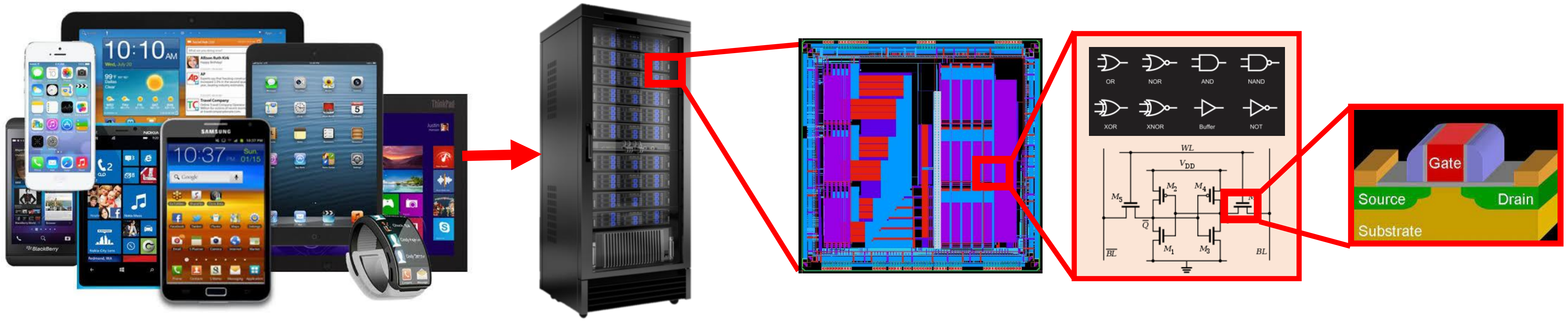
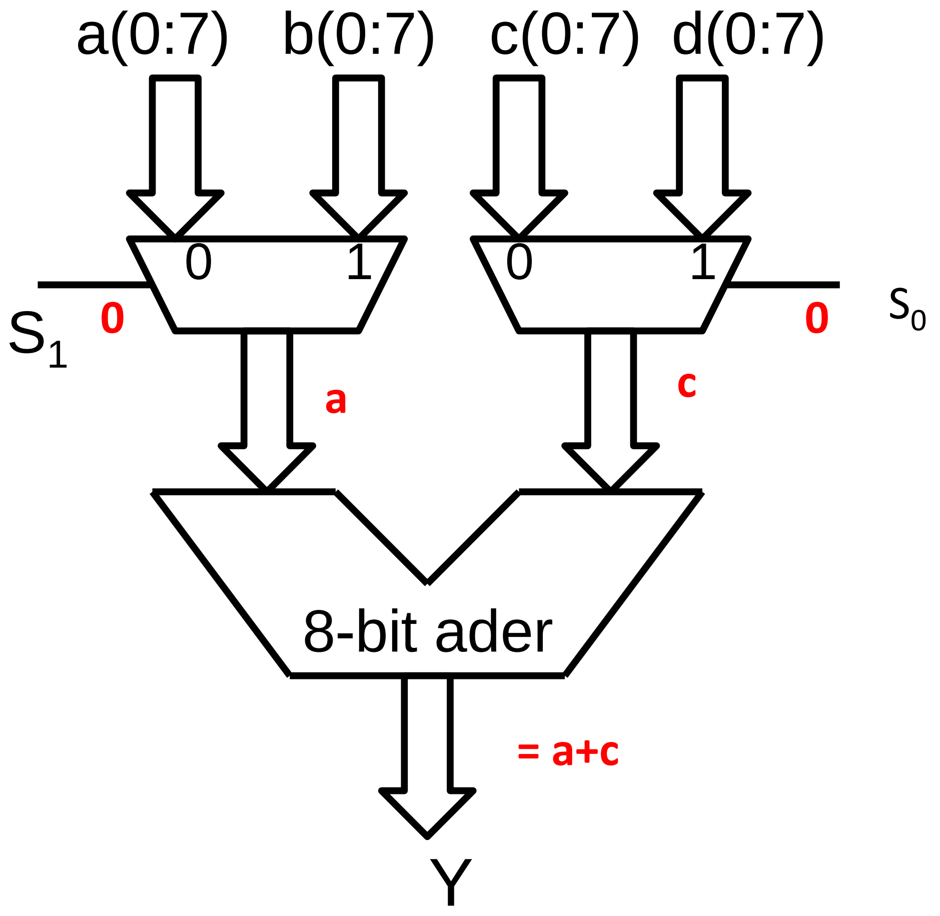


ESC201: Introduction to Electronics

Module 6: Digital Circuits

Dr. Shubham Sahay,
Associate Professor,
Department of Electrical Engineering,
IIT Kanpur

Sharing Hardware With MUX



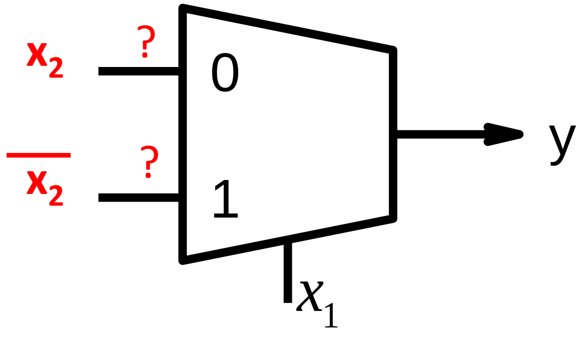
S_1	S_0	$y =$
0	0	a+c
0	1	a+d
1	0	b+c
1	1	b+d

