

CMSC 207- Lecture 7

CHAPTER 2: The Logic of Compound Statements (2.4 & 2.5)

Dr. Ahmed Tarek

The drawing in Figure 2.4.1(a) shows the appearance of the two positions of a simple switch. When the switch is closed, current can flow from one terminal to the other; when it is open, current cannot flow.

Imagine that such a switch is part of the circuit shown in figure 2.4.1(b). The light bulb turns on if, and only if, current flows through it. And this happens if, and only if, the switch is closed.

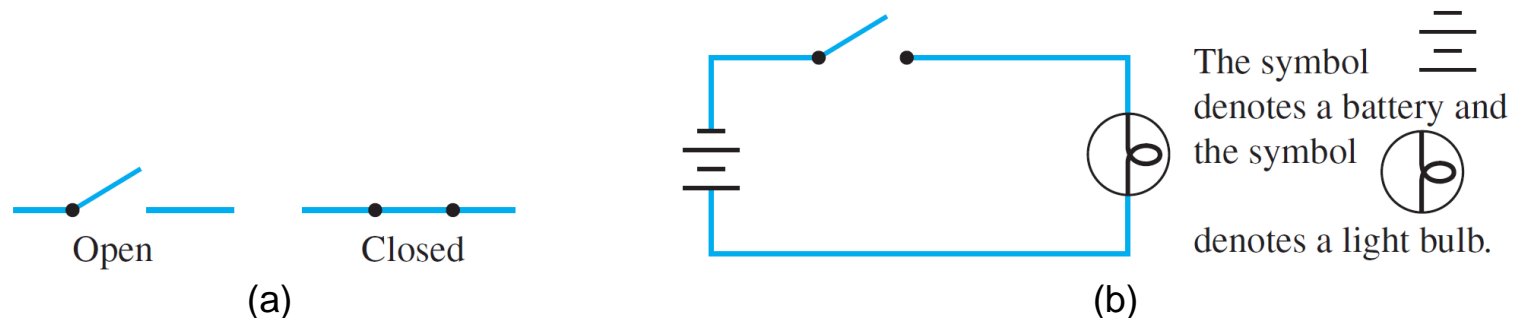
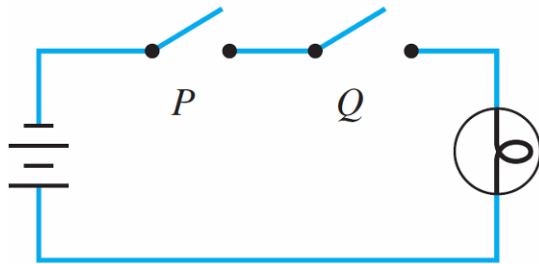
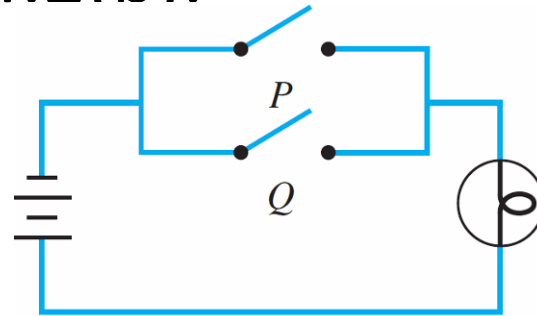


Figure 2.4.1

Now consider the more complicated circuits of Figures 2.4.2(a) and 2.4.2(b).



(a) Switches “in series”



(b) Switches “in parallel”

Figure 2.4.2

In the circuit of Figure 2.4.2(a) current flows and the light bulb turns on if, and only if, *both* switches *P* and *Q* are closed. The switches in this circuit are said to be **in series**.

In the circuit of Figure 2.4.2(b) current flows and the light bulb turns on if, and only if, *at least one* of the switches P or Q is closed. The switches in this circuit are said to be **in parallel**. All possible behaviors of these circuits are described by Table 2.4.1.

Switches		Light Bulb
P	Q	State
closed	closed	on
closed	open	off
open	closed	off
open	open	off

(a) Switches in Series

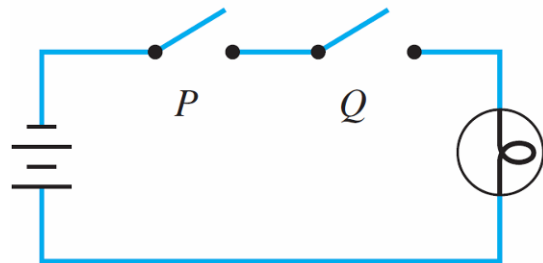
Switches		Light Bulb
P	Q	State
closed	closed	on
closed	open	on
open	closed	on
open	open	off

(b) Switches in Parallel

Table 2.4.1

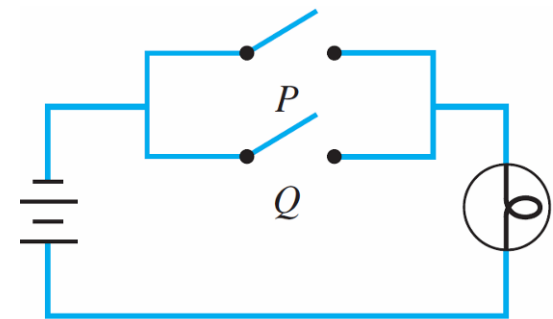
Observe that if the words *closed* and *on* are replaced by T and *open* and *off* are replaced by F, Table 2.4.1(a) becomes the truth table for *and* and Table 2.4.1(b) becomes the truth table for *or*.

