Computer Architecture
Tutorial 3

# Combinational and Sequential Logic Circuits

Artem Burmyakov, Muhammad Fahim, Alexander Tormasov

September 17, 2020

Innopolis University

# Terminology

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

Examples:

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

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:

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

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

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

Many 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 |

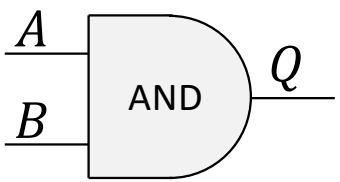
Truth table for function $F = A * B$:

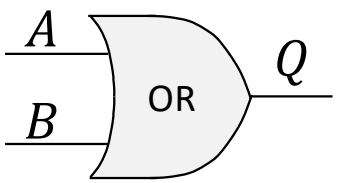
| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{F = A * B}$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

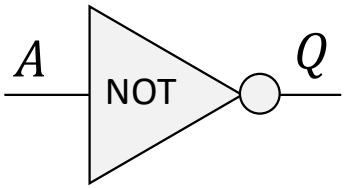
# Terminology

| | |
|---|---|
| **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** | A physical or abstract device implementing a Boolean function |

Sample logic gates:



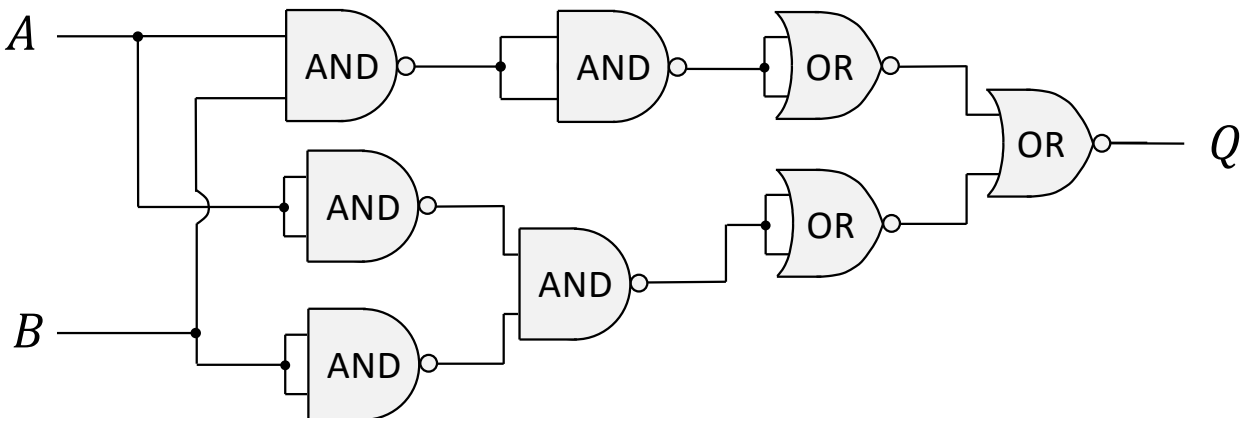



# Terminology

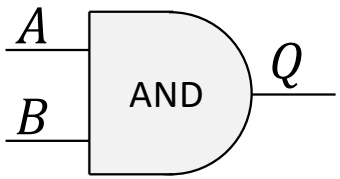
| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | A physical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

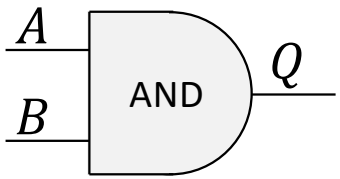
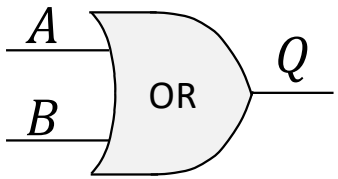
Sample logic circuit:

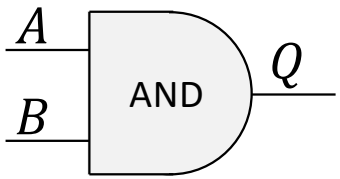
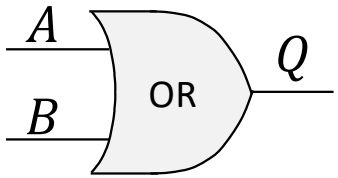
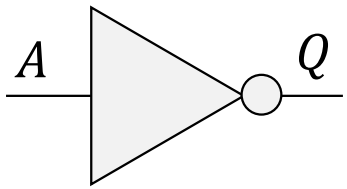


# Terminology

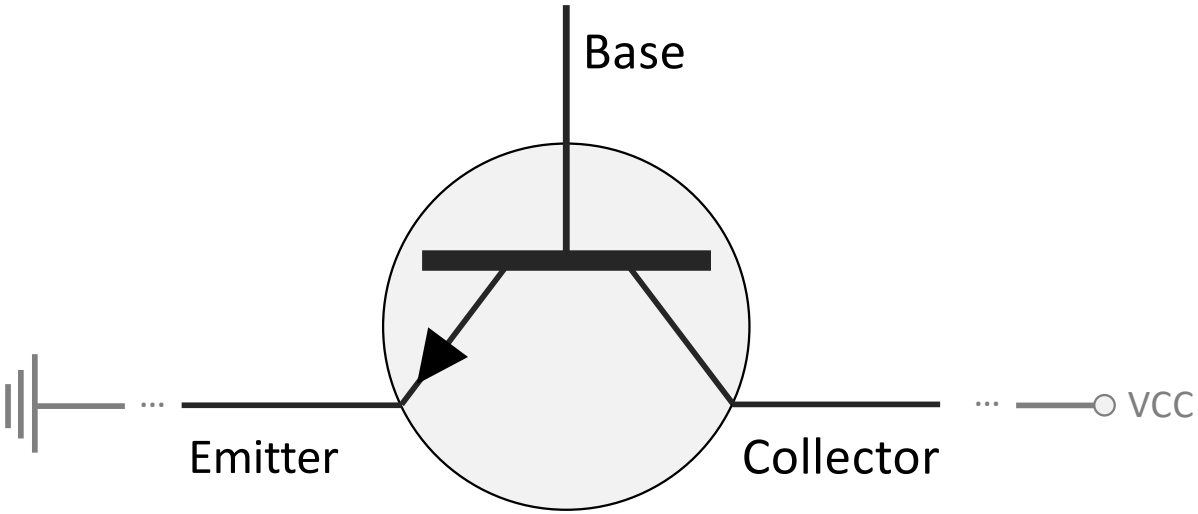
| | |
|---|---|
| **A Boolean function**<br>(introduced by George Boole) | A function taking one or multiple binary inputs and producing exactly one binary output |
| **Truth table** | A table specifying the output of a Boolean function for every combination of input values |
| **A logic gate** | A physical or abstract device implementing a Boolean function |
| **A logic circuit** | A composition of logic gates, to perform a certain function |
| **An Integrated Circuit**<br>(a monolythic integrated circuit, or 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 of integrated circuits over discrete ones:<br>- Lower cost of manufacturing (for mass-production);<br>- Performance (e.g. the propagation delay – to be discussed next);<br>- Compact size<br>Used for CPU |

<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>
</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: the key building block for logic gates

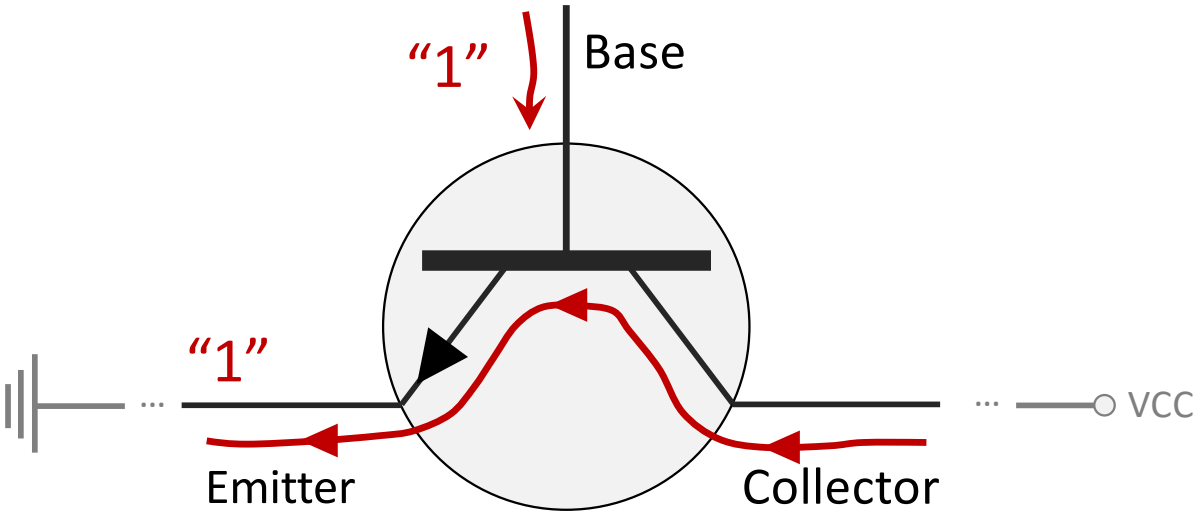



Transistor is a fast switch:

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: the key building block for logic gates



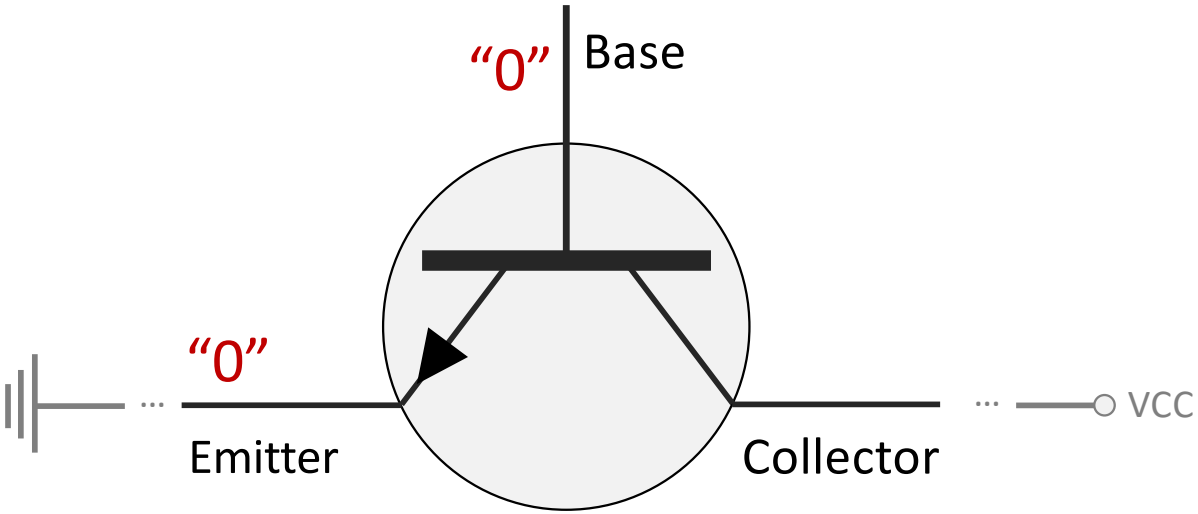

Note: Current direction depends on the transistor type (e.g. NPN or PNP)

Transistor is a fast switch:

