Práctica 6. Multiplicación números en formato IEEE 754

Tecnología y Organización de Computadores

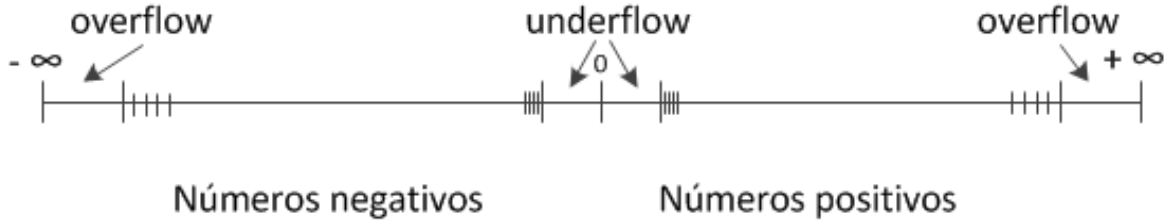
Grado en Ingeniería Informática

Universidad Complutense de Madrid

Índice

IEEE 754

- Estándar de representación de números en coma flotante

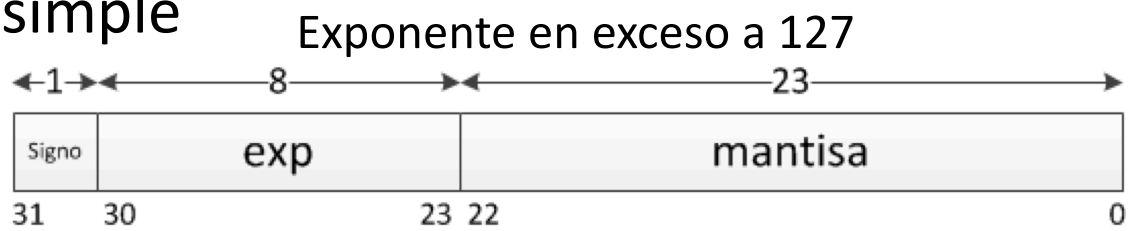


- Formatos:
  - Precisión simple: 32 bits
  - Precisión doble: 64 bits
  - Precisión extendida:
    - Simple: 48 bits
    - Doble: 80 bits

Formato simple

- Formatos IEEE 754
  - Precisión simple: 32 bits
  - Precisión doble 64 bits
  - Precisiones extendidas: 48 bits (simple) y 80 bits (doble).
- Precisión simple

0 positivo
1 negativo



- Clases

Clase	Exponente	Mantisa	Valor representado
Cero	0	0	0
Número de-normalizado	0	$\neq 0$	
Número normalizado	$[1, 254]$	any	
Infinito	255	0	∞
Not-a-number	255	$\neq 0$	NaN

Precisión simple

- Número normalizado

$$v = s \cdot m \cdot 2^{e}$$

donde:

s= +1 cuando signo = 0 (número positivo)

e= exp-127 (exponente en exceso a 127)

m= 1.mantisa

- Ejemplo

exp mantisa

signo → 00011101001101011101010111101011

$$v = +1 \cdot 1{,}42059075832366943359375 \cdot 2^{(58-127)}$$

$$= 2{,}406574353730238423419538770038 3 \cdot 10^{-21}$$

Precisión simple

- Número de-normalizado: aumentar densidad números cercanos al 0 $v = s \cdot m \cdot 2^{e}$
  - e: siempre es -126
  - m= 0.mantisa
- Ejemplos

exp mantisa

signo → 10000000001101011101010111101011

$$v = -1 \cdot 0{,}42059075832366943359375 \cdot 2^{(-126)}$$

$$= -4{,}944020604175354169938045927605 6 \cdot 10^{-39}$$

exp mantisa

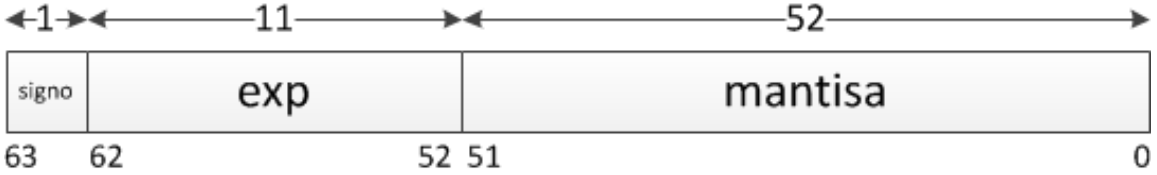
signo → 10000000000000000000000000000001

$$v = -1 \cdot 2^{-23} \cdot 2^{(-126)}$$

$$= -1{,}401298464324817070923729583289 9 \cdot 10^{-45}$$

Precisión doble

- Formato



- Clases de valores

Clase	e	m
Cero	0	0
Número desnormalizado	0	≠ 0
Número normalizado	[1,1022]	cualquiera
Infinitos	1023	0
NaN	1023	≠ 0

