

Introducción al Diseño Lógico (E0301)

Ingeniería en Computación

Gerardo E. Sager

Clase 8 curso 2025

Clase 13

- Temas a tratar
 - Suma – Resta, Multiplicación y División binarias.
 - Operación básica de una unidad aritmético lógica (ALU).
 - Operación de un circuito sumador/restador paralelo.
 - Circuitos Integrados ALU para diferentes operaciones lógicas y aritméticas sobre datos de entrada.


Suma y resta binarias

- Los números binarios se suman de la misma manera que los números decimales.
- Decimal: cuando números suman más que 9 hay un acarreo (carry).
- Binario: cuando números suman más que 1 hay un acarreo (carry).
- Adición es la operación aritmética básica empleada en la sustracción, multiplicación y división.

Suma y resta binarias

- Suma o Adición binaria de un bit

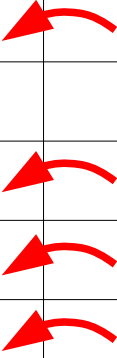
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	(carry) 1 0



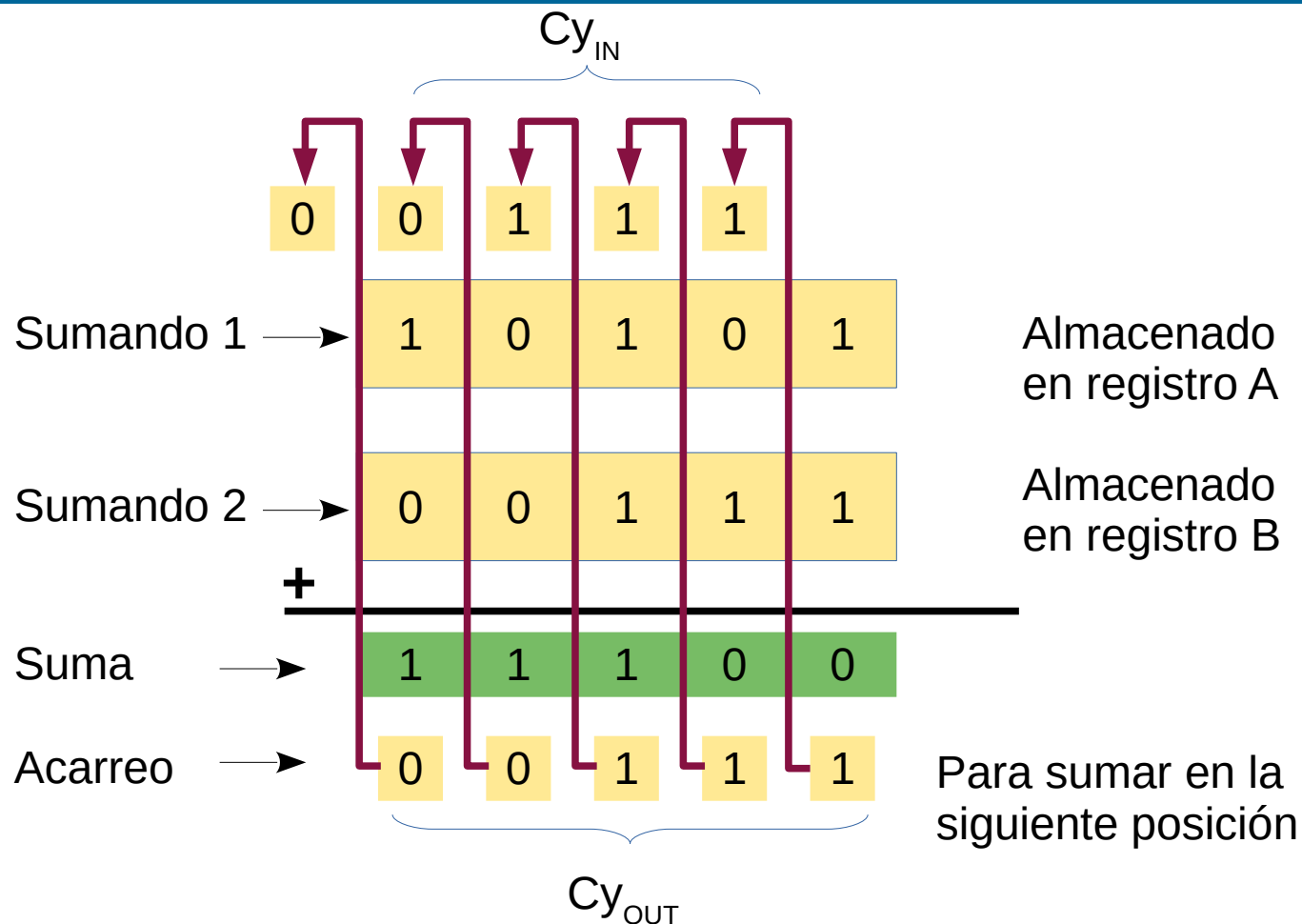
Suma binaria con acarreo

- Suma o Adición binaria de un bit con dos operandos y acarreo de entrada (Cy_{IN})
- El resultado puede producir un bit de acarreo a la salida (Cy_{OUT})

Cy_{IN}	A	B	Cy_{OUT}	$A+B+Cy_{IN}$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

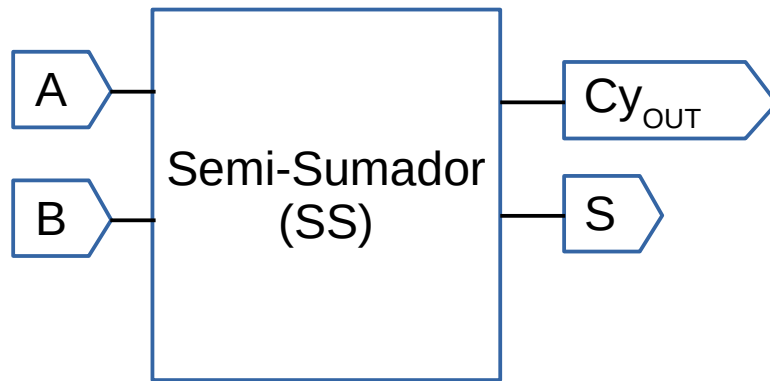


Sumador binario paralelo

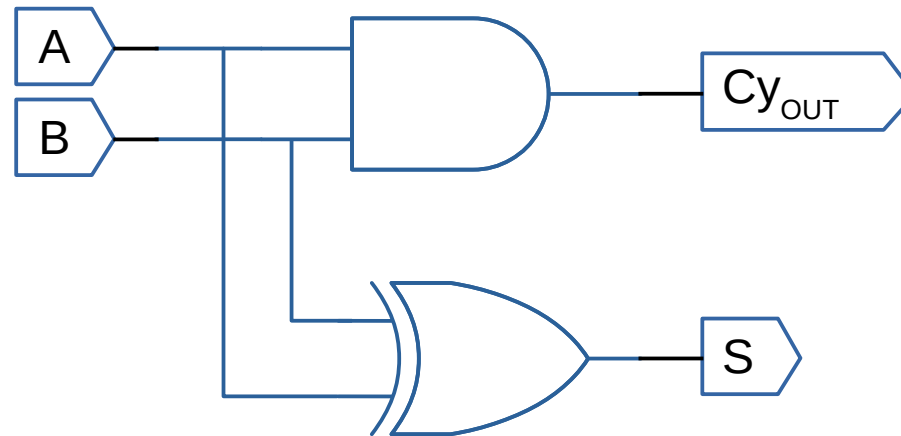


Acá se ve Como para cada columna tengo un Cy_{IN} y un Cy_{OUT}

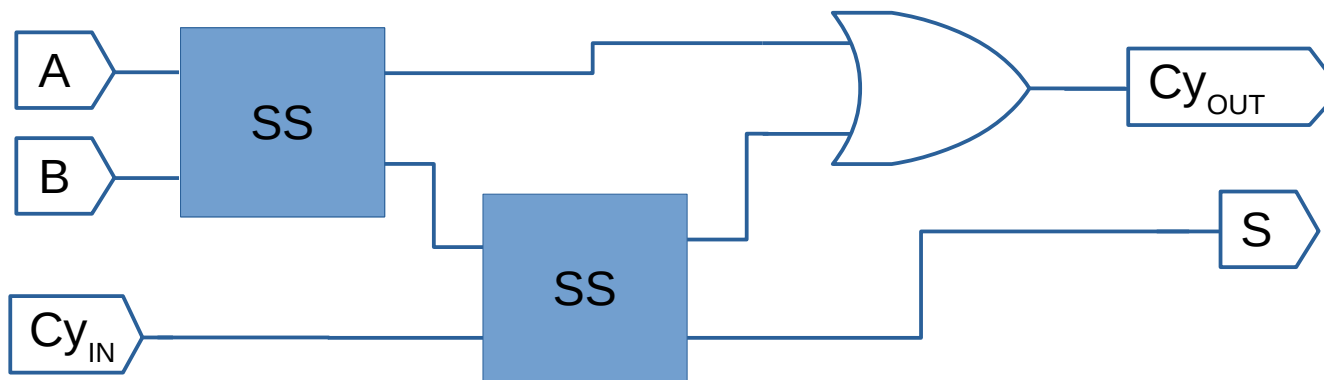
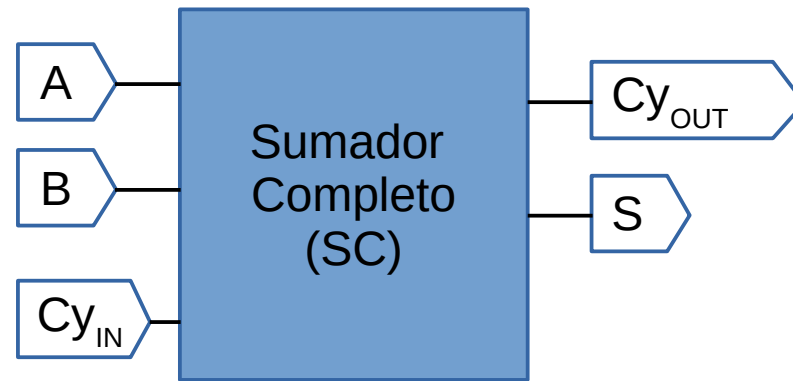
Semisumador (Half Adder)



