Computer Architecture
Tutorial 3

# Combinational Logic Circuits

Artem Burmyakov, Alexander Tormasov

September 09, 2021

INNOPOLIS UNIVERSITY

# Terminology

## Terminology

| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |

Sample boolean variables:

$A, B, C, D, F \in \{0,1\}$

Expressions in one notation:

$F(A, B) = A * B$

$F(A, B, C) = A * B + C$

$F(A, B, C, D) = A * (B + C) + D$

Another notation:

$F(A, B) = \mathbf{AND}(A, B)$

$F(A, B, C) = \mathbf{OR}(\mathbf{AND}(A, B), C)$

$F(A, B, C, D) = \mathbf{OR}(\mathbf{AND}(A, \mathbf{OR}(B, C)), D)$

Other notations are used as well

## Terminology

| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |

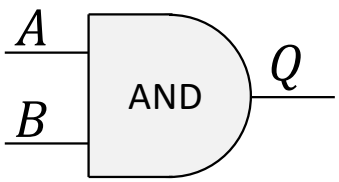
Truth table for function $F = A * B$:

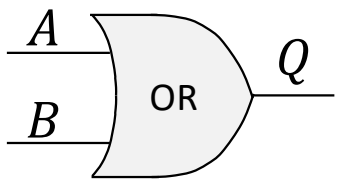
| $A$ | $B$ | $F = A * B$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

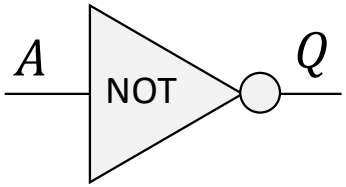
## Terminology

| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | An abstract or physical device implementing a Boolean function |

Sample logic gates:



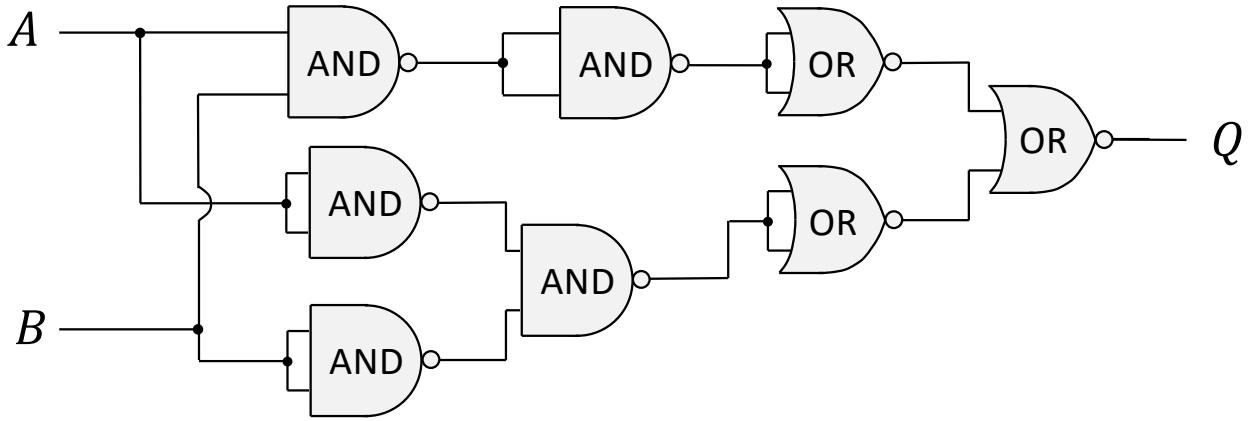



## Terminology

| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | An abstract or physical device implementing a Boolean function |
| **A logic circuit** | Composition of logic gates, to implement a certain function |

Sample logic circuit:

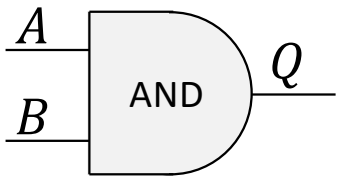


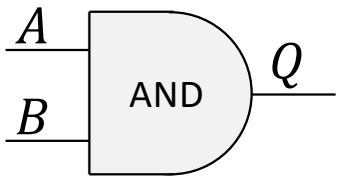
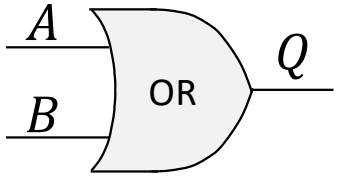
## Terminology

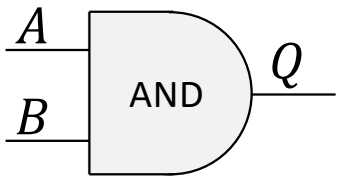
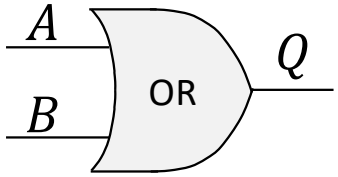
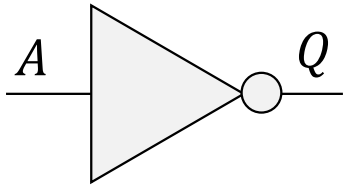
| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | An abstract or physical device implementing a Boolean function |
| **A logic circuit** | Composition of logic gates, to implement a certain function |
| **An Integrated Circuit**<br>(informally: chip, or microchip) | The antonym for a “discrete electrical circuit”, that is a set of discrete electrical devices, which can be assembled and reassembled again; |

## Terminology

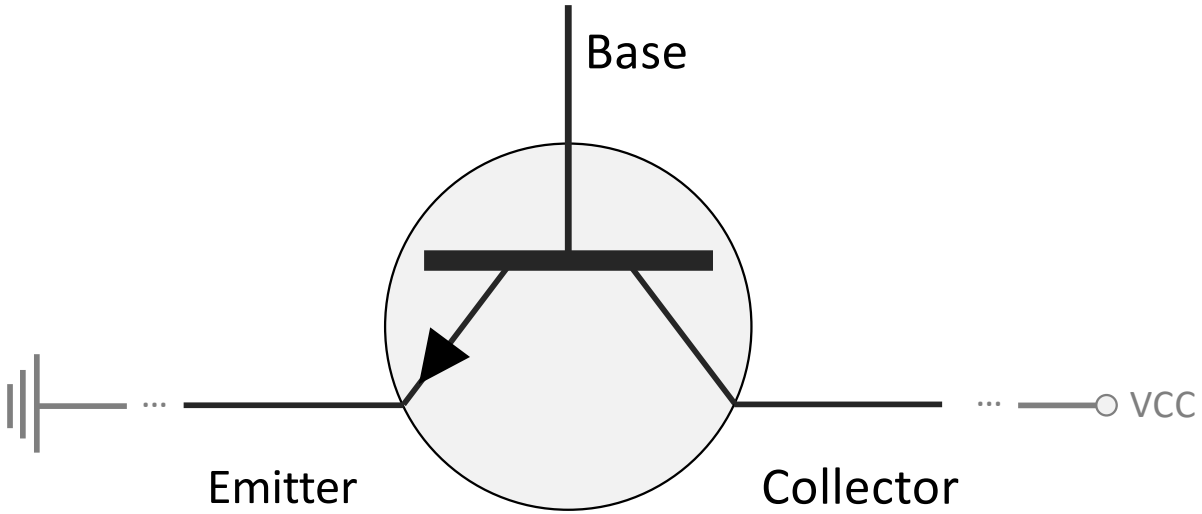
| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | An abstract or physical device implementing a Boolean function |
| **A logic circuit** | Composition of logic gates, to implement a certain function |
| **An Integrated Circuit**<br>(informally: chip, or microchip) | The antonym for a “discrete electrical circuit”, that is a set of discrete electrical devices, which can be assembled and reassembled again;<br>Key advantages over discrete circuits:<br>- Compact size;<br>- Performance (e.g. the propagation delay – to be discussed next);<br>- Lower manufacturing price |

| Logic Gate | Symbolic Representation | Truth Table |
|---|---|---|
| **AND** |   | <table><tr><th>*A*</th><th>*B*</th><th>*Q=AB*</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> |

<table>
<tr><th>Logic Gate</th><th>Symbolic Representation</th><th>Truth Table</th></tr>
<tr>
<td>AND</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
<tr>
<td>OR</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><th>Logic Gate</th><th>Symbolic Representation</th><th>Truth Table</th></tr>
<tr>
<td>AND</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
<tr>
<td>OR</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
<tr>
<td>NOT</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>Q=A’</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
</table>

Transistor: building block for logic gates

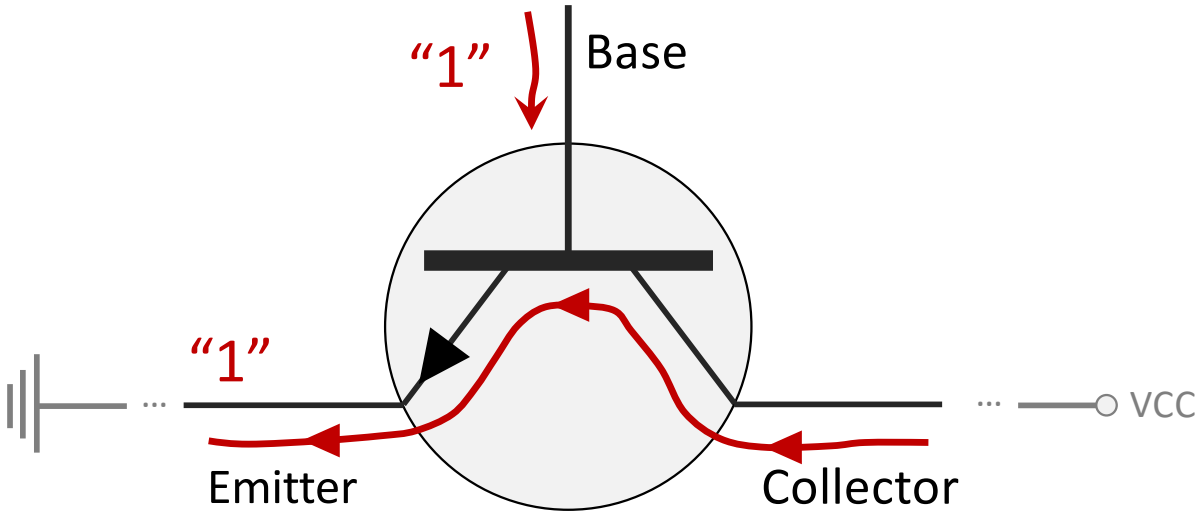



Transistor is a fast switch:

If there is voltage at the Base terminal (Base = “1”), then current flows between Emitter and Collector;

Otherwise (Base = “0”), there is no current

# Transistor: building block for logic gates



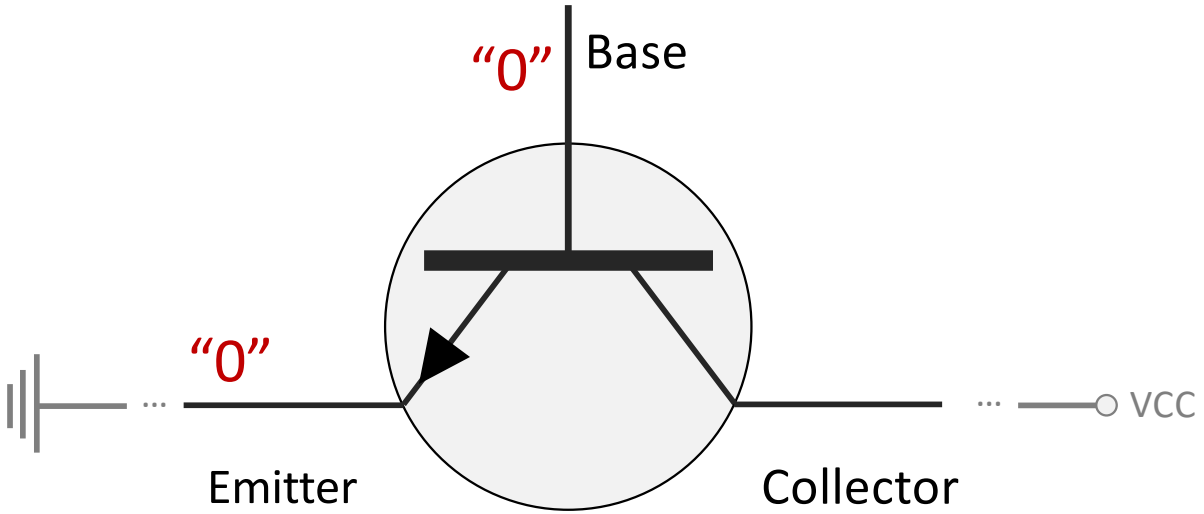

Note: Current direction depends on the transistor type (e.g. NPN or PNP)

Transistor is a fast switch:

If there is voltage at the Base terminal (Base = "1"), then current flows between Emitter and Collector;