Consequently, the switching circuit of Figure 2.4.2(a) is said to correspond to the logical expression $P \wedge Q$, and that of Figure 2.4.2(b) is said to correspond to $P \vee Q$.



(a) Switches “in series”

Figure 2.4.2



(b) Switches “in parallel”

Application: Digital Logic Circuits

More complicated circuits correspond to more complicated logical expressions. This correspondence has been used extensively in the design and study of circuits.

Electrical engineers continue to use the language of logic when they refer to values of signals produced by an electronic switch as being “true” or “false.” But they generally use the symbols 1 and 0 rather than T and F to denote these values. The symbols 0 and 1 are called **bits**, short for *binary digits*.

Black Boxes and Gates

Combinations of signal bits (1's and 0's) can be transformed into other combinations of signal bits (1's and 0's) by means of various circuits.

Because a variety of different technologies are used in circuit construction, computer engineers and digital system designers find it useful to think of certain basic circuits as black boxes.

Black Boxes and Gates

The inside of a black box contains the detailed implementation of the circuit and is often ignored while attention is focused on the relation between the **input** and the **output** signals.

The operation of a black box is completely specified by constructing an **input/output table** that lists all its possible input signals together with their corresponding output signals.



Black Boxes and Gates

For example, the black box picture has three input signals. Since each of these signals can take the value 1 or 0, there are eight possible combinations of input signals.

Black Boxes and Gates

One possible correspondence of input to output signals is as follows:

Input			Output
<i>P</i>	<i>Q</i>	<i>R</i>	<i>S</i>
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

An Input/Output Table

Black Boxes and Gates

An efficient method for designing more complicated circuits is to build them by connecting less complicated black box circuits. Three such circuits are known as NOT-, AND-, and OR-gates.

A **NOT-gate** (or **inverter**) is a circuit with one input signal and one output signal. If the input signal is 1, the output signal is 0.

Conversely, if the input signal is 0, then the output signal is 1. An **AND-gate** is a circuit with two input signals and one output signal. If both input signals are 1, then the output signal is 1.

Black Boxes and Gates

Otherwise, the output signal is 0. An **OR-gate** also has two input signals and one output signal. If both input signals are 0, then the output signal is 0. Otherwise, the output signal is 1.

The actions of NOT-, AND-, and OR-gates are summarized in Figure 2.4.3, where P and Q represent input signals and R represents the output signal.


Type of Gate	Symbolic Representation	Action								
NOT		<table><tr><th>Input</th><th>Output</th></tr><tr><th>P</th><th>R</th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	Input	Output	P	R	1	0	0	1
Input	Output									
P	R									
1	0									
0	1									

Figure 2.4.3

Black Boxes and Gates


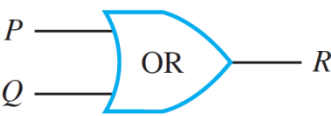
Type of Gate	Symbolic Representation	Action																		
AND		<table> <tr> <th colspan="2">Input</th><th>Output</th></tr> <tr> <th><i>P</i></th><th><i>Q</i></th><th><i>R</i></th></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> </table>	Input		Output	<i>P</i>	<i>Q</i>	<i>R</i>	1	1	1	1	0	0	0	1	0	0	0	0
Input		Output																		
<i>P</i>	<i>Q</i>	<i>R</i>																		
1	1	1																		
1	0	0																		
0	1	0																		
0	0	0																		
OR		<table> <tr> <th colspan="2">Input</th><th>Output</th></tr> <tr> <th><i>P</i></th><th><i>Q</i></th><th><i>R</i></th></tr> <tr> <td>1</td><td>1</td><td>1</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>0</td><td>0</td><td>0</td></tr> </table>	Input		Output	<i>P</i>	<i>Q</i>	<i>R</i>	1	1	1	1	0	1	0	1	1	0	0	0
Input		Output																		
<i>P</i>	<i>Q</i>	<i>R</i>																		
1	1	1																		
1	0	1																		
0	1	1																		
0	0	0																		

Figure 2.4.3 (continued)

Black Boxes and Gates

It should be clear from Figure 2.4.3 that the actions of the NOT-, AND-, and OR-gates on signals correspond exactly to those of the logical connectives \sim , \wedge , and \vee on statements, if the symbol 1 is identified with T and the symbol 0 is identified with F.

Gates can be combined into circuits in a variety of ways. If the rules shown on the next page are obeyed, the result is a **combinational circuit**, one whose output at any time is determined entirely by its input at that time without regard to previous inputs.

Rules for a Combinational Circuit

Never combine two input wires.

2.4.1

A single input wire can be split partway and used as input for two separate gates.

2.4.2

An output wire can be used as input.

2.4.3

No output of a gate can eventually feed back into that gate.

2.4.4

Rule (2.4.4) is violated in more complex circuits, called **sequential circuits**, whose output at any given time depends both on the input at that time and also on previous inputs.

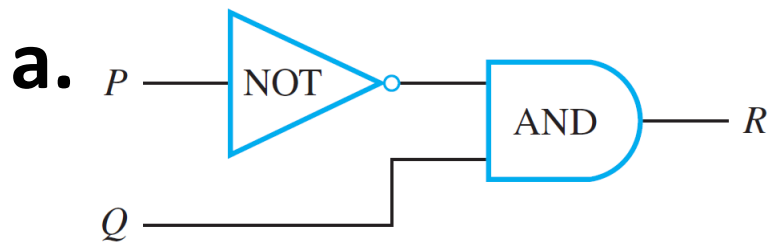
The Input/Output Table for a Circuit

If you are given a set of input signals for a circuit, you can find its output by tracing through the circuit gate by gate.

