# DIGITAL ELECTRONICS & MICROPROCESSORS LAB

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

II B.Tech - I Semester

Prepared By:

Dr. G. Reddy Hemantha,
Associate Professor,
Dept. of ECE,
Aditya College of Engineering,
Madanapalle.

# PREFACE

The significance of the Digital Electronics & Microprocessors Lab is renowned in the various fields of engineering applications. For Electronics and Communication Engineer and a Computer Science & Engineer, it is obligatory to have the practical ideas about the applications of Digital Electronics & Microprocessors. By this perspective, we have introduced a Laboratory manual cum Observation for Digital Electronics & Microprocessors Lab.

The manual uses the plan, cogent and simple language to explain the fundamental aspects of Digital Electronics & Microprocessors in practical. The manual prepared very carefully with our level best. It gives all the steps in executing an experiment.

# DO'S AND DON'TS IN THE LAB

**DO'S:-**

1. Proper dress has to be maintained while entering in the Lab.
2. Students should carry observation notes and record completed in all aspects.
3. Know the thoery behind the experiment before coming to the lab.
4. Students should be at their concerned desktop/work bench, unnecessary moment is restricted.
5. Identify the pins or leads of the IC before making connections.
6. Know the biasing voltage required for different families of IC's and connect power supply and ground terminals to the respective pins of IC's.
7. Mount the IC properly on the IC socket/bread board.
8. Handle instruments with utmost care.
9. Know the procedure to execute ALP.
10. ALP and its theoretical result should be there in the observation before coming to the next lab.
11. After completing the ALP Students should verify the ALP by the Lab Instructor.
12. The practical result should be noted down into their observations and result must be shown to the Lecturer In-Charge for verification.
13. Students must ensure that all switches are in the OFF position, desktop is shut down properly.

**DON'Ts:-**

1. Don't come late to the Lab.
2. Don't leave the Lab without making proper shut down of desktop and keeping the chairs properly.
3. Don't leave the Lab without verification by Lab instructor.
4. Don't leave the lab without the permission of the Lecturer In-Charge.
5. Do not eat food, drink beverages or chew gum in the laboratory.
6. Do not open any irrelevant internet sites on lab computer
7. Do not use a flash drive on lab computers.
8. Long hair, dangling jewelry and loose or baggy clothing are a hazard in the laboratory

## Course Objectives

- To understand all the concepts of Logic Gates and Boolean Functions.
- To learn about Combinational Logic and Sequential Logic Circuits.
- To design logic circuits using Programmable Logic Devices.
- To understand basics of 8086 Microprocessor and 8051 Microcontroller.
- To understand architecture of 8086 Microprocessor and 8051 Microcontroller.
- To learn Assembly Language Programming of 8086 and 8051

## Course Outcomes

After completion of the course, students will be able to

- Design any Logic circuit using basic concepts of Boolean Algebra.
- Design any Logic circuit using basic concepts of PLDs.
- Design and develop any application using 8086 Microprocessor.
- Design and develop any application using 8051 Microcontroller

# LIST OF EXPERIMENTS

## DIGITAL ELECTRONICS:

1. Verification of Truth Table for AND, OR, NOT, NAND, NOR and EX-OR gates.
2. Realisation of NOT, AND, OR, EX-OR gates with only NAND and only NOR gates.
3. Karnaughmap Reduction and Logic Circuit Implementation.
4. Verification of DeMorgan's Laws.
5. Implementation of Half-Adder and Half-Subtractor.
6. Implementation of Full-Adder and Full-Subtractor.
7. Four Bit Binary Adder
8. Four Bit Binary Subtractor using 1's and 2's Complement.

## MICROPROCESSORS (8086 Assembly Language Programming):

1. 8 Bit Addition and Subtraction.
2. 16 Bit Addition.
3. BCD Addition .
4. BCD Subtraction.
5. 8 Bit Multiplication.
6. 8 Bit Division.
7. Searching for an Element in an Array.
8. Sorting in Ascending and Descending Orders.
9. Finding Largest and Smallest Elements from an Array.
10. Block Move

## PROCEDURE TO CREATE FILES AND TO EXECUTE PROGRAMS:

→ Navigate to MASM directory.

→ Create an ASM file using notepad.

