

Estructura de Computadores

Tema 4

Aritmética en coma flotante

DISCA

A decorative graphic consisting of four horizontal white lines of varying lengths, arranged in a staggered, overlapping fashion, located in the bottom right corner of the slide.

Índice

- Introducción
 - Medida del rendimiento
- La norma IEEE y su implementación en el MIPS
 - Formato, valores especiales, redondeo
 - 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 1, 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)

Introducción

DISCA



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 es el número máximo de operaciones por segundo. Viene determinada por el tiempo de operación de cada circuito.
- En las comparativas entre computadores o entre aceleradores gráficos: es el número de operaciones de CF que ejecuta el dispositivo por segundo.
 - Depende del número y características de los operadores incluídos y del uso que se hace de ellos.
 - **Productividad punta** de un computador o de un acelerador gráfico: suma de las productividades de los operadores de CF incluídos. 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

La norma IEEE 754 y su implementación en el MIPS

■

DISCA

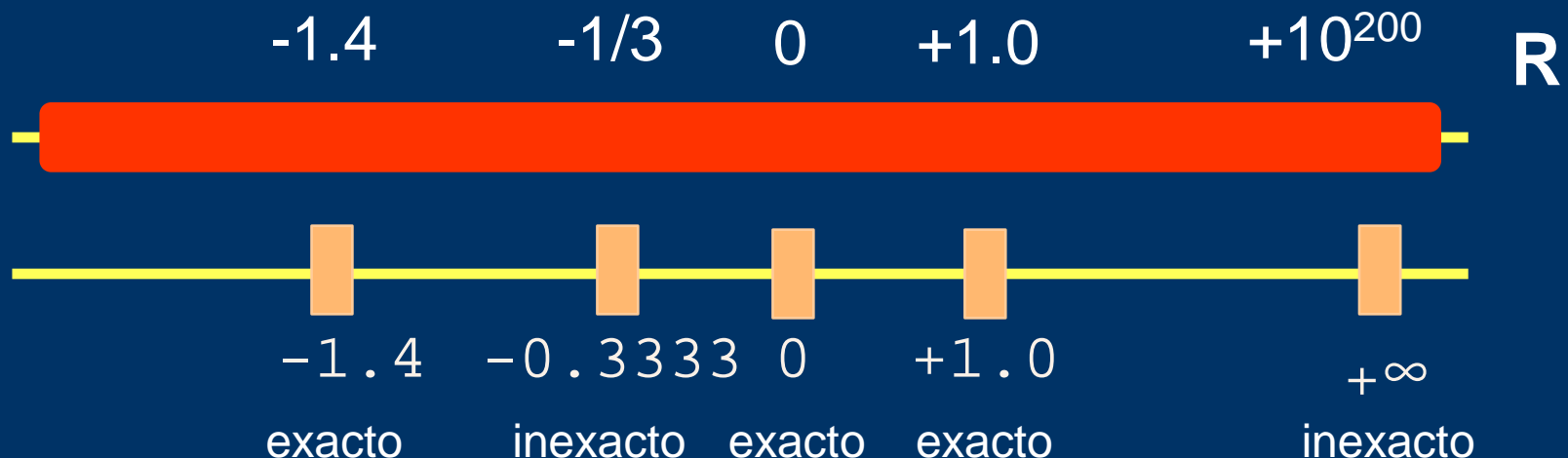


Notas sobre la representación de números

- **Representación del conjunto R (reales positivos y negativos)**
 - El conjunto R es un conjunto denso: entre dos números reales cualesquiera hay infinitos números reales
 - La representación del computador es limitada i 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 R
 - Hay números reales que tienen representación exacta otros que no, como los números con parte decimal periódica
 - ¿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)
 - El formato impone más restricciones a la representación: habrá algunas palabras de bits con una significación matemática especial, como el valor infinito o el cero

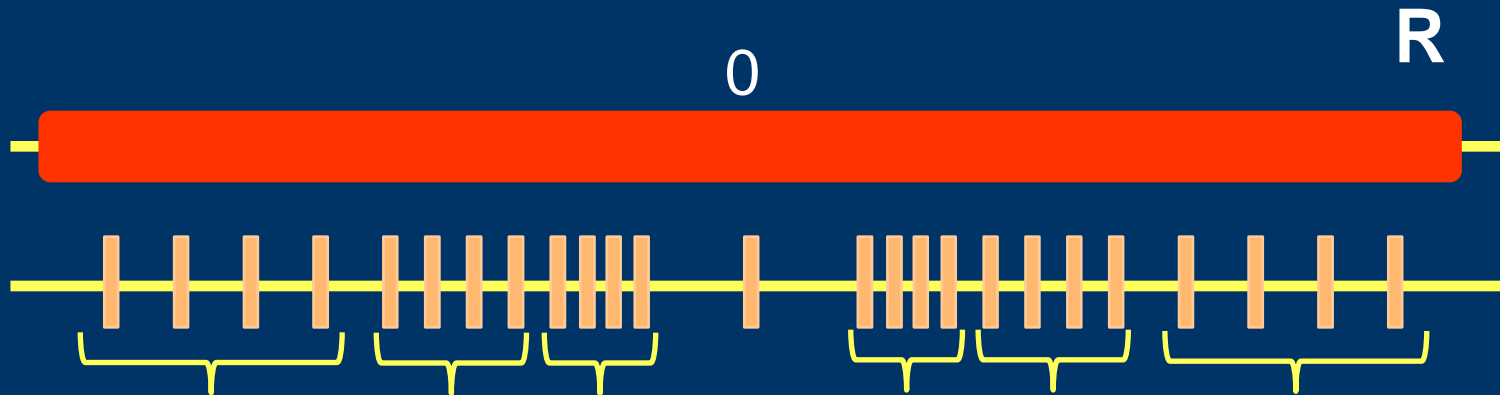
Notas sobre la representación de números