Mux is often used when resources have to be shared

Boolean Functions with MUX

Implementing Boolean expressions using Multiplexers

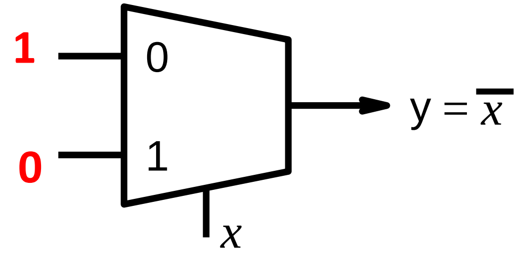
$$y = x_1 \overline{x_2} + \overline{x_1} x_2$$

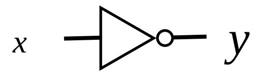


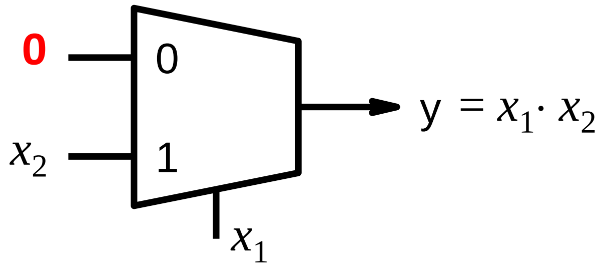
x_1	x_2	y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

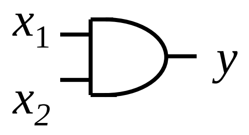
$y = x_2$ when $x_1 = 0$

$y = \overline{x_2}$ when $x_1 = 1$





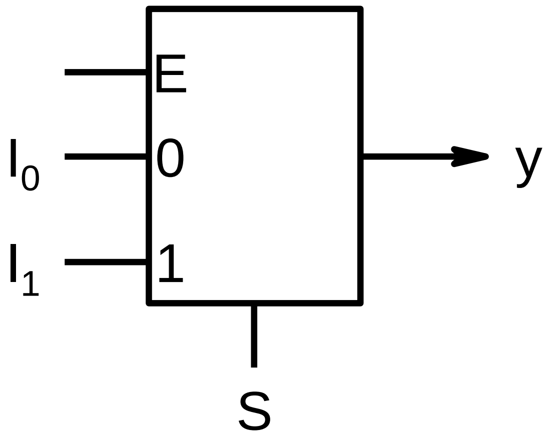




AND combined with NOT → NAND

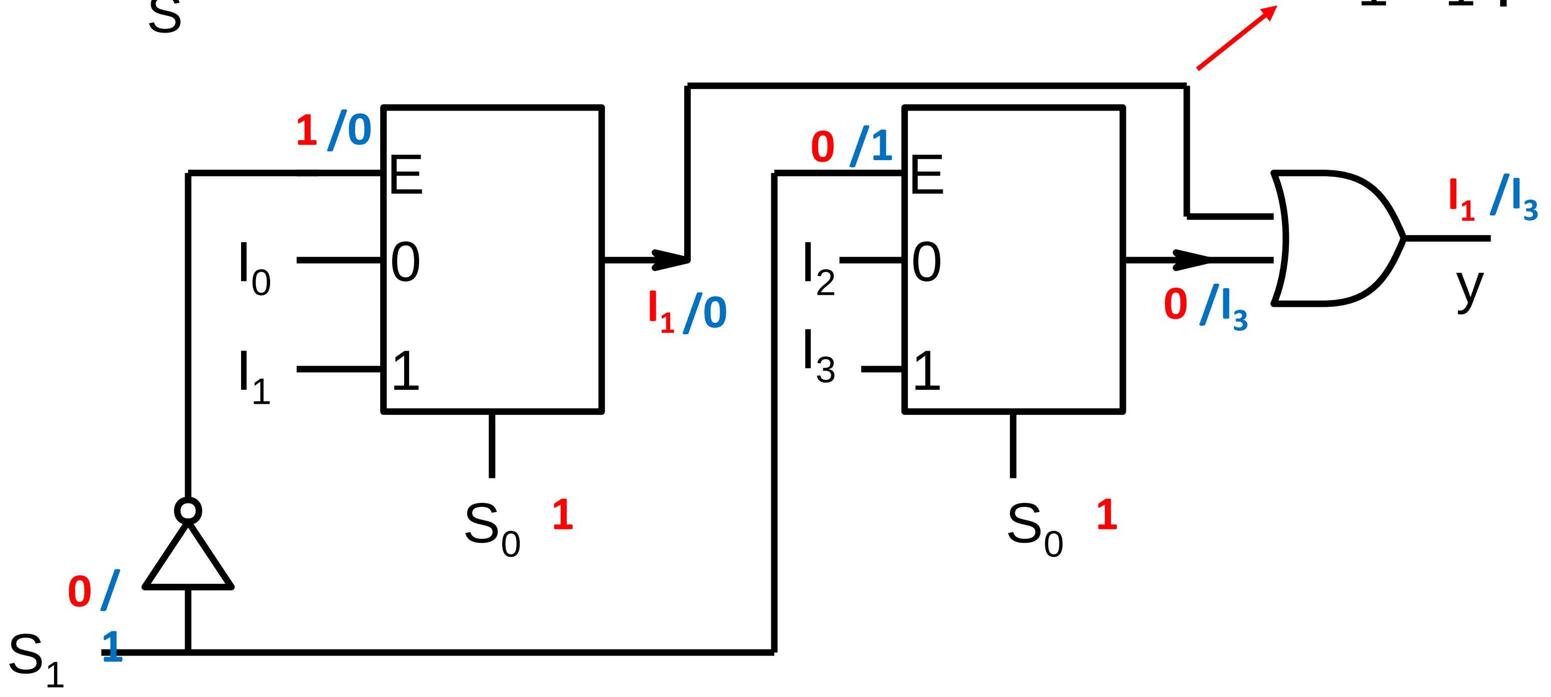
A universal gate!

