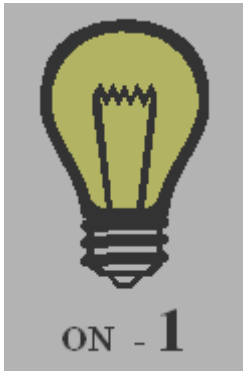
Chapter 4

Fundamentals of Logic

4.1

The Binary Concept

Binary **refers to the idea that many things can be thought of as existing in only one of two states.**



The binary states are 1 and 0

The 1 and 0 can represent:

ON or OFF

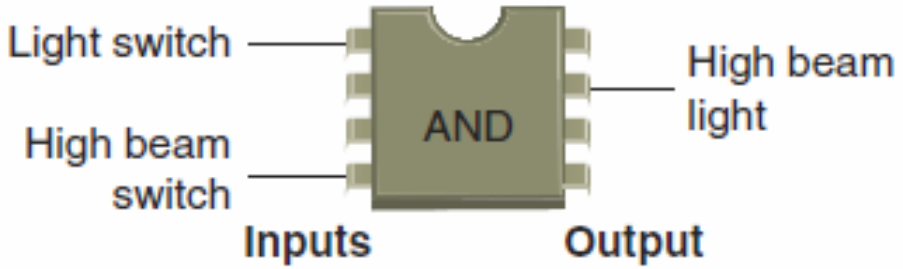
Open or Closed

True or False

High or Low

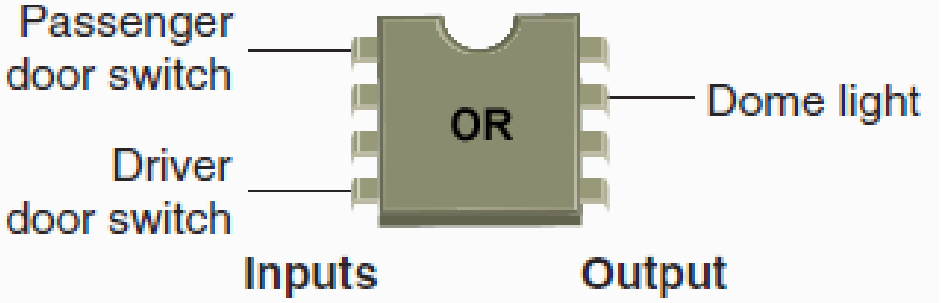
A *logic gate* is a circuit with several inputs but only one output that is activated by particular combinations of input conditions.

Logical AND



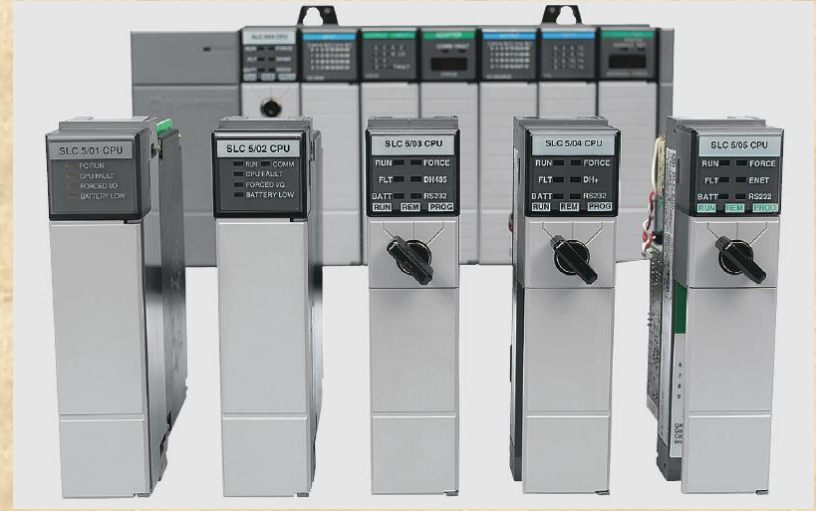
The high beam light can be turned on only when the light switch AND the high beam switch are closed.

Logical OR



The dome light will be turned on whenever the passenger door switch OR the driver door switch is activated.

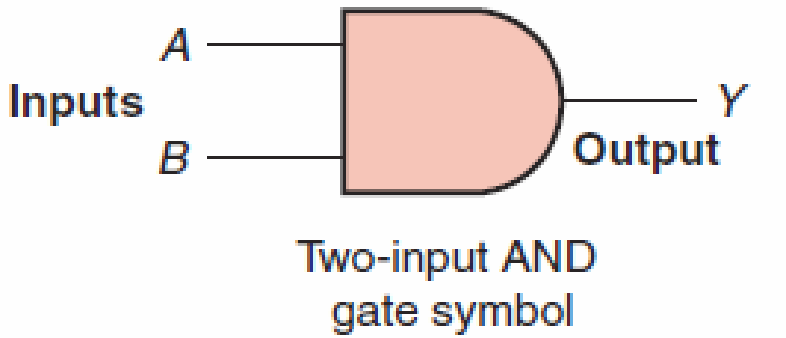
4.2



AND, OR, and NOT Functions

The *AND* Function

An AND gate is a device with two or more inputs and one output.



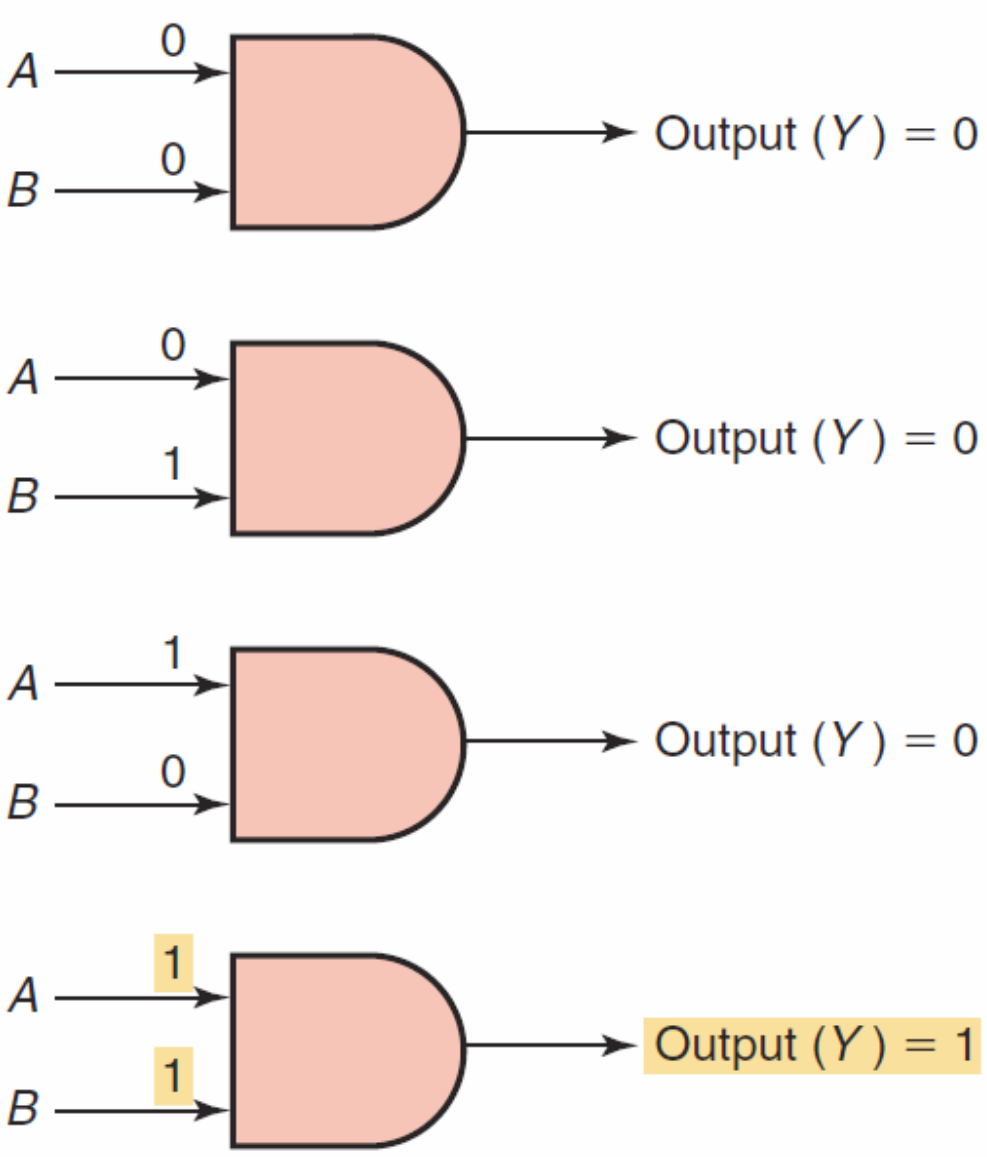
Two-input AND gate symbol

AND truth table

Inputs		Output
A	*B*	*Y*
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate output is 1 only if all inputs are 1.

The *AND* Function



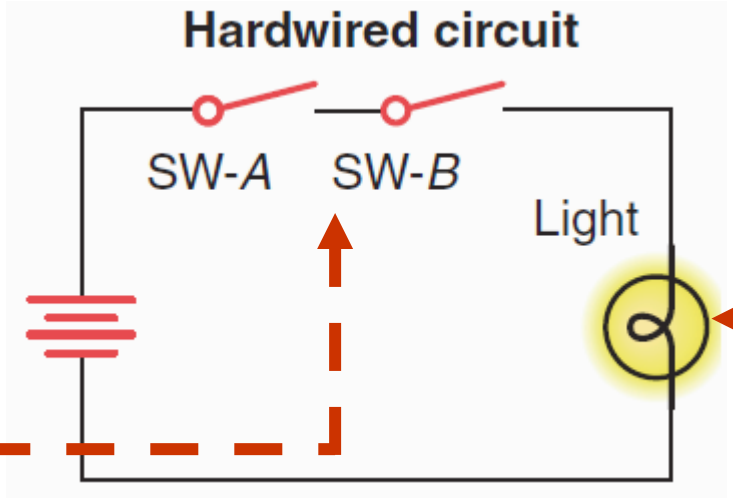
Truth table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

- **If all inputs are 1, the output will be 1.**
- **If any input is 0, the output will be 0.**

The *AND* Function

The AND logic gate operates similarly to control devices connected in series

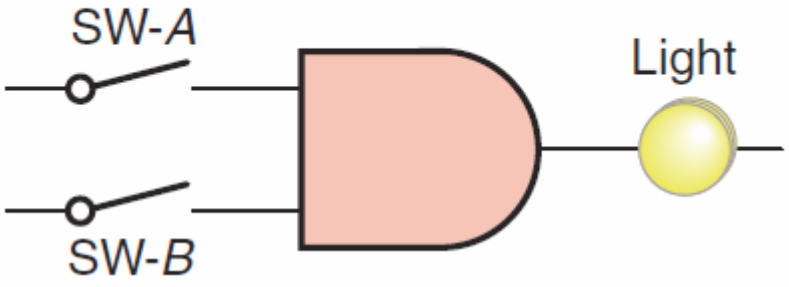


The light will be on only when both switch A and switch B are closed.

Truth table

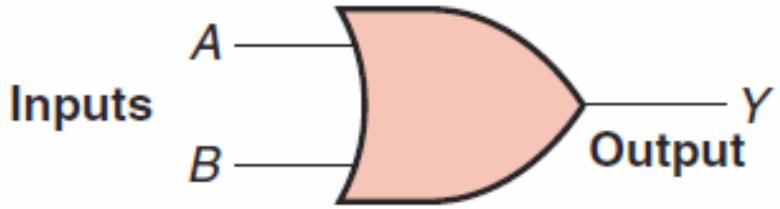
SW-*A*		SW-*B*		Light	
Open	(0)	Open	(0)	Off	(0)
Open	(0)	Closed	(1)	Off	(0)
Closed	(1)	Open	(0)	Off	(0)
Closed	(1)	Closed	(1)	On	(1)

Logic representation



The *OR* Function

An OR gate can have any number of inputs but only one output.



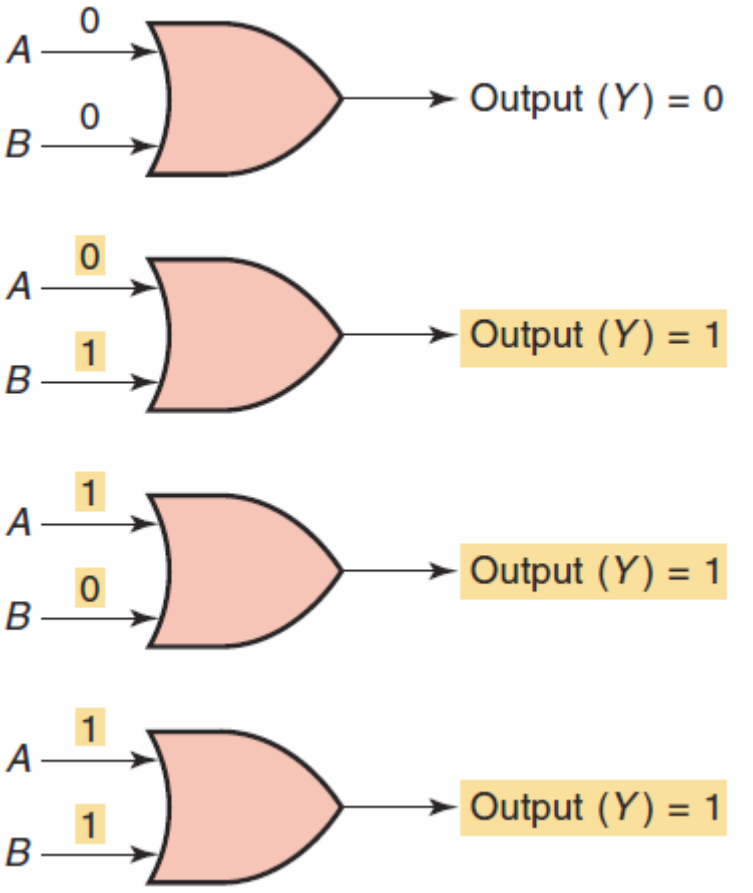
Two-input OR gate symbol

OR truth table

Inputs		Output
A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate output is 1 if one or more inputs are 1.

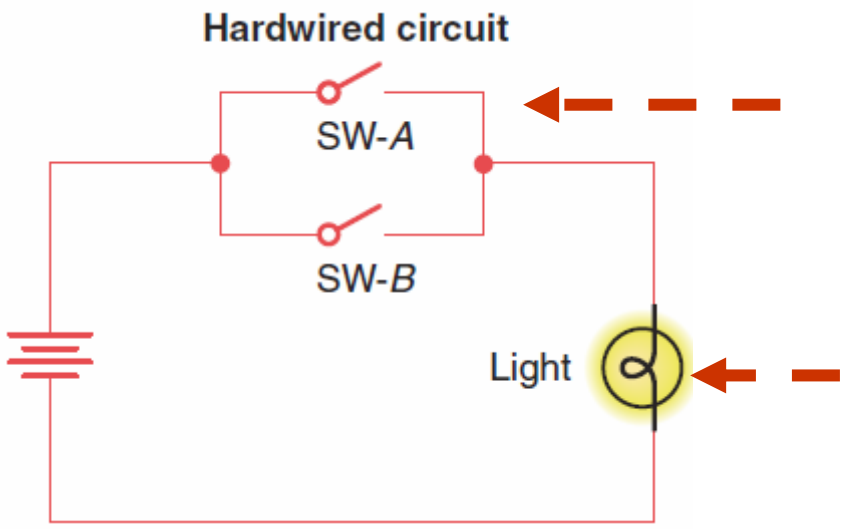
The *OR* Function



Truth table		
Inputs		Output
A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	1

- **If one or more inputs are 1, the output is 1.**
- **If all inputs are 0, the output will be 0.**

The *OR* Function



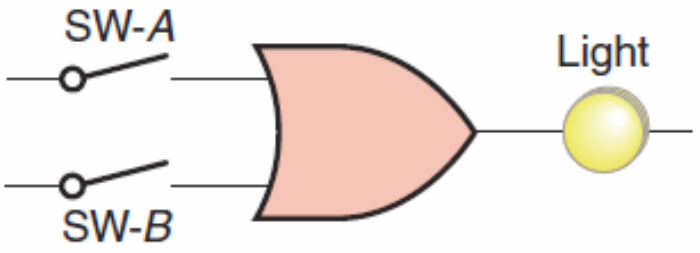
The OR logic gate operates similarly to control devices connected in parallel.

The light will be on if switch A or switch B or both are closed.

Truth table

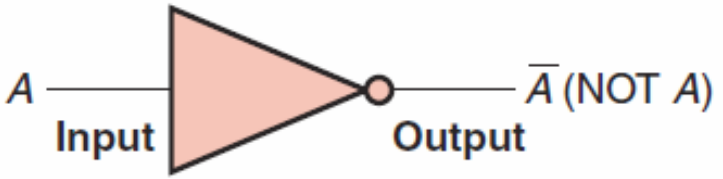
SW-*A*		SW-*B*		Light	
Open	(0)	Open	(0)	Off	(0)
Open	(0)	Closed	(1)	On	(1)
Closed	(1)	Open	(0)	On	(1)
Closed	(1)	Closed	(1)	On	(1)

Logic representation



The *NOT* Function

The NOT function can have only one input.



NOT truth table

A	NOT *A*
0	1
1	0

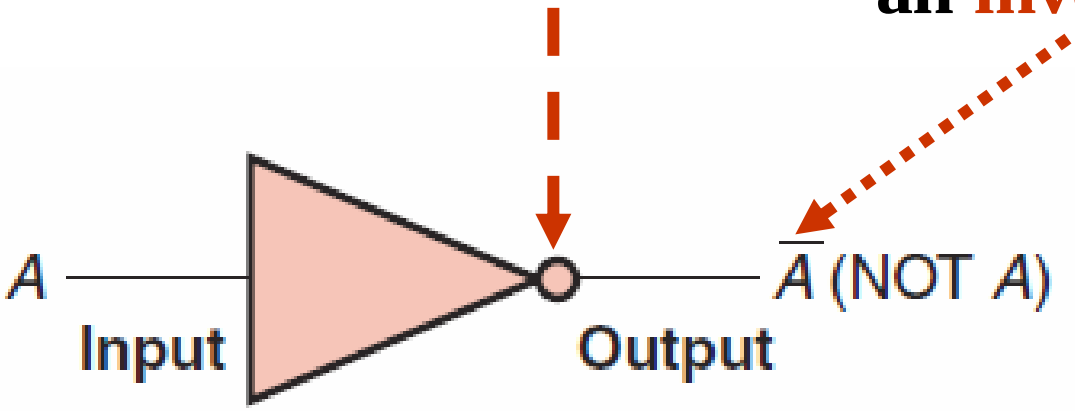
The NOT output is 1 if the input is 0.

The output is 0 if the input is 1.

The *NOT* Function

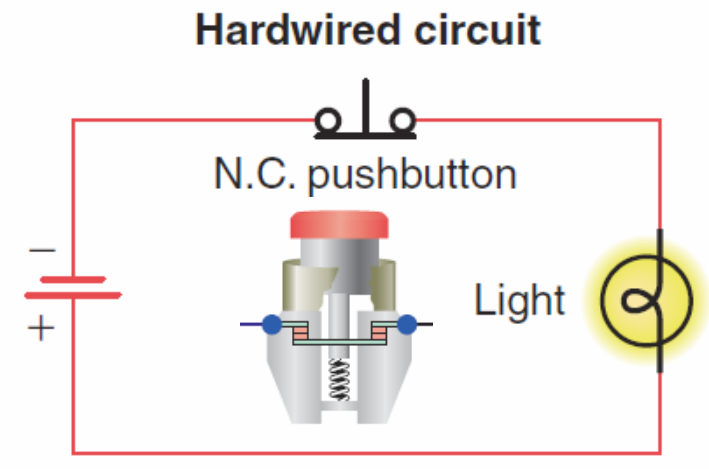
The circle indicates that an inversion of the logical function has taken place.

The bar across the top of the letter, indicates an inverted output.



