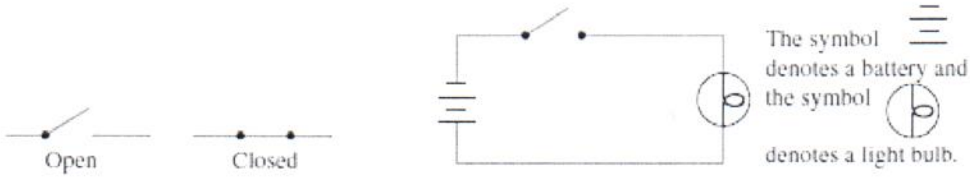
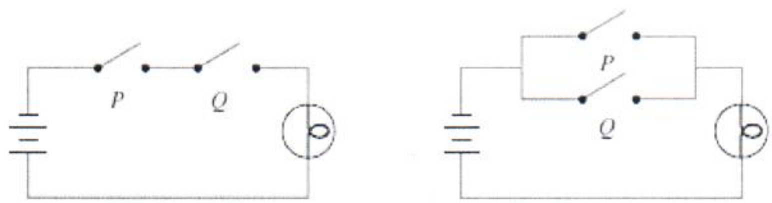




Switches "in series"
(a)

Switches "in parallel"
(b)

(a) Switches in Series

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

"and"
connective

(b) Switches in Parallel

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

"or"
connective

Find the state (on/off) of the light bulb in each situation. How do these tables compare to some logical connectives that we have learned?

switches in series are like "and" connectives

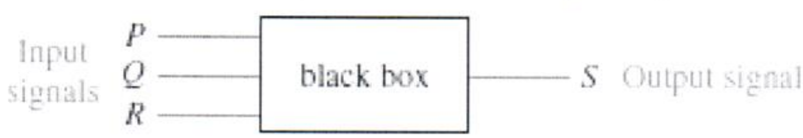
closed/on : T / open/off : F

switches in parallel are like "or" connectives

In the setting of circuit design, it is customary to use 1 and 0 instead of T and F.

T → 1
F → 0

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 of its possible input signals together with their corresponding output signals.

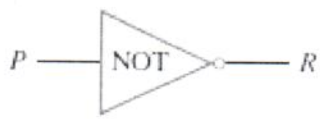
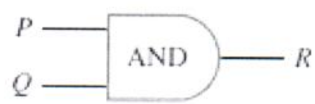
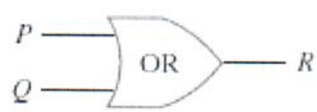
It may be something like this:

ex

INPUT			OUTPUT
P	Q	R	S
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

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.

<table>
<tr><th>Type of Gate</th><th>Symbolic Representation</th><th>Action</th></tr>
<tr><td>NOT</td><td>
</td><td>
<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>
</td></tr>
<tr><td>AND</td><td>
</td><td>
<table>
<tr><th colspan="2">Input</th><th>Output</th></tr>
<tr><th>P</th><th>Q</th><th>R</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>
</td></tr>
<tr><td>OR</td><td>
</td><td>
<table>
<tr><th colspan="2">Input</th><th>Output</th></tr>
<tr><th>P</th><th>Q</th><th>R</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>
</td></tr>
</table>

Gates can be combined into circuits in a variety of ways. If the following rules are obeyed, the result is a **combinational circuit**.

(A combinational circuit has the property that its output at any time is determined entirely by its input at that time without regard to previous inputs.)

R1: Never combine two input wires.

R2: A single input wire can be split partway and used as input for two separate gates.

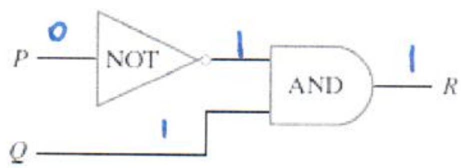
R3: An output wire can be used as input.

R4: No output of a gate can eventually feed back into that gate.

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: Indicate the output of the circuits for the given input signals.

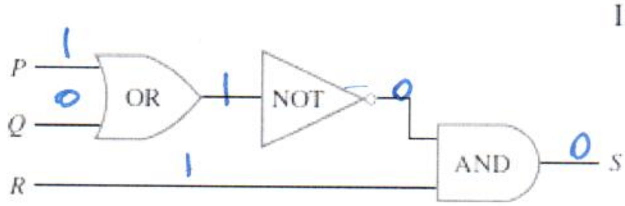
a.



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

$R = 1$

b.

