



TTE Training Ltd.

**Phase 1
Electrical Course Notes**

E-CN-003



<http://www.tteltd.co.uk>

Logic Theory

Binary Numbers

Introduction

In the **decimal** number system there are ten digits 0 to 9, the true significance of any digit depending on its position in the number: for example -

Thousands Hundreds Tens Units

2 4 3 1

really means :

$$(2 \times 10^3) + (4 \times 10^2) + (3 \times 10^1) + (1 \times 10^0)$$

Remembering that $10^0 = 1$ we see that, counting the units column as No. 0, the tens as No. 1 and so on, the value of a digit is equal to that digit multiplied by ten (called the 'radix' of the system) raised to a power equal to the column number. In general a digit 'a' has a value A given by

$$\mathbf{A = a \times r^n}$$

Where r = the radix (ten in this case)

and n = the column number.

Now imagine that we had only **five** digits available or in other words that the symbols 5, 6, 7, 8 and 9 did not exist. Our counting would have to be done with the only digits known to us, that is 0, 1, 2 3 and 4 so that our number system would have a radix of five.

$$r = 5$$

and the values of 'a' would be restricted to :

$$a = 0, 1, 2, 3 \text{ or } 4$$

This system would be called a 'quinary' system.

Then, for example, digit 2 in column No. 3 has a value

$$2 \times 5^3 = (\text{in decimal equivalent}) 250$$

In digital computer engineering a quinary system is not convenient, for many reasons, (see Practical Aspects in this Theory) to employ devices to represent decimal or even quinary numbers as each device would have to be capable of taking up ten or five distinct states. Instead the number of states is restricted to the smallest that will allow a change, that is two, and these are commonly denoted by digits 0 to 1.

This then is the '**binary**' system whose radix is **two**, hence has only two numerals, (0 and 1). The first five columns in a binary system have decimal values:

$$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16.$$

Thus the binary number **1 0 1 1 0** = decimal $1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = \mathbf{22_{10}}$

The small suffix ten in this result is to indicate that the number is a decimal one. Sometimes confusion can arise otherwise, e.g $1\ 0\ 1_{10} = \text{one hundred and one in decimal}$ but $1\ 0\ 1_2 = 5_{10}$

Thus we should have written above:

$$1\ 0\ 1\ 1\ 0_2 = 22_{10}$$

Binary Decimal Code

The words 'binary digit' are usually abbreviated to '**bit**'. A four-bit number can represent $2^4 = 16$ different numbers but the maximum number is $2^4 - 1 = 15$ because zero is included in the 16 possibilities.

Although computers use binary numbers internally, operators are more accustomed to interpreting decimal numbers.

Thus when numbers have to be fed in via a keyboard or fed out to an electric typewriter or to decimal indicators it is convenient to use a system in which the individual decimal digits are separately represented by binary instead of as a whole. For example :

$$26_{10} = 11010_2 \text{ (standard binary)}$$

$$\text{but} = \overset{2}{\boxed{0010}} \cdot \overset{6}{\boxed{0110}} - \text{BCD}$$

Such a system is called '**binary decimal**' or '**binary-coded decimal**'; **BCD** for short.

Obviously, to be able to represent all digits up to 9 there must be four bits to each decimal column so two groups of four bits can represent all numbers from 0 to 99.

Converting Decimal to Binary

The conversion process from decimal to binary is very simple. By dividing the decimal value by 2, the remaining figure (either 1 or 0) will represent the binary figure. For example, consider the decimal figure of 157 :

157 / 2 = 78, with a remainder of	1
78 / 2 = 39, with a remainder of	0
39 / 2 = 19, with a remainder of	1
19 / 2 = 9, with a remainder of	1
9 / 2 = 4, with a remainder of	1
4 / 2 = 2, with a remainder of	0
2 / 2 = 1, with a remainder of	0
1 / 2 = 0, with a remainder of	1

The binary representation is read from bottom to top hence, for the decimal sum 157, its binary representation is, 10011101

Practical Aspects

In electronic digital computers using integrated circuits the binary states are represented by voltages whose range rarely exceeds 5 volts. Usually Binary 0 is represented by a voltage near zero and Binary 1 by one near +5V.

There is always some variation in these voltages in practice and so the rule is made that any voltage between, say 0V and +0.4V will be called binary 0 and any between say +2.4V and +5V will be called binary 1.

In these circumstances you can see that, even disregarding the problems of setting the nominal values of voltage to more than two levels it would be impossible to maintain them within non-overlapping bounds when the tolerances were accounted for. This is one reason why decimal number systems are not used.

Another is, as we shall see later on in the Binary Addition assignment, that binary arithmetic is very much simpler than decimal.

The BCD system is usually restricted to inputs and outputs but there have been several computers built which use it throughout for all arithmetic and other operations.

Logic Gates

Logic circuits are used extensively in digital electronics. The basic logic gates, AND, OR, NAND and NOR are designed to be interconnected into larger, more complex, circuit arrangements. An understanding of how these circuits operate is an essential pre-requisite to understanding a wide range of digital circuits and systems.

The AND Gate

AND gates will only produce a logic 1 output when all inputs are simultaneously at logic 1. Any other input combination results in a logic 0 output. The Boolean expression for the output, Y, of an AND gate with inputs, **A** and **B**, is :

$$\mathbf{Y = A.B}$$

The operation of a simple AND gate can be illustrated by connecting two switches in series with a battery and a lamp. It should be obvious that the lamp will only operate when both switches (S1 AND S2) are closed*.

The OR Gate

OR gates will produce a logic 1 output whenever any one, or more, inputs are at logic 1. Alternatively, an OR gate will only produce a logic 0 output whenever all of its inputs are simultaneously at logic 0. The Boolean expression for the output, Y, of an OR gate with inputs, **A** and **B**, is :

$$\mathbf{Y = A+B}$$

The operation of a simple OR gate can be illustrated by connecting two switches in parallel and then connecting this arrangement in series with a battery and a lamp. The lamp will operate when either of the two switches (S1 OR S2) are closed*.

***** : in both the AND and OR gates, each switch can only be in one of the two states (i.e., open or closed) at any given time, the open and closed conditions are mutually exclusive. Furthermore, since a switch cannot exist in any other state than completely open or completely closed (i.e., there is no intermediate or half-open state) the circuit is based on binary or 'two-state' logic.

