TEMA 3 “Unidad de Cálculo”

1. Operaciones de Cálculo habituales

Las unidades de cálculo suelen implementar:

- Comparación (Igual, mayor que, menor que).
- Suma.
- Resta.
- Multiplicación.
- División.
- Otras operaciones: Seno, Coseno, Tangentes, Logaritmo, Exponenciación.

1.1. Tipos de operandos

La clasificación de los operandos:

- **Por tamaño:** Byte, Word, Doble word, Cuádruple word.
- **Por tipo:**
  - Entero sin signo.
  - Entero con signo: Signo-Magnitud, Complemento a 2 y Exceso M.
  - Números reales: Punto fijo y Punto flotante.

1.2. Implementación de las Operaciones

Elementos necesarios:

- Hardware:
  - Registros (de almacenamiento, de desplazamiento, etc.).
  - Circuitos combinacionales aritméticos (sumadores, restadores, etc.).
  - Elementos de interconexión (buses, líneas, etc.).
  - Lógica adicional.
- Software (Algoritmo aritmético): Diagrama de flujo clásico => Secuencia de microoperaciones.

Interdependencia Hardware/Software.

2. Comparación

Importancia: Mecanismo para la toma de decisiones en otras operaciones.
La comparación se realiza a nivel de magnitud. Una forma de realizarla es mediante un circuito específico. **Igualdad:**

- Dos bits (i-ésimos) son iguales si $X_i = 1$:

$$X_i = A_i \cdot B_i + \overline{A_i} \cdot \overline{B_i}, i = 0 \ldots n-1$$

A	B	A·B	A‘·B’	X
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

- Dos números son iguales si:

$$\prod_{i=0}^{n-1} X_i = 1$$

2.1. Comparación de la magnitud

Mayor que:

- Un número “a” es mayor que otro “b” si, comenzando desde el dígito más significativo; se cumple:

- X_j = 1 (para $j = n\text{-}1 \ldots i+1$) y a_i = 1 y b_i = 0.
- Ej. A, B de 4 bits:

$$(A > B) = A_3 \cdot \overline{B_3} + X_3 \cdot A_2 \cdot \overline{B_2} + X_3 \cdot X_2 \cdot A_1 \cdot \overline{B_1} + X_3 \cdot X_2 \cdot X_1 \cdot A_0 \cdot \overline{B_0}$$

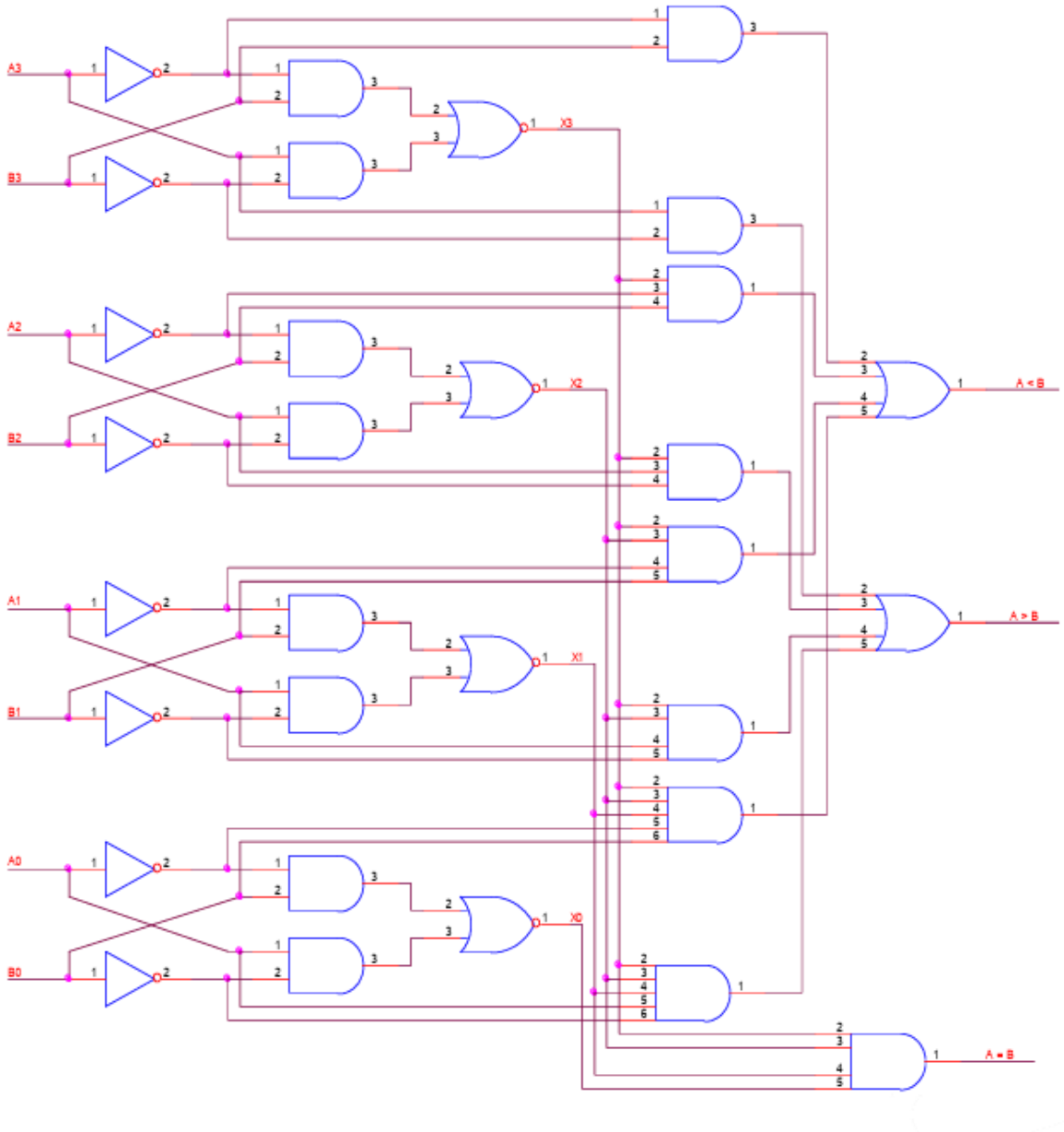
Menor que:

- Un número “a” es menor que otro “b” si, comenzando desde el dígito más significativo; se cumple:
  - X_j = 1 (para $j = n\text{-}1 \ldots i+1$) y a_i = 0 y b_i = 1.
  - Ej. A, B de 4 bits:

$$(A < B) = \overline{A_3} \cdot B_3 + X_3 \cdot \overline{A_2} \cdot B_2 + X_3 \cdot X_2 \cdot \overline{A_1} \cdot B_1 + X_3 \cdot X_2 \cdot X_1 \cdot \overline{A_0} \cdot B_0$$

2.2. Circuito Combinacional de Comparación

La ventaja principal de realizar la comparación mediante un circuito específico es que es la opción más rápida. La desventaja es que es la opción más cara. De ahí que se suelen buscar otras alternativas.



2.3. Comparación mediante Resta

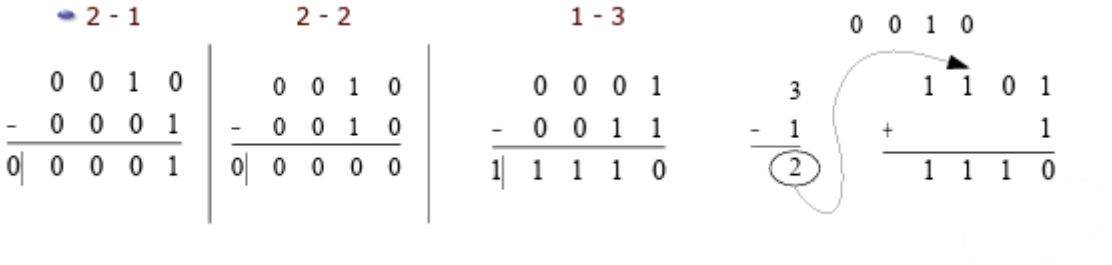
Una forma de saber si dos números son iguales es restándolos.

- Si son iguales, el resultado será 0.
- Si el primero es mayor que el segundo, el resultado será positivo.
- Si el primero es menor que el segundo, el resultado será negativo.