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

Otherwise (B=0), there is no current

# Transistor: the key building block for logic gates

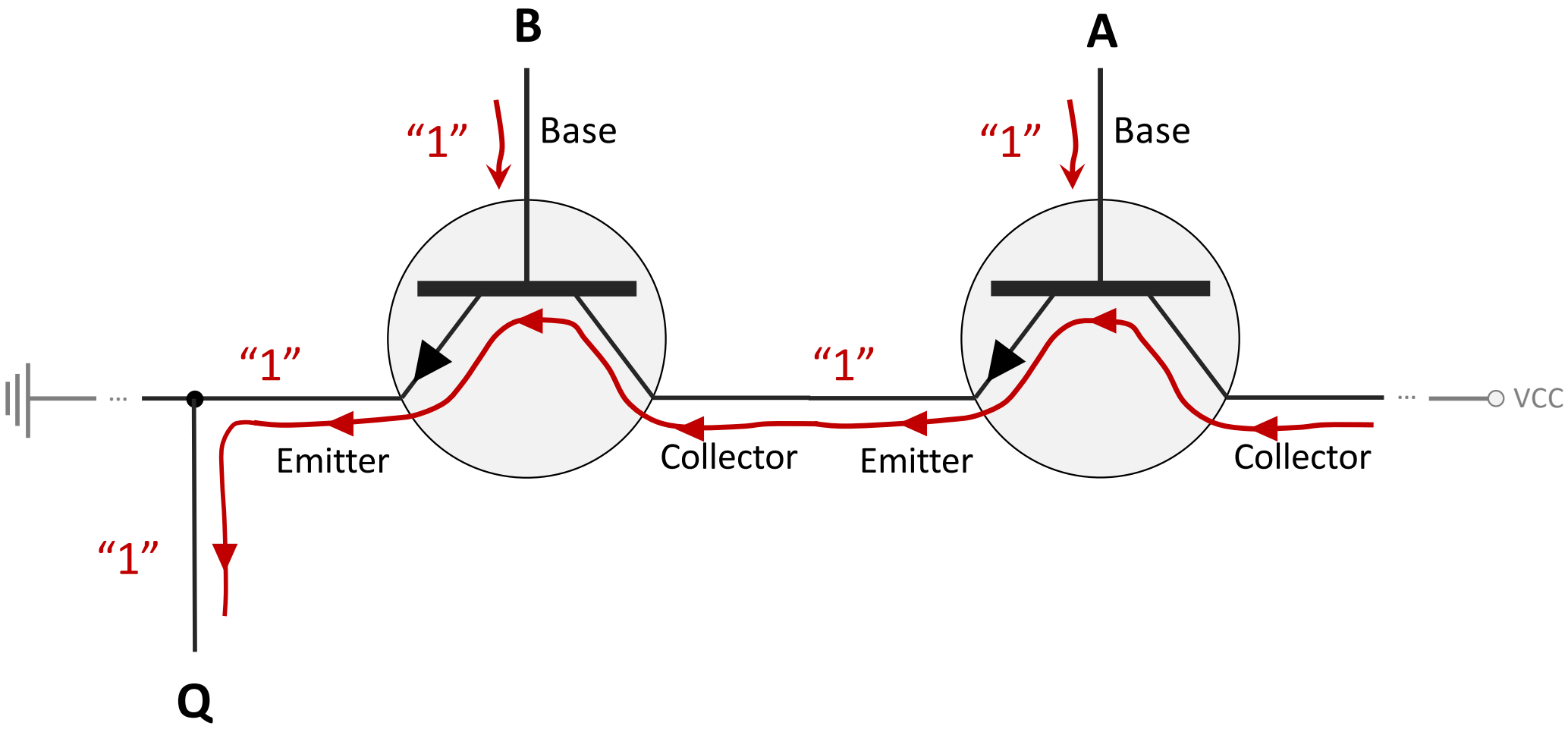



Transistor is a fast switch:

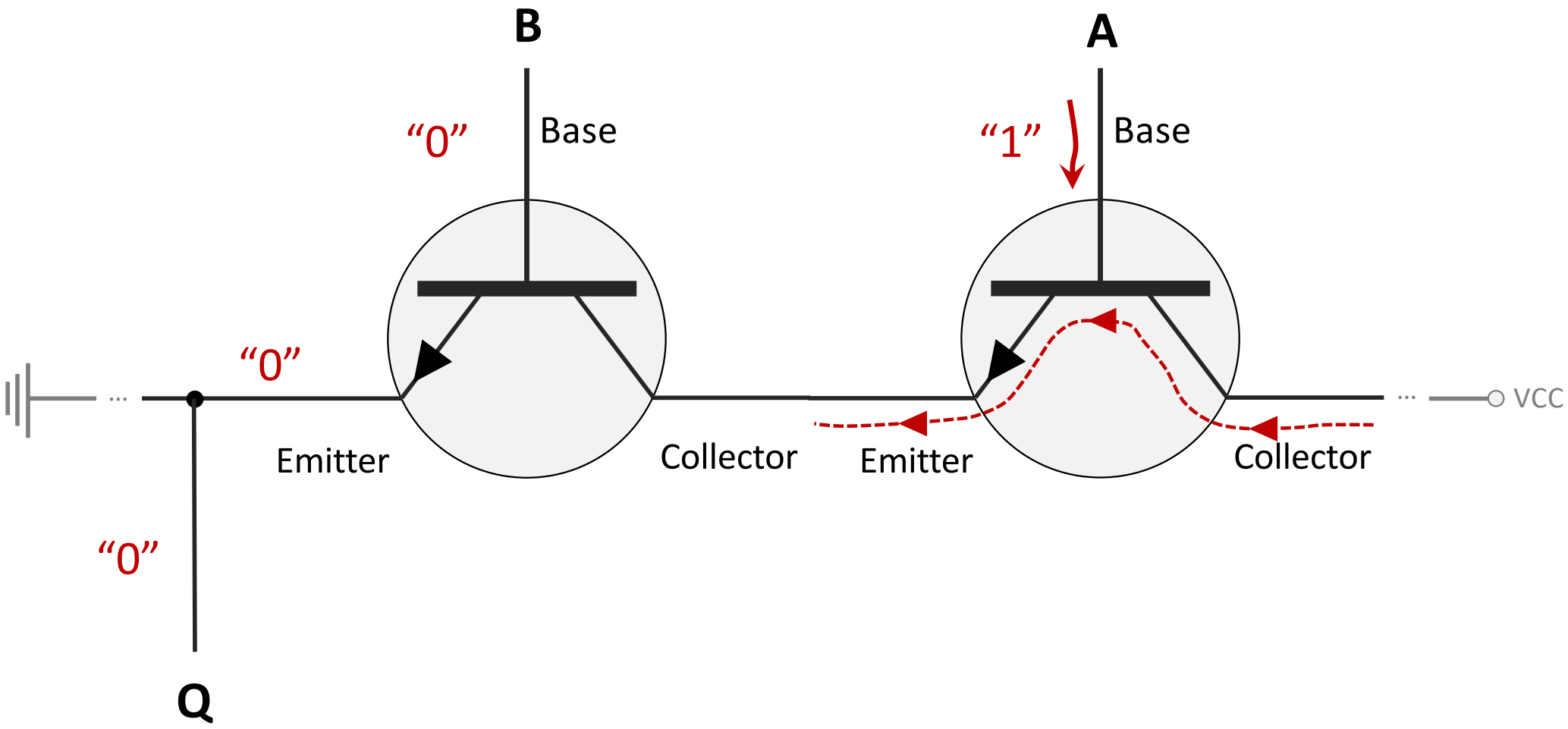
If there is voltage at the Base terminal (B=1), then current flows between Emitter and Collector;

Otherwise (B=0), there is no current

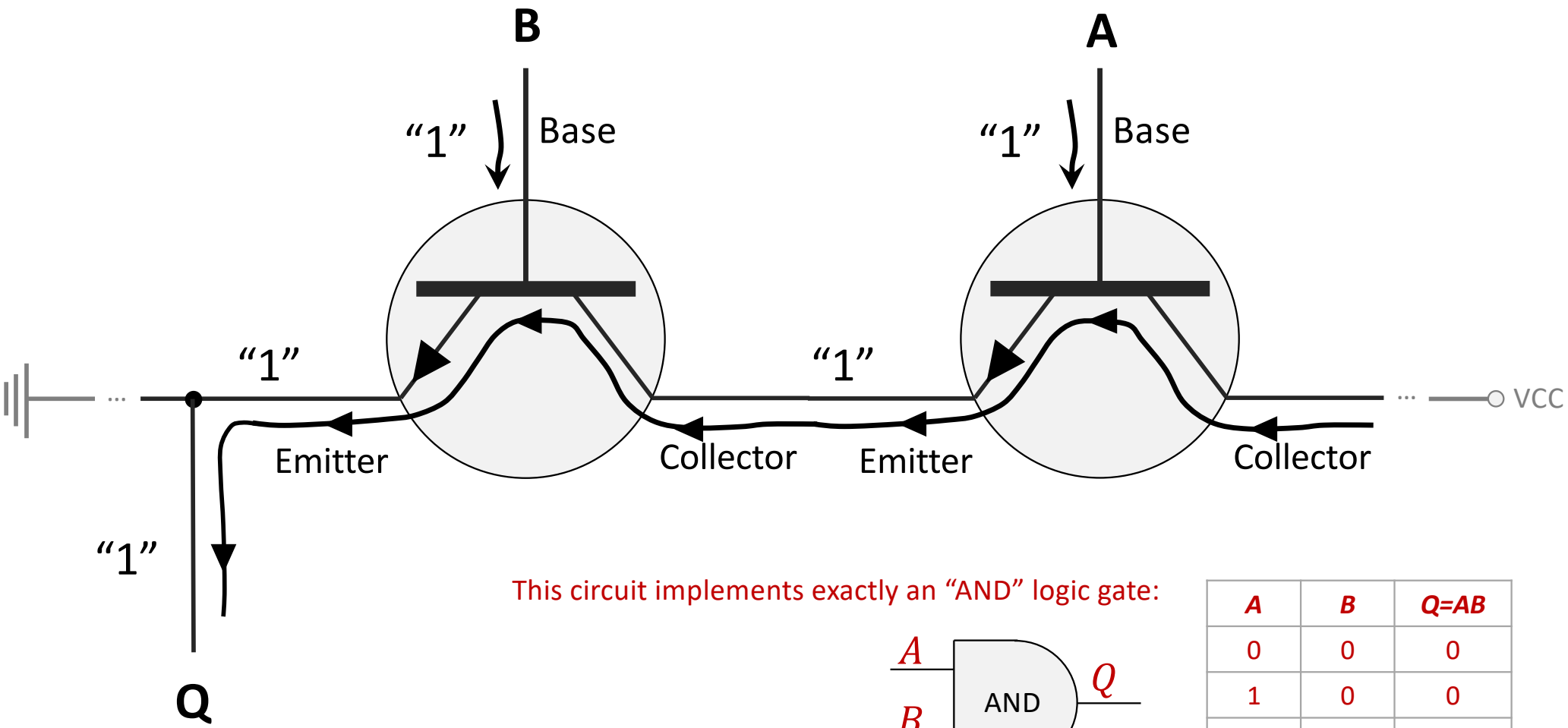
Transistor: the key building block for logic gates



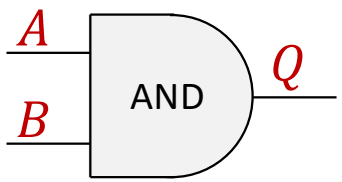
Transistor: the key building block for logic gates



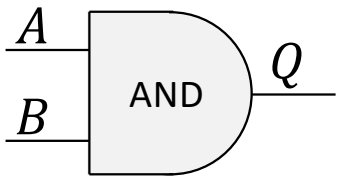
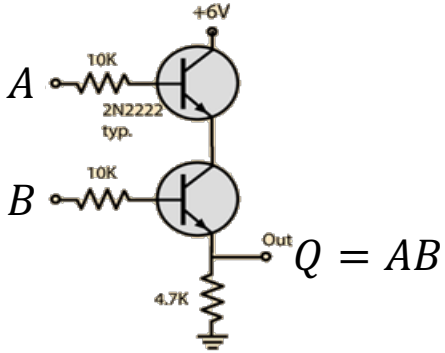
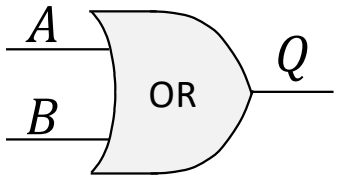
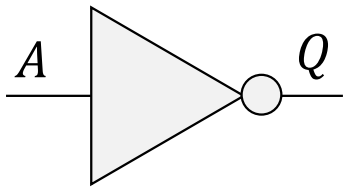
# Transistor: the key building block for logic gates



