Combinational Logic

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema	Andrew Rigas	Anisha Bhatia
Audrey Yang	Craig Lee	Daniel Duan
David LuoZhang	Eddy Yang	Ernest Ng
Heyi Liu	Janavi Chadha	Jason Hom
Katherine Wang	Kyrie Dowling	Mohamed Abaker
Noam Elul	Patricia Agnes	Patrick Kehinde Jr.
Ria Sharma	Sarah Luthra	Sofia Mouchtaris

When poll is active, respond at **pollev.com/tqm**

Text **TQM** to **37607** once to join

How Many CU's are you taking? (including this class)

6 or more

5 or 5.5

4 or 4.5

3 or 3.5

2 or 2.5

less than 2

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Logistics

- HW01 bits.c: **This Friday** 9/16 @ 11:59 pm
  - Will require VM setup
  - Has you "program" in C
  - Should have everything you need
  - Terminal & starting demo in Recitations this week

- HW02 Combinational Logic: to be released this week
  - Written Homework, submitted to gradescope
  - **NO EXTENSIONS OVER 72 HOURS**

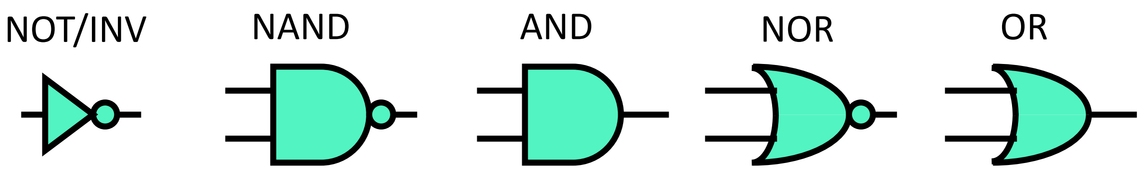
- Check-in01: **Due Monday @ 4:59 pm**
  - Coming out soon

Lecture Outline

- **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:

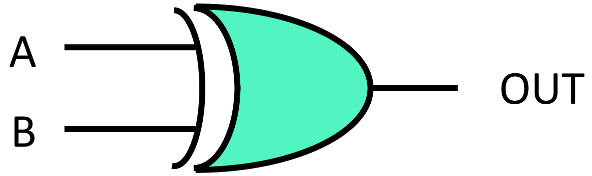


- We can build more complex "gates" that we can use as building blocks for a processor
- This Lecture: start implementing binary arithmetic >:]

Aside: XOR Gate

- Performs the XOR operation

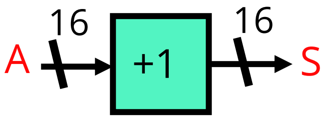
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0



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
```



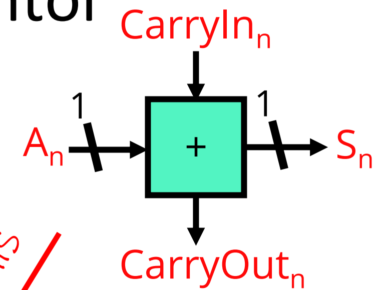
- Theoretical Approach:
  - Use a PLA-like technique to implement the circuit
  - Problem: 2^{16} or 65536 different inputs, 16-bit output
  - This is impractical

One Bit Incrementor "PLA"

- Implementing a single-column of an incrementor

