FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

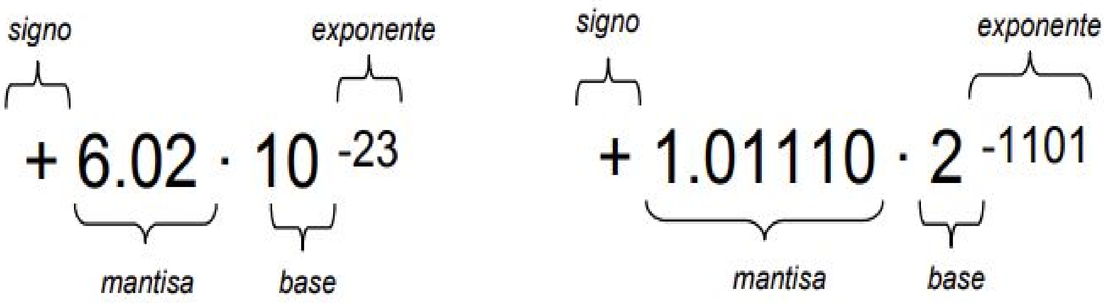
Circuitos Logicos Programables
Carrera de especialización sistemas embebidos
ING.Jose Mauricio Lara Tapia

Multiplicador de punto flotante

1. **INTRODUCCIÓN.**

Bien hay que saber que se denomina números de punto flotante a las representaciones internas de procesador que modelan a los números reales. En forma externa, se representan números con punto decimal tal como 3.1415926 o en notación científica 9.512x10-5, con un solo dígito a la izquierda del punto decimal;
es decir, 9.512x10-4.

La representación en punto flotante está basada en el formato estándar de la IEEE 754 y el el manejo de la notación científica:
El punto decimal no se halla en una posición fija dentro de la secuencia de bits, sino que su posición indica como una potencia de la base.



En todo número de punto flotante se distinguen tres componentes:

- **Signo:** Indica el signo del número (0 = positivo 1= negativo
- **Mantisa:** Contiene la magnitud del número (en binario puro)
- **Exponente:** Contiene el valor de la potencia de la base (sesgado)
- La base queda implícita y es común a todos los números, la más usada es 2

El valor de la secuencia de bits se puede representar en la siguiente ecuación:

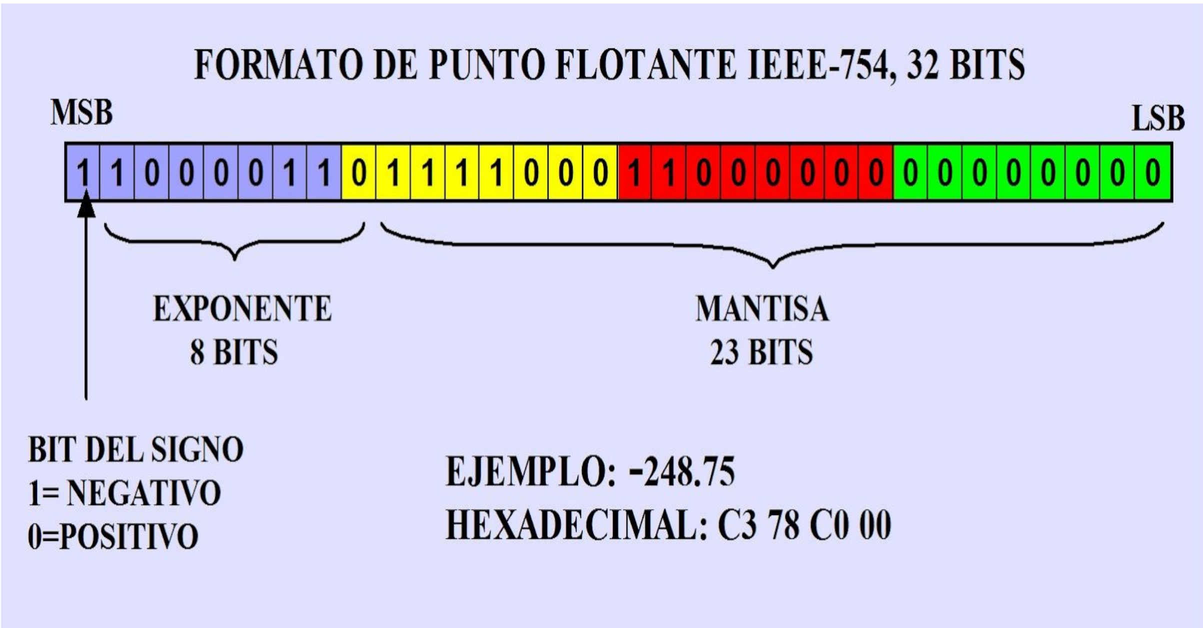
$$(-1)^{s1}\ (Pn\,Mn)\ *\ 2^{e1-127}$$

- siendo $(-1)^{s1}$ el bit de signo.
- siendo $(Pn\,Mn)$ Mantisa normalizada.
- siendo 2^{e1-127} el valor del exponente.