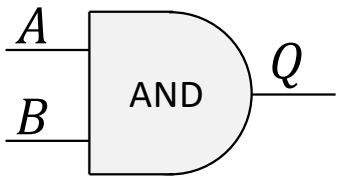
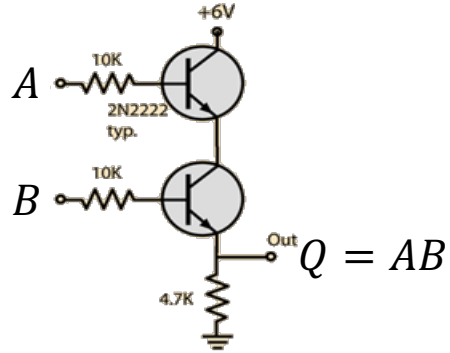
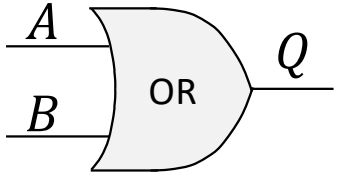
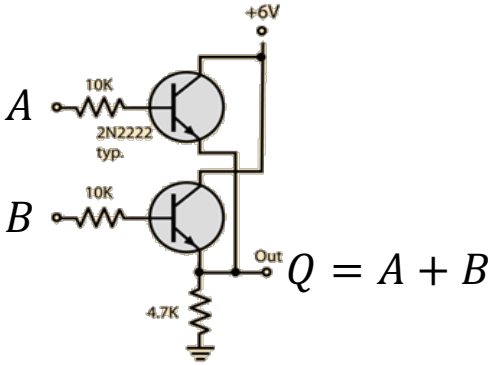
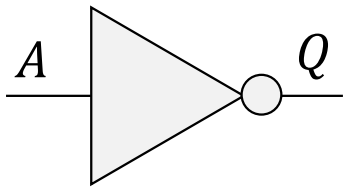

This circuit implements exactly an "AND" logic gate:



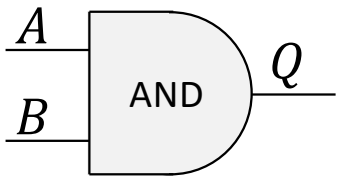
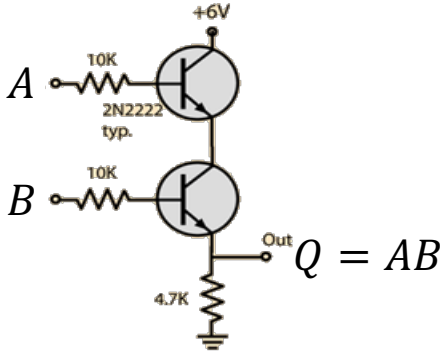
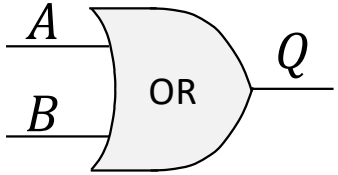
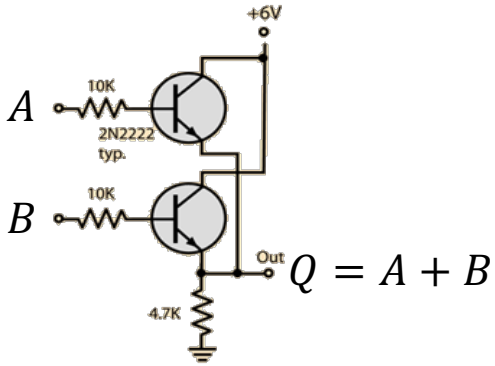
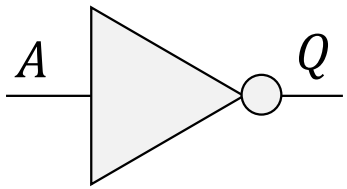
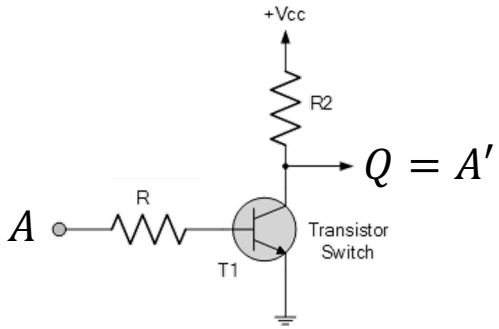

| *A* | *B* | *Q=AB* |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

<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

| Logic Gate | Symbolic Representation | Truth Table | Implementation with Transistors |
|---|---|---|---|
| **AND** |  | |  |
| **OR** |  | |  |
| **NOT** |  | |  |

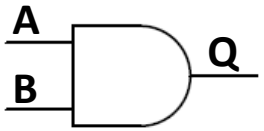
| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

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

| A | Q=A’ |
|---|---|
| 0 | 1 |
| 1 | 0 |

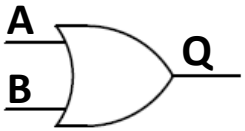
Transistor implementations are taken from http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1 and https://www.electronics-tutorials.ws/logic/logic_4.html

# Selected Logic Gates



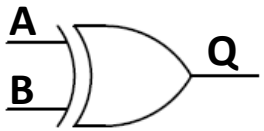

**AND**

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



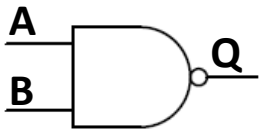

**OR**

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



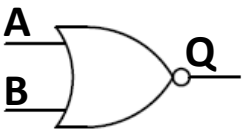

**XOR**

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



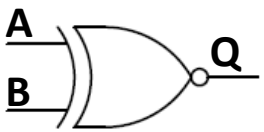

**NAND**

| 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 |

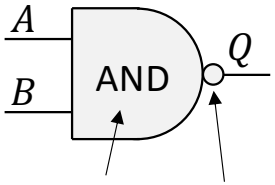



**XNOR**

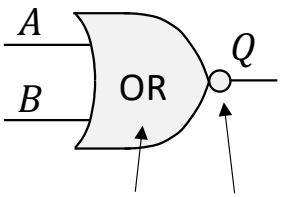
| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

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

# Universal Logic Gates: NAND and NOR

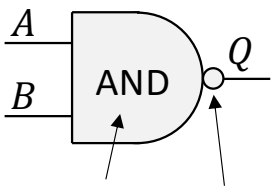



NAND = AND + NOT

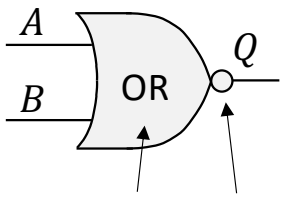



NOR = OR + NOT

# Universal Logic Gates: NAND and NOR



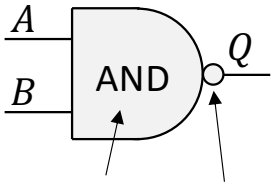

NAND = AND + NOT



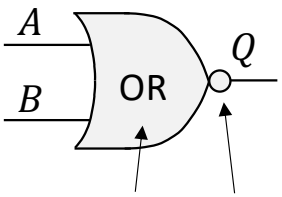

NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

# Universal Logic Gates: NAND and NOR



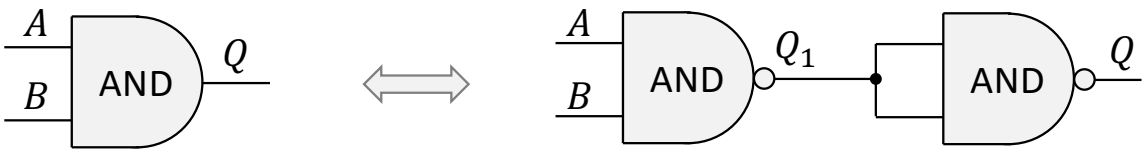

NAND = AND + NOT



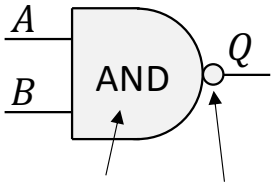

NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

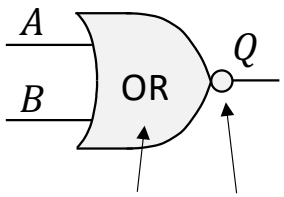
Example: The implementation of AND logic gate by using NAND



# Universal Logic Gates: NAND and NOR



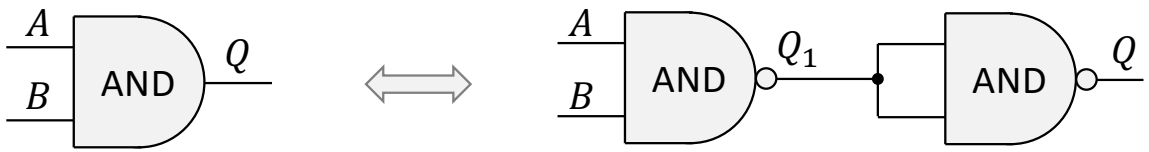

NAND = AND + NOT




NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

Example: The implementation of AND logic gate by using NAND

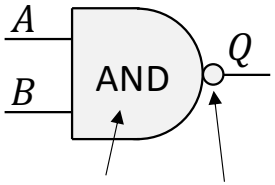



To proof circuits equivalence, we demonstrate the equivalence of outputs for the same sets of inputs:

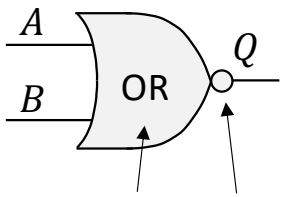
| ***A*** | ***B*** | ***Q=AB*** |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

| ***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



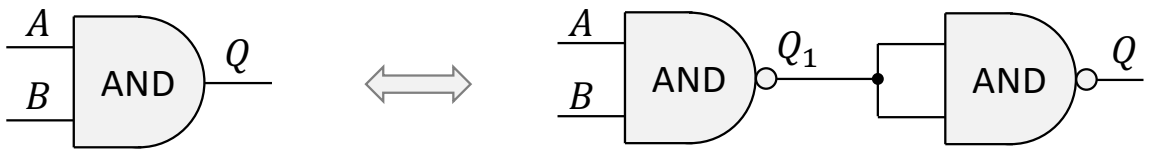

NAND = AND + NOT




NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

Example: The implementation of AND logic gate by using NAND

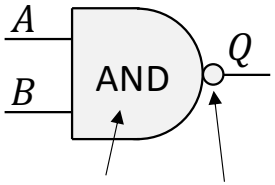



To proof circuits equivalence, we demonstrate the equivalence of outputs for the same sets of inputs:

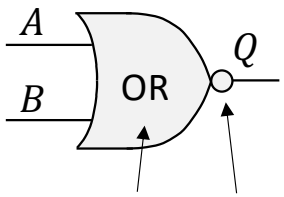
| *A* | *B* | *Q=AB* |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

| *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



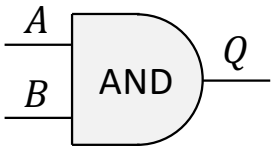

NAND = AND + NOT



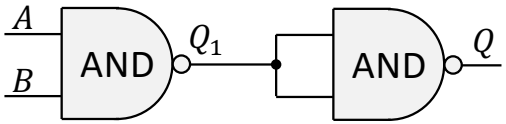

NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

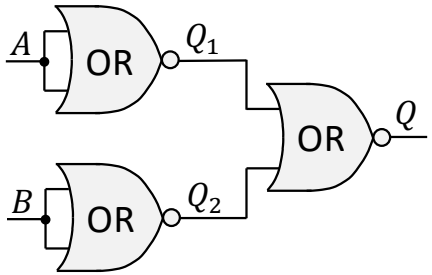
Original gate:



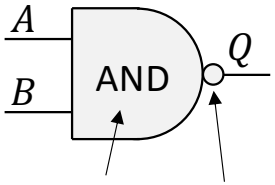

NAND:



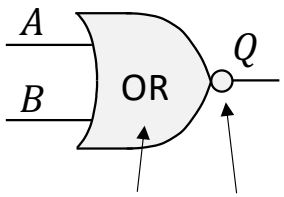

NOR:



# Universal Logic Gates: NAND and NOR



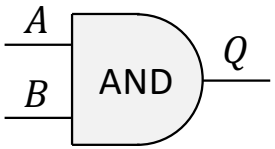

NAND = AND + NOT

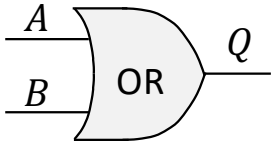



NOR = OR + NOT

A universal logic gate can implement ABSOLUTELY any Boolean function, without need of using other logic gates

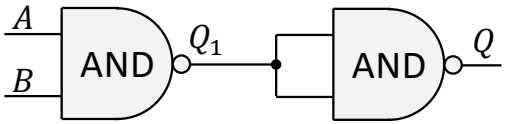
Original gate:

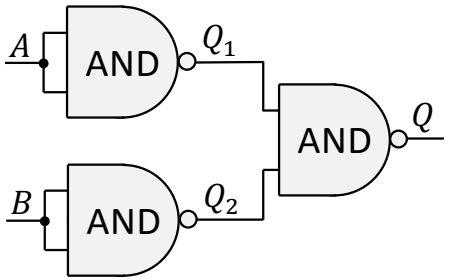


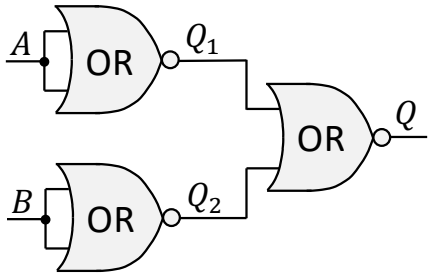

…

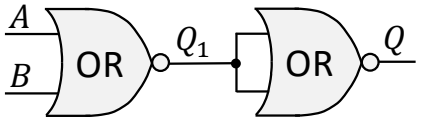
NAND:



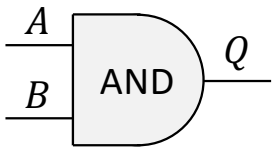

NOR:

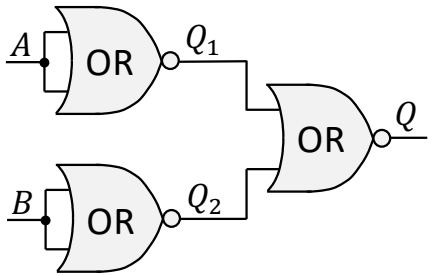


# How to Prove the Equivalence of Logic Circuits?

Way 1: Truth Tables



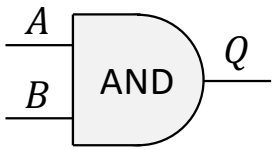



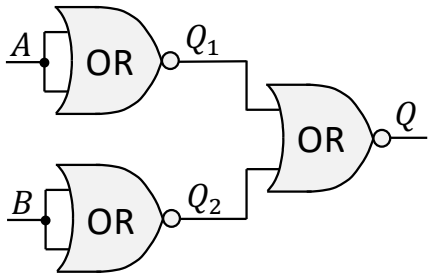

| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

| 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



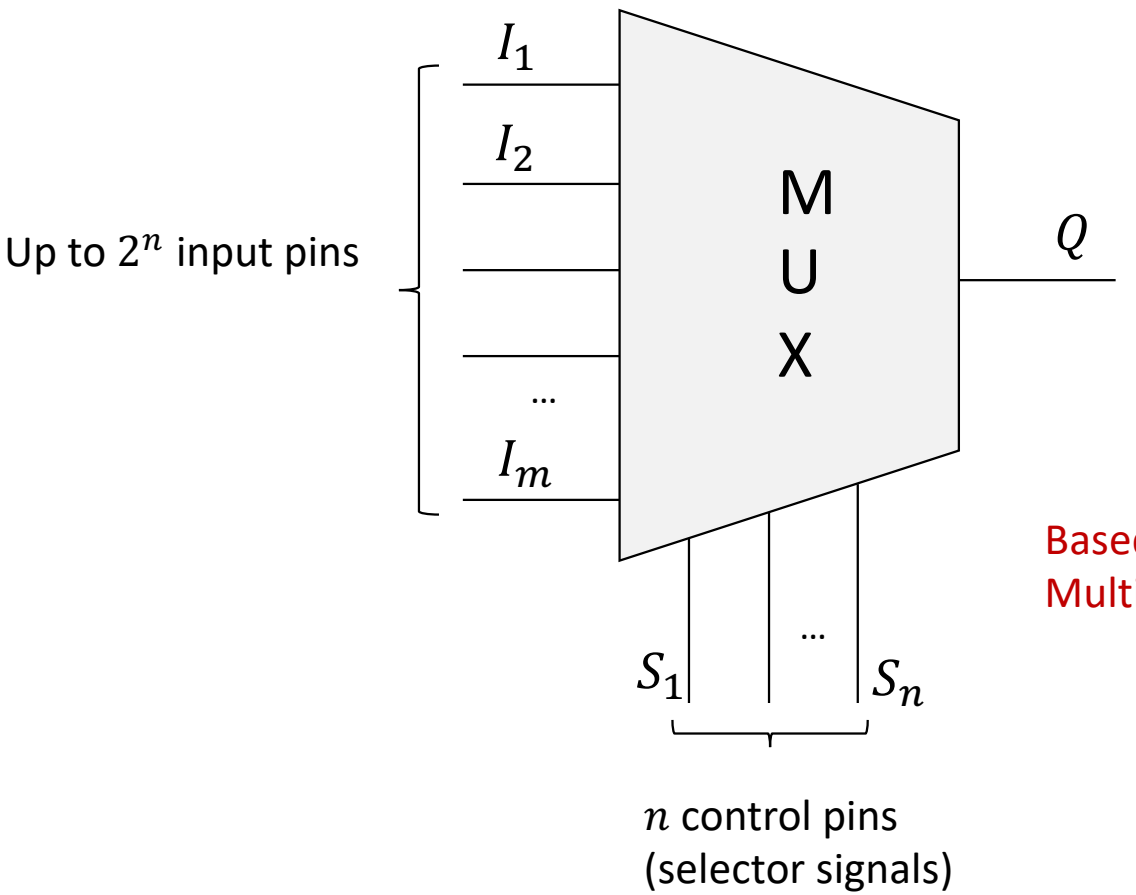

| A | B | Q=AB |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

| 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

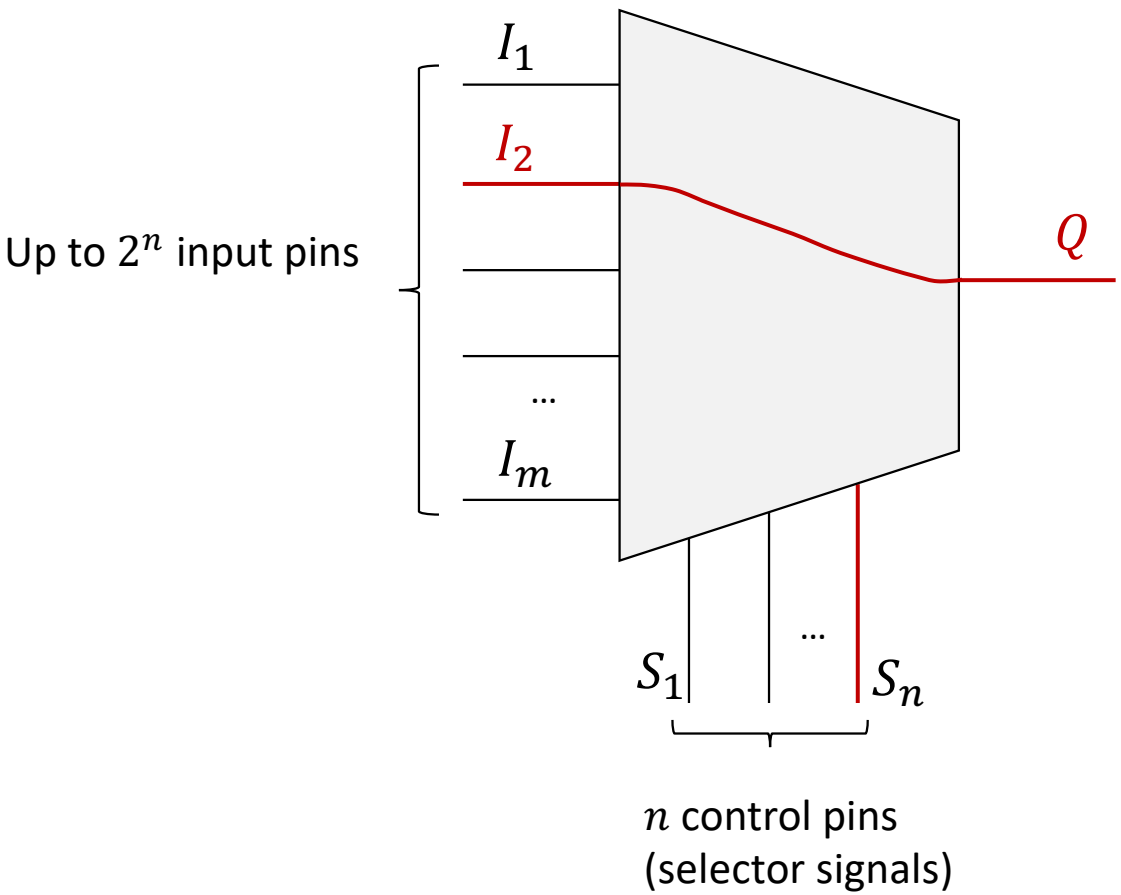
| Name | AND Form | OR Form |
|---|---|---|
| Identity Law | 1A = A | 0 + A = A |
| Null Law | 0A = 0 | 1 + A = 1 |
| Idempotent Law | AA = A | A + A = A |
| Inverse Law | AA' = 0 | A + A' = 1 |
| Commutative Law | AB = BA | A + B = B + A |
| Associative Law | (AB)C = A(BC) | (A + B) + C = A + (B + C) |
| 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: An Example of a Combinational Logic Circuit



