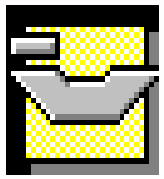


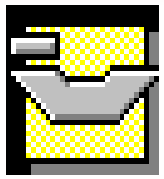
ARQUITECTURAS DE COMPUTADORES

2º CURSO INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

TEMA 3 – UNIDAD ARITMÉTICO-LÓGICA

JOSÉ GARCÍA RODRÍGUEZ
JOSÉ ANTONIO SERRA PÉREZ

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



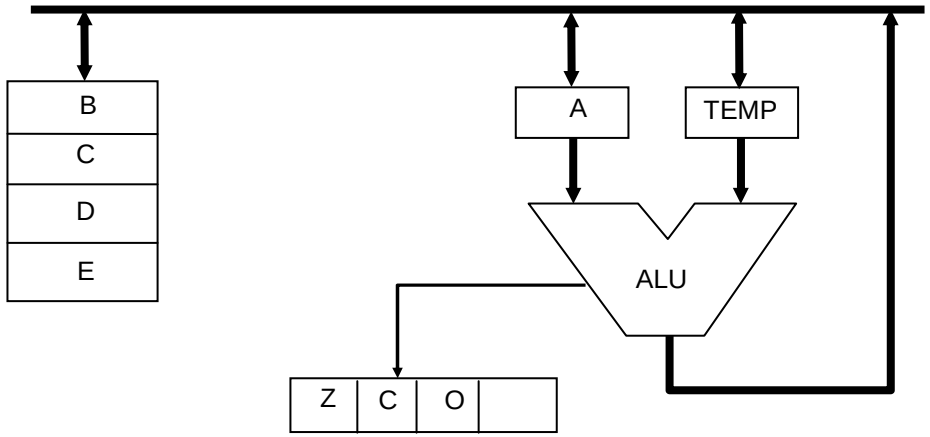
La ALU

La Unidad Aritmética y Lógica

- Introducción
- Operaciones Lógicas
- La suma y la resta
  - Sumador con propagación de acarreo (CPA)
  - Circuito sumador-restador
  - Desbordamiento
  - Sumadores con anticipación de acrreo (CLA)
- La multiplicación
  - Multiplicación binaria sin signo
  - Multiplicación binaria con signo
  - Algoritmo de Booth
- La división
- Conclusiones

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE

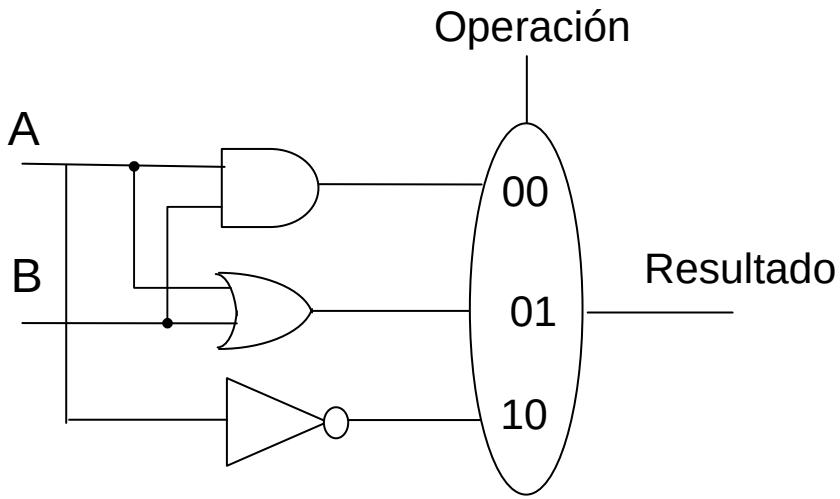
Introducción



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

Operaciones lógicas

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



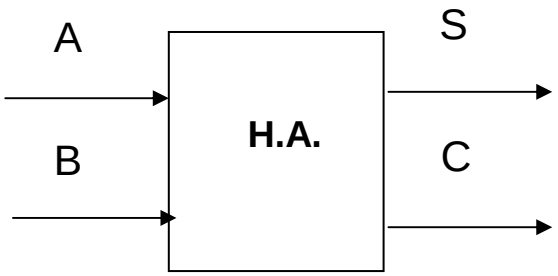
La suma y la resta

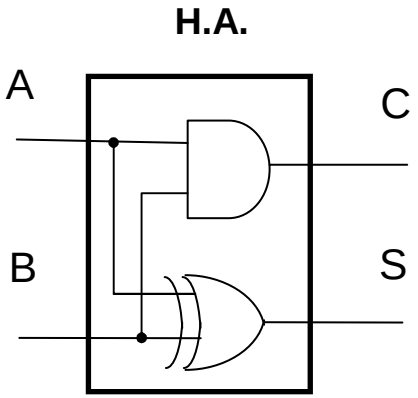
Semisumador Binario (H.A.)

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

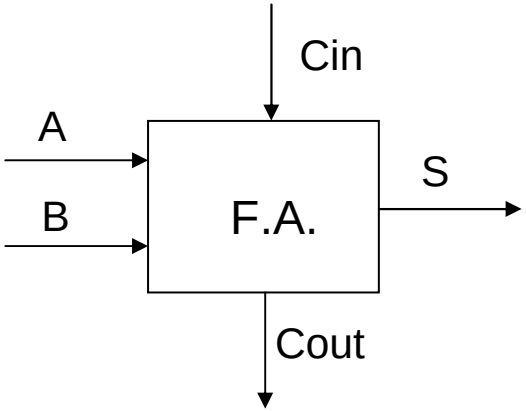
$$S = \overline{X} \cdot Y + X \cdot \overline{Y} = X \oplus Y$$

$$C = X \cdot Y$$





Sumador completo (F.A.)



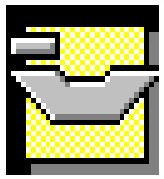
Entradas			Salidas	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

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

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

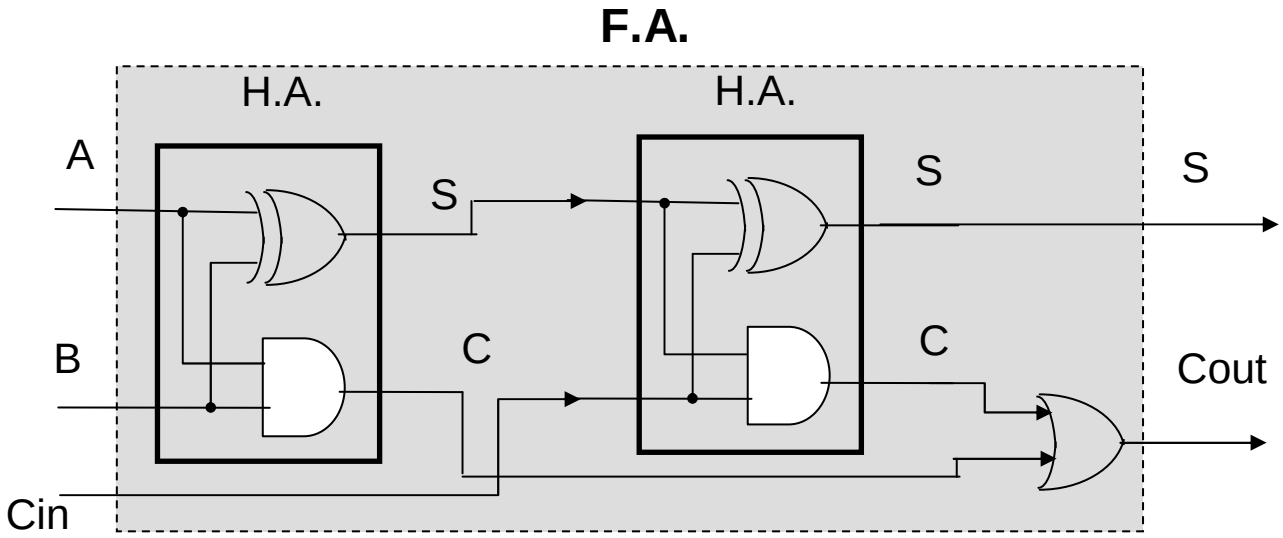
$$Cout = AB + Cin(A \oplus B)$$

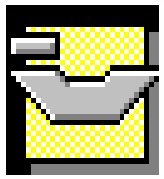


La suma
y la resta

Sumador completo (F.A.)

- Con semi-sumadores (H.A.)

