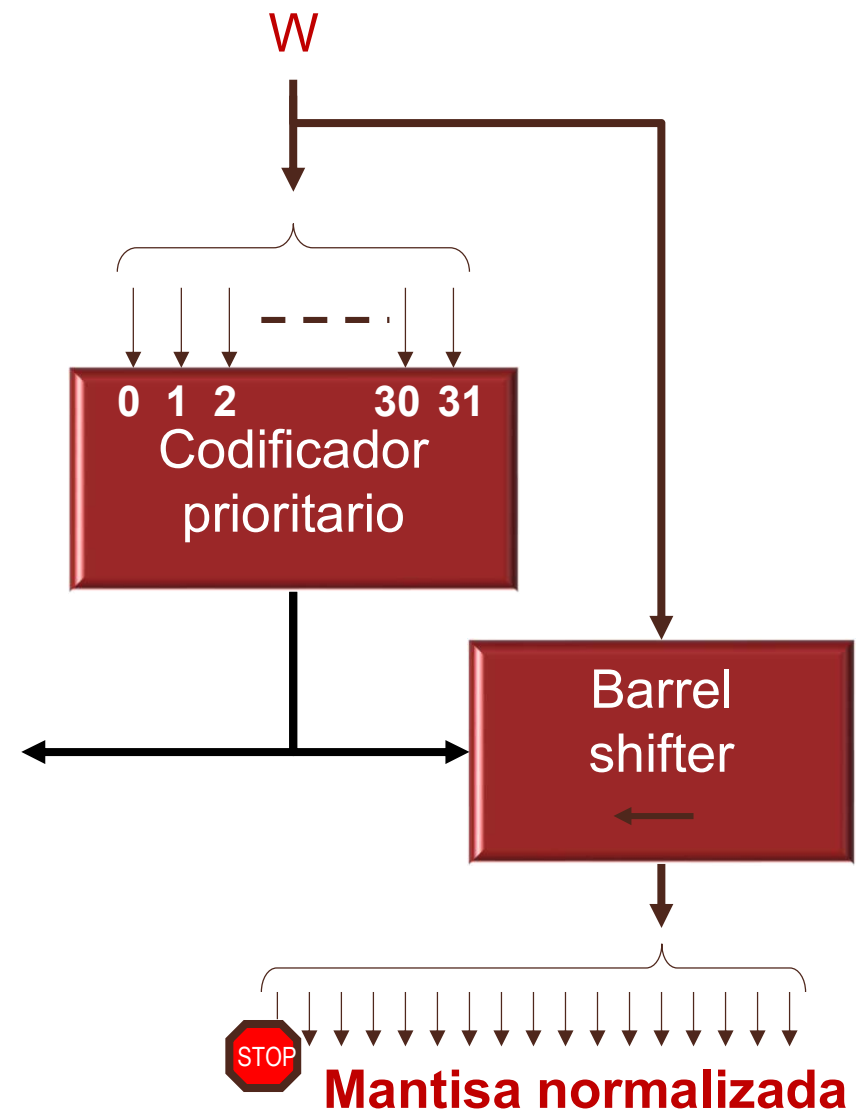


La normalización

Circuito de normalización

- ✓ Un codificador prioritario (que codifica la entrada de menor índice con un 1) calcula Z
- ✓ Un barrel shifter desplaza la mantisa hacia la izquierda y elimina los ceros sobrantes
- ✓ Se descarta el bit implícito

Codificador prioritario							Z
W_{31}	W_{30}	W_{29}	W_{28}	...	W_1	W_0	
1	X	X	X	...	X	X	00000
0	1	X	X	...	X	X	00001
0	0	1	X	...	X	X	00010
...
0	0	0	0	...	0	1	11111



mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

- Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{rcl}
 (-1)^{S_A} \cdot 1.M_A \cdot 2^{E_A} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_A & E_A+127 & M_A \\ \hline \end{array} & \equiv fA \\
 \times \quad (-1)^{S_B} \cdot 1.M_B \cdot 2^{E_B} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_B & E_B+127 & M_B \\ \hline \end{array} & \equiv fB \\
 \hline
 (-1)^{S_R} \cdot 1.M_R \cdot 2^{E_R} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_R & E_R+127 & M_R \\ \hline \end{array} & \equiv fdest
 \end{array}$$

```
mul.s fdest, fA, fB    # fdest ← fA × fB
```