The result of the NOT operation is always the inverse of the input, and the NOT function is, therefore, called an inverter.

The *NOT* Function

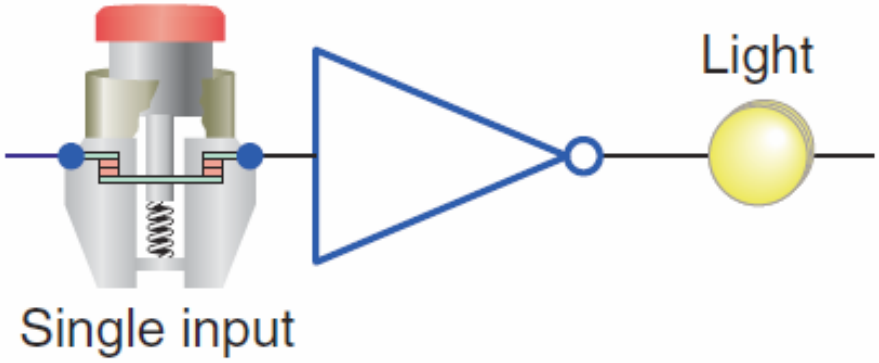


The NOT function can be performed on a contact input simply by using a normally closed instead of a normally open contact.

Truth table

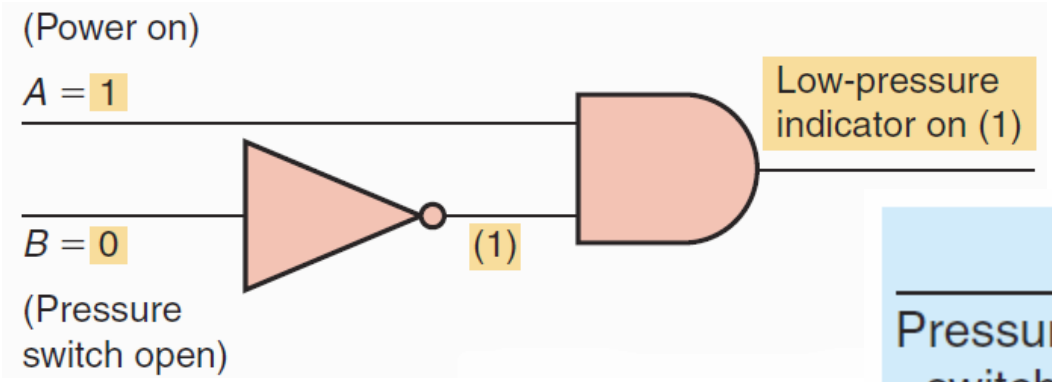
Pushbutton		Light	
Not pressed	(0)	On	(1)
Pressed	(1)	Off	(0)

Logic representation



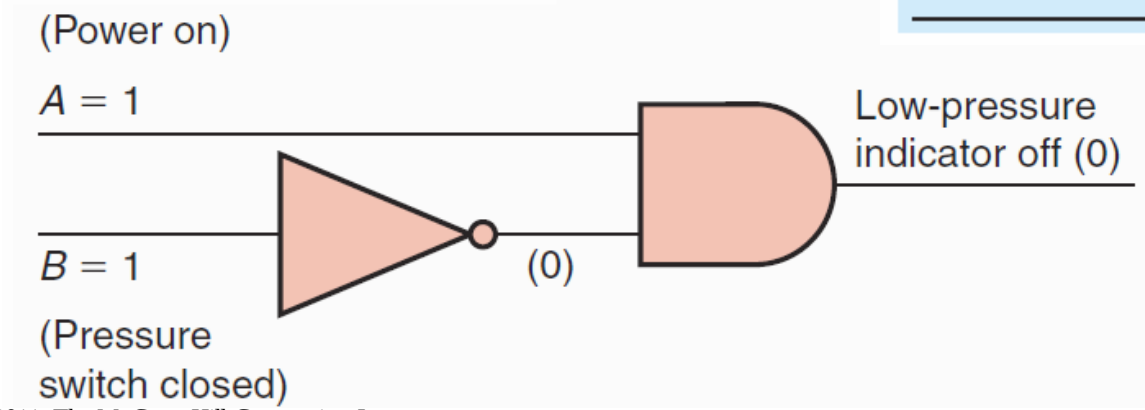
The *NOT* Function

NOT function used in conjunction with an AND gate on a low pressure indicator circuit.



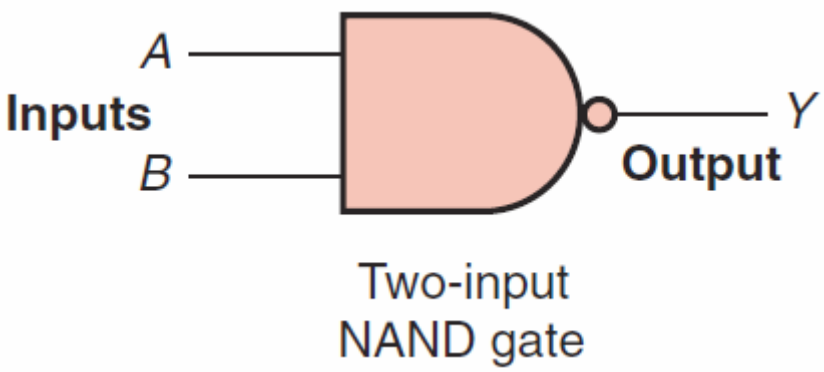
Truth table

Pressure switch	Power	Pressure indicator
0	1	1
1	1	0



The *NAND* Function

An AND gate with an inverted output is called a NAND gate.

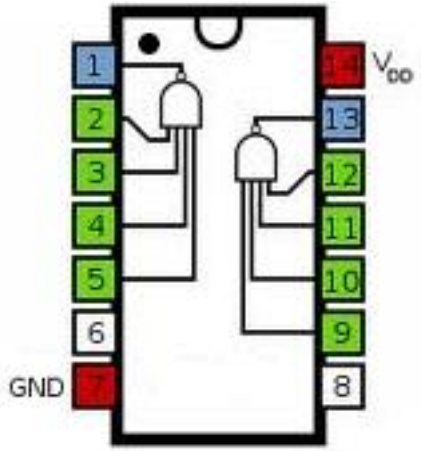


Two-input NAND gate

NAND truth table

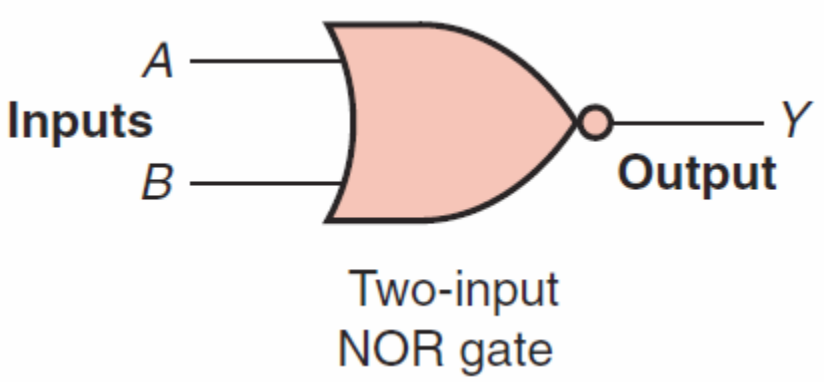
Inputs		Output
A	*B*	*Y*
0	0	1
0	1	1
1	0	1
1	1	0

The NAND gate has the opposite outputs to the AND gate.



The *NOR* Function

A OR gate with an inverted output is called a NOR gate.

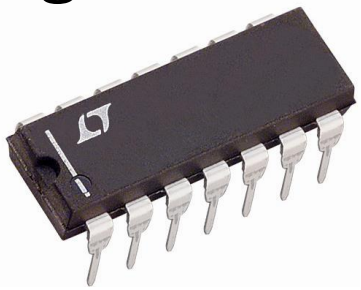


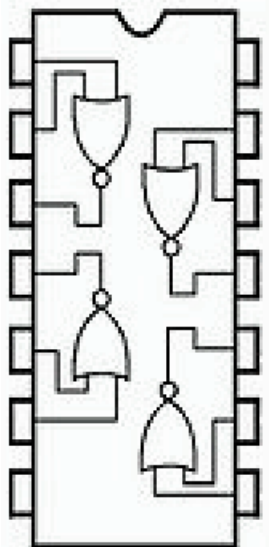
Two-input NOR gate

NOR truth table

Inputs		Output
A	*B*	*Y*
0	0	1
0	1	0
1	0	0
1	1	0

The NOR gate has the opposite outputs to the OR gate.





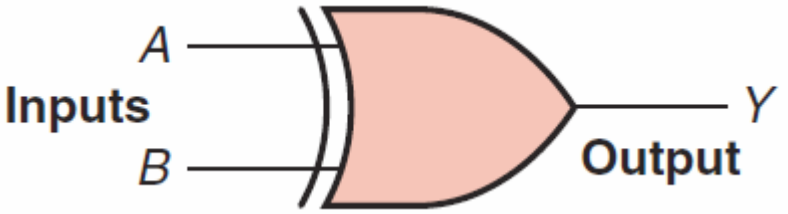
The Exclusive-OR (XOR) Function

Truth table

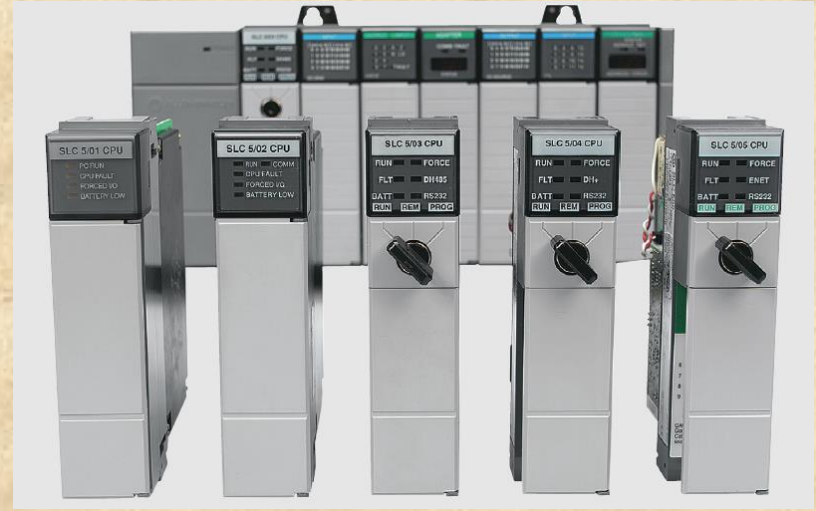
Inputs		Output
A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	0

The output of the XOR gate is HIGH (1) only when one input or the other is HIGH, but not both.

The exclusive-OR gate is commonly used for the comparison of two binary numbers.

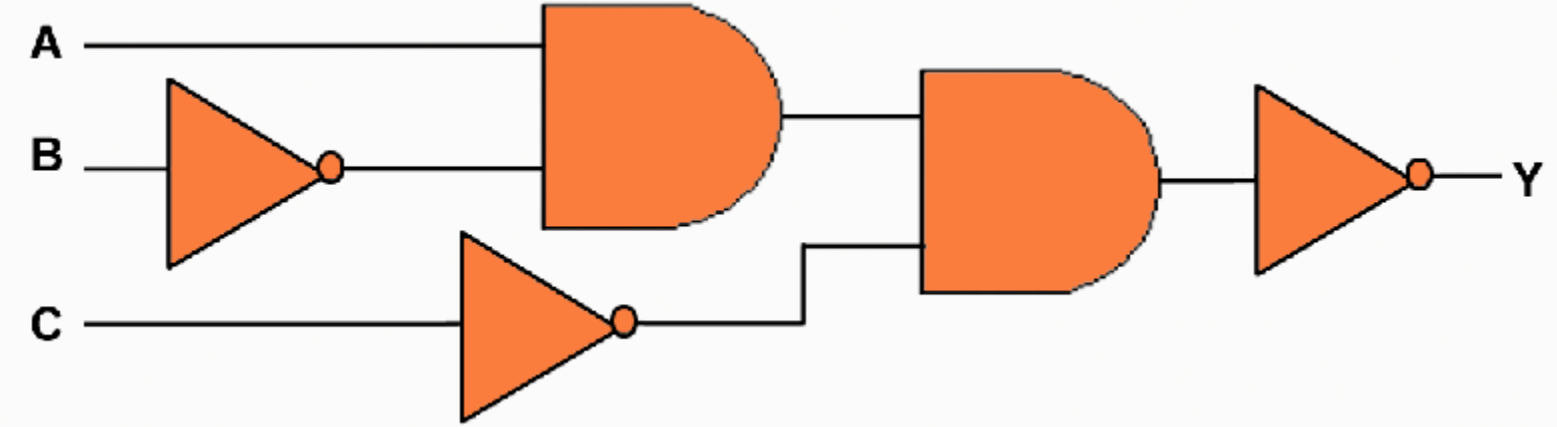


4.3



Boolean Algebra

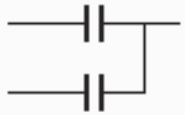
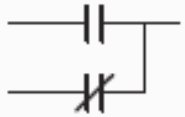
The mathematical study of the binary number system and logic is called *Boolean algebra.*



Boolean algebra is used to write combinations of logic statements that are used to solve PLC programming problems.

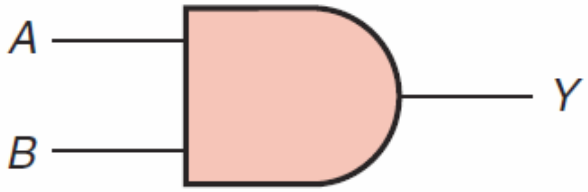
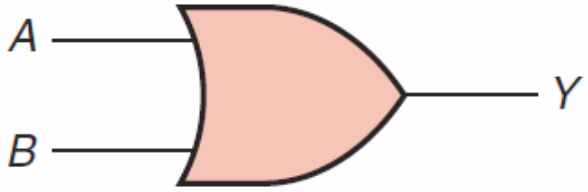
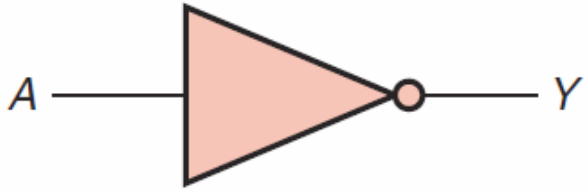
Typical Boolean Instructions or Statements

Instructions are based on the basic Boolean operators of AND, OR, and NOT

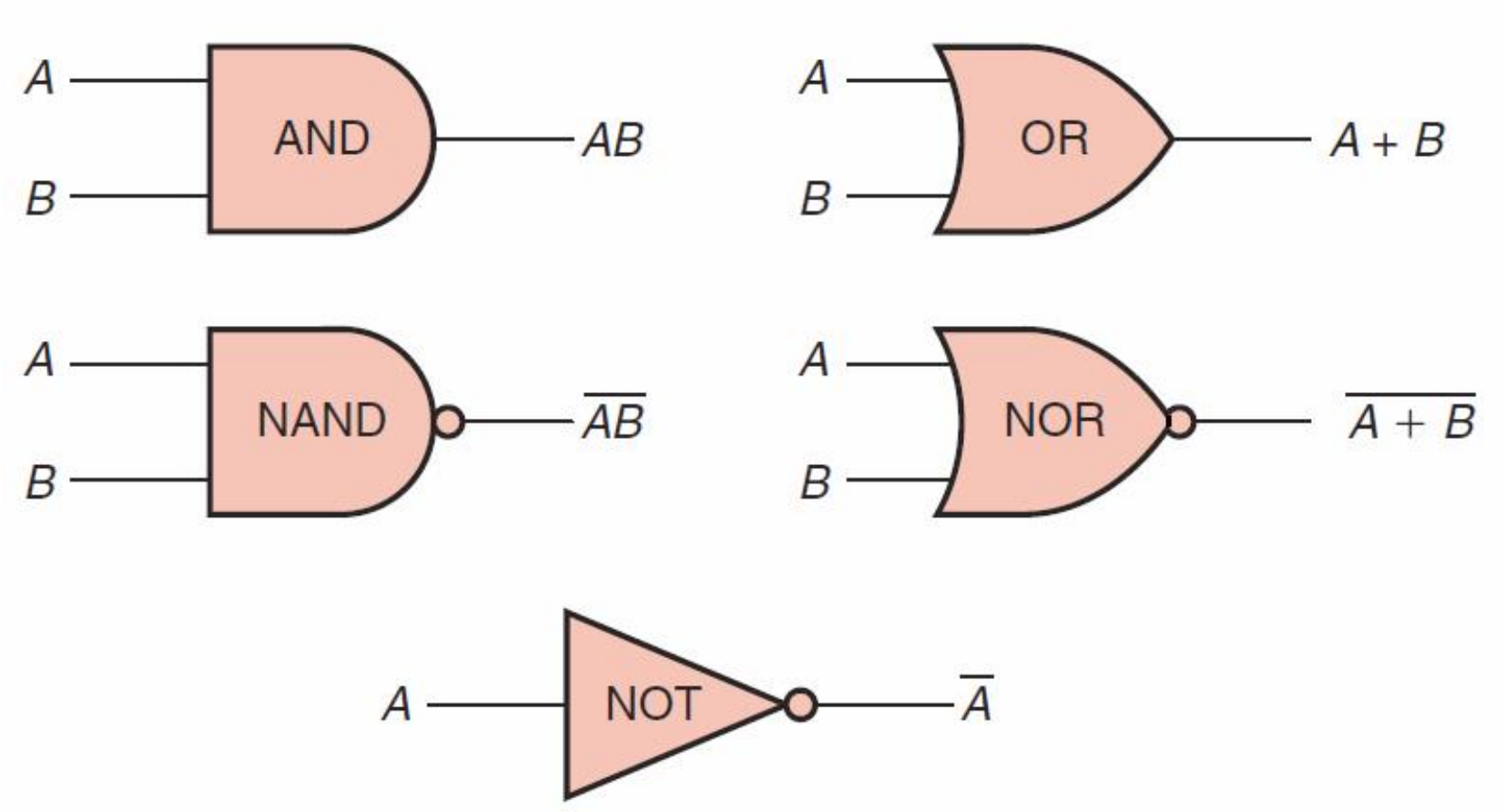
Boolean Instruction and Function	Graphic Symbol
Or (OR) Logically ORs a normally open contact in parallel with another contact in a rung.	
Or Not (OR NOT) Logically ORs a normally closed contact in parallel with another contact in a rung.	
And (AND) Logically ANDs a normally open contact in series with another contact in a rung.	
And Not (AND NOT) Logically ANDs a normally closed contact in series with another contact in a rung.	

Even though these instructions are programmed in a list format they implement the same logic as relay ladder logic.

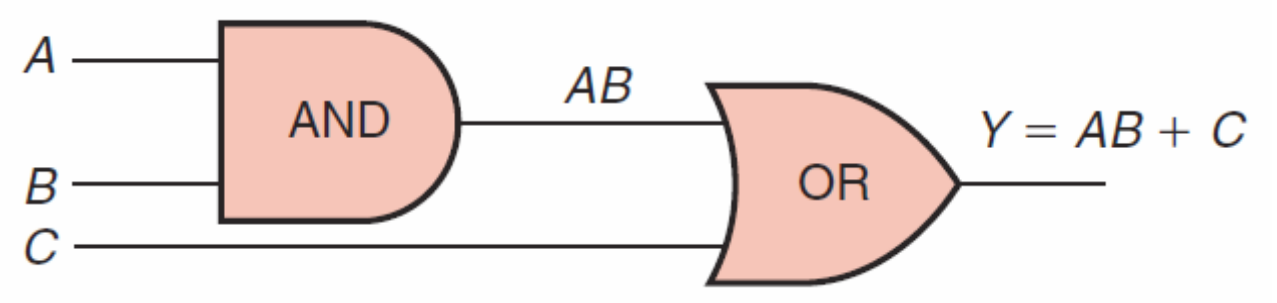
Boolean algebra as related to AND, OR, and NOT functions.