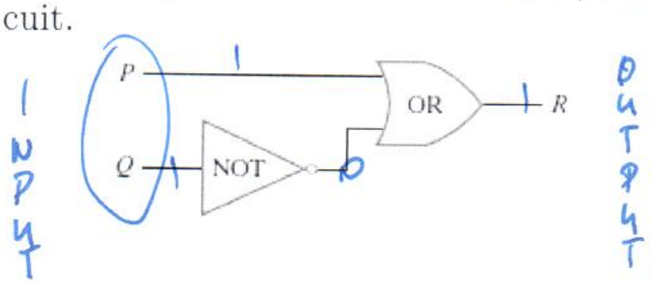


Input signals: $P = 1$, $Q = 0$, $R = 1$

$S = 0$

If we are not given any specific input signals, we might want to construct the entire input/output table of the circuit.

Example: Construct the input/output table for the following circuit.



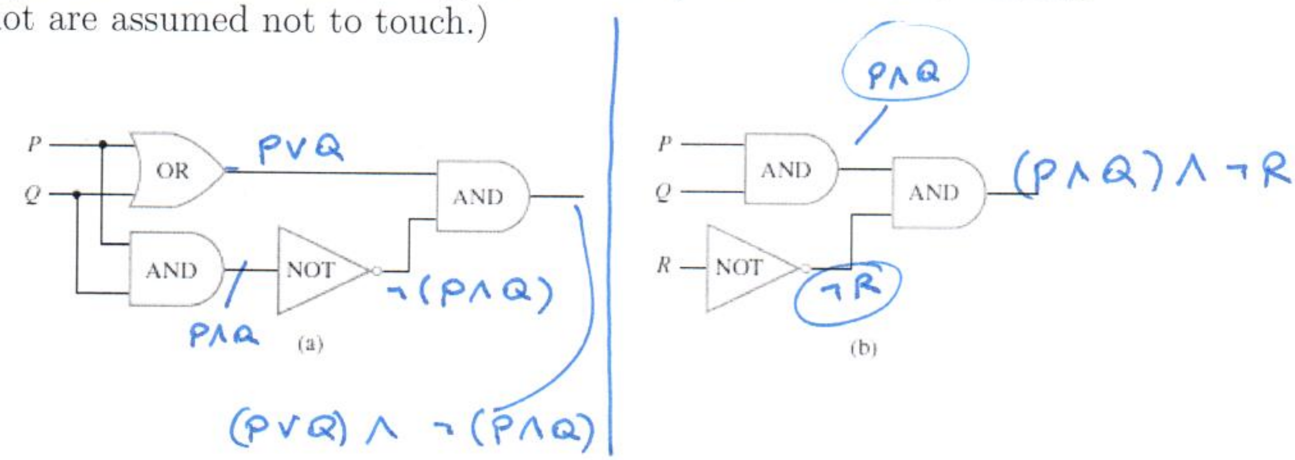
INPUT		OUTPUT
P	Q	R
1	1	1
1	0	1
0	1	0
0	0	1

An expression composed of variables (such as P, Q, etc.) and the connectives $\neg, \wedge, \vee$ is called a **Boolean expression**.

Given a circuit consisting of combined **NOT**-, **AND**-, and **OR**-gates, a corresponding Boolean expression can be obtained by tracing the actions of the gates on the input variables.

Example: 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.)



Observe that the output of the circuit shown in Example (b) is 1 for exactly one combination of inputs.

$$(P \wedge Q) \wedge \neg R \equiv \underbrace{P \wedge Q \wedge \neg R}$$

This has value 1 only if
$P = 1$, $Q = 1$ & $R = 0$

The circuit can be said to "recognize" one particular combination of inputs. It is called a recognizer circuit.

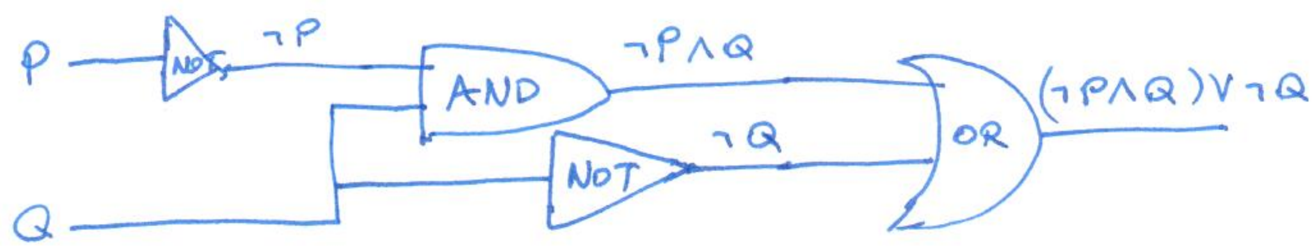
The problem of constructing circuits corresponding to given Boolean expressions is just as interesting.

Example: Construct circuits for the following Boolean expressions.