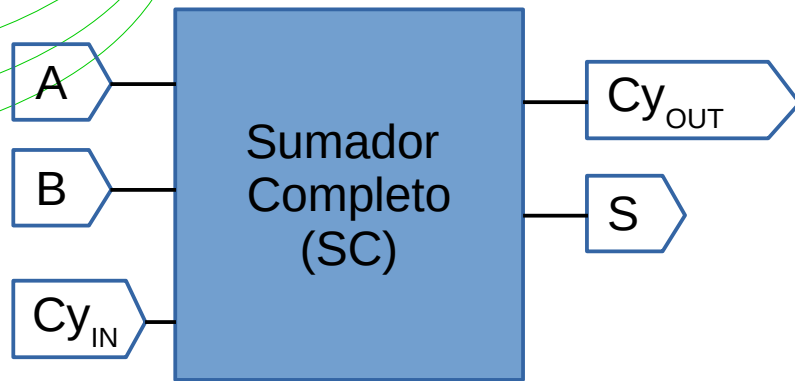
A	B	S	Cy_{OUT}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sumador Completo (Full Adder)



Sumador Completo con propagación anticipada de acarreo (Look Ahead Carry Full Adder) de un bit



	$\bar{B} \bar{C}y_{IN}$	$\bar{B} C y_{IN}$	$B C y_{IN}$	$B \bar{C}y_{IN}$
\bar{A}	0	1	0	1
A	1	0	1	0

S

A	B	Cy_{IN}	S	Cy_{OUT}
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

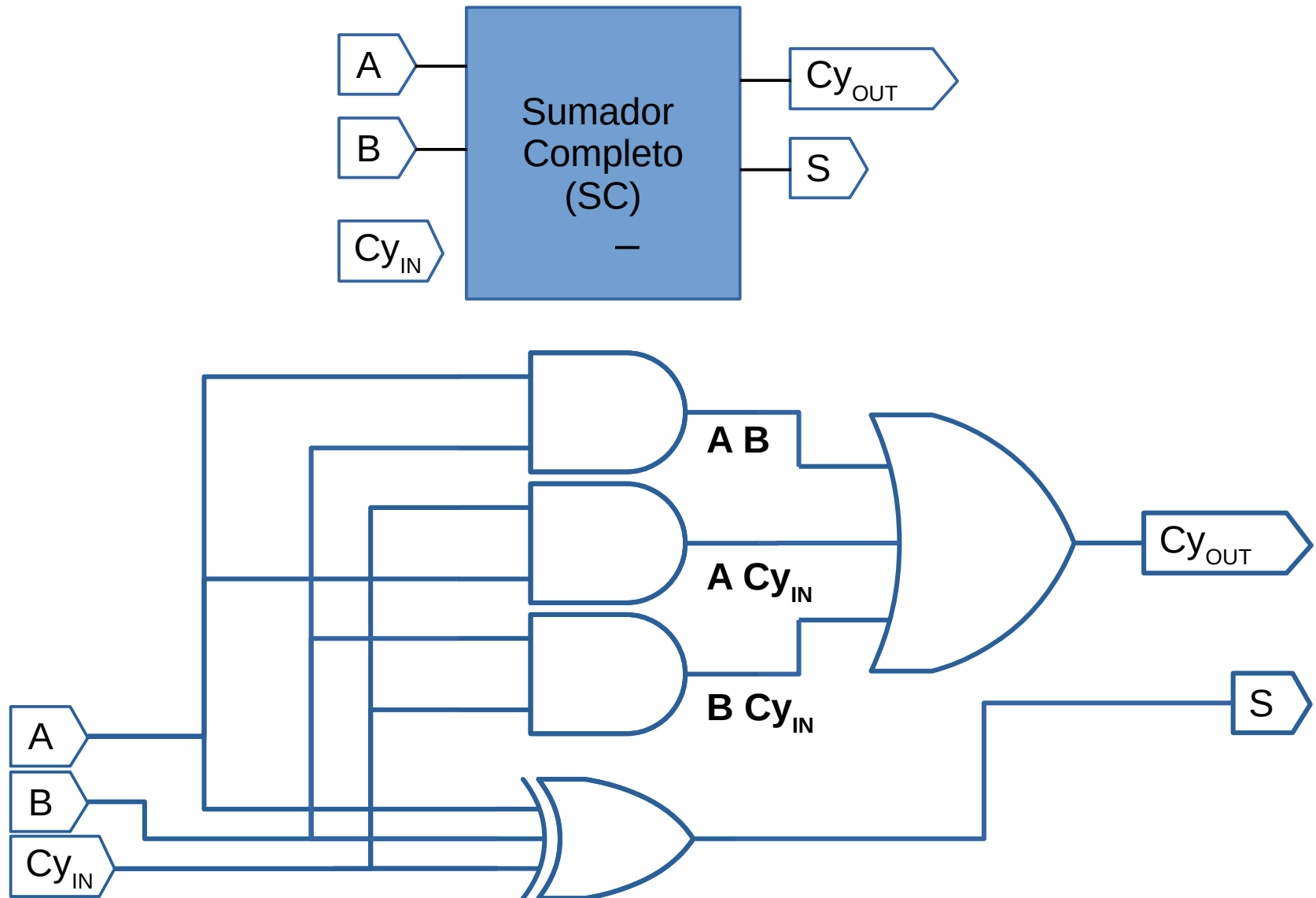
$$S = A \oplus B \oplus Cy$$

	$\bar{B} \bar{C}y_{IN}$	$\bar{B} C y_{IN}$	$B C y_{IN}$	$B \bar{C}y_{IN}$
\bar{A}	0	0	1	0
A	0	1	1	1

Cy_{OUT}

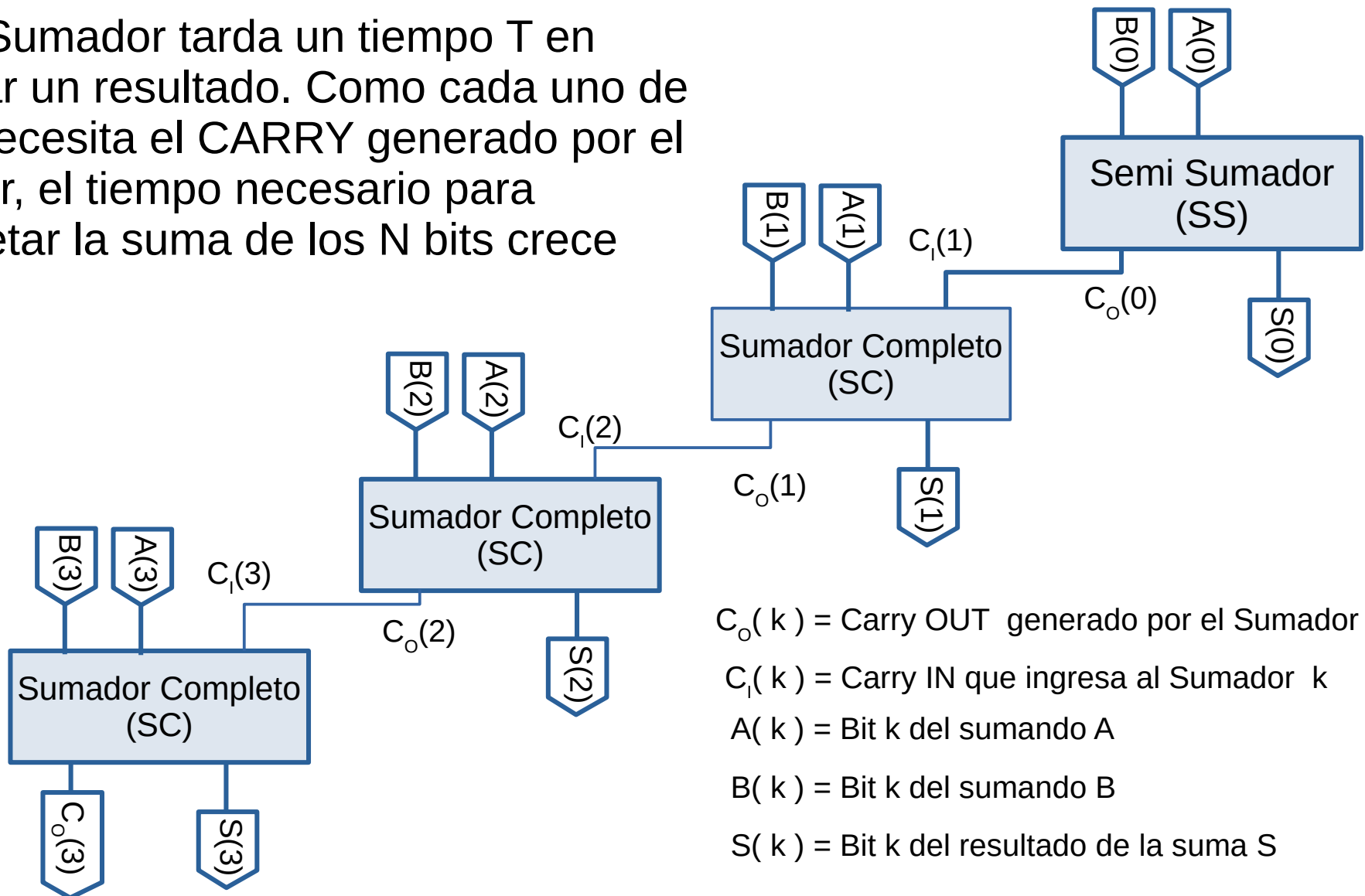
$$Cy_{OUT} = AB + ACy + BCy$$

Sumador Completo con propagación anticipada de acarreo (Look Ahead Carry Full Adder)



