# COMP311: *COMPUTER ORGANIZATION!*

Lecture 6: Multiplexers, Gate Minimization Pt. 1

tinyurl.com/comp311-sp26

# Logistics Updates

- Quiz schedule updated
  - *Quiz 2 & 3 shifted by one week.*
- Quiz 2 is next class! Next Wednesday.
- Quiz 1 grades posted
  - *Mean: 90*
  - *Median: 91*

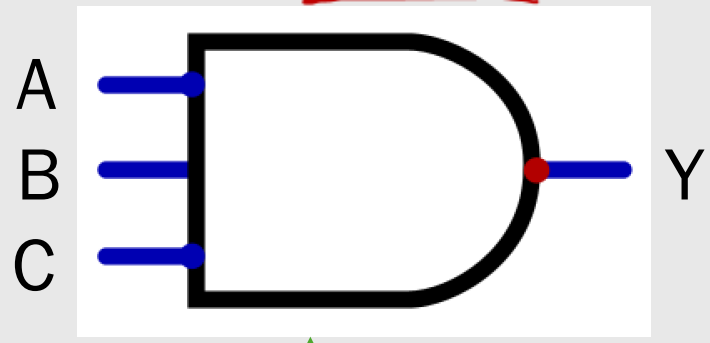
# Quiz Topics

Review Tues 5pm

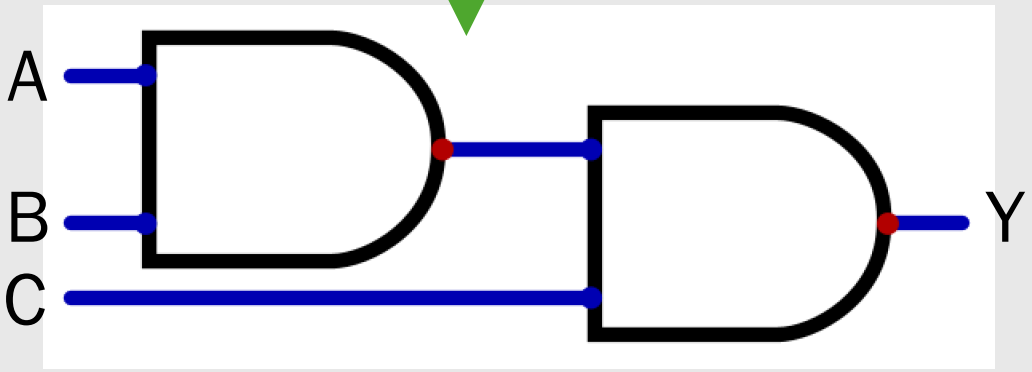
- Topics:
  - *Tracing CMOS gates*
  - *Going between circuit diagrams, boolean equations, and truth tables*
  - *Be able to draw basic circuit diagrams made of logic gates!*
  - *Multiplexers*
  - *Counting the number of transistors*
  - *Boolean algebra simplification*
  - *Whatever we get through today*

# LOGIC GATES WITH 2+ INPUTS

# 3-input AND



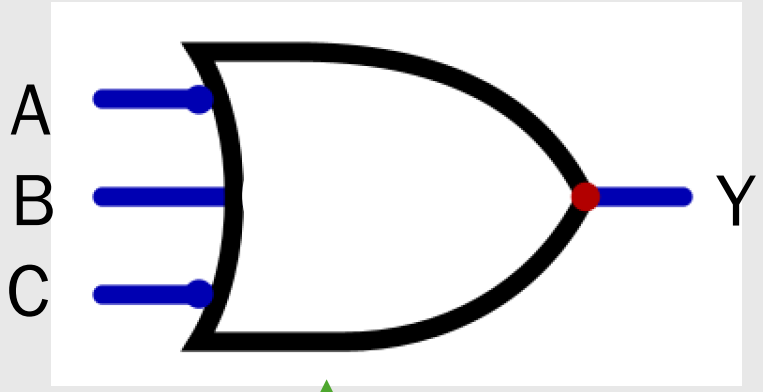

logically equivalent circuits



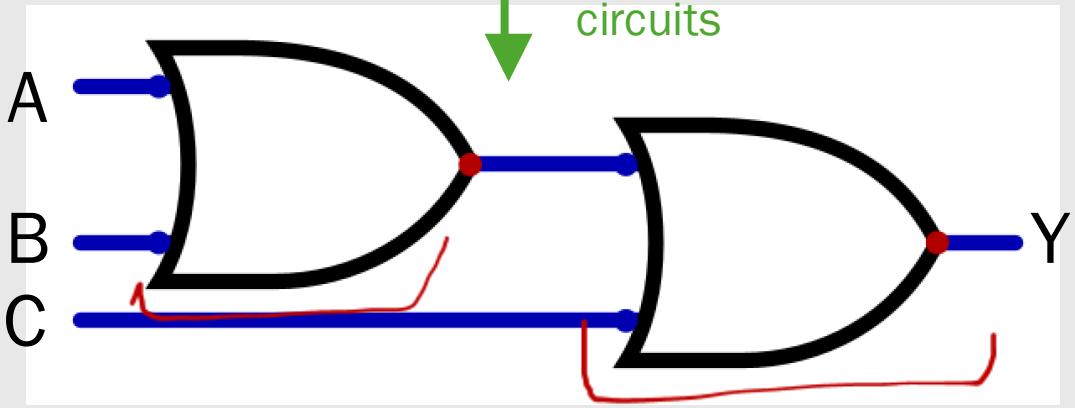

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

A · B · C

# 3-input OR



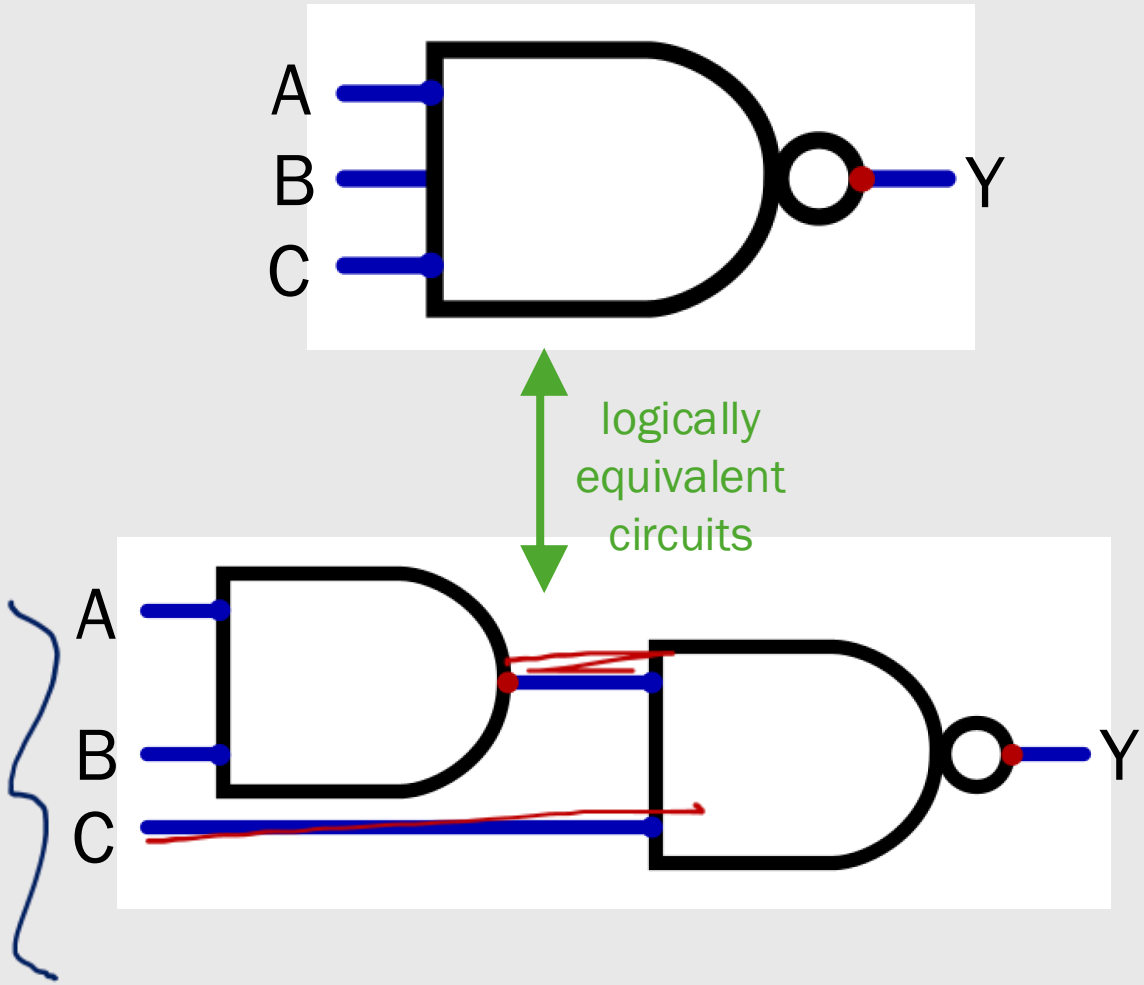

logically equivalent circuits




A + B + C

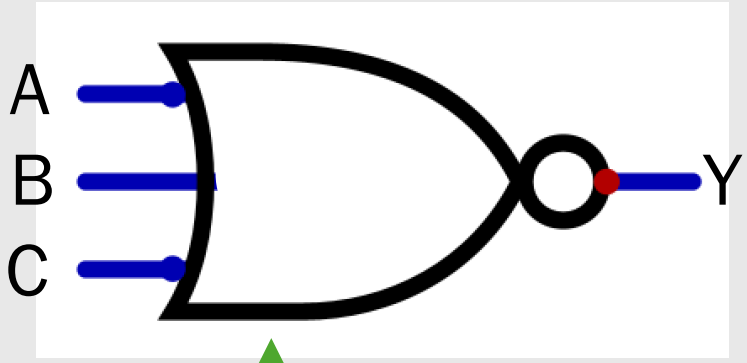
| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# 3-input NAND

