

# Multiplexers, ALU Design

**CS 64: Computer Organization and Design Logic**

**Lecture #14**

**Fall 2019**

Ziad Matni, Ph.D.

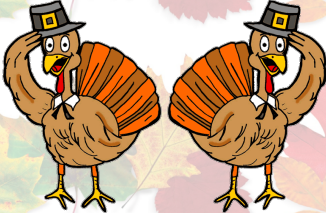

Dept. of Computer Science, UCSB

# Administrative

---

- Lab 6 due Today!
- Lab 7 will be posted today
- You have 2 more labs after this...
- This Friday, the TAs will be in the lab
  - Attendance is not mandatory, but likely very useful for you to be there
  - Work on your Lab 7 b/c it's due next Wednesday
- Next week, there's NO LAB!

# Schedule for the Rest of the Quarter

Date	Topic	Lab Due
W 11/20	Combinatorial Logic / Sequential Logic	Lab 6
M 11/25	Sequential Logic	
W 11/27	NO CLASS! 	 Lab 7
M 12/2	Finite State Machines (FSM)	
W 12/4	FSM / Ethics & Social Impact of CS	Lab 8
F 12/6	(no class)	Lab 9

**Final Exam is on Tuesday, Dec. 10<sup>th</sup> at 12:00 PM in this classroom – details will follow**

# Lecture Outline

---

- More on Combinatorial Logic
- Introduction to Sequential Logic: **R-S Latches**

# Any Questions From Last Lecture?

---

## Any Questions About the Lab?

## 5 Minute Pop Quiz! (Solution)

- Given the following K-Map for binary function **F**:

$B \backslash AC$		00	01	11	10
0	1			1	
1	1	1	1		

- a) Group properly and write the optimized function **F**

$$F = !B!C + BC + !A!C$$

- b) Draw the circuit

**See black board**

# Multiplexer

---

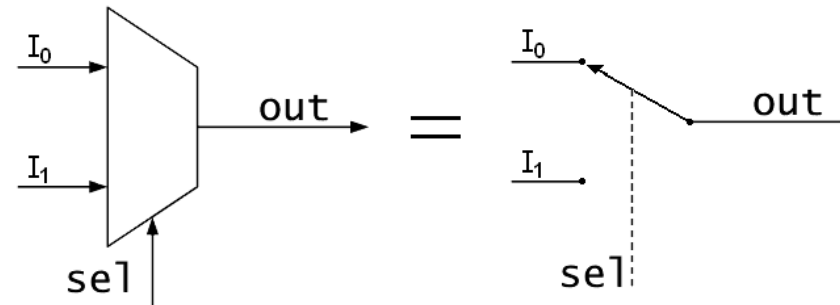
- A logical selector:
  - Select either input A or input B to be the output

```
// if s = 0, output is a
// if s = 1, output is b
int mux(int a, int b, int s)
{
    if (!s) return a;
    else return b;
}
```

# Multiplexer

(Mux for short)

- Typically has 3 *groups of* inputs and 1 output
  - IN: 2 data , 1 select
  - OUT: 1 data



- 1 of the input data lines gets selected to become the output, based on the 3<sup>rd</sup> (select) input
  - If “Sel” = 0, then  $I_0$  gets to be the output
  - If “Sel” = 1, then  $I_1$  gets to be the output
- The opposite of a Mux is called a **Demultiplexer** (or **Demux**)



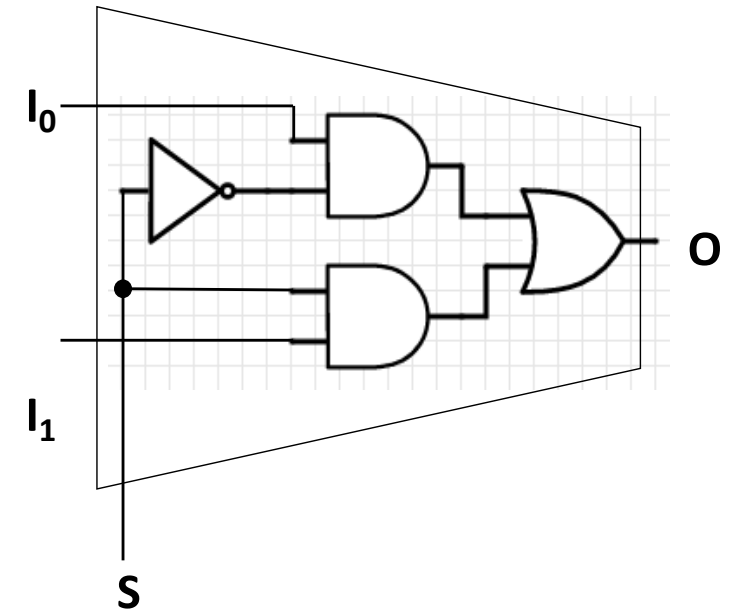
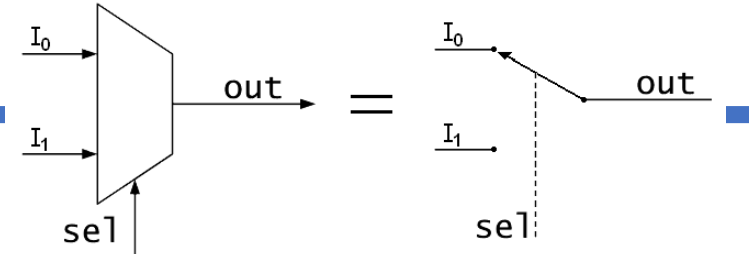
# Mux Truth Table and Logic Circuit