The NAND Gate

NAND (i.e., NOT-AND) gates will only produce a logic 0 output when all inputs are simultaneously at logic 1. Any other input combination will produce a logic 1 output. A NAND gate, therefore, is nothing more than an AND gate with its output inverted! The circle shown at the output of the NAND gate symbol denotes this inversion. The Boolean expression for the output, Y , of a NAND gate with inputs, A and B , is :

$$Y = \overline{A \cdot B}$$

The NOR gate

NOR (i.e., NOT-OR) gates will only produce a logic 1 output when all inputs are simultaneously at logic 0. Any other input combination will produce a logic 0 output. A NOR gate, therefore, is simply an OR gate with its output inverted. The circle shown at the output of the NOR gate symbol denotes this inversion. The Boolean expression for the output, Y , of a NOR gate with inputs, A and B , is :

$$Y = \overline{A + B}$$

Exclusive OR gate

Exclusive-OR gates will produce a logic 1 output whenever either one of the inputs is at logic 1 and the other is at logic 0. Exclusive-OR gates produce a logic 0 output whenever both inputs have the same logical state (i.e., when both are at logic 0 or both are at logic 1). The Boolean expression for the output, Y , of an exclusive-OR gate with inputs, A and B , is:

$$Y = A \cdot \overline{B} + B \cdot \overline{A}$$

Exclusive NOR gate

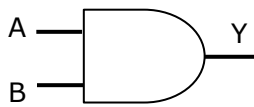
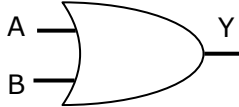
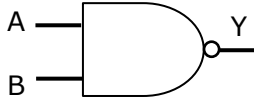
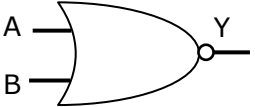
Exclusive-NOR gates will produce a logic 0 output whenever either one of the inputs is at logic 1 and the other is at logic 0. Exclusive-NOR gates produce a logic 1 output whenever both inputs have the same logical state (i.e., when both are at logic 0 or both are at logic 1). The Boolean expression for the output, Y , of an exclusive-NOR gate with inputs, A and B , is:

$$Y = \overline{A \cdot \overline{B} + B \cdot \overline{A}}$$

Buffers

Buffers do not affect the logical state of a digital signal, i.e., a logic 1 input results in a logic 1 output whereas a logic 0 input results in a logic 0 output. Buffers are normally used to provide extra current drive at the output but can also be used to regularise the logic levels present at an interface. The Boolean expression for the output, Y , of a buffer with an input, X , is : $Y = X$

Basic Logic Functions

AND	OR	NAND	NOR																																																												
Y is true if A and B are true	Y is true if A or B is true	Y is false if A and B are true	Y is false if A or B is true																																																												
																																																															
$Y = A.B$	$Y = A+B$	$Y = A.B$	$Y = A+B$																																																												
<table><tr><th>A</th><th>B</th><th>Y</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>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>A</th><th>B</th><th>Y</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>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>A</th><th>B</th><th>Y</th></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>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>A</th><th>B</th><th>Y</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>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y																																																													
0	0	0																																																													
0	1	0																																																													
1	0	0																																																													
1	1	1																																																													
A	B	Y																																																													
0	0	0																																																													
0	1	1																																																													
1	0	1																																																													
1	1	1																																																													
A	B	Y																																																													
0	0	1																																																													
0	1	1																																																													
1	0	1																																																													
1	1	0																																																													
A	B	Y																																																													
0	0	1																																																													
0	1	0																																																													
1	0	0																																																													
1	1	0																																																													

The NAND gate is almost universally used in integrated circuit logic although all types of operation can be obtained. Design tools and techniques are better adapted to the use of the three fundamental operations of OR, AND, NOT, hence methods of converting the resulting expressions to the exclusive use of NAND have to be used.

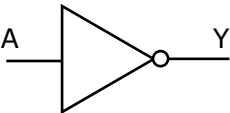
Inverters

Inverters are used to complement the logical state (i.e., a logic 1 input results in a logic 0 output and vice versa). Inverters also provide extra current drive and, like buffers, are used in interfacing applications where they provide a means of regularising logic levels present at the input or output of a digital system. The Boolean expression for the output, **Y**, of an inverter with an input, A, is : $Y = \overline{A}$

NOT

Y is true if A is false

$Y = \text{NOT } A$



A	Y
1	0
0	1

Basic Logic Operations

The first chapter demonstrated that binary numbers comprise of a collection of binary digits, or bits, each of which can be either 0 or 1.

These values can be represented in hardware by various means including toggle switch positions and lamp states. Whilst these can be used to represent the binary variables, they can also be used to represent quite different things. For example, 1 and 0 can represent any of the pairs of opposites shown in the following table :

"1"	"0"
TRUE	FALSE
IN	OUT
UP	DOWN
YES	NO
WET	DRY

In general, binary variables can represent any pair of concepts in which the existence of one implies the non-existence of the other. Notice carefully the idea of "opposite-ness" or "NOT-ness".

"1"	is NOT	"0"
FALSE	is NOT	TRUE
TRUE	is NOT	FALSE
UP	is NOT	DOWN
"0"	is NOT	"1"
etc, etc		

The ideas of true and false have a particularly close association historically with the development of algebraic methods of dealing with binary variables, the earliest application being the study of formal logic. This is why computer circuits are referred to as logic elements and why we use the term "truth table".

The algebra that is used to describe the various operations is called "Boolean Algebra" within which the variables are denoted by letters, e.g. A, B, C, etc., or X, Y, Z. The difference is that whereas in ordinary algebra A could be any value at all, in Boolean Algebra it can only be either 0 or 1.

$$A = 0 \text{ or } A = 1$$

The operator of NOT can be applied to a single variable as : NOT A. This is called the "complement" or "negation" of A and is usually symbolised by a bar over the variable.

$$\text{NOT } A = \overline{A}$$

The Truth Table for NOT can now be written in a more general fashion, hence :

A	\overline{A}
0	1
1	0

If we consider applying this to applications with more than one variable, for example, A and B, the Truth Table for the NAND function will be :

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

By associating 1 with true and 0 with false, Z could be described as being true when it was not true that A and B were both 1, or, put more simply ;

$$Z = \text{NOT (A and B)}$$

NOT-AND is usually abbreviated to NAND so we have

$$Z = A \text{ NAND } B$$

Even this abbreviated notation is not very convenient and hence the notation $A.B$ is adopted to mean A AND B, and may be written as :

$$Y = \text{NOT } A.B = \overline{A.B}$$

Using the bar notation. When no ambiguity is possible, the dot is frequently dropped to give just AB.

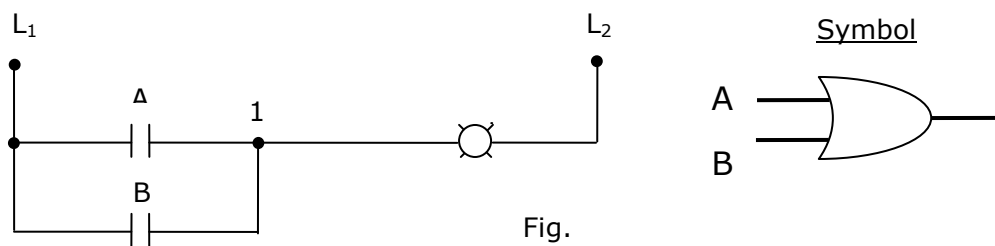
The Truth Table for AND provides some important basic identities that assists the manipulation of Boolean Expressions.