Precisión doble

- Número normalizado

$$v = s \cdot m \cdot 2^e$$

donde:

  - s= +1 cuando signo =0 (número positivo) y s = -1 cuando signo = 1
  - e= exp-1023 (exponente en exceso a 1023)
  - m= 1.mantisa

- Número de-normalizado: aumentar densidad números cercanos al 0
  - e: siempre es -1022
  - m= 0.mantisa

Excepciones

- Operación inválida
- Resultado inexacto
- Overflow/Underflow
- División por cero

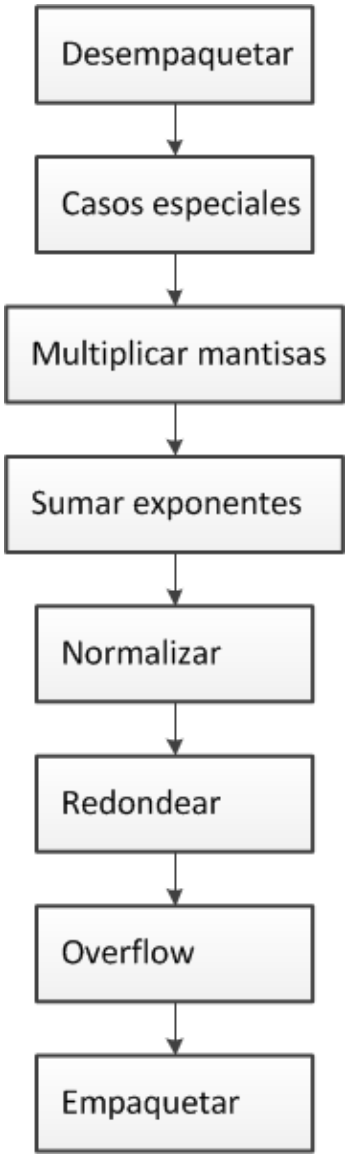
Redondeo

- Siempre que un número de más precisión tiene que convertirse a un número de menor precisión hay que hacer “redondeo”.
- 4 modos de redondeo
  - Redondeo al más cercano (opción por defecto)
    - “The representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero shall be delivered.”
  - Redondeo a más infinito (por exceso)
    - “(…) the result shall be the format's value(possibly +∞) closest to and no less than the infinitely precise result”
  - Redondeo a menos infinito (por defecto)
    - “(…) the result shall be the format's value (possibly - ∞) closest to and no greater than the infinitely precise result”
  - Redondeo a cero (“truncation” o “chopping”)
    - “(…) the result shall be the format's value closest to and no greater in magnitude than the infinitely precise result”

Redondeo (I)

- IEEE 754 exige que el resultado de las operaciones debe ser el mismo que si se hubieran hecho con precisión infinita y después redondeo
  - "(…) every operation specified in Section 5 shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to one of the modes in this section."
- ¿Cómo? Añadiendo tres bits a la derecha: G, R y S
  - G (Guard bit) al hacer un desplazamiento de un bit a derecha este bit evita perdida de precisión.
  - R (Round bit) al hacer un desplazamiento de un bit a izq. (durante normalización) este bit indica hacia dónde hacer el redondeo.
  - S (sticky bit) indica si los bits descartados tienen un valor $\frac{ulp}{2}$ siendo ulp (unit of least precision) la distancia entre dos números consecutivos representables con el mismo exponente.

