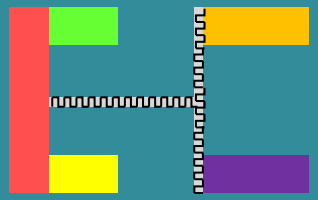
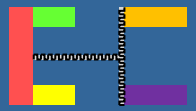


TEMA 2.
UNIDAD ARITMÉTICO-LÓGICA

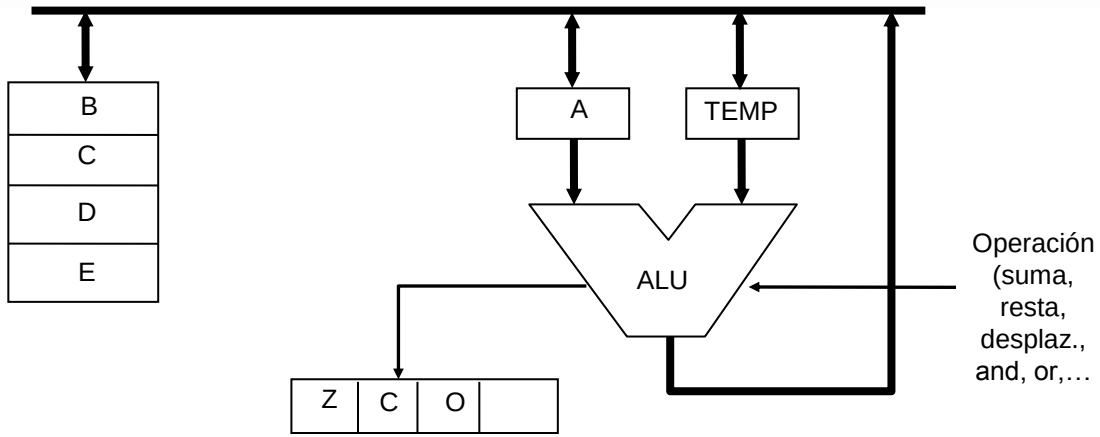


UNIDAD ARITMÉTICO-LÓGICA

Índice

- Introducción
- Unidad lógica
- Operadores de desplazamiento
- Unidad aritmética entera
  - Sumar y restar
  - Multiplicar y dividir
- Unidad aritmética flotante. IEEE 754
  - Sumar y restar
  - Multiplicar y dividir
  - Técnicas de redondeo

ESTRUCTURA GENERAL ALU



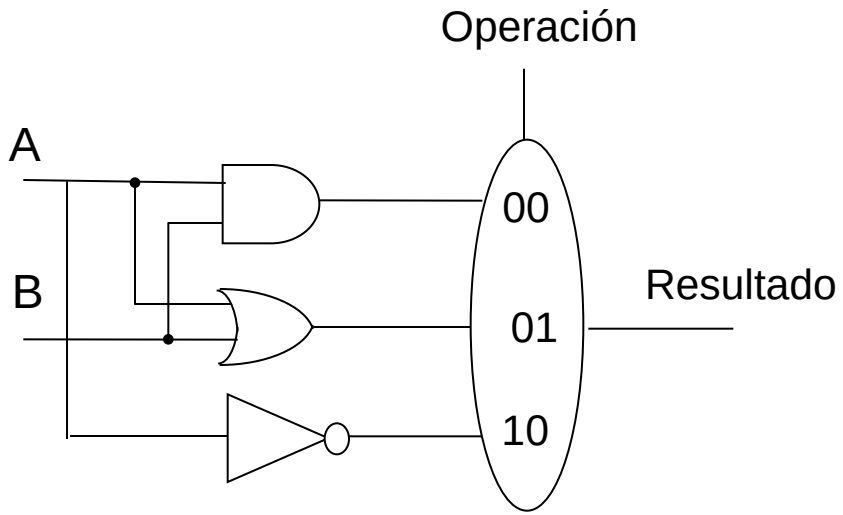
- Operador aritmético, lógico, desplazamiento (uno o varios) (ALU)
- El Acumulador
- Uno o varios registros temporales
- Indicadores de resultado
  - Acarreo (C)
  - Negativo (N)
  - Desbordamiento (O)
  - Cero (Z)

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

OPERACIONES LÓGICAS

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

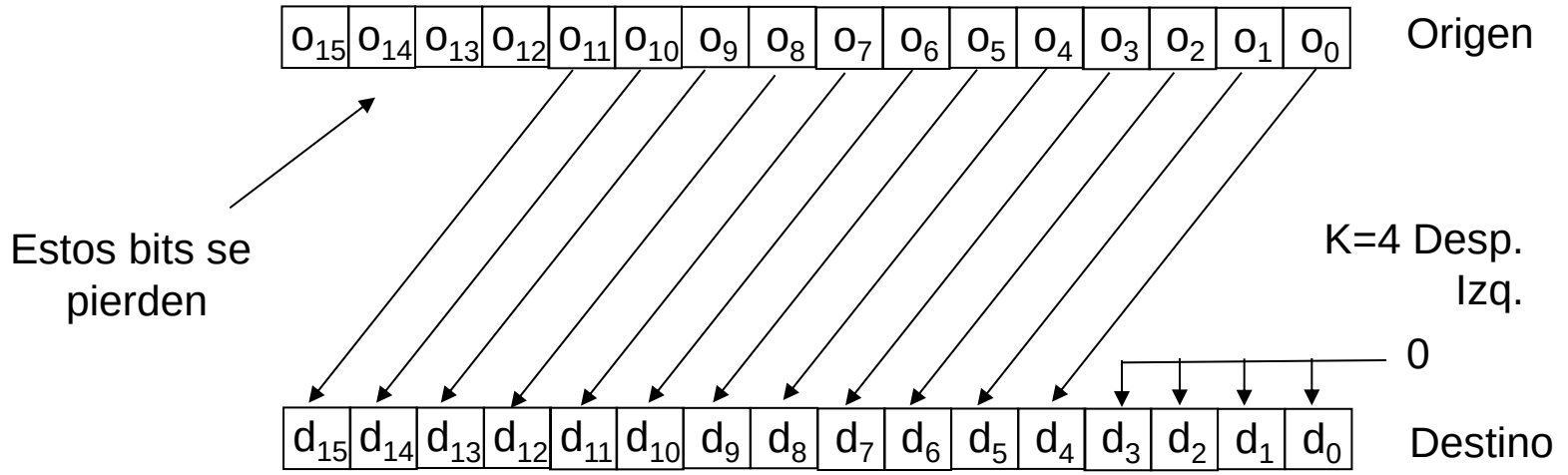


Operaciones de desplazamiento

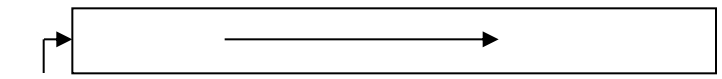
- Consisten en trasladar los bits de una palabra hacia la izquierda o derecha.
- Dependiendo de cómo se traten los extremos, se obtienen tres tipos de desplazamientos:
  - Lógicos
  - Circulares
  - Aritméticos

DESPLAZAMIENTOS LÓGICOS

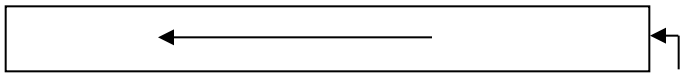
- Los valores extremos se completan con ceros, aunque se pueden plantear desplazamientos lógicos con inclusión de unos en lugar de ceros



Habitualmente, el origen y destino es la misma palabra.



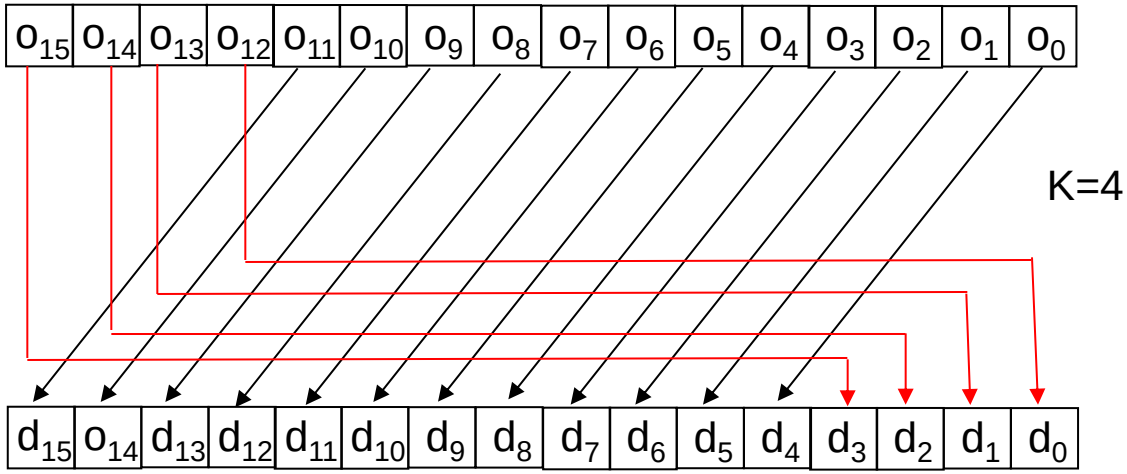
Desplazamiento lógico a la derecha

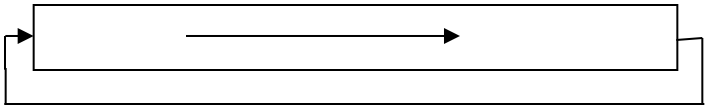


Desplazamiento lógico a la izquierda

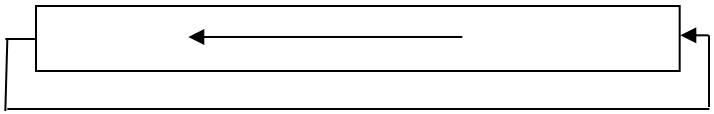
DESPLAZAMIENTOS CIRCULARES

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





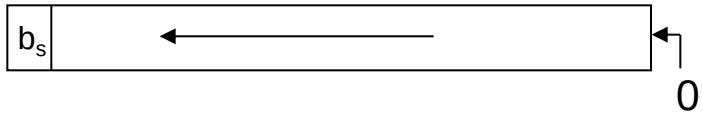
Desplazamiento circular a la derecha



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

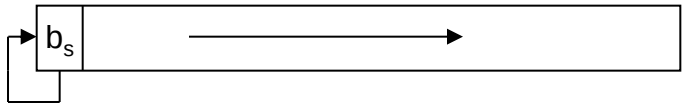
O= 0100 0000 = 64_{10} x2 → D= 1000 0000 = -128_{10} → incorrecto

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

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

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

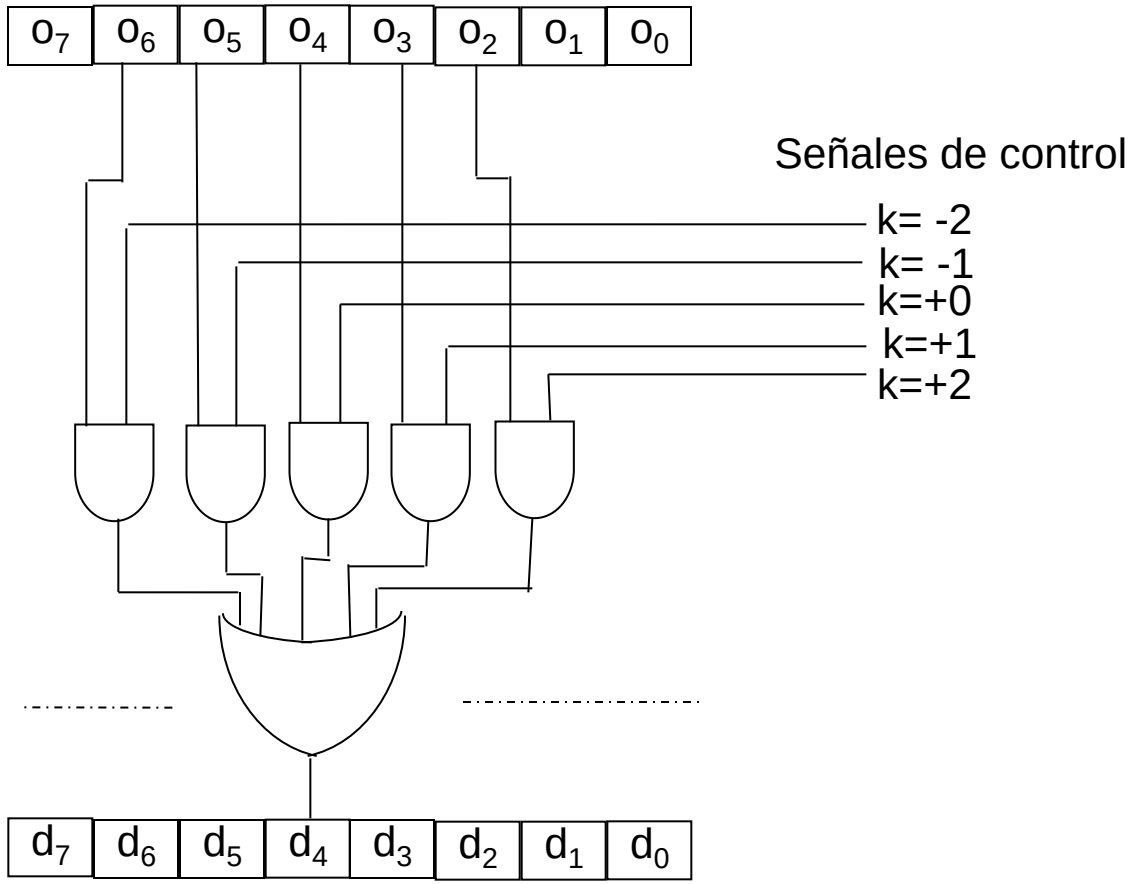
O= 0000 1010 = 10_{10} /2 → D= 0000 0101 = 5_{10}

