Simplification & Combinational Logic

Introduction to Computer Systems, Fall 2024

Instructors: Joel Ramirez Travis McGaha

Head TAs: Adam Gorka Daniel Gearhardt
Ash Fujiyama Emily Shen

TAs:

Ahmed Abdellah
Angie Cao
August Fu
Caroline Begg
Cathy Cao
Claire Lu
Eric Sungwon Lee
Ethan Weisberg
Garrett O'Malley Kirsch
Hassan Rizwan
Iain Li
Jerry Wang
Juan Lopez
Keith Mathe
Maya Huizar
Meghana Vasireddy
Perrie Quek
Sidharth Roy
Sydnie-Shea Cohen
Vivi Li
Yousef AlRabiah

pollev.com/tqm

- How are you? Any Questions from last lecture?

Upcoming Due Dates

- HW03 (RPN):
  - Due Friday

- HW04 will release on Friday, will be due before Fall break.
  - **THIS IS A WRITTEN HW, AT MAX 72 HOURS LATE**
  - It should be pretty short.
  - We want to give you some practice on hardware that we are sure we can get graded and back to you before the midterm.
  - Will try to get HW05 back to you before midterm as well, but aren't certain about it.

- Lecture check-in posted soon (tonight or tomorrow)

Lecture Outline

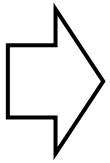
- **PLAs & Simplification**
- Incrementor
- Adder & Subtracter
- Mux
- Multiplier & Others

PLA's

- What if we only had a truth table to create a gate circuit?

- PLA: **P**rogrammable **L**ogic **A**rray
  - A device where we can configure AND, OR and NOT gates to implement a function

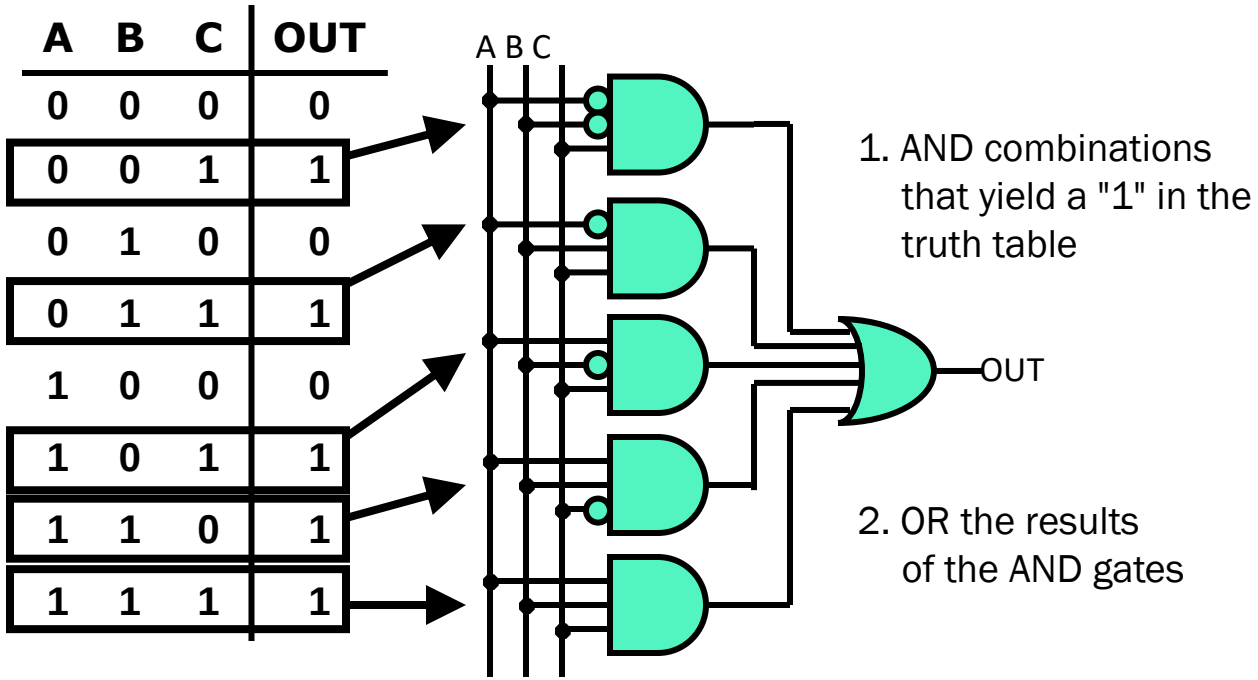
A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