```
 0000000011001011
+0000000000000001
 ----------------
 0000000011001100
```

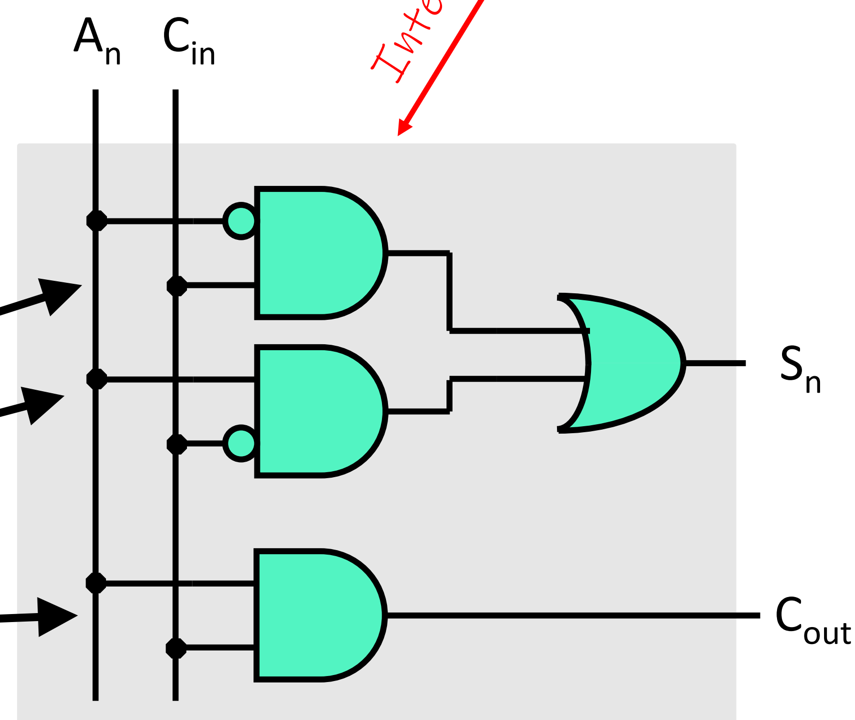
(Ignore LSB for now)



Internals

  - Inputs: A_n, Carry$_{in}$
  - Outputs: S_n, Carry$_{out}$

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. $\sim(C_{in} \wedge A_n)$

D. $A_n \wedge C_{in}$

E. I'm not sure

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. ${\sim}(C_{in} \wedge A_n)$

D. $\boxed{A_n \wedge C_{in}}$

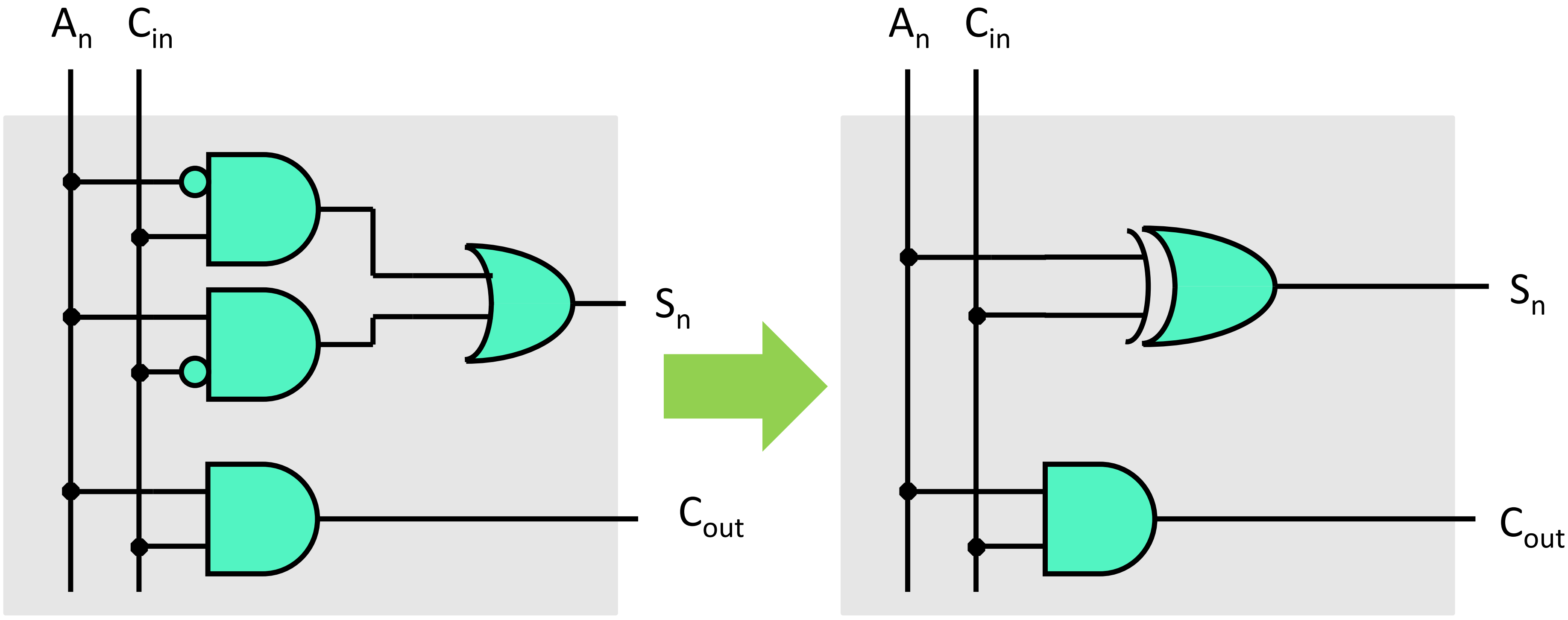
E. I'm not sure

^ is xor

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

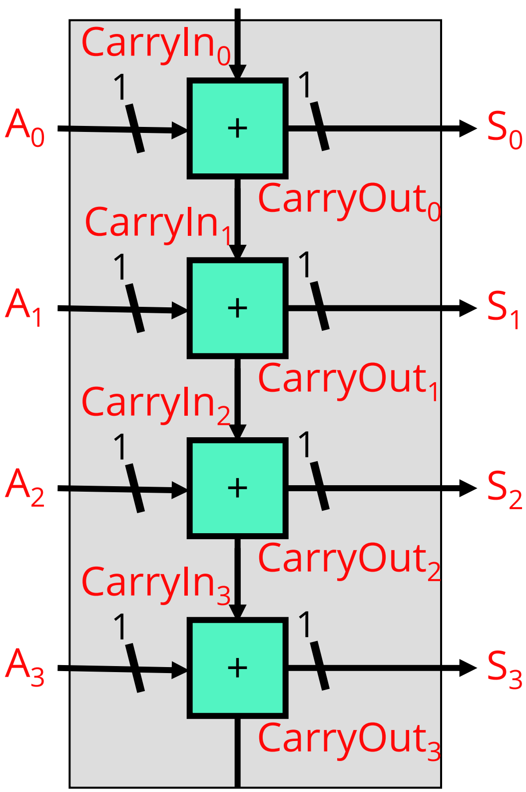
