BOOLEAN LOGIC

Michael D’Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP

LEVELS OF ABSTRACTION

LEVELS OF ABSTRACTION

Software

↓

Programmable Logic

↓

Hardware

Fundamental Theorem of Software Engineering:

"We can solve any problem by introducing an extra level of indirection." – Andrew Koenig

INTRO TO LOGIC GATES

Hardware Level: Transistors

- Transistors are electrical switches that either allow current to flow or not. i.e. they have two states.
- Binary - a data representation that uses two symbols: 0 and 1
- Boolean - a data type that consists of only two possible values:
  - 0 = FALSE = OFF = 
  - 1 = TRUE = ON = 
- Why two states?

Programmable Logic Level: Gates

- Gates are composed of multiple transistors.
- Called "gates" because they switch current on or off.
- Based on the inputs, Boolean operations either produce a TRUE (1) or FALSE (0) result.
- Basic Boolean logic operations
  - AND
  - OR
  - NOT

Truth Tables

- A way of representing the result of a Boolean operation.
- Inputs are shown on the left hand side.
- Output is shown on the right hand side.

Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Truth Tables: 2 Truths and a Lie

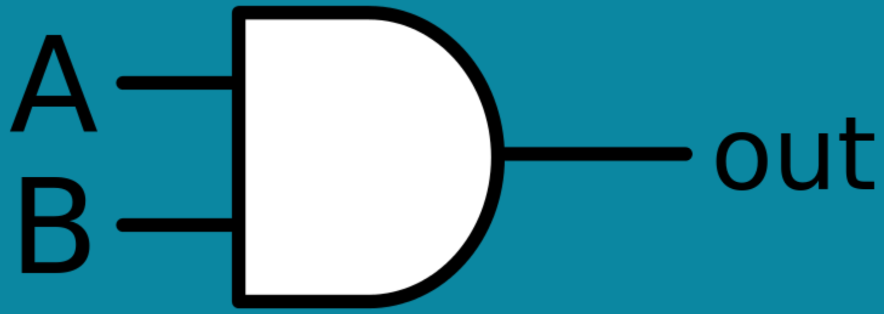
- Truths = 1; Lie = 0
- Complete the truth table to have an output of 1 if the inputs are correct (i.e. 2 truths and 1 lie).

Input A	Input B	Input C	Output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Tables: 2 Truths and a Lie

- Truths = 1; Lie = 0
- Complete truth table to have an output of 1 if the inputs are correct.

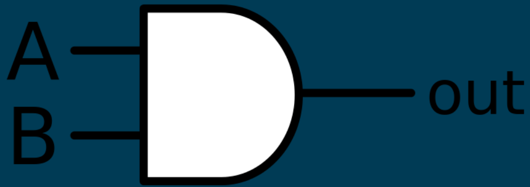
Input A	Input B	Input C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



AND

What is an AND gate?

AND



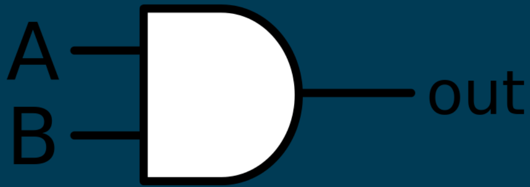
- Exp: We need a hammer AND nails to hang a picture frame.
- We can't just have one.

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

- For the AND operation to be true, we need both to be true.

- Known as a logical conjunction.

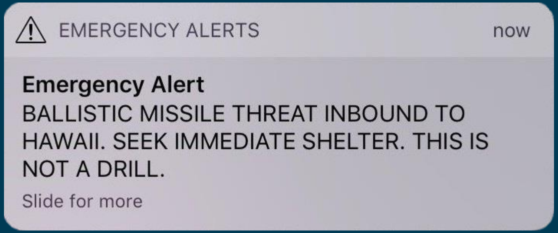
AND

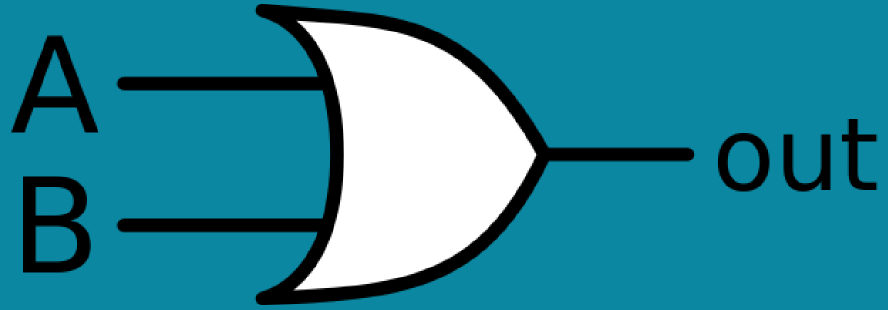


- Exp: We need a hammer AND nails to hang a picture frame.
- We can't just have one.

Input A	Input B	Output
0	0	0
0	1	0
1	0	0
1	1	1

- For the AND operation to be true, we need both to be true.
- Known as a logical conjunction.

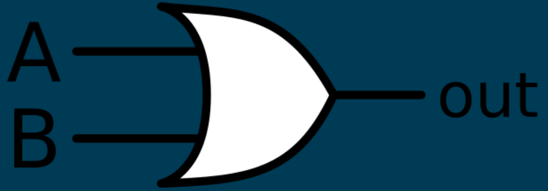




OR

What is an OR gate?

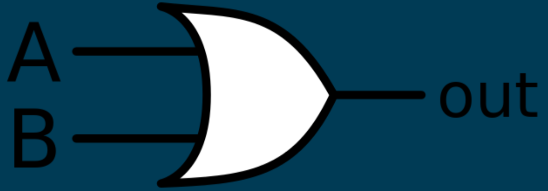
OR



- Exp: "If you need anything, don't hesitate to call or drop by."
- You can do one, the other, or both.
- Considered Inclusive OR.
- For the OR operation to be true, we only need 1 to be true but both can be true.
- Known as a logical disjunction.

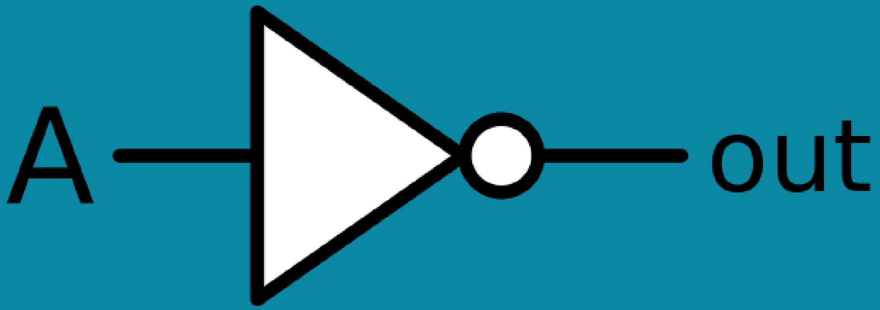
Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

OR



- Exp: “If you need anything, don’t hesitate to call or drop by.”
- You can do one, the other, or both.
- Considered Inclusive OR.
- For the OR operation to be true, we only need 1 to be true but both can be true.
- Known as a logical disjunction.

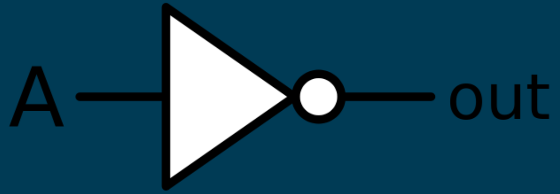
Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	1



NOT

What is a NOT gate?

NOT

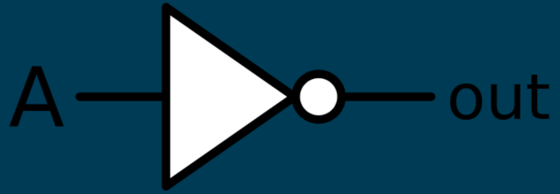


- Exp: If the sun is shining, you do not need an umbrella.

Input A	Output
0	
1	

- This operation negates or inverts the value.
- Also known as an inverter.

NOT



- Exp: If the sun is shining, you do not need an umbrella.
- This operation negates or inverts the value.
- Also known as an inverter.

