

Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2343 – Arquitectura de Computadores

Operaciones Aritméticas y Lógicas Parte 2: Unidad Aritmética Lógica (ALU)

**Profesor:** Hans Löbel

# Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.



# Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.



## Álgebra booleana permite definir sentencias lógicas complejas

- Lo mejor es que podemos representar operaciones aritméticas fácilmente

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

$$C = A \wedge B$$

# Como cambiar de analógico a digital

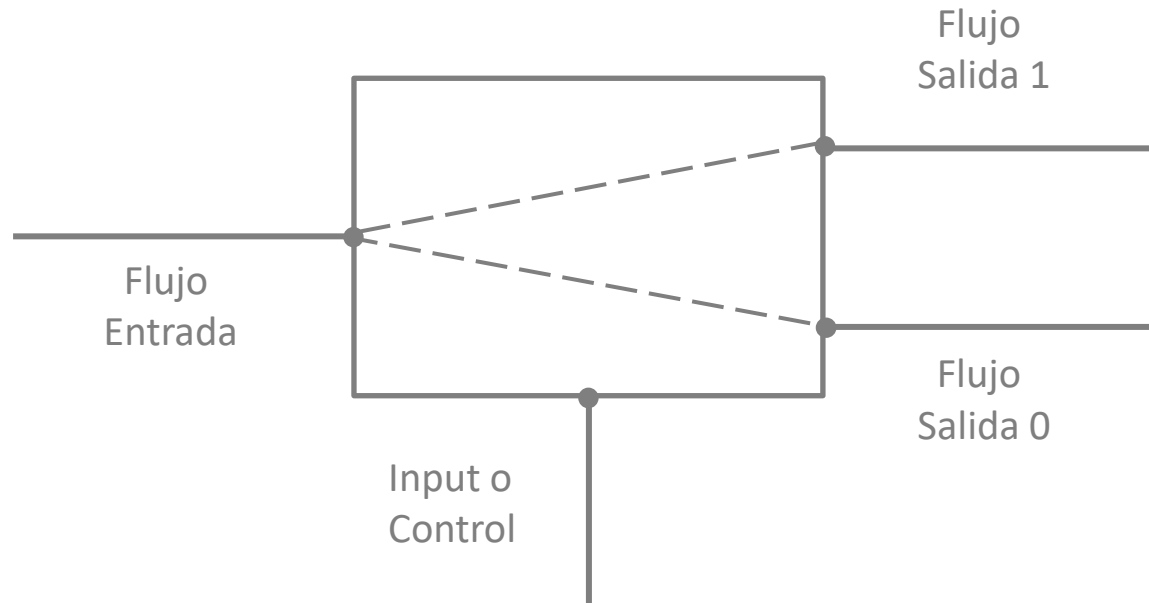
- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.



Podemos implementar físicamente los componentes de la lógica booleana

- El gran aporte de Shannon, fue describir implementaciones físicas de AND, OR y NOT.
- Por cada uno de estos conectivos definió una compuerta lógica.
- Luego, implementó cada compuerta mediante relés.

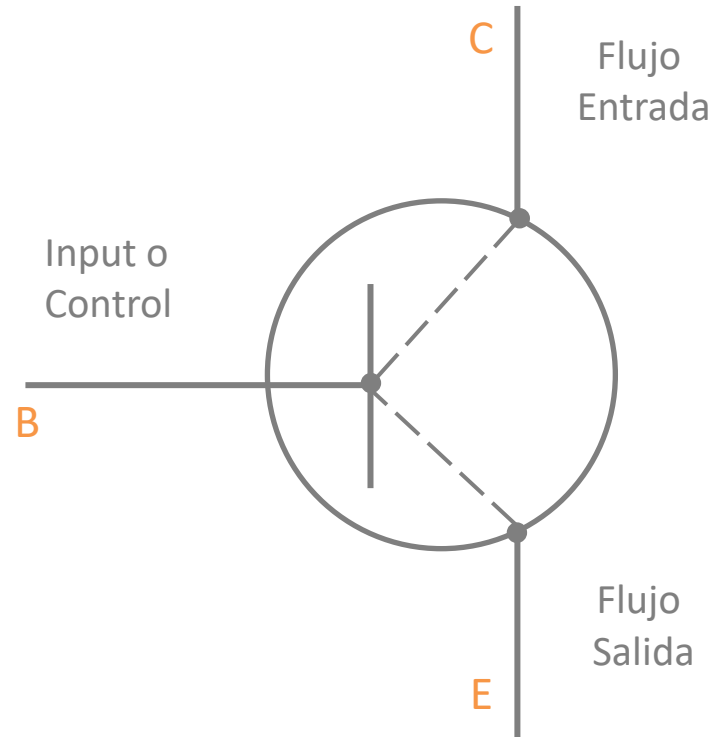
# Relé



Sin flujo de control => Input == 0 → Flujo por salida 0

Con flujo de control => Input == 1 → Flujo por salida 1

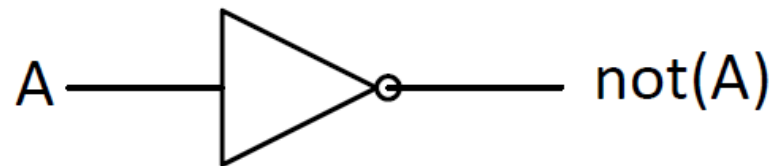
# Transistor



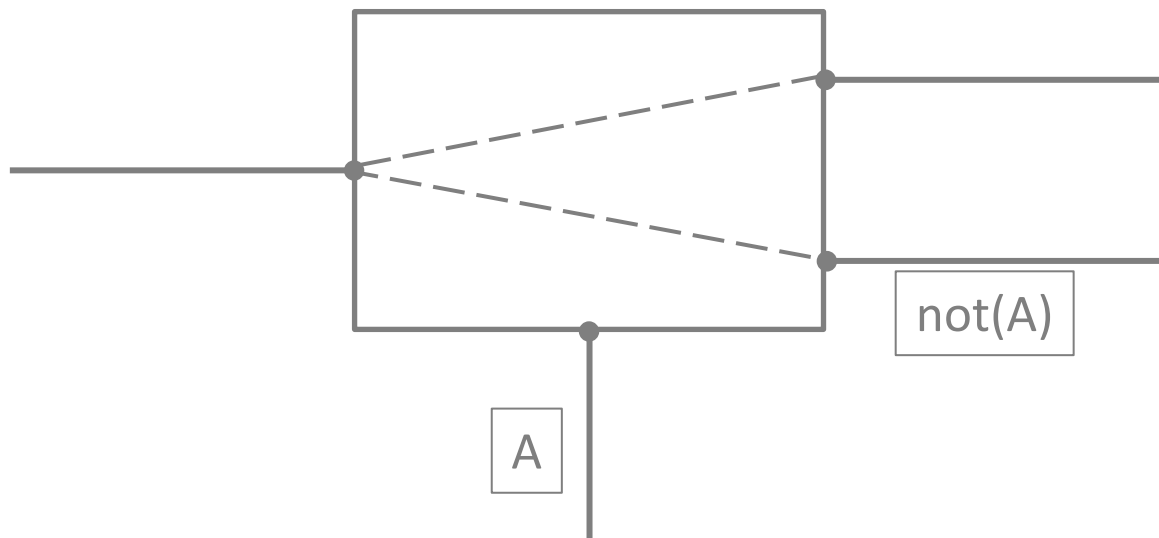
$B == 0 \rightarrow$  Sin flujo por  $E$