- 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



Notas sobre la representación de números

- Patrón de la representación de los números reales
 - No hay valores representados muy cerca del cero
 - Para un mismo valor de exponente los números representados están separados por la misma distancia
 - Cuanto más grande es el exponente más distancia hay entre dos números representados consecutivos (la densidad de representación disminuye)

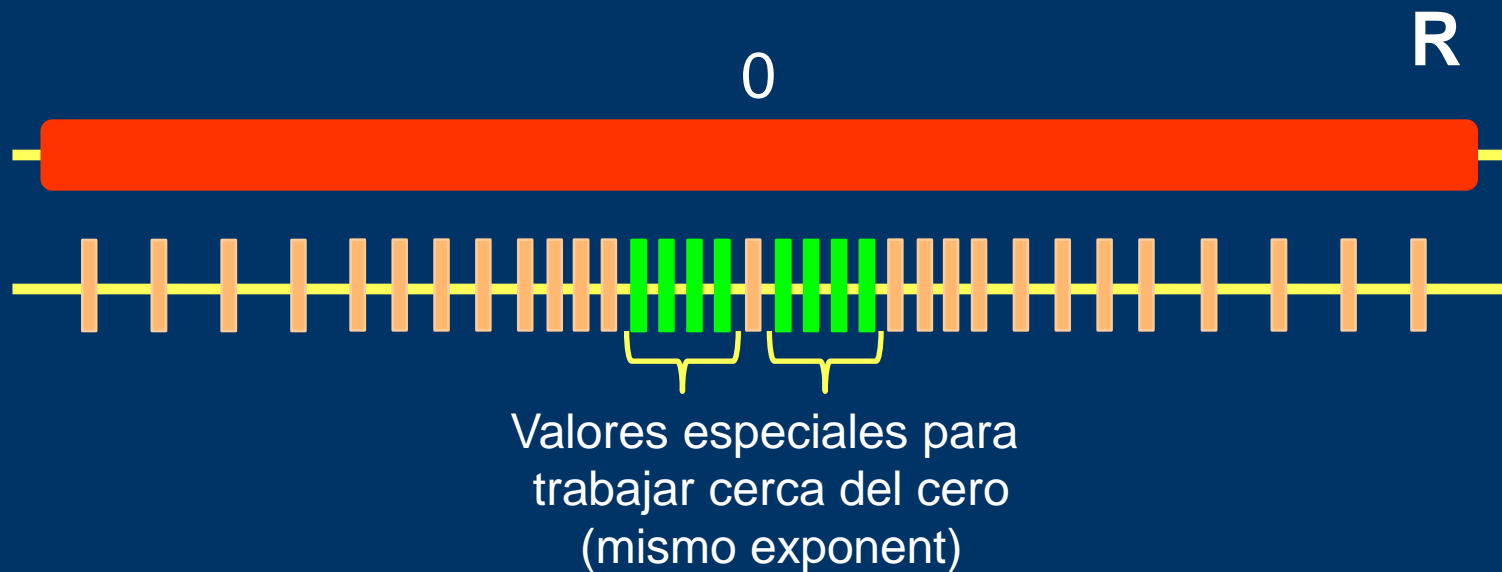


Cada grupo de valores tiene el mismo exponente y diferentes mantisas

Notas sobre la representación de números

■ 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)
- No todas las unidades de coma flotante soportan este subconjunto de valores



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

■ Representación

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

- Simple precisión (SP)

1	8	23
S	E	M

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

1	11	52
S	E	M

 $(-1)^S \cdot 1.M \cdot 2^{E-1023}$

– Valores subnormales

- (SP y DP)

S	000..00	M ≠ 0
---	---------	-------

 $(-1)^S \cdot 0.M \cdot 2^{-126}$
 $(-1)^S \cdot 0.M \cdot 2^{-1022}$

– Valores especiales (SP y DP)

± 0

S	000...00	000...00
---	----------	----------

$\pm \infty$

S	111...11	000...00
---	----------	----------

NaN

x	111...11	M ≠ 0
---	----------	-------

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*.
 - 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** → **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*:
 - Hacia $+\infty$
 - Hacia $-\infty$
 - Hacia 0
 - Escoger el más próximo de los dos (este es el modo por omisión)