dado que un número puede tener varias representaciones ($0.110 * 2^5 = 110 * 2^2 = 0.0110 * 2^6$) los números suelen ser normalizados:

- Un número está normalizado si tiene la forma $1.xxx... * 2^{xx...}$ o $(0.1xx... * 2^{xx...})$
- Dado que los números normalizados en base 2 tienen siempre un 1 a la izquierda, este suele quedar implícito (pero debe ser tenido en cuenta al calcular el valor de la secuencia)

Sea el siguiente formato de punto flotante de 32 bits (base 2, normalizado)



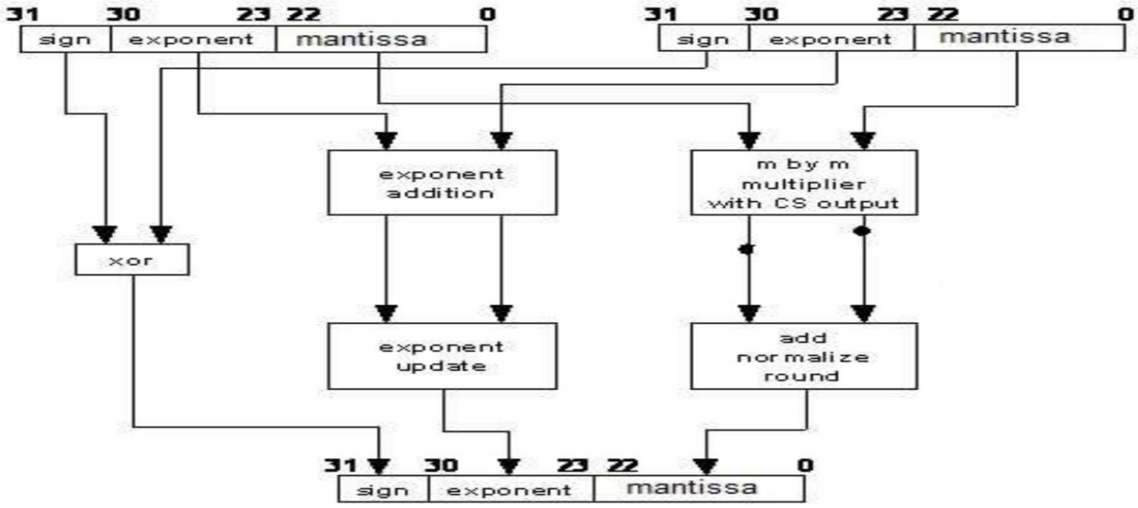
Multiplicación de punto flotante

Considerando el formato estándar de la IEEE 754 y la notación científica se pueden realizar operaciones lógicas como la multiplicación.

Sean X e Y números representados en punto flotante con valor :
- X= $(-1)^{s1}\ (1.Mant1) * 2^{e1}$
- Y= $(-1)^{s2}\ (1.Mant2) * 2^{e2}$

El producto de estos dos numeros sera otro numero Z con valor:
- Z= $(-1)^{s1+s2}\ (1.Mant1 * 1.Mant2) * 2^{e1+e2-127}$

considerando la resultante Z el desarrollo del presente trabajo responde al siguiente diagrama general



Para lidiar con el bit de signo

Se tiene una una compuerta XOR dado que el comportamiento de producto de signos es igual a la tabla de verdad de la misma

PRODUCTO DE SIGNOS		
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1

TABLA DE VERDAD XOR		
0	0	0
0	1	1
1	0	1
1	1	1

Para lidiar con la parte de la mantisa

La mantisa de Z se calcula mediante multiplicación binaria de las mantisas de X e Y, además de las mantisas la operación se tiene que operar con su valor implícito como 1.xxxxx y la operacion de ejemplo quedaría de la siguiente manera:

```
                        1.00000000000000000000000
                      x 1.10000000000000000000000
-----------------------------------------------------
                        000000000000000000000000
                       00000..........
              00000........
         10000....
       10000..........
-----------------------------------------------------
   0 1.100000000000000000000000000000000000000000000    <= Producto resultante 48 bits
```

---

la multiplicación binaria de ambas mantisas en consecuencia arroja 48 bits, 2 bits implícitos y 46 bits de mantisa. De toda la trama de bits se considera a la mantisa resultante desde el bit 46 al 24.