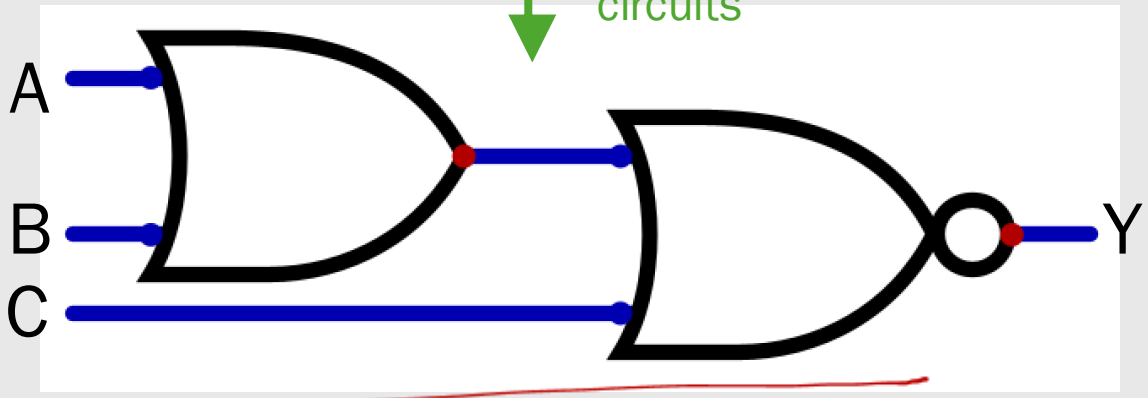



| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# 3-input NOR



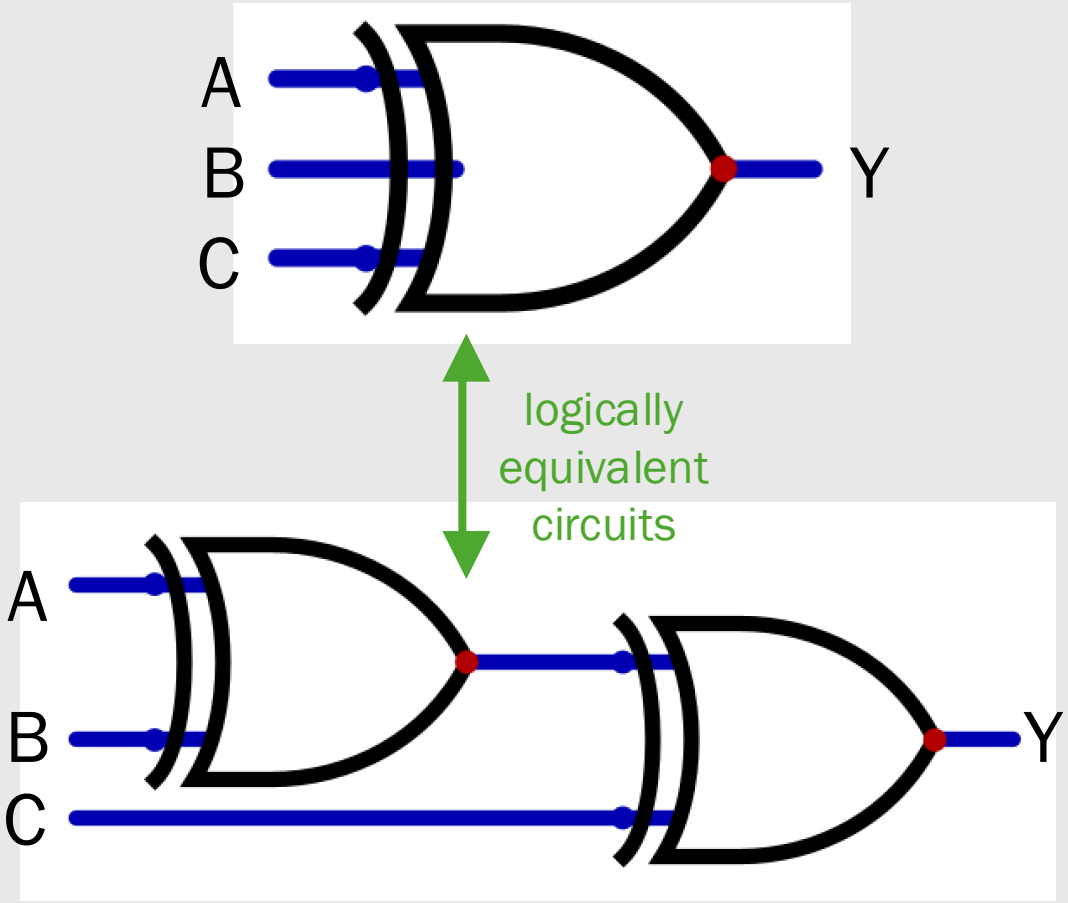

logically equivalent circuits




(A+B)+C

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

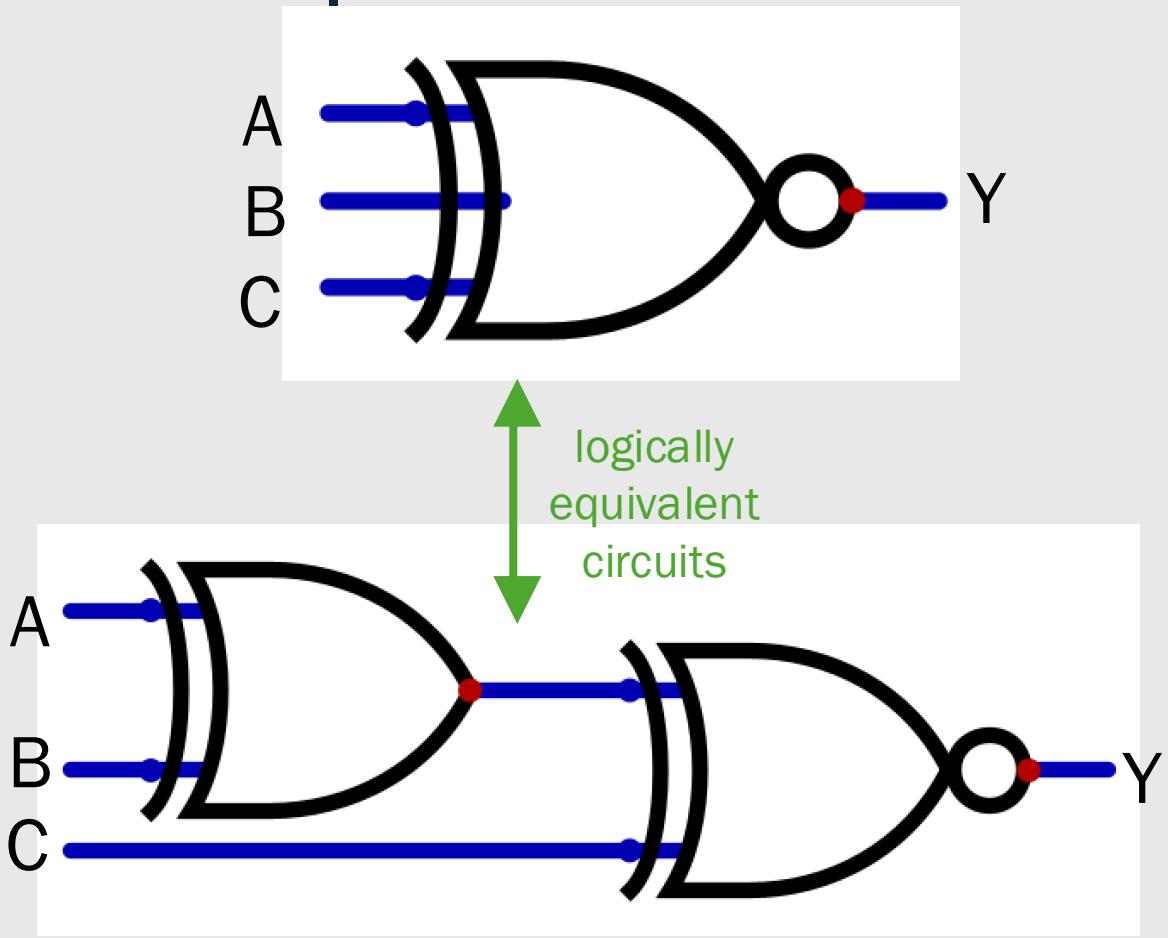
# 3-input XOR




| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

For any number of inputs, the output is 1 when an odd number of inputs is 1

# 3-input XNOR




| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Recall: Transistor Count

| Gate | Number of Transistors |
|---|---|
| NOT | 2 |
| AND | 6 |
| OR | 6 |
| NAND | 4 |
| NOR | 4 |
| XOR | 12 |
| XNOR | 12 |