- **Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{rcl}
 (-1)^{S_A} \cdot 1.M_A \cdot 2^{E_A} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_A & E_A+127 & M_A \\ \hline \end{array} & \equiv f_A \\
 \times & & \\
 (-1)^{S_B} \cdot 1.M_B \cdot 2^{E_B} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_B & E_B+127 & M_B \\ \hline \end{array} & \equiv f_B \\
 \hline
 (-1)^{S_R} \cdot 1.M_R \cdot 2^{E_R} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_R & E_R+127 & M_R \\ \hline \end{array} & \equiv f_{dest} \\
 \swarrow & & \\
 & SR = SA \oplus SB &
 \end{array}$$

mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

- **Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

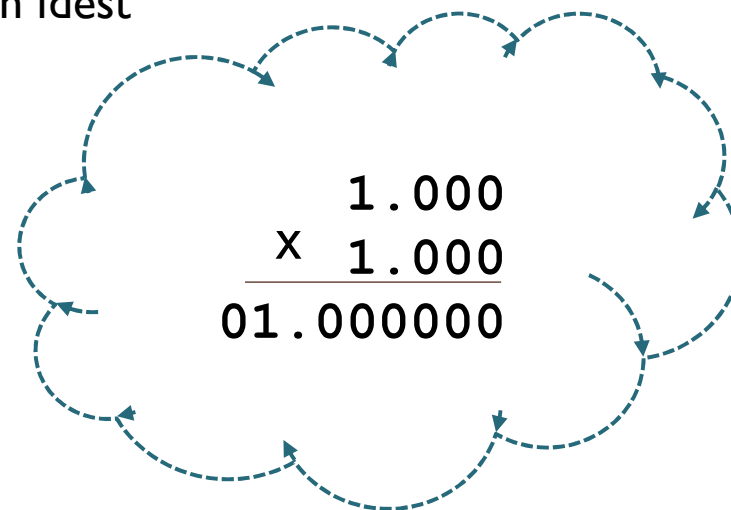
$$\begin{array}{r} \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_A = 1, XX...X \end{array} \\ \times \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_B = 1, XX...X \end{array} \\ \hline \begin{array}{c} 01, X.....X \\ 10, X.....X \\ 11, X.....X \end{array} \\ \xleftrightarrow{64b} \end{array}$$

mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

- Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{r}
 \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_A = 1, XX...X \end{array} \\
 \times \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_B = 1, XX...X \end{array} \\
 \hline
 \begin{array}{l}
 01, X.....X \quad \text{Ya normalizado} \\
 10, X.....X \\
 11, X.....X \\
 \xleftrightarrow{64b}
 \end{array}
 \end{array}$$

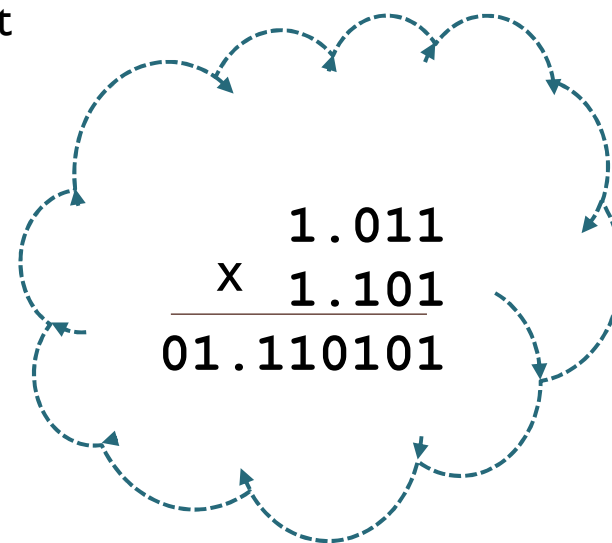


mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

- Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{r}
 \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_A = 1, XX...X \end{array} \\
 \times \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_B = 1, XX...X \end{array} \\
 \hline
 \begin{array}{l}
 01, X.....X \quad \text{Ya normalizado} \\
 10, X.....X \\
 11, X.....X \\
 \xleftrightarrow{64b}
 \end{array}
 \end{array}$$