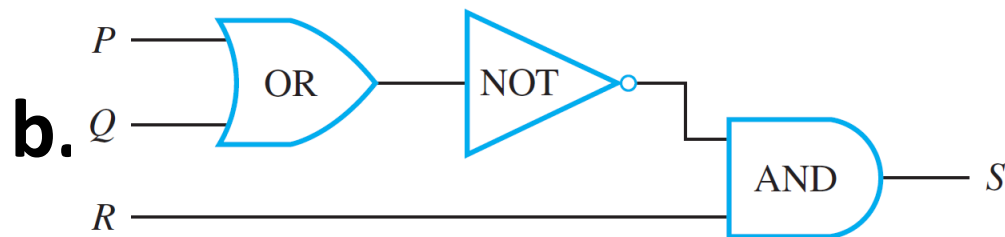
Example 1 – *Determining Output for a Given Input*

Indicate the output of the circuits shown below for the given input signals.

Input signals: $P = 0$ and $Q = 1$



Input signals: $P = 1$, $Q = 0$, $R = 1$

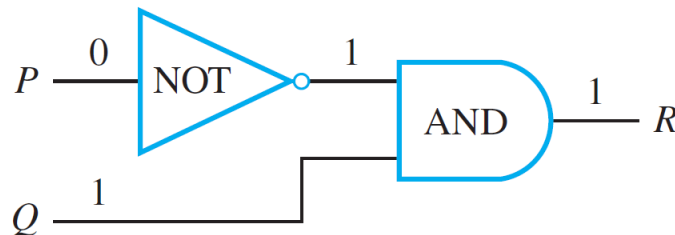


Example 1(a) – *Solution*

Move from left to right through the diagram, tracing the action of each gate on the input signals.

The NOT-gate changes $P = 0$ to a 1, so both inputs to the AND-gate are 1; hence the output R is 1.

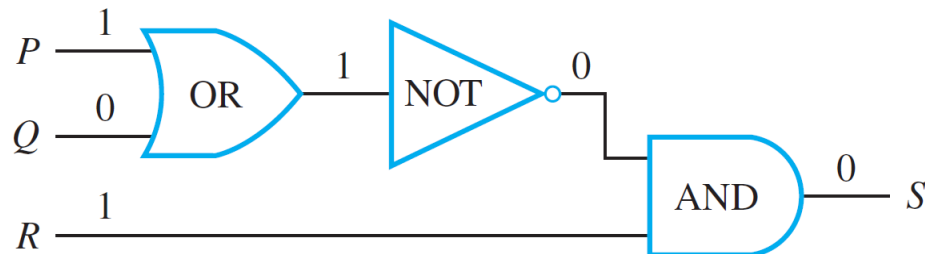
This is illustrated by annotating the diagram as shown below



Example 1(b) – *Solution*

The output of the OR-gate is 1 since one of the input signals, P , is 1. The NOT-gate changes this 1 into a 0, so the two inputs to the AND-gate are 0 and $R = 1$.

Hence the output S is 0. The trace is shown below.



The Boolean Expression Corresponding to a Circuit

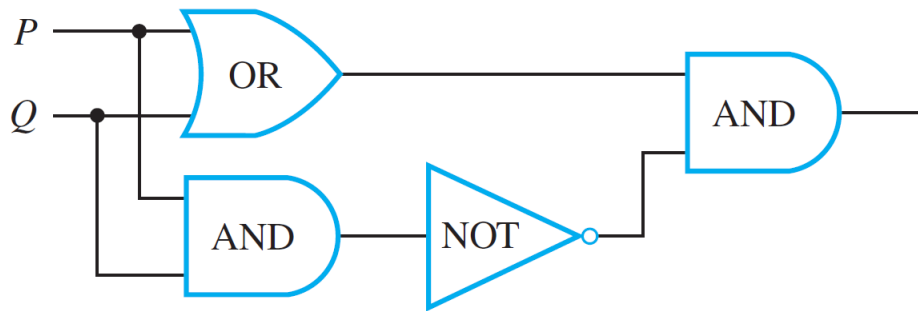
In logic, variables such as p , q and r represent statements, and a statement can have one of only two truth values: T(true) or F(false).

A statement form is an expression, such as $p \wedge (\sim q \vee r)$, composed of statement variables and logical connectives.

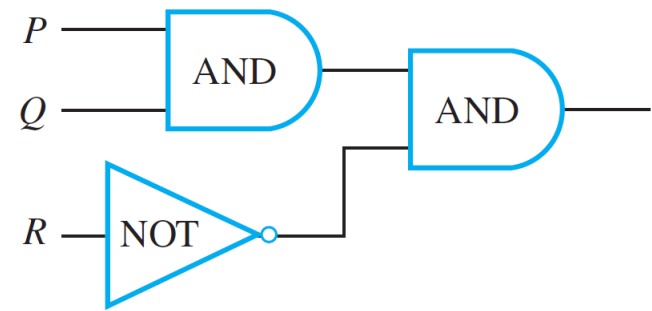
any variable, such as a statement variable or an input signal, that can take one of only two values is called a **Boolean variable**. An expression composed of Boolean variables and the connectives \sim , \wedge , and \vee is called a **Boolean expression**.