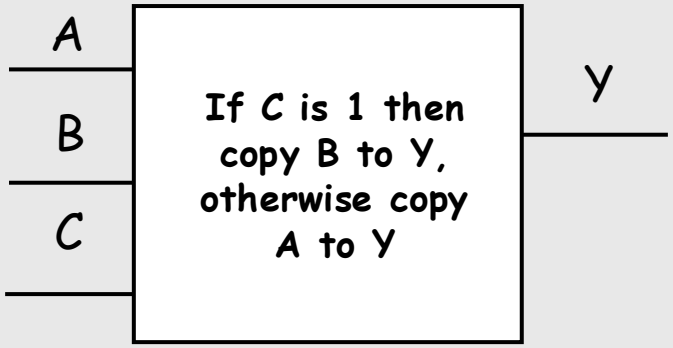
# Transistor Count

| Gate | # of Transistors |
|---|---|
| NOT | 2 |
| n-input AND | 2n + 2 |
| n-input OR | 2n + 2 |
| n-input NAND | 2n |
| n-input NOR | 2n |
| 2-input XOR | 12 |
| 3-input XOR | 28 |
| 2-input XNOR | 12 |
| 3-input XNOR | 28 |

# MULTIPLEXERS

# An interesting 3-input gate…!

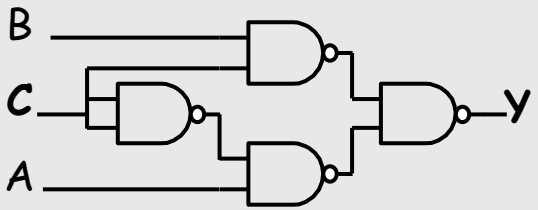
Based on C, select the A or B input to be copied to Y.



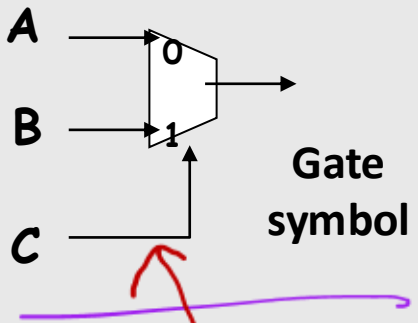

**Truth Table**

| C | B | A | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**2-input Multiplexer**




schematic



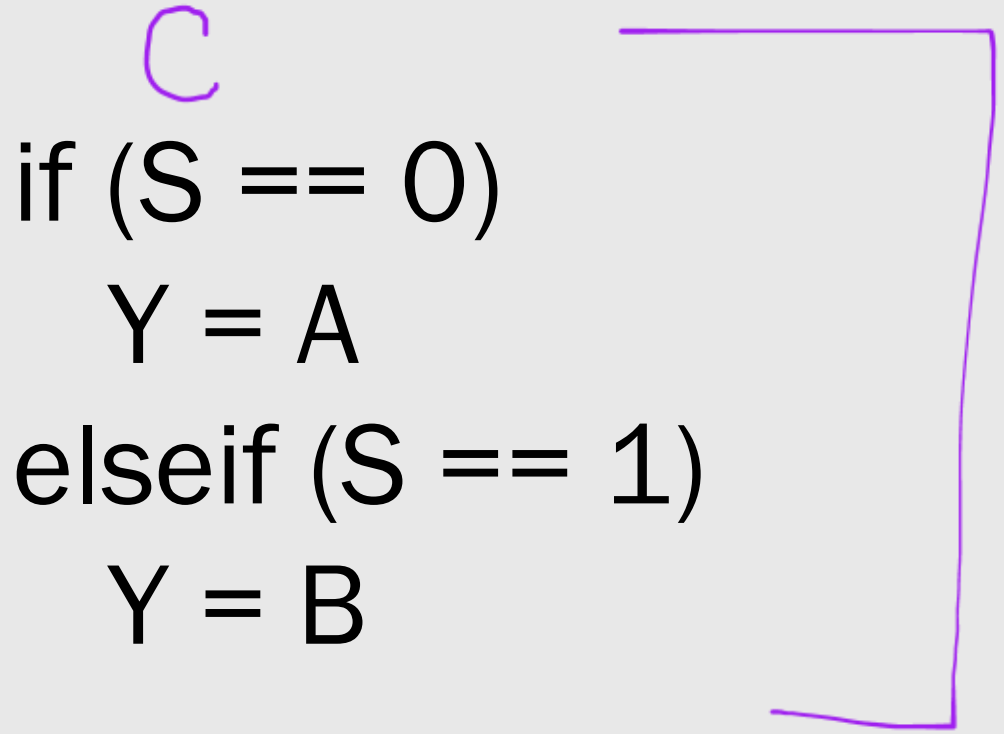

Gate symbol

NANDs

select


14

# A Circuit for If-Statements! ☺



# In Assembly…

```
    beq  x10, x0, S_is_zero      # if S == 0, jump
    addi x14, x0, 1
    beq  x10, x14, S_is_one      # if S == 1, jump

S_is_zero:
    add  x13, x11, x0            # Y = A

S_is_one:
    add  x13, x12, x0            # Y = B
```

New instruction:

***beq***: branch if equal

# Multiplexers

Truth Table

Equation

Diagram

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

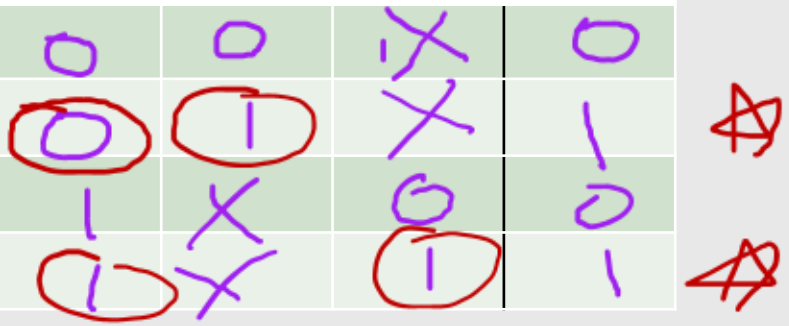
# Multiplexers

2:1 Mux

## Truth Table

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |


| S | A | B | Y |
|---|---|---|---|

Equation

$$Y = A\bar{S} + BS$$

Diagram

# Multiplexers

## Truth Table

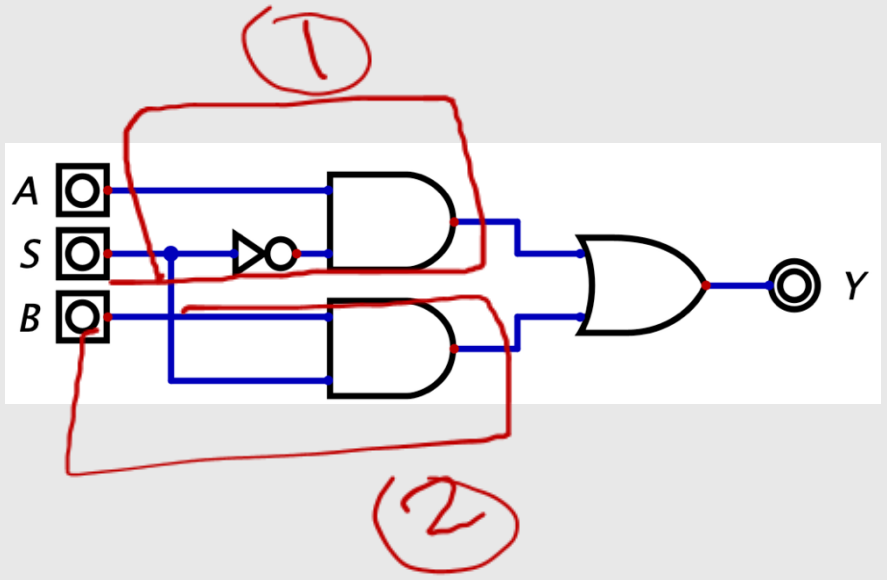
| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | X | 0 |
| 0 | 1 | X | 1 |
| 1 | X | 0 | 0 |
| 1 | X | 1 | 1 |