## 1-bit Mux

$I_0$	$I_1$	$S$	$O$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$S$	$I_0 \ I_1$			
	00	01	11	10
0			1	1
1		1	1	

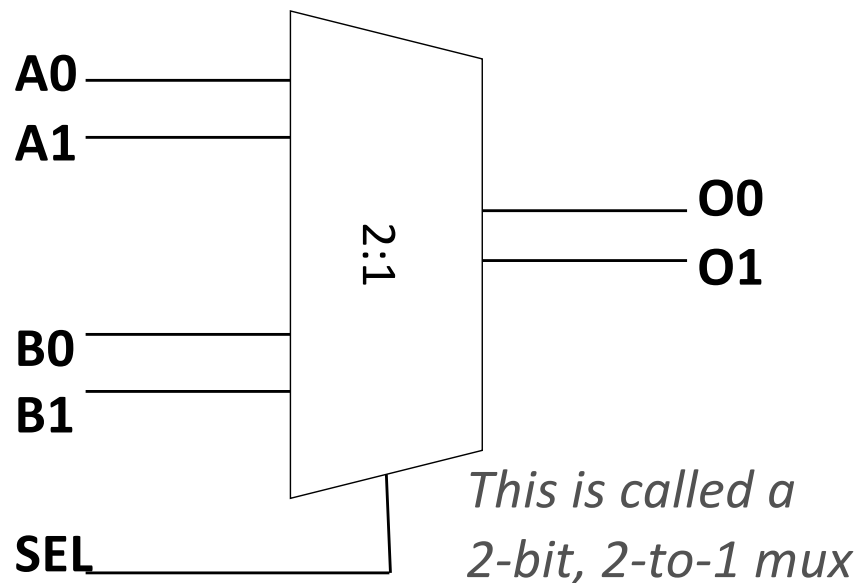
$$O = S \cdot I_1 + S' \cdot I_0$$



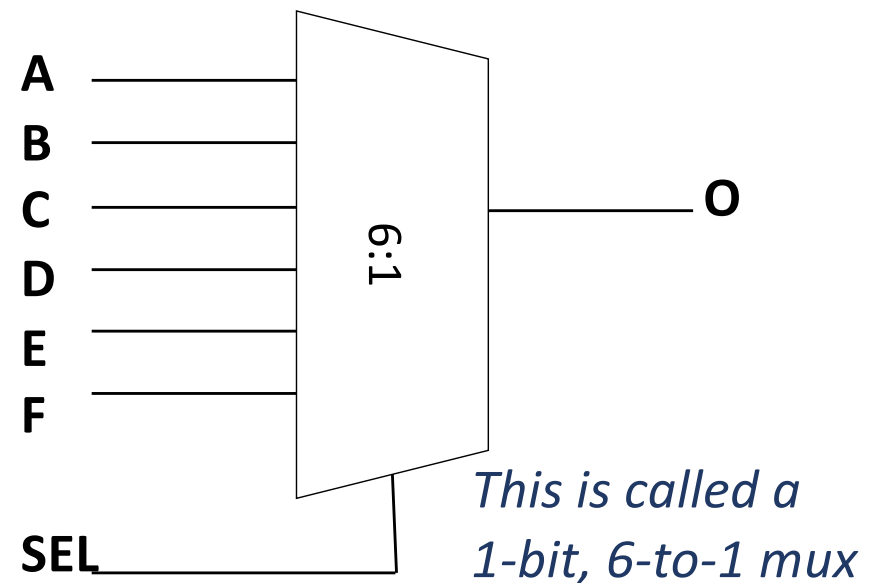
• = lines are physically connected

# Mux Configurations

Muxes can have I/O that are multiple bits

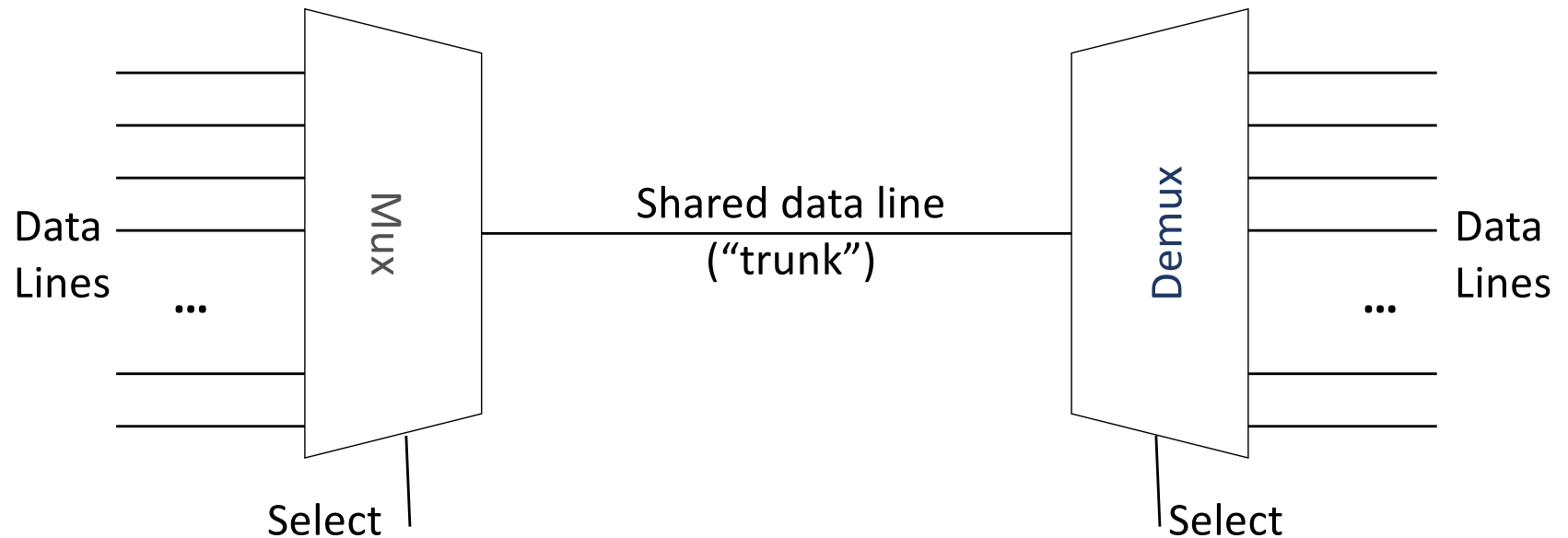


Or they can have more than two data inputs

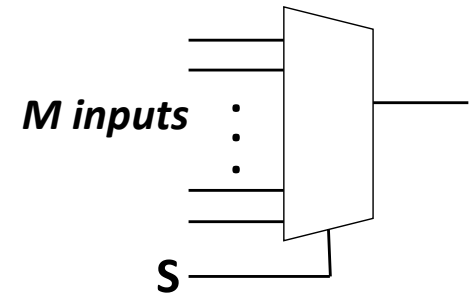


# The Use of Multiplexers

- Makes it possible for several signals (variables) to share one resource
  - Very commonly used in data communication lines



# Selection Lines in Muxes



- General mux description: **N-bit, M-to-1**
- Where:            N = how “wide” the input is (# of input bits, min. 1)  
                      M = how many inputs to the mux (min. 2)
- The “select” input (S) has to be able to select **1 out of M inputs**
  - So, if  $M = 2$ ,    S should be at least 1 bit    *(S = 0 for one line, S = 1 for the other)*
  - But if  $M = 3$ ,    S should be at least **2 bits**    *(why?)*
  - If  $M = 4$ ,                            S should be ???    *(ANS: at least 2 bits)*
  - If  $M = 5$ ,                            S should be ???    *(ANS: at least 3 bits)*

