

# FACULTAD DE INGENIERIA Y CIENCIAS EXACTAS

MATERIA: ARQUITECTURA DE COMPUTADORES (3.4.072)

Circuitos Combinacionales UAL de 1 Bit



Ciudad de Buenos Aires,

# Arquitectura de Computadoras

(3.4.072)

## ● Álgebra de Boole:

- Aproximadamente en el año 1850 George Boole, desarrolló un sistema algebraico para **formular proposiciones** con **símbolos**.
- Consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios “**1**” y “**0**” y a tres operadores:
  - **AND (y)**
  - **OR (o)**
  - **NOT (no)**
- Una variable Booleana representa un ***bit*** que quiere decir:

***Binary digIT***

# Arquitectura de Computadoras

## (3.4.072)

### ● Operador AND:

$x$	$y$	$x \ y$
0	0	0
0	1	0
1	0	0
1	1	1

Si una de las entradas es 0, entonces la salida es 0

# Arquitectura de Computadoras

( 3.4.072)

## ● Operador OR:

$x$	$y$	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Si una de las entradas es 1, entonces la salida es 1

# Arquitectura de Computadoras

( 3.4.072)

## ● Operador NOT:

$x$	$x$
0	1
1	0

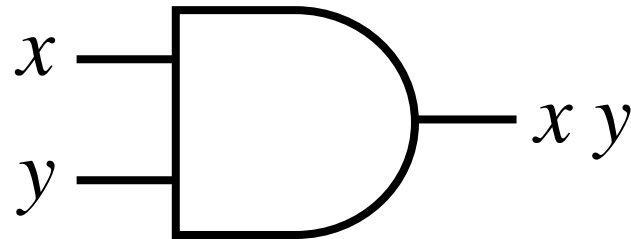
La salida es la negación de la entrada

# Arquitectura de Computadoras

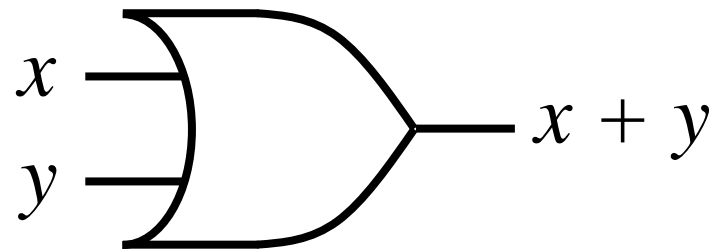
(3.4.072)

- **Símbolos de los operadores:**

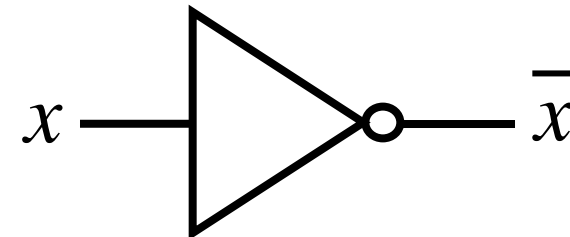
- **Operador AND:**



- **Operador OR:**



- **Operador NOT:**



# Arquitectura de Computadoras

## (3.4.072)

### ● Ejercicio:

- Realizar la tabla de verdad de la siguiente ecuación:

$$w = x \bar{y} + y z$$

$x$	$y$	$z$	$x\bar{y}$	$yz$	$w$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	0	1	1

# Arquitectura de Computadoras

## (3.4.072)

- El álgebra de Boole tiene propiedades, postulados, axiomas, teoremas y leyes.

Por Ej.: Las Leyes de De Morgan

- $\overline{(x + y)} = \bar{x} \bar{y}$
- $\overline{(x y)} = \bar{x} + \bar{y}$

que nos permiten cambiar operadores AND por operadores OR



# Arquitectura de Computadoras

(3.4.072)

## ● Circuitos Combinacionales:

- Un circuito combinatorial es aquel cuya salida depende exclusivamente de la combinación de las señales en sus entradas.

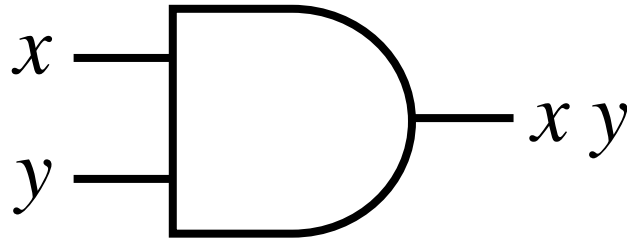
Es decir:

- *No depende del valor de su salida*
  - *No depende del tiempo*
- Desde el punto de vista de las computadoras cualquier dispositivo que realice la función de un operador booleano se denomina **compuerta lógica**.

# Arquitectura de Computadoras

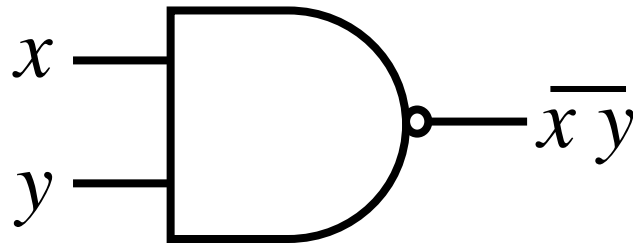
## (3.4.072)

### ● Compuerta AND:



$x$	$y$	$xy$
0	0	0
0	1	0
1	0	0
1	1	1

### ● Compuerta NAND:

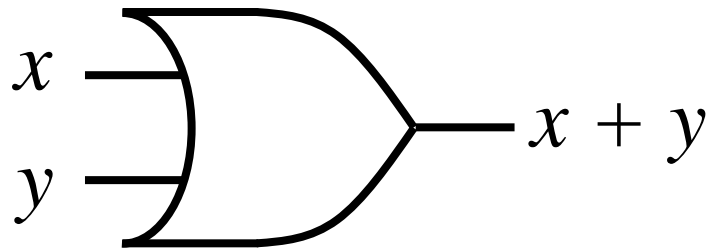


$x$	$y$	$\overline{xy}$
0	0	1
0	1	1
1	0	1
1	1	0

# Arquitectura de Computadoras

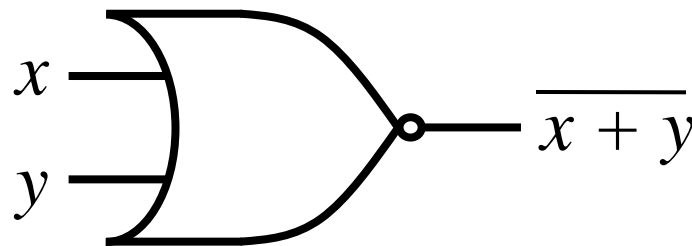
## (3.4.072)

### ● Compuerta OR:



$x$	$y$	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

### ● Compuerta NOR:

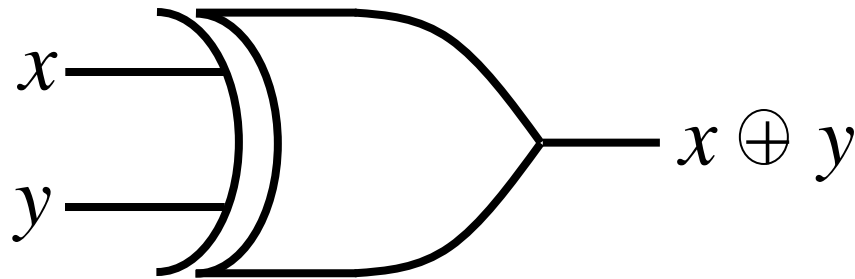


$x$	$y$	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

# Arquitectura de Computadoras

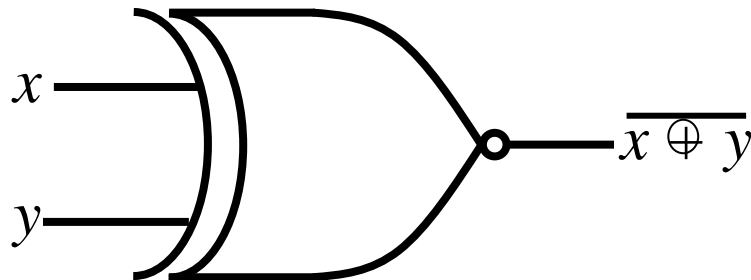
## (3.4.072)

### ● Compuerta XOR (*OR Exclusivo*):



$x$	$y$	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

### ● Compuerta XNOR (*XOR Exclusivo*):



$x$	$y$	
0	0	1
0	1	0
1	0	0
1	1	1

# Arquitectura de Computadoras

## (3.4.072)

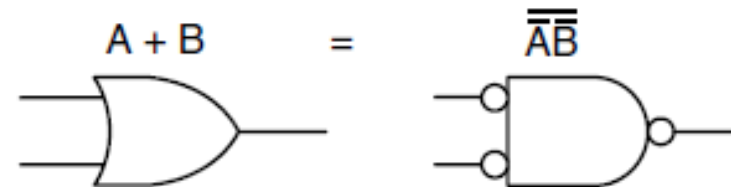
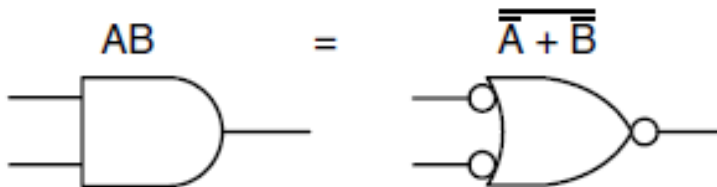
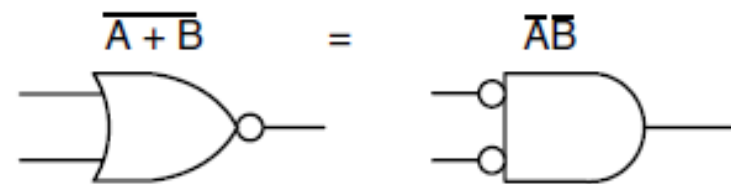
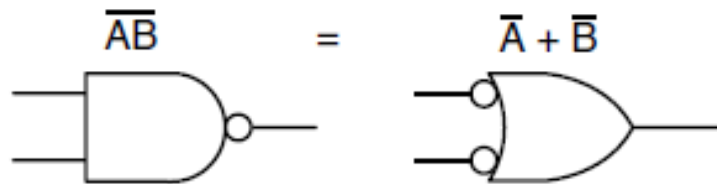
### Postulados del Algebra de Boole

