3 Logic Gates and Simple Computer Circuits

Consider the following statements

- All humans are mortals
- Socrates is human
- Therefore Socrates is mortal

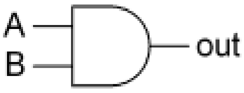
This system of reasoning is an example of *logic*.

3.1 *Logic gates*

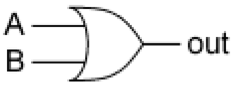
Modern digital circuits are easier to build using two voltage levels (high/low, true/false, 1/0, etc.) rather than several voltage levels. Electronic circuits called logic gates are used to implement logic in electronic devices.

The functioning of logic gates can be described in terms of their *truth tables*, tables that show for every possible combination of inputs, the corresponding outputs. This section presents the common logic gate symbols, their truth tables, and for each, the algebraic expression that describes its functioning.

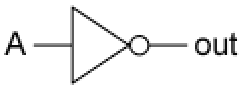
3.1.1 AND Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expressions</th></tr>
<tr><td> </td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td><td>$out = AB$
$out = A \cdot B$</td></tr>
<tr><td colspan="3">The AND gate gives an output of ‘1’ only when both inputs are ‘1’</td></tr>
</table>

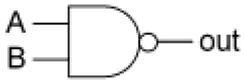
3.1.2 OR Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td> </td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td><td>$out = A + B$</td></tr>
<tr><td colspan="3">The OR gate gives an output of ‘1’ when either input is ‘1’</td></tr>
</table>

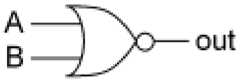
3.1.3 NOT Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td>
</td><td><table>
<tr><th>INPUT</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>out</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table></td><td>$out = \overline{A}$</td></tr>
<tr><td colspan="3">Also called the inverter, the NOT gate reverses or complements its input</td></tr>
</table>

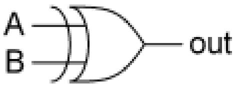
3.1.4 NAND Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td>
</td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>out</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table></td><td>$out = \overline{AB}$
$out = \overline{A.B}$</td></tr>
<tr><td colspan="3">The NAND gate complements the results that the AND gate would have given</td></tr>
</table>

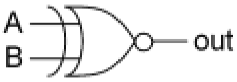
3.1.5 NOR Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td>
</td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>out</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table></td><td>$out = \overline{A + B}$</td></tr>
<tr><td colspan="3">The NOR gate complements the result that the OR gate would have given</td></tr>
</table>

3.1.6 Exclusive-Or (XOR) Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td>
</td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table></td><td>$out = A \oplus B$</td></tr>
<tr><td colspan="3">The XOR gate gives an output of ‘1’ when either, but not both inputs, is ‘1’</td></tr>
</table>

3.1.7 Exclusive-NOR (XNOR) Gate

<table>
<tr><th>Symbol</th><th>Truth Table</th><th>Acceptable Algebraic Expression</th></tr>
<tr><td>
</td><td><table>
<tr><th colspan="2">INPUTS</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td><td>$out = \overline{A \oplus B}$</td></tr>
<tr><td colspan="3">The XNOR gate complements the output of the XOR gate</td></tr>
</table>

3.2 Simple Computer Circuits

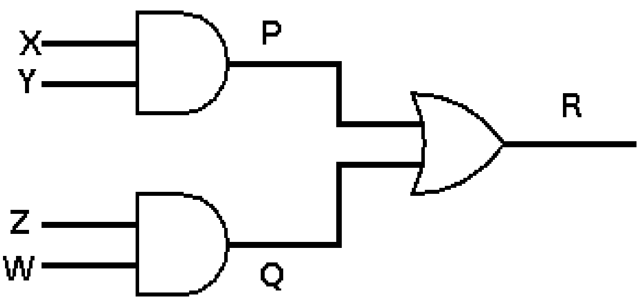
In this section, we shall see how logic gates can be connected together to provide some logic function.

We start by generating logic expressions and truth tables for a sample of logic circuits. We shall however, not generate logic equations or logic circuits from truth tables; neither shall we attempt to simplify logic expressions.

Then, we introduce two broad categories of logic circuits: combinational and sequential logic circuits, and illustrate practical examples of these types of circuits.