Otherwise (Base = "0"), there is no current

Transistor: building block for logic gates

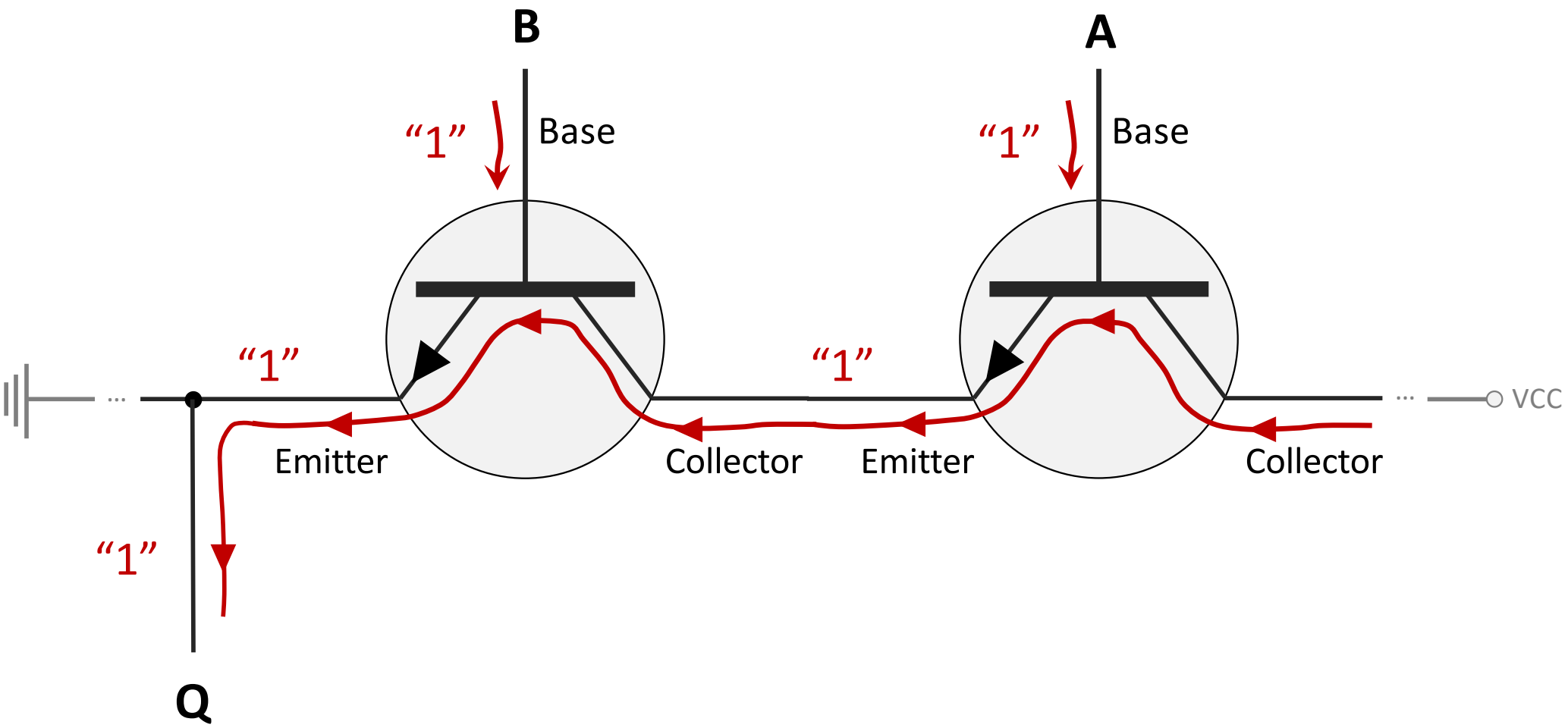



Transistor is a fast switch:

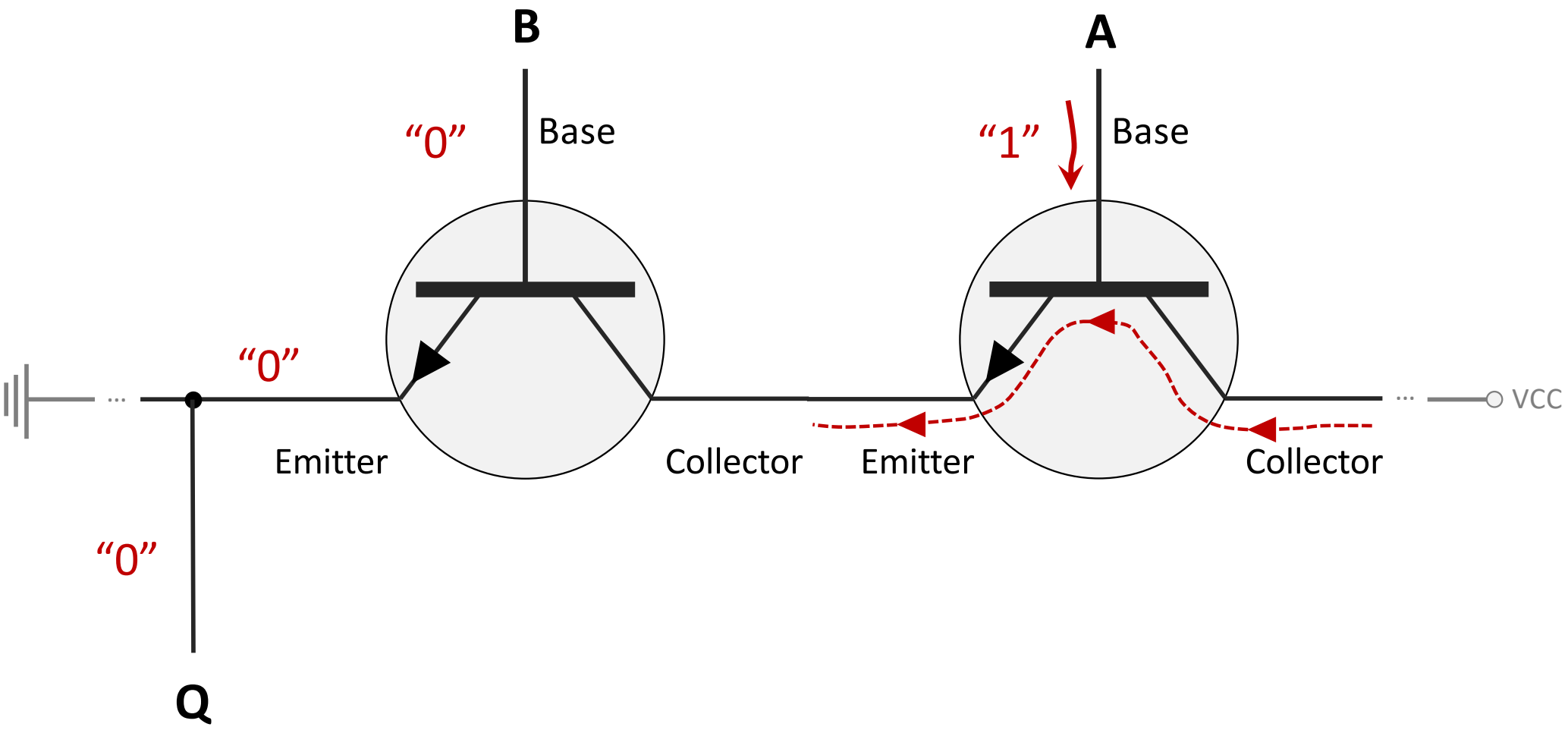
If there is voltage at the Base terminal (Base = "1"), then current flows between Emitter and Collector;

Otherwise (Base = "0"), there is no current

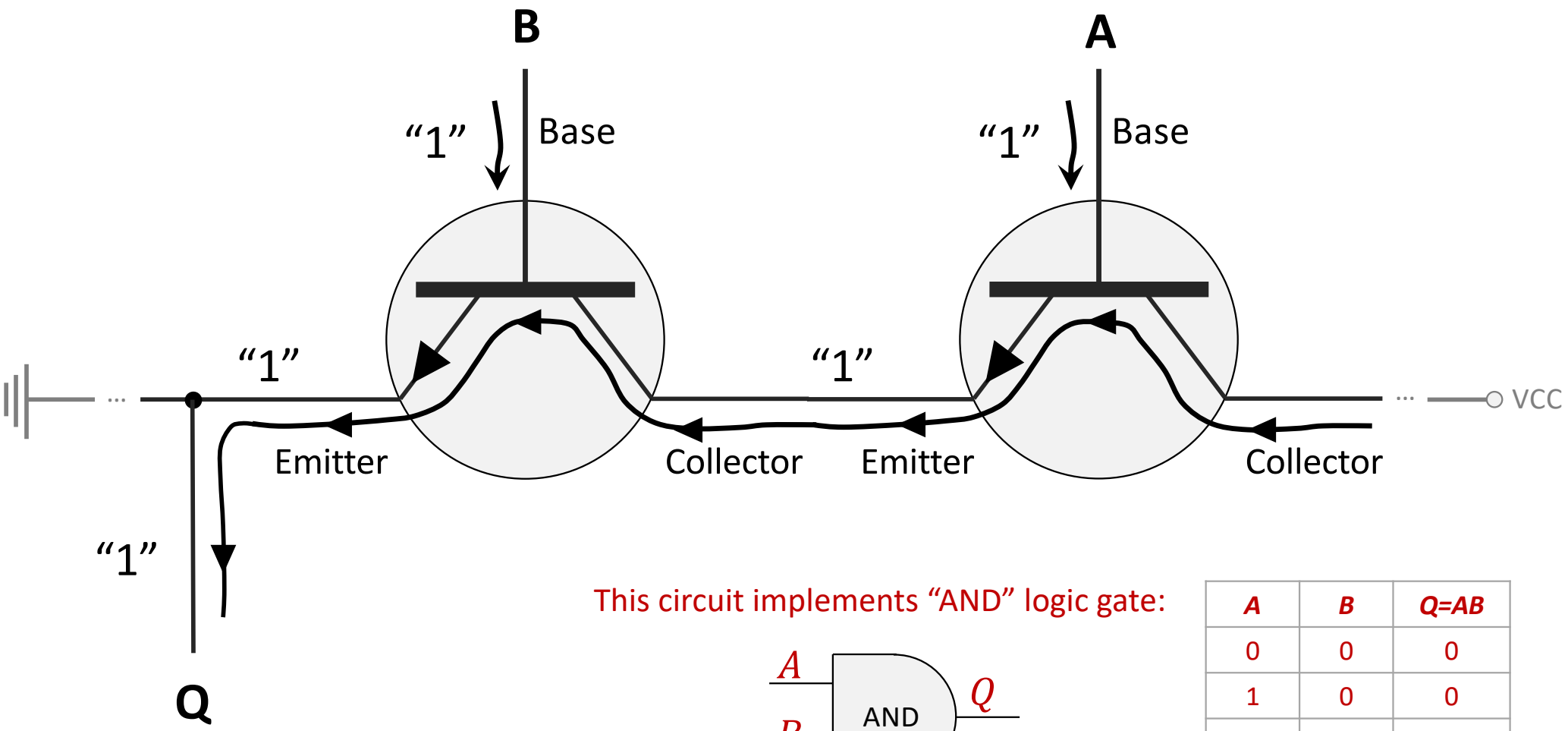
Transistor: building block for logic gates



Transistor: building block for logic gates



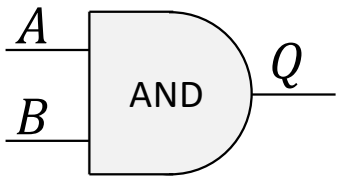
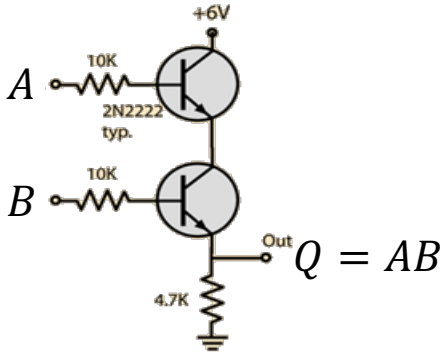
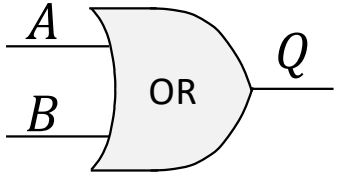
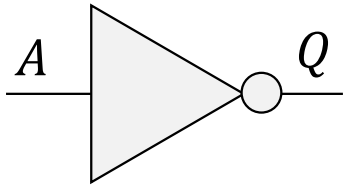
Transistor: building block for logic gates



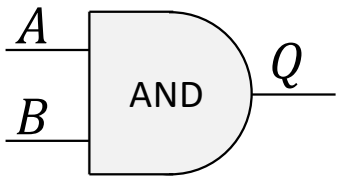
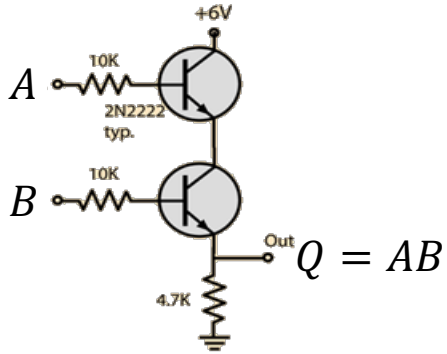
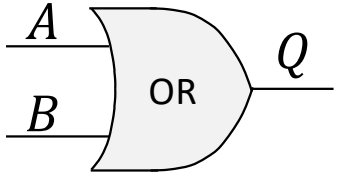
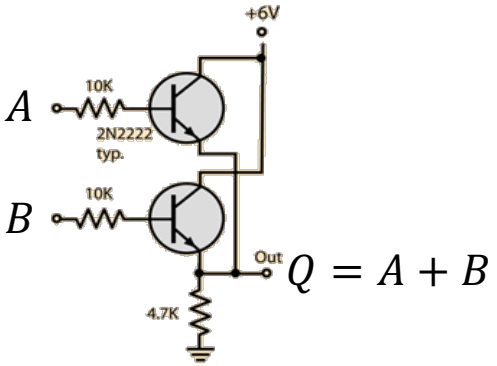
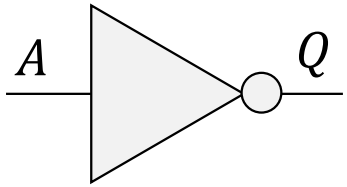

This circuit implements "AND" logic gate:



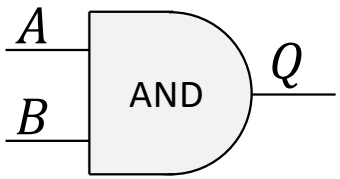
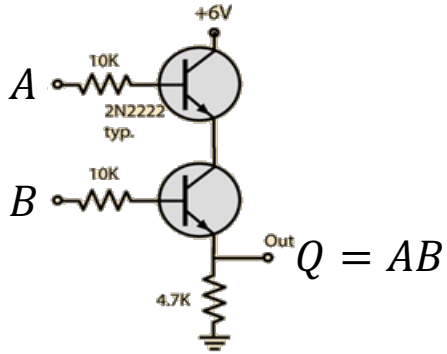
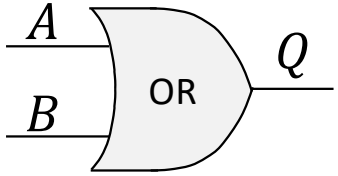
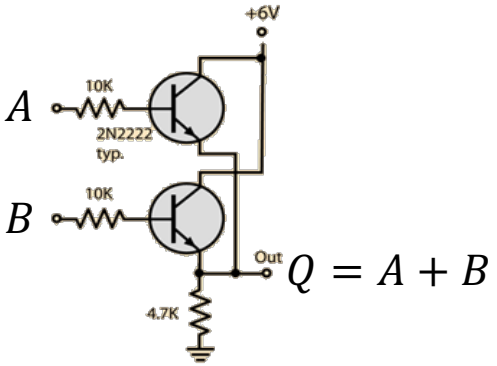
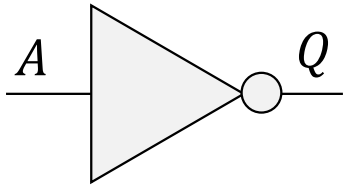
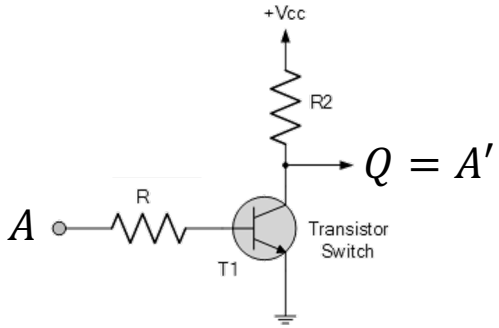

| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

<table>
<tr><th>Logic Gate</th><th>Symbolic Representation</th><th>Truth Table</th><th>Implementation with Transistors</th></tr>
<tr>
<td>AND</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>OR</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>Q=A’</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td></td>
</tr>
</table>