# Combining Muxes Together

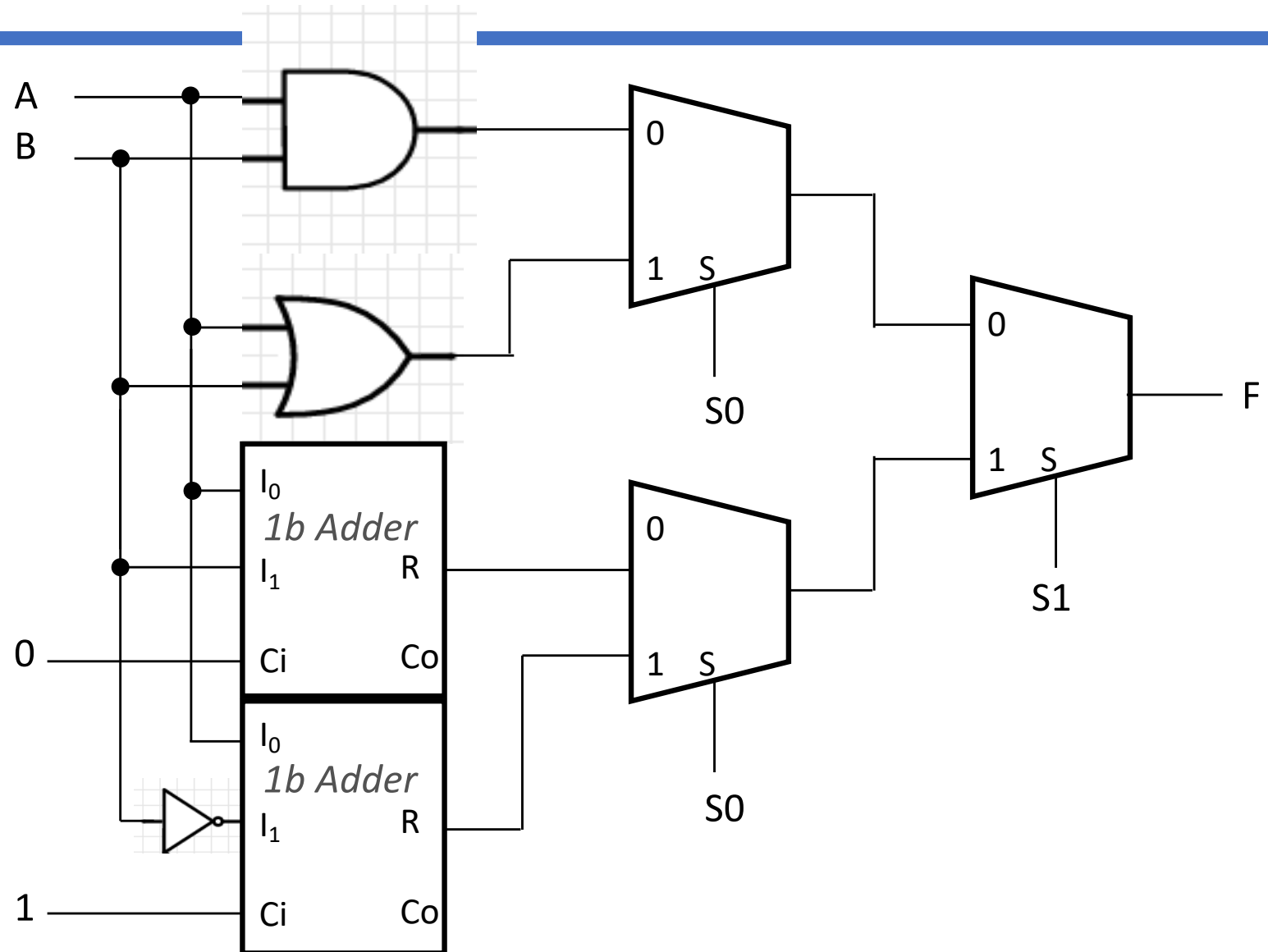
---

Can I do a **4:1** mux from 2:1 muxes?

Generally, you can do  **$2^n:1$**  muxes from 2:1 muxes.