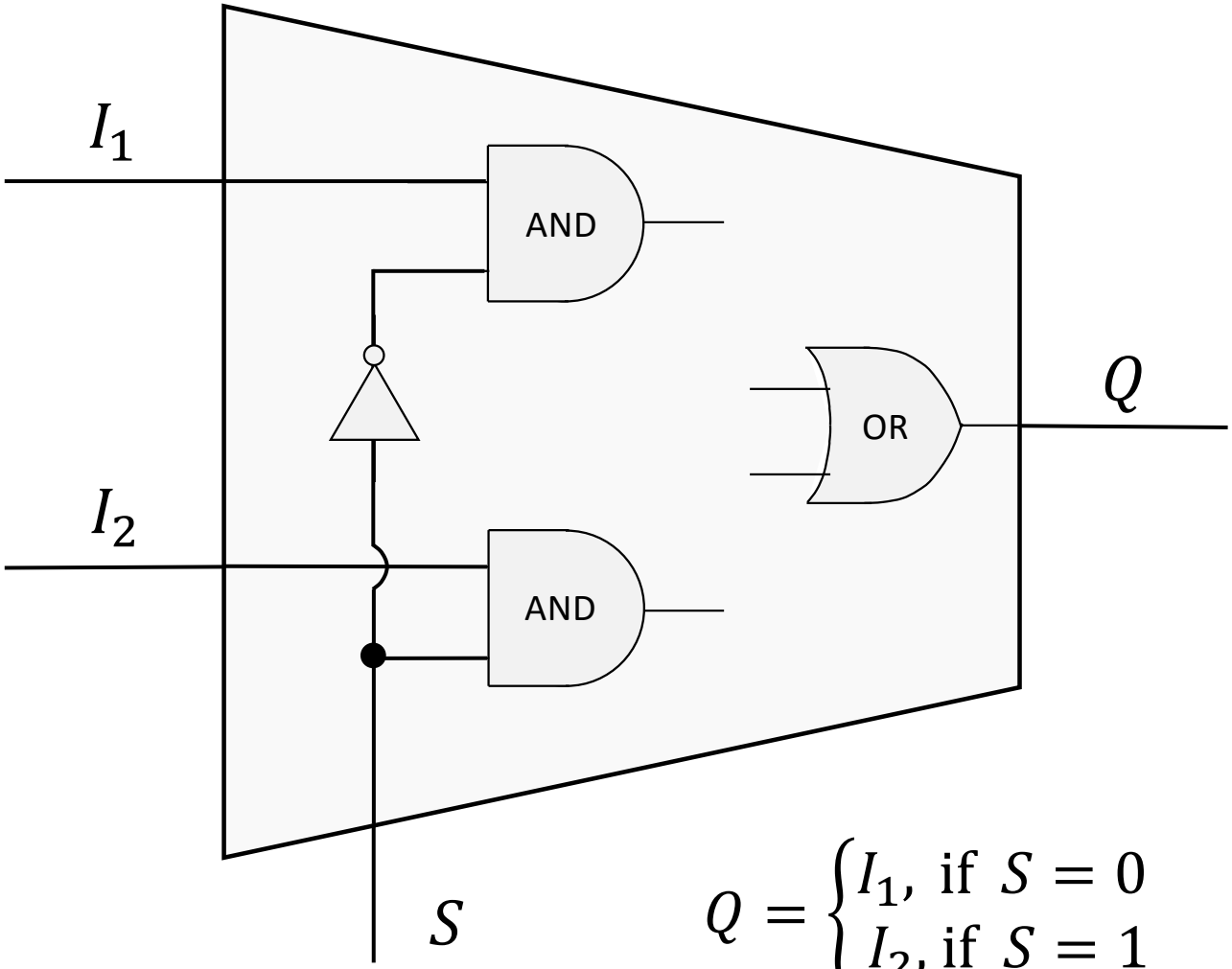

Based on the signals of control pins,
Multiplexor sets its ouput to one of its inputs

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



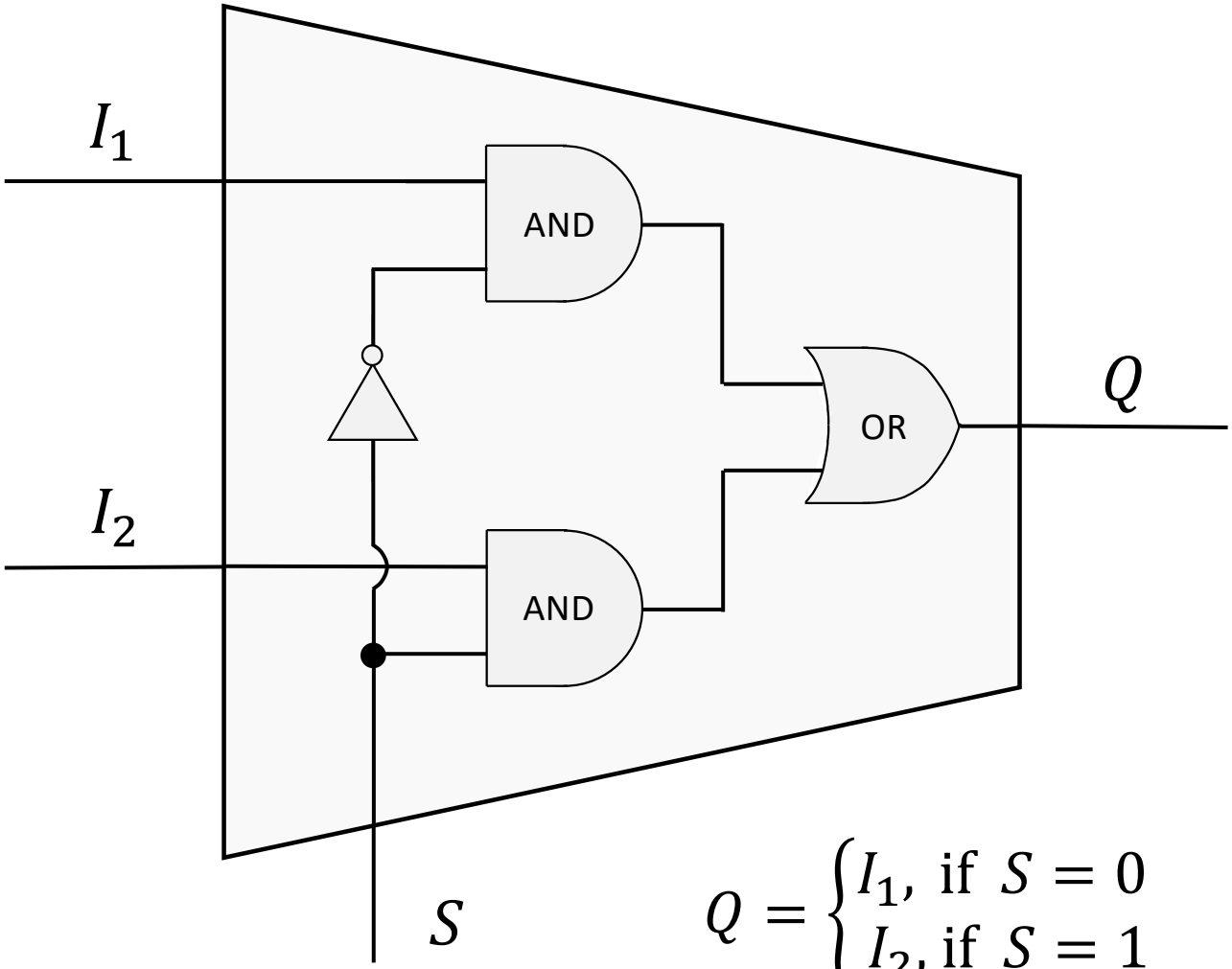

Based on the signals of control pins, sets its ouput to one of its inputs

# Sample Implementation of a 2-to-1 Multiplexor



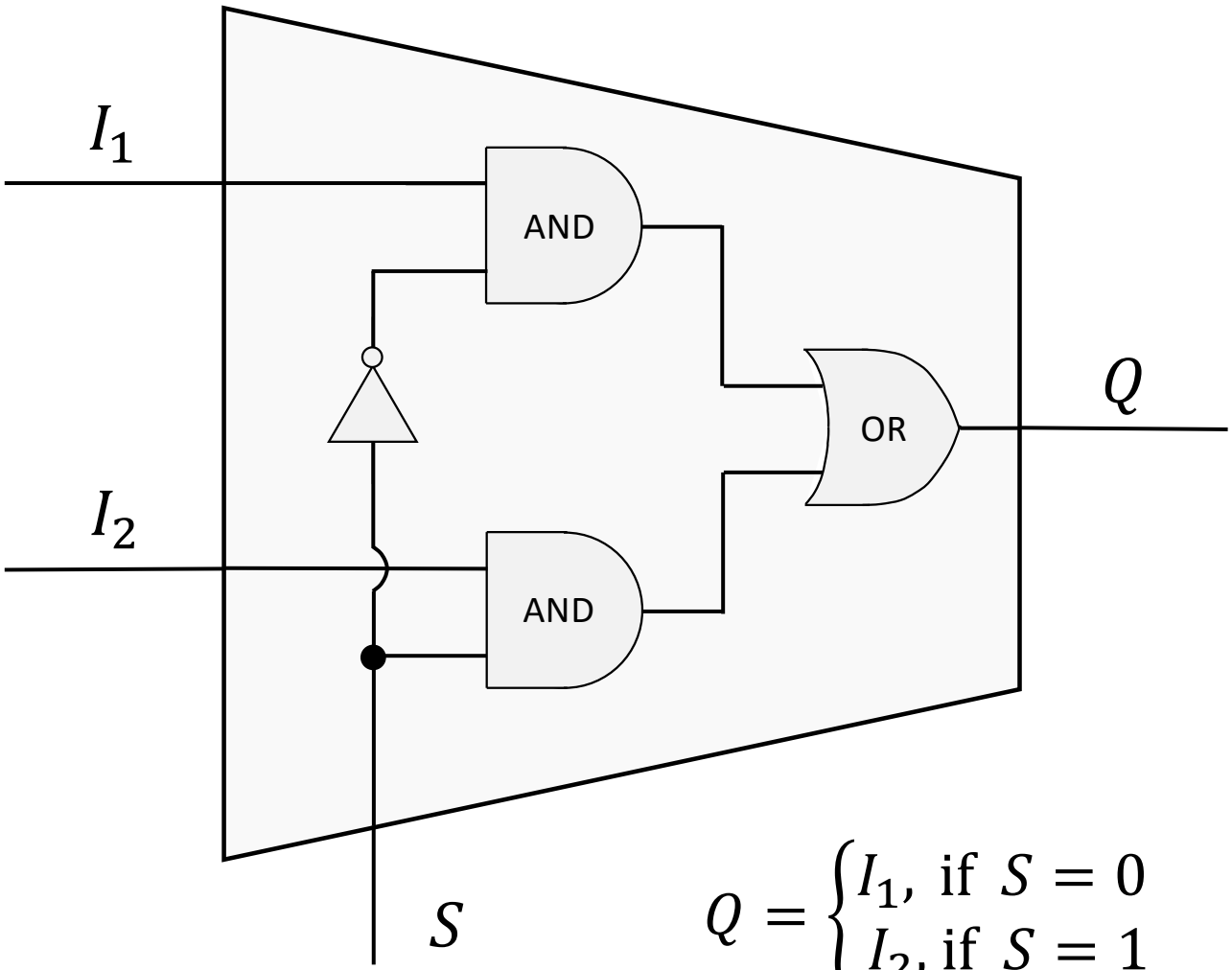

$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ I_2, \text{ if } S = 1 \end{cases}$$

# 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

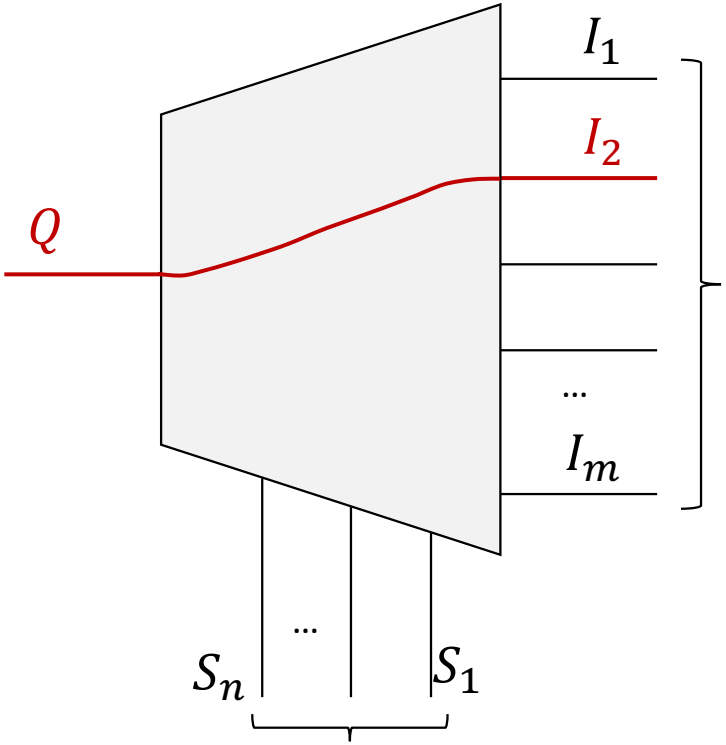



$$Q = \begin{cases} I_1, \text{ if } S = 0 \\ I_2, \text{ if } S = 1 \end{cases}$$

Verification of
the implementation 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** |