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

# Arquitectura de Computadoras

## (3.4.072)

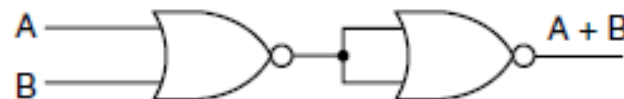
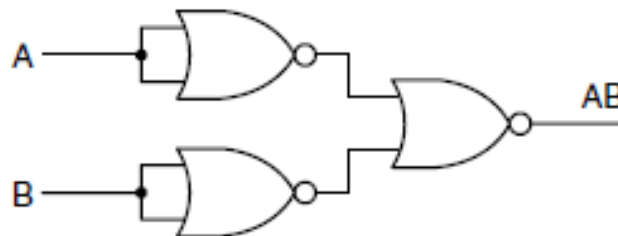
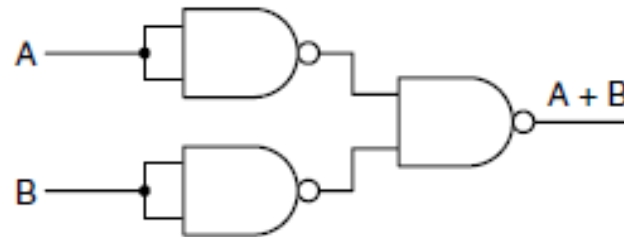
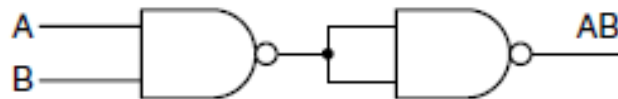
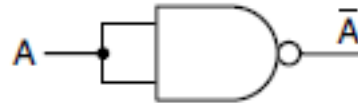
Equivalencias de las funciones



# Arquitectura de Computadoras

## (3.4.072)

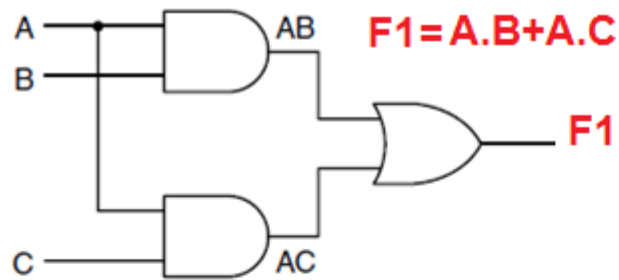
Equivalencias de las funciones



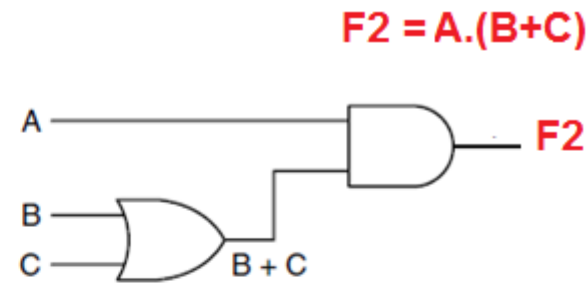
# Arquitectura de Computadoras

## (3.4.042 / 3.4.072)

- Construir la tabla de verdad de un circuito
- Dada la **ecuación lógica** construir el circuito
- Determinar si los circuitos son equivalentes



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1



A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1



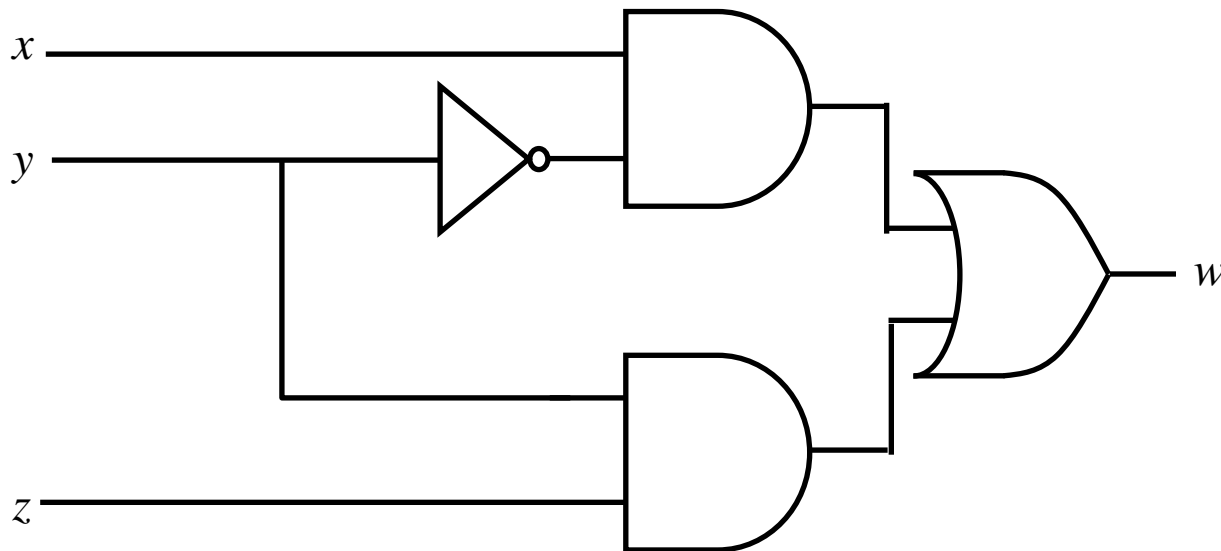
# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Ejercicio:

- Diseñe un circuito combinacional que responda a la siguiente ecuación:

$$w = x \bar{y} + y z$$

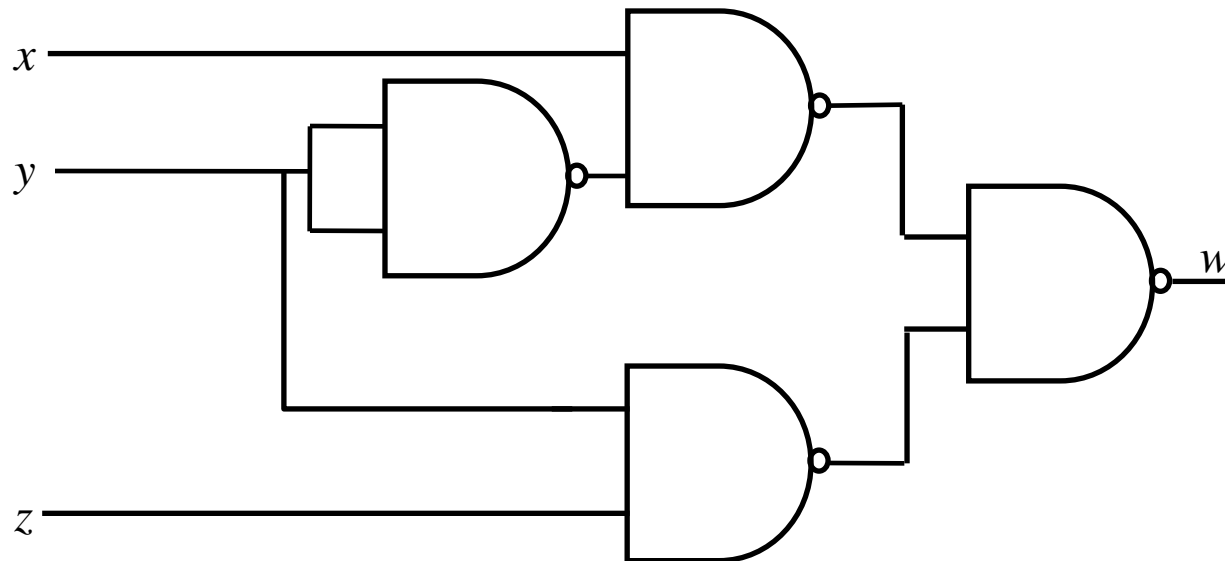


# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Ejercicio:

- Construya la tabla de verdad a la que responde el siguiente circuito lógico:



# Arquitectura de Computadoras

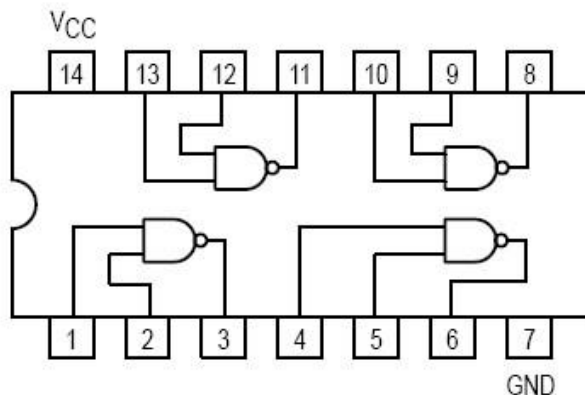
(3.4.072)