Utilizando un restador completo:

- A > B => Último Préstamo = 0 => Resultado = A – B.
- A = B => Último Préstamo = 0 => Resultado = 0.
- A < B => Último Préstamo = 1 => Resultado = 2^n – (B – A) = C_2 (B – A).

Ejemplo:



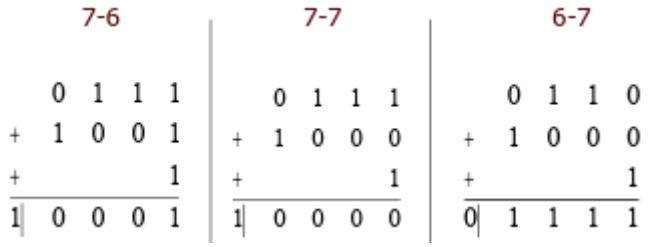
La resta se puede realizar mediante la suma en complemento a 2 del sustraendo.

- A – B = A + C_2 (B) = A + C_1 (B) + 1.

Utilizando un sumador completo:

- A > B => Último Acarreo = 1 => Resultado = A – B.
- A = B => Último Acarreo = 1 => Resultado = 0.
- A < B => Último Acarreo = 0 => Resultado = 2^n – (B – A) = C_2 (B – A).

Ejemplo:



3. Enteros con signo: Representación Signo - Magnitud

Definición:

- Un registro Ac estará compuesto por 2 registros: A_s |A.
- Se toma el bit más significativo como bit de signo (A_s).
  - Si A_s = 1 => Negativo.
  - Si A_s = 0 => Positivo.
- El resto de bits del entero representa la magnitud asociada (A).

Ejemplo:

- 1 001b => -1d.
- 0 001b => +1d.

Problemas:

- Hay que tratar A_s aparte de la magnitud.
- Doble 0 (1 000b // 0 000b).

4. Enteros con signo: Representación Complemento a 2 con signo

Definición:

- Un registro Ac compuesto por 2 registros A_sA.
- Se toma el bit más significativo como bit de signo (A_s).
  - Si A_s = 1 => Negativo.
  - Si A_s = 0 => Positivo.
- El resto de bits del entero representan:
  - Si A_s = 1 => A es la magnitud en complemento a 2.
  - Si A_s = 0 => A es la magnitud.

Ejemplo:

- 0 111b => +7d.
- 1 111b => negativo (111b => not(000b) + 1 = 1) => -1d.

Características:

- No hay que tratar A_s aparte de la magnitud, sino de manera integrada.
- No hay doble 0.

3.1. Suma/Resta Representación Signo - Magnitud

Operaciones combinadas:

- Suma.
- Resta.

Tabla de operaciones efectivas según signos:

Operación	Sumar magnitudes	Restar magnitudes		
		Si A>B	Si A<B	Si A=B
(+A) + (+B)	+(A + B)			
(+A) + (-B)		+(A – B)	-(B – A)	+(A – B)
(-A) + (+B)		-(A – B)	+(B – A)	+(A – B)
(-A) + (-B)	-(A + B)			
(+A) - (+B)		+(A – B)	-(B – A)	+(A – B)
(+A) - (-B)	+(A + B)			
(-A) - (+B)	-(A + B)			
(-A) - (-B)		-(A – B)	+(B – A)	+(A – B)

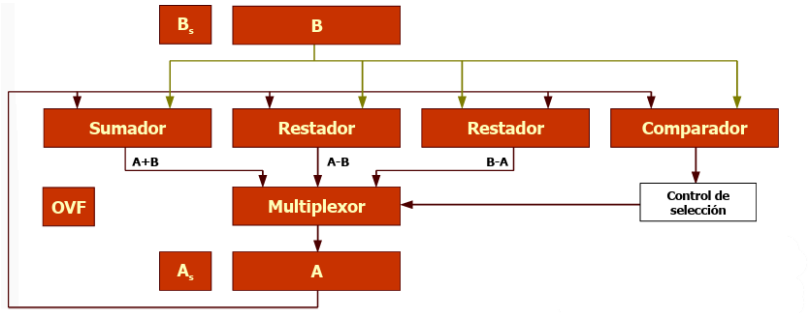
Signo del resultado:

- Si es suma: signo, el de A.
- Si es resta:
  - Si A>B, el de A.
  - Si A<B, el contrario de A.
  - Si A=B, +.

3.1.1. Hardware para la Suma/Resta Representación Signo-Magnitud

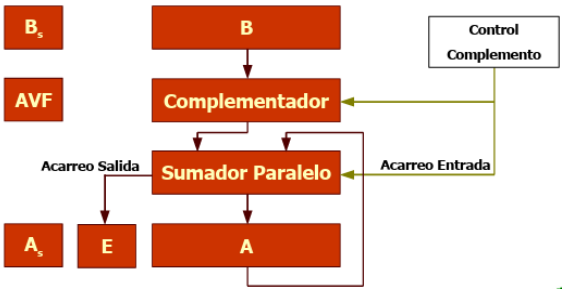
Para realizar las microoperaciones anteriores, en principio, necesitaríamos:

- Un registro para cada operando y un biestable para almacenar el bit de signo de cada uno de los operandos.
- Un biestable para almacenar el sobreflujo de suma (OVF).
- Un circuito comparador, un circuito sumador para calcular A+B, un circuito restador para calcular A-B y un circuito restador para calcular B-A.

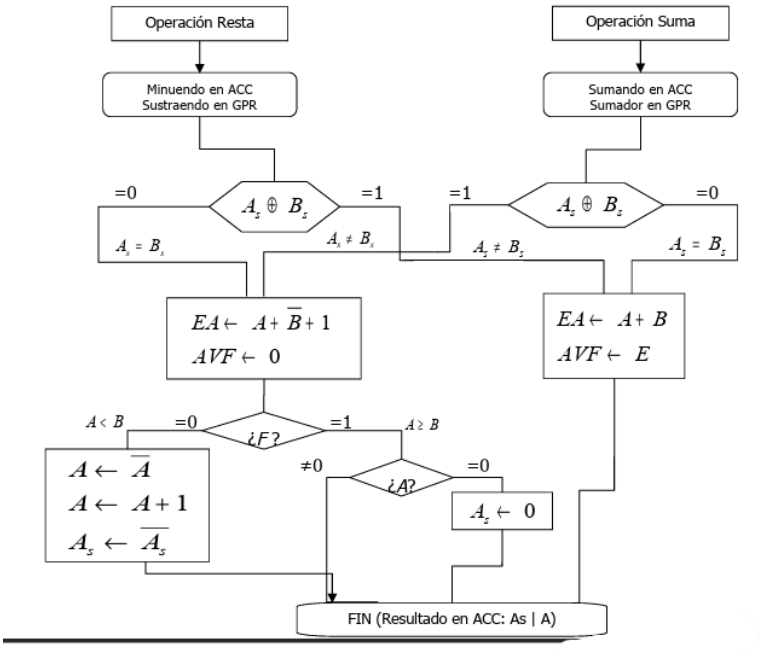


En el caso anterior, estaríamos derrochando hardware. Una mejor implementación consistiría en lo siguiente:

- Registros para los dos operandos (A_sA y B_sB).
- Registro de Sobreflujo (AVF).
- Registro de Acarreo (E).
- Circuito Sumador/Restador (Complemento a 2).



3.1.2. Algoritmo Suma/Resta Representación Signo - Magnitud



3.2. Multiplicación Representación Signo - Magnitud

La multiplicación de dos números de N bits (+1 de signo) da como resultado un número de 2·N bits (+1 de signo).

- **El signo del resultado será:**
  - +, si los dos números tienen el mismo signo.
  - -, si los dos números tienen signo distinto.

- La magnitud del resultado será el producto de las magnitudes de los dos operandos.