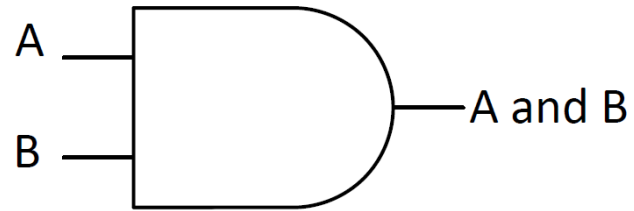
$B == 1 \rightarrow$  Flujo por  $E$



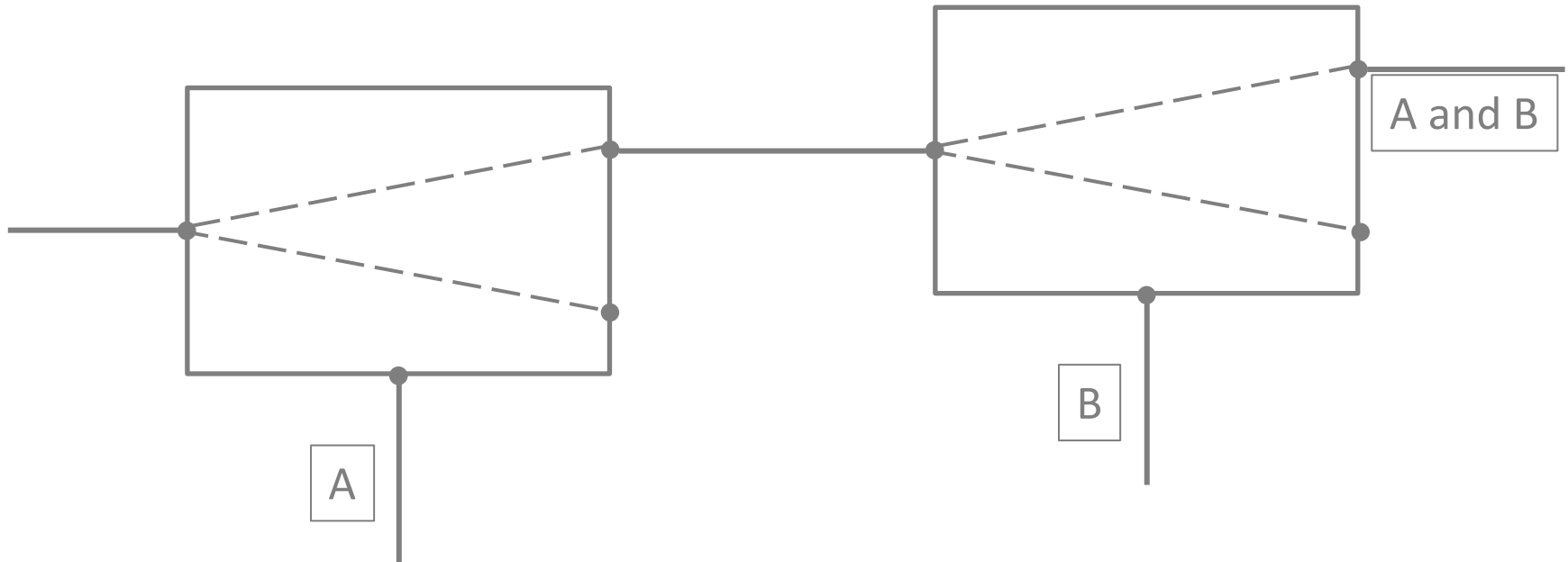


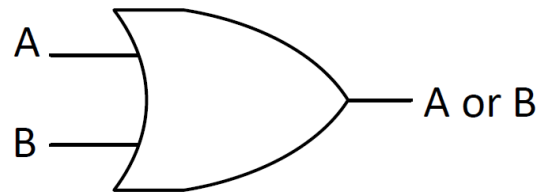
A	not(A)
0	1
1	0



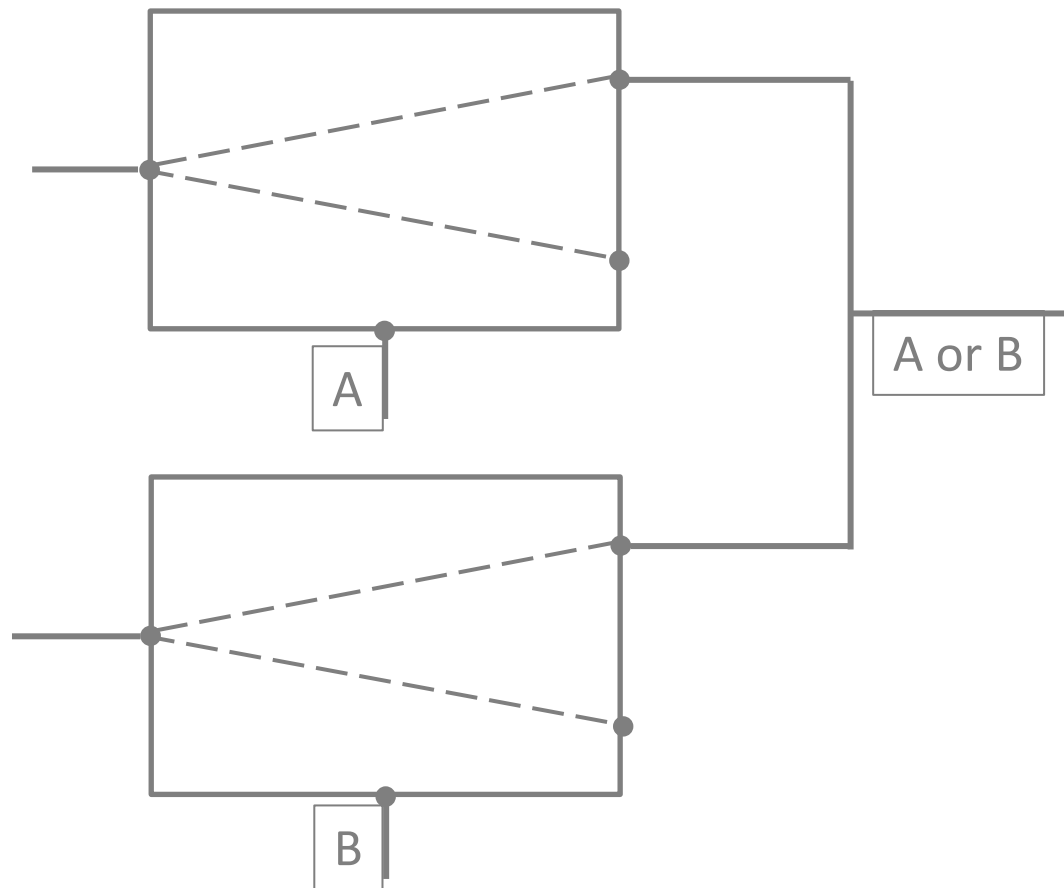


A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1



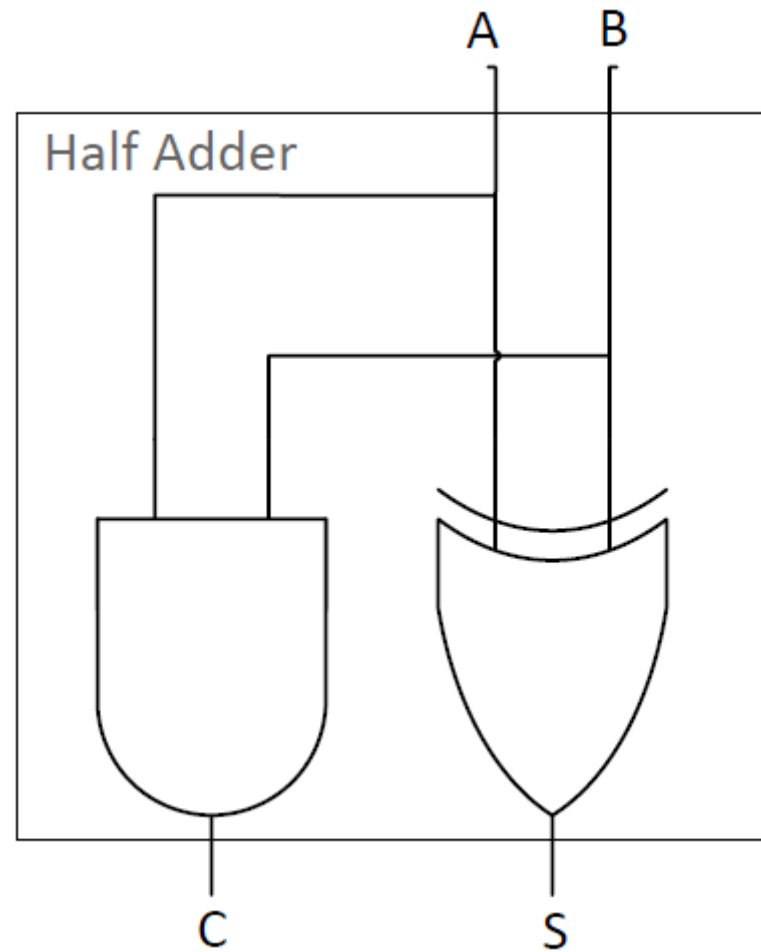


A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1



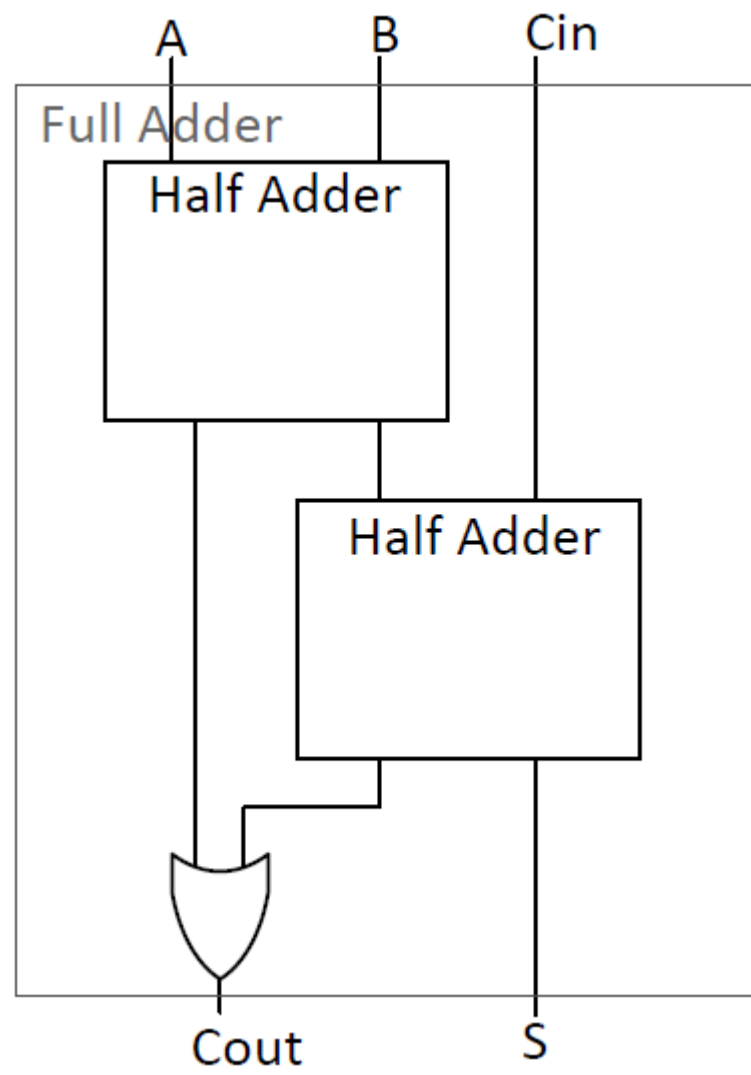
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