0.0	0
0.1	0
1.0	0
1.1	1
0.1	1.0
	0

It is clear from the table that AB and BA are the same thing. An expression like this is called a **"logical product"** because of its similarity to arithmetic multiplication.

Digital Logic Functions

Simple logic functions can be constructed using a hypothetical lamp circuit. Using standard binary notation for the status of the switches and lamp (0 for "off" and 1 for "on"), a truth table can be made to show how logic works.

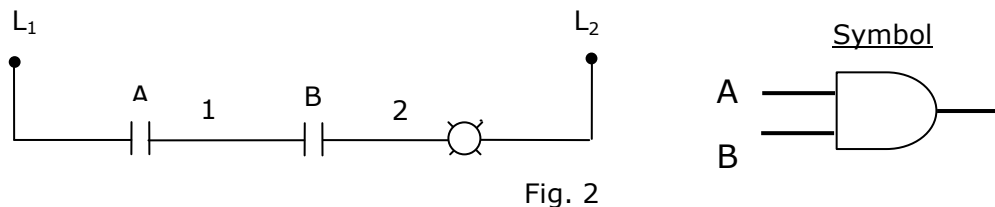


In Fig. 1 the lamp will come on if either contact A OR contact B is actuated.

Truth Table - OR Gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

The AND logic function can be mimicked by connecting the two contacts in series instead of parallel (see Fig. 2)

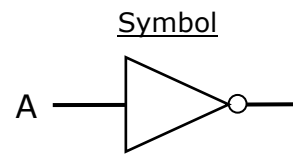
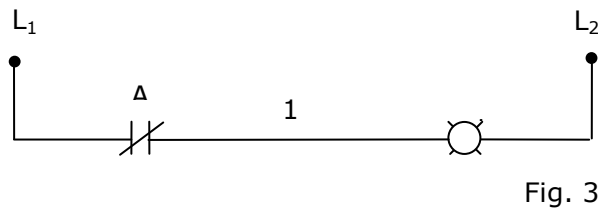


The lamp will come on only if contacts A AND B are actuated.

Truth Table - AND Gate

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

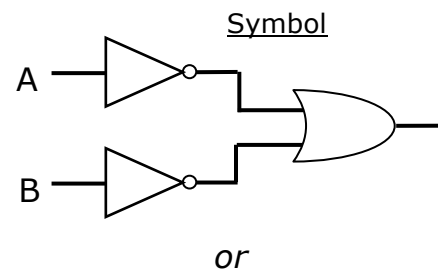
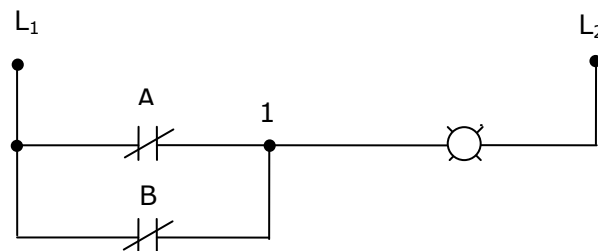
The logical inversion, or NOT, function can be performed on a contact input by using a normally-closed contact instead of a normally-open contact.



A	Output
0	1
1	0

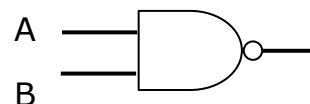
In Fig. 3 the lamp will come on if the contact is *NOT* actuated, and switches off when the contact is actuated.

If we take the OR function and invert each "input" through the use of normally-closed contacts, we will end up with a NAND function. In Boolean Algebra, this effect of gate function identity changing with the inversion of input signals is described by DeMorgan's Theorem (described in the following pages)



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Fig.



In Fig. 4 the lamp will come on if either contact A OR contact B is un-actuated. It will only go off if both contacts are actuated simultaneously.

Likewise, if we take our AND function and invert each “input” through the use of normally-closed contacts, the result will be a NOR function :

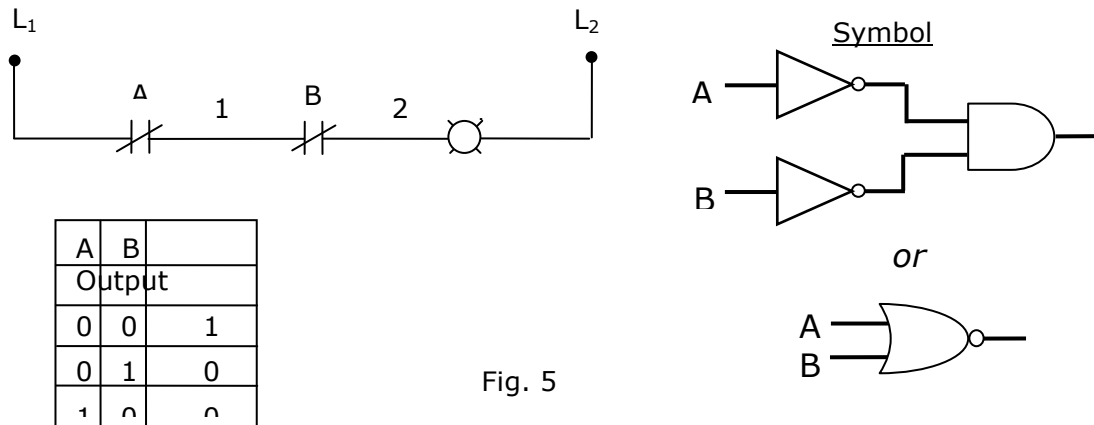


Fig. 5

When ladder circuits are compared with their logic gate counterparts, the following is revealed:

- parallel contacts are equivalent to an OR gate
- series contacts are equivalent to an AND gate
- normally-closed contacts are equivalent to a NOT gate

Combination logic functions can be also built by grouping contacts in series-parallel arrangements. The following examples demonstrate an EXCLUSIVE-OR function constructed from a combination of AND, OR and inverter (NOT) gates :

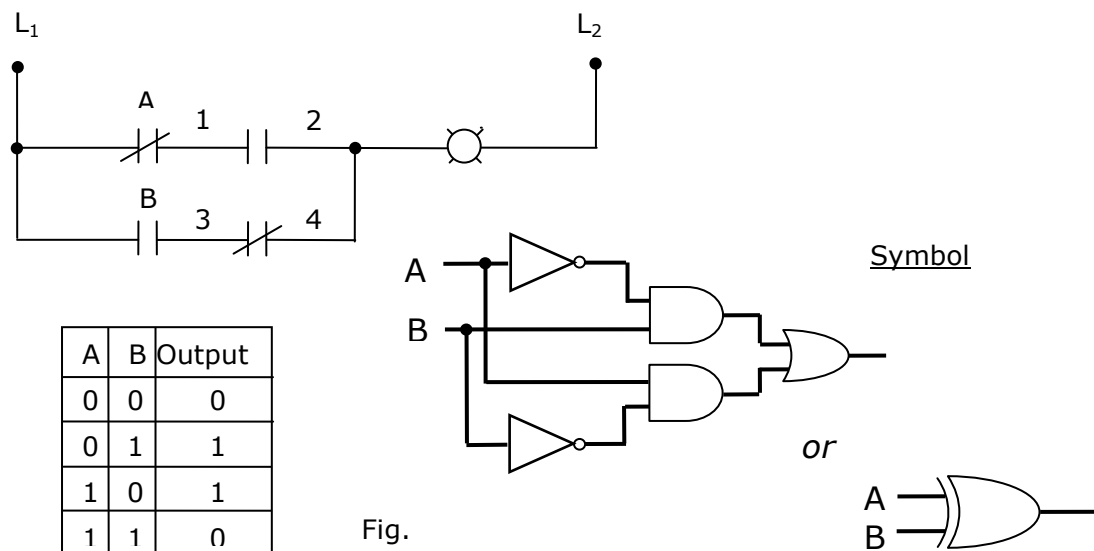


Fig.