Multiplicación



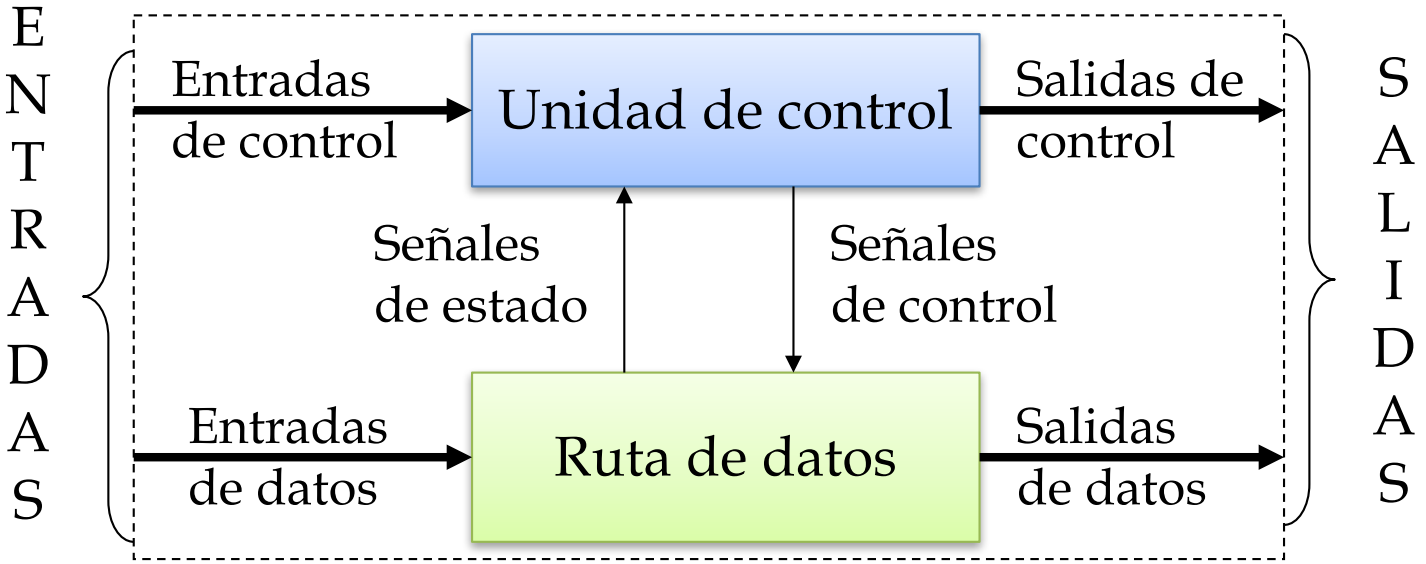
Ejemplo:

$$\begin{array}{r} 4.555\cdot10^{4} \\ \times\ 2.996\cdot10^{7} \\ \hline 13.646780\cdot10^{11} \end{array} \longrightarrow 1.3646780\cdot10^{12} \longrightarrow 1.365\cdot10^{12}$$

normalizar redondeo

¿Qué hacer?

- Diseñar un sistema algorítmico que implemente la multiplicación "simplificada" de dos números en formato simple IEEE 754
  - Números normalizados
  - Operandos de entrada (op1 y op2) nunca un NaN
- El diseño se implementará como un sistema algorítmico



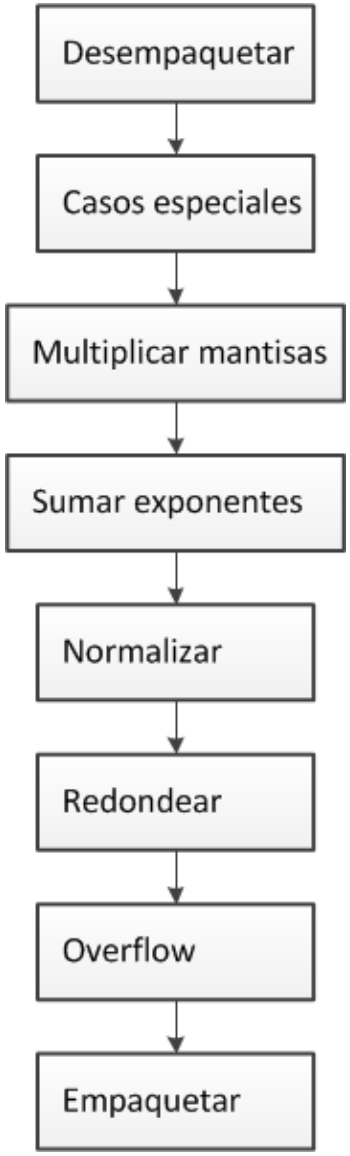
Puertos

- Definición entidad

```
entity mult_ieee754 is
    port  (
    clk : in  std_logic;
    rst : in  std_logic;
    -- operandos entrada/salida
    op1 : in  std_logic_vector(31 downto 0);
    op2 : in  std_logic_vector(31 downto 0);
    res : out std_logic_vector(31 downto 0);
    -- control
    ini : in  std_logic;
    fin : out std_logic);
end entity mult_ieee754;
```