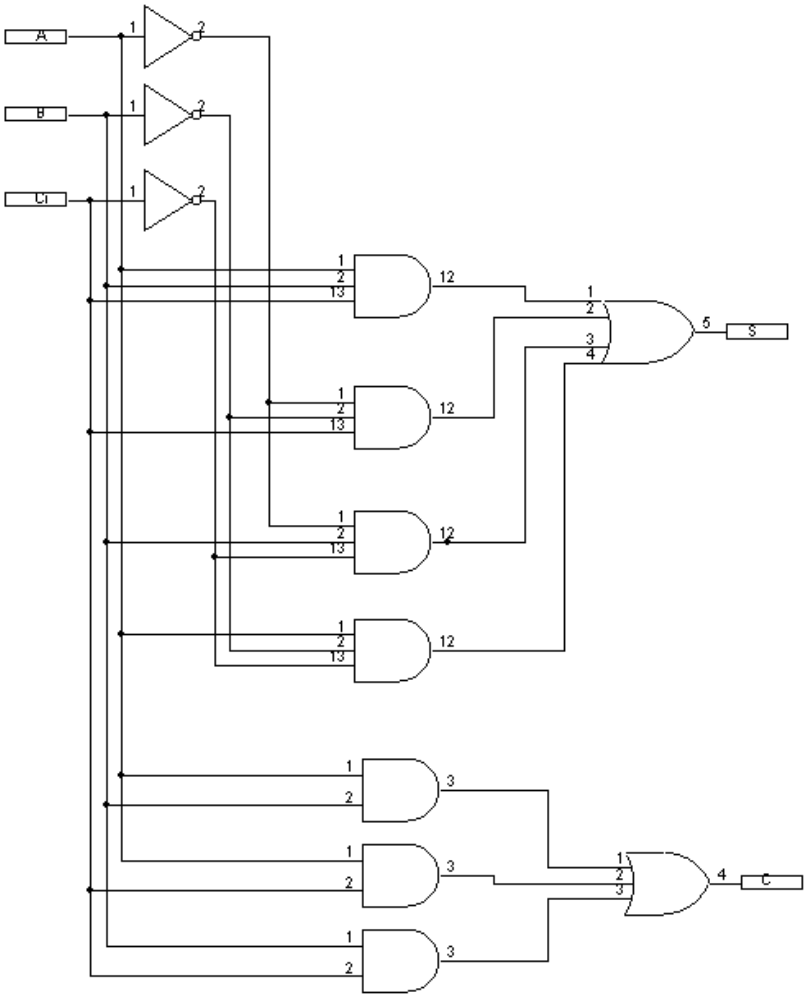




La suma
y la resta

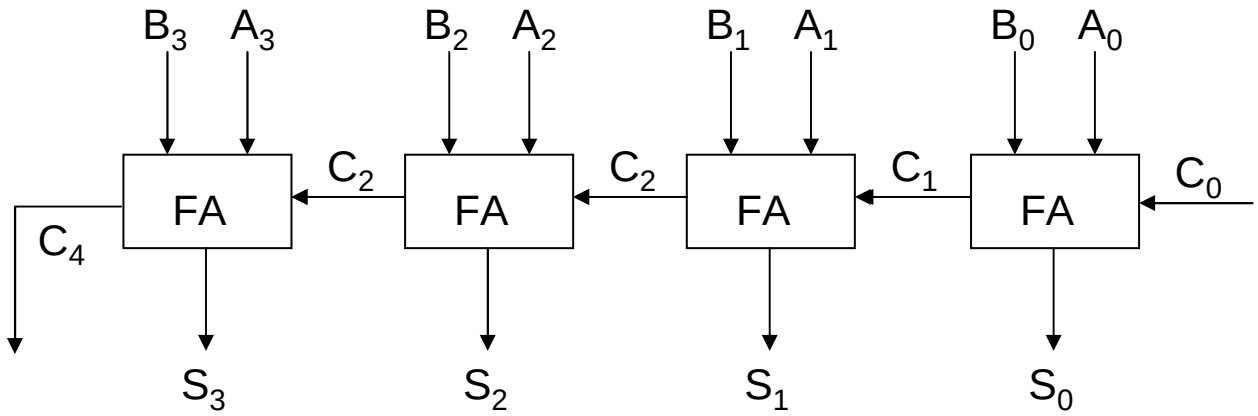
Sumador completo (F.A.)

- Con puertas



Sumador con propagación de acarreo

- Para sumar dos números de n bits se necesita colocar en cascada n sumadores completos.
- El acarreo se propaga de una etapa a la siguiente: Sumador con Propagación de Acarreo (Carry Propagated Adder)

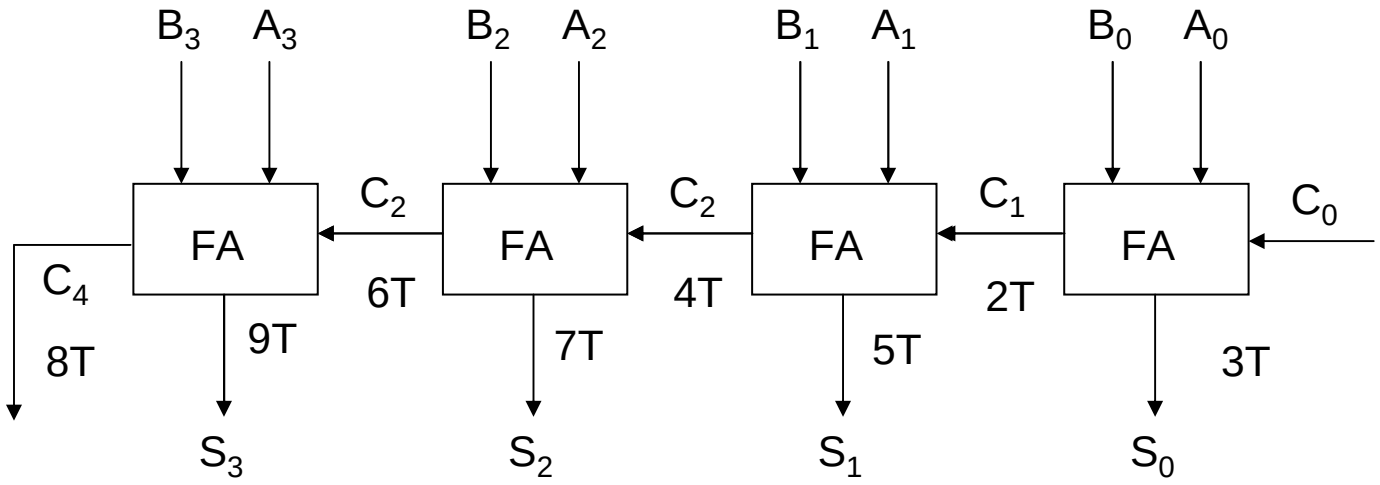


Sumador con propagación de acarreo

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

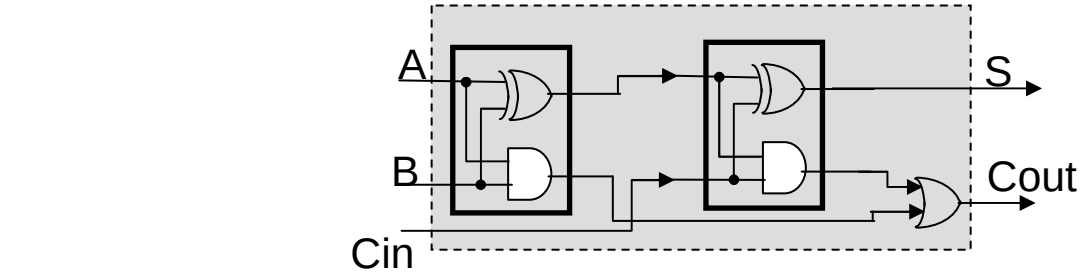
$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin$$

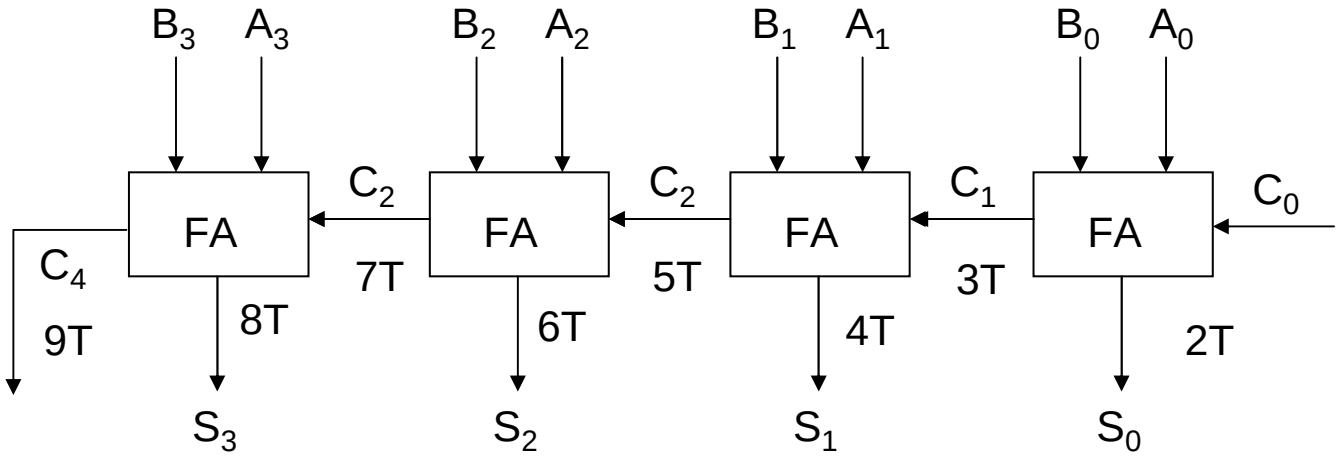
$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin$$



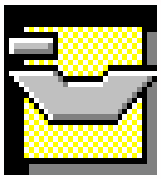
Sumador con propagación de acarreo

- Sumadores completos construidos con semisumadores:





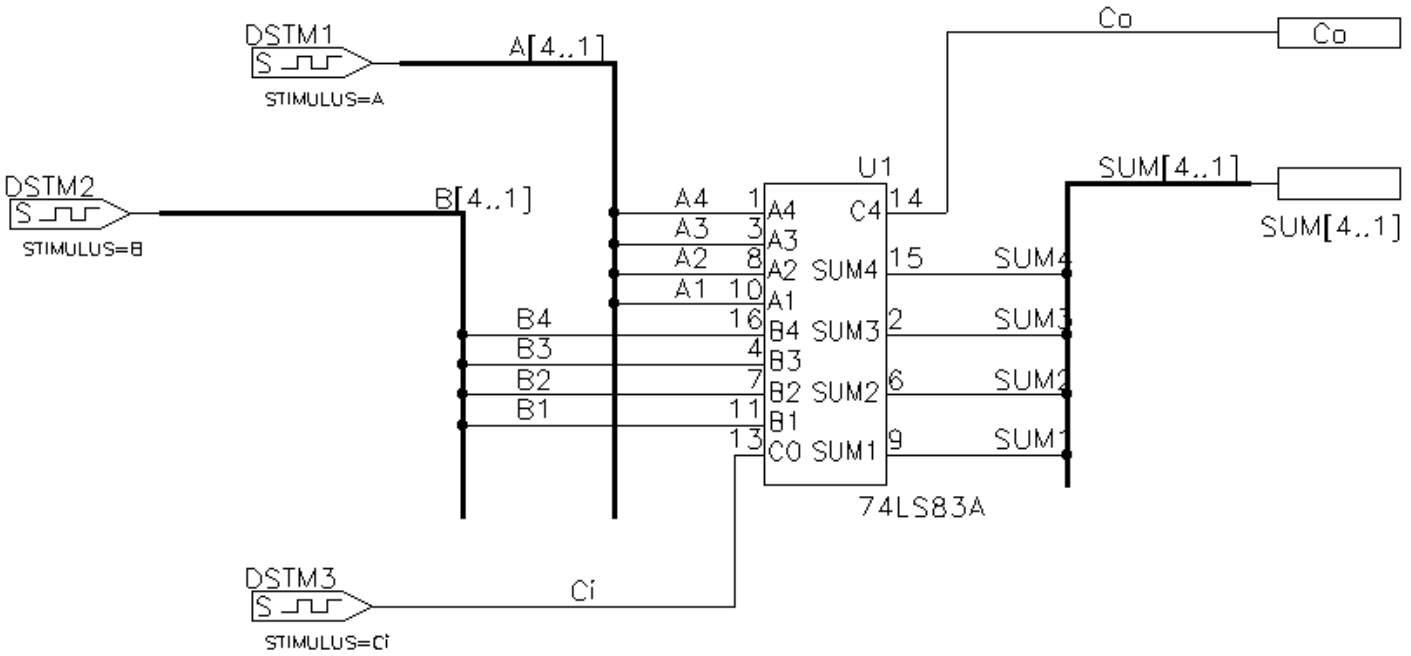
$$Tiempo\_Total = (2 \cdot n + 1)T$$

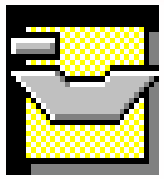


La suma
y la resta

Sumador 74ls83

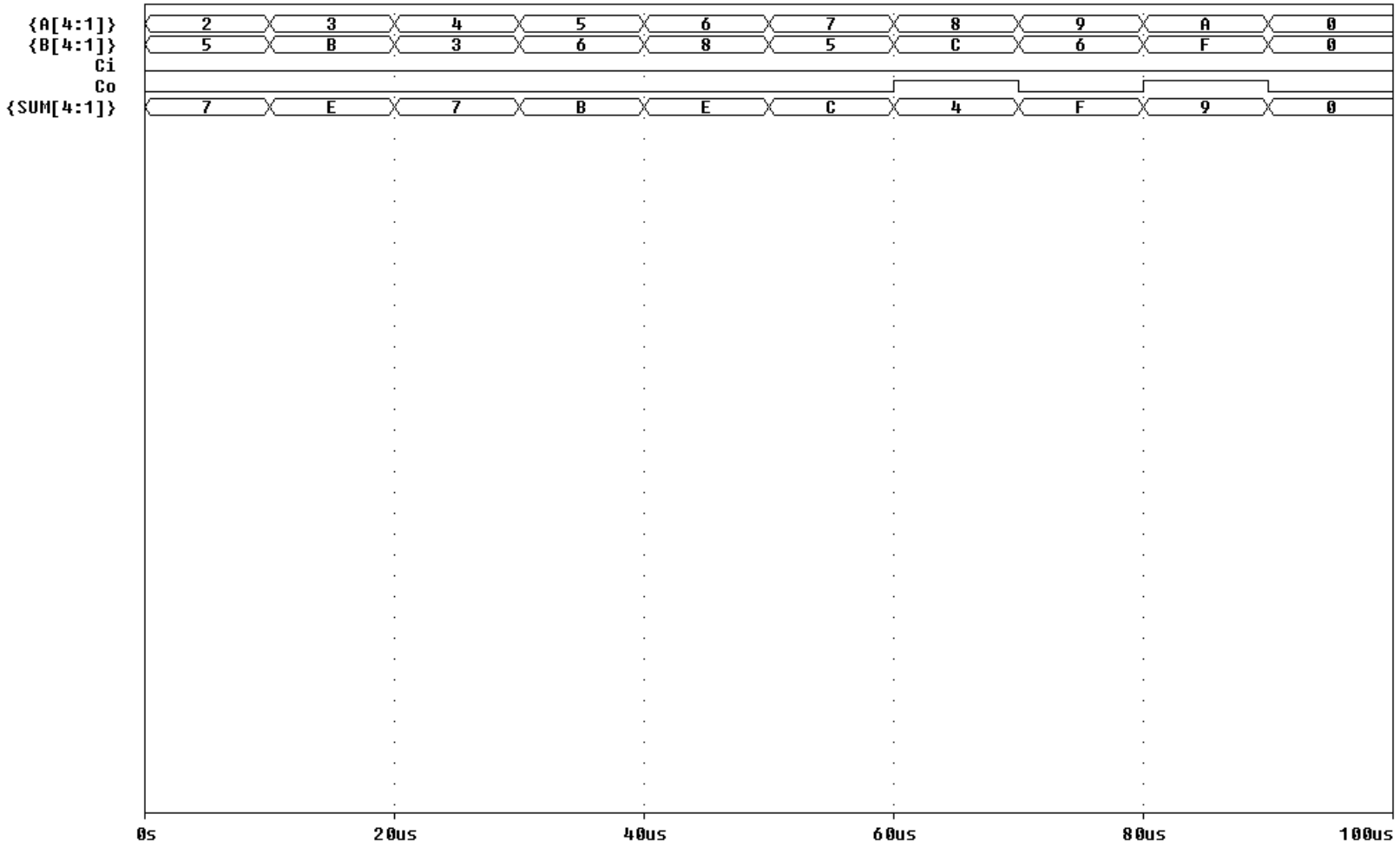
- Sumadores integrado

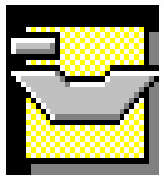




La suma
y la resta

Sumador 74ls83

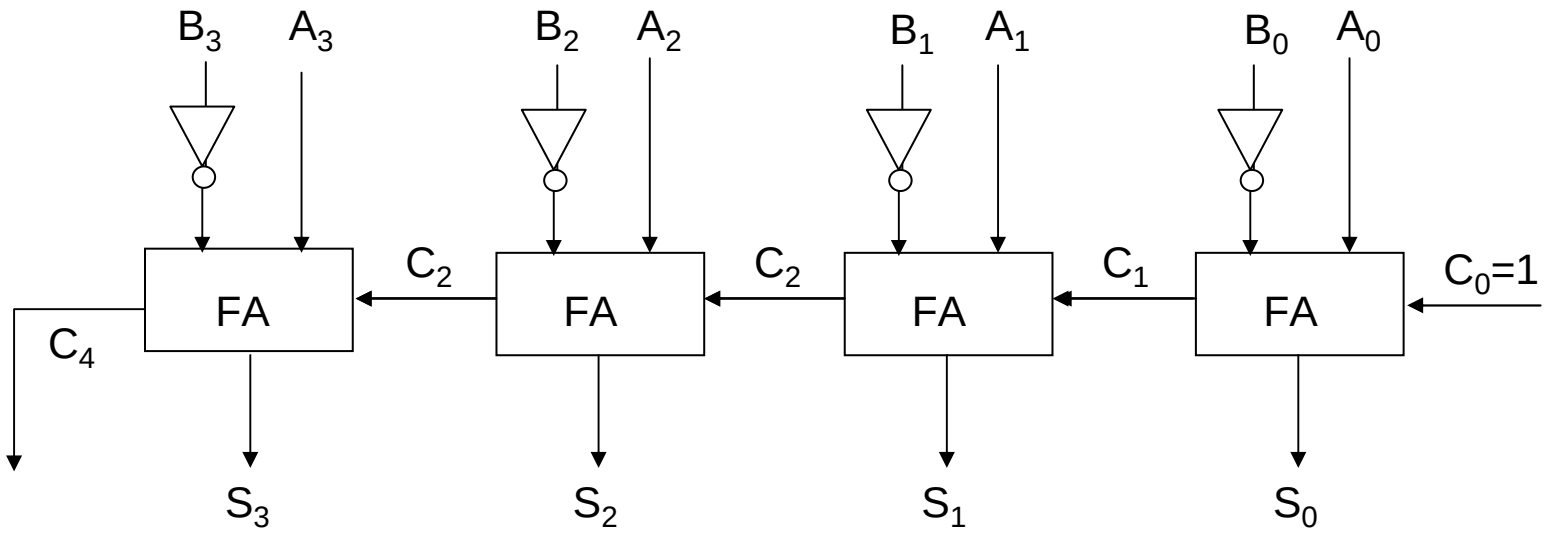


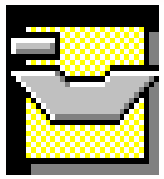


La suma
y la resta

Circuito restador

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



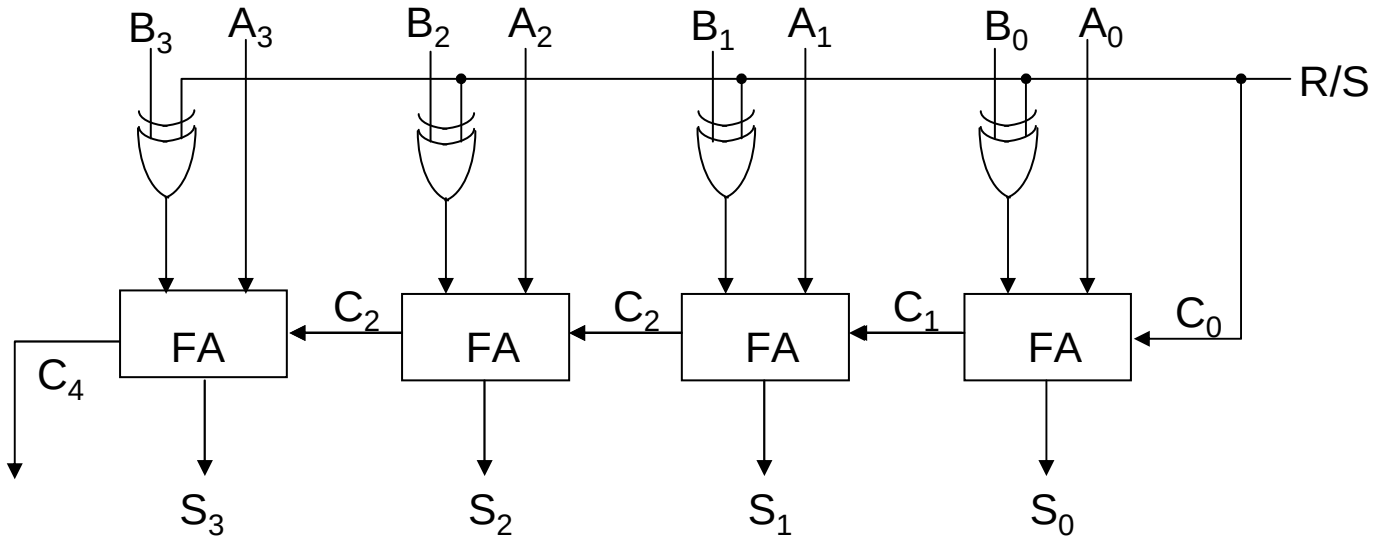


La suma
y la resta

Circuito Sumador-Restador

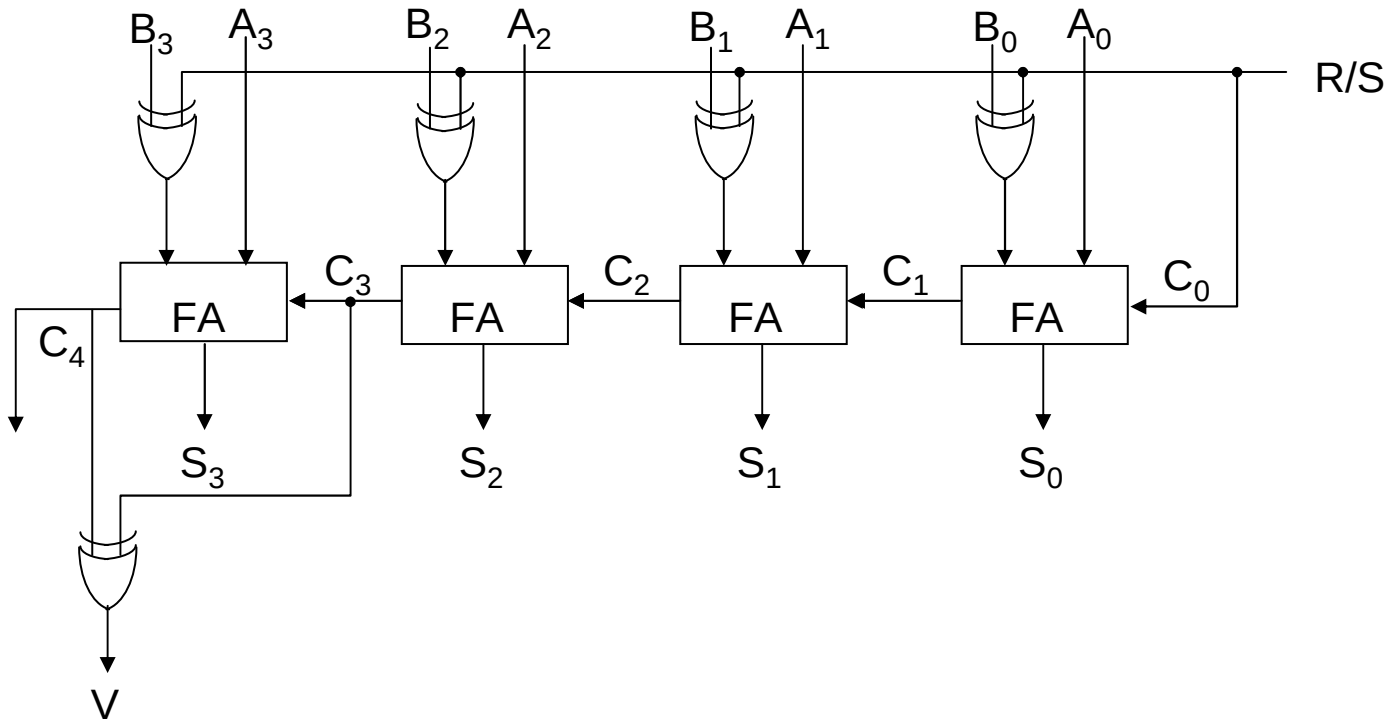
R/S	B_i	Entrada al FA
0	0	0
0	1	1
1	0	1
1	1	0

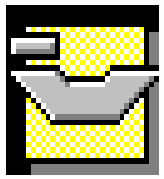
$$Tiempo\_Total = 2(n+1)T$$



Detección de desbordamiento

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





La suma
y la resta

Detección de desbordamiento

1. Caso suma de dos positivos

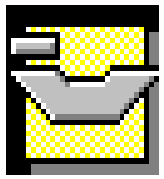
