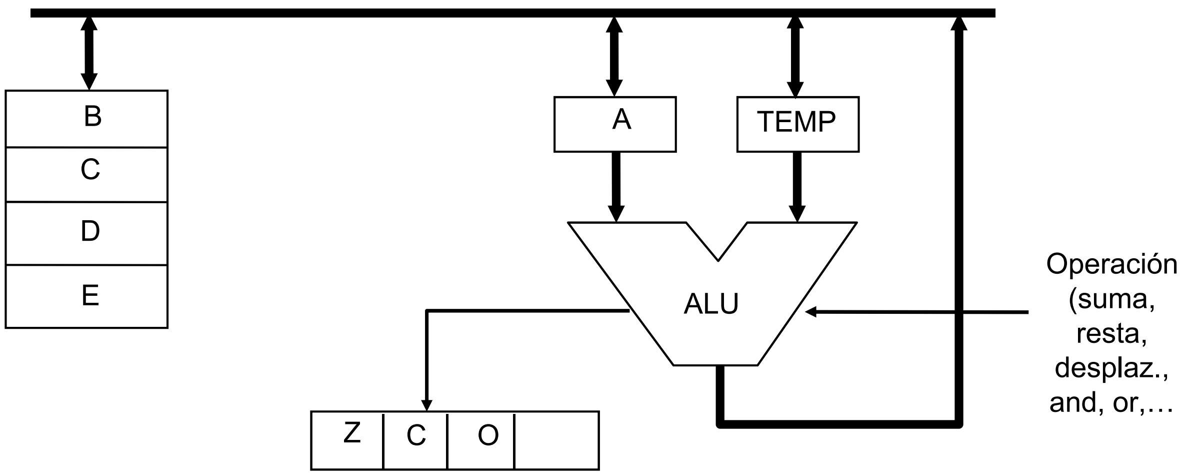
TEMA 2.
UNIDAD ARITMÉTICO-LÓGICA

Índice

ESTRUCTURA GENERAL ALU

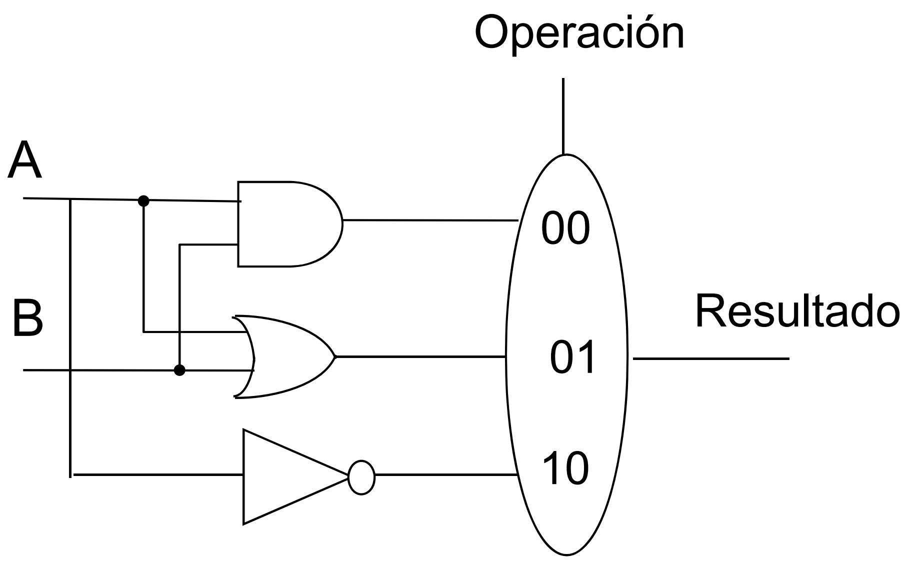


- Operador aritmético, lógico, desplazamiento (uno o varios) (ALU)
- El Acumulador
- Uno o varios registros temporales
- Indicadores de resultado
  - Acarreo (C)
  - Negativo (N)
  - Desbordamiento (O)
  - Cero (Z)

OPERACIONES TÍPICAS

	Potencia de cálculo			
Operación	Mínima	Baja	Media	Alta
Suma/Resta en binario	Comb	Comb	Comb	Comb
Suma/Resta en coma flotante	Prg/Copr	Prg/UC	UC	Secu
Multiplicación en binario	Prg/Copr	Prg/UC	UC	Comb
Multiplicación en coma flotante	Prg/Copr	Prg/UC	UC	Secu
División en binario	Prg/Copr	Prg/UC	UC	Secu
División en coma flotante	Prg/Copr	Prg/UC	UC	Secu
Operaciones lógicas	Comb	Comb	Comb	Comb
Desplazamientos unitarios	Comb	Comb	Comb	Comb
Desplazamientos múltiples	Prg	Prg/UC	UC	Comb

- Fáciles de implementar ⇒ Correspondencia directa con Hardware.
- Puertas lógicas AND, OR, OR-EXCLUSIVA, INVERSORES,...

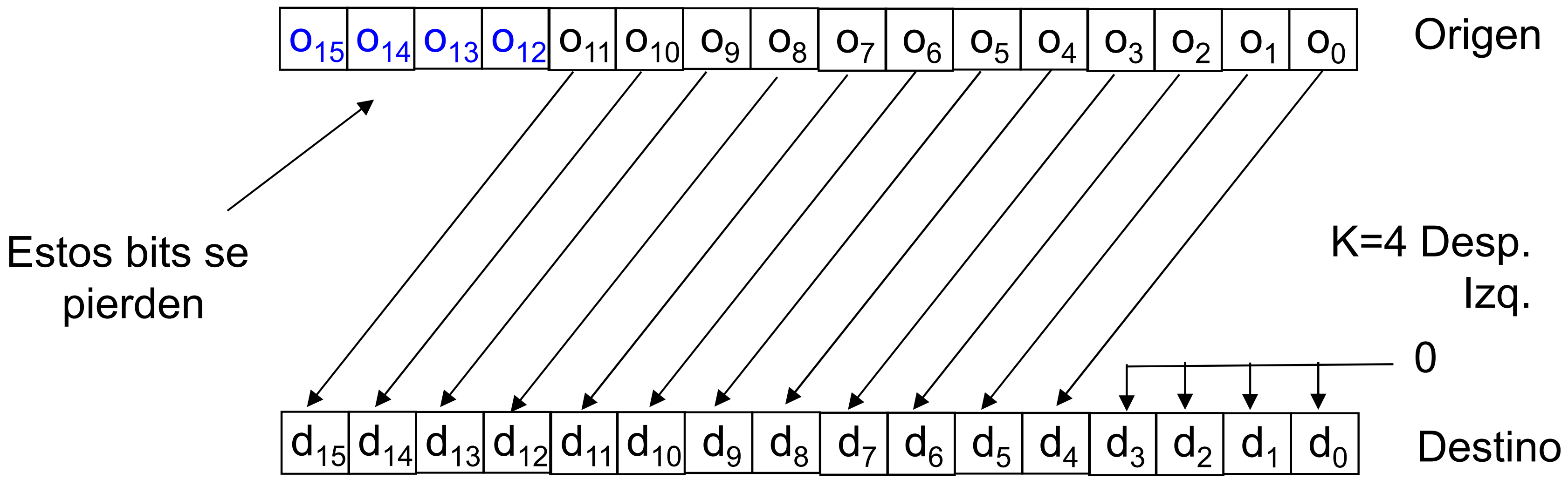


OPERACIONES DE DESPLAZAMIENTO

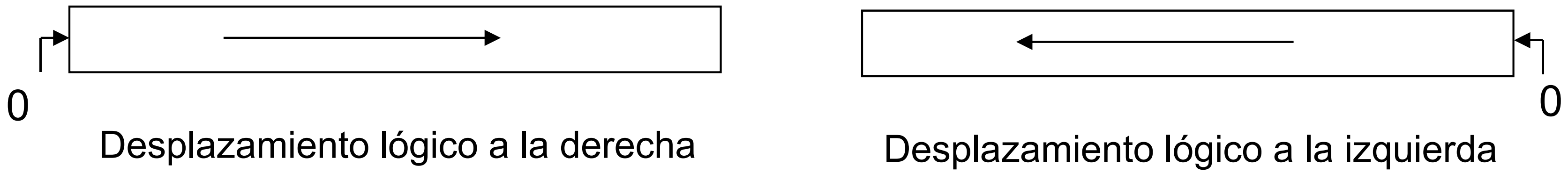
- Consisten en trasladar los bits de una palabra hacia la izquierda o derecha.
- Dependiendo de cómo se traten los extremos, se obtienen tres tipos de desplazamientos:
  - Lógicos
  - Circulares
  - Aritméticos

DESPLAZAMIENTOS LÓGICOS

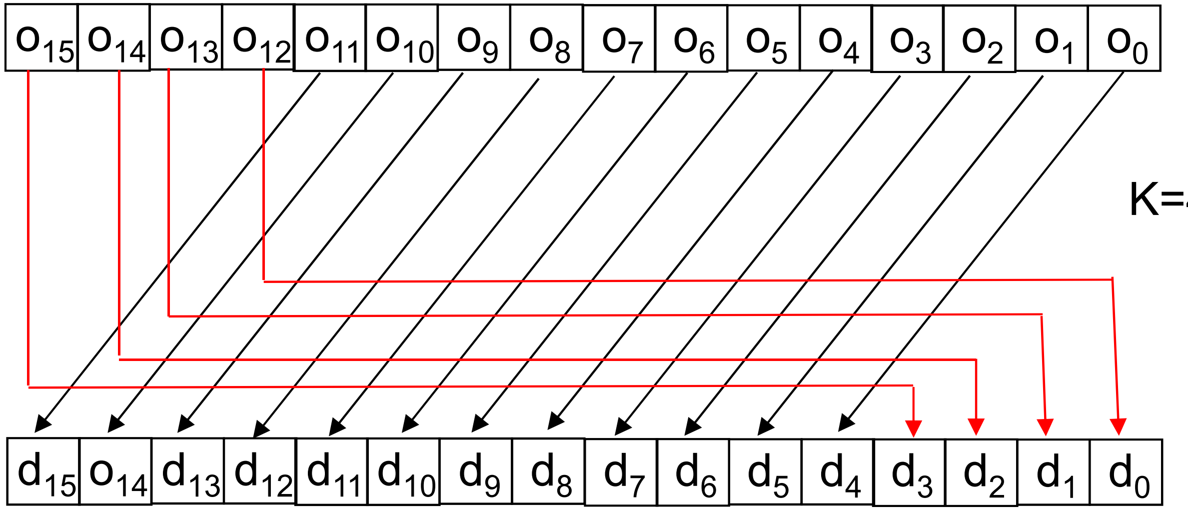
- Los valores extremos se completan con ceros, aunque se pueden plantear desplazamientos lógicos con inclusión de unos en lugar de ceros

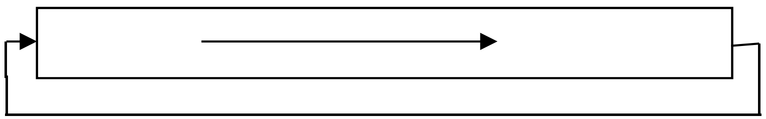


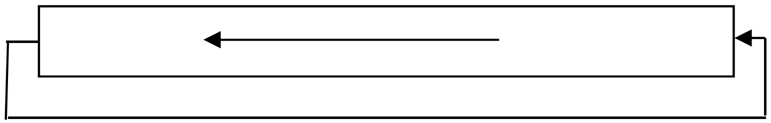
Habitualmente, el origen y destino es la misma palabra.



- Los bits del origen que sobran por un lado, se insertan en el destino por el otro, matemáticamente:

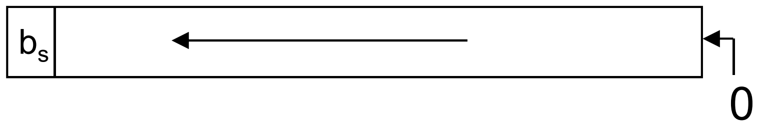



Desplazamiento circular a la derecha


Desplazamiento circular a la izquierda

DESPLAZAMIENTOS ARITMÉTICOS

- Se tiene en cuenta el bit de signo y se representa en complemento a 2.
- Desplazamiento a la **izquierda** (multiplicación por 2):



Desplazamiento aritmético a la izquierda

  - Se van perdiendo bits de signo y hay que introducir ceros por la derecha para números positivos
  - Para que no haya overflow hay que comprobar el bit de signo después de la operación:
    - Si el número es positivo, se desplaza y da negativo -> Overflow
    - Si el número es negativo, se desplaza y da positivo -> Overflow

DESPLAZAMIENTOS ARITMÉTICOS

- Ejemplos:

O= 0000 1010 = 10_{10} x2 → D= 0001 0100 = 20_{10} → correcto

O= 0100 0000 = 64_{10} x2 → D= 1000 0000 = -128_{10} → incorrecto

O= 1111 1100 = -4_{10} x2 → D= 1111 1000 = -8_{10} → correcto

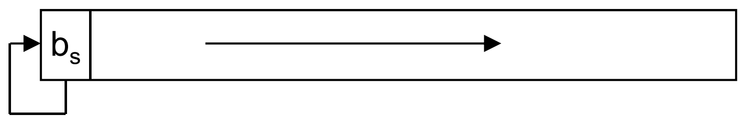
O= 1000 0100 = -124_{10} x2 → D= 0000 1000 = 8_{10} → incorrecto

Negativos en complemento a 2:
1 ←→ 0 y +1

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

DESPLAZAMIENTOS ARITMÉTICOS

- Desplazamiento a la derecha (división por 2):
  - Hay que conservar el signo del dato, hay que recircularlo.
  - Hay que introducir ceros por la izquierda para números positivos
  - Para números negativos hay que introducir unos



Desplazamiento aritmético a la derecha

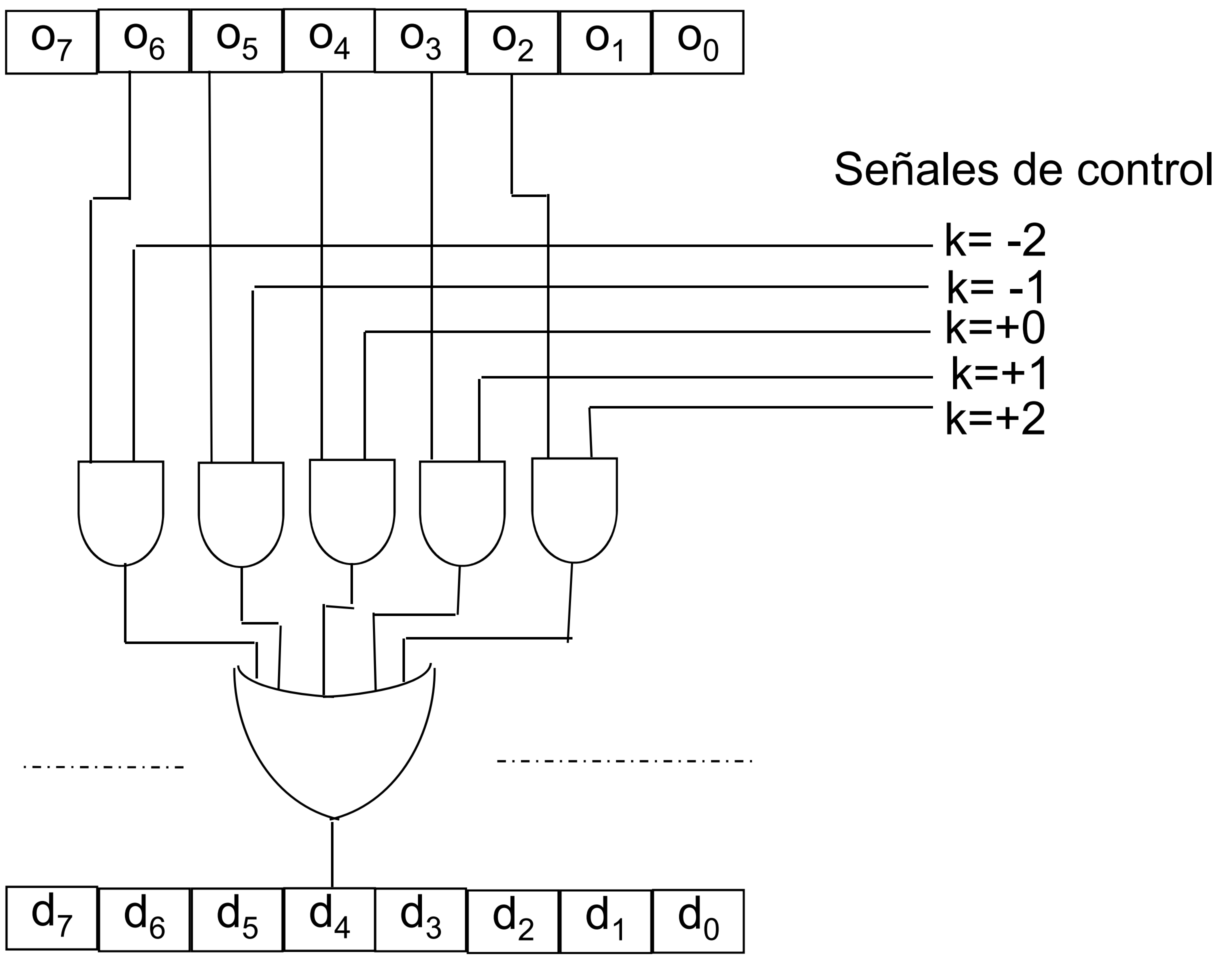
O= 0000 1010 = 10_{10} /2 → D= 0000 0101 = 5_{10}

O= 1111 1100 = -4_{10} /2 → D= 1111 1110 = -2_{10}

Puertas lógicas

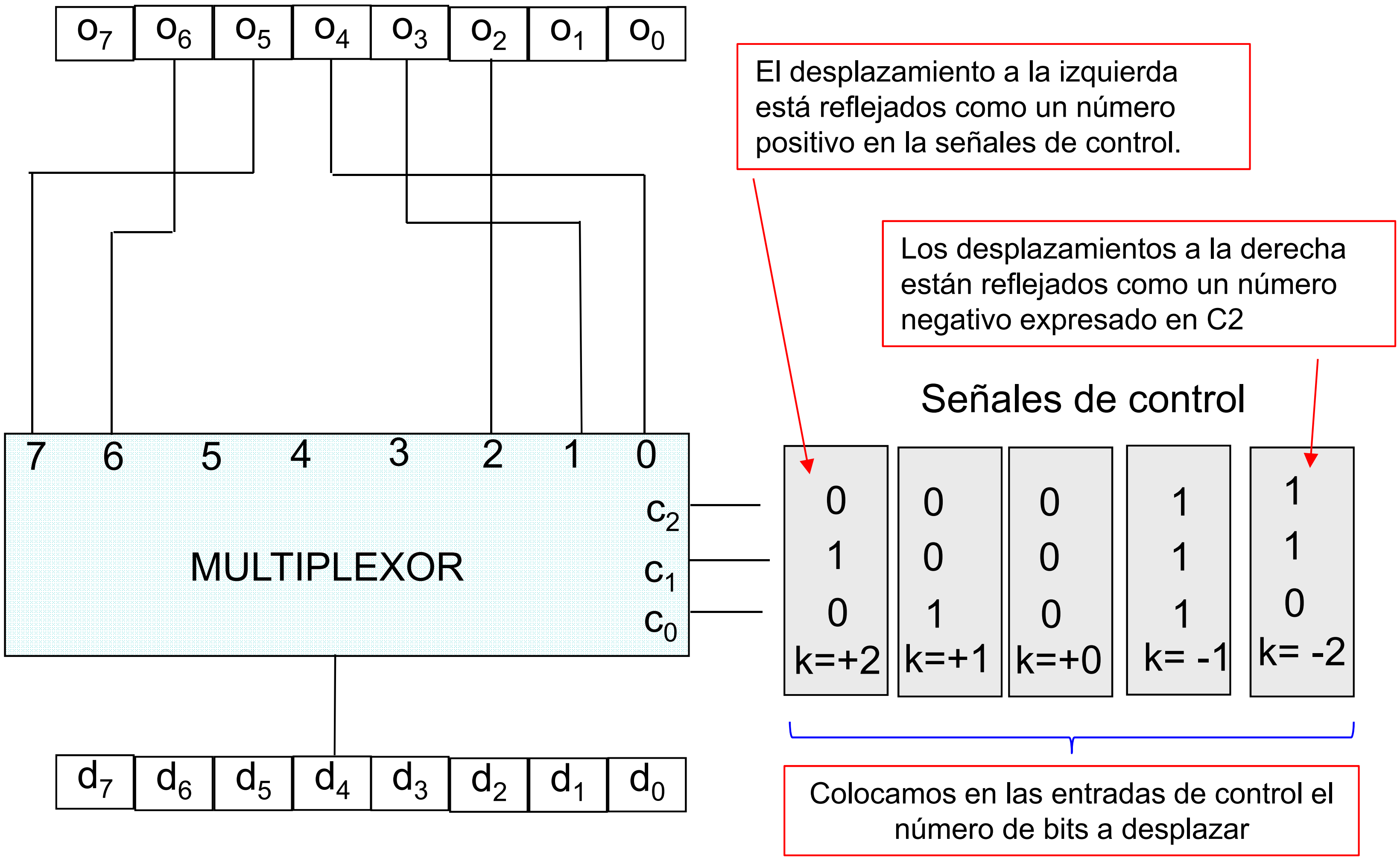
- La complejidad es elevada.
- Las señales de control son las mismas para cada bit.

Unidad de desplazamiento de 2 bits para el bit 4



Multiplexores

Unidad de desplazamiento de 2 bits para el bit 4



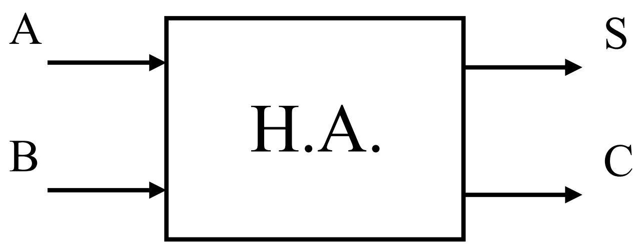
Unidad aritmética entera. Sumar y restar

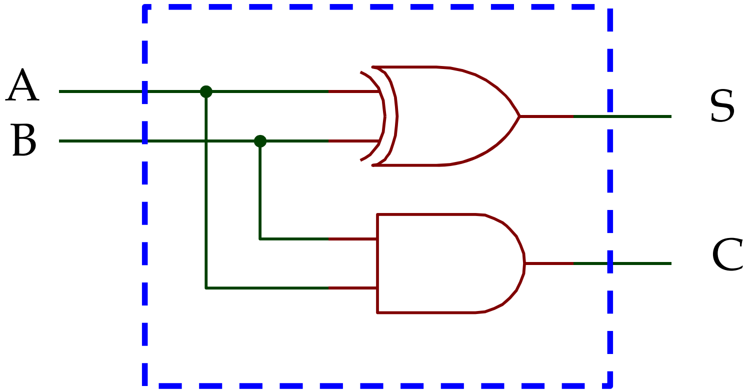
- Semisumador Binario (H.A.)

Entradas		Salidas	
A	**B**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \bar{A}B + A\bar{B} = A \oplus B$$

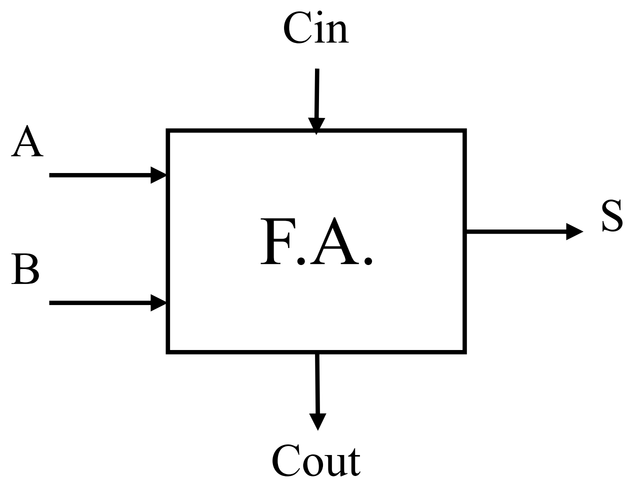
$$C = AB$$





SUMADOR COMPLETO (F.A.)

- Sumador Binario (F.A.)



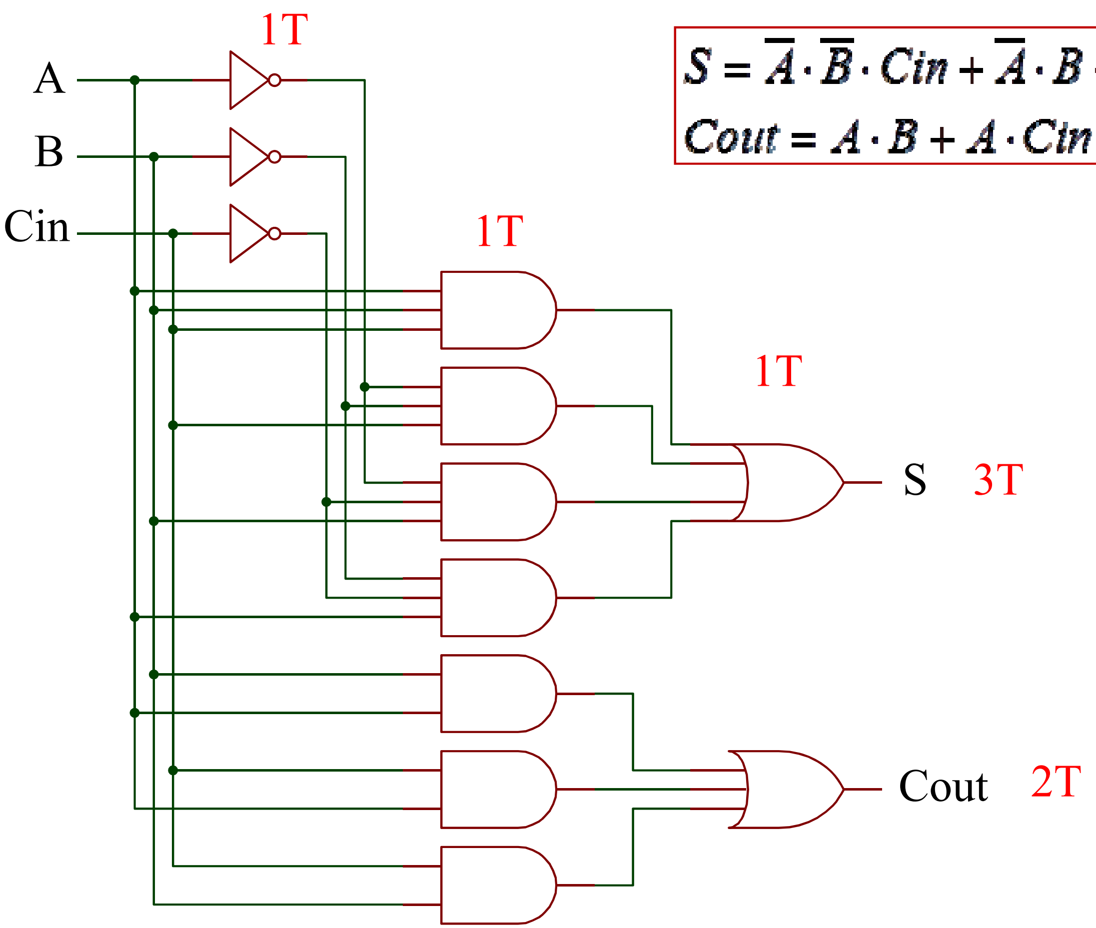
Entradas			Salidas	
A	**B**	**Cin**	**S**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin = (A \oplus B) \oplus Cin$$

$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin = A \cdot B + Cin(A \oplus B)$$

Unidad aritmética entera. Sumar y restar

Sumador completo (FA) – Puertas lógicas (expresión booleana)

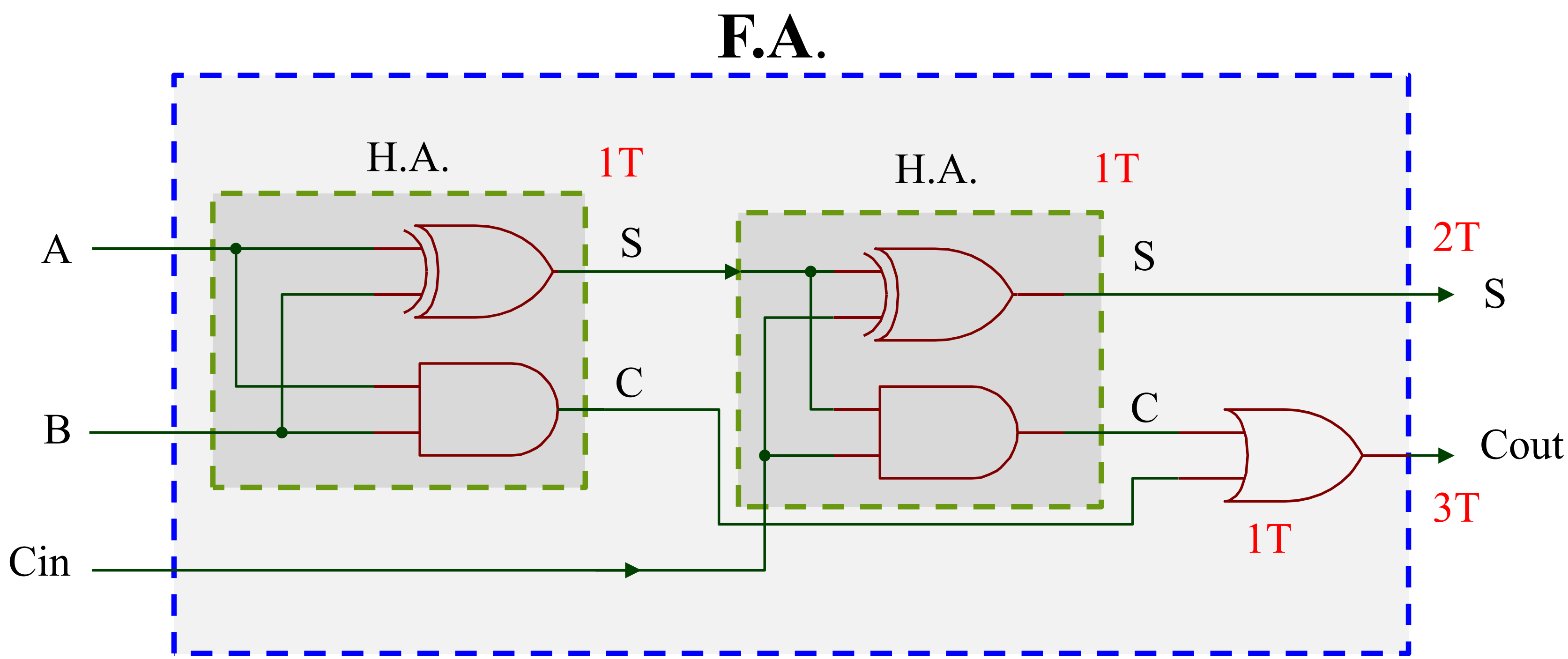


$$S = \overline{A}\cdot\overline{B}\cdot Cin + \overline{A}\cdot B\cdot\overline{Cin} + A\cdot\overline{B}\cdot\overline{Cin} + A\cdot B\cdot Cin$$

$$Cout = A\cdot B + A\cdot Cin + B\cdot Cin$$

Unidad aritmética entera. Sumar y restar

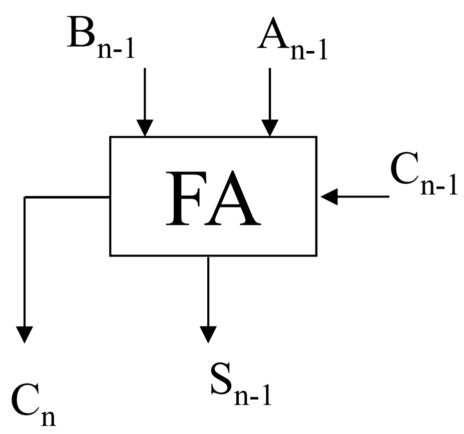
- Sumador completo (FA) - Semisumadores

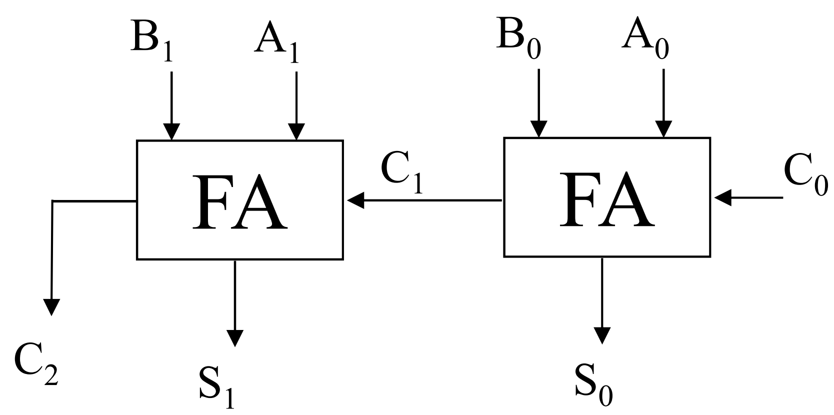


SUMADOR CON PROPAGACIÓN DE ACARREO (CPA)

- La estructura para sumar dos números de ***n*** bits es colocar en cascada ***n*** sumadores completos.

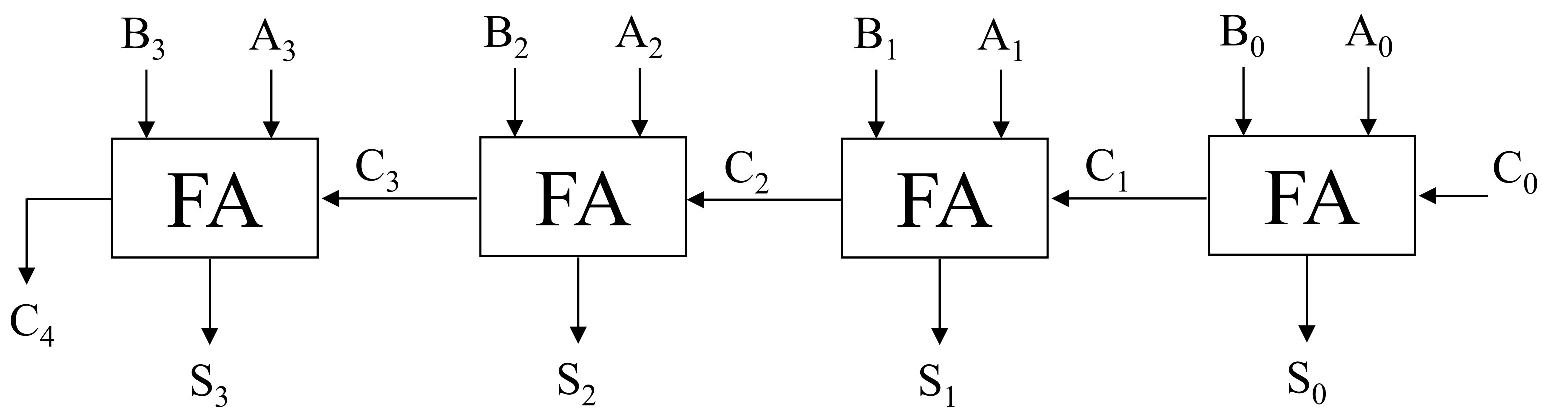
- El acarreo se propaga de una etapa a la siguiente: Sumador con Propagación de Acarreo (*Carry Propagated Adder*)





Unidad aritmética entera. Sumar y restar

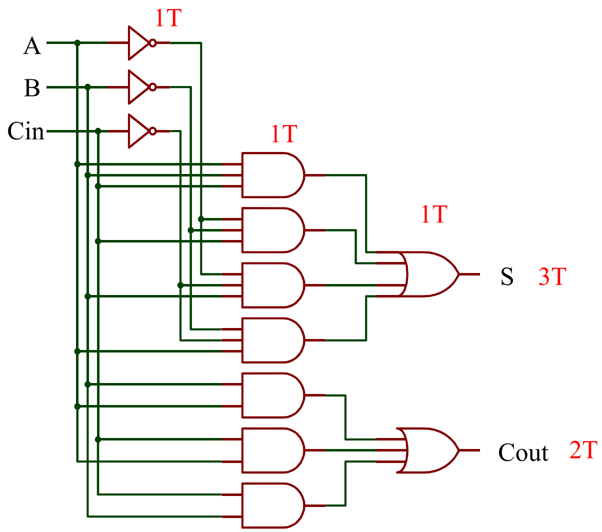
- Sumador con propagación de acarreo de 4 bits.