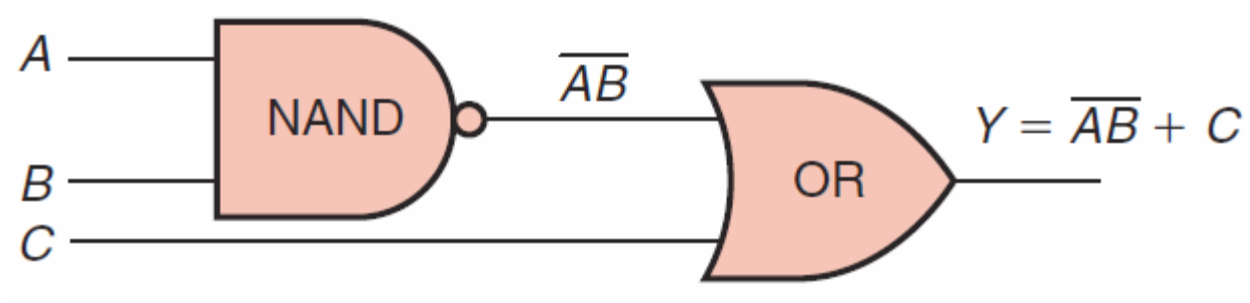
Logic symbol	Logic statement	Boolean equation	Boolean notations
	*Y* is 1 if *A* and *B* are 1	$Y = A \cdot B$ or $Y = AB$	Symbol Meaning • and + or – not ° invert = result in
	*Y* is 1 if *A* or *B* is 1	$Y = A + B$	
	*Y* is 1 if *A* is 0 *Y* is 0 if *A* is 1	$Y = \overline{A}$	

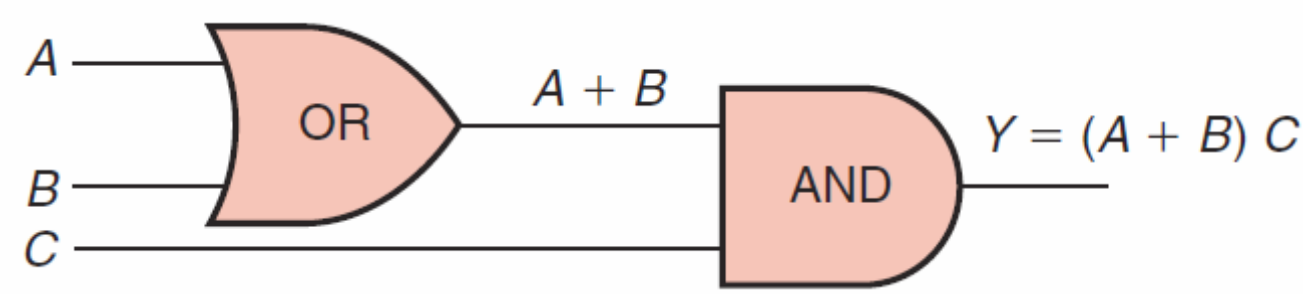
Logic operators used form *logical statements*.

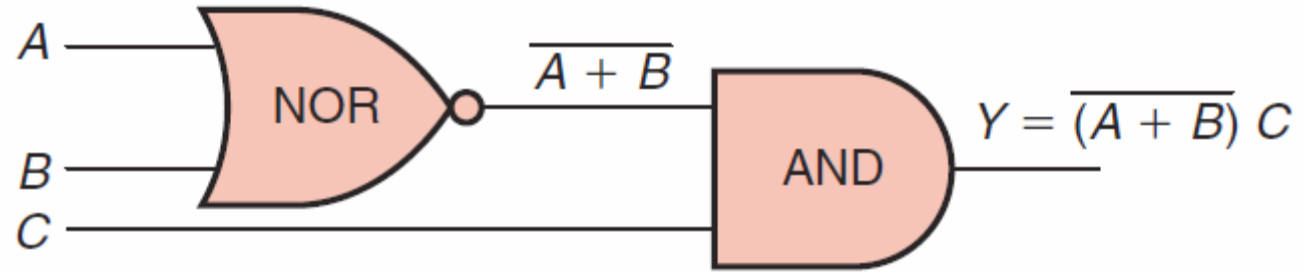


Logic operators used to form *Boolean equations.*









Some laws of Boolean algebra are similar to those of ordinary algebra.

COMMUTATIVE LAW

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

ASSOCIATIVE LAW

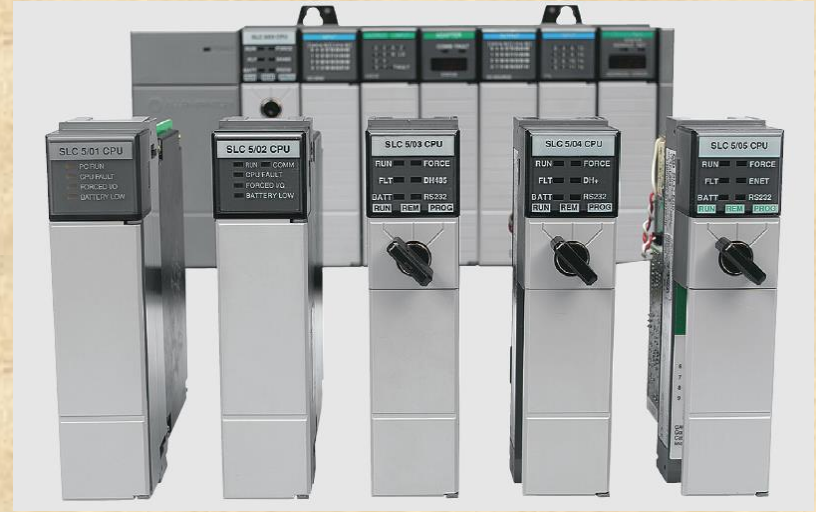
$$(A + B) + C = A + (B + C)$$
$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

DISTRIBUTIVE LAW

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$
$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

This law holds true only in Boolean algebra

4.4



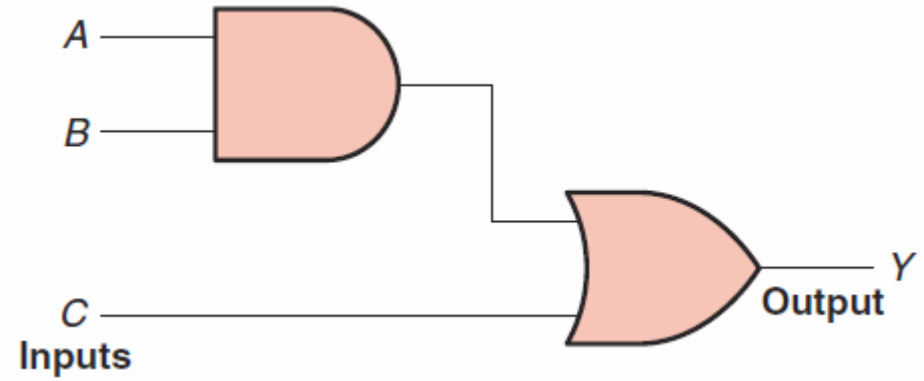
Developing Logic Gate Circuits from Boolean Expressions

Logic gate circuit developed from the Boolean expression $Y = AB + C$.

Gates required:

1 - AND gate with input *A* and *B*

1 - OR gate with input *C* and output from previous AND gate



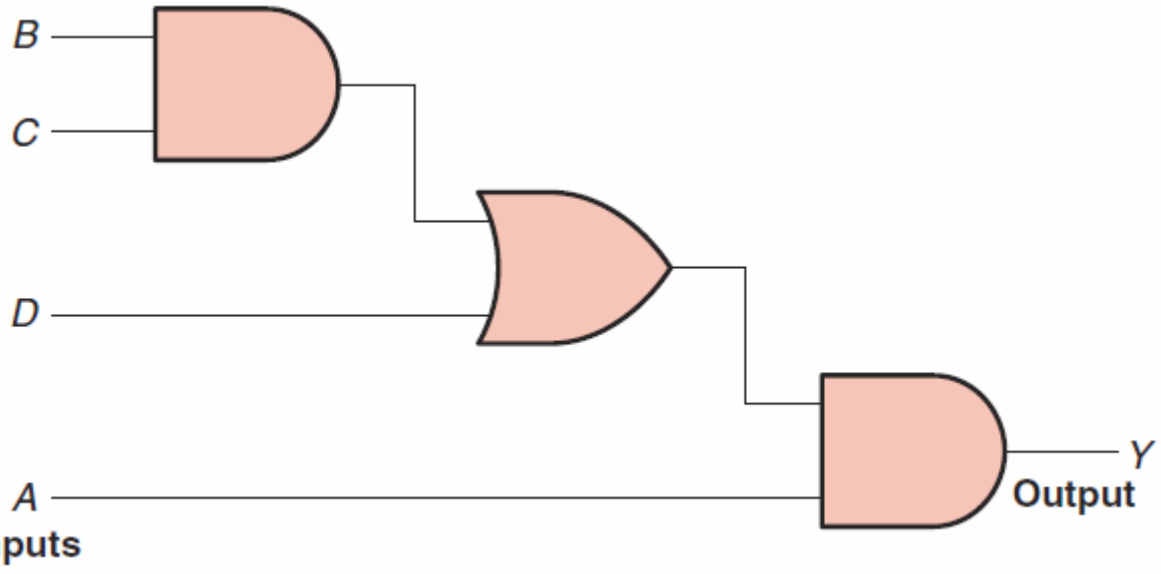
Logic gate circuit developed from the Boolean expression $Y = A\,(BC + D)$.

Gates required:

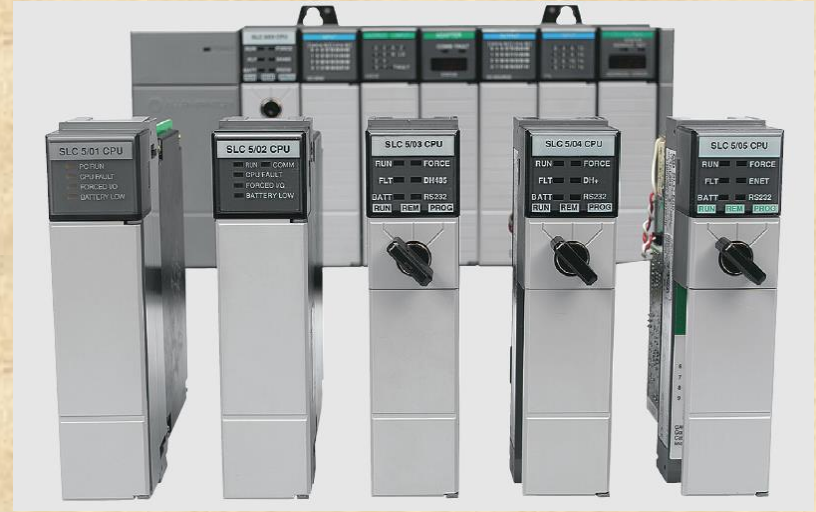
1 - AND gate with input B and C

1 - OR gate with inputs *BC* and *D*

1 - AND gate with inputs *A* and the output from the OR gate

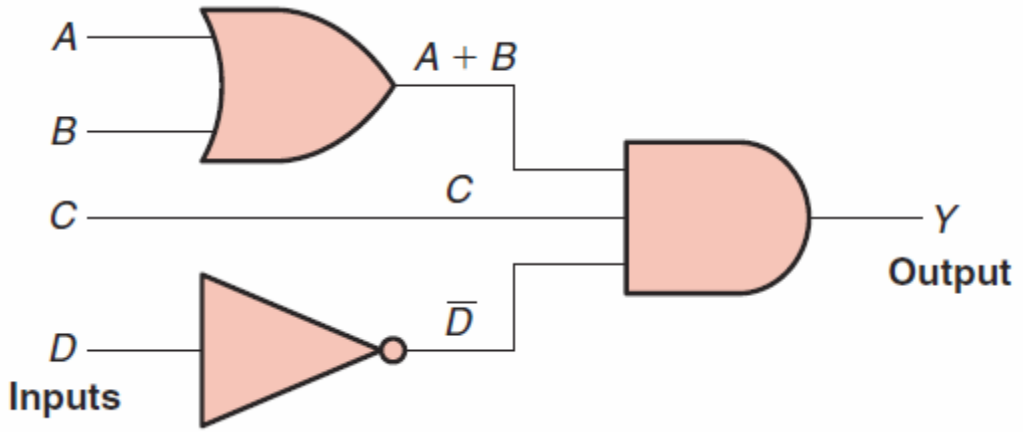


4.5



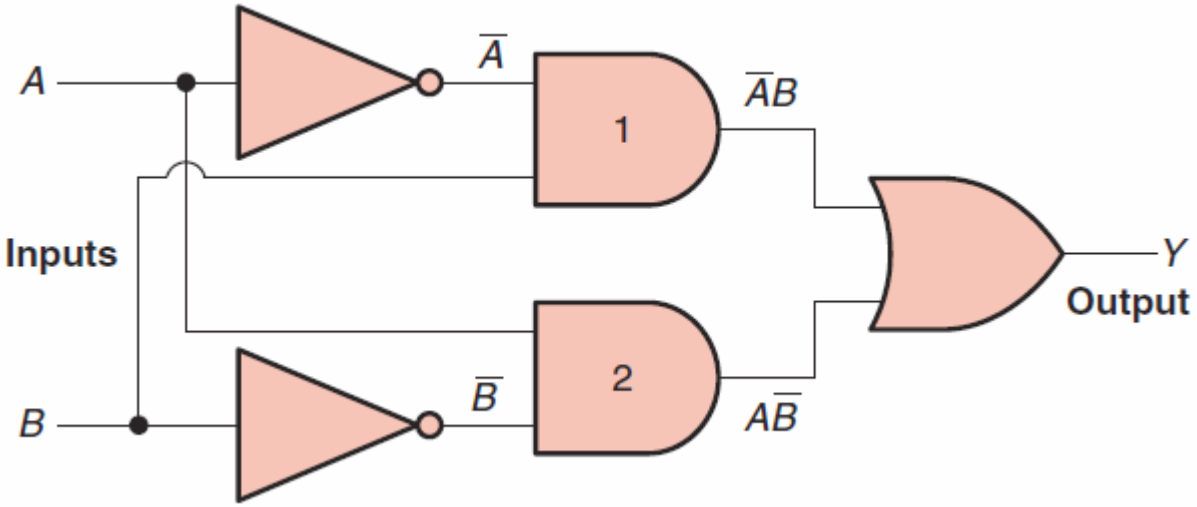
Producing the Boolean Equation for a Given Logic Gate Circuit

Determining the logic circuit Boolean equation



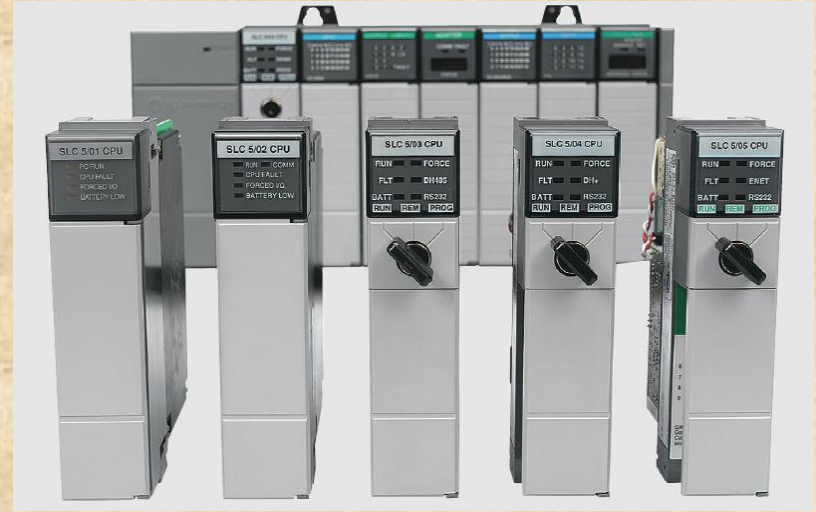
- The output of the OR gate is $A + B$
- The output of the inverter is $\overline{D}$
- Based on the input combination applied to the AND gate the Boolean equation for the circuit is $Y = C\,\overline{D}\,(A + B)$

Determining the logic circuit Boolean equation



- The output of AND gate 1 is $\overline{A}B$
- The output of AND gate 2 is $A\overline{B}$
- Based on the combination of inputs applied to the OR gate the Boolean equation for the circuit is $Y = \overline{A}B + A\overline{B}$

4.6

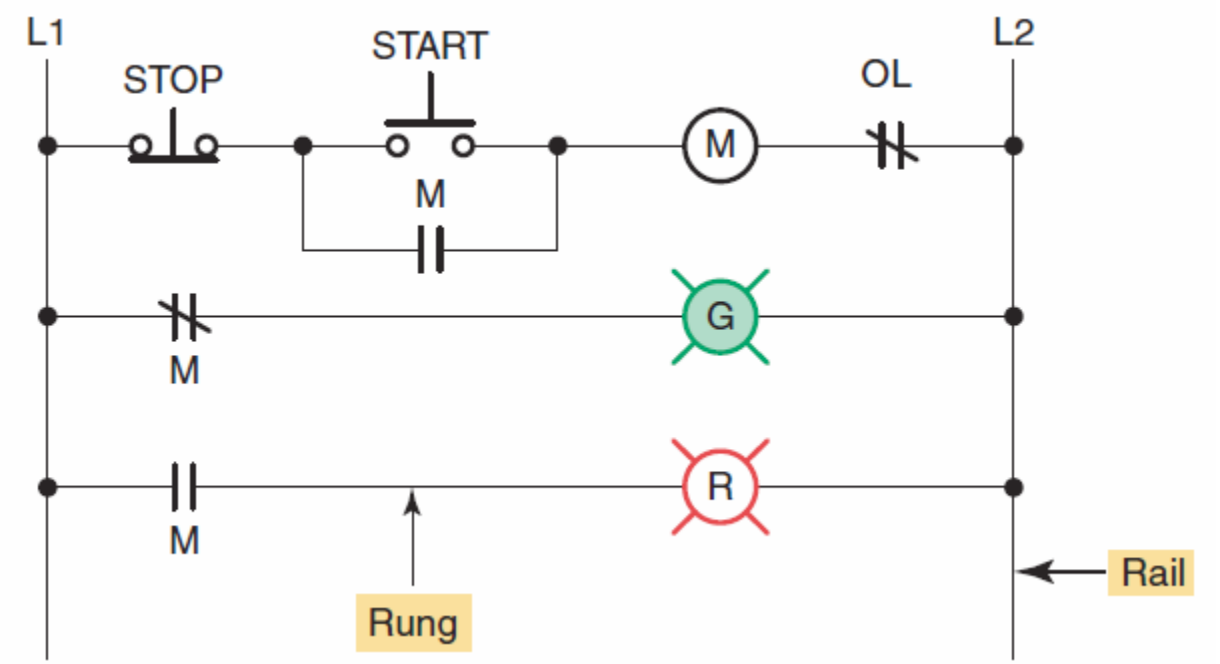


Hardwired Logic versus Programmed Logic

The term *hardwired logic* refers to logic control functions that are determined by the way devices are electrically interconnected.