		S			
	C4	C3	C2	C1	
	0	1	1	1	
		0	1	1	1
		0	1	1	1
		1	1	1	0

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



La suma
y la resta

Sumador con anticipación de acarreo

- 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:

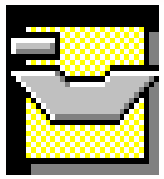
$$C_0 = G_0 + P_0 \cdot C_{-1}$$

$$C_1 = G_1 + P_1 \cdot C_0$$

$$C_2 = G_2 + P_2 \cdot C_1$$

$$C_3 = G_3 + P_3 \cdot C_2$$

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



La suma y la resta

Sumador con anticipación de acarreo

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

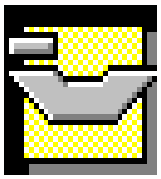
$$C_0 = G_0 + P_0 \cdot C_{-1}$$

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

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

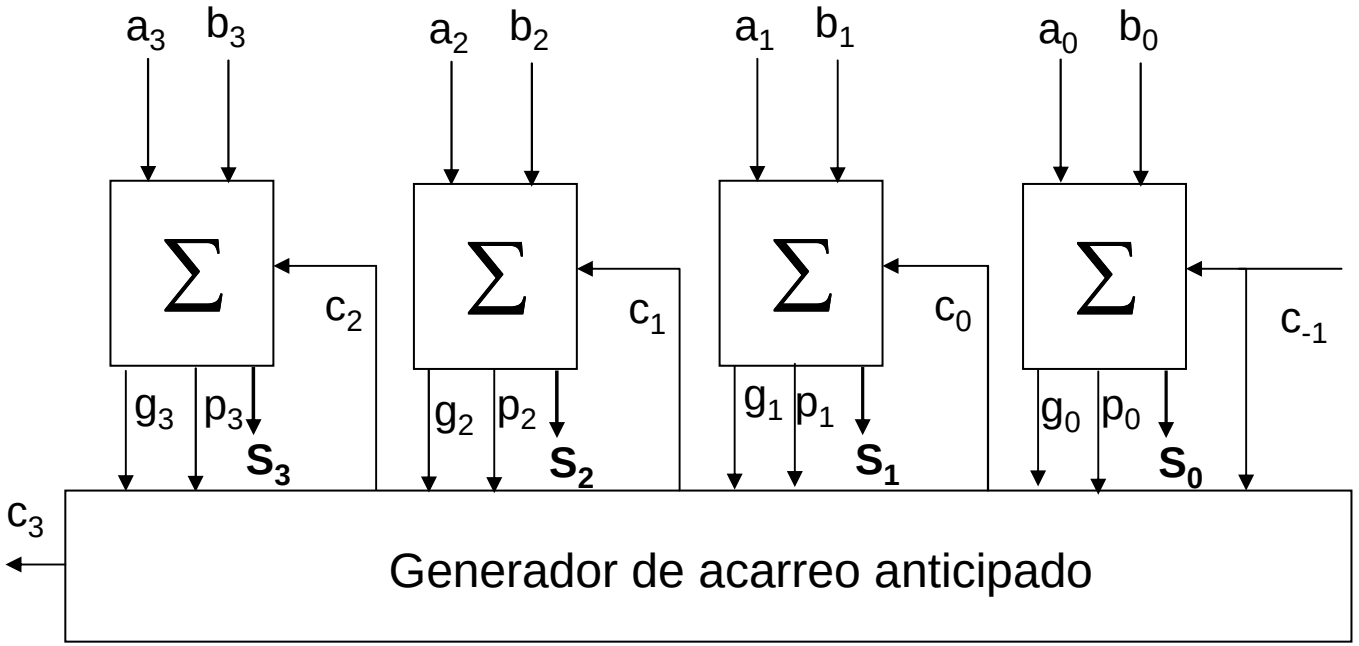
$$C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

- 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.