La norma IEEE 754

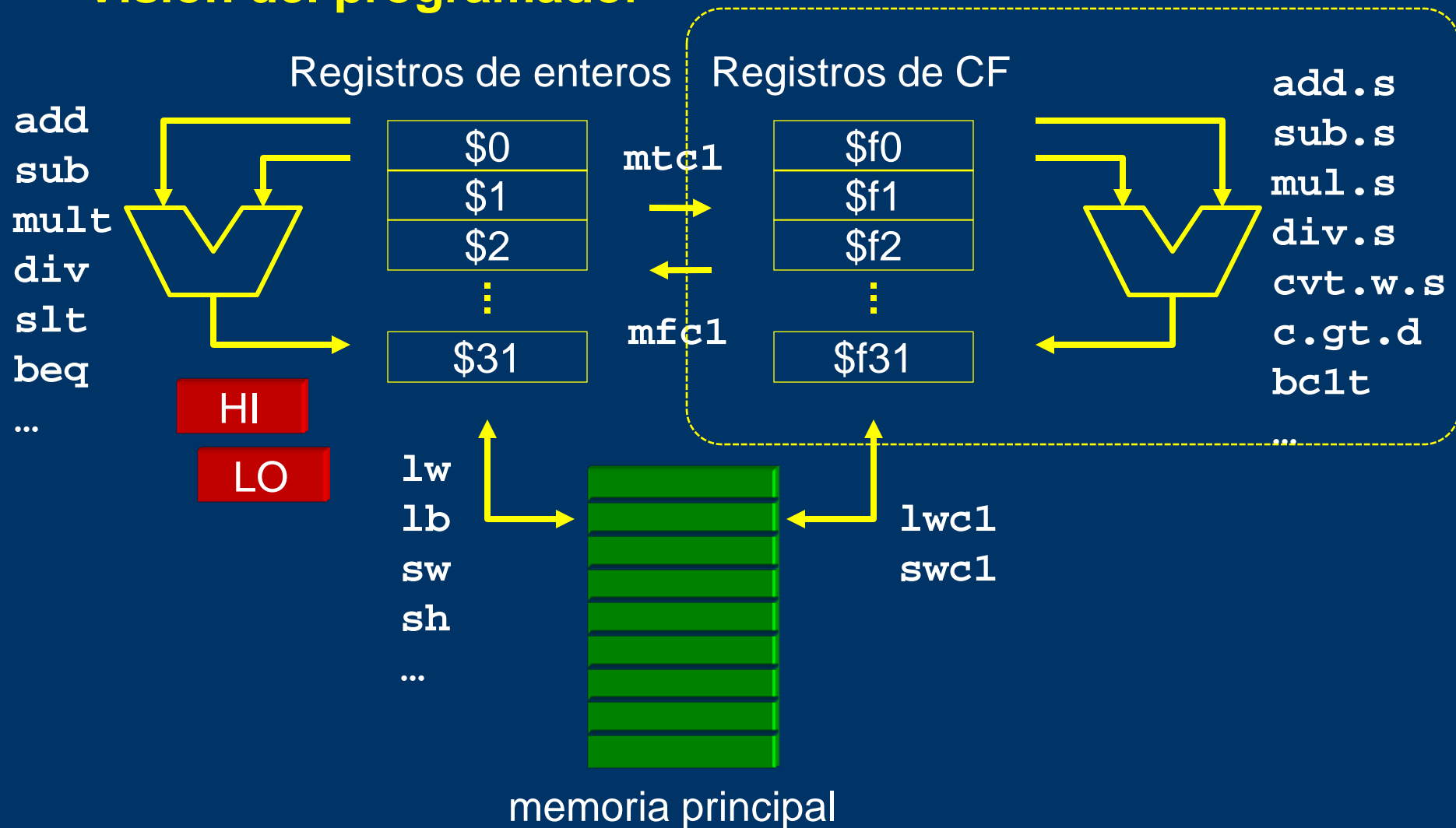
■ El redondeo hacia el más próximo (sesgado al par)

- 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)
- Ejemplo:

<u>M</u>	<u>se elige</u>	<u>M resultante</u>
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

La coma flotante en el MIPS

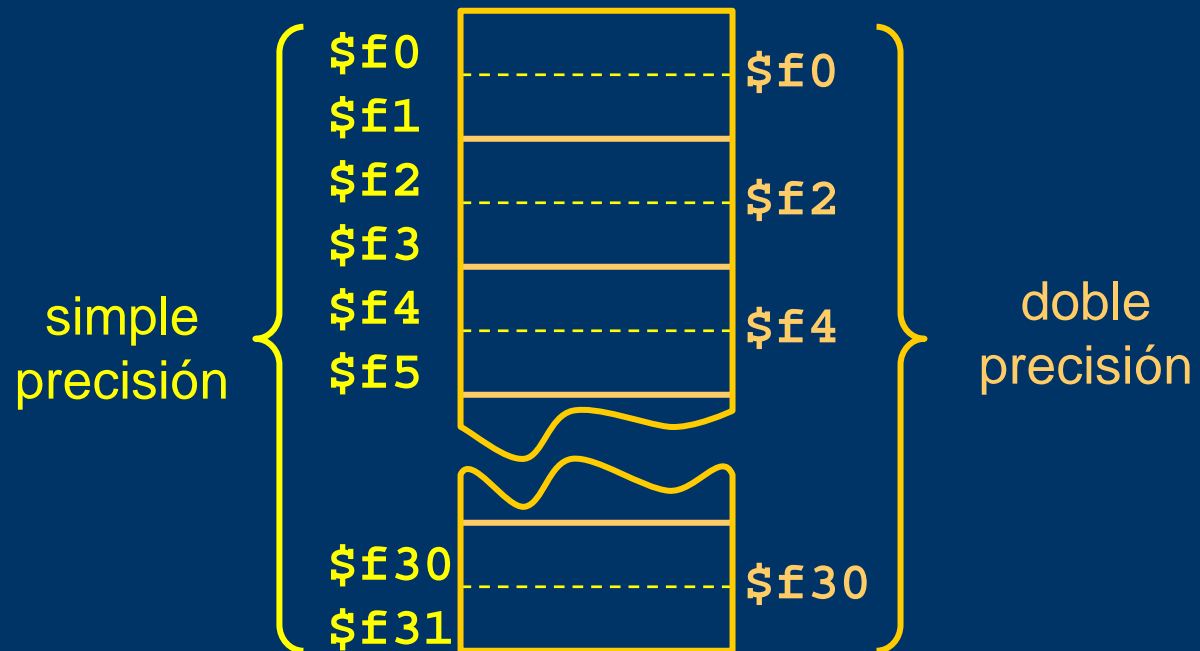
■ Visión del programador



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 $\$f0, \$f2, \dots, \$f30$
- 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