→ Save the program with an extension of **.asm.**

→ Make ensure that the file is saved with an extension of **.asm.**

→ Click on DosBox icon.

→ Type **mount e e:\**

→ Type **e:** to go to e drive.

→ Type **cd masm** to change masm directory.

→Type **exec filename** to check for errors

→If no errors or warnings, type **afdebug filename** to see the result.

To open execution window which shows all the registers and memory locations

→Use f1 for single step execution

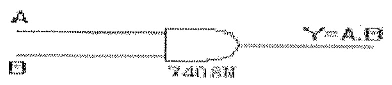
→Use f7, f8, f9, f10 for navigation through the window

→Use f3 to go to starting command line

# DIGITAL ELECTRONICS

**AND GATE:**

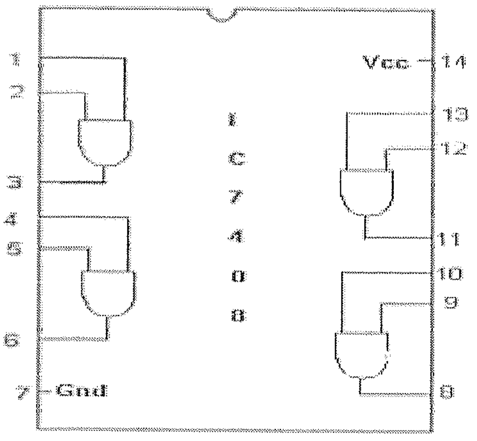
**SYMBOL:**



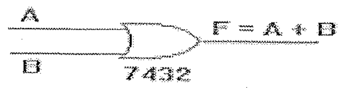
TRUTH TABLE

| A | B | A.B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**PIN DIAGRAM:**



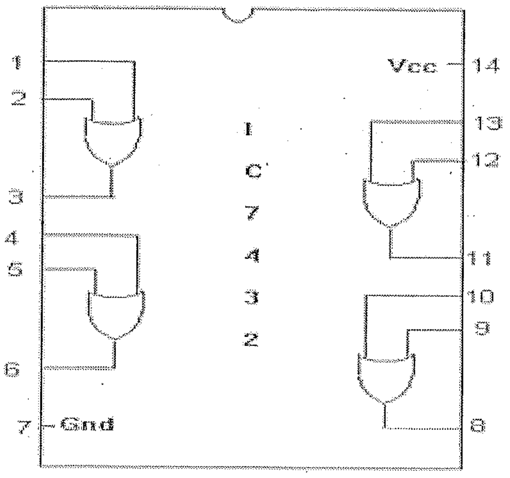
**OR GATE:**
**SYMBOL:**



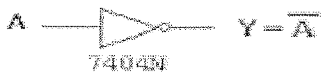
TRUTH TABLE

| A | B | A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**PIN DIAGRAM:**



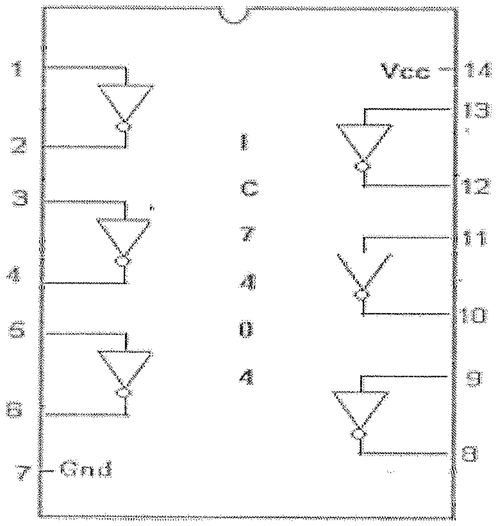
**NOT GATE:**
**SYMBOL:**



TRUTH TABLE :

| A | $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

**PIN DIAGRAM:**



# 1. LOGIC GATES

**AIM**: - To verify the truth tables for AND, OR, NOT, NAND, NOR and EX-OR gates.

**APPARATUS REQUIRED**: -

Digital lab trainer kit, single strand wires, breadboard, TTL IC's AND (IC-7408), OR (IC-7432), NAND (IC-7400), NOR (IC-7402), NOT (IC-7404) and XOR (IC-7486).