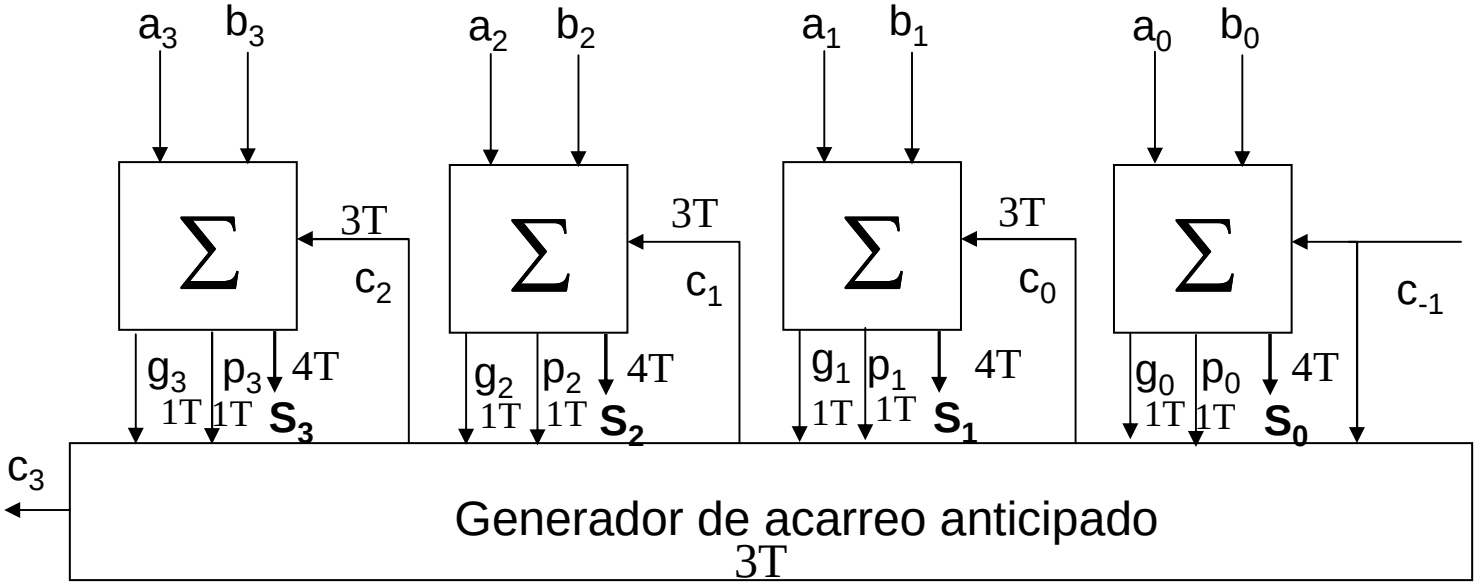


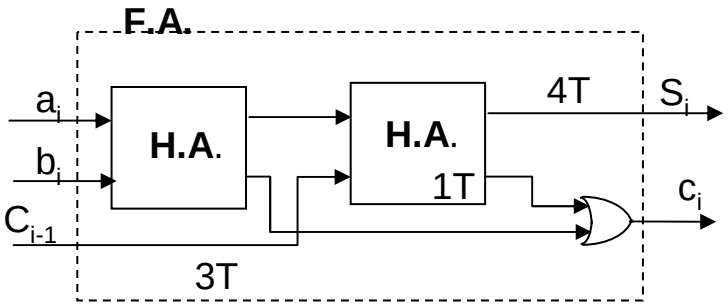
La suma
y la resta

Sumador con anticipación de acarreo

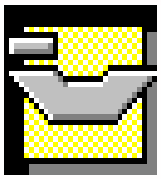


Sumador con anticipación de acarreo



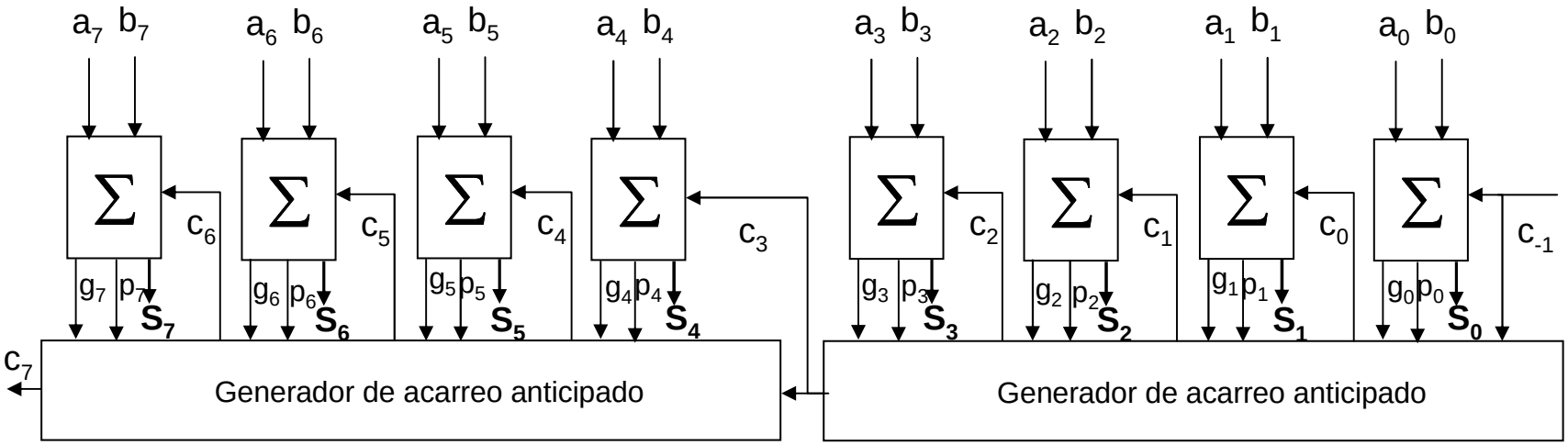


Sumadores construidos
con semisumadores

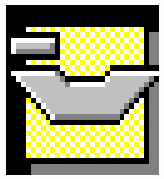


La suma
y la resta

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.



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

Multiplicación binaria sin signo

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.

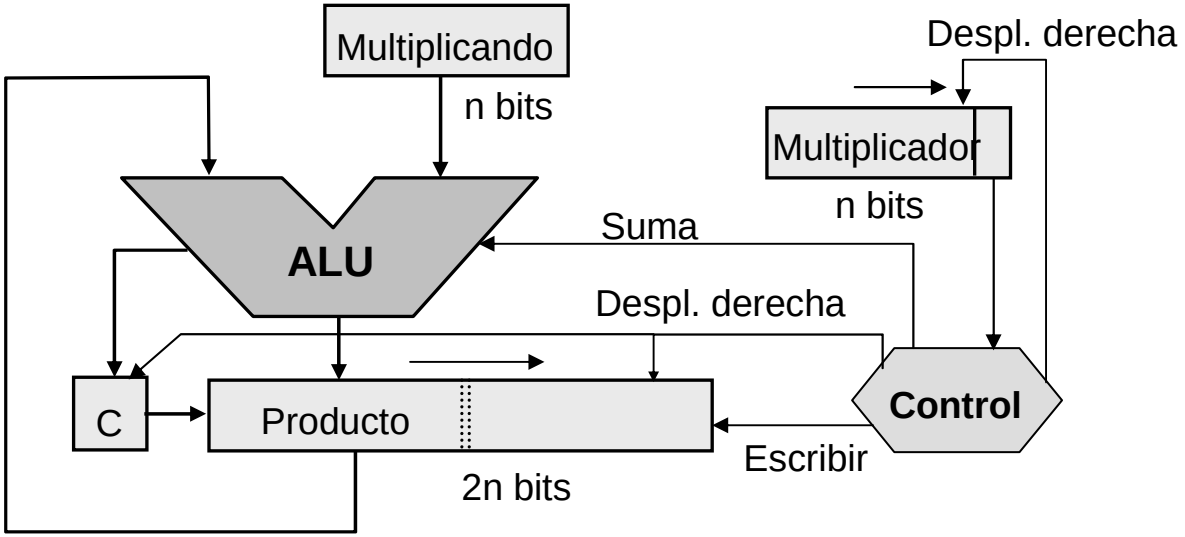
  Fin entonces

  Desplazar 1 bit a la derecha el registro producto

  Desplazar 1 bit a la derecha el registro multiplicador

Fin repetir

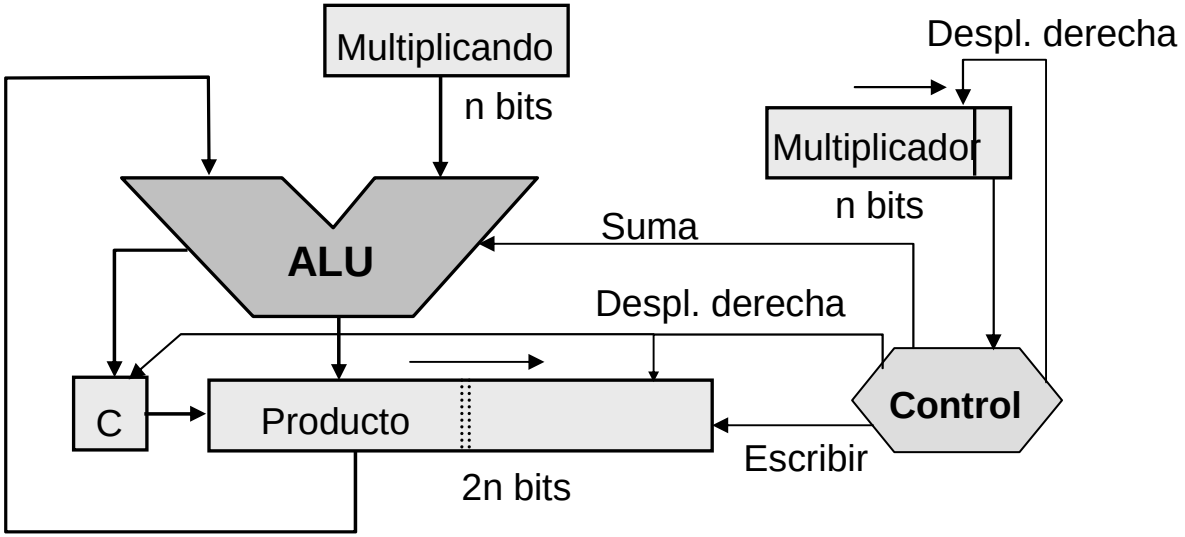
Versión preliminar



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



Multiplicación binaria sin signo

Repetir n veces

  Si el bit 0 del registro producto=1 entonces

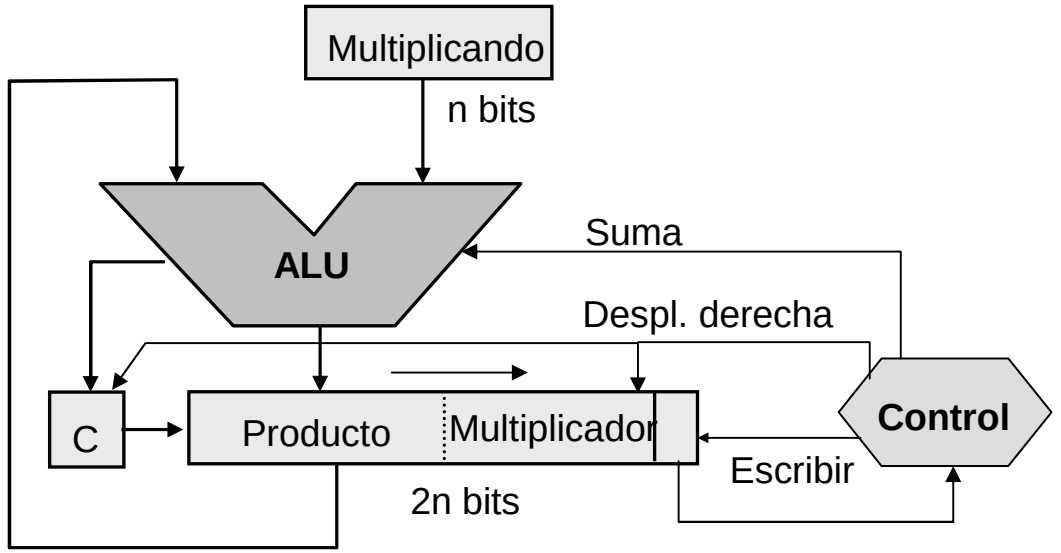
    Sumar el multiplicando a la mitad izquierda del producto y colocar el resultado en la mitad izquierda del producto.

  Fin entonces

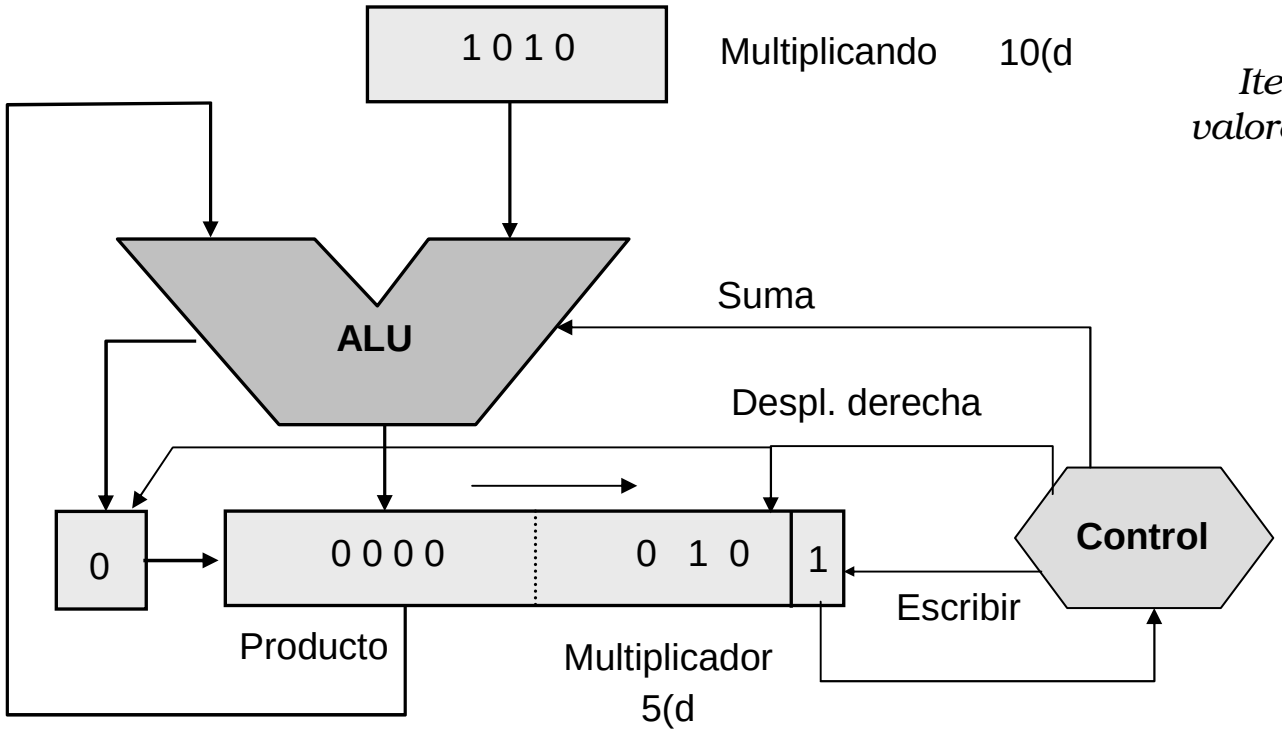
  Desplazar 1 bit a la derecha el registro producto