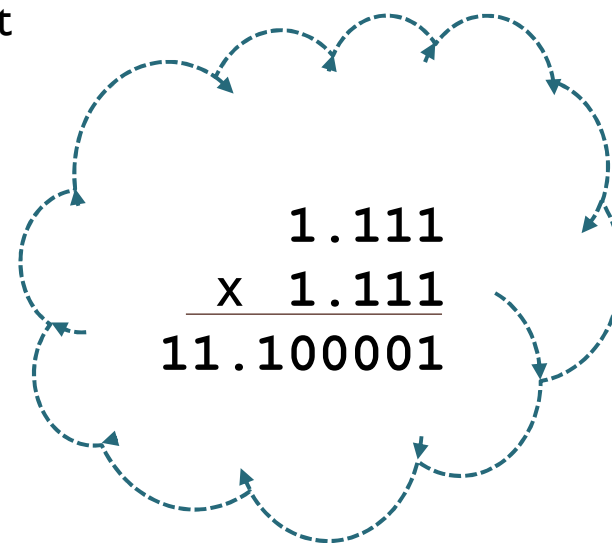


mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

- Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{r}
 \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_A = 1, XX...X \end{array} \\
 \times \begin{array}{c} \xleftrightarrow{32b} \\ 1.M_B = 1, XX...X \end{array} \\
 \hline
 \begin{array}{l}
 01, X.....X \quad \text{Ya normalizado} \\
 10, X.....X \\
 11, X.....X \quad \left. \vphantom{\begin{array}{l} 10, X.....X \\ 11, X.....X \end{array}} \right\} \text{Hay que normalizar} \\
 \xleftrightarrow{64b}
 \end{array}
 \end{array}$$



mul.s fdest, fA, fB # fdest \leftarrow fA \times fB

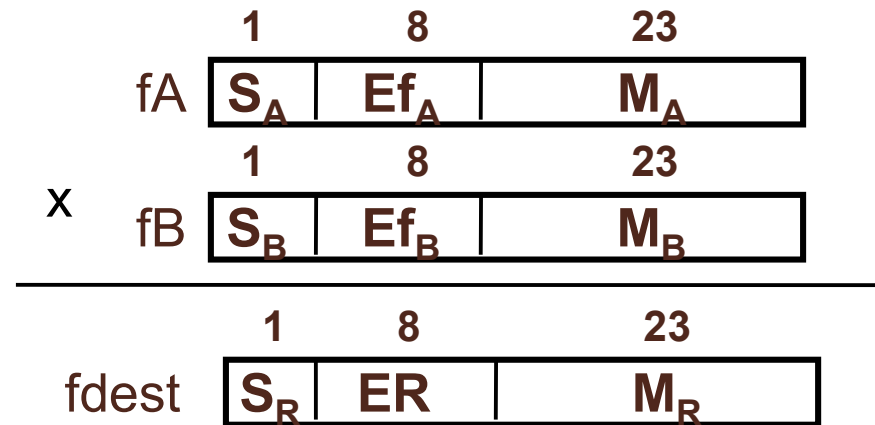
- Descripción de la operación:**

- ✓ Realiza la operación de multiplicación de dos valores en simple precisión almacenados en fA y fB, el resultado lo deja en fdest
- ✓ La operación matemática es la siguiente:

$$\begin{array}{rcl}
 (-1)^{S_A} \cdot 1.M_A \cdot 2^{E_A} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_A & E f_A = E_A + 127 & M_A \\ \hline \end{array} & \equiv fA \\
 \times & & \\
 (-1)^{S_B} \cdot 1.M_B \cdot 2^{E_B} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_B & E f_B = E_B + 127 & M_B \\ \hline \end{array} & \equiv fB \\
 \hline
 (-1)^{S_R} \cdot 1.M_R \cdot 2^{E_R} & \begin{array}{|c|c|c|} \hline 1 & 8 & 23 \\ \hline S_R & E f_R = E_R + 127 & M_R \\ \hline \end{array} & \equiv fdest
 \end{array}$$

$\rightarrow E_R = E_A + E_B$
 $E f_R = E_A + E_B + 127 = E f_A + E f_B - 127 + \text{normalización si procede}$