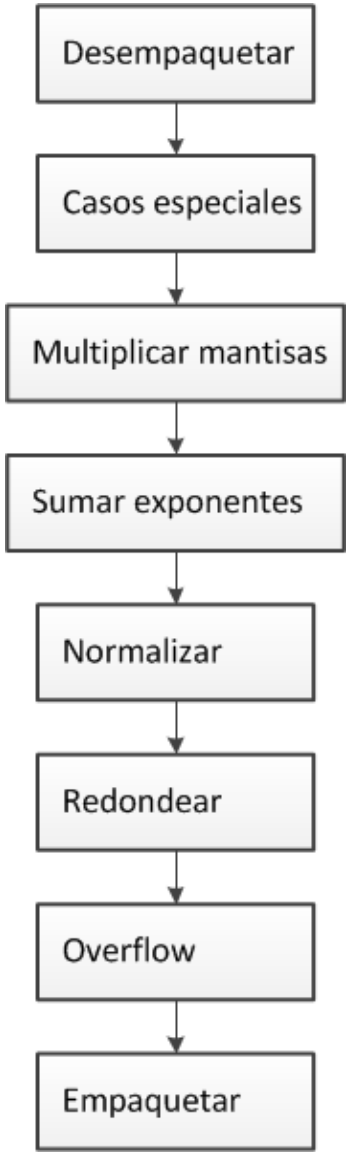
- Funcionamiento síncrono. Todos los registros activos por flanco de subida.
- La señal de reloj será clk.
- La señal de reset, rst, asíncrona activa a nivel alto.
- Datos de entrada (op1 y op2) y salida (res) registrados

Algoritmo



1. Extraer signo, exponente y mantisa de cada operando

Algoritmo



1. Extraer signo, exponente y mantisa de cada operando

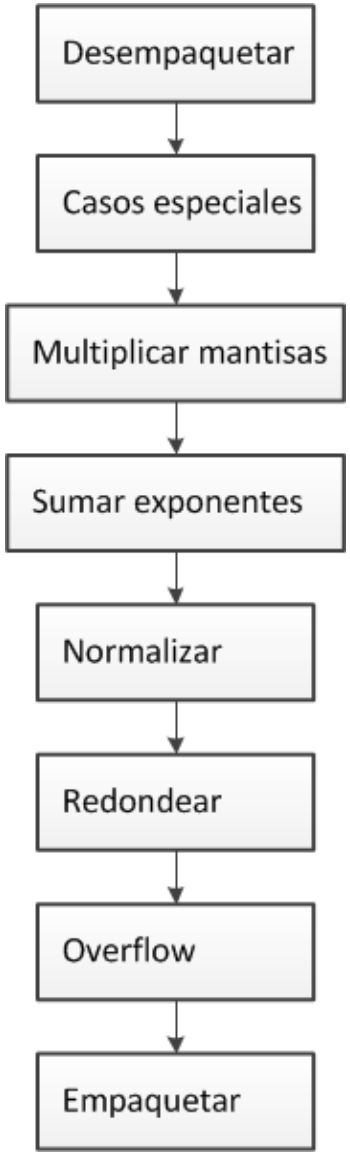
2. Gestión operandos entrada igual a 0 o infinito

op1	op2	res
0	any	0
∞	0	NaN
∞	any	∞
∞	∞	∞

En la tabla anterior:

- Los valores de las columnas op1 y op2 son intercambiables
- Hay que tener en cuenta los signos

Algoritmo

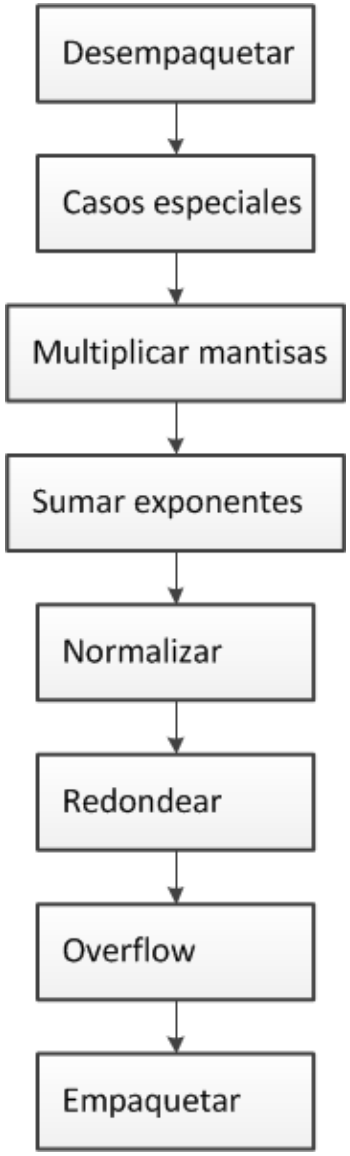


1. Extraer signo, exponente y mantisa de cada operando

2. Gestión operandos entrada igual a 0 o infinito

3. Las mantisas se multiplican como numero en binario puro
   - Cada mantisa es un número de 24 bits: a los 23 bits de la mantisa hay que concatenar a su izquierda un 1
   - El resultado es un numero de 48 bits
   - El punto decimal estará entre los bits 45 y 46