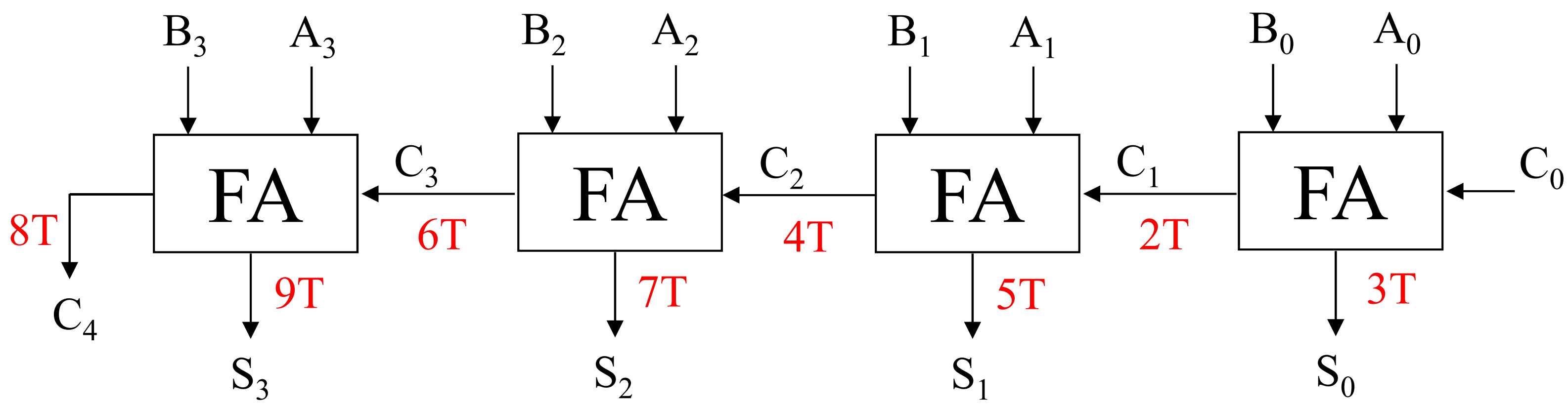
- Sumadores construidos con sumadores completos a partir de puertas lógicas:

$$S = \overline{A}\cdot\overline{B}\cdot Cin + \overline{A}\cdot B\cdot\overline{Cin} + A\cdot\overline{B}\cdot\overline{Cin} + A\cdot B\cdot Cin$$

$$Cout = A\cdot B + A\cdot Cin + B\cdot Cin$$

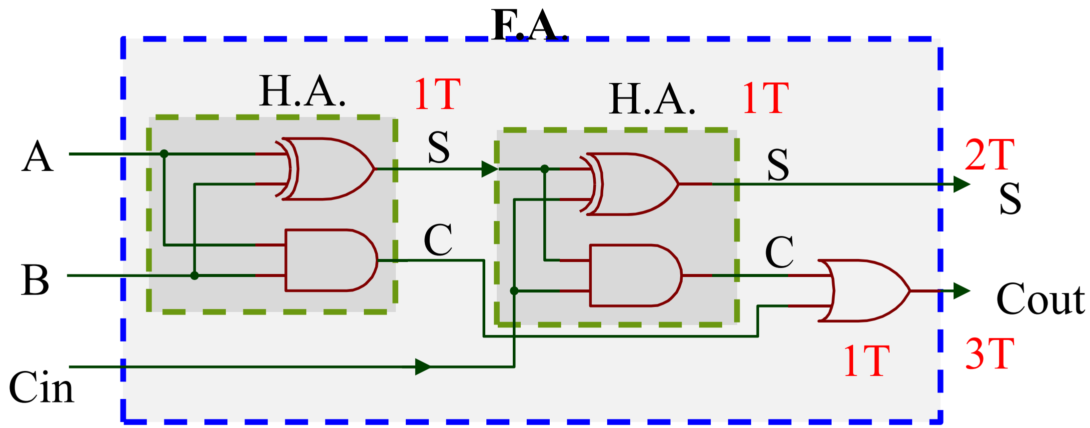


$$\text{Tiempo Total} = (2n+1)T$$

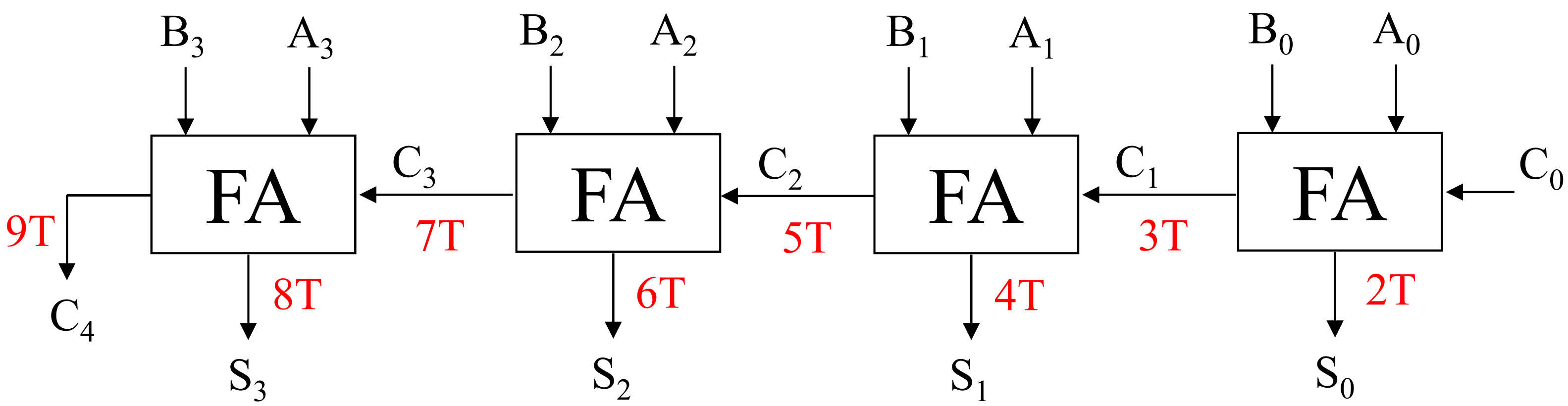


Unidad aritmética entera. Sumar y restar

- Sumadores completos construidos con semisumadores:

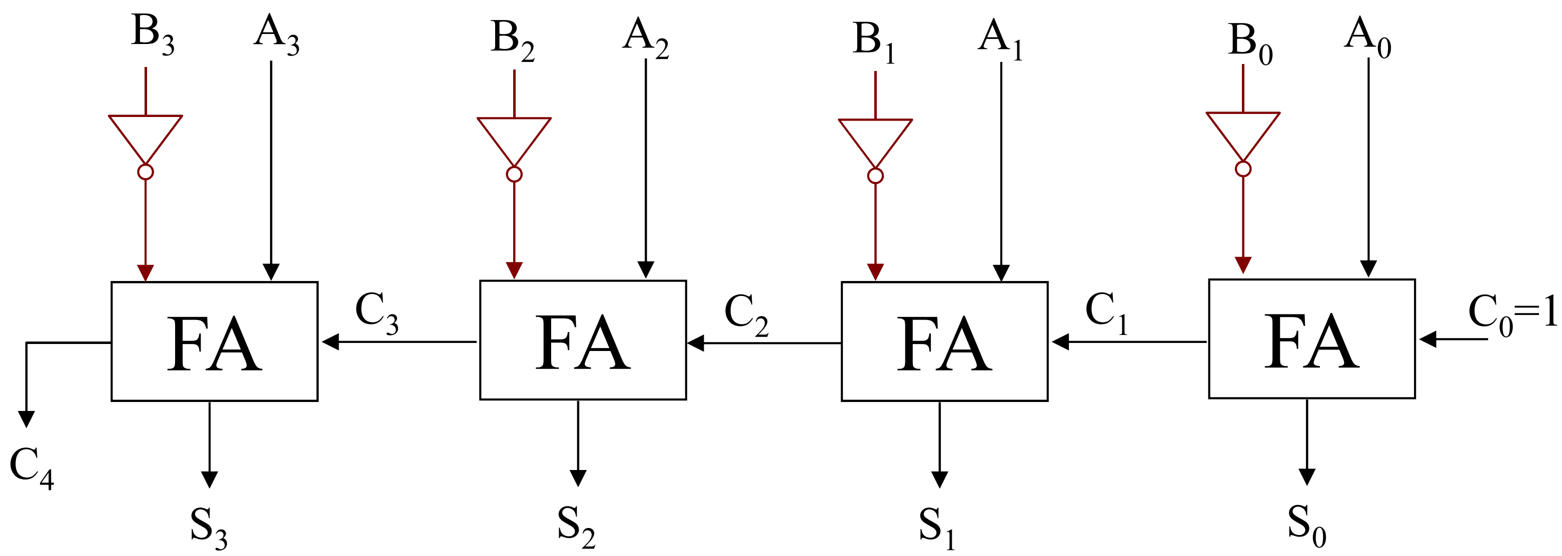


$$\text{Tiempo Total} = (2n+1)T$$



Unidad aritmética entera. Sumar y restar

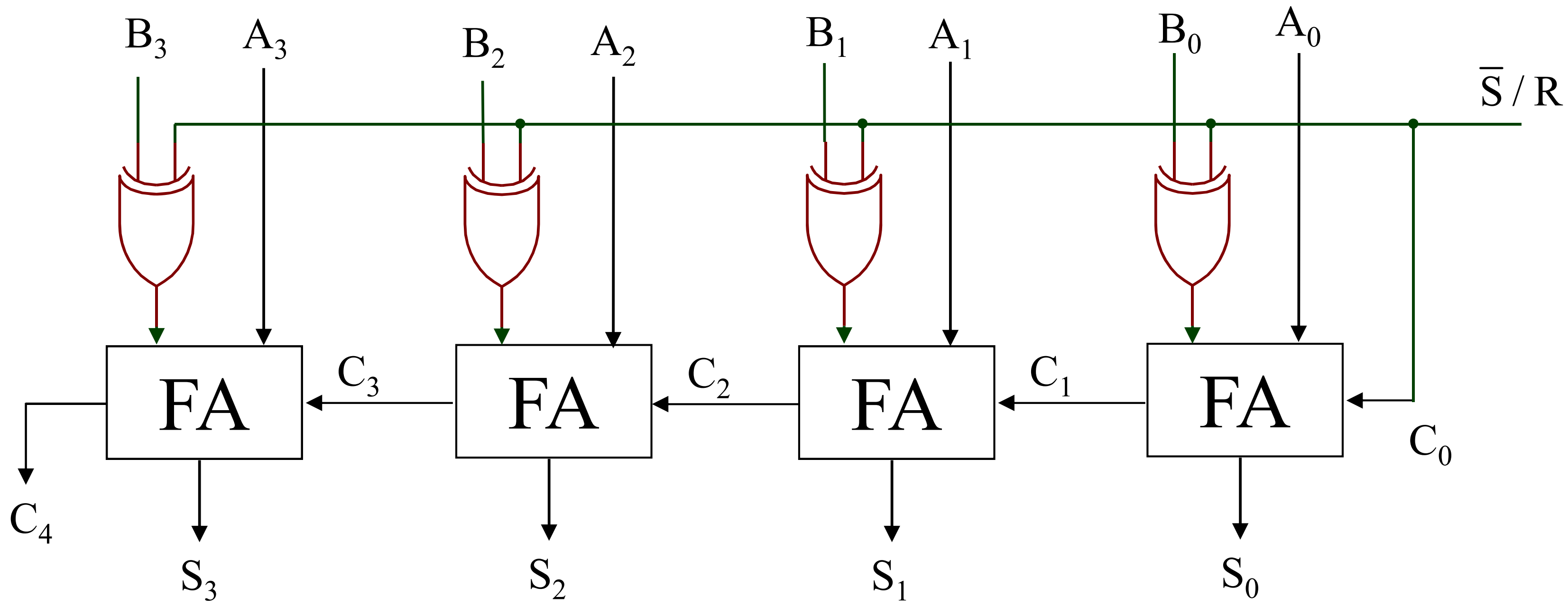
- Puesto que se trabaja con números expresados en complemento a 2 (C2) → C2(B) = C1(B) + 1.
- A - B = A + (C1(B) + 1)



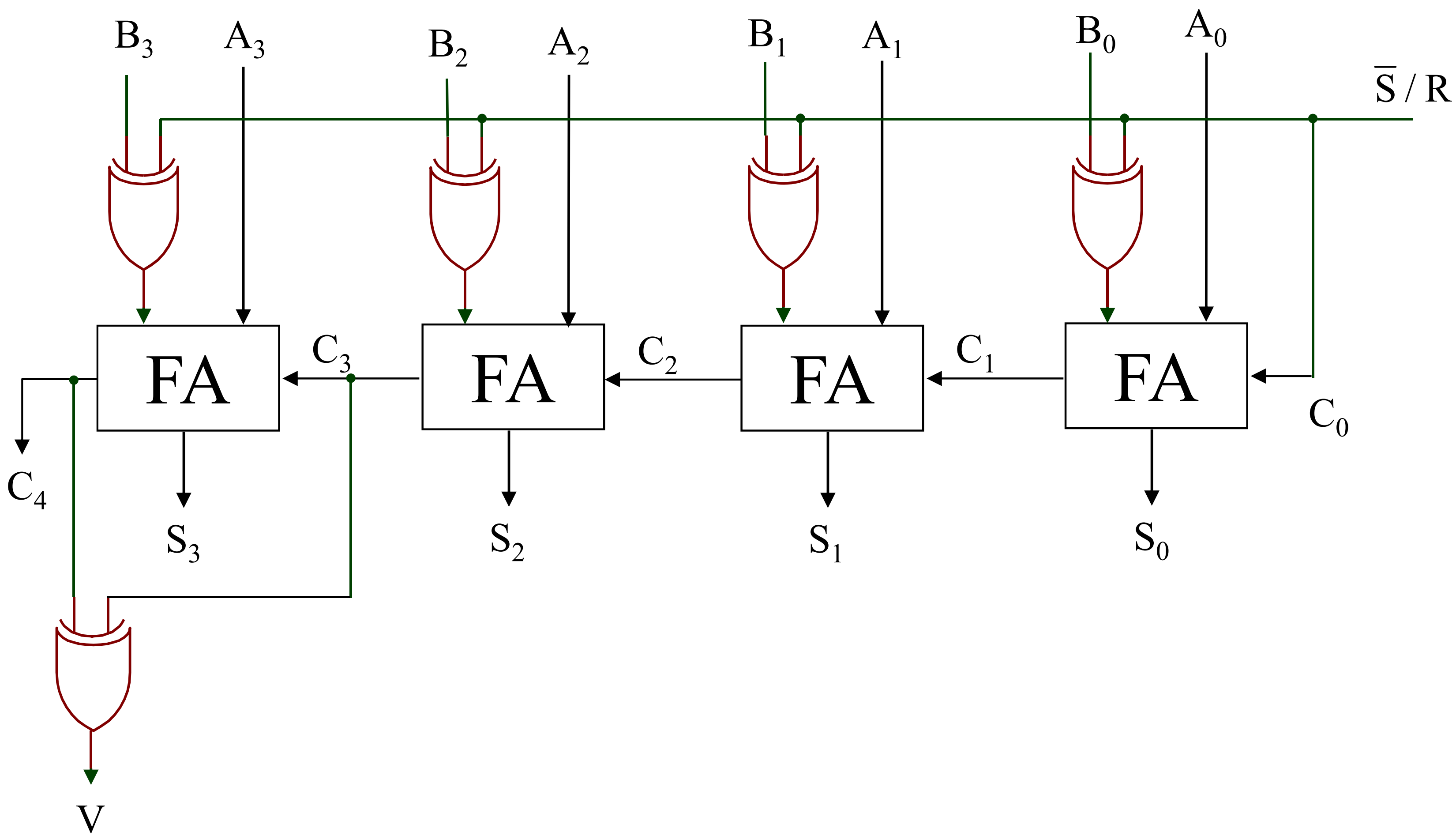
CIRCUITO SUMADOR-RESTADOR

$\overline{S}/R$	B_i	Entrada al FA
0	0	0
0	1	1
1	0	1
1	1	0

$$\text{Tiempo Total} = 2(n+1)T$$



Sumador-Restador en C2 con detección de desbordamiento.



Detección de desbordamiento

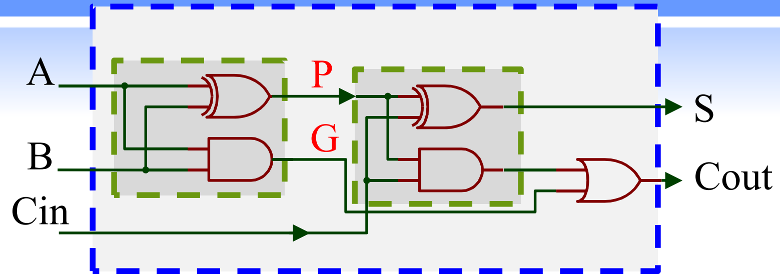
1. Caso suma de dos positivos

C4	C3	C2	C1		
0	1	1	1		
	0	1	1	1	(+7)
	0	1	1	1	(+7)
	1	1	1	0	¿ -2 ? → OV

2. Caso suma de dos negativos

C4	C3	C2	C1		
1	0	1	1		
	1	0	0	1	(-7)
	1	0	1	1	(-5)
	0	1	0	0	¿ 4 ? → OV

SUMADOR CON ANTICIPACIÓN DE ACARREO (CLA)

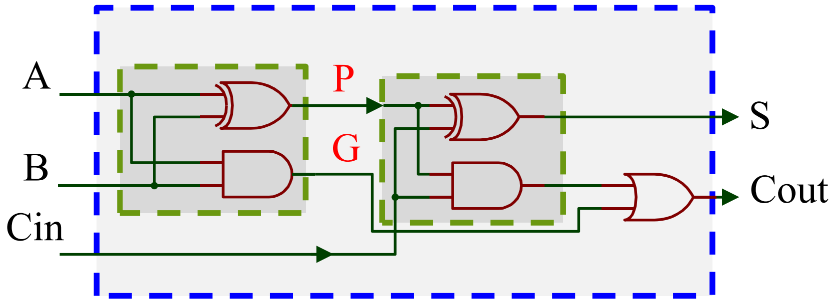


- Carry Lookahead Adder (CLA)
- Suponer A y B números de 4 bits
- Señal generadora de acarreo : $G_i = A_i \cdot B_i$
- Señal propagadora de acarreo: $\begin{cases} P_i = A_i \oplus B_i \\ P_i = A_i + B_i \end{cases}$
- Acarreo de la etapa i: $C_i = G_i + P_i \cdot C_{i-1}$
- Particularizando para A y B:

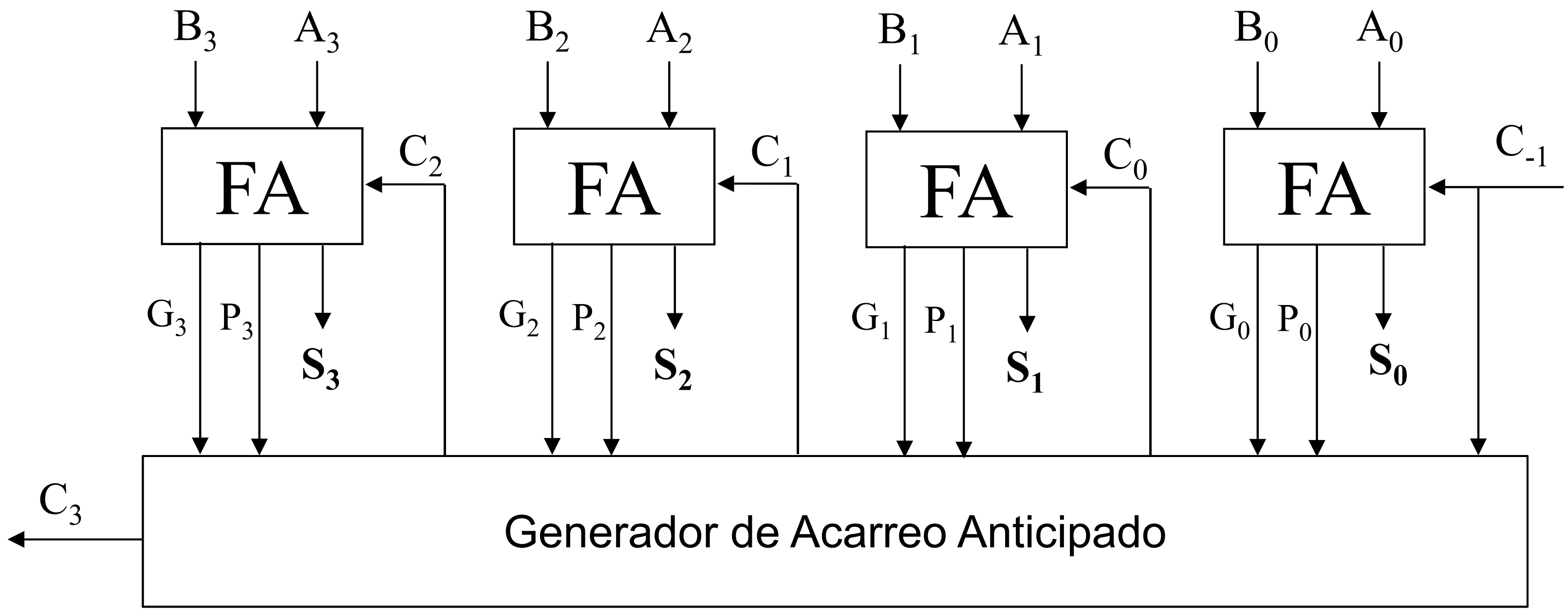
$$C_0 = G_0 + P_0 \cdot C_{-1}$$
$$C_1 = G_1 + P_1 \cdot C_0$$
$$C_2 = G_2 + P_2 \cdot C_1$$
$$C_3 = G_3 + P_3 \cdot C_2$$

- Desarrollando las expresiones y poniéndolas en función de C_{-1}:

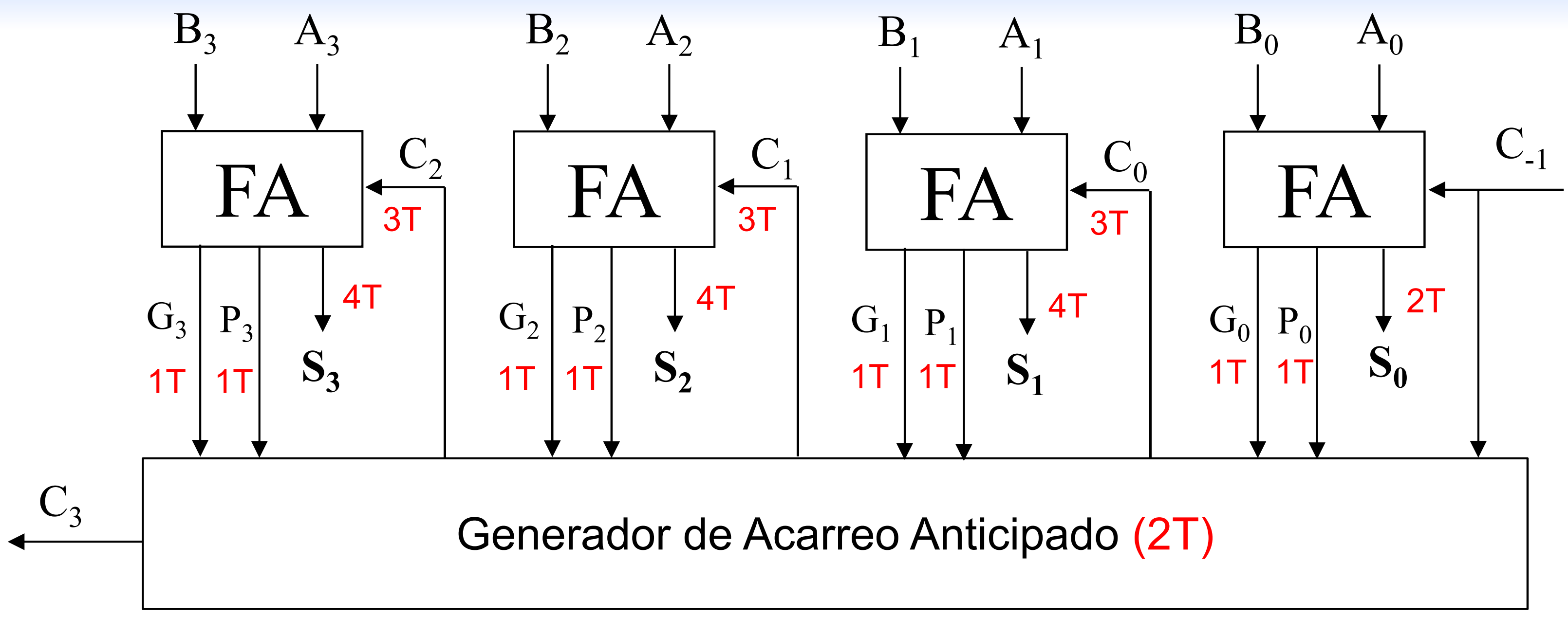
$$C_0 = G_0 + P_0 \cdot C_{-1}$$
$$C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_{-1}$$
$$C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$
$$C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

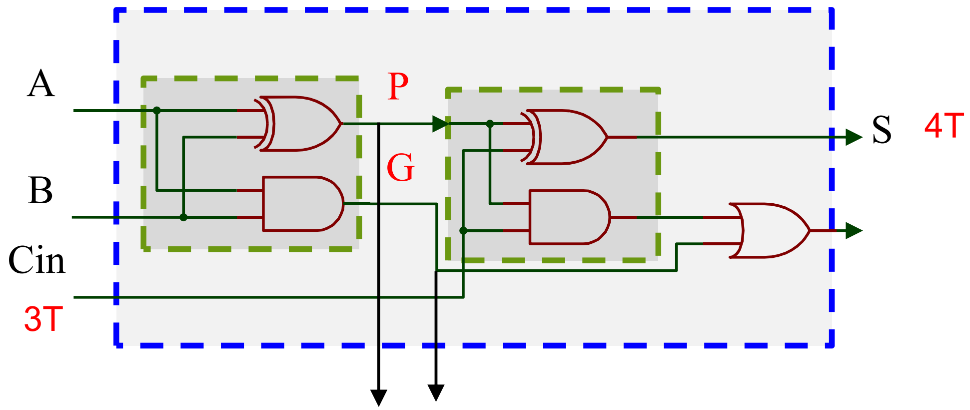


- Todos los acarreos dependen de A_i y B_i (y de C_{-1}).
- Estas expresiones se resuelven como suma de productos.
- Tres niveles de puertas lógicas para obtener cada uno de los acarreos.


B3
A3
B2
A2
B1
A1
B0
A0
FA
FA
FA
FA
C2
C1
C0
C-1
G3
P3
S3
G2
P2
S2
G1
P1
S1
G0
P0
S0
C3
Generador de Acarreo Anticipado

SUMADOR CON ANTICIPACIÓN DE ACARREO (CLA)

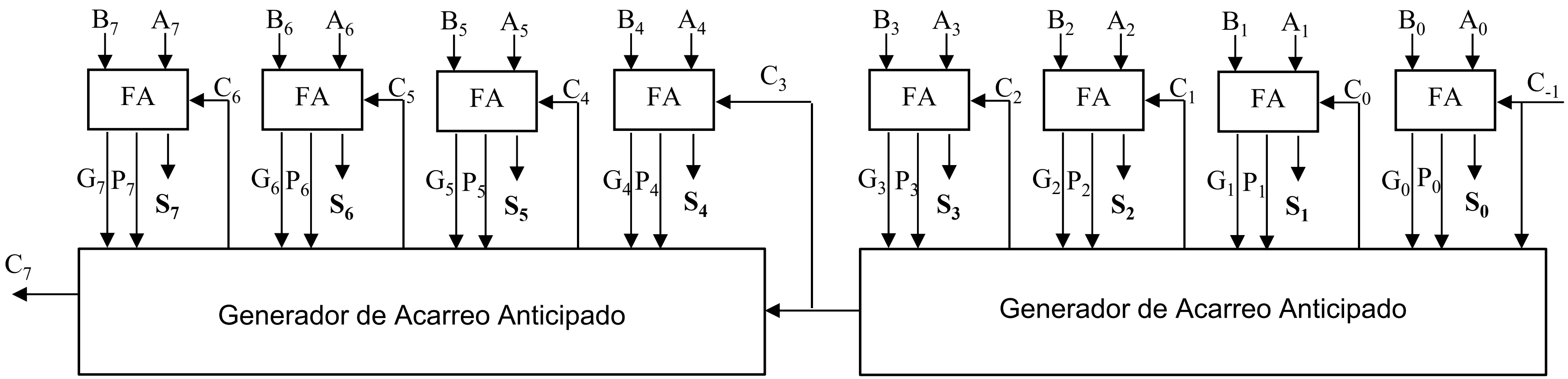




Sumadores construidos con semisumadores

http://www.ecs.umass.edu/ece/koren/arith/simulator/Add/lookahead/

SUMADOR CON ANTICIPACIÓN DE ACARREO (CLA): EJEMPLO DE 8 BITS



Calcula los retardos en este CLA suponiendo que los sumadores se construyen con semisumadores.

Compara el resultado con el de un sumador CPA de 8 bits.

LA MULTIPLICACIÓN

- Algoritmo de sumas y desplazamientos
- Si el multiplicando es de *n* bits y el multiplicador de *m* bits, entonces el producto tendrá una longitud de *n*+*m* bits.
- Multiplicación binaria: sencilla ya que hay que multiplicar por 1 o por 0.

Multiplicando				5	3	2
Multiplicador				4	3	1
				5	3	2
		1	5	9	6	
	2	1	2	8		
Producto	2	2	9	2	9	2

	1	0	0
		1	0
	0	0	0
1	0	0	
1	0	0	0

Repetir n veces

Si el bit 0 del multiplicador=1 entonces

Sumar el multiplicando a la mitad izquierda del producto y colocar el resultado en la mitad izquierda del producto.

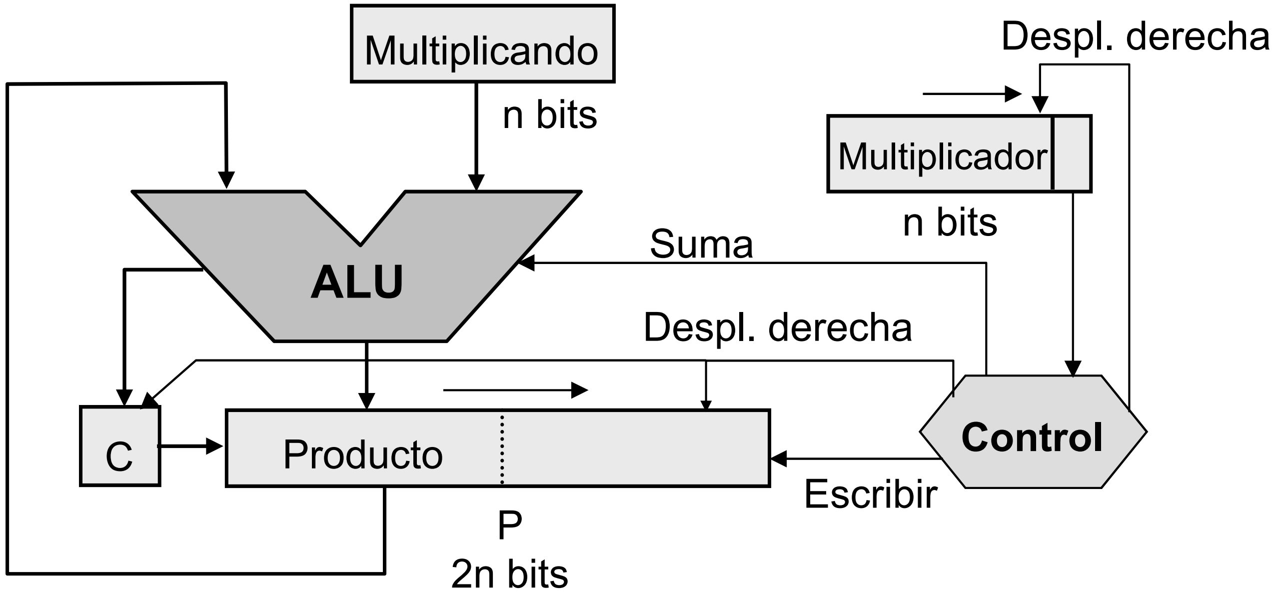
Fin entonces

Desplazar 1 bit a la derecha el registro producto

Desplazar 1 bit a la derecha el registro multiplicador

Fin repetir

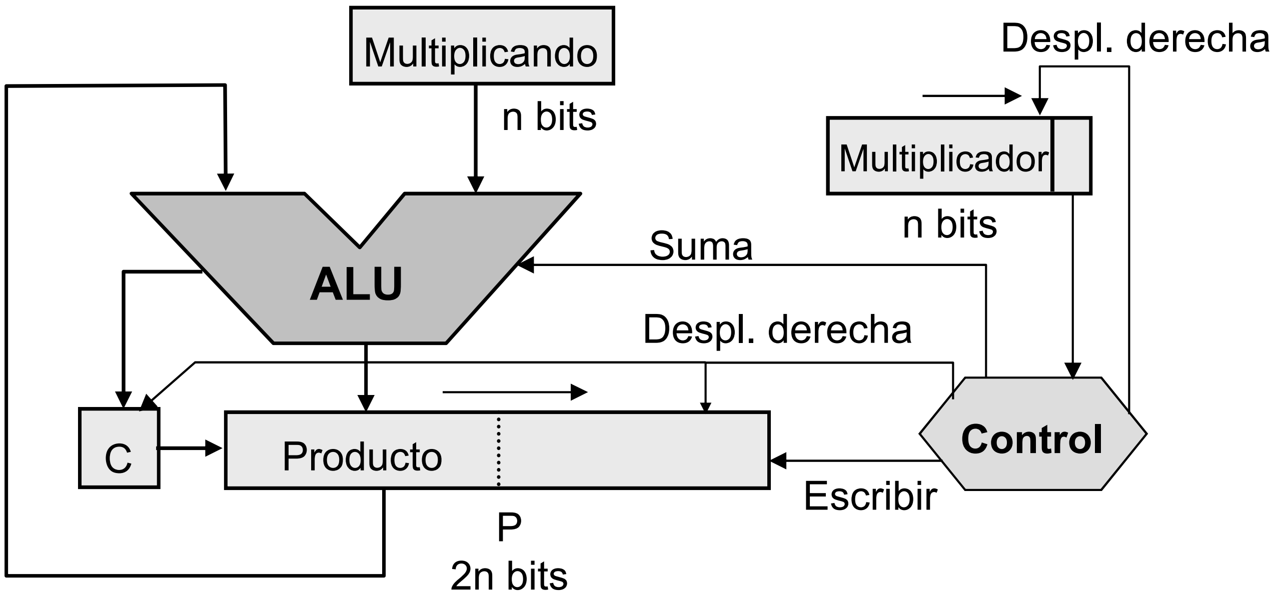
Versión preliminar



MULTIPLICACIÓN BINARIA SIN SIGNO

Multiplicando					1	0	1	1	11_d
Multiplicador					1	1	0	1	13_d
					1	0	1	1	
				0	0	0	0		
			1	0	1	1			
		1	0	1	1				
Producto	1	0	0	0	1	1	1	1	143_d

Versión preliminar

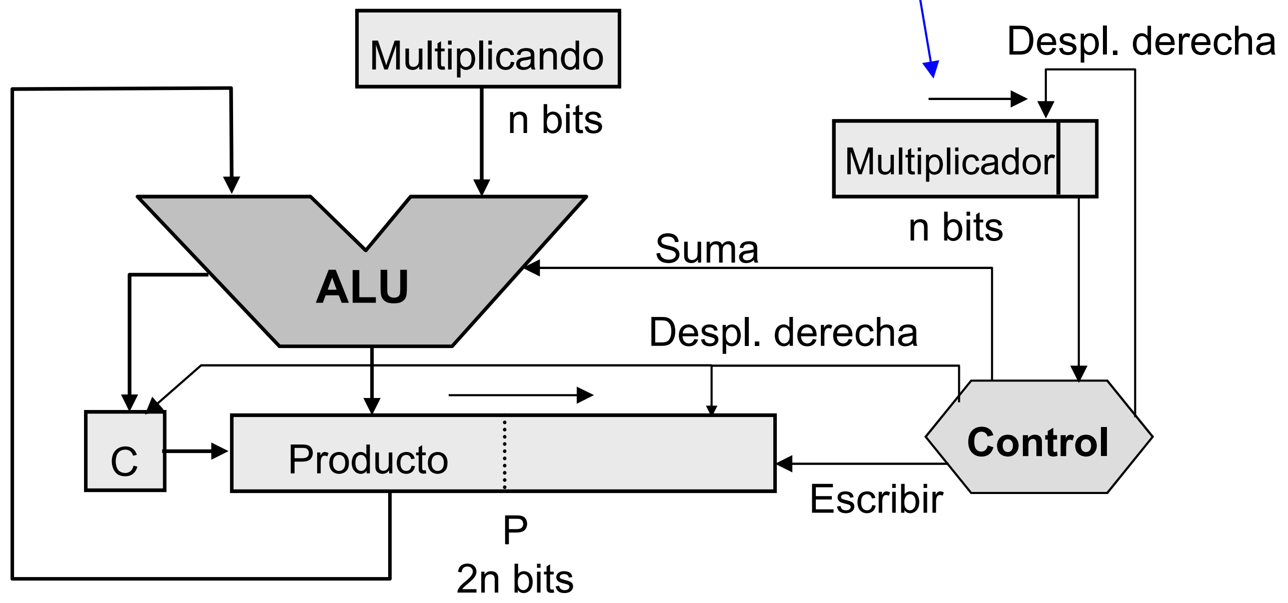


MULTIPLICACIÓN BINARIA SIN SIGNO

Multiplicando					1	0	1	1	11_d
Multiplicador					1	1	0	1	13_d
					1	0	1	1	
				0	0	0	0		
			1	0	1	1			
		1	0	1	1				
Producto	1	0	0	0	1	1	1	1	143_d

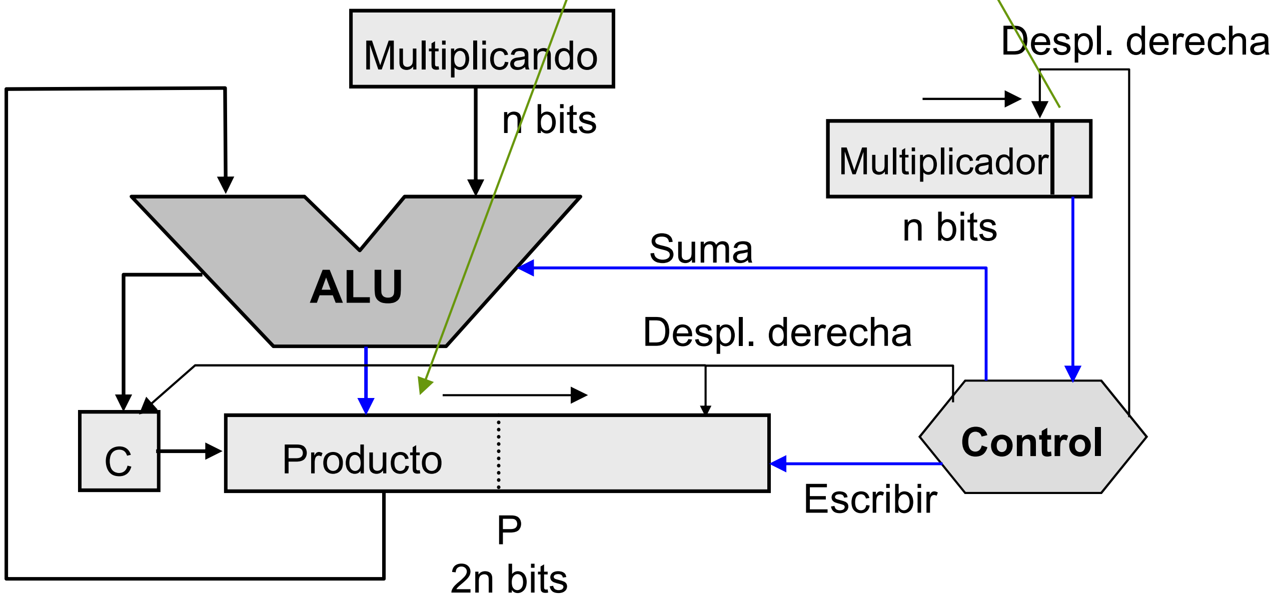
Versión preliminar

Cargamos el *multiplicando* y el multiplicador en los registros correspondientes y ponemos el registro *Producto* a cero



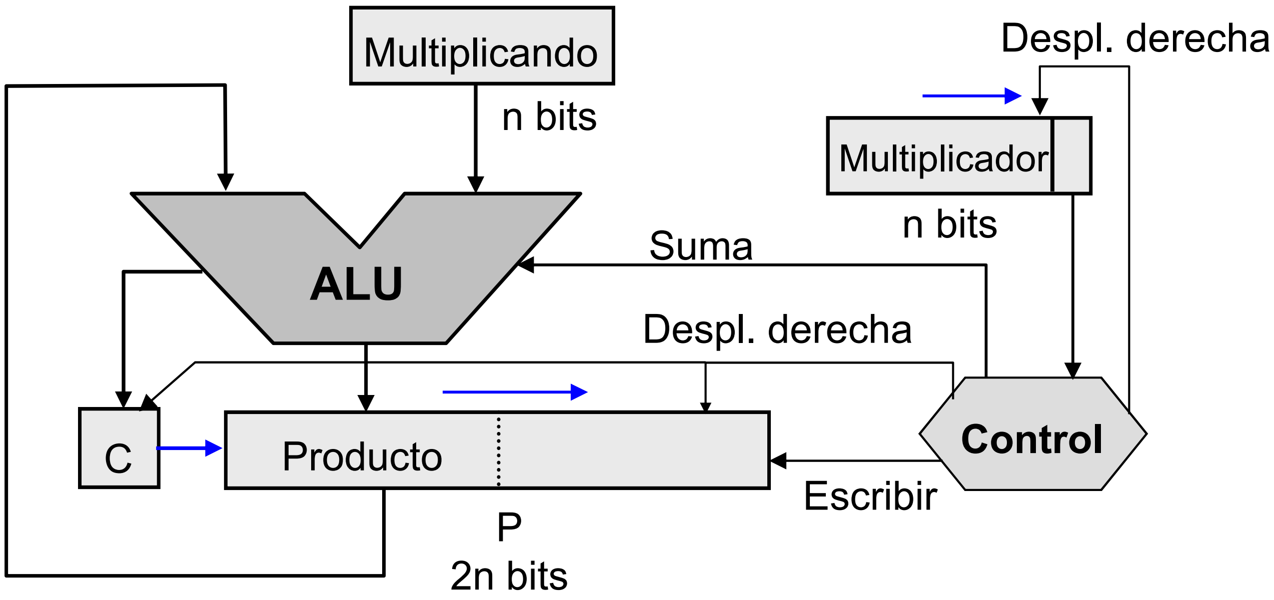
Multiplicando		1	0	1	1
Multiplicador		1	1	0	1
		1	0	1	1

1. Como el bit de la derecha de *Multiplicador* es 1, se suma *Multiplicando* y *Producto*. El resultado queda en *producto*.