# Demultiplexor

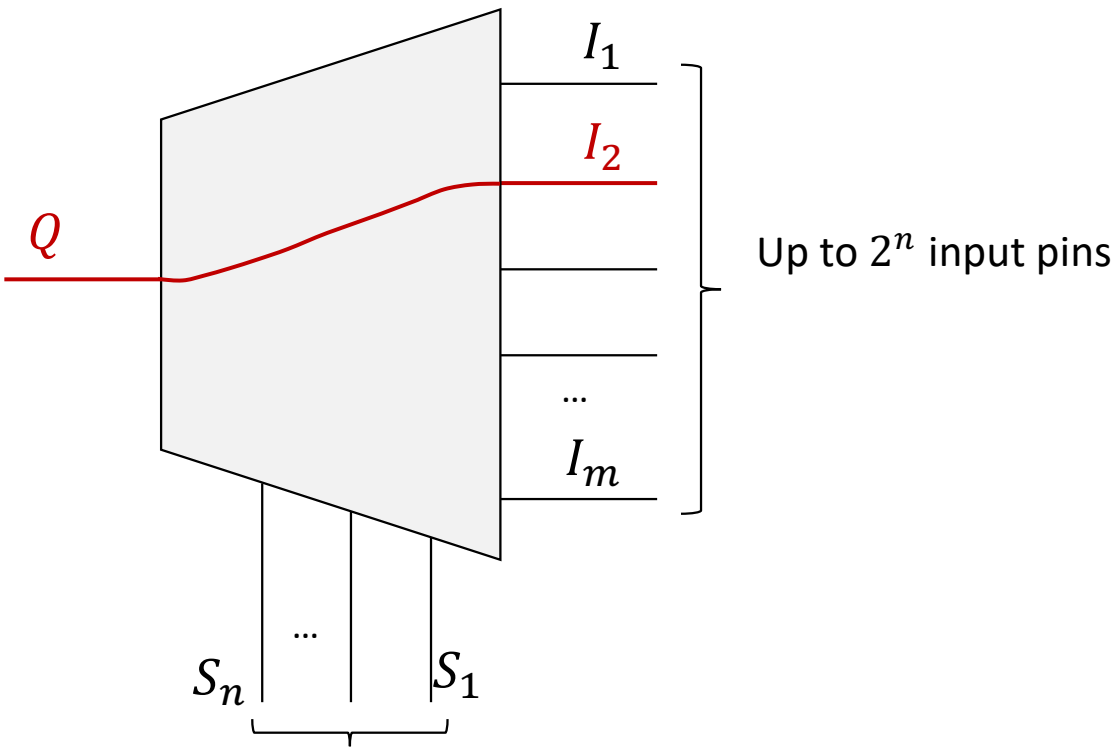



Up to $2^n$ input pins

$n$ control pins
(selector signals)

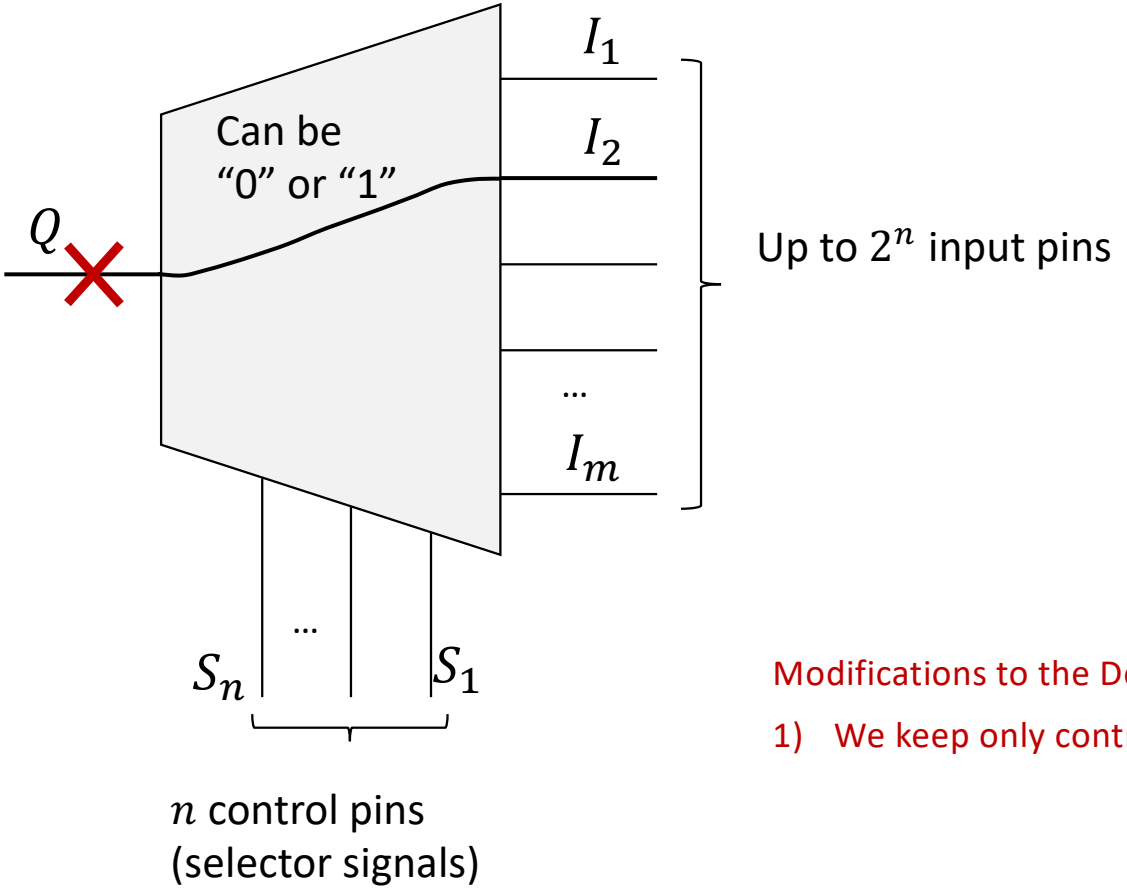
Based on the signals of control pins,
Demultiplexor sends its input to one of its outputs

# Demultiplexor



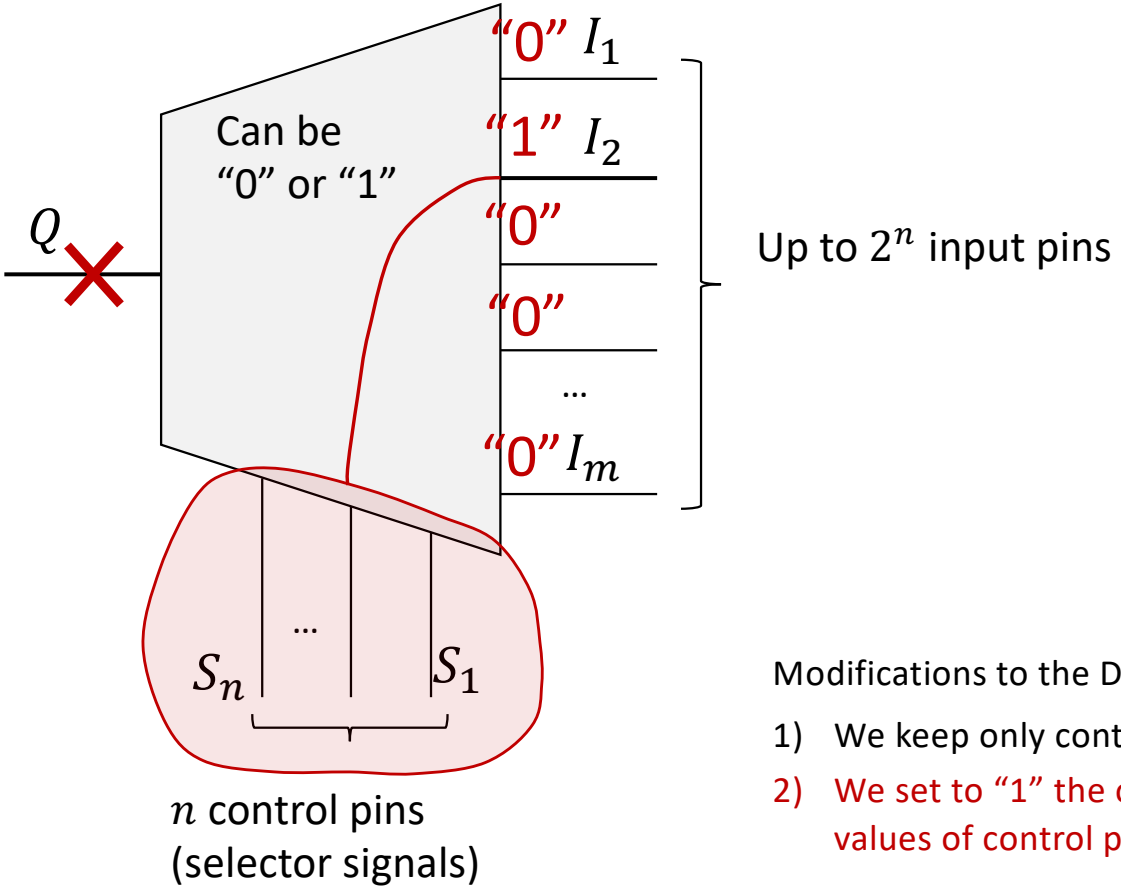

$n$ control pins
(selector signals)

Based on the signals of control pins,
Demultiplexor sends its input to one of its outputs




Modifications to the Decoder:

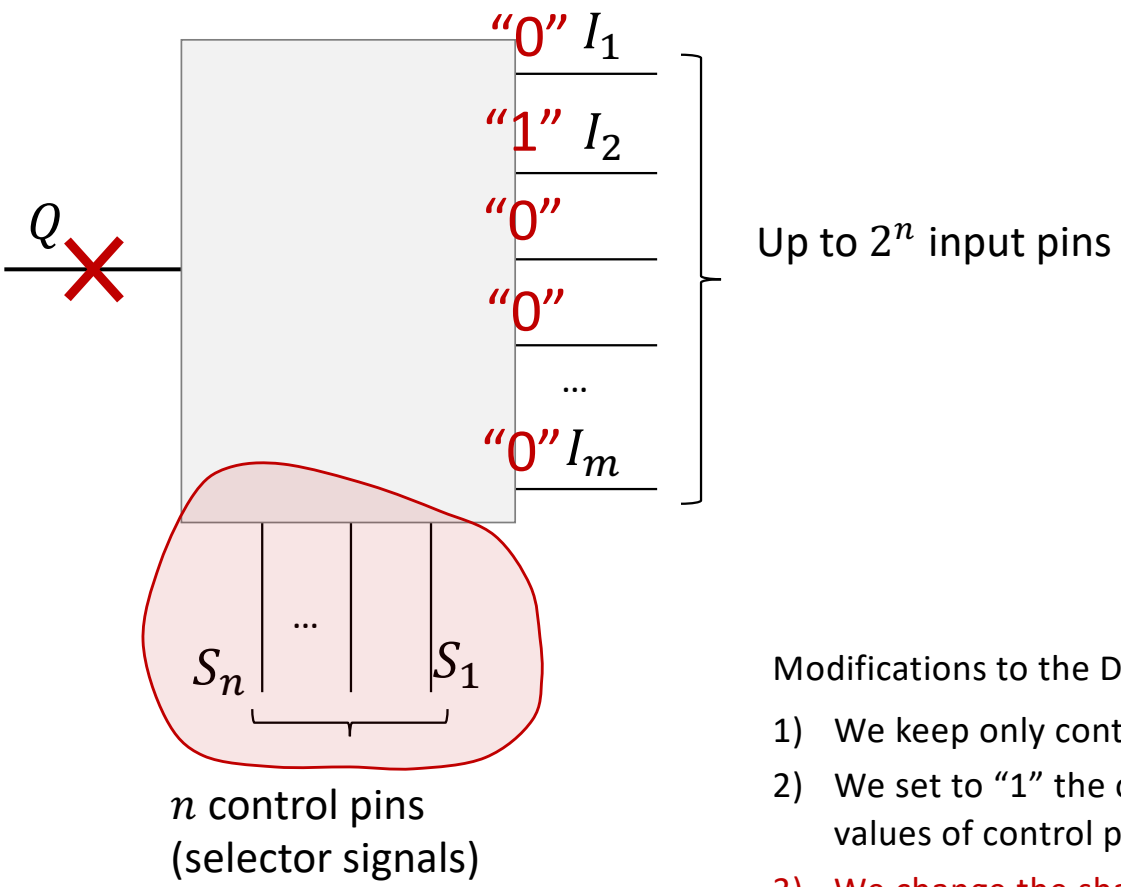
1) We keep only control pins, and remove input pin Q;