Algoritmo



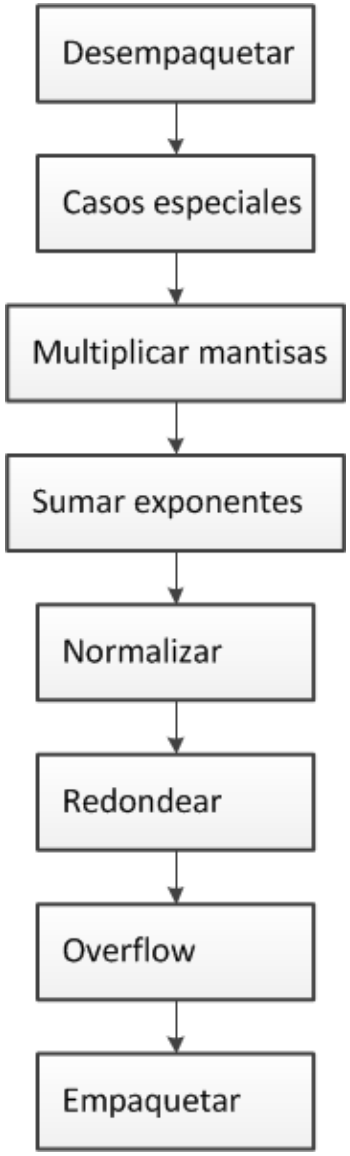
1. Extraer signo, exponente y mantisa de cada operando

2. Gestión operandos entrada igual a 0 o infinito

3. Las mantisas se multiplican como numero en binario puro

4. Los exponentes se suman como números en binario puro
   - El resultado es un numero de 9 bits al que hay que restarle un exceso a 127

Algoritmo



1. Extraer signo, exponente y mantisa de cada operando

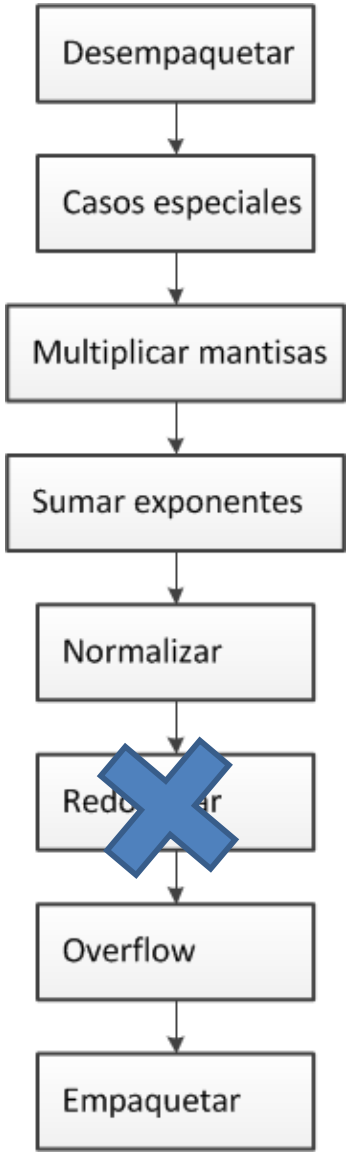
2. Gestión operandos entrada igual a 0 o infinito