# What Does This Circuit Do?

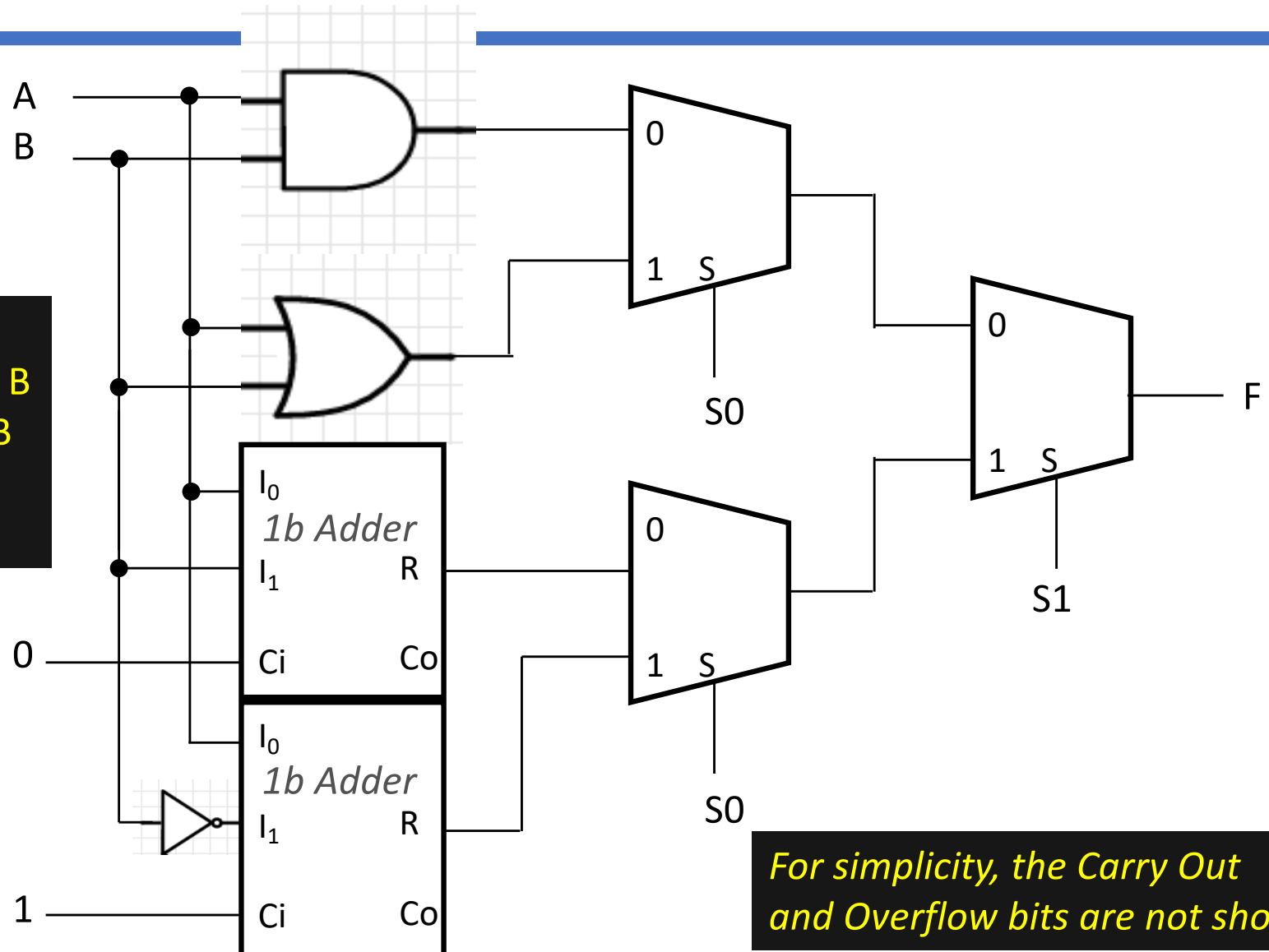
Class Ex.