GATES

Implementing a PLA From a Truth Table

- NOT, AND, OR can implement any truth table function

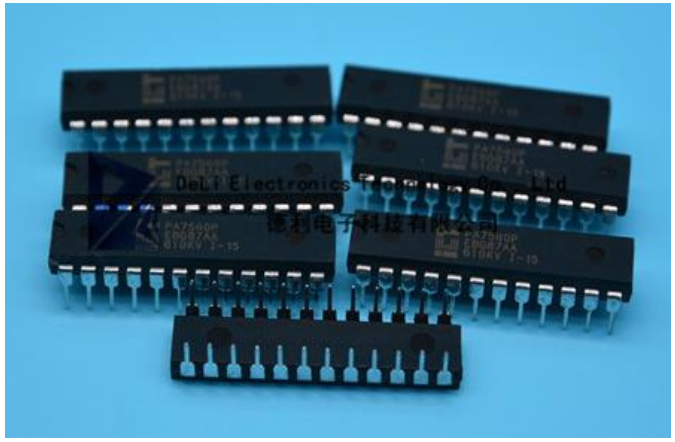


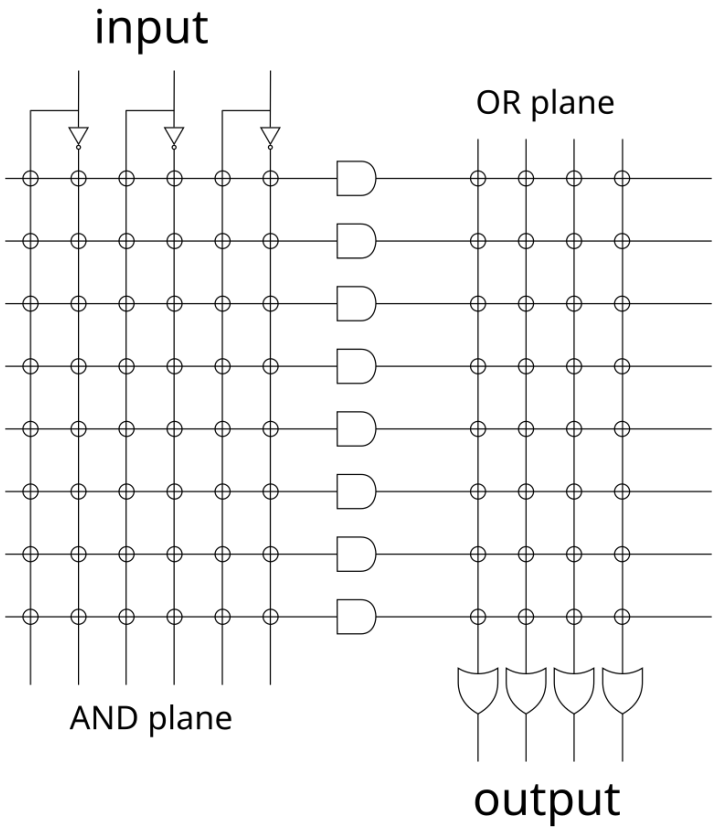
IN a PLA, this structure is always followed →

Notice, 5 rows that cause a "1" in the output…5 AND gates
Notice, 1 output, only 1 OR gate
Notice, negations always happen before the AND gates

Why This Format?

- PLA's are already manufactured chips that you can buy and then "program" to behave how you like



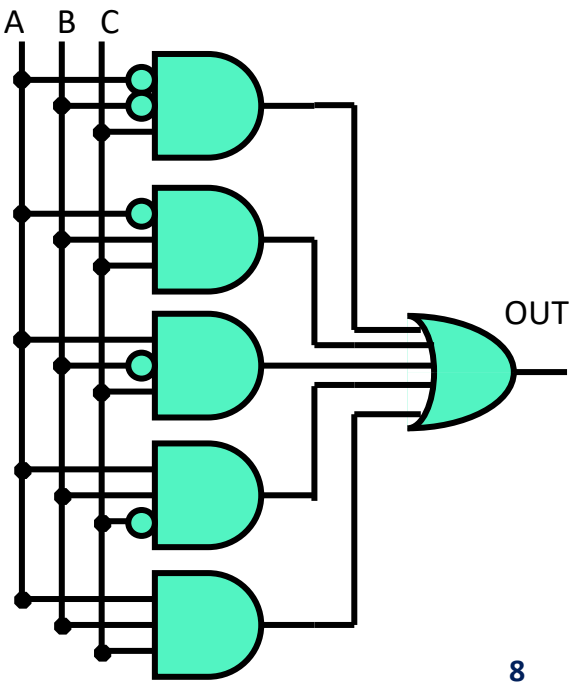


By Ilia Kr. - Own work. Created using Inkscape, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=22305322

PLA to Boolean expression

pollev.com/tqm

- Given this PLA, we can convert it to a Boolean expression
  - (~A & ~B & C) | (~A & B & C) | (A & ~B & C) | (A & B & ~C) | (A & B & C)
  - **(C & ((~A & ~B) | (~A & B) | (~A & B) | (A & B)))** | (A & B & ~C)
    - // distributive property
    - // TODO: Simplify the rest, what do you get?



PLA to Boolean expression

- Given this PLA, we can convert it to a Boolean expression
  - (~A & ~B & C) | (~A & B & C) | (A & ~B & C) | (A & B & ~C) | (A & B & C)
  - **(C & ((~A & ~B) | (~A & B) | (~A & B) | (A & B)))**| (A & B & ~C)
    - // distributive property
  - (C & **1**) | (A & B & ~C)
    - // a lot of identity properties that were omitted for space
  - C | (A & B & ~C) *// Identity*
  - (C | A) & (C | B) & (C | ~C) *// Distributive*
  - (C | A) & (C | B) & 1 *// Identity*
  - (C | A) & (C | B) *// identity*
  - C | (A & B) *// distributive*

PLAs Pros & Cons

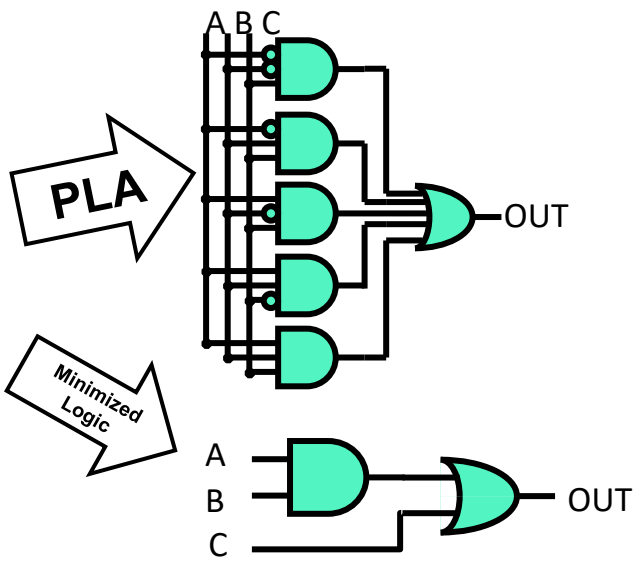
- A PLA can be used to implement ANY logical function
  - Provides you with an incredibly easy tool to use
  - If you can generate a truth table to model desired behavior
    - PLA gives you a way generate the gate level implementation
  - *However, PLAs don't give the most efficient solution*
    - *In terms of "run-time" and transistor cost*

Logic Function

F=(A AND B) OR C

Truth Table

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

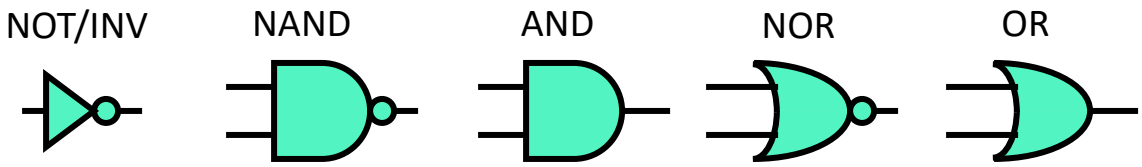


Lecture Outline

- PLAs & Simplification
- **Incrementor**
- Adder & Subtracter
- Mux
- Multiplier & Others

Combinational Logic

- Boolean functions where the output is a pure function of the inputs
  - There is no “memory” or “stored state”
- So far, we have basic logic gates from last lecture:



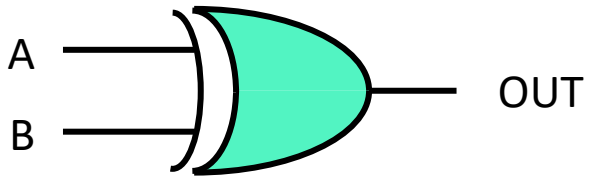
- We can build more complex "gates" that we can use as building blocks for a processor