3. Las mantisas se multiplican como numero en binario puro

4. Los exponentes se suman como números en binario puro

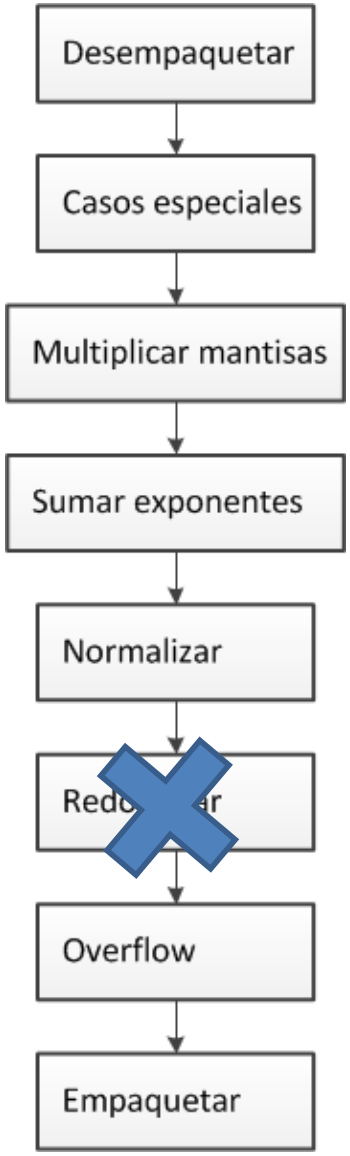
5. Se desplaza el punto decimal de la mantisa
   - Dicho punto debe quedar a la derecha del primer uno más significativo.
   - Por cada desplazamiento a izquierda (solo hará falta un desplazamiento) se debe incrementar el exponente en +1

Algoritmo



1. Extraer signo, exponente y mantisa de cada operando
2. Gestión operandos entrada igual a 0 o infinito
3. Las mantisas se multiplican como numero en binario puro
4. Los exponentes se suman como números en binario puro
5. Se desplaza el punto decimal de la mantisa
6. Nuestro multiplicador simplificado no redondea

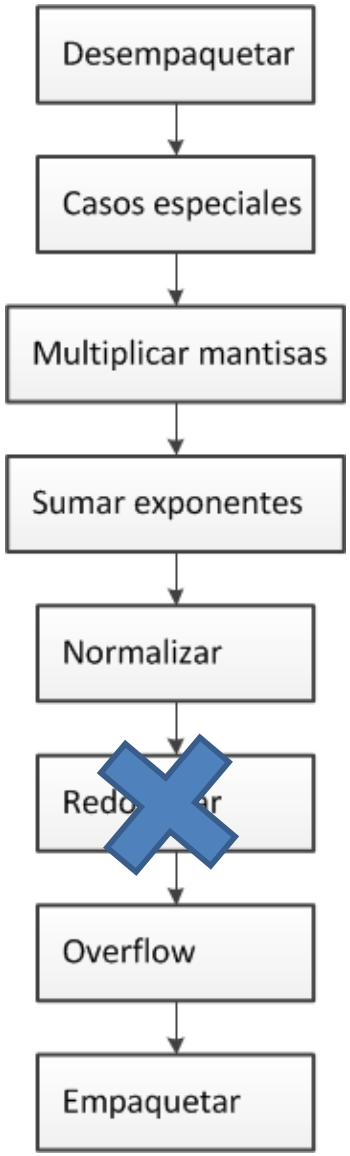
Algoritmo



1. Extraer signo, exponente y mantisa de cada operando
2. Gestión operandos entrada igual a 0 o infinito
3. Las mantisas se multiplican como numero en binario puro
4. Los exponentes se suman como números en binario puro
5. Se desplaza el punto decimal de la mantisa
6. Nuestro multiplicador simplificado no redondea
7. Si el exponente es 255 o mayor entonces overflow
   - El campo mantisa se debe poner a 0 para representar el infinito