**MOTOROLA**

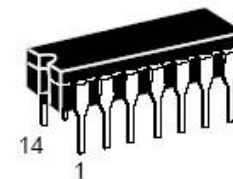
## QUAD 2-INPUT NAND GATE

- ESD > 3500 Volts



**SN54/74LS00**

**QUAD 2-INPUT NAND GATE**  
**LOW POWER SCHOTTKY**



**J SUFFIX**  
**CERAMIC**  
**CASE 632-08**

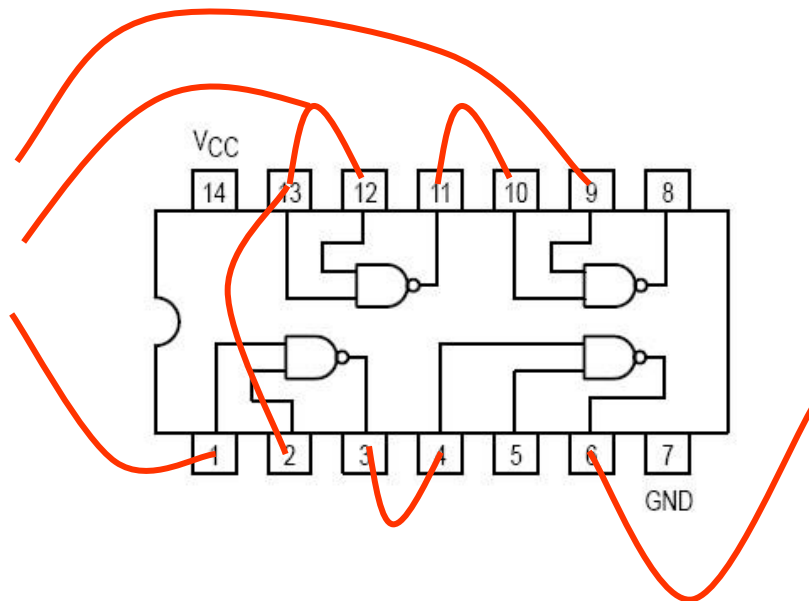
# Arquitectura de Computadoras

(3.4.072)



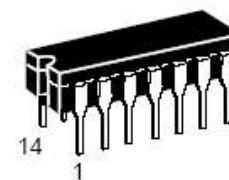
## QUAD 2-INPUT NAND GATE

- ESD > 3500 Volts



**SN54/74LS00**

**QUAD 2-INPUT NAND GATE**  
**LOW POWER SCHOTTKY**



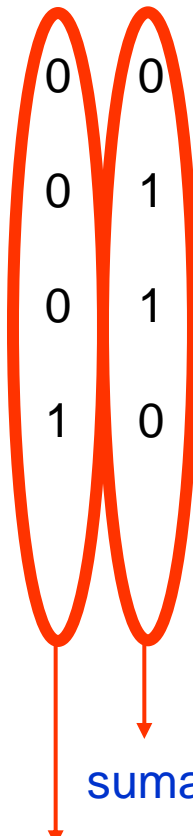
**J SUFFIX**  
**CERAMIC**  
**CASE 632-08**

# Arquitectura de Computadoras

## (3.4.072)

### ADICIÓN BINARIA:

	<i>A</i>	+	<i>B</i>	=	dec	bin	
Regla 1:	0	+	0	=	0	0	0
Regla 2:	0	+	1	=	1	0	1
Regla 3:	1	+	0	=	1	0	1
Regla 4:	1	+	1	=	2	1	0



acarreo

suma

# Arquitectura de Computadoras

## (3.4.072)

### ● Suma de dos bits:

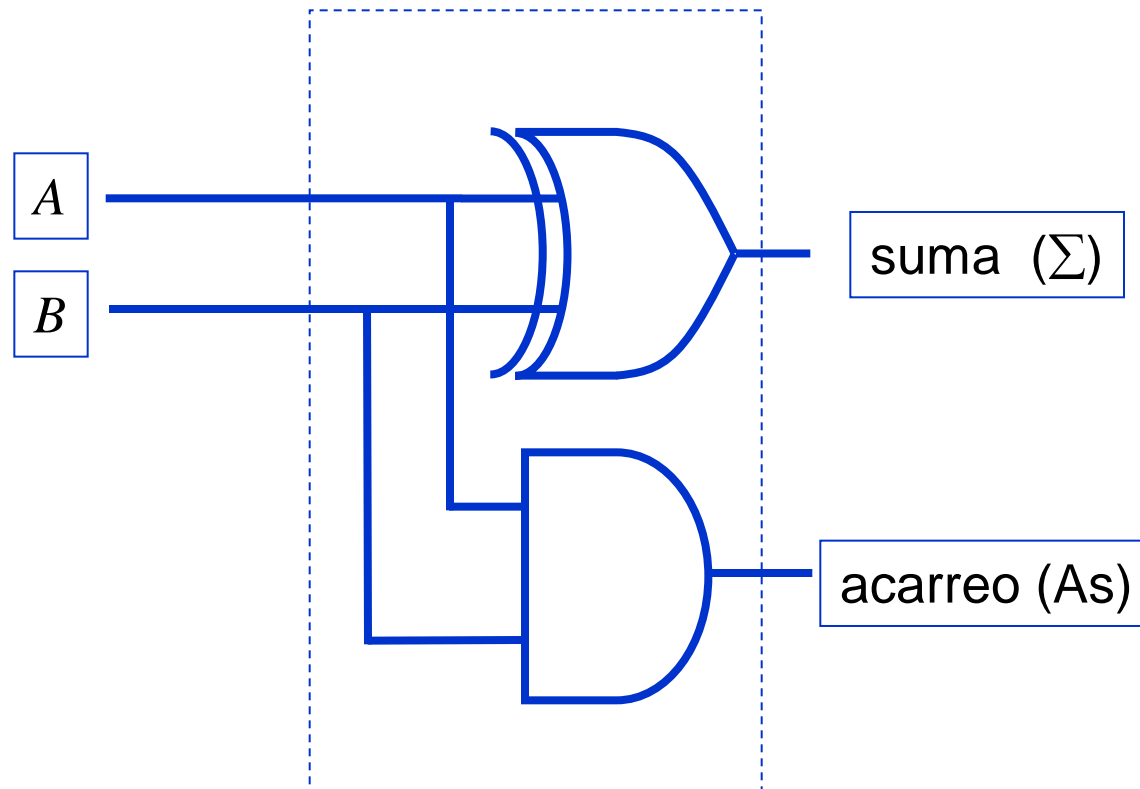
$A$	$B$	suma	acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

¿Cómo sería el circuito combinacional de suma y acarreo?

# Arquitectura de Computadoras

(3.4.072)

## ● Sumador de dos bits:



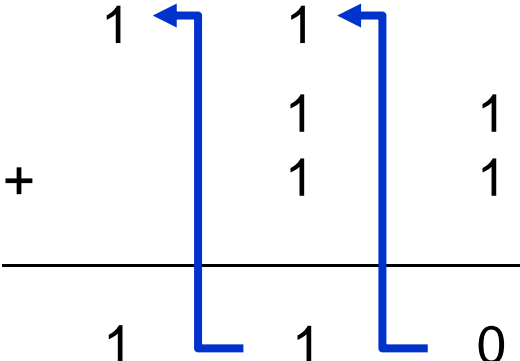
*half adder (1/2 sumador)*

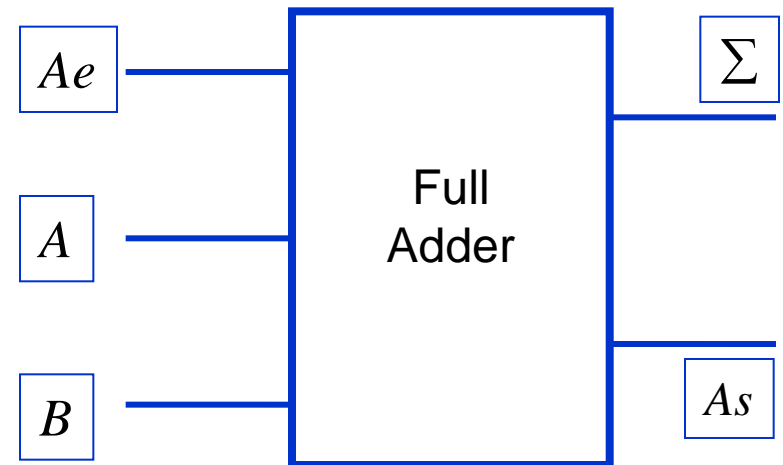
# Arquitectura de Computadoras

## (3.4.072)

*¿Cómo se suman números de dos bits?*

Ej:

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ + \quad 1 \quad 1 \quad 1 \\ \hline 1 \quad 1 \quad 0 \end{array}$$




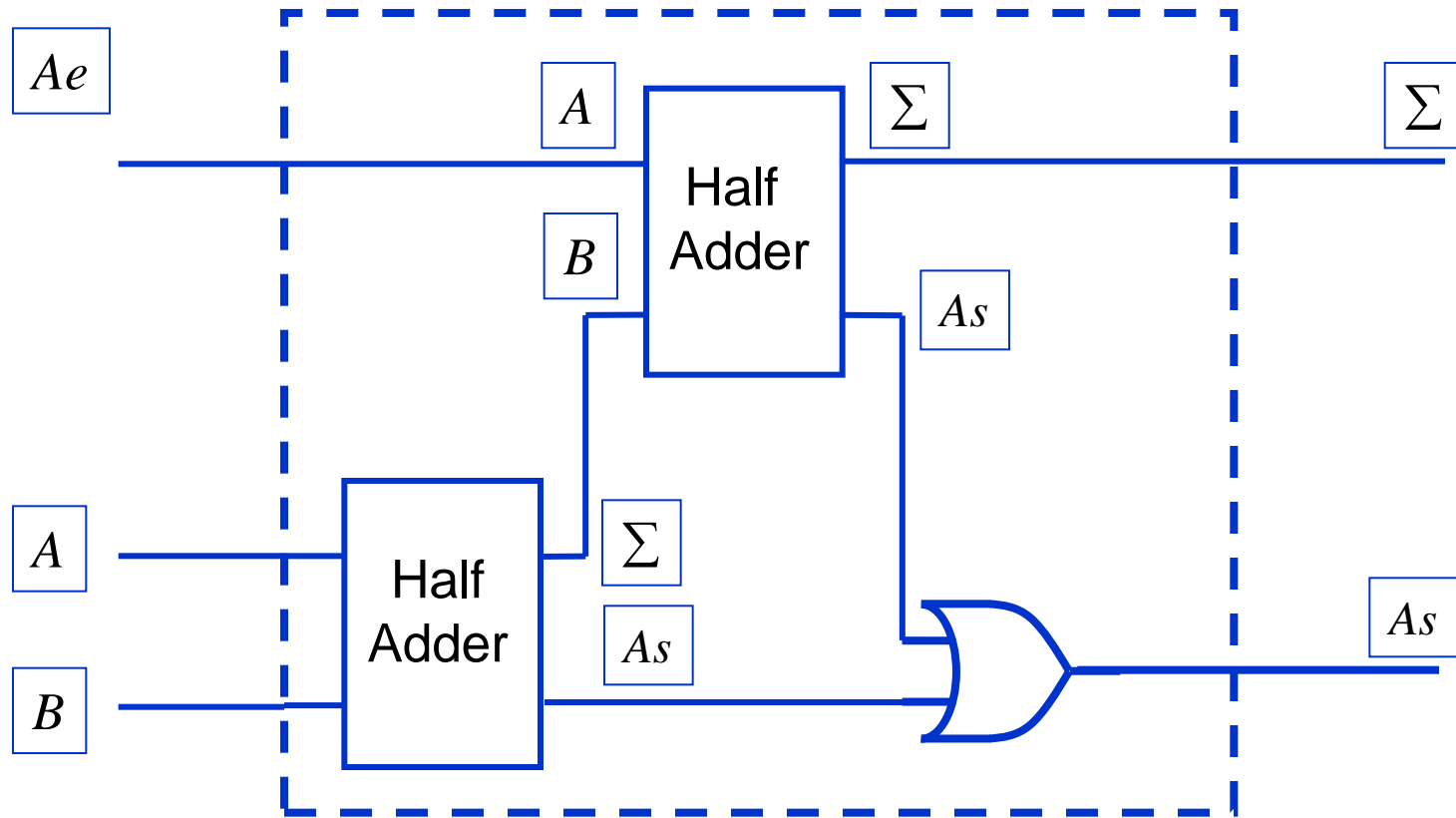
Se necesita un Full Adder (sumador completo) que considere el acarreo.