One Bit Incrementor Alternative

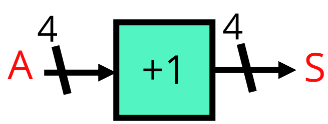
- Can implement with an XOR gate instead



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:





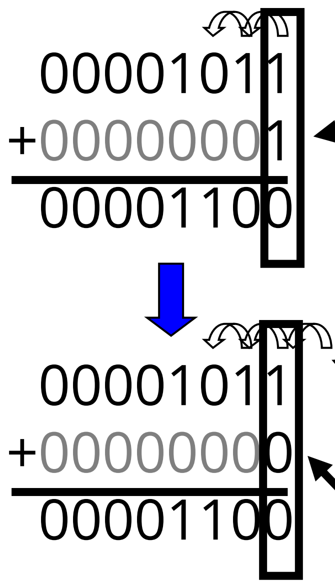
4-bit incrementer "implemented" using 4 1-bit half-adders

Can easily scale to 16-bits

…but how do we start off the least-significant bit?

N-bit incrementor LSB

- How do we handle the Least significant bit?

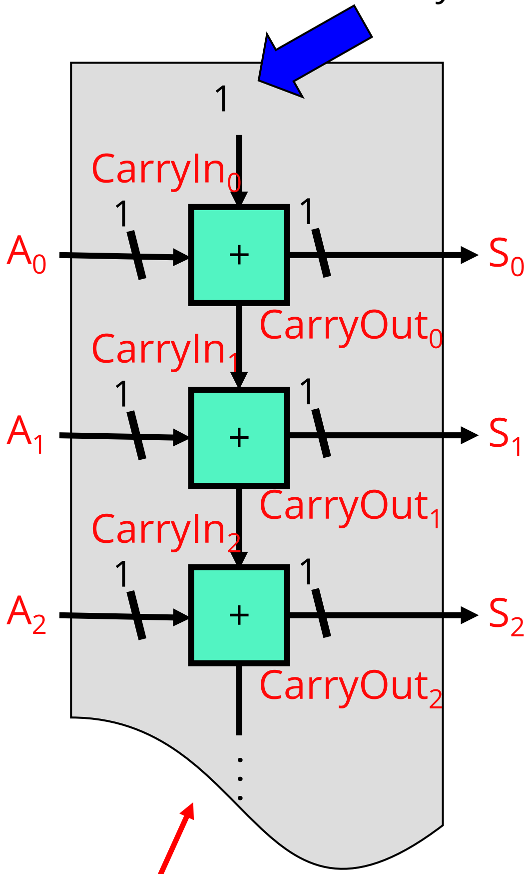


How do we handle the initial 1?

C_{in} = 1

No longer needed; implicitly encoded with C_{in}

We "carry in" a 1



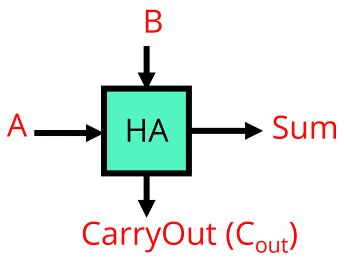