- This Lecture: start implementing binary arithmetic >:]

Aside: XOR Gate

- Performs the XOR operation

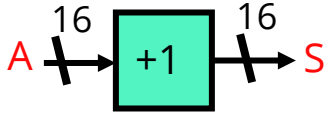
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0



Creating an Incrementor

- Let's create a 16-bit incrementor!
  - Input: A (as a 16 bit 2C integer)
  - Output: S = A + 1 (as a 16-bit 2C integer)
  - Ignore the overflow case for now

```
 0000000011001011
+0000000000000001
 0000000011001100
```



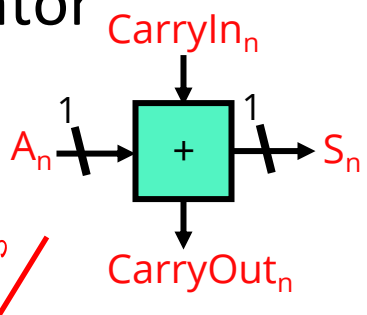
- Theoretical Approach:
  - Use a PLA-like technique to implement the circuit
  - Problem: 2^{16} or 65536 different inputs, 16-bit output
  - This is impractical

One Bit Incrementor "PLA"

- Implementing a single-column of an incrementor

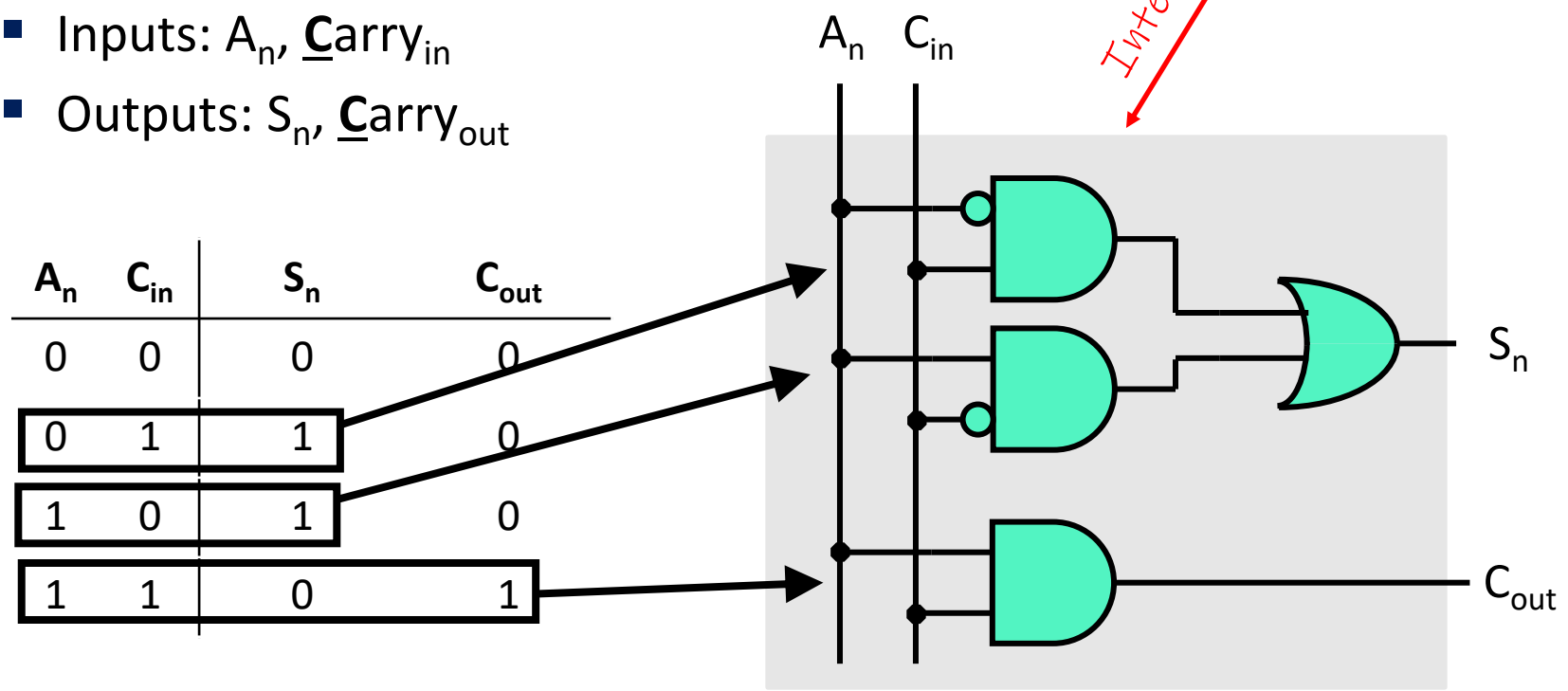
```
 0000000011001011
+0000000000000001
 0000000011001100
```

(Ignore LSB for now)



- Inputs: A_n, **C**arry$_{in}$
- Outputs: S_n, **C**arry$_{out}$

A_n	C_{in}	S_n	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Poll Everywhere

pollev.com/tqm

- Which of the follow is an equivalent expression for S_n?

A. $(A_n \ \& \ \sim C_{in}) \ \& \ (\sim A_n \ \& \ C_{in})$

B. $(A_n \mid \sim C_{in}) \ \& \ (\sim A_n \mid C_{in})$

C. $\sim(C_{in} \ \hat{} \ A_n)$

D. $A_n \ \hat{} \ C_{in}$

E. I'm not sure

^ is xor

A_n	C_{in}	S_n
0	0	0
0	1	1
1	0	1
1	1	0

Poll Everywhere

pollev.com/tqm

- Which of the follow is an equivalent expression for S_n?

