MEC 3102 – PRODUCTION ENGINEERING I AND ELECTRICITY & ELECTRONICS II

Department of Mechanical Engineering

The University of Zambia

Name: Mr. Munkombwe Boyd,

Old School of Engineering Blg.,
Cell: 0968315273/0950435239
Email: boyd.munkombwe@unza.zm

2nd Series Lecture 1[3]

LOGIC GATES

Introduction

- To process digital information we use special electronic components that respond to binary signals.
- To design efficient digital circuits, we also need a special numbering system and a special type of algebra.
- Computers consist of large numbers of logic gates and memory elements organized to process data at high speed.
- Binary data or instructions are stored temporarily in registers. Binary counters are used in calculations and to keep track of computer operations.
- Instructions and data are stored at specified locations in memory and can be retrieved at will.
- Thus, we will examine the binary number system and learn to apply logic theorems to binary relations.

Binary numbers

- In the decimal number system a quantity is represented by the value and the position of a digit. The number 503.14 means

$$500 \quad +0 \quad +3 \quad +\frac{1}{10} \quad +\frac{4}{100}$$

- Using powers of 10, this can be rewritten as

$$5\times10^{2}+0\times10^{1}+3\times10^{0}+1\times10^{-1}+4\times10^{-2}$$

- In other words, 10 is the base and each position to the left or right of the decimal point corresponds to a power of 10.
- A base 12 or duodecimal system was used by the Babylonians, and we still use 12 in subdividing the foot, the year, and the clock face.
- In representing data by an ON-OFF switch position, there are two possibilities and the corresponding numbers are 1 and 0.
- In such a binary system, the base is 2 and the decimal number 10 is written as 1010 since

$$1\times2^{3}+0\times2^{2}+1\times2^{1}+0\times2^{0}=8+0+2+0=10$$

- In electronic logic circuits the numbers **1** and **0** usually correspond to two easily distinguished voltage levels specified by the circuit designer. For instance, in TTL (Transistor Transistor Logic), **0** corresponds to a voltage near zero and **1** to a voltage near +5 V.

Binary to Decimal conversion

- In a binary number, each position to the right or left of the "binary point" corresponds to a power of 2, and each power of 2 has a decimal equivalent.
- To convert a binary number to its decimal equivalent, add the decimal equivalents of each position occupied by a 1.

For instance $\mathbf{110001} = 2^5 + 2^4 + 0 + 0 + 0 + 2^0 = 32 + 16 + 1 = 49$

$$\mathbf{101.01} = 2^2 + 0 + 2^0 + 0 + 2^{-2} = 4 + 1 + \frac{1}{4} = 5.25$$

Decimal to Binary conversion

- A decimal number can be converted to its binary equivalent by the inverse process, that is, by expressing the decimal number as a sum of powers of 2.

- To convert a decimal integer to its equivalent;
  - progressively divide the decimal number by 2, noting the remainders; the remainders taken in reverse order form the binary equivalent.
- To convert a decimal fraction to its binary equivalent;
  - progressively multiply the fraction by 2, removing and noting the carries; the carries taken in forward order form the binary equivalent.

Example 1

✓Convert decimal 28.375 to its binary equivalent.

Solution.

✓Using the double-dabble method on the integer,

✓ The binary equivalent is **11100**.

2	28	remainder
	14	0
	7	0
	3	1
	1	1
	0	1

✓ Then converting the fraction

$$0.375 \times 2 = 0.75 \text{ with a carry of } 0$$
$$0.75 \times 2 = 1.50 \text{ with a carry of } 1$$
$$0.5 \times 2 = 1.00 \text{ with a carry of } 1$$

✓ The binary equivalent is .**011**.

✓ Therefore, 28.375 is equivalent to binary **11100.011**.