# Class Ex.

## What Does This Circuit Do?

<u>S1</u>	<u>S0</u>	<u>F</u>
0	0	A && B
0	1	A    B
1	0	A + B
1	1	A - B



*For simplicity, the Carry Out and Overflow bits are not shown*

### Input Controls



Toggle Switch



Push Button



Clock



High Constant



Low Constant

### Output Controls



Light Bulb



Digit

### Logic Gates



Buffer



NOT Gate



AND Gate



NAND Gate



OR Gate



NOR Gate

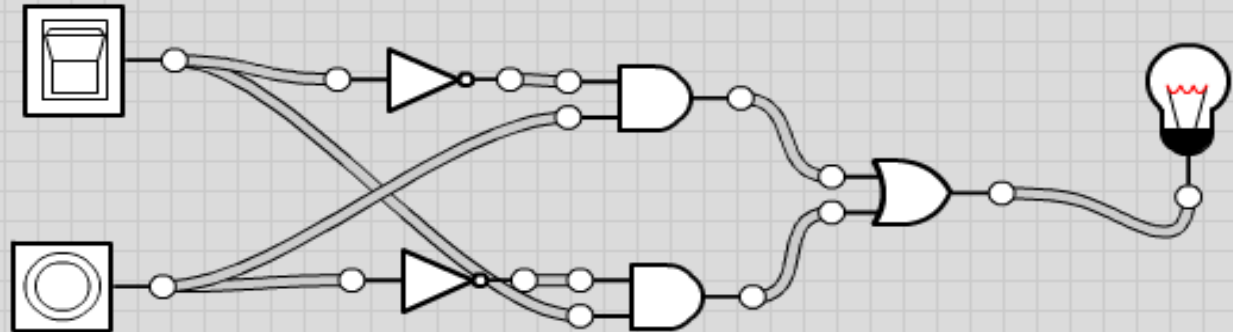


XOR Gate



XNOR Gate

# Simulation of Combinatorial Logic



- Go to:

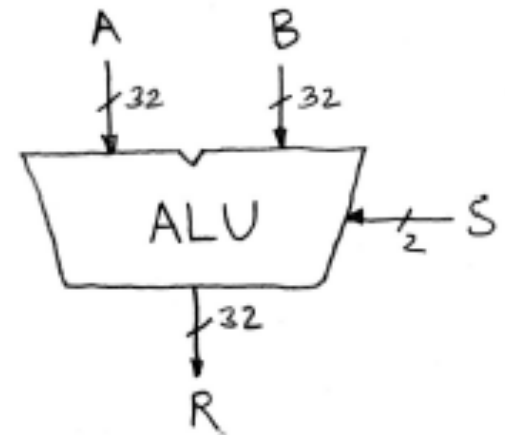
<https://logic.ly/demo/>

**IN-CLASS DEMONSTRATION**



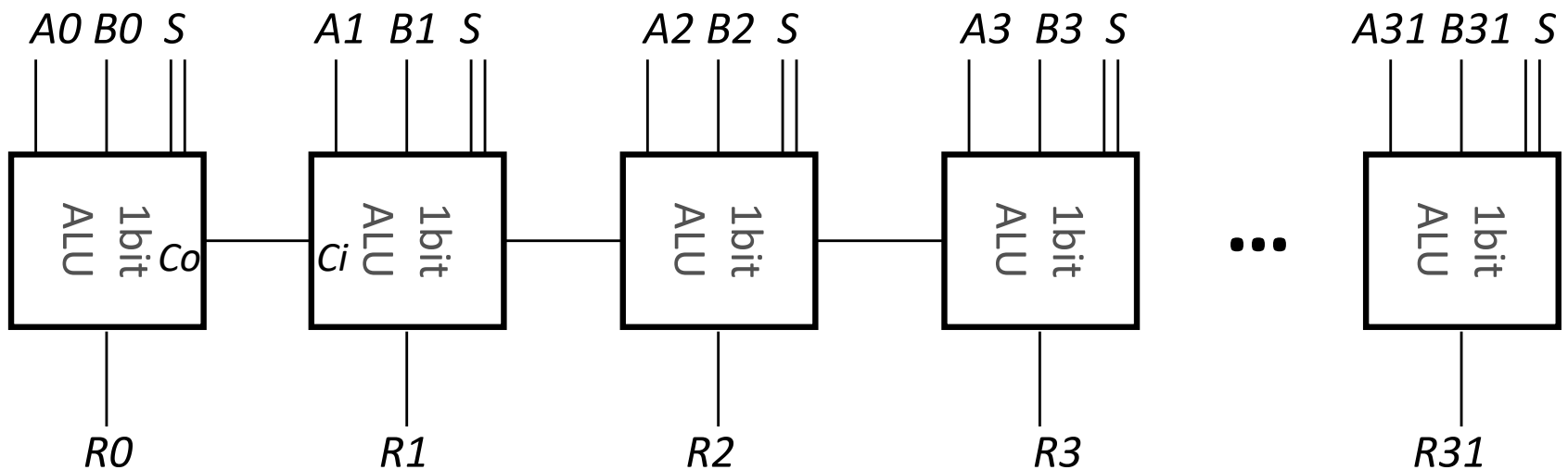
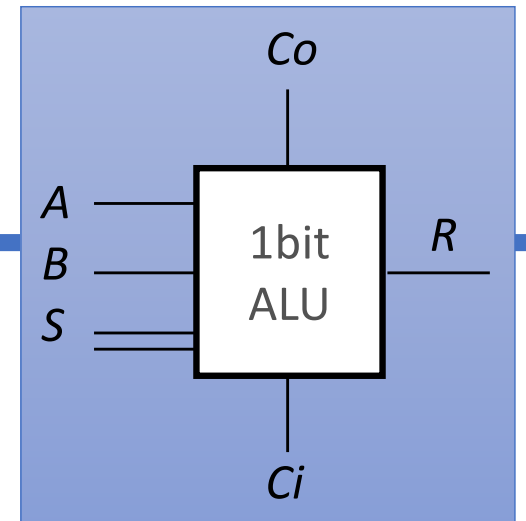
# Arithmetic-Logic Unit (ALU)