La coma flotante en el MIPS

■ Intercambio con la memoria y los registros de enteros

operación	instrucción
lectura $\$ft \leftarrow Mem[X+\$rs]$	<code>lwc1 \$ft,X(\$rs)</code>
escritura $Mem[X+\$rs] \leftarrow \ft	<code>swc1 \$ft,X(\$rs)</code>
transferencia $\$fs \leftarrow \rt	<code>mtc1 \$rt,\$fs</code>
transferencia $\$rt \leftarrow \fs	<code>mfc1 \$rt,\$fs</code>

fs i ft: registros de coma flotante
rs i rt: són 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
```

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

■ 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 `$f7` a CF en doble precisión contenido en `$f4||$f5`
- En combinación con las instrucciones de transferencia con el banco de registros de enteros, se puede hacer aritmética con variables de tipos diversos

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`

<u>operación</u>	<u>instrucción</u>
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>
comparación	<code>c.cond._ fs,ft</code>
copia	<code>mov._ fd,fs</code>
cambio de signo	<code>neg._ fd,fs</code>
valor absoluto	<code>abs._ fd,fs</code>

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` 0 `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
```



Operadores de coma flotante

DISCA



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

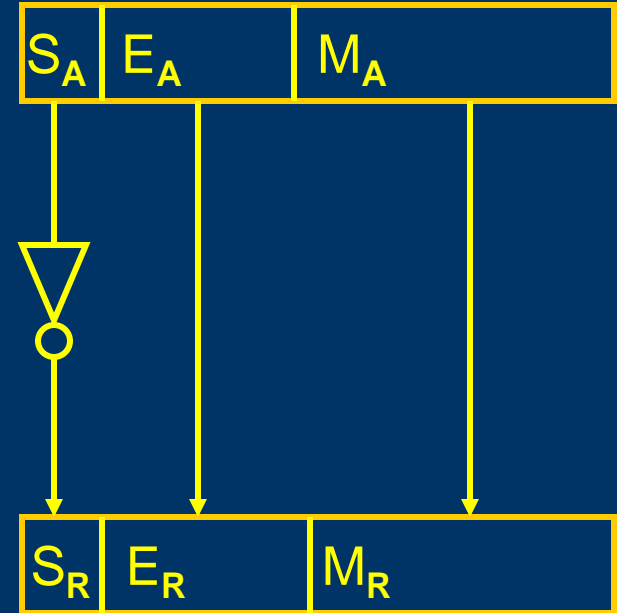
■ Ejemplos de operadores

- NEG.S i 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 i MULT.D (multiplicación)
 - Estructura básica
 - Detalle: la renormalización

Cambio de signo

■ Operador

- Dos versiones:
 - Simple precisión:
 - Entrada: S_A (1 bit), E_A (8 bits) y M_A (23 bits)
 - Salida: S_R (1 bit), E_R (8 bits) y M_R (23 bits)
 - Doble precisión:
 - Entrada: S_A (1 bit), E_A (11 bits) y M_A (52 bits)
 - Salida: S_R (1 bit), E_R (11 bits) y M_R (52 bits)
- Cambia el signo: $S_R = \text{not } S_A$
- Copia el exponente: $E_R = E_A$
- Copia la mantisa: $M_R = M_A$



Emulación del cambio de signo

```
float x = 1.0;
```

```
x = -x;
```

```
x:      .float 1.0
```

```
lwc1 $f2, x          # $f2 <- x (1.0)
mfc1 $t0, $f2        # $t0 <- $f2
lui $t1, 0x8000      # $t1 <- 0x80000000
xor $t0, $t0, $t1    # $t0 <- -1.0
mtc1 $t0, $f2        # $f2 <- $t0
swc1 $f2, x          # x <- $f2 (-1.0)
```

Conversión de simple a doble precisión (cvt.d.s)

■ Especificación del operador:

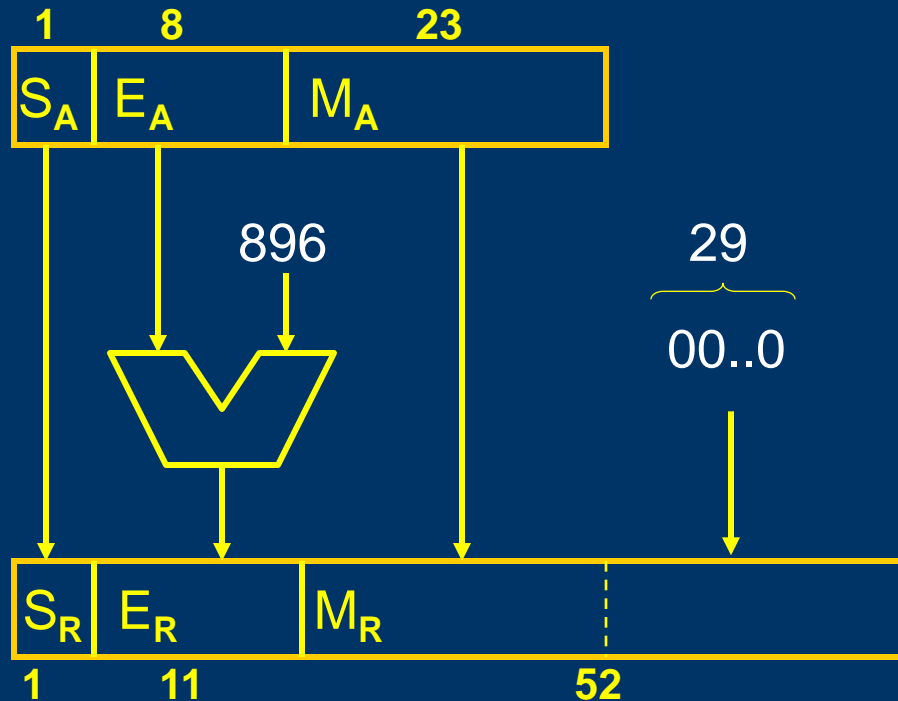
- Entrada: S_A (1 bit), E_A (8 bits) y M_A (23 bits)
- Salida: S_R (1 bit), E_R (11 bits) y M_R (52 bits)
- El signo no cambia: $S_R = S_A$
- Exponente: hay que cambiar de exceso 127 a exceso 1023
 - $E_R = E_A + 896$
- Mantisa: hay que añadir $52 - 23 = 29$ ceros a la derecha
 - $M_R = M_A \parallel 00\dots 0$
- Detalle del trato correcto de los valores especiales:

	E_A	S_R	E_R	M_R
zero y subnormal:	00000000_2	S_A	00000000000_2	$M_A \parallel 00\dots 0$
$\pm\infty$ y NaN:	11111111_2	S_A	11111111111_2	$M_A \parallel 00\dots 0$
valores corrientes: (otros valores)		S_A	$E_A + 896$	$M_A \parallel 00\dots 0$

Conversión de simple a doble precisión (cvt.d.s)

■ El operador básico

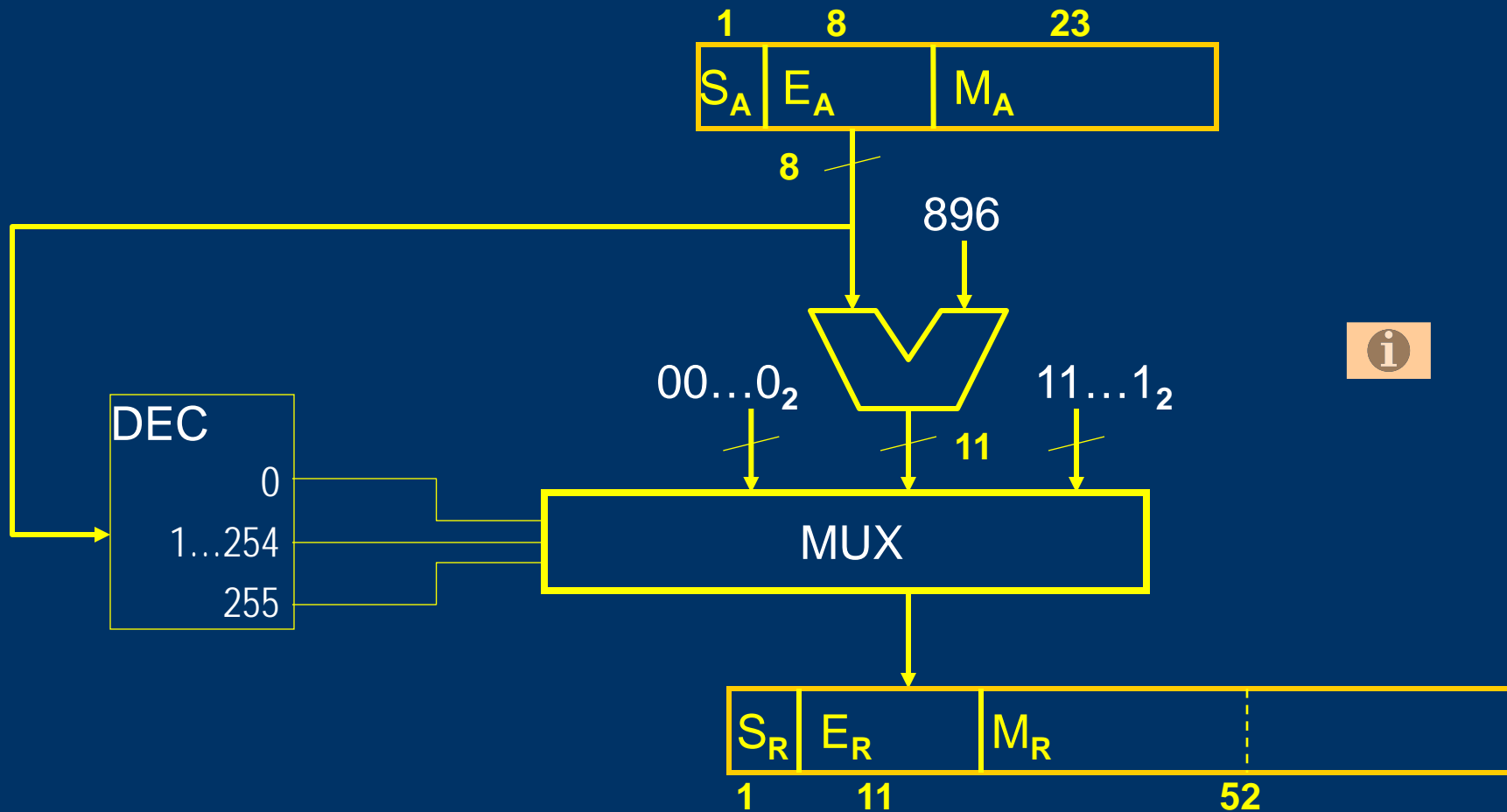
- No trata los valores especiales



Conversión de simple a doble precisión (cvt.d.s)