# Arquitectura de Computadoras

## (3.4.072)

### ● Estructura del sumador completo:

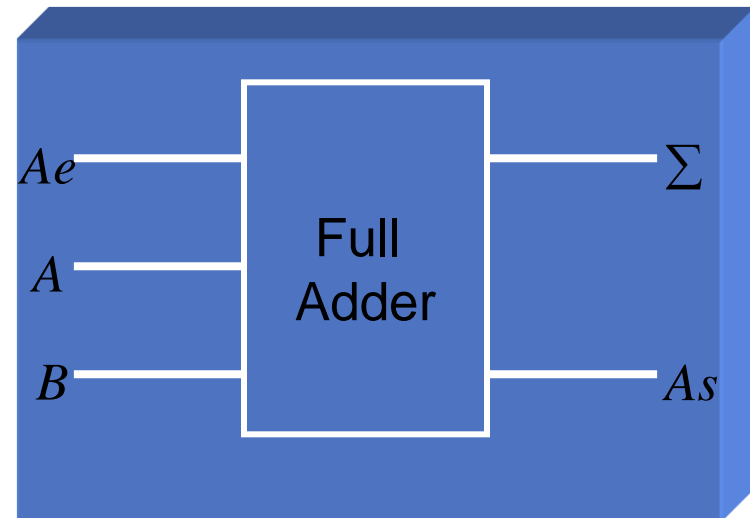
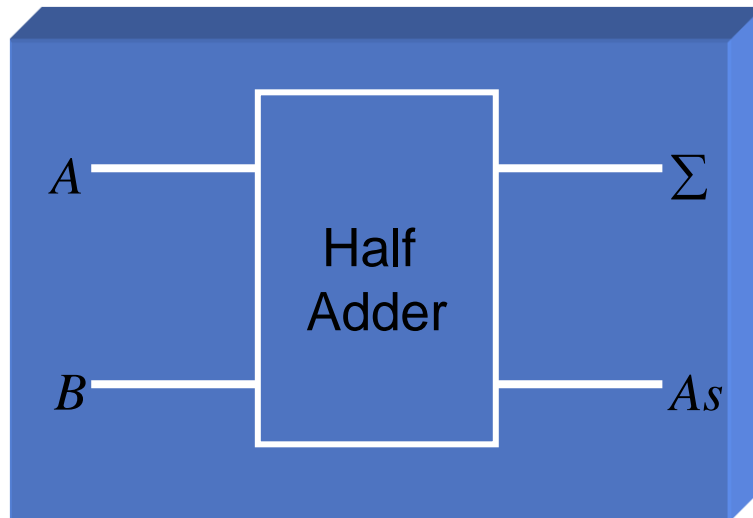


# Arquitectura de Computadoras

## (3.4.072)

Ejercicio: diseñar un sumador de cuatro bits usando half y/o full adders.

$$\begin{array}{r} A_4 A_3 A_2 A_1 \\ + B_4 B_3 B_2 B_1 \\ \hline C_5 C_4 C_3 C_2 C_1 \end{array}$$



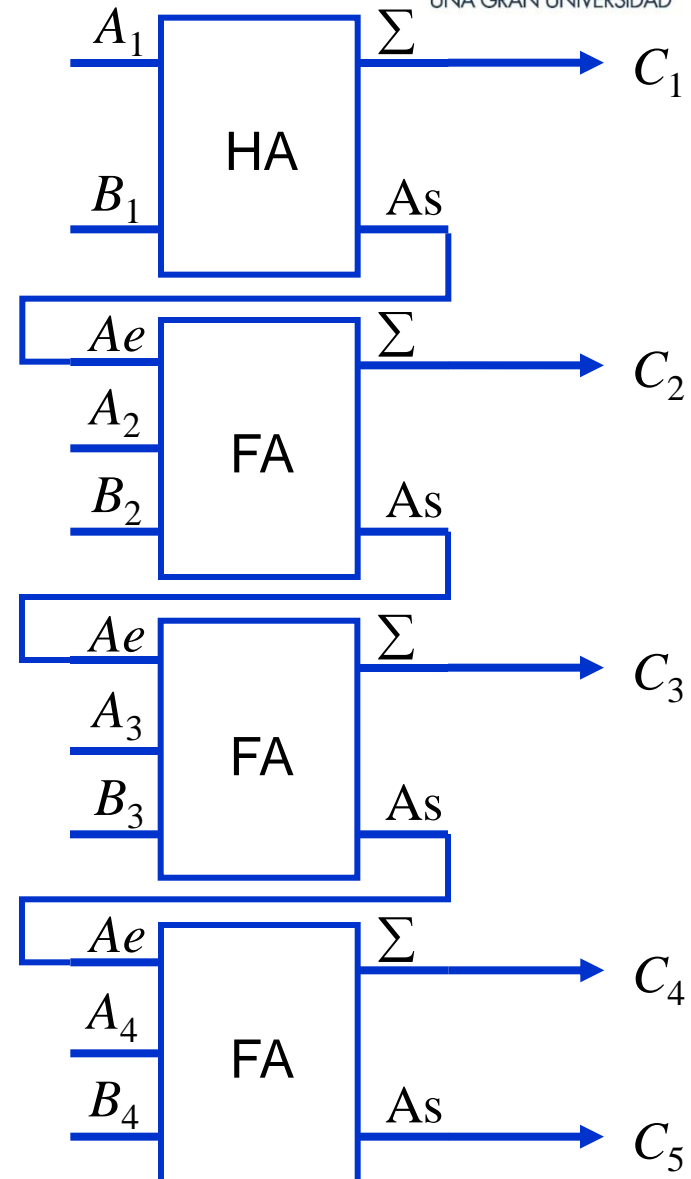
# Arquitectura de Computadoras

(3.4.072)

sumador de cuatro bits

$$\begin{array}{r} A_4 A_3 A_2 A_1 \\ + B_4 B_3 B_2 B_1 \\ \hline C_5 C_4 C_3 C_2 C_1 \end{array}$$

Especificaciones Técnicas



# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Sustracción binaria:

Para restar dos números binarios se puede utilizar el complemento a 2.

El complemento a 2 de un número binario es su complemento a 1 + 1.

Ej:     0010 1011  $\rightarrow$  B

	1101	0100	$\rightarrow$	Ca1	de	B
+		1				
<hr/>						
	1101	0101	$\rightarrow$	Ca2	de	B

# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Símbolos de los operadores:

Para calcular la resta binaria  $C = A - B$

- se calcula:  $B'$  como el complemento a 2 de  $B$
- se procede a calcular:  $C = A + B'$

Ejemplo:  $57|_{10} - 34|_{10}$ :

57:    0011 1001 (A)

34:    0010 0010 (B)

not    1101 1101 not (B)

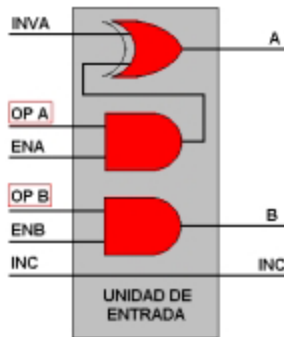
+1    1101 1110  $B'$

0001 0111  $A+B'$      $\Rightarrow$  0001 0111 =  $23|_{10}$

# Arquitectura de Computadoras

## (3.4.072)

- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**

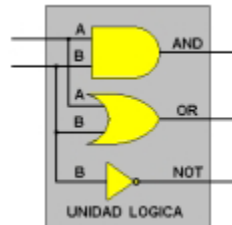
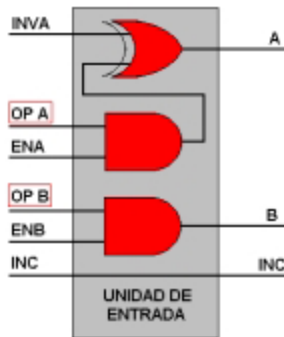


1 - Unidad de Entrada: *preprocesa operandos Bit A y Bit B (Habilita A y B , Invierte A)*