Para lidiar con la parte del exponente

El exponente de Z se calcula mediante una suma binaria de los exponentes de X e Y y la resultante de la suma se le hace una resta binaria de 127.

Existen casos donde llega a registrar te otra variable en la obtención del exponente esto se debe por la multiplicación de la mantisa que da como resultante 48 bits, el caso viene cuando el bit más significativo que sería el 48 tiene a ser un 1 lógico, lo cual en consecuencia existiría un incremento en la exponente actualizando el resultado.

if (bit_48==1){

Desplazamiento = 1

$$exponente\ resultante\ = exponente\ X + exponente\ Y\ + Desplazamiento$$

}

esle {

Desplazamiento=0

$$exponente\ resultante\ = exponente\ X + exponente\ Y\ + Desplazamiento$$

Implementación VHDL

- **Operación con mantisa en código VHDL:**

En un número de coma flotante de 32 bits, se dedican 23 bits para determinar la mantisa. La mantisa del producto se calcula mediante la multiplicación binaria de 23 bits de los operandos.
Esto se logra implementando el siguiente código.

```
    --adicion de 1 al bit mas significativo
    --para realizar la multiplicacion
    x_mantissa(23) := '1';
    y_mantissa(23) := '1';

    --multiplicacion y sumatoria de las mantisas
    for j in 0 to 23 loop
       multiplicacion_temporal := (others => '0');
       if (y_mantissa(j)='1')then
       --se asigna el valor de X_mantisa a multiplicacion temporal
       -- y no se no afecta al valor original
             multiplicacion_temporal(23+j downto j) := x_mantissa;
       end if;
       -- el valor multiplicado se almacena en un valor temporal
       --para no afectar de manera directa a la resultante
       multiply_store_temp := multiply_store;

       --sumador completo de las operaciones de multiplicacion en 48 bits
       for i in 0 to 47 loop
             multiply_store(i) := multiply_store_temp(i) xor multiplicacion_temporal(i) xor carry1;
             carry1  := ( multiply_store_temp(i) and multiplicacion_temporal(i) ) or ( multiply_store_temp(i) and carry1 ) or ( multiplicacion_temporal(i) and carry1 );
       end loop;
end loop;
```

```
if (multiply_store(47)='1')then

            z_mantissa := multiply_store(46 downto 24);
            carry1 := '0';--reasignacion a cero
            carry2 := '0';--reasignacion a cero
            multiply_rounder(0) := multiply_store(23);

            --operacion de redondeo de la mantisa
            for i in 0 to 22 loop
                  carry1 := z_mantissa(i);
                  z_mantissa(i) := carry1 xor multiply_rounder(i) xor carry2;
                  carry2 := (multiply_rounder(i) and carry1) or (multiply_rounder(i) and carry2) or (carry1 and carry2);

            end loop;
      else
            z_mantissa := multiply_store(45 downto 23);

            carry1 := '0';--reasignacion a cero
            carry2 := '0';--reasignacion a cero

            multiply_rounder(0) := multiply_store(22);


            for i in 0to 22 loop
                  carry1 := z_mantissa(I) ;
                  z_mantissa(I) :=  carry1 xor multiply_rounder(I) xor carry2 ;
                  carry2 := (multiply_rounder(I) and carry1 ) or ( multiply_rounder(I) and carry2 ) or ( carry1 and carry2 ) ;
             end loop;
      end if;
```

- **operación de exponentes con código VHDL**

Se asignan 8 bits para representar el exponente del número de coma flotante.
El exponente del producto se calcula sumando exponentes de los dos operandos y luego restando 127 de él.
Esto se logra mediante el siguiente bloque de código.

```
--agregado simple del x_exponente e y_exponente
for i in 0 to 8 loop
        suma_exponente(i) := x_exponente(i) xor y_exponente(i) xor carry1;
        carry1 := (x_exponente(i) and y_exponente(i)) or (x_exponente(i) and carry1) or (y_exponente(i) and carry1);
end loop;

carry1 := '0';--reasignacion a cero
carry2 := '0';--reasignacion a cero

if (multiply_store(47)='1')then
        --asignando valor a la mantisa de Z
        --incremento de exponente por desplasamiento de coma
        for i in 0 to 8 loop
                carry1 := suma_exponente(i) ;
                suma_exponente(i) := carry1 xor temporal2(i) xor carry2 ;
                carry2 := (temporal2(i) and carry1) or (carry1 and carry2) or (temporal2(i) and carry2 ) ;
        end loop;

--resta de z_exponente -127
for i in 0 to 8 loop
        carry1 := suma_exponente(i) ;
        suma_exponente(i) :=  carry1 xor temporal1(I) xor carry2 ;
        carry2 := ( carry2 and Not carry1 ) or ( temporal1(i) and Not carry1 ) or (temporal1(i) and carry2) ;
end loop;
```