Multiplicando	1	0	1	1	
Multiplicador	0	1	1	0	
	0	1	0	1	1

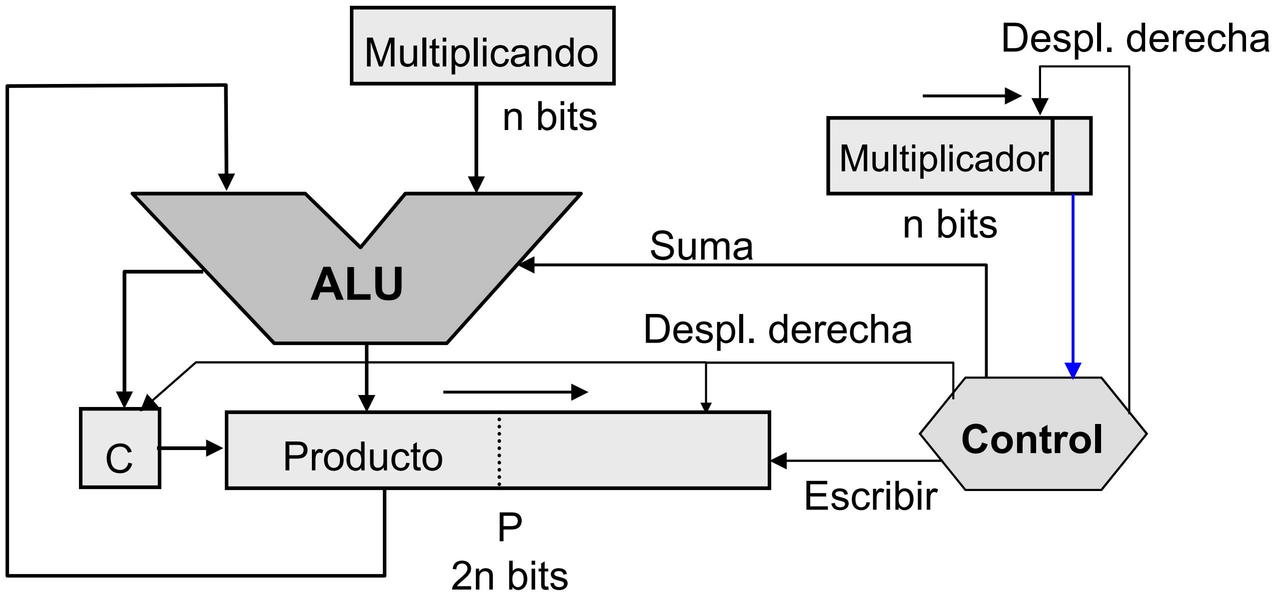
2. Desplazamos a la derecha *Multiplicador* y *Producto*.



Unidad aritmética entera. Multiplicar y dividir

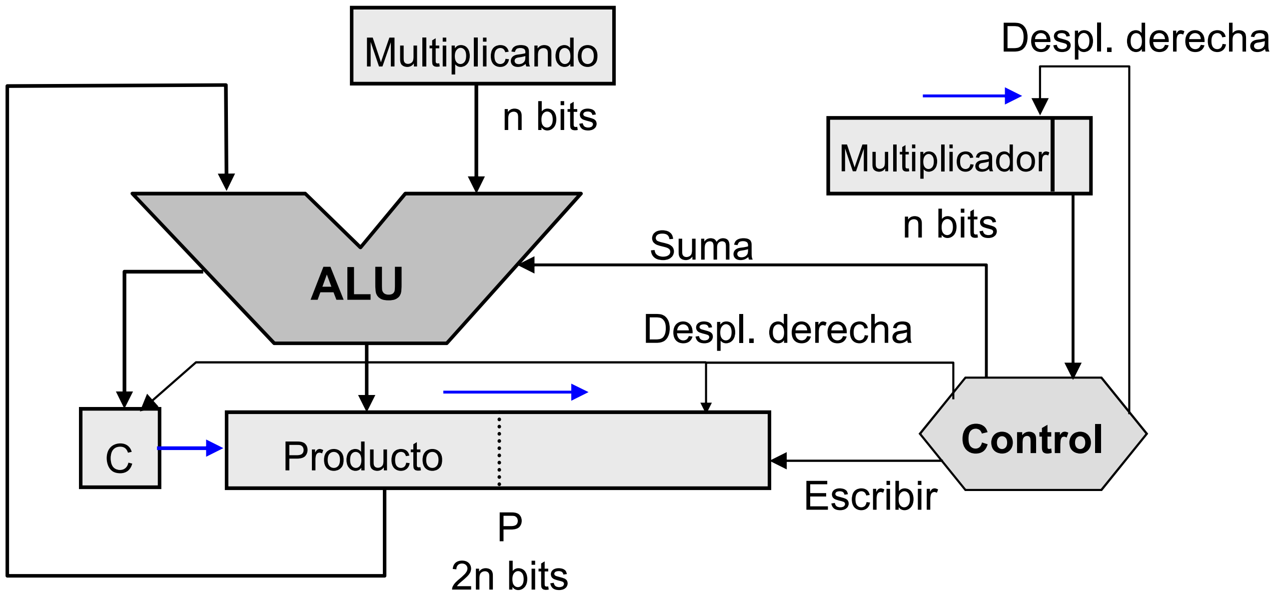
Multiplicando		1	0	1	1
Multiplicador		0	1	1	0
		0	1	0	1	1

3. El bit de la derecha de *Multiplicador* es 0. No se realiza la suma.



Multiplicando	1 0 1 1
Multiplicador	0 0 1 1
	0 0 1 0 1 1

4. Desplazamos a la derecha *Multiplicador* y *Producto*.

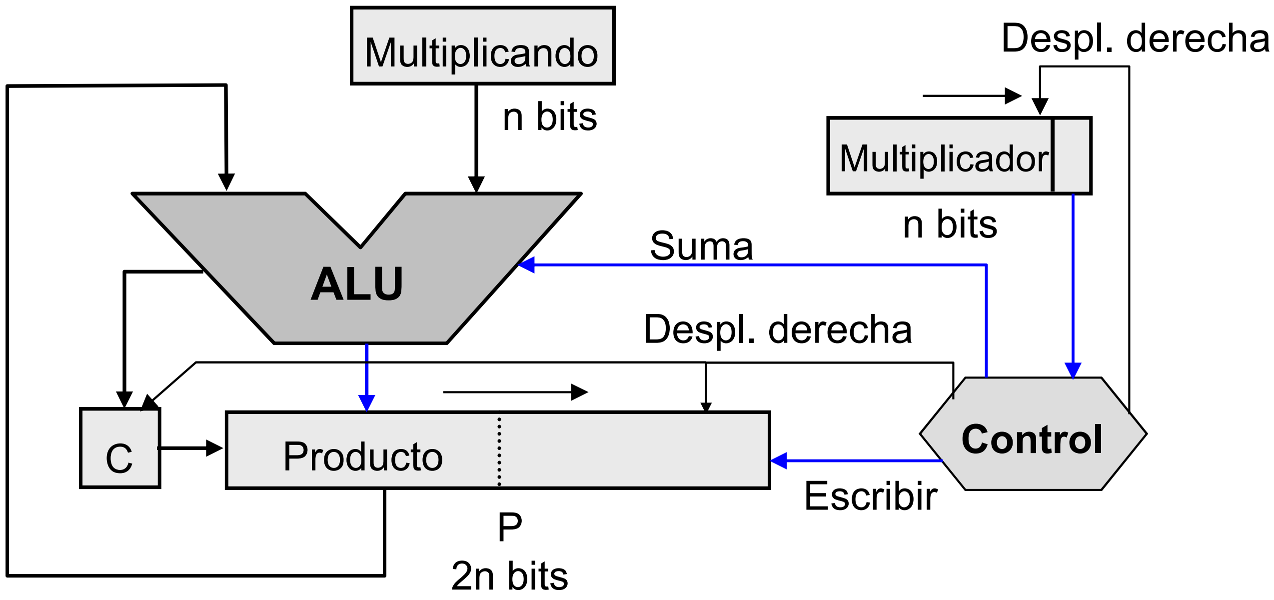


MULTIPLICACIÓN BINARIA SIN SIGNO

Unidad aritmética entera. Multiplicar y dividir

Multiplicando			1	0	1	1
Multiplicador			0	0	1	1
	0	0	1	0	1	1
	1	0	1	1		
	1	1	0	1	1	1

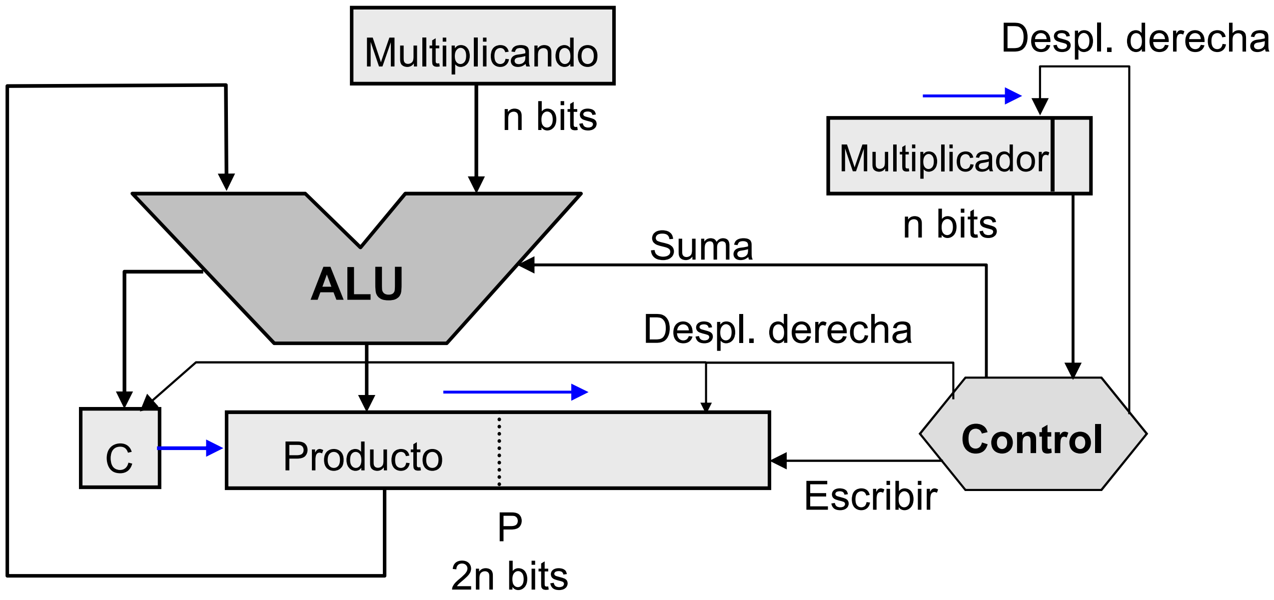
5. Como el bit de la derecha de *Multiplicador* es 1, se suma *Multiplicando* y *Producto*. El resultado queda en *producto*.



MULTIPLICACIÓN BINARIA SIN SIGNO

Multiplicando				1	0	1	1
Multiplicador				0	0	0	1
	0	1	1	0	1	1	1

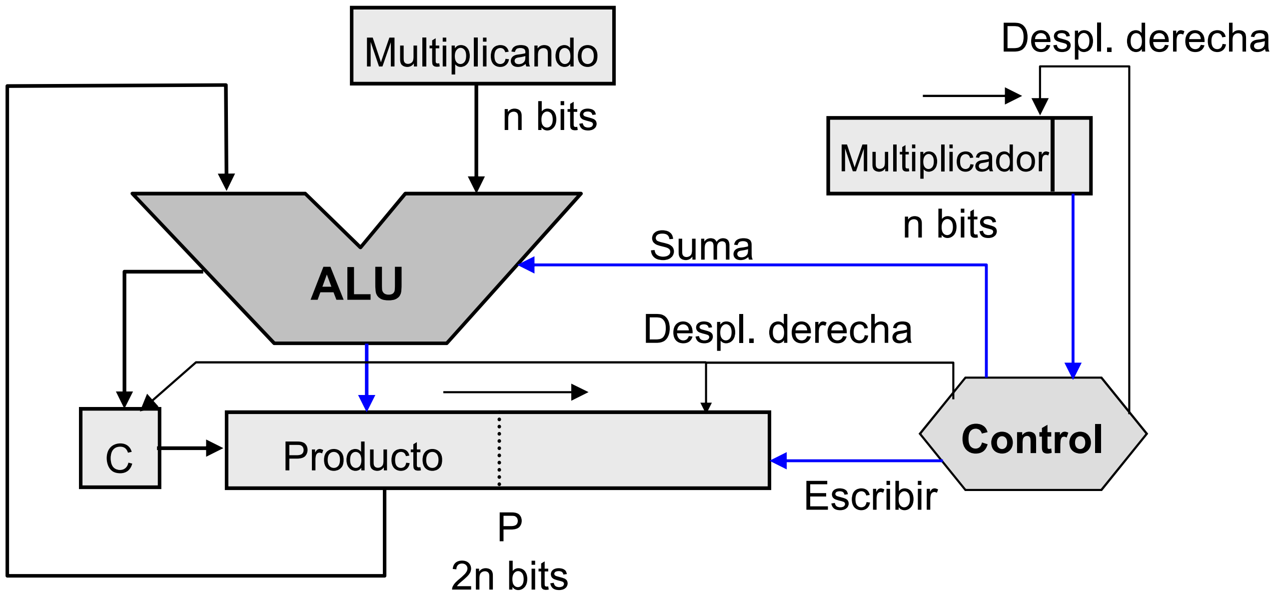
6. Desplazamos a la derecha *Multiplicador* y *Producto*.



Unidad aritmética entera. Multiplicar y dividir

Multiplicando			1	0	1	1		
Multiplicador			0	0	0	1		
		0	1	1	0	1	1	1
		1	0	1	1			
	1	0	0	0	1	1	1	1

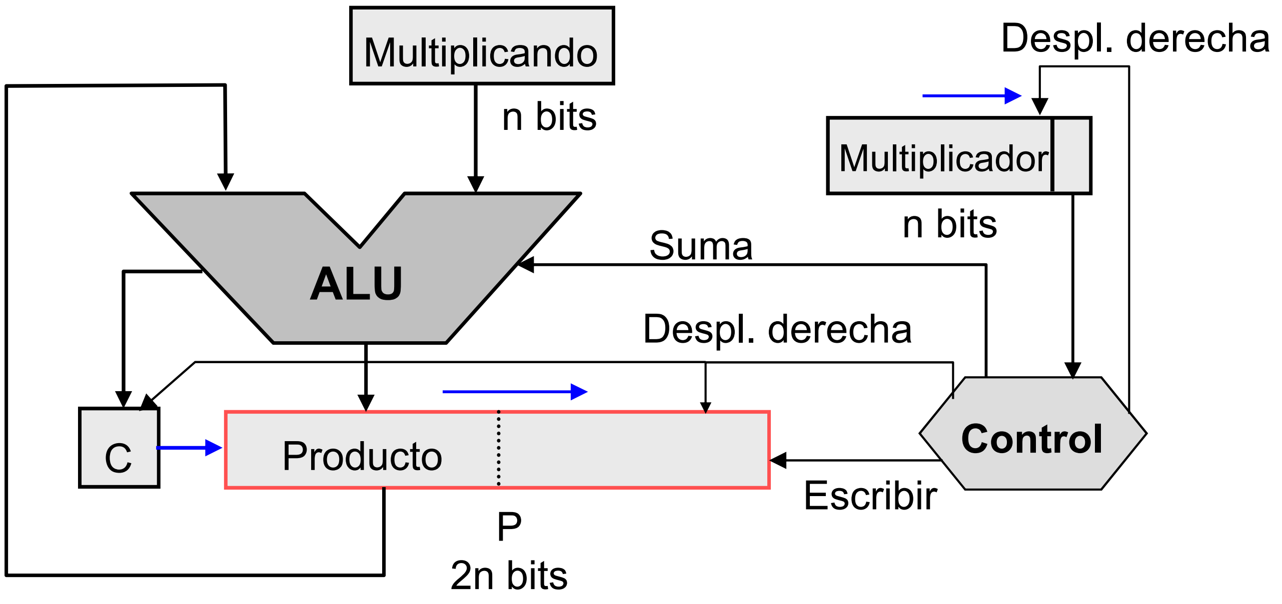
7. Como el bit de la derecha de *Multiplicador* es 1, se suma *Multiplicando* y *Producto*. El resultado queda en *producto*.



Unidad aritmética entera. Multiplicar y dividir

Multiplicando					1	0	1	1
Multiplicador					0	0	0	0
	1	0	0	0	1	1	1	1

8. Desplazamos a la derecha *Multiplicador* y *Producto*.
El resultado final queda en el registro P de 2n bits.



Repetir n veces

Si el bit 0 del registro producto=1 entonces

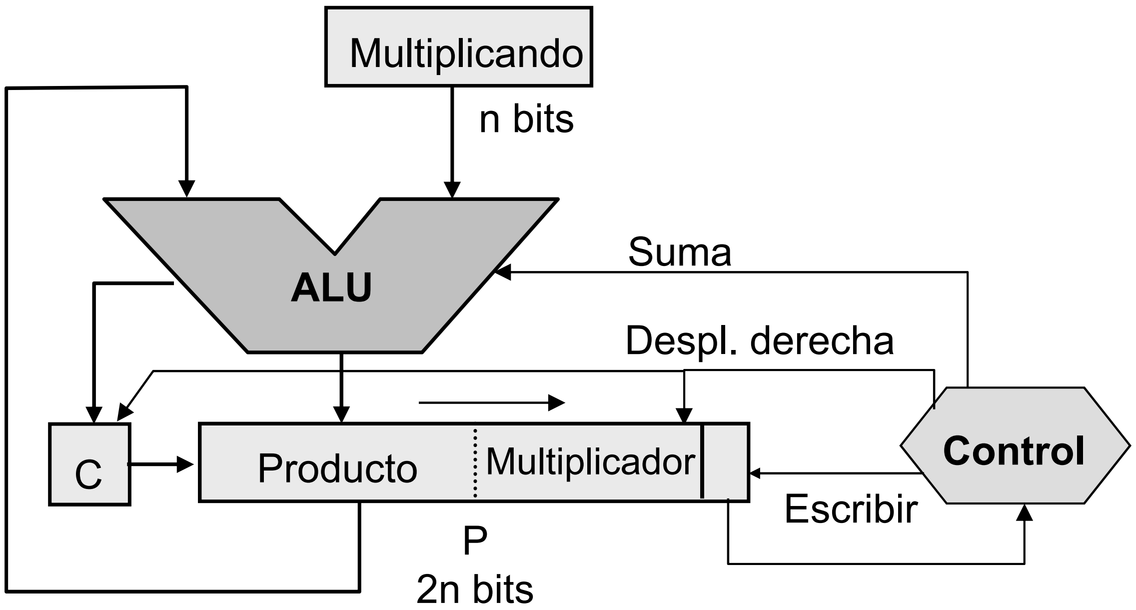
Sumar el multiplicando a la mitad izquierda del producto ($prod_H$) y colocar el resultado en la mitad izquierda del producto → **$prod_H$=$prod_H$ + multiplicando**

Fin entonces

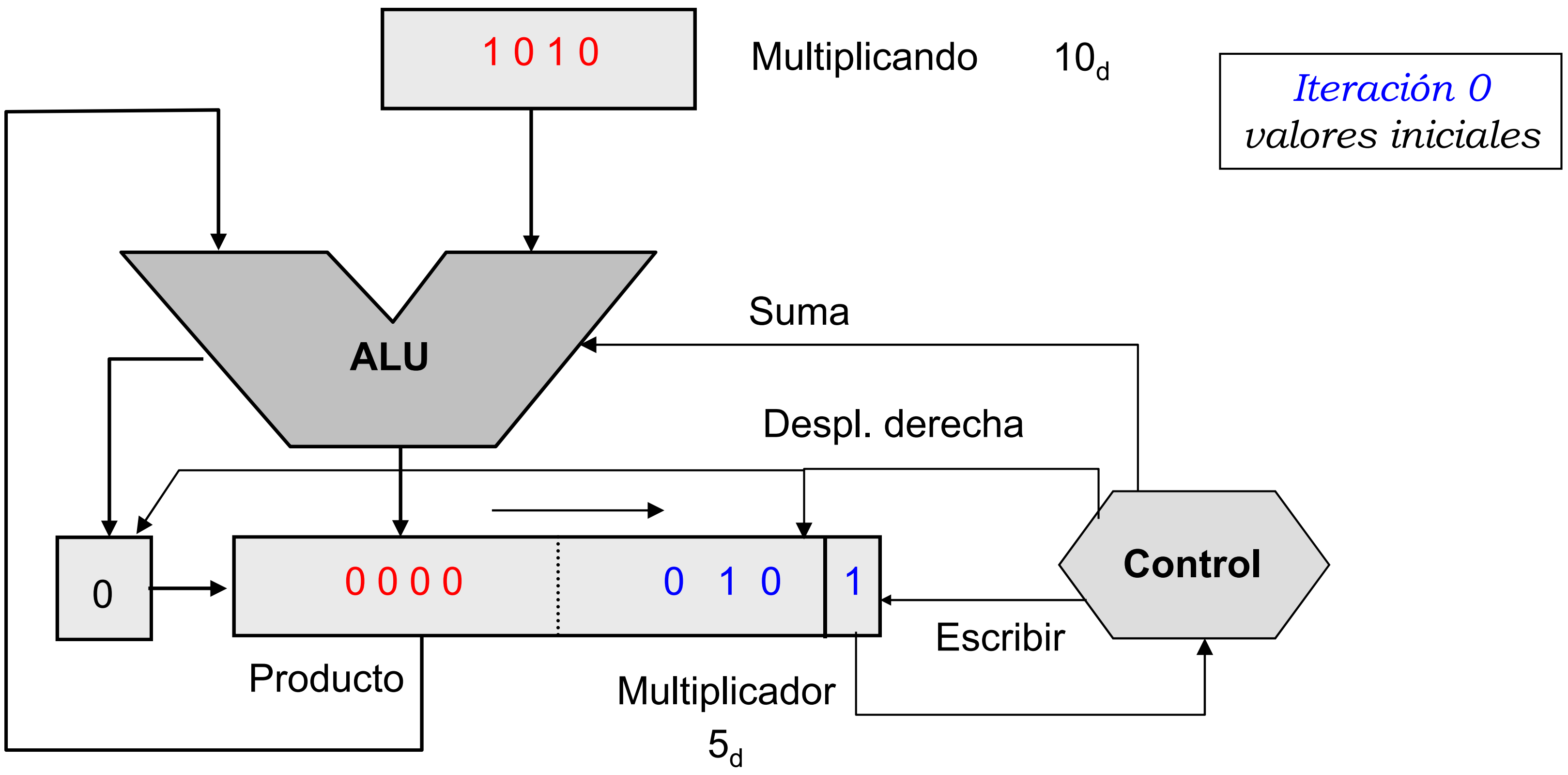
Desplazar 1 bit a la derecha el registro producto

Fin repetir

Versión final

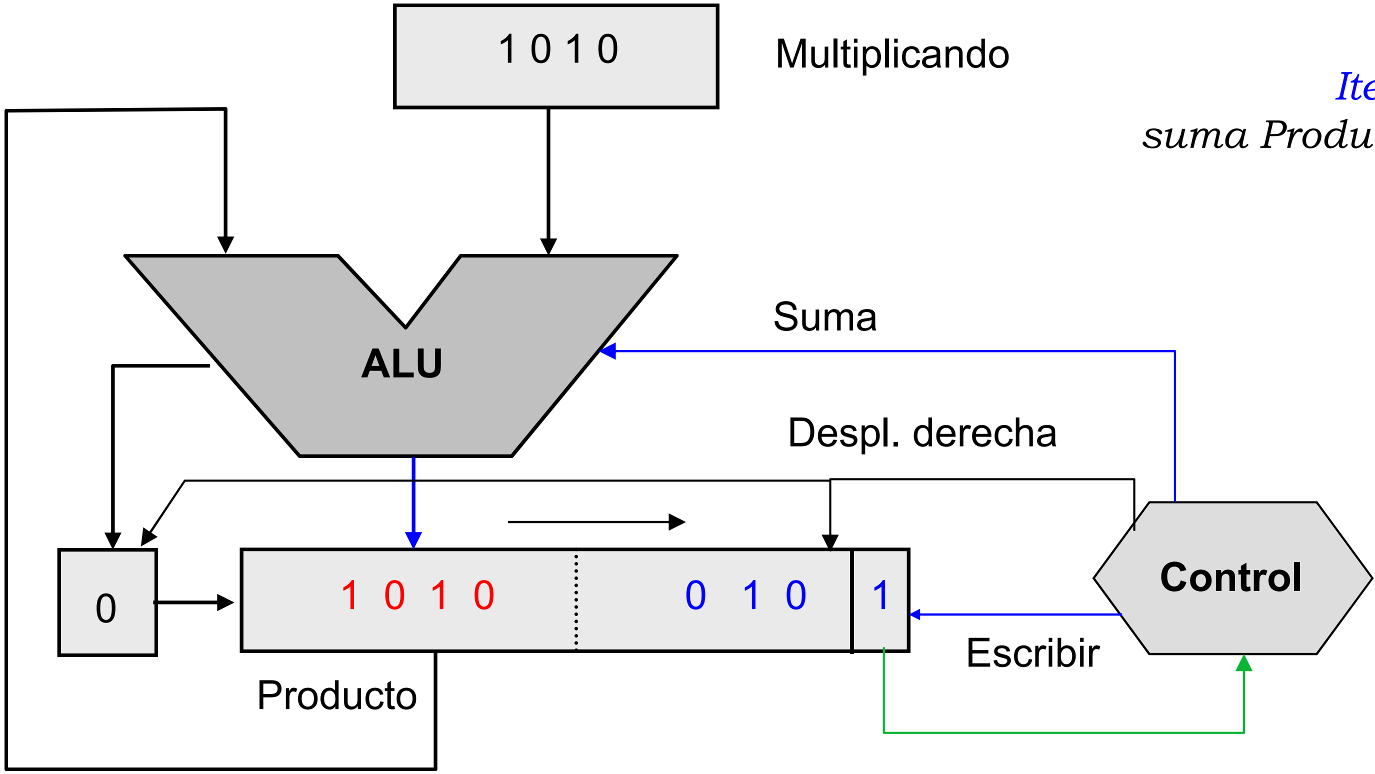


MULTIPLICACIÓN BINARIA SIN SIGNO



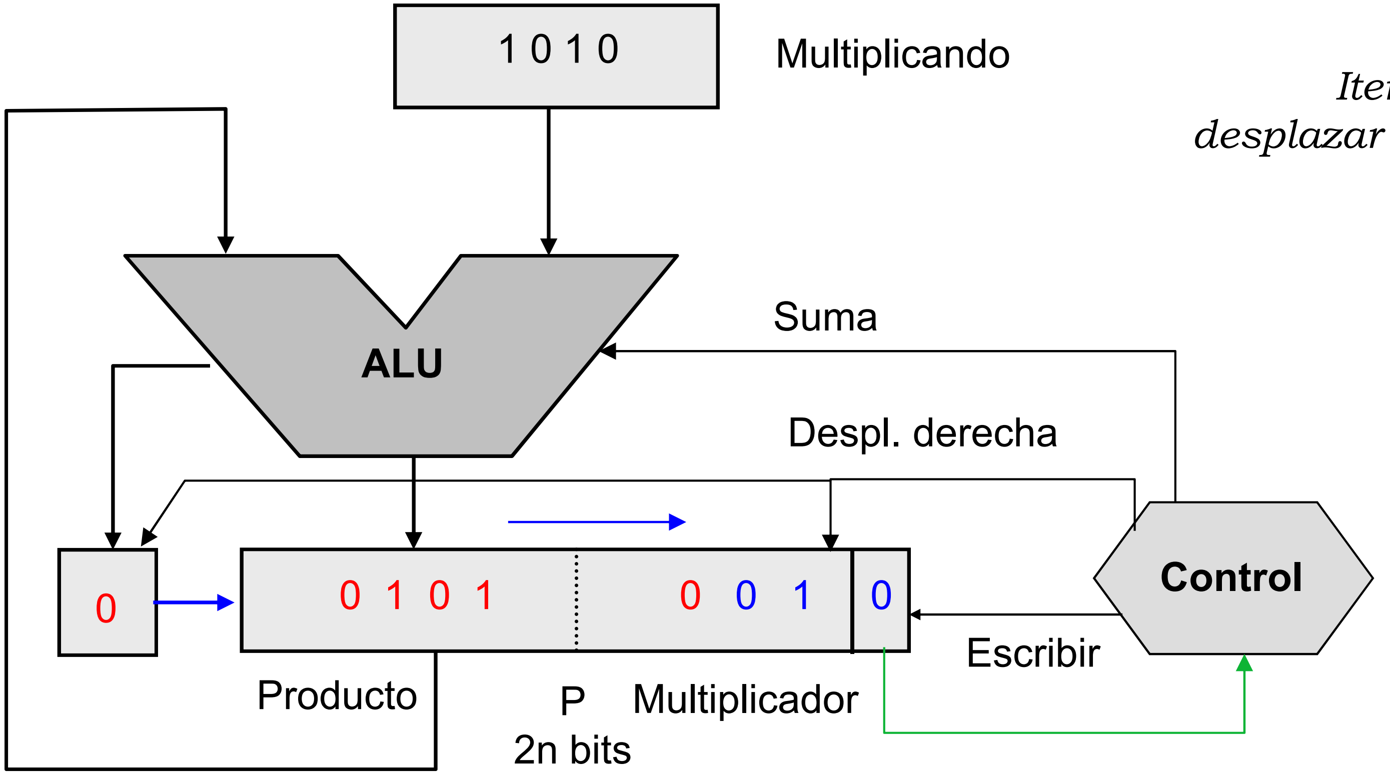
MULTIPLICACIÓN BINARIA SIN SIGNO

Iteración 1
suma Producto y Multiplicando

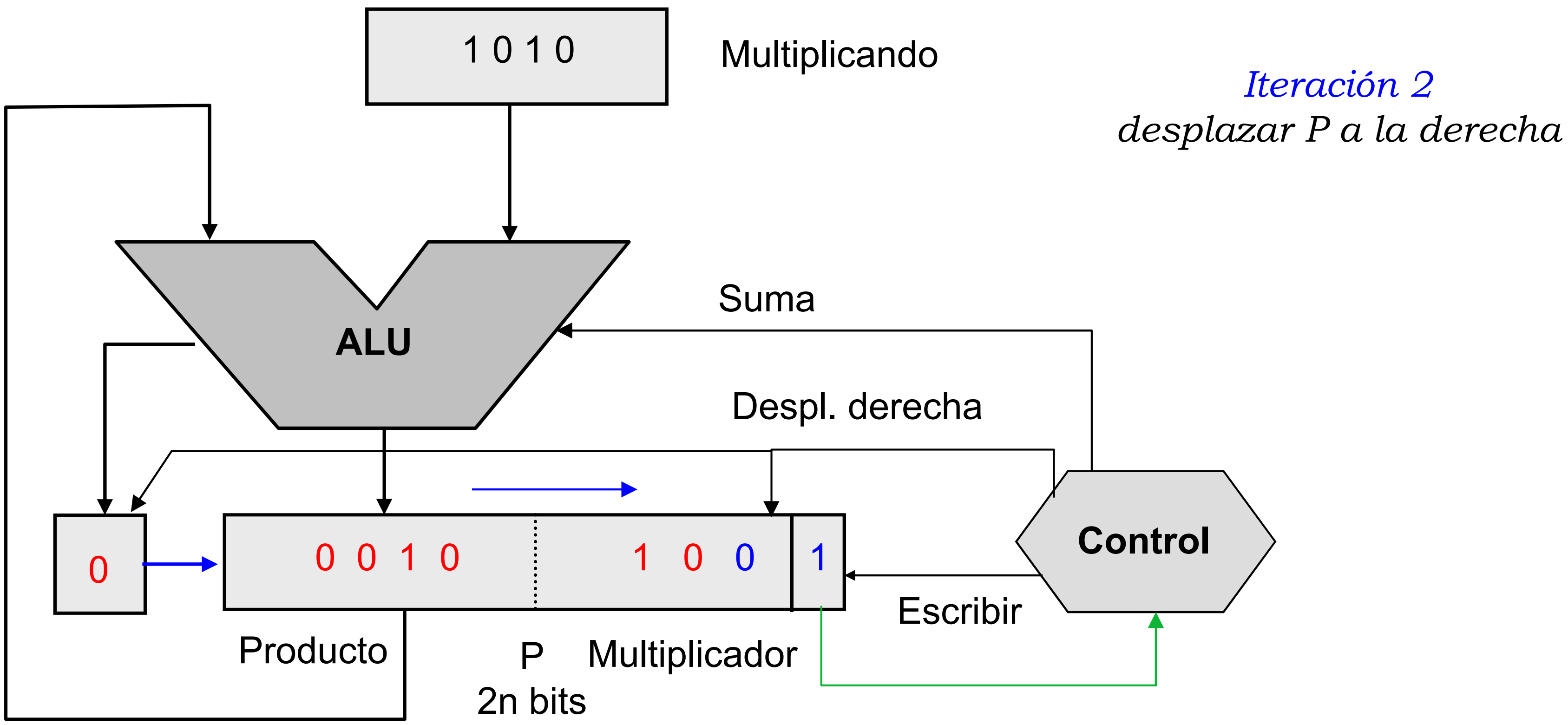


Multiplicación binaria sin signo

Iteración 1
desplazar P a la derecha.