Example 3 – *Finding a Boolean Expression for a Circuit*

Find the Boolean expressions that correspond to the circuits shown below. A dot indicates a soldering of two wires; wires that cross without a dot are assumed not to touch.



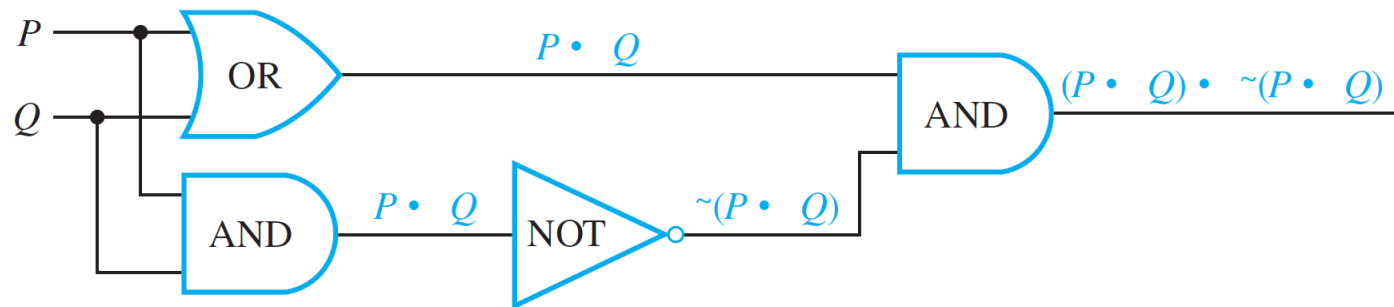
(a)



(b)

Example 3(a) – *Solution*

Trace through the circuit from left to right, indicating the output of each gate symbolically, as shown below.

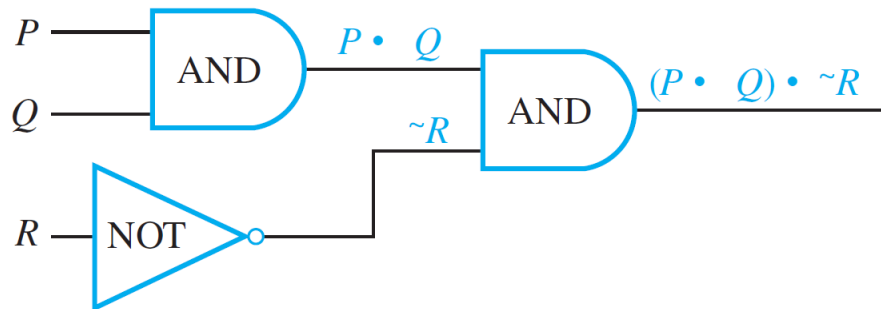


The final expression obtained, $(P \vee Q) \wedge \sim(P \wedge Q)$, is the expression for exclusive or: P or Q but not both.

Example 3(b) – *Solution*

The Boolean expression corresponding to the circuit is

$(P \wedge Q) \wedge \sim R$, as shown below.



Observe that the output of the circuit shown in Example 3(b) is 1 for exactly one combination of inputs ($P = 1$, $Q = 1$, and $R = 0$) and is 0 for all other combinations of inputs.

For this reason, the circuit can be said to “**recognize**” one particular combination of inputs. The output column of the input/output table has a 1 in exactly one row and 0’s in all other rows.

P	Q	R	$(P \wedge Q) \wedge \sim R$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Input/Output Table for a Recognizer

- **Definition**

A **recognizer** is a circuit that outputs a 1 for exactly one particular combination of input signals and outputs 0’s for all other combinations.

Example 4 – *Constructing Circuits for Boolean Expressions*

Construct circuits for the following Boolean expressions.

a. $(\sim P \wedge Q) \vee \sim Q$

b. $((P \wedge Q) \wedge (R \wedge S)) \wedge T$

b. **Solution:**

- c. Write the input variables in a column on the left and then go from the right side of the diagram to the left, working from the outermost part of the expression to the innermost part. Since the last operation executed when evaluating $(\sim P \wedge Q) \vee \sim Q$ is \vee , put an OR-gate at the extreme right of the diagram.

Example 4 – *Solution*

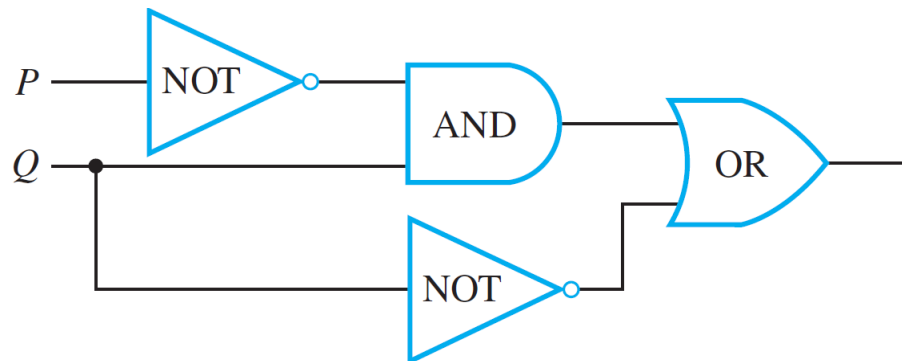
One input to this gate is $\sim P \wedge Q$, so draw an AND-gate to the left of the OR-gate and show its output coming into the OR-gate.