## Equation

$$Y = A\bar{S} + BS$$

## Diagram



# Multiplexers

Truth Table

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | X | 0 |
| 0 | 1 | X | 1 |
| 1 | X | 0 | 0 |
| 1 | X | 1 | 1 |

Equation

Diagram

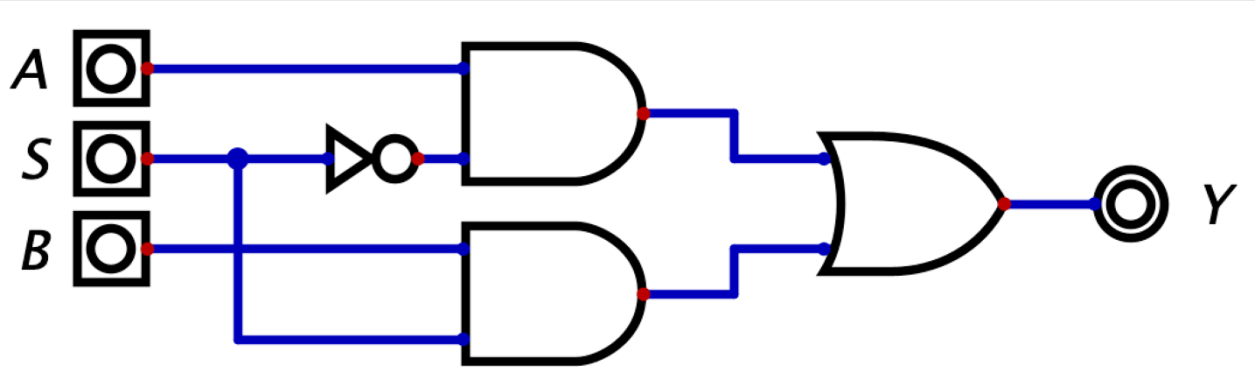
***Don't-cares can be used to simplify logic!***

For input combinations that never occur, the designer may assume either 1 or 0, whichever is more useful to achieving a minimum-cost implementation.

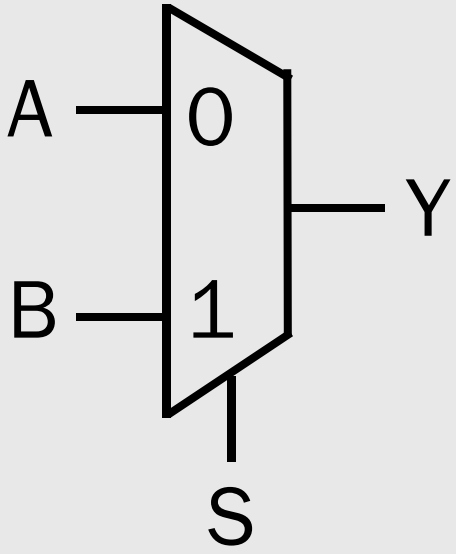
Y

# 2:1 Multiplexer (2:1 mux)

- A circuit that chooses an output from among two inputs based on the value of a select signal



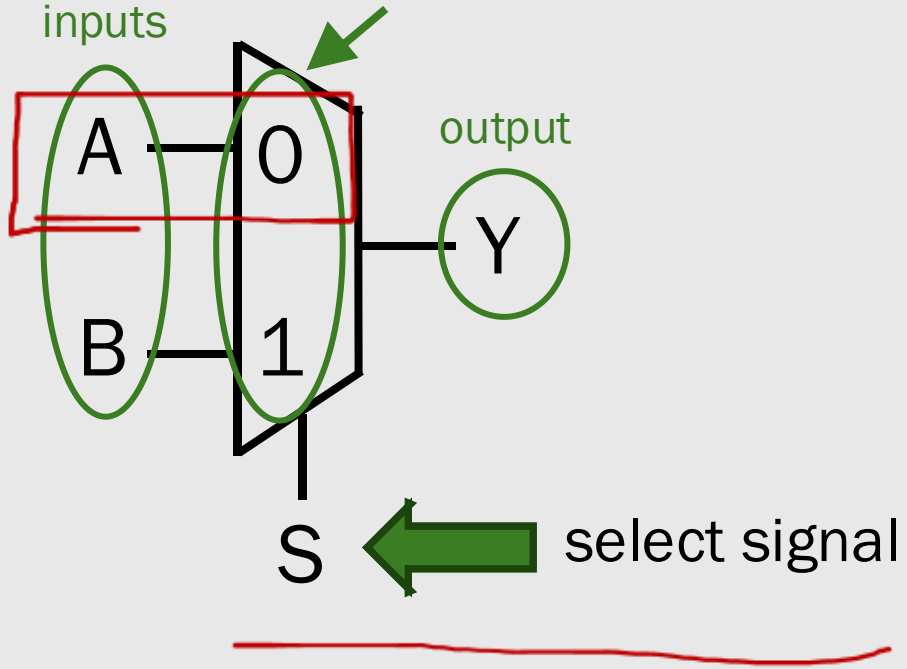

2:1 Multiplexer Schematic
(Gate Diagram)




2:1 Multiplexer Symbol

# 2:1 Multiplexer (2:1 mux)

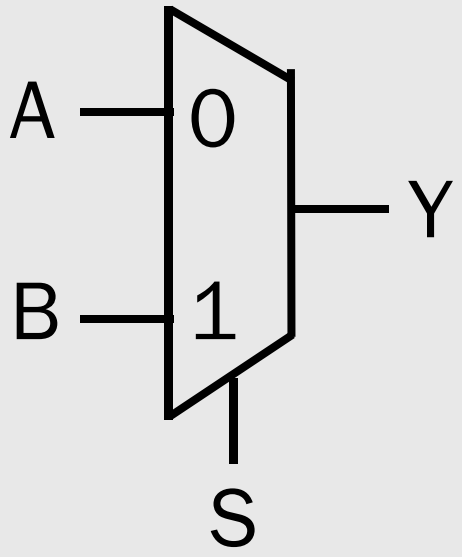
These labels tell us how the multiplexer chooses the output based on the value of the select signal



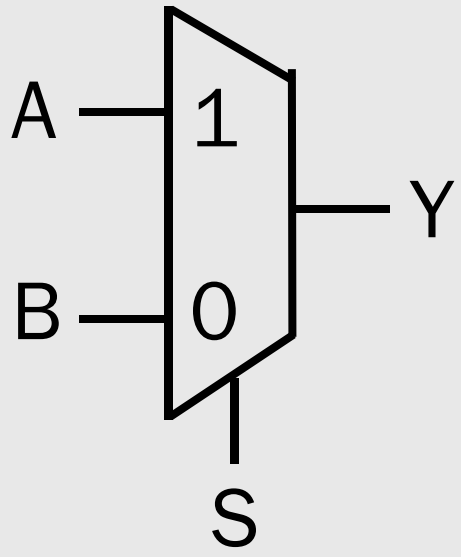

When S = 0, Y = A

When S = 1, Y = B

# 2:1 Multiplexer (2:1 mux)



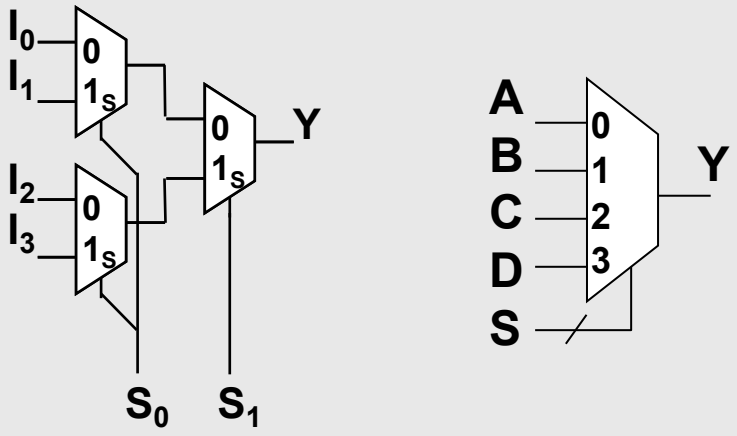

When S = 0, Y = A
When S = 1, Y = B




When S = 0, Y = B
When S = 1, Y = A

# MUX Compositions and Shortcuts

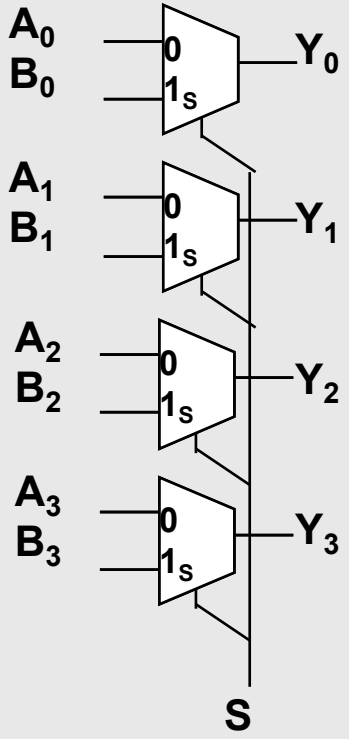
A 4-input Mux
(implemented as a tree)