In Fig. 6 (previous page) :

- the top rung (NC contact A in series with NO contact B) is the equivalent of the top NOT/AND gate combination
- the bottom rung (NO contact A in series with NC contact B) is the equivalent of the bottom NOT/AND gate combination.
- the parallel connection between the two rungs at wire no. 2 forms the equivalent of the OR gate, by allowing the output from either the top OR bottom rungs to energise the lamp

To make the EXCLUSIVE-OR function, two contacts per input are used, one for the direct input and the other for the "inverted" input. The two "A" contacts are physically actuated by the same mechanism, as are the two "B" contacts. The common association between the contacts is denoted by the label of the contact. There is no limit to how many contacts per switch can be represented in a ladder diagram as each new contact (either normally-open or normally-closed) on any switch or relay used in the diagram is marked with the same label.

Occasionally multiple contacts on a single switch or relay are designated by a compound-label, such as "A-1" and "A-2" (instead of two "A" labels) in order to identify which set of contacts is being used for a particular part of the circuit.

The *output* of any switch-generated logic function may be inverted by using a relay with a normally-closed contact. For example, energising a load based on the inverse, or NOT, of a normally-open contact, (see Fig. 7 below).

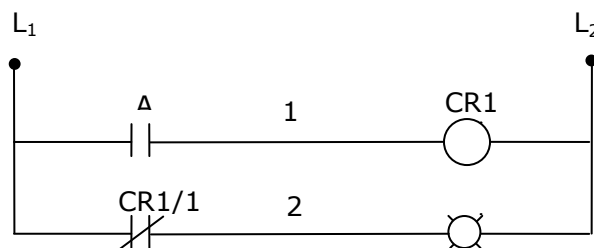


Fig. 7

Symbol		
A	CR1	
Output		
0	0	1

When the coil of the control relay, CR1, is energised, the contact on the second rung opens hence de-energising the lamp. From switch A to the coil CR1, the function is non-inverted. The normally-closed contact, CR1/1, (actuated by CR1), provides a logical inverter function to drive the lamp opposite to that of the switch actuation status.

By applying this inversion strategy to one of the inverted-input functions, such as the OR-to-NAND, the output can be inverted with a relay to create a non-inverted function, (see Fig. 8).

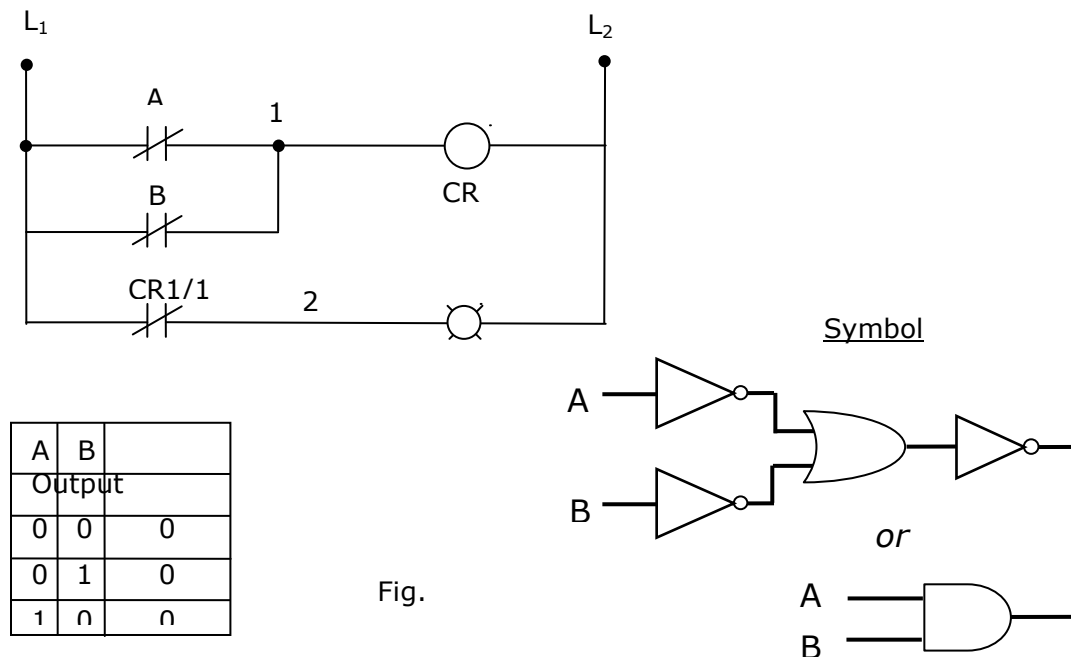


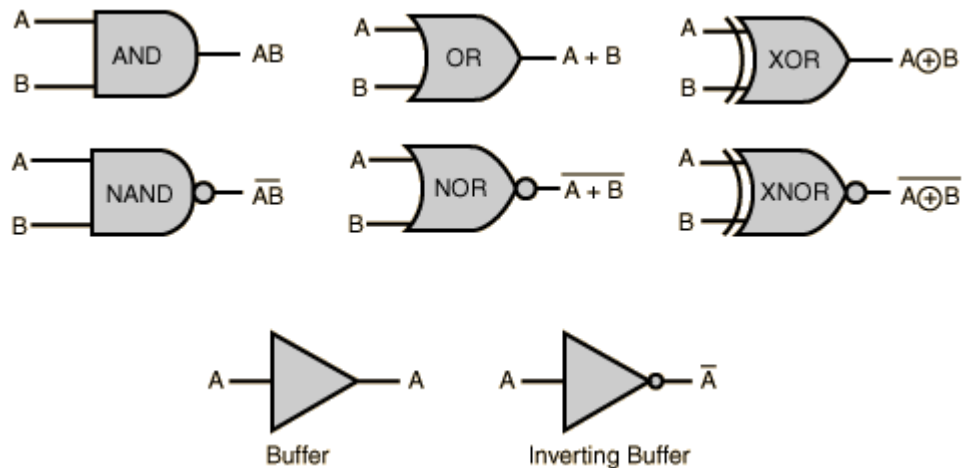
Fig.

From the switches to the coil CR1, the logical function is that of a NAND gate. The normally-closed contact, CR1/1, provides one final inversion to turn the NAND function into an AND function.