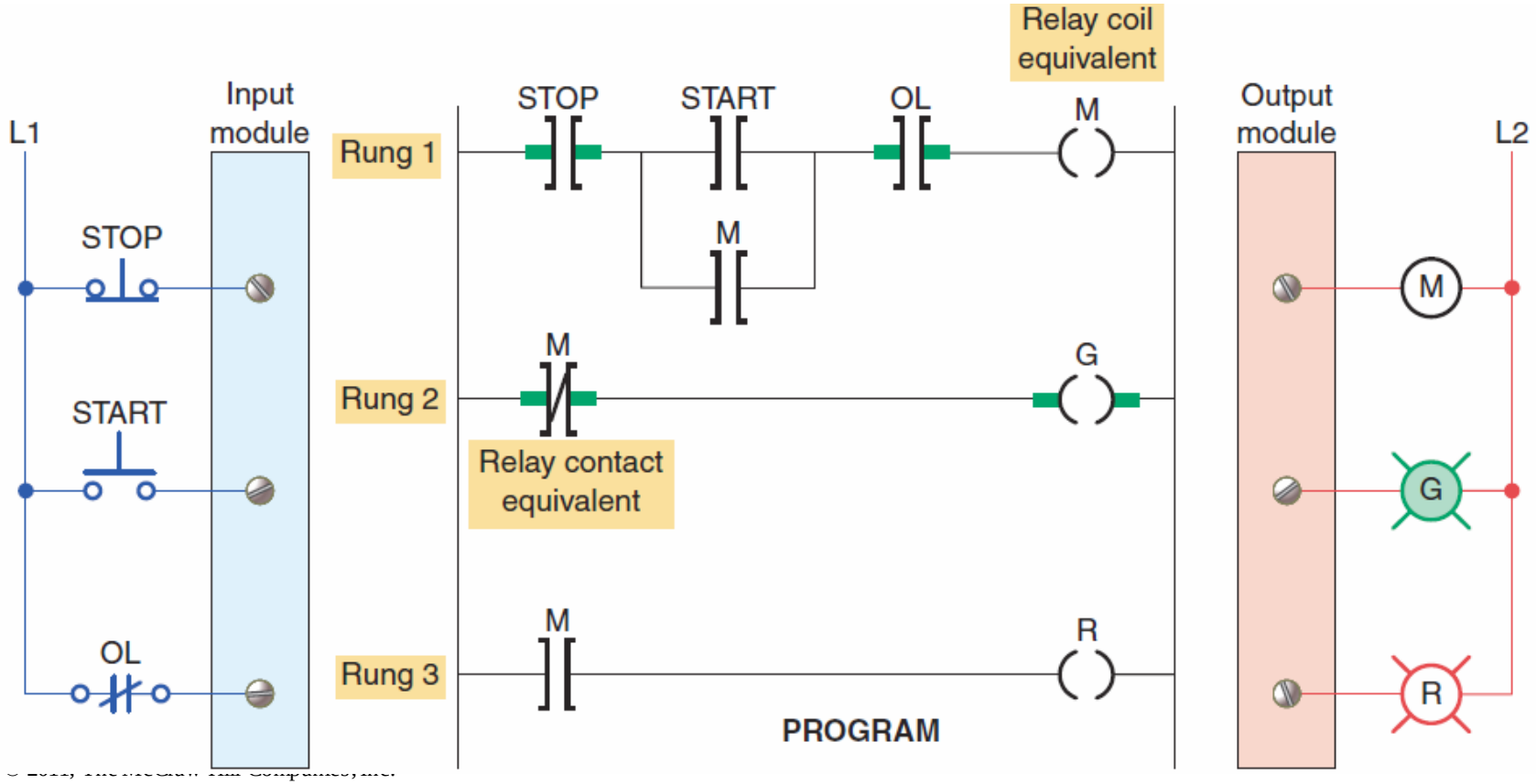
Hardwired logic is fixed and changeable only by altering the way devices are electrically interconnected.

Hardwired motor control program.



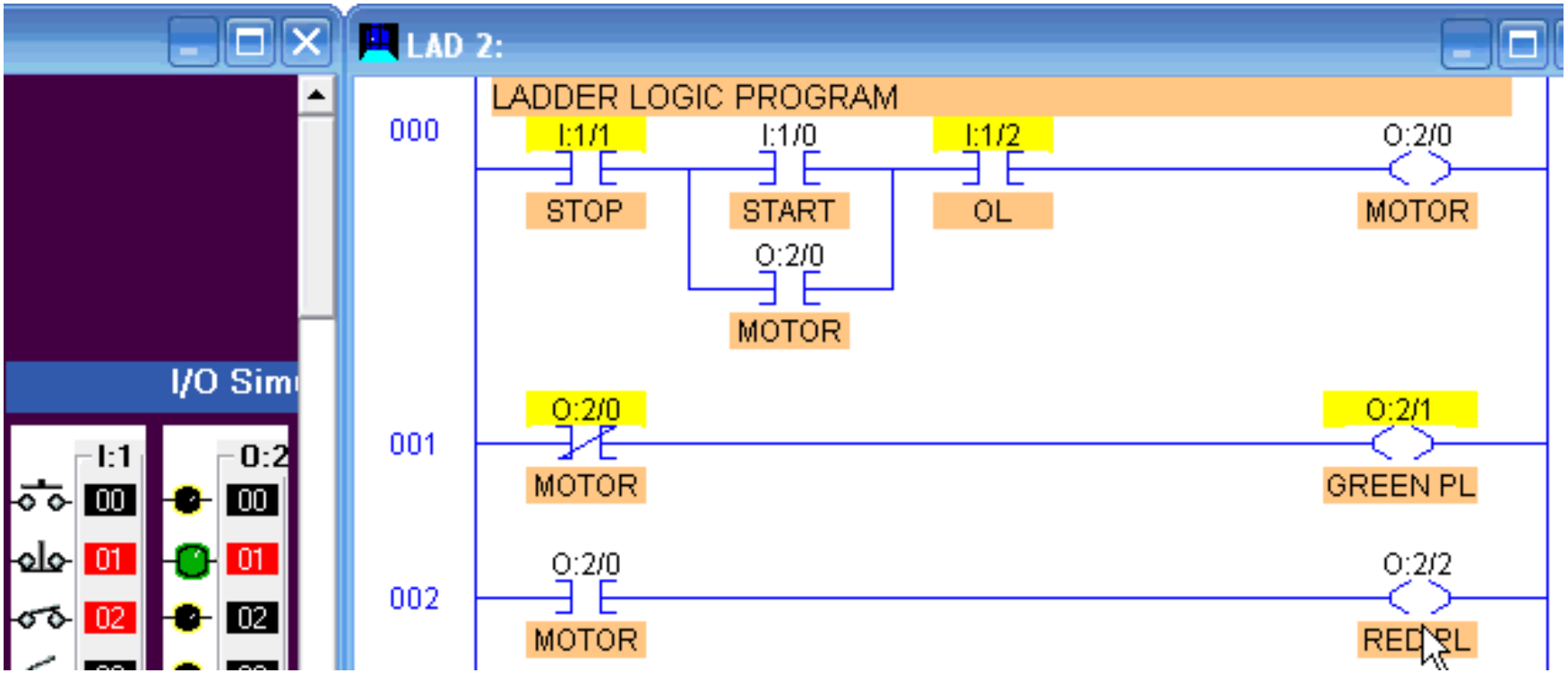
Programmable control **is based on the basic logic functions, which are programmable and easily changed.**

PLC programmed motor control program.



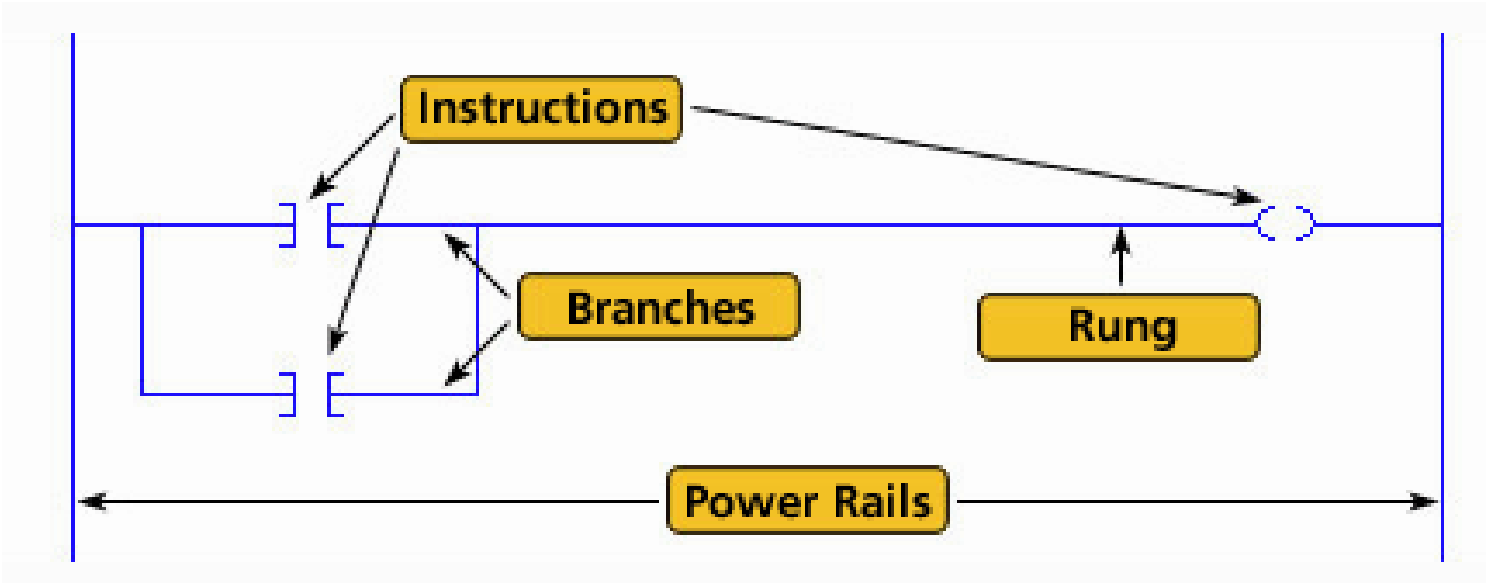
Simulated PLC motor control program.

Ladder Logic Program



I/O

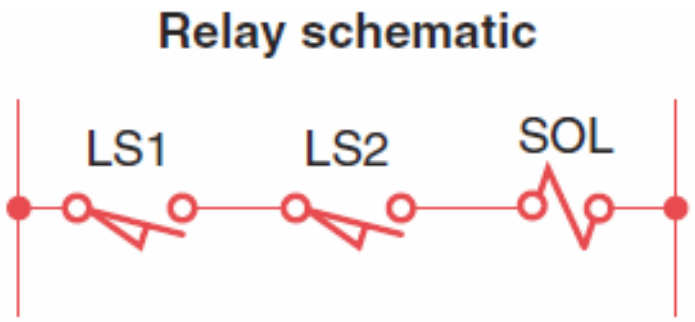
The most common PLC programming language is *ladder logic* .

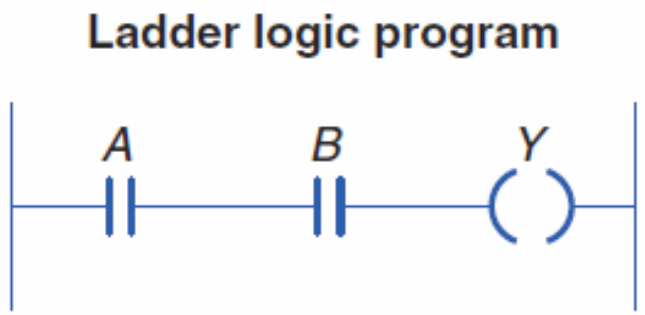


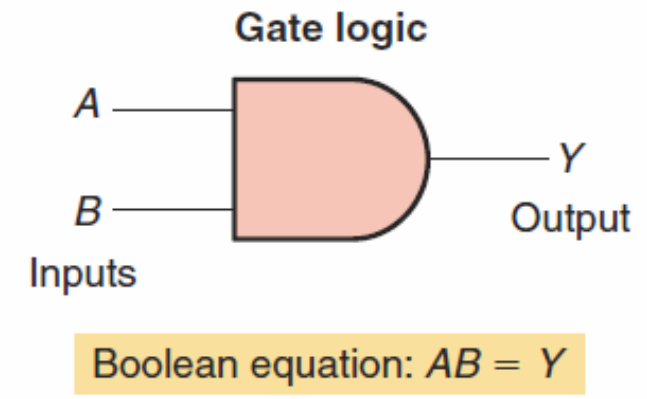
A ladder logic program consists of several rungs, each of which controls an output.

Each rung is a combination of input conditions (symbols) connected from left to right, with the symbol that represents the output at the far right.

Relationship between the relay ladder *schematic*, the ladder logic *program*, and the equivalent logic *gate circuit*.

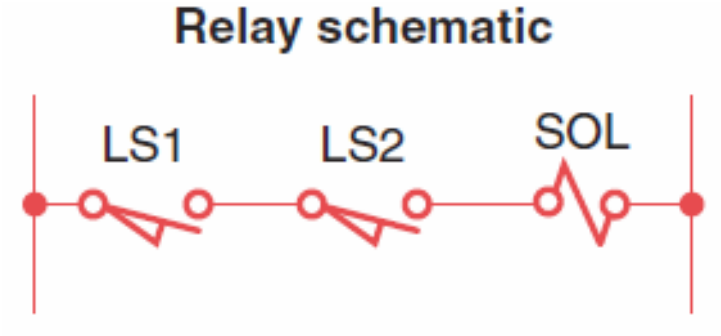


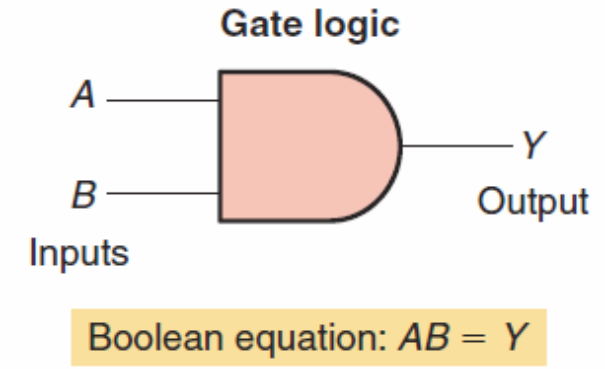




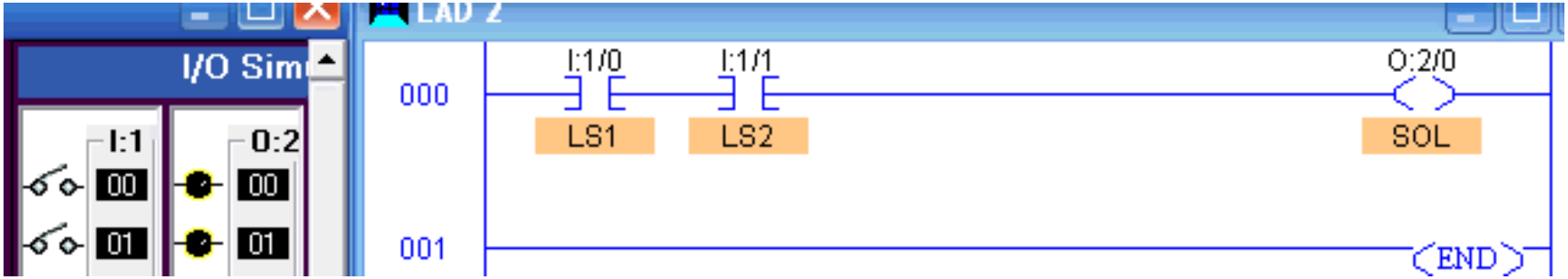
Two limit switches connected in series and used to control a solenoid valve.

Program simulation of two switches in series used to control an output.



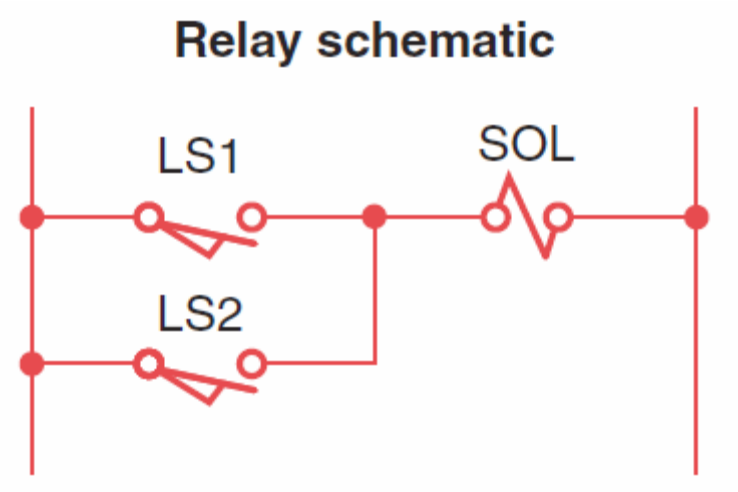


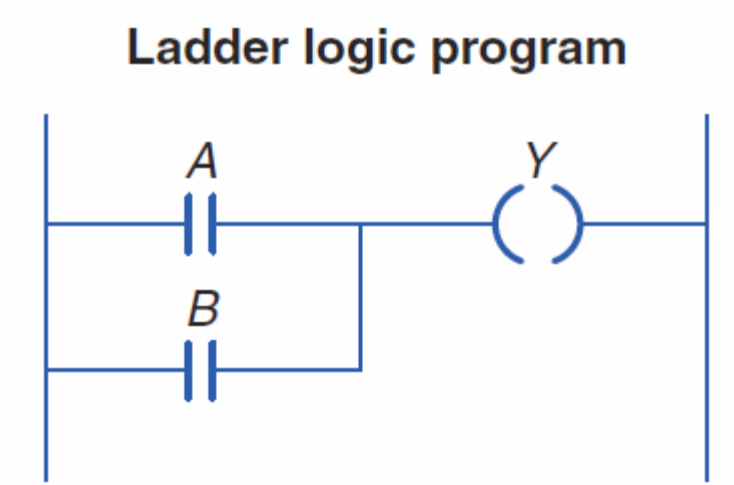
Program

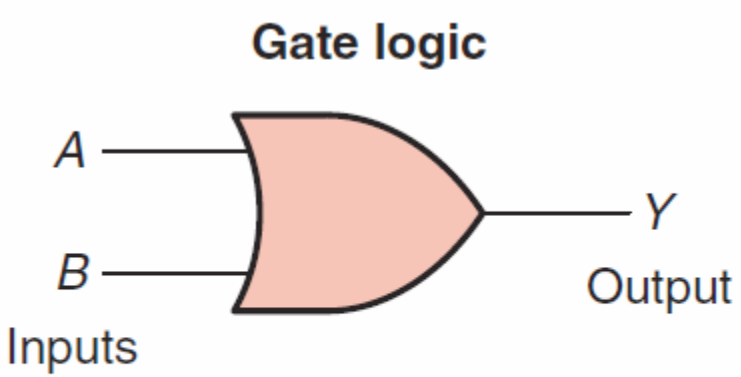


I/O

Two limit switches connected in *parallel* and used to control a solenoid valve.

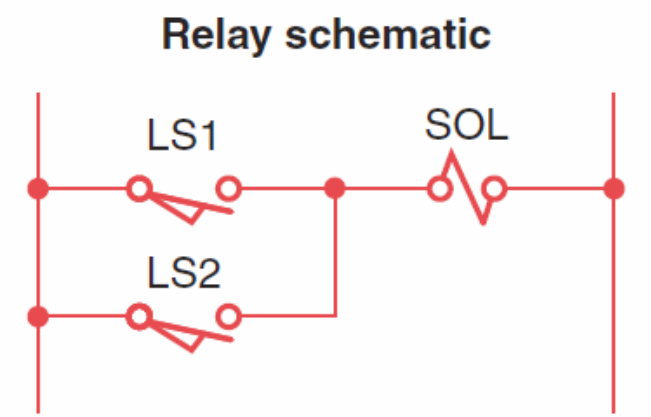


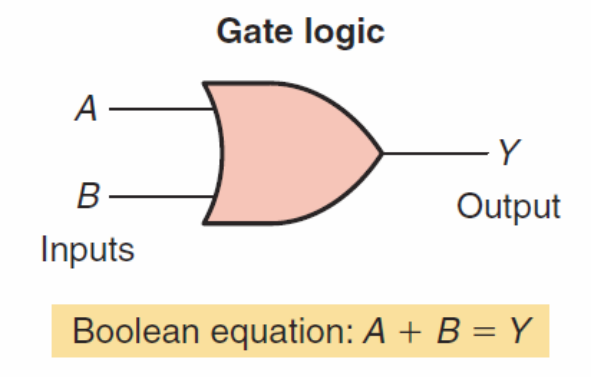




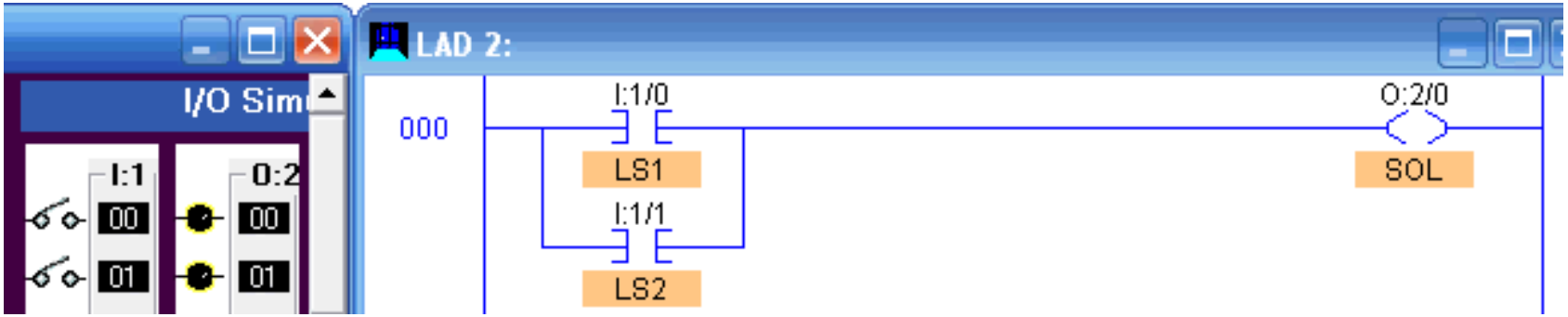
Boolean equation: $A + B = Y$

Program simulation of two switches in parallel used to control an output.