Ejemplo de multiplicación de dos magnitudes:

					1	0	1	1	1
				x	1	0	0	1	1
					1	0	1	1	1
				1	0	1	1	1	
			0	0	0	0	0		
		0	0	0	0	0			
	1	0	1	1	1				
0	1	1	0	1	1	0	1	0	1

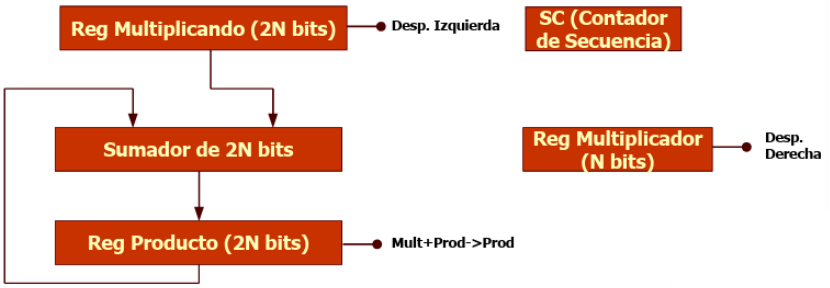
La técnica a seguir es repetir N veces el siguiente bucle:
- Si el bit i-ésimo es del multiplicador es 1: Sumar el multiplicando desplazado a la izquierda “i” veces al resultado parcial (de 2N bits).

El hardware que necesitaríamos para multiplicar las dos magnitudes consistiría (en un principio) en:
- Dos registros de 2N bits para el multiplicando y el resultado.
- Un registro de N bits para el multiplicador.
- Un sumador completo de 2N bits.
- Un contador de secuencia (SC) para realizar las N iteraciones.

Se comenzaría introduciendo el multiplicando en los N bits menos significativos del Reg Multiplicando.

El bucle consistiría en comprobar el bit menos significativo del Reg Multiplicador:
- Si el bit es 1: Se suman Reg Multiplicando y Reg Producto.
- Si el bit es 0: No se hace nada.

El bucle terminaría desplazando el Reg Multiplicando una posición a la izquierda y el Reg Multiplicador una posición a la derecha.

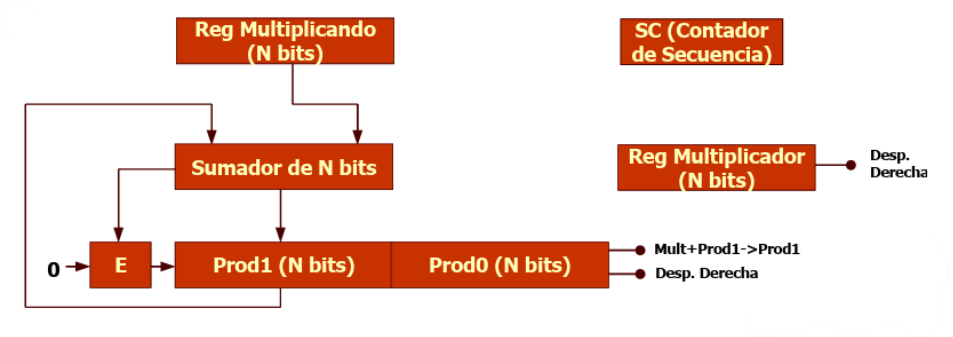


El problema de esta implementación es que la mitad de los bits del Reg Multiplicando siempre valen cero. Con lo cual, estamos usando hardware de más.

Otra opción es dejar el multiplicando fijo y, en lugar de ir desplazándolo a la izquierda, se pasará a desplazar a la derecha el Reg Producto.

De esta forma, el Reg Multiplicando pasa a ser de N bits y el circuito sumador necesario pasa a ser, también, de N bits.

Reg Prod pasa a dividirse en dos registros de N bits (Prod1 y Prod0), en donde Prod1 es el que está conectado a la salida del sumador de N bits.



Con esta nueva implementación, el bucle cambiará un poco. Al igual que antes, se comienza comprobando el bit menos significativo del Reg Multiplicador:

- Si el bit es 1: Se suma Reg Multiplicando con el registro Prod1.
- Si el bit es 0: No se hace nada.

El bucle terminará, en este caso, desplazando los registros Prod1/Prod0 y Reg Multiplicador una posición a la derecha.

Después de N iteraciones, el producto se encontrará en el conjunto formado por los registros Prod1 y Prod0.

Esta implementación se puede mejorar un poco más. Al principio, el valor de Prod0 es irrelevante y, con cada desplazamiento a la derecha generado en el bucle, se va ocupando poco a poco con valores significativos.

Casualmente, el Reg Multiplicador es también de N bits y, con cada desplazamiento a la derecha generado en el bucle, va perdiendo bits significativos.

Por tanto, ambos registros se pueden combinar en uno solo.

Relacionando este hardware optimizado con registros utilizados en operaciones anteriores tenemos que:

- El Reg Multiplicando de N bits se puede relacionar con el registro B.
- El registro Prod1 se puede relacionar con el registro A.

No hay equivalencia entre el registro Prod0 y registros utilizados anteriormente. Con lo cual, tendremos que introducir un nuevo registro unido a A, de modo que puedan realizar desplazamientos a la derecha como conjunto. Este nuevo registro lo llamaremos Q. De este registro, el bit menos significativo (Q_n) será accesible como bit de estado.

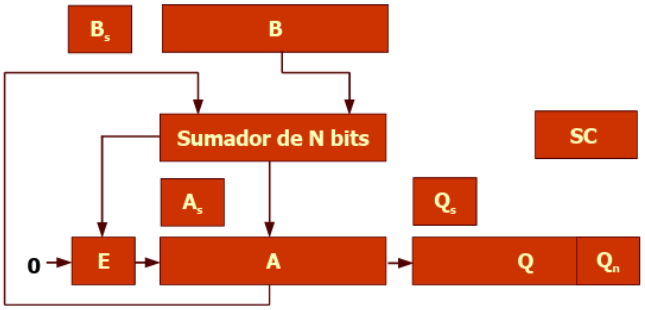
Además de estos registros, tal y como se ha visto en los esquemas anteriores, hará falta un registro Contador de Secuencia (SC), además de un bit E para especificar el acarreo.

El algoritmo que utiliza esta implementación partirá con el multiplicando en BS B y el multiplicador en Qs Q. Se comprobará el bit Q_n :

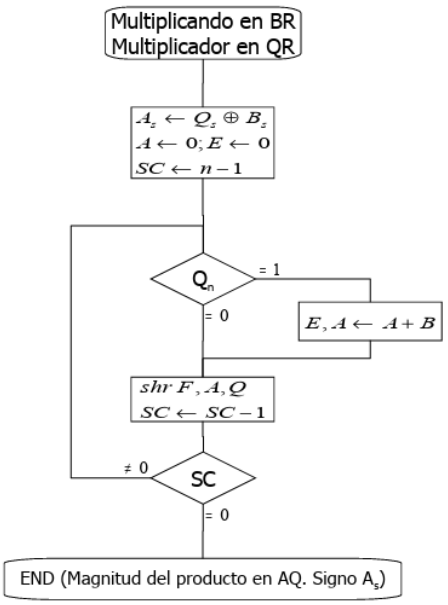
- Si vale 1: se suman las magnitudes B y A.
- Si vale 0: no se hace nada.

El bucle terminará desplazando un bit a la derecha el conjunto formado por E, A y Q.

Después de N iteraciones, el resultado de la multiplicación se encontrará en A_sAQ.



3.2.1. Algoritmo Multiplicación Representación Signo - Magnitud



3.2. Multiplicación Representación Signo - Magnitud

Ejemplo (multiplicación solo magnitud):

- N: 5 ; Multiplicando (B): 10111 ; Multiplicador (Q): 10011.
- Resultado: 0110110101.

```
                1 0 1 1 1
              x 1 0 0 1 1
-----------------------------
                1 0 1 1 1
              1 0 1 1 1
            0 0 0 0 0
          0 0 0 0 0
        1 0 1 1 1
-----------------------------
0 1 1 0 1, 1 0 1 0 1
```

	E	A	Q	SC
Multiplicando B: 10111. Multiplicador Q: 10011.	0	00000	10011	5
Q_n = 1 => Sumar B	0	10111	10011	
shr E,A,Q	0	01011	11001	4
Q_n = 1 => Sumar B	1	01011 + 10111 00010	11001	
shr E,A,Q	0	10001	01100	3
Q_n = 0 => shr E,A,Q	0	01000	10110	2
Q_n = 0 => shr E,A,Q	0	00100	01011	1
Q_n = 1 => Sumar B	0	00100 + 10111 11011	01011	
shr E,A,Q	0	01101	10101	0

3.3. División Representación Signo - Magnitud

La división de un número de 2N bits por otro de N bits (+1 de signo) da como resultado N bits de cociente (+1 de signo) y N bits de resto (+1 de signo).