# Arquitectura de Computadoras

## (3.4.072)

- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**

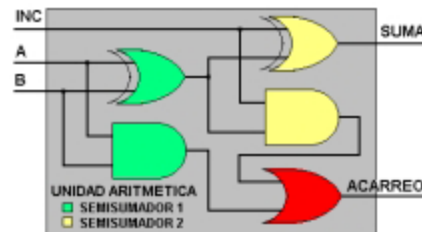
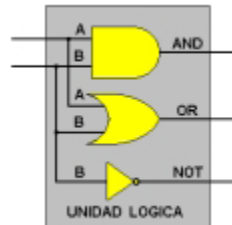
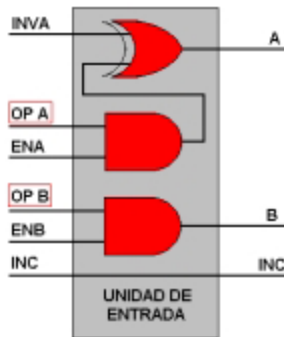


2 - Unidad Lógica: *Realiza tres operaciones A and B , A or B y not B*

# Arquitectura de Computadoras

## (3.4.072)

- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**



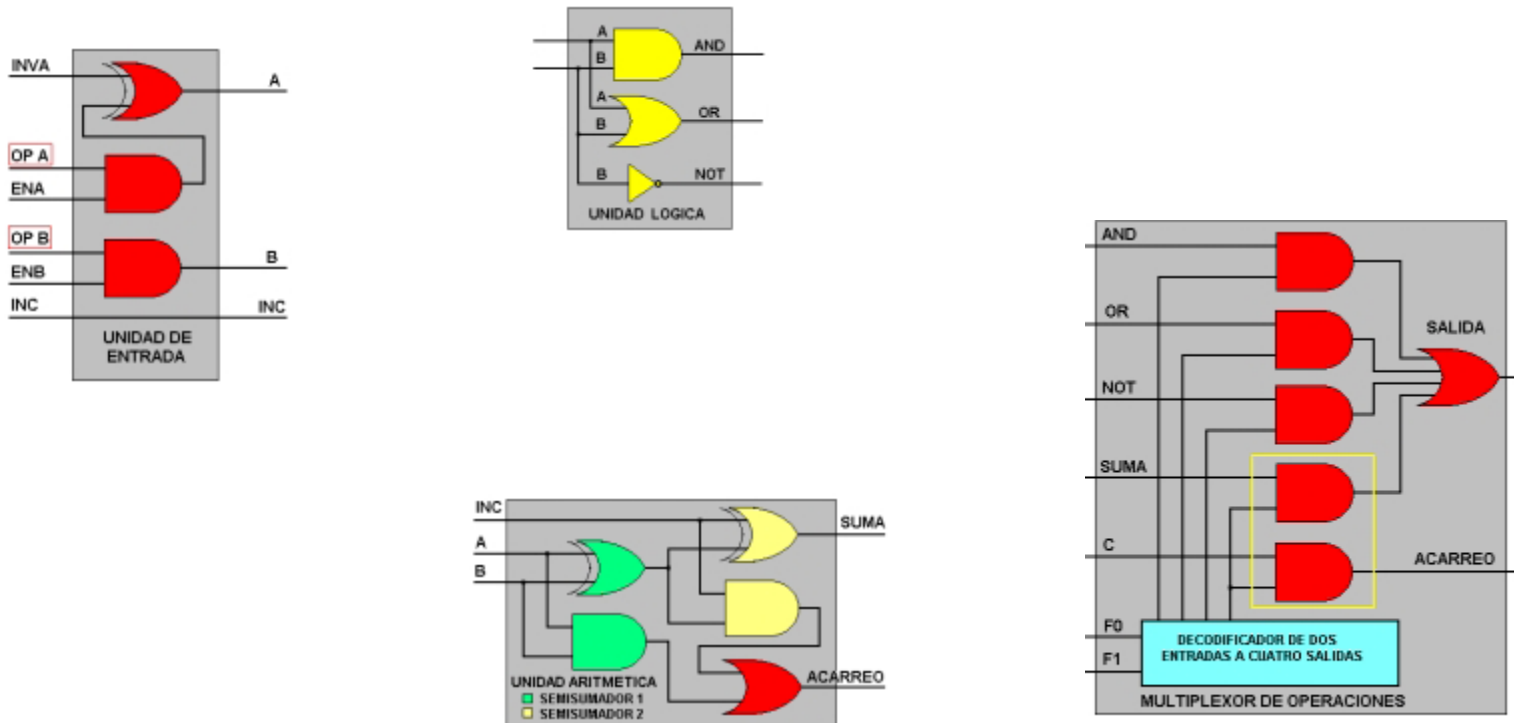
3- Unidad Aritmética: *Realiza la suma aritmética del Bit A y el Bit B ( Inc. y Acarreo)*



# Arquitectura de Computadoras

## (3.4.072)

### ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

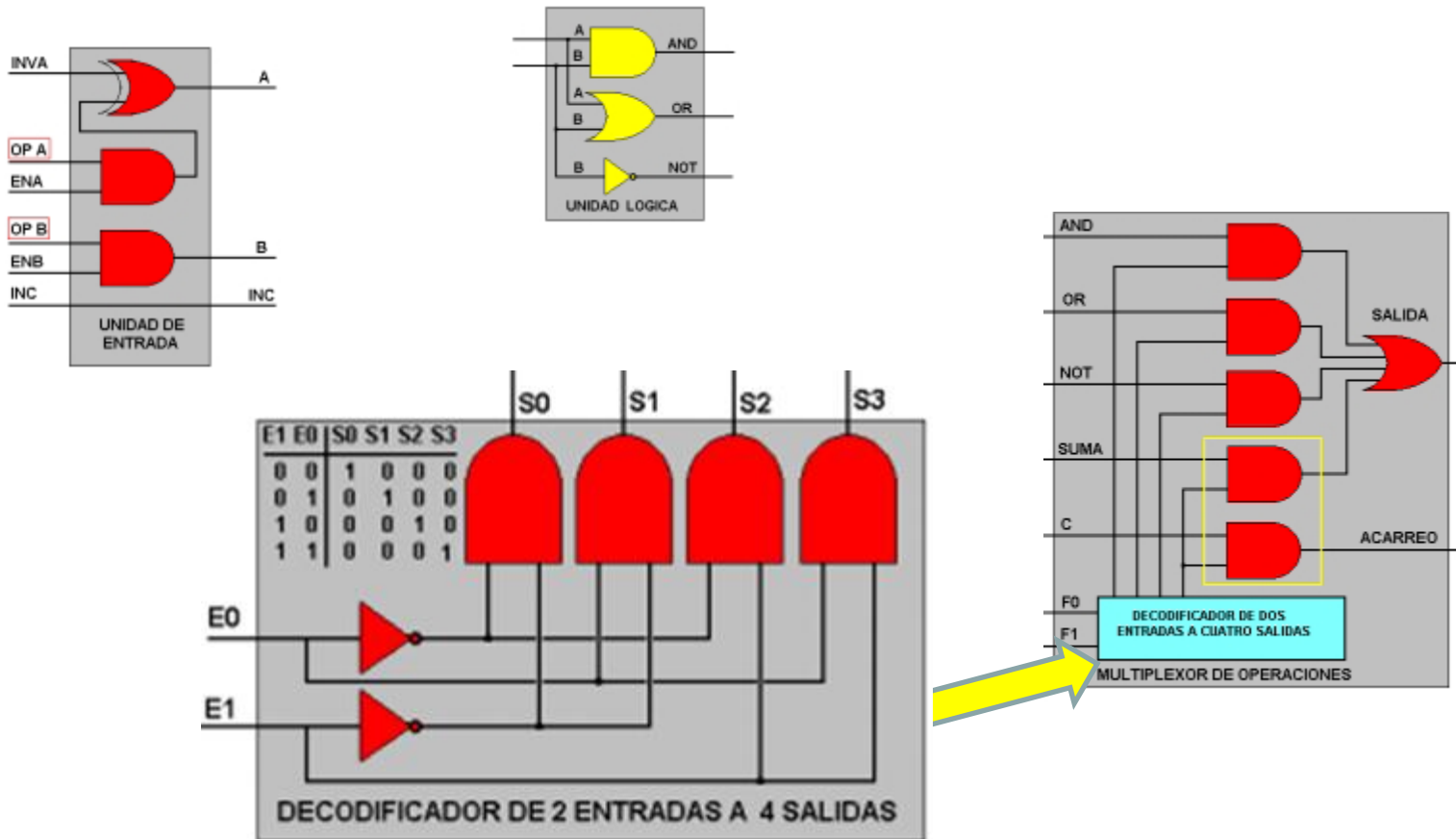


4 - Multiplexor de Operaciones: *Selecciona una operación de 4 , usa un decodificador*