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

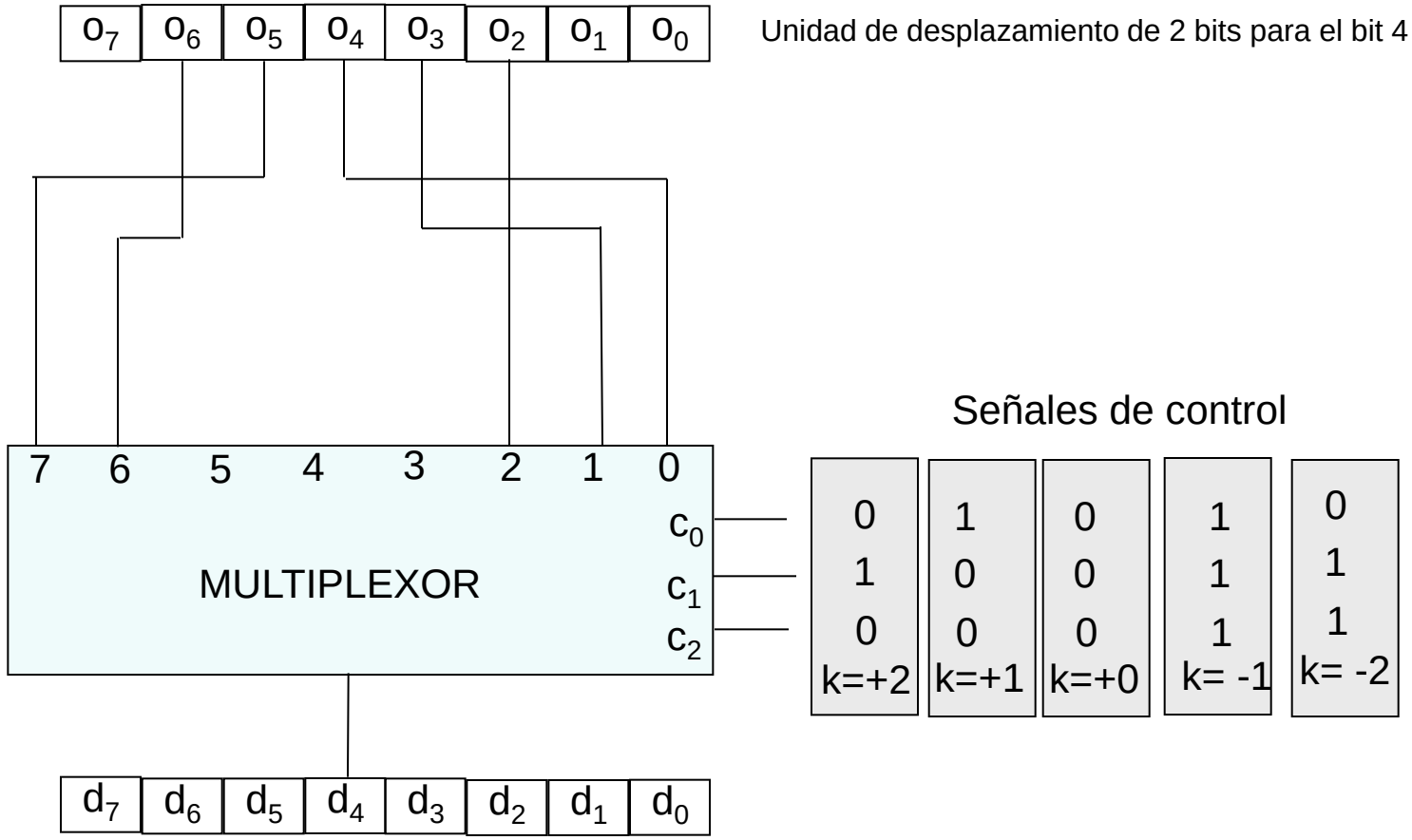
Puertas lógicas

- 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



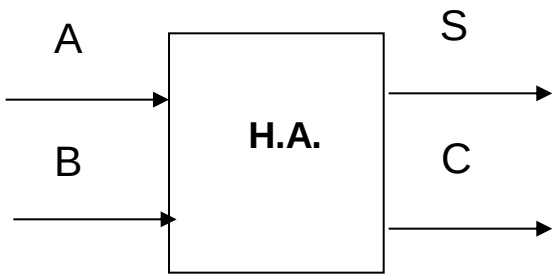
Unidad de desplazamiento de 2 bits para el bit 4

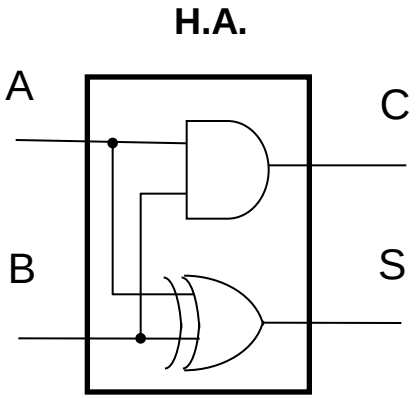
- Semisumador Binario (H.A.)

Entradas		Salidas	
X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{X} \cdot Y + X \cdot \overline{Y} = X \oplus Y$$

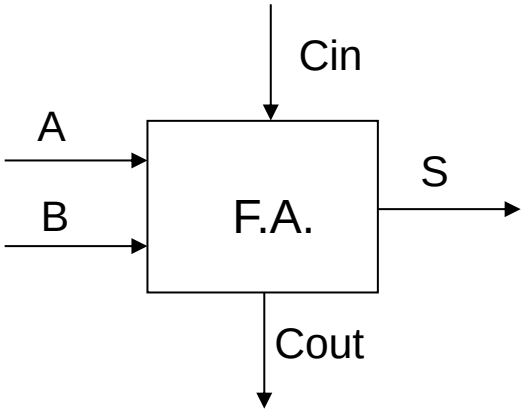
$$C = X \cdot Y$$





SUMADOR COMPLETO (F.A.)

- Semisumador Binario (H.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

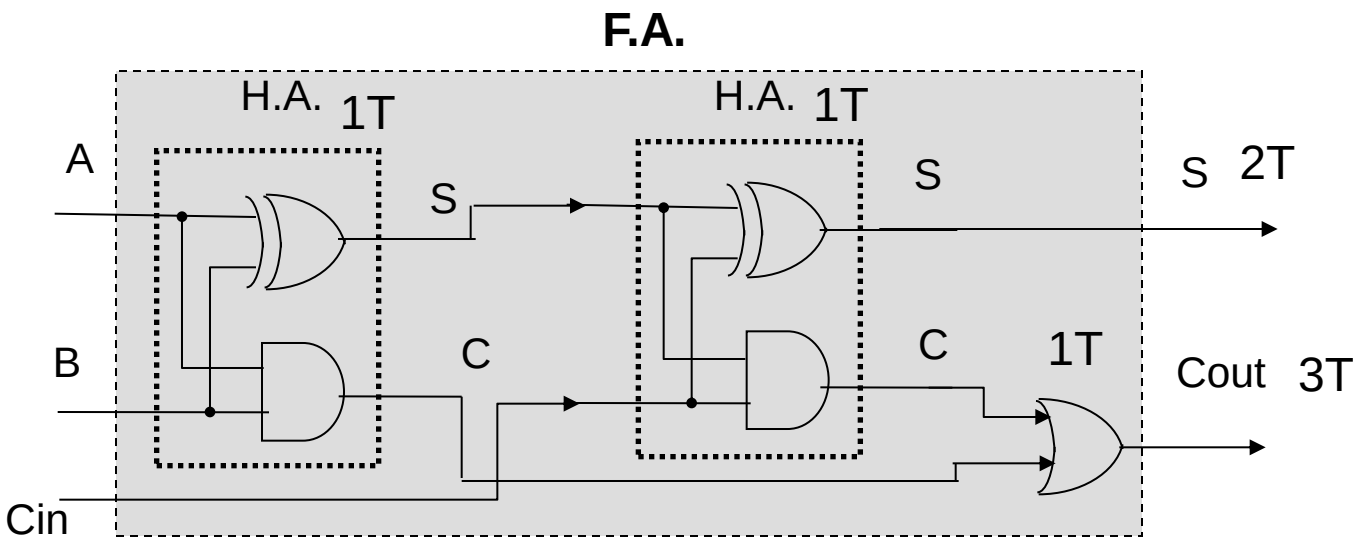
$$S = \overline{A}\cdot\overline{B}\cdot Cin + \overline{A}\cdot B\cdot\overline{Cin} + A\cdot\overline{B}\cdot\overline{Cin} + A\cdot B\cdot Cin$$

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

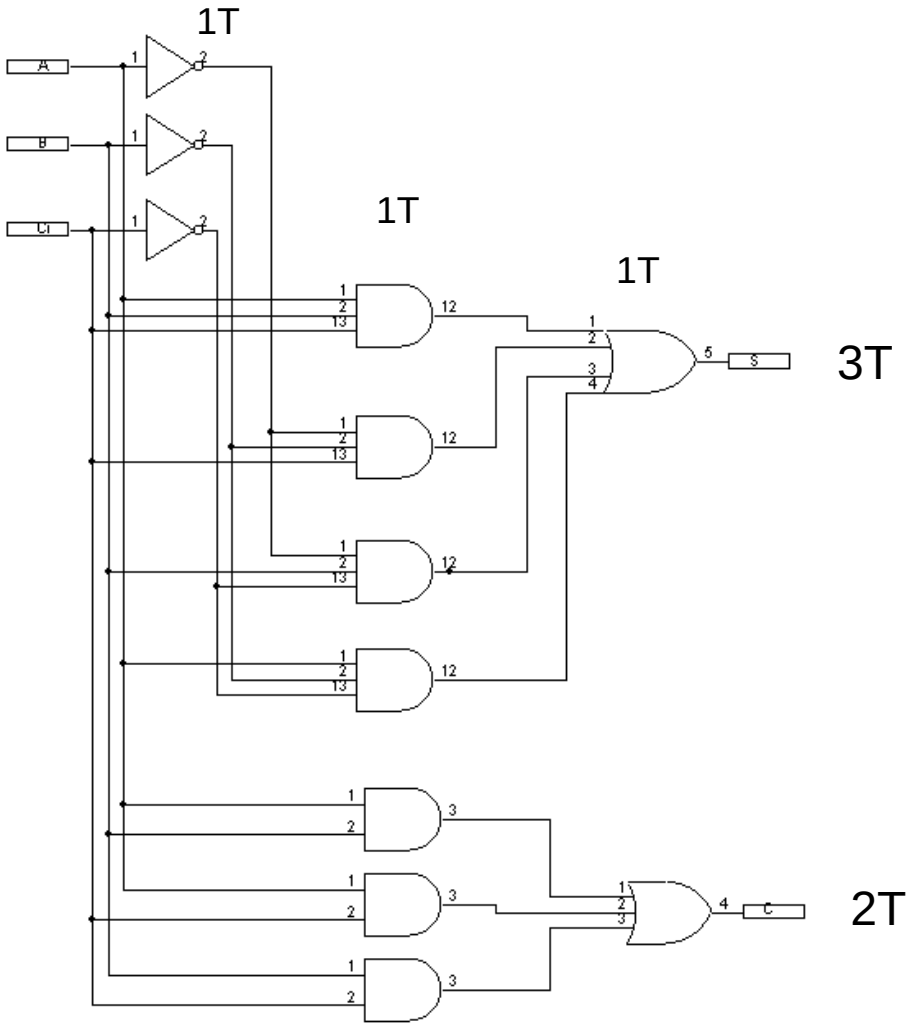
$$S = (A\oplus B)\oplus Cin$$

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

- Sumador completo (FA) - Semisumadores

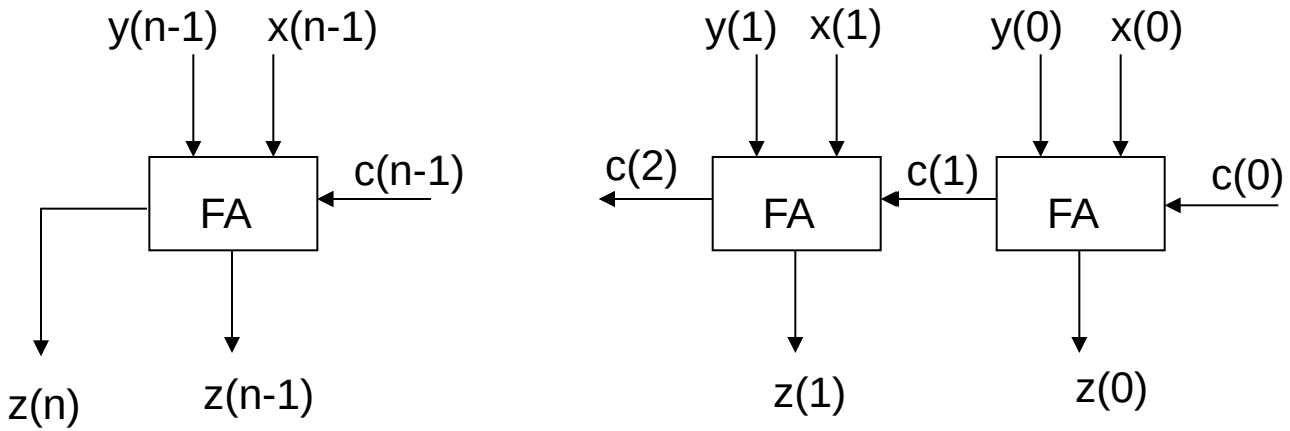


- Sumador completo (FA) – Puertas lógicas



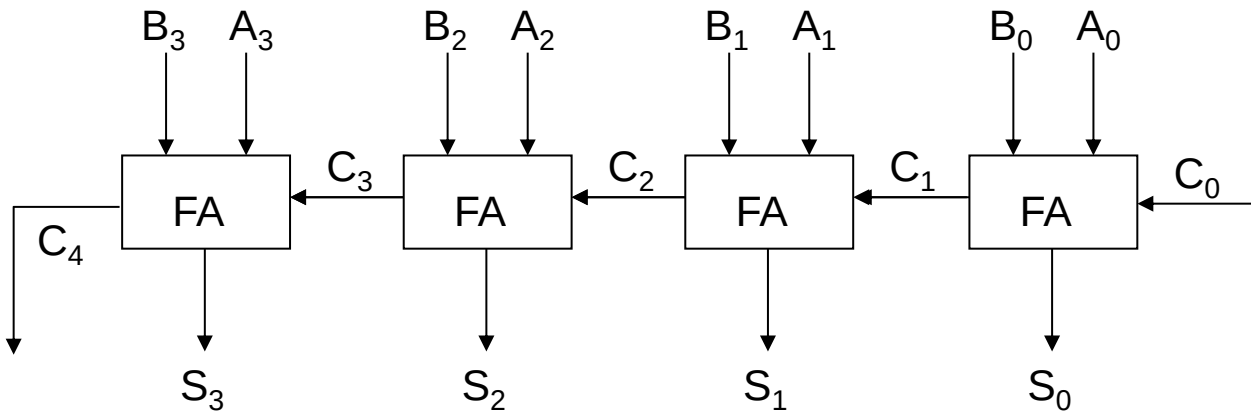
SUMADOR CON PROPAGACIÓN DE ACARREO (CPA)

- 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)



Sumador con propagación de acarreo (CPA)

- Sumador con propagación de acarreo de 4 bits.