Transistor implementations are taken from http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1 and https://www.electronics-tutorials.ws/logic/logic_4.html

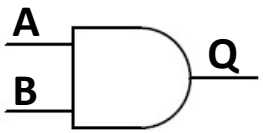
<table>
<tr><th>Logic Gate</th><th>Symbolic Representation</th><th>Truth Table</th><th>Implementation with Transistors</th></tr>
<tr>
<td>AND</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>OR</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>NOT</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>Q=A’</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td></td>
</tr>
</table>

Transistor implementations are taken from http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1 and https://www.electronics-tutorials.ws/logic/logic_4.html

<table>
<tr><th>Logic Gate</th><th>Symbolic Representation</th><th>Truth Table</th><th>Implementation with Transistors</th></tr>
<tr>
<td>AND</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>OR</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Q=A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>NOT</td>
<td> 
</td>
<td>
<table>
<tr><th>A</th><th>Q=A’</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

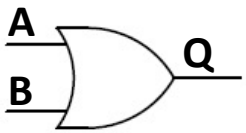
Transistor implementations are taken from http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1 and https://www.electronics-tutorials.ws/logic/logic_4.html

# Basic Logic Gates



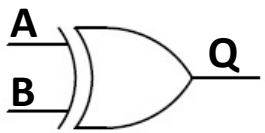

**AND**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



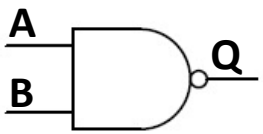

**OR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



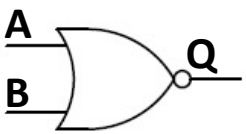

**XOR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



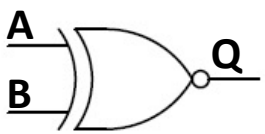

**NAND**

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




**NOR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

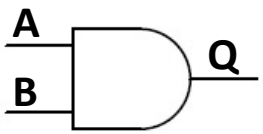



**XNOR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

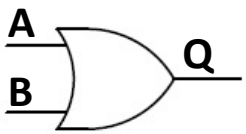
Image is taken from https://instrumentationtools.com/logic-gates/

# Basic Logic Gates



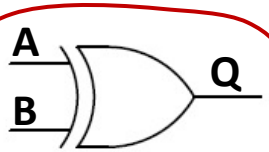

**AND**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




**OR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

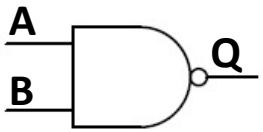



**XOR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

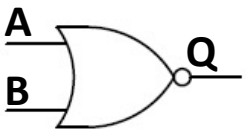
**XOR = eXceptional OR**:

unlike OR, if both inputs are “1”s, the output is “0”



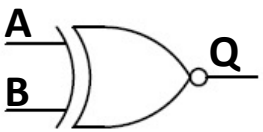

**NAND**

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




**NOR**

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

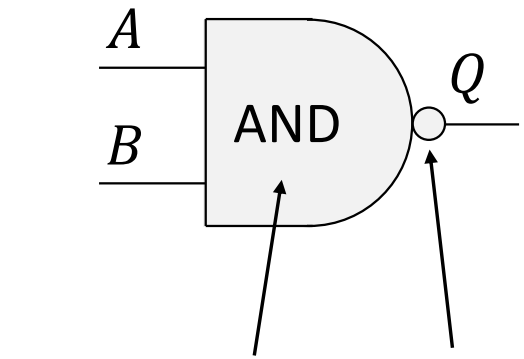



**XNOR**

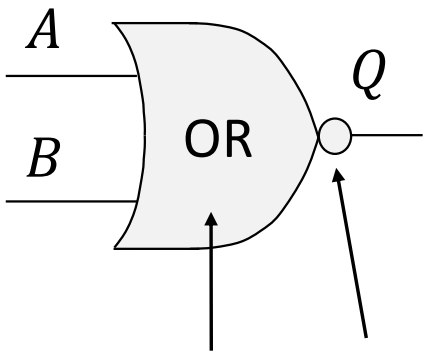
| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Image is taken from https://instrumentationtools.com/logic-gates/

# NAND and NOR Logic Gates



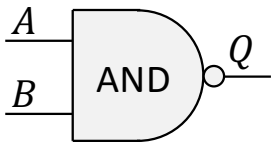

NAND = AND + NOT

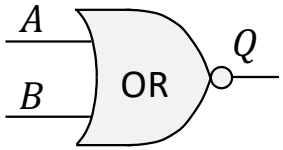



NOR = OR + NOT

# Universal Logic Gates: NAND and NOR

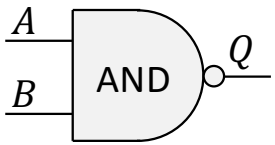
We can implement any other gate, by using one of them

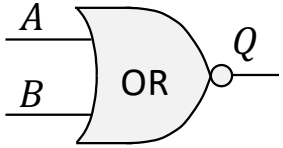


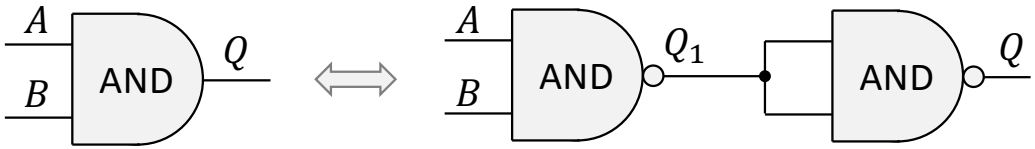
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them



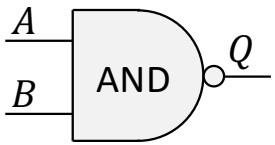




## Example: AND gate through NAND

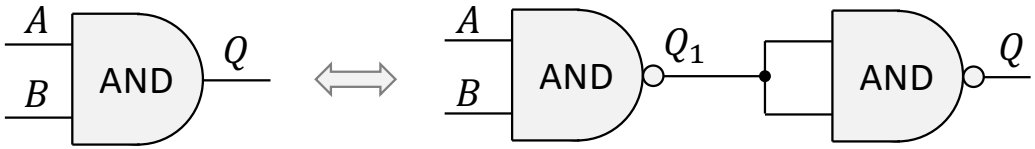


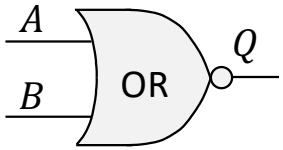
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them




Example: AND gate through NAND



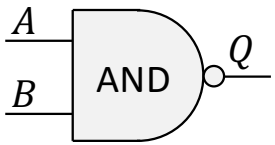

To proof circuits equivalence, we compare truth tables:

| *A* | *B* | *Q=AB* |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

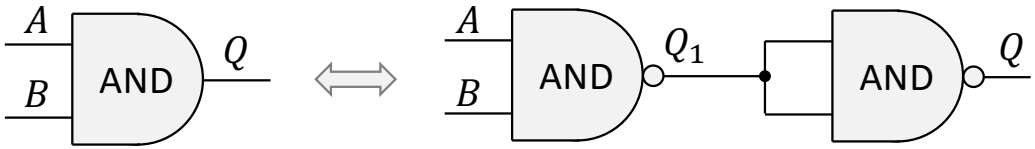
| *A* | *B* | $Q_1$ | $Q_1$ | *Q* |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

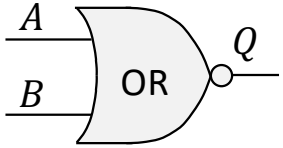
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them




Example: AND gate through NAND



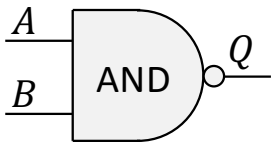

To proof circuits equivalence, we compare truth tables:

| *A* | *B* | *Q=AB* |
|---|---|---|
| **0** | **0** | **0** |
| **1** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **1** | **1** |

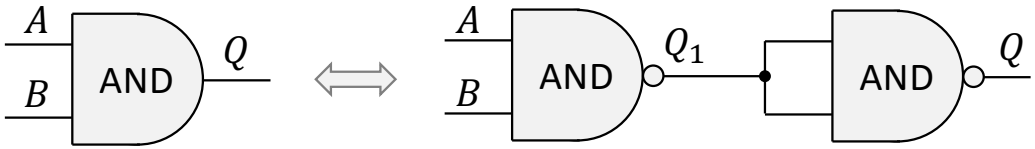
| *A* | *B* | $Q_1$ | $Q_1$ | *Q* |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

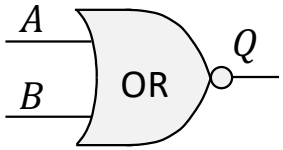
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them




Example: AND gate through NAND



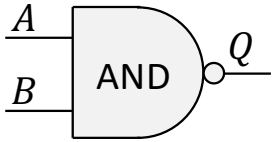

To proof circuits equivalence, we compare truth tables:

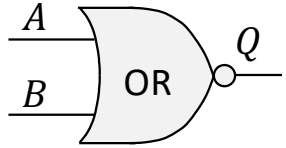
| *A* | *B* | *Q=AB* |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

| *A* | *B* | $Q_1$ | $Q_1$ | *Q* |
|---|---|---|---|---|
| **0** | **0** | 1 | 1 | **0** |
| **1** | **0** | 1 | 1 | **0** |
| **0** | **1** | 1 | 1 | **0** |
| **1** | **1** | 0 | 0 | **1** |

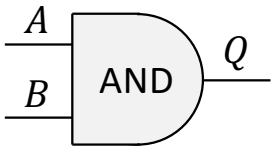
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them

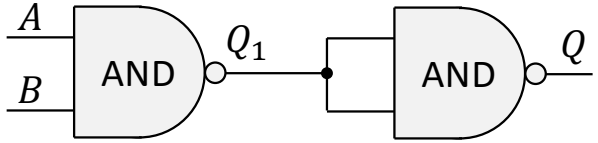


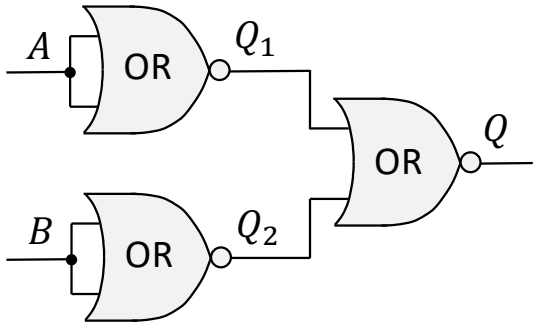

Original gate:




NAND impl.:

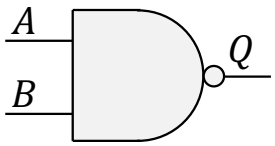
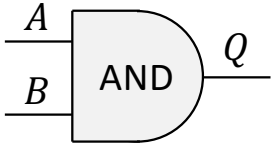
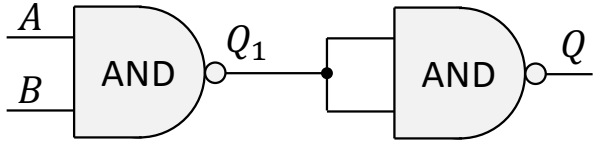
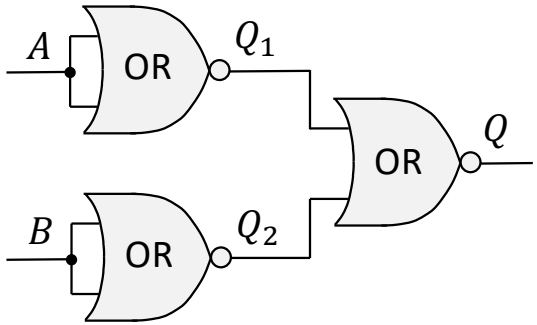
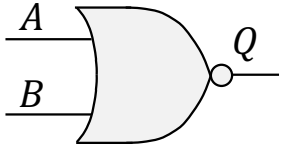
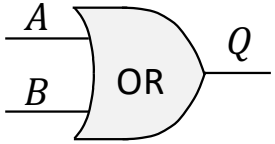
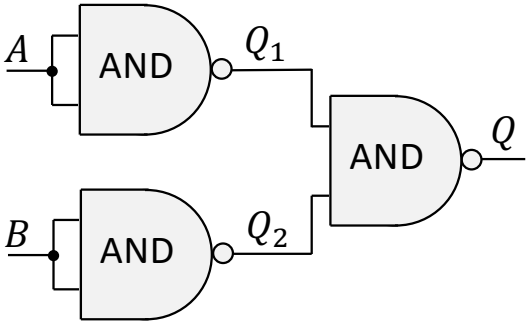
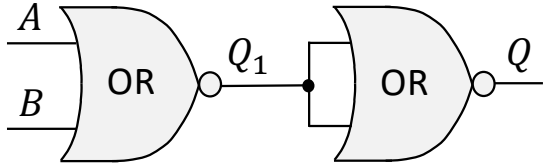



NOR impl.:



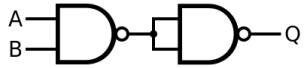
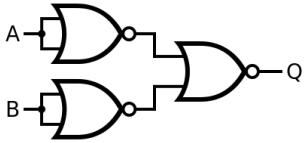
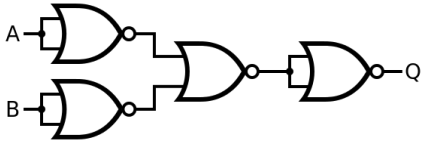
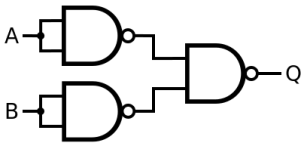
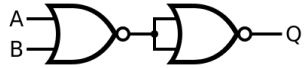
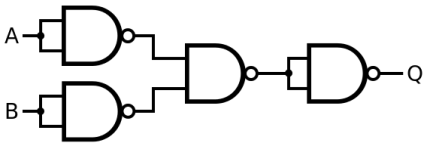
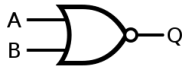
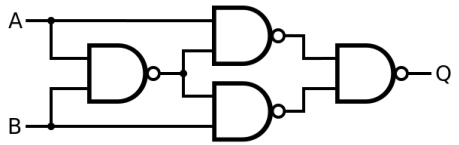
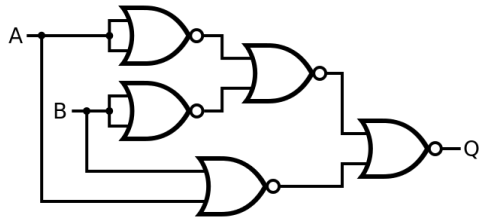
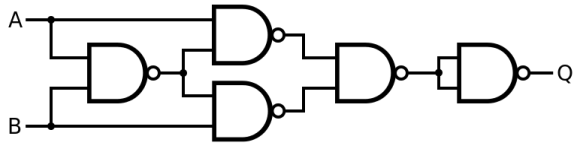
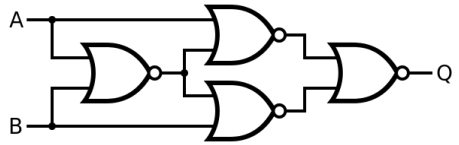
# Universal Logic Gates: NAND and NOR

We can implement any other gate, by using one of them

| | Original gate: | NAND impl.: | NOR impl.: |
|---|---|---|---|
| NAND |  |  |  |  |
| NOR |  |  |  |  |

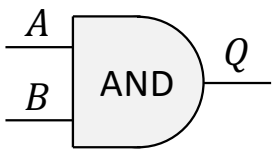
# Universal Logic Gates: NAND and NOR

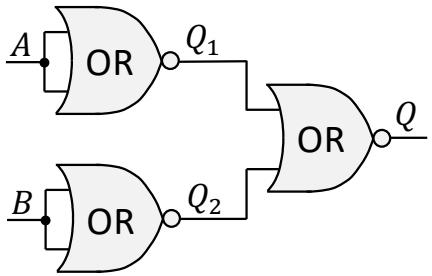
We can implement any other gate,
by using one of them

| type | NAND construction | NOR construction |
|---|---|---|
| NOT |  |  |
| AND |  |  |
| NAND |  |  |
| OR |  |  |
| NOR |  |  |
| XOR |  |  |
| XNOR |  |  |

# How to Prove the Equivalence of Logic Circuits?

## Way 1: Truth Tables



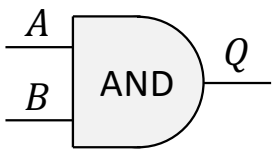




| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

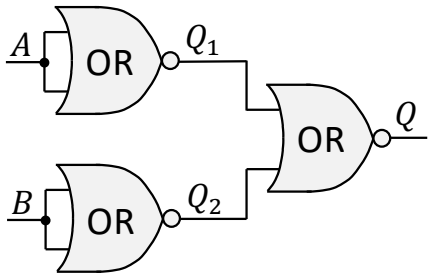
| A | B | $Q_1$ | $Q_2$ | Q |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |

# How to Prove the Equivalence of Logic Circuits?

## Way 1: Truth Tables




| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

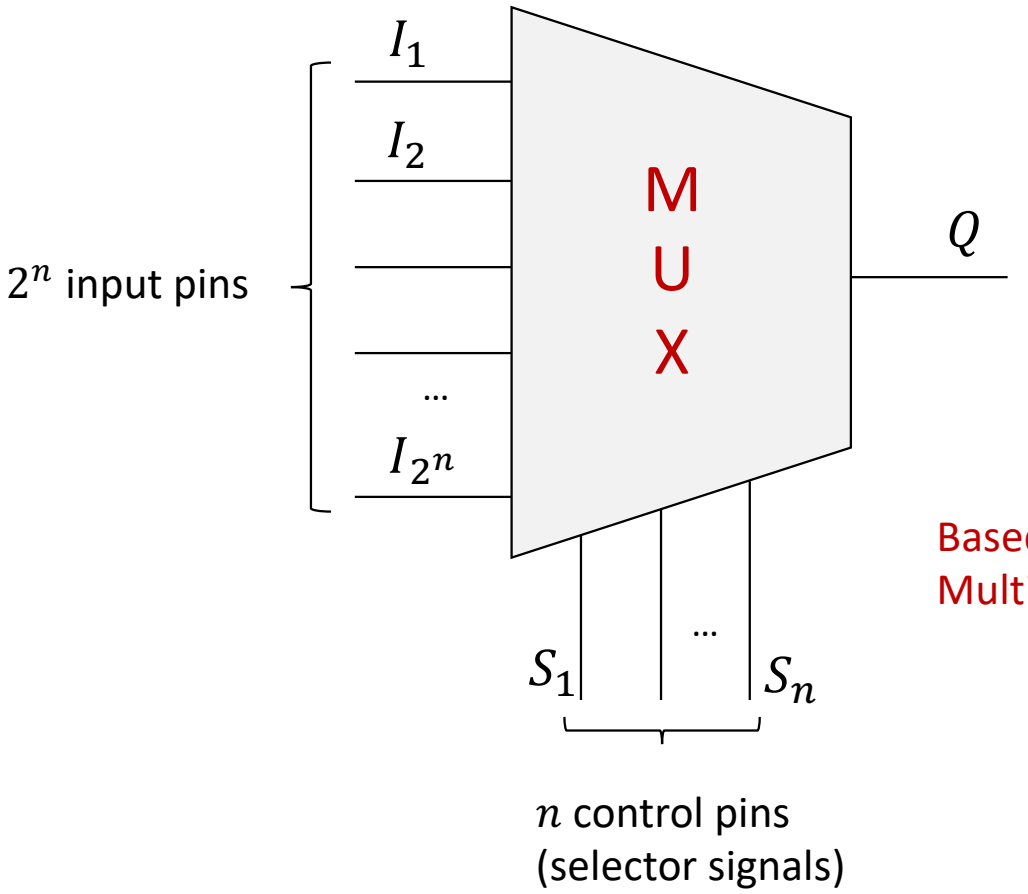



| A | B | $Q_1$ | $Q_2$ | Q |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |

## Way 2: Boolean Algebra Laws

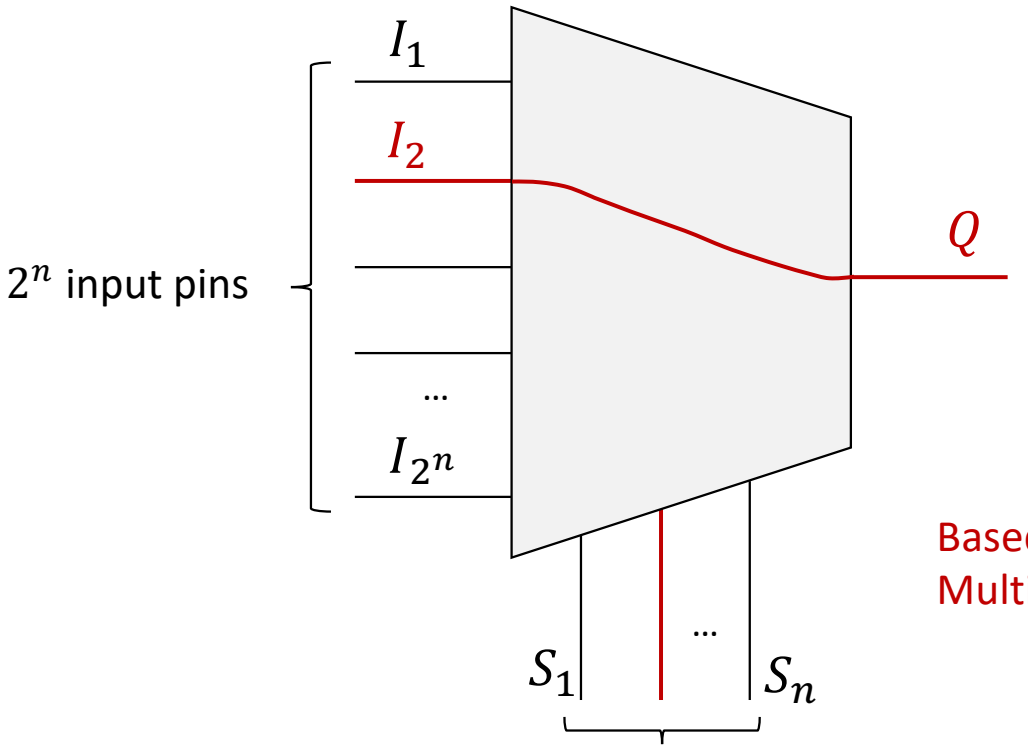
| Name | AND Form | OR Form |
|---|---|---|
| Identity Law | $1A = A$ | $0 + A = A$ |
| Null Law | $0A = 0$ | $1 + A = 1$ |
| Idempotent Law | $AA = A$ | $A + A = A$ |
| Inverse Law | $AA' = 0$ | $A + A' = 1$ |
| Commutative Law | $AB = BA$ | $A + B = B + A$ |
| Associative Law | $(AB)C = A(BC)$ | $(A + B) + C = A + (B + C)$ |
| Distributive Law | $A+BC = (A + B)(A + C)$ | $A(B + C) = AB + AC$ |
| Absorption Law | $A(A + B) = A$ | $A + AB = A$ |
| De Morgan's Law | $\overline{AB} = \overline{A} + \overline{B}$ | $\overline{A + B} = \overline{A}\,\overline{B}$ |