Binary Arithmetic

- ✓Since the binary system uses the same concept of value and position of the digits as the decimal system, we expected the associated arithmetic to be similar but easier.
- ✓The binary multiplication table is very short. For addition in binary, we add column by column, carrying where necessary into higher position columns.
- ✓In subtraction, we subtract column by column, borrowing where necessary from higher position columns.
- ✓In subtracting a large number from a smaller, we can subtract the smaller from the larger and change the sign just as we do with decimals.

Example 2

Convert the numbers in color to the other form and perform the indicated operations.

14	$\mathbf{1110}$	13	$\mathbf{1101}$	1010	$\mathbf{10}$
$+11$	$+\mathbf{1011}$	-10	$-\mathbf{1010}$	-1101	$-\mathbf{13}$
25	$\mathbf{11001}$	3	$\mathbf{0011}$	-0011	$-\mathbf{3}$

- ✓ In **multiplication**, we obtain partial products using the binary multiplication table, and then add the partial products. $(0\times0=0,\ 0\times1=0,\ 1\times0=0,\ 1\times1=1)$
- ✓ In **division**, we perform repeated subtractions just as in long division of decimals, See Example 3.

Example 3

✓Convert the numbers in colour to the other form and perform the indicated operations.

(a)

$$\begin{array}{r} 14.5 \\ \times\ 1.25 \\ \hline 725 \\ 290 \\ 145 \\ \hline 18.125 \end{array}$$

(b)

$$\begin{array}{r} \mathbf{101.1} \\ \mathbf{11.1}\overline{)\mathbf{10011.01}} \\ \mathbf{111} \\ \hline \mathbf{1010} \\ \mathbf{111} \\ \hline \mathbf{111} \\ \mathbf{111} \\ \hline \mathbf{0} \end{array}$$

(a)

$$\begin{array}{r} \mathbf{1110.1} \\ \mathbf{\times\ 1.01} \\ \hline \mathbf{11101} \\ \mathbf{00000} \\ \mathbf{11101} \\ \hline \mathbf{10010.001} \end{array}$$

(b)

$$\begin{array}{r} 5.5 \\ 3.5\overline{)19.25} \\ 175 \\ \hline 175 \\ 175 \\ \hline 0 \end{array}$$

Bits, Bytes, and Words

- ✓A single binary digit is called a "bit." All information in digital system is represented by a sequence of bits. An 8-bit sequence is called a "byte"; a 4-bit sequence is a "nibble."
- ✓The number of bits in the data sequences processed by a given computer is a key characteristic called the "word length". Computers handle data in words of 4 to 64 bits.
- ✓An 8-bit microprocessor can receive, process, store, and transmit data or instructions in form of bytes. Eight bits can be arranged in $2^8 = 256$ different combinations.

Two's Complement Notation

- ✓A better notation for computers, one that is easily implemented in hardware, is based on the fact that adding the complement of a number is equivalent to subtracting the number.
- ✓A "complement" is that which completes; the "*n*'s complement" of a number *x* is equal to. $n - x$

Two's Complement Notation

- ✓For example, the 10's complement of 3 is 7. To evaluate $9-3$, i.e., to subtract 3 from 9, we can "add the 10's complement" of 3 (i.e., $10-3=7$) to obtain

  $9+7=16$ to yield 6 after discarding the final carry.

- ✓In the decimal system, the 10's complement of a multidigit number is easily found by taking the 9's complement of each digit (by inspection) and then adding 1.

- ✓In general, to subtract a two-digit number *B* from *A* we use the relationship

$$A-B=A+\left[100-B\right]-100=A+\left[\left(99-B\right)+1\right]-100$$

  where $(99-B)$ is the 9's complement.

- ✓ In the binary system, arithmetic is simplified if negative numbers are in signed 2's complement notation. In this notation, the MSB (Most Significant Bit) is the sign bit: **0** for plus, **1** for minus. To form the 2's complement of any number, positive or negative:
  - ❖ Form the 1's complement by changing **1**s *to* **0**s *and* **0**s *to* **1**s. *Add* **1**.

✓ If the result of an arithmetic operation has a 1 sign bit, it is a negative number in 2's complement notation; to obtain the true magnitude, subtract 1 and form the 1's complement.