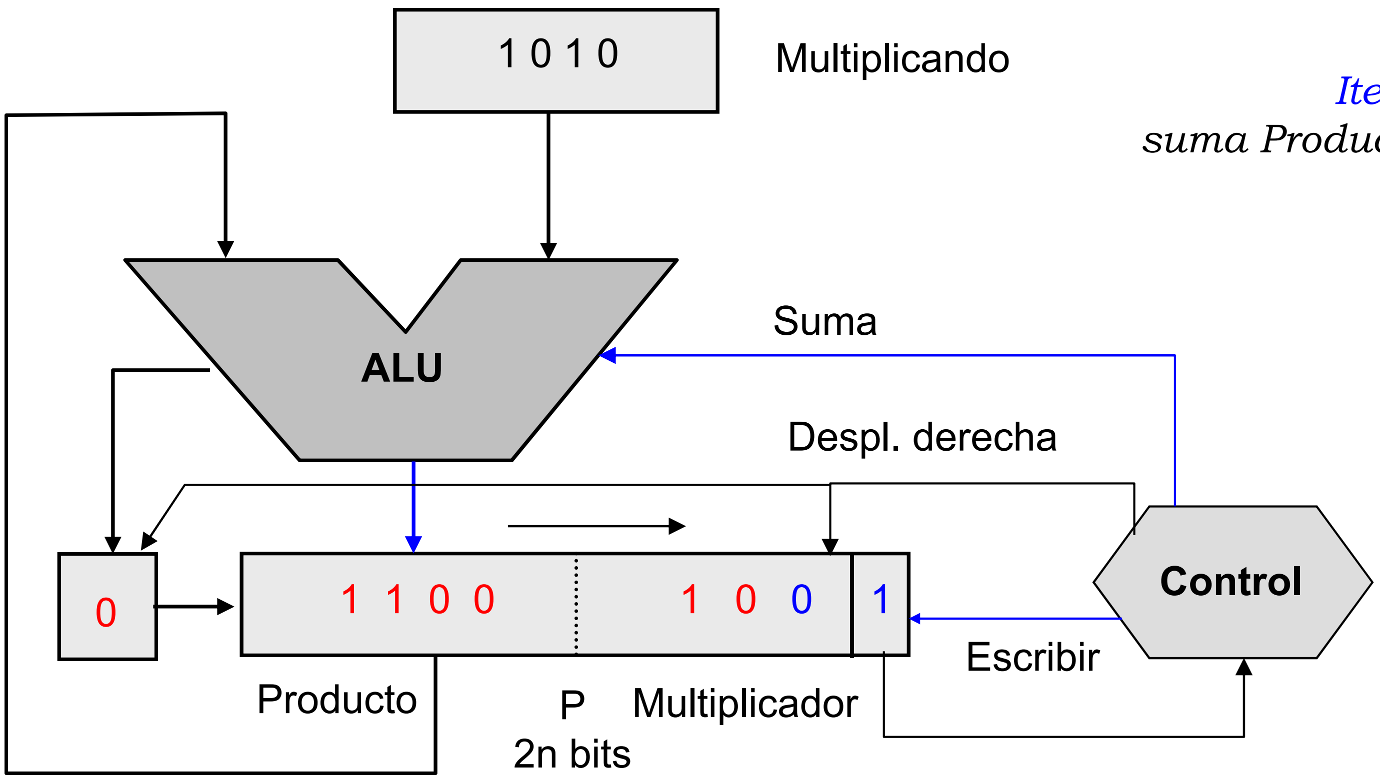


Multiplicación binaria sin signo

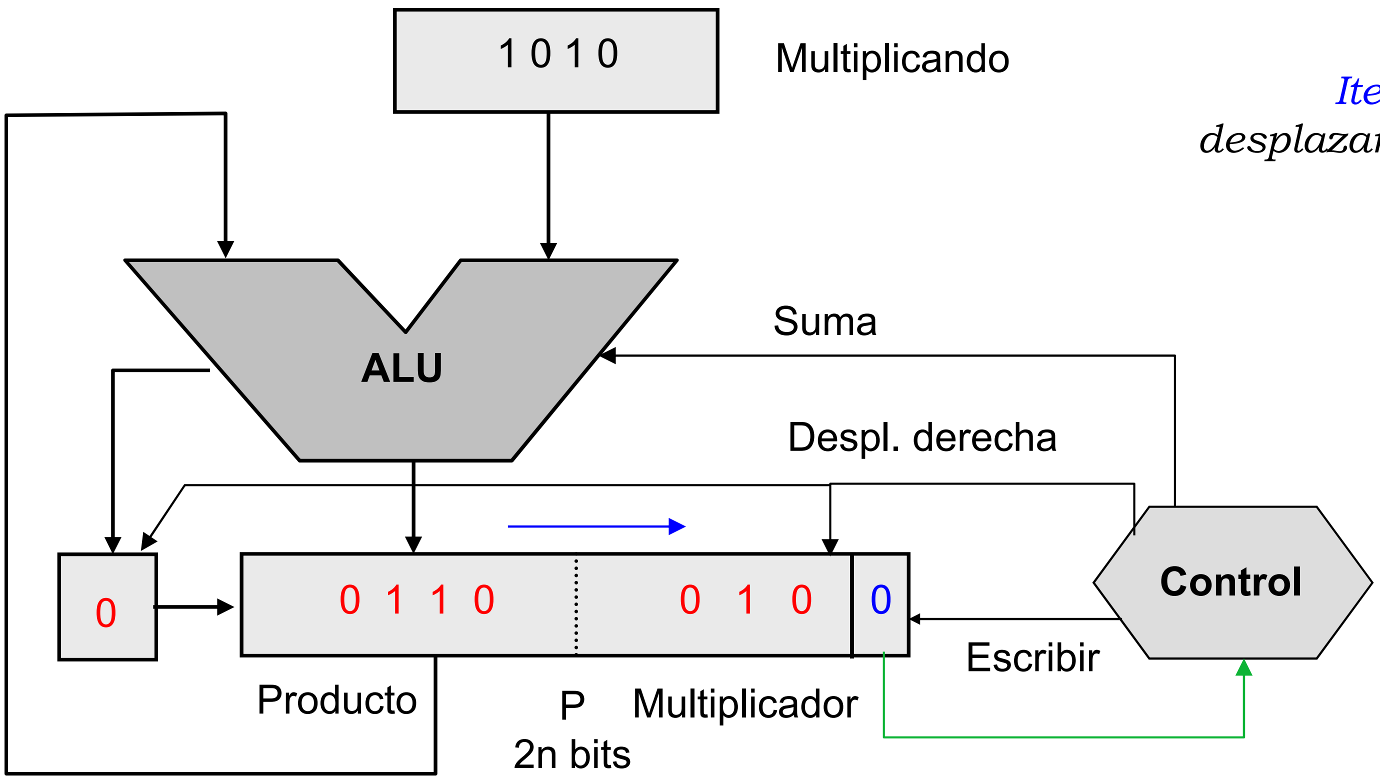


Multiplicación binaria sin signo

Iteración 3
suma Producto y Multiplicando



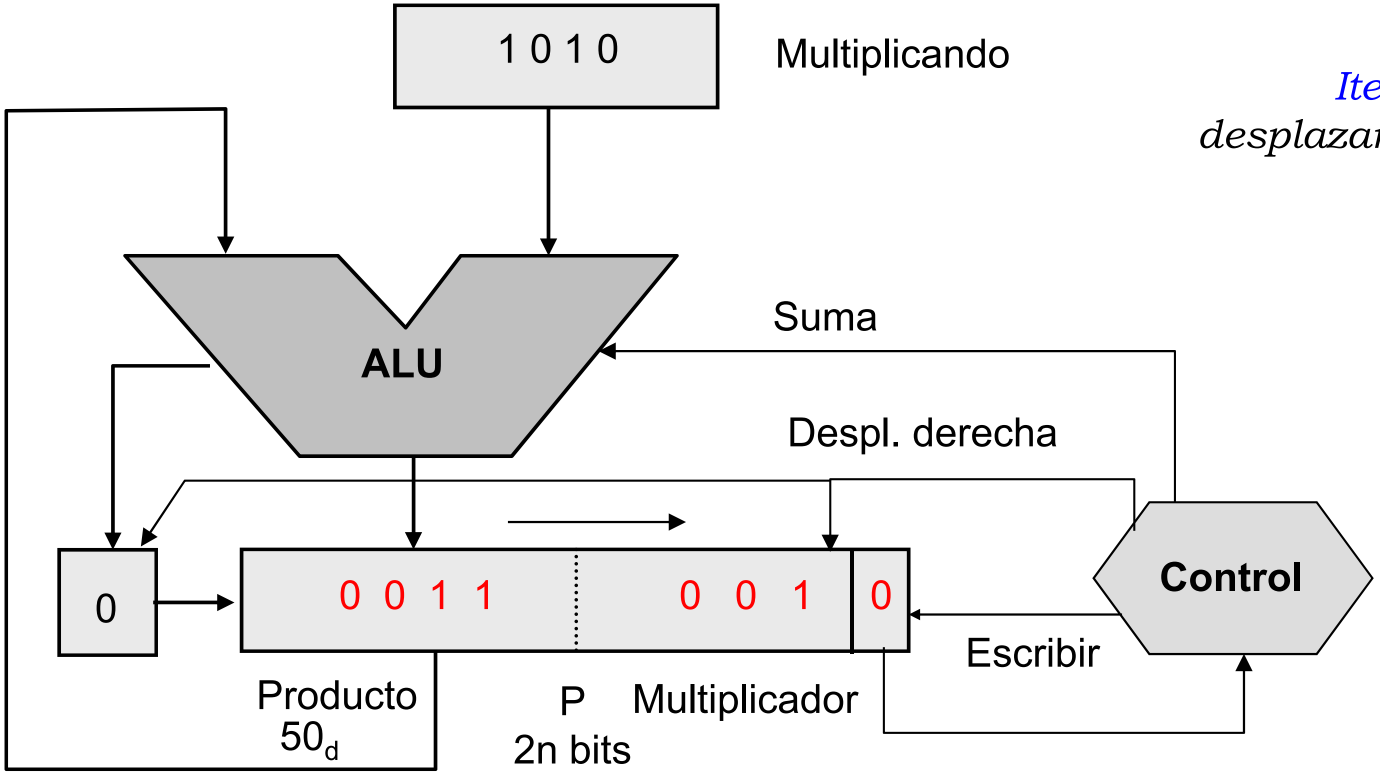
Multiplicación binaria sin signo



Iteración 3
desplazar P a la derecha

MULTIPLICACIÓN BINARIA SIN SIGNO

Iteración 4
desplazar P a la derecha

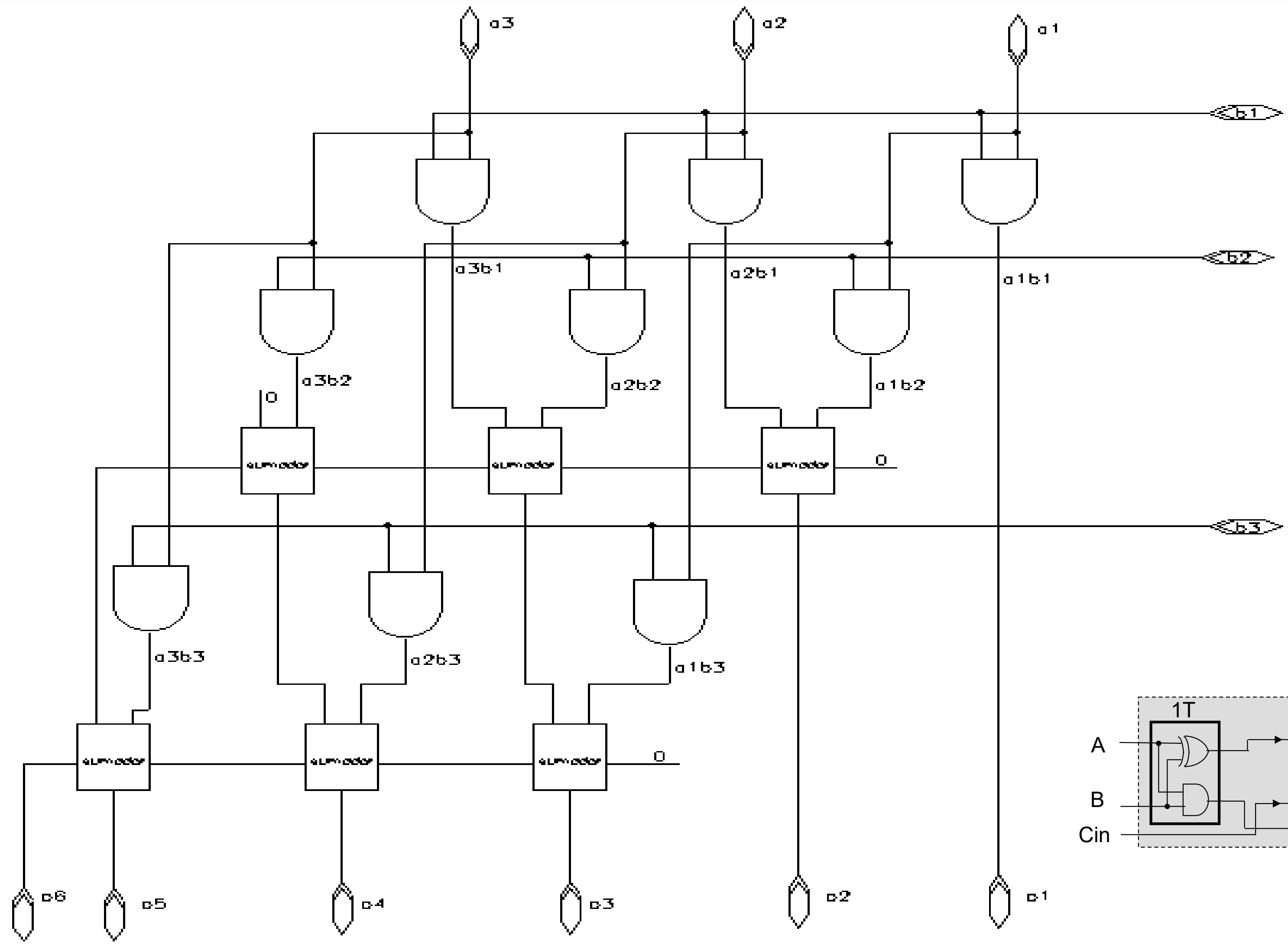


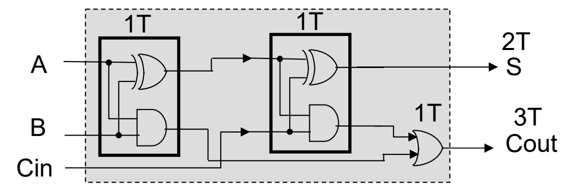
Multiplicando = 1010 10_d
Multiplicador = 0101 5_d

Producto	Multiplicando	Acción	Iteración
0000 010**1**	1010	Valores iniciales	0
1010 0101	1010	Sumar $prod_H$ + multiplicando	1
0101 001**0**	1010	Desplazar 1 bit a la derecha	1
0010 100**1**	1010	Despl. 1 bit a la derecha	2
1100 1001	1010	Sumar $prod_H$ + multiplicando	3
0110 010**0**	1010	Desplazar 1 bit a la derecha	3
0011 0010	1010	Desplazar 1 bit a la derecha	4

50_d

Multiplicación rápida (c. combinacional)





MULTIPLICACIÓN BINARIA CON SIGNO (M. TRADICIONAL)

- Supongamos números expresados en C2
- A = 1010 (-6_d) y B = 0011 ($+3_d$)
- Apliquemos algoritmo de sumas y desplazamientos

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

Versión errónea

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

Versión correcta (ext. de signo)

Unidad aritmética entera. Multiplicar y dividir

- Ejemplos:

```
5*5=25
     0101
   X 0101
     0101
    0000
   0101
  0000
 00011001
```

```
(-5)*5= -25
     1011
   X 0101
     1011
    0000
   1011
  0000
 00110111 ≠-25
```

```
(-5)*5= -25
     1011
   X 0101
 11111011
 0000000
 111011
 00000
 11100111 = -25
```

Los productos parciales deben representarse en C2

```
5*(-5)= - 25
     0101
   X 1011
     0101
    0101
   0000
  0101
 00110111 ≠-25
```

```
5*(-5)= - 25
     0101
   X 1011
 00000101
 0000101
 000000
 11011
 11100111 = -25
```

En la última iteración hay que restar el multiplicando

- El algoritmo de Booth simplifica el multiplicador para tratar de realizar un menor número productos parciales buscando polinomios compuestos por menos potencias de 2

$$V(m) = -m_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} m_i 2^i$$

$$V(m) = -m_{n-1} 2^{n-1} - \sum_{i=0}^{n-2} m_i 2^i + 2 \cdot \sum_{i=0}^{n-2} m_i 2^i$$

$$V(m) = -\left(m_{n-1} 2^{n-1} + m_{n-2} 2^{n-2} + \quad + m_1 2^1 + m_0 2^0 \right) + 2 \cdot \left(m_{n-2} 2^{n-2} + m_{n-3} 2^{n-3} + \quad + m_1 2^1 + m_0 2^0 \right) =$$

$$= -\left(m_{n-1} 2^{n-1} + m_{n-2} 2^{n-2} + \quad + m_i 2^i + m_0 2^0 \right) + \left(m_{n-2} 2^{n-1} + m_{n-3} 2^{n-2} + \quad + m_1 2^2 + m_0 2^1 \right)$$

$$V(m) = (m_{n-2} - m_{n-1}) 2^{n-1} + (m_{n-3} - m_{n-2}) 2^{n-2} + \quad + (m_1 - m_2) 2^2 + (m_0 - m_1) 2^1 + (0 - m_0) 2^0$$

MULTIPLICACIÓN BINARIA CON SIGNO (ALGORITMO DE BOOTH)

- Supongamos Multiplicando = 2 y Multiplicador = 7 (en binario 0010 x 0111)
- Booth recodifica el 7 y lo expresa como +100-1 = $2^3 + 0^2 + 0^1 - 1^0 = 7$
- Cada elemento recodificado b_i se obtienen a partir de la resta del elemento que ocupa la posición *i* en el número original del que se encuentra inmediatamente a su derecha.

				0	0	1	0	**Multiplicando**
			x	+1	0	0	-1	**Multiplicador según A. Booth**
1	1	1	1	1	1	1	0	**Restamos el multiplicando**
0	0	0	0	0	0			**2 despl. (2 ceros en el multiplicador)**
0	0	0	1	0				**Sumamos el multiplicando**
0	0	0	0	1	1	1	0	

Multiplicación binaria con signo (Algoritmo de Booth)

Bit n	Bit n-1	Sustitución
0	0	0 (no hay transición)
1	0	-1 (transición hacia negativo)
0	1	+1 (transición hacia positivo)
1	1	0 (no hay transición)

Se establece que $\mathbf{q_{-1} = 0}$ para calcular Booth del bit menos significativo

Ejemplo: Multiplicando = 11101110 (-18) y Multiplicador = 01111010 (0) (122)

Recodificación del multiplicador según Booth = +1000-1+1-10 = $2^7-2^3+2^2-2^1$ = 122

								1	1	1	0	1	1	1	0	
							x	+1	0	0	0	-1	+1	-1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0		
1	1	1	1	1	1	1	1	1	0	1	1	1	0			
0	0	0	0	0	0	0	0	1	0	0	1	0				
1	1	1	1	0	1	1	1	0	0	0	0					
1	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0	(-2196)

MULTIPLICACIÓN BINARIA CON SIGNO (ALGORITMO DE BOOTH)

Inicialmente $q_{-1}=0$

Repetir n veces

Si $q_0 = 1$ y $q_{-1} = 0$ entonces

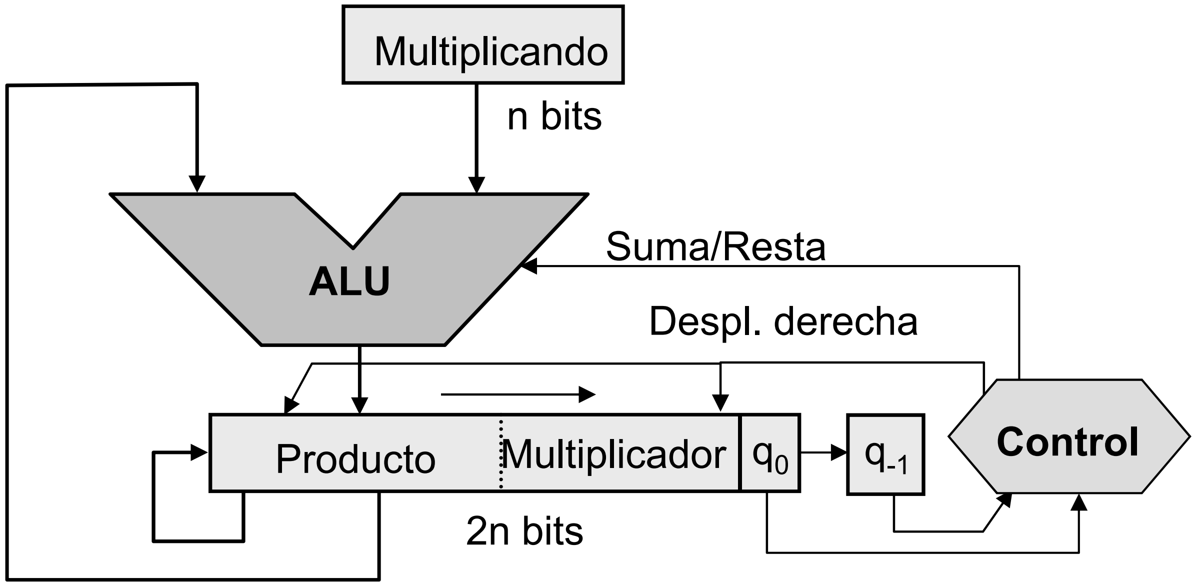
Producto$_h$ = Producto$_h$ - Multiplicando

Si $q_0 = 0$ y $q_{-1}=1$ entonces

Producto$_h$ = Producto$_h$ + Multiplicando

Desplazamiento aritmético a la derecha de Producto y q_{-1}

Fin repetir.



MULTIPLICACIÓN BINARIA CON SIGNO (ALGORITMO DE BOOTH)

Unidad aritmética entera. Multiplicar y dividir

Multiplicando = 1010 (-6)
Multiplicador = 1110 (-2)

Multiplicando	Producto	q_{-1}	Acción	Iteración
1010	0000 111**0**	**0**	Valores iniciales	0
1010	0000 1110	0	00 → Ninguna operación	1
1010	0000 011**1**	**0**	Despl. arit. 1 bit dcha.	1
1010	0110 0111	0	10→ **$prod_H$=$prod_H$ - multiplicando**	2
1010	0011 001**1**	**1**	Despl. arit. 1 bit dcha.	2
1010	0011 0011	1	11 → Ninguna operación	3
1010	0001 100**1**	**1**	Despl. arit. 1 bit dcha.	3
1010	0001 1001	1	11 → Ninguna operación	4
1010	**0000 1100**	1	Despl. arit. 1 bit dcha.	4

Multiplicando = 0110 (+6)
Multiplicador = 1110 (-2)

Multiplicando	Producto	q_{-1}	Acción	Iteración
0110	0000 111**0**	**0**	Valores iniciales	0
0110	0000 1110	0	00 → Ninguna operación	1
0110	0000 011**1**	**0**	Despl. arit. 1 bit dcha.	1
0110	1010 0111	0	10→ **$prod_H$=$prod_H$ - multiplicando**	2
0110	1101 001**1**	**1**	Despl. arit. 1 bit dcha.	2
0110	1101 0011	1	11 → Ninguna operación	3
0110	1110 100**1**	**1**	Despl. arit. 1 bit dcha.	3
0110	1110 1001	1	11 → Ninguna operación	4
0110	**1111 0100**	1	Despl. arit. 1 bit dcha.	4

Multiplicación binaria con signo (Algoritmo de Booth)

Multiplicando = 1010 (-6)
Multiplicador = 0010 (+2)

Multiplicando	Producto	q_{-1}	Acción	Iteración
1010	0000 0010	**0**	Valores iniciales	0
1010	0000 0010	0	00 → Ninguna operación	1
1010	0000 000**1**	**0**	Despl. arit. 1 bit dcha.	1
1010	0110 0001	0	10→ **$prod_H$=$prod_H$ - multiplicando**	2
1010	0011 000**0**	**1**	Despl. arit. 1 bit dcha.	2
1010	1101 0000	1	01 → **$prod_H$=$prod_H$+ multiplicando**	3
1010	1110 1000	**0**	Despl. arit. 1 bit dcha.	3
1010	1110 1000	0	00 → Ninguna operación	4
1010	**1111 0100**	1	Despl. arit. 1 bit dcha.	4

Unidad aritmética entera. Multiplicar y dividir

Multiplicando = 0110 (+6)
Multiplicador = 0010 (+2)

Multiplicando	Producto	q_{-1}	Acción	Iteración
0110	0000 0010	**0**	Valores iniciales	0
0110	0000 0010	0	00 → Ninguna operación	1
0110	0000 000**1**	**0**	Despl. arit. 1 bit dcha.	1
0110	1010 0001	0	10→ **$prod_H$=$prod_H$ - multiplicando**	2
0110	1101 000**0**	**1**	Despl. arit. 1 bit dcha.	2
0110	0011 0000	1	01 → **$prod_H$=$prod_H$+ multiplicando**	3
0110	0001 1000	**0**	Despl. arit. 1 bit dcha.	3
0110	0001 1000	0	00 → Ninguna operación	4
0110	**0000 1100**	1	Despl. arit. 1 bit dcha.	4

- La división la podemos expresar como:

  Dividendo = Cociente x Divisor + Resto

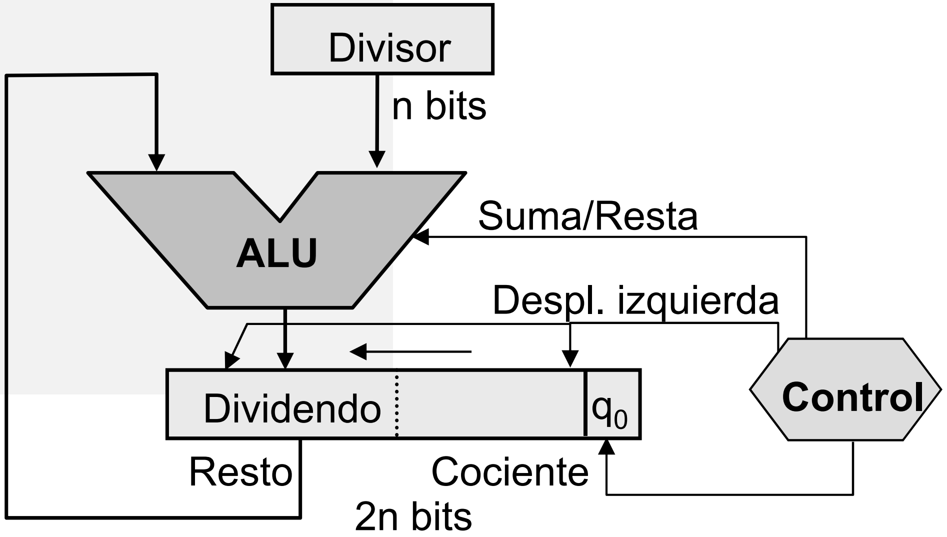
- El resto es más pequeño que el divisor. Hay que reservar el doble de espacio para el dividendo.

- Supondremos operandos positivos.

```
Dividendo →  10010011 (147)    | 1011 (11)   ←Divisor
             10010 (18)          01101 13    ←Cociente
              1011
             ---------------
             001110 (14)
               1011
             ---------------
               001111 (15)
                 1011
             ---------------
                 0100 (4)         ←Resto
```

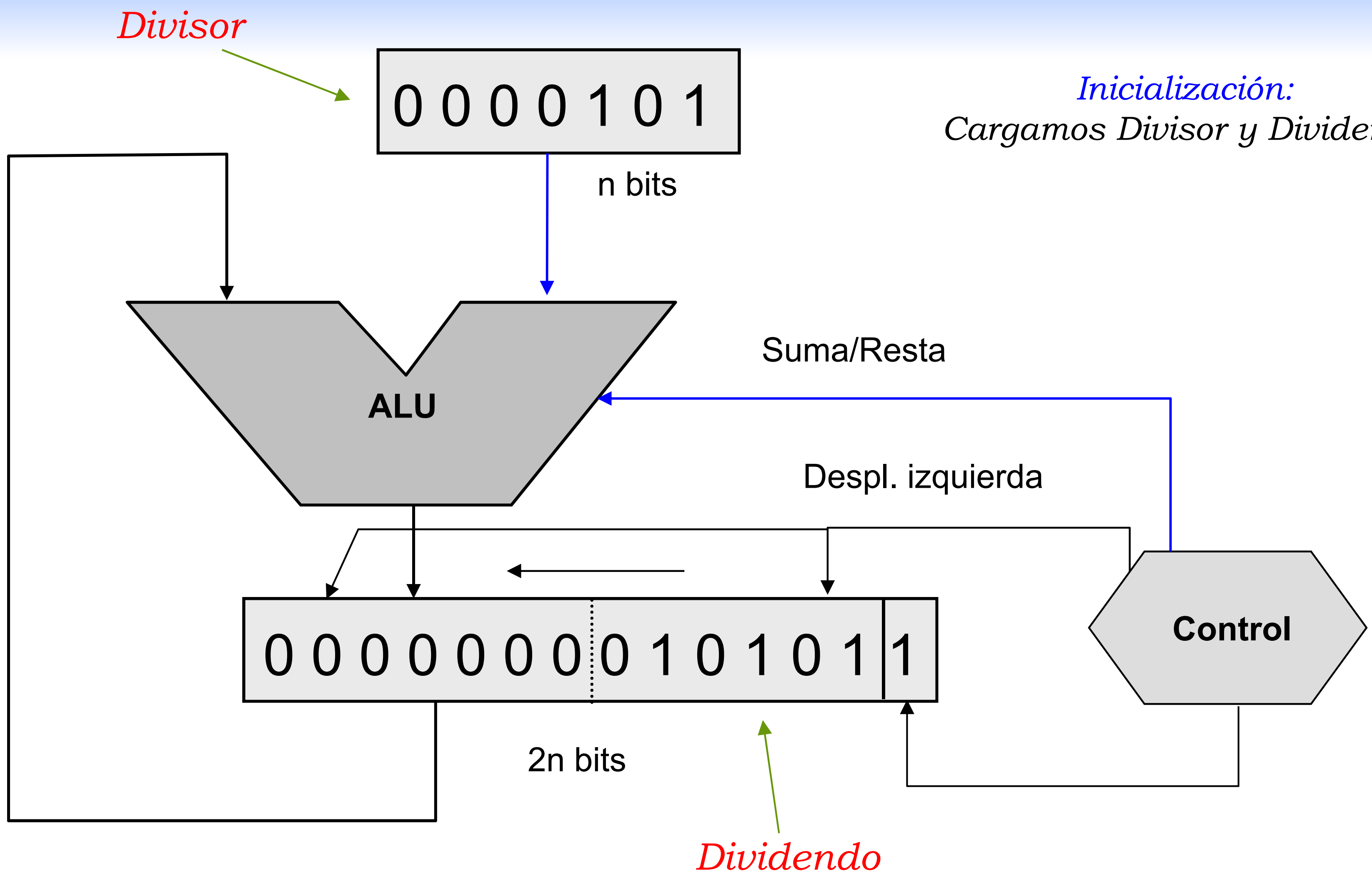
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Dividendo$_h$ = Dividendo$_h$ - Divisor
Repetir n veces
    Si Dividendo$_h$ < 0 entonces
    Desplazar el Dividendo a la izquierda
    Dividendo$_h$ = Dividendo$_h$ + Divisor
    Sino
        Desplazar el Dividendo a la izquierda
        Dividendo$_h$ = Dividendo$_h$ - Divisor
    Fin Si
    Si Dividendo$_h$ < 0 entonces
        q_0=0
    Sino
        q_0=1
    Fin Si
Fin Repetir
Si Dividendo$_h$ < 0 entonces
    Dividendo$_h$ = Dividendo$_h$ + Divisor
Fin Si

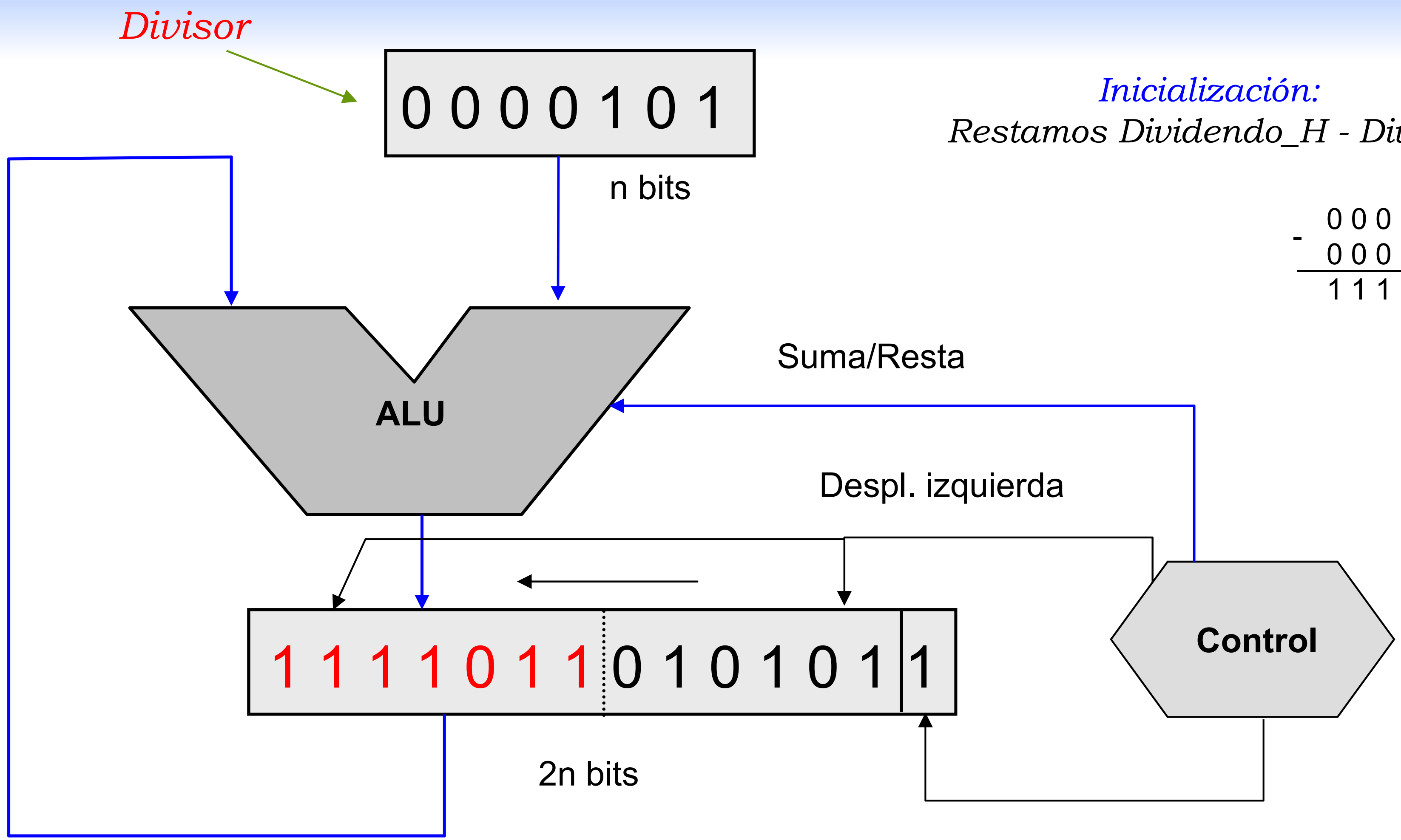


Unidad aritmética entera. Multiplicar y dividir

Inicialización:
Cargamos Divisor y Dividendo



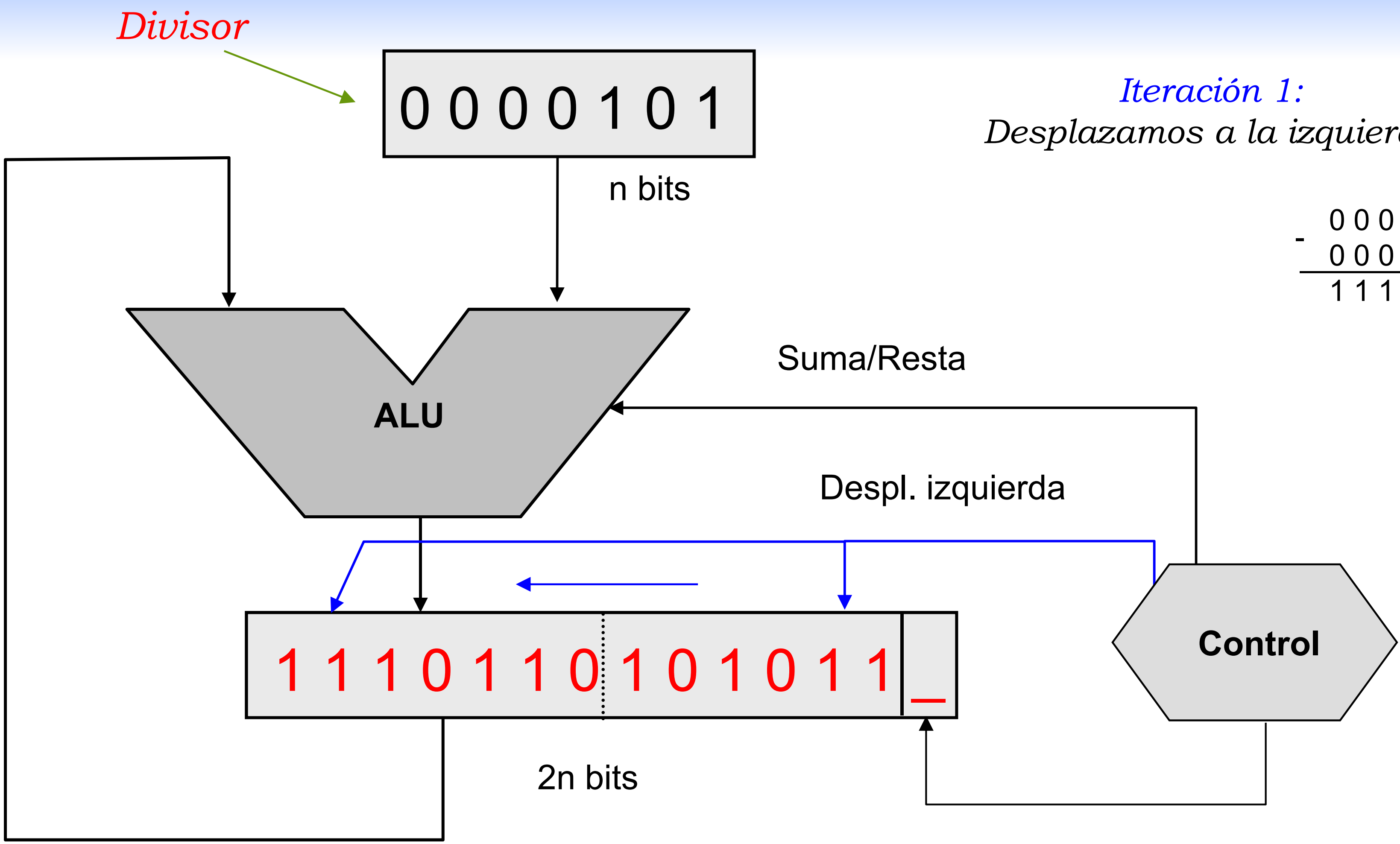
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN



Inicialización:
Restamos Dividendo_H - Divisor

```
  0 0 0 0 0 0 0
- 0 0 0 0 1 0 1
  -------------
  1 1 1 1 0 1 1
```

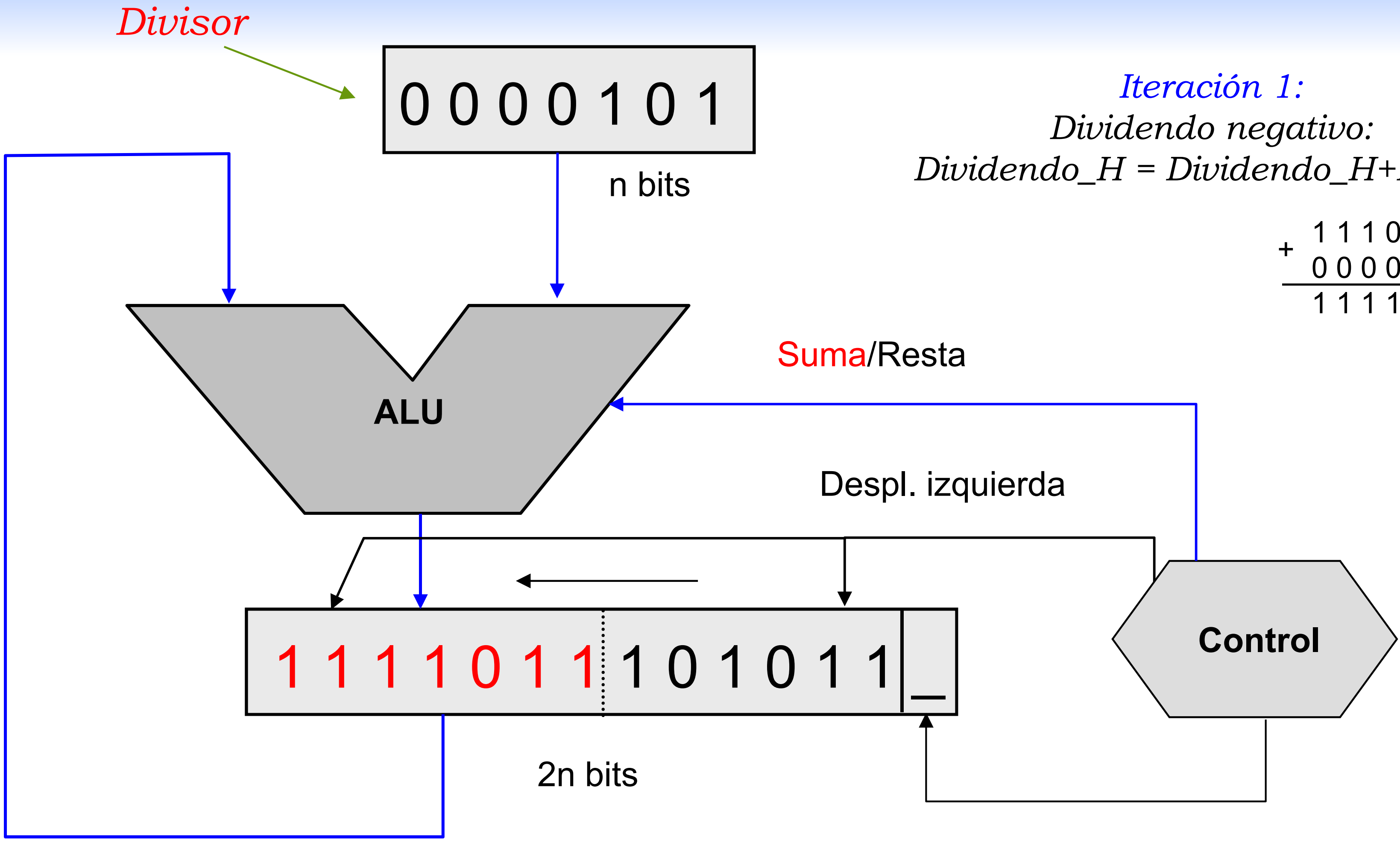
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN



Iteración 1:
Desplazamos a la izquierda

$$\begin{array}{r} 0\,0\,0\,0\,0\,0\,0 \\ -\ \ 0\,0\,0\,0\,1\,0\,1 \\ \hline 1\,1\,1\,1\,0\,1\,1 \end{array}$$

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

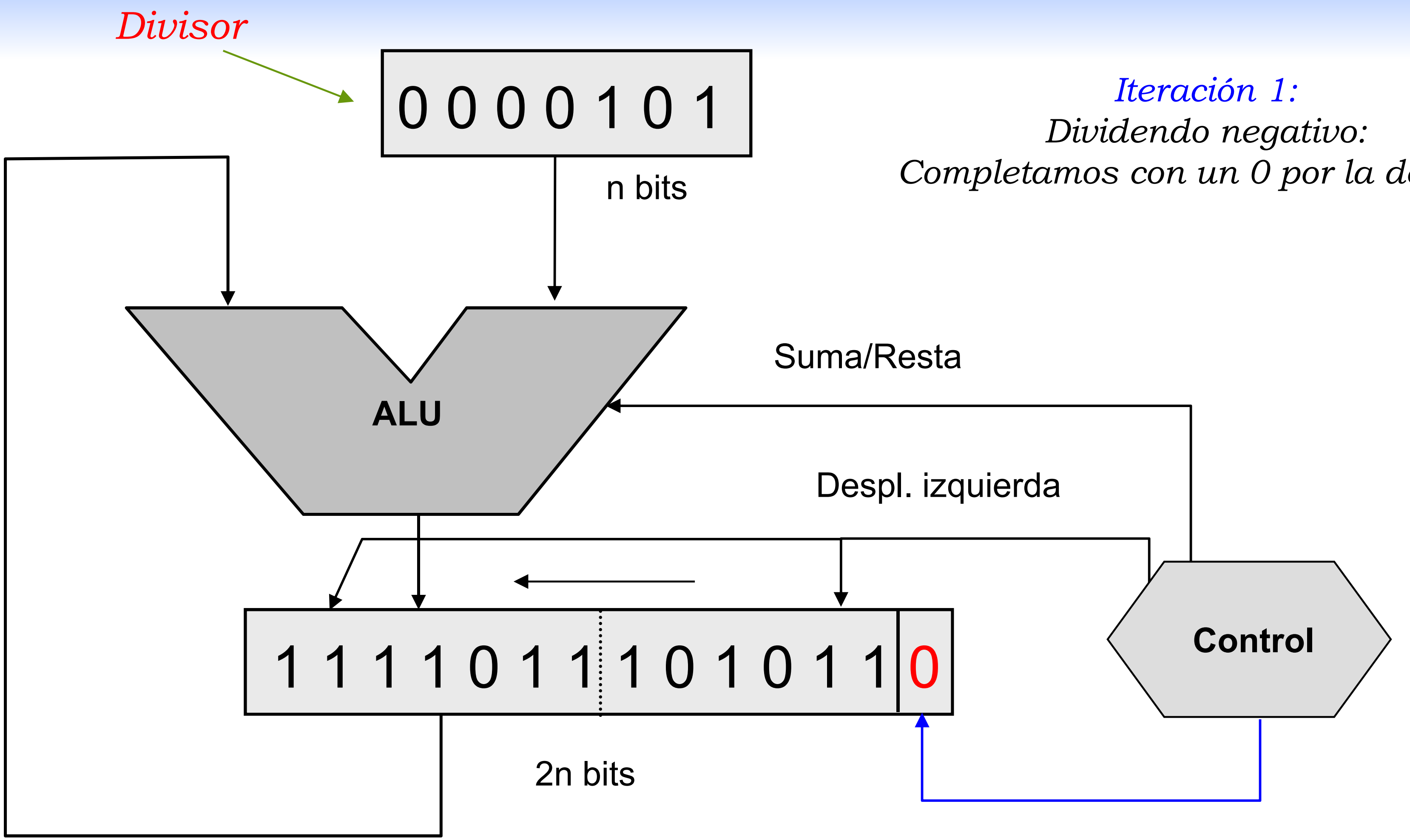


Iteración 1:
Dividendo negativo:
Dividendo_H = Dividendo_H+Divisor

$$\begin{array}{r} 1110110 \\ +\ 0000101 \\ \hline 1111011 \end{array}$$

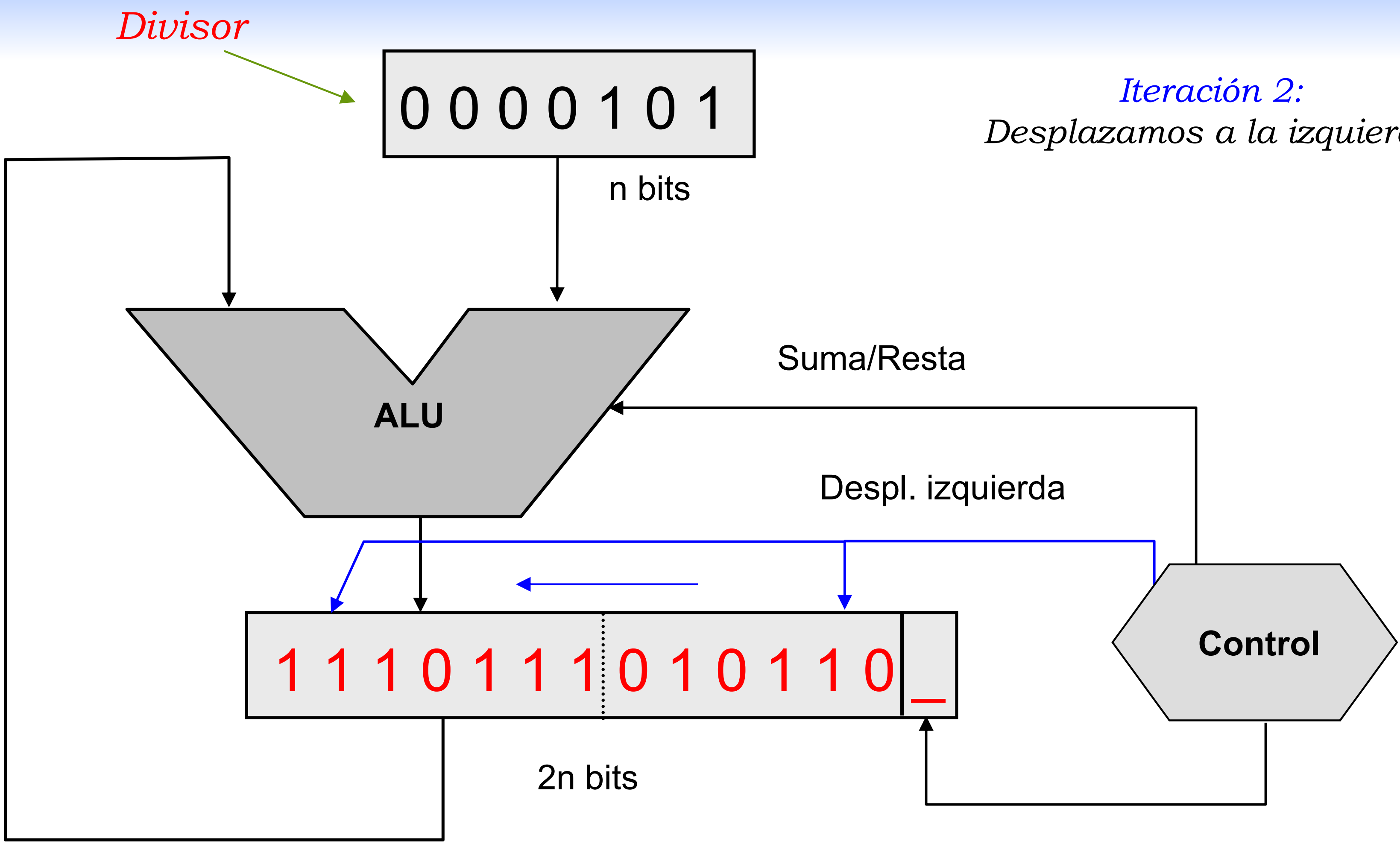
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

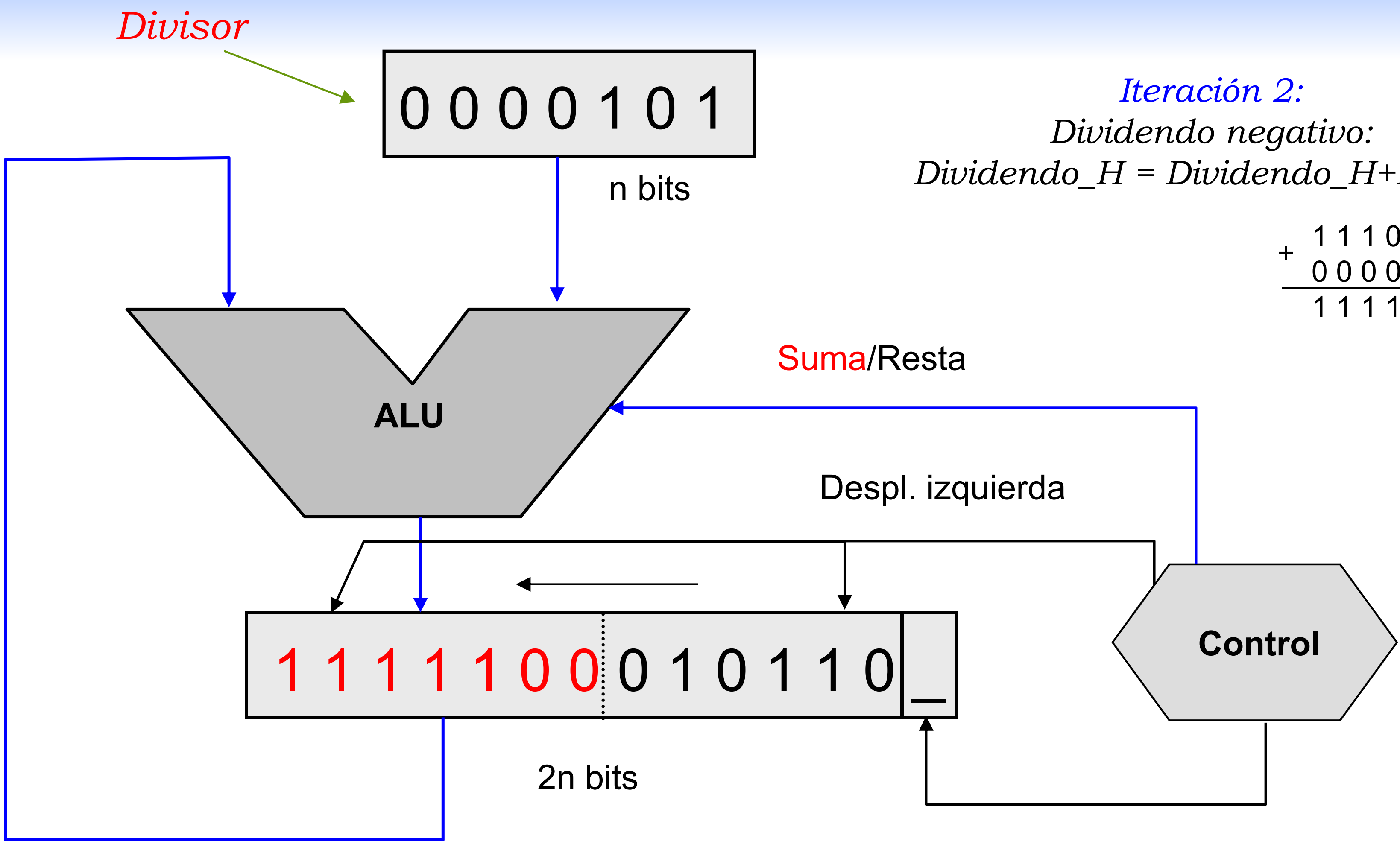
Iteración 1:
Dividendo negativo:
Completamos con un 0 por la derecha



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 2:
Desplazamos a la izquierda

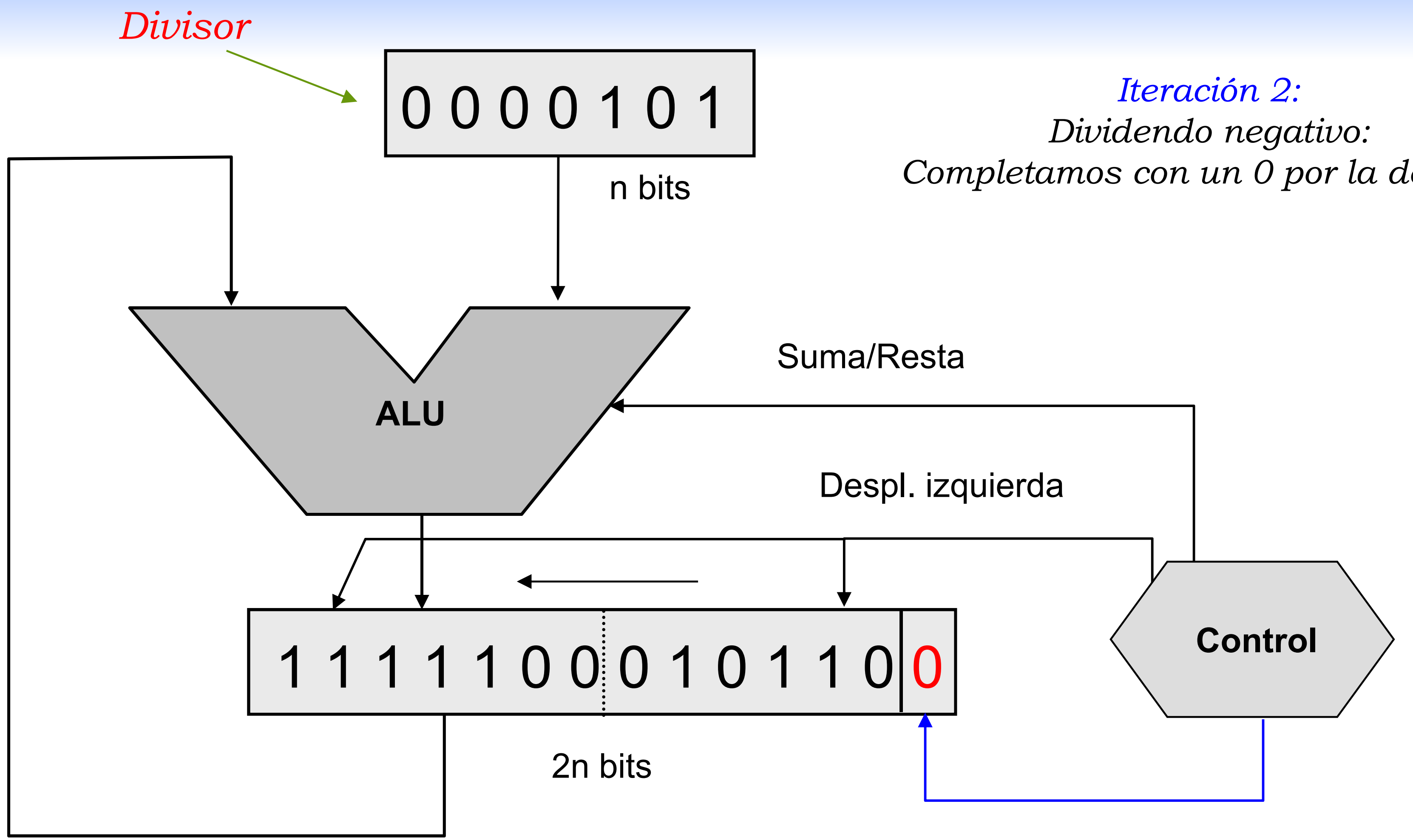




Iteración 2:
Dividendo negativo:
Dividendo_H = Dividendo_H+Divisor

$$\begin{array}{r} 1110111 \\ +\ 0000101 \\ \hline 1111100 \end{array}$$

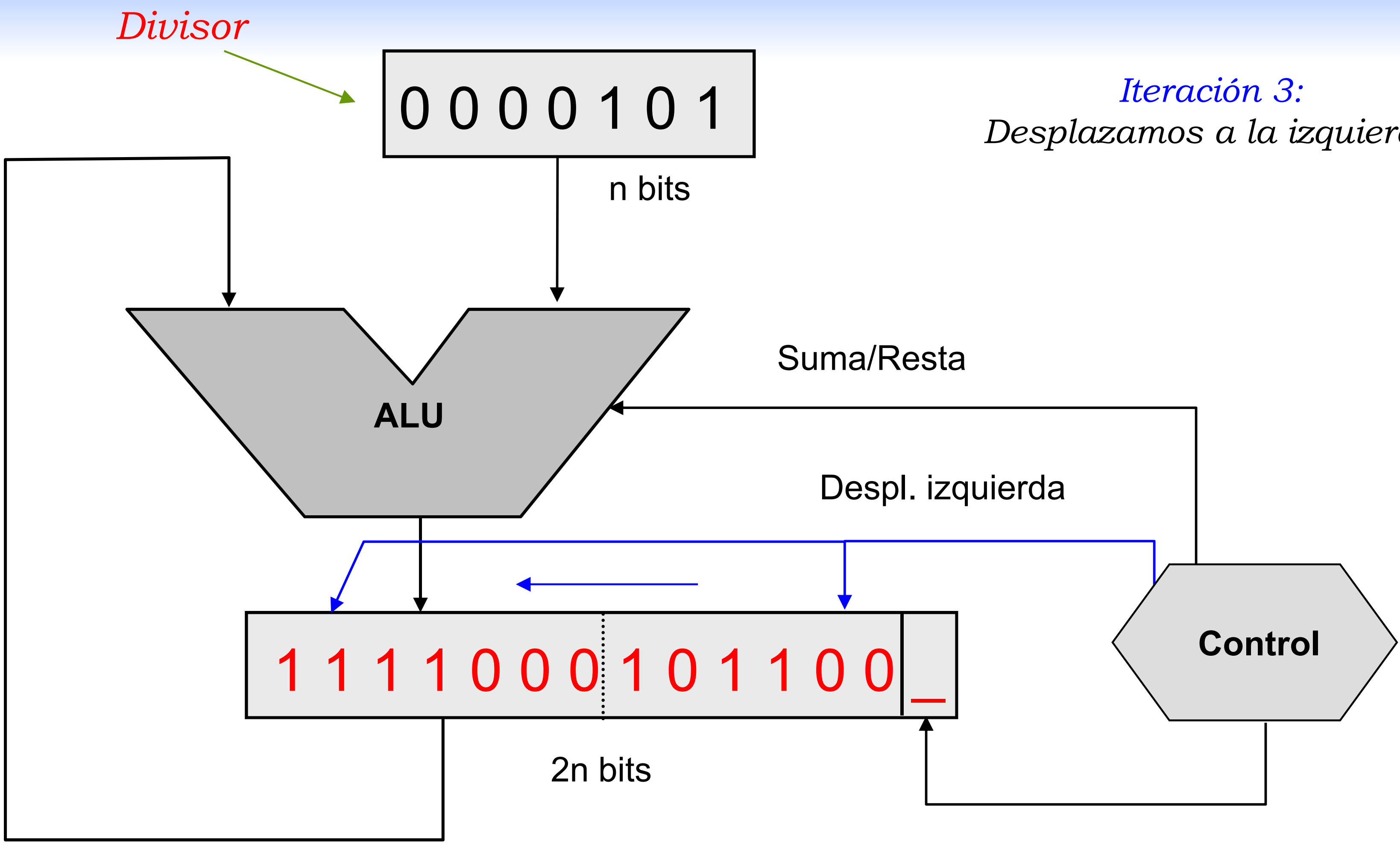
Unidad aritmética entera. Multiplicar y dividir



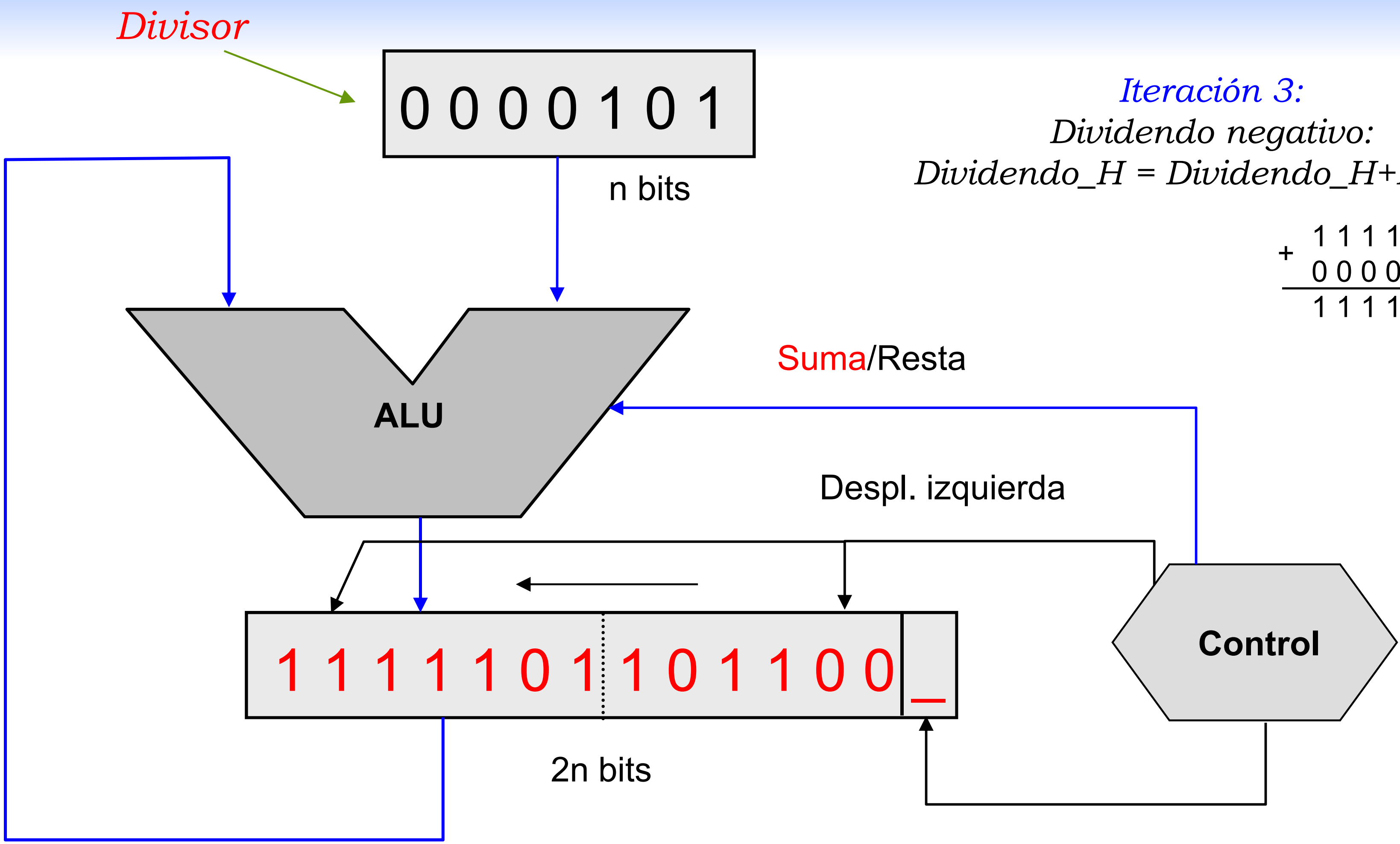
Iteración 2:
Dividendo negativo:
Completamos con un 0 por la derecha

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 3:
Desplazamos a la izquierda



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

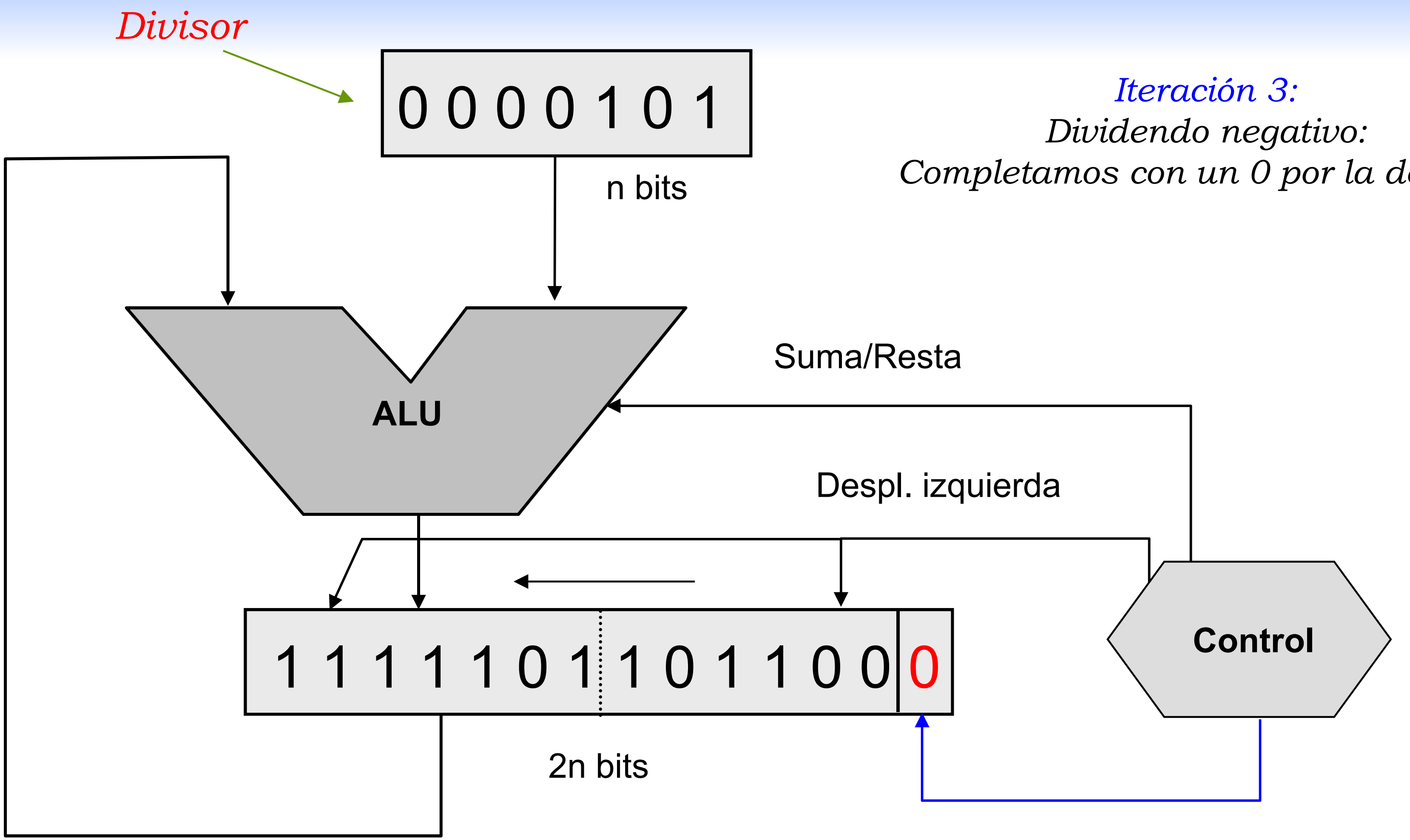


Iteración 3:
Dividendo negativo:
Dividendo_H = Dividendo_H+Divisor

```
  1 1 1 1 0 0 0
+ 0 0 0 0 1 0 1
  -------------
  1 1 1 1 1 0 1
```

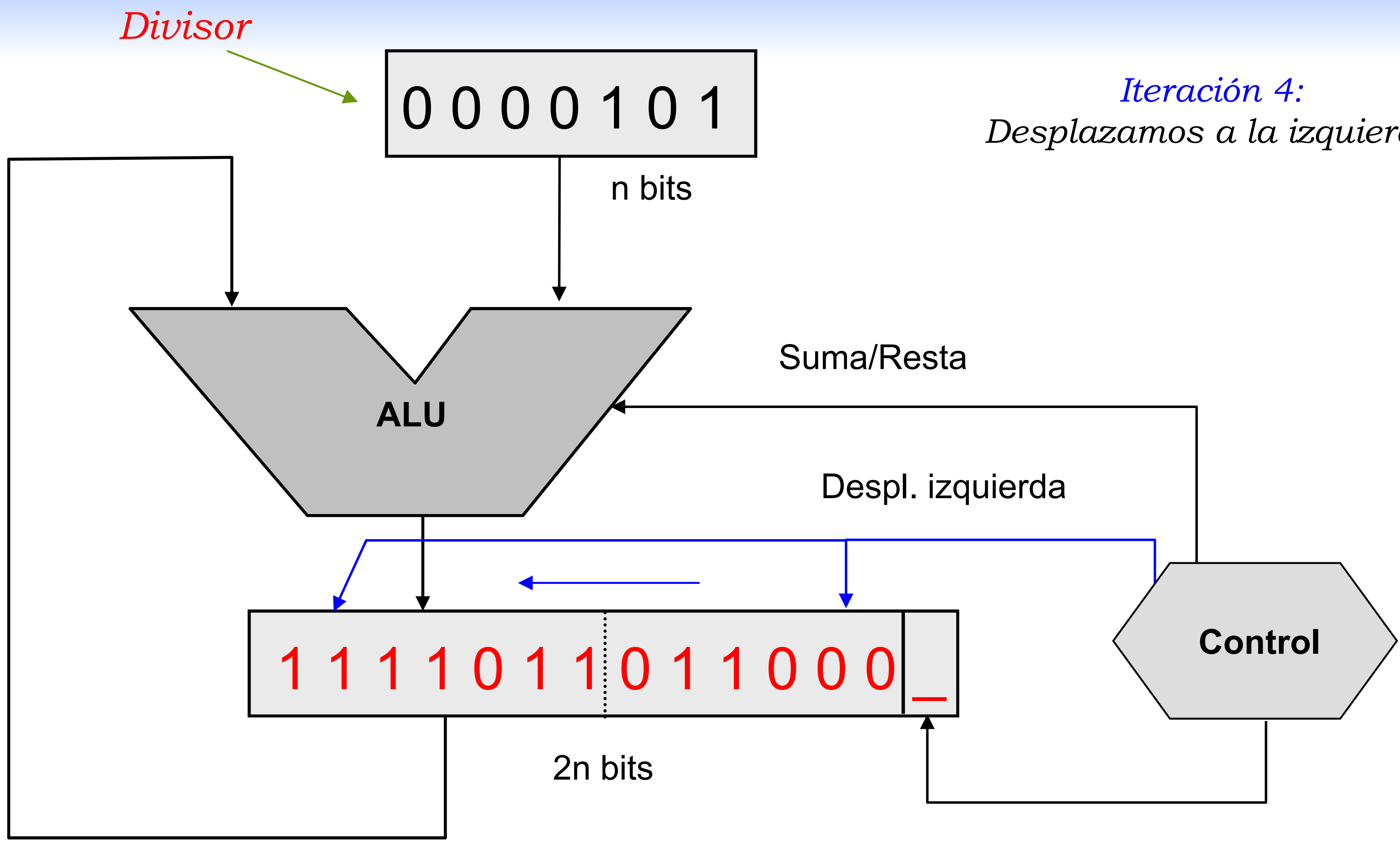
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 3:
Dividendo negativo:
Completamos con un 0 por la derecha

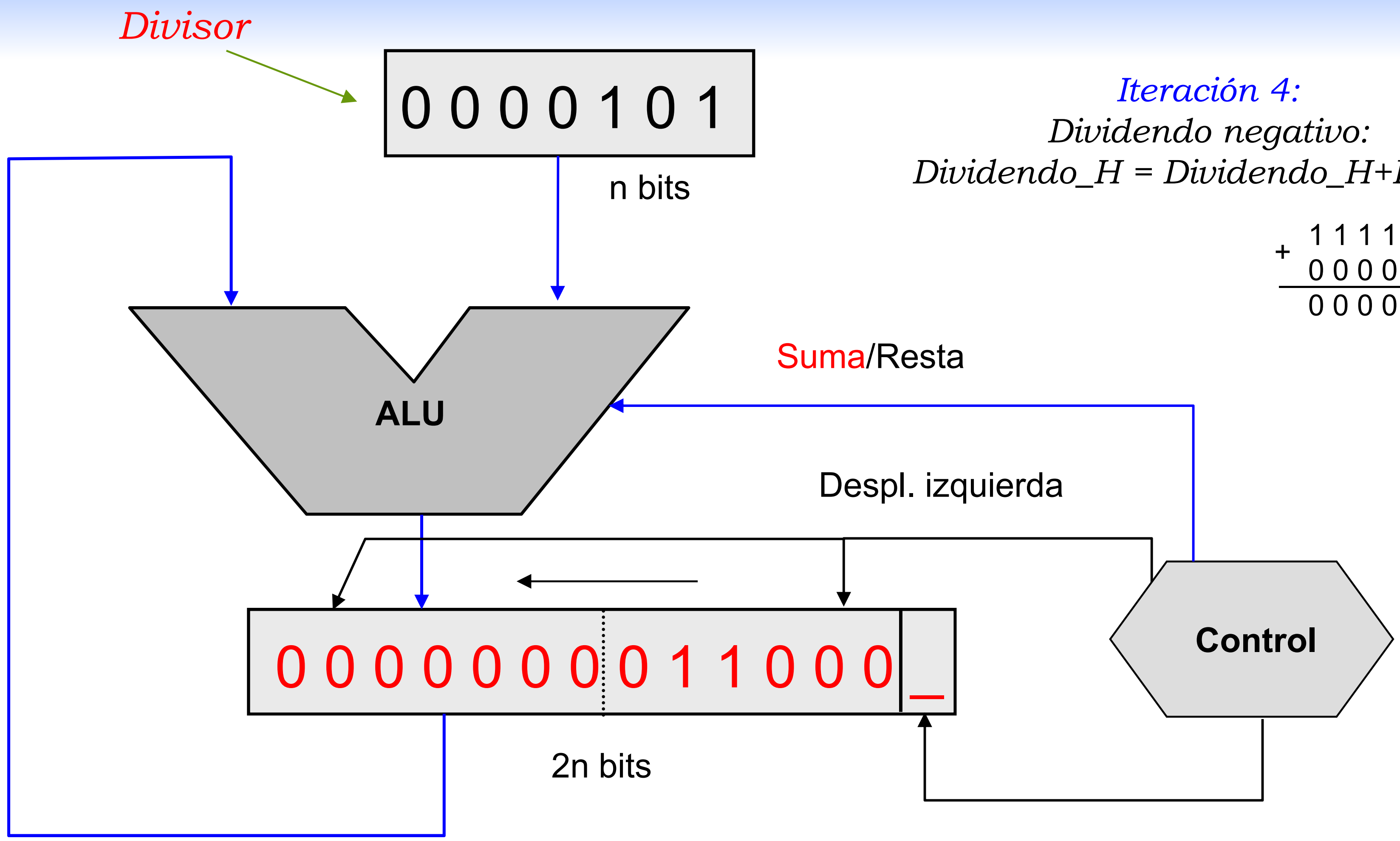


DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 4:
Desplazamos a la izquierda