■ Tratamiento de los valores especiales

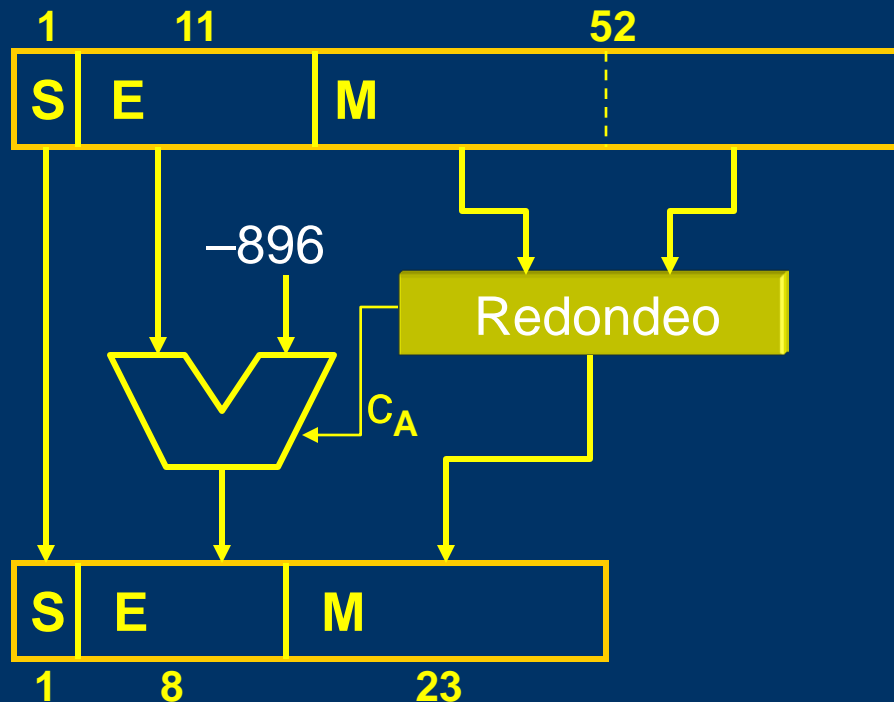
- Detalle del cálculo del exponente



Conversión de doble a simple precisión (cvt.s.d)

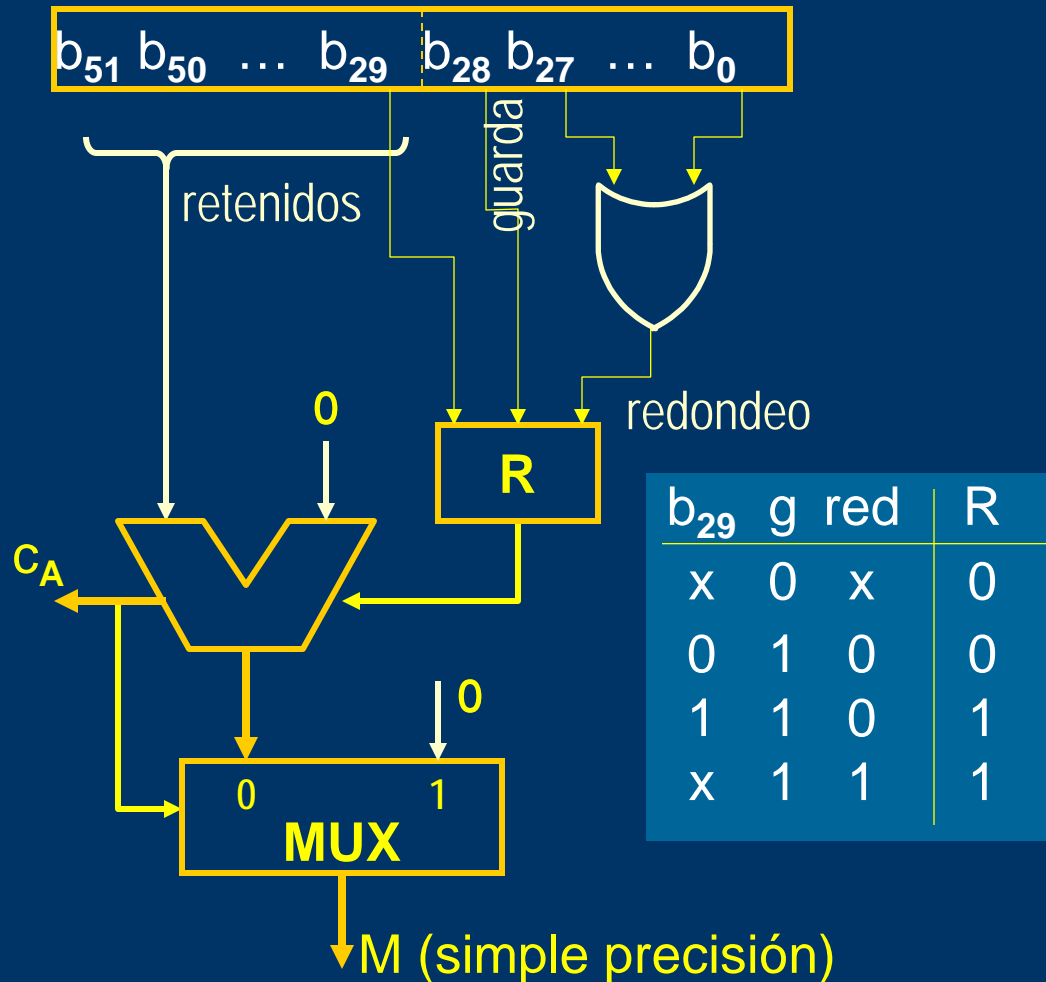
■ Estructura del operador

- El signo no cambia
- Exponente: hay que cambiar de 11 bits en exceso 1023 a 8 bits en exceso 127
 - puede darse desbordamiento
- Mantisa: hay que eliminar 29 bits por la derecha y redondear



El redondeo

■ Circuito para el redondeo al más próximo



Si $R=0$, truncar:

implícito

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

Si $R=1$, incrementar:

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

Si $c_A=1$, corregir

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

Conversión de entero a doble precisión (cvt.d.w)

■ Filosofía del operador

- Si es positivo, un entero W de 32 bits se puede escribir como $+0.W \times 2^{32}$
- Si es negativo, W se reescribe como $-0.(-W) \times 2^{32}$
- La mantisa W comienza por una serie de Z ceros ($0 \leq Z \leq 32$)
- Habrá que normalizar la mantisa desplazándola $Z+1$ posiciones hacia la izquierda (eliminando el bit entero) y restar $Z+1$ al exponente