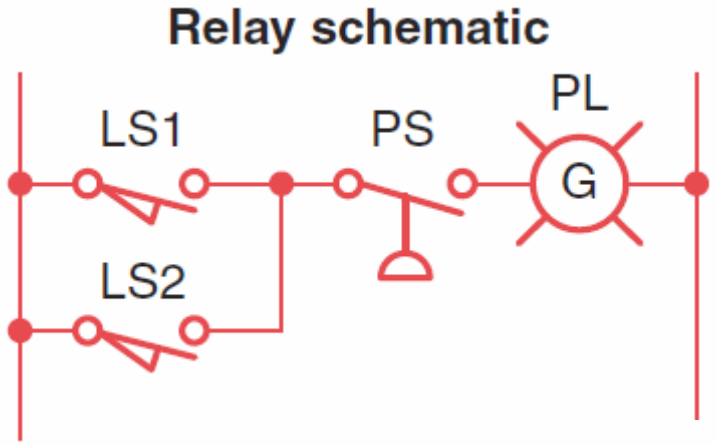


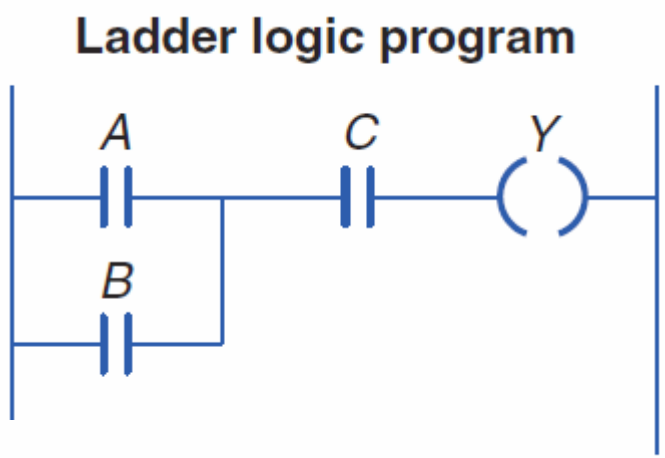
Program

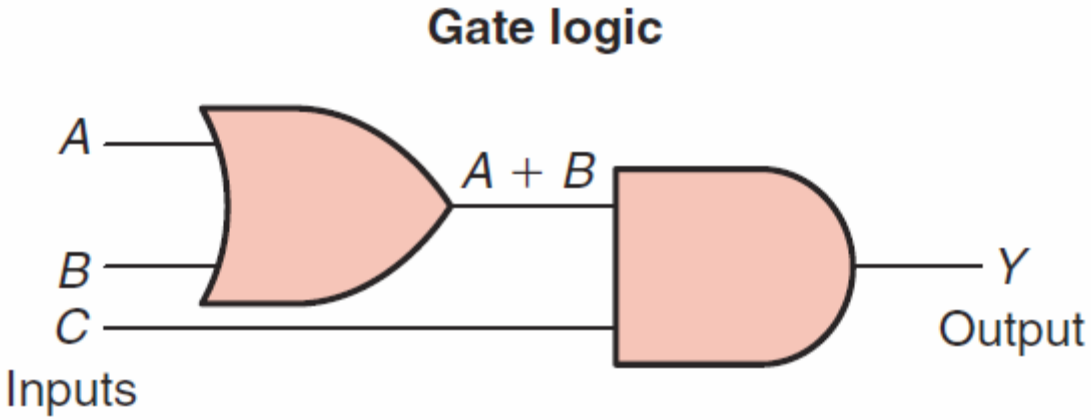


I/O

Two limit switches connected in *parallel* with each other and in *series* with a pressure switch to control a pilot light.

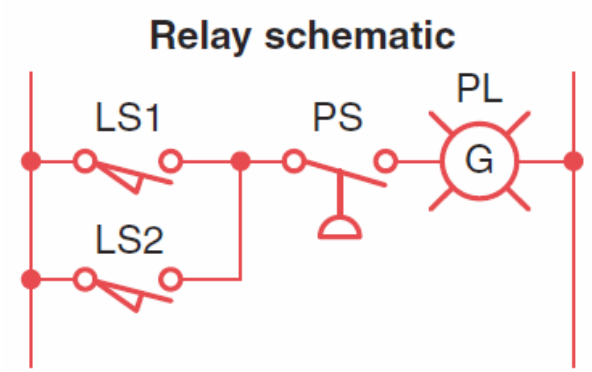


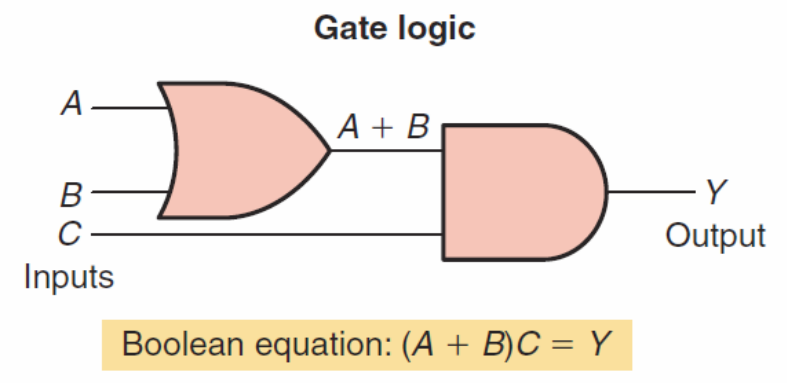




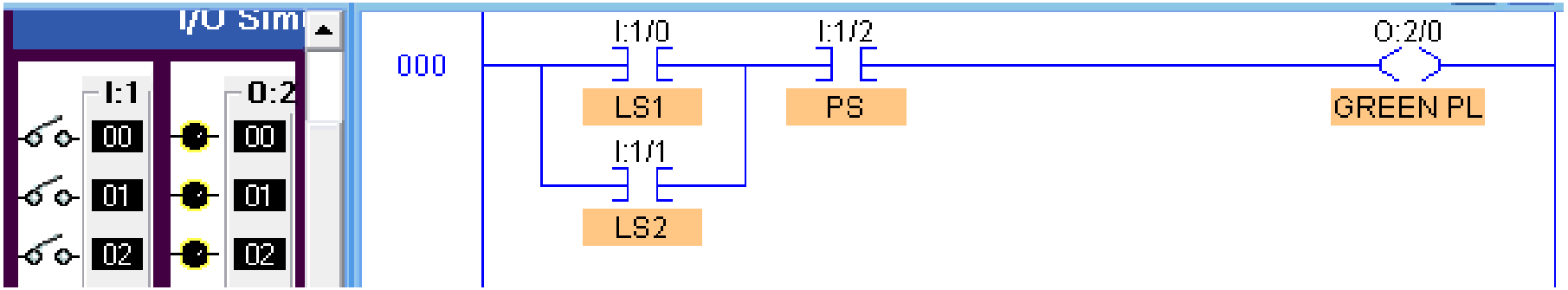
Boolean equation: $(A + B)C = Y$

Program simulation of two switches in parallel and this pair in series with a switch used to an output.



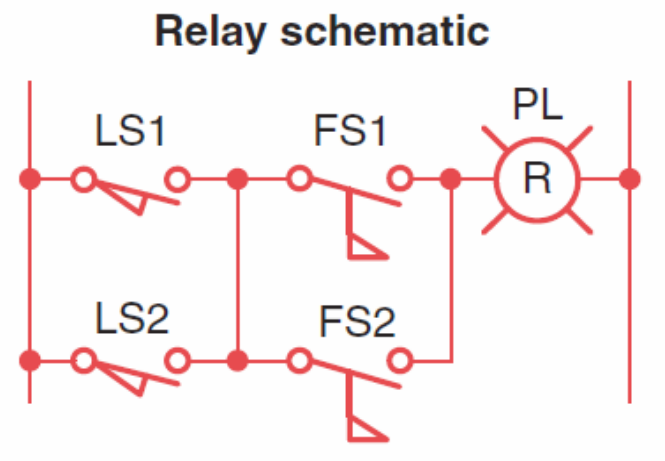


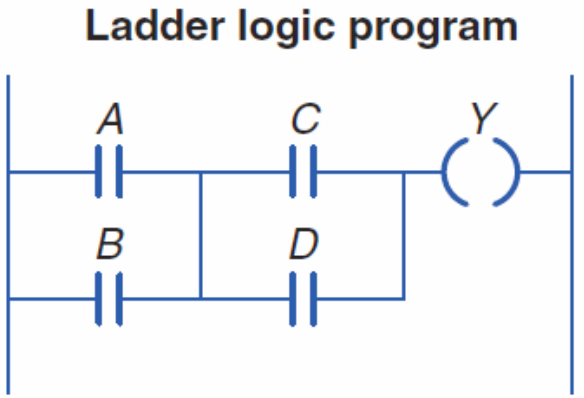
Program

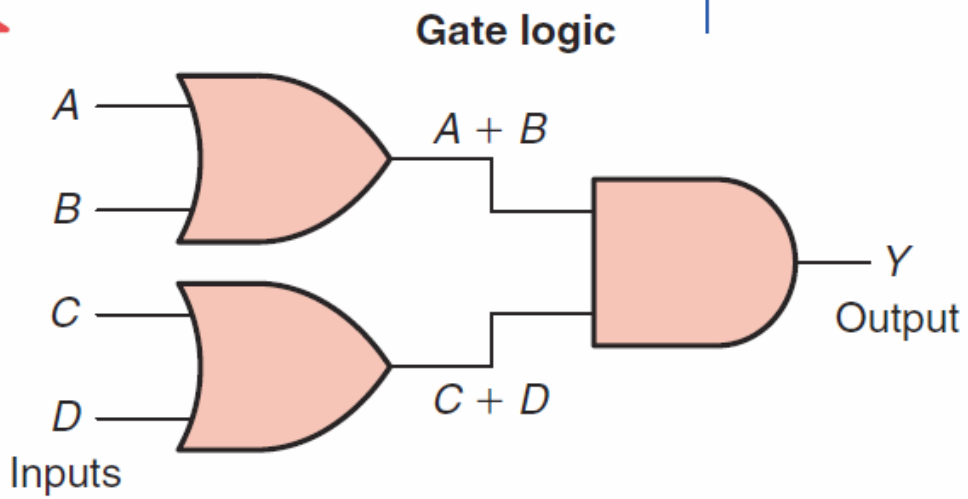


I/O

Two limit switches connected in *parallel* with each other and in *series* with two sets of flow switches (in *parallel*), and used to control a pilot light.

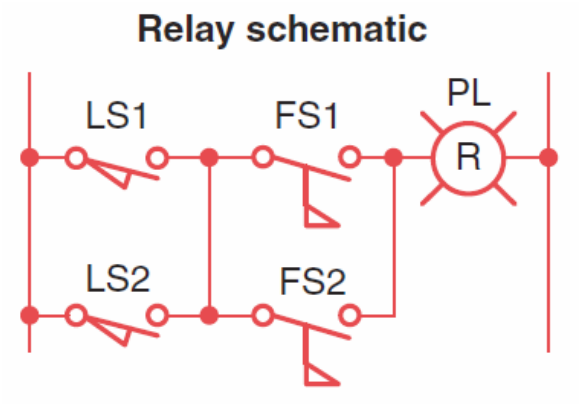


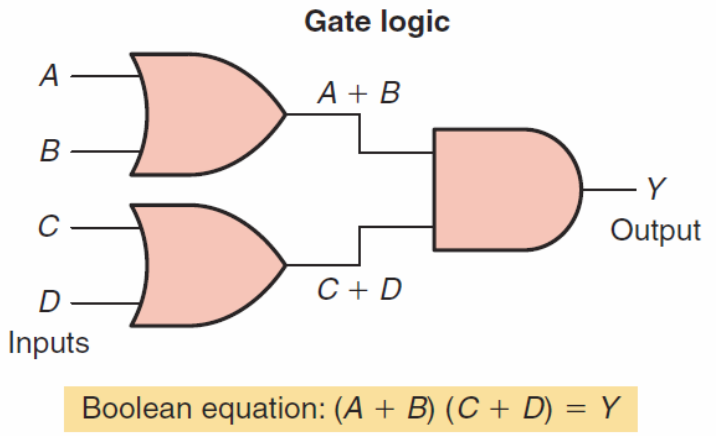




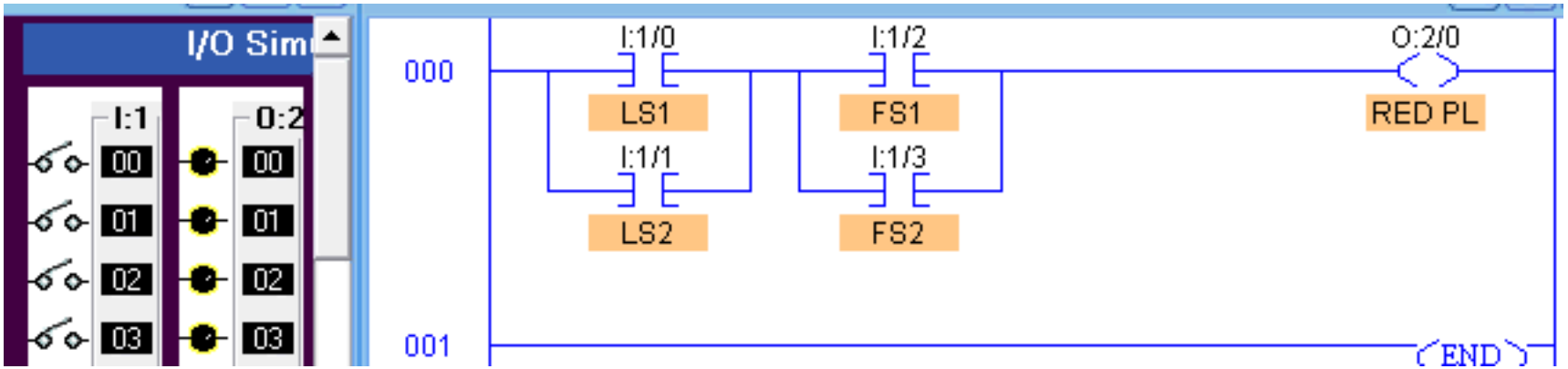
Boolean equation: $(A + B)(C + D) = Y$

Program simulation of two switches in parallel and in series with a second pair of switches in parallel.



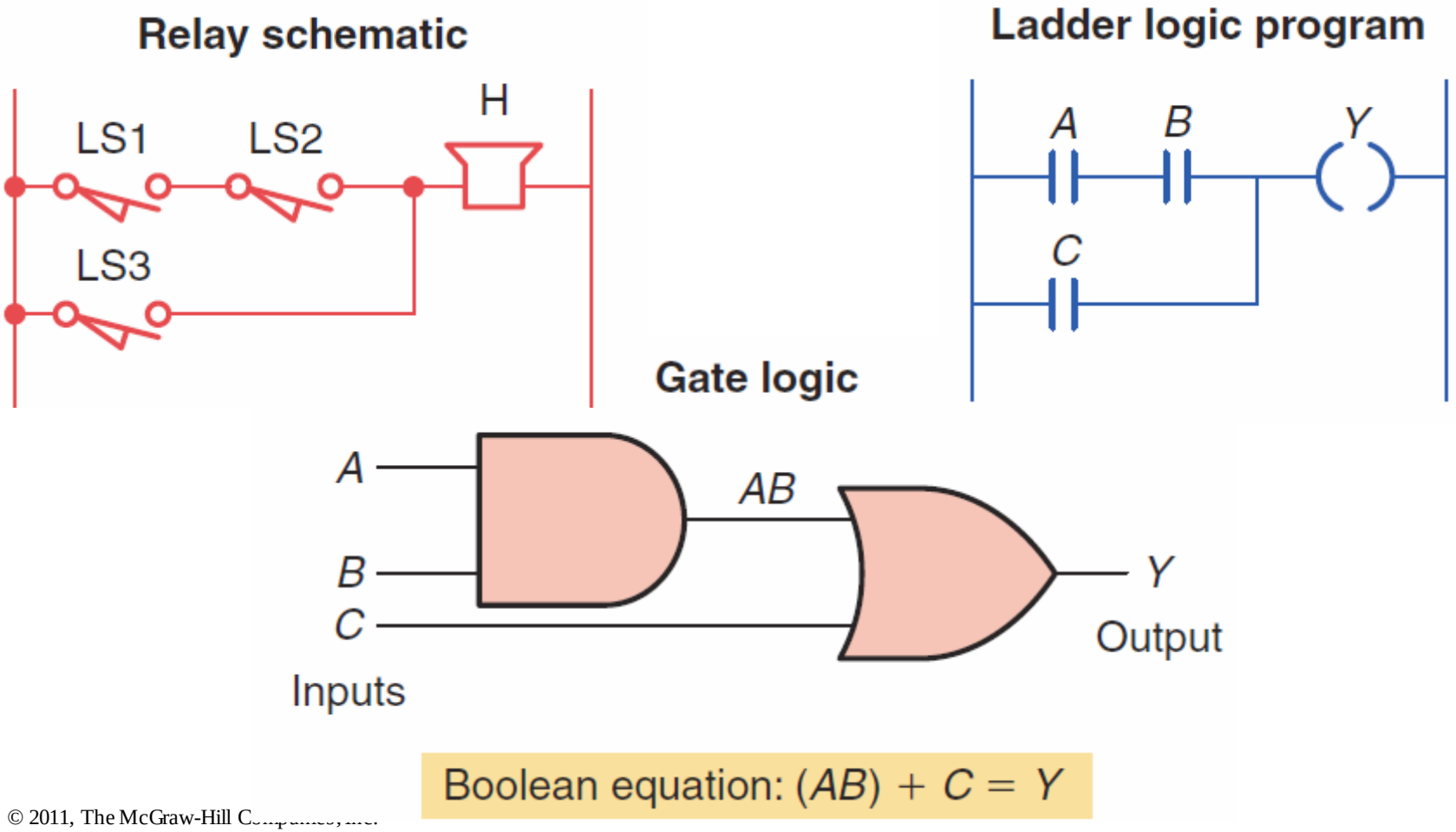


Program

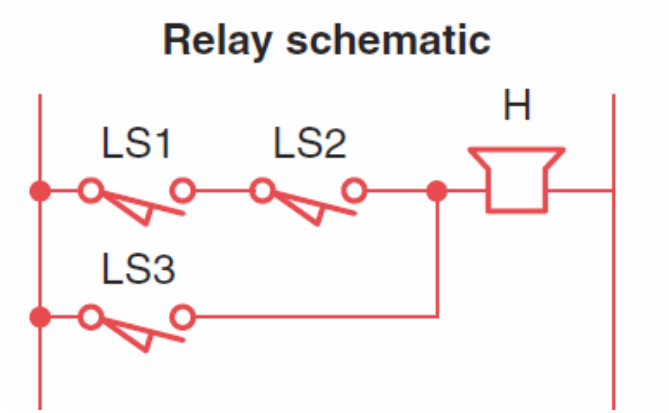


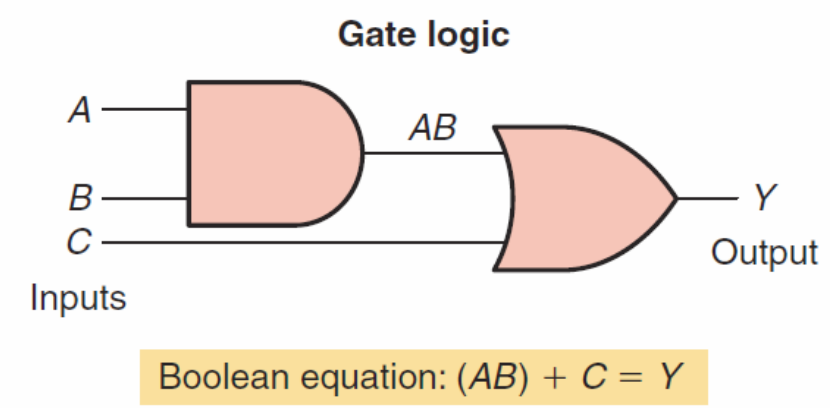
I/O

Two limit switches connected in *series* with each other and in *parallel* with a third limit switch, and used to control a warning horn.