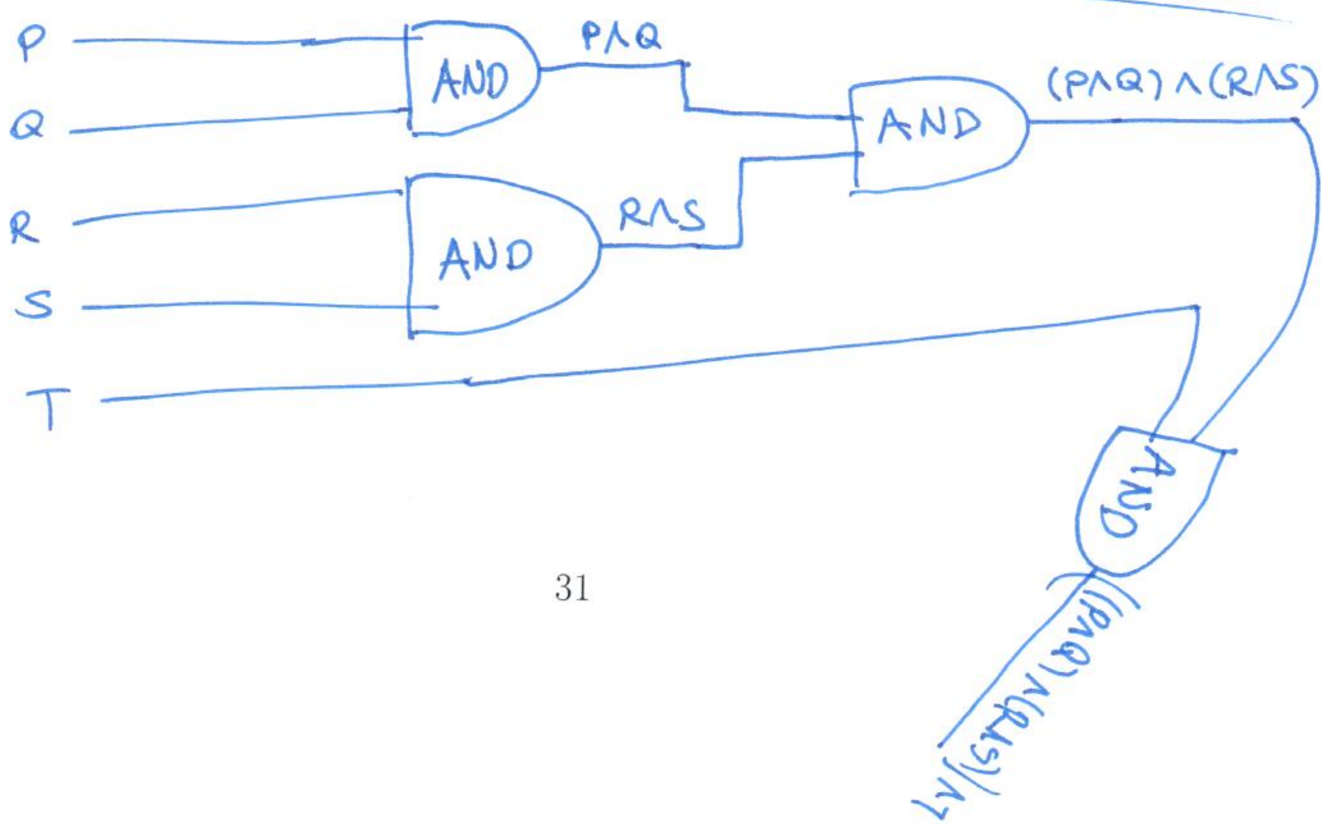
(i) $(\neg P \wedge Q) \vee \neg Q$

(ii) $((P \wedge Q) \wedge (R \wedge S)) \wedge T$

i)



ii)



Now we address the question of how to design a circuit (or find a Boolean expression) corresponding to a given input/output table.

The way to do this is to put several recognizers together in parallel. This brings us to the notion of the disjunctive normal form of a proposition.

A **disjunctive normal form (DNF)** of a proposition is a disjunction of conjunctions which may involve negations of the variables.

Examples:

$(P \wedge Q) \vee (R \wedge S) \vee (P \wedge \neg S)$

$(\neg S) \vee (S \wedge \neg P)$

Given the truth table of a proposition/expression, to find its DNF:

1. Identify each row for which the output is 1.

2. For each such row, construct an $\wedge$-expression that produces a 1 (or true) for the exact combination of input values for that row, and a 0 (or false) for all other combinations of input values.

3. Bring together the $\wedge$-expressions of the previous step with $\vee$'s.

Example: Design a circuit for the following input/output table.

INPUT		OUTPUT
P	Q	R
1	1	1
1	0	0
0	1	1
0	0	0

First find an expression in **DNF** with this table as its truth table.

DNF :

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

To create the required circuit for the input/output table we would construct a recognizer circuit for each expression corresponding to output 1, and then we'd connect them through or gates.