Example 4

a) Obtain the 10's complement of 15 and 24.

b) Represent -15 and -24 in 8-bit signed 2's complement notation.

c) Perform $24-15$ and $15-24$ directly and by complement notation.

Solution

a) Form the 9's complement of each digit, then add 1:

$15 \rightarrow 84+1=85$ $24 \rightarrow 75+1=76$

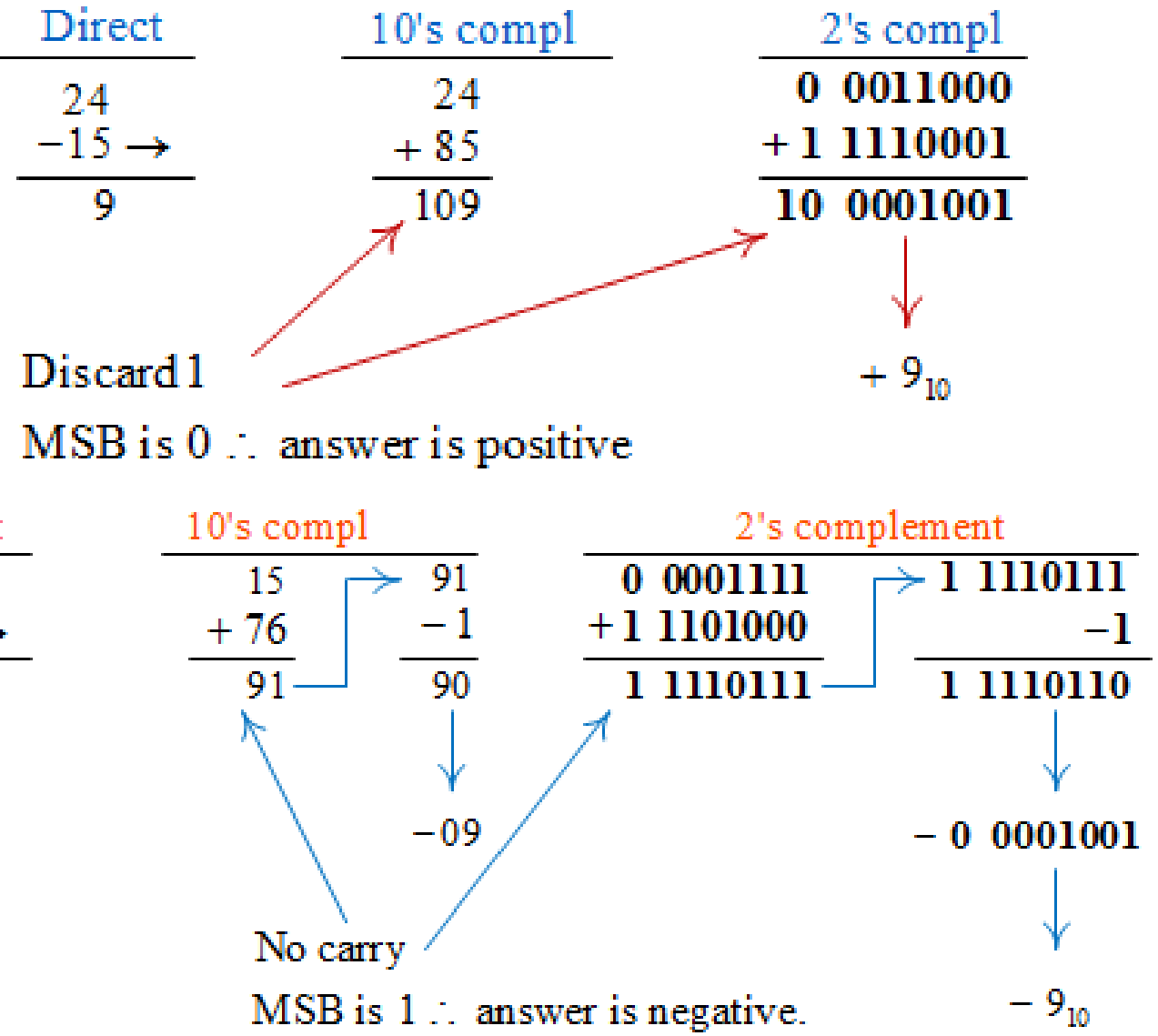
b) Form the 1's complement of each digit, then add 1.