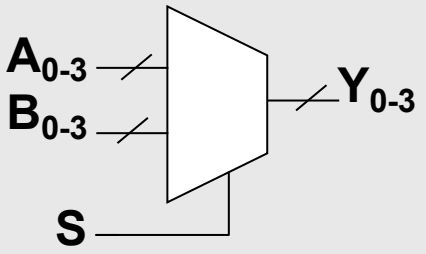






A 4-bit wide 2-input Mux



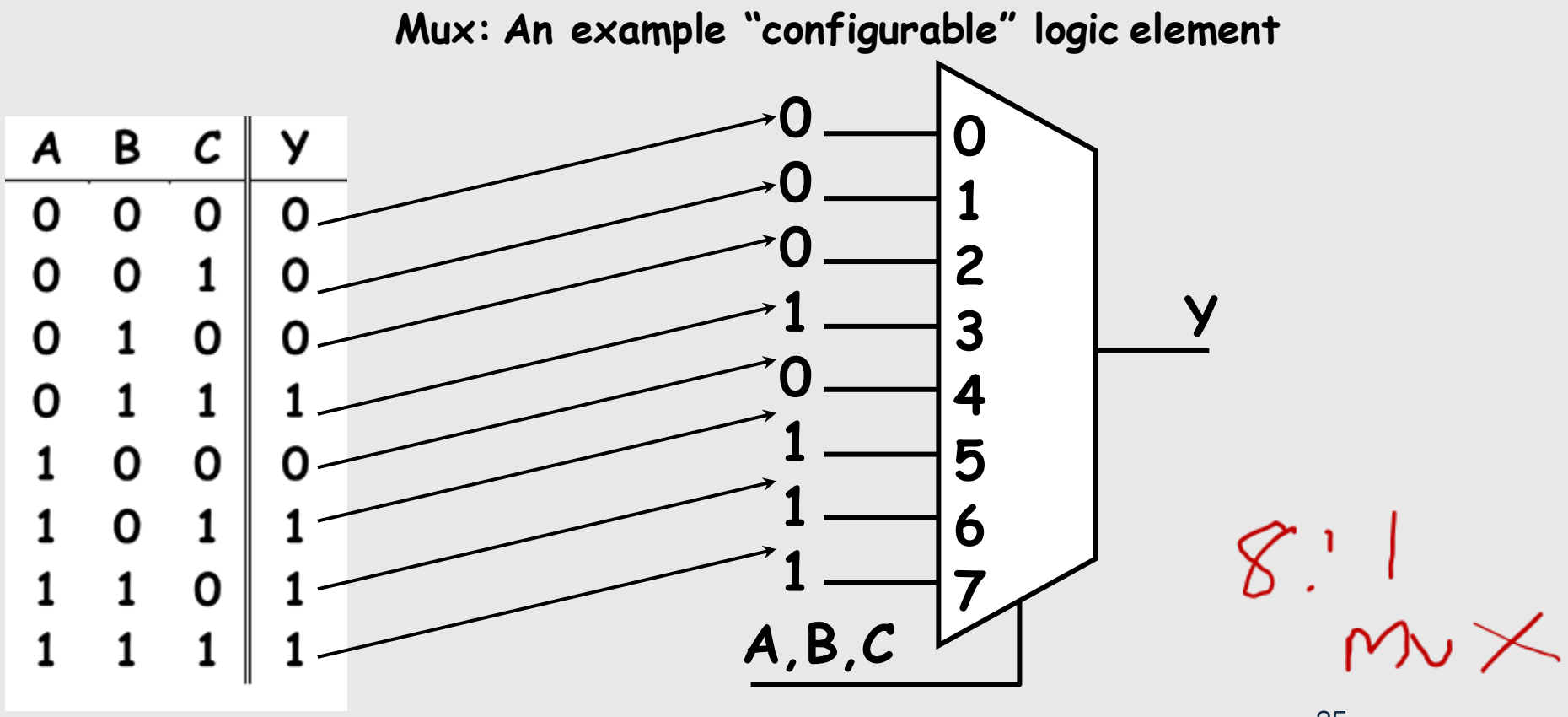


24

# MUX Function Synthesis

A mux plus constants is enough to implement *any* Boolean function

**Consider implementation of some arbitrary Combinational function, F(A,B,C)... using a MULTIPLEXER as the only circuit element:**



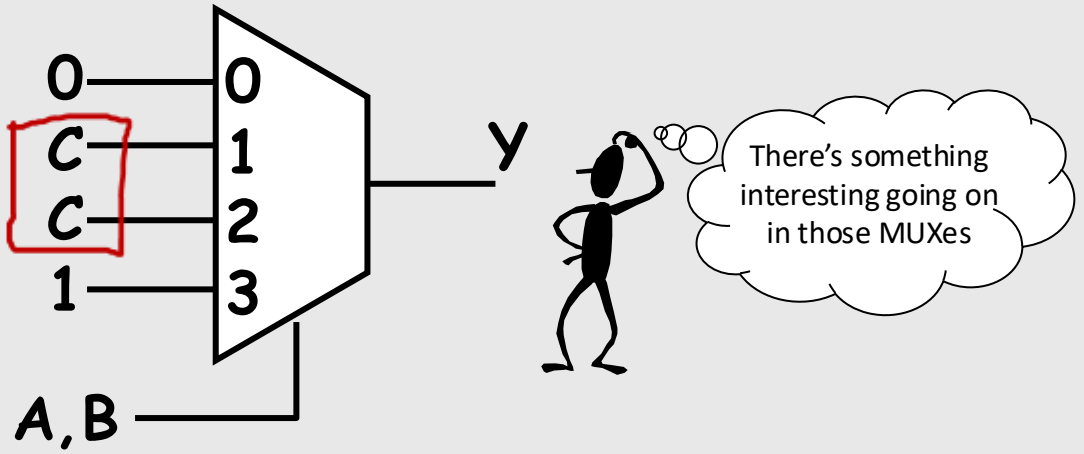
# MUX Logic Tricks

**We can apply certain optimizations to MUX Function synthesis**

Desired Logic Function

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**- Largely by inspection or exhaustive search**




**- N-input function == N-1 input MUX & 1 inverter**

If adjacent truth-table rows differ by just one bit, that bit can become a mux input


26

# FORMING EQUATIONS FROM TRUTH TABLES

# From Last Time: Sum of Products Form

When two or more products (AND) are summed (OR) together

Sum of Products

$Y = AB + AC$

Not Sum of Products

$Y = (A + B)(C + A)$

Approach: write down a boolean expression for every ‘1’ in the output

# Ex: Forming Equations from Truth Tables

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

# Ex: Forming Equations from Truth Tables

| A | B | C | Y | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $\bar{A}\bar{B}\bar{C}$ |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 1 | $\bar{A}B\bar{C}$ |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $A\bar{B}C$ |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 0 | |

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C$$

# Forming Equations from Truth Tables

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# Forming Equations from Truth Tables

| A | B | C | Y | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | $\bar{A}\bar{B}C$ |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | $A\bar{B}\bar{C}$ |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 1 | $AB\bar{C}$ |
| 1 | 1 | 1 | 1 | $ABC$ |

$$Y = \bar{A}\bar{B}C + A\bar{B}\bar{C} + AB\bar{C} + ABC$$

# A Systematic Design Approach

**Truth Table**

| C | B | A | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

-it's systematic!
-it works!
-it's easy!
-we are done! (?)

1) Write the functional spec as a truth table
2) Write down a Boolean expression for every '1' in the output

```
Y = (!C && !B && A) || (!C && B && A)
  || (C && B && !A) || (C && B && A)
```

3) Wire up the ideal gates, replace them with equivalent realizable gates, call it a day, and go home!