**THEORY**: -

Logic gates are electronic circuits, which perform a logical operation on one or more logical inputs and produce a single output. Logic gates are the basic building blocks of any digital system.

## BASIC GATES:

**AND GATE:**

The AND gate performs a logical multiplication commonly known as AND function. The output is high when both the inputs are high. The output is low level when any one of the inputs is low.

**OR GATE:**

The OR gate performs a logical addition commonly known as OR function. The output is high when any one of the inputs is high. The output is low level when both the inputs are low.

**NOT GATE:**

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

## UNIVERSAL GATES:

**NAND GATE:**

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low. The output is low level when both inputs are high.

**NOR GATE:**

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

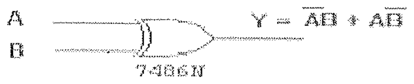
## ADVANCED GATES:

**X-OR GATE:**

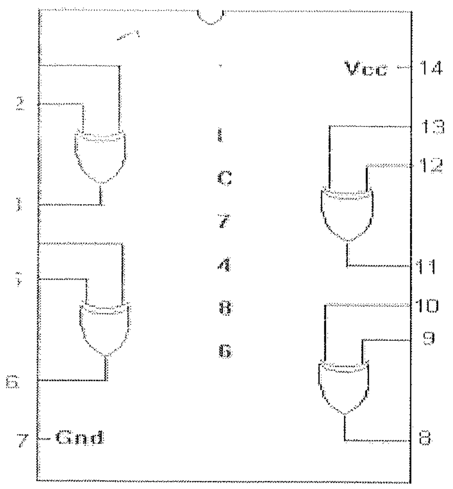
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

## XOR GATE:

### SYMBOL:



### PIN DIAGRAM:

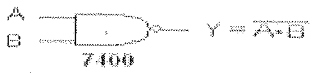


TRUTH TABLE :

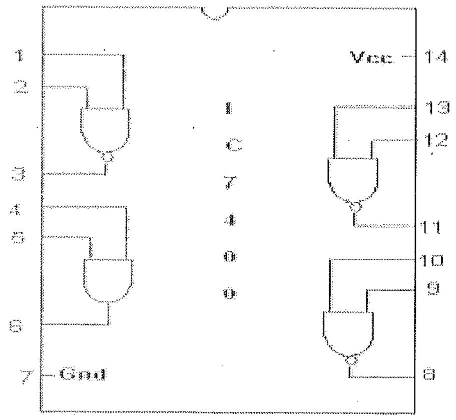
| A | B | $\overline{A}B + A\overline{B}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## NAND GATE:

### SYMBOL:



### PIN DIAGRAM:

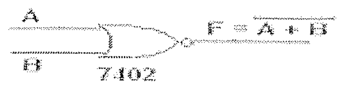


TRUTH TABLE

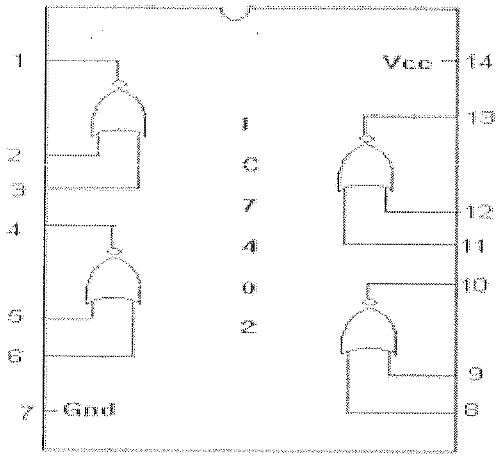
| A | B | $\overline{A \cdot B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## NOR GATE:

### SYMBOL:



### PIN DIAGRAM:



TRUTH TABLE

| A | B | $\overline{A+B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**PROCEDURE:**

1. Place the breadboard gently on the observation table.
2. Fix the IC which is under observation between the half shadow line of breadboard, so there is no shortage of voltage.
3. Connect the wire to the main voltage source (Vcc) whose other end is connected to last pin of the IC (14 place from the notch).
4. Connect the ground of IC (7th place from the notch) to the ground terminal provided on the digital lab kit.
5. Give the input at any one of the gate of the ICs i.e. 1st, 2nd, 3rd, 4th gate by using connecting wires.
6. Connect output pins to the led on digital lab kit.
7. Switch on the power supply.
8. If LED glows red then output is true, otherwise output is false, which is numerically denoted as 1 and 0respectively. The Color can change based on the IC manufacturer it's just verification of the Truth Table not the color change.