A. $(A_n \;\&\; \sim C_{in}) \;\&\; (\sim A_n \;\&\; C_{in})$

B. $(A_n \mid \sim C_{in}) \;\&\; (\sim A_n \mid C_{in})$

C. $\sim(C_{in} \;\hat{}\; A_n)$

D. $A_n \;\hat{}\; C_{in}$

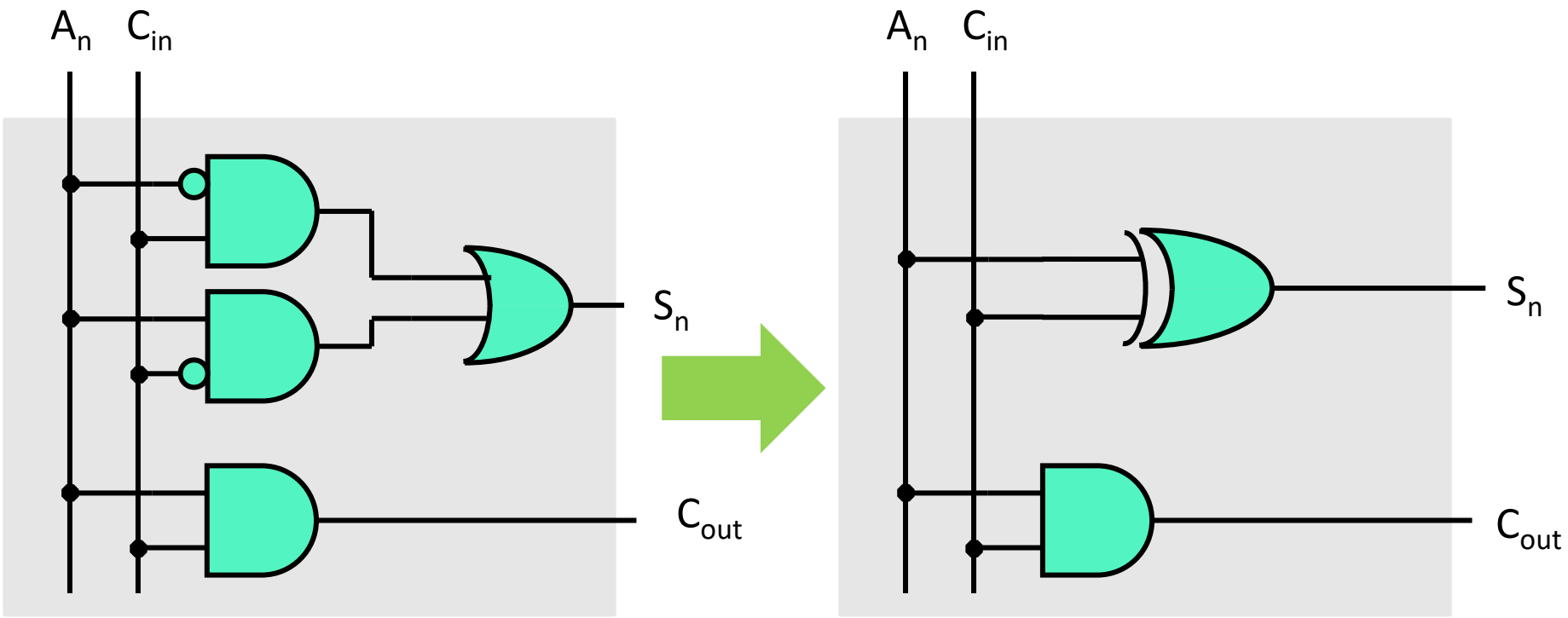
^ is xor

E. I'm not sure

A_n	C_{in}	S_n
0	0	0
0	1	1
1	0	1
1	1	0

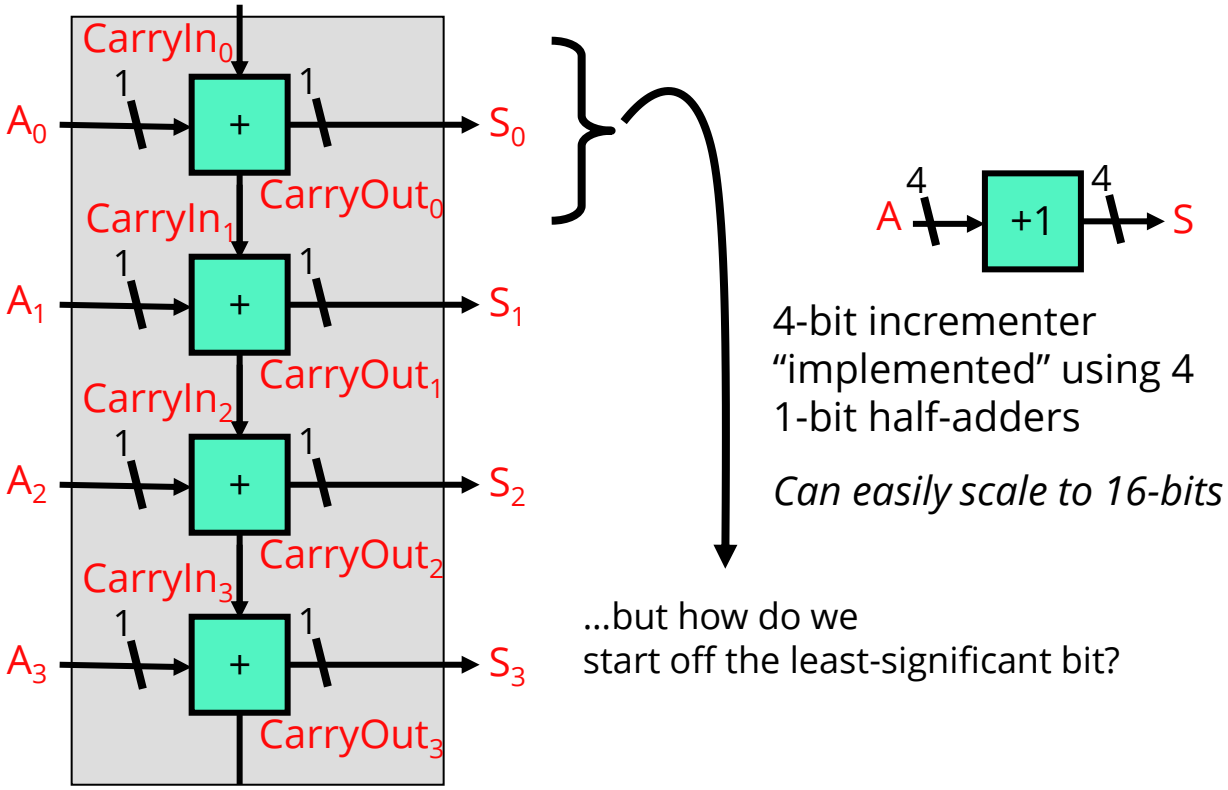
One Bit Incrementor Alternative

- Can implement with an XOR gate instead



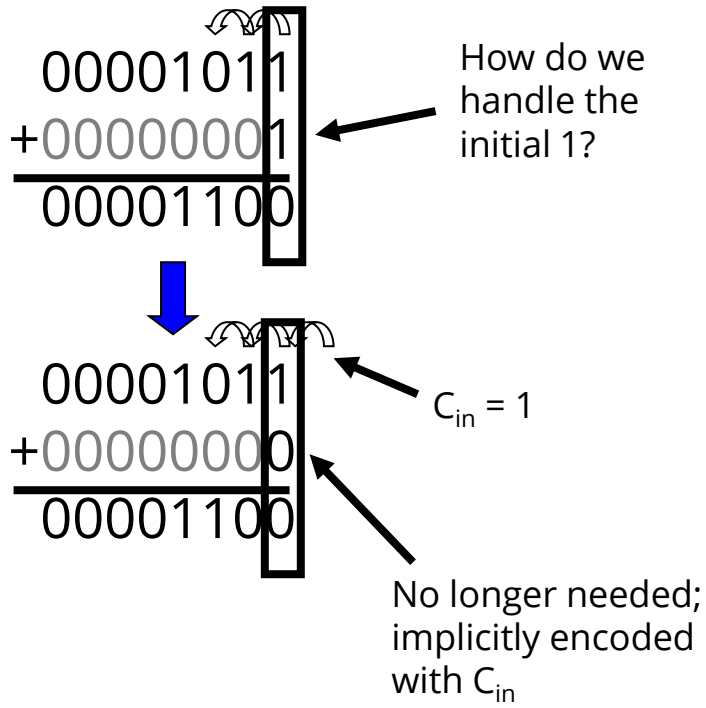
N-bit Incrementor

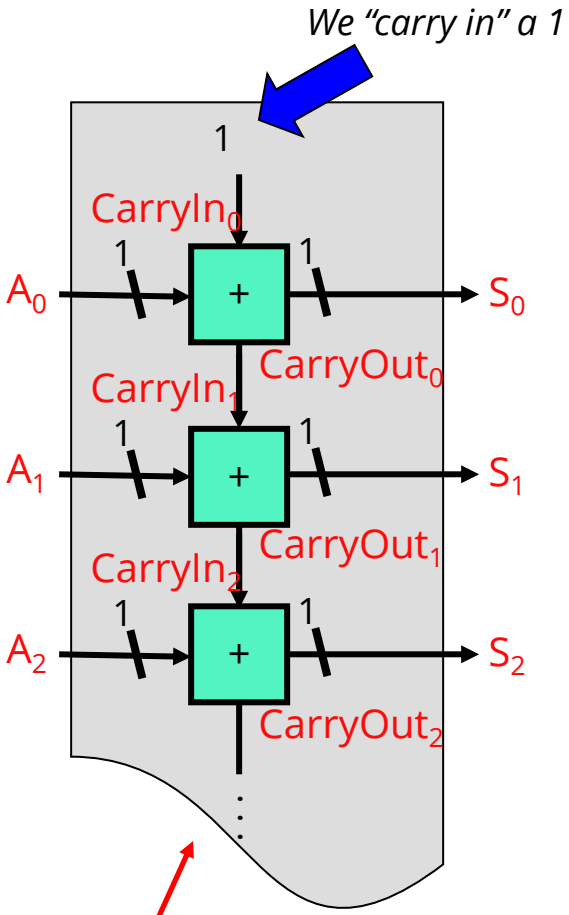
- We can chain the 1-bit Incrementors together
  - Carry-out for bit N, is Carry-in for bit N+1
- 4-bit Incrementor example:



N-bit incrementor LSB