Since one input to the AND-gate is $\sim P$, draw a line from P to a NOT-gate and from there to the AND-gate. Since the other input to the AND-gate is Q , draw a line from Q directly to the AND-gate.

In-class Assignment 1

Example 4 – *Solution*

The other input to the OR-gate is $\sim Q$, so draw a line from Q to a NOT-gate and from the NOT-gate to the OR-gate. The circuit you obtain is shown below.



Example 4 – *Solution*

b. To start constructing this circuit, put one AND-gate at the extreme right for the \wedge between $((P \wedge Q) \wedge (R \wedge S))$ and T .

To the left of that put the AND-gate corresponding to the \wedge between $P \wedge Q$ and $R \wedge S$.

To the left of that put the AND-gates corresponding to the \wedge 's between P and Q and between R and S .

Example 4 – *Solution*

The circuit is shown in Figure 2.4.4.

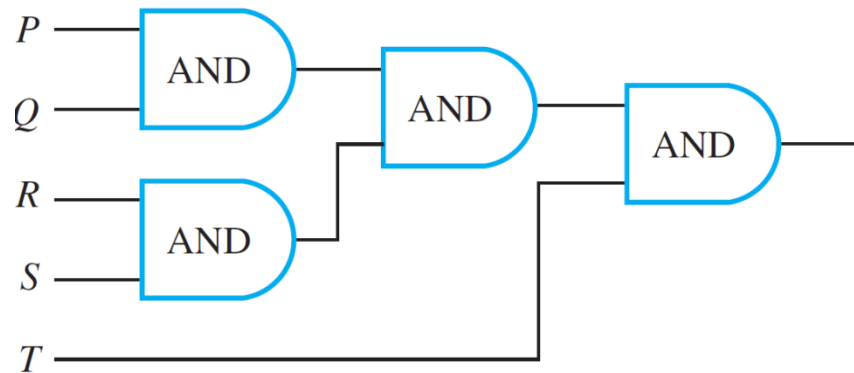


Figure 2.4.4

The Circuit Corresponding to a Boolean Expression

Thus, for example,

$$((P \wedge Q) \wedge (R \wedge S)) \wedge T \equiv (P \wedge (Q \wedge R)) \wedge (S \wedge T).$$

It also follows that the circuit in Figure 2.4.5, which corresponds to $(P \wedge (Q \wedge R)) \wedge (S \wedge T)$, has the same input/output table as the circuit in Figure 2.4.4, which corresponds to $((P \wedge Q) \wedge (R \wedge S)) \wedge T$.

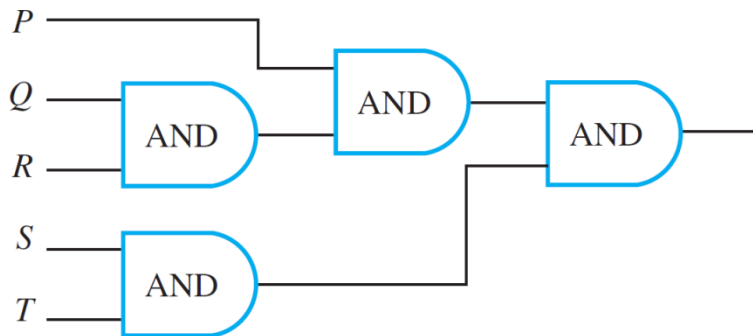


Figure 2.4.5

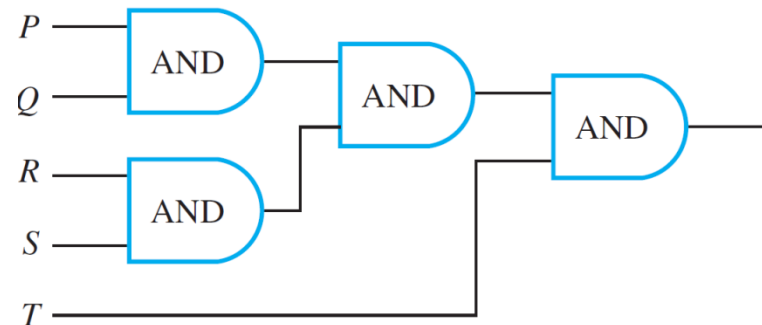


Figure 2.4.4

Each of the circuits in Figures 2.4.4 and 2.4.5 is, therefore, an implementation of the expression $P \wedge Q \wedge R \wedge S \wedge T$.

Such a circuit is called a **multiple-input AND-gate** and is represented by the diagram shown in Figure 2.4.6.