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;

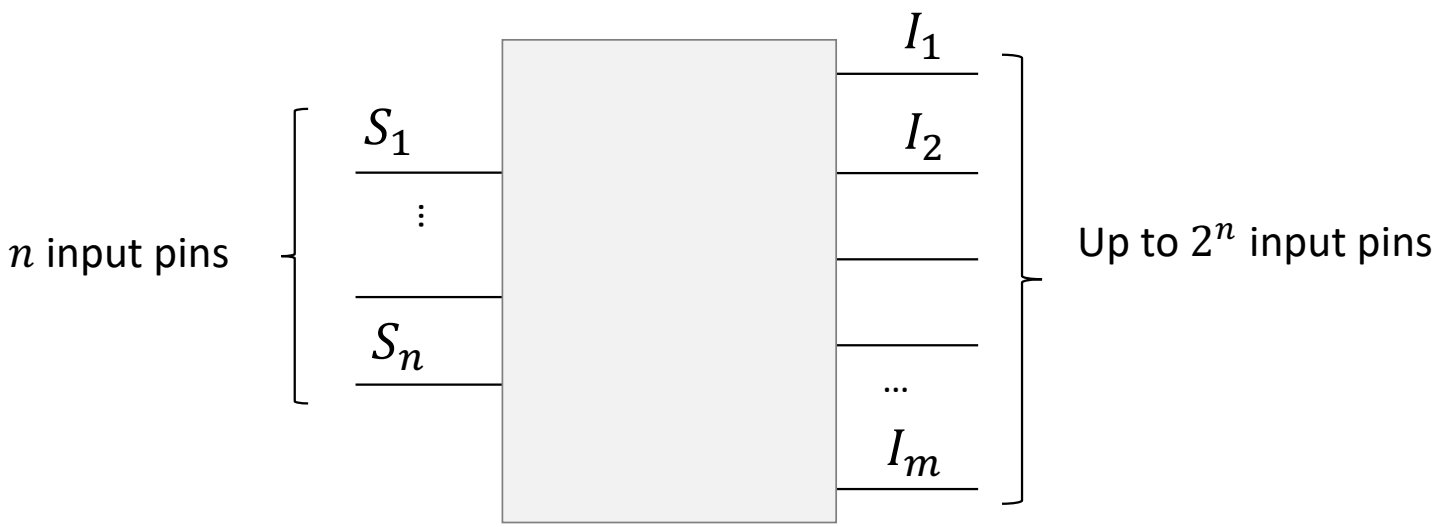
# Decoder




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

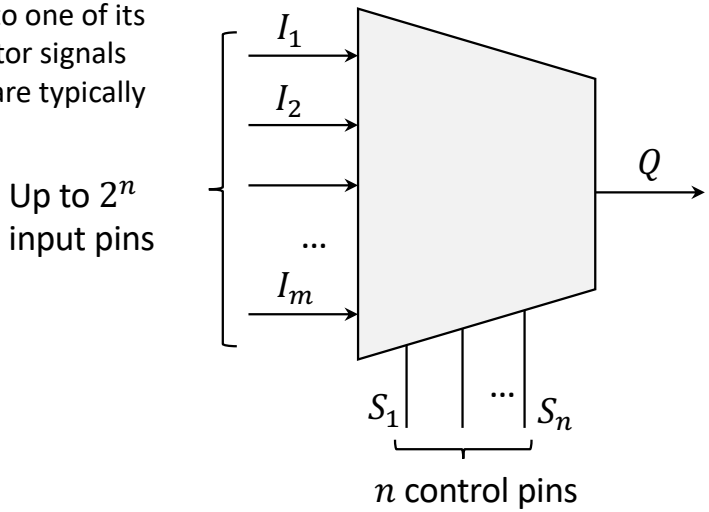



Modifications to the Decoder:

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

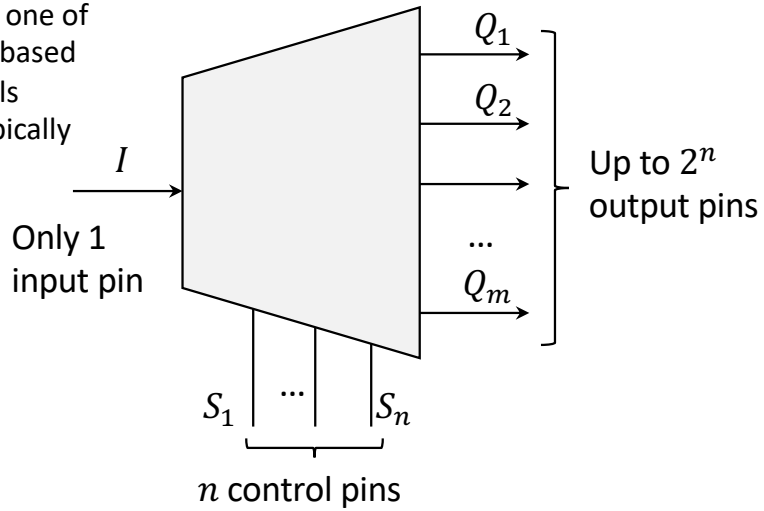
## Multiplexor

Sets a specific output to one of its inputs, based on selector signals (all other output pins are typically set to “0”)



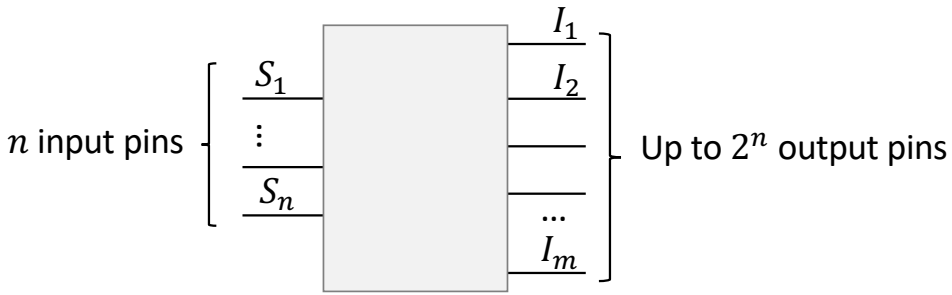

## Demultiplexor

Sends input signal to one of its multiple outputs, based on the selector signals (all other outputs typically remain “0”)



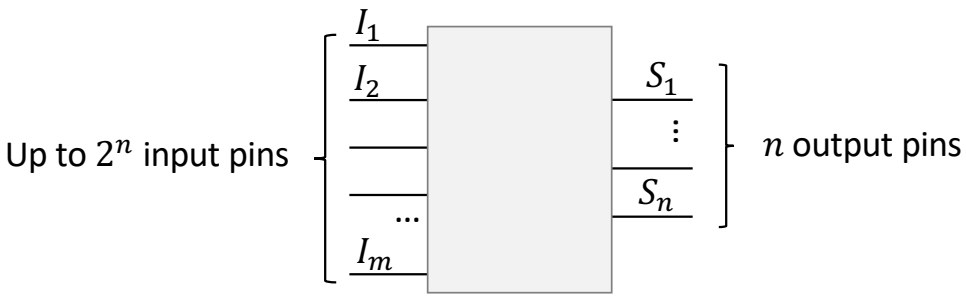

## Decoder

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



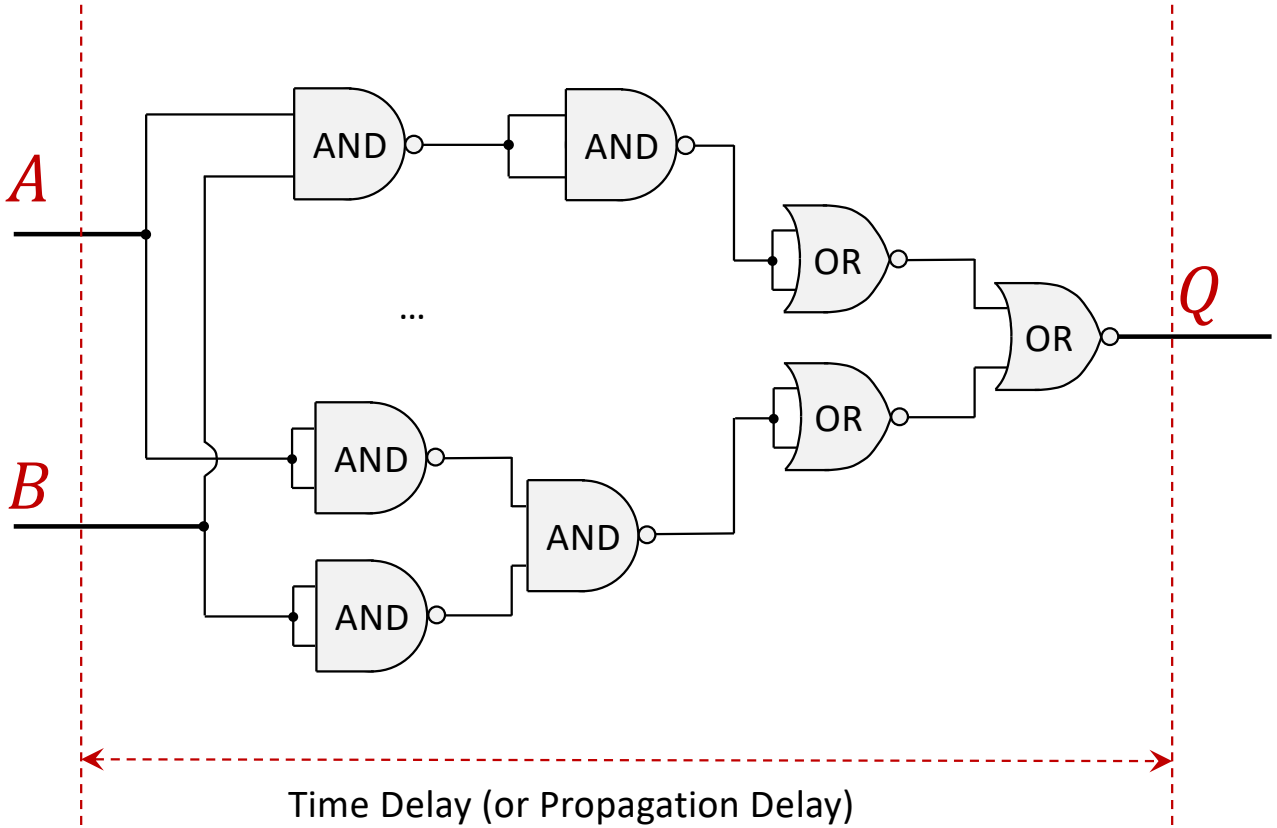

## Encoder

The opposite function of an encoder;
Only one input pin is assumed to be “1”, while all others – “0”