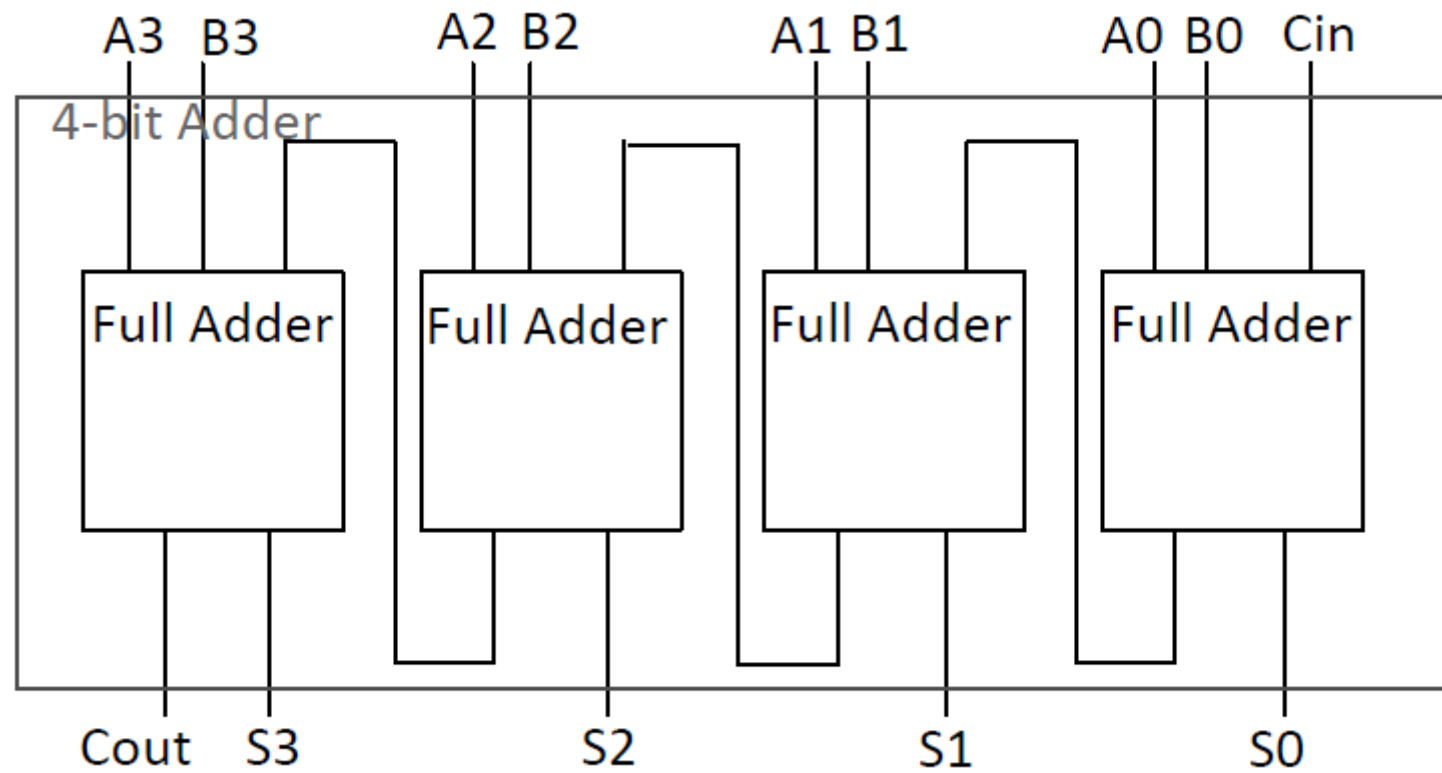
- $S = A \oplus B$
- $C = A \wedge B$



La clave consiste en tener módulos combinables y expandibles

- Aprovechemos la versatilidad de la lógica booleana y ampliamos el sumador a 4 bits.
- ¿Sirve poner 4 half-adders en paralelo?
- Primero debemos construir un full-adder, que considere el carry de una operación anterior.