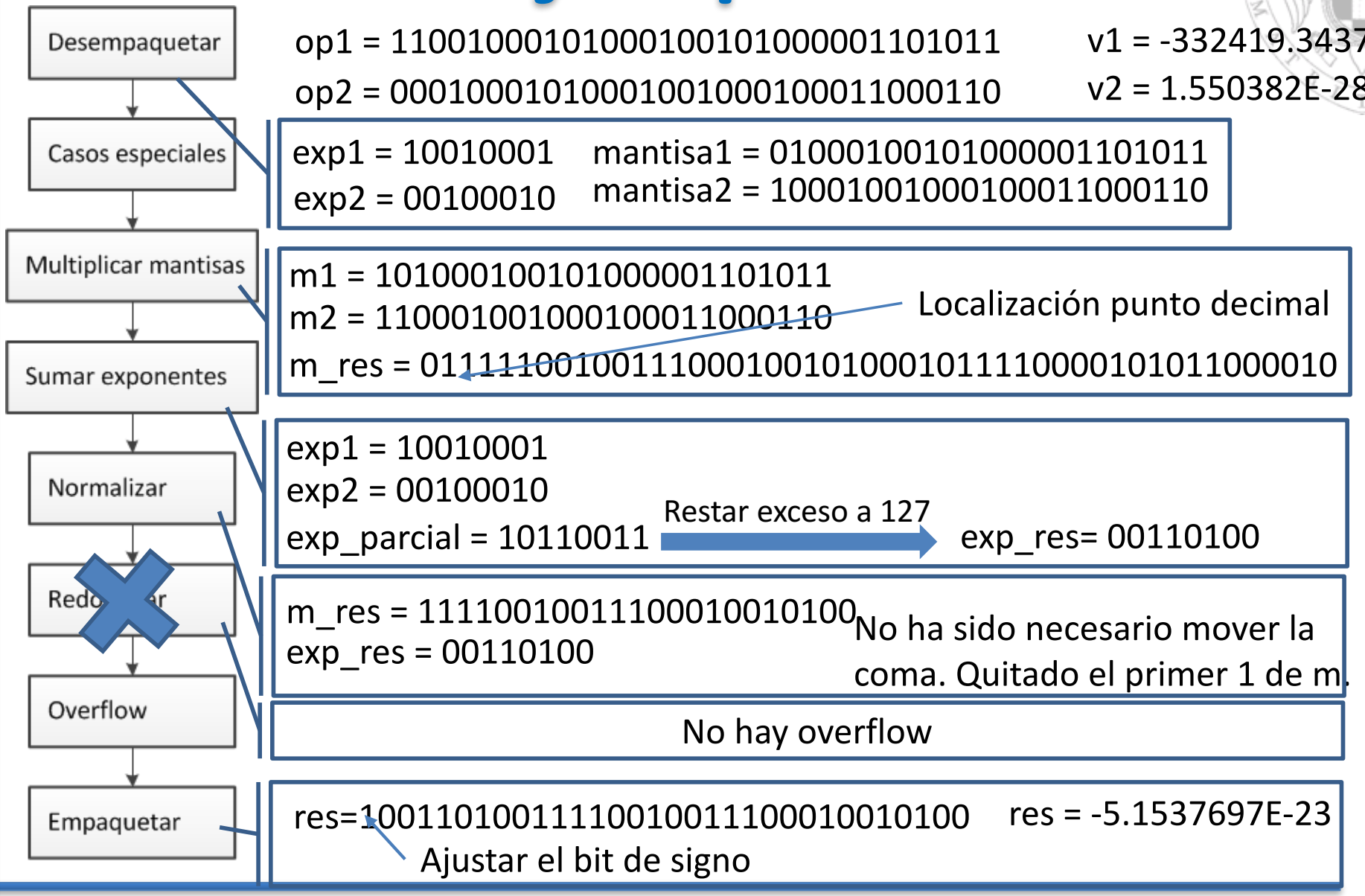
   Si el exponente es negativo entonces underflow
   - El campo mantisa y exponente se debe poner a 0