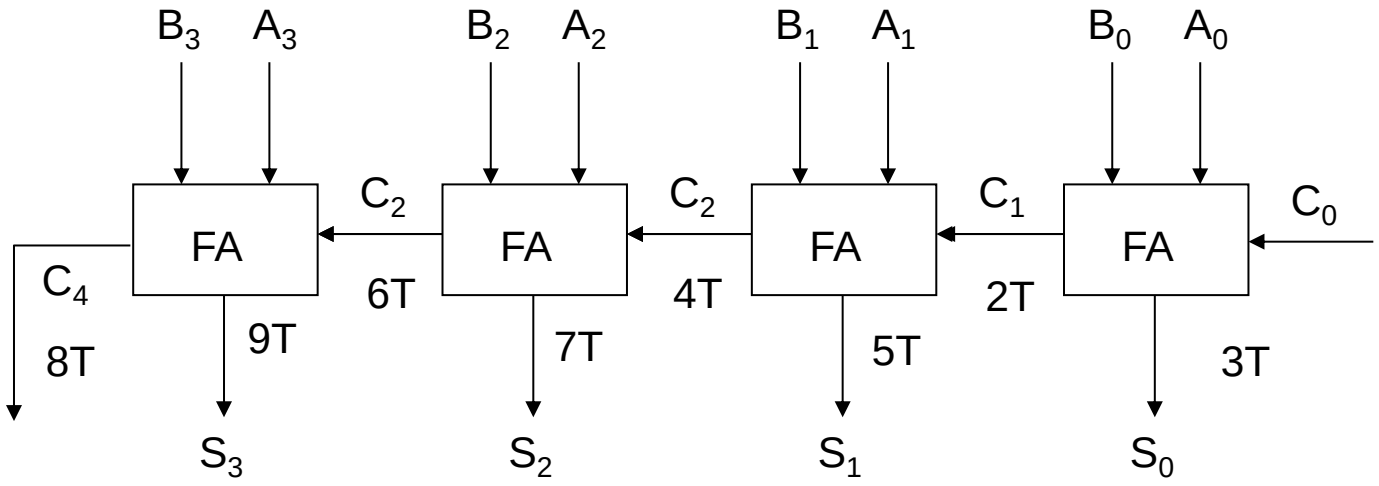


SUMADOR CON PROPAGACIÓN DE ACARREO

- Sumadores construidos con puertas lógicas a partir de la expresión:

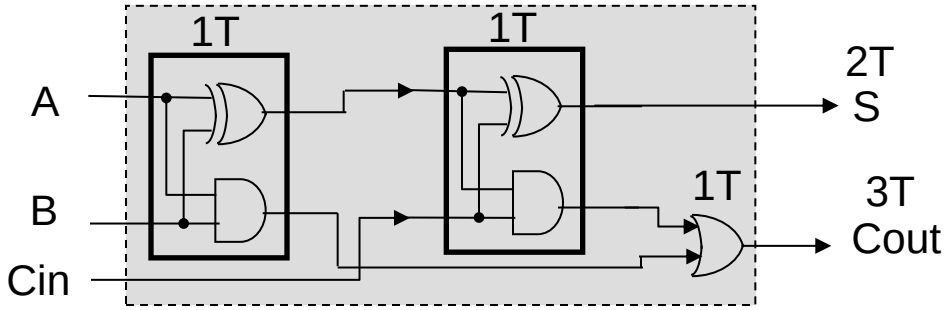
$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin$$

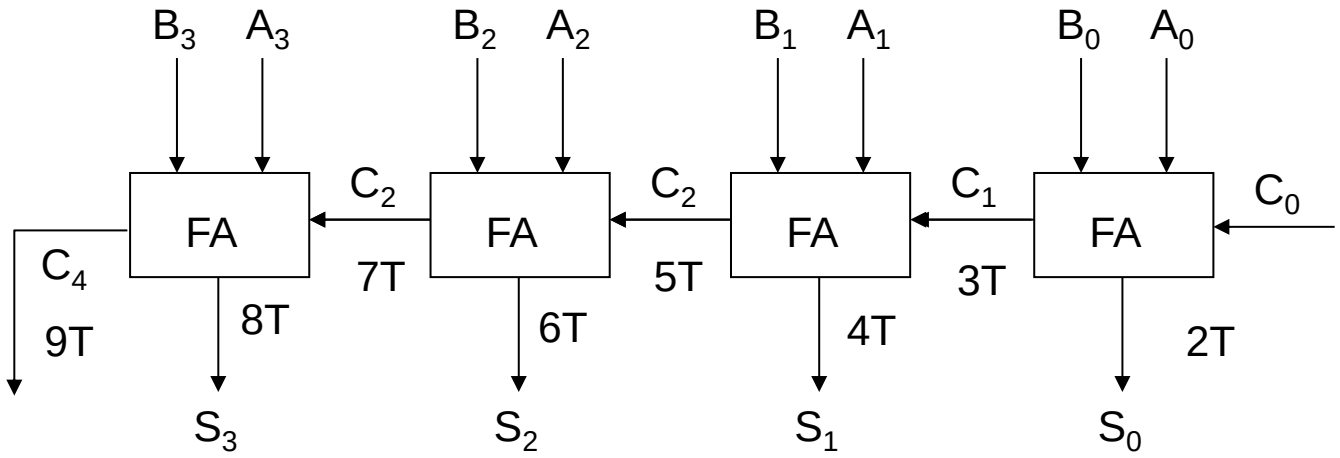
$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin$$



SUMADOR CON PROPAGACIÓN DE ACARREO

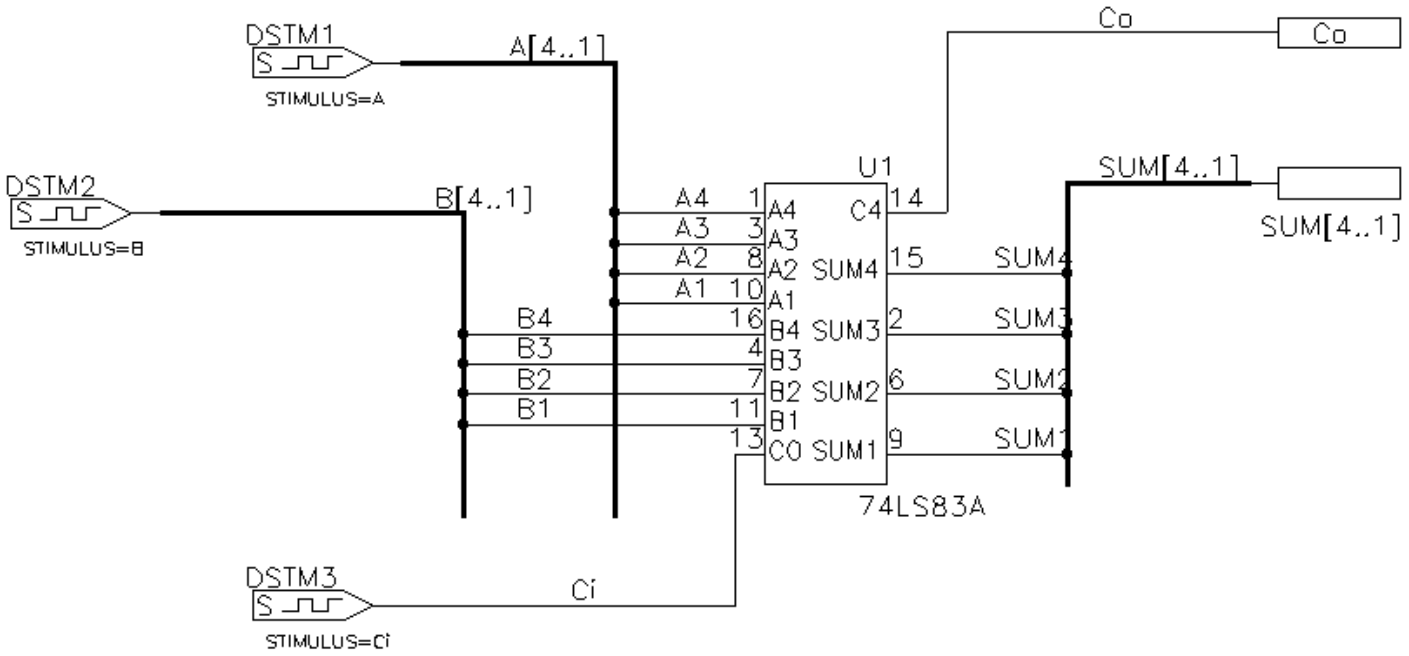
- Sumadores completos construidos con semisumadores:





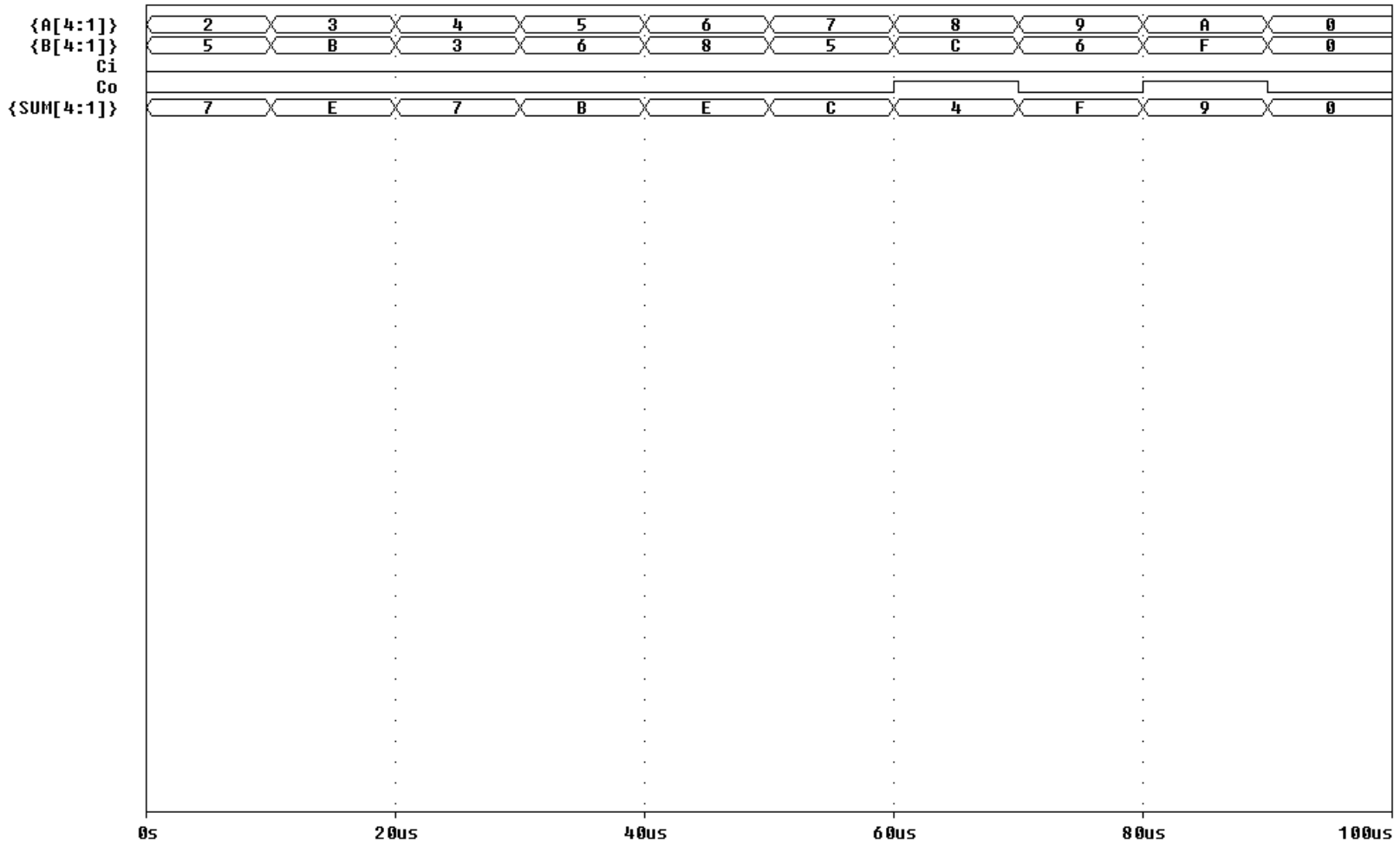
$$Tiempo\_Total = (2 \cdot n + 1)T$$

- Sumadores integrado



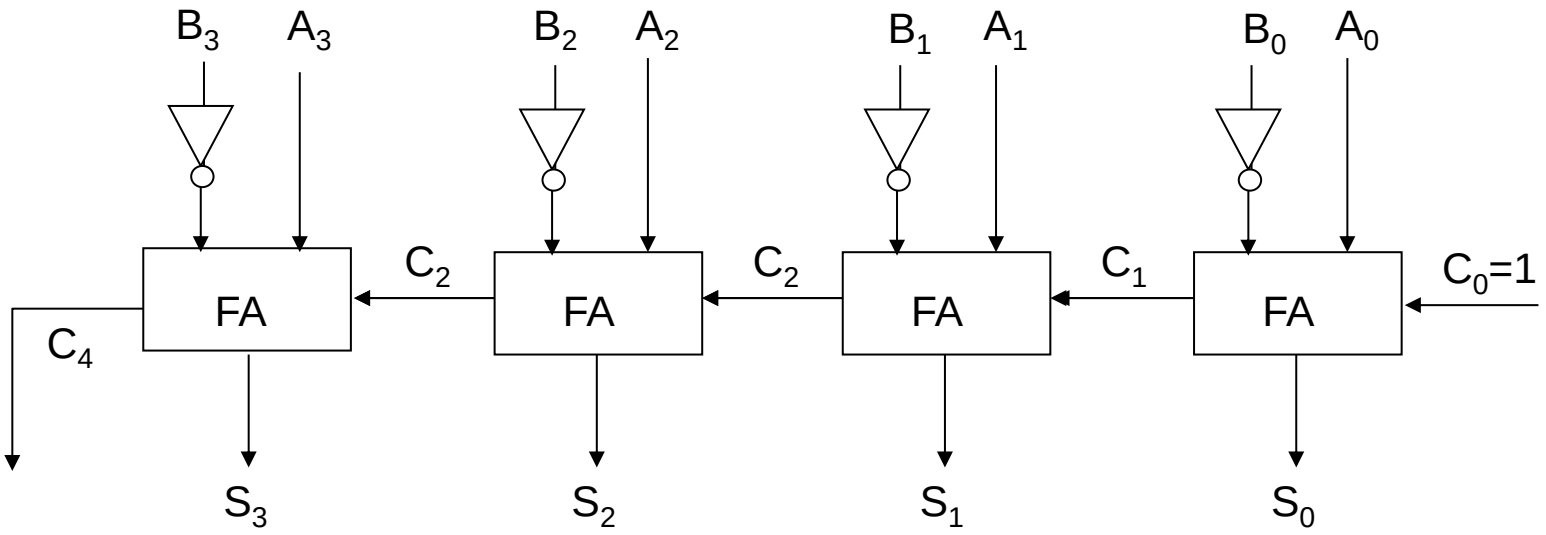
SUMADOR 74LS83

Unidad aritmética entera. Sumar y restar



CIRCUITO RESTADOR

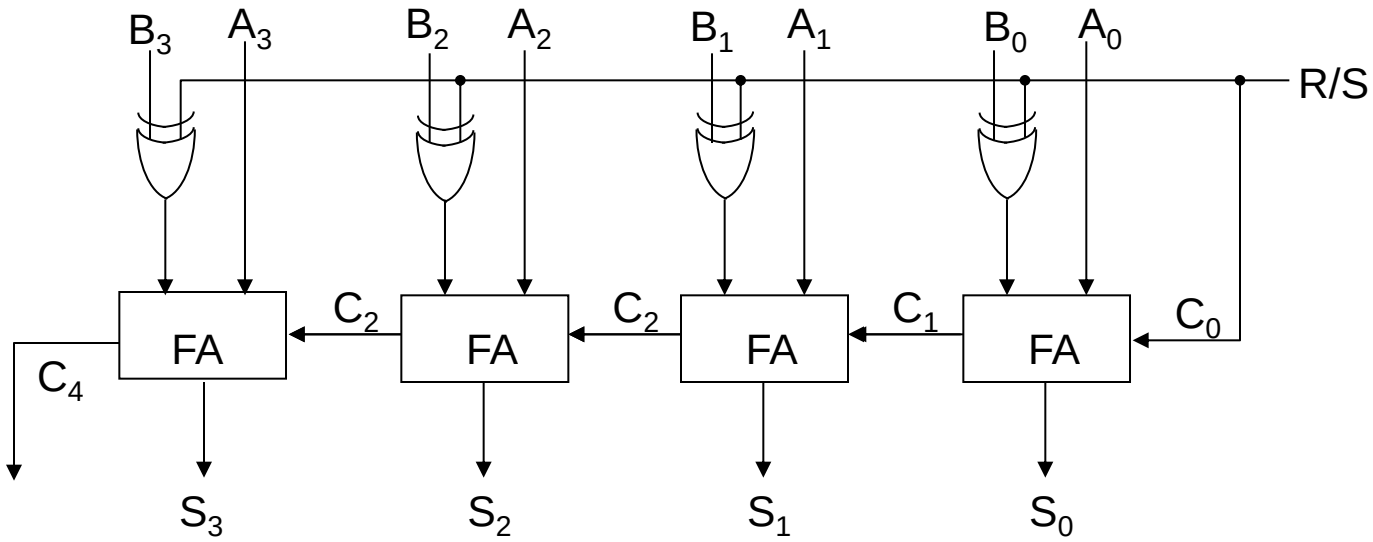
- Suponer que se trabaja con números expresados en complemento a 2.
- A - B = A + (C1(B) + 1)



CIRCUITO SUMADOR-RESTADOR

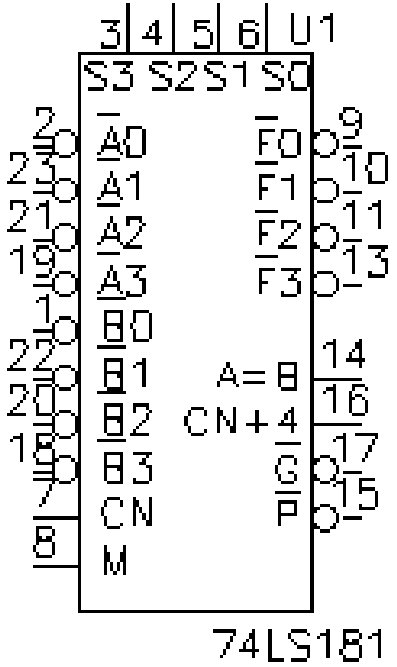
R/S	B_i	Entrada al FA
0	0	0
0	1	1
1	0	1
1	1	0

$$Tiempo\_Total = 2(n+1)T$$

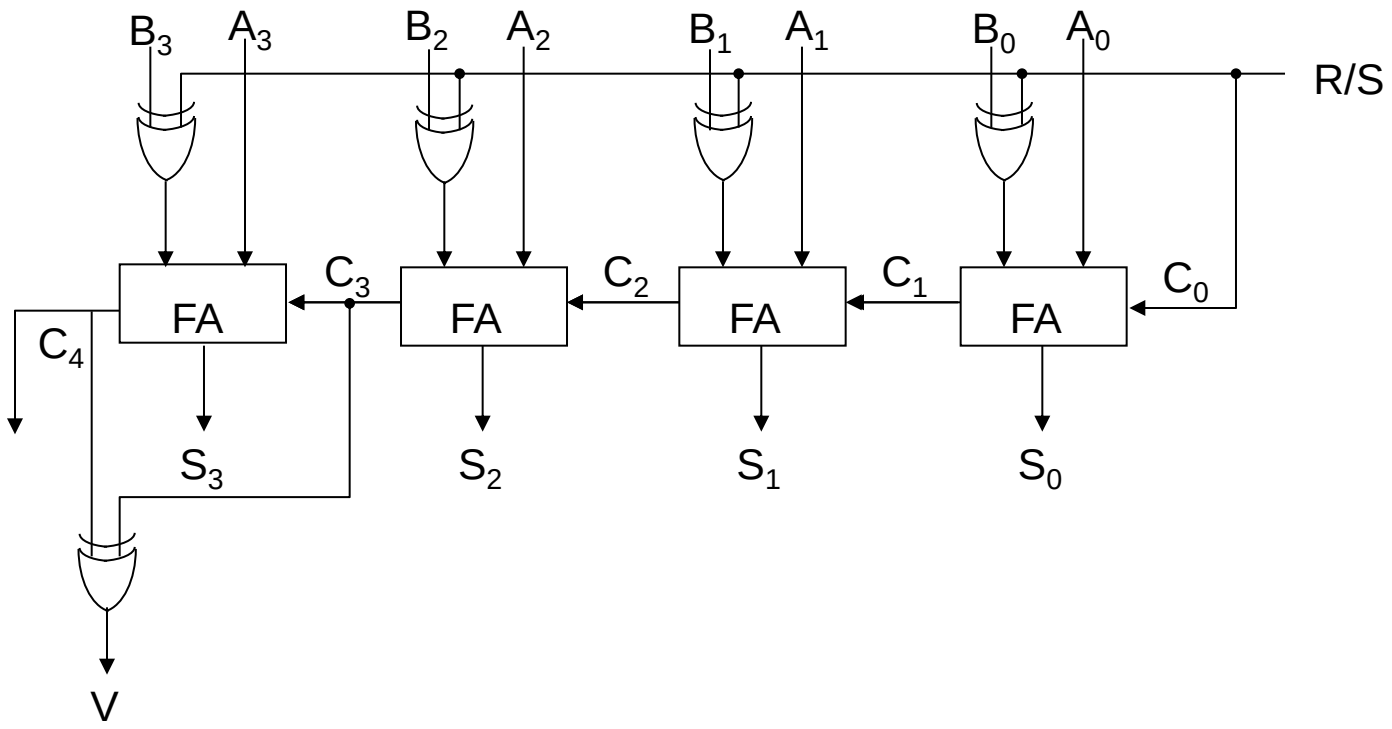


CIRCUITO SUMADOR-RESTADOR 74LS181

Unidad aritmética entera. Sumar y restar



Sumador-Restador en complemento a 2 con detección de desbordamiento.



DETECCIÓN DE DESBORDAMIENTO

1. Caso suma de dos positivos

			S		
C4	C3	C2	C1		
0	1	1	1		
	0	1	1	1	
	0	1	1	1	
	1	1	1	0	OV

2. Caso suma de dos negativos

			S		
C4	C3	C2	C1		
1	0	1	1		
	1	0	0	1	
	1	0	1	1	
	0	1	0	0	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:

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

SUMADOR CON ANTICIPACIÓN DE ACARREO

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

$$C_0 = G_0 + P_0 \cdot C_{-1}$$

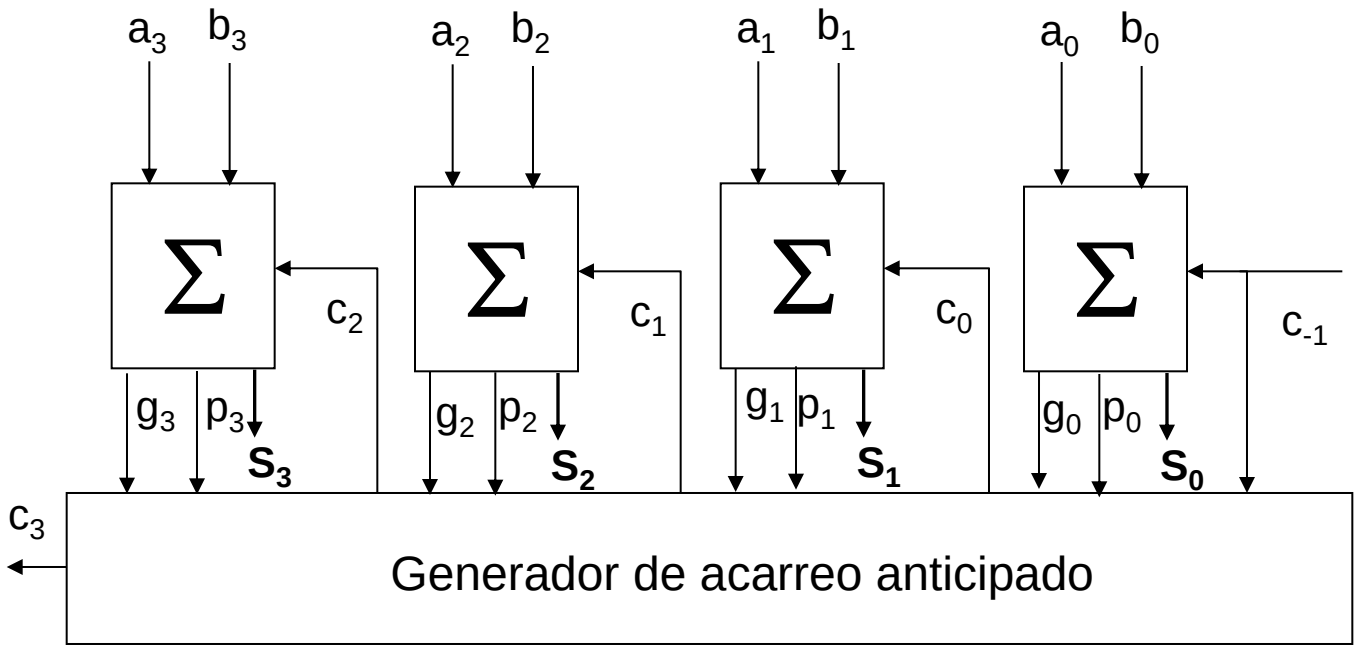
$$C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_{-1}$$

$$C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

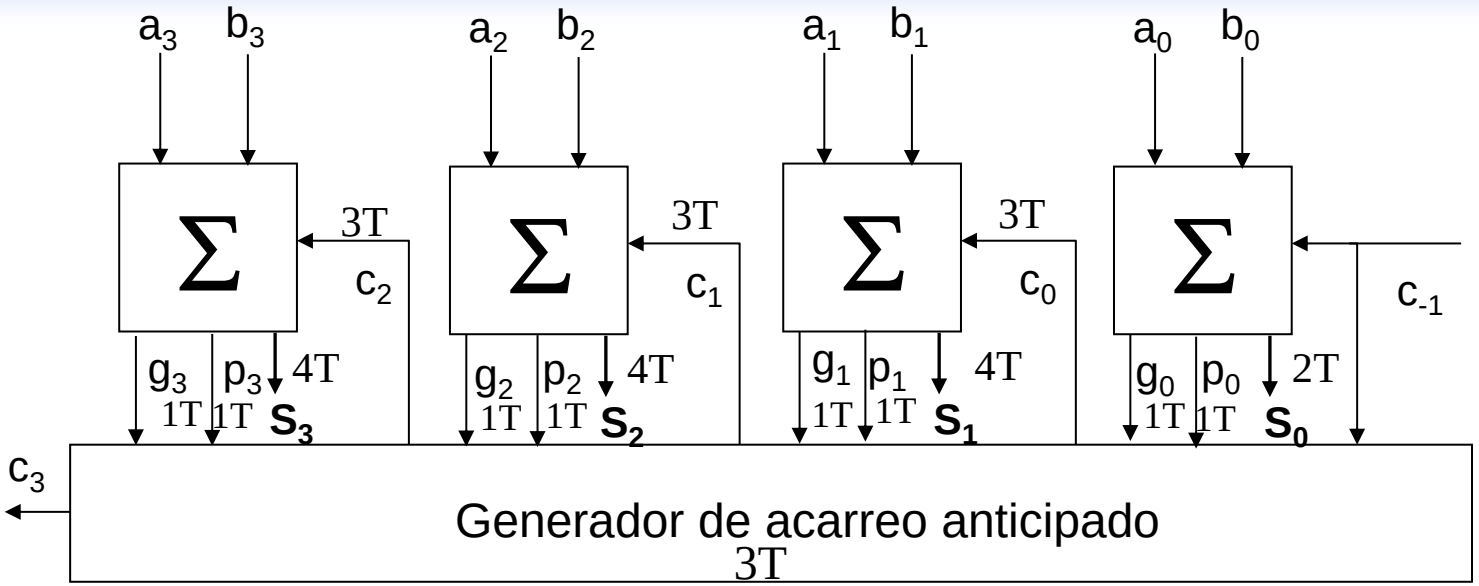
$$C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

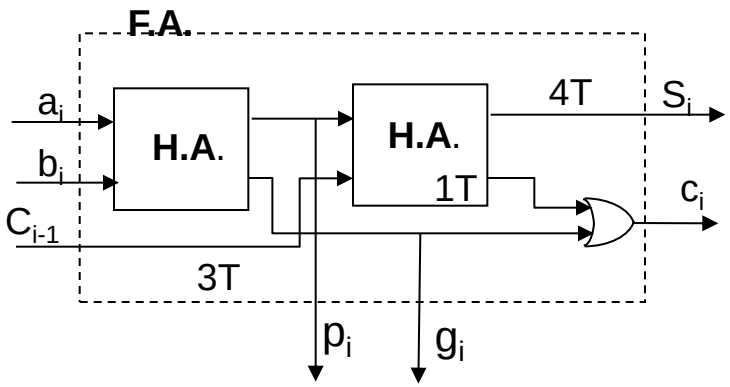
- Todos los acarreos dependen de a_i y b_i.
- Estas expresiones se resuelven como suma de productos.
- Tres niveles de puertas lógicas para obtener cada uno de los acarreos.