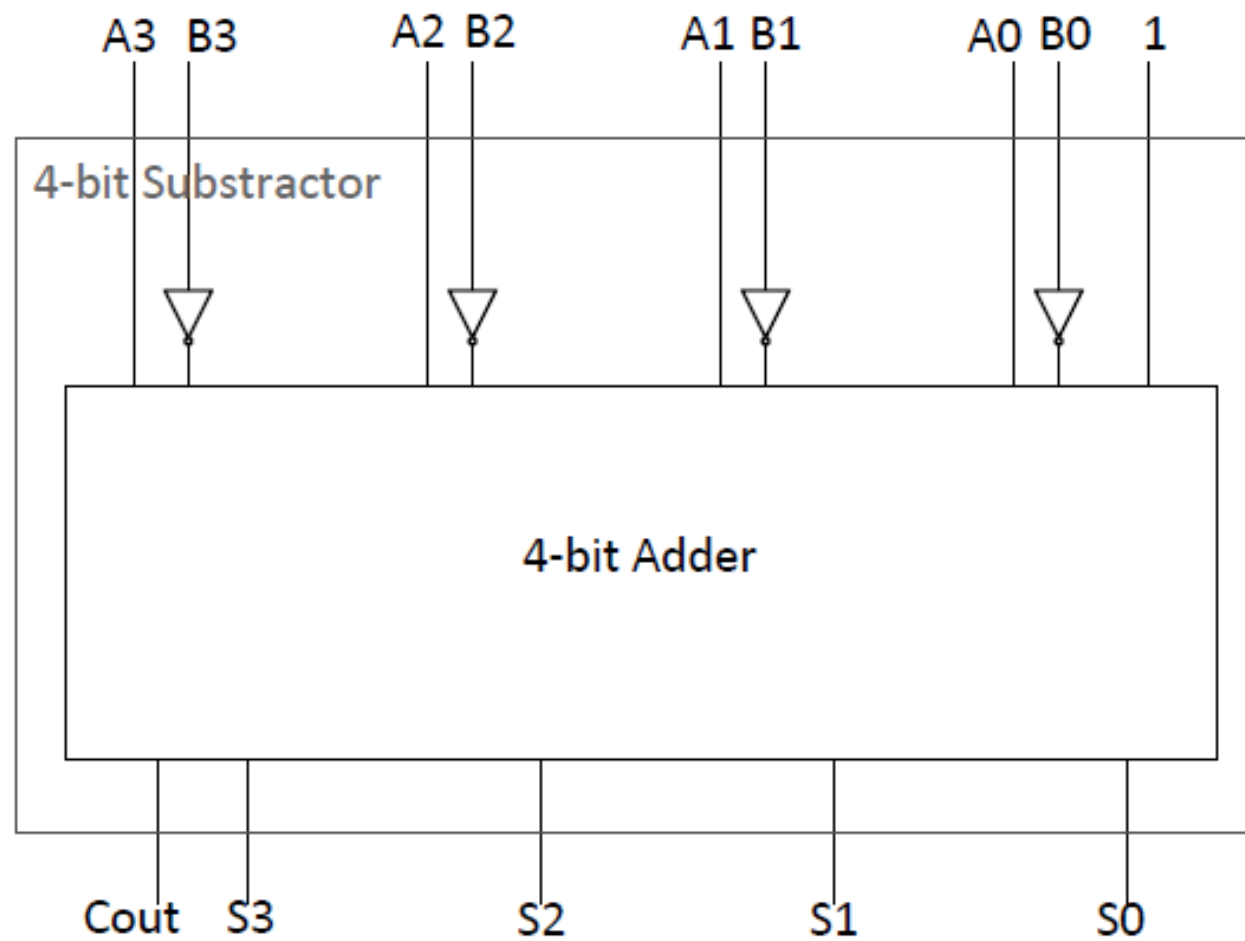




La clave consiste en tener módulos combinables y expandibles

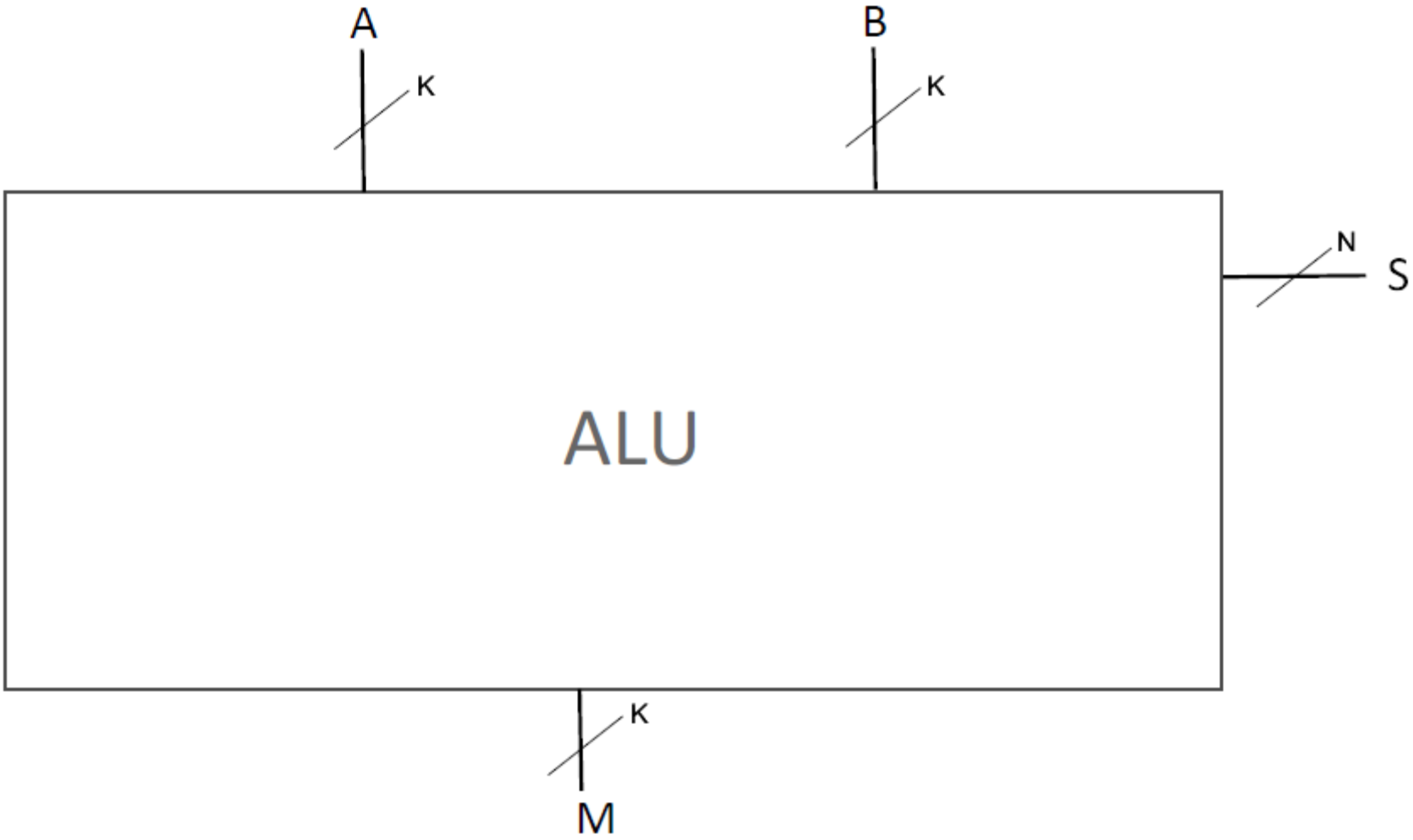
- ¿Qué debemos hacer para diseñar un restador?
- Afortunadamente, no necesitamos mucho más.
- Aprovechando el **complemento a 2**, sumamos al minuendo el inverso aditivo del sustraendo y además ponemos en 1 el carry-in.

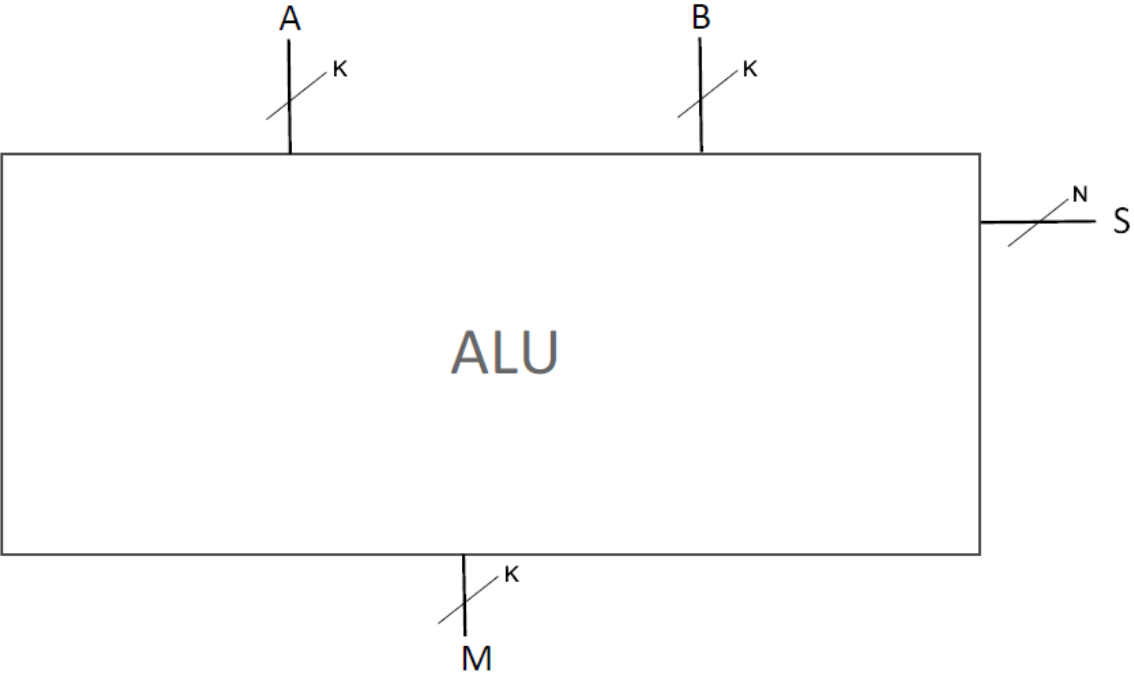




## El centro de operaciones de un computador

- Un computador realiza principalmente cálculos, por lo que es fundamental construir una unidad eficiente para esto.
- El centro de operaciones de un computador es la unidad aritmética lógica (ALU), por la cual pasan todos los datos.