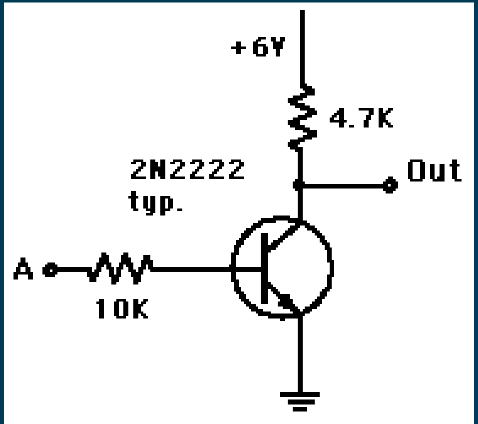
Input A	Output
0	1
1	0

Activity: Designing BJT Logic Gates

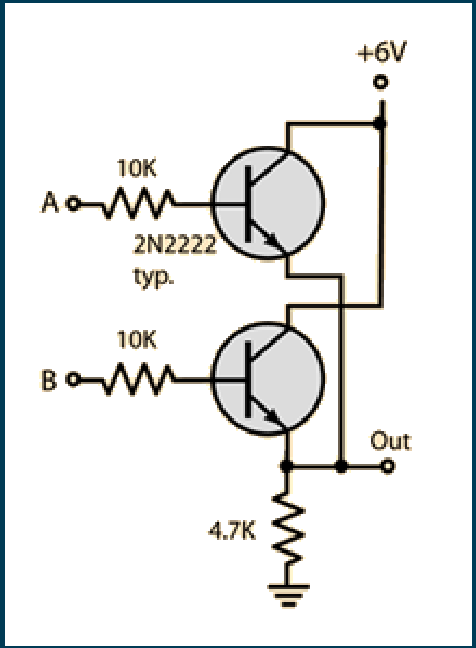
- How would you construct the three basic logic gates (NOT, OR, AND) out of npn BJTs?
- Think about their truth tables!

Activity: Designing BJT Logic Gates

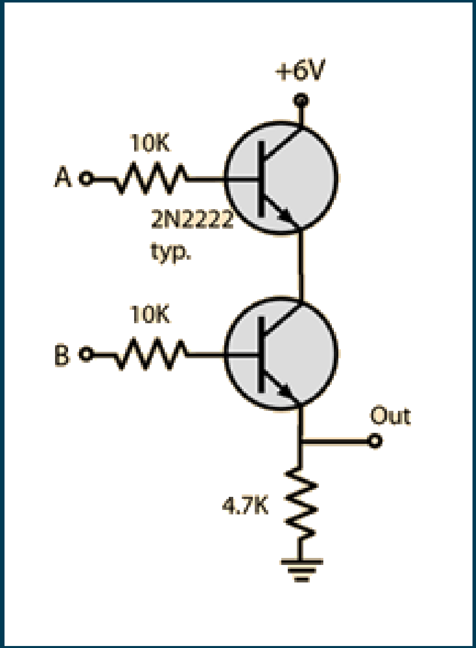
NOT



OR



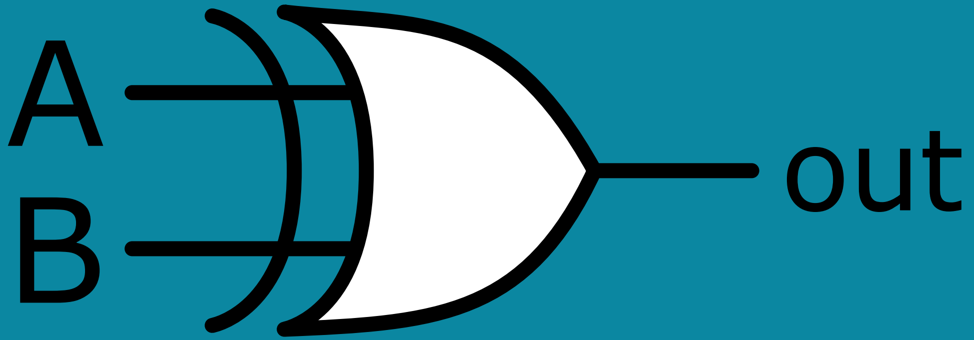
AND



3 Basic Boolean Logic Operations

- From these, we can perform any computation.

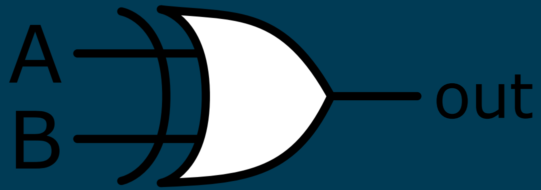
- But before we discuss that, there are also a few more important logic gates that can be constructed from these 3 basic logic gates.



XOR

What is an exclusive or (XOR) gate?

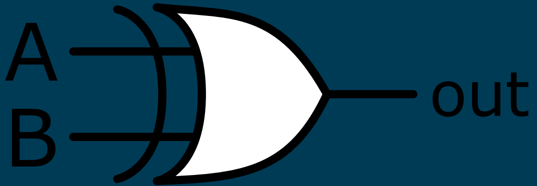
XOR



- Exp: You can go to Au Bon Pain or Sazon for breakfast. You can’t go to both.
- You can do one, the other, but you can’t do both.
- Considered Exclusive OR.
- For the XOR operation to be true, we need 1 and only 1 to be true.

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

XOR



- Exp: You can go to Au Bon Pain or Sazon for breakfast. You can’t go to both.
- You can do one, the other, but you can’t do both.
- Considered Exclusive OR.
- For the XOR operation to be true, we need 1 and only 1 to be true.

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Activity: Design an XOR Gate

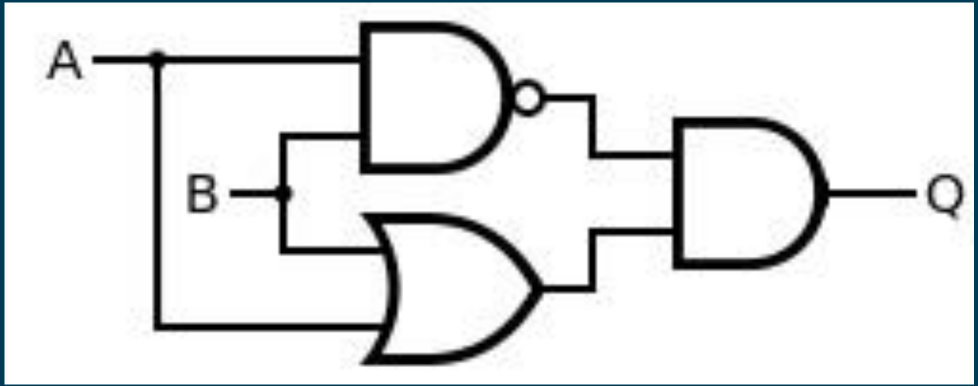
- How would you build an XOR gate?

- Use only the 3 basic Boolean logic gates
  - AND
  - OR
  - NOT

- Try starting with the truth table.

XOR Gate Construction

- Many ways to construct it, but here is one way.

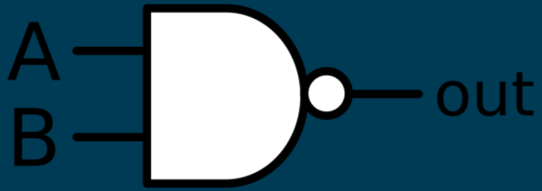


INVERTED GATES

Inverted Gates

- There also exists inverted or negated logic gates.
- They are:
  - NAND – negated and
  - NOR – negated inclusive or
  - XNOR – negated exclusive or
- Can you draw the truth tables for these?

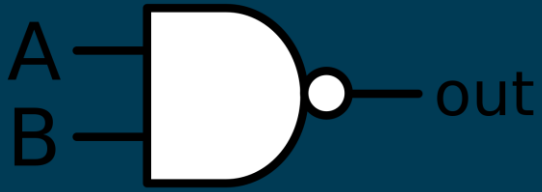
NAND



- Negated AND gate.
- For the NAND operation to be true, we need 1 input to be false.

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

NAND

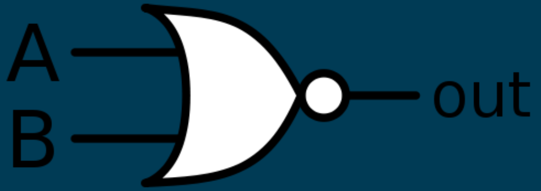


- Negated AND gate.