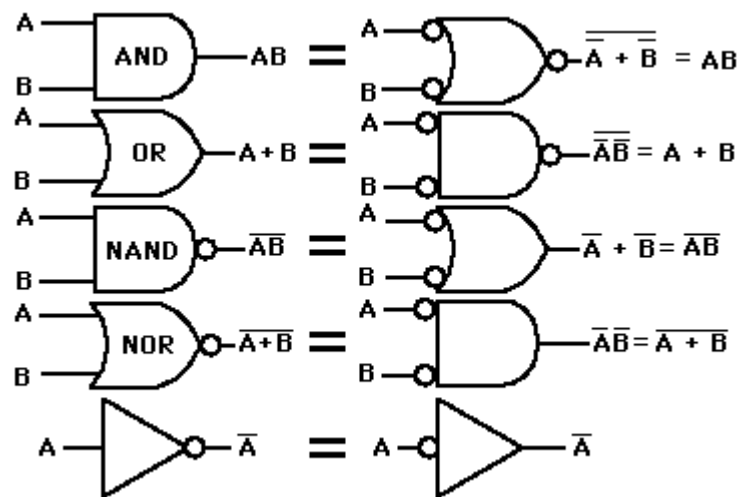
In summary :

- parallel contacts are logically equivalent to an OR gate
- series contacts are logically equivalent to an AND gate
- normally-closed contacts (NC) are logically equivalent to a NOT gate
- a relay must be used to invert the *output* of a logic gate function, whilst normally-closed switch contacts are sufficient to represent inverted gate *inputs*

Basic Logic Gates



Negative Logic Gates



Each of the basic [gates](#) has a negative logic equivalent as shown. The equivalence is shown by the application of [DeMorgan's theorem](#). It amounts to changing AND's to OR's or vice versa and inverting all input and output lines compared to the implementation in gates shown at left.

DeMorgan's Theorem

The most important logic theorem for digital electronics, this theorem says that any logical binary expression remains unchanged if we

1. Change all variables to their complements.
2. Change all AND operations to OR.
3. Change all OR operations to AND.
4. Take the complement of the entire expression.

For example :

$$\overline{A + B} = \bar{A} \cdot \bar{B} \quad \& \quad \overline{A \cdot B} = \bar{A} + \bar{B}$$

$$\overline{A \cdot B \cdot C \text{ etc}} = \bar{A} + \bar{B} + \bar{C} \text{ etc} \quad \& \quad \overline{A + B + C \text{ etc}} = \bar{A} \cdot \bar{B} \cdot \bar{C} \text{ etc}$$

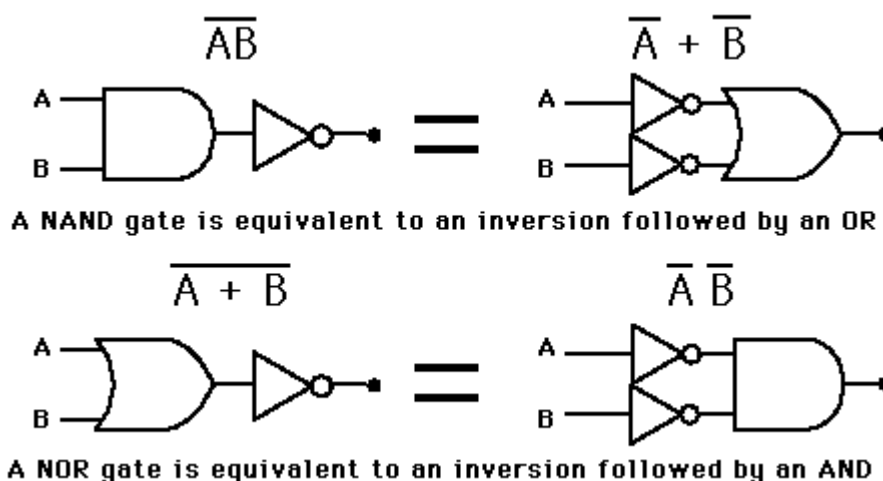
$$\overline{\bar{A} + B} = \bar{\bar{A}} \cdot \bar{B} = A \cdot \bar{B}$$

$$\overline{A \cdot \bar{B} \cdot \bar{C}} = \bar{A} + \bar{\bar{B}} + \bar{\bar{C}} = \bar{A} + B + C$$

A practical operational way to look at DeMorgan's Theorem is that the inversion bar of an expression may be broken at any point and the operation at that point replaced by its opposite (i.e., AND replaced by OR, or vice versa).

DeMorgan's Theorem in Gates

Two forms of DeMorgan's Theorem implemented with basic gates.

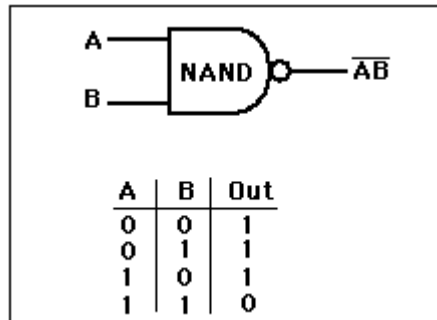


DeMorgan Applications

DeMorgan's Theorem is useful in the implementation of the basic gate operations with alternative gates, particularly with NAND and NOR gates which are readily available in Integrated Circuit (IC) form.