$-15_{10} \rightarrow \mathbf{-1111} \rightarrow \mathbf{-00001111} \rightarrow \mathbf{11110000+1} \rightarrow \mathbf{11110001} \rightarrow \mathbf{1\ 1110001}$

$-24_{10} \rightarrow \mathbf{-11000} \rightarrow \mathbf{-00011000} \rightarrow \mathbf{11100111+1} \rightarrow \mathbf{11101000} \rightarrow \mathbf{1\ 1101000}$

Example 4

c) Solution



Logic gates and Boolean algebra

- ✓We have seen that binary arithmetic and decimal arithmetic are similar in many respects. To work with logic relations in digital form, we need a set of rules for symbolic manipulation that will enable us to simplify complex expressions and solve for unknowns.
- ✓Simply put, we need a **digital algebra**. Nearly 100 years before the first digital computer, George Boole, an English mathematician (1815-1864), formulated a basic set of rules governing the true-false statements of logic.
- ✓Eighty-five years later (1938), Claude Shannon , at that time a graduate student a MIT, pointed out the usefulness of Boolean algebra in solving telephone switching problems and established the analysis of such problems on a firm mathematical basis.
- ✓**Boolean algebra** is valuable in manipulating binary variables in **OR**, **AND**, or **NOT** relations and in the analysis and design of all types of digital systems.

Introduction

- ✓Logic gates are electronic circuits that can be used to implement the most elementary logic expressions also known as Boolean expressions.
- ✓The logic gate is the most basic building block of combinational logic. There are three basic logic gates, namely the **OR gate**, the **AND gate** and the **NOT gate**.
- ✓Other logic gates that are derived from these basic gates are the **NAND gate**, the **NOR gate**, the **EXCLUSIVE-OR gate** and the **EXCLUSIVE-NOR gate**.

Truth Table

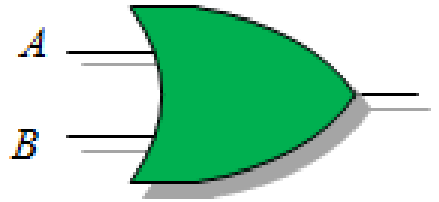
- ✓A truth table lists all possible combinations of input binary variables and the corresponding outputs of a logic system.
- ✓When the number of input binary variables is only one, then there are only two possible inputs, i.e., '**0**' and '**1**'.
- ✓If the number of inputs is two, there can be four possible input combinations, i.e., **00**, **01**, **10** and **11**.

OR Gate

✓An OR gate performs an ORing operation on two or more logic variables. The OR operation on two independent logic variables *A* and *B* is written as

$Y = A + B$ and reads as *Y* equals *A* OR *B*.

✓An OR gate is a logic circuit with two or more inputs and one output. The output of an OR gate is LOW only when all of its inputs are LOW, otherwise, it is HIGH for all other possible input combinations.



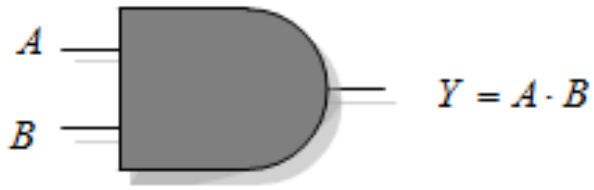
A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	1

Figure 4.1: Two-input OR gate.

✓ Figure 4.1 shows the logic circuit symbol and truth table of a two-input OR gate.