Expanding MUX

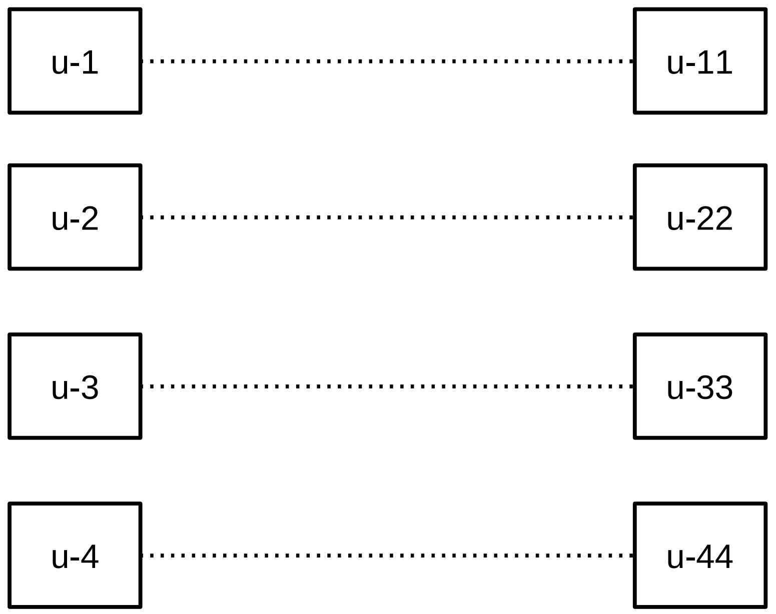


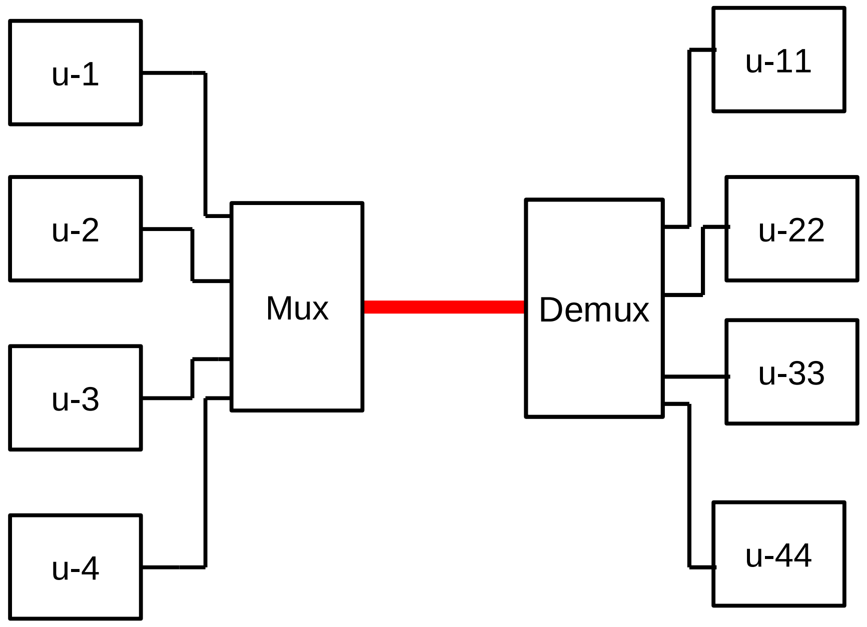
E	S	y
0	x	0
1	0	I_0
1	1	I_1

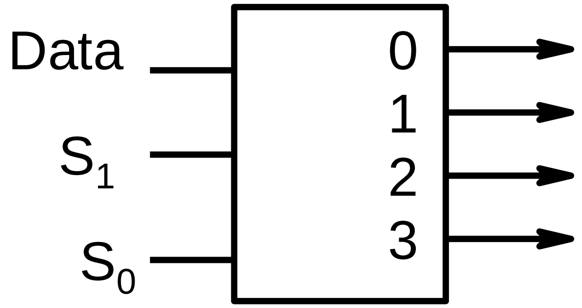
S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