# Arquitectura de Computadoras

## (3.4.072)

### ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

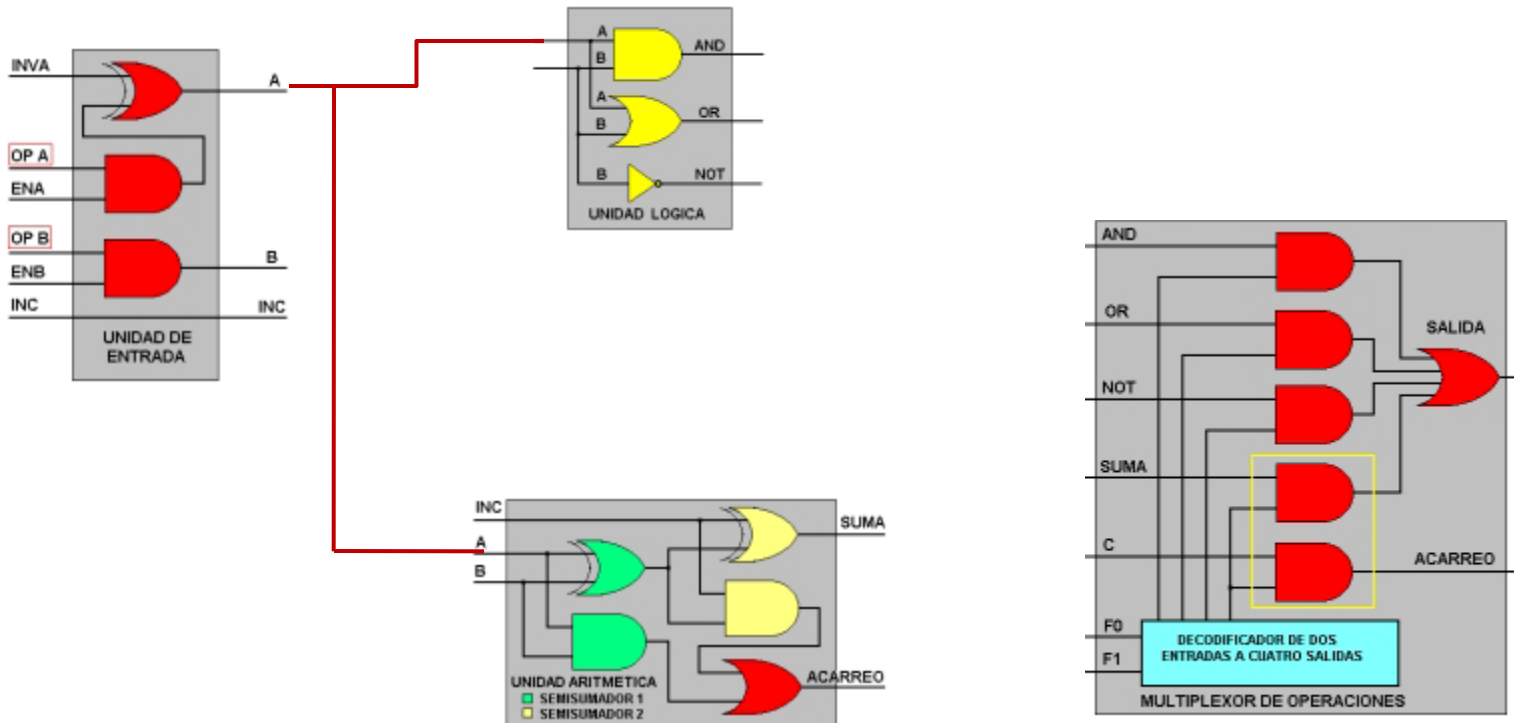


5 – Decodificador: *Activa la salida que se indica con el código binario de entrada*