- How do we handle the Least significant bit?





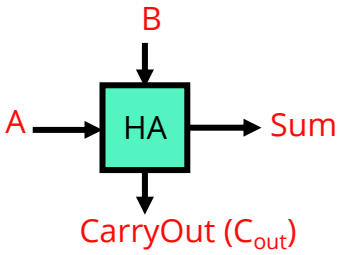
REMEMBER: This is all made of logic gates

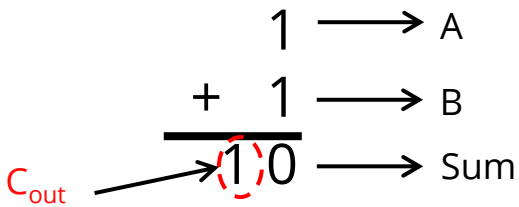
Lecture Outline

- PLAs & Simplification
- Incrementor
- **Adder & Subtracter**
- Mux
- Multiplier & Others

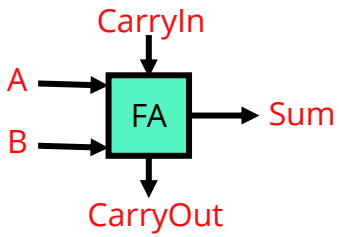
Adder

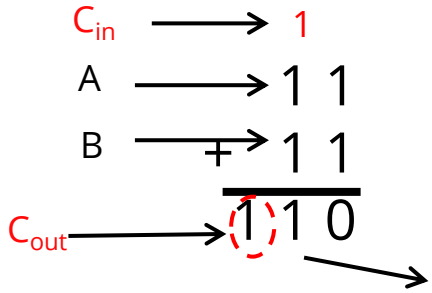
- Similar to incrementor, but doesn't quite work:
  - Incrementor only had to add 2 bits





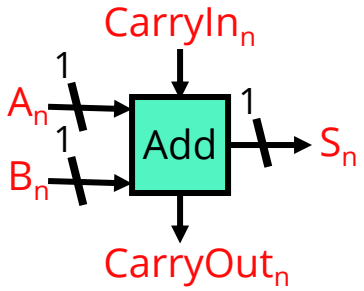
  - Works for the LSB, since there is no "carry in" for the LSB
  - Bits other than the LSB may need to add two bits + carry in