SUMADOR CON ANTICIPACIÓN DE ACARREO



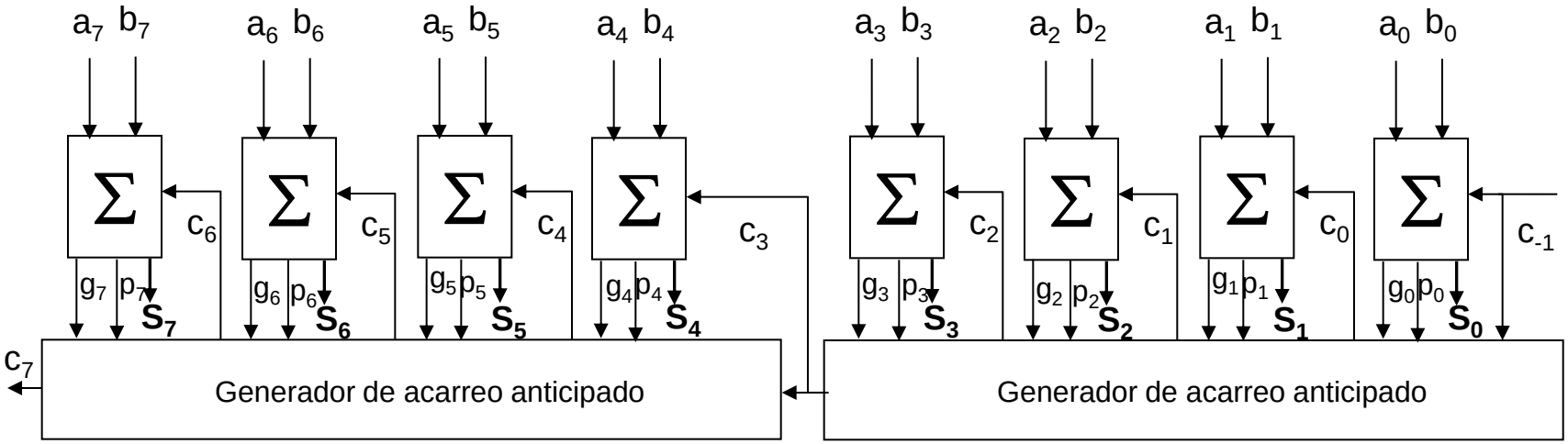
SUMADOR CON ANTICIPACIÓN DE ACARREO





Sumadores construidos con semisumadores

EJEMPLO (SUMADOR CLA DE 8 BITS)

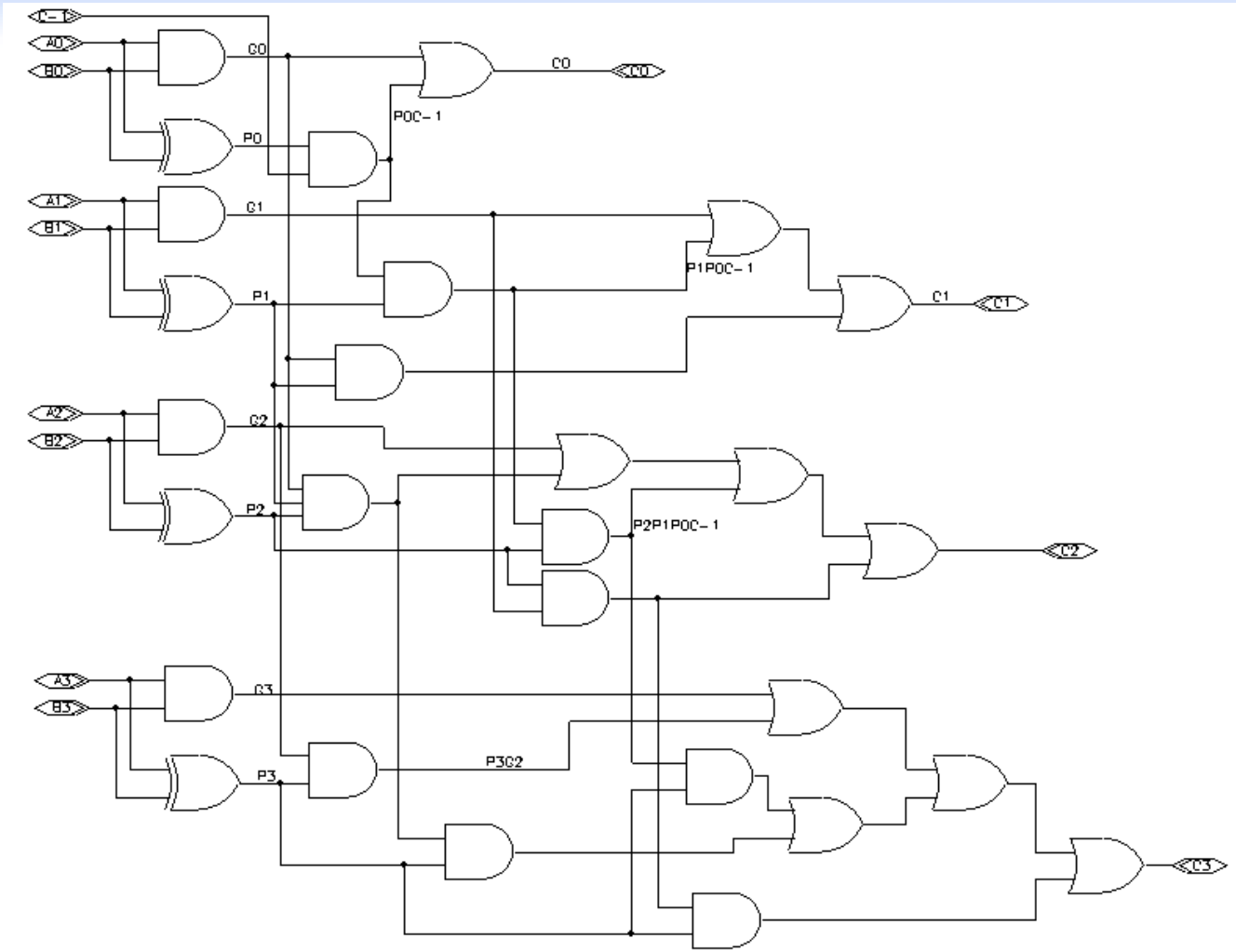


Calcular los retardos en este CLA suponiendo que los sumadores se construyen con semisumadores.

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

EJEMPLO (CLA DE 4 BITS)

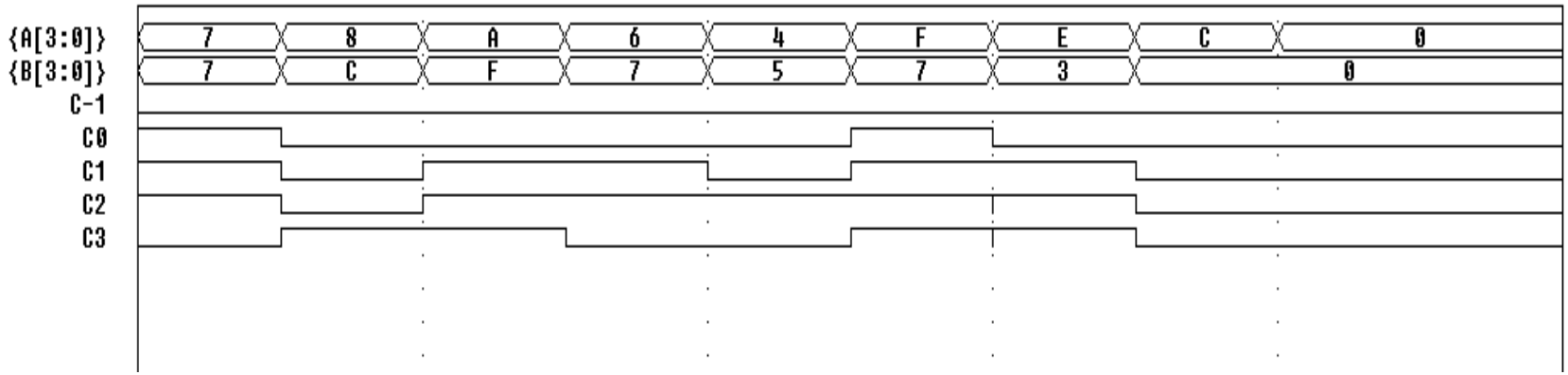
Unidad aritmética entera. Sumar y restar



EJEMPLO (CLA DE 4 BITS)

C3	C2	C1	C0	
0	1	1	1	
	0	1	1	1
	0	1	1	1
	1	1	1	0

C3	C2	C1	C0	
1	1	1		
	1	0	1	0
	1	1	1	1
	1	0	0	1



La multiplicación

- Algoritmo de sumas y desplazamientos
- Si multiplicando de n bits y 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

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.

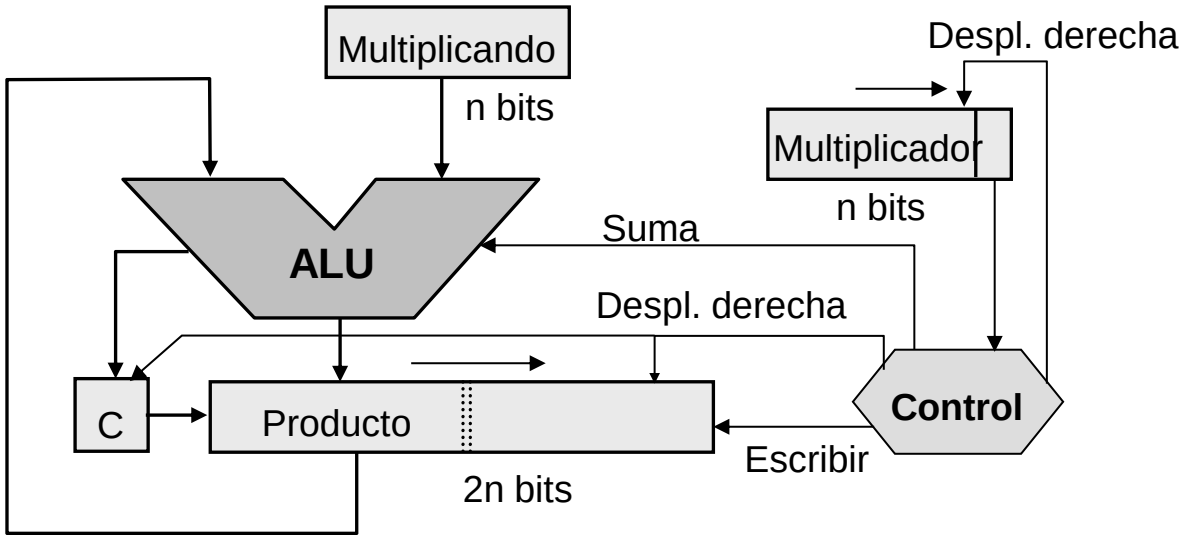
Fin entonces

Desplazar 1 bit a la derecha el registro producto

Desplazar 1 bit a la derecha el registro multiplicador

Fin repetir

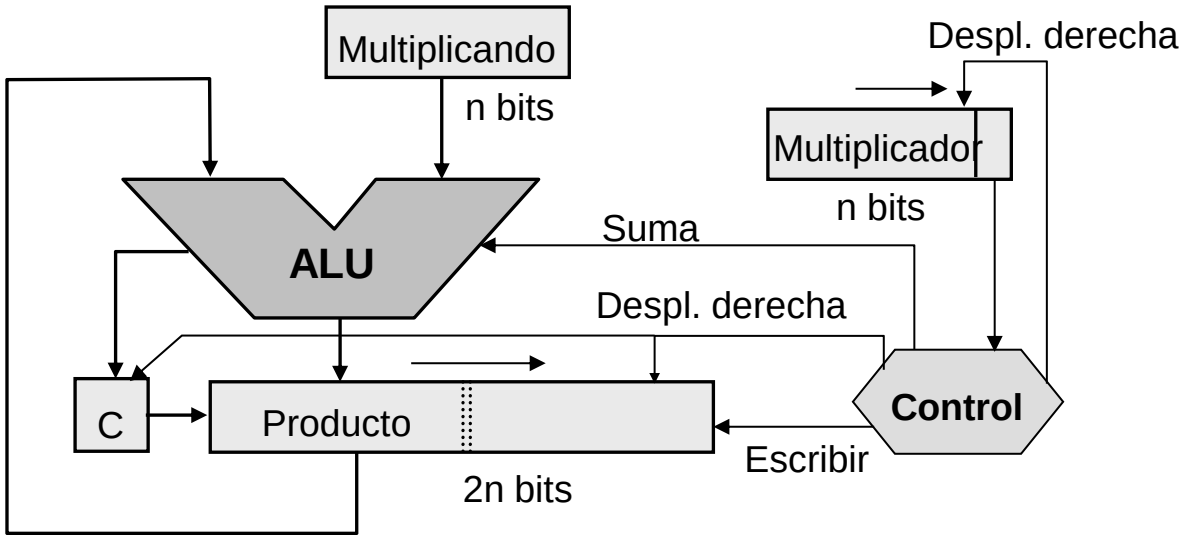
Versión preliminar



Multiplicación binaria sin signo

Multiplicando				1	0	1	1	
Multiplicador				1	1	0	1	
				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

Versión preliminar



Repetir n veces

Si el bit 0 del registro producto=1 entonces

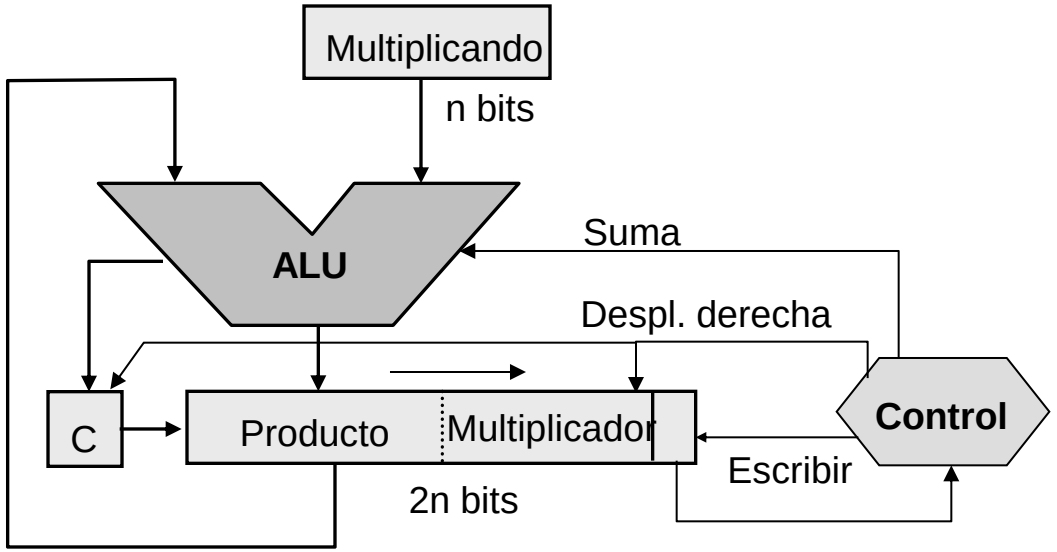
Sumar el multiplicando a la mitad izquierda del producto y colocar el resultado en la mitad izquierda del producto.