**RESULT:**

Thus the logic gates are studied and their truth tables were verified.

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

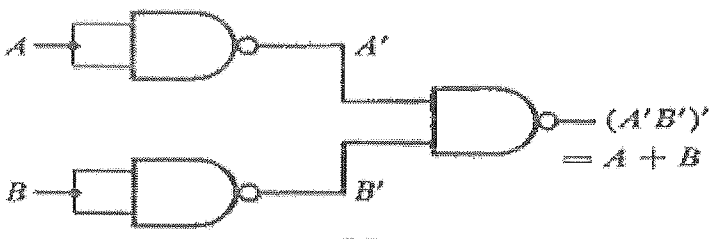
| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## NAND gates as OR gate:

From DeMorgan's theorems: (A.B)' = A' + B'

(A'.B')' = A'' + B'' = A + B

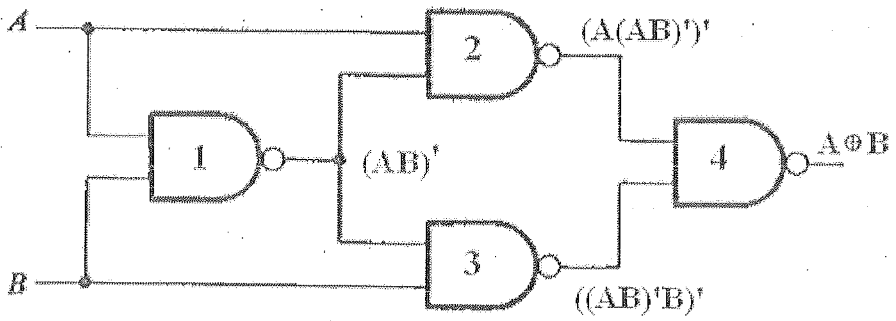
So, give the inverted inputs to a NAND gate, obtain OR operation at output.



OR

## NAND gates as X-OR gate:

The output of a two input X-OR gate is shown by: Y = A'B + AB'. This can be achieved with the logic diagram shown in the below.



X-OR

| Gate No. | Inputs | Output |
|---|---|---|
| 1 | A, B | (AB)' |
| 2 | A, (AB)' | (A (AB)')' |
| 3 | (AB)', B | (B (AB)')' |
| 4 | (A (AB)')', (B (AB)')' | A'B + AB' |

Now the output from gate no. 4 is the overall output of the configuration.

Y = ((A (AB)')' (B (AB)')')'
= (A(AB)')'' + (B(AB)')''
= (A(AB)') + (B(AB)')
= (A(A' + B')) + (B(A' + B'))
= (AA' + AB') + (BA' + BB')
= ( 0 + AB' + BA' + 0 )
= AB' + BA'
Y = AB' + A'B

# 2. VERIFICATION OF NAND AND NOR GATES AS UNIVERSAL GATES

**AIM: -**

Verify the NAND and NOR gates as universal logic gates.

**APPARATUS REQUIRED: -**

Logic trainer kit, NAND gates (IC 7400), NOR gates (IC 7402), wires.

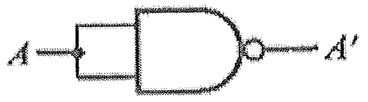
**THEORY: -**

NAND gate is actually a combination of two logic gates: AND gate followed by NOT gate. So its output is complement of the output of an AND gate.
This gate can have minimum two inputs; output is always one. By using only NAND gates, we can realize all logic functions AND, OR, NOT, X-OR, X-NOR, NOR. So this gate is also called universal gate.

**NAND gates as NOT gate:**

A NOT produces complement of the input. It can have only one input, tie the inputs of a NAND gate together. Now it will work as a NOT gate. Its output is

Y = (A.A)'
Y = (A)'

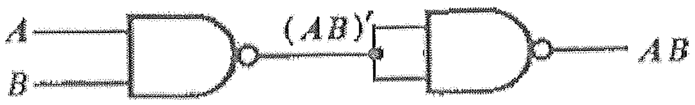


NOT (inverter)

**NAND gates as AND gate:**

A NAND produces complement of AND gate. So, if the output of a NAND gate is inverted, overall output will be that of an AND gate.