Multiplexor (or Selector) circuit: Example of a Combinational Logic Circuit



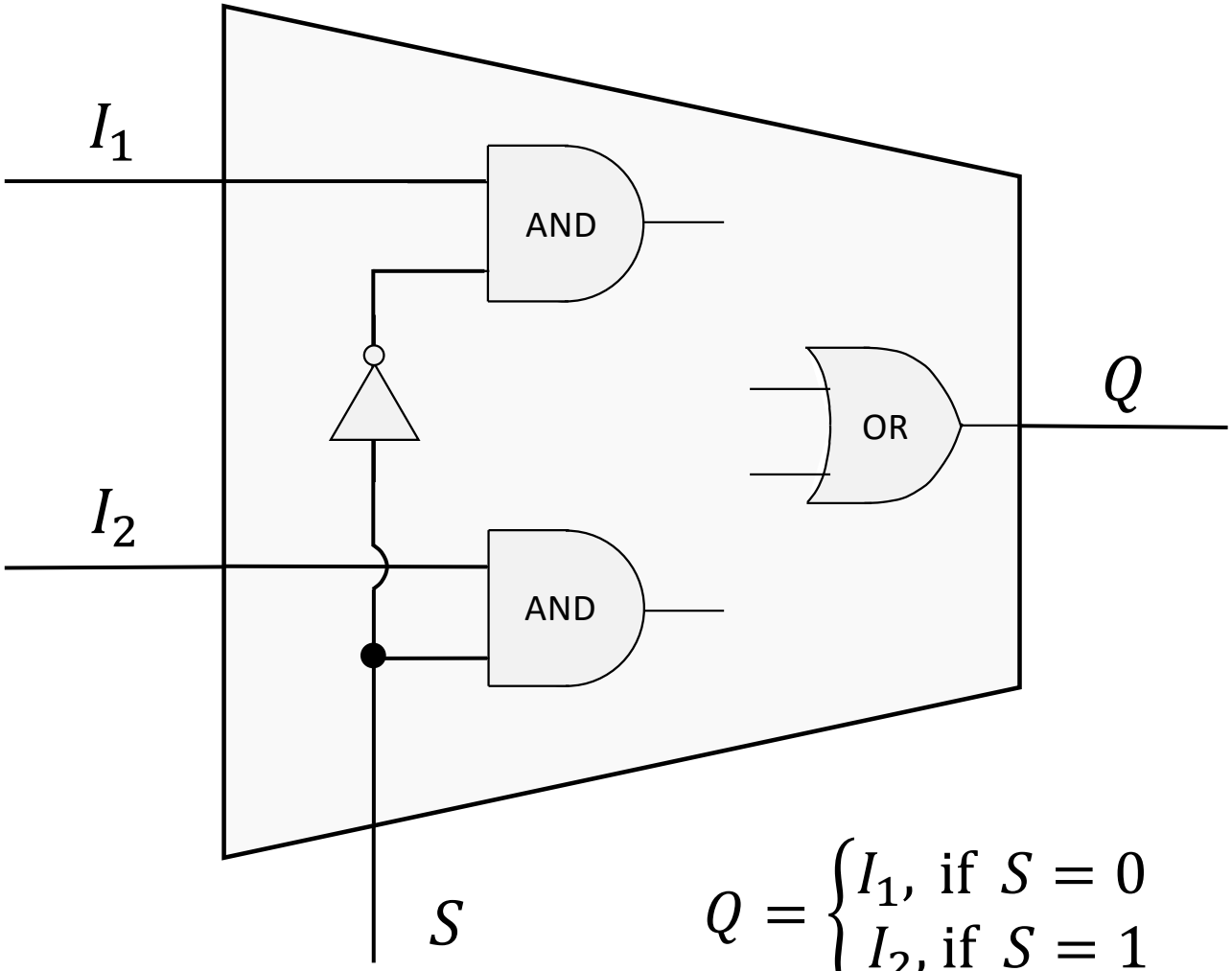

Based on control pin signals,
Multiplexor sets its ouput to one of its inputs

# Multiplexor (or Selector): an Example of a Combinational Logic Circuit



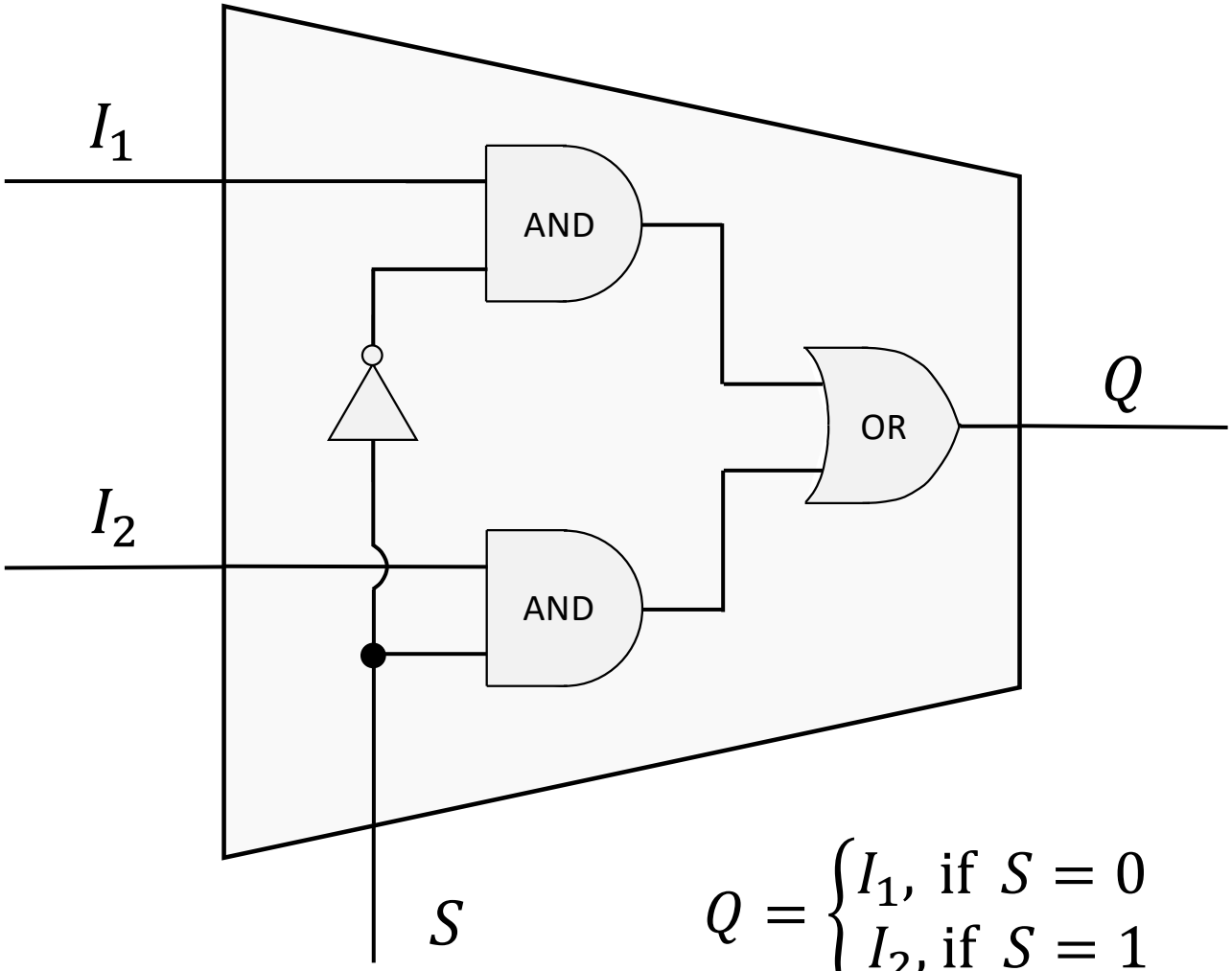

Based on control pin signals,
Multiplexor sets its ouput to one of its inputs

# Sample Implementation of a 2-to-1 Multiplexor



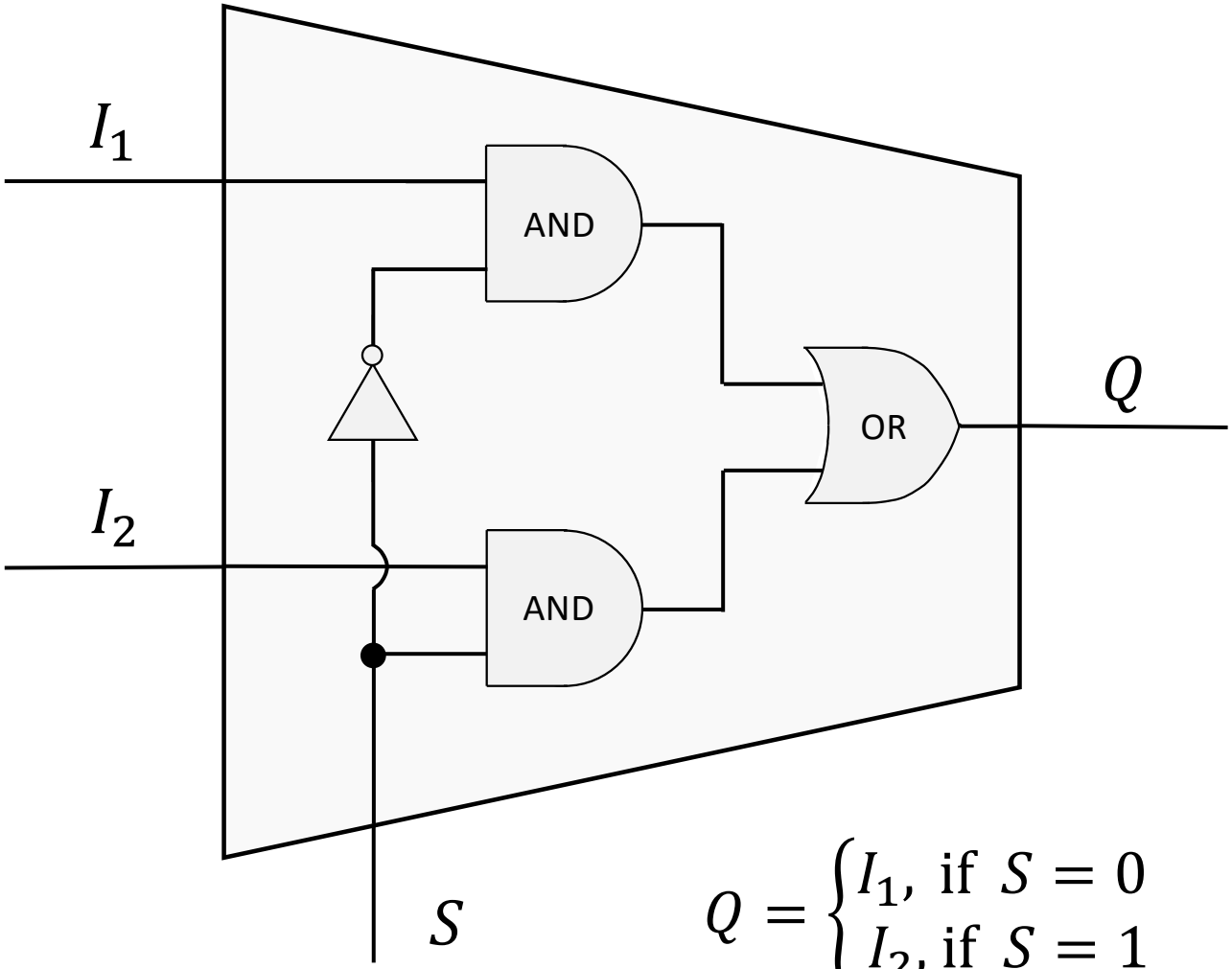

$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ I_2, \text{ if } S = 1 \end{cases}$$

# Sample Implementation of a 2-to-1 Multiplexor




$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ I_2, \text{ if } S = 1 \end{cases}$$

# Sample Implementation of a 2-to-1 Multiplexor

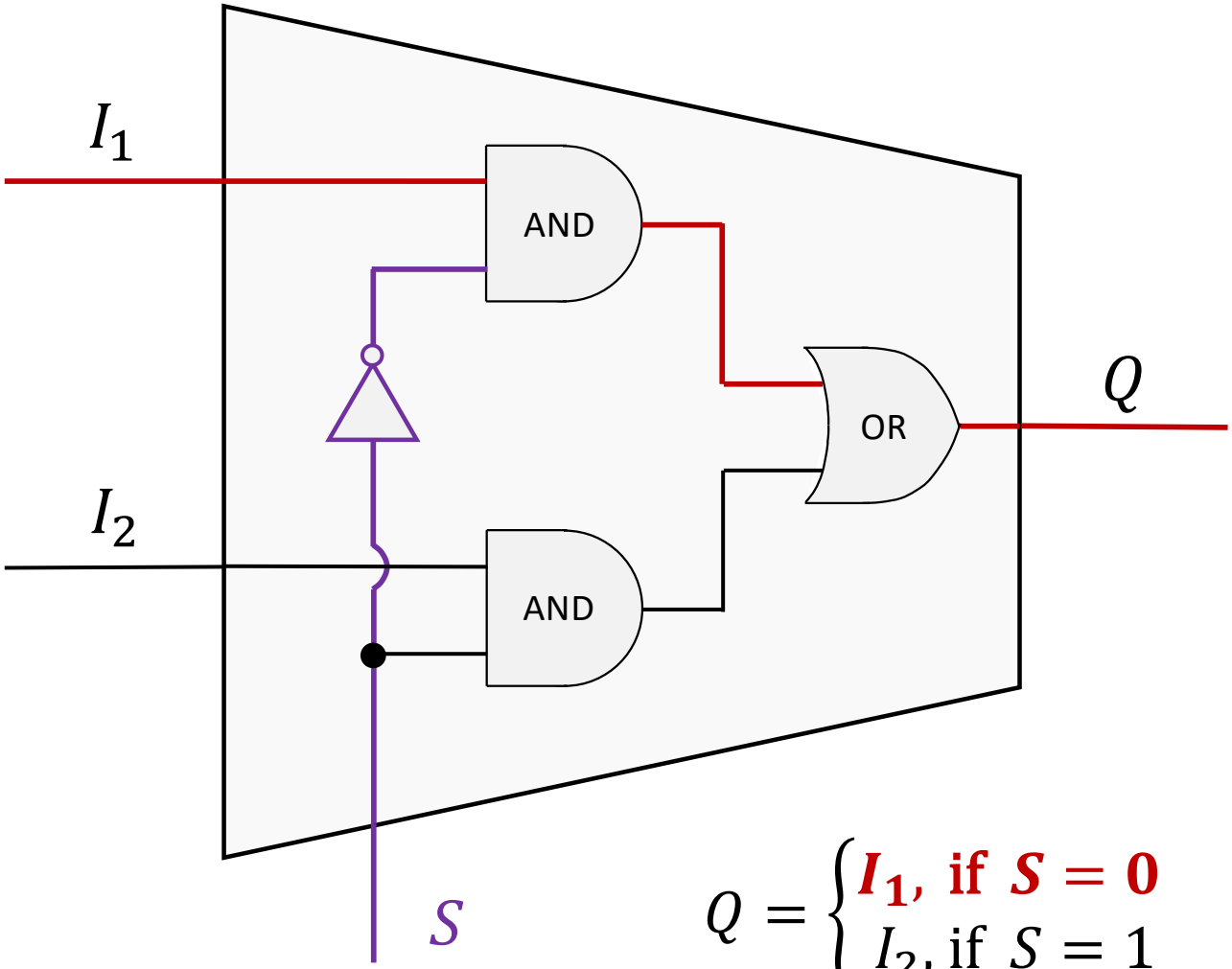



$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ I_2, \text{ if } S = 1 \end{cases}$$

Verification of correctness:

| $I_1$ | $I_2$ | $S$ | $Q$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

# Sample Implementation of a 2-to-1 Multiplexor

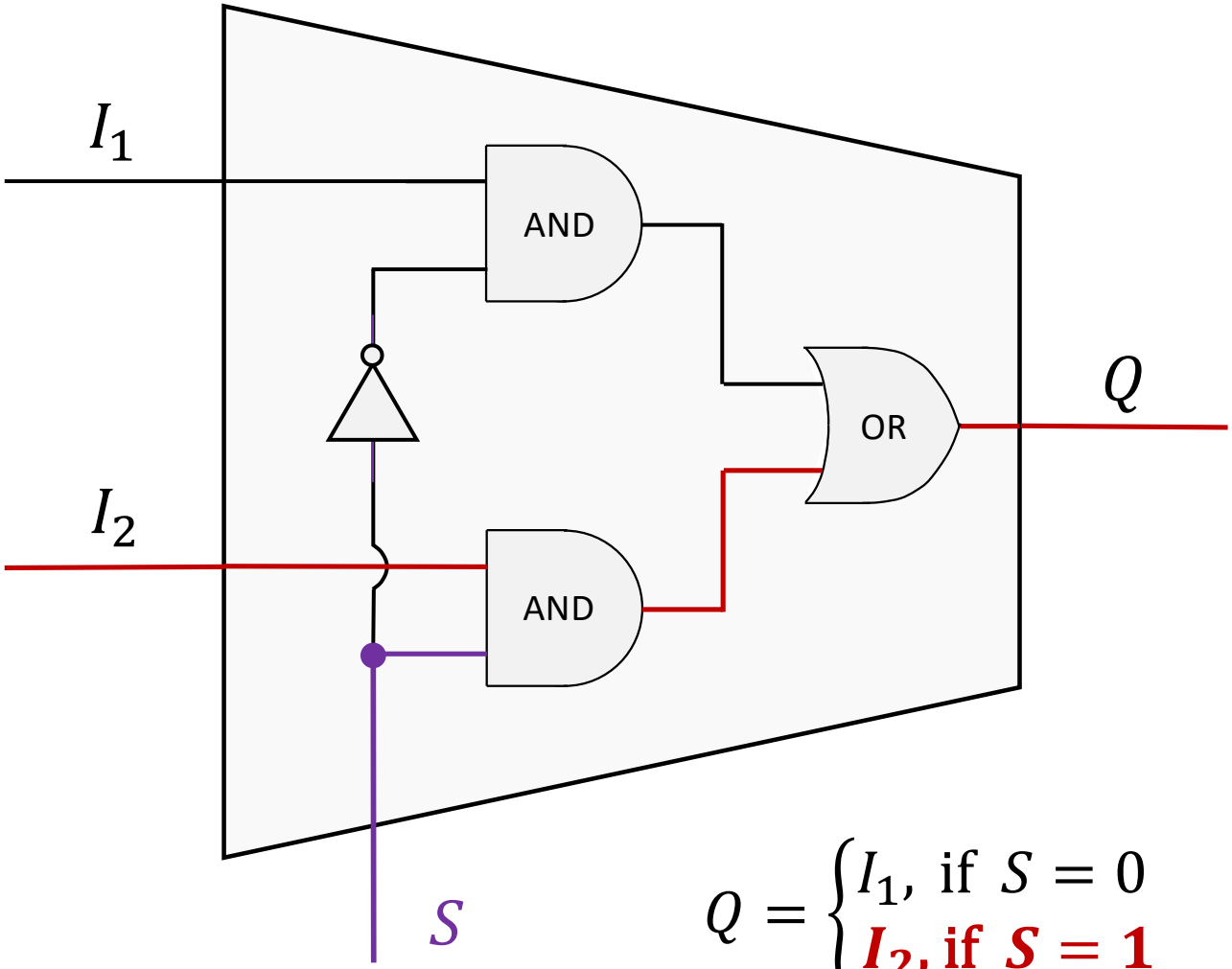



$$Q = \begin{cases} \boldsymbol{I_1}, \text{ if } \boldsymbol{S = 0} \\ I_2, \text{ if } S = 1 \end{cases}$$