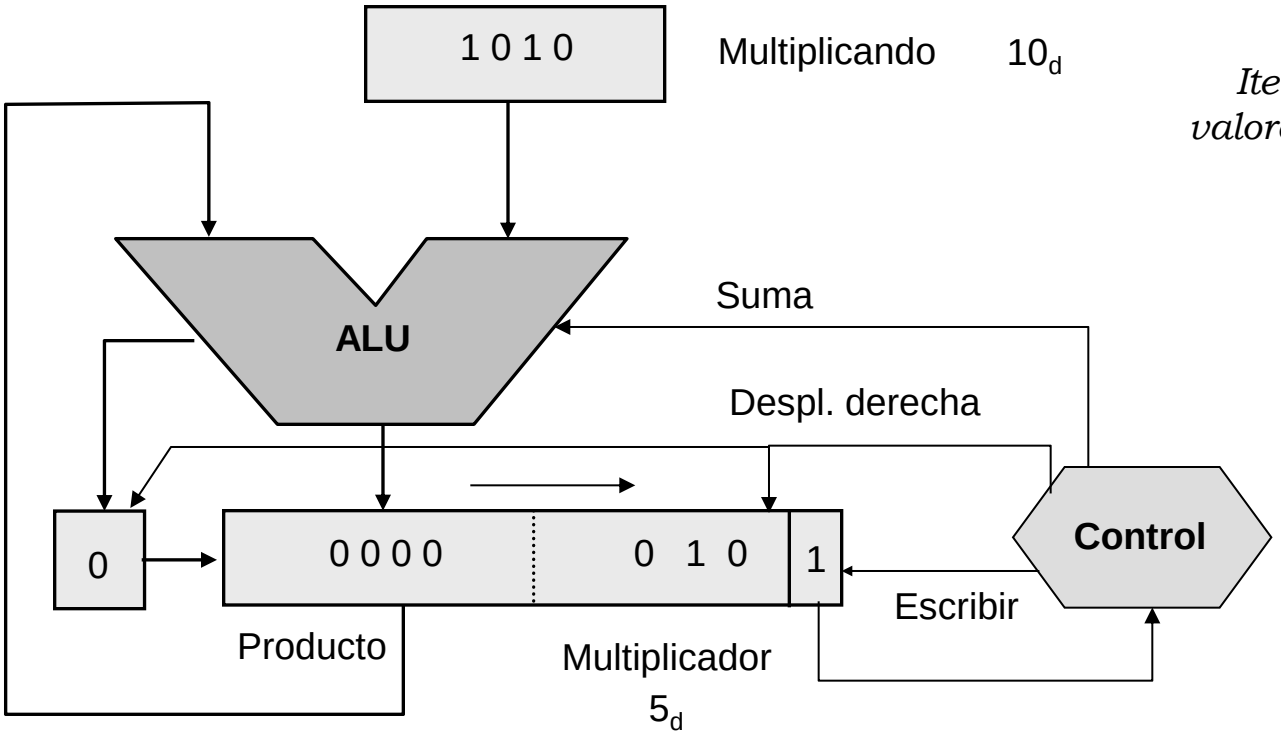
Fin entonces

Desplazar 1 bit a la derecha el registro producto

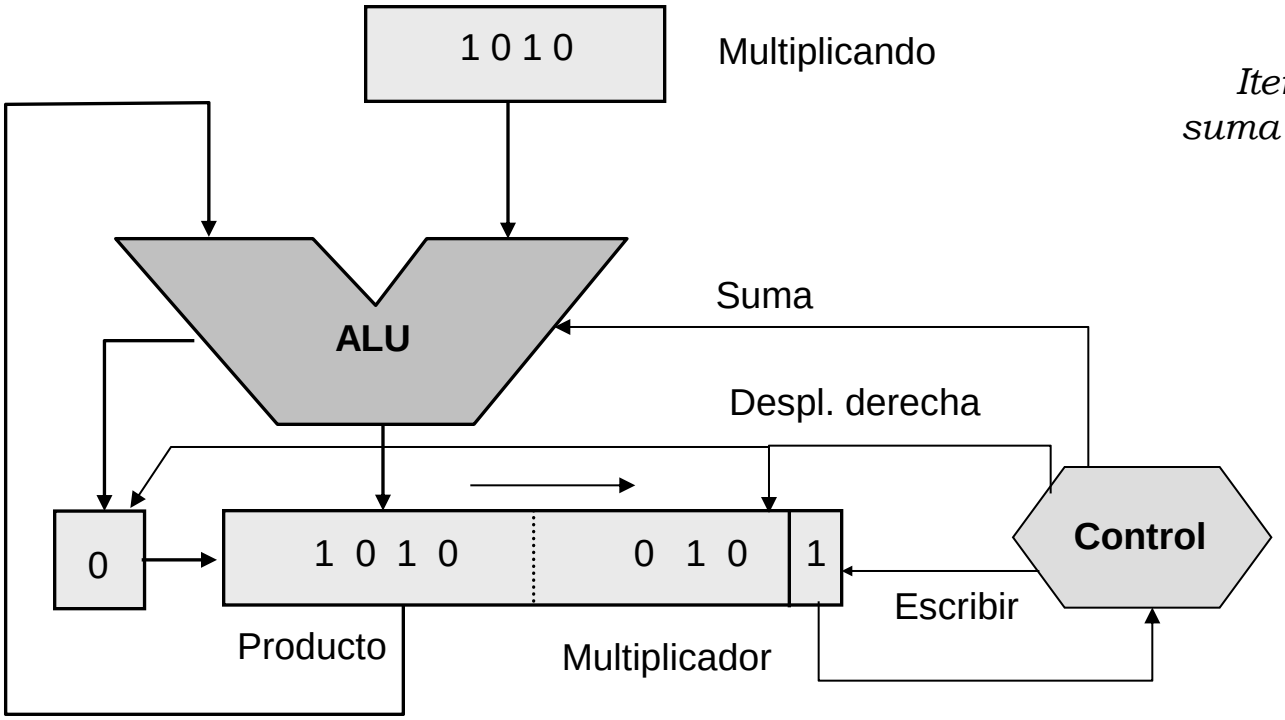
Fin repetir

Versión final



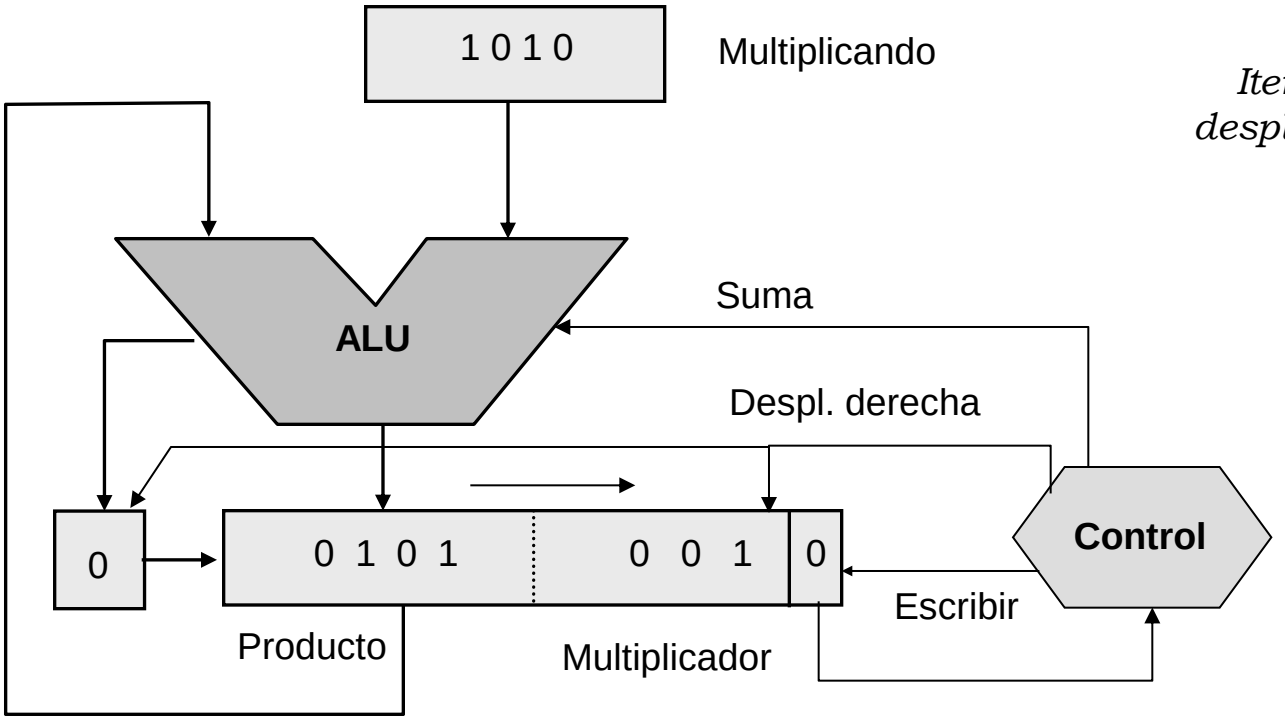

1 0 1 0
Multiplicando
10_d
Iteración 0
valores iniciales
ALU
Suma
Despl. derecha
0
0 0 0 0
0 1 0
1
Control
Escribir
Producto
Multiplicador
5_d

MULTIPLICACIÓN BINARIA SIN SIGNO



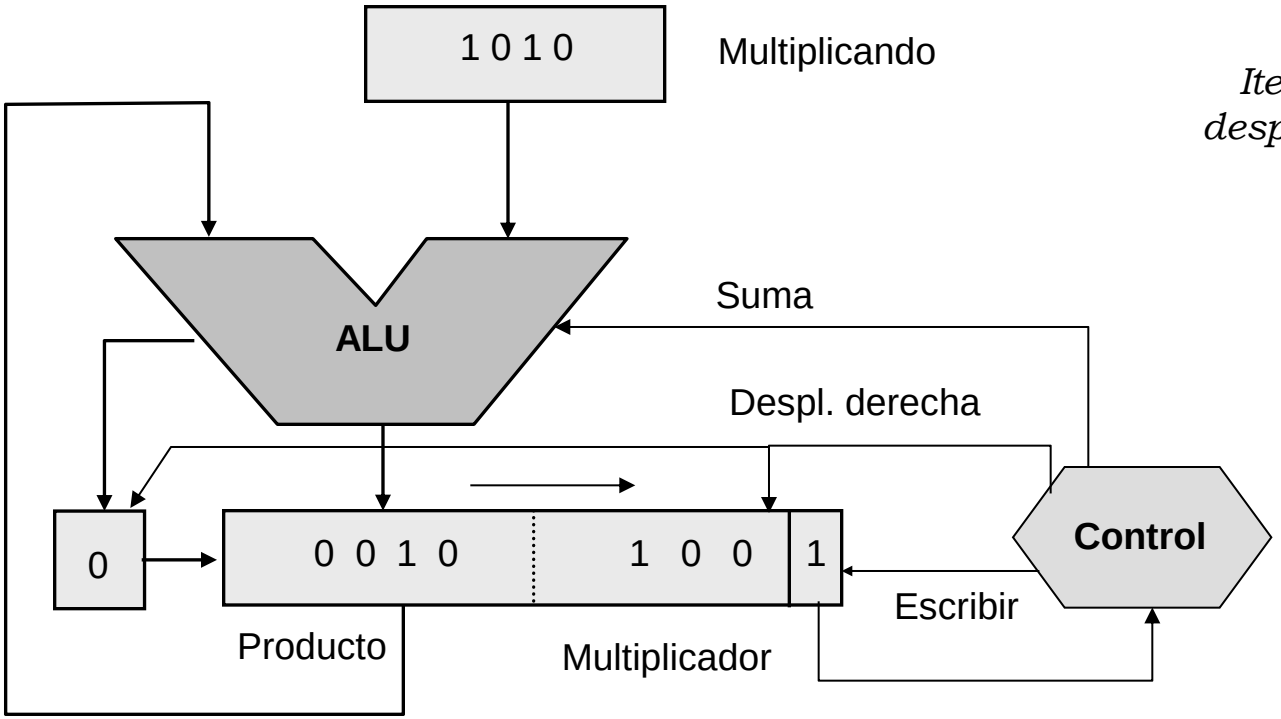
Iteración 1
suma P y Mando

MULTIPLICACIÓN BINARIA SIN SIGNO



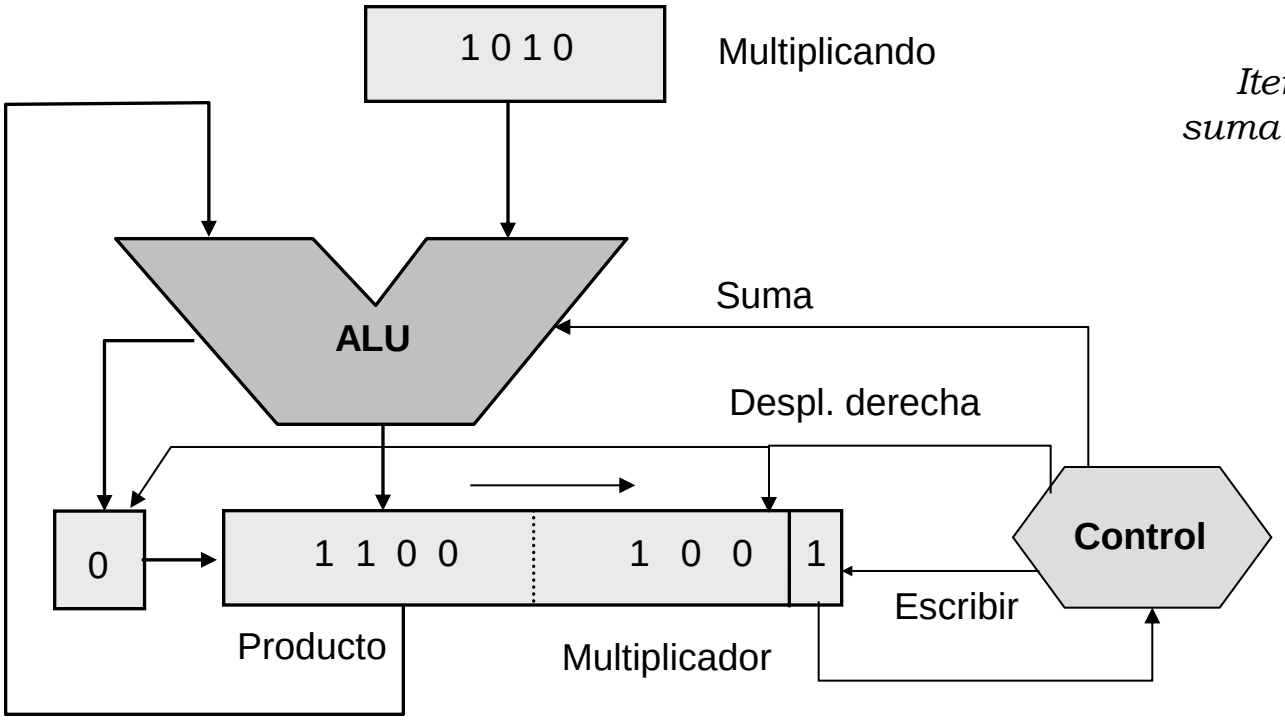
Iteración 1
despla. P dcha.