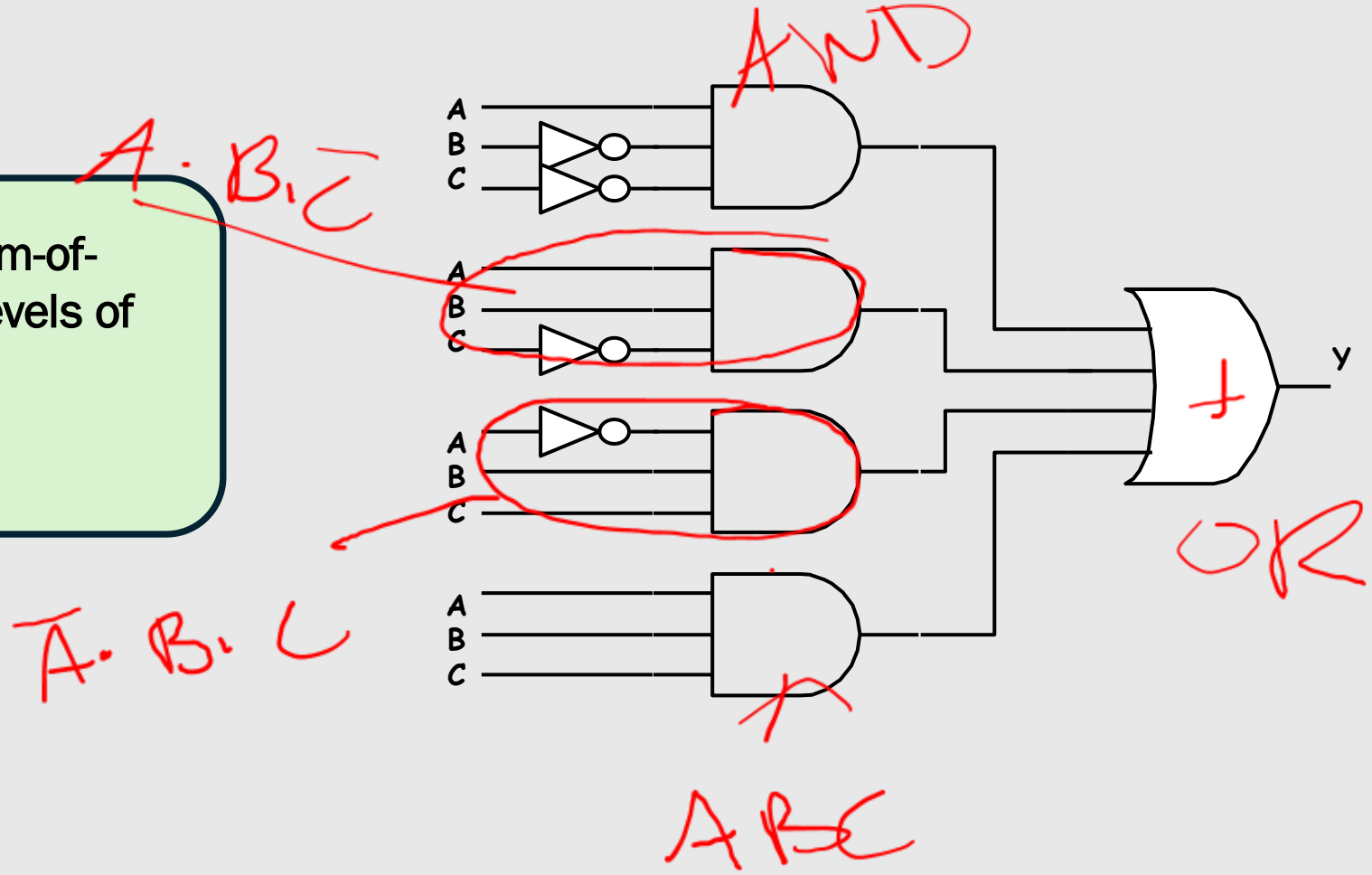
This approach will always give us logic expressions in a particular form:

*SUM-OF-PRODUCTS*

# Straightforward Synthesis

We can implement sum-of-products with just 3 levels of logic, using...

*INVERTERS/AND/OR!*



# Let's Try it Out

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

I want to build a circuit that will output 1 if A > B, and 0 if A <= B. A and B are both two-bit unsigned numbers.

Additionally, define $A_1$ and $B_1$ as the left bits of A and B, and $A_0$ and $B_0$ as the right bits.

$Y =$

# Let's Try it Out

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

I want to build a circuit that will output 1 if A > B, and 0 if A <= B. A and B are both two-bit unsigned numbers.

Additionally, define $A_1$ and $B_1$ as the left bits of A and B, and $A_0$ and $B_0$ as the right bits.

$$y = \bar{A}_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 B_0 + A_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 \bar{B}_0$$

*Is the simplest possible form??* 🧐

# GATE MINIMIZATION

# Optimizing our circuits…

- Fewer transistors means….
  - *Smaller area!*
  - *Lower power consumption!*
  - *Shorter time from input to output!*
  - *Lower cost!*
- How will we do this?
  - *Visual inspection*
  - *Boolean algebra simplification*
  - *In practice, → advanced synthesis algorithms to create **logically equivalent,** more efficient circuits!* ☺

# Recall: Transistor Count

| Gate | Number of Transistors |
|---|---|
| NOT | 2 |
| AND | 6 |
| OR | 6 |
| NAND | 4 |
| NOR | 4 |
| XOR | 12 |
| XNOR | 12 |

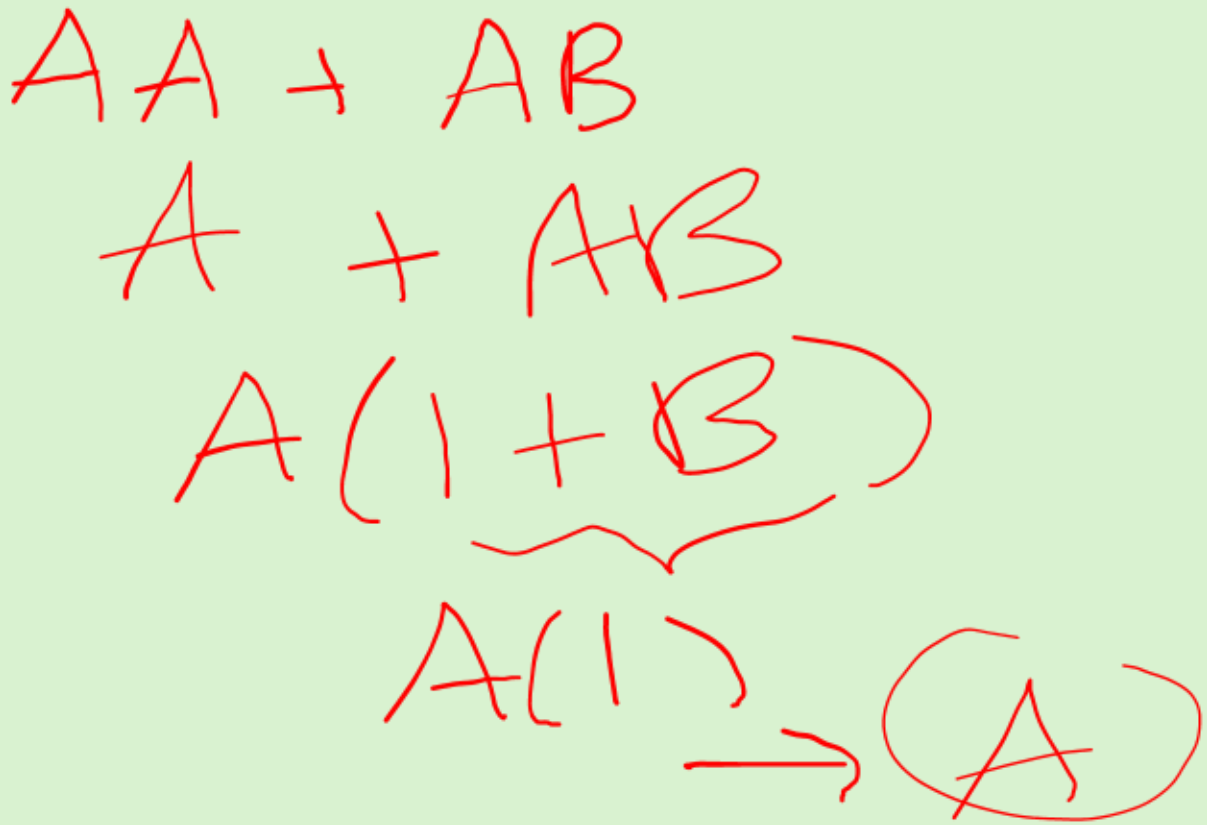
# Boolean Algebra Laws

| Name | OR Form | AND Form |
|---|---|---|
| Identity | $A + 0 = A$ | $A \cdot 1 = A$ |
| Null | $A + 1 = 1$ | $A \cdot 0 = 0$ |
| Idempotent | $A + A = A$ | $A \cdot A = A$ |
| Complement | $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ |
| Commutative | $A + B = B + A$ | $A \cdot B = B \cdot A$ |
| Associative | $(A + B) + C = A + (B + C)$ | $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ |
| Distributive | $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ |
| De Morgan's | $\overline{A + B} = \overline{A} \cdot \overline{B}$ | $\overline{A \cdot B} = \overline{A} + \overline{B}$ |