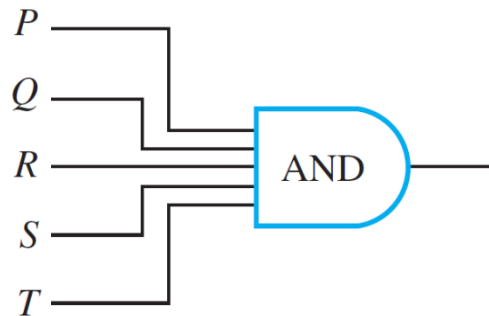


Figure 2.4.6

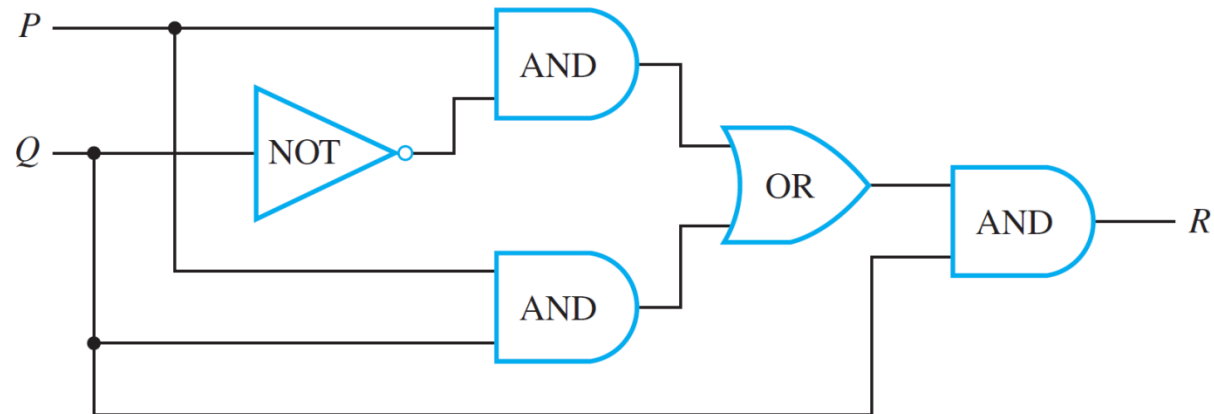
Simplifying Combinational Circuits

- **Definition**

Two digital logic circuits are **equivalent** if, and only if, their input/output tables are identical.

Example 6 – Showing That Two Circuits Are Equivalent

Find the Boolean expressions for each circuit in Figure 2.4.8. Show that these expressions are logically equivalent when regarded as statement forms.



(a)



(b)

Example 6 – *Solution*

The Boolean expressions that correspond to circuits (a) and (b) are $((P \wedge \sim Q) \vee (P \wedge Q)) \wedge Q$ and $P \wedge Q$, respectively.

By Theorem 2.1.1,

$$((P \wedge \sim Q) \vee (P \wedge Q)) \wedge Q$$

$$\equiv (P \wedge (\sim Q \vee Q)) \wedge Q \quad \text{by the distributive law}$$

$$\equiv (P \wedge (Q \vee \sim Q)) \wedge Q \quad \text{by the commutative law for } \vee$$

Example 6 – *Solution*

$$\equiv (P \wedge \mathbf{t}) \wedge Q \quad \text{by the negation law}$$

$$\equiv P \wedge Q \quad \text{by the identity law.}$$

It follows that the truth tables for $((P \wedge \sim Q) \vee (P \wedge Q)) \wedge Q$ and $P \wedge Q$ are the same.

Hence the input/output tables for the circuits corresponding to these expressions are also the same, and so the circuits are equivalent.

NAND and NOR Gates

Another way to simplify a circuit is to find an equivalent circuit that uses the least number of different kinds of logic gates.

Two gates not previously introduced are particularly useful for this: NAND-gates and NOR-gates. A NAND-gate is a single gate that acts like an AND-gate followed by a NOT-gate. A NOR-gate acts like an OR-gate followed by a NOT-gate.

NAND and NOR Gates



Thus the output signal of a NAND-gate is 0 when, and only when, both input signals are 1, and the output signal for a NOR-gate is 1 when, and only when, both input signals are 0.

The logical symbols corresponding to these gates are $|$ (for NAND) and \downarrow (for NOR), where $|$ is called a **Sheffer stroke** (after H. M. Sheffer, 1882–1964) and \downarrow is called a **Peirce arrow** (after C. S. Peirce, 1839–1914). Thus

$$P | Q \equiv \sim(P \wedge Q) \quad \text{and} \quad P \downarrow Q \equiv \sim(P \vee Q).$$

NAND and NOR Gates

The table below summarizes the actions of NAND and NOR gates.

Type of Gate	Symbolic Representation	Action																		
NAND		<table> <tr> <th colspan="2">Input</th><th>Output</th></tr> <tr> <th>P</th><th>Q</th><th>$R = P \mid Q$</th></tr> <tr> <td>1</td><td>1</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>0</td><td>0</td><td>1</td></tr> </table>	Input		Output	P	Q	$R = P \mid Q$	1	1	0	1	0	1	0	1	1	0	0	1
Input		Output																		
P	Q	$R = P \mid Q$																		
1	1	0																		
1	0	1																		
0	1	1																		
0	0	1																		
NOR		<table> <tr> <th colspan="2">Input</th><th>Output</th></tr> <tr> <th>P</th><th>Q</th><th>$R = P \downarrow Q$</th></tr> <tr> <td>1</td><td>1</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>0</td><td>0</td><td>1</td></tr> </table>	Input		Output	P	Q	$R = P \downarrow Q$	1	1	0	1	0	0	0	1	0	0	0	1
Input		Output																		
P	Q	$R = P \downarrow Q$																		
1	1	0																		
1	0	0																		
0	1	0																		
0	0	1																		

NAND and NOR Gates

It can be shown that any Boolean expression is equivalent to one written entirely with Sheffer strokes or entirely with Peirce arrows.

Thus any digital logic circuit is equivalent to one that uses only NAND-gates or only NOR-gates.