S2	S1	S0	M
0	0	0	Suma
0	0	1	Resta
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Xor
1	1	0	Shift left
1	1	1	Shift right

## Enfrentemos primero las operaciones aritméticas

- Nuestro primer paso será un sumador/restador de 4 bits
- ¿Podemos construir esto usando sólo los elementos vistos hasta ahora en clases?

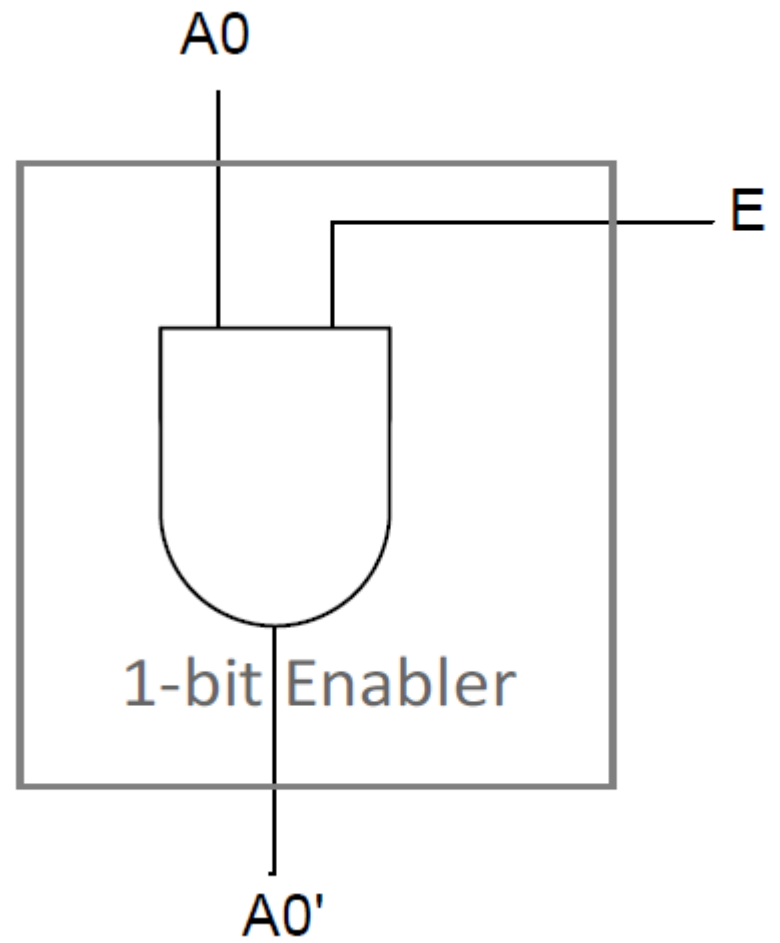
## Elementos de control son necesarios

- Para diseñar un el sumador/restador aún nos faltan algunos elementos lógicos.
- Estos tienen relación con el control y la selección de las salidas y operaciones que se realizan.
- Estos elementos se conocen como **Enablers** y **Multiplexores** o **Mux**.