# Arquitectura de Computadoras

## (3.4.072)

### ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

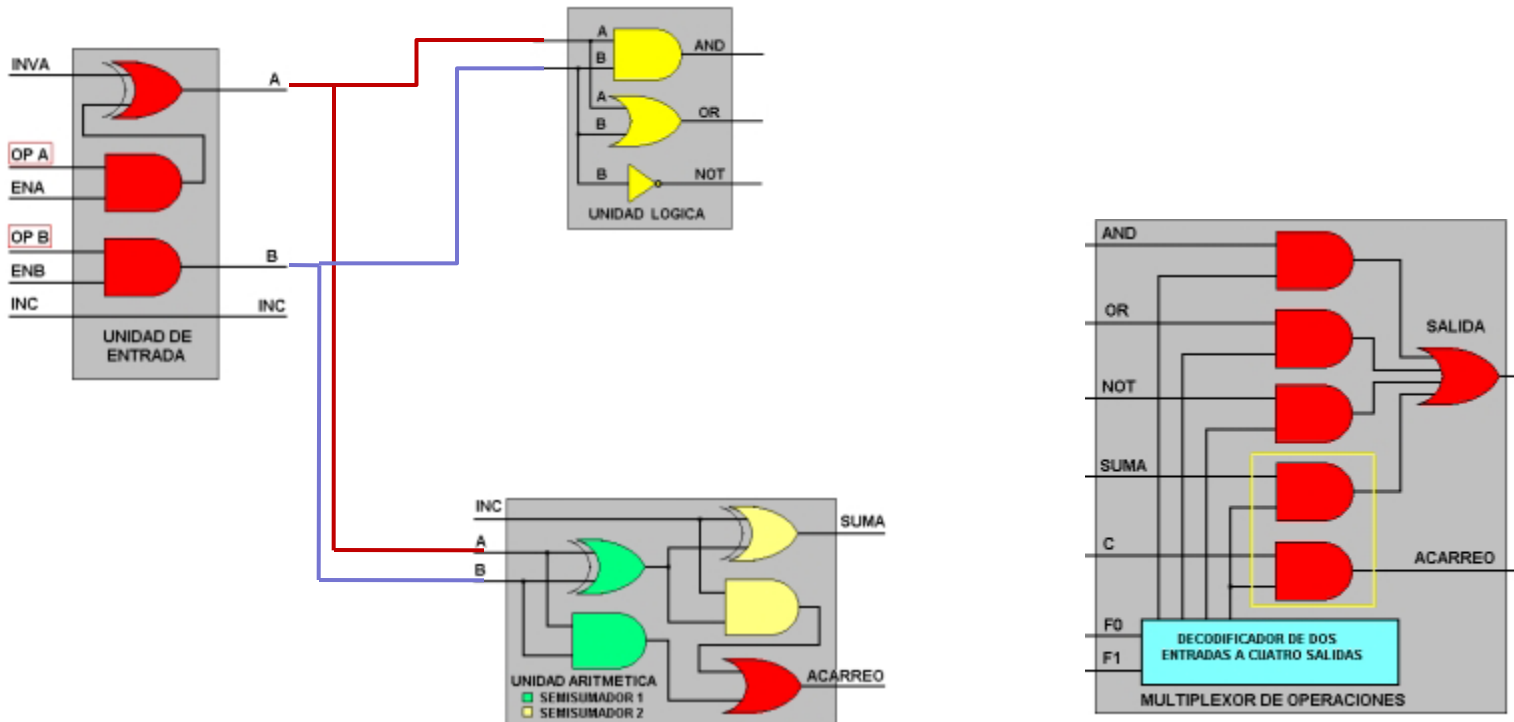


7- Entrada A: *Primer Operando Bit A*

# Arquitectura de Computadoras

## (3.4.072)

### ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

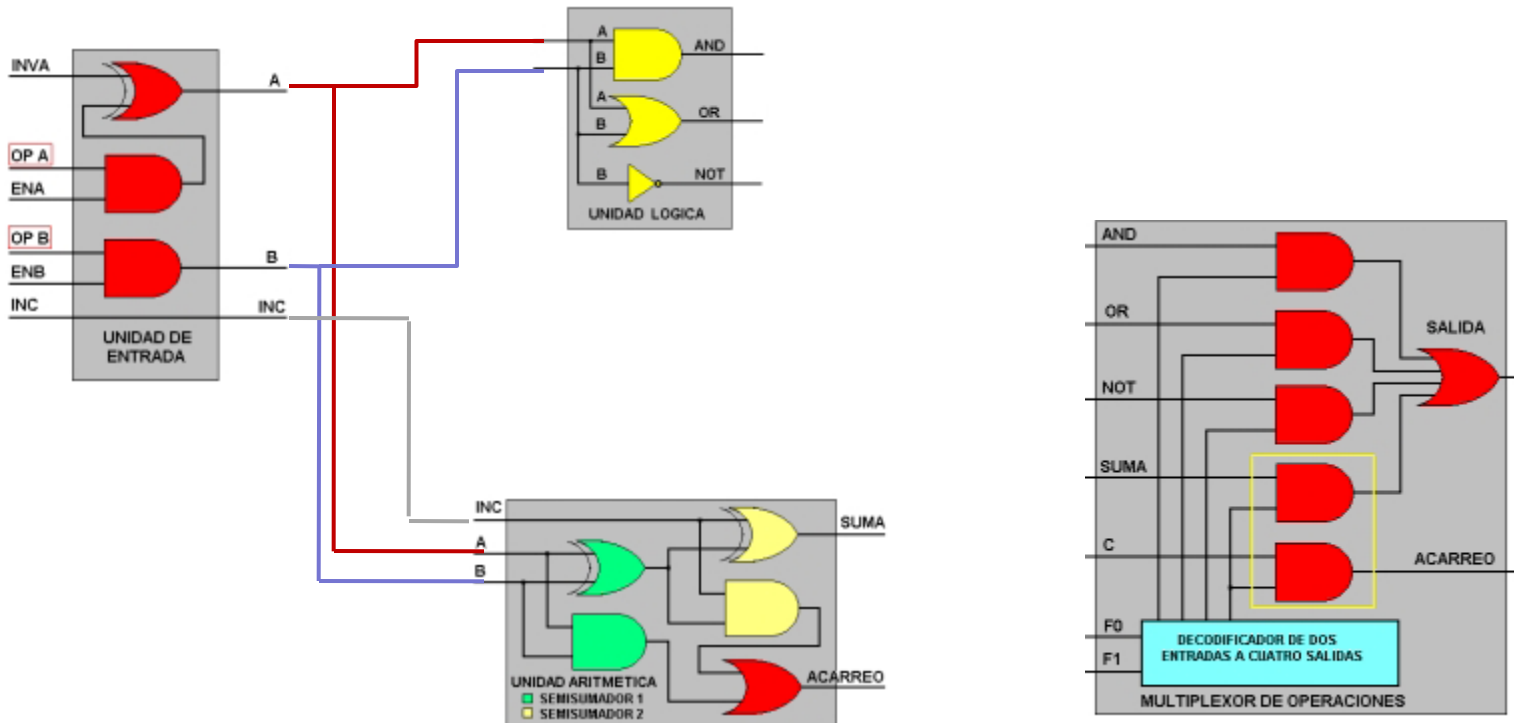


8 - Entrada B: *Segundo operando Bit B*

# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

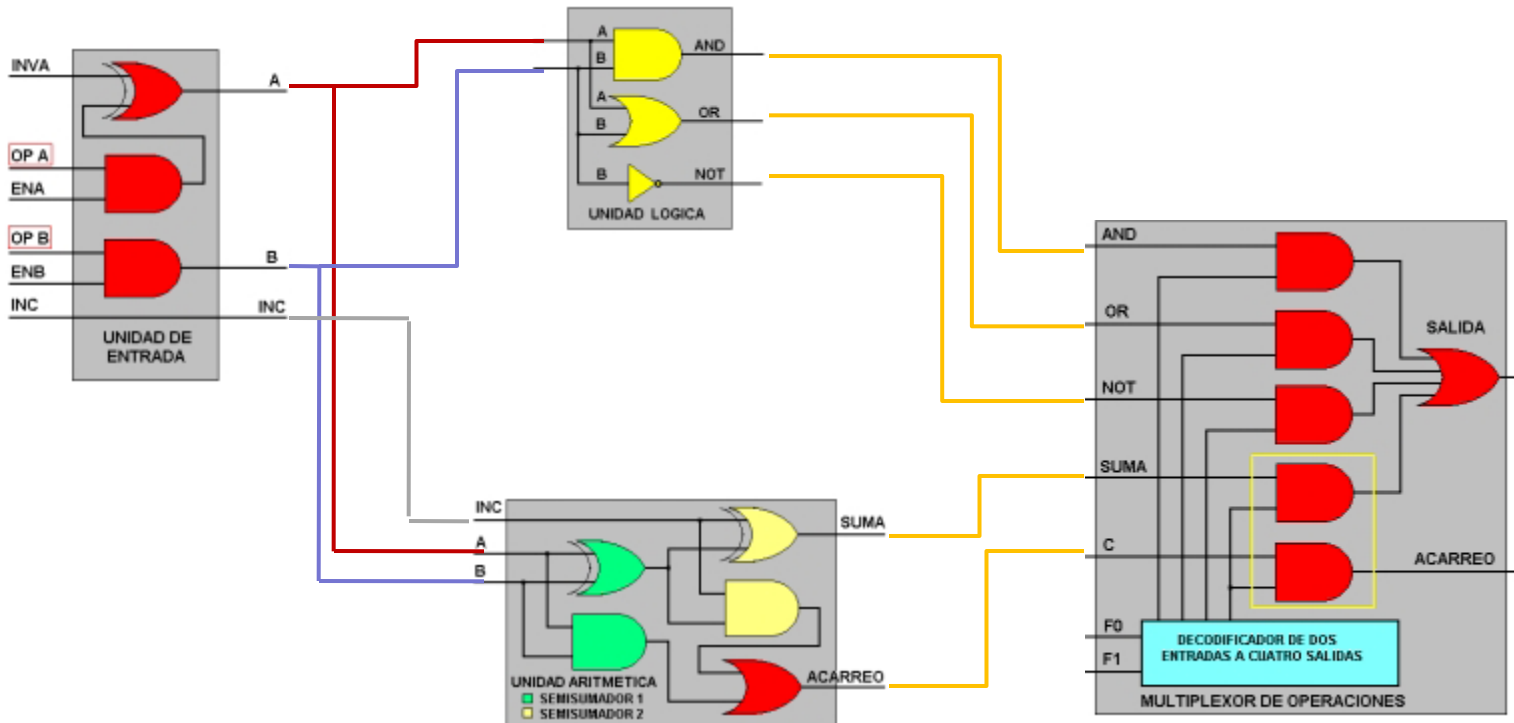


9 – Incremento: *Transporte o acarreo de entrada del módulo aritmético.*

# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

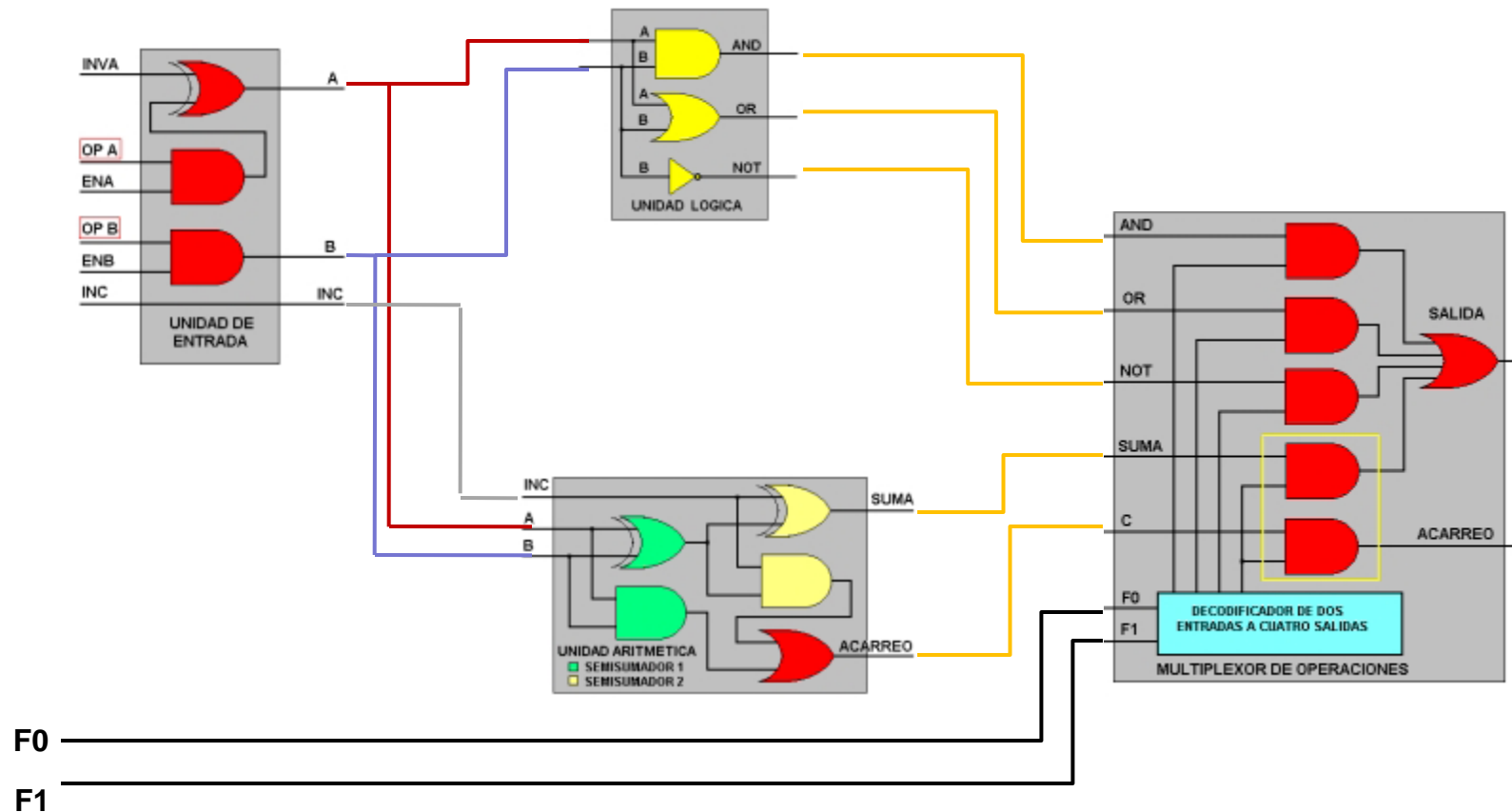


10 - Salidas de Operaciones: *Todas las Operaciones Aritmético lógicas al MUX4-1*

# Arquitectura de Computadoras

(3.4.042 / 3.4.072)