- **El signo del cociente será:**

    - +, si los dos números tienen el mismo signo.
    - -, si los dos números tienen signo distinto.

- La magnitud del cociente será la división de las magnitudes de los dos operandos.
- El signo del resto será el mismo que el del dividendo.

La técnica a seguir es repetir N veces el siguiente bucle:

- Resta, si cabe, los N bits del divisor al residuo parcial.
    - Si cabe, desplazar el cociente un bit a la izquierda, poniendo su bit menos significativo a 1.
    - Si no cabe, desplazar el cociente un bit a la izquierda, poniendo su bit menos significativo a 0.
- El bucle termina desplazando un bit a la derecha el registro divisor.

El hardware que necesitaríamos para dividir las dos magnitudes consistiría (en un principio) en:

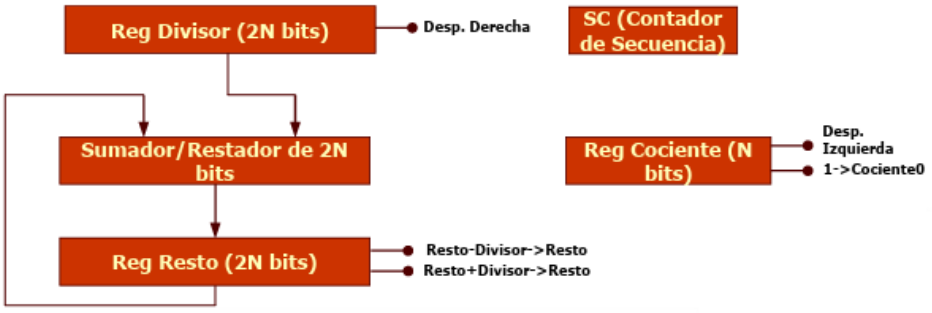
- Dos registros de 2N bits para el divisor y el dividendo (que, tras finalizar el algoritmo, contendrá el resto).
- Un registro de N bits para el cociente.
- Un circuito sumador/restador completo de 2N bits.
- Un contador de secuencia (SC) para realizar las N iteraciones.

Se comenzaría introduciendo el divisor en los N bits más significativos del Reg Divisor.

El bucle consistiría en restar el Reg Divisor al Reg Resto:

- Si resto ≥ 0: Se desplaza Reg Cociente un bit a la izquierda, poniendo su bit menos significativo a 1.
- Si resto < 0: Se restaura el valor de Reg Resto sumándole el Reg Divisor. Se desplaza Reg Cociente un bit a la izquierda, dejando su bit menos significativo a 0.

El bucle terminaría desplazando Reg Divisor un bit a la derecha.

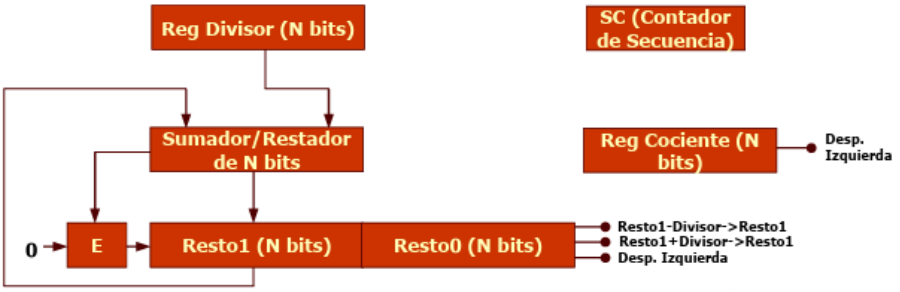


El problema de esta implementación es que la mitad de los bits del Reg Divisor siempre valen cero. Por tanto, estamos derrochando hardware con esta implementación.

Otra opción es dejar el divisor fijo y, en lugar de ir desplazándolo a la derecha, se pasará a desplazar a la izquierda el Reg Resto.

De esta forma, el Reg Divisor pasa a ser de N bits y el circuito sumador/restador necesario pasa a ser, también, de N bits.

Reg Resto pasa a dividirse en dos registros de N bits (Resto1 y Resto0), en donde Resto1 es el que está conectado a la salida del sumador/restador de N bits.



Esta nueva distribución del hardware, implicará algunos cambios en el bucle. El bucle empieza restando Reg Divisor a Resto1:

- Si Resto1 ≥ 0: Se desplaza Reg Cociente un bit a la izquierda, poniendo su bit menos significativo a 1.
- Si Resto1 < 0: Se restaura el valor de Resto1 sumándole el Reg Divisor. Se desplaza Reg Cociente un bit a la izquierda, dejando su bit menos significativo a 0.

El bucle terminará, en este caso, desplazando los registros Resto1/Resto0 una posición a la izquierda.

Después de N iteraciones, el cociente se encontrará en Reg Cociente y el resto en Resto1.

Esta implementación se puede mejorar un poco más. Al principio, el valor de Reg Cociente es irrelevante y, con cada desplazamiento a la izquierda generado en el bucle, se va ocupando poco a poco con valores significativos.

Casualmente, Resto0 es también de N bits y, con cada desplazamiento a la izquierda generado en el bucle, va perdiendo bits significativos.

Por tanto, ambos registros se pueden combinar en uno solo.

Relacionando este hardware optimizado con registros utilizados en operaciones anteriores tenemos que:

- El Reg Divisor de N bits se puede relacionar con el registro B.
- El registro Resto1 se puede relacionar con el registro A.

Para Resto0 necesitamos un registro conectado al acumulador, de modo que puedan realizar desplazamientos a la izquierda como conjunto. Para este caso nos puede servir el mismo registro Q que fue utilizado en la multiplicación S-M, con la diferencia que debe posibilitar los desplazamientos a la izquierda. Además, el bit menos significativo (Q_n) debe tener un terminal de control que habilite su puesta a 1.

El algoritmo que utiliza esta implementación partirá con el divisor en BS B y el dividendo en A_sAQ.

Se empieza desplazando a la izquierda EAQ.

- Si E=1, sabemos directamente que el conjunto EA es mayor que lo que hay en B (el divisor). Luego podemos restar B a A e introducimos un 1 en Q_n.
- Si E=0, no sabemos en principio si lo que hay en A es mayor o igual que lo que hay en B. Con lo cual, restamos (sumamos B'+1) a A y vemos el acarreo:
    - Si E=1, quiere decir que lo que había en A era mayor o igual que lo que había en B, luego la operación de resta fue correcta, pasando a introducir 1 en Q_n .

- Si E=0, quiere decir que lo que había en A era menor que lo que había en B. Como no teníamos que haber realizado la resta, volvemos a sumar el contenido de B a A para restaurar el valor que teníamos antes de restar.

Después de N iteraciones, el cociente estará en Q_sQ y el resto en A_sA.

El único problema que queda por resolver es el del sobreflujo de división.

El sobreflujo de división se produce cuando el resultado de la división necesita más de N bits para ser representado.

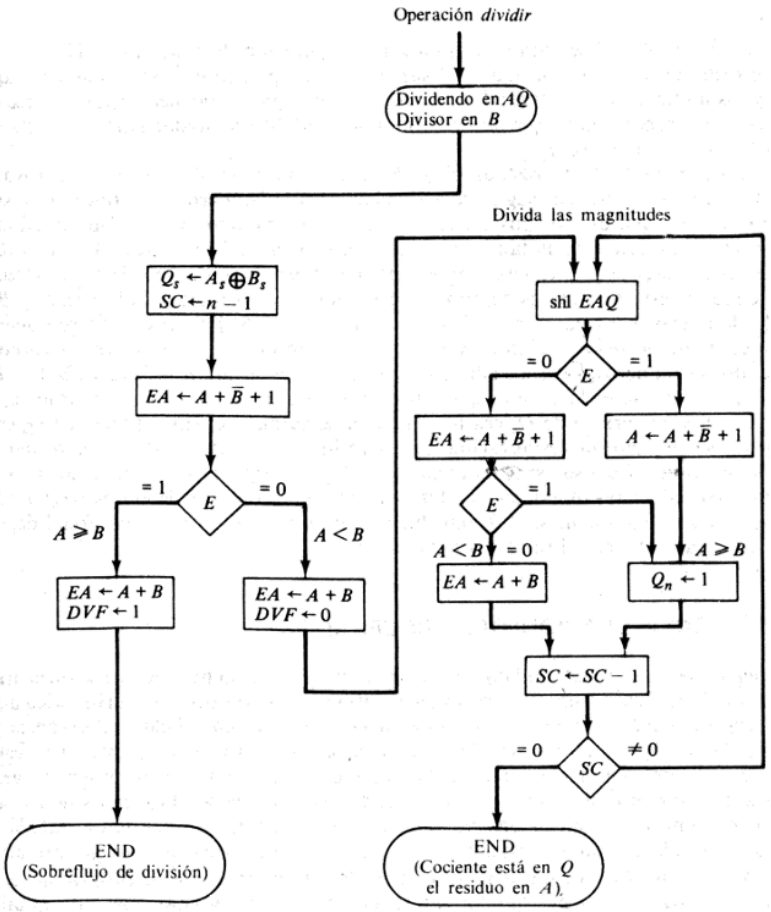
Un método para identificar si existirá sobreflujo de división o no, consiste en comparar los N bits más significativos de la magnitud del dividendo, con los N bits del divisor:

- Si los N bits del divisor representan un número menor que los N bits más significativos del dividendo, existirá sobreflujo. Como el algoritmo dará lugar a un resultado incorrecto, directamente no se realiza, mandando un mensaje de error.
- En caso contrario, no existirá sobreflujo, con lo que se puede proceder con el algoritmo.

Para especificar mediante hardware si existe sobreflujo o no, se puede utilizar un biestable (DVF) que sirva de bit de estado del sistema:

- Si DVF=0 no existe sobreflujo de división.
- Si DVF=1 existe sobreflujo de división.

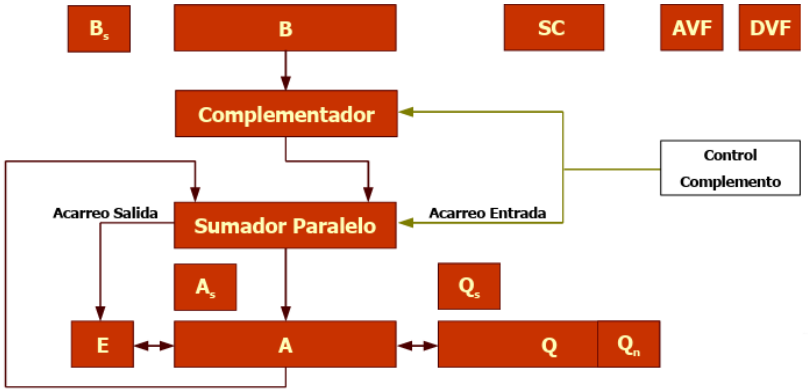
3.3.1. Algoritmo División con Restauración Representación Signo - Magnitud



3.3. División Representación Signo - Magnitud

Como conclusión, la misma circuitería que fue usada para la multiplicación (con algún retoque), también sirve para llevar a cabo el algoritmo de división. Pero no sólo eso, como para la división es necesario tener un sumador/restador, también servirá la misma circuitería para realizar las instrucciones de suma y resta en signo-magnitud.

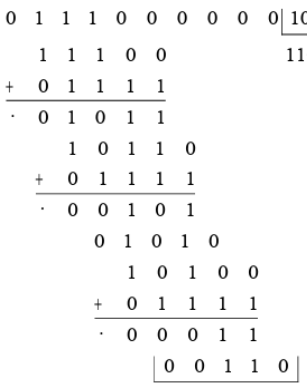
El hardware necesario para realizar las cuatro operaciones en signo-magnitud será, por tanto:



3.3.2. División con restauración Representación Signo - Magnitud

Ejemplo (división solo magnitud):

- N: 5 ; Divisor (B): 10001.
- Dividendo (AQ): 01110 00000.
- Cociente (Q): 11010 ;
- Resto (A): 00110.
- B complemento + 1 = 01111.



	E	A	Q	SC
Divisor B: 10001. Dividendo AQ: 01110 00000	0	01110	00000	5
Shl EAQ	0	11100	00000	
Sumar B'+1	1	11100 + 01111 01011	00000	
E=1 => Q_n = 1	1	01011	00001	4
Shl EAQ	0	10110	00010	
Sumar B'+1	1	10110 + 01111 00101	00010	
E=1 => Q_n = 1	1	00101	00011	3
Shl EAQ	0	01010	00110	
Sumar B'+1	0	01010 + 01111 11001	00110	
E=0 => Q_n = 0	0	11001	00110	2
Sumar B (Restaurar el registro)	1	11001 + 10001 01010	00110	
Shl EAQ	0	10100	01100	
Sumar B'+1	1	10100 + 01111 00011	01100	
E=1 => Q_n = 1	1	00011	01101	1
Shl EAQ	0	00110	11010	
Sumar B'+1	0	00110 + 01111 10101	11010	
E=0 => Q_n = 0	0	10101	11010	
Sumar B (Restaurar el registro)	1	10101 + 10001 00110	11010	0
Despreciar E. Resto en A		00110		
Cociente en Q			11010	

4.1. Suma/Resta Representación Complemento a 2

Hardware:

- Ac: A_s|A.
- BR/GPR: B_s|B.
- El bit de signo está integrado dentro del registro completo.
- Sumador completo.

Suma: Realizar la operación utilizando el sumador completo: Ac ← Ac + BR.

Resta: Realizar la operación realizando el complemento a 2 del sustraendo: Ac ← Ac + BR complementado + 1.

Problema:

- Posibilidad de desbordamiento. No validez del resultado => Faltan bits para representar el valor.
- Información Estado: Registro V.
- Una forma de calcular el sobreflujo de la suma es analizando los dos últimos acarreos proporcionados por el sumador completo:
    - Si los dos últimos acarreos son iguales, no hay sobreflujo (V=0).
    - Si los dos últimos acarreos son distintos, hay sobreflujo (V=1).

4.1.1. Desplazamientos Representación Complemento a 2

Desplazamiento lógico

- Mover el contenido de cada bit de la posición actual a la posición inmediatamente anterior o posterior (desplazamiento derecha o izquierda respectivamente).
- El bit inicial o final, según el desplazamiento, se rellena con 0.
- No se mantiene el signo negativo.

Desplazamiento aritmético o con mantenimiento del signo

- Mover el contenido de cada bit de la posición actual a la posición inmediatamente anterior o posterior (desplazamiento derecha o izquierda respectivamente), manteniendo el bit de signo: Aplicar el desplazamiento lógico y alterar el bit de signo (si necesario).
- Para desplazamientos a derecha, ingresar en el bit de signo el mismo valor que existía antes del desplazamiento.
- Para desplazamientos a izquierda, complementar el bit de signo, si varía el valor del bit de signo antes y después del desplazamiento lógico.

4.2. Multiplicación Representación en Complemento a 2

Multiplicación con números en complemento a 2:

- Se puede utilizar el mismo algoritmo de multiplicación en representación signo-magnitud:
    - **Multiplicador positivo y Multiplicando positivo:**
        - Algoritmo multiplicación signo-magnitud olvidándose del signo.
    - **Multiplicador positivo y Multiplicando negativo:**
        - Algoritmo multiplicación signo-magnitud.
        - Desplazamientos aritméticos (desplazamientos manteniendo el signo).
    - **Multiplicador negativo:**
        - No se puede utilizar el algoritmo multiplicación signo-magnitud.
        - El multiplicador siempre debe ser positivo.

Algoritmo Multiplicación I

- Asegurar que ambos (multiplicando/multiplicador) sean positivos.
    - Calcular el signo del resultado (XOR).
    - Se convierten en positivos.
    - Se aplica el algoritmo multiplicación signo-magnitud.
    - Si el signo del resultado debe ser negativo, aplicar complemento a 2.