- Recall: the ALU does all the computations necessary in a CPU
- The previous circuit was a simplified ALU:
  - When  $S = 00$ ,  $R = A + B$
  - When  $S = 01$ ,  $R = A - B$
  - When  $S = 10$ ,  $R = A \text{ AND } B$
  - When  $S = 11$ ,  $R = A \text{ OR } B$



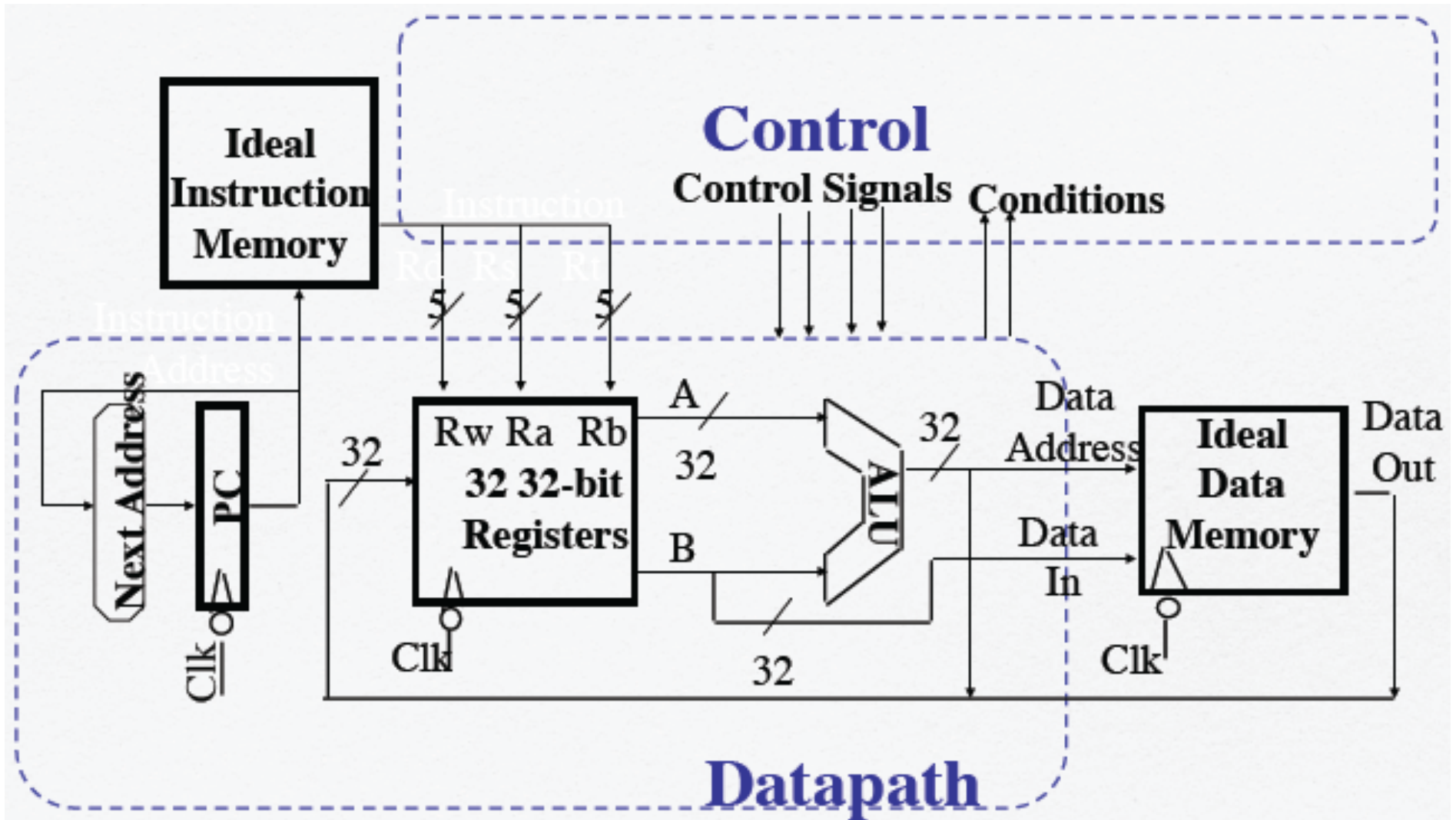
# Simplified ALU

- We can string 1-bit ALUs together to make bigger-bit ALUs (e.g. 32b ALU)