Fin repetir

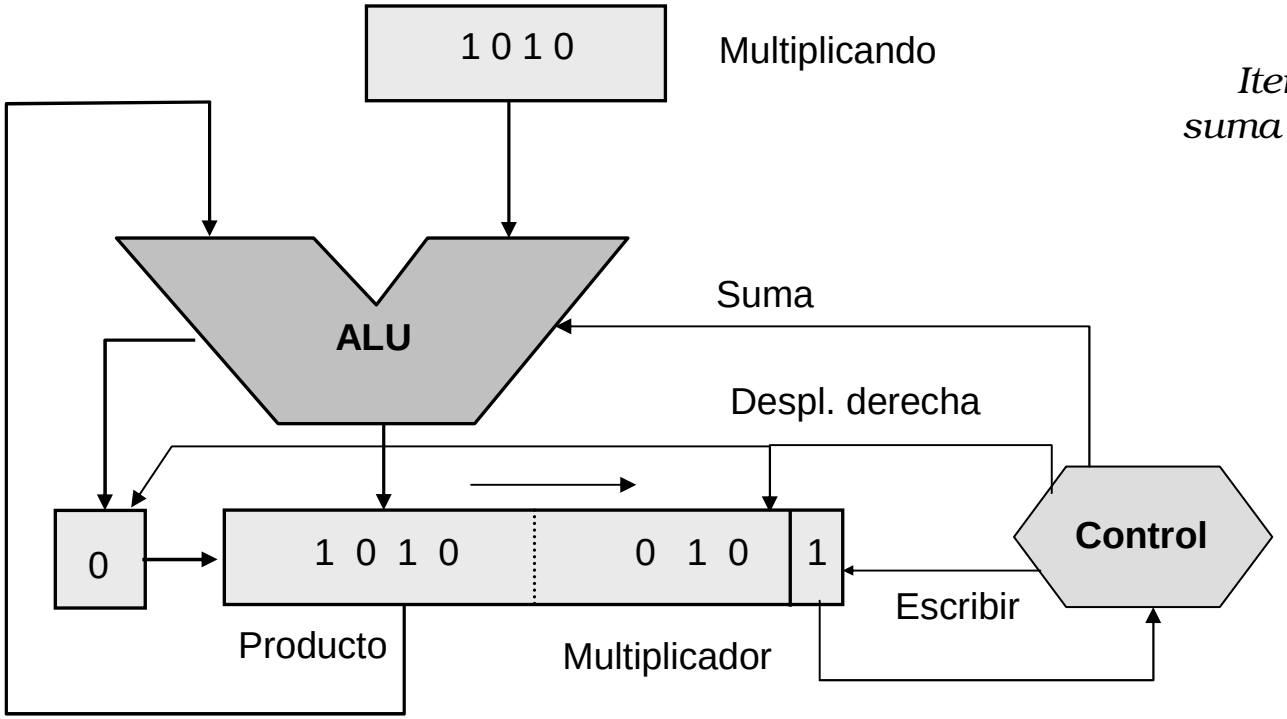
Versión final



Multiplicación binaria sin signo

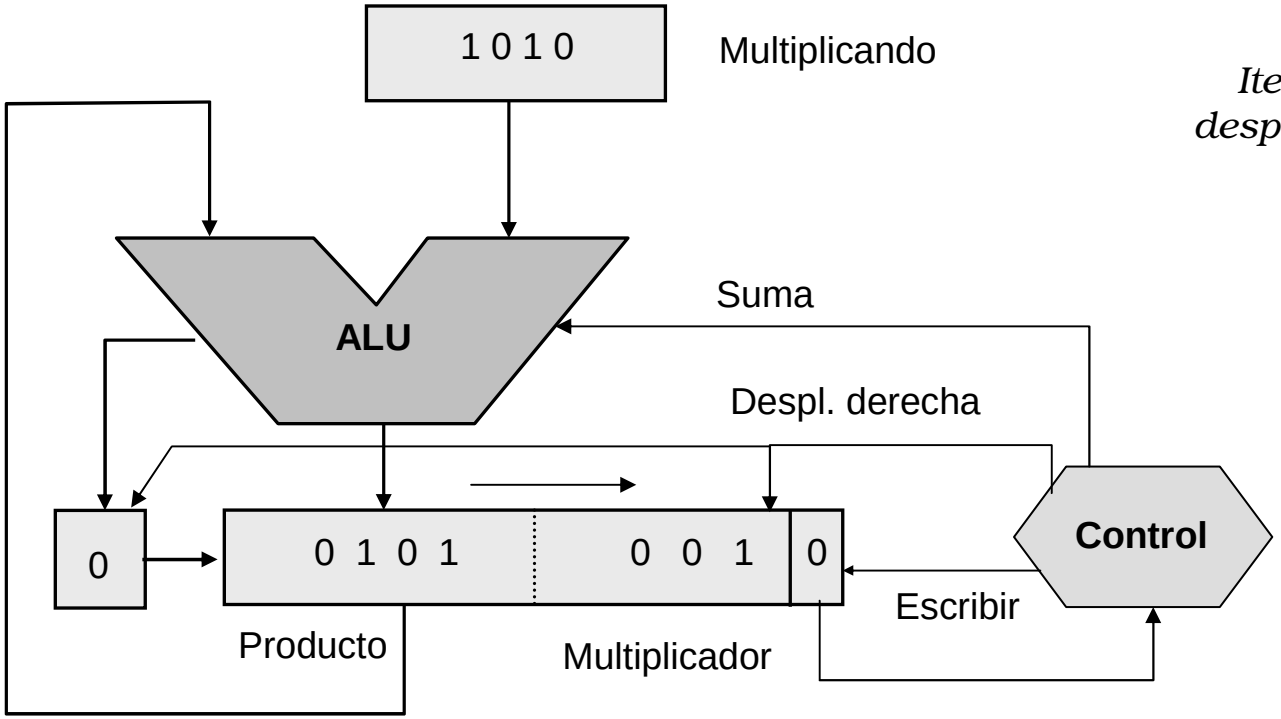


Multiplicación binaria sin signo



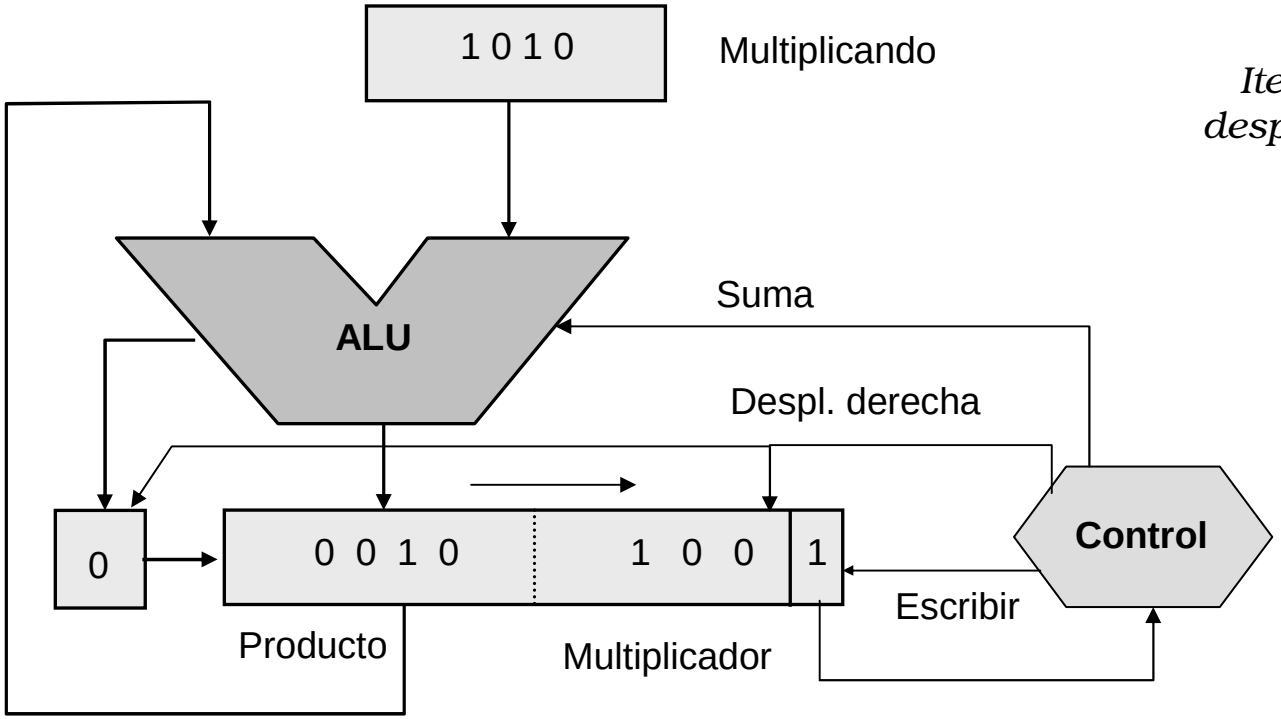
Iteración 1
suma P y Mando

Multiplicación binaria sin signo



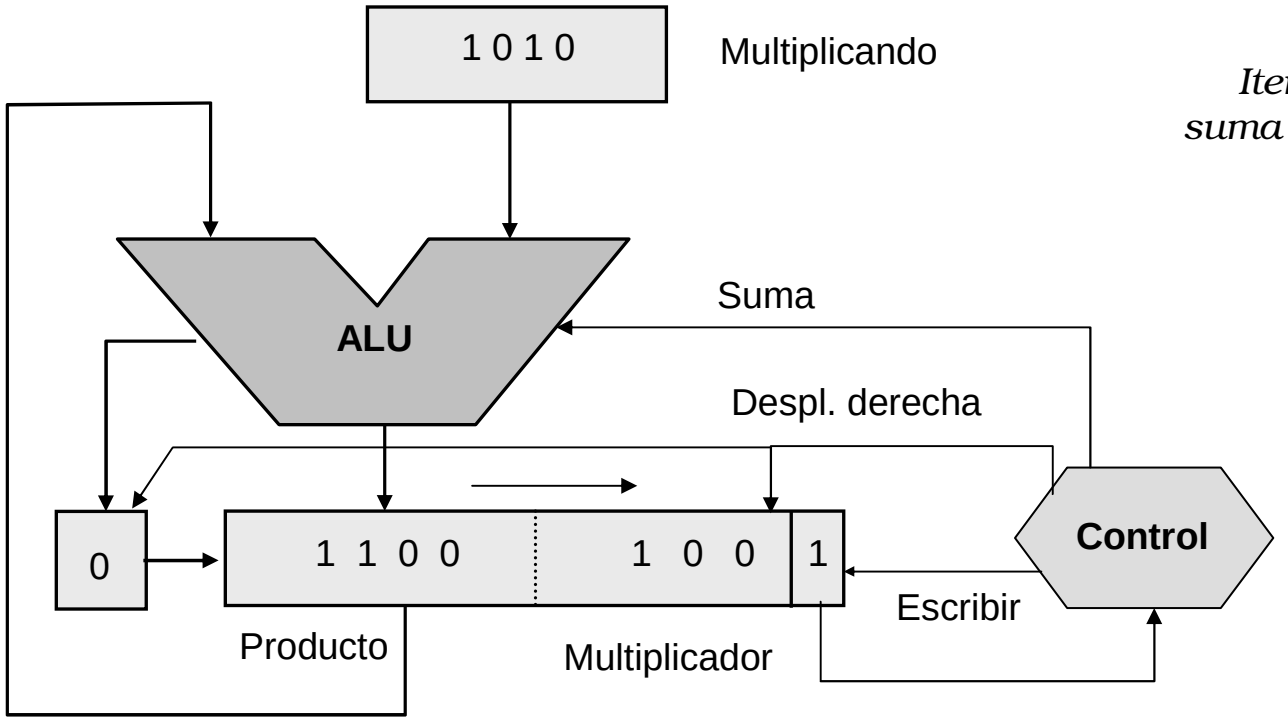
Iteración 1
despla. P dcha.