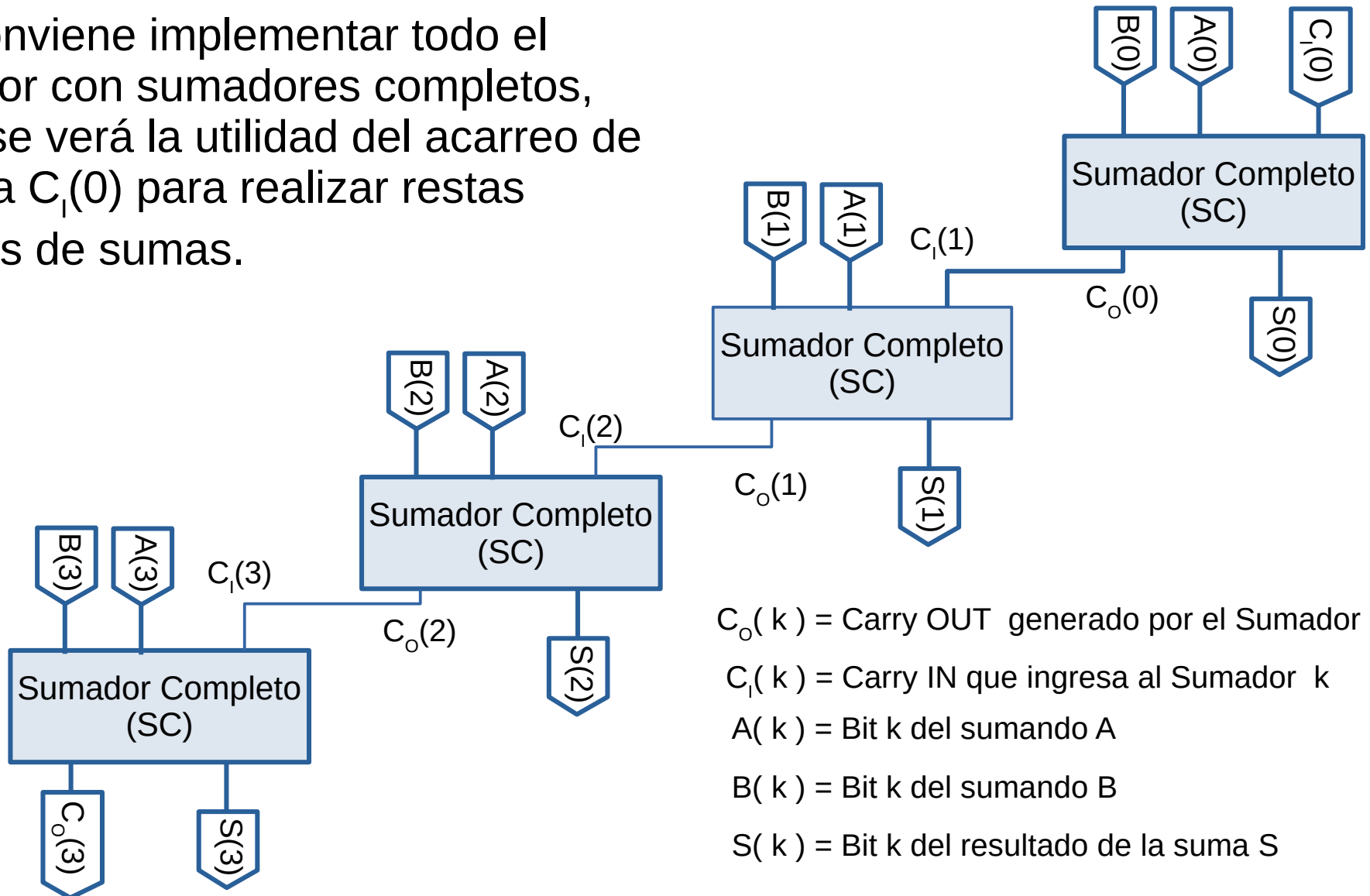
Sumador con acarreo serie (N bits)

Cada Sumador tarda un tiempo T en generar un resultado. Como cada uno de ellos necesita el CARRY generado por el anterior, el tiempo necesario para completar la suma de los N bits crece con N



Sumador con acarreo serie (N bits)

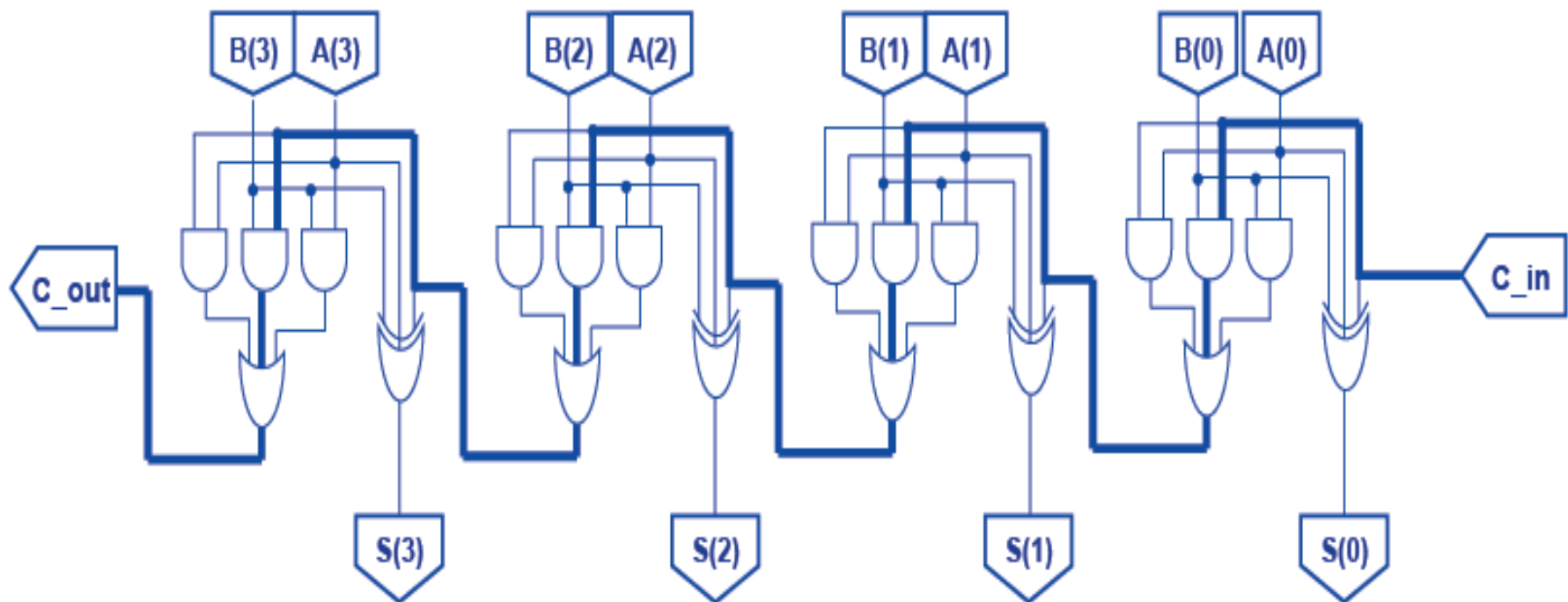
Nos conviene implementar todo el sumador con sumadores completos, luego se verá la utilidad del acarreo de entrada $C_i(0)$ para realizar restas además de sumas.



Sumador con acarreo serie (N bits)

En trazo grueso se marca el camino por donde se generan los acarreos para cada etapa.

Se observa claramente que la ultima etapa obtendra su resultado, luego de que esten listos los de las etapas anteriores.



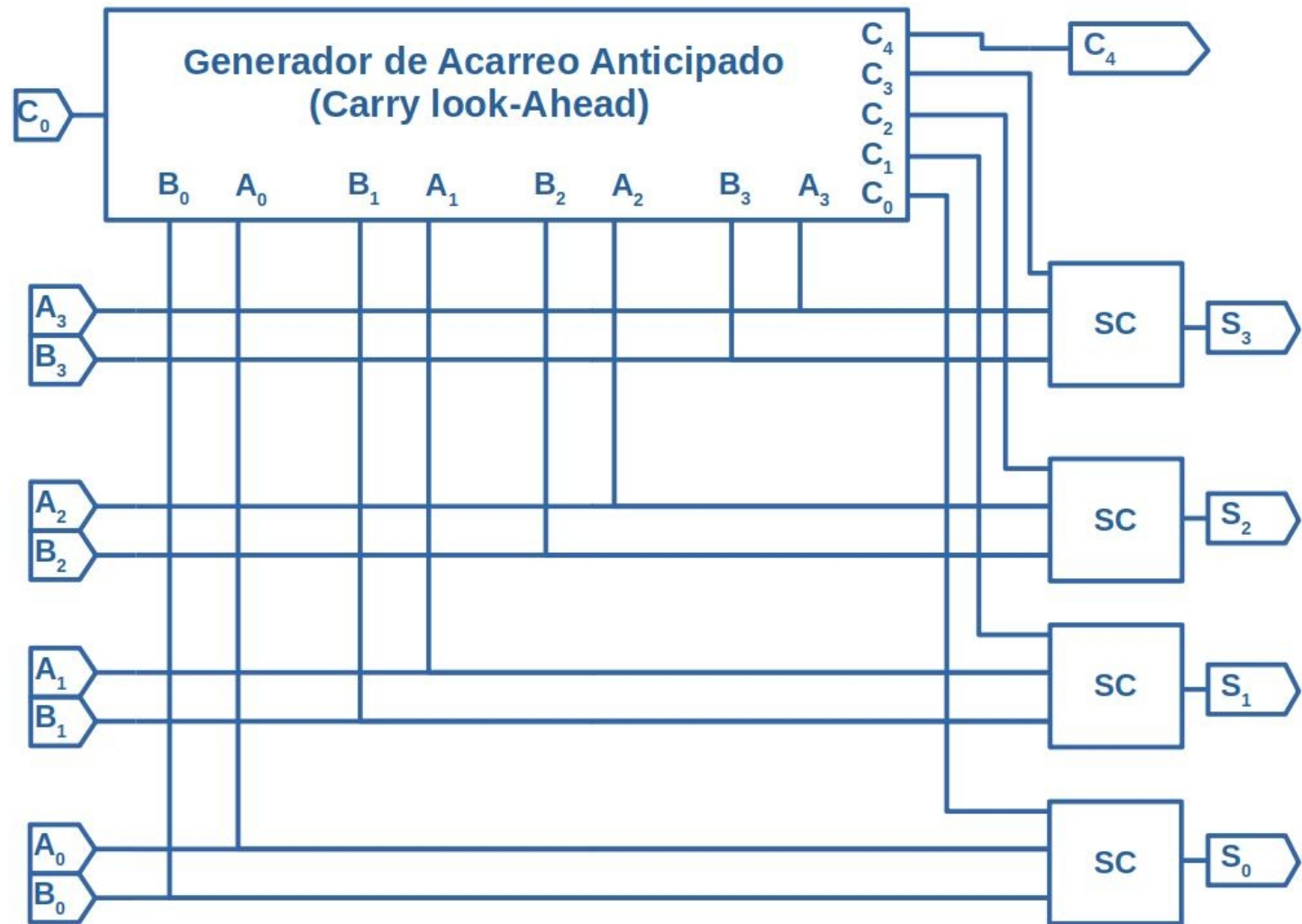
Propagación del Acarreo

- Los sumadores paralelos están limitados en velocidad por el tiempo de propagación del acarreo.