AND Gate

- ✓An AND gate is a logic ckt having two or more inputs and one output. Its output is HIGH only when all of its inputs are in the HIGH state.
- ✓In all other cases, the output is LOW. The logic symbol and truth table of a two input AND gate are shown in Figure 4.2.



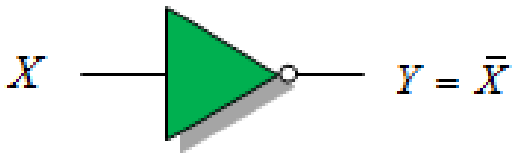
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 4.2: Two-input AND gate.

- ✓ The AND operation on two independent logic variables *A* and *B* is written as $Y = A \cdot B$ and reads as *Y* equals *A* AND *B*.

Not Gate

- ✓A NOT gate is a one-input, one-output logic ckt whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa.
- ✓Figure 4.3 shows the circuit symbol and the truth table for a NOT gate.



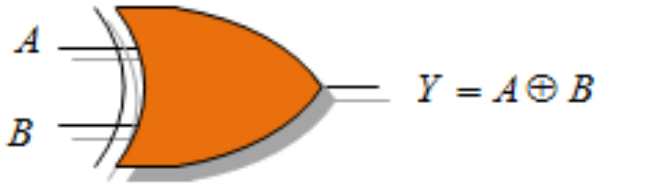
X	Y
0	1
1	0

Figure 4.3: Circuit symbol and truth table of a NOT ckt.

- ✓ The NOT operation on a logic variable X is denoted as $\bar{X}$ or X'. That is, if X is the input to a NOT ckt, then its Y is given by $Y = \bar{X}$ *or* X' and reads as Y equals NOT X.

Exclusive-OR Gate

- ✓Commonly written as EX-OR gate, is a two-input gate. Its output is logic ‘1’ when the inputs are unlike and logic ‘0’ when the inputs are like.
- ✓The output of a multiple-input EX-OR logic function is a logic ‘1’ when the number of 1s in the input sequence is odd and logic ‘0’ when the number of 1s in the input sequence is even, including zero.
- ✓That is an all 0s input sequence also produces a logic ‘0’ at the output.



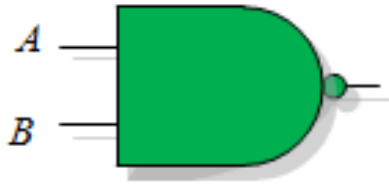
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.4: Circuit symbol and truth table of an EX-OR ckt.

- ✓ The output of a two-input EX-OR gate is expressed by $Y = A \oplus B$

NAND Gate

✓NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate. NAND gate operation is logically expressed as $Y = \overline{A \cdot B}$.



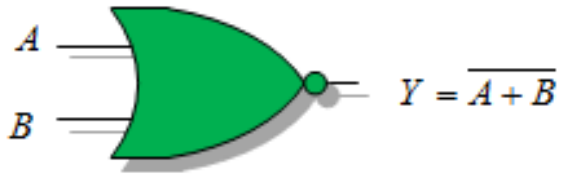
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 4.5: Circuit symbol and truth table of a NAND gate.

✓ Figure 4.5 shows the circuit symbol and the truth table of a two-input NAND gate.

NOR Gate

✓NOR stands for NOT OR. An OR gate followed by a NOT ckt makes it a NOR gate. The output of a two-input NOR gate is logically expressed as $Y = \overline{A+B}$.



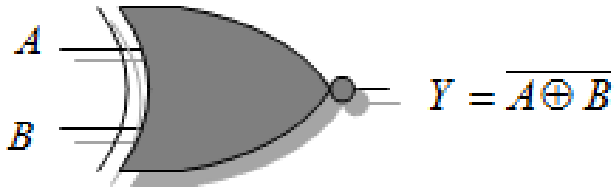
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 4.6: Circuit symbol and truth table of a NOR gate.

✓ Figure 6 shows the circuit symbol and the truth table of a two-input NOR gate.