3.2.1 Logic Equations and Truth Tables from Logic Circuits

Example 1



Intermediate Outputs

$P = X \cdot Y$

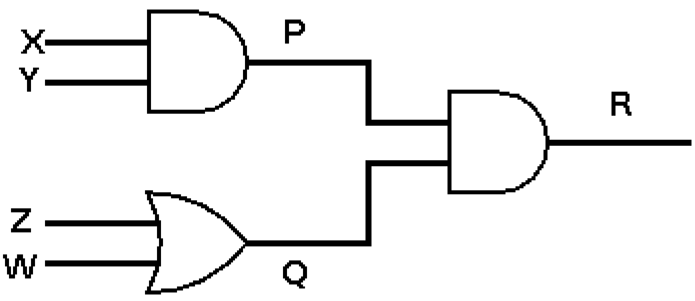
$Q = Z \cdot W$

Output

$$R = P + Q$$
$$R = X \cdot Y + Z \cdot W$$

INPUTS				INT OUTPUTS		OUTPUT
X	**Y**	**Z**	**W**	**P**	**Q**	**R**
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Example 2



Intermediate Outputs

$P = X \cdot Y$

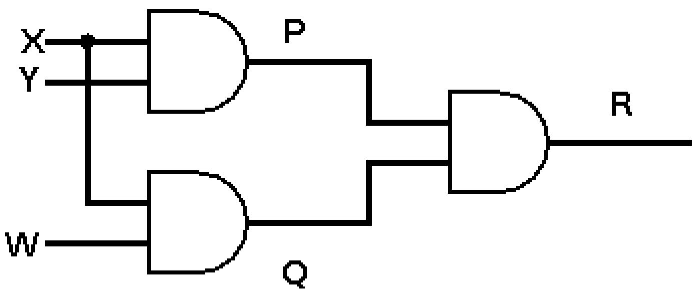
$Q = Z + W$

Output

$$R = P \cdot Q$$
$$R = (X \cdot Y) \cdot (Z + W)$$

INPUTS				INT OUTPUTS		OUTPUT
X	**Y**	**Z**	**W**	**P**	**Q**	**R**
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

Example 3



Intermediate Outputs

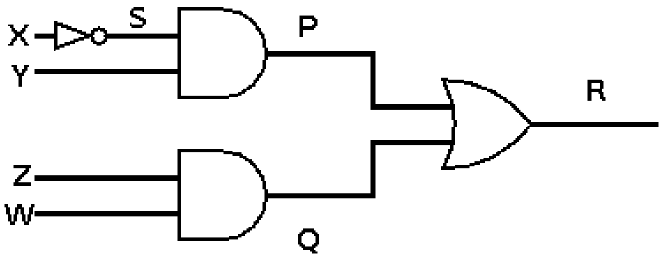
$P = X \cdot Y \qquad Q = X \cdot W$

Output:

$$R = P \cdot Q$$
$$R = (X \cdot Y) \cdot (X \cdot W)$$

INPUTS			INT OUTPUTS		OUTPUT
X	**Y**	**W**	**P**	**Q**	**R**
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Example 4



Intermediate Outputs

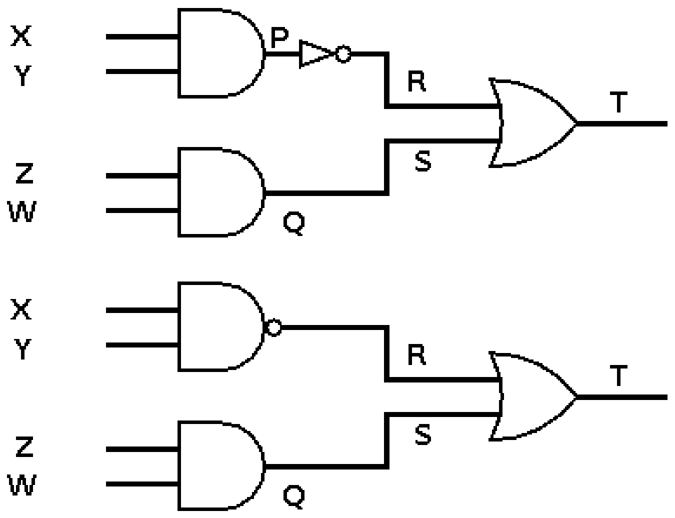
$S = \overline{X} \qquad P = S \cdot Y \qquad Q = X \cdot W$