Multiplicación binaria sin signo



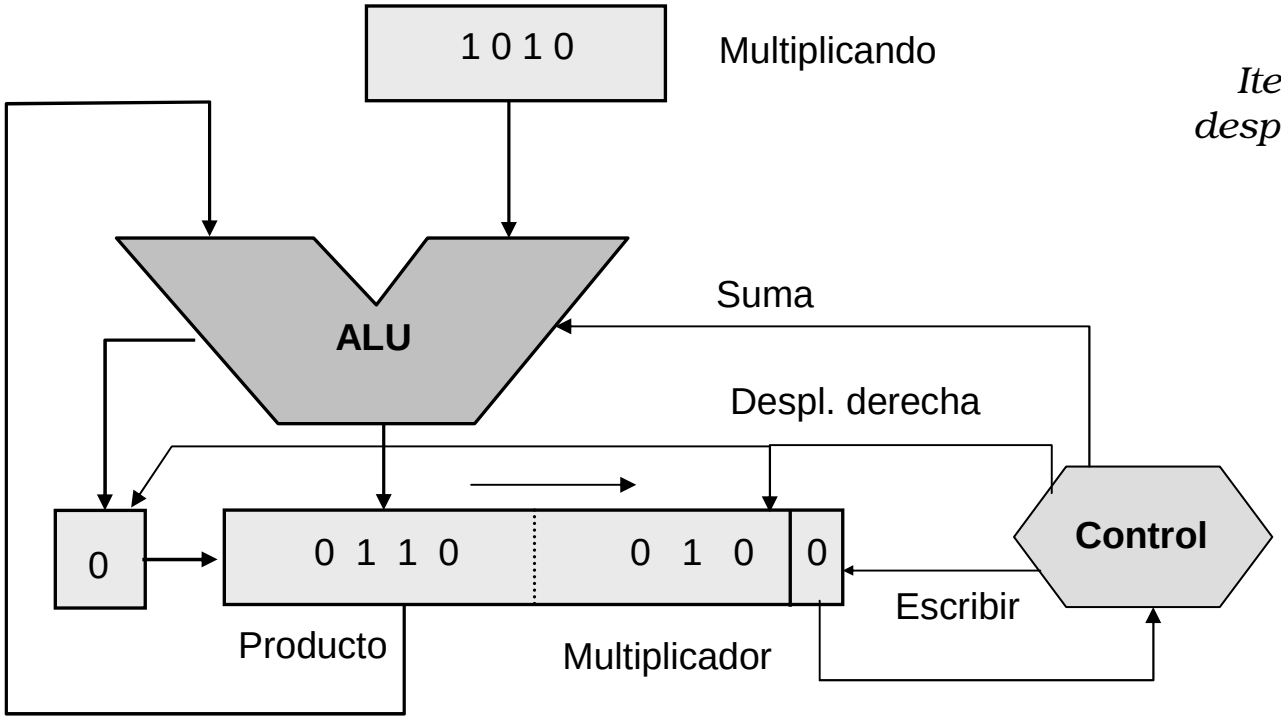
Iteración 2
despla. P dcha

Multiplicación binaria sin signo



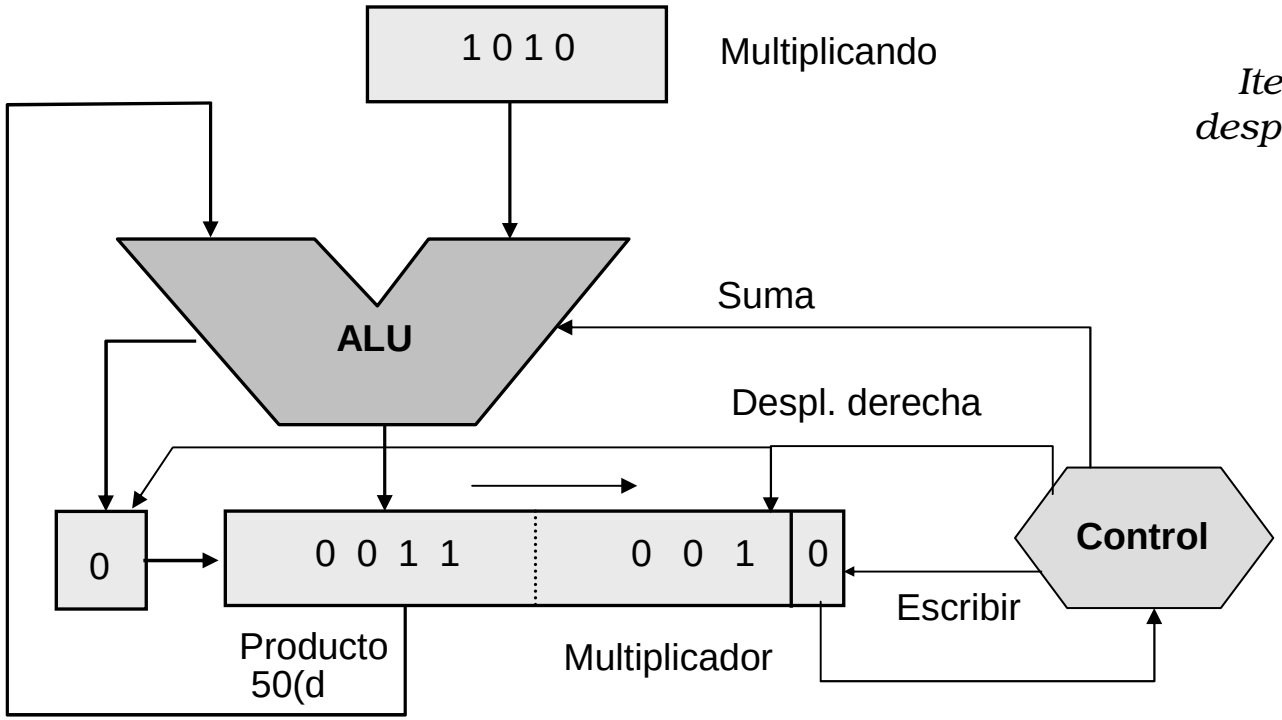
Iteración 3
suma P y Mando

Multiplicación binaria sin signo

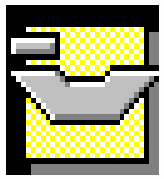


Iteración 3
despla. P dcha.

Multiplicación binaria sin signo



Iteración 4
despla. P dcha.