Verification of correctness:

| $I_1$ | $I_2$ | $S$ | $Q$ |
|---|---|---|---|
| **0** | 0 | **0** | **0** |
| **1** | 0 | **0** | **1** |
| **0** | 1 | **0** | **0** |
| **1** | 1 | **0** | **1** |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

# Sample Implementation of a 2-to-1 Multiplexor

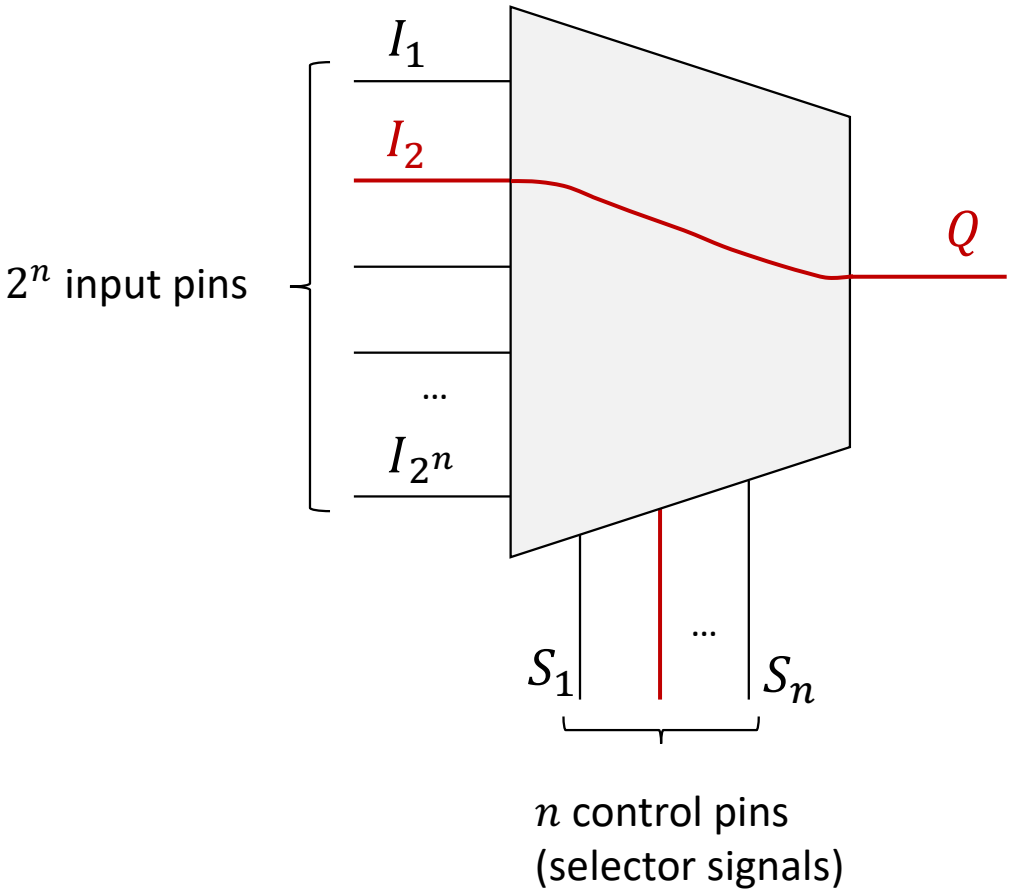



$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ \boldsymbol{I_2, \text{ if } S = 1} \end{cases}$$

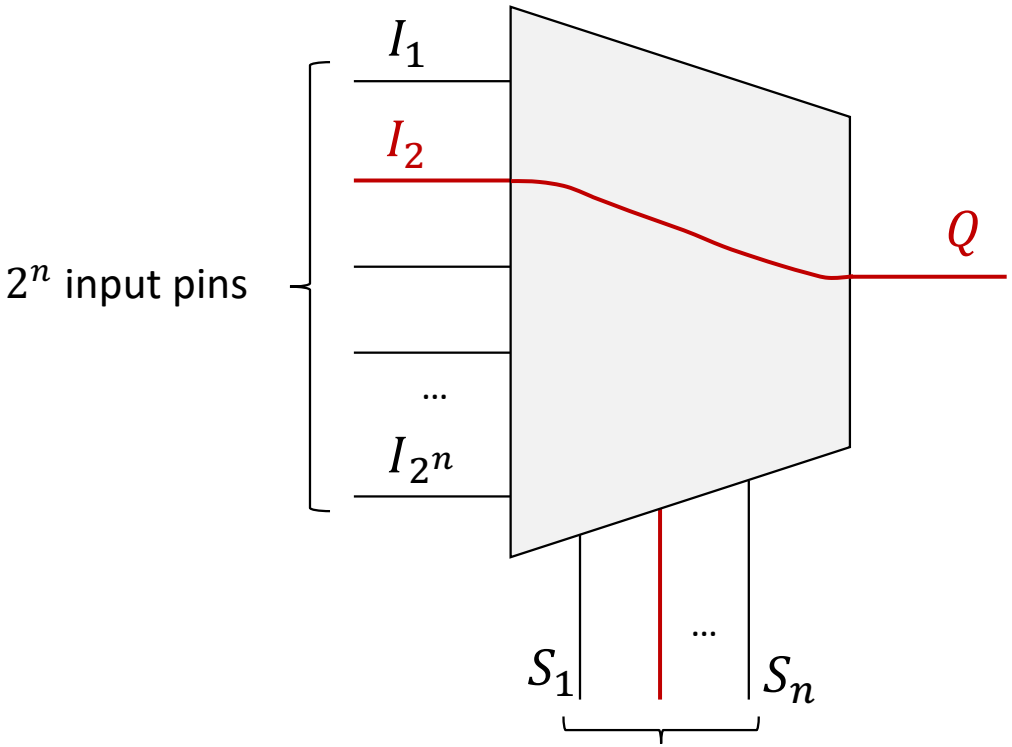
Verification of correctness:

| $I_1$ | $I_2$ | $S$ | $Q$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 0 | **0** | **1** | **0** |
| 1 | **0** | **1** | **0** |
| 0 | **1** | **1** | **1** |
| 1 | **1** | **1** | **1** |