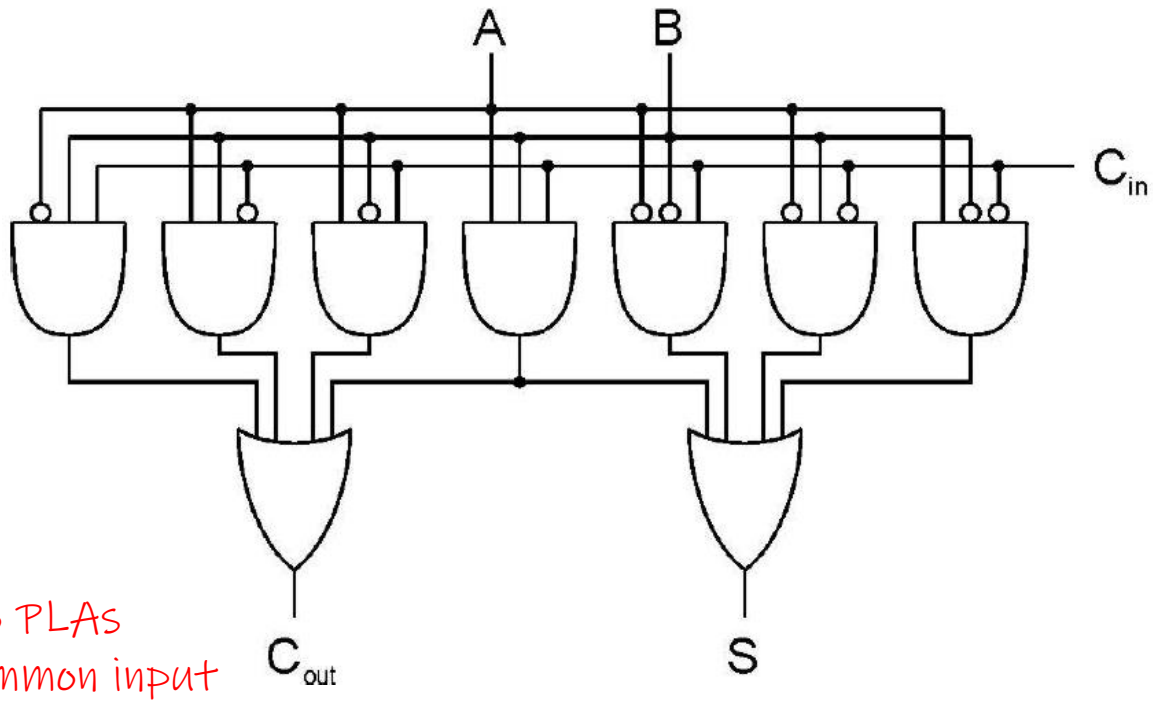


One-Bit Adder

- Like incrementor, we will build a 1-bit component first
- Start from a truth table
- Create a PLA from it

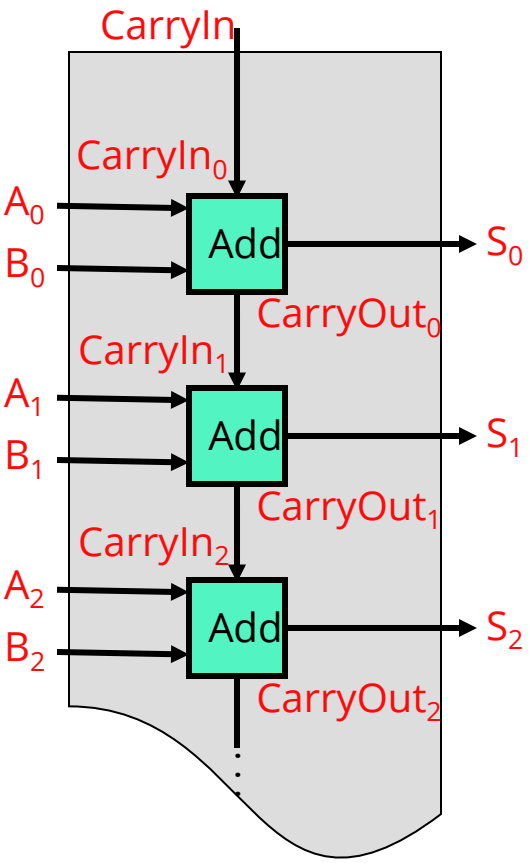


A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

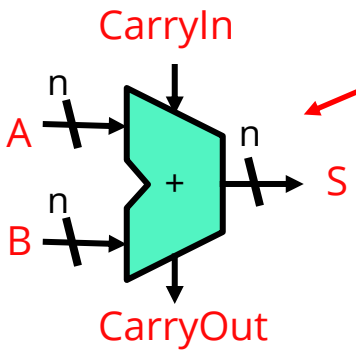


This is just two PLAs fused on the common input

N-Bit Adder



Gate Level Abstraction



CarryOut: useful for detecting overflow

CarryIn: assumed to be zero if not present

Aside: Efficiency

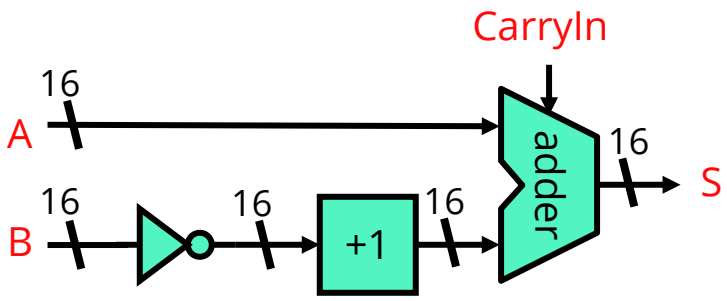
- **Full Disclosure:**
  - Our adder: Ripple-carry adder
  - No one really uses ripple-carry adders
  - Why? *way* too slow
  - Latency proportional to n

- **We can do better:**
  - Many ways to create adders with latency proportional to $log_2(n)$
  - In theory: constant latency (build a big PLA)
  - In practice: too much hardware, too many high-degree gates
  - "Constant factor" matters, too
  - If you continue to CIS 471, you'll encounter "carry look ahead adders", more efficient architecture

Subtractor

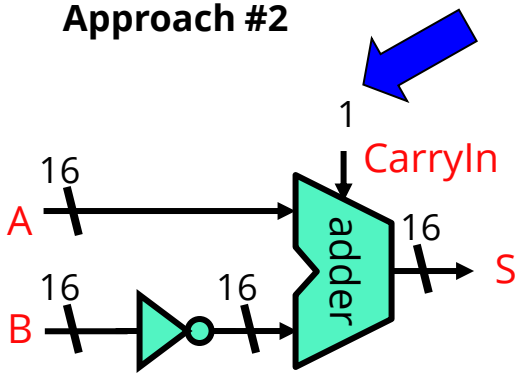
- Build a subtractor from an adder
  - Calculate A – B = A + –B
  - Negate B
  - Recall –B = NOT(B) + 1

Approach #1



We "carry in" a 1
(no longer need incrementer)

Approach #2



Why is approach #2 better?

Can we combine this with the adder?