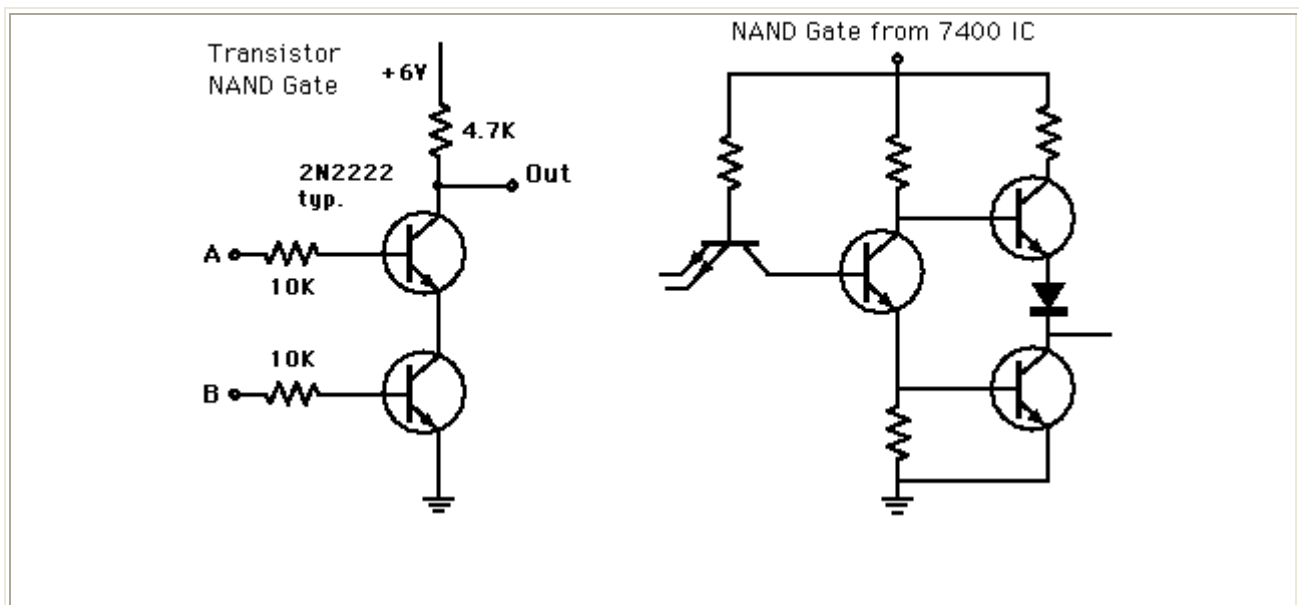
The NAND Gate

The output is high when either of inputs A or B is high, or if neither is high. In other words, it is normally high, going low only if both A and B are high.

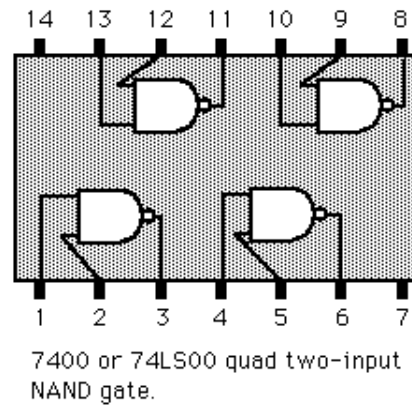
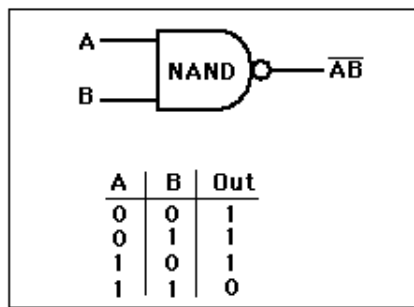


The NAND gate and the NOR gate can be said to be universal gates since combinations of them can be used to accomplish any of the basic operations and can thus produce an inverter, an OR gate or an AND gate. The non-inverting gates do not have this versatility since they can't produce an invert.

Making a NAND Gate

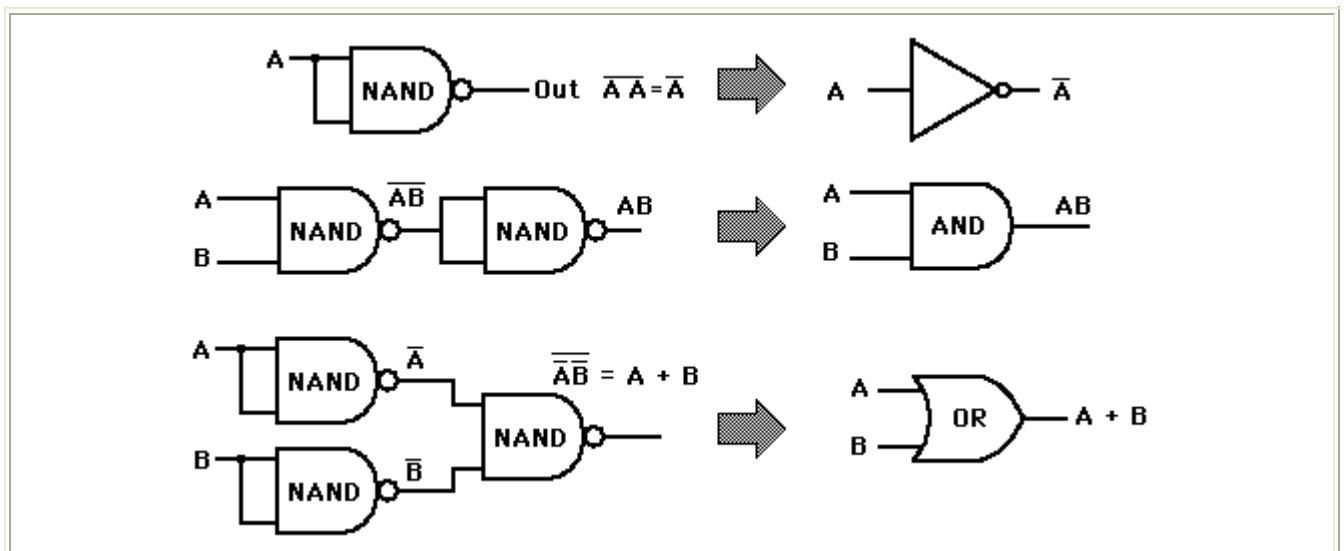


Integrated Circuit (IC) 7400 Quad NAND Gate



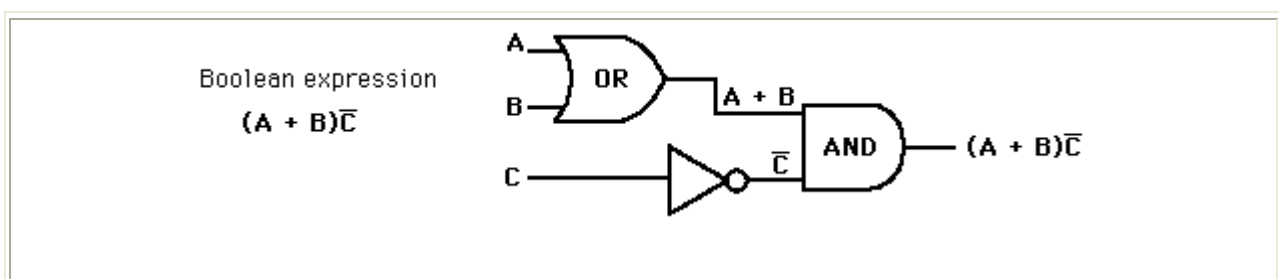
NAND Gate Operations

The NAND gate is called a universal gate because combinations of it can be used to accomplish all the basic functions.

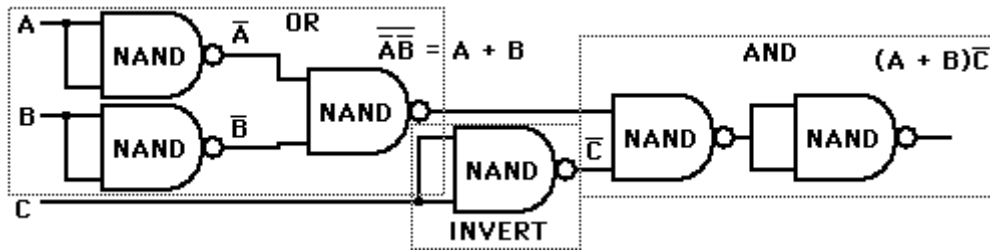


NAND Gate Application

Suppose you want a high output when either A or B is high but C is low. The Boolean expression and straightforward gate version of this are :



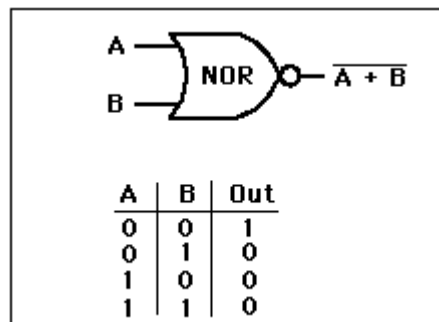
But the same task can be accomplished with NAND gates only since NAND's are universal gates. Integrated circuits such as the 7400 make this practical.