REMEMBER: This is all made of logic gates

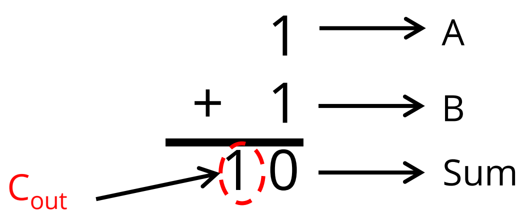
Lecture Outline

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

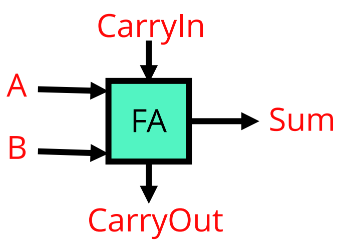
Adder

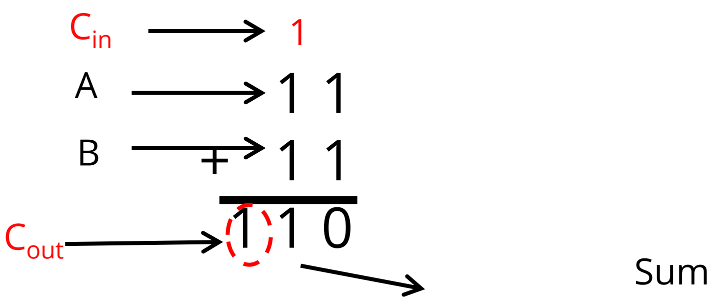
- Similar to incrementor, but doesn't quite work:
  - Incrementor only had to add 2 bits





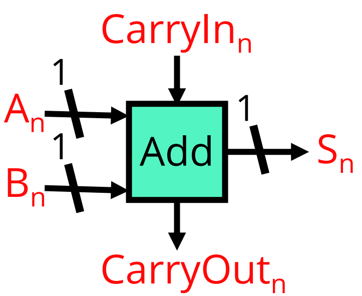
  - Works for the LSB, since there is no "carry in" for the LSB
  - Bits other than the LSB may need to add two bits + carry in



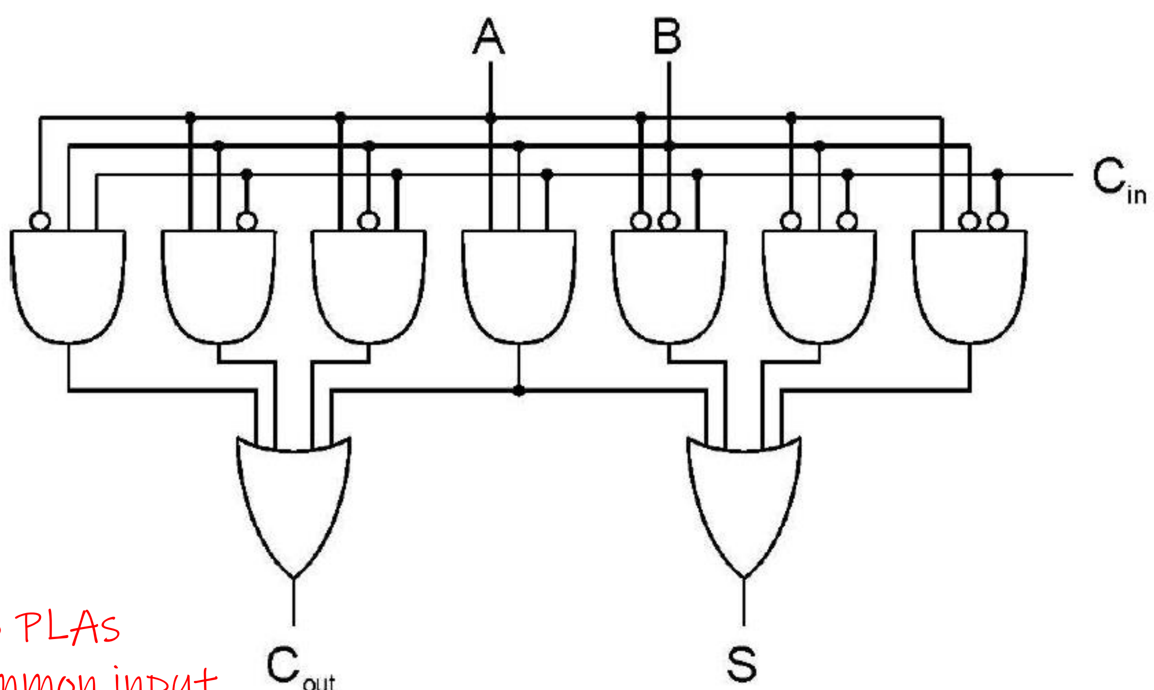


One-Bit Adder

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

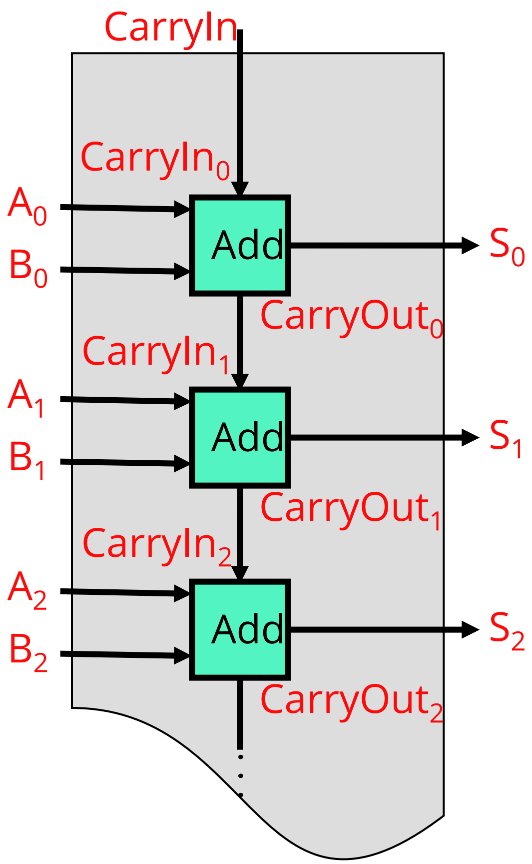