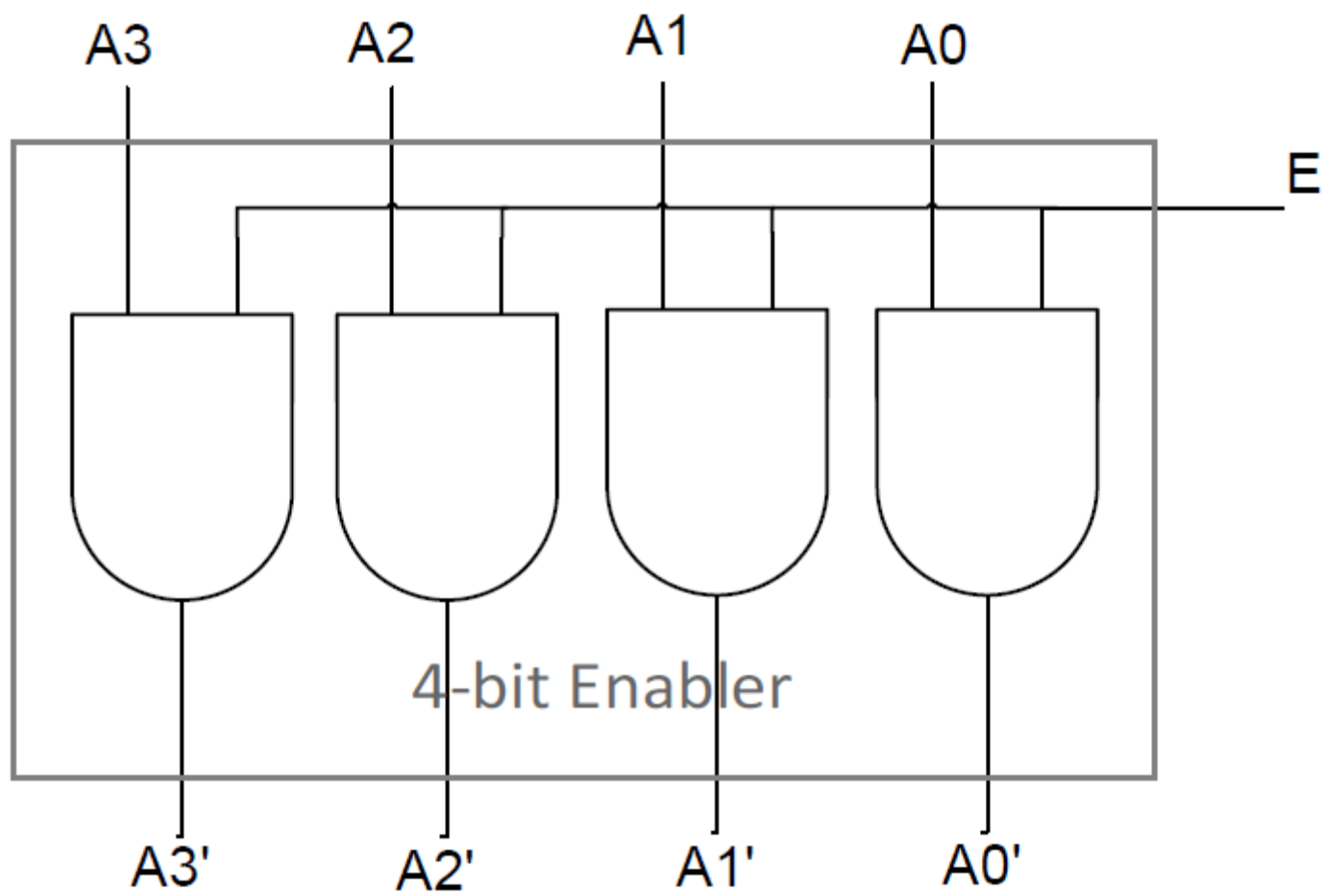
## Buses otorgan simplificación en el diseño de los componentes

- Las conexiones usadas anteriormente sólo transmitían datos, ej. sumador.
- Ahora, además de eso, necesitamos transmitir órdenes de control.
- Así, se hace la distinción entre buses de datos y buses de control.