MULTIPLICACIÓN BINARIA SIN SIGNO



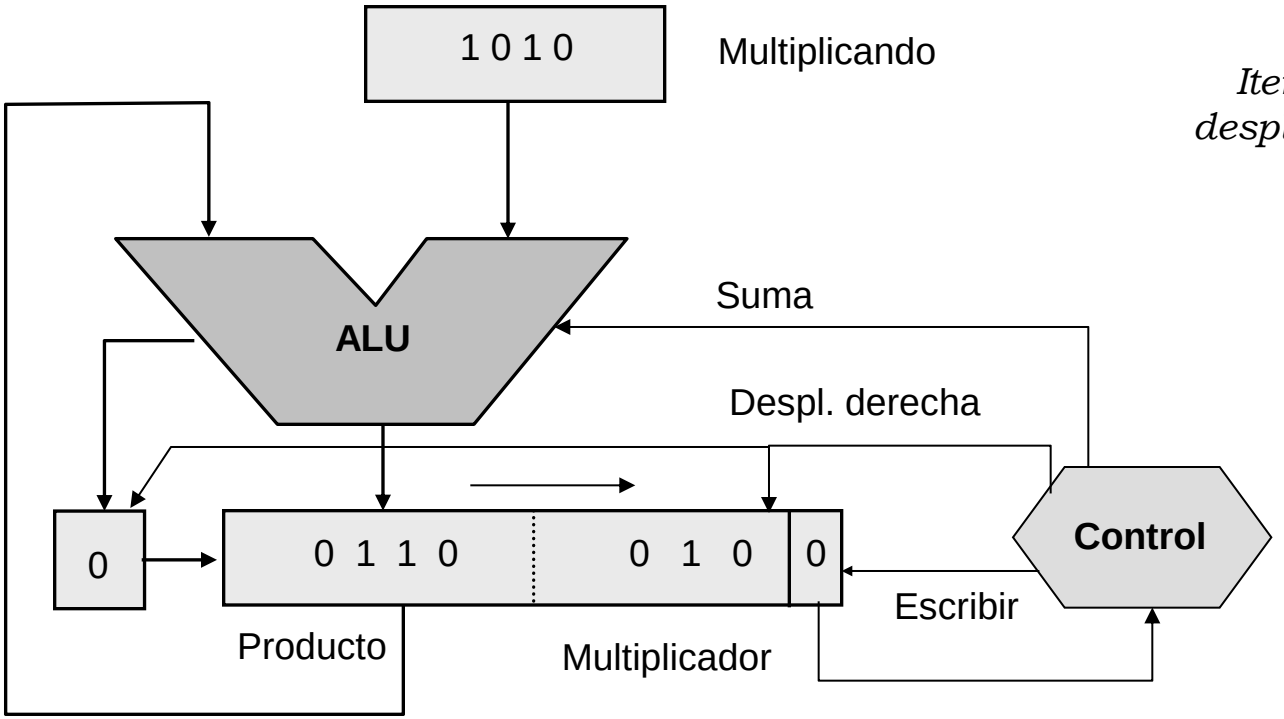
Iteración 2
despla. P dcha

MULTIPLICACIÓN BINARIA SIN SIGNO

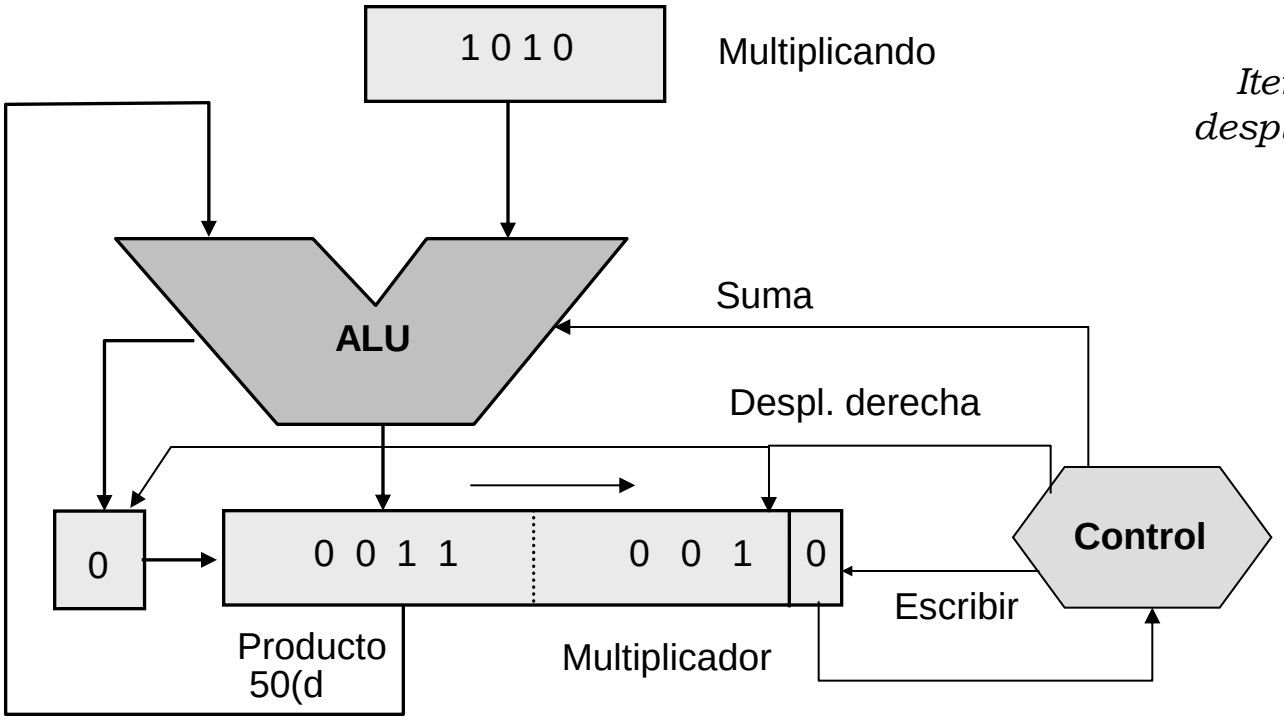


Iteración 3
suma P y Mando

MULTIPLICACIÓN BINARIA SIN SIGNO



Iteración 3
despla. P dcha.



Iteración 4
despla. P dcha.

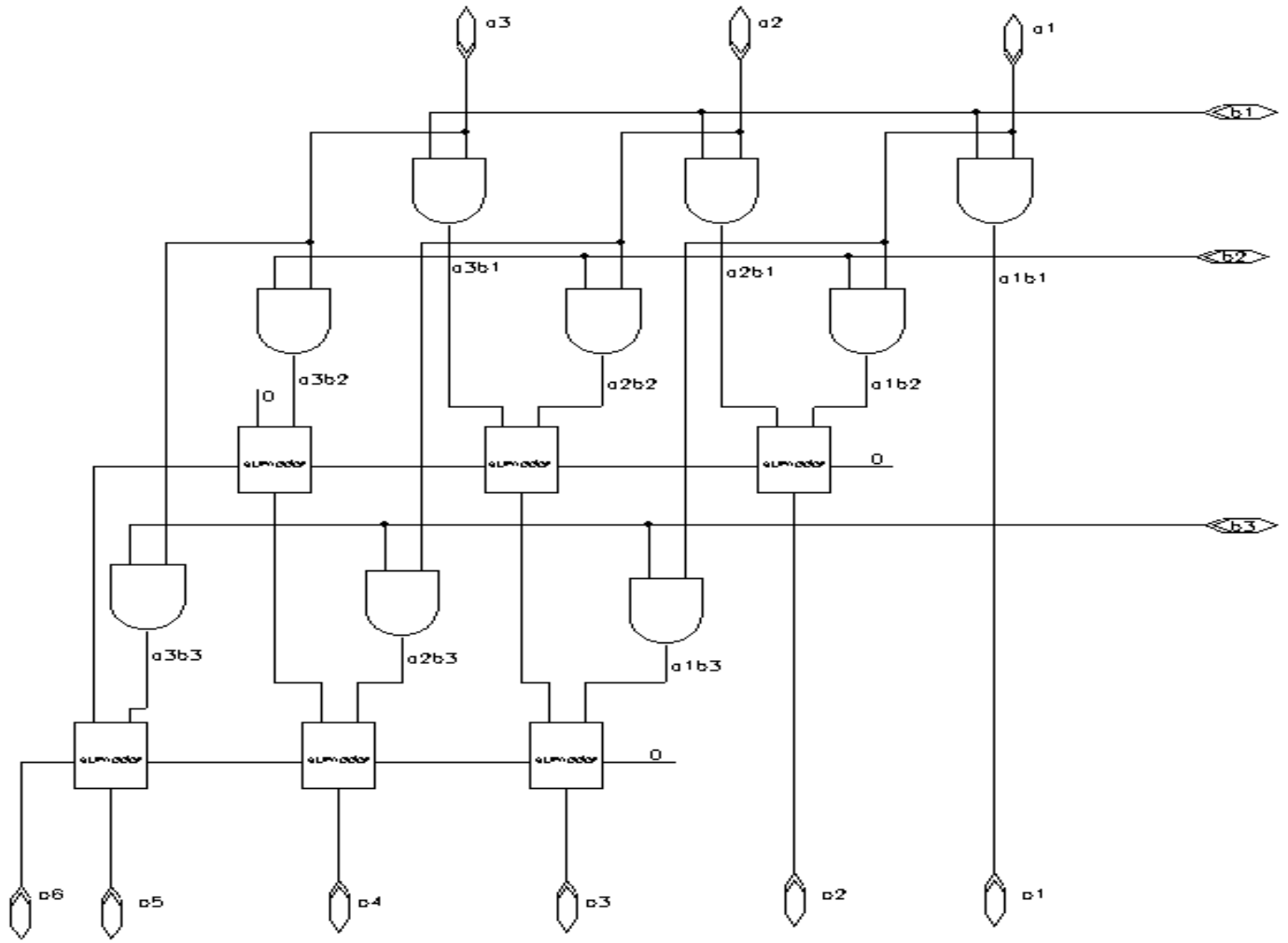
MULTIPLICACIÓN BINARIA SIN SIGNO

Multiplicando = 1010
Multiplicador = 0101

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

MULTIPLICACIÓN RÁPIDA

Unidad aritmética entera. Multiplicar y dividir



MULTIPLICACIÓN BINARIA CON SIGNO

- Supongamos números expresados en Ca2
- A = 1010 y B = 0011
- 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

ALGORITMO DE BOOTH

- Supongamos Multiplicando = 2 y Multiplicador = 7 (en binario 0010 x 0111)
- Booth expresó 7 = 8 - 1 y sustituyo el multiplicador por esta descomposición:

0111 = 1000 - 0001 = +100-1

				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	

Bit actual	Bit a la izquierda	Sustitución
0	0	0 (no hay transición)
0	1	-1 (transición hacia negativo)
1	0	+1 (transición hacia positivo)
1	1	0 (no hay transición)

Ejemplo: Multiplicando = 11101110 y Multiplicador = 01111010

Recodificación del multiplicador según Booth = +1000-1+1-10

								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

Inicialmente $q_{-1}=0$

Repetir n veces

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

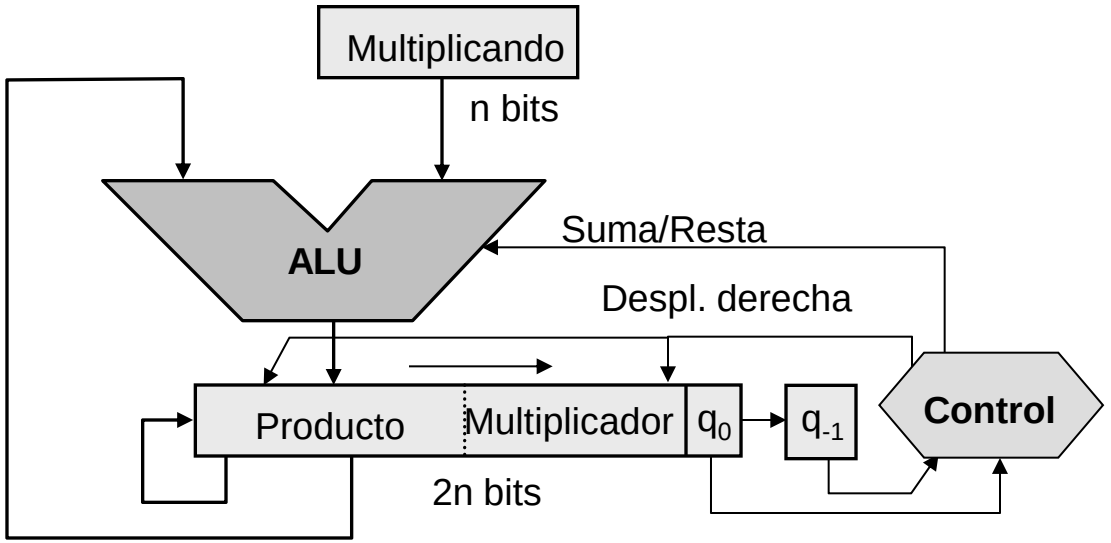
$Producto_h = producto_h$ - Multiplicando

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.