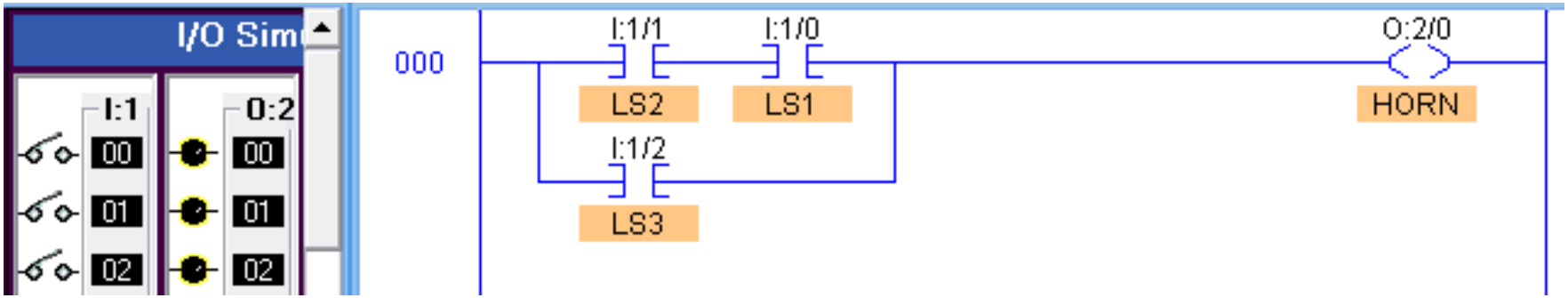


Program simulation of two switches in series and in parallel with a third switch.



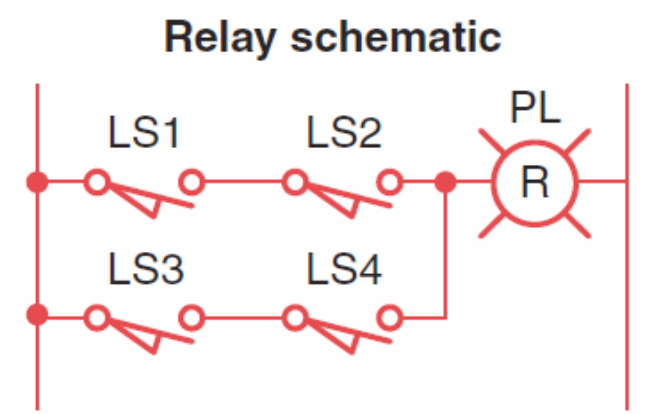


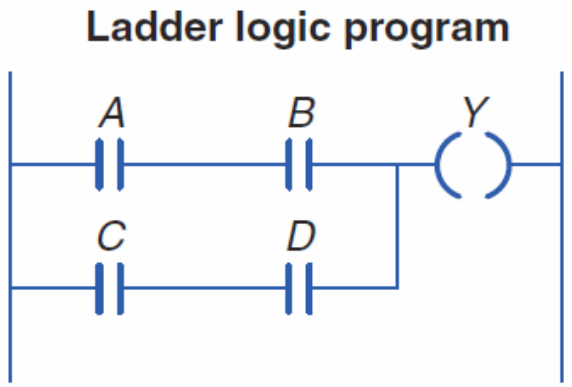
Program

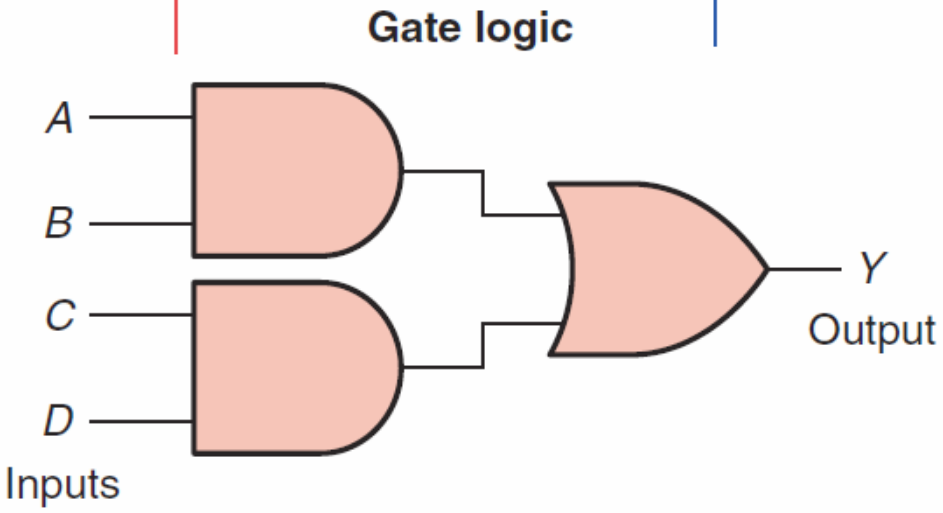


I/O

Two limit switches connected in *series* with each other and in *parallel* with two other limit switches (in *series*), and used to control a pilot light.

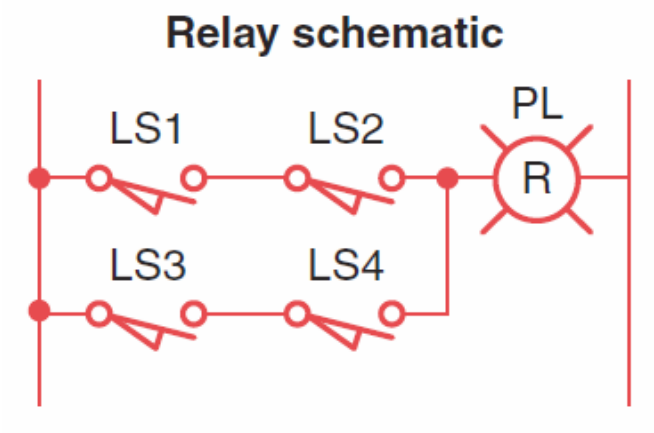


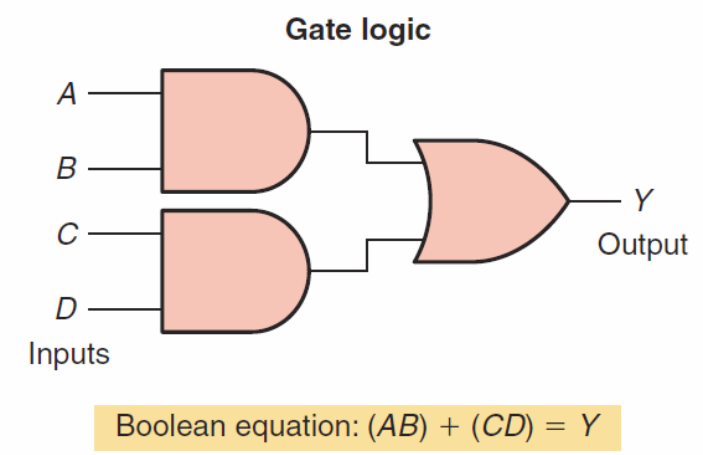




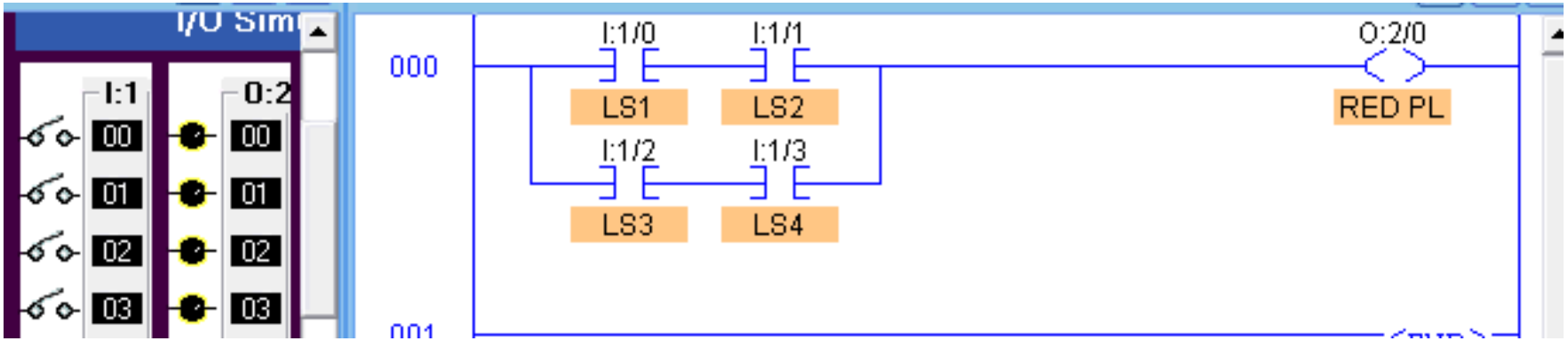
Boolean equation: $(AB) + (CD) = Y$

Program simulation of two switches in series and in parallel with a second pair in series.



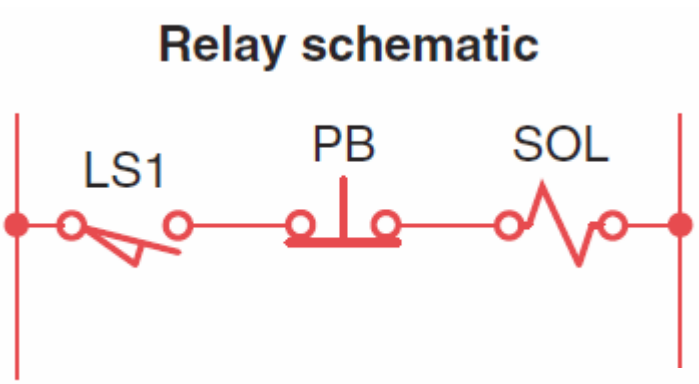


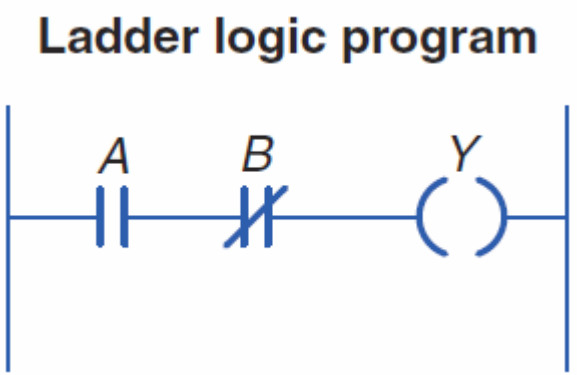
Program

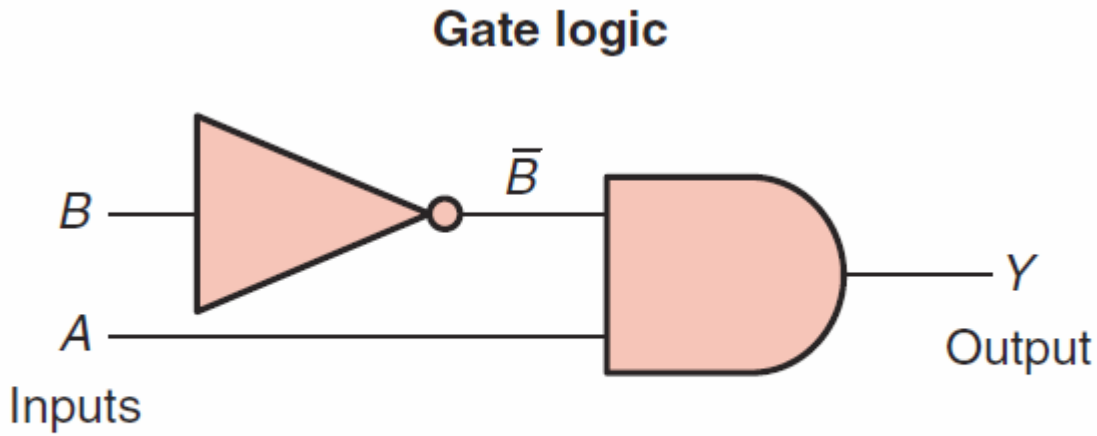


I/O

One limit switch connected in series with a *normally closed pushbutton* and used to control a solenoid valve.

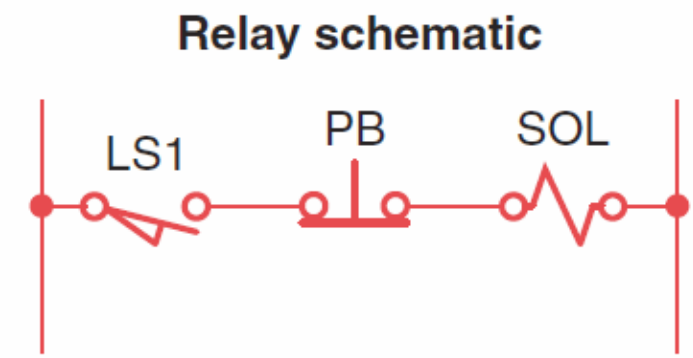


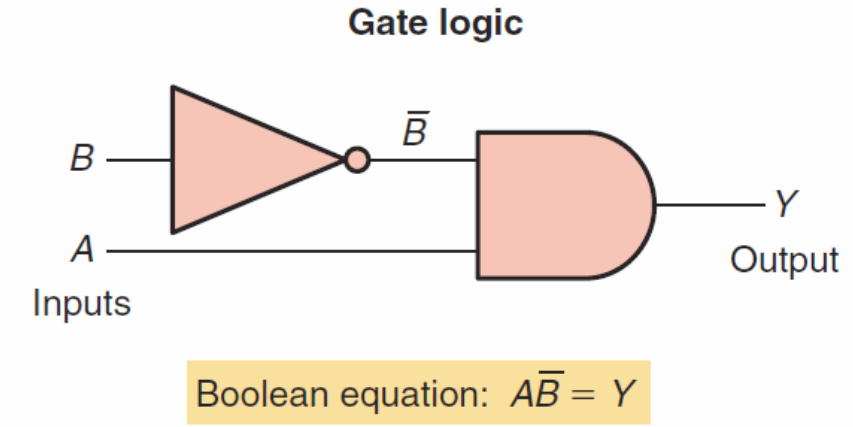




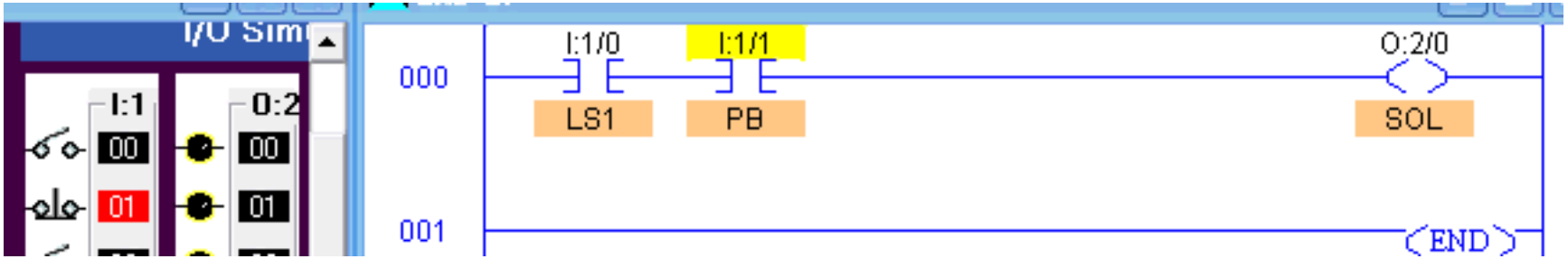
Boolean equation: $A\overline{B} = Y$

Program simulation of one switch connected in series with a normally closed pushbutton.



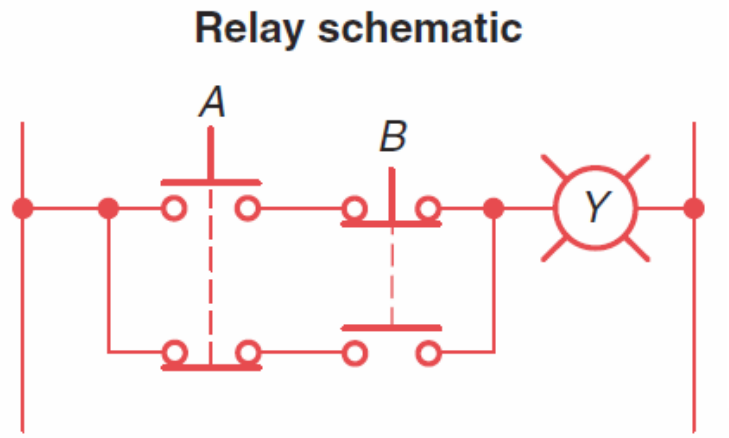


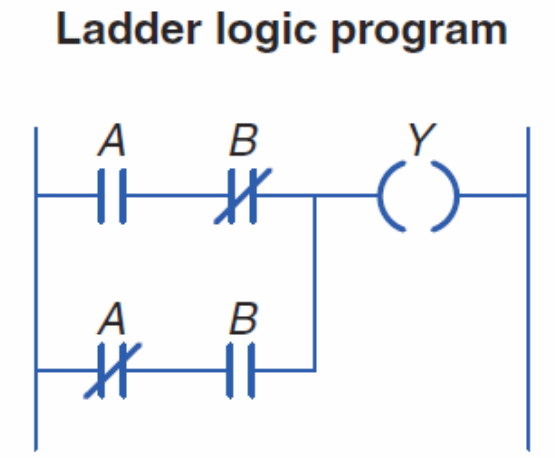
Program

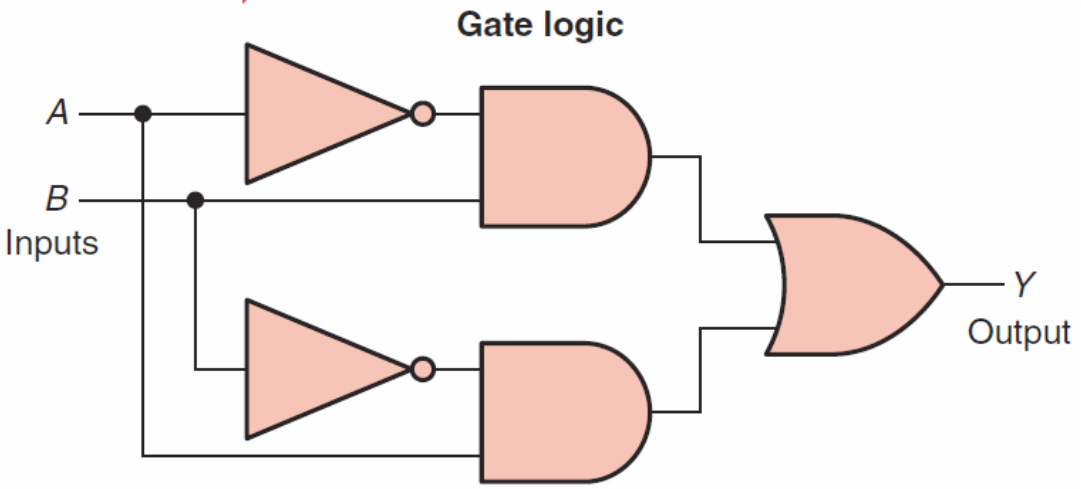


I/O

The output lamp of this circuit is ON only when pushbutton A or B is pressed, *but not both*.