Y = ((A.B)')'
Y = (A.B)

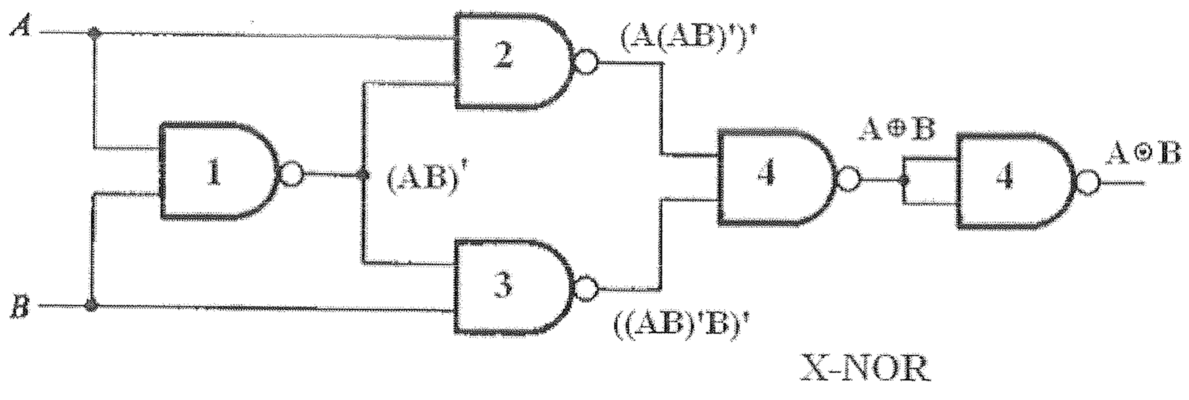


AND

**NAND gates as X-NOR gate:**

X-NOR gate is actually X-OR gate followed by NOT gate. So give the output of X-OR gate to a NOT gate, overall output is that of an X-NOR gate.

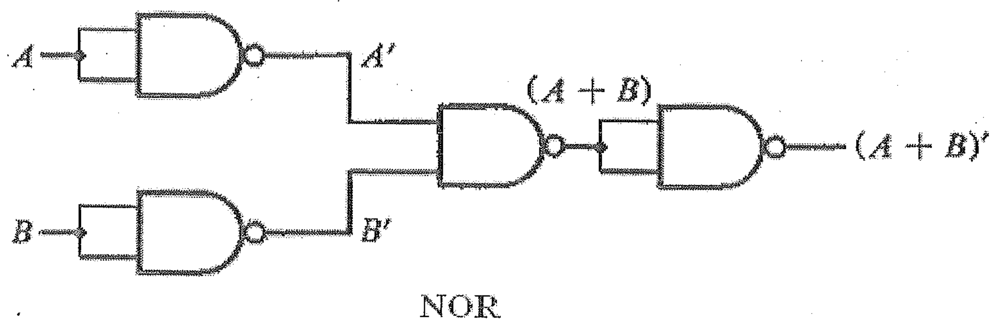
Y = AB+ A'B'



**NAND gates as NOR gate**

A NOR gate is an OR gate followed by NOT gate. So connect the output of OR gate to a NOT gate, overall output is that of a NOR gate.

Y = (A + B)'



**PROCEDURE:**

1. Connect the trainer kit to ac power supply.
2. Connect the NAND gates for any of the logic functions to be realized.
3. Connect the inputs of first stage to logic sources and output of the last gate to logic indicator.
4. Apply various input combinations and observe output for each one.
5. Verify the truth table for each input/ output combination.
6. Repeat the process for all logic functions.
7. Switch off the power supply.

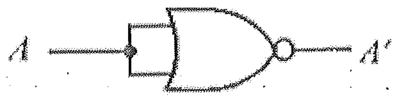
**THEORY:**

NOR gate is actually a combination of two logic gates: OR gate followed by NOT gate. So its output is complement
of the output of an OR gate. This gate can have minimum two inputs; output is always one. By using only NOR gates, we can realize all logic functions: AND, OR, NOT, X-OR, X-NOR, NAND. So this gate is also called universal gate.