- For the NAND operation to be true, we need 1 input to be false.

Input A	Input B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR

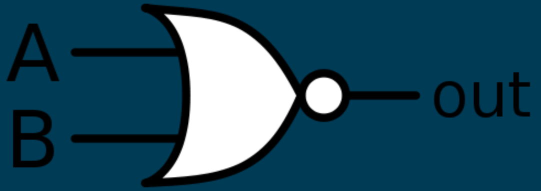


- Negated OR gate.

- For the NAND operation to be true, we need both inputs to be false.

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

NOR

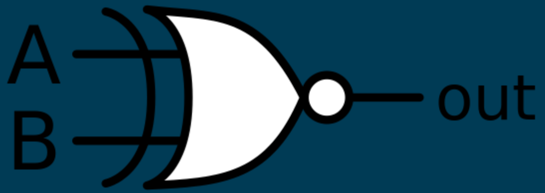


- Negated OR gate.

- For the NAND operation to be true, we need both inputs to be false.

Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	0

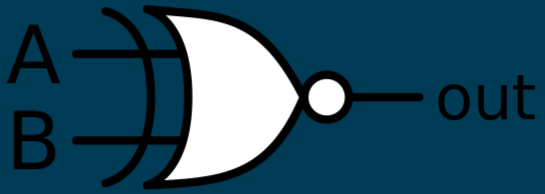
XNOR



- Negated XOR gate.
- For the XNOR operation to be true, we need both inputs to be false OR both inputs to be true.

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

XNOR



- Negated XOR gate.
- For the XNOR operation to be true, we need both inputs to be false OR both inputs to be true.

Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	1

NOW WHAT?

- How do we get from logic gates to software?
- How do we construct something useful?
- How is a computer built out of 3 logic operations?

Software

Programmable Logic

Hardware

BOOLEAN ALGEBRA

Conventions

- NOT: $\bar{A}$ is A inverted
  - Exp. $\bar{A} \cdot B$
- OR: $+$
  - Exp. $A + B$
- AND: $\cdot$
  - Exp. $A \cdot B$

Boolean Algebra Laws

- Annulment Law – a term AND'ed with a "0" equals 0 or OR'ed with a "1" will equal 1.
  - $A \cdot 0 = 0$
  - $A + 1 = 1$
- Identity Law – a term OR'ed with a "0" or AND'ed with a "1" will always equal that term.
  - $A + 0 = A$
  - $A \cdot 1 = A$
- Idempotent Law – An input that is AND'ed or OR'ed with itself is equal to that input.
  - $A + A = A$
  - $A \cdot A = A$

Boolean Algebra Laws (contd.)

- Complement Law – A term AND´ed with its complement equals "0" and a term OR´ed with its complement equals "1".
  - $A \cdot \bar{A} = 0$
  - $A + \bar{A} = 1$
- Commutative Law – The order of application of two separate terms is not important.
  - $A \cdot B = B \cdot A$
  - $A + B = B + A$
- Double Negation Law – A term that is inverted twice is equal to the original term.
  - $\bar{\bar{A}} = A$

Boolean Algebra Laws (contd.)

- de Morgan's Law

1. Two separate terms NOR'ed together is the same as the two terms inverted (Complement) and AND'ed.
   - Example: $\overline{A+B} = \bar{A} \cdot \bar{B}$
2. Two separate terms NAND'ed together is the same as the two terms inverted (Complement) and OR'ed.
   - Example: $\overline{A \cdot B} = \bar{A} + \bar{B}$

Boolean Algebra Laws (contd.)

- Distributive Law – This law permits the multiplying or factoring out of an expression.
  - A.(B + C) = A.B + A.C (OR Distributive Law)
  - A + (B.C) = (A + B).(A + C) (AND Distributive Law)
- Absorptive Law – This law enables a simplification of an expression by absorbing like terms.
  - A + (A.B) = A (OR Absorption Law)
  - A(A + B) = A (AND Absorption Law)
- Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables.
  - A + (B + C) = (A + B) + C = A + B + C (OR Associate Law)
  - A(B.C) = (A.B)C = A . B . C (AND Associate Law)

TRUTH TABLES TO BOOLEAN EXPRESSIONS

Truth Tables to Boolean Expressions

- Construct expressions line by line when the output is true.
- Example: if the input 111 resulted in a true output, the expression for that line would be $A \cdot B \cdot C = 1$.
- What is the resulting Boolean expression for the 2 truths and a lie truth table on the right?

Input A	Input B	Input C	Output
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	0

Example: 2 Truths and a Lie

- $Output = \bar{A}BC + A\bar{B}C + AB\bar{C}$
- How would we simplify this?

Input A	Input B	Input C	Output
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	0

Example: 2 Truths and a Lie

- We can't!
- Many times you will try to apply the Boolean algebra laws in circles until you find it's not able to be simplified.

Example: Boolean Algebra

- How would you simplify this?

A	B	C	Output
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

A	B	C	Output	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC = 1$
1	0	0	0	
1	0	1	1	$A\overline{B}C = 1$
1	1	0	1	$AB\overline{C} = 1$
1	1	1	1	$ABC = 1$

$$\text{Output} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Example Solution

$$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$\downarrow$ Factoring **BC** out of 1st and 4th terms

$$BC(\overline{A} + A) + A\overline{B}C + AB\overline{C}$$

$\downarrow$ Applying identity $\mathbf{A + \overline{A} = 1}$

$$BC(1) + A\overline{B}C + AB\overline{C}$$

$\downarrow$ Applying identity $\mathbf{1A = A}$

$$BC + A\overline{B}C + AB\overline{C}$$

$\downarrow$ Factoring **B** out of 1st and 3rd terms

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

$\downarrow$ Applying rule $\mathbf{A + \overline{A}B = A + B}$ to the $C + A\overline{C}$ term

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

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

$\downarrow$ Distributing terms

$$BC + AB + A\overline{B}C$$

$\downarrow$ Factoring **A** out of 2nd and 3rd terms

$$BC + A(B + \overline{B}C)$$

$\downarrow$ Applying rule $\mathbf{A + \overline{A}B = A + B}$ to the $B + \overline{B}C$ term

$$BC + A(B + C)$$

$\downarrow$ Distributing terms

$$BC + AB + AC$$

or Simplified result

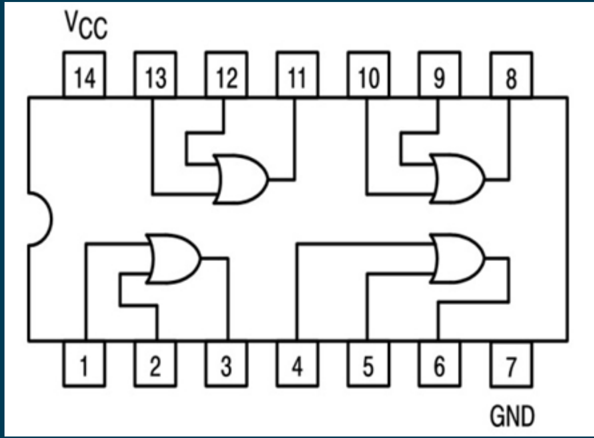
$$AB + BC + AC$$

ACTIVITIES

Logic Gates

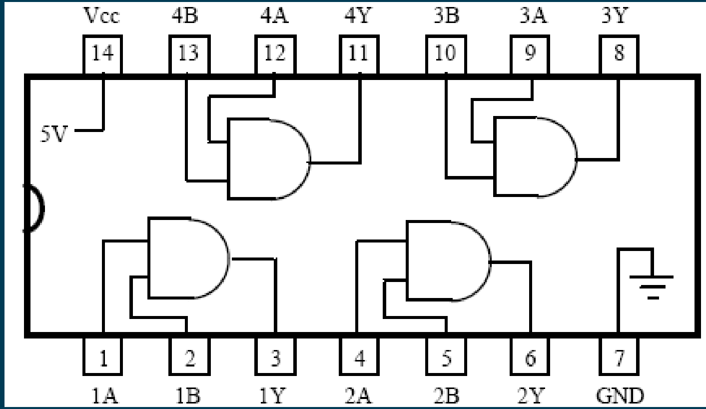
- SN74LS32N – OR Gate – 2 inputs – 4 channels
- SN74LS08N – AND Gate – 2 inputs – 4 channels
- SN74LS04N – NOT Gate – 1 input – 6 channels