Lecture Outline

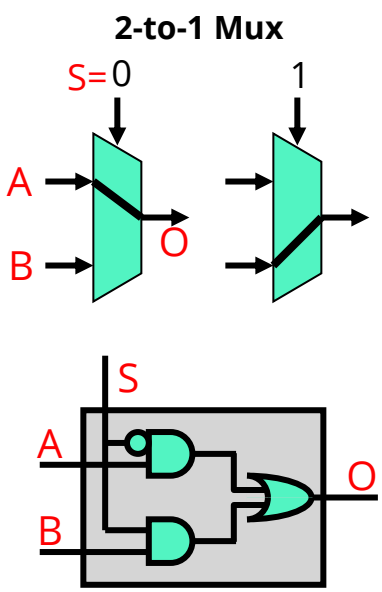
- PLAs & Simplification
- Incrementor
- Adder & Subtracter
- **Mux**
- Multiplier & Others

The Multiplexer

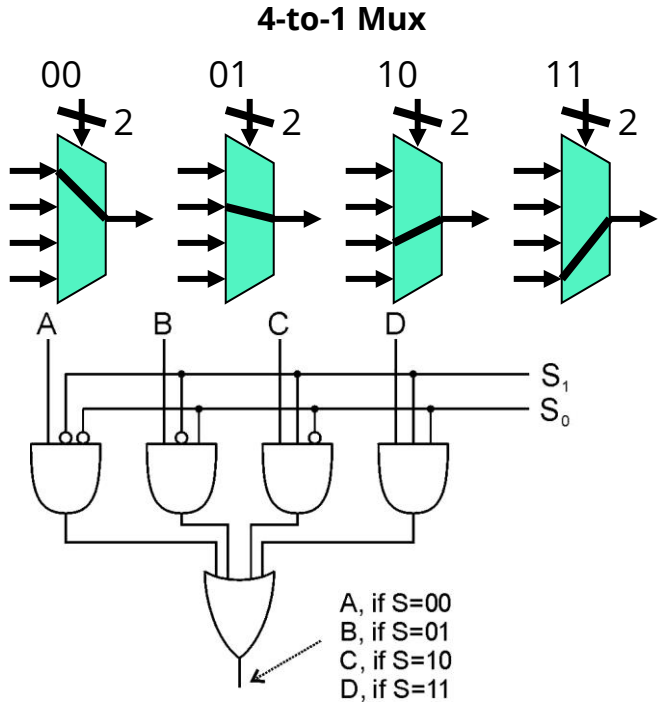
- **Selector/Chooser of signals**
- Shorthand: "Mux"

Note: selector bits map all "0" to the top input, and increment each input "down"

If you don't want to follow this ordering, label your MUX in the Hw



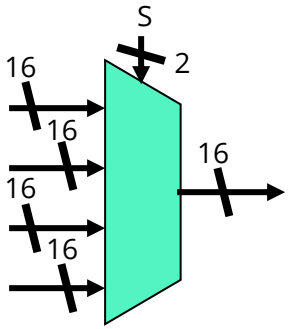
Input "S" *selects* A or B to attach to "O" *output*
Acts like an "IF/ELSE" statement



The Multiplexor In General

- In General
  - N select bits chooses from 2^N inputs
  - **An incredibly useful building block**

- Multi-bit Muxes
  - Can switch an entire "bus" or group of signals
  - Switch n-bits with *n* muxes with the same select bits