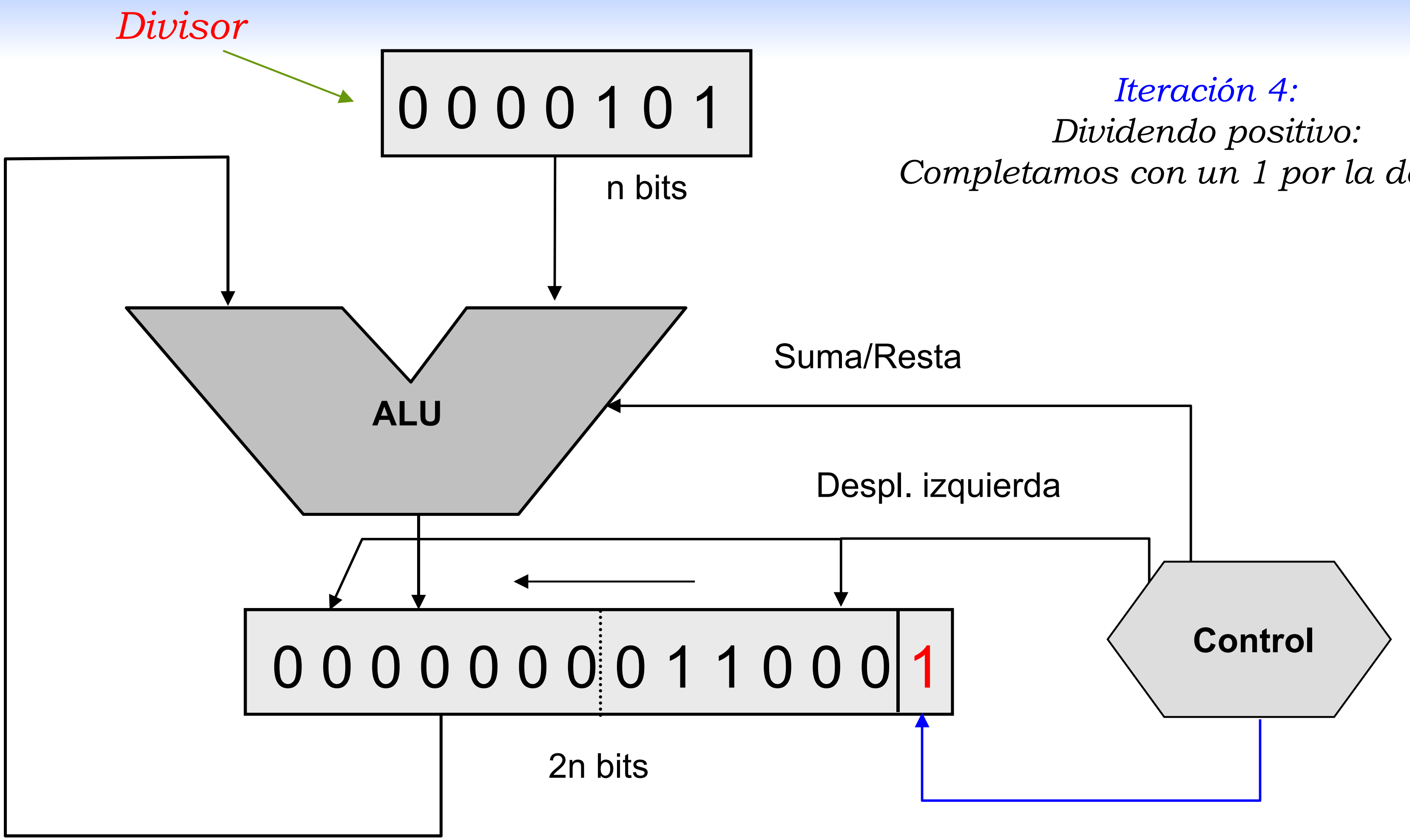


DIVISIÓN: ALGORITMO SIN RESTAURACIÓN



Iteración 4:
Dividendo negativo:
Dividendo_H = Dividendo_H+Divisor

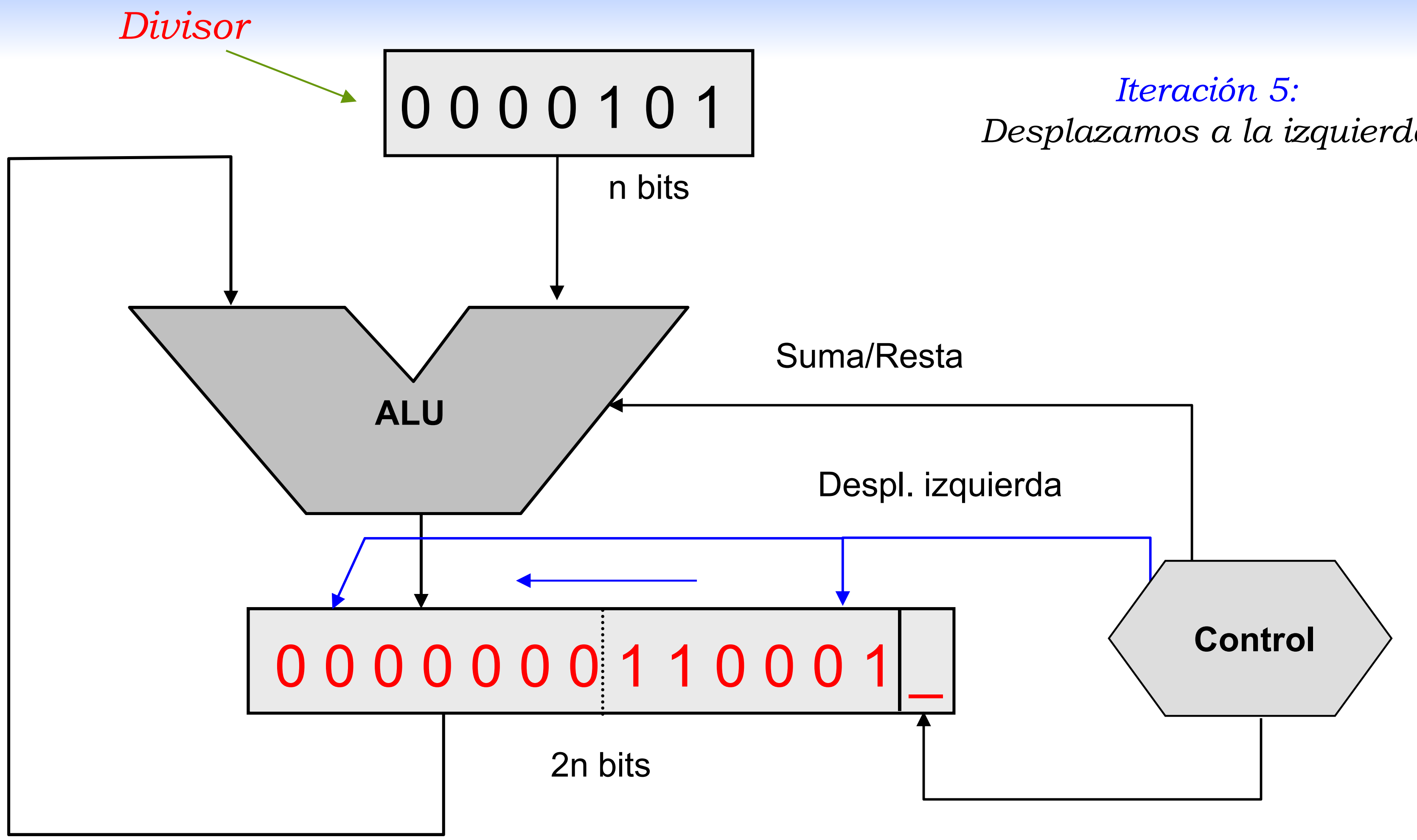
$$\begin{array}{r} 1111011 \\ +\ 0000101 \\ \hline 0000000 \end{array}$$



Iteración 4:
Dividendo positivo:
Completamos con un 1 por la derecha

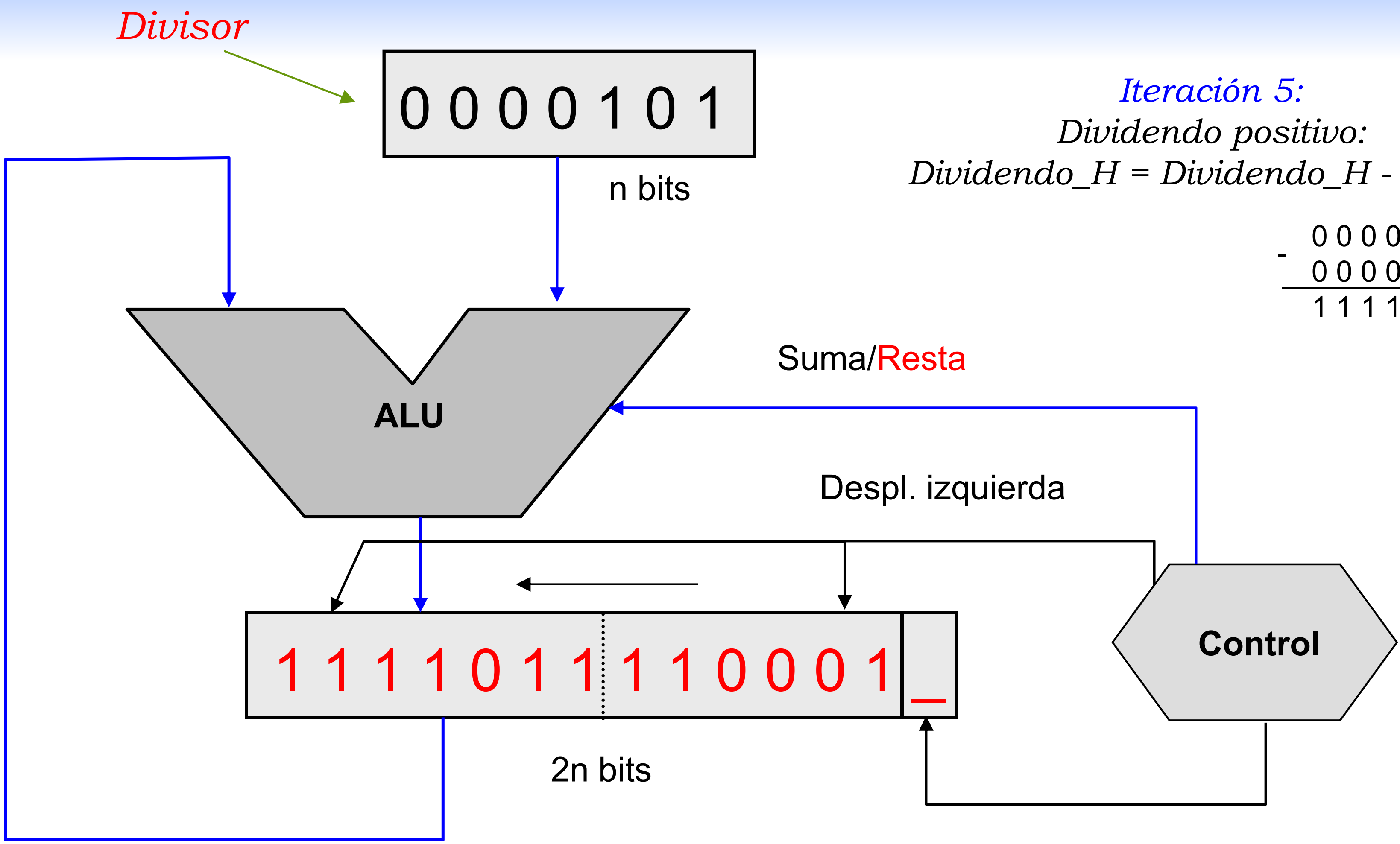
DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 5:
Desplazamos a la izquierda



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Unidad aritmética entera. Multiplicar y dividir

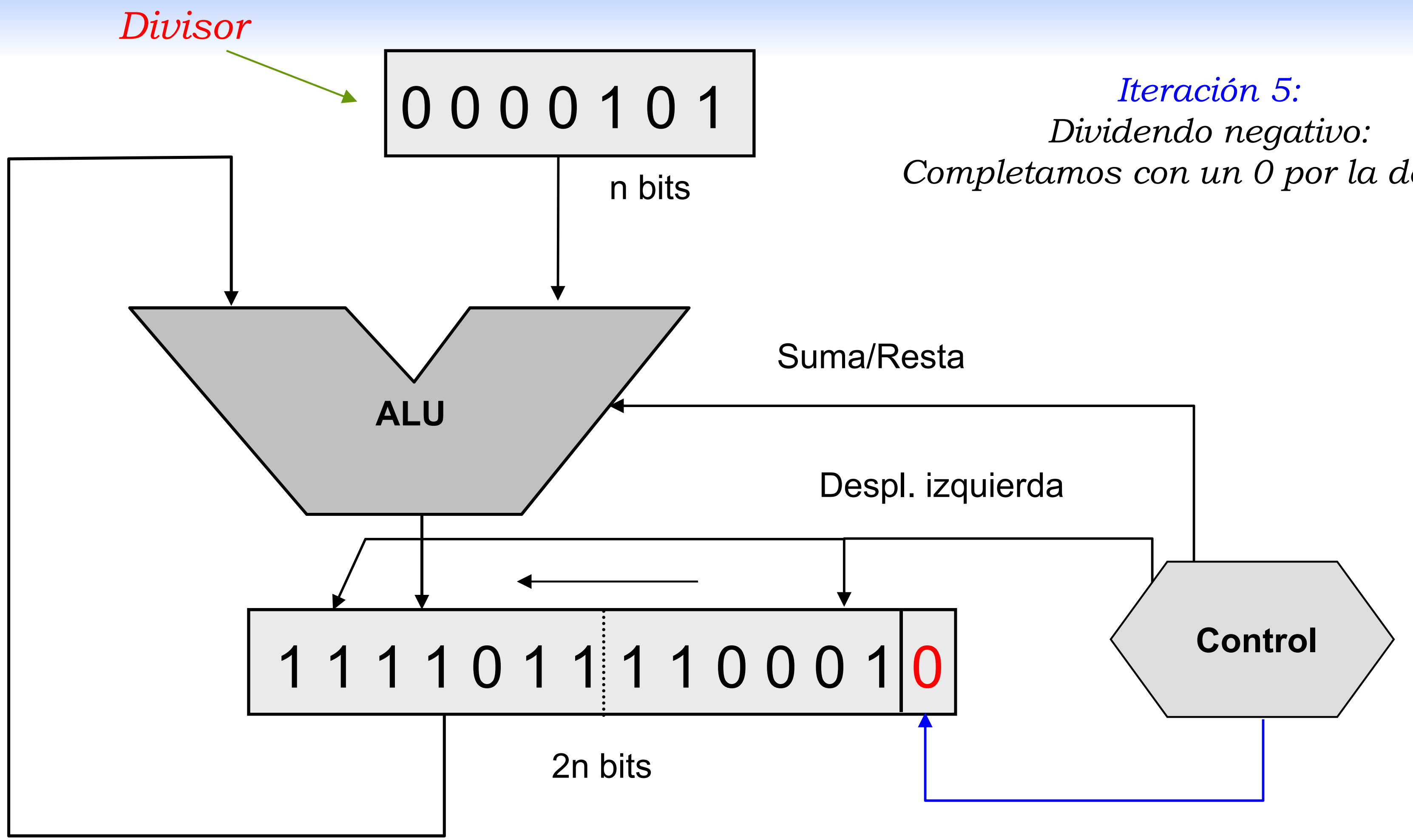


Iteración 5:
Dividendo positivo:
Dividendo_H = Dividendo_H - Divisor

```
  0 0 0 0 0 0 0
- 0 0 0 0 1 0 1
  -------------
  1 1 1 1 0 1 1
```

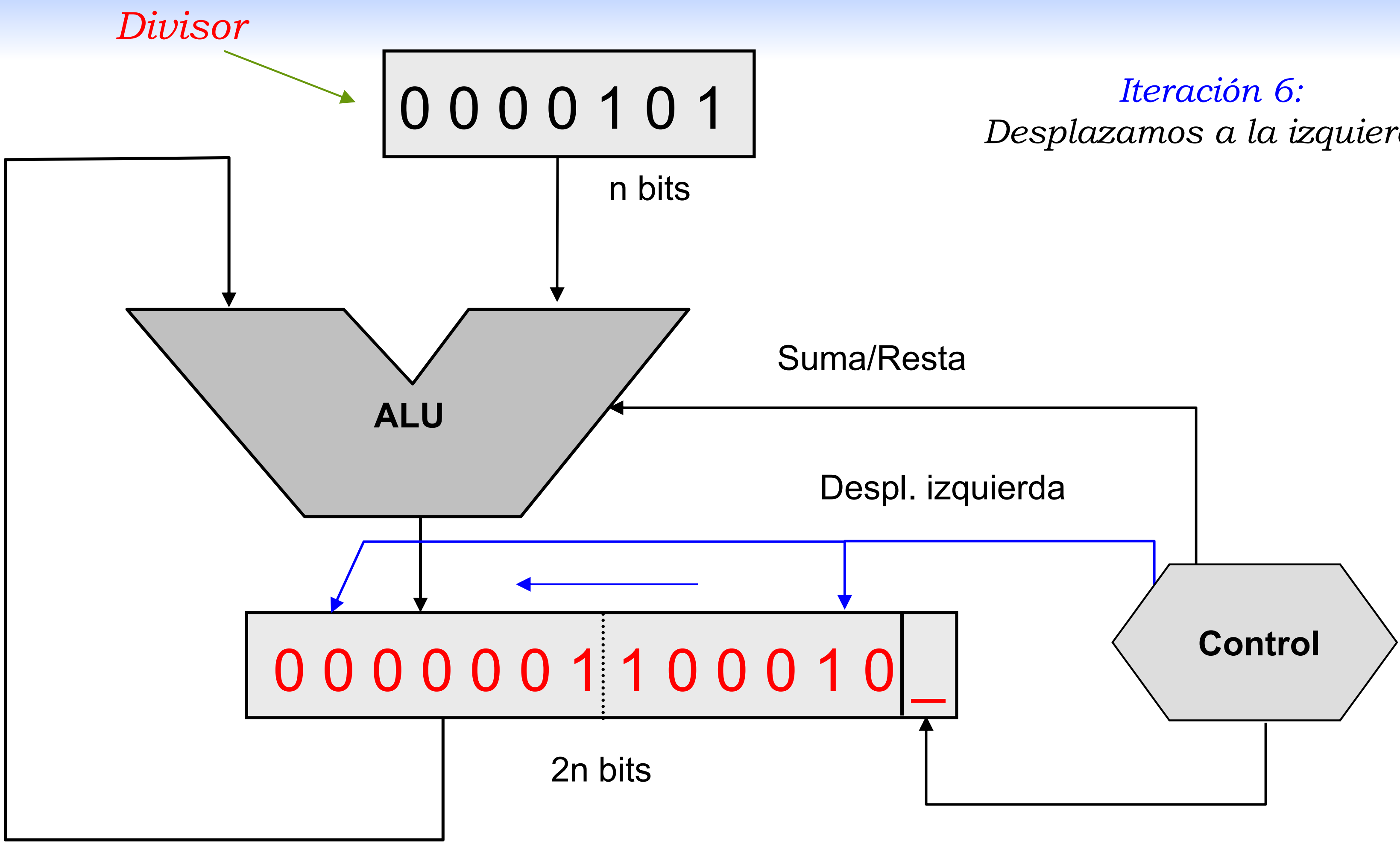
Unidad aritmética entera. Multiplicar y dividir

Iteración 5:
Dividendo negativo:
Completamos con un 0 por la derecha

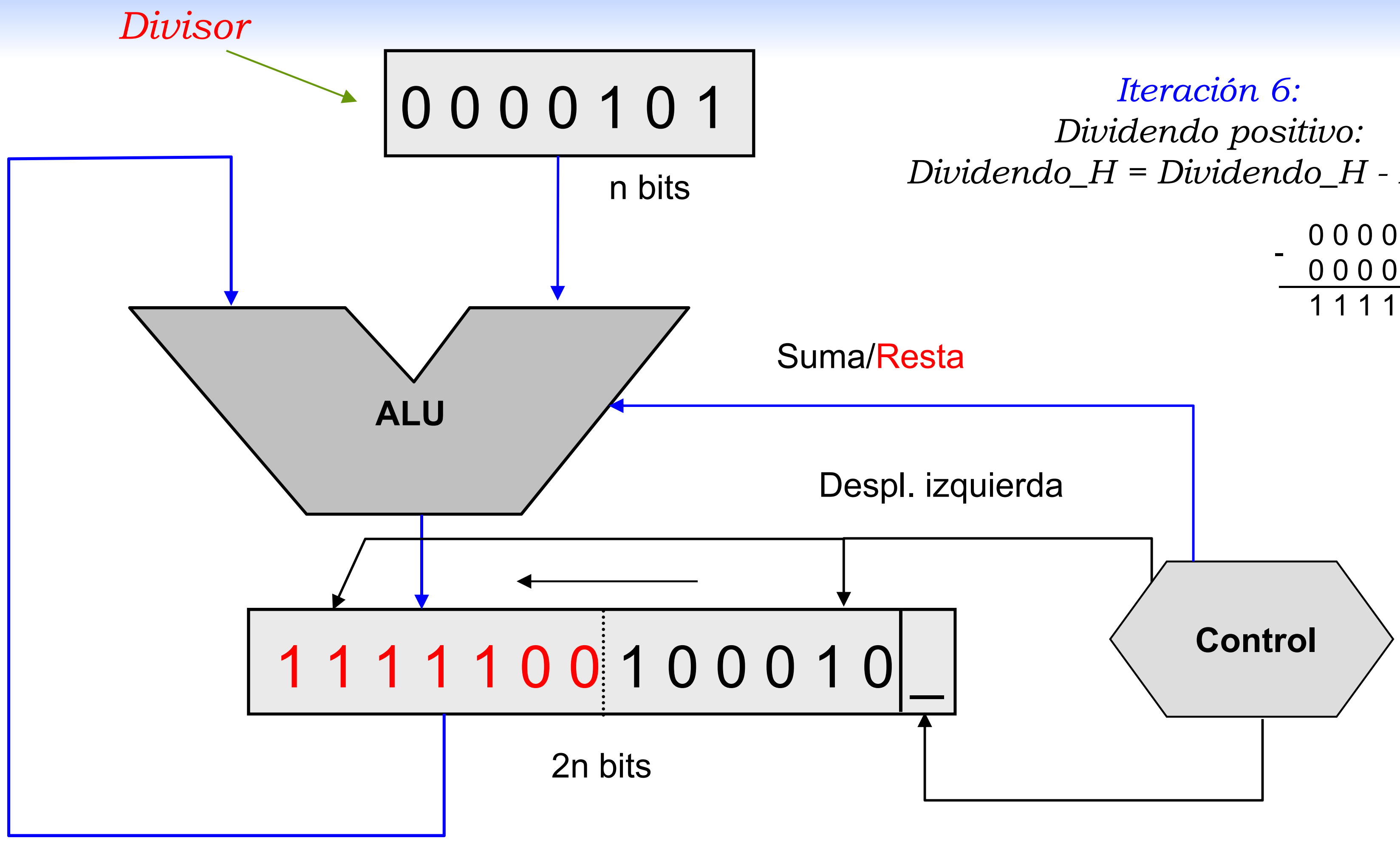


DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Iteración 6:
Desplazamos a la izquierda



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

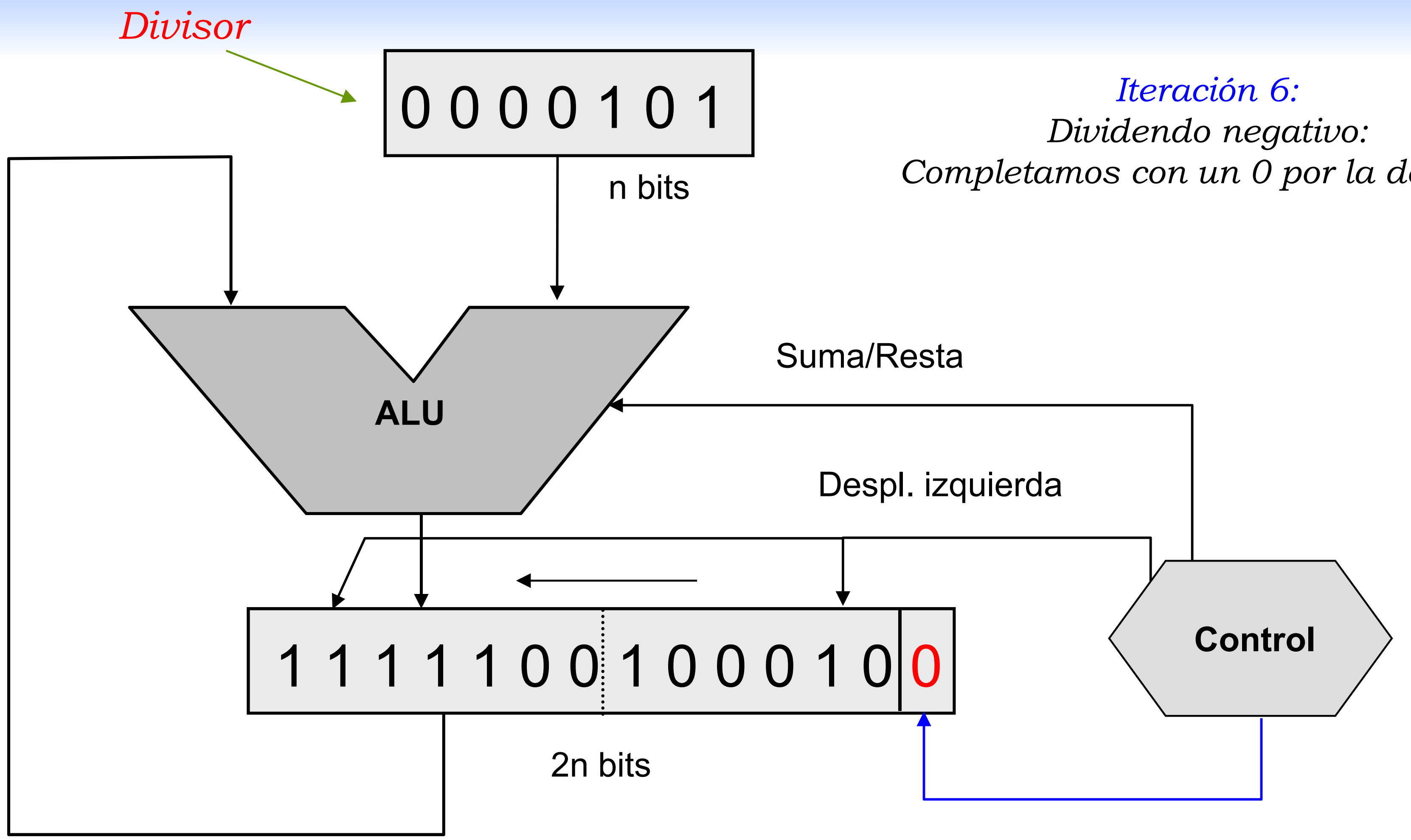


Iteración 6:
Dividendo positivo:
Dividendo_H = Dividendo_H - Divisor

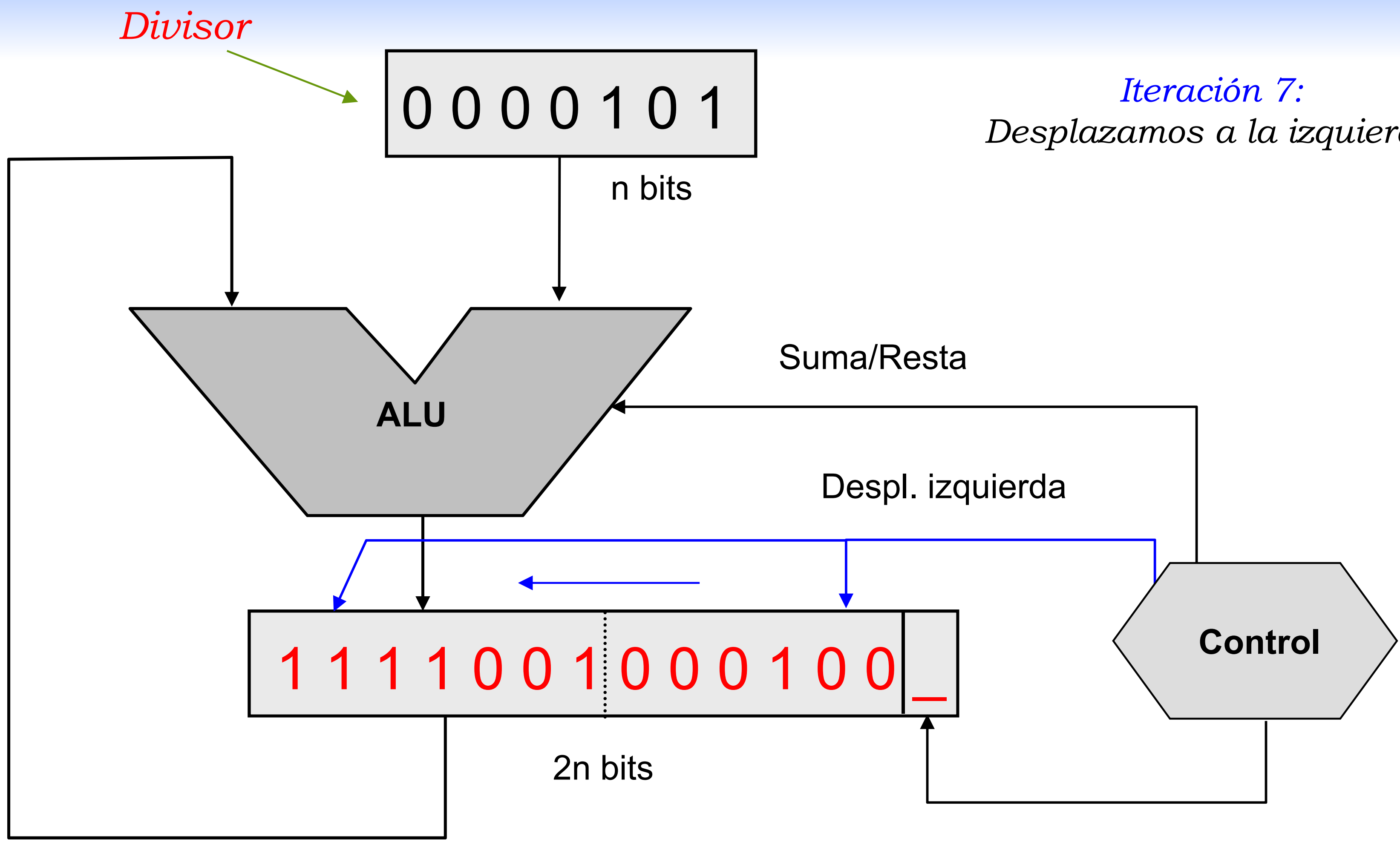
```
  0 0 0 0 0 0 1
- 0 0 0 0 1 0 1
  -------------
  1 1 1 1 1 0 0
```

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

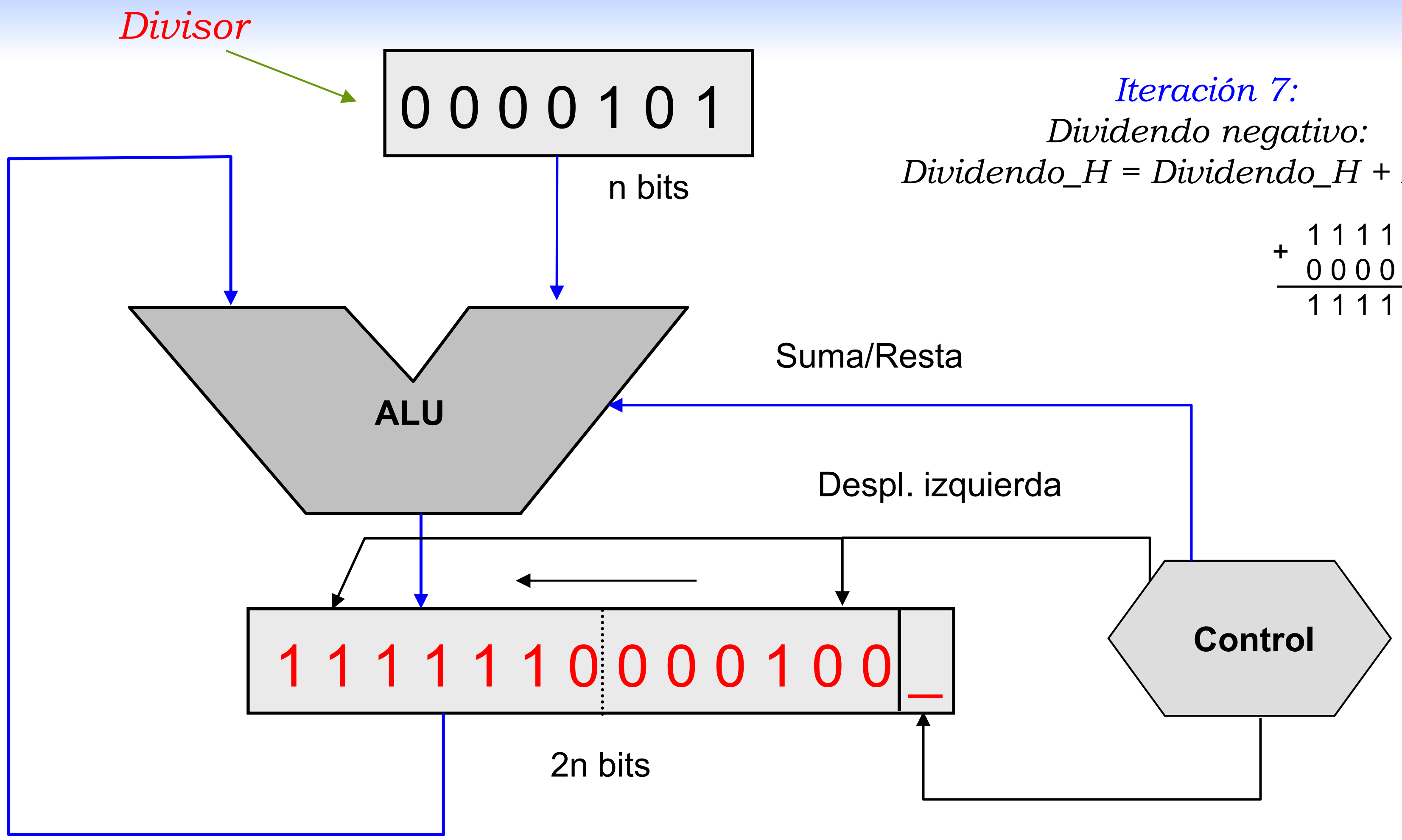
Iteración 6:
Dividendo negativo:
Completamos con un 0 por la derecha



Iteración 7:
Desplazamos a la izquierda



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN



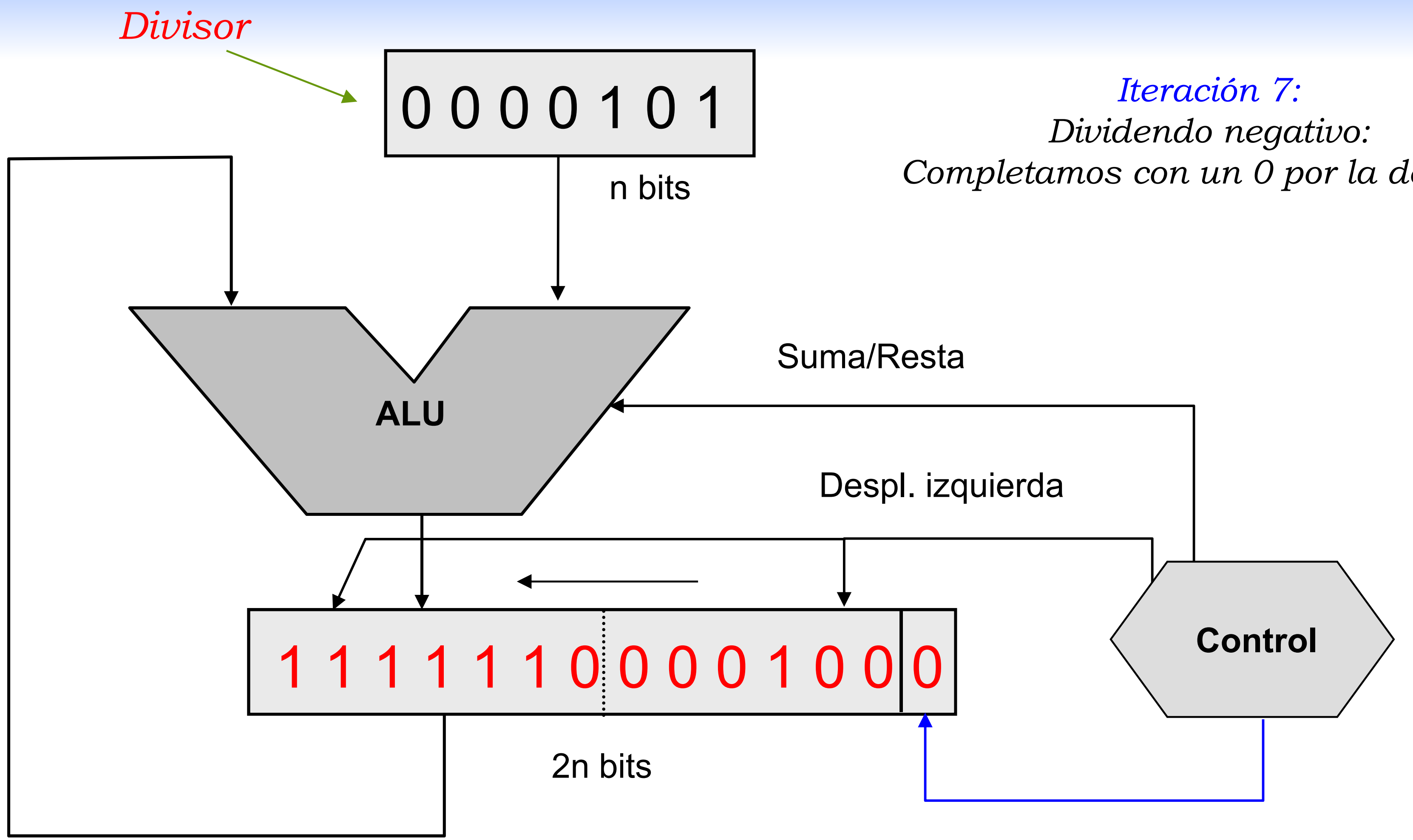
Iteración 7:
Dividendo negativo:
Dividendo_H = Dividendo_H + Divisor

```
  1 1 1 1 0 0 1
+ 0 0 0 0 1 0 1
  -------------
  1 1 1 1 1 1 0
```

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

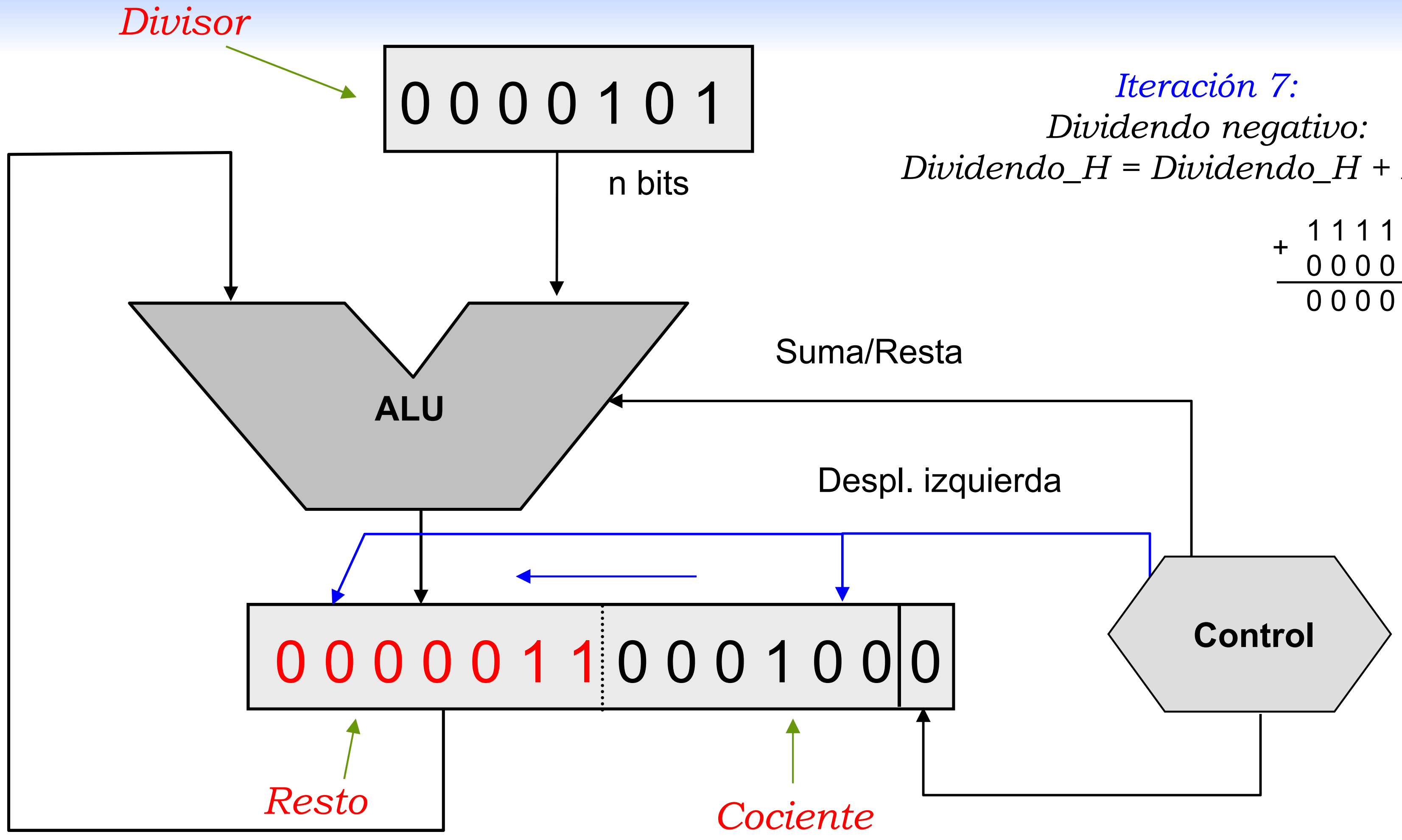
Unidad aritmética entera. Multiplicar y dividir

Iteración 7:
Dividendo negativo:
Completamos con un 0 por la derecha



DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Unidad aritmética entera. Multiplicar y dividir



Iteración 7:
Dividendo negativo:
Dividendo_H = Dividendo_H + Divisor

$$\begin{array}{r} 1111110 \\ +\ 0000101 \\ \hline 0000011 \end{array}$$

Esta última suma (restauración) solo la haremos en el caso de que el resto resulte negativo

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Dividendo	Divisor	Acción	Iteración
0000 0111 (7)	0010 (2)	Valores iniciales	0
1110 0111	0010	Dividendo_h - Divisor	0
1100 111_	0010	Desplazar Izda. $\text{Dividendo}_h < 0 \Rightarrow$	1
1110 111_	0010	Dividendo_h + Divisor	1
1110 111**0**	0010	$\text{Dividendo}_h < 0 \Rightarrow q_0 = 0$	1
1101 11**0**_	0010	Desplazar Izda. $\text{Dividendo}_h < 0 \Rightarrow$	2
1111 11**0**_	0010	Dividendo_h + Divisor	2
1111 1**100**	0010	$\text{Dividendo}_h < 0 \Rightarrow q_0 = 0$	2
1111 1**00**_	0010	Desplazar Izda. $\text{Dividendo}_h < 0 \Rightarrow$	3
0001 1**00**_	0010	Dividendo_h + Divisor	3
0001 1**001**	0010	$\text{Dividendo}_h >= 0 \Rightarrow q_0 = 1$	3
0011 **001**_	0010	Desplazar Izda. $\text{Dividendo}_h > 0 \Rightarrow$	4
0001 **001**_	0010	Dividendo_h - Divisor	4
0001 **0011**	0010	$\text{Dividendo}_h >= 0 \Rightarrow q_0 = 1$	4

↑ ↑

Resto(1) Cociente (3)

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Unidad aritmética entera. Multiplicar y dividir

Dividendo	Divisor	Acción	Iteración
00000 01101	00101	Valores iniciales (13÷5)	0
11011 01101	00101	DividendoH - Divisor	0
10110 1101_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	1
11011 1101_	00101	DividendoH + Divisor	1
11011 1101**0**	00101	Dividendo < 0 ⇒ q_0 = 0	1
10111 101**0**_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	2
11100 101**0**_	00101	DividendoH + Divisor	2
11100 101**00**	00101	Dividendo < 0 ⇒ q_0 = 0	2
11001 01**00**_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	3
11110 01**00**_	00101	DividendoH + Divisor	3
11110 01**000**	00101	Dividendo < 0 ⇒ q_0 = 0	3
11100 1**000**_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	4
00001 1**000**_	00101	DividendoH + Divisor	4
00001 1**0001**	00101	Dividendo > 0 ⇒ q_0 = 1	4
00011 **0001**_	00101	Desplazar Izquierda. Dividendo > 0 ⇒	5
11110 **0001**_	00101	DividendoH – Divisor	5
11110 **00010**	00101	Dividendo < 0 ⇒ q_0 = 0	5
00011 **00010**	00101	DividendoH < 0 ⇒ DividendoH + Divisor	5

Resto

Cociente

DIVISIÓN: ALGORITMO SIN RESTAURACIÓN

Unidad aritmética entera. Multiplicar y dividir

Dividendo	Divisor	Acción	Iteración
00000 01110	00100	Valores iniciales (14÷4)	0
11100 01110	00100	DividendoH - Divisor	0
11000 1110_	00100	Desplazar Izquierda. Dividendo < 0 ⇒	1
11100 1110_	00100	DividendoH + Divisor	1
11100 1110**0**	00100	Dividendo < 0 ⇒ q_0 = 0	1
11001 110**0**_	00100	Desplazar Izquierda. Dividendo < 0 ⇒	2
11101 110**0**_	00100	DividendoH + Divisor	2
11101 110**00**	00100	Dividendo < 0 ⇒ q_0 = 0	2
11011 10**00**_	00100	Desplazar Izquierda. Dividendo < 0 ⇒	3
11111 10**00**_	00100	DividendoH + Divisor	3
11111 10**000**	00100	Dividendo < 0 ⇒ q_0 = 0	3
11111 0**000**_	00100	Desplazar Izquierda. Dividendo < 0 ⇒	4
00011 0**000**_	00100	DividendoH + Divisor	4
00011 0**0001**	00100	Dividendo > 0 ⇒ q_0 = 1	4
00110 **0001**_	00100	Desplazar Izquierda. Dividendo > 0 ⇒	5
00010 **0001**_	00100	DividendoH – Divisor	5
00010 **00011**	00100	Dividendo > 0 ⇒ q_0 = 1	5

Resto

Cociente

- Representación para números fraccionarios
  - Coma fija 1234,567
  - Logarítmica log 123,456 = 2,0915122
  - Coma flotante 1,234566 x 10^3
- Ventajas de estandarizar una representación determinada
  - Posibilidad de disponer de bibliotecas de rutinas aritméticas
  - Técnicas de implementación en hardware de alto rendimiento
  - Construcción de aceleradores aritméticos estándar, etc.
- El estándar más empleado es el 754-1985 del IEEE.