Mayor número de bits introducen mas retardos.

- El esquema **look-ahead carry** (acarreo anticipado) se utiliza en dispositivos de mayor velocidad para reducir el retardo.

Sumador con Acarreo Anticipado



Acarreo anticipado

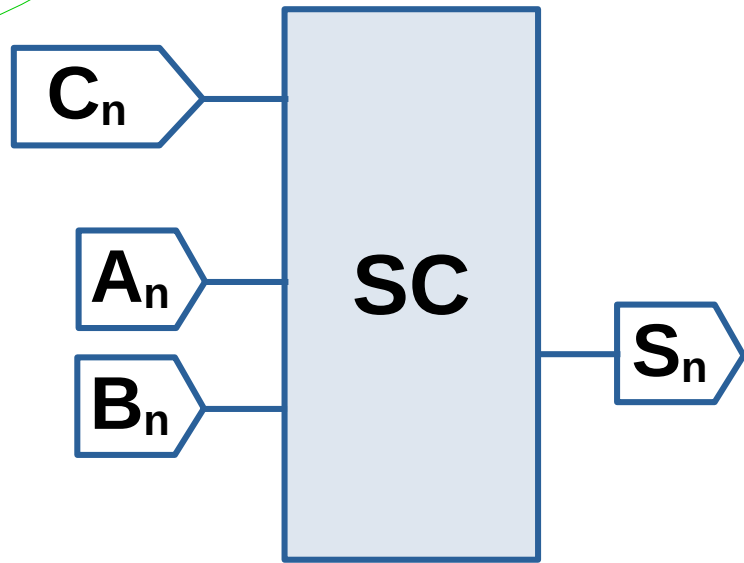
- Queremos hacer una función que genere el acarreo en una suma de N bits sin tener que esperar a que se propague por cada sumador completo.
- Definimos dos funciones que dependen solo de los bits de la etapa actual.
 - Generate: $\mathbf{G_n=1}$ si $\mathbf{A_n}$ y $\mathbf{B_n}$ generan acarreo
 - Propagate: $\mathbf{P_n=1}$ si cuando hay acarreo de entrada $\mathbf{C_n}$, los valores de $\mathbf{A_n}$ y $\mathbf{B_n}$ propagan el acarreo a la salida.
- Entonces el carry que entra a la etapa $\mathbf{n+1}$ es

$$\mathbf{C_{n+1} = G_n + P_n C_n}$$

- Desarrollando esta expresión hasta llegar a $\mathbf{C_0}$ (el carry de entrada al sumador) se obtiene la función lógica de cada carry de entrada de los FA del sumador *Carry-Look-Ahead*. Por ejemplo:

$$\mathbf{C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0}$$

Acarreo anticipado



A_n	B_n	G_n
0	0	0
0	1	0
1	0	0
1	1	1

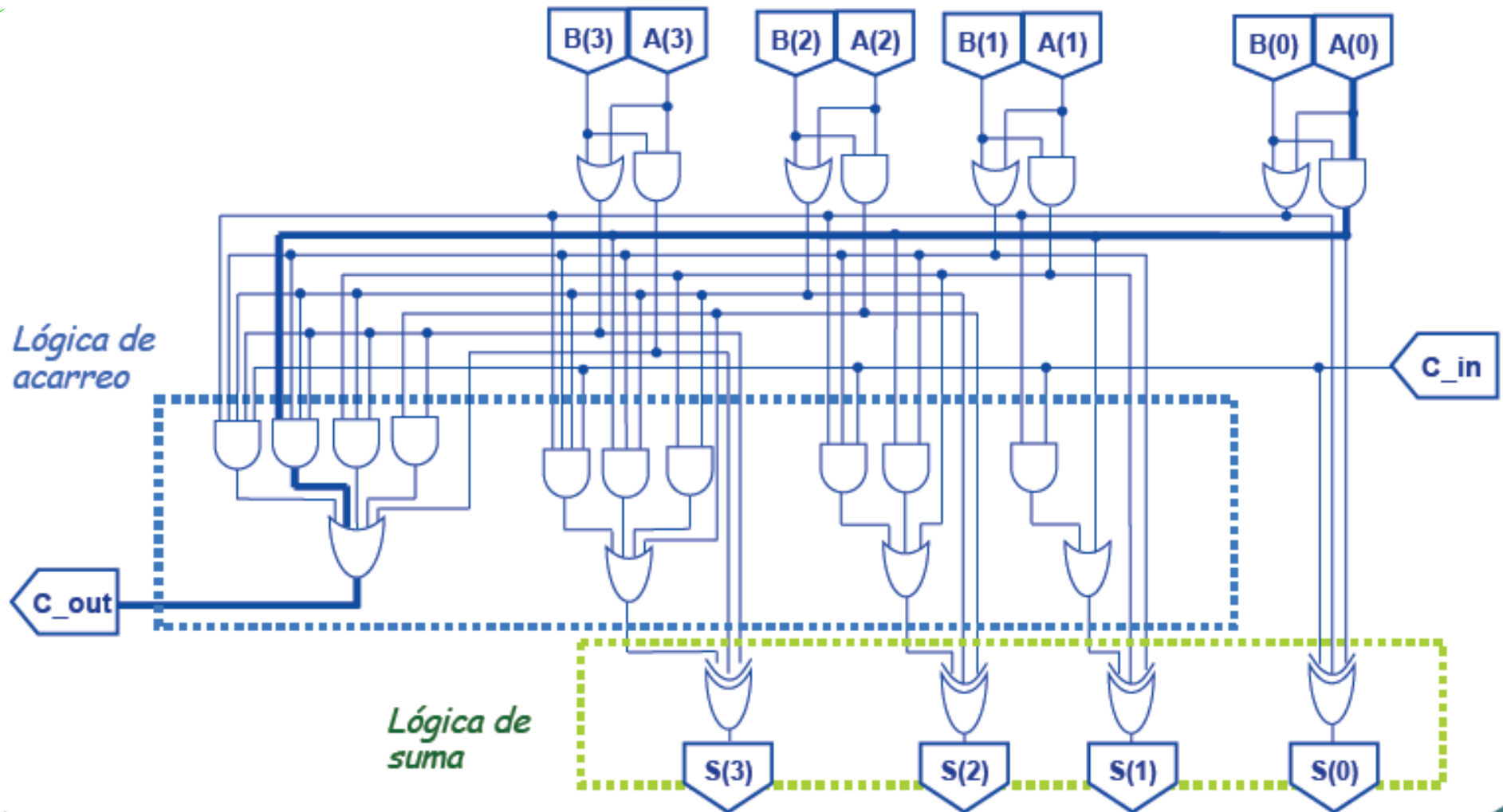
$$G_n = A_n B_n$$

A_n	B_n	P_n
0	0	0
0	1	1
1	0	1
1	1	1

$$P_n = A_n + B_n$$

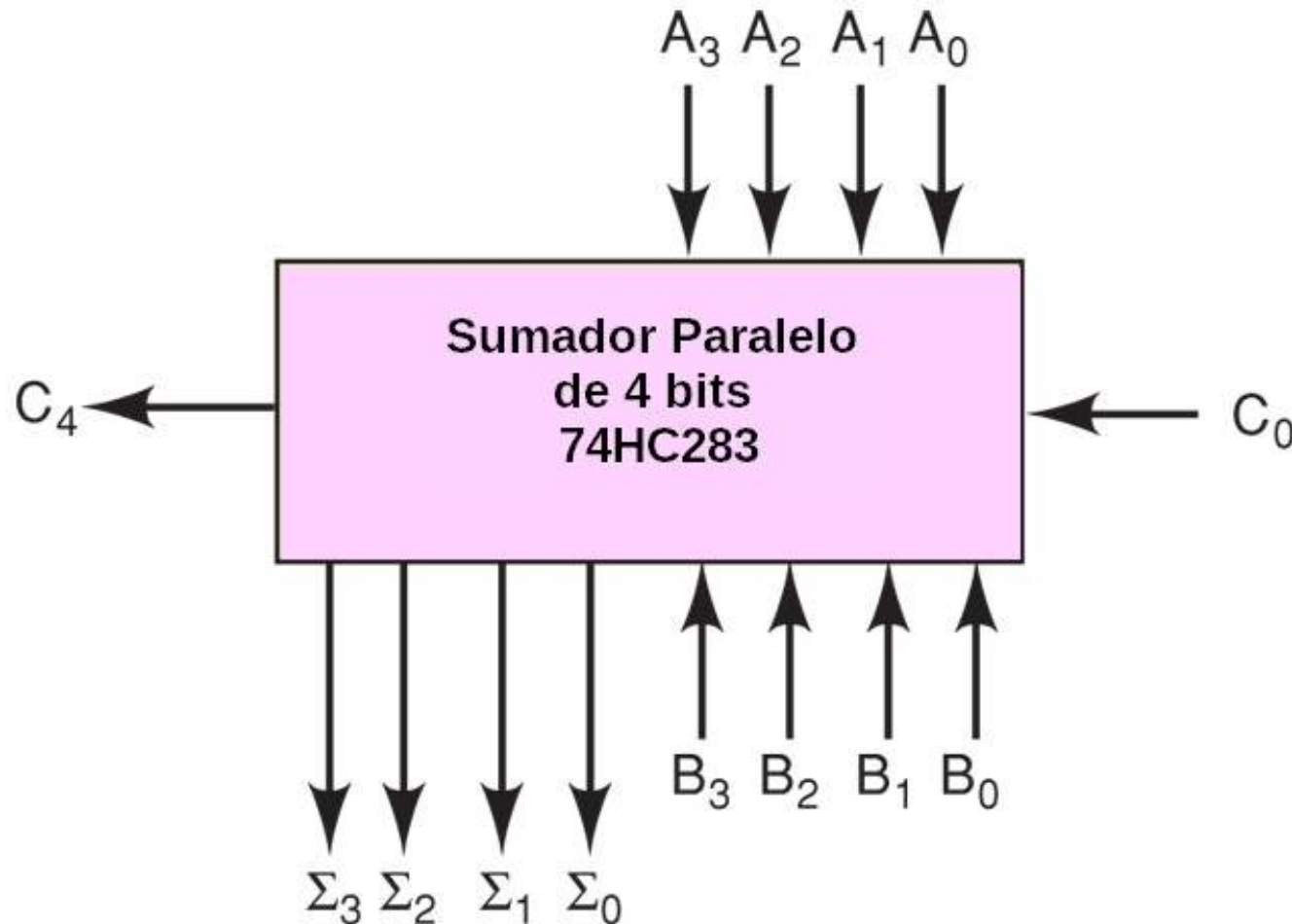
$$C_{n+1} = G_n + P_n C_n$$

Sumador con Acarreo Anticipado



Sumador Paralelo (legacy)