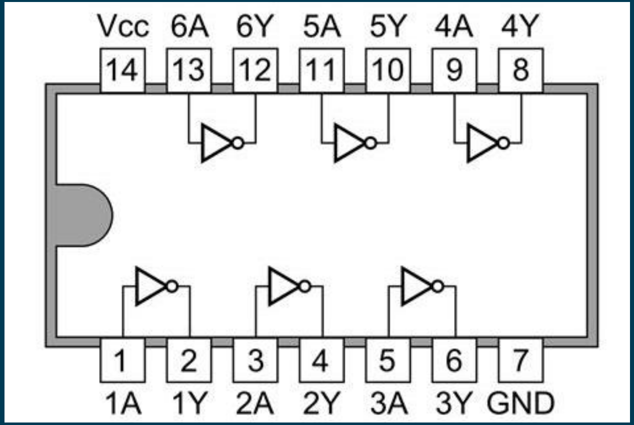
Logic Gate Pinouts



OR

AND





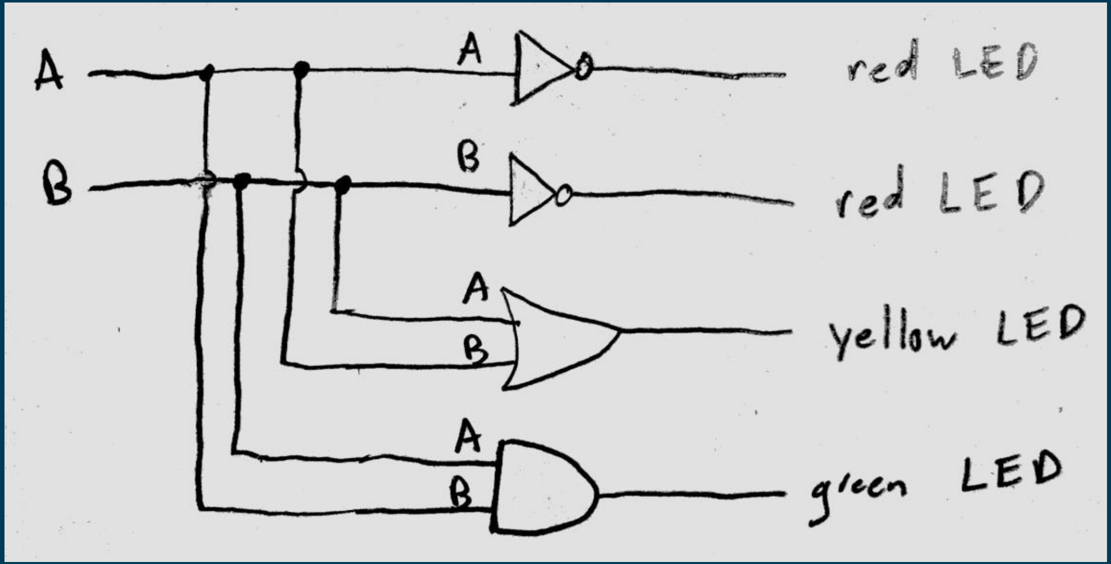
NOT

How should we set up the inputs?

- How do we wire the switch?
- Can't leave it floating. Need to provide discrete inputs: logic low and logic high.
- Should limit voltage to protect LEDs.

Activity: Building 3 Basic Logic Ops

- Construct a circuit that has 2 inputs and an output from each gate.
- It should turn on an LED when the output is high.



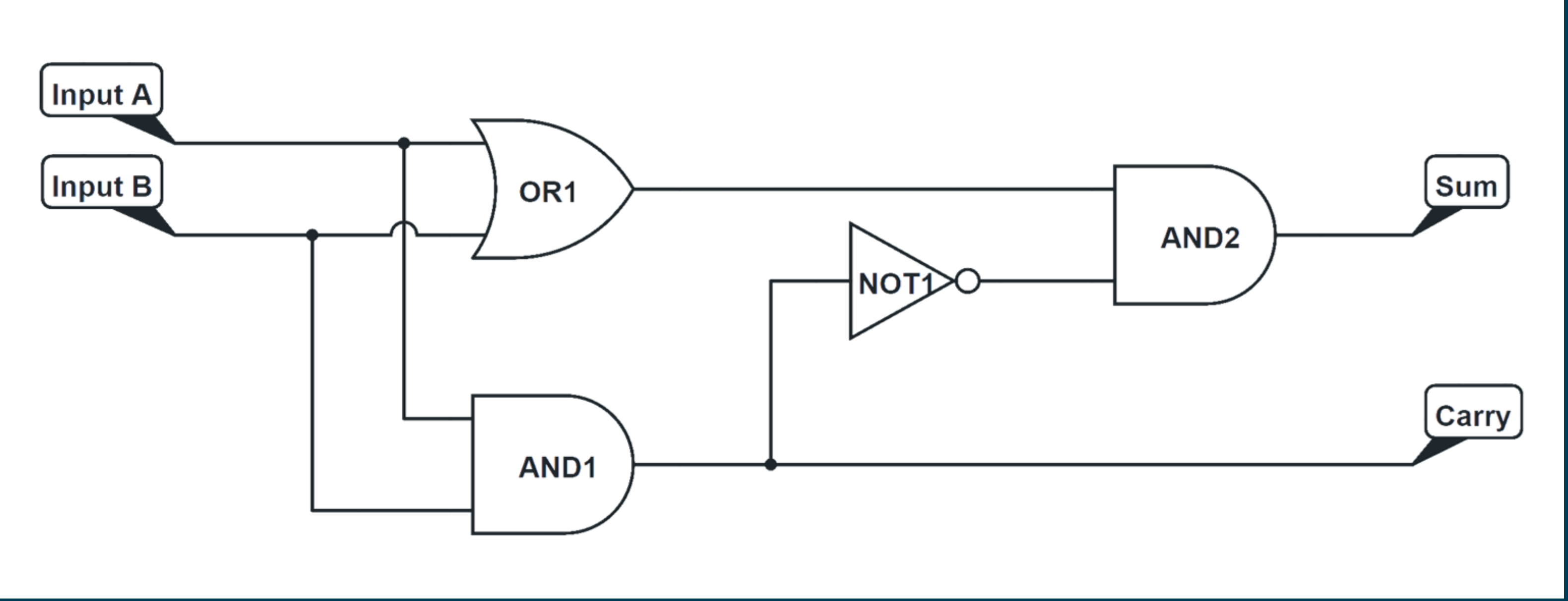
Activity: Adder

- Construct an adder using AND, OR, NOT logic gates to add a 1-bit number to another 1-bit number.
- Start by drawing the truth table!
- Construct it to test.
- Adding a 1-bit number to a 1-bit number results in a 2-bit output.
- Drive 2 LEDs to show the 2 output bits.

Activity: Adder

- This is called a half adder!

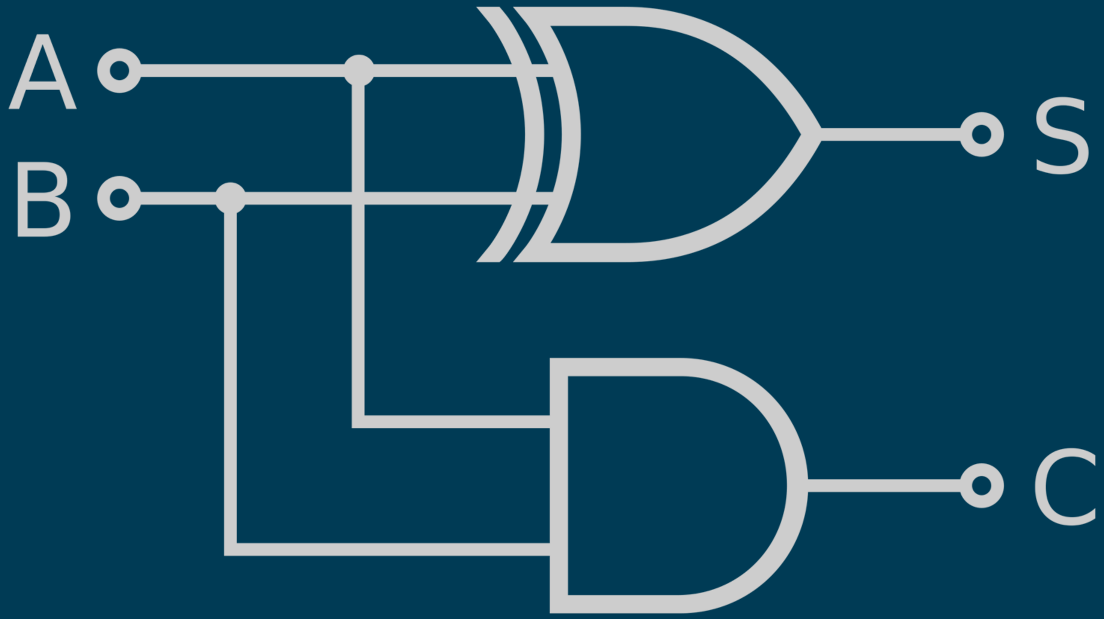
Input A	Input B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Half Adder

- More easily constructed with an XOR gate.