# Abstract Schematic of the MIPS CPU

*Relevant to a future lab...*



# Combinatorial vs. Sequential Logic

- The CPU schematic shows  
*both **combinatorial** and **sequential*** logic blocks

- **Combinatorial Logic**

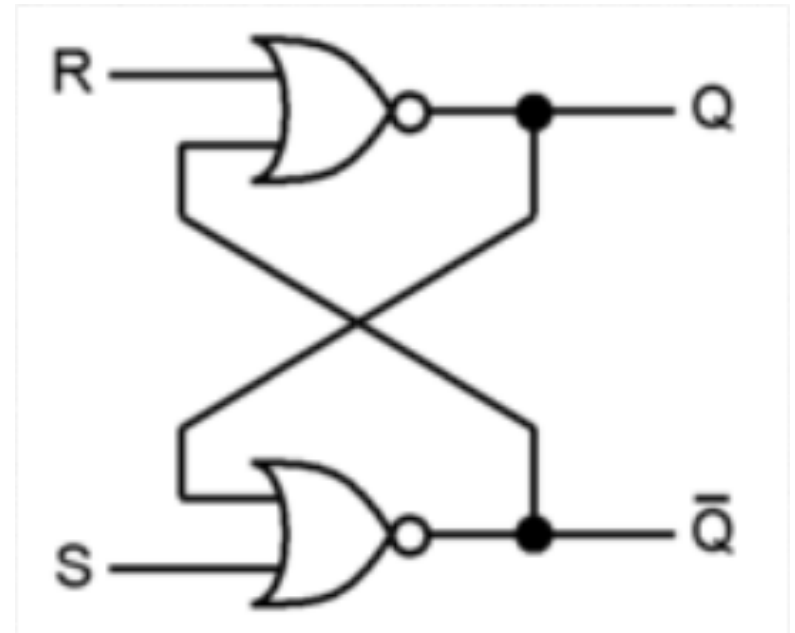
- Combining multiple logic blocks
- The output is a function **only** of the present inputs
- There is no memory of past “states”

- **Sequential Logic**

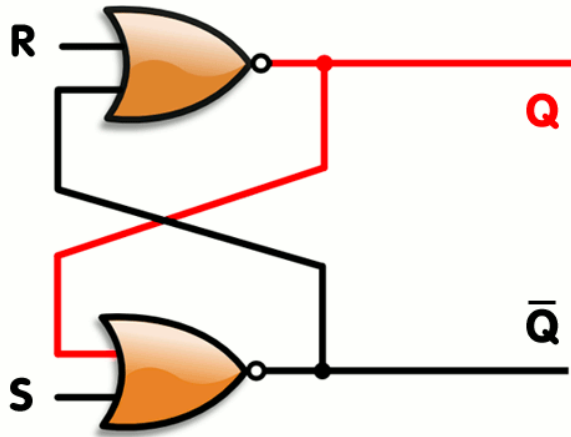
- Combining multiple logic blocks
- The output is a function of **both** the present inputs and **past** inputs
- There exists a memory of past “states”

# The S-R Latch

- Only involves 2 NORs
- The outputs are ***fed-back*** to the inputs
- The result is that the output state (either a 1 or a 0) is ***maintained*** *even if the input changes!*



# How a S-R Latch Works



- Note that if one NOR input is **0**, the output becomes the inverse of the other input
- So, if output Q already exists and if **S = 0, R = 0**, then Q will remain at whatever it was before! (**hold output state**)
- If **S = 0, R = 1**, then Q becomes 0 (**reset output**)
- If **S = 1, R = 0**, then Q becomes 1 (**set output**)
- Making S = 1, R = 1 is not allowed (**gives an undetermined output**)

S	R	Q <sub>0</sub>	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

# YOUR TO-DOs

---

- Are you doing your readings??
- Go to Friday lab (we won't take attendance)
- Work on Lab 7, which is due **next Wednesday**

**</LECTURE>**