Algoritmo Multiplicación II

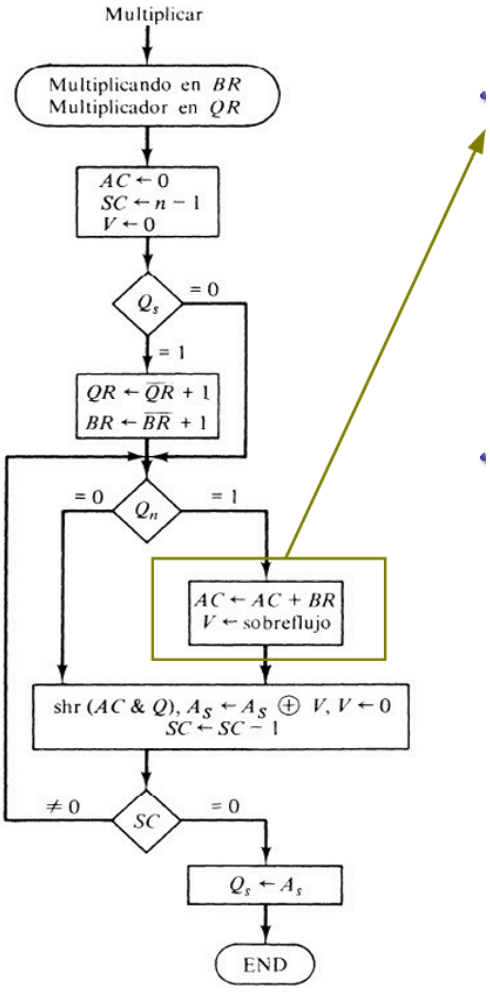
- Asegurar que el multiplicador sea positivo.
    - Comprobar el signo del multiplicador (Q_s).
    - Si es negativo, cambiar el signo de los dos (Multiplicador positivo y multiplicando positivo o negativo).
    - Aplicar el algoritmo multiplicación signo-magnitud.

Cuando se produce sobreflujo de suma con números en complemento a dos, se produce una inversión de signo en el resultado:

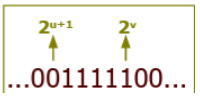
- Si se suman dos números positivos, se obtiene como resultado un número negativo.
- Si se suman dos números negativos, se obtiene como resultado un número positivo.

Por tanto, después de realizar la suma parcial, a la hora de hacer el desplazamiento aritmético a la derecha:

- Si no hubo sobreflujo tras la suma (V=0), se desplazan todos los bits manteniendo el signo.
- Si hubo sobreflujo tras la suma (V=1), se desplazan todos los bits a la derecha. Como el bit de signo es incorrecto, se complementa.



Algoritmo de multiplicación de Booth

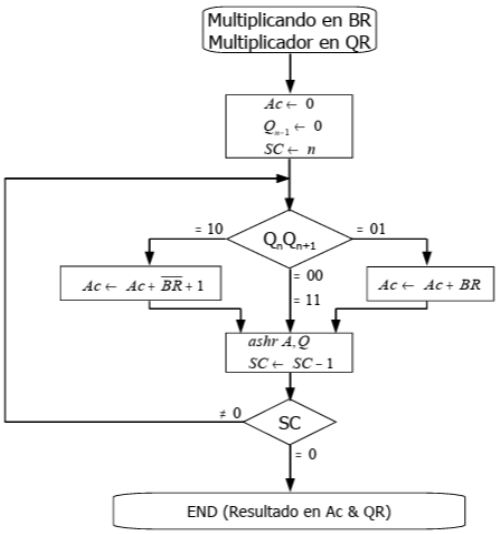


- Una hilera de 0's seguidos no requiere ninguna suma -> ...0011111[00]...
- Una hilera de 1's seguidos puede simplificarse sumando (2^u+1 - 2^v), donde u es la última posición de la hilera de 1's y v la primera posición de dicha hilera.
- Ante la primera aparición de un 1 tras 0 en el multiplicador, se resta el multiplicando al valor parcial -> ...001111[10]0…
- Se realizan desplazamientos del valor parcial si el bit tratado es idéntico al anterior. El desplazamiento debe ser aritmético para no perder el signo.
- En la aparición de un 0 tras un 1 en el multiplicador, se suma el multiplicando al valor parcial -> ...0[01]111100...

Hardware adicional:

- Registro de 1 bit Q_{n+1} que contiene el valor de Q_n en la iteración anterior.

4.2.1. Algoritmo de multiplicación de Booth Representación Complemento a 2



Ejemplo (multiplicación de Booth):

- N: 7; Multiplicando (BR): 0101100 (44); Multiplicador (QR): 1011110 (-34).
- Resultado (AQ): 11101000101000 (-1496).
- B complementado + 1 = 1010100.

							0	1	0	1	1	0	0
						x	1	0	1	1	1	1	0
1	1	1	0	1	0	0	0	1	0	1	0	0	0

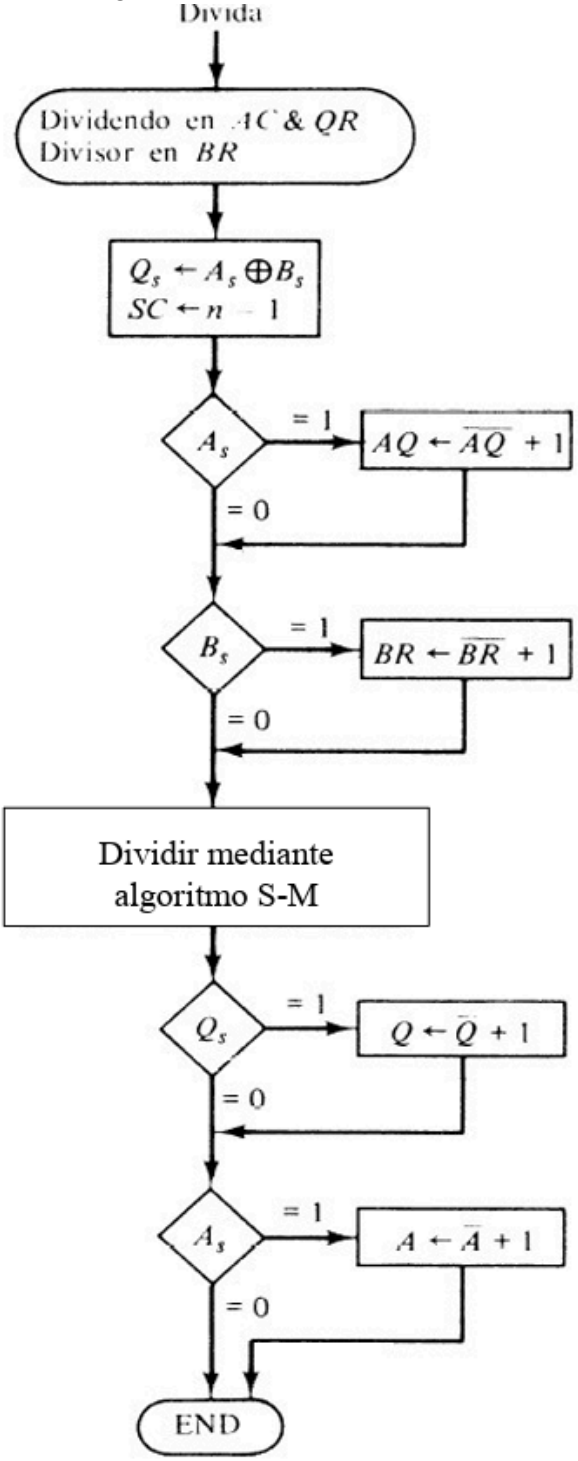
	44
x	- 34
	- 1496

A	Q	Q_{n+1}	SC	
0000000	1011110	0	7	ashr / dec SC
0000000	0101111	0	6	Resta B
0000000 + 1010100 1010100	0101111			ashr /dec SC
1101010	0010111	1	5	ashr /dec SC
1110101	0001011	1	4	ashr /dec SC
1111010	1000101	1	3	ashr /dec SC
1111101	0100010	1	2	Suma B
1111101 + 0101100 0101001	0100010			ashr /dec SC
0010100	1010001	0	1	Resta B
0010100 + 1010100 1101000	1010001			ashr /dec SC
1110100	0101000			AQ: Resultado

4.3. División Representación en Complemento a 2

División con números en complemento a 2:

- Debido a las complicaciones que puede generar la división entre sumas y restas en los residuos parciales (según el signo relativo entre dividendo y divisor) y también a la hora de generar el cociente en la representación correcta, lo más conveniente es:
    - Determinar el signo del cociente a partir de los signos del dividendo y el divisor.
    - Convertir los dos números en positivos.
    - Dividir ambos números siguiendo el algoritmo de Signo-Magnitud.
    - Si el resultado es negativo, se complementa.