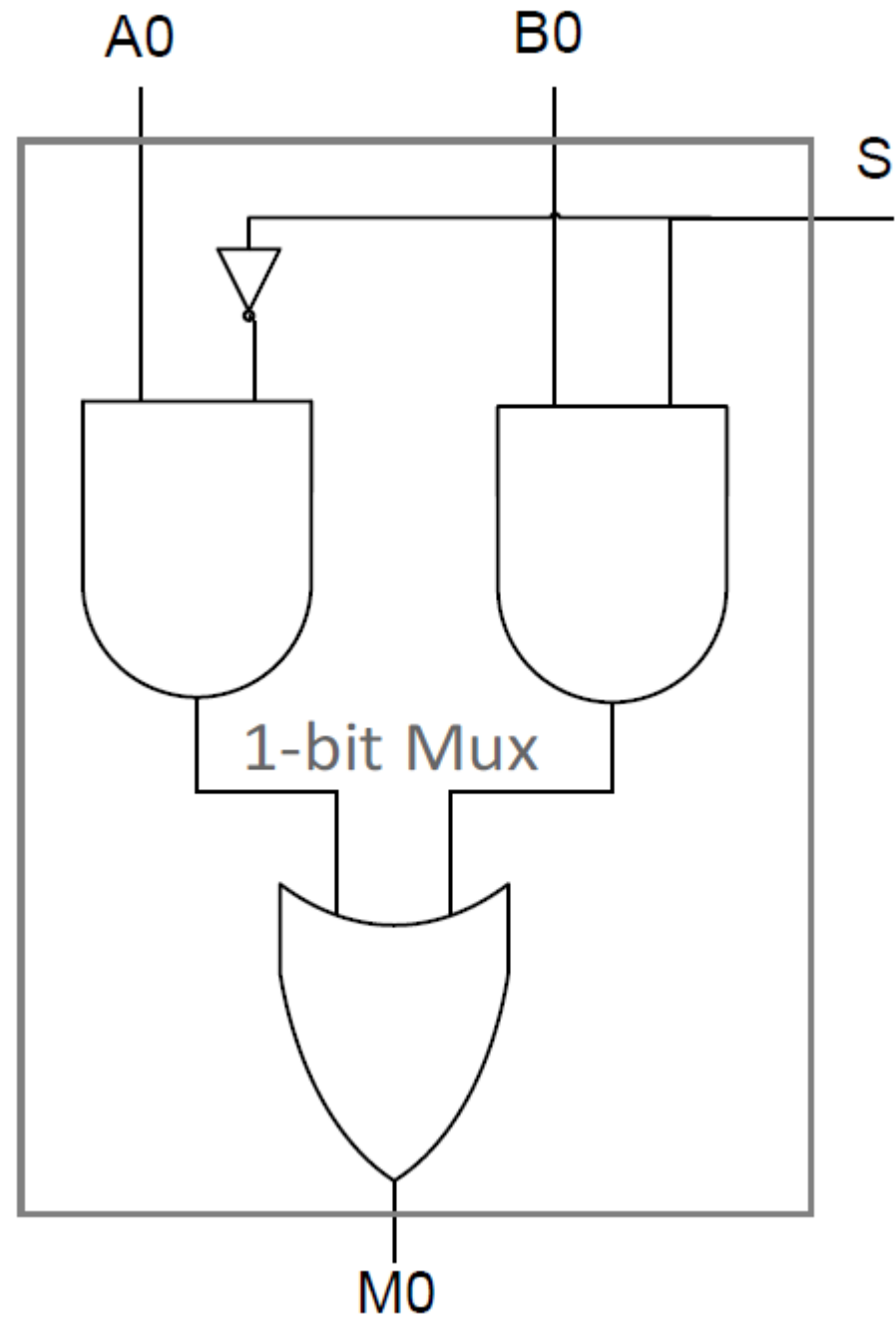
E	$A0'$
0	0
1	$A0$

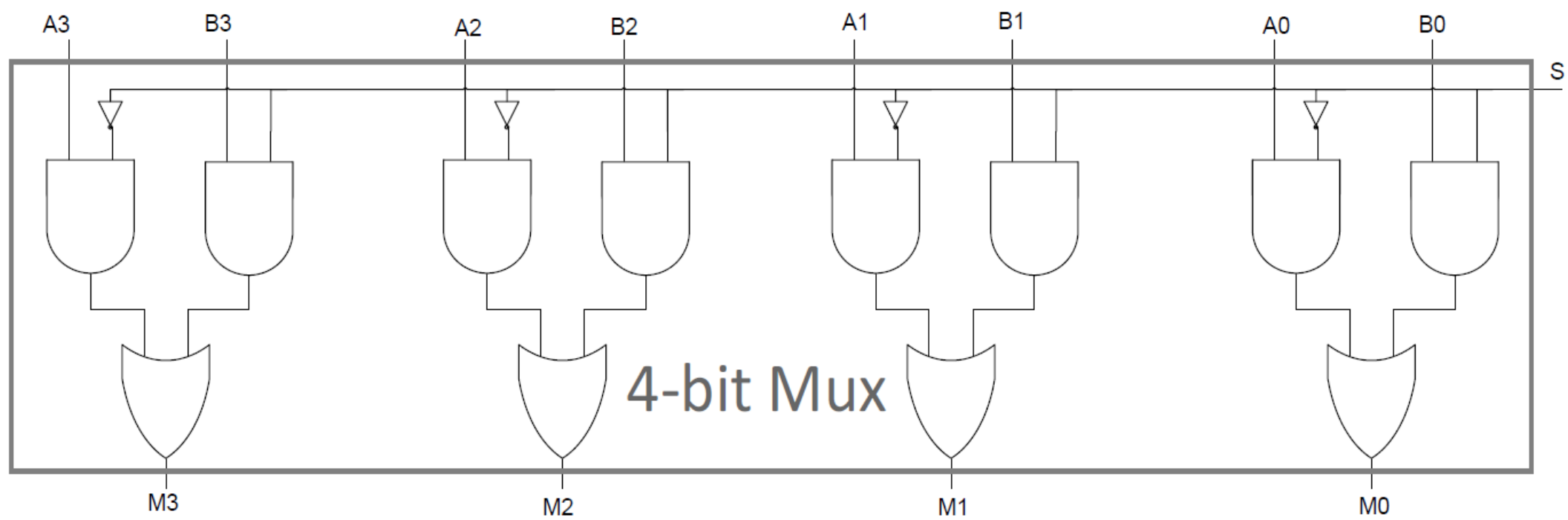




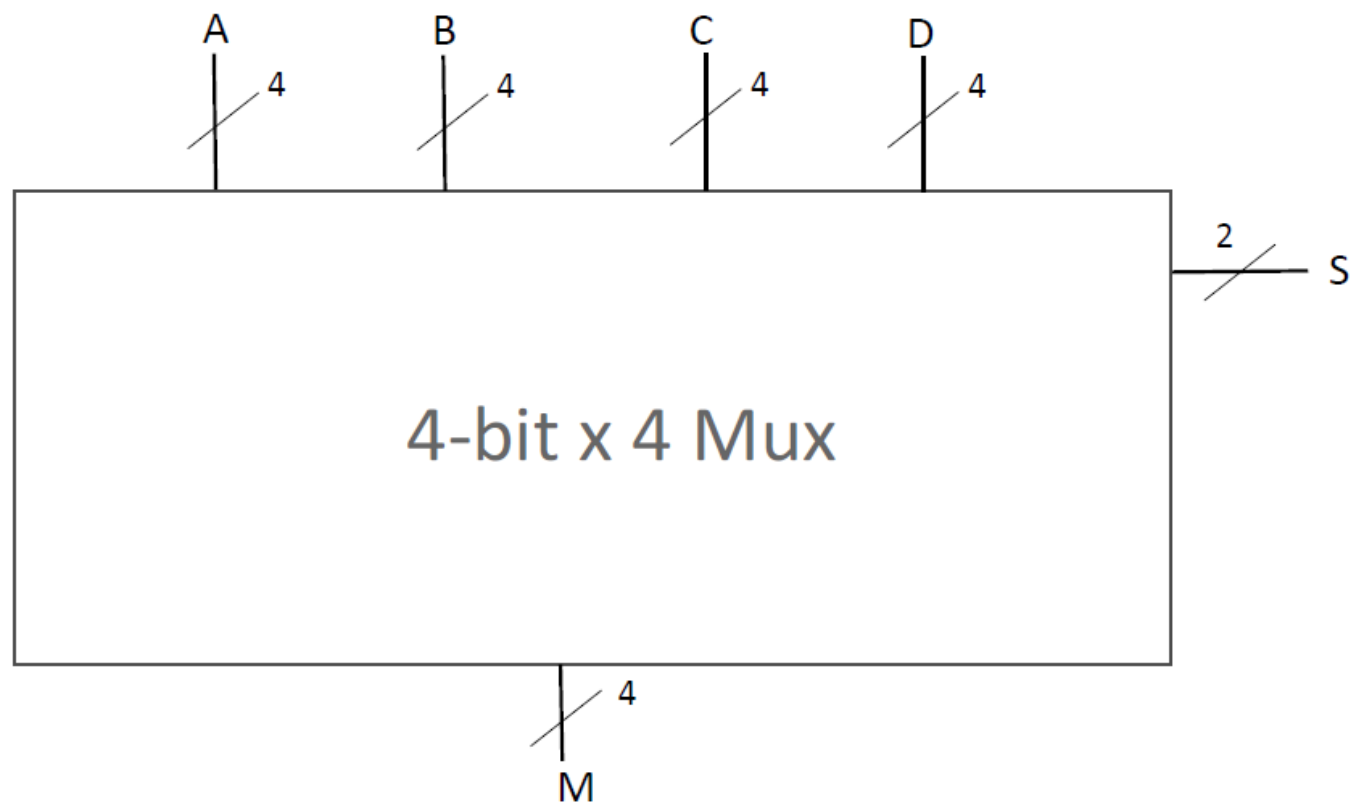


S	M0
0	A0
1	B0

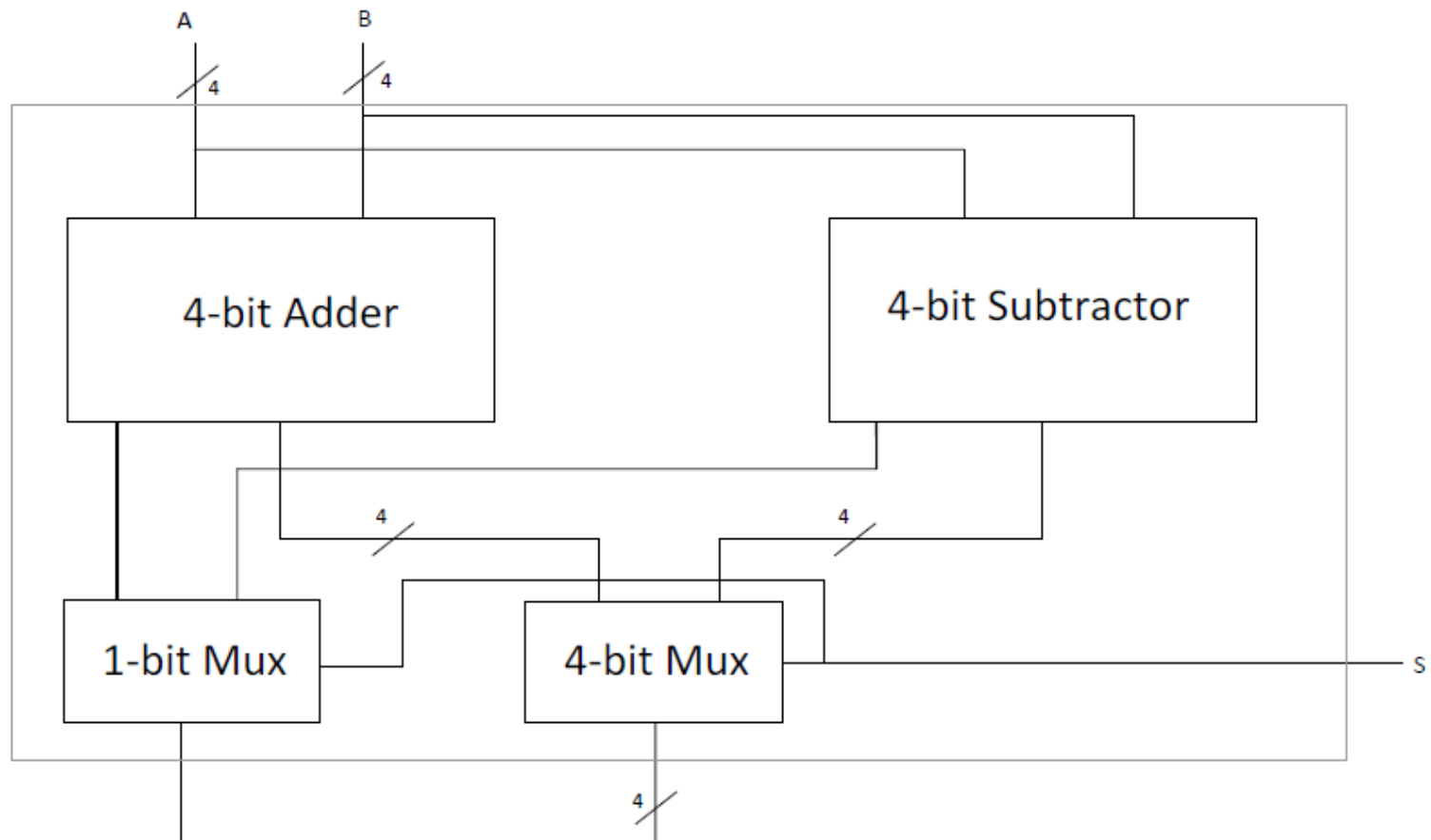




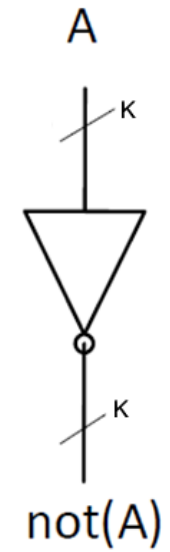
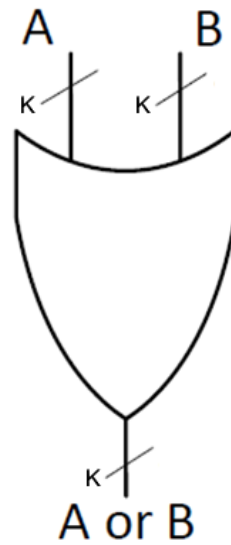
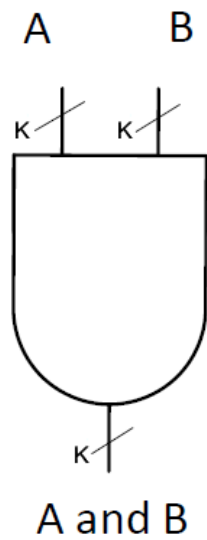
S1	S1	M
0	0	A
0	1	B
1	0	C
1	1	D



Cómo queda entonces el **sumador/restador**



## Operaciones lógicas *bitwise*



## Sólo falta definir el ¿shifting?

- Una operación muy ocupada es el **shifting** o desplazamiento, que consiste en mover todas las cifras a la izquierda o la derecha.
- Ej: `shift_left(0101) = 1010`.
- ¿Por qué es esto importante?
- En binario, desplazar a la izquierda es análogo a multiplicar por 2, mientras que desplazar a la derecha es dividir por 2.
- ¿Qué pasa acá?: `shift_left(1101) = ?`
- Si no queremos perder cifras, utilizamos **rotate**.
- Ej: `rotate_right(0101) = 1010`.

## Ya tenemos todos los elementos necesarios para construir la ALU

- Con todas las operaciones anteriores definidas, tenemos suficiente para completar una ALU de K bits con 8 operaciones.

S2	S1	S0	M
0	0	0	Suma
0	0	1	Resta
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Xor
1	1	0	Shift left
1	1	1	Shift right



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2343 – Arquitectura de Computadores

Operaciones Aritméticas y Lógicas Parte 2: Unidad Aritmética Lógica (ALU)

**Profesor:** Hans Löbel