DeMultiplexer



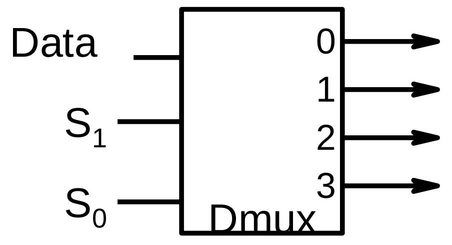




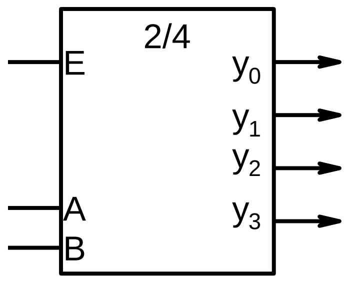
S_1	S_0	y_0	y_1	y_2	y_3
0	0	I_0	0	0	0
0	1	0	I_1	0	0
1	0	0	0	I_2	0
1	1	0	0	0	I_3

DeMUX Gate Implementation

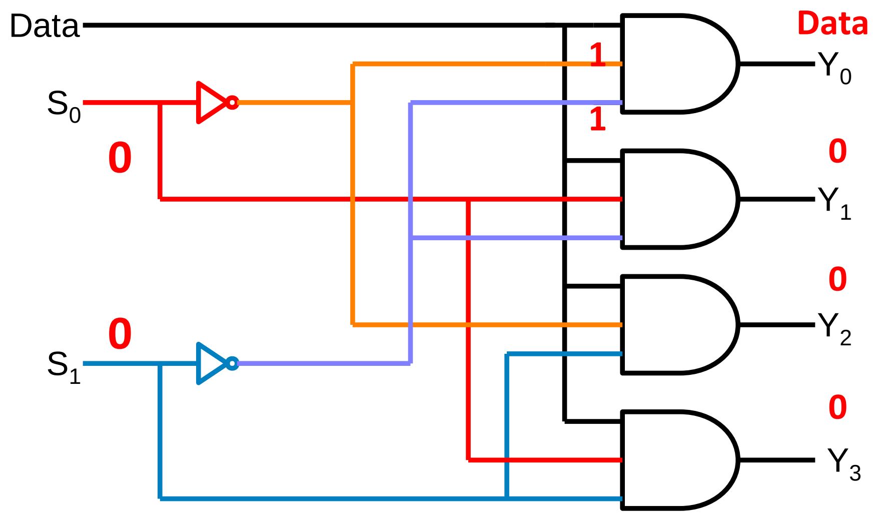
Demultiplexer is very much like a decoder

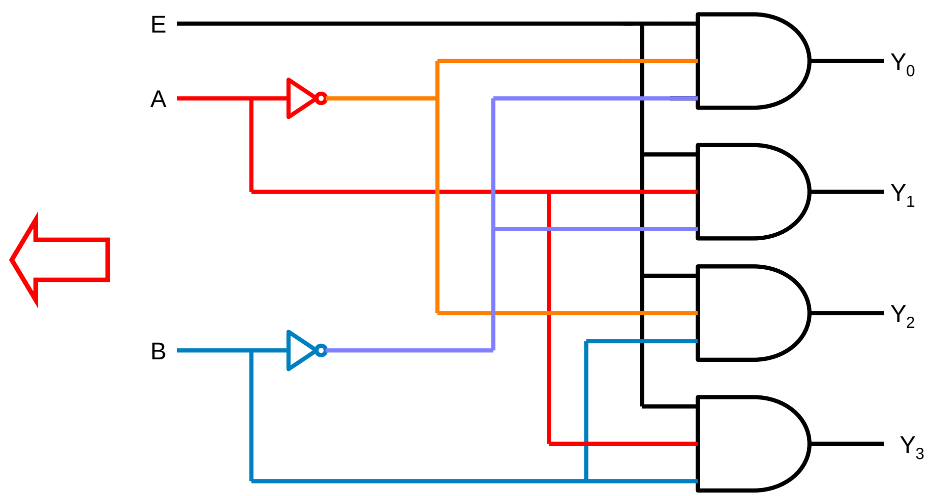


S_1	S_0	y_0	y_1	y_2	y_3
0	0	I_0	0	0	0
0	1	0	I_1	0	0
1	0	0	0	I_2	0
1	1	0	0	0	I_3



E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1





Arithmetic Addition – Half Adder

If we represent numbers in binary, the digits are either a 0 or a 1.

One may try and represent conventional addition in terms of Boolean Algebra.

Let us carry out all possible addition exercises for one bit numbers.

Addition Operation

```
  A
+ B
-----
Co S
-----
```

A: augend
B: addend
+ : Arithmetic addition symbol
S: sum
C_0: carry out

There are four possible scenarios for one digit bianry:

```
(i)    0      (ii)    0      (iii)   1      (iv)    1
     + 0            + 1            + 0            + 1
     -----          -----          -----          -----
       0              1              1             10
     -----          -----          -----          -----
```

In the last case, there is a "carry out" of 1.
We can say that "carry out" in other cases were 0.