5. Representación Punto Flotante

Reales en notación científica:

- N = S·M·b^e.
  - N: Número real.
  - S: Signo.
  - M: Mantisa.
    - Parte entera.
    - Parte decimal.
    - Mezcla de ambas.
  - b: Base
  - e: exponente

- **Ejemplo:**
  - N = -4560 · 10^32.
  - N = 5.629 · 10^20.
  - N = 0.0011101b · 2^1011b.

Ventajas notación científica:

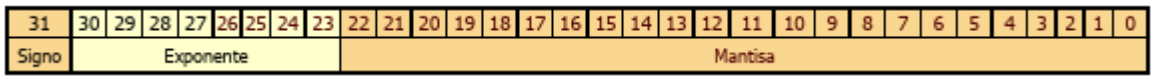
- Facilidad para variar el punto decimal
  - 5.62 · 10^20 = 56.2 · 10^19 = 0.562 · 10^21.
  - 0.01101·2^1101 = 0.1101 · 2^1100.

Representación normalizada binaria:

- La mantisa representa solo la parte decimal, es decir, es un valor con parte entera igual a 0.
- Además en la mantisa, el primer dígito tras el punto es distinto de 0.
- Los valores negativos se implementan mediante representación Signo-Magnitud.
- El exponente se representa sesgado.
- La base no se incluye (se sabe que es 2).

Representación 32bits (simple precisión):

- 1 bit de signo.
- 8 bits de exponente con sesgo (sesgo=128).
- 23 bits de mantisa (normalizada).



Valor:

$(-1)^{signo} \cdot (0.Mantisa) \cdot 2^{(Exponente-128)}$

Ejemplo:

- 0 10000111 11000000000000000000000.
- Signo: 0 => Positivo.
- Exponente Sesgado: 10000111.
  - Exponente sin sesgo: 10000111 – 10000000 = 00000111.
- Mantisa: 0.11000000000000000000000.
- Valor decimal: + 0.11b · 2^111b = 0.75 · 2^7 = 0.75 · 128 = 96.0.

5.1. Estándar IEEE 754 Representación Punto Flotante

Estándar internacional para definición de punto flotante. Dos precisiones:

- Simple 32bits / Doble 64 bits.
- Formato “signo-exponente-mantisa”.
- Valor decimal:

$$N = (-1)^{signo} \cdot 1.mantisa \cdot 2^{exponente-sesgo}$$

    - Sesgo = 127 (para simple precisión); Sesgo = 1023 (doble precisión).
- Se aumenta en un bit la capacidad de representación.

5.2. Representación de Números Singulares

- **Cero:**
    - Signo = X ; Exponente = 0 ; Mantisa = 0.

- **Infinito Positivo:**
    - Signo = 0 ; Exponente = 255 ó 2047 ; Mantisa = 0.

- **Infinito Negativo:**
    - Signo = 1 ; Exponente = 255 ó 2047 ; Mantisa = 0.

- **Números denormalizados:**
    - Signo = X ; Exponente = 0 ; Mantisa ≠ 0.

- **No un número (NaN):**
    - Signo = X ; Exponente = 255 ó 2047 ; Mantisa ≠ 0.

5.3. Hardware para Operaciones en Punto Flotante

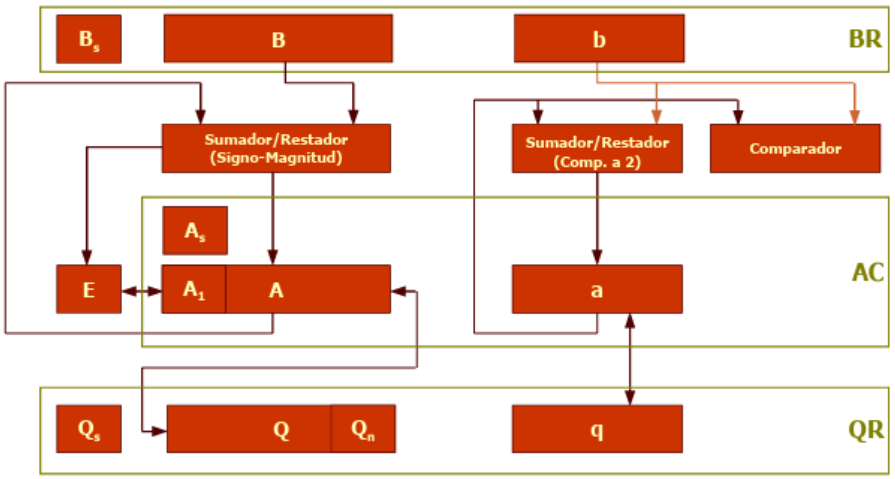
3 Registros compuestos:

- Signo | Mantisa | Exponente.
- Ac: A_s | A | a.
- BR: B_s | B | b.
- QR: Q_s | Q | q.

Circuitos aritméticos que operan independientemente mantisas / exponentes.

- Operaciones Mantisas: Basados en representación signo-magnitud.
- Operaciones Exponentes: Basados en complemento a 2.

- Sumador/Restador para mantisas.
- Sumador/Restador y Comparador para exponentes.

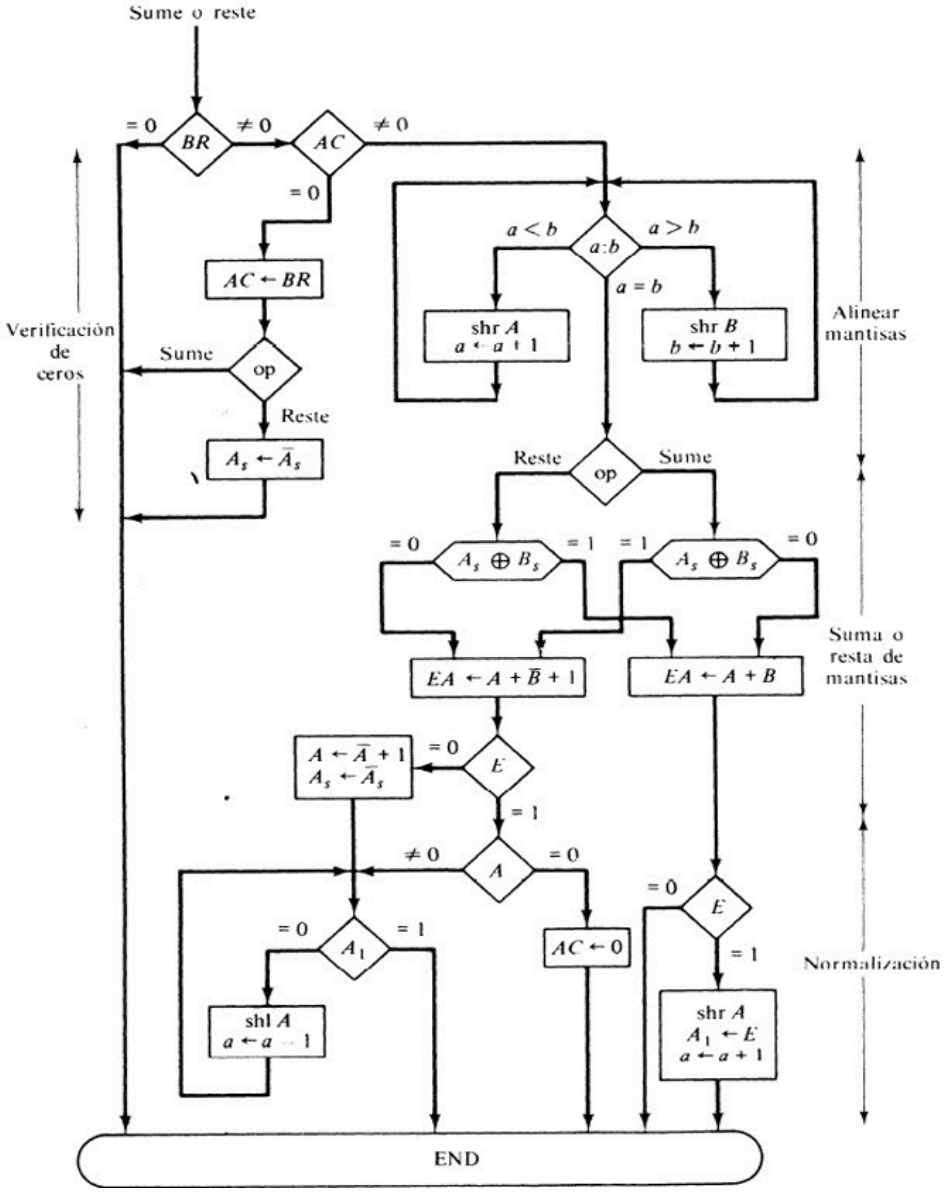


5.3.1. Operaciones aritméticas en Punto Flotante

Suma/Resta.