# Boolean Algebra Practice

Using the laws that you have just learned, simplify the following Boolean expression:

$$\mathbf{A \cdot (A + B)}$$



# Boolean Algebra Practice

$$\mathbf{A \cdot (A + B)}$$

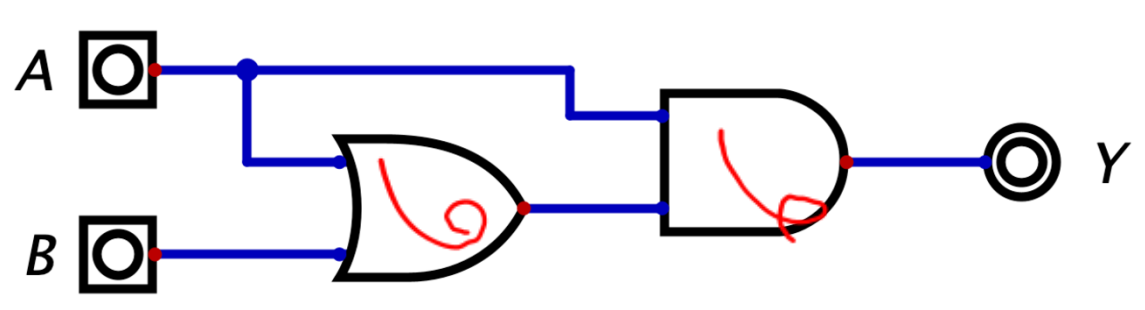
$$AA + AB$$
$$A + AB$$
$$A(1 + B)$$
$$A(1)$$
$$A$$

| Name | OR Form | AND Form |
|---|---|---|
| Identity | A + 0 = A | A · 1 = A |
| Null | A + 1 = 1 | A · 0 = 0 |
| Idempotent | A + A = A | A · A = A |
| Complement | $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ |
| Commutative | A + B = B + A | A · B = B · A |
| Associative | (A + B) + C = A + (B + C) | (A · B) · C = A · (B · C) |
| Distributive | A · (B + C) = (A · B) + (A · C) | A + (B · C) = (A + B) · (A + C) |
| De Morgan's | $\overline{A + B} = \overline{A} \cdot \overline{B}$ | $\overline{A \cdot B} = \overline{A} + \overline{B}$ |

Absorption Law: $A \cdot (A + B) = A$

# Comparing Number of Transistors Used

## Original Circuit




$$Y = A \cdot (A + B)$$

Number of Transistors = 12

## Simplified Circuit


$$Y = A$$

Number of Transistors = 0

# Boolean Algebra Practice

Simplify the following Boolean expression: $B + \overline{B}C$

| Name | OR Form | AND Form |
|---|---|---|
| Identity | $A + 0 = A$ | $A \cdot 1 = A$ |
| Null | $A + 1 = 1$ | $A \cdot 0 = 0$ |
| Idempotent | $A + A = A$ | $A \cdot A = A$ |
| Complement | $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ |
| Commutative | $A + B = B + A$ | $A \cdot B = B \cdot A$ |
| Associative | $(A + B) + C = A + (B + C)$ | $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ |
| Distributive | $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ |
| De Morgan's | $\overline{A + B} = \overline{A} \cdot \overline{B}$ | $\overline{A \cdot B} = \overline{A} + \overline{B}$ |
| Absorption | $A \cdot (A + B) = A$ | $A + (A \cdot B) = A$ |

# Boolean Algebra Practice

$$B + \bar{B}C$$

$$(B + \bar{B})(B + C)$$
$$(1)(B + C)$$
$$B + C$$

| Name | OR Form | AND Form |
|---|---|---|
| Identity | A + 0 = A | A · 1 = A |
| Null | A + 1 = 1 | A · 0 = 0 |
| Idempotent | A + A = A | A · A = A |
| Complement | $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ |
| Commutative | A + B = B + A | A · B = B · A |
| Associative | (A + B) + C = A + (B + C) | (A · B) · C = A · (B · C) |
| Distributive | A · (B + C) = (A · B) + (A · C) | A + (B · C) = (A + B) · (A + C) |
| De Morgan's | $\overline{A + B} = \overline{A} \cdot \overline{B}$ | $\overline{A \cdot B} = \overline{A} + \overline{B}$ |

# Recall: De Morgan's Theorem

De Morgan's Theorem: express conjunctions & disjunctions (ANDS, ORS) solely in terms of negation!

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

$$\overline{AB} = \overline{A} + \overline{B}$$

# Applying DeMorgan's

$$\overline{AB + CD}$$

- How would you apply DeMorgan's to this expression?
  - *We group our terms based on the lowest operator precedence under the bar*
  - The order of operations for Boolean Algebra is **NOT, then AND, then OR.** Expressions inside brackets are always evaluated first.
  - *In this case, the operator with the lowest precedence is OR*

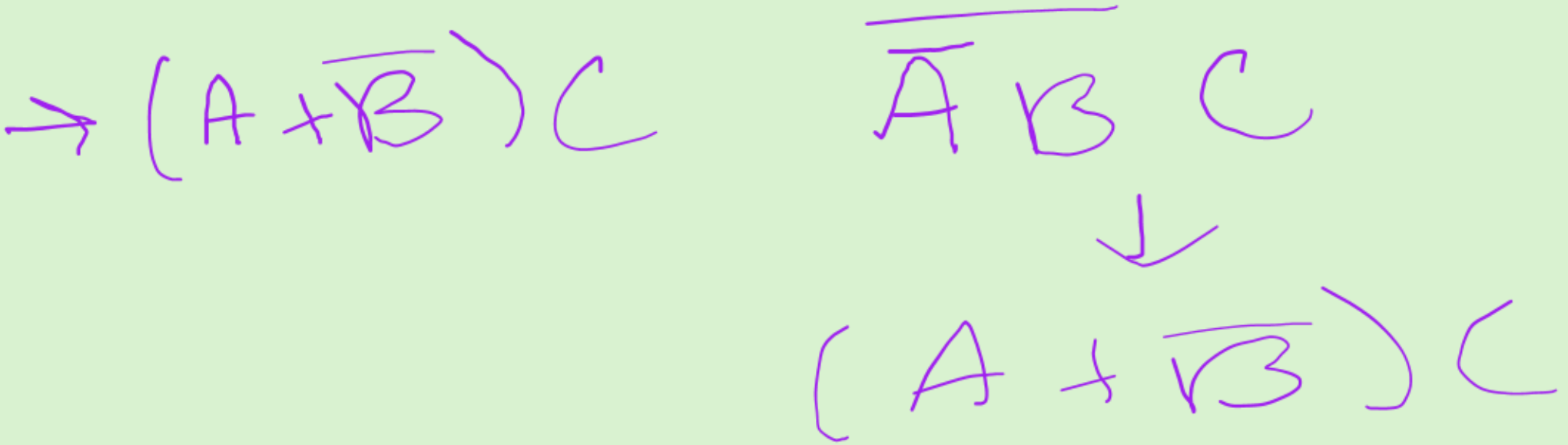
(1) $(\overline{AB})(\overline{CD})$

(2) $(\bar{A} + \bar{B})(\bar{C} + \bar{D})$

# DeMorgan's Practice

$$\overline{\bar{A}B + \bar{C}}$$

- How would you apply DeMorgan's to this expression?



# DeMorgan's Practice

$$\overline{\bar{A}B + \bar{C}}$$

- How would you apply DeMorgan's to this expression?

$$\overline{\bar{A}B}C$$

$$(A + \bar{B})C$$

# DeMorgan's Practice

- How would you apply DeMorgan's to this expression?

$$\overline{(\bar{A} + B)(C + D)\overline{(E + F)} + G}$$

$$\overline{X + Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{X\,Y} = \overline{X} + \overline{Y}$$

# DeMorgan's Practice

- How would you apply DeMorgan's to this expression?