Input A	Input B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

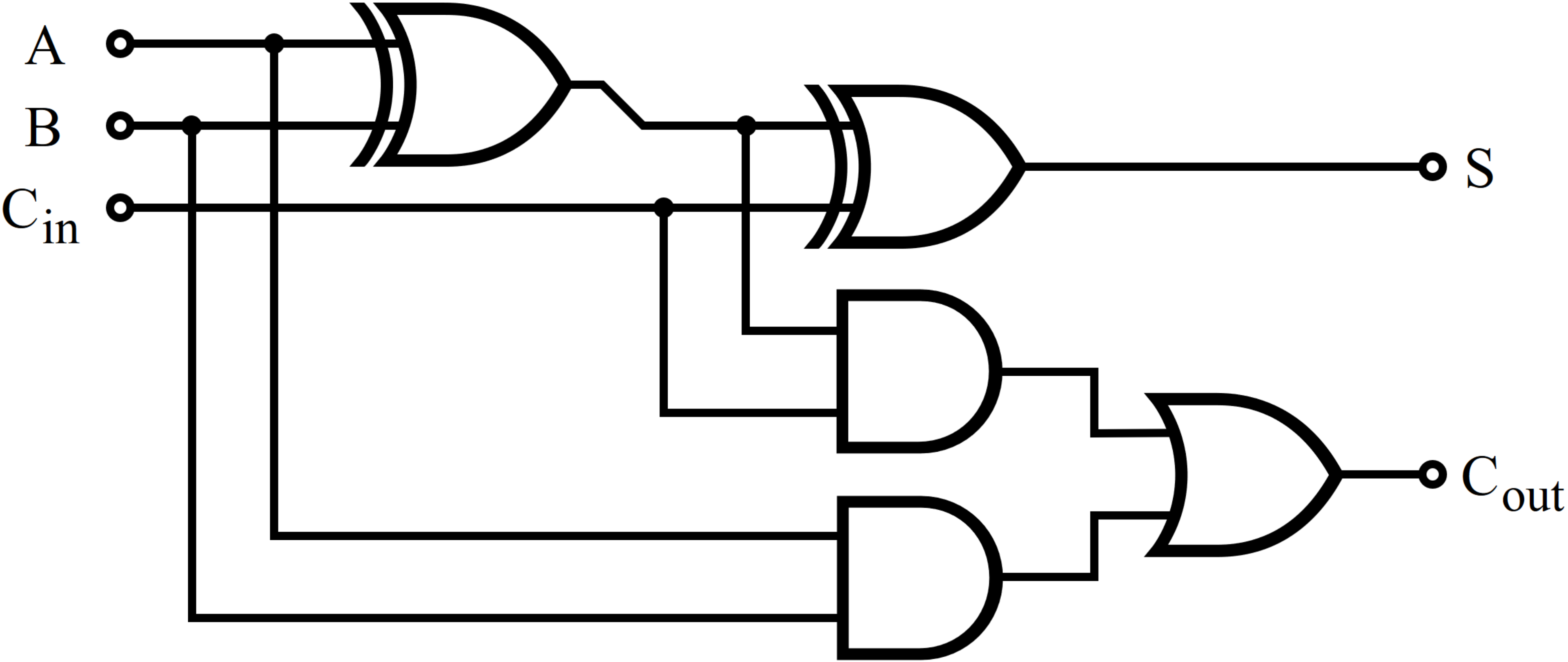


Activity: Full Adder

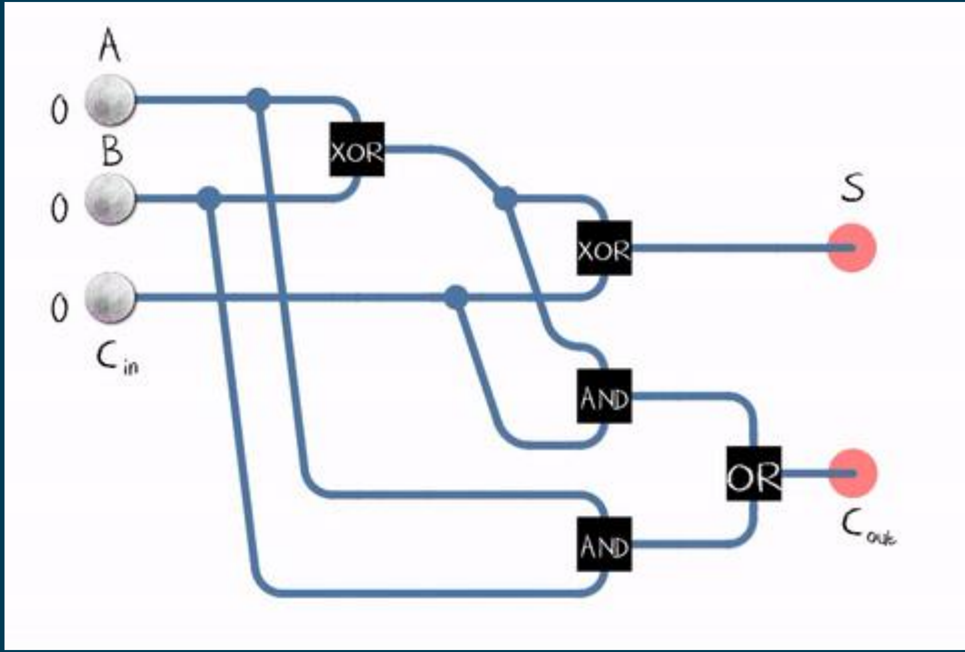
- If that's a half-adder, presumably there's a full adder.
- What would it look like?
- What would it allow us to do?

- Hint: how could we chain adders together to be able to add more bits at a time?