Exclusive-NOR Gate

✓EX-NOR means NOT of EX-OR, i.e., the logic gate that we get by complementing the output of an EX-OR gate. Logically the output is given by $Y = \overline{A \oplus B}$.



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

Figure 4.7: Circuit symbol and truth table of a EX-NOR gate.

- ✓ Figure 4.7 shows the circuit symbol and the truth table of a two-input EX-NOR gate.
- ✓ The output of a two-input EX-NOR gate is a logic ‘1’ when the inputs are like and logic ‘0’ when they are unlike.

- ✓Boolean algebra is mathematics of logic. It is one of the most basic tools available to the logic designer and thus can be effectively used for simplification of complex logic.
- ✓Boolean algebra, quite interestingly, is simpler than ordinary algebra. It is also composed of a set of symbols and a set of rules to manipulate these symbols. Nevertheless, the differences between Boolean and ordinary algebra are as follows:

1. In ordinary algebra, the letter symbols can take on any number of values including infinity. In Boolean algebra, they can take on either of two values, that is, 0 and 1.
2. The values assigned to a variable have a numerical significance in ordinary algebra, whereas in its Boolean counterpart they have a logical significance.
3. While ‘ . ’ and ‘+’ are respectively the signs of multiplication and addition in ordinary algebra, in Boolean algebra ‘ . ’ means AND operation and ‘+’ means an OR operation.

4. Boolean algebra captures the essential properties of both logic operations such as AND, OR and NOT and set operations such as intersection, union and complement.
5. Boolean algebra may also be defined to be a set *A* supplied with two operations of logical AND (Λ) , logical OR (V), a unitary operation NOT $(-)$, and two elements, namely logical FALSE (0) and logical TRUE (1). This set is such that, for all elements of this set, the postulates or axioms relating to the associative, commutative, distributive, absorption and complementation properties of these elements hold good.

Postulates of Boolean Algebra

- $0+0=0, \quad 0\cdot 0=0$.
- $0+1=1,\ 1+0=1; \quad 0\cdot 1=0,\ 1\cdot 0=0$.
- $1+1=1, \quad 1\cdot 1=1$.
- $\overline{1}=0, \quad \overline{0}=1$.

- ✓ Many theorems of Boolean algebra are based on these postulates, which can be used to simplify Boolean expressions.

Theorems of Boolean Algebra

- ✓ **Theorem 1** (Operations with '0' and '1')

$$(\text{a}) \quad 0 \cdot X = 0 \quad \text{and} \quad (\text{b}) \quad 1 + X = 1$$

Where *X* is not necessarily a single variable – it could be a term or even a large expression.

Proof of theorem 1(a): we substitute all possible values of *X*, that is, 0 and 1, into the give expression and check if the LHS equals the RHS.

- ❑ For $X = 0,\ \text{LHS} = 0 \cdot X = 0 \cdot 0 = 0 = \text{RHS}$.
- ❑ For $X = 1,\ \text{LHS} = 0 \cdot 1 = 0 = \text{RHS}$.

- ✓ Thus , $0 \cdot X = 0$ irrespective of the value of *X*, and hence the proof.

Proof of theorem 1(b) is similar: In general, according to theorem 1,

$$0 \cdot (\text{Boolean expression}) = 0 \quad \text{and} \quad 1 + (\text{Boolean expression}) = 1$$

- ✓ For example, $0\cdot(A\cdot B+B\cdot C+C\cdot D)=0$ and $1+(A\cdot B+B\cdot C+C\cdot D)=1$, where *A*, *B* and *C* are Boolean variables.
- ✓ **Theorem 2** (operations with '0' and '1')

$$(\text{a})\quad 1\cdot X=X \quad \text{and} \quad (\text{b})\quad 0+X=X$$

where *X* could be a variable, a term or even a large expression.

Proof of theorem 2(a):.

- ❑ For $X=0,\ \text{LHS}=1\cdot 0=0=\text{RHS}$.
- ❑ For $X=1,\ \text{LHS}=1\cdot 1=1=\text{RHS}$.