Example 7 – Rewriting Expressions Using the Sheffer Stroke

Use Theorem 2.1.1 and the definition of Sheffer stroke to show that

$$\text{a. } \sim P \equiv P | P \quad \text{and} \quad \text{b. } P \vee Q \equiv (P | P) | (Q | Q).$$

Solution:

$$\begin{aligned} \text{a. } \sim P &\equiv \sim(P \wedge P) && \text{by the idempotent law for } \wedge \\ &\equiv P | P && \text{by definition of } |. \end{aligned}$$

$$\begin{aligned} \text{b. } P \vee Q &\equiv \sim(\sim(P \vee Q)) && \text{by the double negative law} \\ &\equiv \sim(\sim P \wedge \sim Q) && \text{by De Morgan's laws} \end{aligned}$$

Example 7 – *Solution*

$$\equiv \sim((P \mid P) \wedge (Q \mid Q)) \quad \text{by part (a)}$$

$$\equiv (P \mid P) \mid (Q \mid Q) \quad \text{by definition of } \mid.$$

Binary Representation of Numbers

- $27 = 16 + 8 + 2 + 1$
- $= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
- $= 11011 \text{ in base } 2$
- The places in binary notation correspond to the various powers of 2.

Binary Representation of Numbers

- The right-most position is the ones place (or 2^0 place), to the left of that is the twos place (or 2^1 place), to the left of that is the fours place (or 2^2 place), and so forth, as illustrated below.

Place	2^4 sixteens	2^3 eights	2^2 fours	2^1 twos	2^0 ones
Binary Digit	1	1	0	1	1

- As in the decimal notation, leading zeros may be added or dropped as desired. For example,

$$003_{10} = 3_{10} = 1 \cdot 2^1 + 1 \cdot 2^0 = 11_2 = 011_2.$$

•In-class Assignment 2

Example 2 – *Converting a Binary to a Decimal Number*

- Represent 110101_2 in decimal notation.

- **Solution:**

$$110101_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$= 32 + 16 + 4 + 1$$

$$= 53_{10}$$

- **In-class Assignment 3**

Example 4 – *Addition in Binary Notation*

- Add 1101_2 and 111_2 using binary notation.

- **Solution:**

- Because $2_{10} = 10_2$ and $1_{10} = 1_2$, the translation of

$1_{10} + 1_{10} = 2_{10}$ to binary notation is
$$\begin{array}{r} 1_2 \\ + 1_2 \\ \hline 10_2 \end{array}$$

Example 4 – *Solution*

- Adding three 1's together also results in a carry of 1 since $3_{10} = 11_2$ (“one one base two”).

$$\begin{array}{r} 1_2 \\ + 1_2 \\ + 1_2 \\ \hline 11_2 \end{array}$$

- Thus the addition can be performed as follows:

$$\begin{array}{rcccc} & 1 & 1 & 1 & \leftarrow \text{carry row} \\ & 1 & 1 & 0 & 1_2 \\ + & & 1 & 1 & 1_2 \\ \hline 1 & 0 & 1 & 0 & 0_2 \end{array}$$

Example 5 – *Solution*

- In binary subtraction it may also be necessary to borrow across more than one column. But when you borrow a 1_2 from 10_2 , what remains is 1_2 .

$$\begin{array}{r} 10_2 \\ - 1_2 \\ \hline 1_2 \end{array}$$

- Thus the subtraction can be performed as follows:

$$\begin{array}{rcccccc} & & 0 & 1 & 1 & & \\ & & \swarrow & \swarrow & & & \\ & 1 & 1 & 0 & 0 & 0_2 & \\ - & & 1 & 0 & 1 & 1_2 & \\ \hline & & 1 & 1 & 0 & 1_2 & \end{array} \quad \leftarrow \text{borrow row}$$

Two's Complements and the Computer Representation of Negative Integers

- A more common approach, using *two's complements*, makes it possible to add integers quite easily and results in a unique representation for 0. The two's complement of an integer relative to a fixed bit length is defined as follows:

- **Definition**

Given a positive integer a , the **two's complement of a relative to a fixed bit length n** is the n -bit binary representation of

$$2^n - a.$$

- There is a convenient way to compute two's complements that involves less arithmetic than direct application of the definition. For an 8-bit representation, it is based on three facts:

- **1.** $2^8 - a = [(2^8 - 1) - a] + 1$.
- **2.** The binary representation of $2^8 - 1$ is 11111111_2 .
- **3.** Subtracting an 8-bit binary number a from 11111111_2 just switches all the 0's in a to 1's and all the 1's to 0's. (The resulting number is called the **one's complement** of the given number.)

Two's Complements and the Computer Representation of Negative Integers

- In general,

To find the 8-bit two's complement of a positive integer a that is at most 255:

- Write the 8-bit binary representation for a .
- Flip the bits (that is, switch all the 1's to 0's and all the 0's to 1's).
- Add 1 in binary notation.

Example 6 – *Finding a Two's Complement*

- Find the 8-bit two's complement of 19.

- **Solution:**

- Write the 8-bit binary representation for 19,
{19₁₀ = (16 + 2 + 1)₁₀ to 1's and all the 1's to 0's, and
add 1.

$$= 00010011_2 \xrightarrow{\text{flip the bits}} 11101100 \xrightarrow{\text{add 1}} 11101101$$

Example 6 – *Solution*

- To check this result, note that

$$11101101_2 = (128 + 64 + 32 + 8 + 4 + 1)_{10}$$

$$= 237_{10}$$

$$= (256 - 19)_{10}$$

$$= (2^8 - 19)_{10},$$

- which *is* the two's complement of 19.