**NOR gates as NOT gate:**

A NOT produces complement of the input. It can have only one input, tie the inputs of a NOR gate together. Now it will work as a NOT gate. Its output is

Y = (A+A)'
Y = (A)'

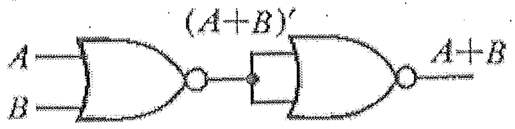


NOT

**NOR gates as OR gate:**

A NOR produces complement of OR gate. So, if the output of a NOR gate is inverted, overall output will be that of an OR gate.

Y = ((A+B)')'
Y = (A+B)

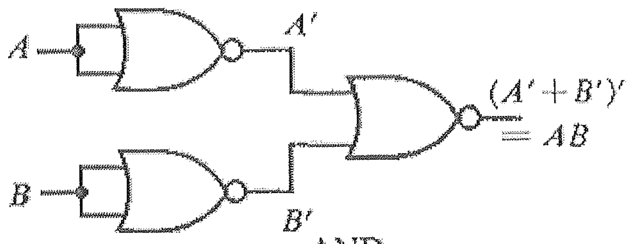


OR

**NOR gates as AND gate:**

From DeMorgan's theorems: (A+B)' = A'B'
(A'+B')' = A''B'' = AB

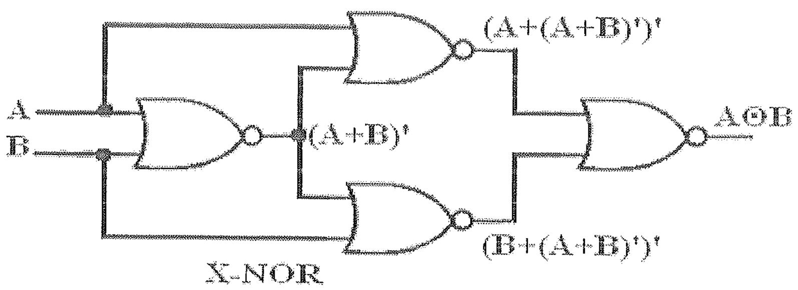
So, give the inverted inputs to a NOR gate, obtain AND operation at output.



AND

## NOR gates as X-NOR gate:

The output of a two input X-NOR gate is shown by: Y = AB + A'B'. This can be achieved with the logic diagram shown in the left side.



X-NOR

| Gate No. | Inputs | Output |
|---|---|---|
| 1 | A, B | (A + B)' |
| 2 | A, (A + B)' | (A + (A+B)')' |
| 3 | (A + B)', B | (B + (A+B)')' |
| 4 | (A + (A + B)')', (B + (A+B)')' | AB + A'B' |

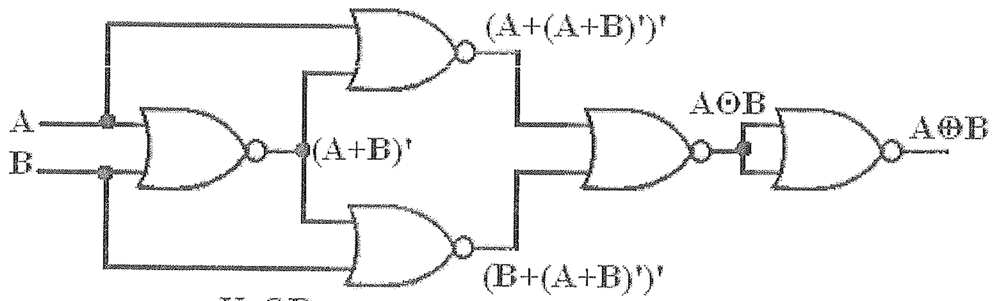
Now the output from gate no. 4is the overall output of the configuration.

$$
\begin{aligned}
Y &= ((A + (A+B)')' (B +( A+B)')')' \\
&= (A+(A+B)')''.(B+(A+B)')'' \\
&= (A+(A+B)').(B+(A+B)') \\
&= (A+A'B').(B+A'B') \\
&= (A + A').(A + B').(B+A')(B+B') \\
&= 1.(A+B').(B+A').1 \\
&= (A+B').(B+A') \\
&= A.(B + A') +B'.(B+A') \\
&= AB + AA' +B'B+B'A' \\
&= AB + 0 + 0 + B'A' \\
&= AB + B'A' \\
Y &= AB + A'B'
\end{aligned}
$$

## NOR gates as X-OR gate:

X-OR gate is actually X-NOR gate followed by NOT gate. So give the output of X-NOR gate to a NOT gate, overall output is that of an X-OR gate.