Algoritmo

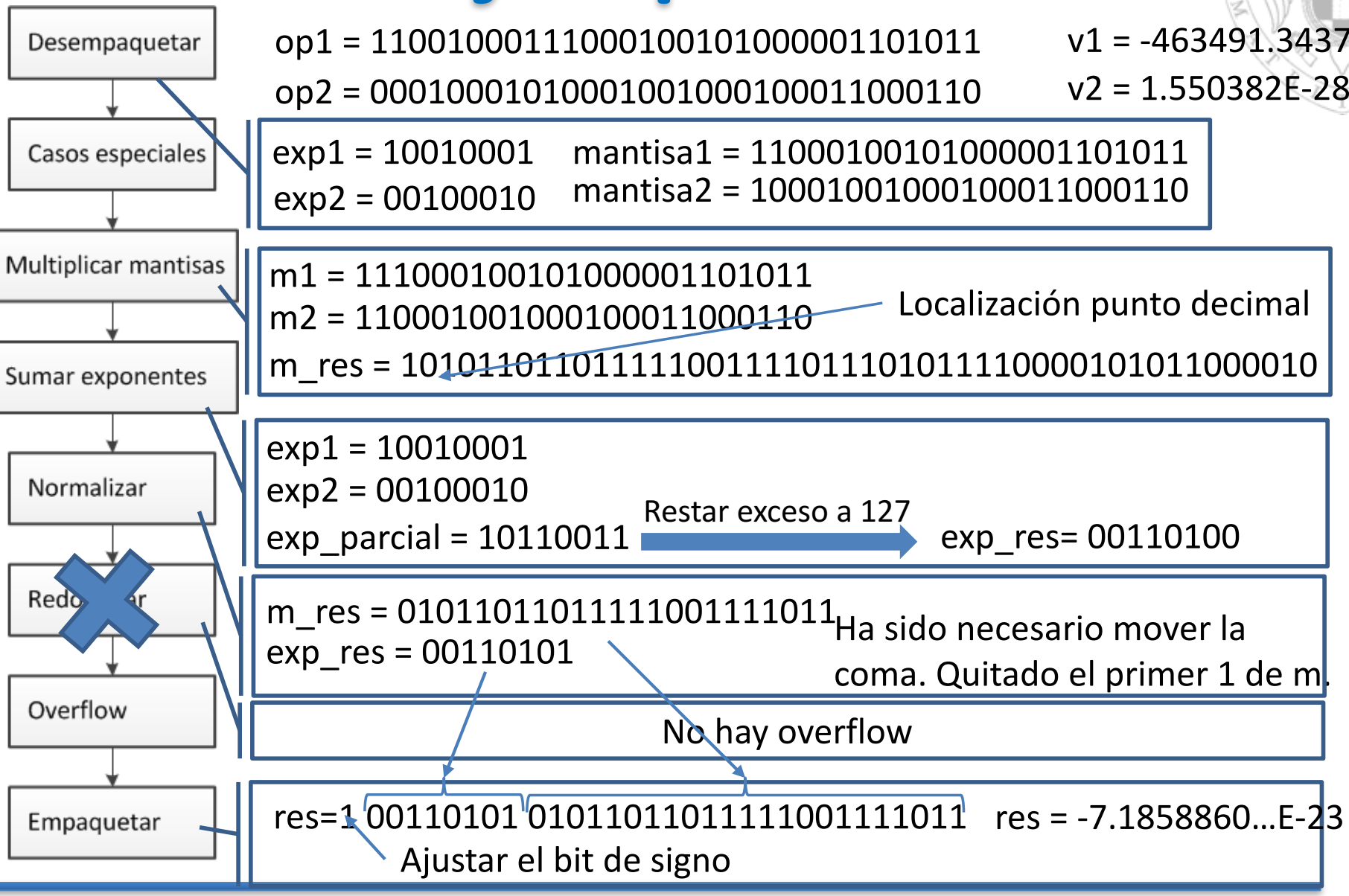


1. Extraer signo, exponente y mantisa de cada operando
2. Gestión operandos entrada igual a 0 o infinito
3. Las mantisas se multiplican como numero en binario puro
4. Los exponentes se suman como números en binario puro
5. Se desplaza el punto decimal de la mantisa
6. Nuestro multiplicador simplificado no redondea
7. Si el exponente es > 255 o <0
8. Se concatena el bit de signo, 8 bits del exponente y los 23 bits más significativos de la mantisa excluido el primer 1

Ejemplo


Desempaquetar
op1 = 11001000101000100101000001101011
v1 = -332419.34375
op2 = 00010001010001001000100011000110
v2 = 1.550382E-28
Casos especiales
exp1 = 10010001
mantisa1 = 01000100101000001101011
exp2 = 00100010
mantisa2 = 10001001000100011000110
Multiplicar mantisas
m1 = 10100010010100000110101 1
m2 = 11000100100010001100011 0
Localización punto decimal
m_res = 01111100100111000100101000101111000010101100001 0
Sumar exponentes
exp1 = 10010001
exp2 = 00100010
exp_parcial = 10110011
Restar exceso a 127
exp_res= 00110100
Normalizar
Redo r
m_res = 11110010011100010010100
exp_res = 00110100
No ha sido necesario mover la coma. Quitado el primer 1 de m.
Overflow
No hay overflow
Empaquetar
res=10011010011110010011100010010100
res = -5.1537697E-23
Ajustar el bit de signo

Ejemplo II


Desempaquetar
Casos especiales
Multiplicar mantisas
Sumar exponentes
Normalizar
Redo ar
Overflow
Empaquetar
op1 = 11001000111000100101000001101011
v1 = -463491.34375
op2 = 00010001010001001000100011000110
v2 = 1.550382E-28
exp1 = 10010001
mantisa1 = 11000100101000001101011
exp2 = 00100010
mantisa2 = 10001001000100011000110
m1 = 11000100101000001101011
m2 = 11000100100010001100011O
Localización punto decimal
m_res = 10101101101111100111011101011110000101011000010
exp1 = 10010001
exp2 = 00100010
exp_parcial = 10110011
Restar exceso a 127
exp_res= 00110100
m_res = 01011011011111001111011
exp_res = 00110101
Ha sido necesario mover la coma. Quitado el primer 1 de m.
No hay overflow
res=1 00110101 01011011011111001111011
res = -7.1858860…E-23
Ajustar el bit de signo