Truth Table

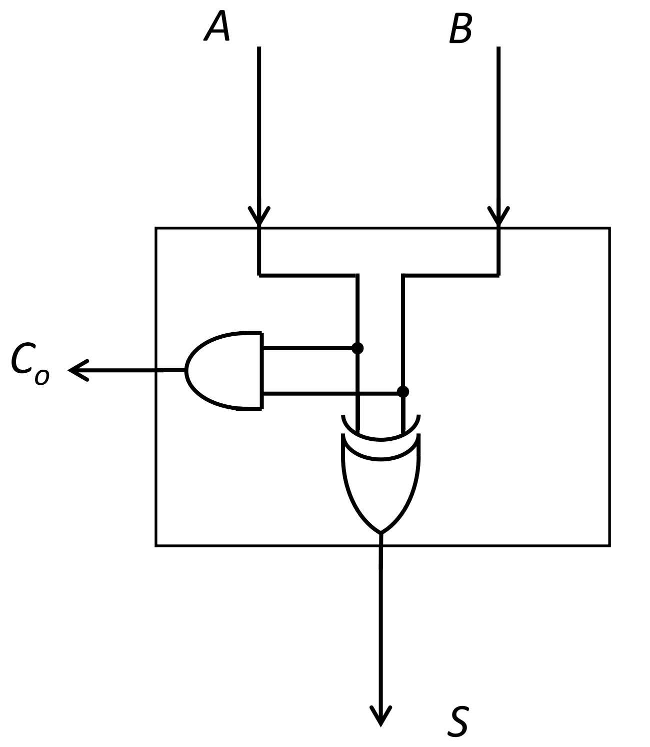
A	B	S	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum $S = A \oplus B$ → XOR

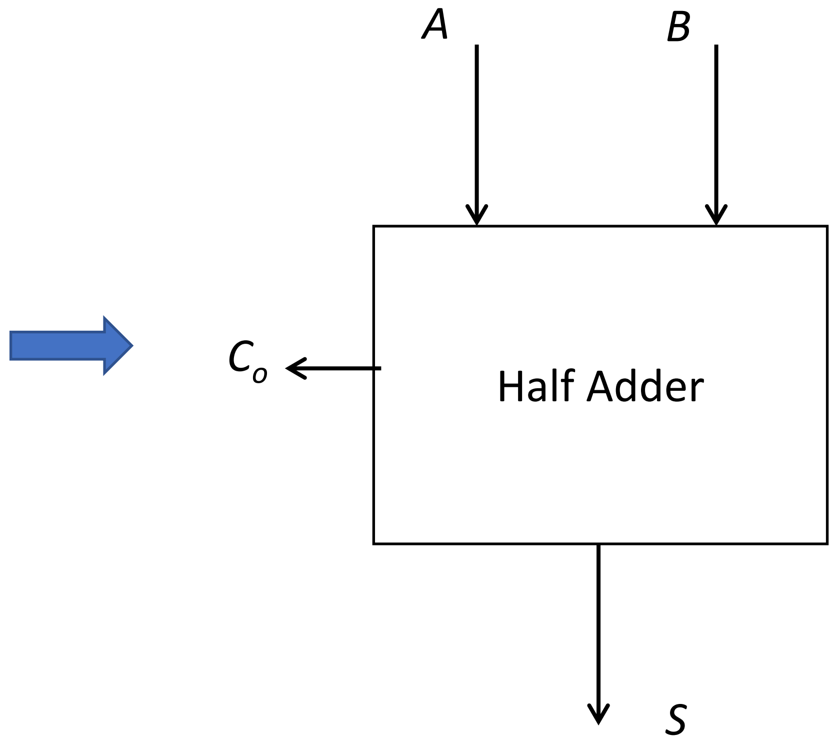
Carry out $C_o = A \cdot B$ → AND

1-bit Half Adder

Implementation



Representation



Inputs:
A (augend) and *B* (addend)

Outputs:
S (sum) and C_o (carry out)

Arithmetic Addition – Full Adder

$$\begin{array}{r} C_i \\ +\,A \\ +\,B \\ \text{-----} \\ C_o\;S \\ \text{-----} \end{array}$$

C

C_i: *Carry in*
A: augend
B: addend
+ : Arithmetic addition symbol
S: sum
C_0: carry out