- Los operandos deben tener el mismo exponente:
    - Se desplaza a la derecha el operando con el exponente menor y se incrementa su exponente, hasta que ambos exponentes coinciden.

- Una vez comprobado que los operandos tienen el mismo exponente: el exponente del resultado será el común y la mantisa del resultado se obtendrá mediante la suma/resta (respectivamente) en signo-magnitud de las mantisas de los operandos.
- Por último, se comprueba si el resultado de la suma/resta está normalizado. Si no está normalizado, se debe realizar su normalización.

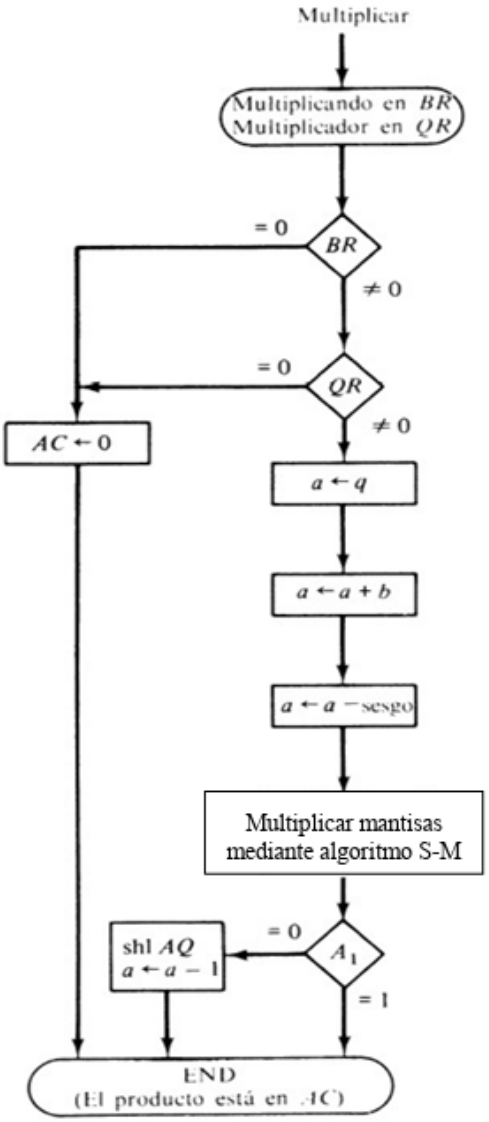


Multiplicación.

- La mantisa del resultado será el producto en signo-magnitud de las mantisas de los operandos.
- El exponente del resultado será la suma de los exponentes de los operandos. Para esto hay que tener una consideración, y es que cada uno de los exponentes de los operandos está sesgado. Para tener el exponente del producto correctamente sesgado, debe restarse el sesgo a la suma de los dos exponentes:

  ExpProd = a + b – sesgo =
  = (ExpReal(a) + sesgo) + (ExpReal(b) + sesgo) – sesgo =
  = ExpReal(a) + ExpReal(b) + sesgo

- Si el producto no está normalizado, sólo es necesario un único desplazamiento a la izquierda del producto, con el correspondiente decremento del exponente.

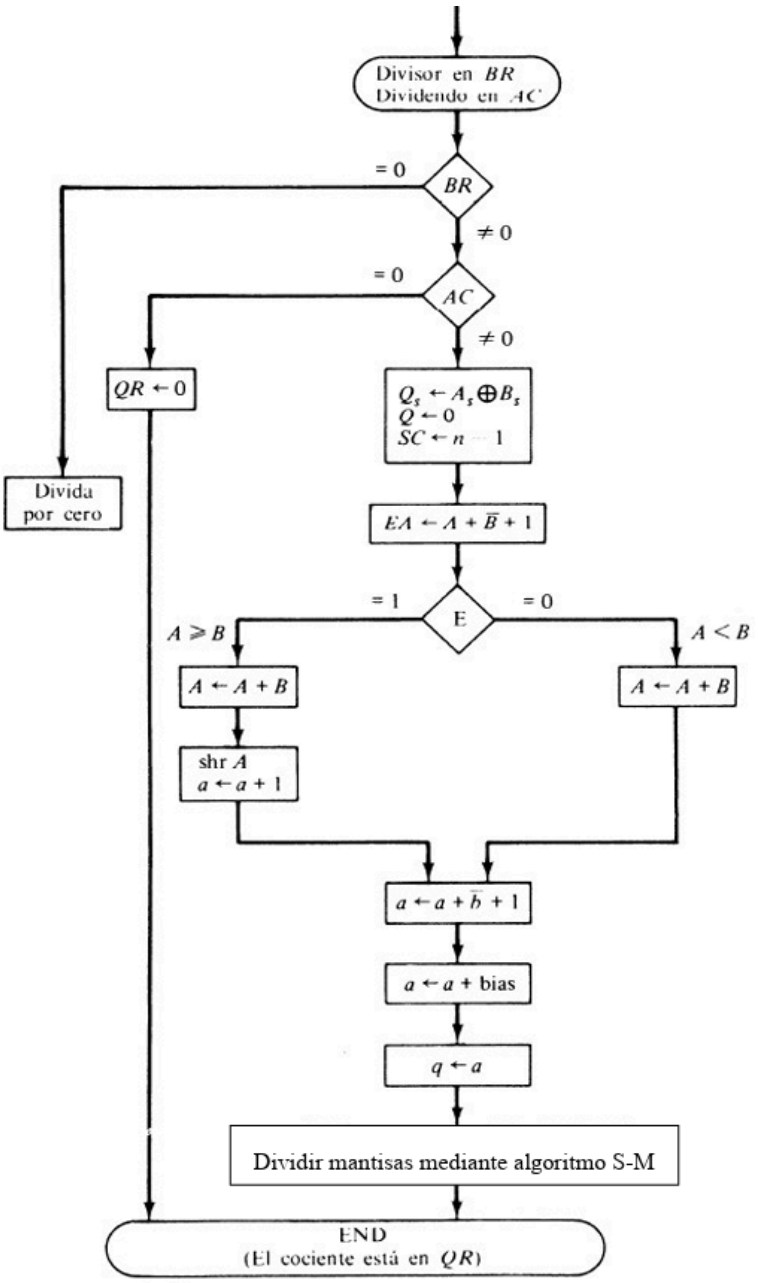


División.

- La mantisa del resultado será la división en signo-magnitud de las mantisas de los operandos.
- El exponente del resultado se obtiene restándole al exponente del dividendo el exponente del divisor. Hay que tener en cuenta que al restar dos números sesgados, desaparece el sesgo. Luego para tener la representación de la división correctamente sesgada, hay que sumarle el sesgo a la resta de exponentes:

$$ExpDiv = a - b + sesgo = (ExpReal(a) + sesgo) - (ExpReal(b) + sesgo) + sesgo =$$
$$= ExpReal(a) - ExpReal(b) + sesgo$$

- El cociente resultante sale directamente normalizado.
- Por último, en punto flotante, no existe el problema del sobreflujo de división. Inicialmente uno comprueba si los N bits más significativos del dividendo representan un número mayor que los N bits del divisor:
    - Si no es mayor, se procede al algoritmo de división.
    - Si es mayor, la solución consiste en realizar un desplazamiento a la derecha sobre el dividendo, con su correspondiente incremento del exponente. Como son números normalizados, después de este desplazamiento, los N bits más significativos del dividendo representan un número menor que los N bits del divisor, con lo que no hay sobreflujo de división.


Divisor en BR
Dividendo en AC
= 0
BR
≠ 0
= 0
AC
≠ 0
QR ← 0
Q_s ← A_s ⊕ B_s
Q ← 0
SC ← n − 1
Divida
por cero
EA ← A + B̄ + 1
= 1
E
= 0
A ⩾ B
A < B
A ← A + B
A ← A + B
shr A
a ← a + 1
a ← a + b̄ + 1
a ← a + bias
q ← a
Dividir mantisas mediante algoritmo S-M
END
(El cociente está en QR)