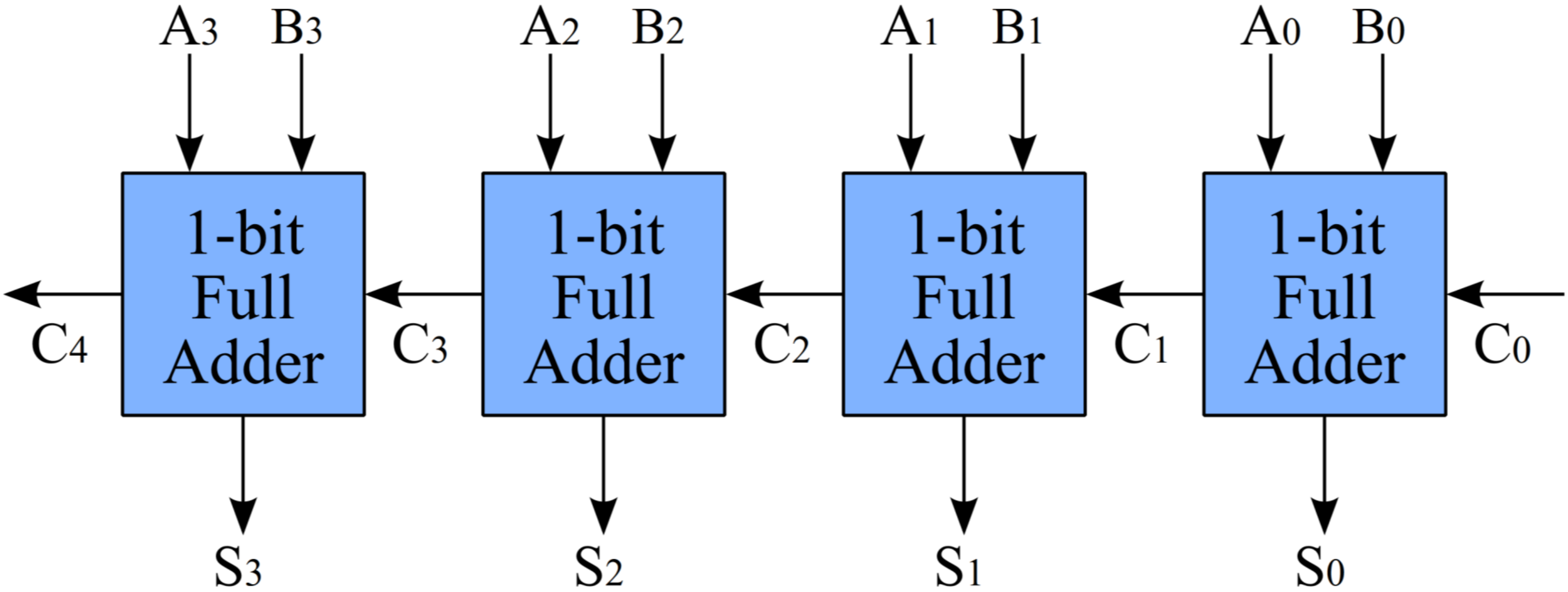
Activity: Full Adder



Full Adder



4-bit Adder



A3
B3
A2
B2
A1
B1
A0
B0
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
C4
C3
C2
C1
C0
S3
S2
S1
S0

COMBINATIONAL AND SEQUENTIAL LOGIC

Types of Circuit Logic

- You just constructed your first pieces of logic.
- These pieces can be used to construct an Arithmetic Logic Unit (ALU).
- The logic you just constructed is considered *combinational.*
- There are two types of logic we will use to construct the basis for computing.

Building Functions from Logic Gates

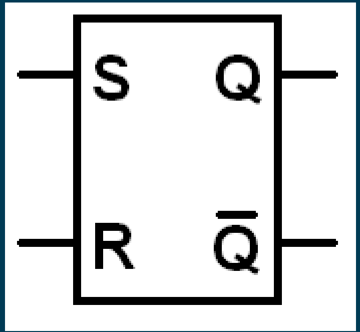
- From the Boolean logic gates/operations we discussed, we can construct two different types of logic.
- *Combinational Logic Circuit*
  - output depends only on the current inputs
  - stateless
- *Sequential Logic Circuit*
  - output depends on the sequence of inputs (past and present)
  - stores information (state) from past inputs

Combinational vs. Sequential

- Combinational Circuit - output depends only on inputs
  - Always gives the same output for a given set of inputs
  - Example: adder always generates sum and carry, regardless of previous inputs
- Sequential Circuit - output depends on stored information (state) plus inputs
  - Stores information - so a given input might produce different outputs, depending on the stored information
  - Example: “volume up” button. Increases the volume, but does so in relation to the present volume level.
  - Useful for building memory elements and state machines.

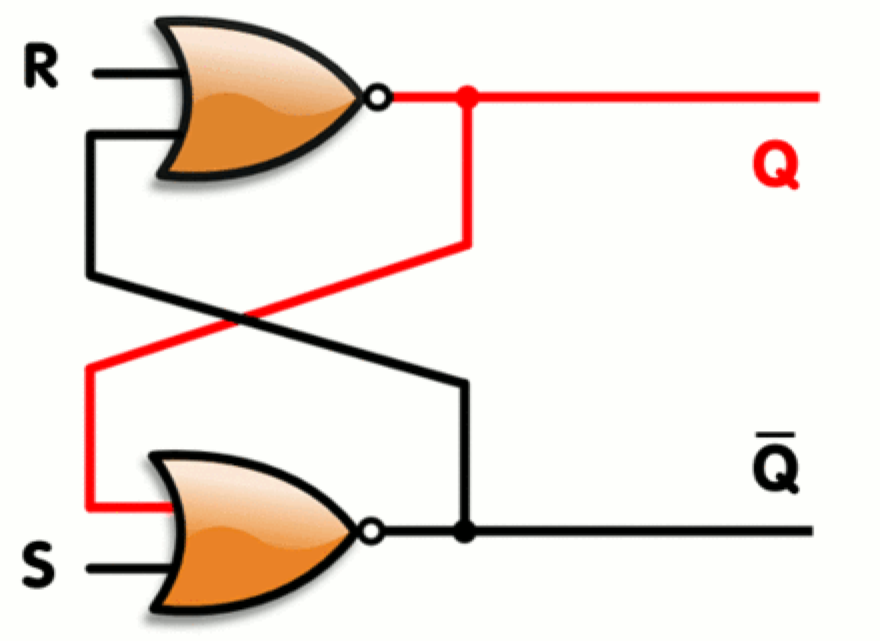
S-R Latch: Simple Storage Element

- R is used to “reset” the element – set it to 0.
- S is used to “set” the element – set it to 1.
- If both R and S are 1, Q could be <u>either</u> 0 or 1.
  - “quiescent” state - holds its previous value



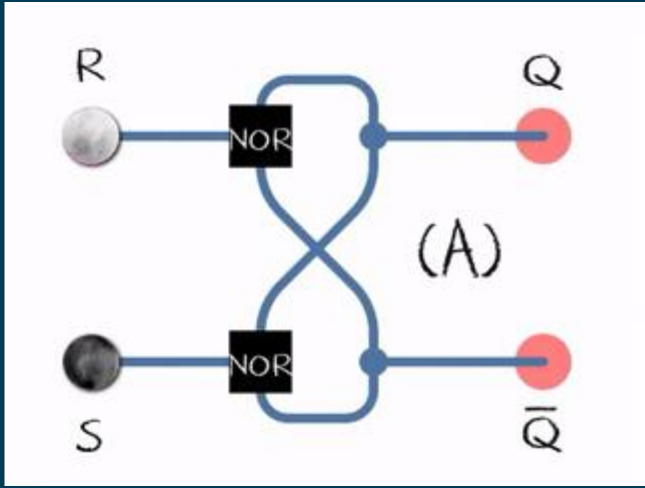
SR Latch or Flip Flop

- A latch circuit that has two stable states and can be used to store information.
- S – Set, drives output Q high
- R – Reset, drives output Q low
- Q' – an inversion of output Q



S	R	Q
0	0	hold
0	1	0 (reset)
1	0	1 (set)
1	1	Forbidden

SR Latch

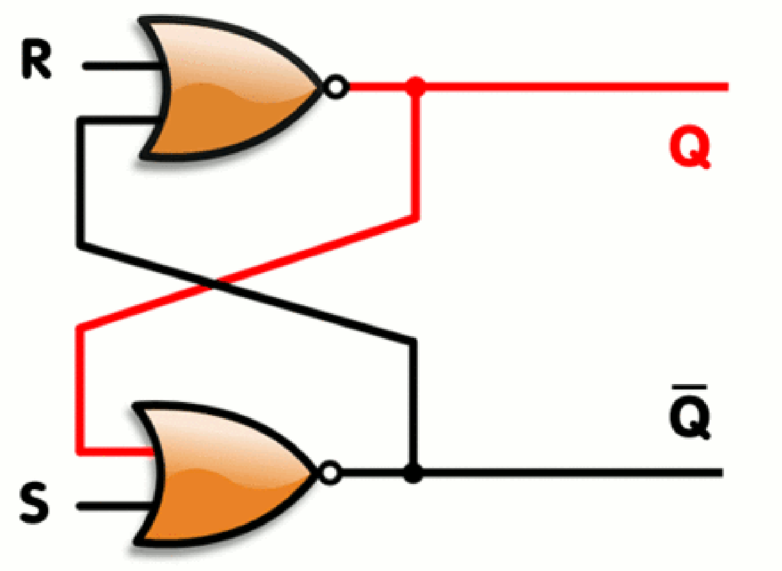


An animated SR latch. Black and white mean logical '1' and '0', respectively.
(A) S = 1, R = 0: set
(B) S = 0, R = 0: hold
(C) S = 0, R = 1: reset
(D) S = 1, R = 1: not allowed
Transitioning from the (D) to (A) leads to an unstable state.

Activity: SR Latch (Building Memory)

- Construct NOR gate out of OR and NOT gates.
- Drive an LED with your output to show that it holds state like a memory element.