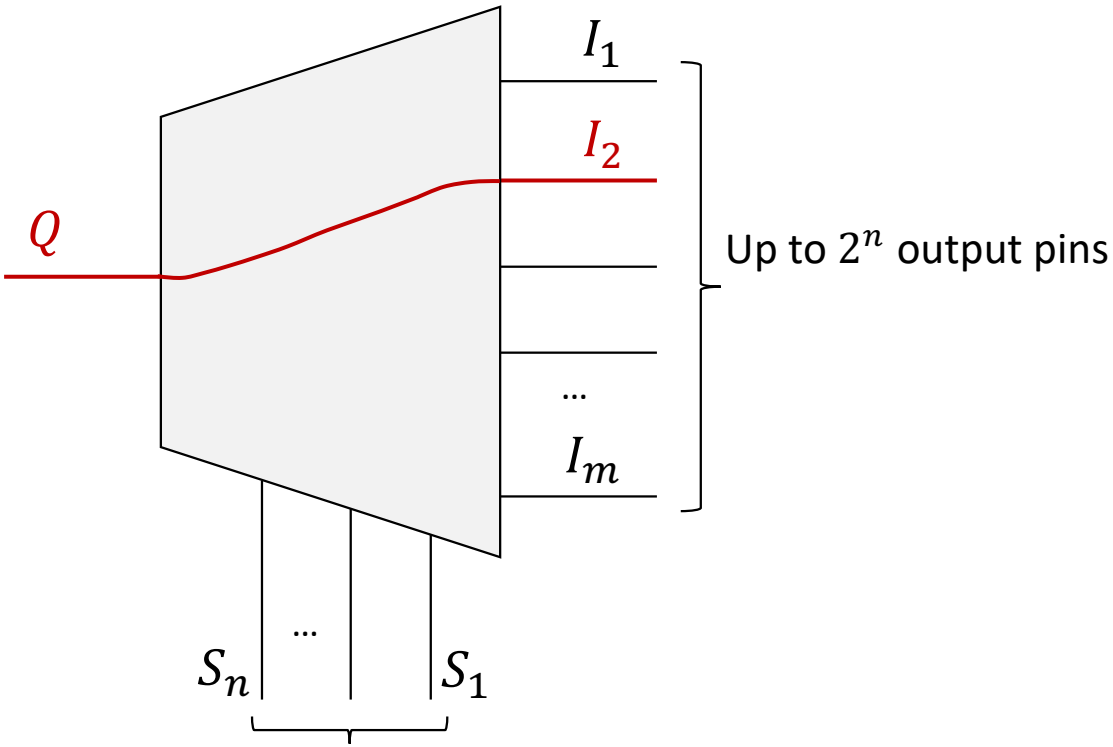
# Multiplexer



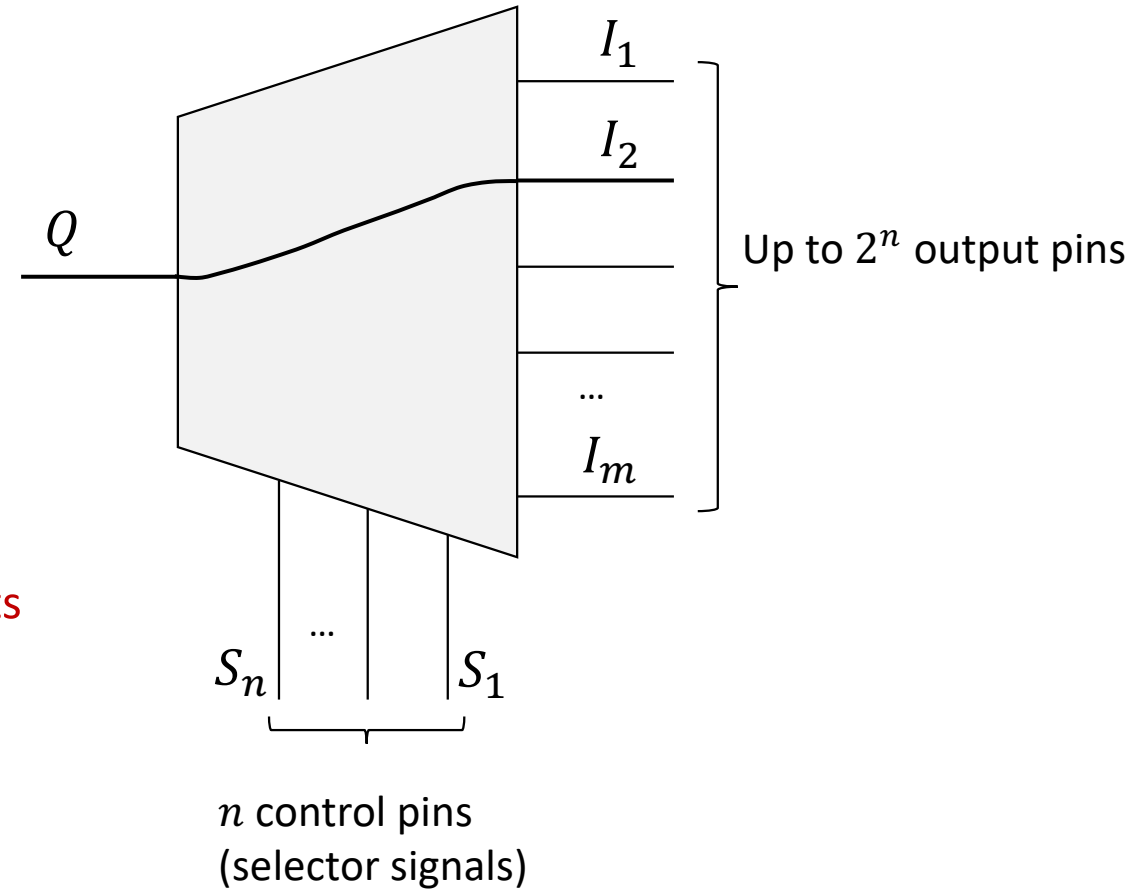
## Multiplexer




## Demultiplexer

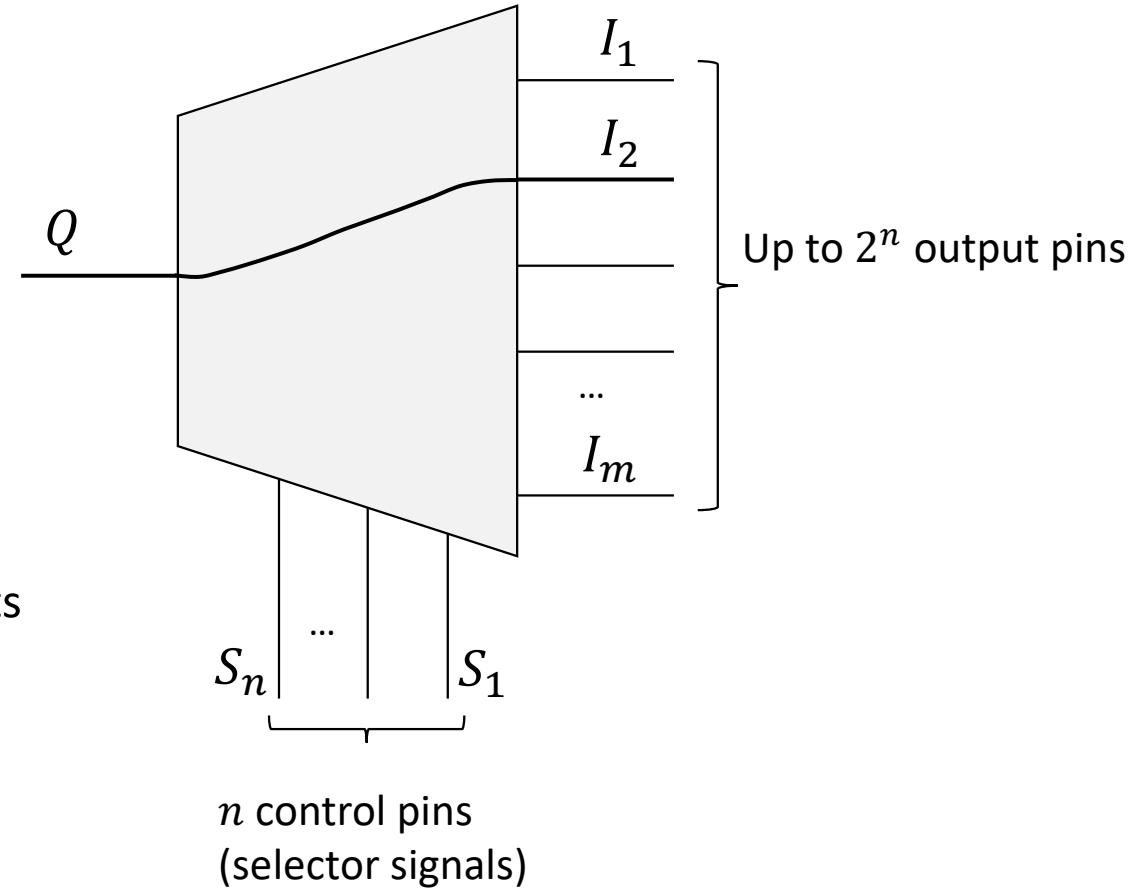


## Demultiplexer



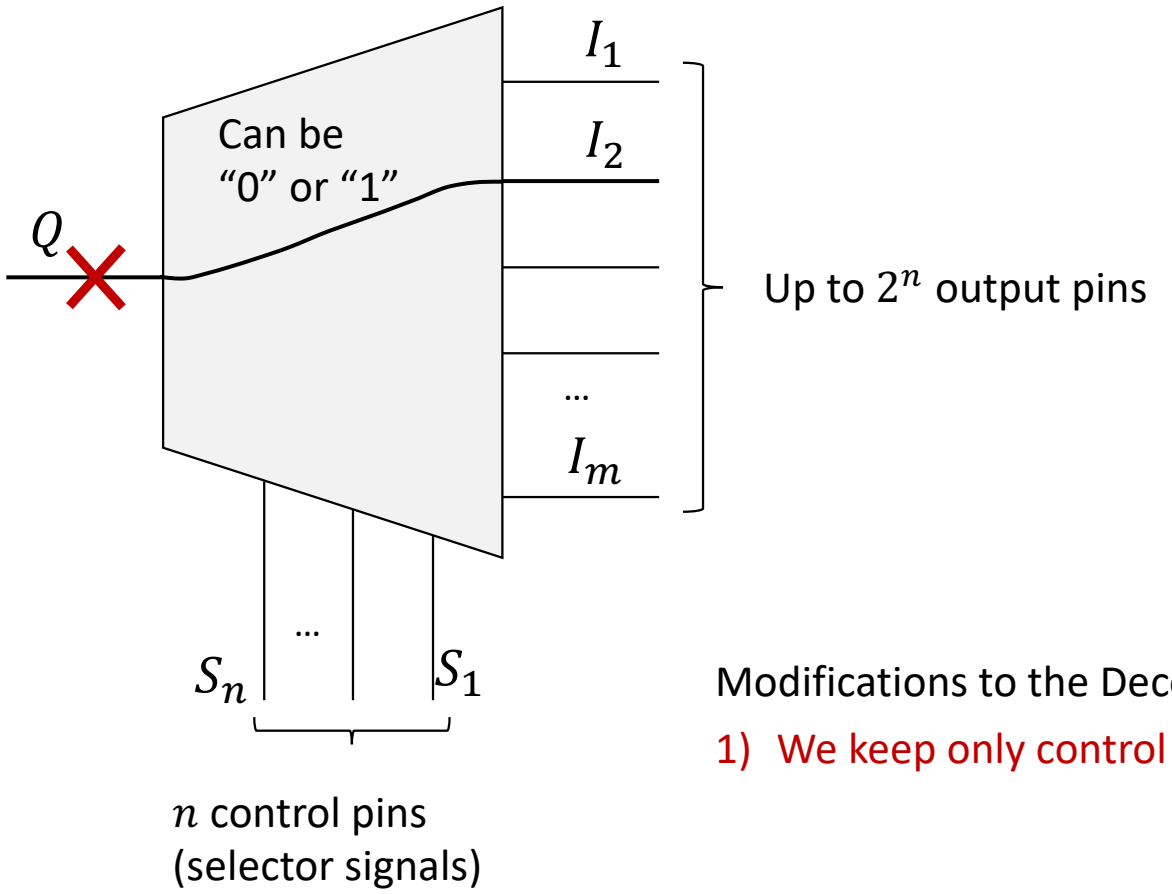

Based on control pin signals,
demultiplexor sends its input to one of its outputs

## Demultiplexer



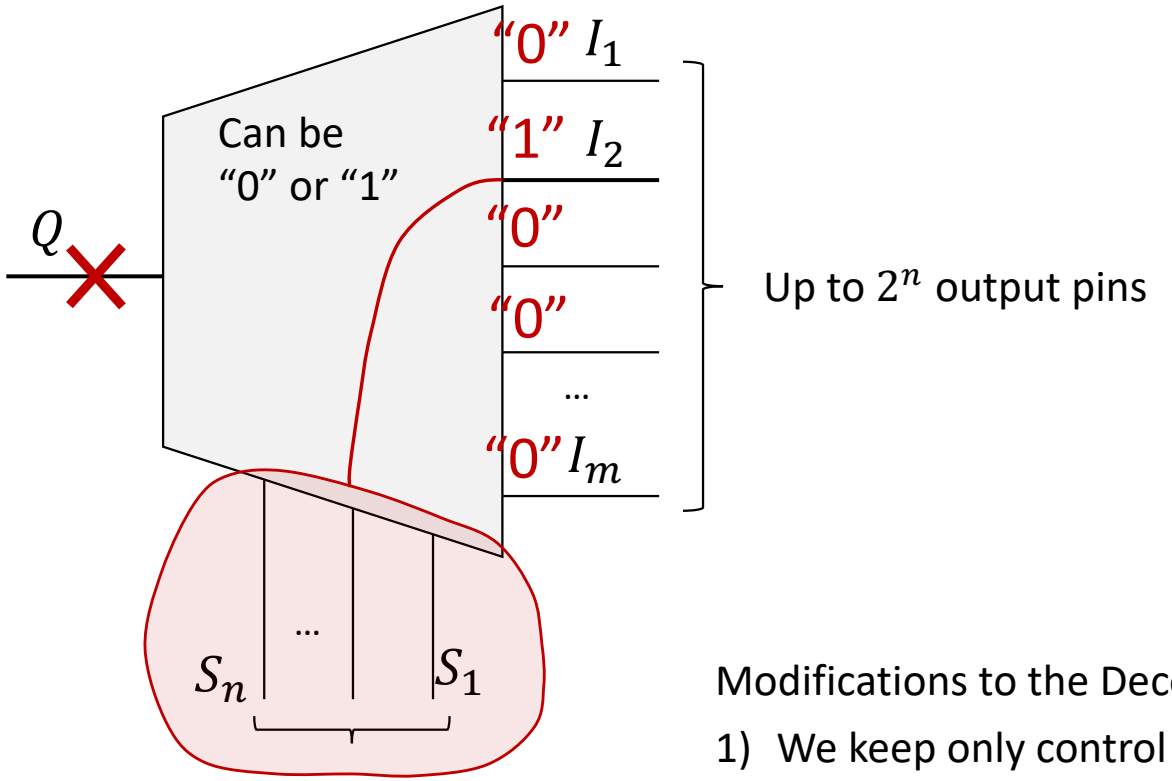

Based on control pin signals,
demultiplexor sends its input to one of its outputs

All other outputs are set to some
default value (e.g. 0)




Modifications to the Decoder:

1) We keep only control pins, and remove input pin Q;



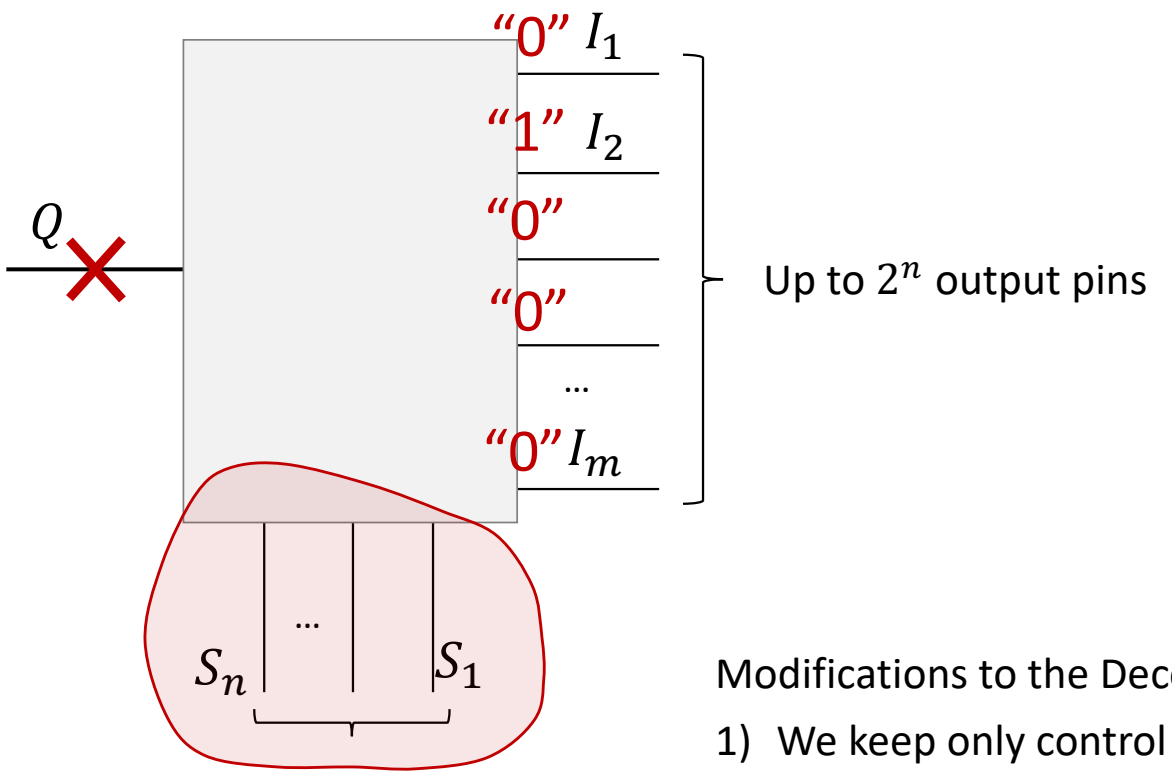

Up to $2^n$ output pins

$n$ control pins
(selector signals)

Modifications to the Decoder:

1) We keep only control pins, and remove input pin Q;
2) We set to "1" the output pin, which corresponds to the values of control pins;

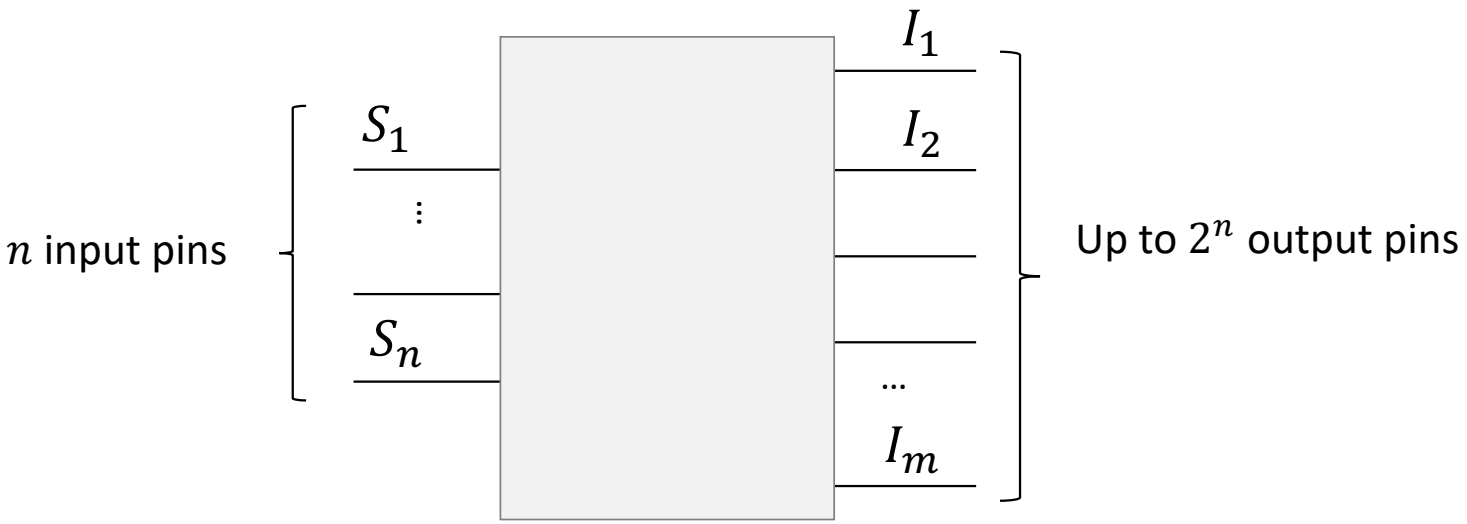
# Decoder




Modifications to the Decoder:

1) We keep only control pins, and remove input pin Q;
2) We set to "1" the output pin, which corresponds to the values of control pins;
3) We change the shape to a rectangle

# Decoder

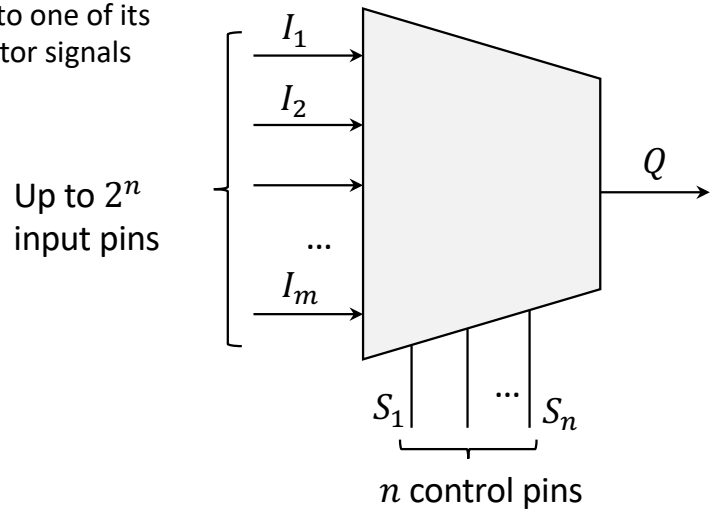



Modifications to the Decoder:

1) We keep only control pins, and remove input pin Q;
2) We set to “1” the output pin, which corresponds to the values of control pins;
3) We change the shape to a rectangle

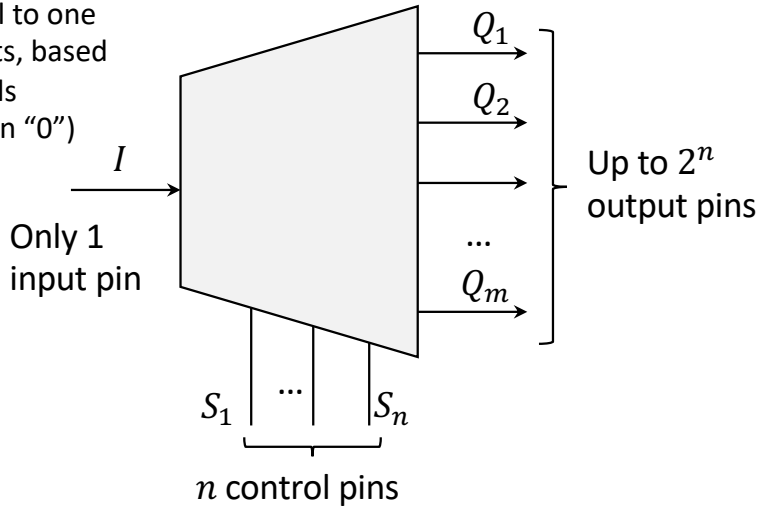
## Multiplexor

Sets a specific output to one of its inputs, based on selector signals



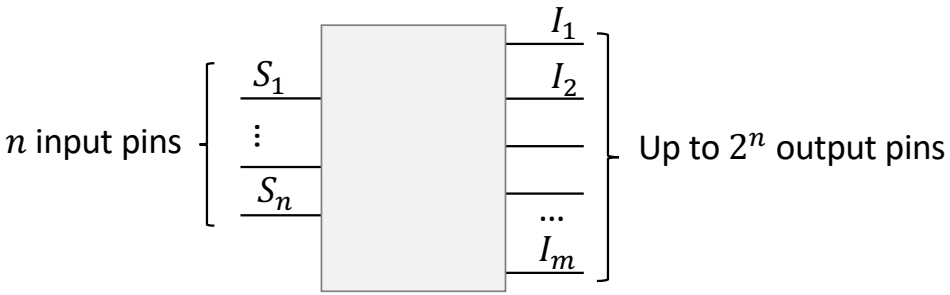

## Demultiplexor

Forwards input signal to one of its multiple outputs, based on the selector signals (other outputs remain "0")



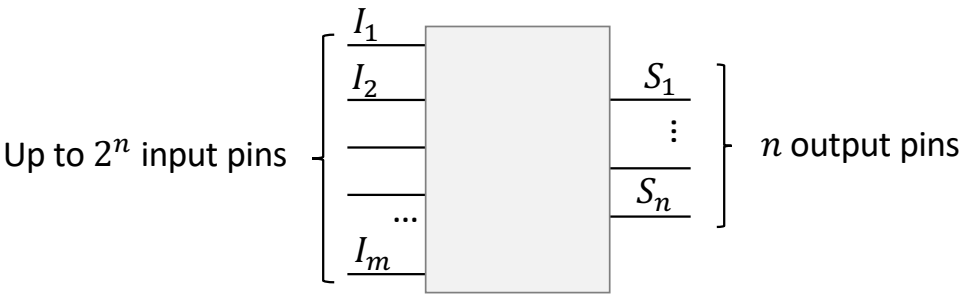

## Decoder

Sets to "1" exactly one output pin, which corresponds to the signals of input pins; All other pins are set to "0"



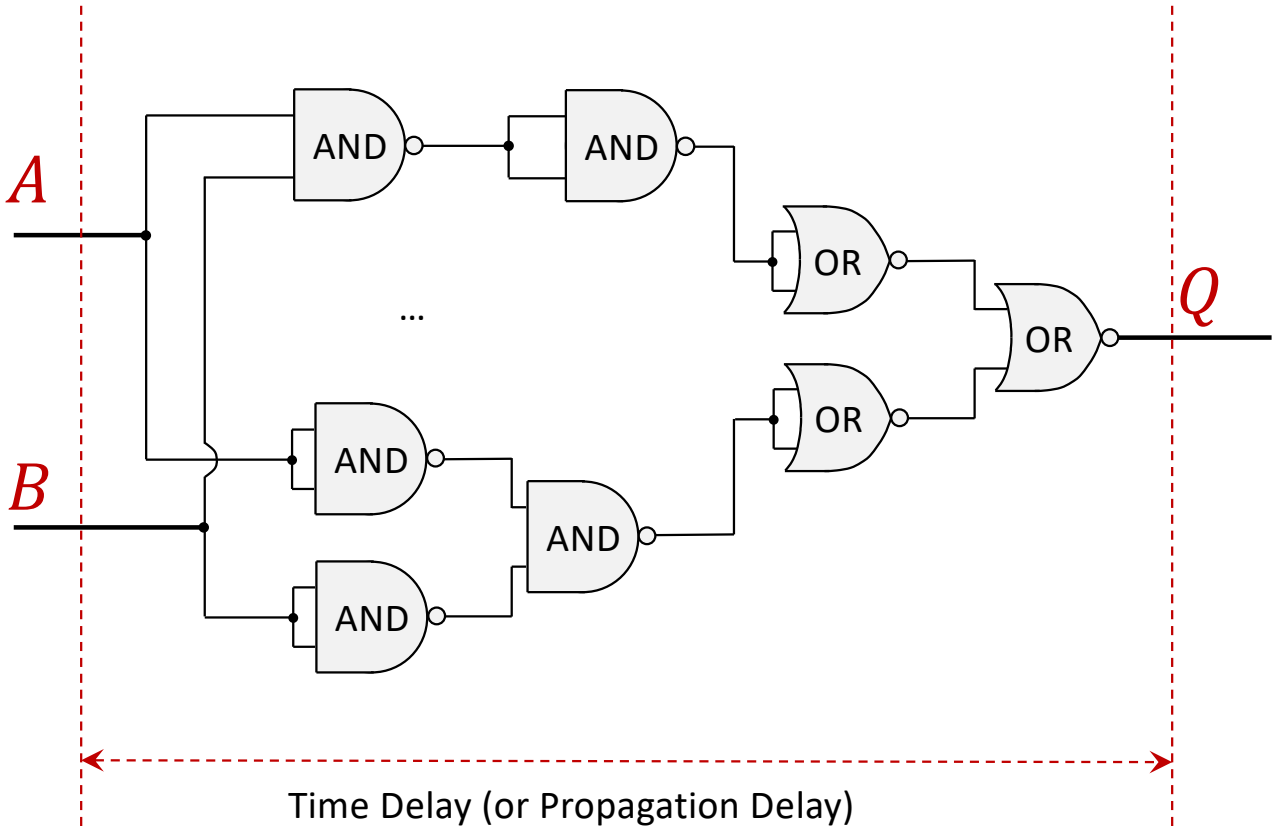

## Encoder

The opposite function of a decoder;
Only one input pin can be set to "1", while all others – "0"



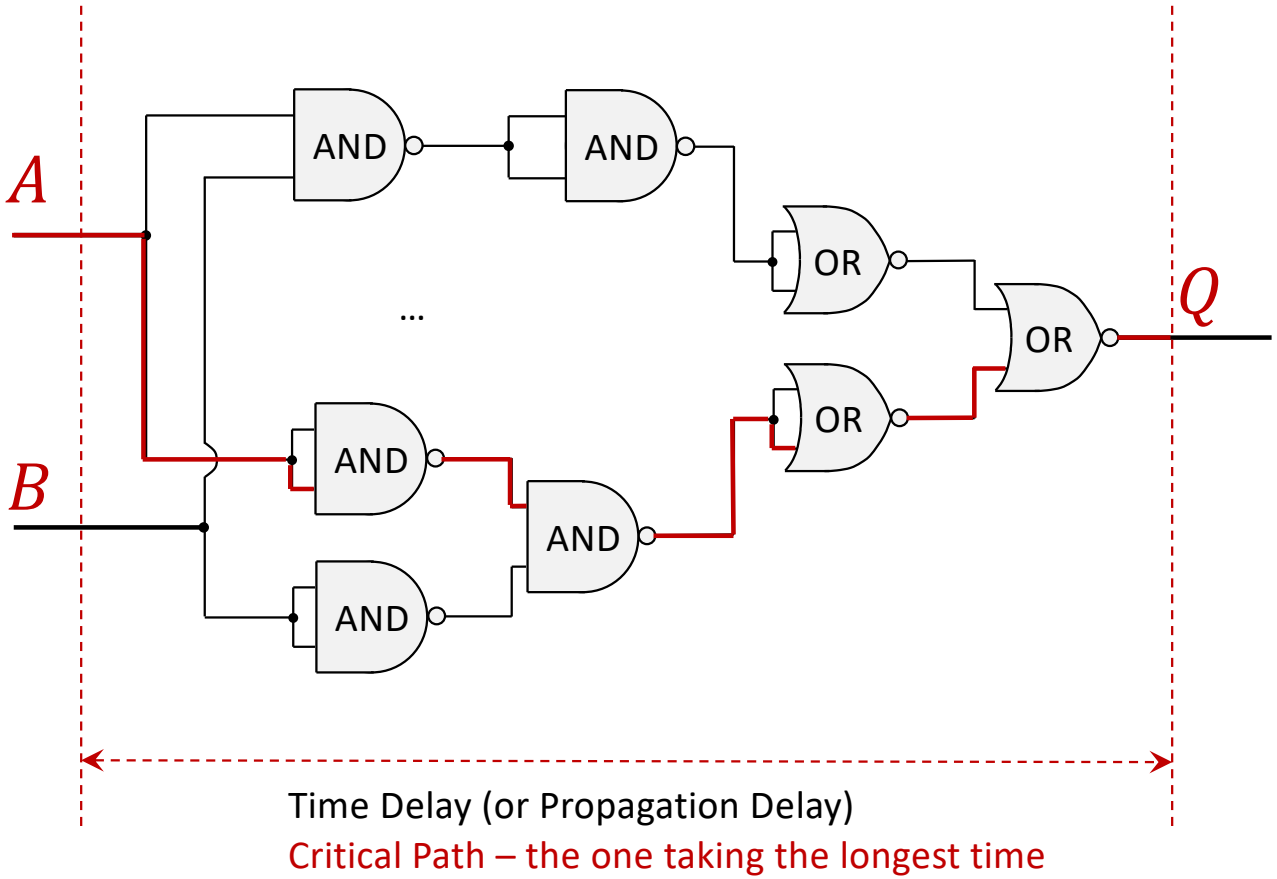
# Propagation Delay

There is ALWAYS a time delay between the change of input signals, and the update of an output signal



# Critical Path and Propagation Delay

There is ALWAYS a time delay between the change of input signals, and the update of an output signal

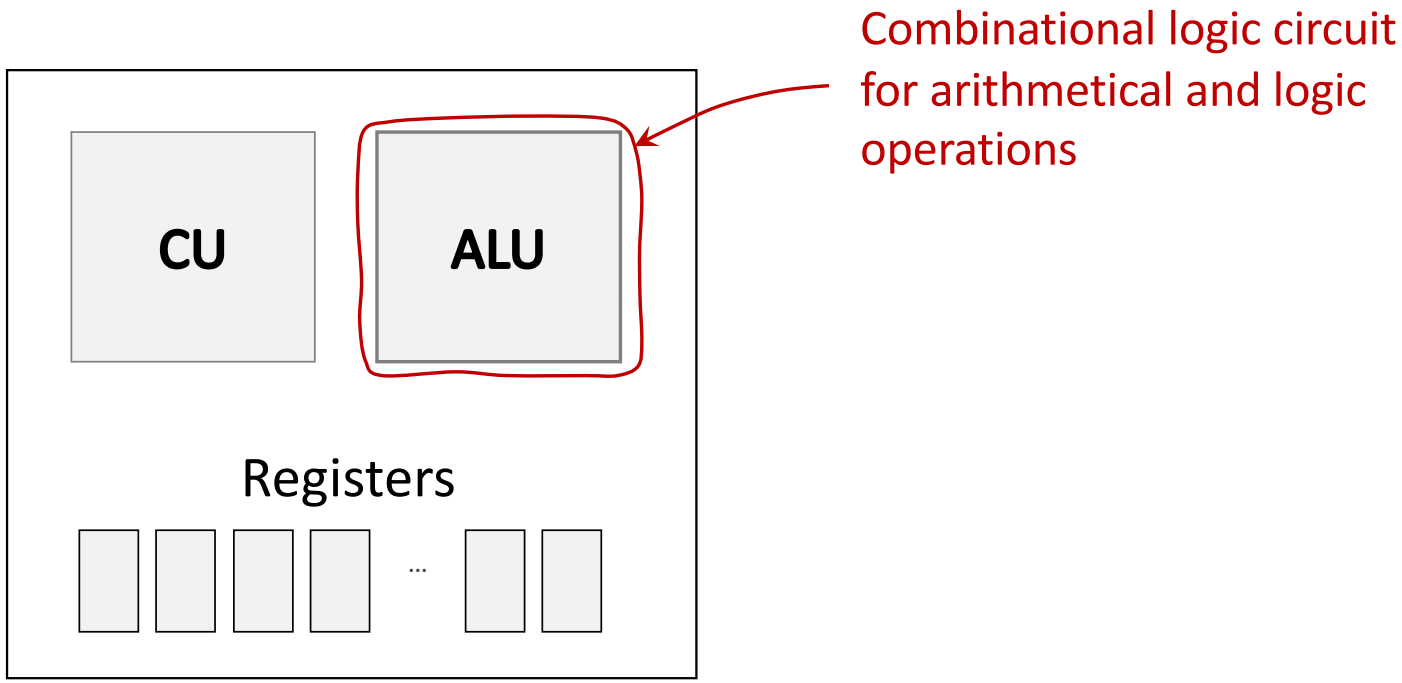


## Combinational Logic Circuit

- Combines input signals and outputs the result;
- Implemented by using logic gates, such as AND, OR, NOT, NAND, etc.
- Does not use memory elements (registers);
- Does not use clock signal (typically)

Typical use: arithmetic operations

# Types of Logic Circuits for CPU Components



# Types of Logic Circuits for CPU Components