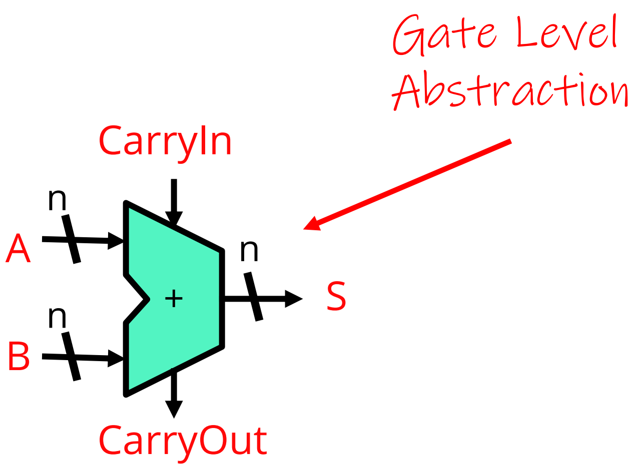
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



This is just two PLAs
fused on the common input

N-Bit Adder





CarryOut: useful for detecting overflow

CarryIn: assumed to be zero if not present

Aside: Efficiency

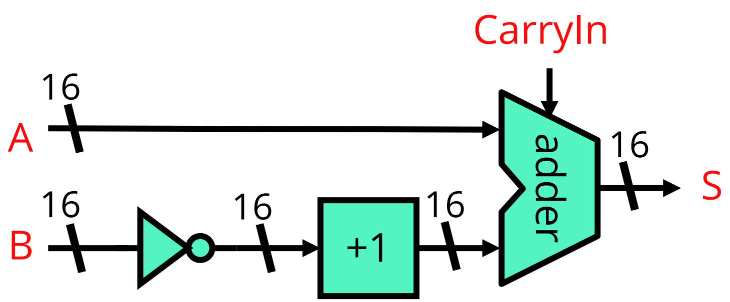
- **Full Disclosure:**
  - Our adder: Ripple-carry adder
  - No one really uses ripple-carry adders
  - Why? *way* too slow
  - Latency proportional to *n*
- **We can do better:**
  - Many ways to create adders with latency proportional to $log_2(n)$
  - In theory: constant latency (build a big PLA)