S	R	Q
0	0	hold
0	1	0 (reset)
1	0	1 (set)
1	1	Forbidden



SOFTWARE LEVEL

LEVELS OF ABSTRACTION

Software

↓

Programmable Logic

↓

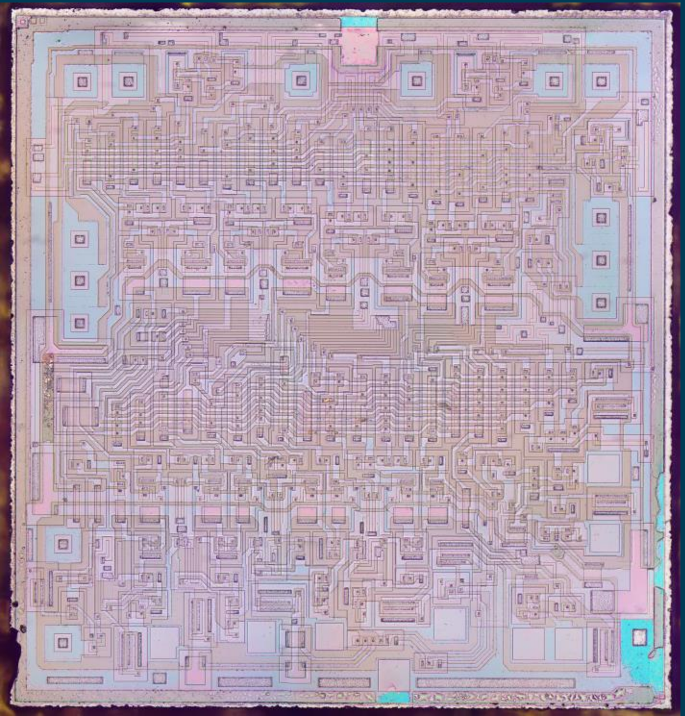
Hardware

Fundamental Theorem of Software Engineering:

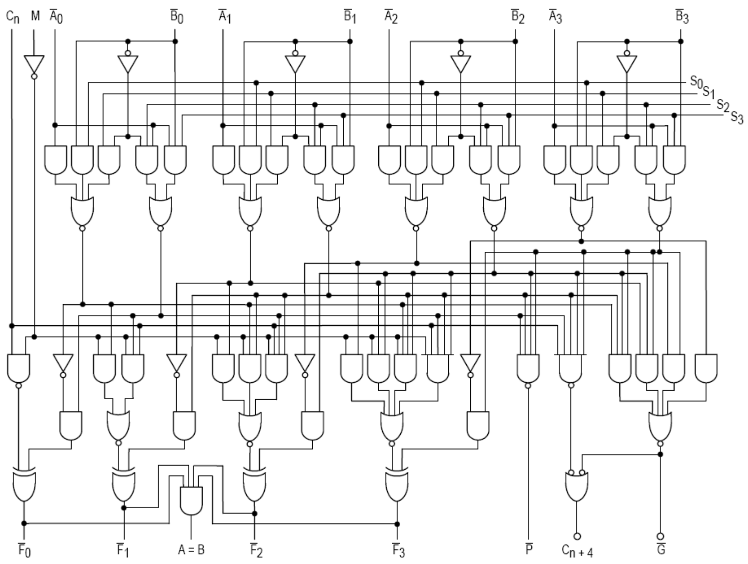
"We can solve any problem by introducing an extra level of indirection." – Andrew Koenig

ARITHMETIC LOGIC UNIT (ALU) - 74181

Hardware abstraction level



Programmable logic abstraction level



Software abstraction level

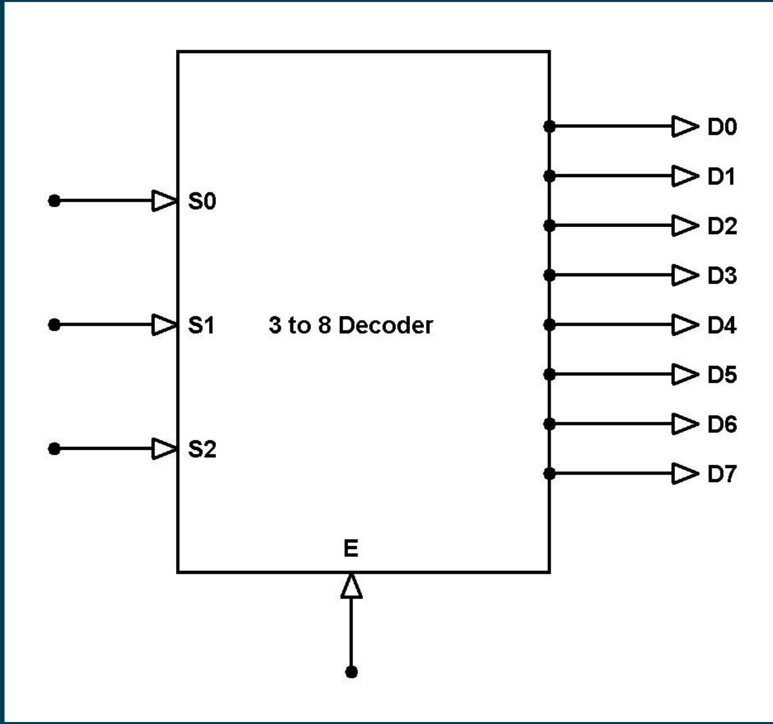
8 + 7 = ?

Or calculating pixel color and location

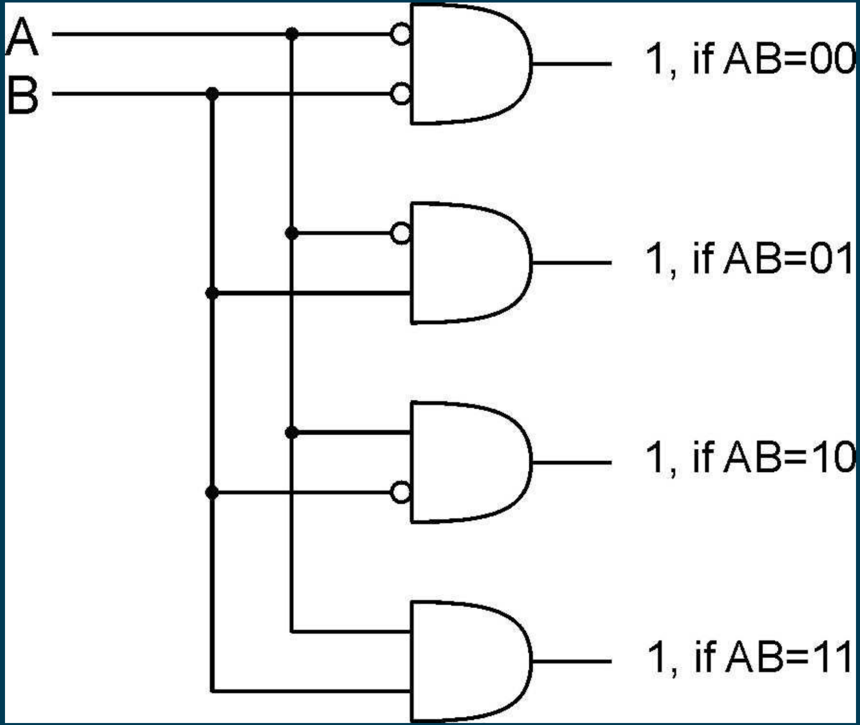
MORE ACTIVITIES

Activity: 3-Bit Binary Decoder

- 3 lines to 8 lines
- Could we use it to drive a 7-segment LED?