Formatos

- Simple precisión (32 bits)

1 bit	8 bits	23 bits
signo	exponente	mantisa

- Doble precisión (64 bits)

1 bit	11 bits	52 bits
signo	exponente	mantisa

- Cuádruple precisión (128 bits)

1 bit	15 bits	112 bits
signo	exponente	mantisa

Unidad aritmética flotante. IEEE 754

- Base del exponente 2
- Exponente representado en exceso $2^{q-1}-1$
  - Exceso a 127 en simple precisión
  - Exceso a 1023 en doble precisión
- Mantisa en valor absoluto; fraccionaria y normalizada con un uno implícito a la izquierda de la coma decimal.
  - Mantisa de la forma 1,XXXXXX
  - El primer uno nunca estará representado
  - Valores posibles entre 1,00000..... y 1,11111....
- S es el signo de la mantisa
- El valor del número representado vendrá dado por:
  - $(-1)^S \times 1{,}M \times 2^{E-127}$ simple precisión
  - $(-1)^S \times 1{,}M \times 2^{E-1023}$ doble precisión

Unidad aritmética flotante. IEEE 754

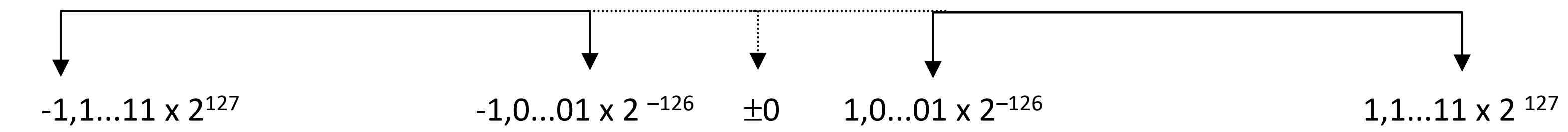
- Casos especiales

E	M	Valores
$2^{q}-1$	$\neq 0$	NaN (no un Número)
$2^{q}-1$	0	$+\infty$ y $-\infty$ según el signo de S
0	0	Cero
0	$\neq 0$	Números desnormalizados

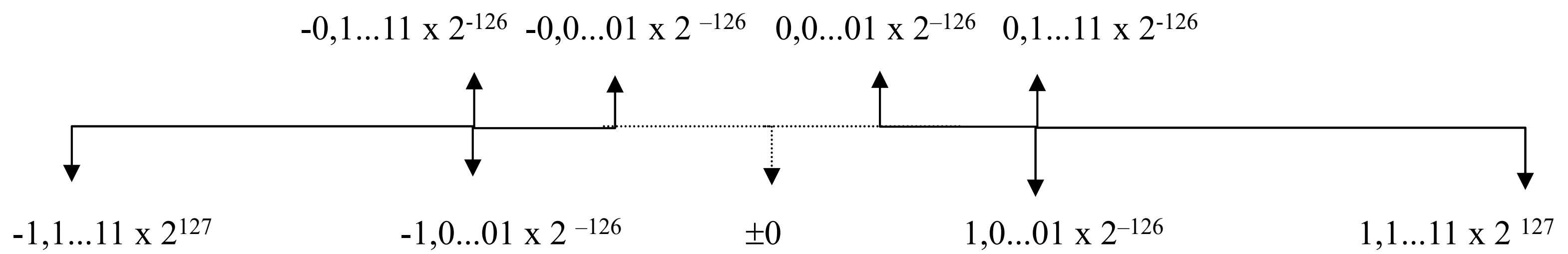
- NaN resultado de operaciones tales como 0/0, $\sqrt{-1}$
- El valor cero tiene dos representaciones +0 y –0.

ESTÁNDAR IEEE 754 PARA COMA FLOTANTE

- Formato desnormalizado
  - $0,M \times 2^{-126}$ simple precisión
  - $0,M \times 2^{-1022}$ doble precisión



Sin números desnormalizados



Con números desnormalizados

Procedimiento

- 1. Representar en coma fija el número decimal.
- 2. Pasar número decimal a binario.
- 3. Normalizar mantisa.
- 4. Normalizar exponente.
- 5. Representar en formato IEEE 754

Unidad aritmética flotante. Sumar y restar

Ejemplo: -0.81375×10^2

- 1. Representar en coma fija el número decimal.
  - -81.375
- 2. Pasar número decimal a binario.
  - Parte entera: 81 -> 1010001_2
  - Parte decimal: 0.375 -> 0.011_2
  - 1010001.011_2
- 3. Normalizar mantisa.
  - 1.010001011×2^6
- 4. Normalizar exponente.
  - E=6+127=133 -> 10000101_2
- 5. Representar en formato IEEE 754

1 bit	8 bits	23 bits
1	10000101	010001011000……….00

Reglas de Suma/Resta

- 1. Seleccionar el número de menor exponente y desplazar su mantisa hacia la derecha tantas posiciones como la diferencia de los exponentes en valor absoluto.
- 2. Igualar el exponente del resultado al exponente mayor.
- 3. Operar las mantisas (según operación seleccionada y signos de ambos números) y obtener el resultado en signo y valor absoluto.
- 4. Normalizar el resultado y redondear la mantisa al número de bits apropiado.

alineamiento

$$\begin{array}{l} 6.144\cdot 10^{2} \\ 9.975\cdot 10^{4} \end{array} \quad \xrightarrow{\text{alineamiento}} \quad \begin{array}{r} 0.06144\cdot 10^{4} \\ +\ 9.97500\cdot 10^{4} \\ \hline 10.03644\cdot 10^{4} \end{array} \quad \xrightarrow{\text{normalización}} \quad \begin{array}{r} 1.003644\cdot 10^{5} \\ +\ 0.000500\cdot 10^{5} \\ \hline 1.004\cdot 10^{5} \end{array}$$

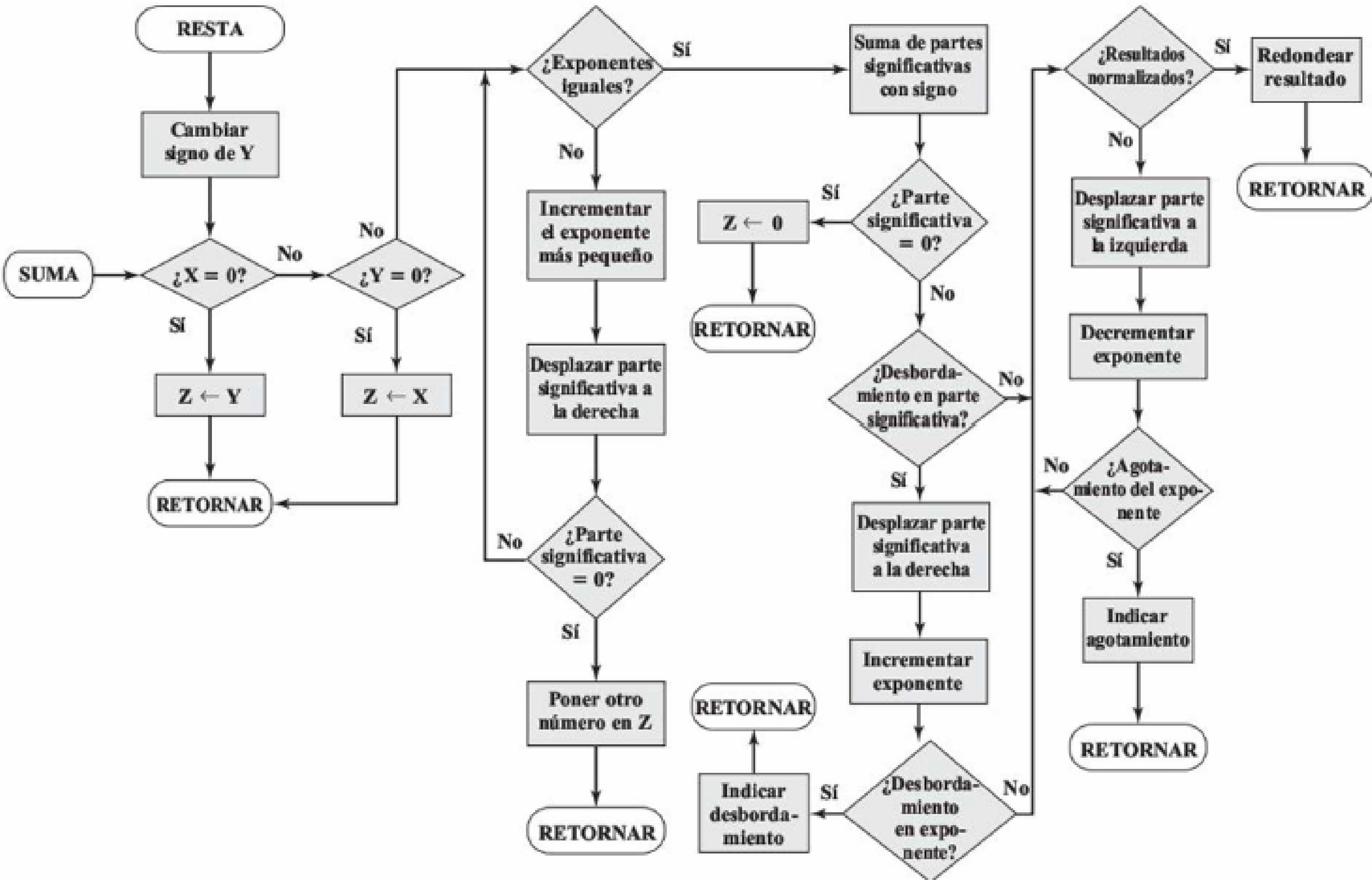
operación

redondeo

Unidad aritmética flotante. Sumar y restar

Selección de la operación real

SA	SB	Operación	Operación Real	Valores
0	0	Suma	A+B	0
0	1	Suma	A-B	Según resultado
1	0	Suma	B-A	Según resultado
1	1	Suma	A+B	1
0	0	Resta	A-B	Según resultado
0	1	Resta	A+B	0
1	0	Resta	A+B	1
1	1	Resta	A-B	Según resultado


RESTA
Cambiar signo de Y
SUMA
¿X = 0?
No
Sí
Z ← Y
RETORNAR
¿Y = 0?
No
Sí
Z ← X
¿Exponentes iguales?
Sí
No
Incrementar el exponente más pequeño
Desplazar parte significativa a la derecha
¿Parte significativa = 0?
No
Sí
Poner otro número en Z
RETORNAR
Suma de partes significativas con signo
¿Parte significativa = 0?
Sí
Z ← 0
RETORNAR
No
¿Desbordamiento en parte significativa?
No
Sí
Desplazar parte significativa a la derecha
Incrementar exponente
¿Desbordamiento en exponente?
Sí
No
Indicar desbordamiento
RETORNAR
¿Resultados normalizados?
Sí
Redondear resultado
RETORNAR
No
Desplazar parte significativa a la izquierda
Decrementar exponente
¿Agotamiento del exponente
No
Sí
Indicar agotamiento
RETORNAR

Circuito Sumador / Restador

- Tenemos 2 números A y B

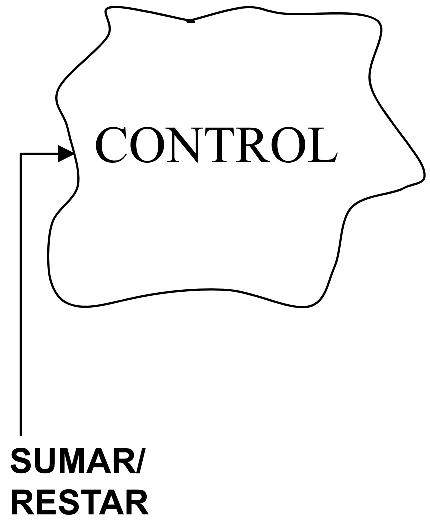
S_A	E_A	M_A

S_B	E_B	M_B

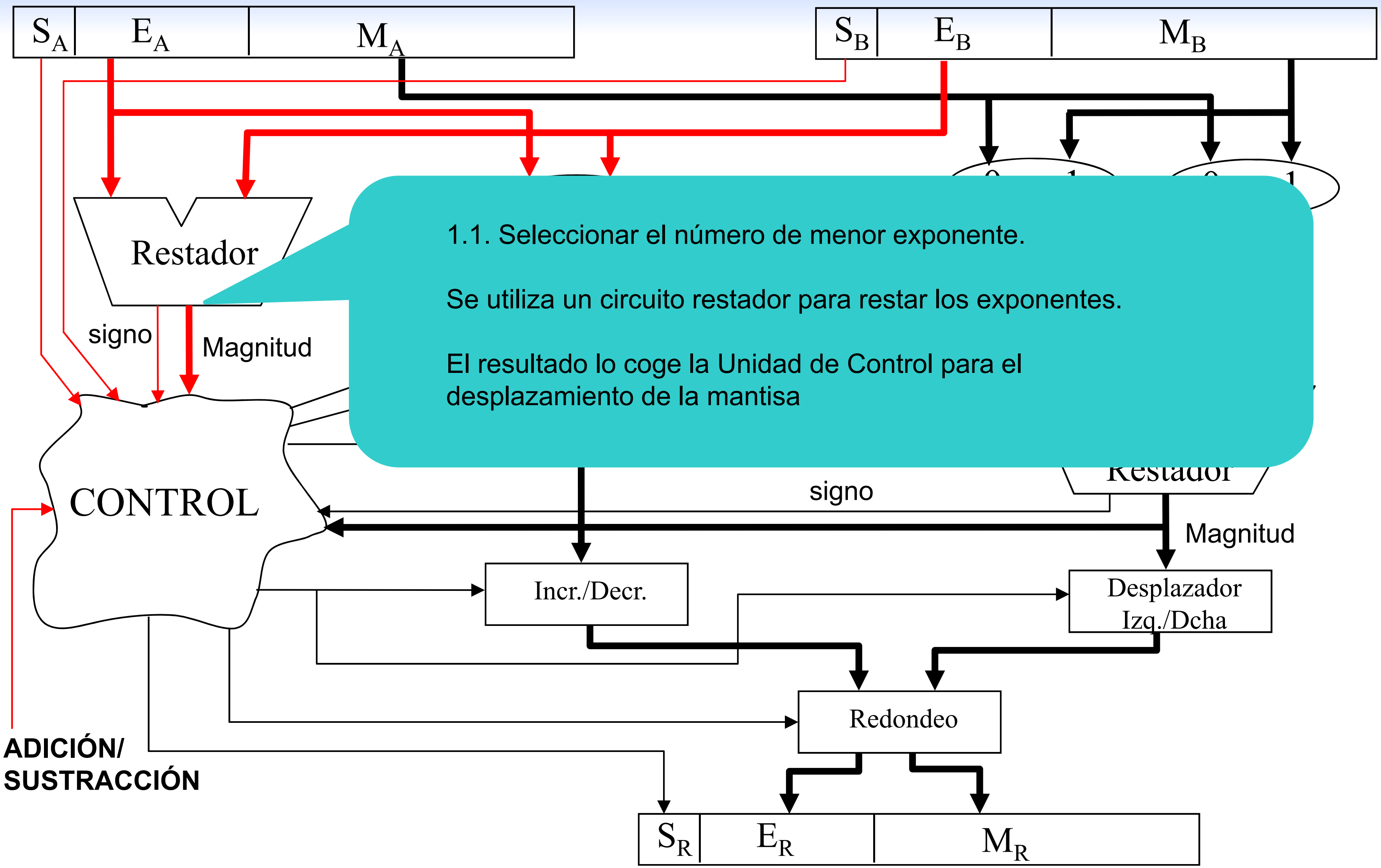
- Queremos sumar/restar
  - R = A + B
  - R = A – B

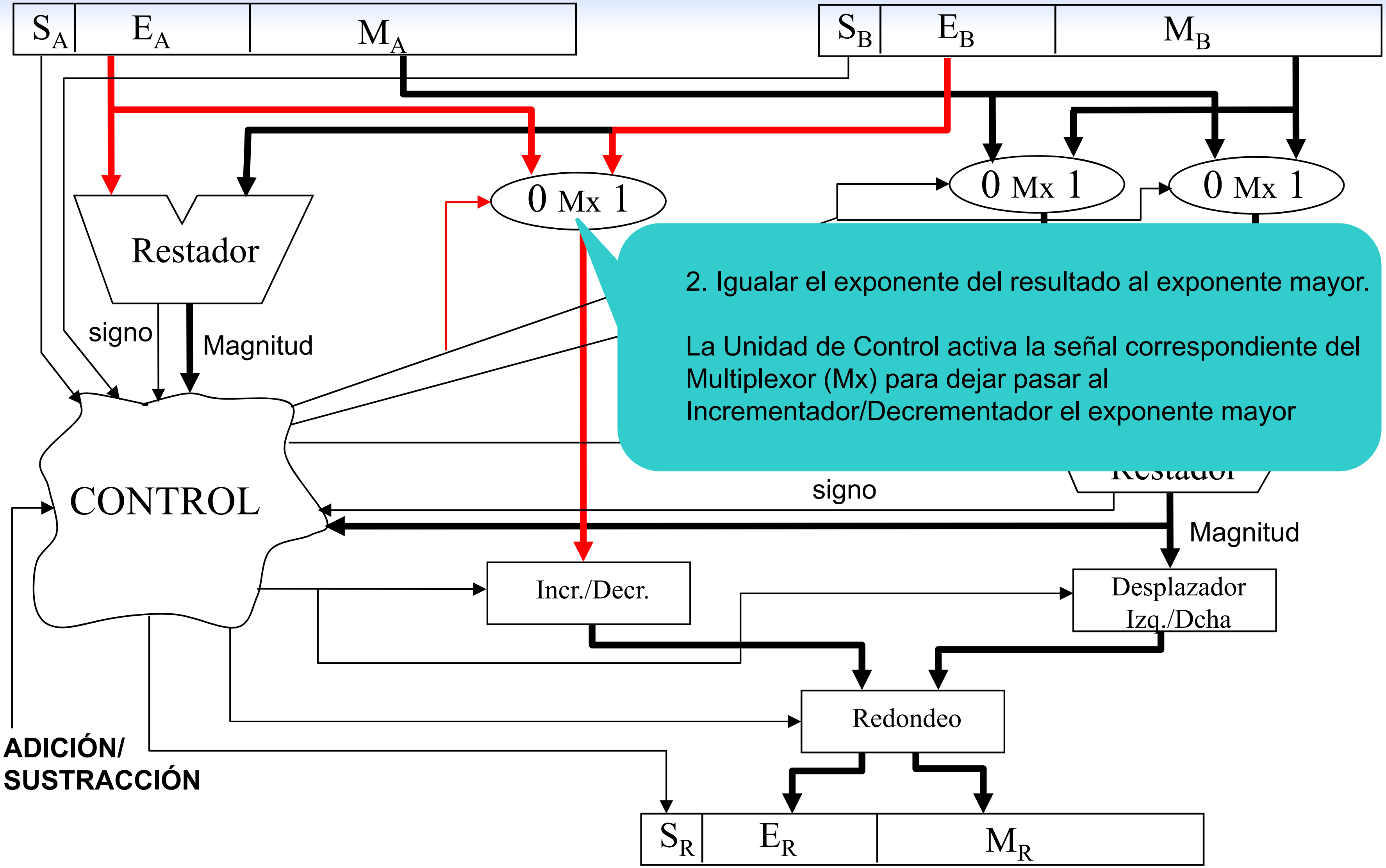
S_R	E_R	M_R

- La unidad de control se encarga
de realizar la operación (SUMAR/RESTAR)
Activando sus señales de salida



Unidad aritmética flotante. Sumar y restar





2. Igualar el exponente del resultado al exponente mayor.

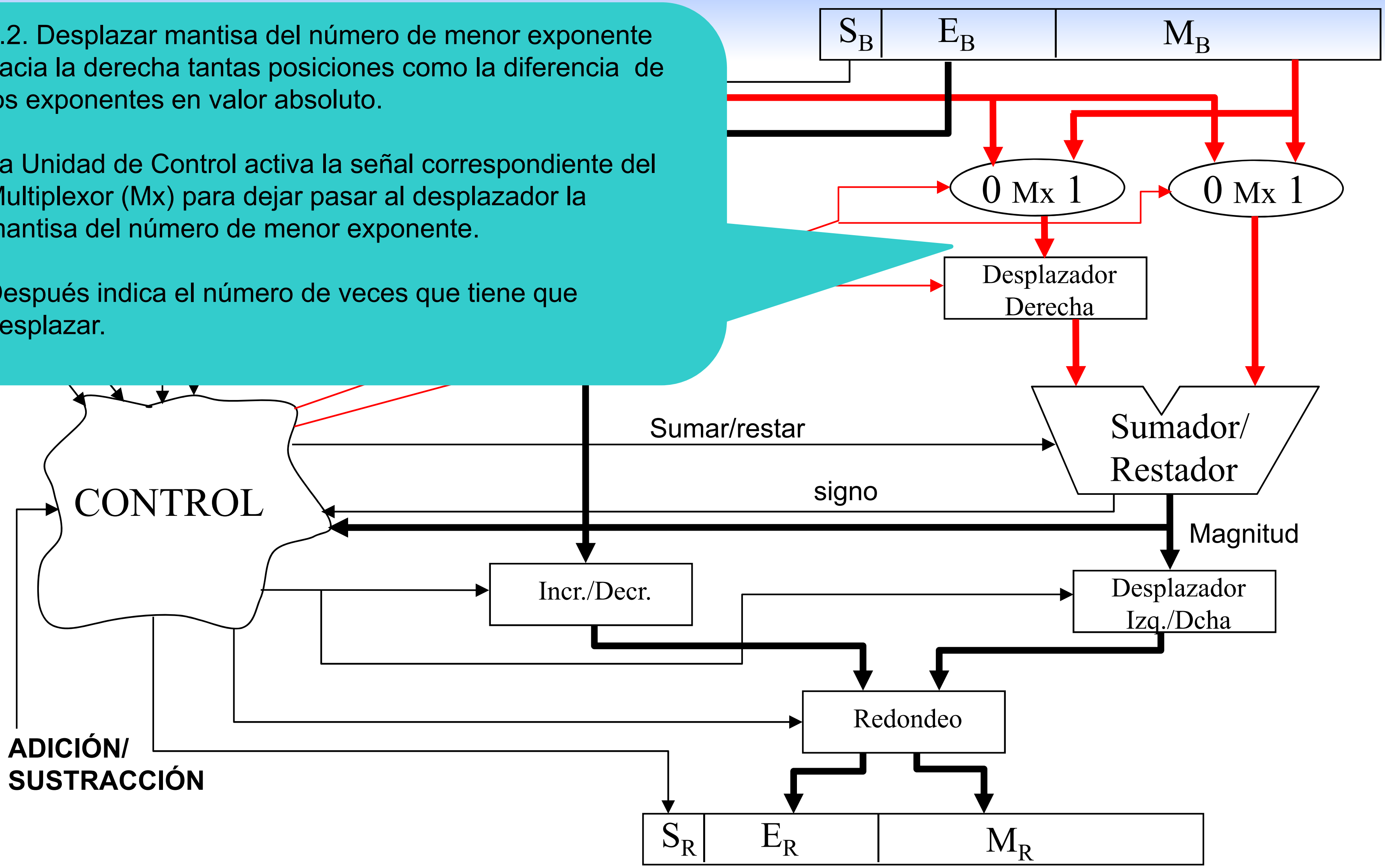
La Unidad de Control activa la señal correspondiente del Multiplexor (Mx) para dejar pasar al Incrementador/Decrementador el exponente mayor

Unidad aritmética flotante. Sumar y restar

1.2. Desplazar mantisa del número de menor exponente hacia la derecha tantas posiciones como la diferencia de los exponentes en valor absoluto.

La Unidad de Control activa la señal correspondiente del Multiplexor (Mx) para dejar pasar al desplazador la mantisa del número de menor exponente.

Después indica el número de veces que tiene que desplazar.



Circuito Sumador/Restador

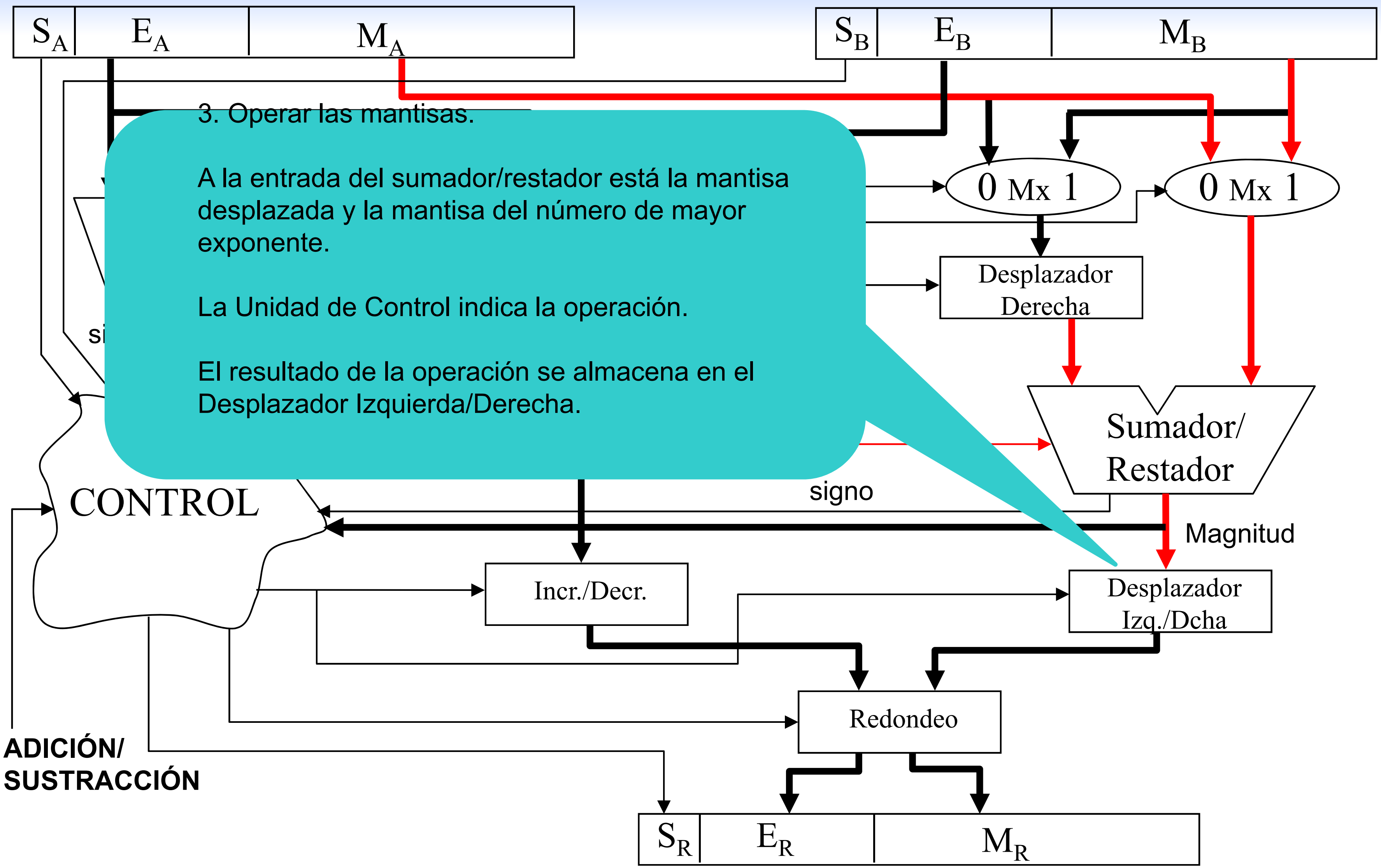
Unidad aritmética flotante. Sumar y restar

3. Operar las mantisas.

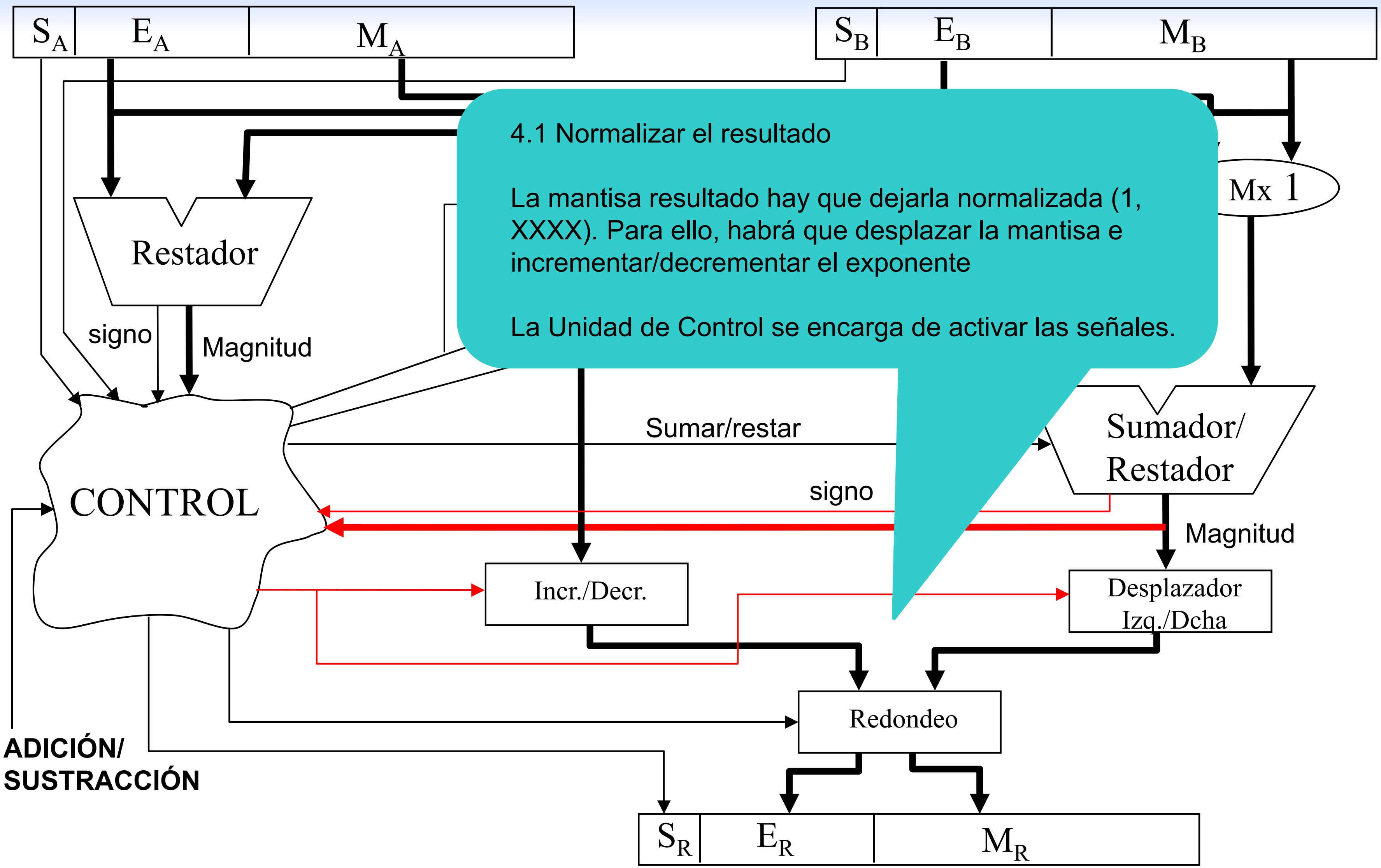
A la entrada del sumador/restador está la mantisa desplazada y la mantisa del número de mayor exponente.

La Unidad de Control indica la operación.

El resultado de la operación se almacena en el Desplazador Izquierda/Derecha.