NOR Gate

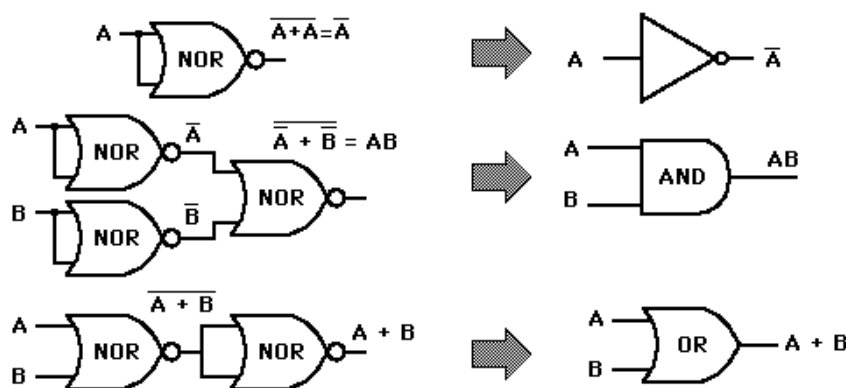
The output is high only when neither A nor B is high. That is, it is normally high but any kind of non-zero input will take it low.

The NOR gate and the NAND gate can be said to be universal gates since combinations of them can be used to accomplish any of the basic operations and can thus produce an inverter, an OR gate or an AND gate. The non-inverting gates do not have this versatility since they can't produce an invert.

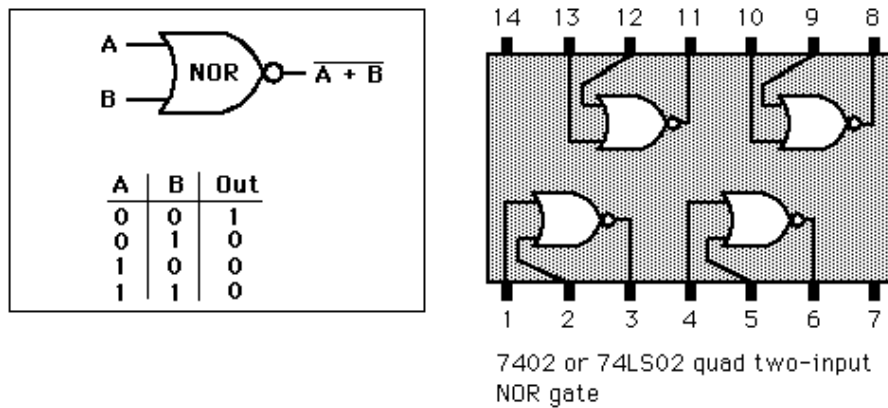


NOR Gate Operations

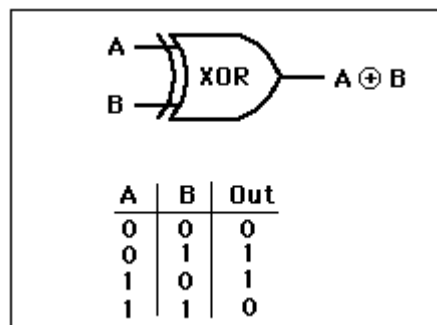
The NOR gate is called a universal gate because combinations of it can be used to accomplish all the basic functions.



Integrated Circuit (IC) 7402 Quad NOR Gate



Exclusive OR Gate



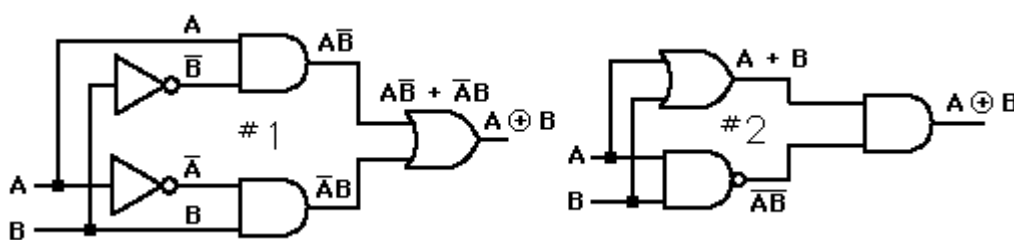
The output is high when either of inputs A or B is high, but not if both A and B are high.

Logically, the exclusive OR (XOR) operation can be seen as either of the following operations :

1. $A \oplus B = A\bar{B} + \bar{A}B$ A AND NOT B OR B AND NOT A

2. $A \oplus B = (A + B)(\overline{AB})$ A OR B AND NOT A AND B

which can be implemented by the gate arrangements shown. They can also be implemented using NAND gates only.

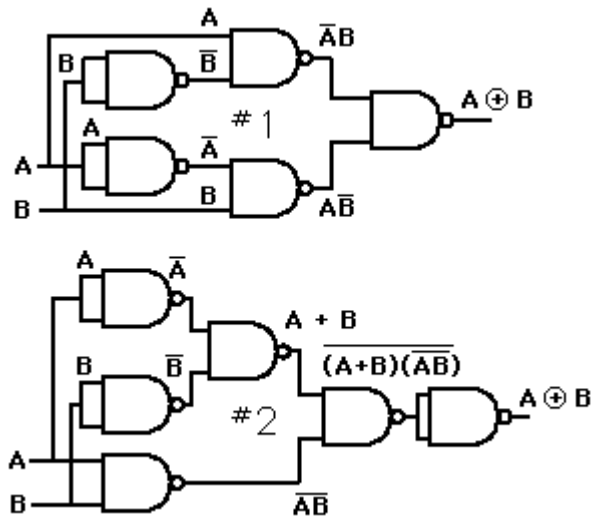


Exclusive OR with NAND

The implementation of the exclusive OR (XOR) operation with just NAND gates illustrates the function of NANDs as universal gates.