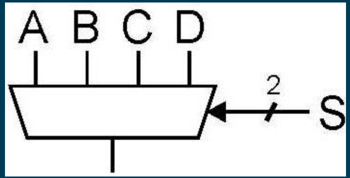


2-bit Decoder



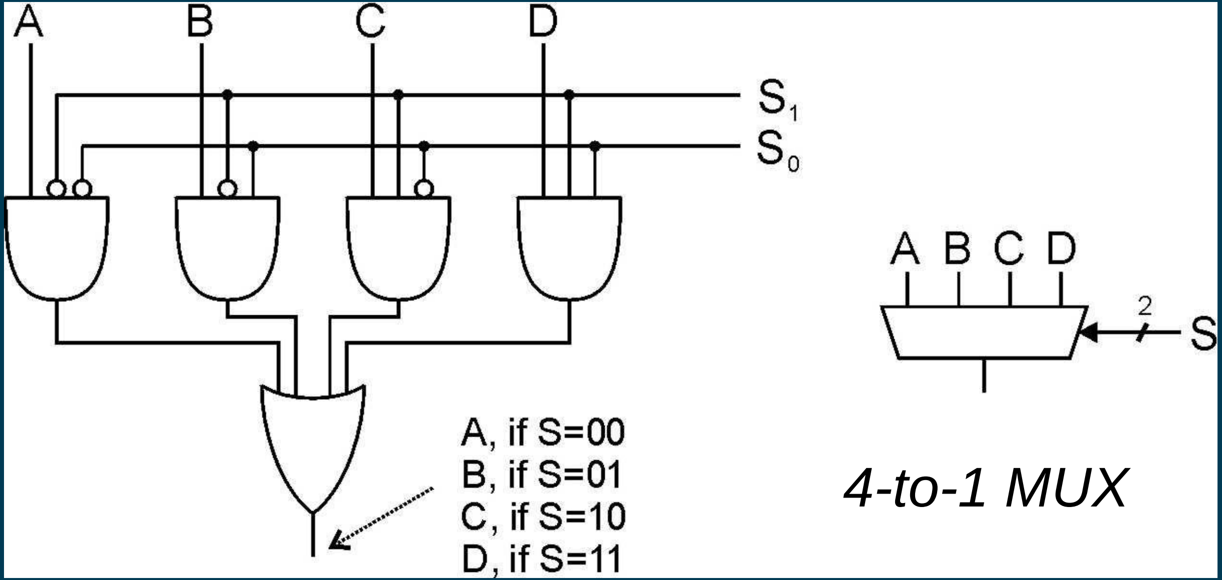
Activity: Multiplexer (MUX)

- *n*-bit selector and 2^n inputs, one output
- Output equals one of the inputs, depending on selector bits.
- Design a 4-to-1 Multiplexer



4-to-1 MUX

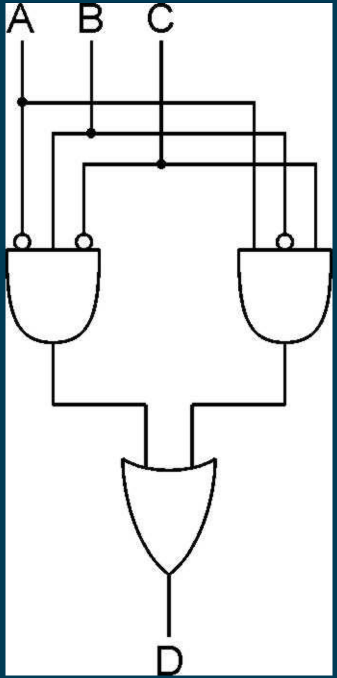
Multiplexer (MUX)



Logical Completeness

- Can implement ANY truth table with AND, OR, NOT.

A	B	C	D
0	0	0	0
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



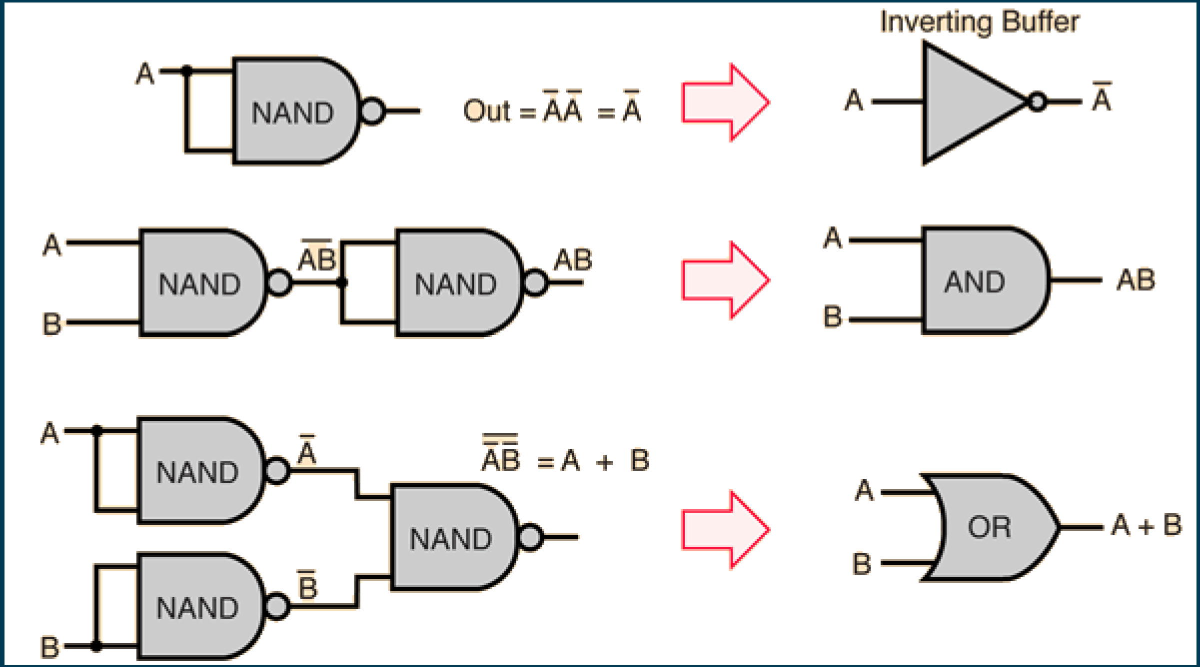
1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

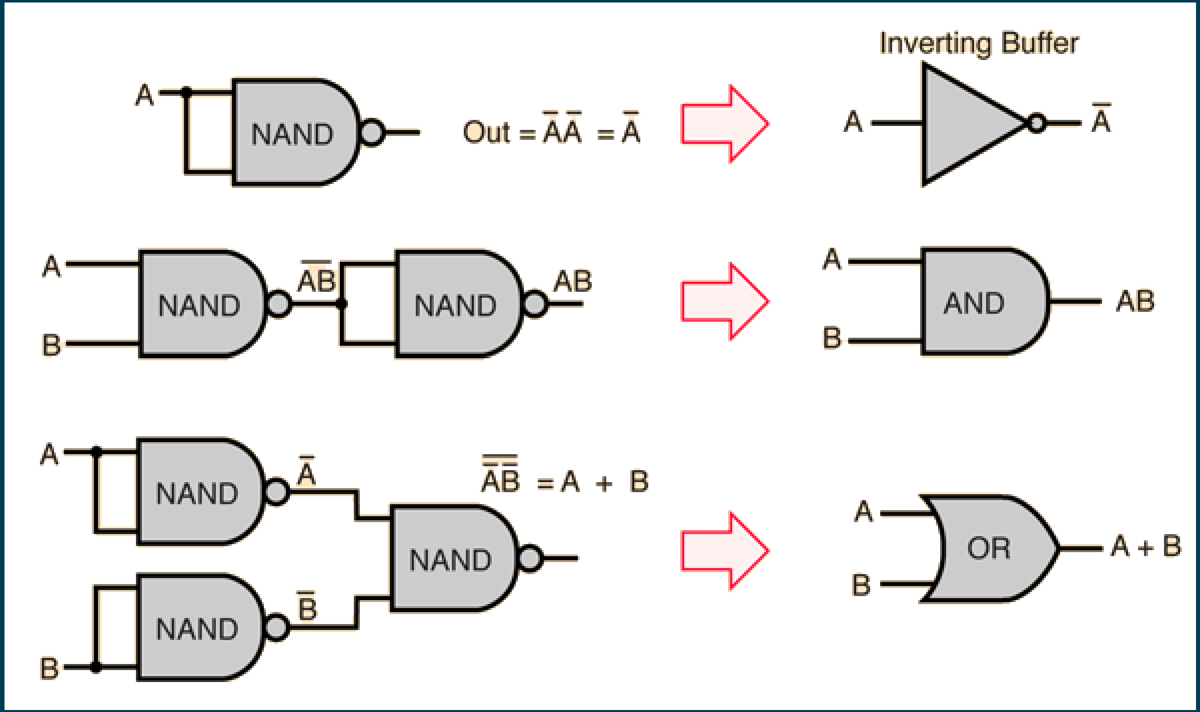
ADDITIONAL INFO

NAND Gates

- You can construct any logic possible using just NAND gates!



NAND Gate Equivalents



Activity: NAND Gate Adder

- How would you design a 2-bit adder using just NAND gates?

NAND Gates Adder Construction