Las líneas A y B representan números de 4 bits a sumar. C_0 es el carry-in, C_4 es el carry-out, Σ es la suma.



Suma de números positivos y negativos

- Si se quiere sumar números positivos y negativos puede representarse el número negativo en complemento a 2 y luego se realiza la suma normal.
- De la misma manera puede realizarse la resta.

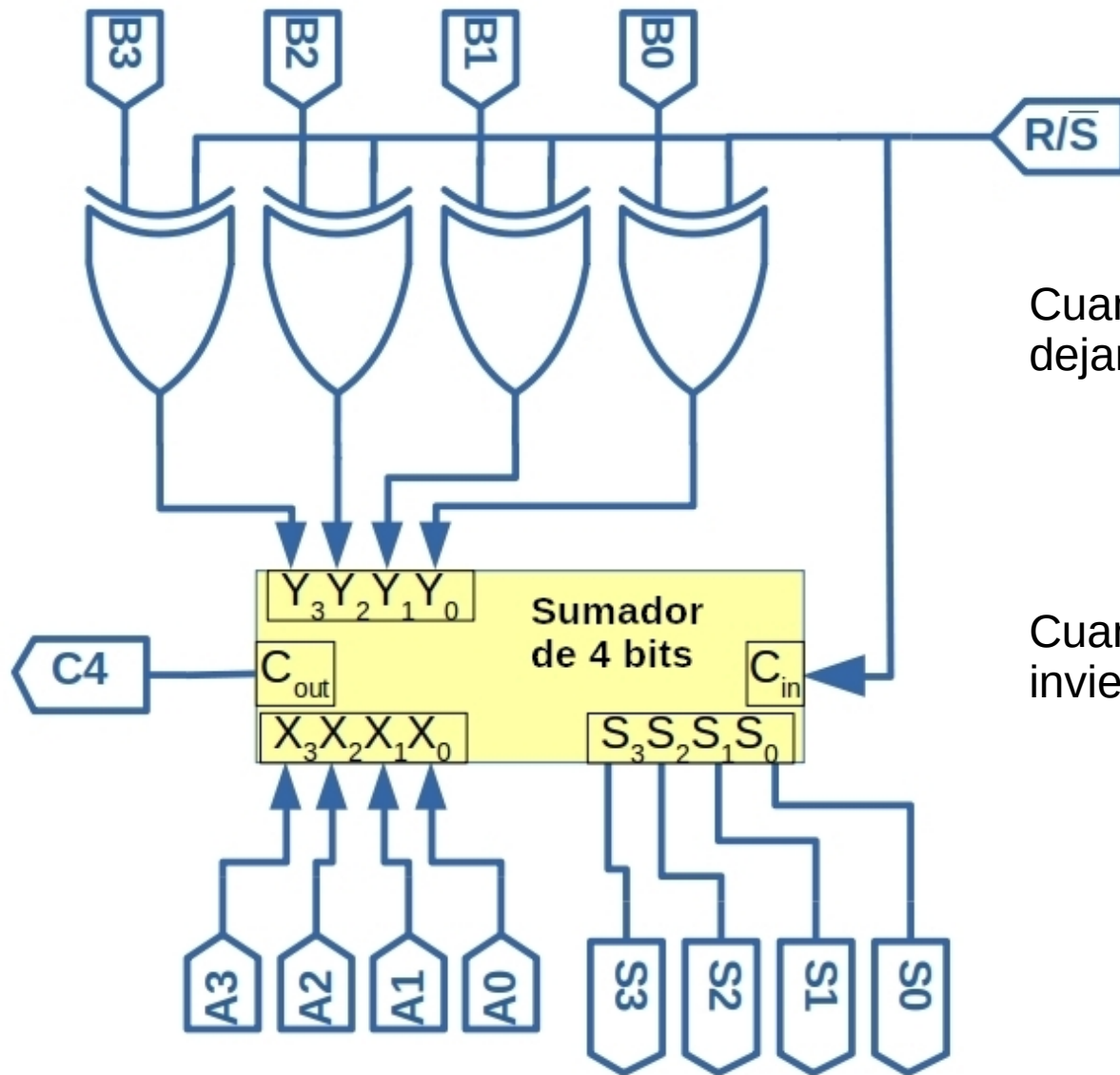
$$A + (-B) = A - B$$

$$(-B) = \text{Ca2}(B) = \overline{B} + 1$$

$$A - B = A + \overline{B} + 1$$

- Podemos aprovechar la última expresión

Suma de números positivos y negativos



Cuando $R/\bar{S} = 0$, las compuertas XOR dejan pasar los datos B y $C_{in} = 0$