Unidad aritmética flotante. Sumar y restar

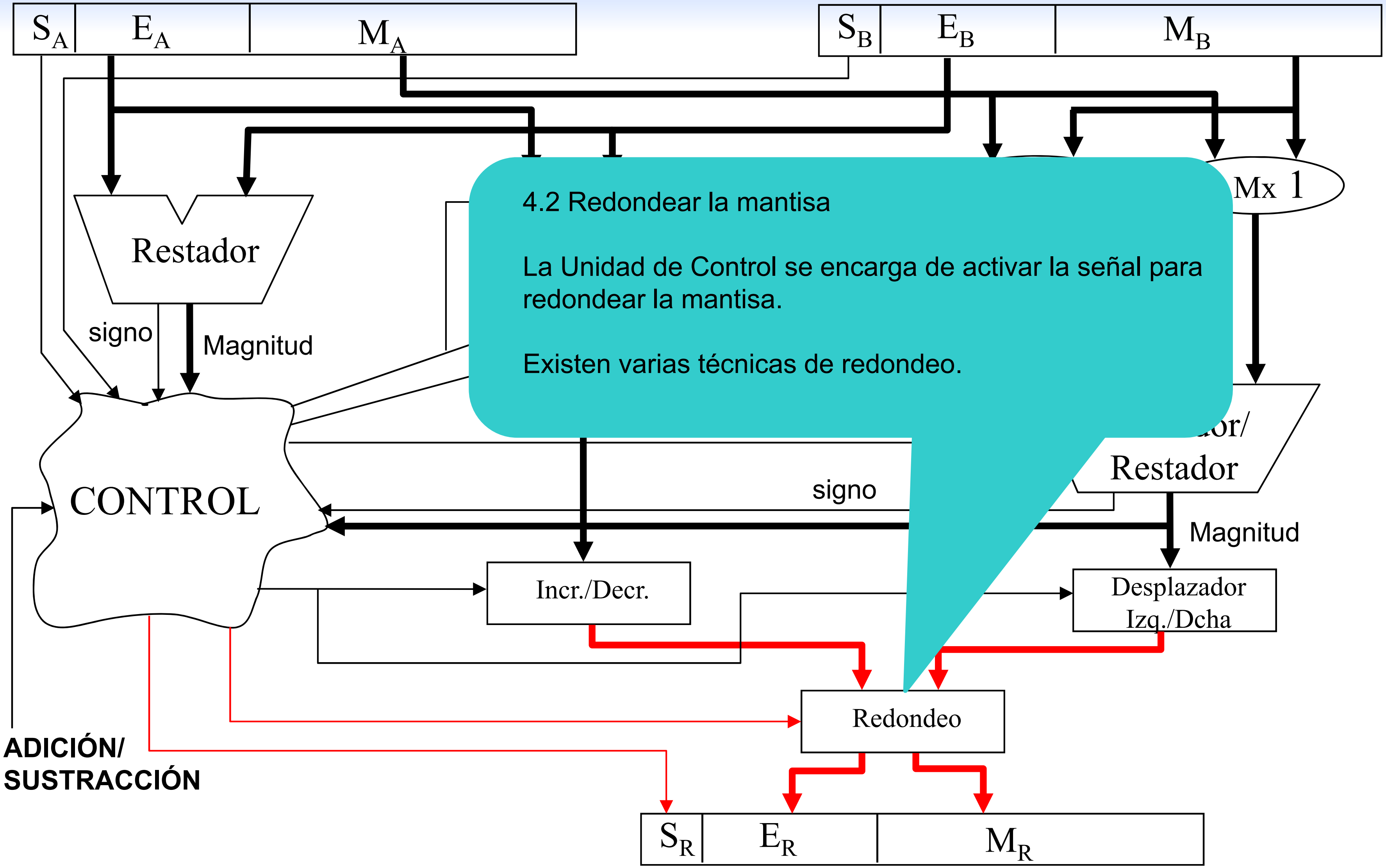


4.1 Normalizar el resultado

La mantisa resultado hay que dejarla normalizada (1, XXXX). Para ello, habrá que desplazar la mantisa e incrementar/decrementar el exponente

La Unidad de Control se encarga de activar las señales.

Unidad aritmética flotante. Sumar y restar



4.2 Redondear la mantisa

La Unidad de Control se encarga de activar la señal para redondear la mantisa.

Existen varias técnicas de redondeo.

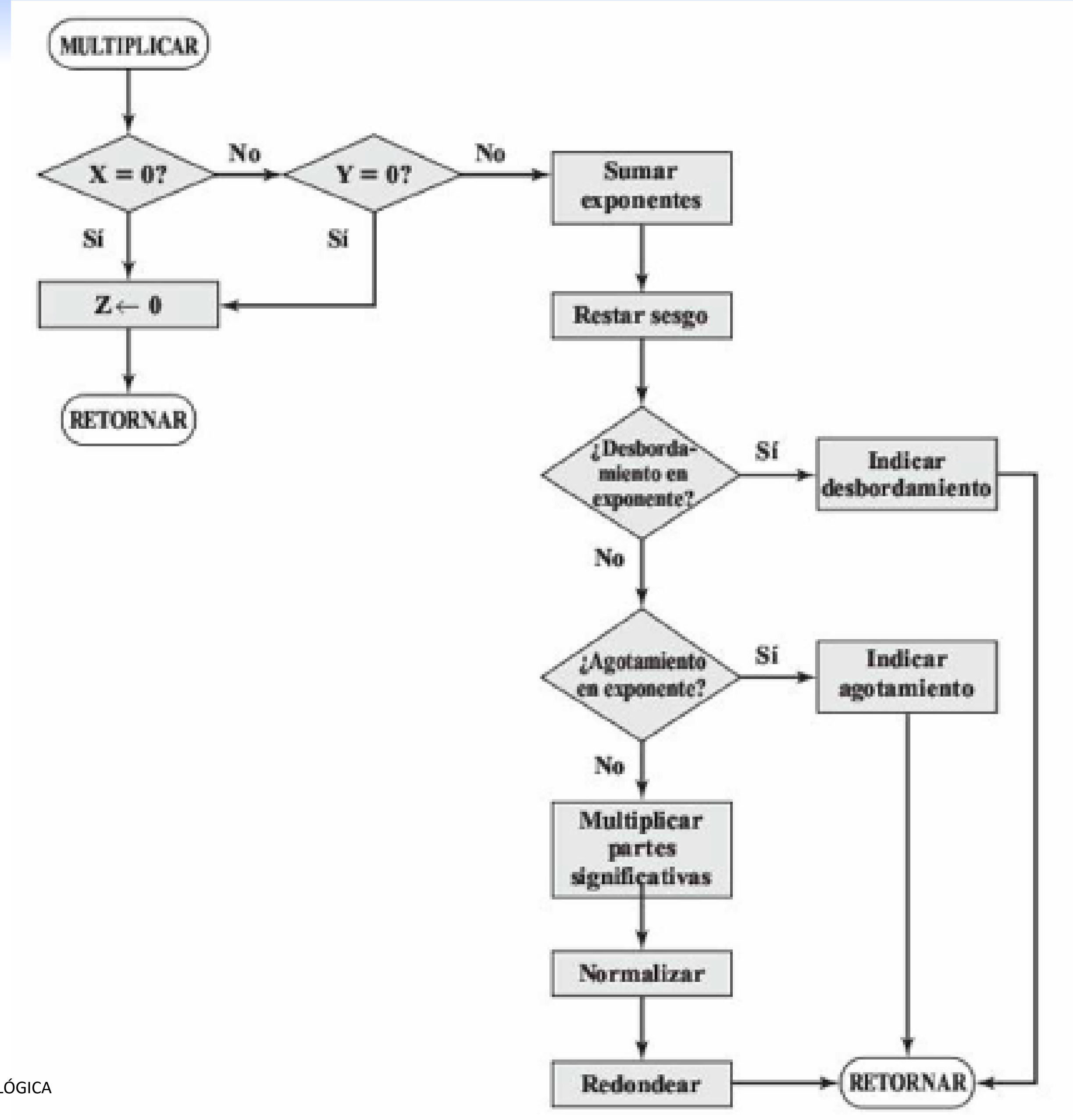
Unidad aritmética flotante. Multiplicar y dividir

Reglas de Multiplicación

- 1. Sumar los exponentes y restar el exceso para obtener el exponente del resultado
- 2. Multiplicar las mantisas para determinar la mantisa del resultado
- 3. Procesar los signos
- 4. Normaliza y redondear si es necesario

Resultado → (s1 ⊕ s2) (1.MANT1 * 1.MANT2) x 2 $^{(e1 + e2 - sesgo)}$

La Multiplicación en Coma Flotante



Unidad aritmética flotante. Multiplicar y dividir

- Reglas de División
  - 1. Restar los exponentes y sumar el exceso para obtener el exponente resultado
  - 2. Dividir las mantisas para determinar la mantisa del resultado.
  - 3. Procesar los signos.
  - 4. Normalizar y redondear si es necesario.

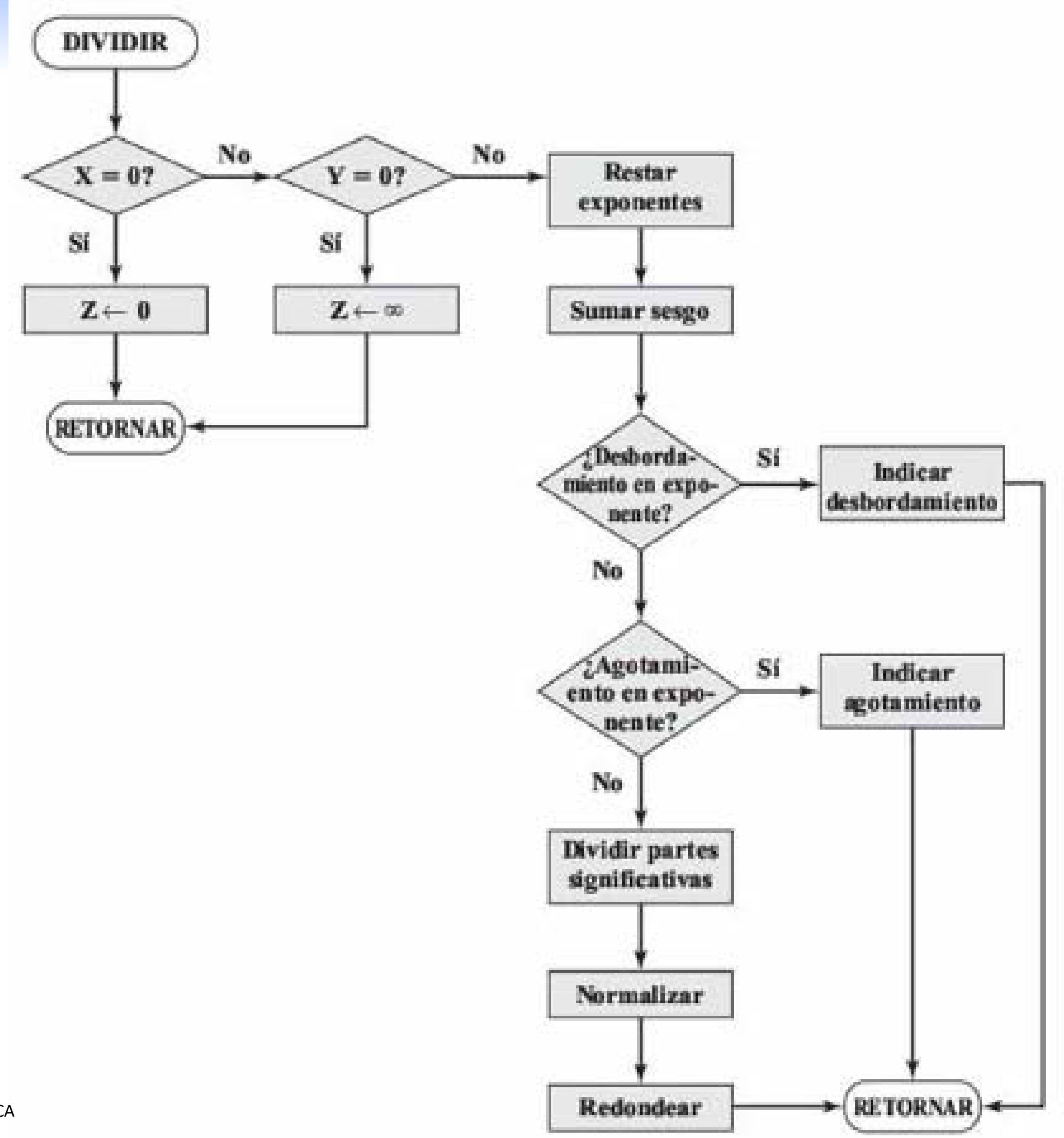
Resultado ➔ (s1 ⊕ s2) (1.MANT1 / 1.MANT2) x 2 $^{(e1 - e2 + sesgo)}$

- Procesamiento de los signos

S_A	S_B	S_R
0	0	0
0	1	1
1	0	1
1	1	0

$$S_R = S_A \oplus S_B$$

La División en Coma Flotante



- Tenemos 2 números A y B

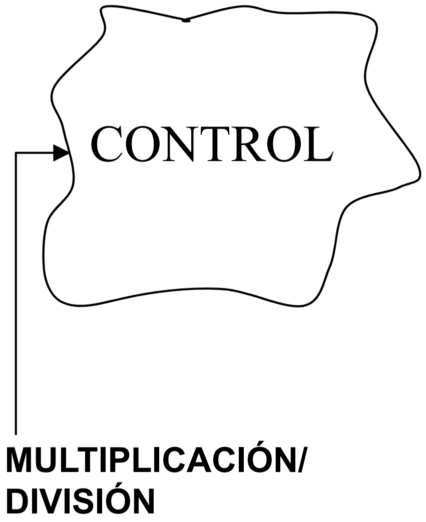
- Queremos multiplicar/dividir
  - $R = A * B$
  - $R = A / B$

S_A	E_A	M_A

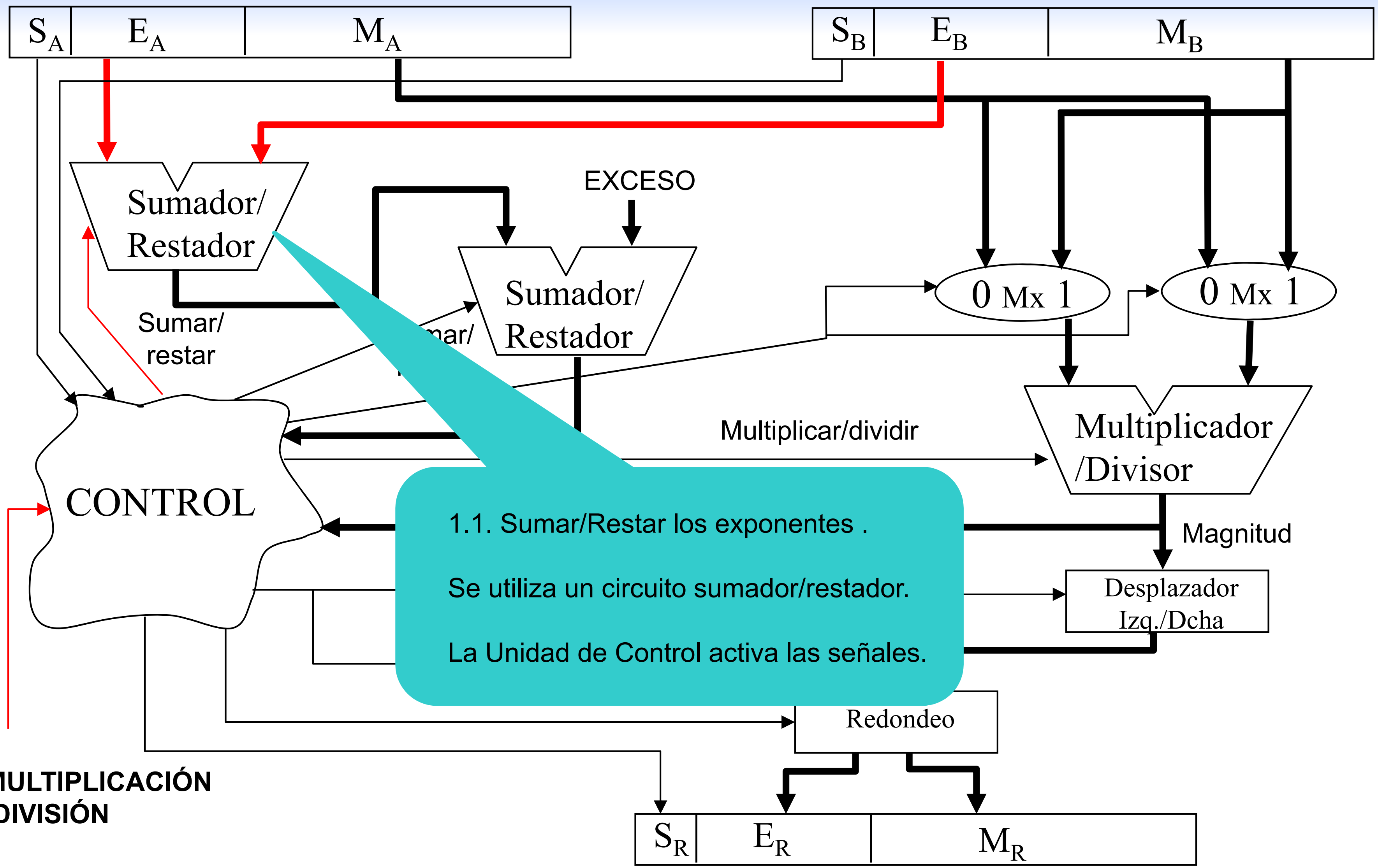
S_B	E_B	M_B

S_R	E_R	M_R

- La unidad de control se encarga
de realizar la operación (MULTIPLICAR/DIVIDIR)
Activando sus señales de salida



Unidad aritmética flotante. Multiplicar y dividir



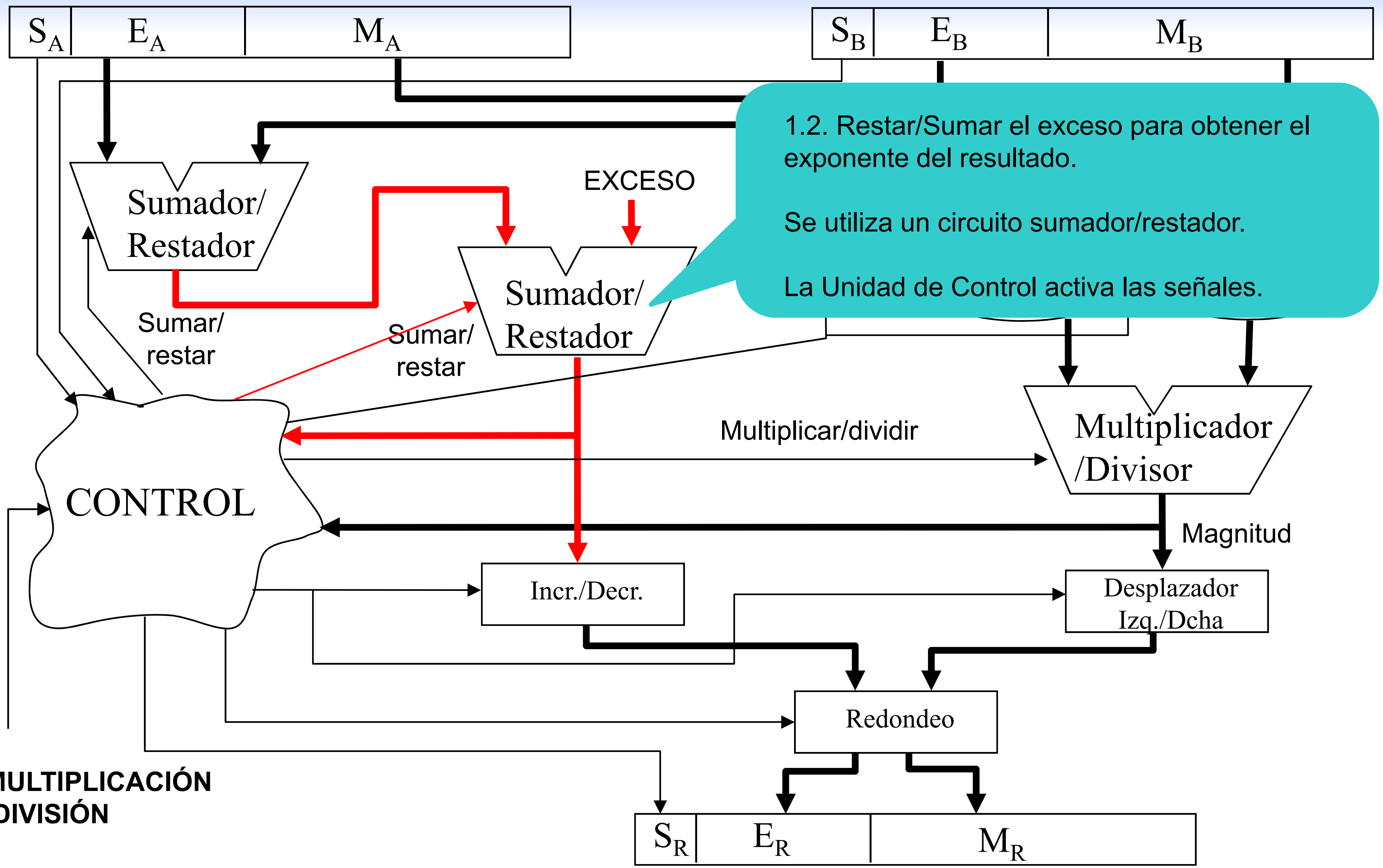
1.1. Sumar/Restar los exponentes .

Se utiliza un circuito sumador/restador.

La Unidad de Control activa las señales.

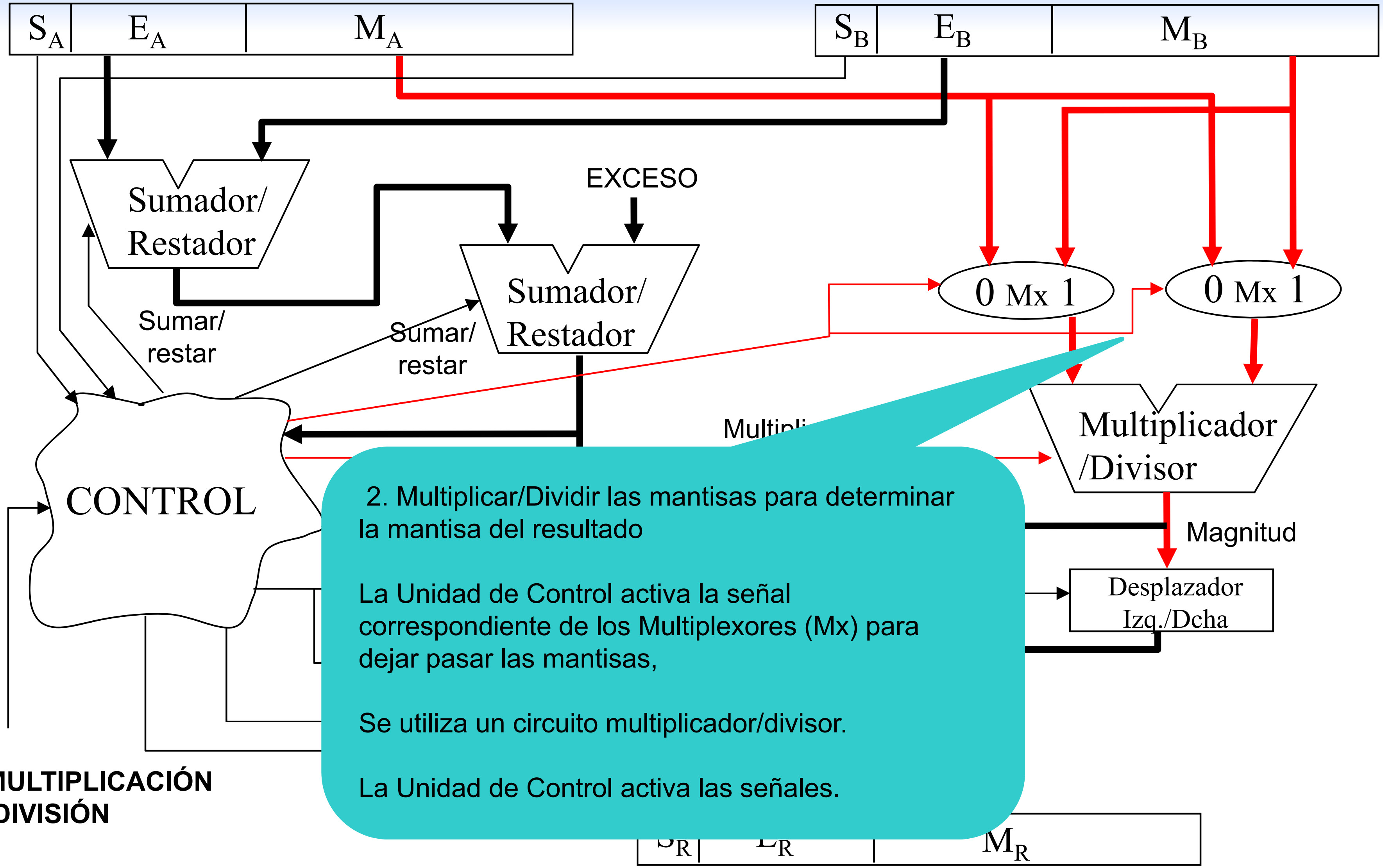
MULTIPLICACIÓN /DIVISIÓN

Unidad aritmética flotante. Multiplicar y dividir



MULTIPLICACIÓN /DIVISIÓN

CIRCUITO MULTIPLICADOR/DIVISOR



MULTIPLICACIÓN /DIVISIÓN

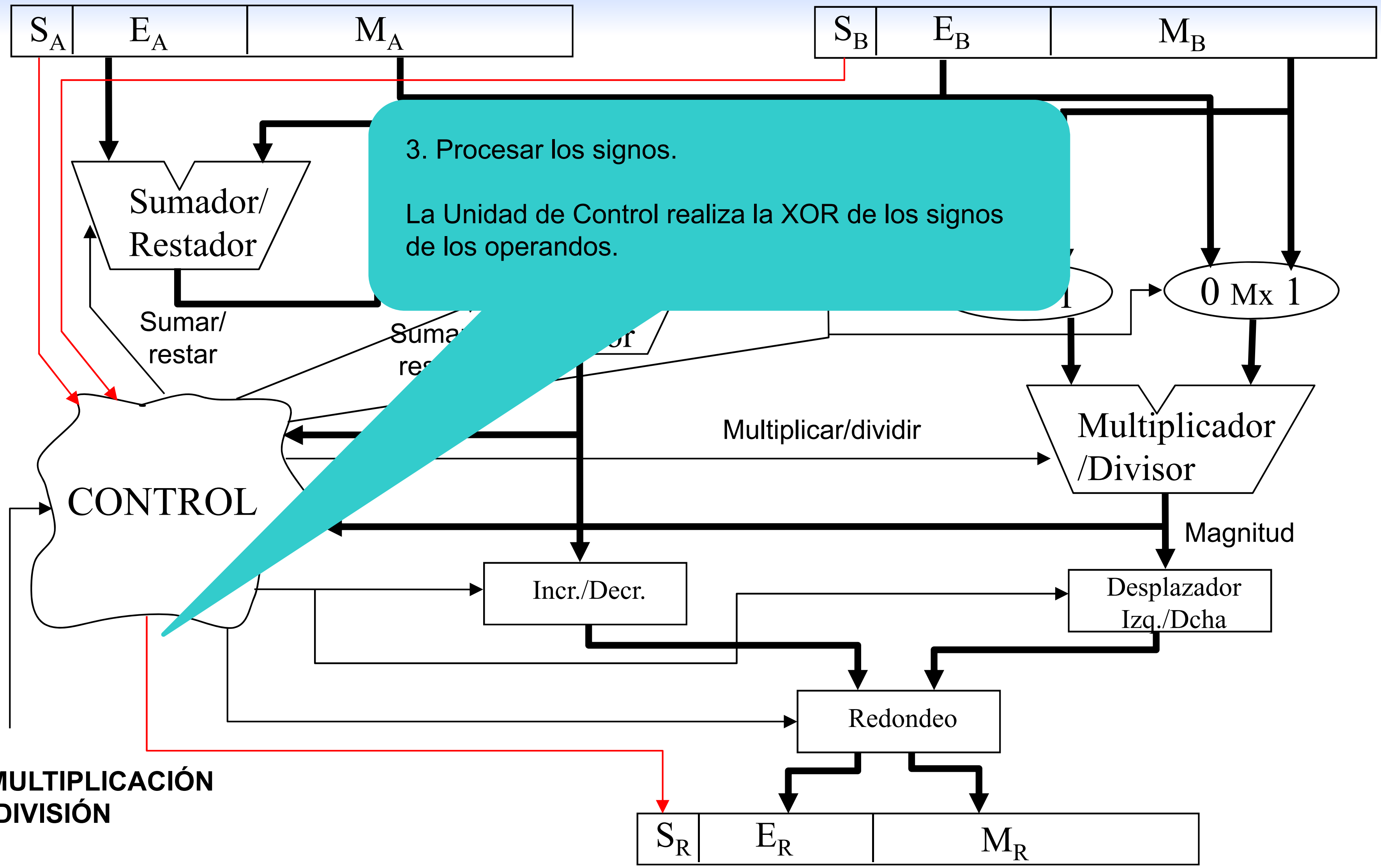
2. Multiplicar/Dividir las mantisas para determinar la mantisa del resultado

La Unidad de Control activa la señal correspondiente de los Multiplexores (Mx) para dejar pasar las mantisas,

Se utiliza un circuito multiplicador/divisor.

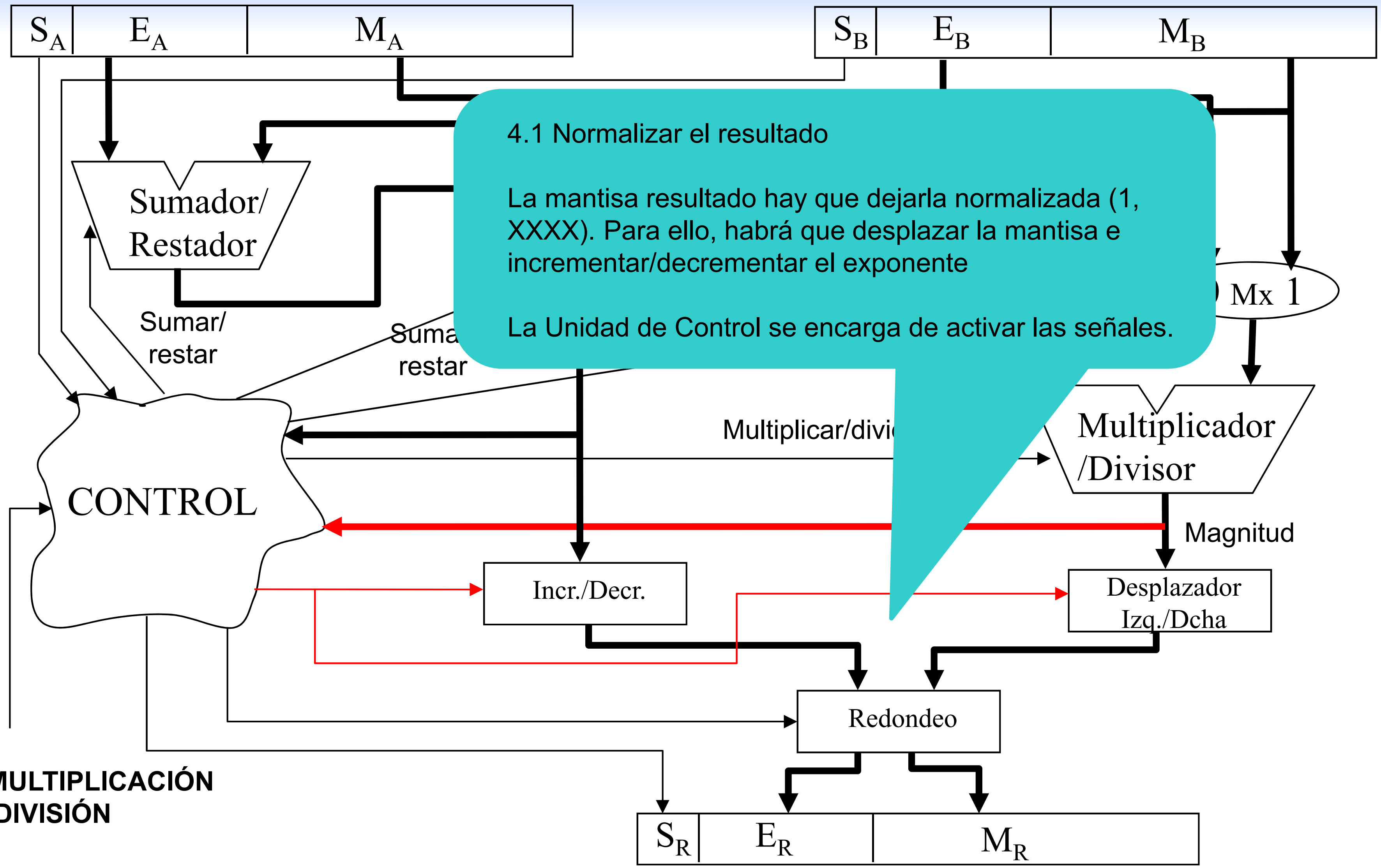
La Unidad de Control activa las señales.

Unidad aritmética flotante. Multiplicar y dividir



MULTIPLICACIÓN /DIVISIÓN

Unidad aritmética flotante. Multiplicar y dividir



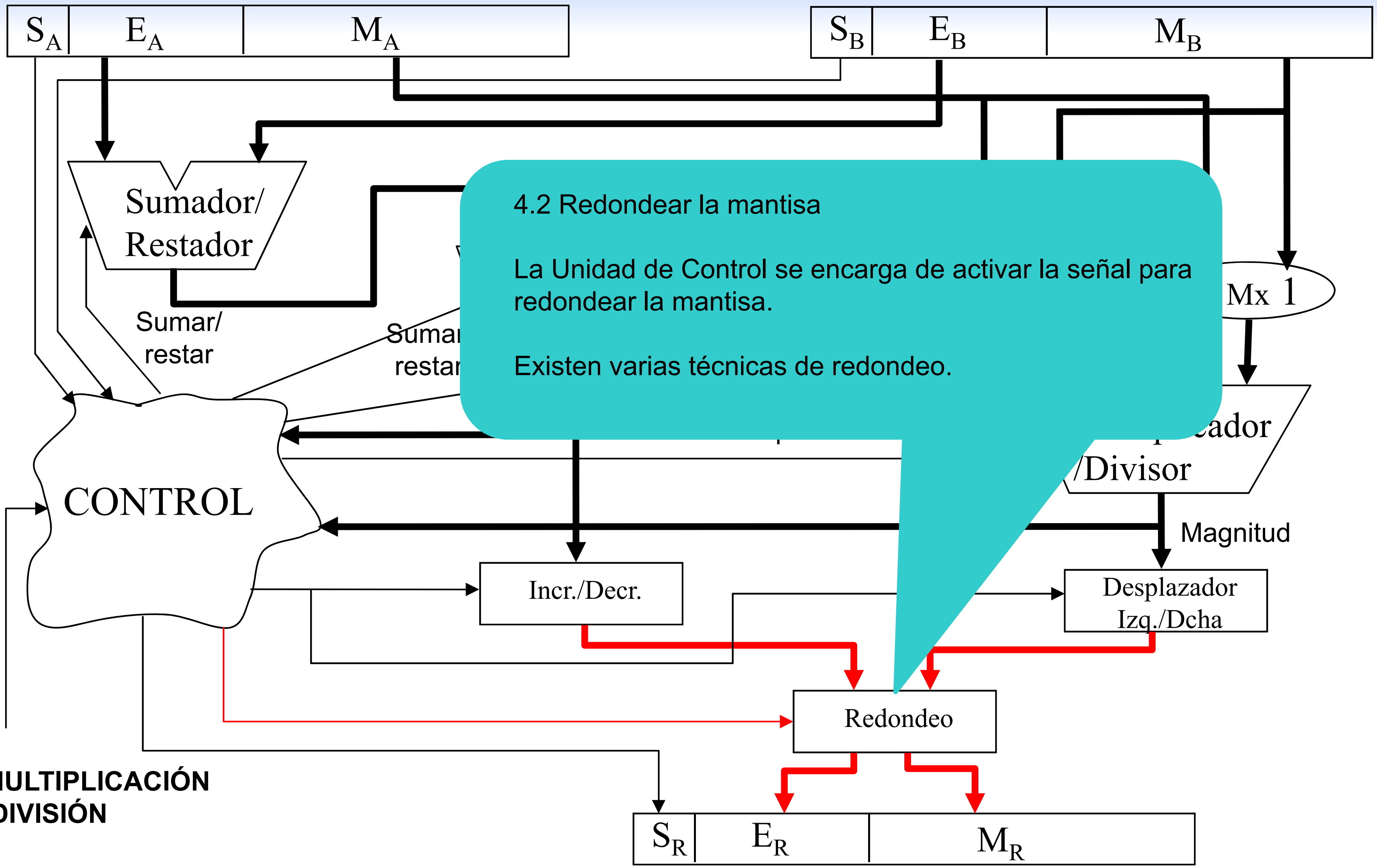
4.1 Normalizar el resultado

La mantisa resultado hay que dejarla normalizada (1, XXXX). Para ello, habrá que desplazar la mantisa e incrementar/decrementar el exponente

La Unidad de Control se encarga de activar las señales.

MULTIPLICACIÓN /DIVISIÓN

Unidad aritmética flotante. Multiplicar y dividir



4.2 Redondear la mantisa

La Unidad de Control se encarga de activar la señal para redondear la mantisa.

Existen varias técnicas de redondeo.

MULTIPLICACIÓN /DIVISIÓN

TÉCNICAS DE REDONDEO

- Las técnicas de redondeo consisten en limitar el número de bits al disponible en el sistema de representación utilizado.
- Normalmente las ALU utilizan **bits de guarda** que luego deben eliminar.
- Dada una cantidad **C**, y un sistema de representación que permite representar los valores $\mathbf{V_0, V_1, \ldots V_r}$.
- El redondeo consiste en asignar a **C** una representación **R** que se le aproxime. Si $V_{i-1} < C < V_i$ el redondeo consiste en asignar $\mathbf{V_{i-1}}$ o $\mathbf{V_i}$ como representación R de la cantidad C
- El error absoluto se define como: $\boldsymbol{\varepsilon = |R - C|}$
- Técnicas de redondeo
  - Truncamiento
  - Redondeo propiamente dicho
  - Bit menos significativo forzado a "uno"

Técnicas de Redondeo: Truncamiento

- Elimina los bits a la derecha que no caben en la representación.
  - Es fácil de implementar.
  - El error del resultado es siempre por defecto.
  - El error puede crecer rápidamente (p. ej. en operaciones consecutivas)

TÉCNICAS DE REDONDEO: REDONDEO AL MÁS PRÓXIMO

- El resultado de redondea al **número representable** más próximo:

$$\text{Si } |V_{i-1} - C| < |V_i - C| \rightarrow R \equiv V_{i-1} \text{ si no } V_i$$

Ejemplo: representación con 8 bits de punto implícito

$C = 0{,}01100000\ 01 \equiv 0{,}375976563$

$V_{i-1} = 0{,}01100000 \equiv 0{,}375$ (número representable anterior)

$V_i = 0{,}01100001 \equiv 0{,}37890625$ (número representable posterior)

$|V_{i-1} - C| = 0{,}000976563 \longrightarrow R = 0{,}01100000$ (más próximo)

$|V_i - C| = 0{,}00390625$

Técnicas de Redondeo: Bit menos significativo forzado a 1

- Consiste en truncar y forzar el bit menos significativo a “uno”
  - Es muy rápido, tanto como el truncamiento
  - Sus errores son tanto por defecto como por exceso.

EJERCICIO 1

- Sumar los números A y B según el estándar IEEE 754 en simple precisión teniendo en cuenta las tres técnicas de redondeo.

A	0	01111111	000000000000……….00
B	0	01100110	000000000000……….00

EJERCICIO 2

Ejercicios

- Obtener el resultado de la operación A+B y AxB en el formato IEEE 754 de los siguientes números representados en este formato. Para obtener el resultado especificar los pasos seguidos utilizando el algoritmo de suma y multiplicación estudiado para números representados en el IEEE 754. Expresar el resultado en hexadecimal.

A= C1340000 B= 3F980000