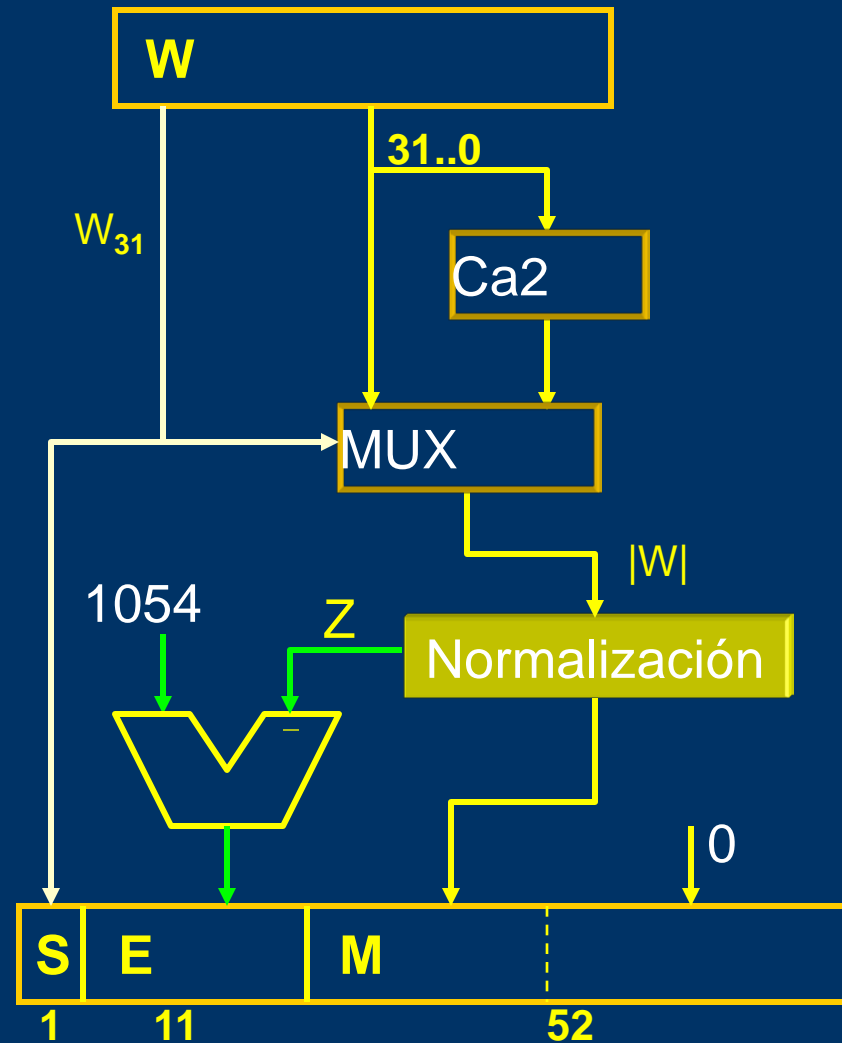
■ Especificación

- Entrada del operador: W (32 bits)
- Salida del operador: SR (1 bit), ER (11 bits) y MR (52 bits)
- $SR = \text{Signo}(W)$
- $MR = |W| \ll Z+1$ (desplazamiento)
 - Si $W < 0$, hay que hacer $W = -W$
- $ER = 1023 + 32 - Z - 1$

Conversión de entero a doble precisión (cvt.d.w)

■ Esquema

- La normalización ha de contar el número Z de bits a cero por la izquierda (hasta el primer 1)
- Ha de desplazar la mantisa Z posiciones hacia la izquierda
 - Hay que representar el exponente $31 - Z$
 - Sumando el exceso, tenemos $E = 1023 + 31 - Z$
- Hay que completar la mantisa con ceros