  - 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

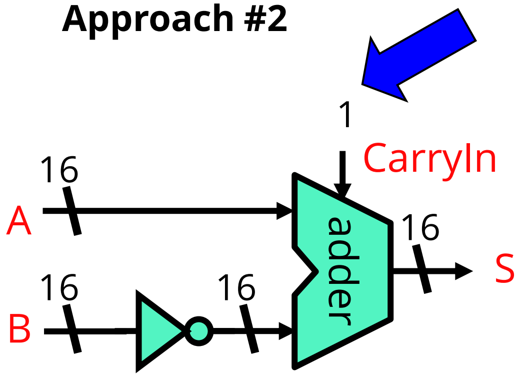
- Build a subtractor from an adder
  - Calculate A – B = A + –B
  - Negate B
  - Recall –B = NOT(B) + 1

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

Approach #1



Approach #2



Why is approach #2 better?

Can we combine this with the adder?

Lecture Outline

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

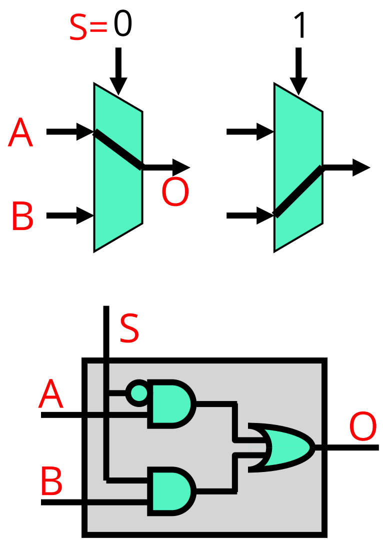
The Multiplexer

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

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

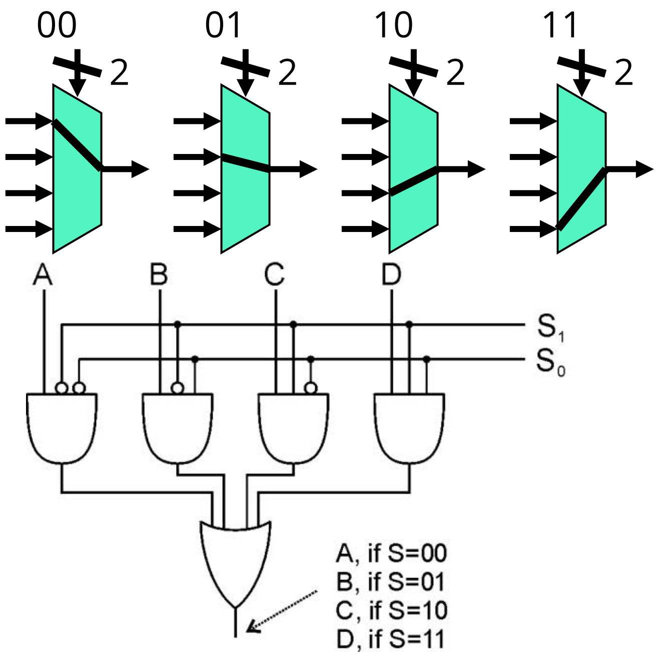