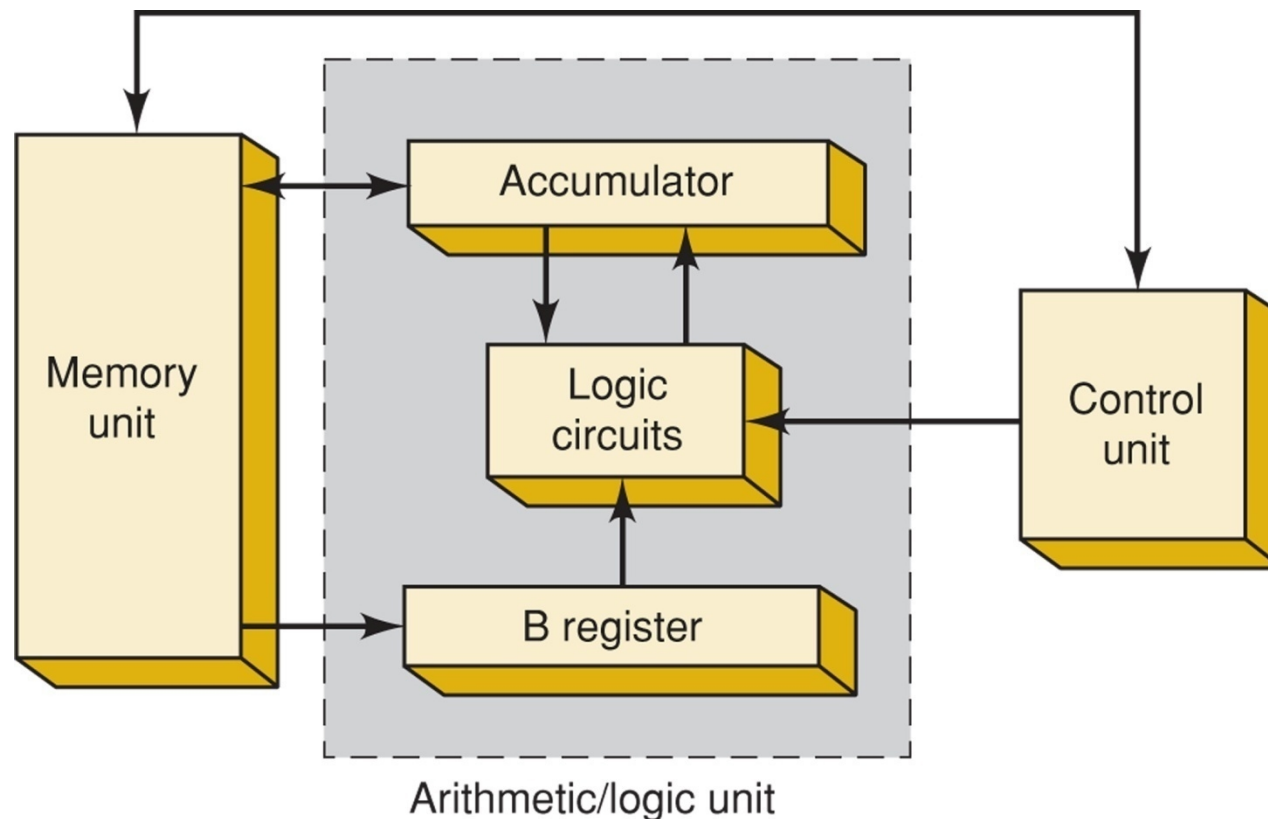
La salida $S = A + B$

Cuando $R/\bar{S} = 1$, las compuertas XOR invierten los datos B y $C_{in} = 1$

La salida $S = A + \bar{B} + 1 = A - B$

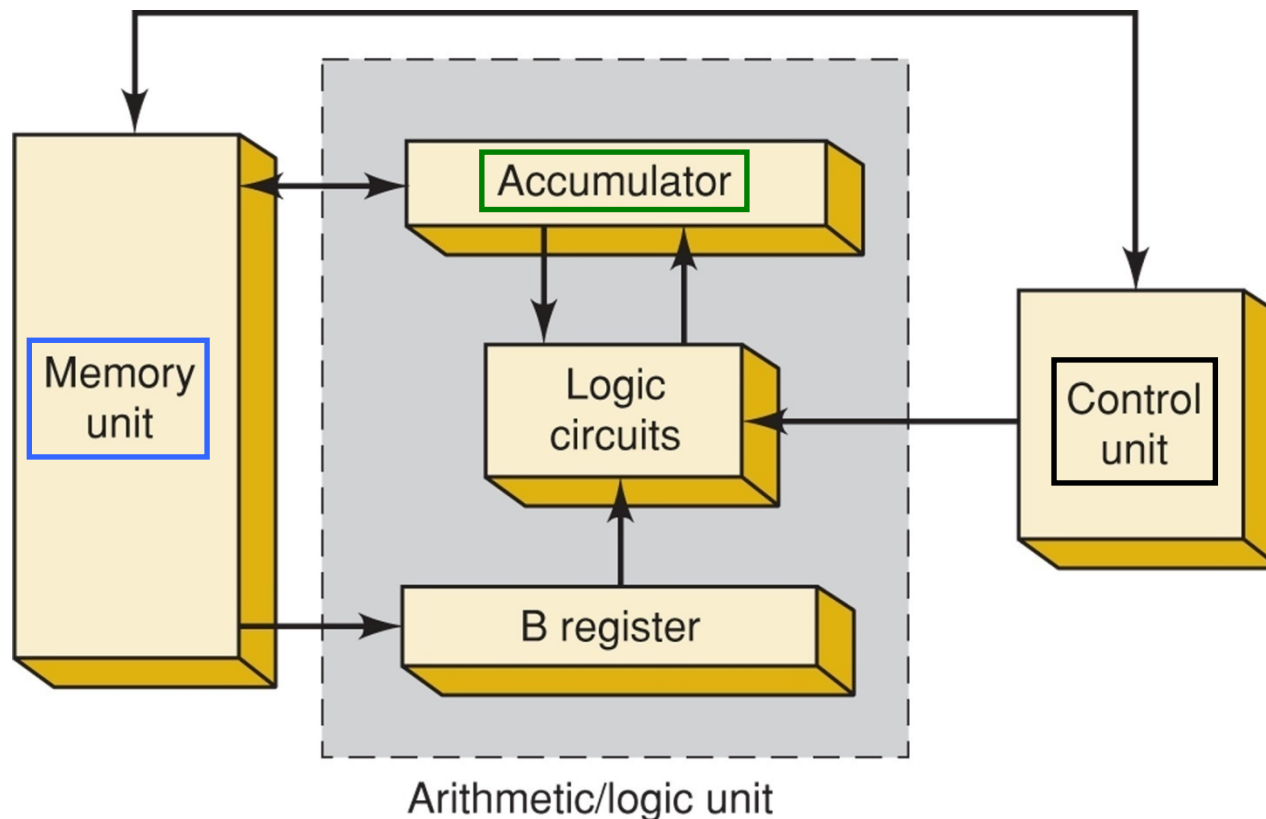
Unidad Aritmético – Logica (ALU)

- Las ALUs pueden realizar distintas funciones lógicas y aritméticas en función de un código binario que se aplica a las entradas de función



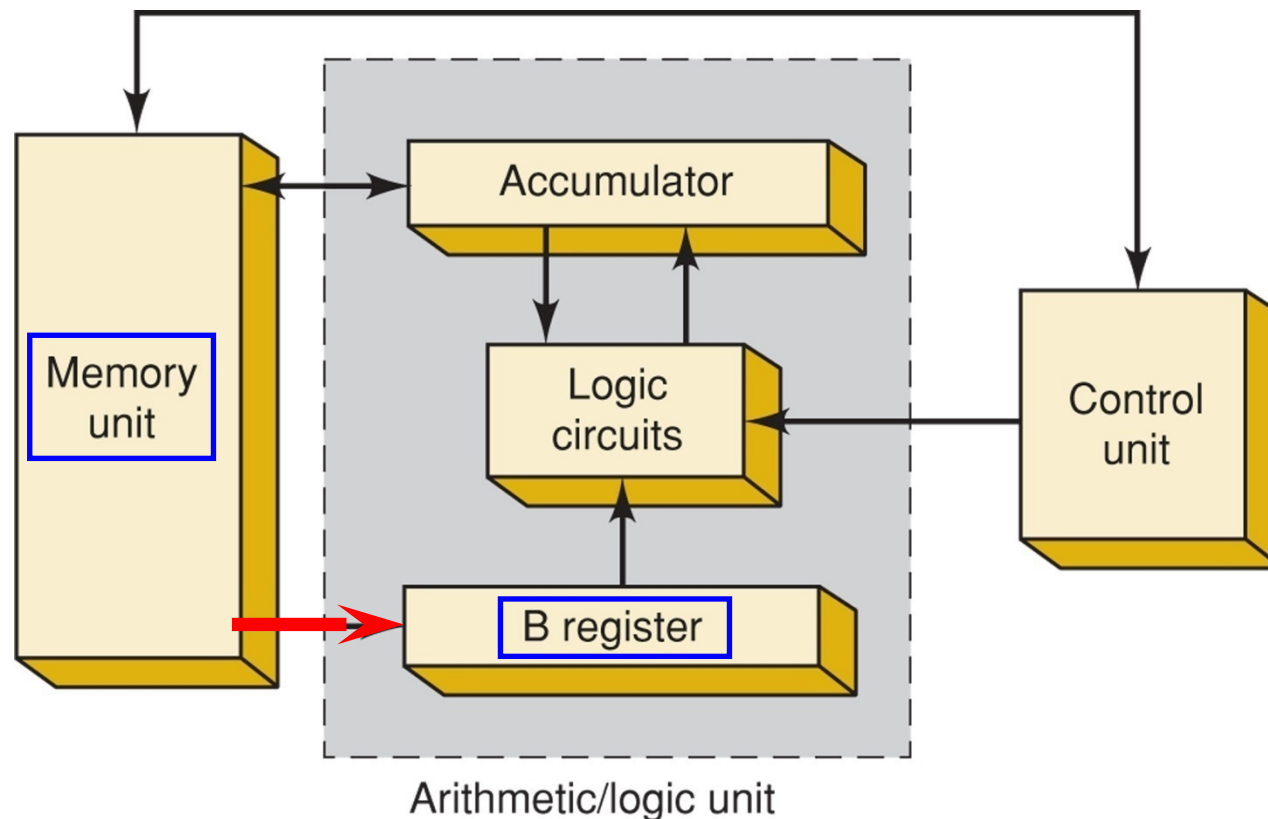
Unidad Aritmético – Logica (ALU)

- La unidad de control es instruída para sumar un número ubicado en una posición de memoria con uno almacenado en el acumulador.



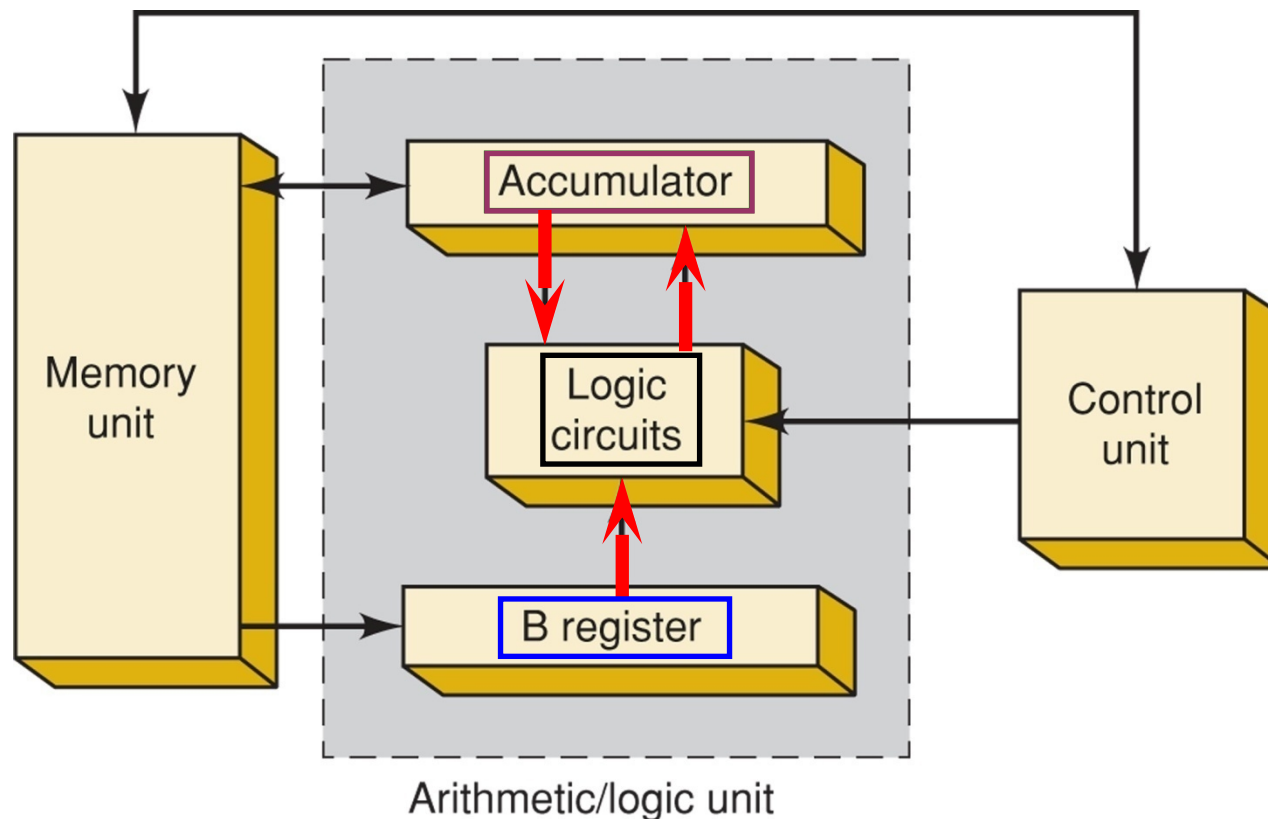
Unidad Aritmético – Logica (ALU)

- El número es transferido desde la memoria hacia el registro B.