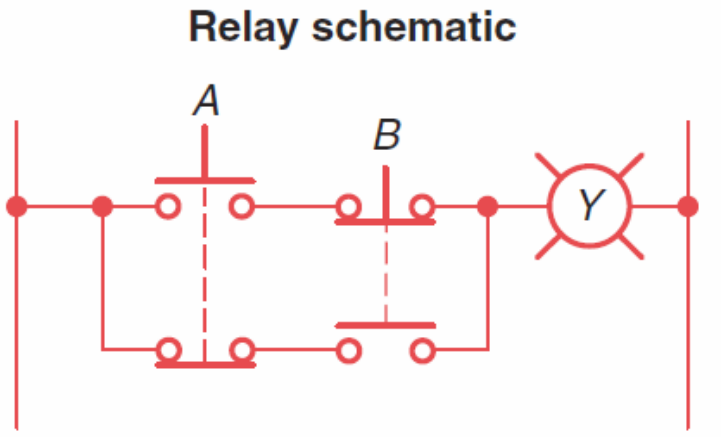


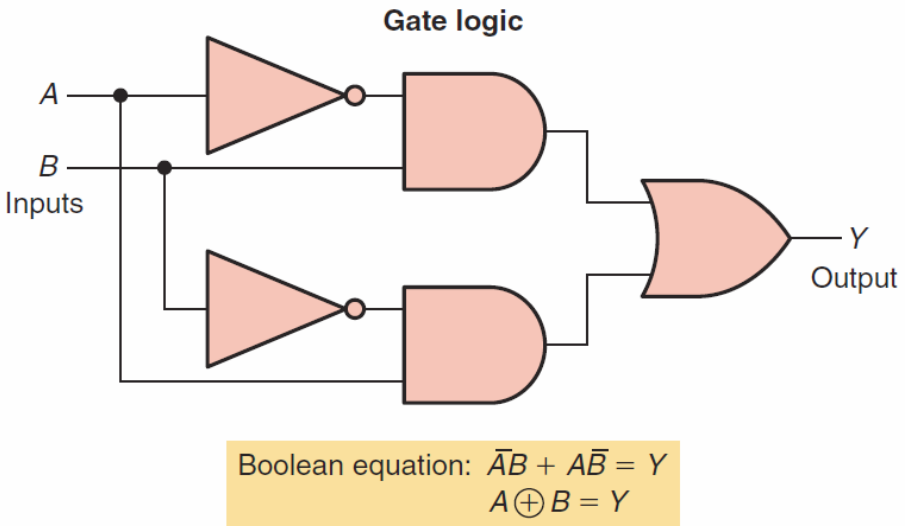


Boolean equation: $\bar{A}B + A\bar{B} = Y$

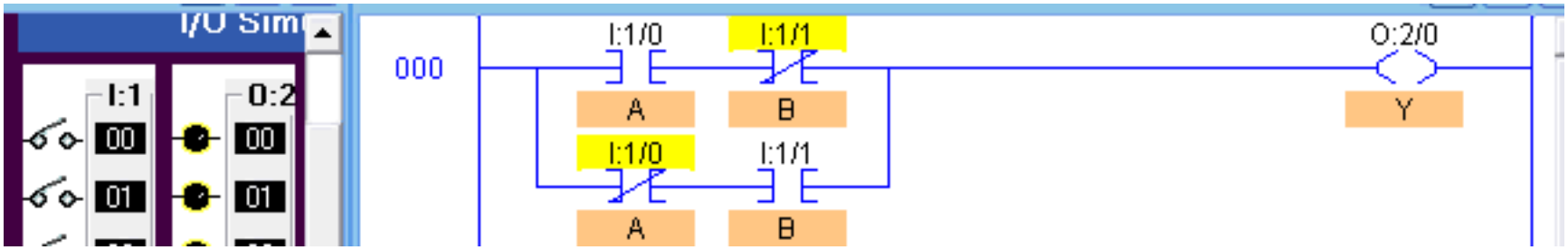
$A \oplus B = Y$

Program simulation of an exclusive OR circuit.



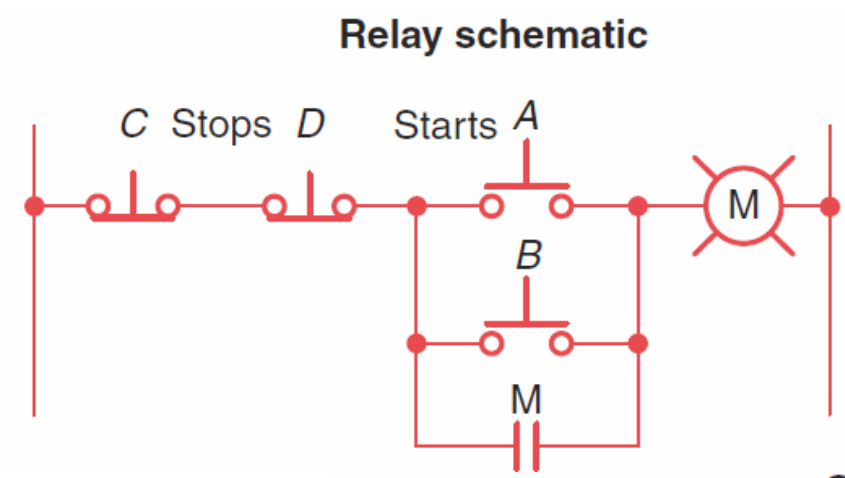


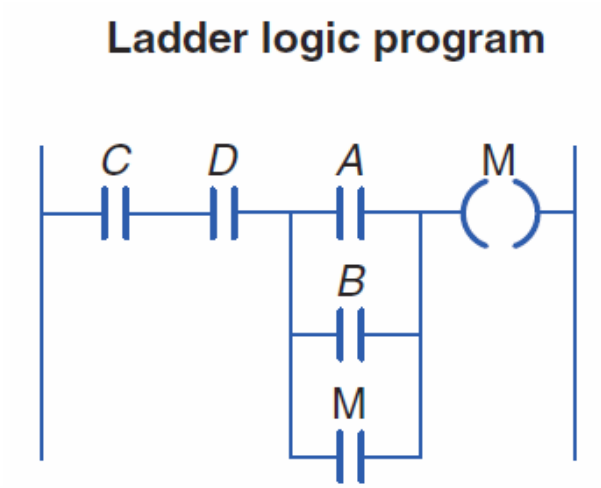
Program

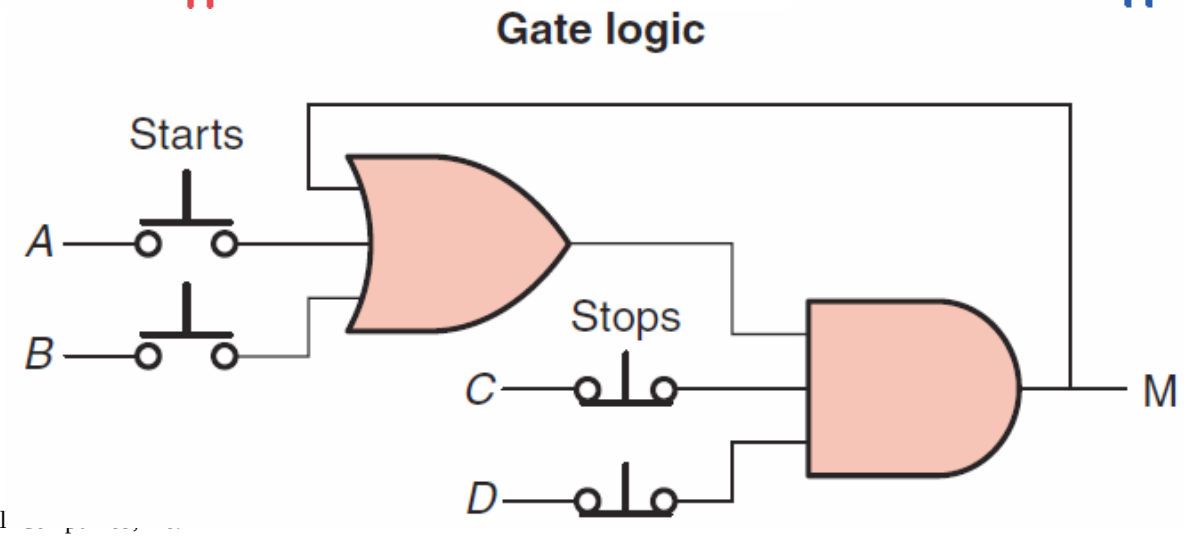


I/O

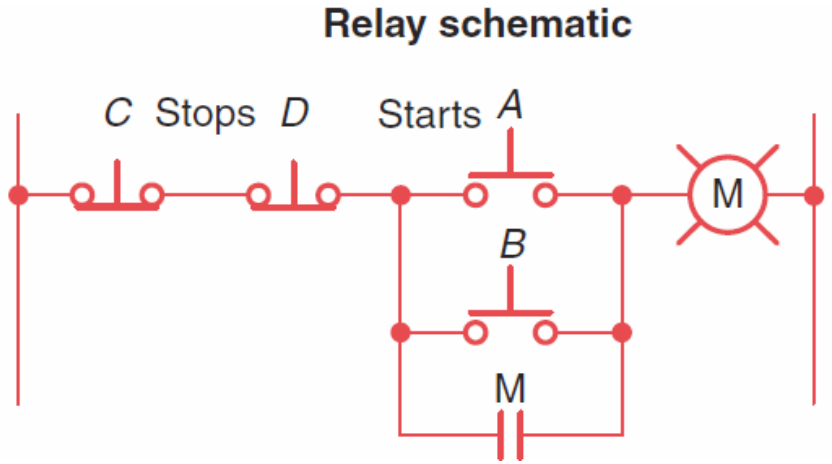
When either start button is depressed, the motor runs. Either stop button stops the motor when it is depressed.

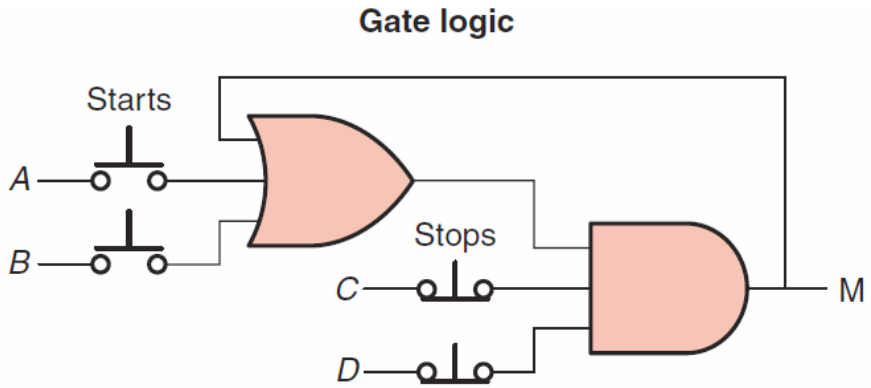




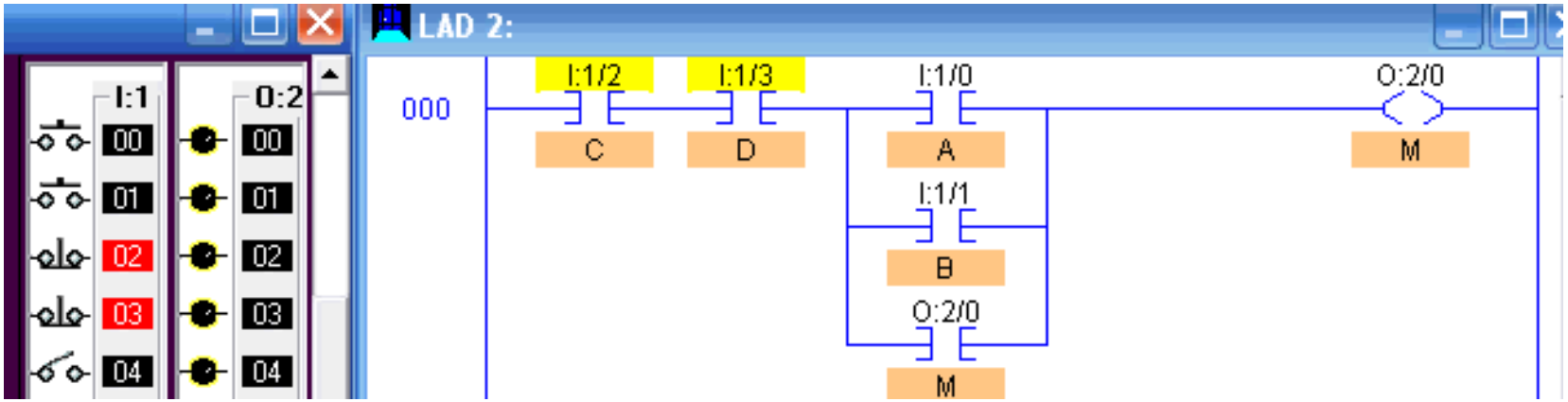


Program simulation of a motor starter circuit.



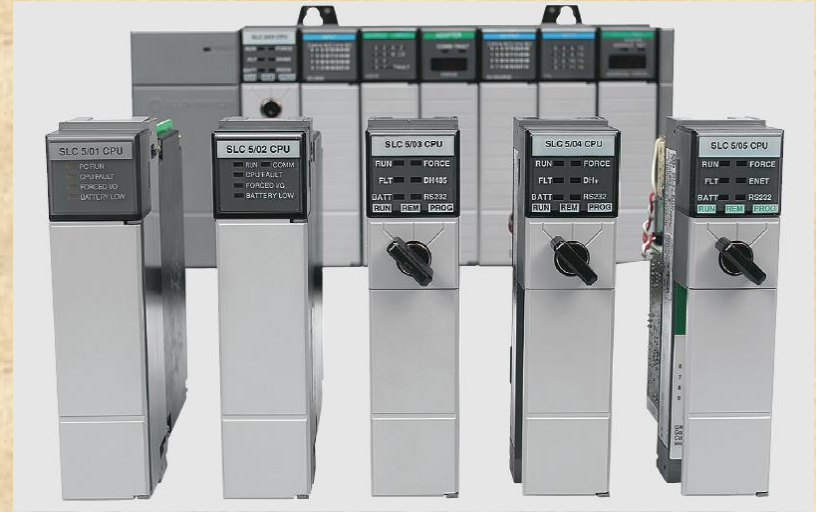


Program



I/O

4.7

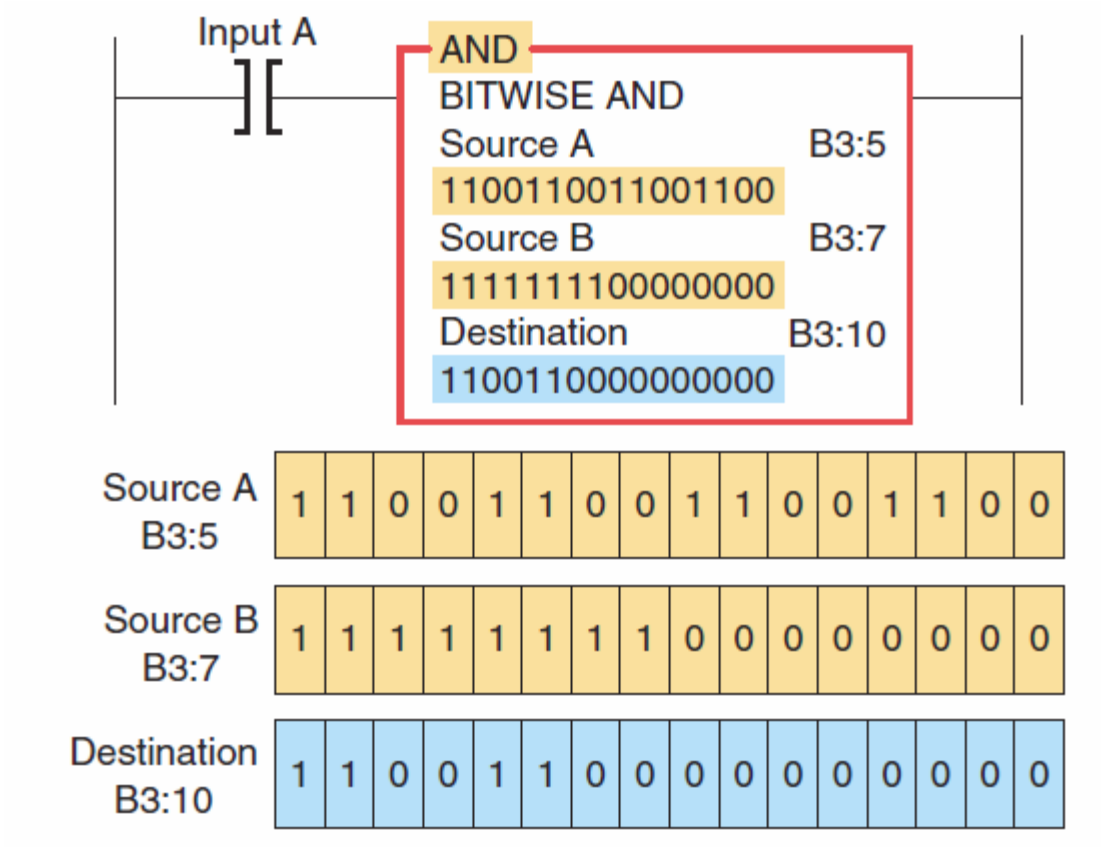


Programming Word Level Logic Instructions

Selecting Word Logic Instructions

If you want to . . .	. . . use this instruction.
Know when matching bits in two different words are both ON	AND
Know when one or both matching bits in two different words are ON	OR
Know when one or the other bit of matching bits in two different words is ON	XOR
Reverse the state of bits in a word	NOT

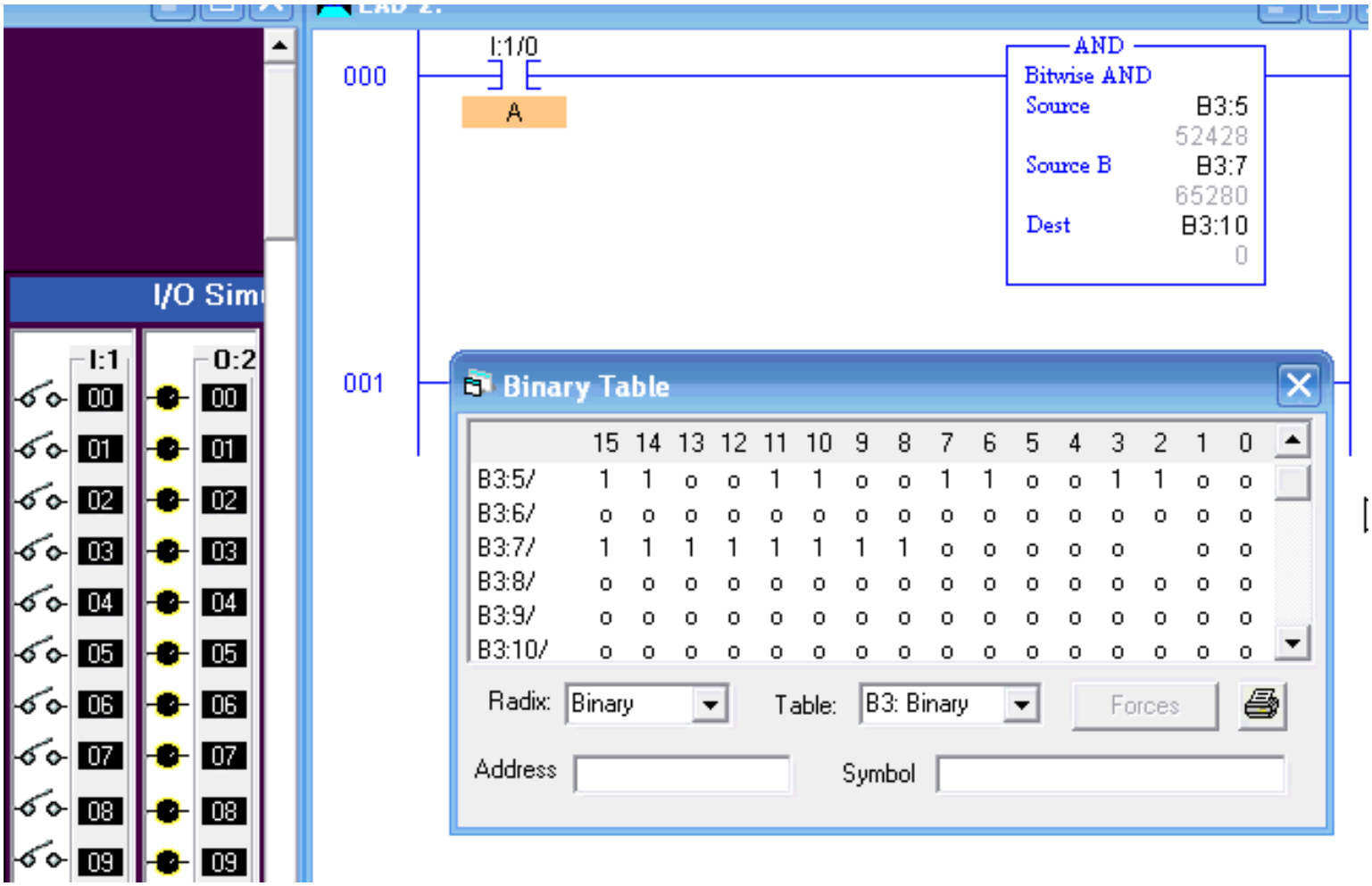
Word-level *AND* instruction.