Output

$$R = P + Q$$
$$R = S \cdot Y + (X \cdot W)$$
$$R = \overline{X} \cdot Y + (X \cdot W)$$

INPUTS				INT OUTPUTS			OUTPUT
X	**Y**	**Z**	**W**	**S**	**P**	**Q**	**R**
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Example 5



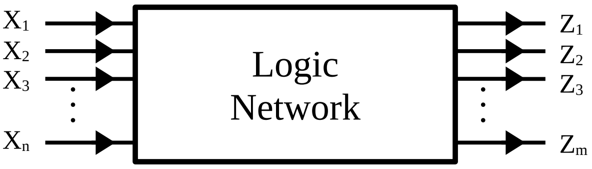
INPUTS				INT OUTPUTS			OUTPUT
X	**Y**	**Z**	**W**	**S**	**P**	**Q**	**R**
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Intermediate Outputs

Output

3.2.2 Combinational Logic Circuits

A combinational logic circuit is one whose output(s) depend(s) only on the inputs; there is no feedback from the output side to the input side. A combinational logic circuit operates in accordance with its truth table regardless of any prior inputs to which the circuit may have been exposed.



All the logic circuits in Section 3.2.1 are examples of combinational logic circuits. As we saw, the functioning of a circuit in given in the form of a logical equation or a truth table. Logic equations normally are given either in the form of *sum of products* or *product of sums* (also known as canonical) forms. Logic equations can be simplified using Boolean algebra, but we do not do that in this course.

Figure 33 shows a logical circuit that can be used to implement the full adder (see truth table in Figure 15, page 11). The student is expected to (1) write a logic equation for the circuit, and (2) generate the truth table from the logic circuit.

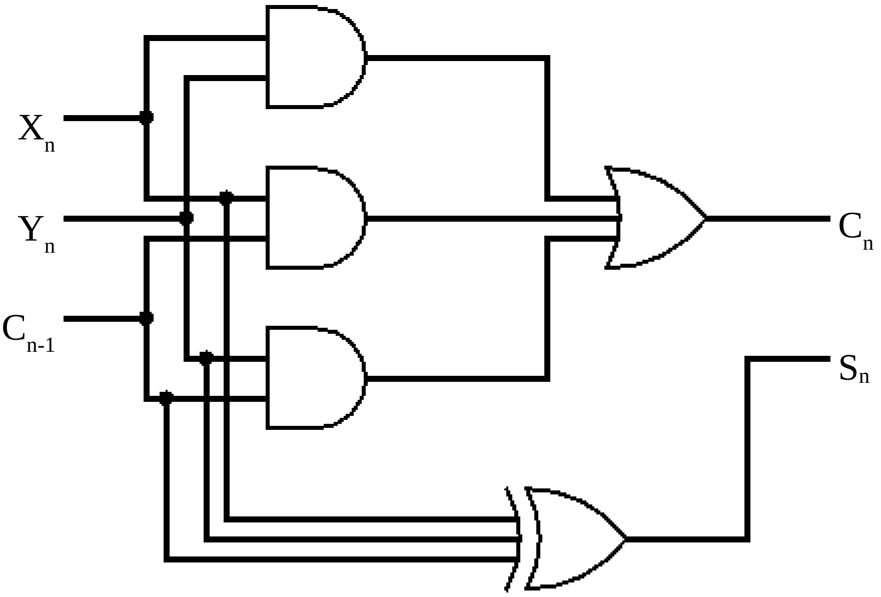
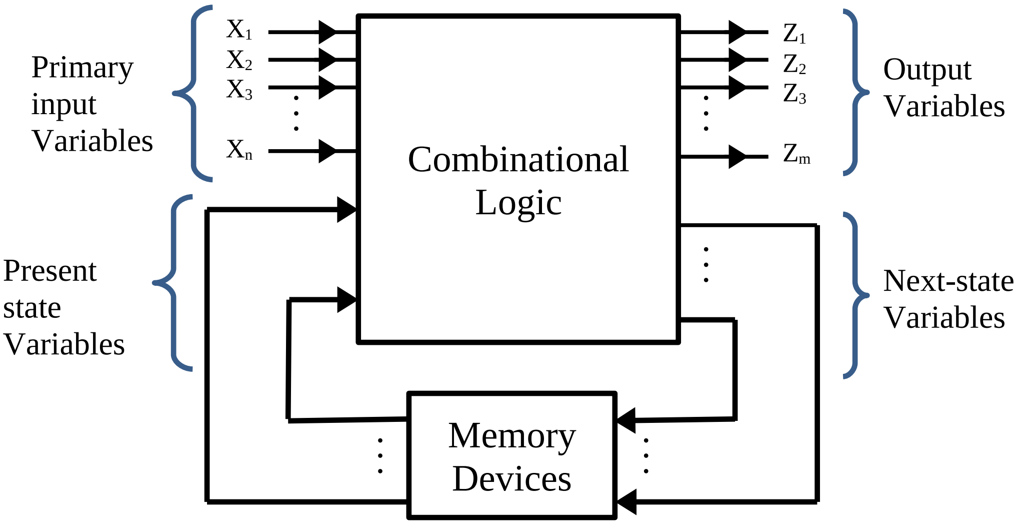