- **Selector/Chooser of signals**
- Shorthand: "Mux"

2-to-1 Mux



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

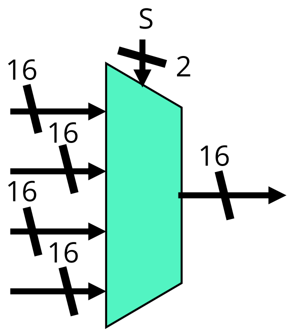
4-to-1 Mux



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

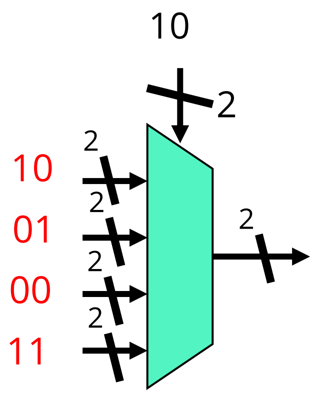
A. 10

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

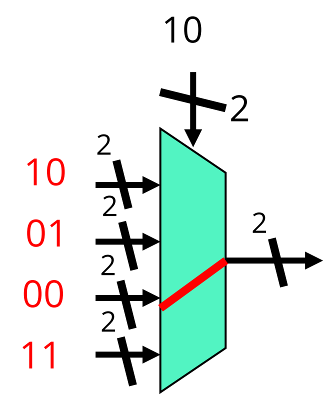
A. 10

B. 01

C. 00

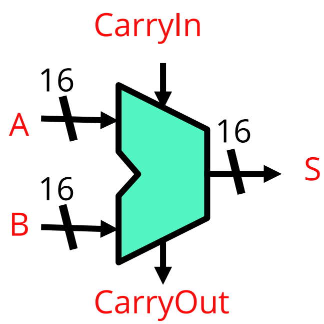
D. 11

E. I'm not sure

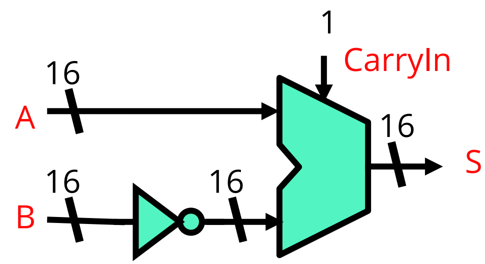


Adder/Subtractor - Approach #1

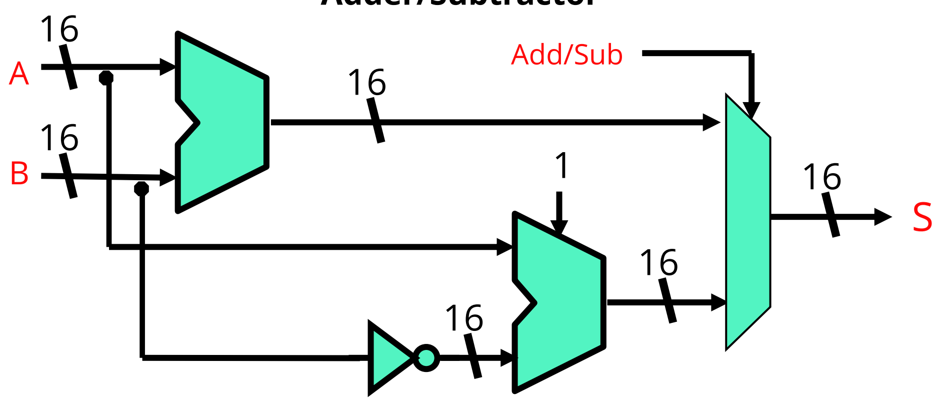
Adder



Subtractor

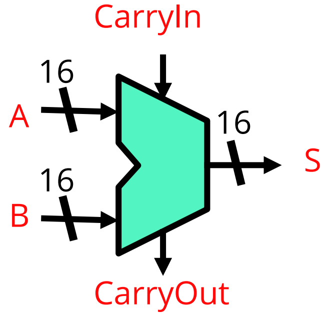