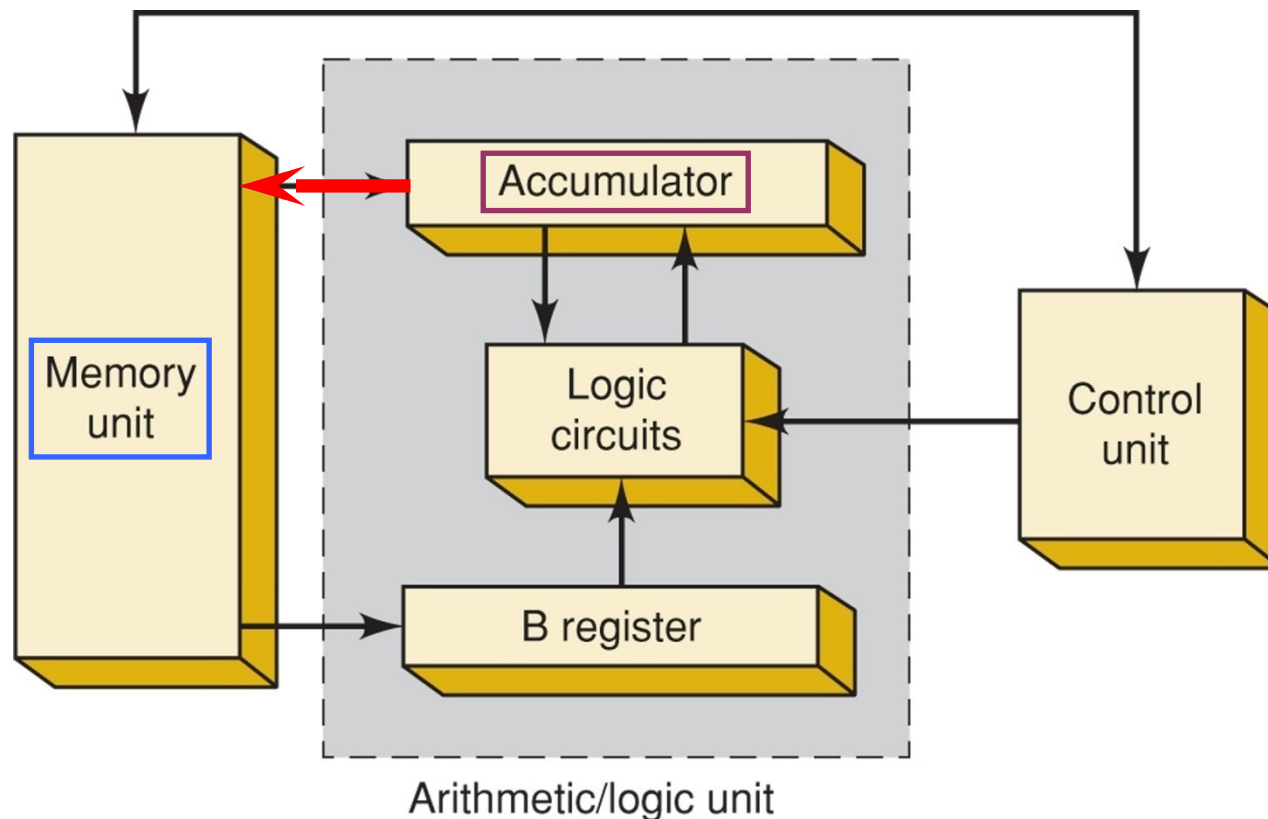
Unidad Aritmético – Logica (ALU)

- El número en el registro B y en el acumulador son sumados en el circuito lógico y la suma es enviada al acumulador para su almacenamiento.

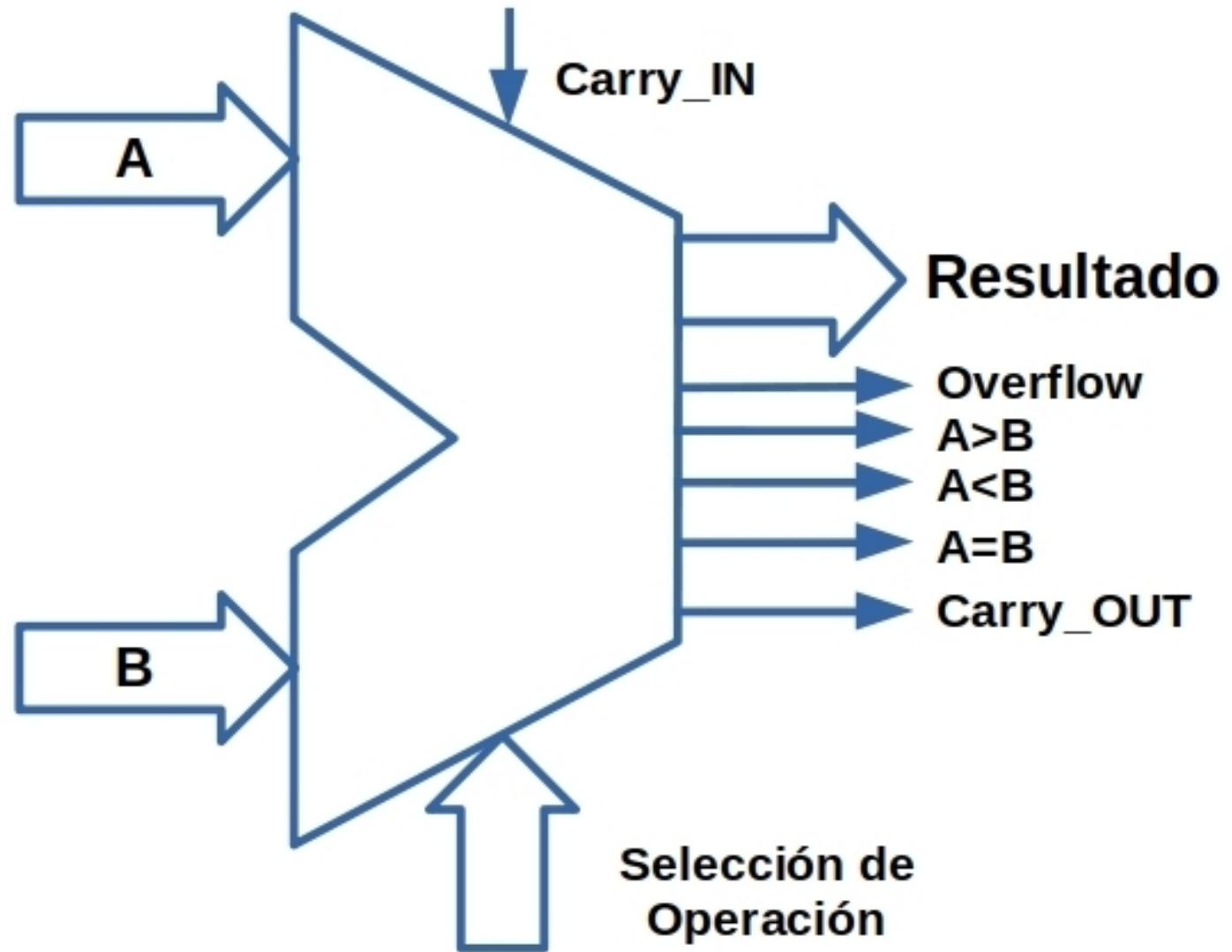


Unidad Aritmético – Logica (ALU)

- El número se mantiene en el acumulador para futuras operaciones, ó puede ser transferido a memoria para su almacenamiento



ALU



Funciones típicas de una ALU

➤ Suma

- $A + B$

➤ Resta

- $A - B$

➤ Complemento a 2

- $--B$

➤ Comparación

- $A > B$

- $A < B$

- $A = B$

➤ Desplazamiento a Izquierda

- $SHL(A)$

➤ Desplazamiento a Derecha

- $SHR(A)$

Operaciones Lógicas (bit a bit)

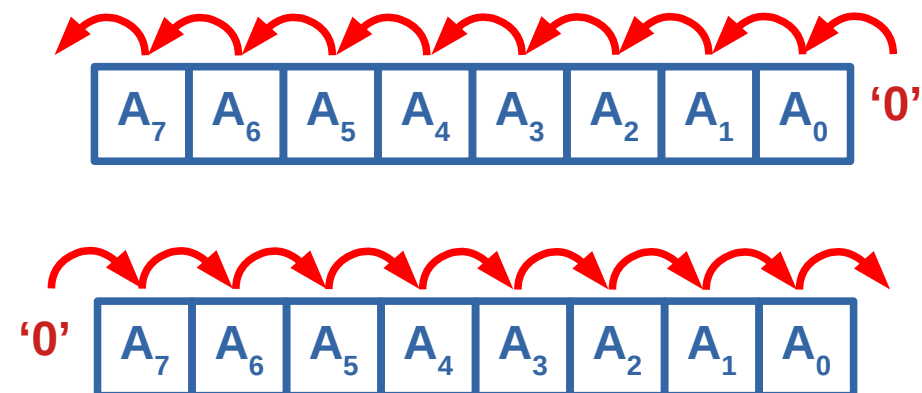
- $A \text{ AND } B$

- $A \text{ OR } B$

- $A \text{ XOR } B$

- $A \text{ XNOR } B$

- $\text{NOT } A$



Multiplicación



Multiplicador decimal y binario

86d	→	1010110b
15d	→	0001111b
<hr/>		
1290d	→	10100001010b

Decimal	
86d	
15d	
<hr/>	
30	5x6
40	5x8 desplazado a izqda 1 p.
6	6x1 desplazado a izqda 1 p.
8	8x1 desplazado a izqda 2 p.
<hr/>	
1290	

- Operandos: **n bits**
- Resultado: **2*n bits**

Multiplicación Binaria

Binario

$$A * B = A * (b_{n-1} * 2^{n-1} + b_{n-2} * 2^{n-2} + \dots + b_1 * 2^1 + b_0 * 2^0)$$

ii '1s' o '0s' !!

La multiplicación binaria de dos números A (m bits) y B (n bits) consiste en una suma de tantos elementos como bits tenga B (n). Cada elemento i es el número A desplazado a la izquierda i veces si el peso correspondiente de B vale '1'. En caso contrario el elemento i es '0'.