Two's Complements and the Computer Representation of Negative Integers

- Observe that because

$$2^8 - (2^8 - a) = a$$

- the *two's complement of the two's complement of a number is the number itself*, and therefore,

To find the decimal representation of the integer with a given 8-bit two's complement:

- Find the two's complement of the given two's complement.
- Write the decimal equivalent of the result.

Example 7 – Finding a Number with a Given Two's Complement

- What is the decimal representation for the integer with two's complement 10101001?

$$10101001_2 \xrightarrow{\text{flip the bits}} 01010110$$

$$\bullet \quad \xrightarrow{\text{add 1}} 01010111_2 = (64 + 16 + 4 + 2 + 1)_{10} \\ = 87_{10}$$

Example 7 – *Solution*

- To check this result, note that the given number is

$$10101001_2 = (128 + 32 + 8 + 1)_{10}$$

$$= 169_{10}$$

$$= (256 - 87)_{10}$$

$$= (2^8 - 87)_{10},$$

8-Bit Representation of a Number

- It follows that the 8-bit two's complement of an integer from 1 through 128 has a leading bit of 1. Note also that the ordinary 8-bit representation of an integer from 0 through 127 has a leading bit of 0.
- Consequently, eight bits can be used to represent both nonnegative and negative integers by representing each nonnegative integer up through 127 using ordinary 8-bit binary notation and representing each negative integer from -1 through -128 as the two's complement of its absolute value.

8-Bit Representation of a Number

- That is, for any integer a from -128 through 127 ,

The 8-bit representation of a

$$= \begin{cases} \text{the 8-bit binary representation of } a & \text{if } a \geq 0 \\ \text{the 8-bit binary representation of } 2^8 - |a| & \text{if } a < 0 \end{cases}.$$

Hexadecimal Notation

- **Hexadecimal notation** is even more compact than decimal notation, and it is much easier to convert back and forth between hexadecimal and binary notation than it is between binary and decimal notation.
- The word *hexadecimal* comes from the Greek root *hex-*, meaning “six,” and the Latin root *deci-*, meaning “ten.” Hence *hexadecimal* refers to “sixteen,” and hexadecimal notation is also called **base 16 notation**.

Hexadecimal Notation

- The sixteen hexadecimal digits are shown in Table 2.5.3, together with their decimal equivalents and, for future reference, their 4-bit binary equivalents.

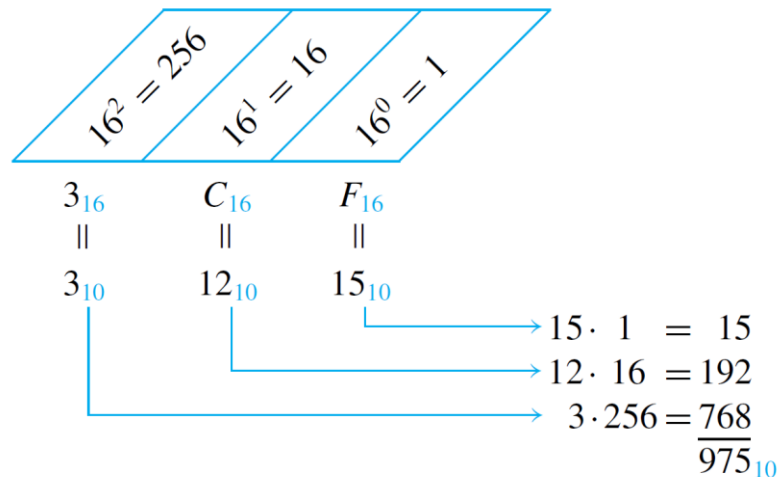
Decimal	Hexadecimal	4-Bit Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Decimal	Hexadecimal	4-Bit Binary Equivalent
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Table 2.5.3

Example 11 – *Converting from Hexadecimal to Decimal Notation*

- Convert $3CF_{16}$ to decimal notation.
- **Solution:**
- Consider the following schema.



Hexadecimal Notation

- Now consider how to convert from hexadecimal to binary notation.
- The following sequence of steps will give the required conversion from hexadecimal to binary notation.

To convert an integer from hexadecimal to binary notation:

- Write each hexadecimal digit of the integer in 4-bit binary notation.
- Juxtapose the results.

Example 12 – *Converting from Hexadecimal to Binary Notation*

- Convert $B09F_{16}$ to binary notation.

- **Solution:**

- $B_{16} = 11_{10} = 1011_2,$





$$0_{16} = 0_{10} = 0000_2,$$

- $9_{16} = 9_{10} = 1001_2,$

- and $F_{16} = 15_{10} = 1111_2.$

Example 12 – *Solution*

- Consequently,

B	0	9	F
			
1011	0000	1001	1111

- and the answer is 1011000010011111₂.

Hexadecimal Notation

- To convert integers written in binary notation into hexadecimal notation, reverse the steps of the previous procedure.

To convert an integer from binary to hexadecimal notation:

- Group the digits of the binary number into sets of four, starting from the right and adding leading zeros as needed.
- Convert the binary numbers in each set of four into hexadecimal digits. Juxtapose those hexadecimal digits.





Example 13 – *Converting from Binary to Hexadecimal Notation*

- Convert 100110110101001_2 to hexadecimal notation.
- **Solution:**
- First group the binary digits in sets of four, working from right to left and adding leading 0's if necessary.

0100 1101 1010 1001.

Example 13 – *Solution*

- Convert each group of four binary digits into a hexadecimal digit.

0100	1101	1010	1001
			
4	D	A	9

- Then juxtapose the hexadecimal digits.

4DA9₁₆

In-class Assignment 4