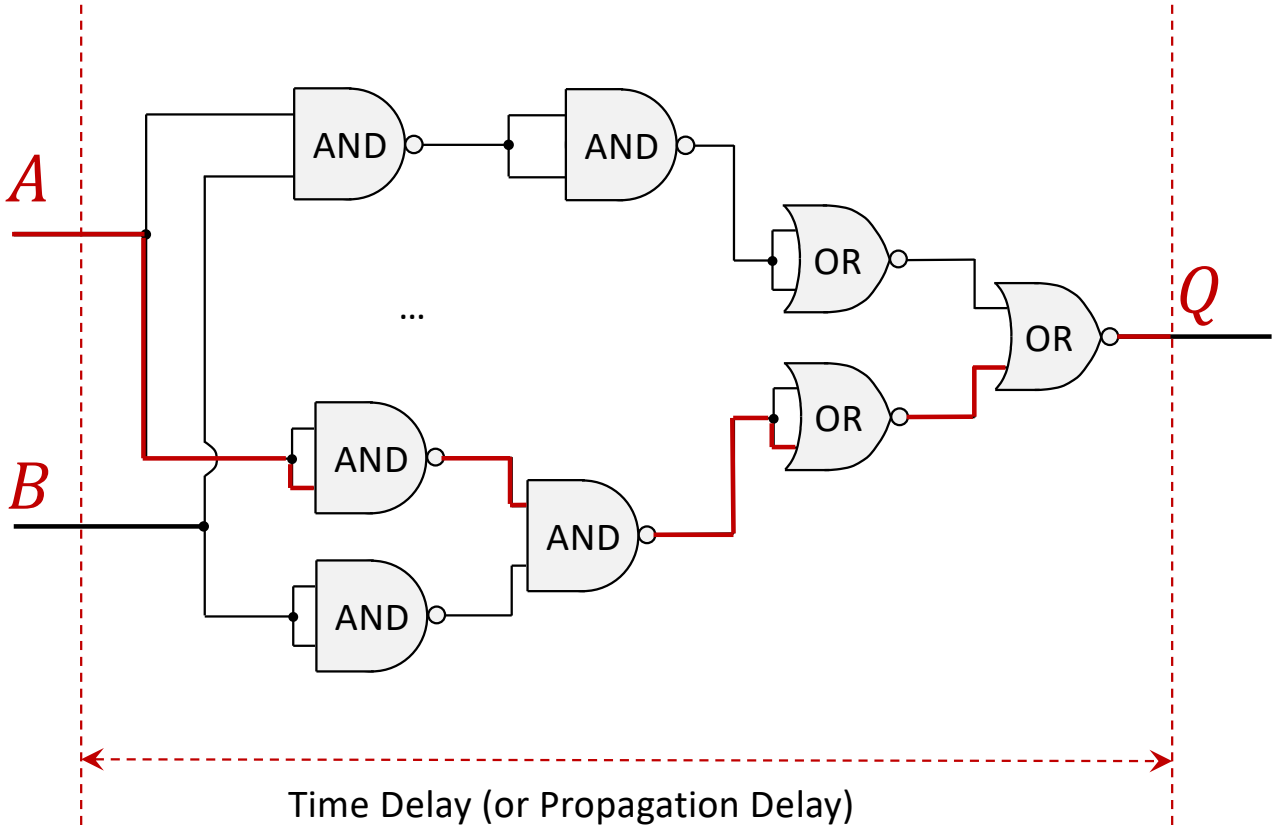
# Propagation Delay

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

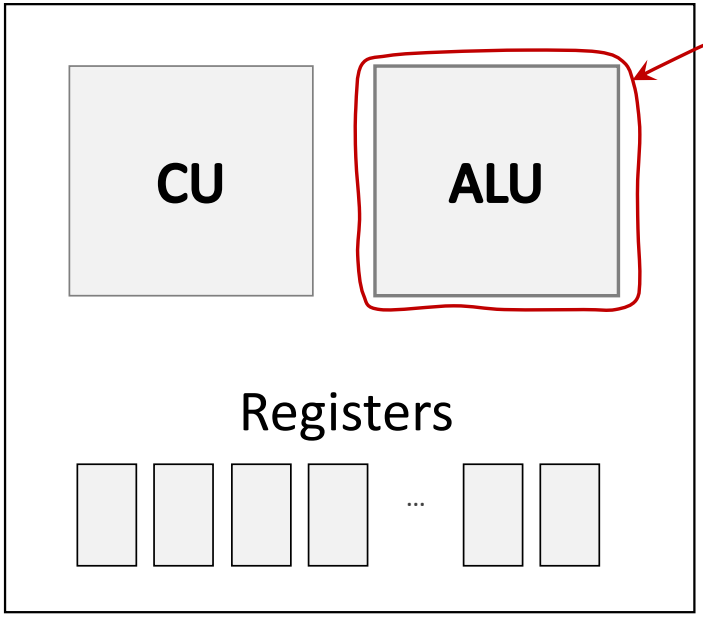
Combines input signals and outputs the result;

Implemented by using logic gates, such as AND, OR, NOT, NAND, etc.

Does not use memory elements (registers);

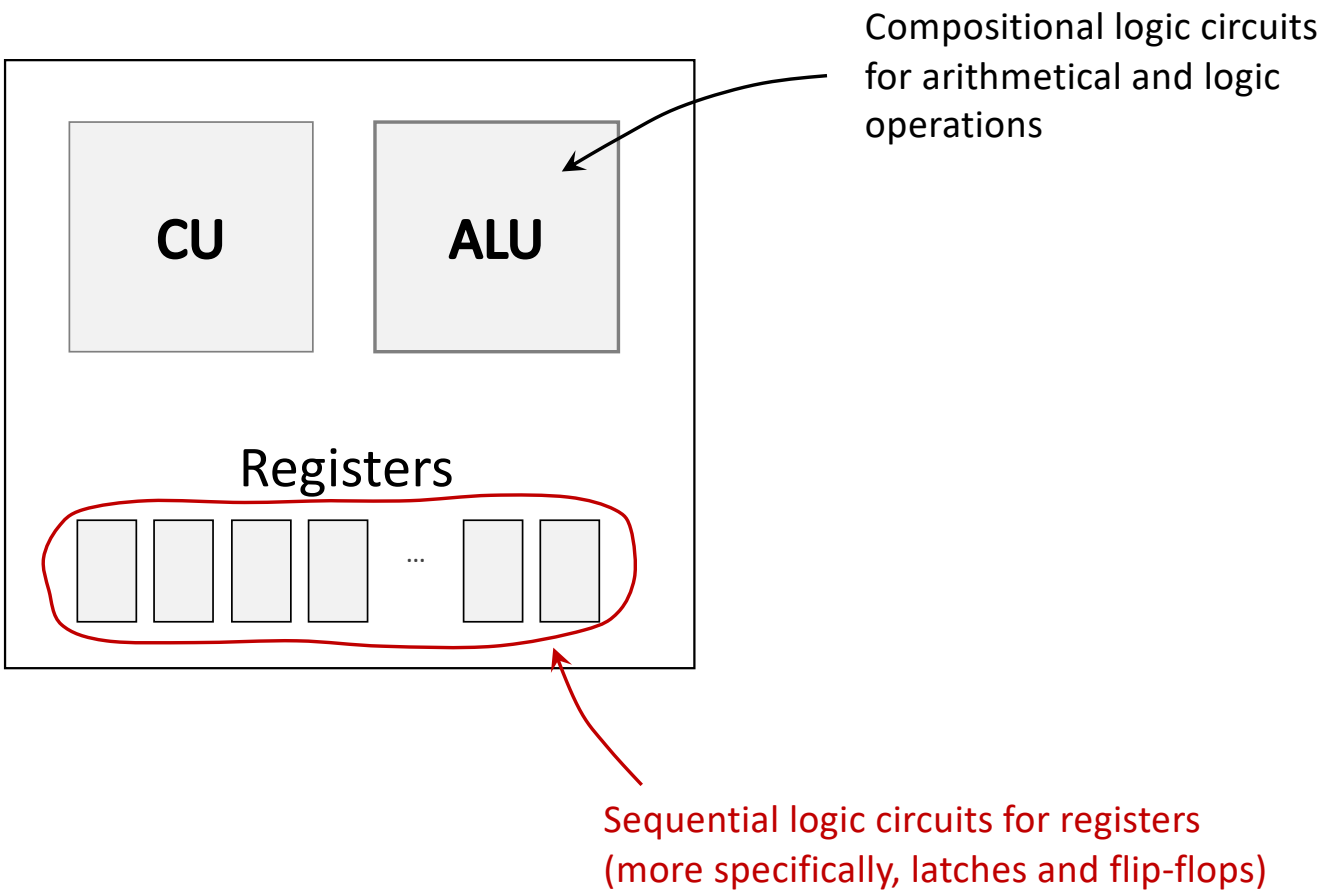
Typical use: arithmetic operations

# Types of Logic Circuits for CPU Components




Compositional logic circuits for arithmetical and logic operations

# Types of Logic Circuits for CPU Components



Sequential Logic = Combinational Logic + Memory Units

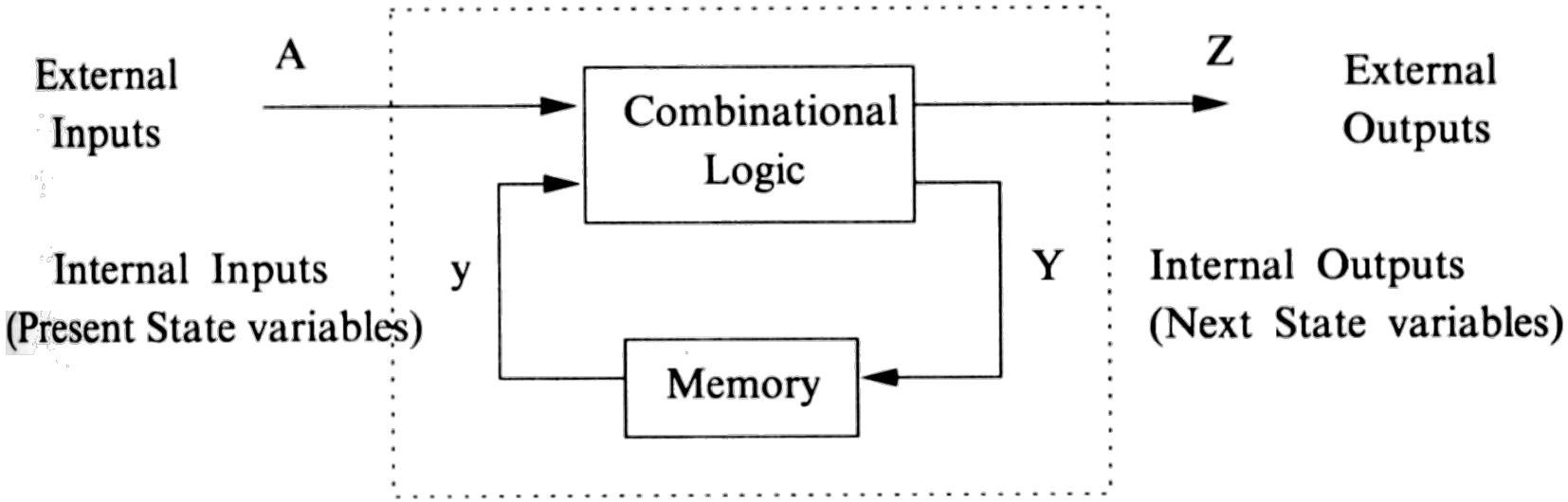



Image is taken from http://www.ee.surrey.ac.uk/Projects/CAL/seq-switching/General_seq_circ.htm

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |
| **Speed (Propagation Delay)** | Fast | Slow |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |
| **Speed (Propagation Delay)** | Fast | Slow |
| **Implementation Complexity** | Simple | Complex |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |
| **Speed (Propagation Delay)** | Fast | Slow |
| **Implementation Complexity** | Simple | Complex |
| **Typical Use Cases** | Arithmetic and boolean operations | Data storage |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |
| **Speed (Propagation Delay)** | Fast | Slow |
| **Implementation Complexity** | Simple | Complex |
| **Typical Use Cases** | Arithmetic and boolean operations | Data storage |
| **Examples** | Multiplexor / Demultiplexor<br>Encoder / Decoder | Flip-flop<br>Latches |

# Sequential vs. Combinational Logic Circuits

| | **Combinational Logic** | **Sequential Logic** |
|---|---|---|
| **Memory Elements** | Does not use memory elements;<br>Cannot implement memory elements | Can implement memory elements;<br>Can use memory elements |
| **Time Characteristics** | Time-independent:<br>the output depends only on present inputs, but not on past inputs | Time-dependent:<br>the output depends not only on present inputs, but on past inputs as well |
| **Speed (Propagation Delay)** | Fast | Slow |
| **Implementation Complexity** | Simple | Complex |
| **Typical Use Cases** | Arithmetic and boolean operations | Data storage |
| **Examples** | Multiplexor / Demultiplexor<br>Encoder / Decoder | Flip-flop<br>Latches |