- **Condición de desbordamiento**

Cuando la magnitud del número es demasiado grande para representar
La condición reconocida por el octavo bit de suma_exponente es 1 y el séptimo bit es 0.
En tal caso, la salida es infinito positivo o infinito negativo
Valor de magnitud más grande 8388607 * 2 + 127

- **Representación negativa de subdesbordamiento**

Cuando la magnitud del número es demasiado pequeña para representar
La condición reconocida por el octavo bit y el séptimo bit de suma_exponente es 1.
En tal caso, la salida es cero positivo o cero negativo (prácticamente ambos valores resultan ser iguales).
Valor de magnitud más pequeño 1 * 2-128

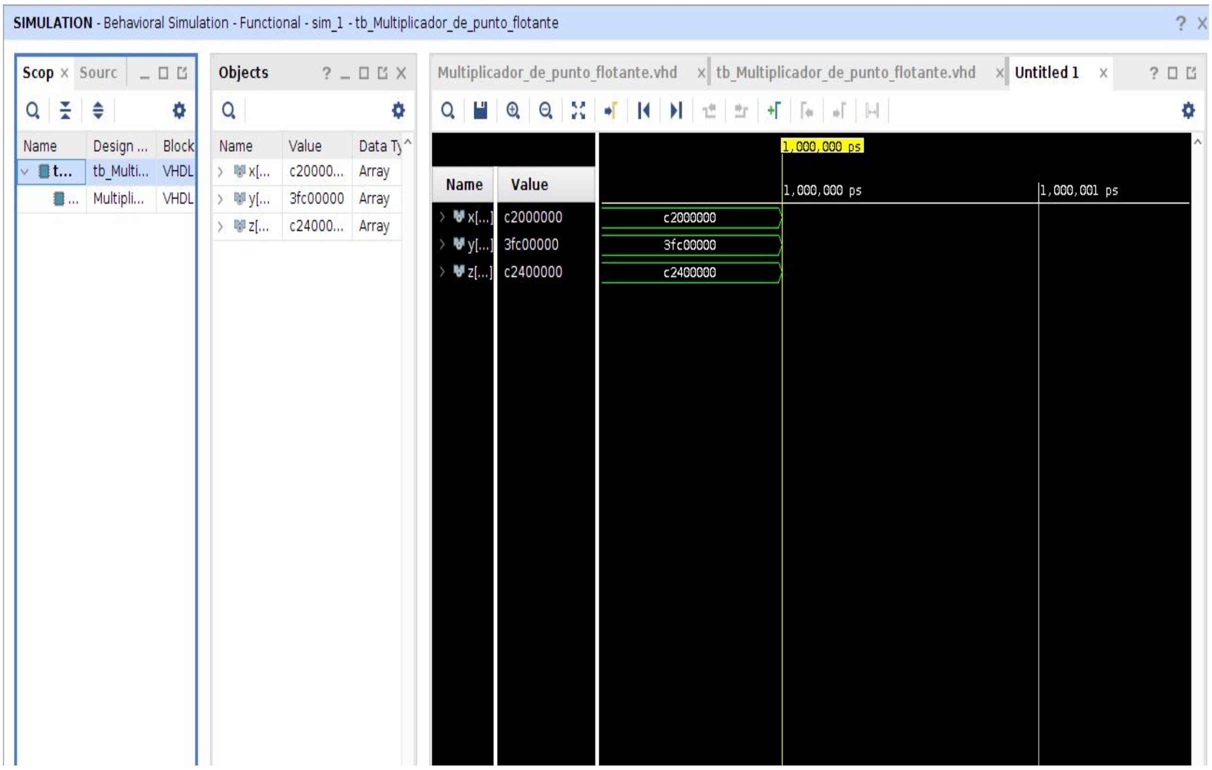
```
if (suma_exponente (8) = '1') then
        --desbordamiento
    if(suma_exponente(7) = '0')then

            z_bit_signo := x_bit_signo xor y_bit_signo;
            z_exponente := "11111111";
            z_mantissa :=(others => '0');
    --representación negativa de subdesbordamiento
    else

            z_bit_signo := '0';
            z_exponente := (others => '0');
            z_mantissa := (others => '0');

    end if;
```

- **Simulación**

DATOS:

X=0xC2000000
Y=0x3FC00000

PRODUCTO RESULTANTE:
Z=0xC240000



- **Esquemático**