Adder/Subtractor

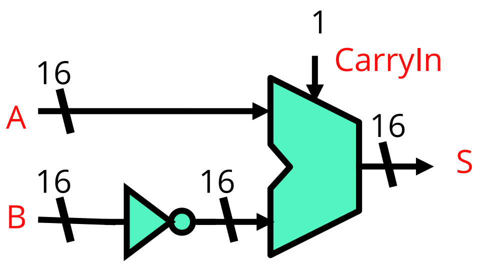


Adder/Subtractor - Approach #2

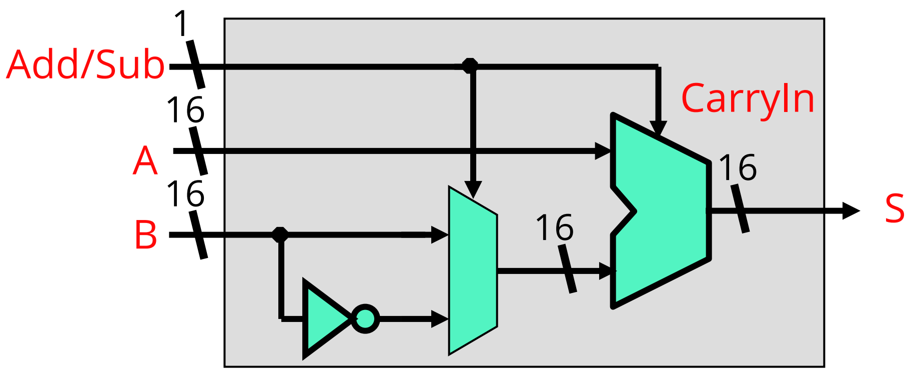
Adder



Subtractor



Adder/Subtractor



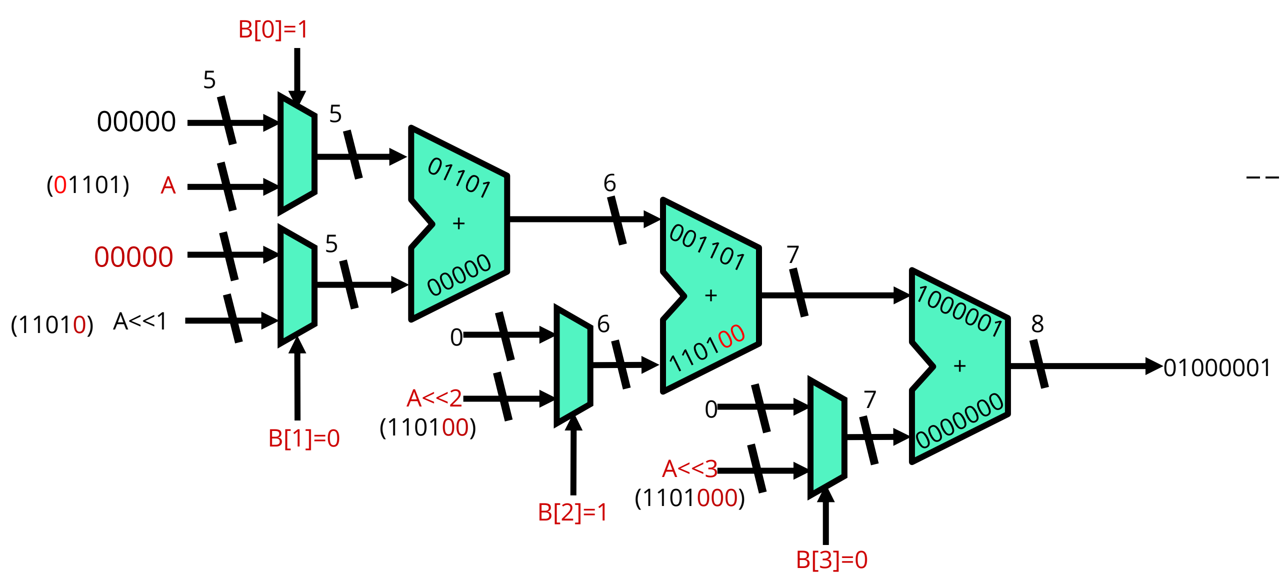
Lecture Outline

- 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     13_10
B=        x 0101      5_10
        -----------
           01101
           00000
          110100
      + 0000000     65_10
        -----------
        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 next lecture)
- Bitwise ops (AND, OR, XOR, …)
  - Easy
- Arbitrary left-right shift
  - Can be done with just muxes (try it if you want!)