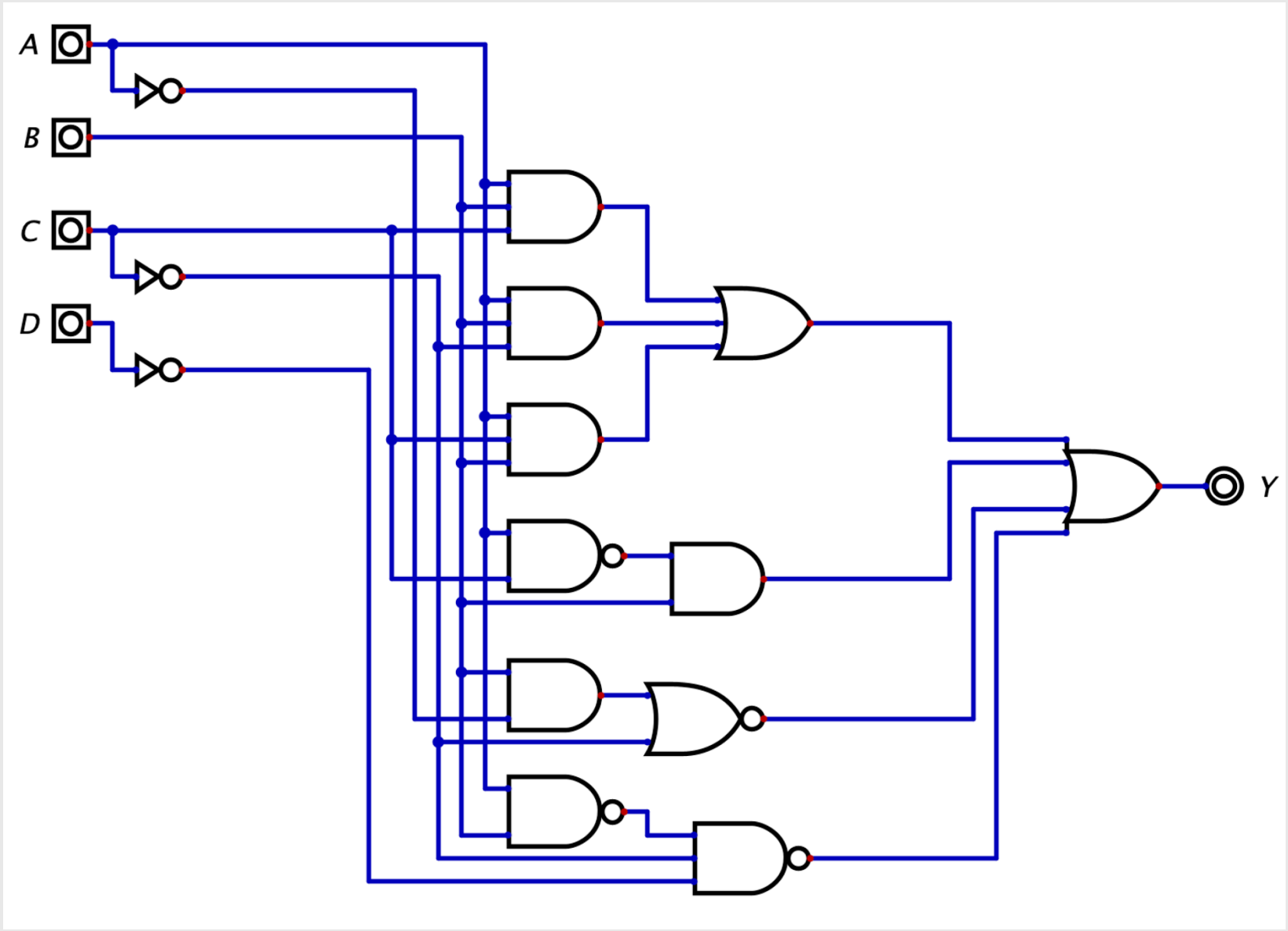
$$\overline{(\bar{A} + B)(C + D)\overline{(E + F)} + G}$$

$$\overline{(\bar{A} + B)(C + D)\overline{(E + F)}} \cdot \bar{G}$$

$$(\overline{\bar{A} + B} + \overline{C + D} + E + F) \cdot \bar{G}$$

$$(A\bar{B} + \bar{C}\bar{D} + E + F) \cdot \bar{G}$$

# Can we optimize this?



# Simplify the following Boolean expression

$$ABC + AB\bar{C} + \overline{AC}B + ACB + \overline{\overline{\overline{AB}\,\overline{C}\,\overline{D}}} + \overline{\overline{A}B + \bar{C}}$$

# Solution

$$ABC + AB\bar{C} + \overline{AC}B + ACB + \overline{\overline{AB}\,\overline{C}\,\overline{D}} + \overline{\bar{A}B + \bar{C}}$$

$$AB(C + \bar{C}) + B(\overline{AC} + AC) + AB + C + D + (\overline{\bar{A}B})C$$

$$AB(1) + B(1) + AB + C + D + (A + \bar{B})C$$

$$AB + B + AB + C + D + AC + \bar{B}C$$

$$AB + B + C + D + AC + \bar{B}C$$

$$B(A + 1) + C(1 + A) + \bar{B}C + D$$

$$B(1) + C(1) + \bar{B}C + D$$

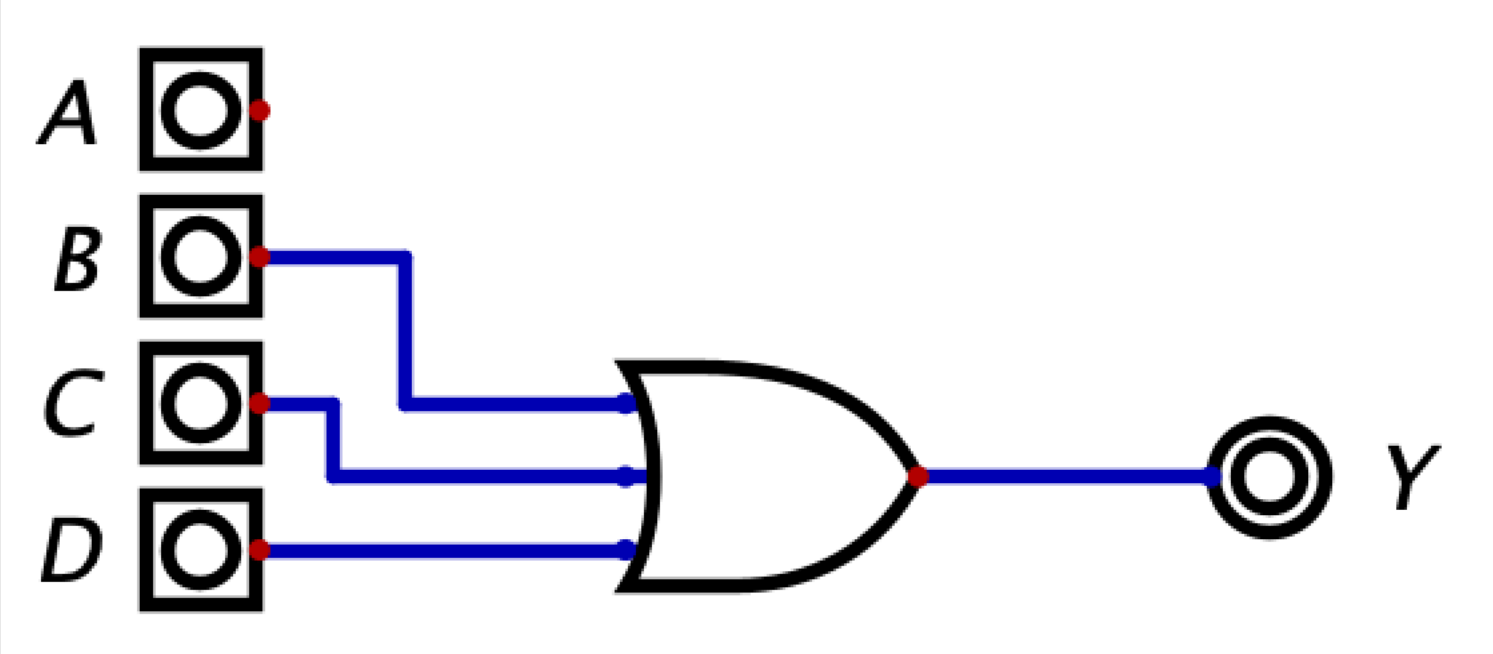
$$B + C + \bar{B}C + D$$

$$B + C(1 + \bar{B}) + D$$

$$B + C(1) + D$$

$$B + C + D$$

# Optimizing Our Gates




Yay! We did it!

# Recall: Our Problem from Earlier

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

I want to build a circuit that will output 1 if A > B, and 0 if A <= B. A and B are both two-bit unsigned numbers.

Additionally, define $A_1$ and $B_1$ as the left bits of A and B, and $A_0$ and $B_0$ as the right bits.

$$\bar{A}_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 B_0 + A_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 \bar{B}_0$$

*Is the simplest possible form??* 🧐

# Simplify?

$$\bar{A}_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{A}_0 \bar{B}_1 B_0 + A_1 A_0 \bar{B}_1 \bar{B}_0 + A_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 \bar{B}_0$$

# Our Problem from Earlier

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

I want to build a circuit that will output 1 if A > B, and 0 if A <= B. A and B are both two-bit unsigned numbers.

Additionally, define $A_1$ and $B_1$ as the left bits of A and B, and $A_0$ and $B_0$ as the right bits.

$$A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$$