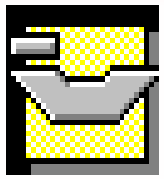
La multiplicación

Multiplicación binaria sin signo

Multiplicando = 1010

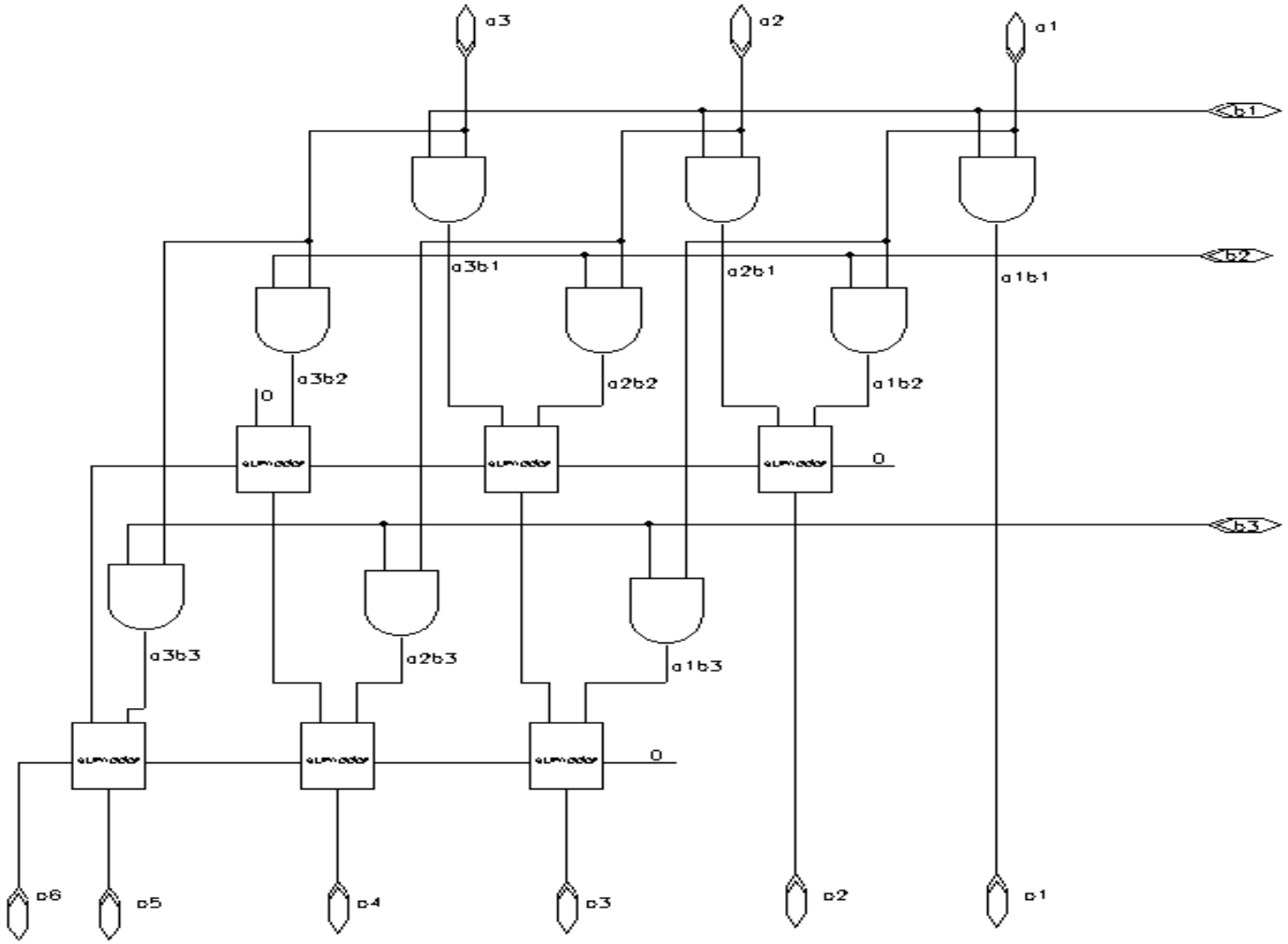
Multiplicador = 0101

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



La multiplicación

Multiplicación rápida



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:

$$0111 = 1000 - 0001 = +100\text{-}1$$

				0	0	1	0	Multiplicando
			x	+1	0	0	-1	Multiplicador según A. Booth
1	1	1	1	1	1	1	0	Restamos el multiplciadndo
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	

Algoritmo de Booth

Bit actual	Bit a la izquierda	Sustitución
0	0	0 (no hay transición)
0	1	-1 (transición hacia negativoo)
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

Algoritmo de Booth

Inicialmente $q_{-1}=0$

Repetir n veces

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

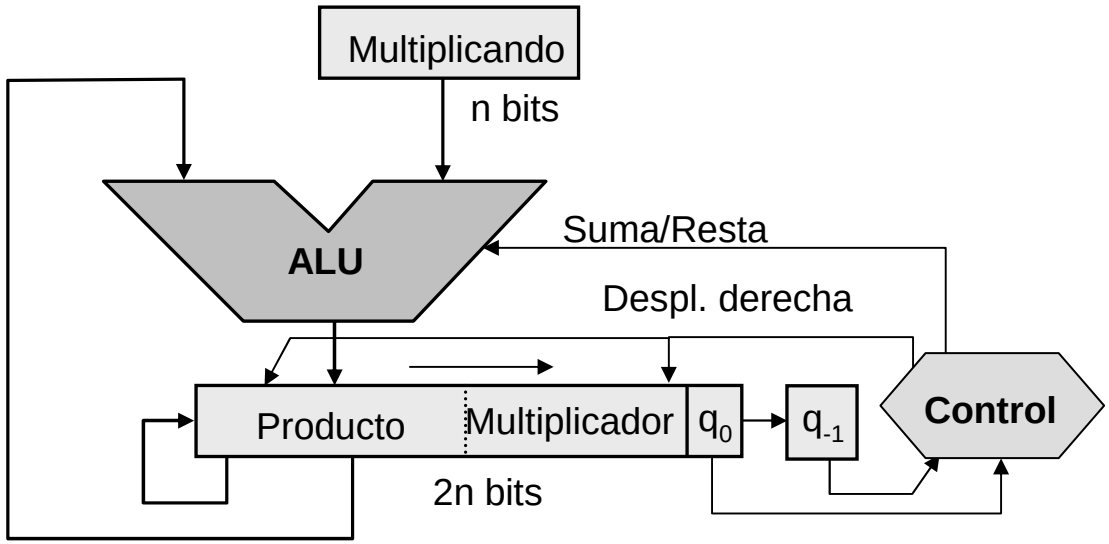
$Producto_h = producto_h$ - Multiplicando

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

$Producto_h = Producto_h$ + Multiplicando

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

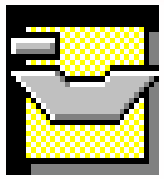
Fin repetir.



Algoritmo de Booth

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



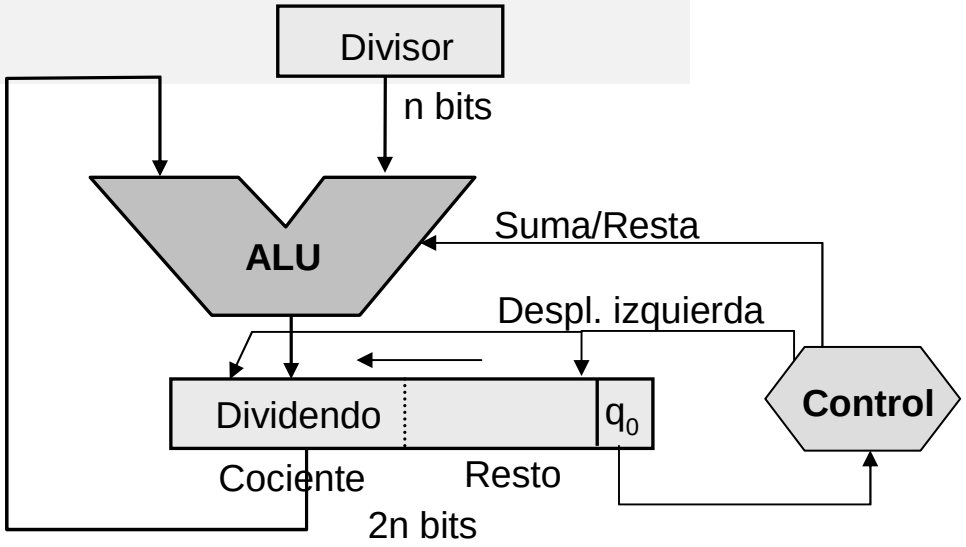
La división

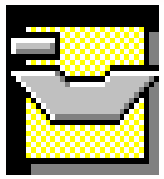
- La división la podemos expresar como:
  Dividendo = Cociente x Divisor + Resto
- El resto es más pequeño que el divisor. Hay que reservar el doble de espacio para el dividendo.
- Supondemos operandos positivos.

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

Algoritmo con restauración

Repetir n veces
    Desplazar el Dividendo a la izquierda
    $Dividendo_h = Dividendo_h - Divisor$
    Si $Dividendo_h < 0$ entonces (no cabe)
        $q_0=0$
        $Dividendo_h = Dividendo_h + Divisor$ (restaurar)
    Sino
        $q_0=1$
    Fin Si
Fin Repetir





Algoritmo con restauración

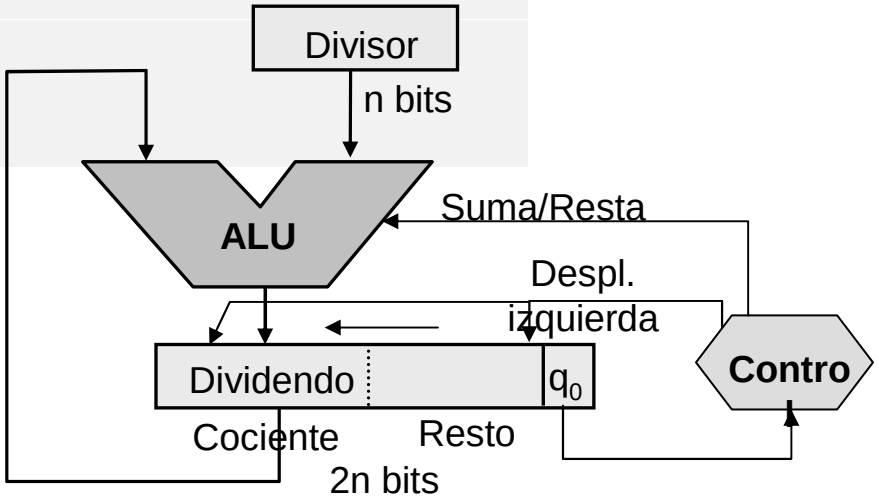
Dividendo	Divisor	Acción	Iteración
0101 0011	0110	Valores iniciales	0
1010 011_	0110	Desplazar un bit a izquierda	1
0100 011_	0110	Restar	1
0100 011**1**	0110	$Dividendo_h > 0 \Rightarrow q_0 = 1$	1
1000 11**1**_	0110	Desplazar un bit a izquierda	2
0010 11**1**_	0110	$Dividendo_h$ - Divisor (Restar)	2
0010 11**11**	0110	$Dividendo_h > 0 \Rightarrow q_0 = 1$	2
0101 1**11**_	0110	Desplazar un bit a izquierda	3
1111 1**11**_	0110	$Dividendo_h$ - Divisor (Restar)	3
1111 1**110**	0110	$Dividendo_h <= 0 \Rightarrow q_0 = 0$	3
0101 1**110**	0110	$Dividendo_h$ + Divisor (Restaurar)	3
1011 **110**_	0110	Desplazar un bit a izquierda	4
0101 **110**_	0110	$Dividendo_h$ - Divisor (Restar)	4
0101 **1101**	0110	$Dividendo_h > 0 \Rightarrow q_0 = 1$	4

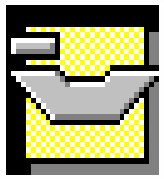
↑ ↑

Resto Cociente

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
```





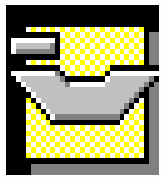
La división

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



Conclusiones

- Sumadores
  - Problemática temporal de los Sumadores con Propagación de Acarreo (CPA), especialmente si n elevado.
  - Los Sumadores con anticipación de acarreo (CLA) mejoran el tiempo de respuesta de los sumadores.
- Multiplicación
  - Problemática de la multiplicación de números con signo.
  - El algoritmo de Booth permite multiplicar números en Ca2 y en algunos casos reduce el números de operaciones si aparecen cadenas de 1's o 0's en el multiplicador.
- La División
  - Algoritmo para la división con restauración para números positivos. Si números negativos, entonces tratamiento previo del signo, y en función de éste se obtiene el signo del resultado.