Multiplicando = 1010
Multiplicador = 1110

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**	Desplazamiento dcha.	1
1010	0110 0111	0	10 → Resta	2
1010	0011 001**1**	**1**	Desplazamiento dcha.	2
1010	0011 0011	1	11 → Ninguna operación	3
1010	0001 100**1**	**1**	Desplazamiento dcha	3
1010	0001 1001	1	11 → Ninguna operación	4
1010	**0000 1100**	1	Desplazamiento dcha.	4

La división

- La división la podemos expresar como:

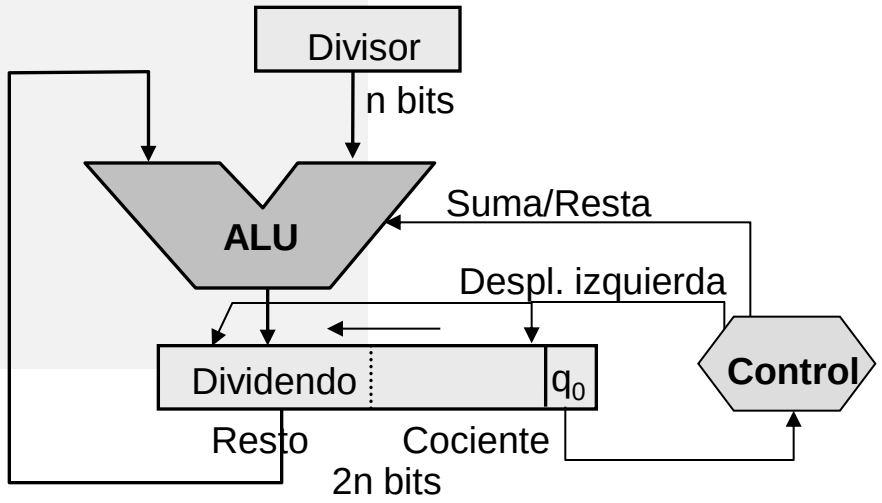
  Dividendo = Cociente x Divisor + Resto

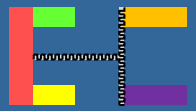
- El resto es más pequeño que el divisor. Hay que reservar el doble de espacio para el dividendo.
- Supondremos operandos positivos.

```
Dividendo →  10010011 | 1011    ←Divisor
             10010    | 01101   ←Cociente
              1011
             ------
             001110
               1011
             ------
             001111
                1011
             ------
                0100            ←Resto
```

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
```





ALGORITMO SIN RESTAURACIÓN

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

↑ ↑

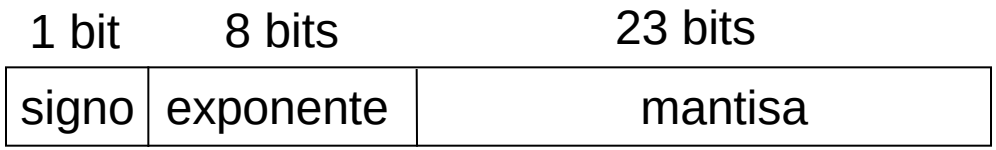
Resto Cociente

Coma flotante

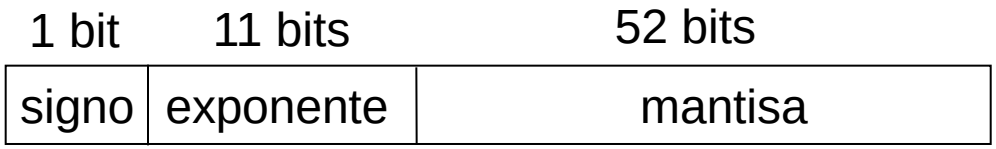
- Representación para números fraccionarios
  - Coma fija 1234,567
  - Logarítmica log 123,456 = 2,0915122
  - Coma flotante $1{,}234566 \times 10^3$
  - Otras
- Ventajas de estandarizar una representación determinada
  - Posibilidad de disponer de bibliotecas de rutinas aritméticas
  - Técnicas de implementación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

ESTÁNDAR IEEE 754 PARA COMA FLOTANTE

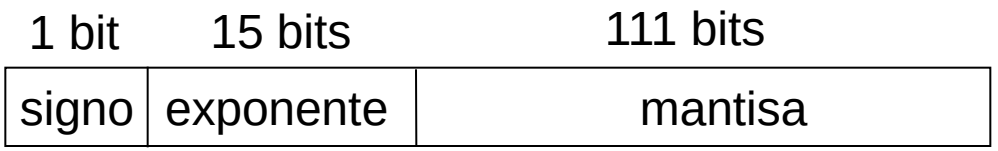
- Formatos
  - Simple precisión (32 bits)



  - Doble precisión (64 bits)



  - Cuádruple precisión (128 bits)



ESTÁNDAR IEEE 754 PARA COMA FLOTANTE

- 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
- Números
  - $(-1)^S \times 1,M \times 2^{E-127}$ simple precisión
  - $(-1)^S \times 1,M \times 2^{E-1023}$ doble precisión

ESTÁNDAR IEEE 754 PARA COMA FLOTANTE

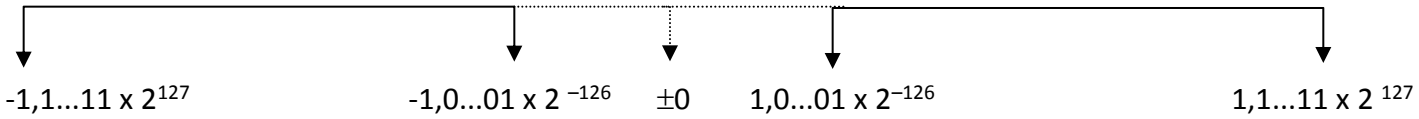
- Casos especiales

E	M	Valores
2^q-1	≠0	NaN (no un Número)
2^q-1	0	+∞ y -∞ según el signo de S
0	0	Cero
0	≠0	Números desnormalizados

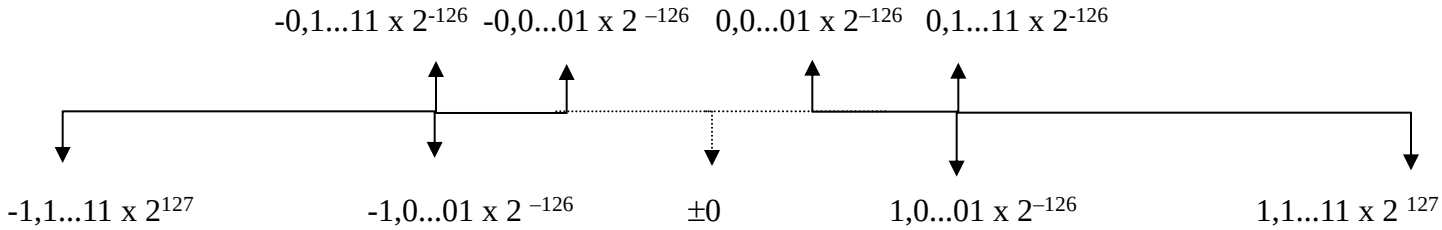
- NaN resultado de operaciones tales como 0/0, $\sqrt{-1}$
- El valor cero tiene dos representaciones +0 y –0.

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

CONVERSIÓN DECIMAL-IEEE 754

- Ejemplo: -0.81375 x 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

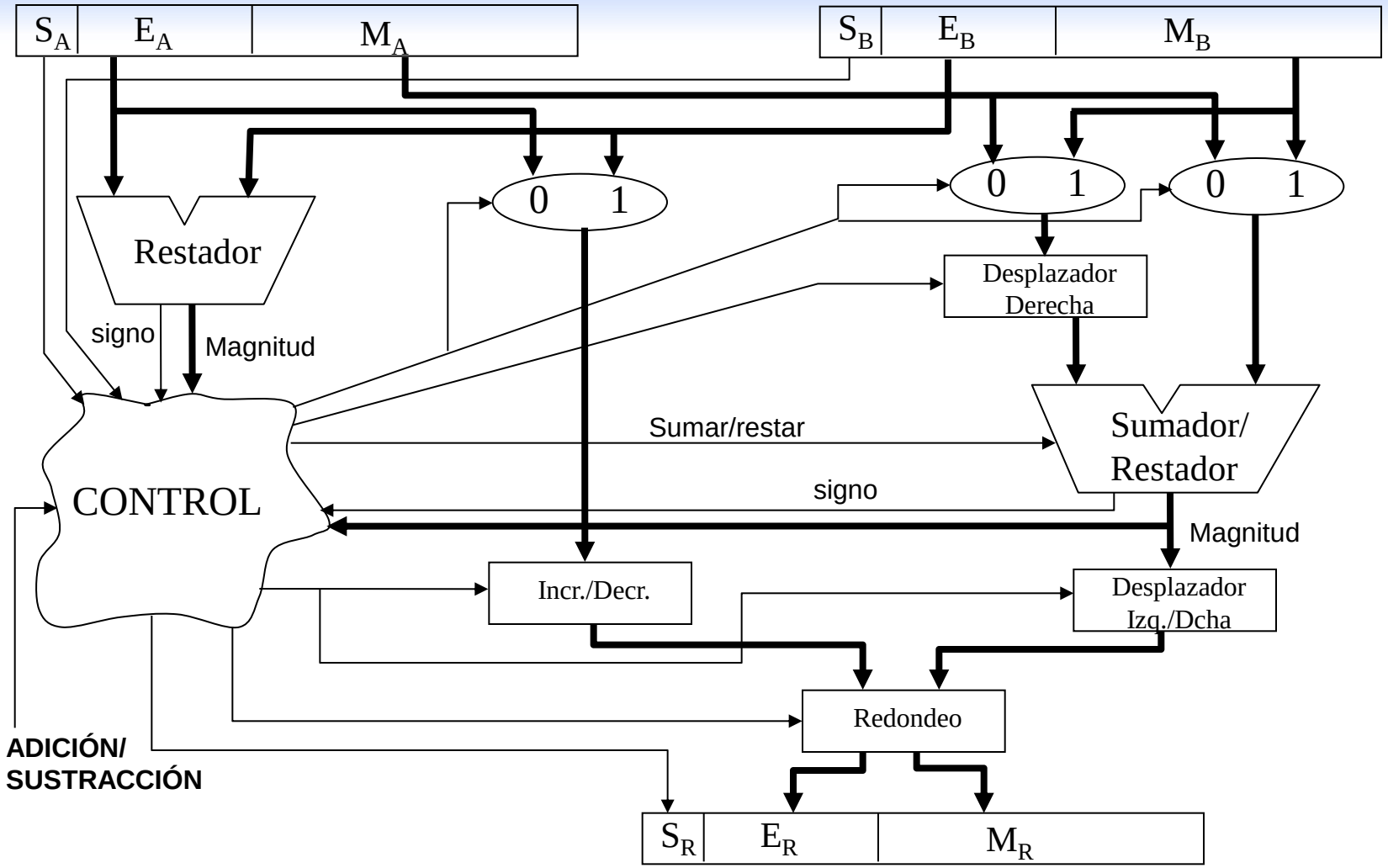
- 1. Seleccionar el número de menor exponente y desplazar su mantisa hacia la derecha tantas posiciones como la diferencia de los exponentes en valor absoluto.
- 2. Igualar el exponente del resultado al exponente mayor.
- 3. Operar las mantisas (según operación seleccionada y signos de ambos números)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

- 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

Circuito Sumador/Restador

Unidad aritmética flotante. Sumar y restar



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

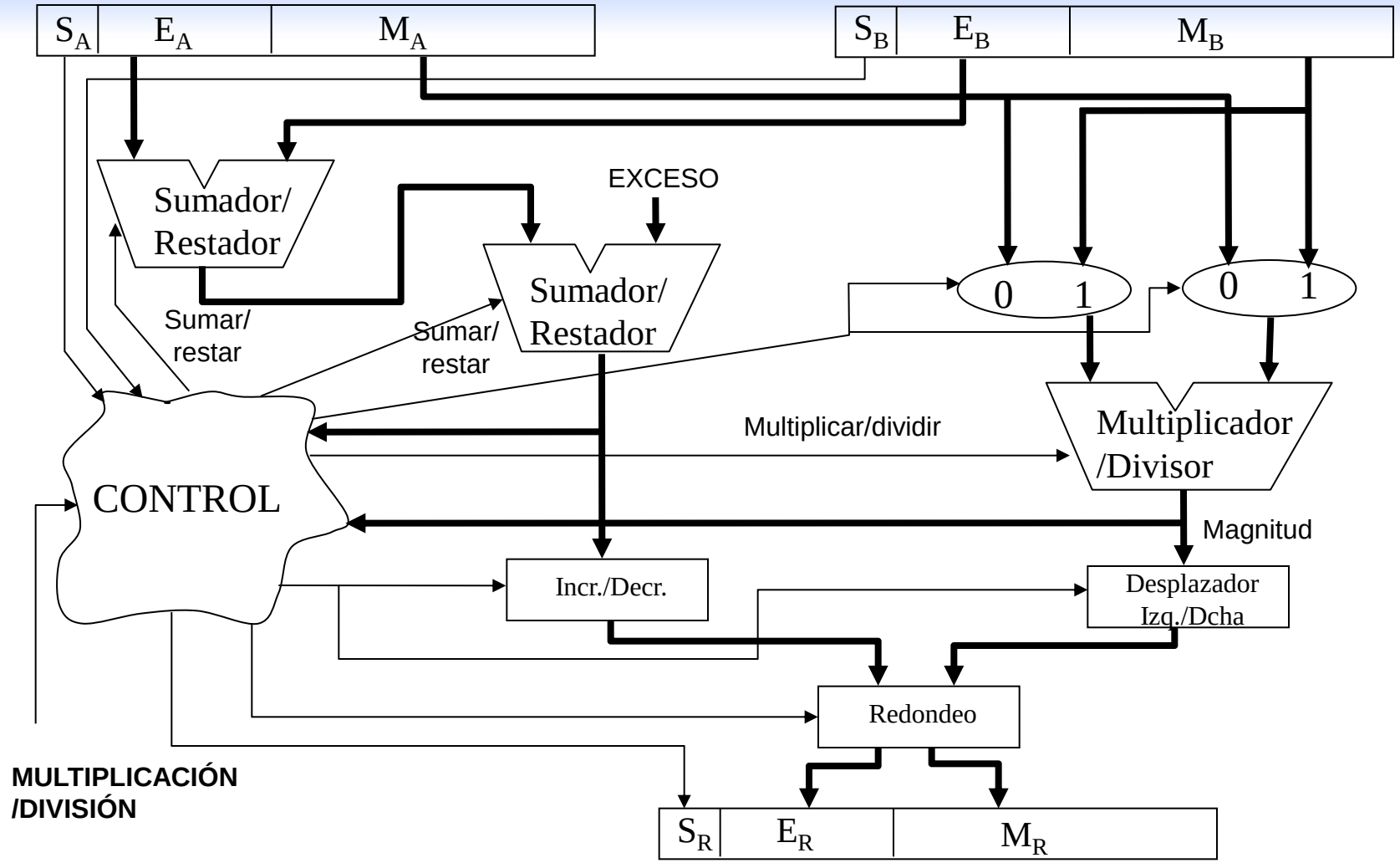
- Reglas de División
  - 1. Restar los exponentes y sumar el exceso para obtener el exponente resultado
  - 2. Dividir las mantisas para determinar la mantisa del resultado.
  - 3. Procesar los signos.
  - 4. Normalizar y redondear si es necesario.

- Procesamiento de los signos

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

$$S_R = S_A \oplus S_B$$

CIRCUITO MULTIPLICADOR/DIVISOR



MULTIPLICACIÓN /DIVISIÓN

TÉCNICAS DE REDONDEO

- Las técnicas de redondeo consisten en limitar el número de bits al disponible en el sistema de representación utilizado.
- 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 V_{i-1} o V_i como representación R de la cantidad C
- El error absoluto se define como: $\varepsilon = |R - C|$
- La resolución se define como: $\Delta = |V_i - V_{i-1}|$
- Técnicas de redondeo
  - Truncamiento
  - Redondeo propiamente dicho
  - Bit menos significativo forzado a “uno”

- 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

REDONDEO AL MÁS PRÓXIMO

- Toma el valor más próximo al que se quiere representar

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

Ejemplo: representación con 8 bits de punto implícito

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

$V_{i-1} = 0{,}01100000 \equiv 0{,}375$

$V_i = 0{,}01100001 \equiv 0{,}37890625$

$|V_{i-1} - C| = 0{,}000976563$

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

$\longrightarrow R = 0{,}01100000$

BIT MENOS SIGNIFICATIVO FORZADO A "UNO"

- 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.