Multiplicación Binaria

$$\begin{array}{r}
 \begin{array}{c}
 A_3 \quad A_2 \quad A_1 \quad A_0 \\
 \times \quad B_3 \quad B_2 \quad B_1 \quad B_0 \\
 \hline
 A_3 B_0 \quad A_2 B_0 \quad A_1 B_0 \quad A_0 B_0 \\
 \\
 + \quad A_3 B_1 \quad A_2 B_1 \quad A_1 B_1 \quad A_0 B_1 \quad 0 \\
 \\
 \quad A_3 B_2 \quad A_2 B_2 \quad A_1 B_2 \quad A_0 B_2 \quad 0 \quad 0 \\
 \\
 \quad A_3 B_3 \quad A_2 B_3 \quad A_1 B_3 \quad A_0 B_3 \quad 0 \quad 0 \quad 0 \\
 \hline
 R_7 \quad R_6 \quad R_5 \quad R_4 \quad R_3 \quad R_2 \quad R_1 \quad R_0
 \end{array}
 \end{array}$$

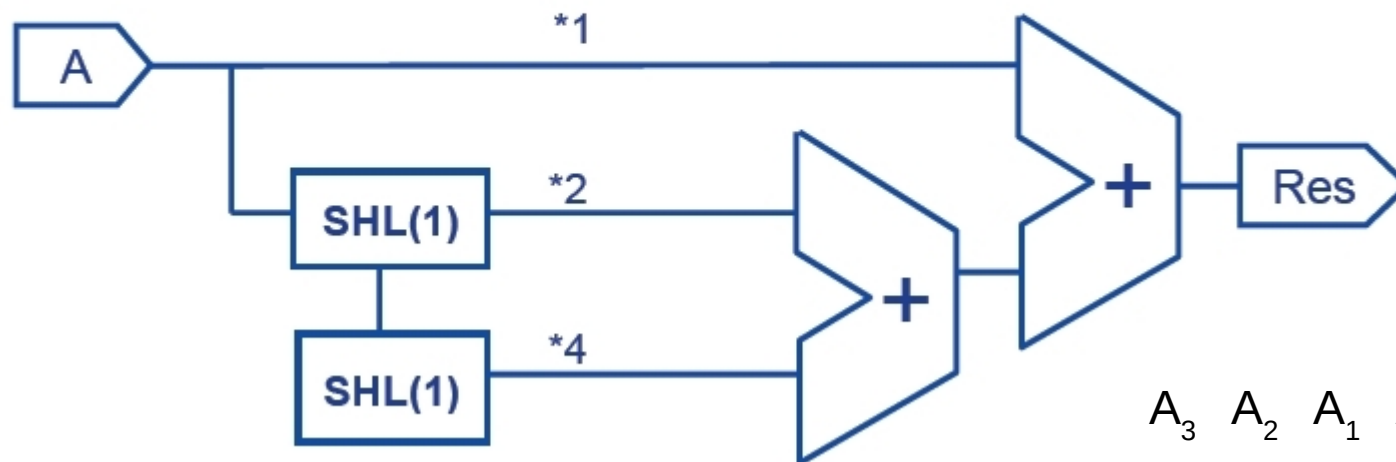
B_n son 1 o 0,

- Cuando $B_n = 1$, se copia el número A, desplazado a la izquierda n posiciones y completando a la derecha con 0.
- Cuando $B_n = 0$, se completa la fila con 0.

Multiplicación Binaria



$$A * 7 = A * (4 + 2 + 1)$$

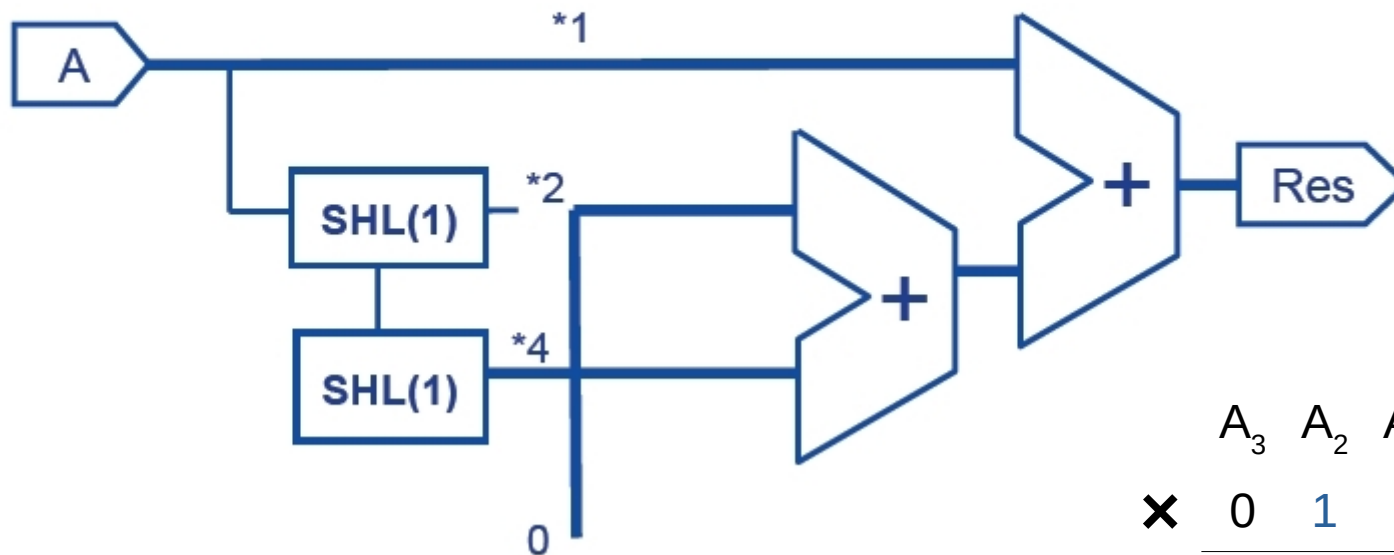


				A_3	A_2	A_1	A_0	(A)		
			\times	0	1	1	1	$\times (7)$		
				A_3	A_2	A_1	A_0	(A)		
		$+$		A_3	A_2	A_1	A_0	0	A SHL(1)	
				A_3	A_2	A_1	A_0	0	0	A SHL(2)
R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	Resultado		

Multiplicación Binaria



$$A * 5 = A * (4 + 0 + 1)$$

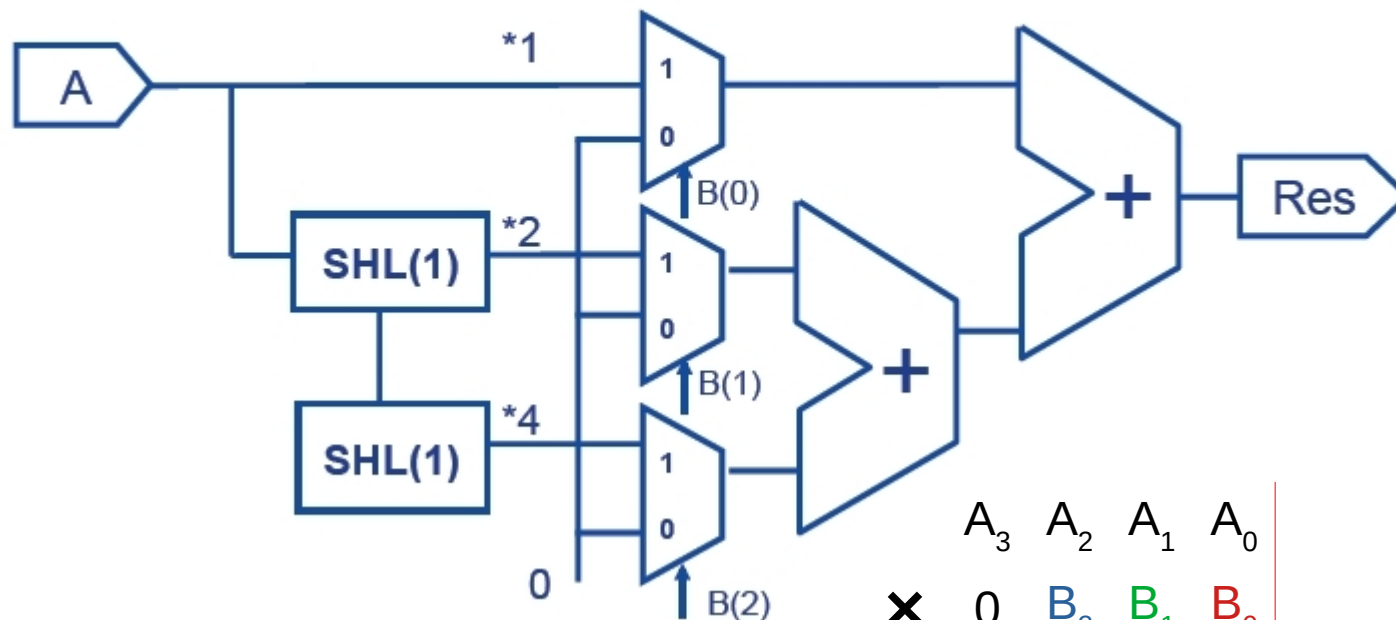


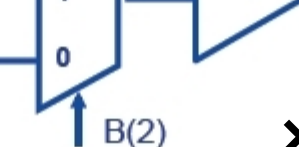
				A_3	A_2	A_1	A_0	(A)
			\times	0	1	0	1	$\times (7)$
				A_3	A_2	A_1	A_0	(A)
	$+$	0	0	0	0	0	0	A SHL(1)
		A_3	A_2	A_1	A_0	0	0	A SHL(2)
R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	Resultado

Multiplicación Binaria



$$A * B = A * (4*B(2)+2*B(1)+1*B(0))$$



	A_3	A_2	A_1	A_0	(A)			
\times	0	B_2	B_1	B_0	$\times (7)$			
	A_3	A_2	A_1	A_0	A si $B_0=1$, 0 si $B_0=0$			
$+$	A_3	A_2	A_1	A_0	A SHL(1) si $B_1=1$, 0 si $B_1=0$			
	A_3	A_2	A_1	A_0	A SHL(2) si $B_2=1$, 0 si $B_2=0$			
R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	Resultado

Multiplicador Binario de 4 bits