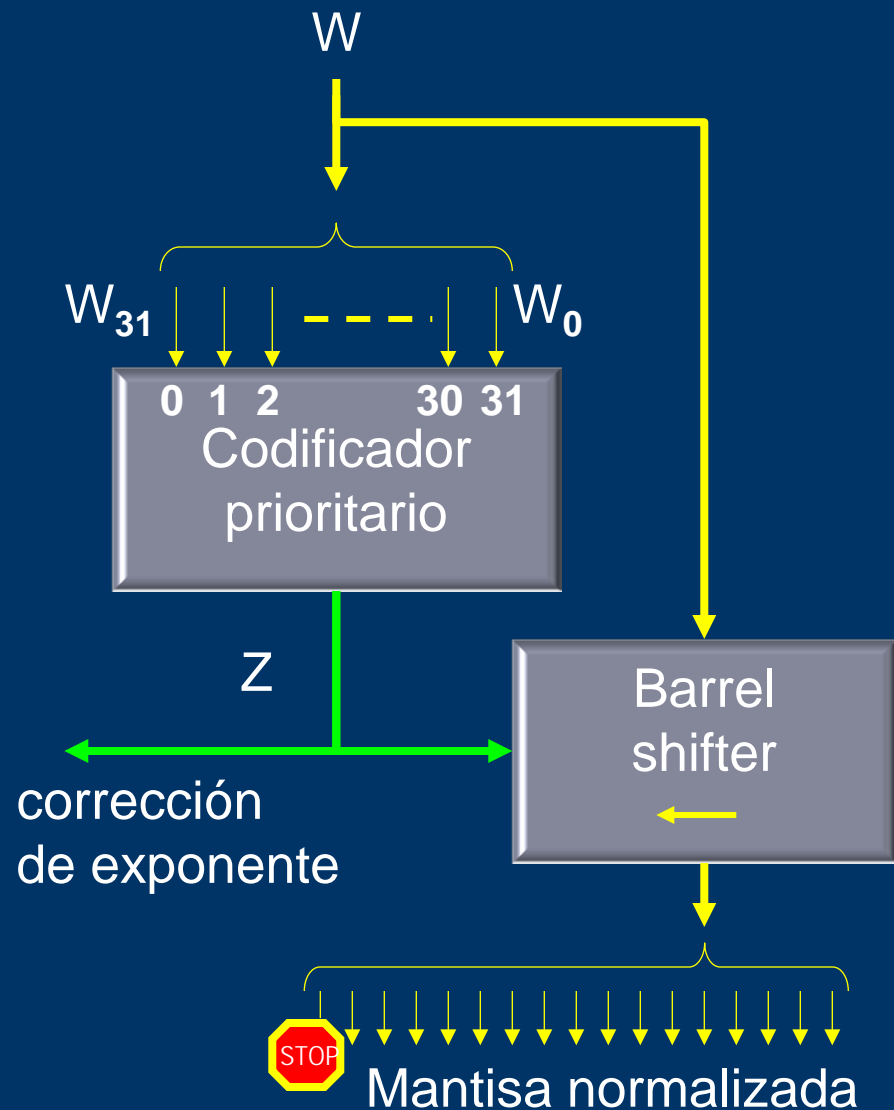
La normalización

■ 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

W_{31}	W_{30}	W_{29}	W_{28}	...	W_1	W_0	Z
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



La multiplicación (mul.s y mul.d)

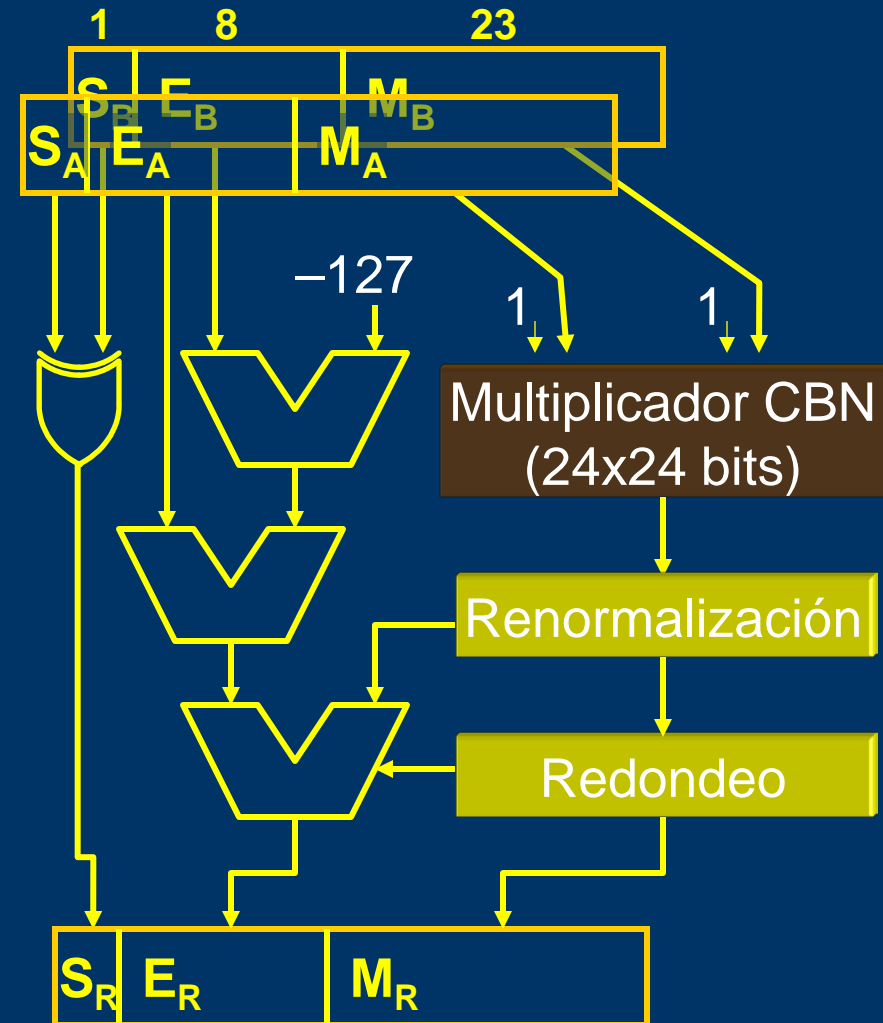
■ Especificación

- Dos versiones: simple precisión y doble precisión
- Dos entradas (A y B):
 - S_A (1 bit), E_A (8/11 bits) y M_A (23/52 bits)
 - S_B (1 bit), E_B (8/11 bits) y M_B (23/52 bits)
- Salida: S_R (1 bit), E_R (8/11 bits) y M_R (23/52 bits)
- Cálculo del signo: $S_R = S_A \text{ xor } S_B$
- Cálculo del exponente: hay que sumar y compensar el exceso
 - Simple precisión $E_R = E_A + E_B - 127$
 - Doble precisión $E_R = E_A + E_B - 1023$
- Cálculo de la mantisa:
 - Multiplicar $1.M_A \times 1.M_B$ (considerando el bit implícito)
 - El multiplicador depende de la precisión: 24x24 bits o 53x53 bits
 - Habrá que renormalizar (quitando el bit implícito) y redondear la mantisa resultante dejándola en 23/52 bits

La multiplicación (mul.s)

▪ Operador (SP)

- Dos sumadores para sumar los exponentes representados en exceso 127
- Multiplicador para las mantisas
 - Opera con el bit implícito añadido: 24x24 bits
- Habrá que renormalizar el resultado (ahora lo veremos) y quitar el bit implícito
 - Tal vez habrá que decrementar el exponente
- El resultado ocupará 47 bits y habrá que redondearlo a 23
 - Tal vez habrá que incrementar el exponente



La renormalización

■ Renormalización después de un producto

- Si la mantisa del producto comienza por 0:
 - Desplazar a la izquierda una posición
- Si comienza por 1:
- En cualquier caso:
 - Eliminar el bit entero implícito

$$\begin{array}{r} 1.000 \\ \times 1.000 \\ \hline 01.000000 \\ \downarrow \leftarrow \\ 0000000 \end{array}$$

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

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