Proof of theorem 2(b) is similar: In general, according to theorem 2,

$1\cdot(\text{Boolean expression})=\text{Boolean expression}$ and $0+(\text{Boolean expression})=\text{Boolean expression}$

.

- ✓ For example, $1\cdot(A\cdot B+B\cdot C+C\cdot D)=0+(A\cdot B+B\cdot C+C\cdot D)=A\cdot B+B\cdot C+C\cdot D$

- ✓ **Theorem 3** (Idempotent or Identity Laws)

$$(\text{a}) \quad X \cdot X \cdot X \cdots X = X \qquad \text{and} \qquad (\text{b}) \quad X + X + X + \cdots + X = X$$

- ✓ Theorem 3(a) is a direct consequence of an AND gate operation, whereas theorem 3(b) represents an OR gate operation when all inputs of the gate have been tied together.
- ✓ The scope of the idempotent laws can be expanded further by considering *X* to be a term or an expression.
- ✓ **Example 1**
- ✓ Apply idempotent laws to simplify the Boolean expression

$$\left(A \cdot \overline{B} \cdot \overline{B} + C \cdot C\right) \cdot \left(A \cdot \overline{B} \cdot \overline{B} + A \cdot \overline{B} + C \cdot C\right)$$

- ✓ **Solution**

$$\left(A \cdot \overline{B} \cdot \overline{B} + C \cdot C\right) \cdot \left(A \cdot \overline{B} \cdot \overline{B} + A \cdot \overline{B} + C \cdot C\right) = \left(A \cdot \overline{B} + C\right) \cdot \left(A \cdot \overline{B} + A \cdot \overline{B} + C\right)$$

$$= \left(A \cdot \overline{B} + C\right) \cdot \left(A \cdot \overline{B} + C\right) = A \cdot \overline{B} + C$$

- **Theorem 4** (Complementation Law)

$$(a)\quad X \cdot \overline{X} = 0 \quad \text{and} \quad (b)\quad X + \overline{X} = 1$$

- By this theorem, in general, any Boolean expression when ANDed to its complement yields a '0' and when ORed to its complement yields a '1', irrespective of the complexity of the expression.

Proof of theorem 4(a):

- For $X = 0,\ \overline{X} = 1,\quad \text{LHS} = X \cdot \overline{X} = 0 \cdot 1 = 0 = \text{RHS}.$
- For $X = 1,\ \overline{X} = 0,\quad \text{LHS} = X \cdot \overline{X} = 1 \cdot 0 = 0 = \text{RHS}.$

- Hence, theorem 4(a) is proved. Since theorem 4(b) is the dual of theorem 4(a), its proof is implied.
- To further illustrate the application of theorem 4, consider

$$(A + B \cdot C)\overline{(A + B \cdot C)} = 0 \quad \text{and} \quad (A + B \cdot C) + \overline{(A + B \cdot C)} = 1$$

✓ **Example 2**

Simplify the following:

$$\left[1+L\cdot M+L\cdot\overline{M}+\overline{L}\cdot M\right]\cdot\left[\left(L+\overline{M}\right)\cdot\left(\overline{L}\cdot M\right)+\left(\overline{L}\cdot\overline{M}\right)\cdot\left(L+M\right)\right]$$

Solution

- We know that $1+(\text{Boolean expression})=1$.
- Also, $\left(\overline{L}\cdot M\right)$ is the complement of $\left(L+\overline{M}\right)$ and $\left(\overline{L}\cdot\overline{M}\right)$ is the complement of $\left(L+M\right)$.
- Therefore, the given expression reduces to $1\cdot\left[0+0\right]=1\cdot 0=0$.

✓ **Theorem 5** (Commutative Laws)

$$(a)\quad X+Y=Y+X \qquad \text{and} \qquad (b)\quad X\cdot Y=Y\cdot X$$

✓ Theorem 5(a) implies that the order in which variables are add or ORed is immaterial. Theorem 5(b) implies that the order in which variables are ANDed is also immaterial.