Poll Everywhere

pollev.com/tqm

- What is the output of the following mux with selector bits 10

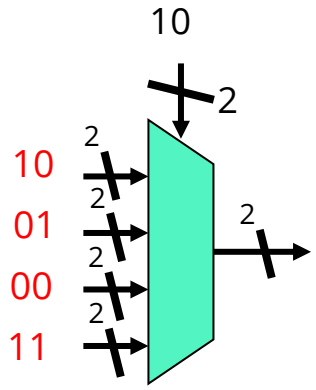
A. 10

B. 01

C. 00

D. 11

E. I'm not sure



Poll Everywhere

pollev.com/tqm

- What is the output of the following mux with selector bits 10

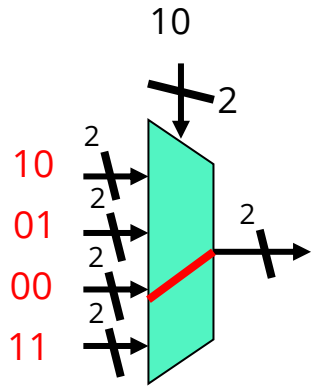
A. 10

B. 01

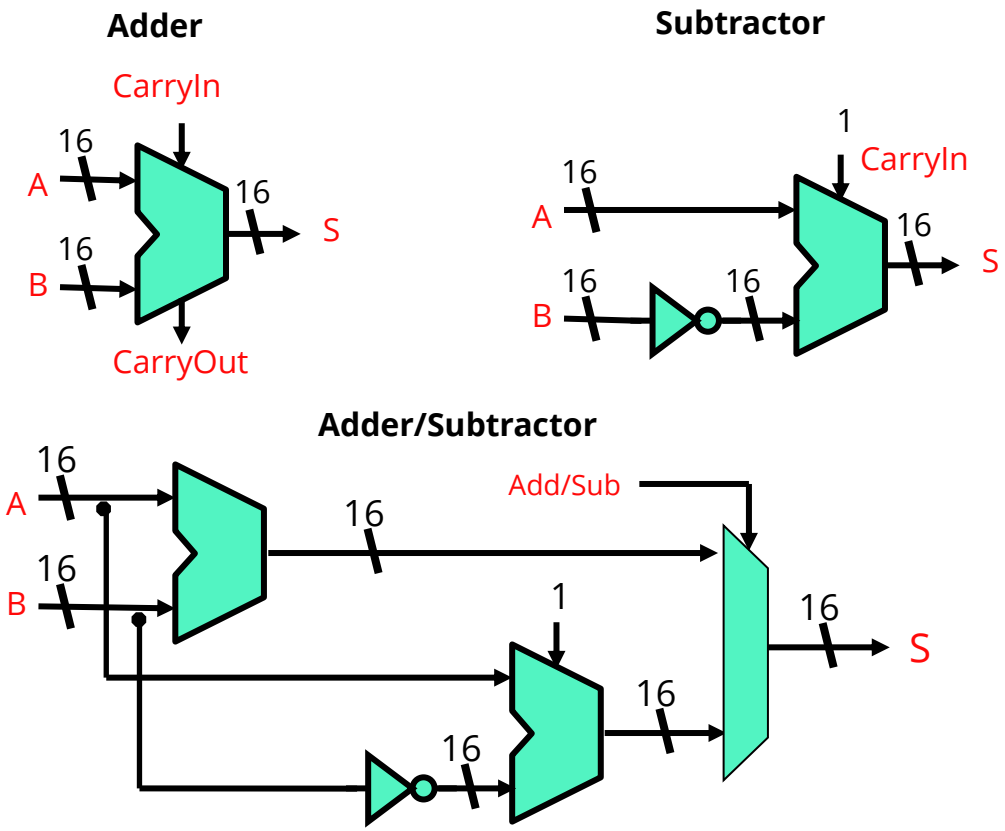
C. 00

D. 11

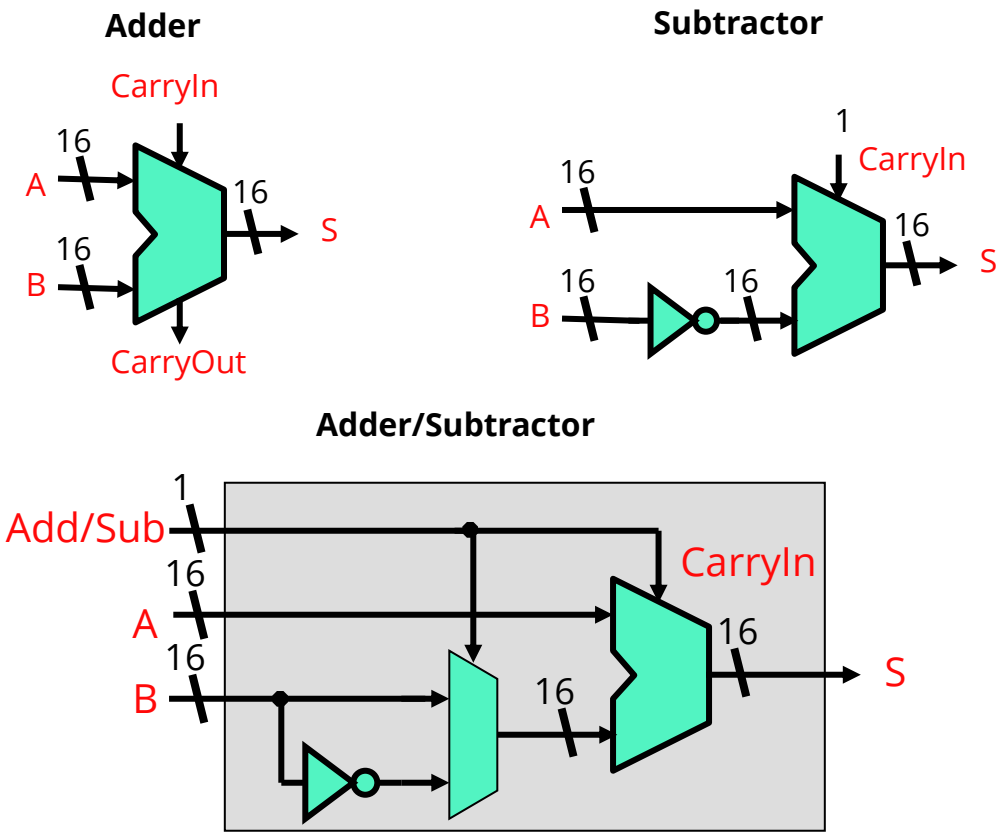
E. I'm not sure



Adder/Subtractor - Approach #1



Adder/Subtractor - Approach #2



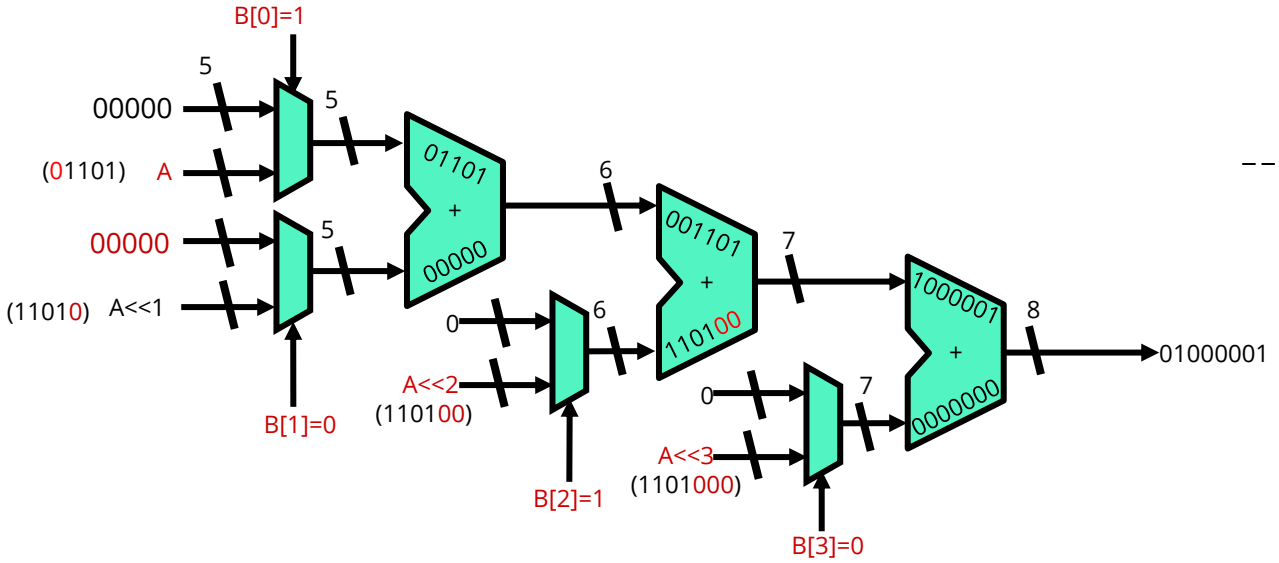
Lecture Outline

- PLAs & Simplification
- Incrementor
- Adder & Subtracter
- Mux
- **Multiplier & Others**

Creating a Multiplier

- Combinational Multiplier using adders & muxes
  - Let's build a 4-bit multiplier that makes an 8-bit product
  - Recall: shifting is the same as multiplying by powers of 2
  - ***Notation in this example: B[0], means LSB bit of B***

```
A=          1101   13₁₀
B=        x 0101    5₁₀
------------------
         01101
         00000
        110100
    + 0000000   65₁₀
------------------
        01000001
```



Arithmetic Algos

- Multiplication:
  - More time efficient algos exist(Karatsuba and others)
- Divide/mod?
  - Much harder than multiplication
  - Most implementations are not combinational, but are sequential (more on sequential logic starting in 2 lectures)
- Bitwise ops (AND, OR, XOR, …)
  - Easy
- Arbitrary left-right shift
  - Can be done with just muxes (try it if you want!)