This instruction tells the processor to perform an AND operation on B3:5 and B3:7 and to store the result in destination B3:10 when input device A is true.

Program simulation of the word level AND instruction.

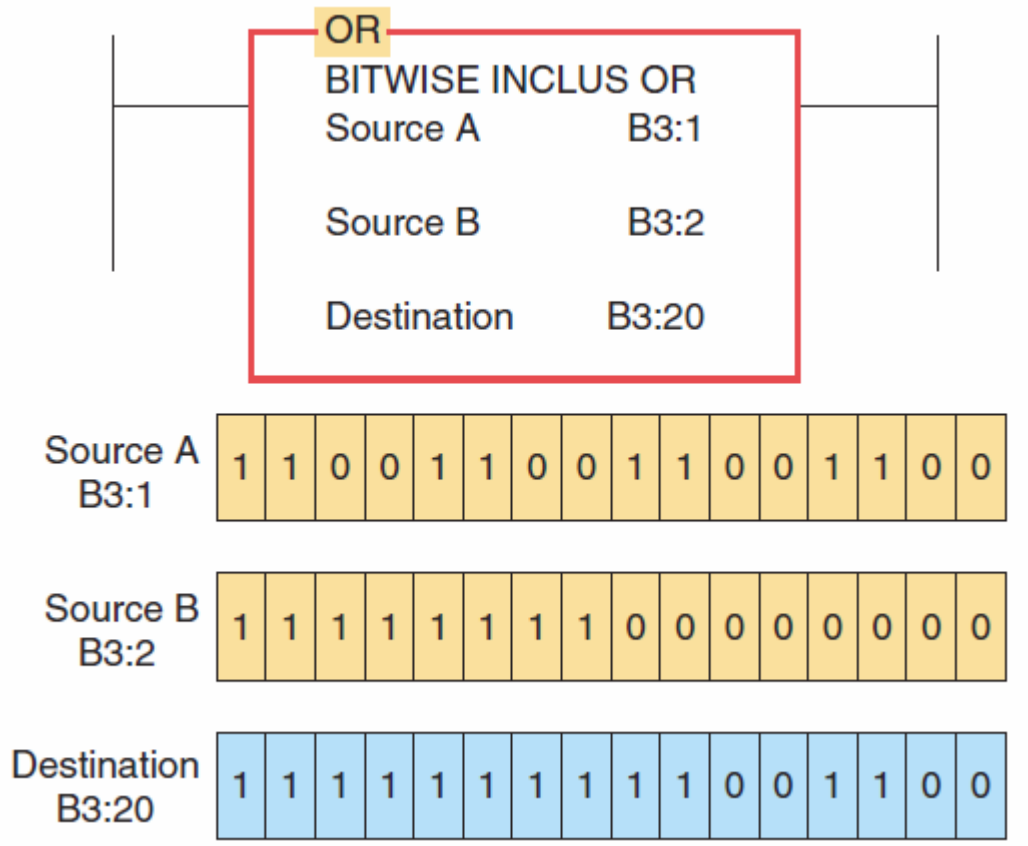
Program



I/O

B3 Data Table

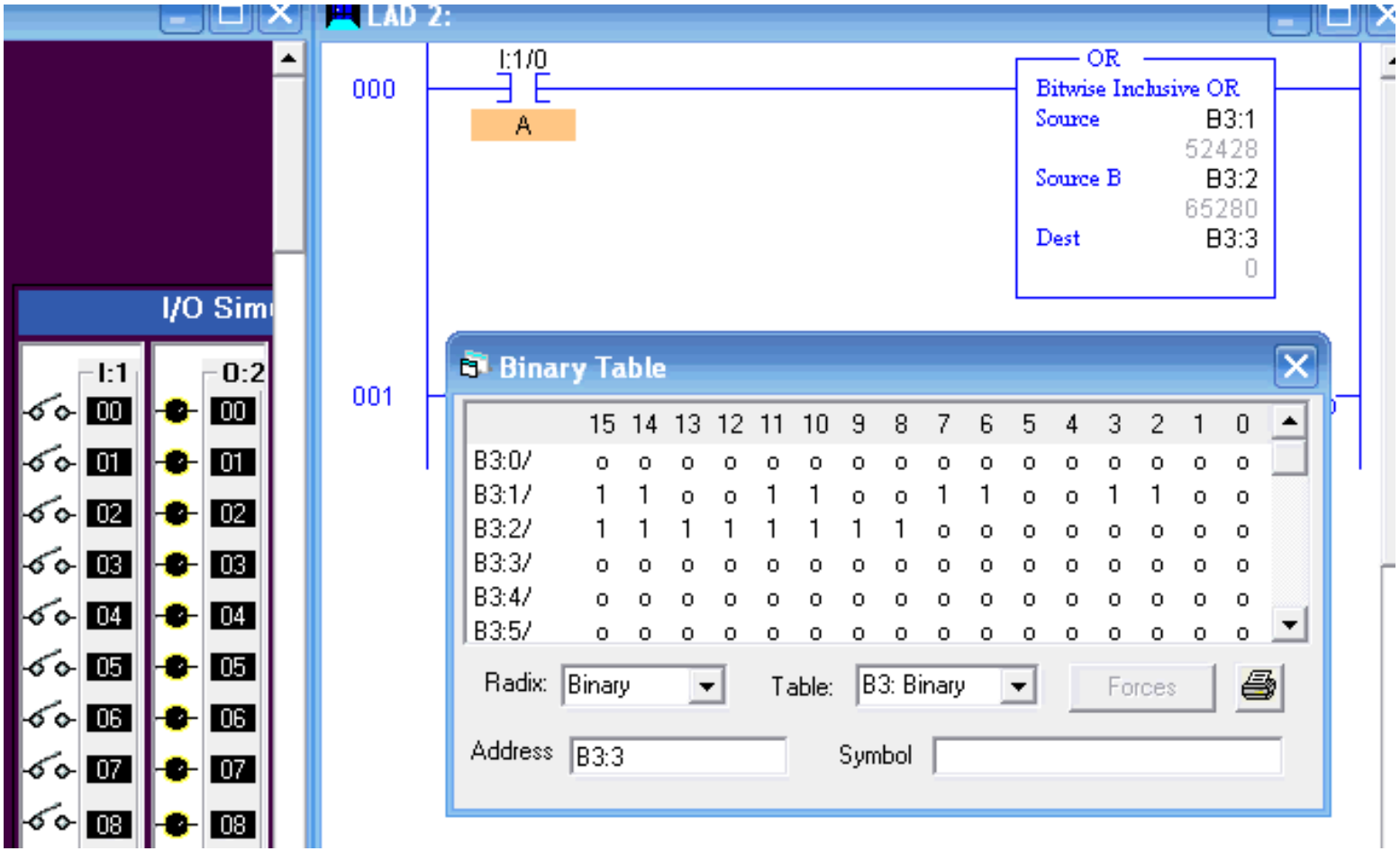
Word-level *OR* instruction.



This instruction ORs the data in Source A, bit by bit, with the data in Source B and stores the result at the destination address.

Program simulation of the word level OR instruction.

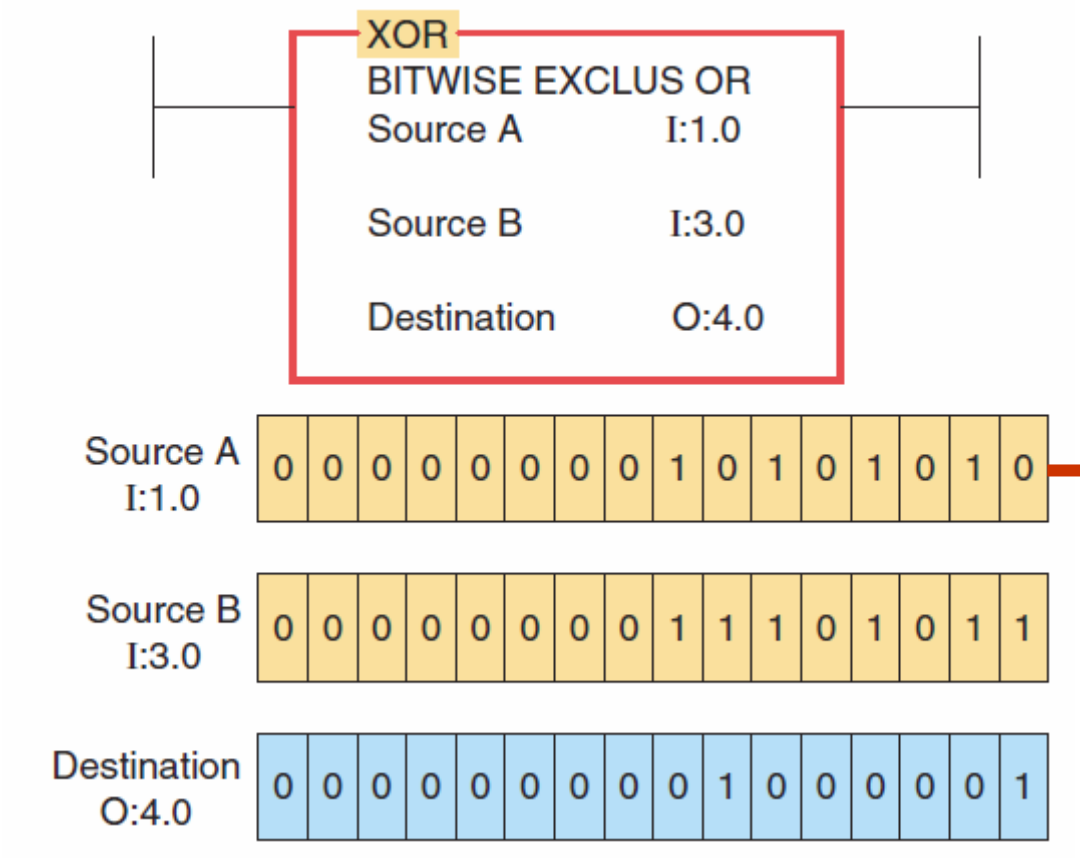
Program

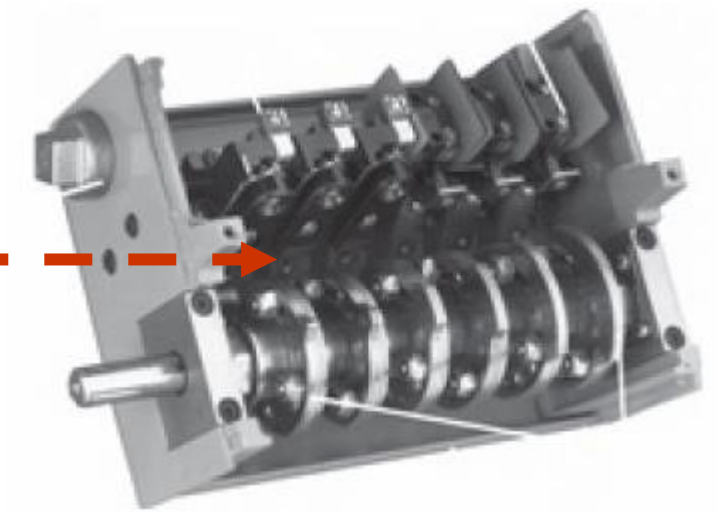


I/O

B3 Data Table

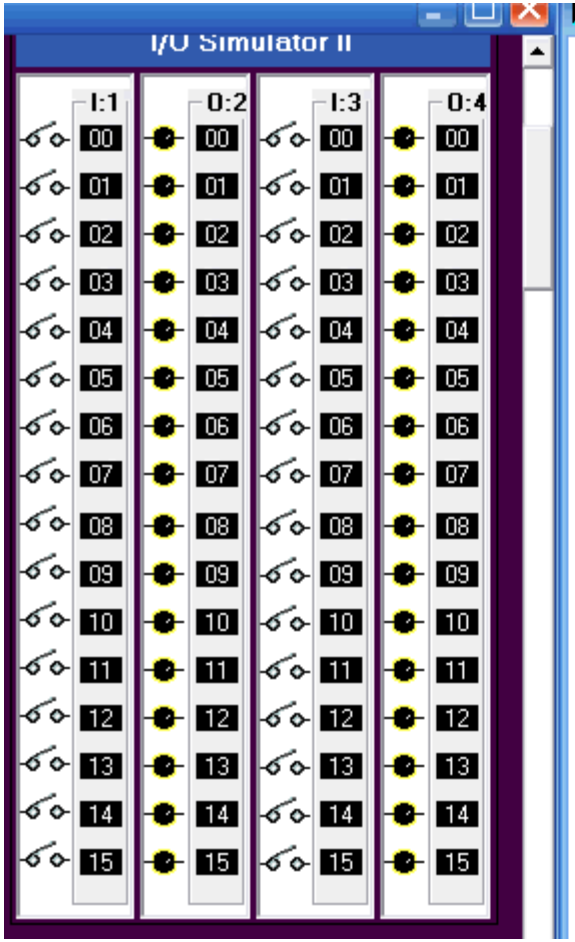
Word-level *XOR* instruction.



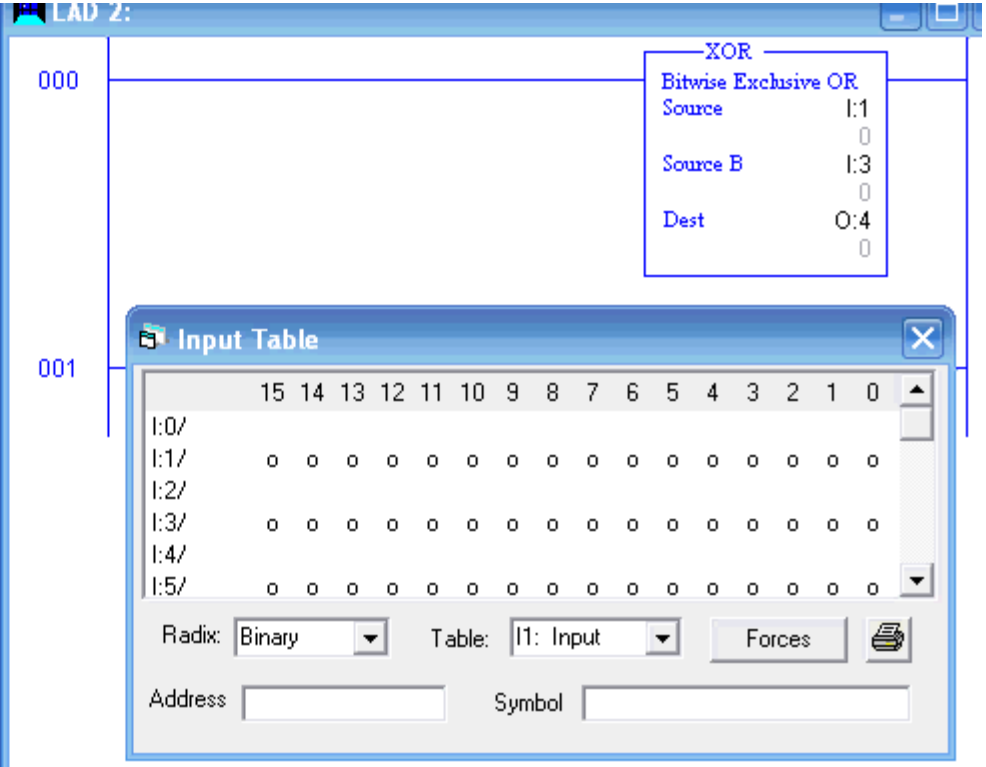


Data from input I:1.0 are compared, bit by bit, with data from input I:3.0. Any mismatches energize the corresponding bit in word O:4.0.

Program simulation of the word level XOR instruction

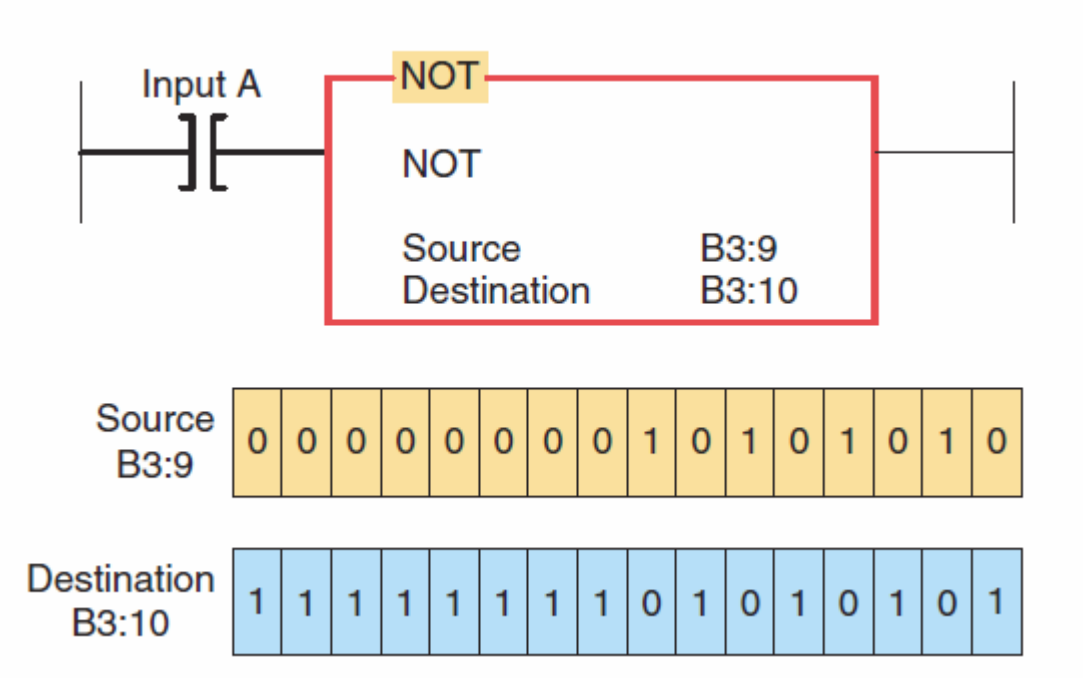


I:1 I:3 O:4



Input Table

Word-level NOT operation.



The bit pattern in B3:10 is the result of the instruction being true and is the inverse of the bit pattern in B3:9.

Program simulation of the word level NOT instruction