mul.s fdest, fA, fB

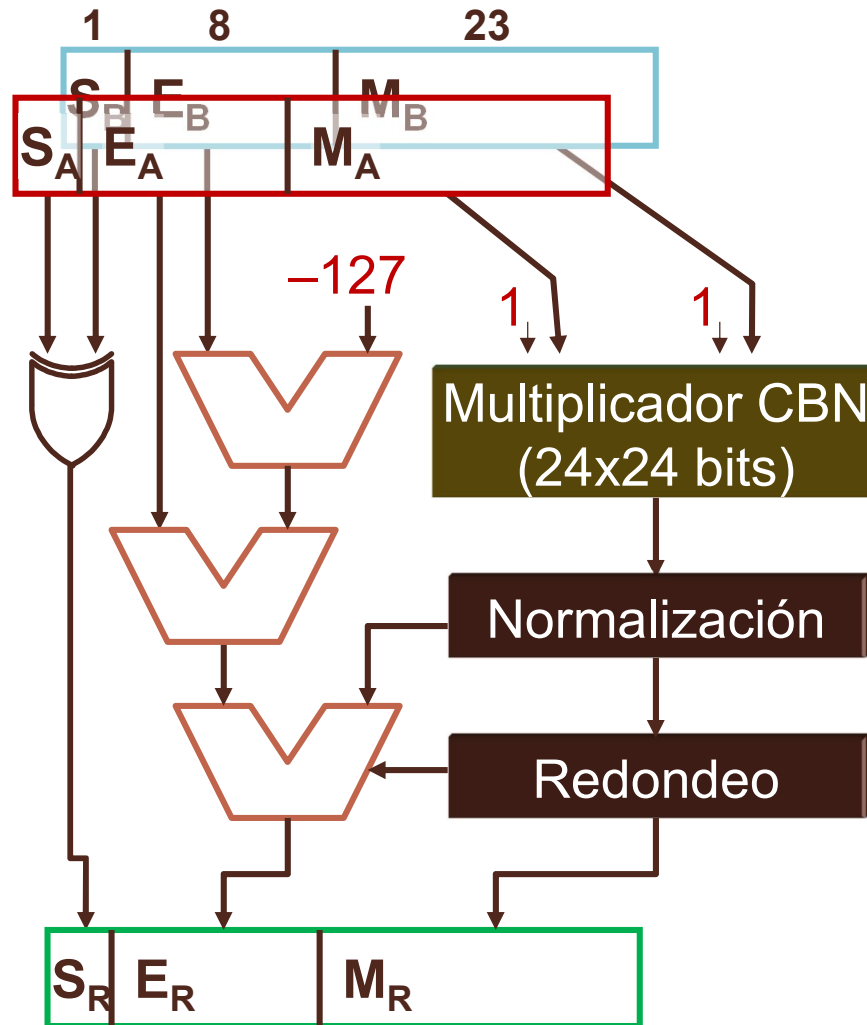


- **Operaciones elementales necesarias:**

- ✓ Cálculo del signo: $SR = SA \oplus SB$
- ✓ Cálculo del exponente: $ER = Ef_A + Ef_B - 127 + \text{posibles normalizaciones}$
- ✓ Cálculo de la mantisa: MR
 - Multiplicar $1.M_A \times 1.M_B$ (considerando el bit implícito, 24bits)
 - El resultado serán $24 \times 24 = 48$ bits
 - Si el bit de mayor peso de la operación es uno, habrá que normalizar el resultado sumando uno al exponente
 - Luego habrá que reducir el resultado para adaptarlo a 23, redondeando y normalizando de nuevo si fuera preciso

mul.s fdest, fA, fB

- El operador básico:

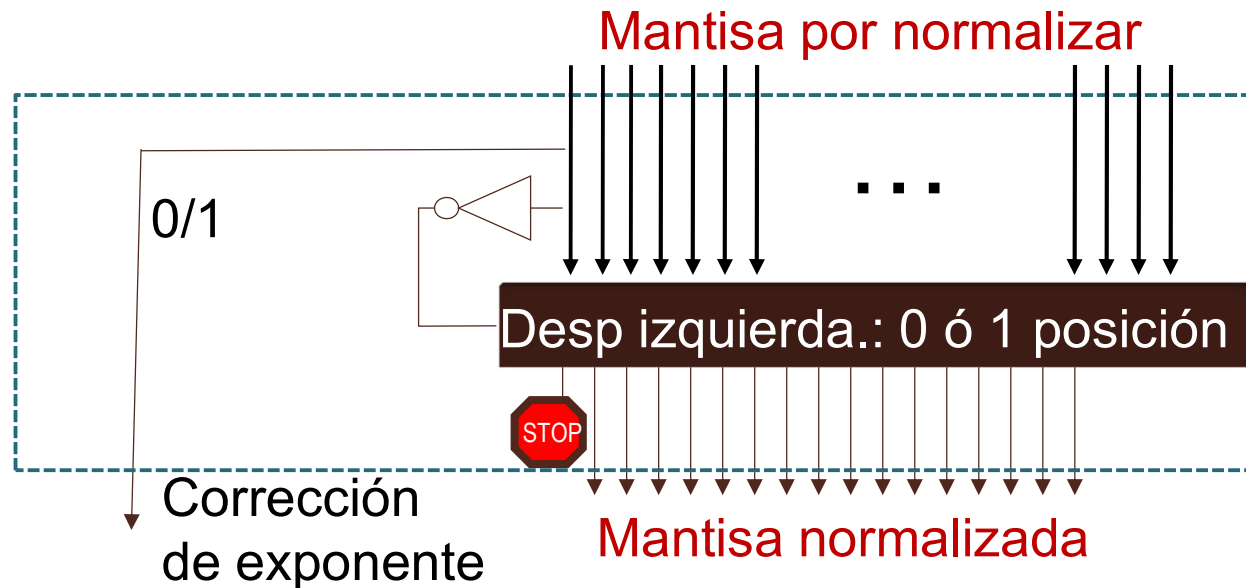


Circuito de normalización

✓ Normalización:

- Si la mantisa comienza por 0: **0X,XXX.....X**
 - Desplazar a la izquierda una posición
 - No se sumará nada al exponente (ya normalizada)
- Si comienza por 1: **1X,XXX...XXX**
 - Se sumará uno al exponente (normalizar)

✓ Eliminar el bit de más peso (implícito)



$$\begin{array}{r} 1.000 \\ \times 1.000 \\ \hline 01.000000 \\ \leftarrow \\ 10.000000 \end{array}$$

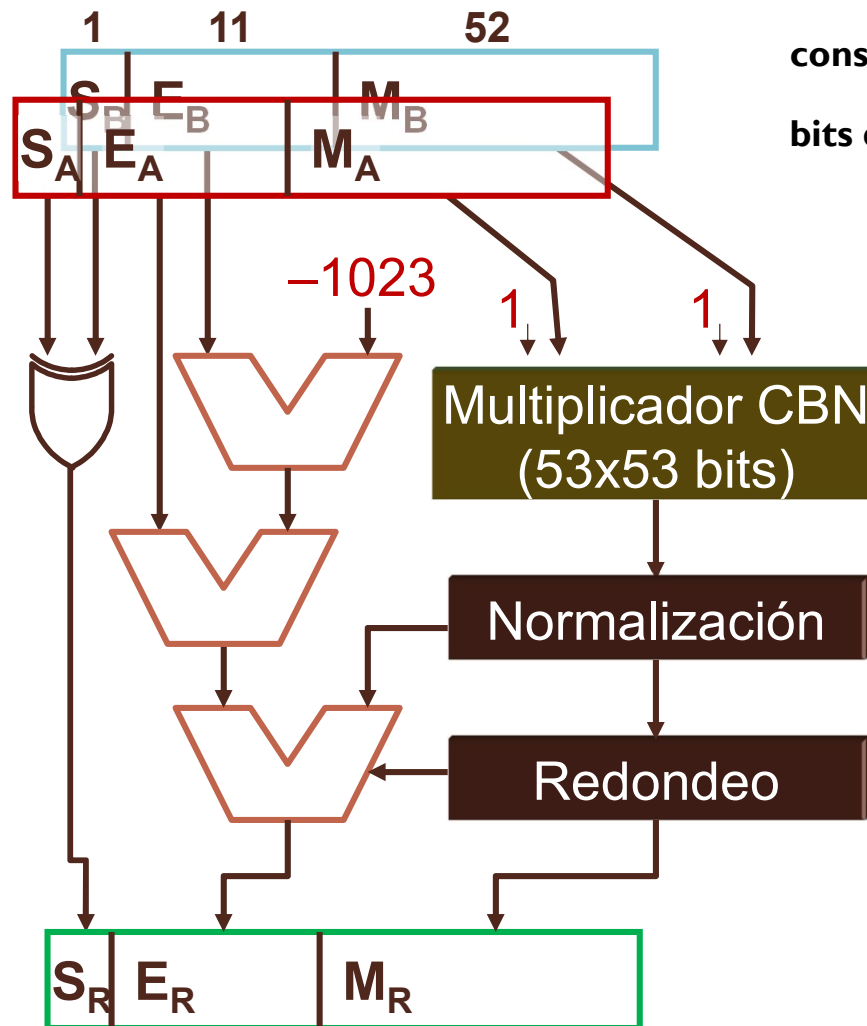
0000000

$$\begin{array}{r} 1.011 \\ \times 1.101 \\ \hline 01.110101 \\ \leftarrow \\ 11.101010 \\ 1101010 \end{array}$$

$$\begin{array}{r} 1.111 \\ \times 1.111 \\ \hline 11.100001 \\ 11.100001 \\ 1100001 \end{array}$$

mul.d fdest, fA, fB

- Adaptar este circuito para que realice una multiplicación en doble precisión consistirá en aumentar el número de bits de los circuitos del operador



Estructura de Computadores

Tema 4

Aritmética en coma flotante

Fin

DISCA