## ● Unidad Aritmético Lógica de 1 Bit, etapas de diseño:

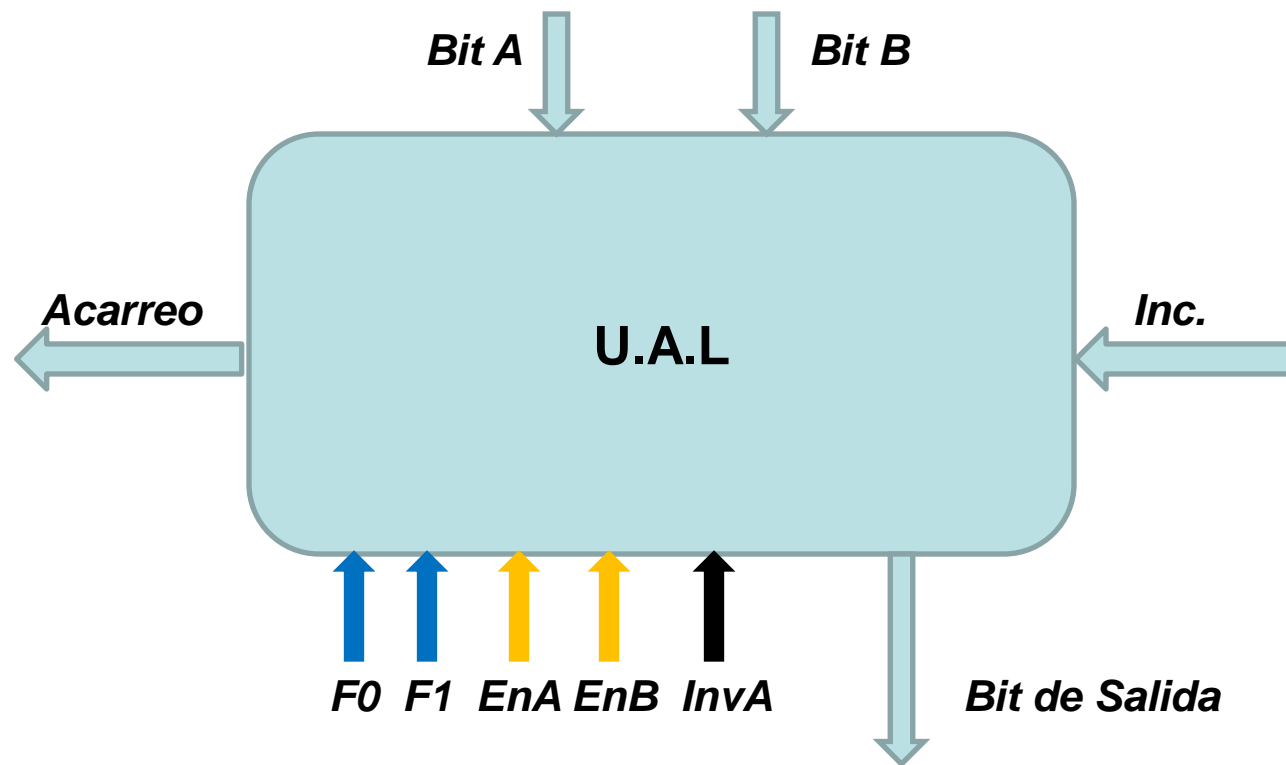


11 - Selector de Operación: *F0* y *F1* con *ENA*, *ENB*, *INVA* e *INC*

# Arquitectura de Computadoras

## (3.4.072)

- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**

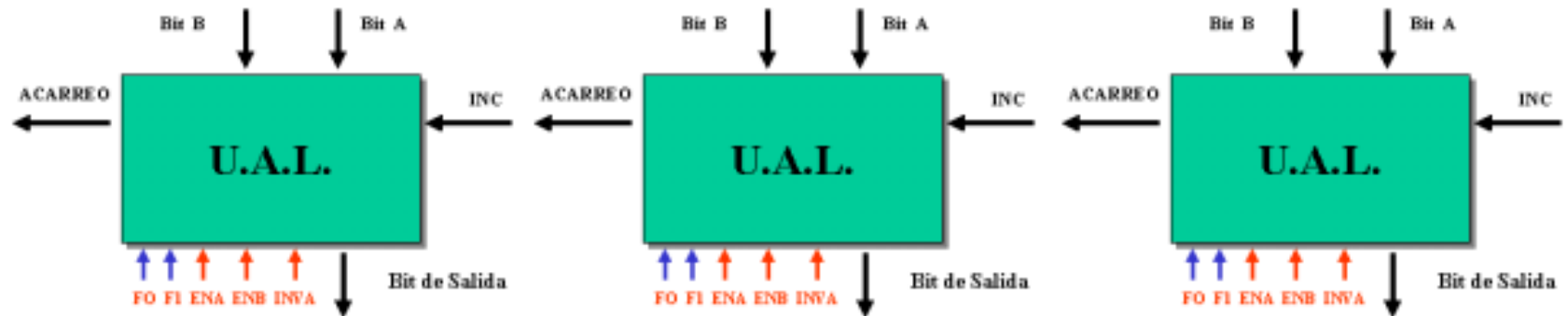




# Arquitectura de Computadoras

## (3.4.072)

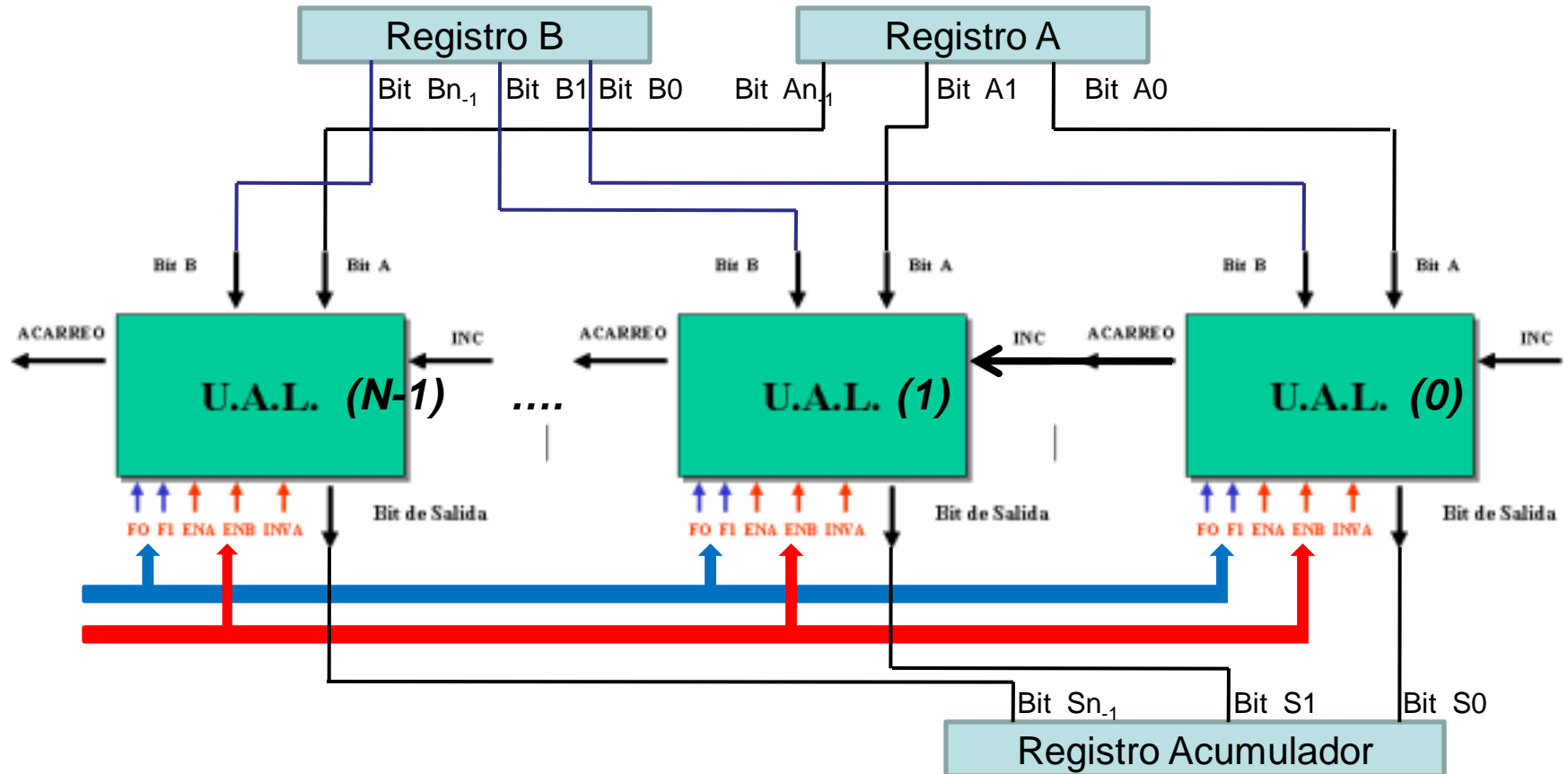
- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**



# Arquitectura de Computadoras

## (3.4.072)

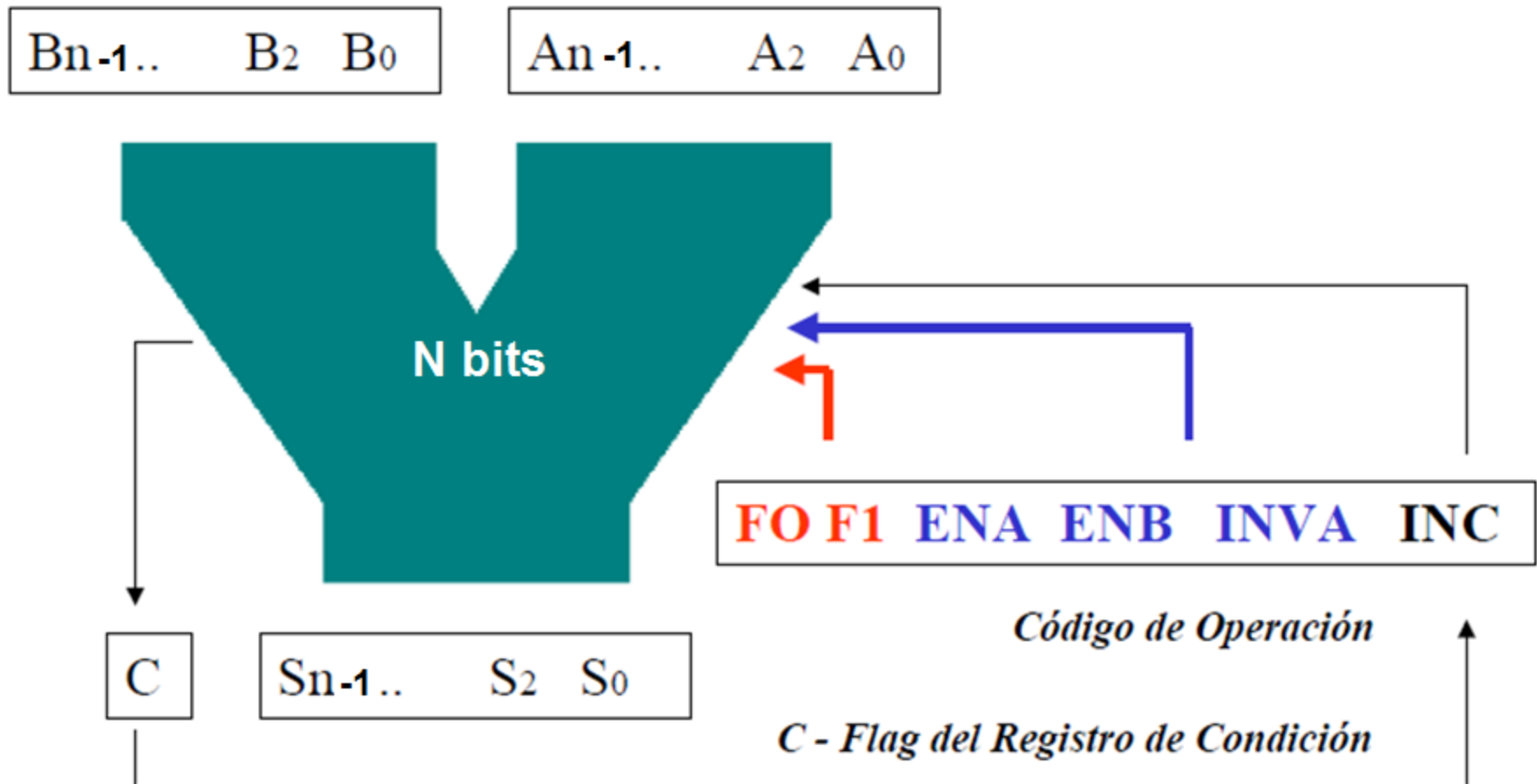
### ● *Unidad Aritmético Lógica de 1 Bit, etapas de diseño:*



# Arquitectura de Computadoras

## (3.4.072)

- **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**



# Arquitectura de Computadoras

## (3.4.072)

● **Unidad Aritmético Lógica de 1 Bit, etapas de diseño:**

$F_0$	$F_1$	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1