C_i	A	B	S	C_o
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

Sum
$S = C_i \oplus A \oplus B$

Carry out
$C_o = C_i \cdot A + C_i \cdot B + A \cdot B$

S

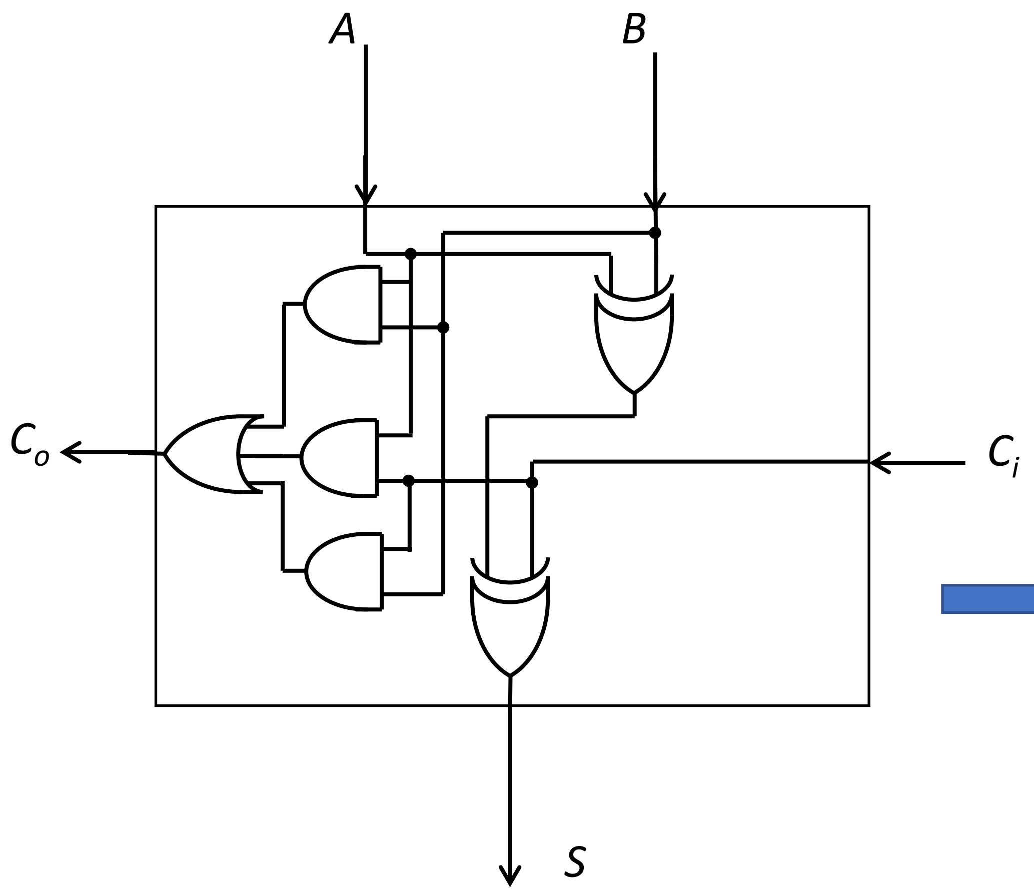
C_i \ $A\,B$	00	01	11	10
0	**0**	**1**	**0**	**1**
1	**1**	**0**	**1**	**0**

C_o

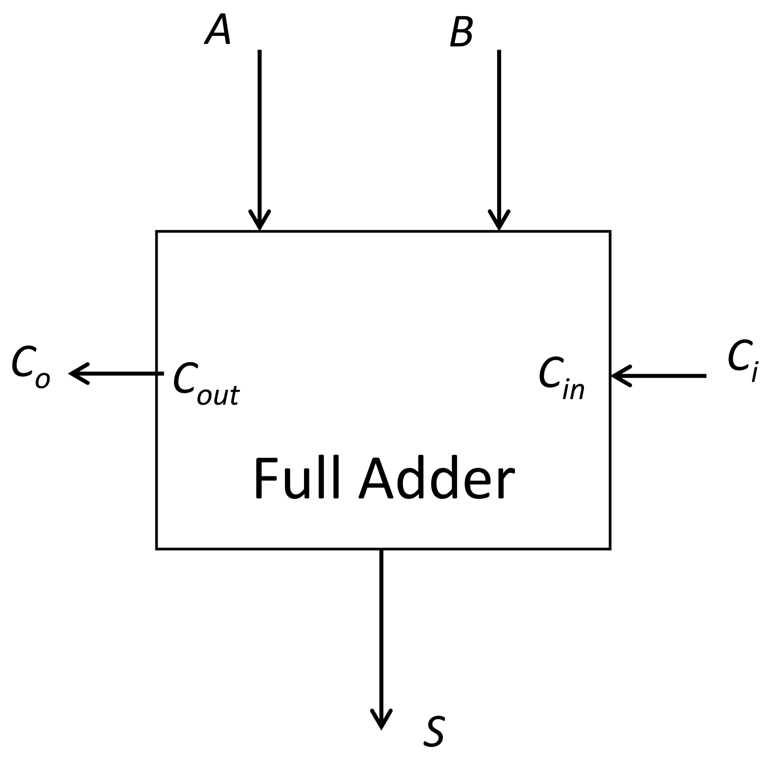
C_i \ $A\,B$	00	01	11	10
0	**0**	**0**	**1**	**0**
1	**0**	**1**	**1**	**1**

1 bit Full Adder

Implementation



Representation



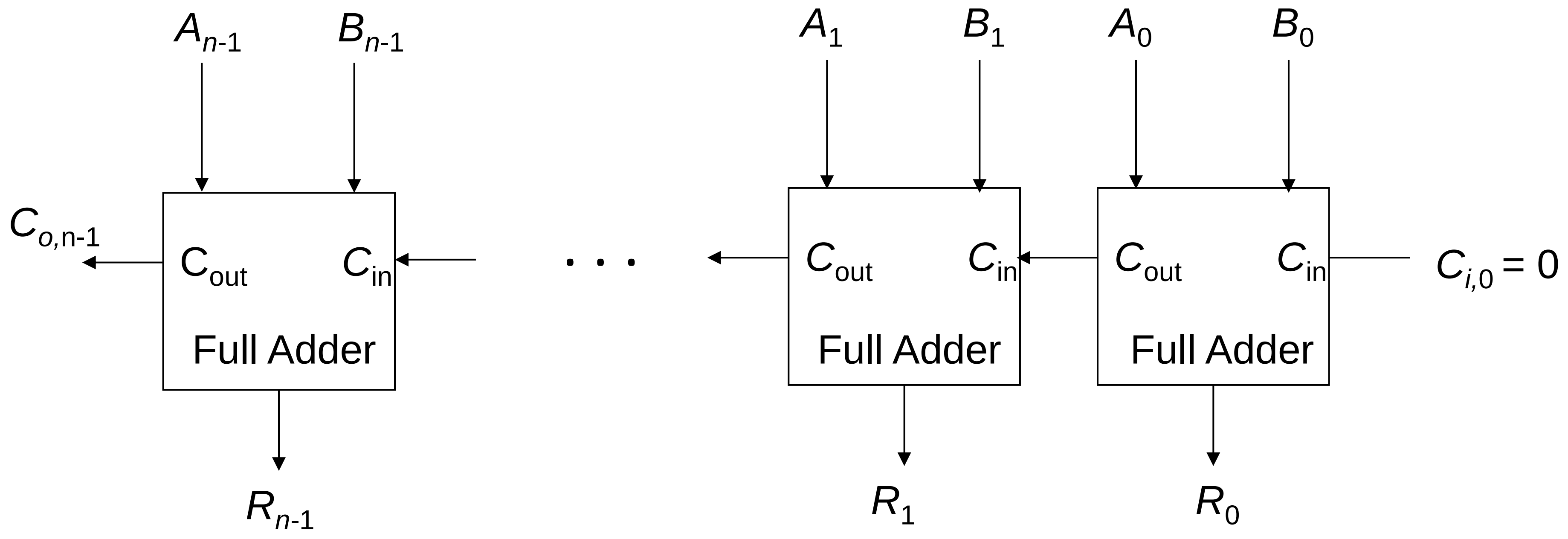
Inputs:
A (augend), B (addend) and C_i (Carry in)