Figure 33: Full Adder Circuit

3.2.3 Sequential Circuits

We saw in the previous section that in combinational logic circuits, the binary value of any output is a function only of one or more of the current inputs, and has no dependence whatsoever on any previous output, i.e., there is no feedback connection from the output to the input section of the circuit.

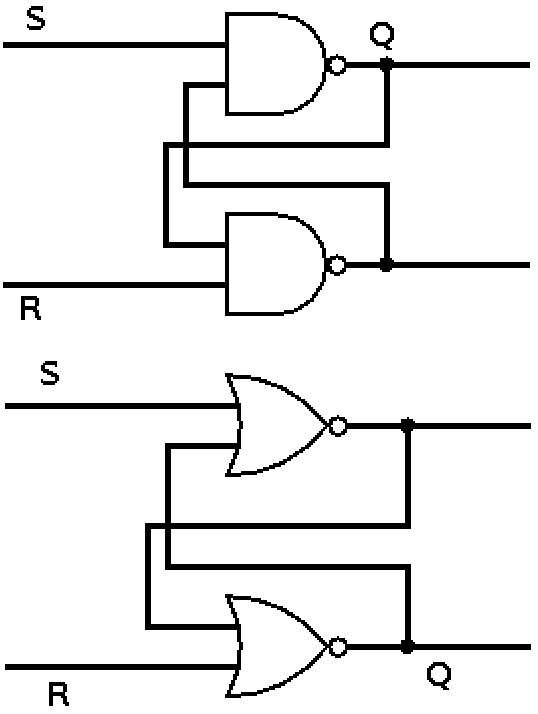
Sequential circuits differ from combinational ones in that there is at least one feedback connection from the output to the input side of the circuit. The current outputs of sequential circuits depend therefore, not only on the current inputs, but on prior inputs as well.



The feedback connections in sequential circuits give them the ability to have *memory*, i.e., at time t_n, the circuit *knows* something about time t_{n-1}.

Flip-flops

A flip-flop or latch circuit, is a circuit that has two stable states and can be used to store state information. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. Flip-flops and latches are a fundamental building block of memory circuits. Figure 34 shows two simple latch circuits and their operation tables: the NAND latch and NOR latch.



S	**R**	**Q**
0	0	1
0	1	1
1	0	0
1	1	Z (Memory state)

NAND LATCH

S	**R**	**Q**
0	0	Z (Memory State)
0	1	0
1	0	1
1	1	0

NOR LATCH

Figure 34: NAND and NOR Latch Circuits correction: swap Q output

In both latch circuits, we notice that three of the four input conditions force the Q output to a particular state (either 0 or 1), while the fourth input condition is passive, as it allows the circuit to hold its last forced value. This passive state is the memory state.