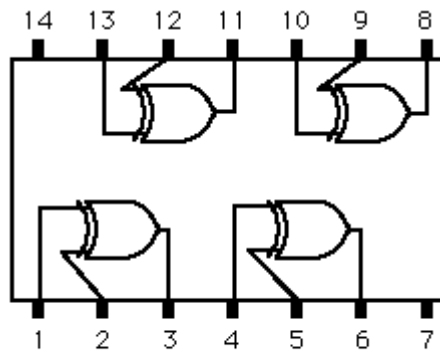
1. $A \oplus B = \overline{A}B + A\overline{B}$ (A AND NOT B OR B AND NOT A)

2. $A \oplus B = (A + B)(\overline{AB})$ (A OR B AND NOT A AND B)



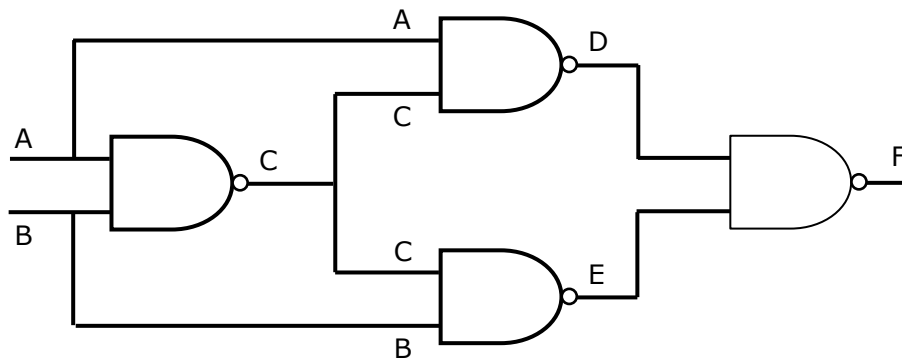
Integrated Circuit (IC) 7486 Exclusive-OR

This is an example of convenient packaging of XOR gates in integrated circuit form.



Exercise

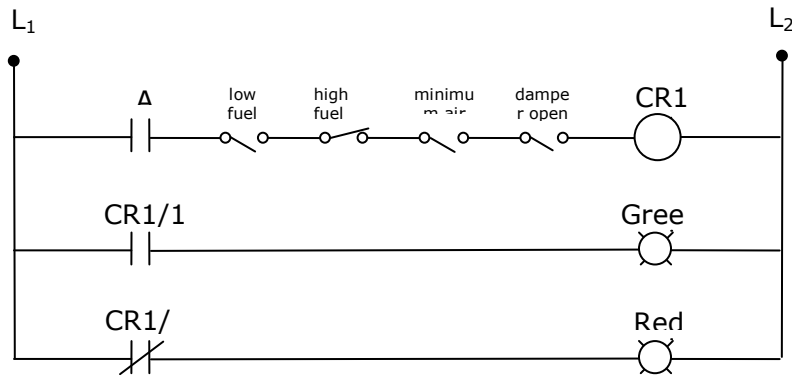
Study the diagram below and complete a truth table for all inputs / outputs



A	B	C	D	E	F
0	0				
0	1				
1	0				
1	1				

Permissive and Interlock Systems

A practical application of switch and relay logic may be found in process control systems where several conditions may have to be met before equipment can be operated. For example, in order for combustion furnaces to operate safely the burners have to be controlled. The control system requires “permissions” from several process switches, such as high and low fuel pressure, exhaust stack damper position, access door position etc. Each process condition is called a permissive and each permissive switch contact is connected in series so that if any one of them detects an unsafe condition, the circuit will be opened :



Green light : conditions met – safe to start

Red light : conditions not met – unsafe to start

Fig. 9

If all conditions are met, CR1 will energise and the green indicator lamp will be lit. In real life the circuit would consist of numerous indicators and the permissive switches would control a main control relay or fuel valve solenoid. If any of the permissive conditions were not met, the series connected contacts will be broken, CR1 will de-energise and the red indicator will light.

Note that the high-fuel pressure contact is normally-closed – this is because the contact opens when the fuel pressure is excessive (too high). The “normal” condition of any pressure switch is when zero (low) pressure is being applied to it hence this circuit uses a switch contact that is normally-closed and opens with high pressure.

Another practical application of relay logic is in electrical control systems where we want to ensure two incompatible events cannot occur at the same time. For example, in a reversible motor control two contactors are connected to switch the polarity of the motor however only one contactor can be energised at any one time, (see Fig. 10).

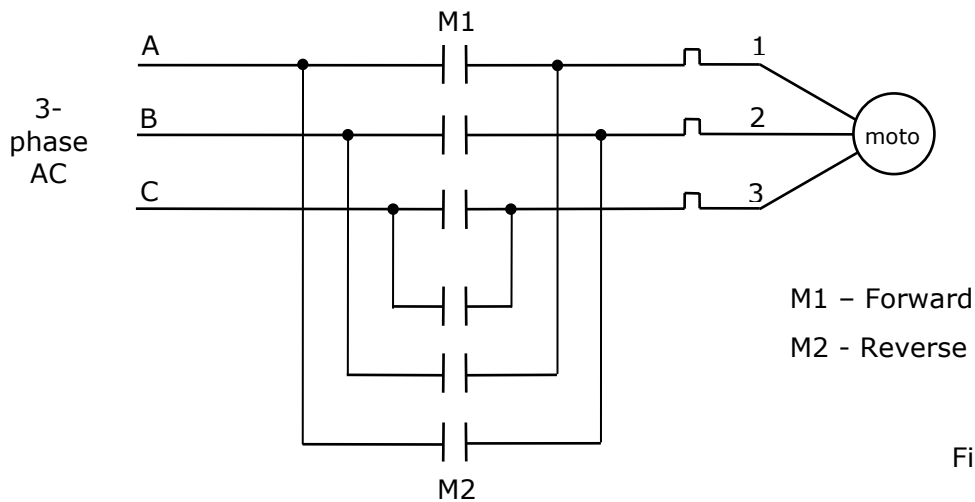


Fig. 10

When contactor M1 is energised, the 3 phases (A, B and C), are connected directly to the terminals 1, 2 and 3 of the motor. When contactor M2 is energised, phases A and B are reversed, A going to motor terminal 2 and B going to motor terminal 1. This reversal of phase connections results in the motors rotating in the opposite direction. The control for the two contactors is as follows (Fig. 11)

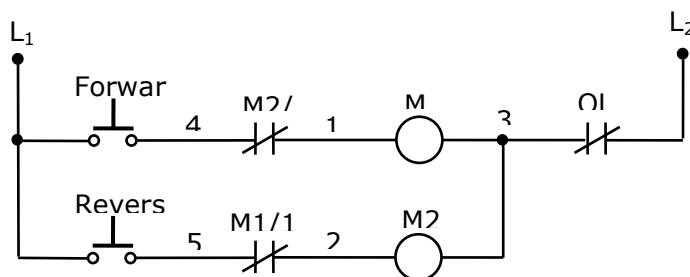


Fig.

* the normally-closed "OL" contact is the thermal overload contact operated by the overload "heater" elements connected in series with each phase of the AC motor. If the heaters get too hot the contact will change from its closed state to an open state which will then prevent either contactor from energising.

To prevent both contactors energising at the same time the circuit is designed by connecting a normally-closed auxiliary contact from the opposing relay in series with the coil. For example, when relay M1 energises, its aux. contact (M1/1) opens hence preventing relay M2 from energising at the same time. This is known as *interlocking*.