Outputs:
S (sum) and C_o (carry out)

Adder for Two *n*-Bit Binary Numbers

Unsigned bits (only positive integer numbers)

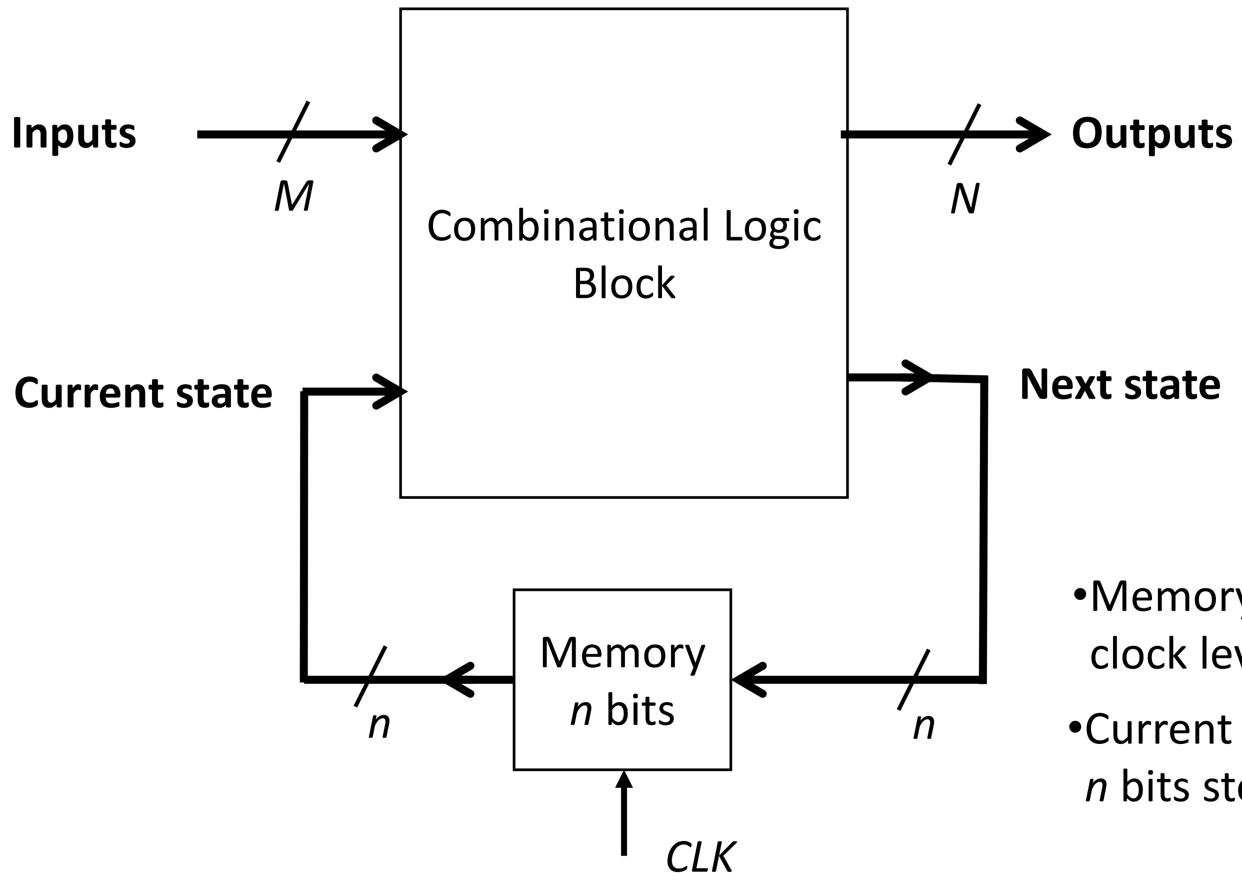
Wights of bit location →	2^{n-1} (MSB)	. . .	2^1	2^0 (LSB)



We make $C_{i,0} = 0$
For full adders of intermediate digit, for k=1 to n-2, $C_{o,k} = C_{i,k+1}$

For unsigned bit addition, overflow occurs for $C_{o,n-1} = 1$

Sequential Circuits



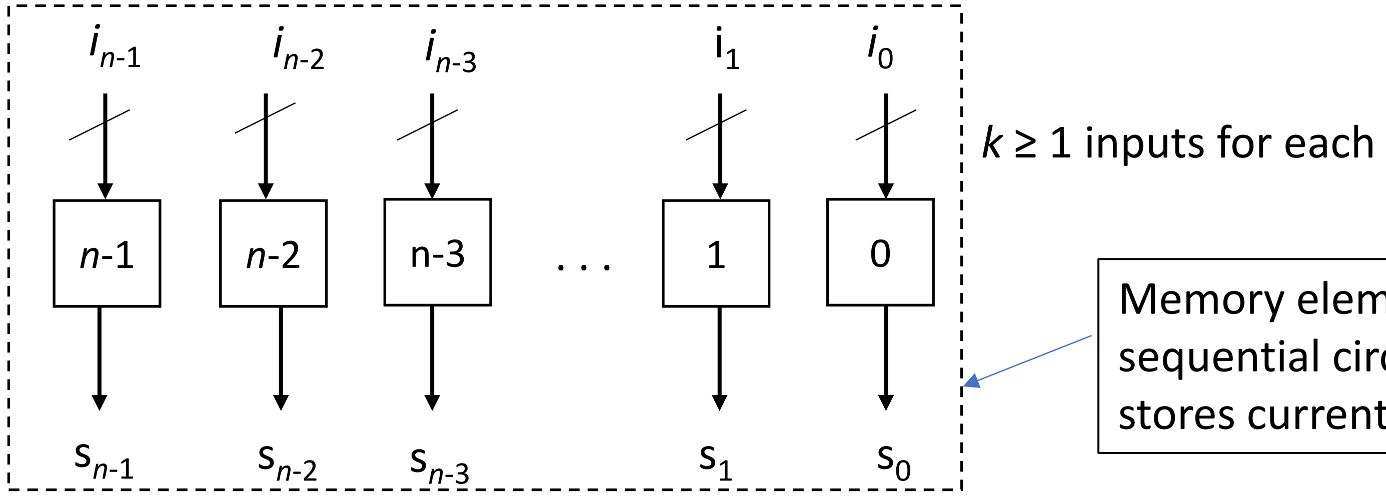
- Memory updated at clock level or clock edge
- Current state defined by *n* bits stored currently in memory

- Output will change as per current state and the input given!
- Can decide what will be the next state based on current state and inputs

  → Beginning of decision making and intelligence!

The output of a set of n latches / registers define the state-machine with 2^n states.

Depending on current state and input, inputs i^s should take state to next state



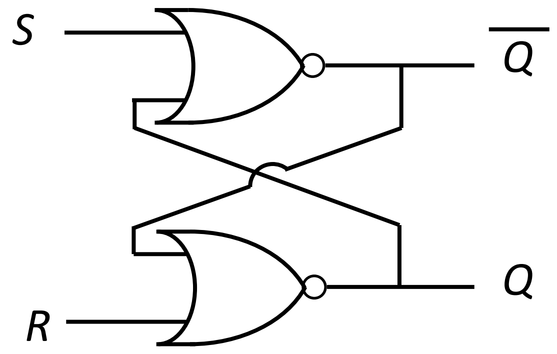
Memory element of sequential circuit that stores current state.

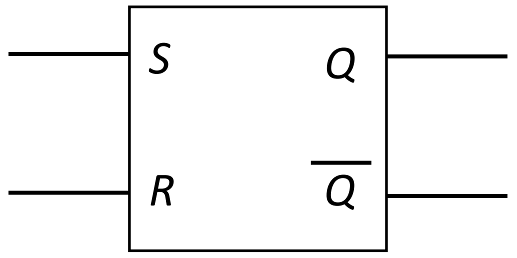
The values of output of latches/registers define the current state of the system

- Here you need Read/Write memory
- Flip-Flops are popular memory building blocks
  - Built with cross coupled gates
- One may build latches or registers using Flip-Flops
  - Can be level triggered or edge triggered
- Some popular latches / registers:
  - *SR* (set-reset), *JK*, *D* (delay), *T* (toggle), …

Static *SR* Flip-Flop* with NOR Gates

S ≡ ‘set’ latch output value *Q*; and *R* ≡ ‘reset’ latch output value *Q*





S	*R*	*Q*	$\overline{Q}$	Remark
0	0	*Q*	$\overline{Q}$	Hold
0	1	0	1	Re-set
1	0	1	0	Set
1	1	0	0	**Prohibited**

For NOR gate, Any input being “1” will make output “0”.

Input *S* = *R* = 1 is prohibited because when it comes out of that state to *S* = *R* = 0,
we do not know whether *Q* becomes 0 or 1, leading to unpredictability of future states.

If *S* = *R* = 1 state occurs, make the next state compulsorily
S = 1, *R* = 0 or *S* = 0, *R* = 1 state.

* ‘Flip-Flop’ as defined in Rabaey et al.’s book is
any bi-stable component “formed by cross-coupling of gates”