Y = A'B+ AB'

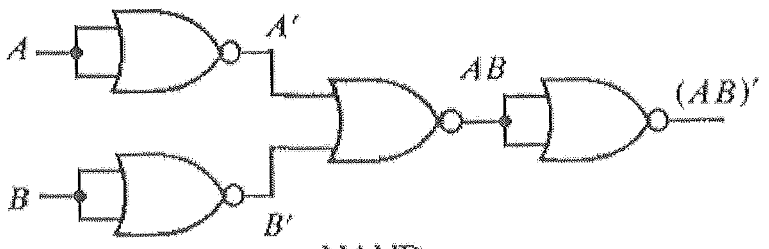


X- OR

**NOR gates as NAND gate:**

A NAND gate is an AND gate followed by NOT gate. So connect the output of AND gate to a NOT gate, overall output is that of a NAND gate.

Y = (AB)'



NAND

**PROCEDURE:**

1. Connect the trainer kit to ac power supply.
2. Connect the NOR gates for any of the logic functions to be realised.
3. Connect the inputs of first stage to logic sources and output of the last gate to logic indicator.
4. Apply various input combinations and observe output for each one.
5. Verify the truth table for each input/ output combination.
6. Repeat the process for all logic functions.
7. Switch off the ac power supply.

**RESULT:**

Thus NAND & NOR verified as universal gates successfully.

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

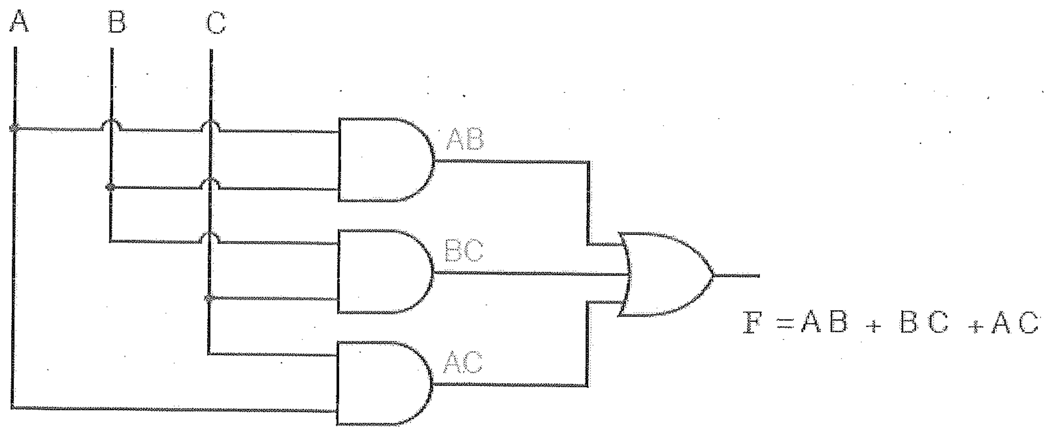
$F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

K Map:

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | | 1 | 1 | 1 |

$F = A\overline{B} + BC + AC$

Logic Implementation:



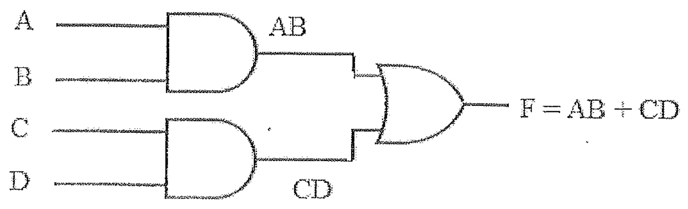
$F = \overline{A}\overline{B}CD + \overline{A}BCD + ABCD + A\overline{B}CD + AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D}$

K Map:

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | |
| 01 | | | 1 | |
| 11 | 1 | 1 | 1 | 1 |
| 10 | | | 1 | |

$F = AB + CD$

Logic Circuit:



## 3. KARNAUGH MAP METHOD

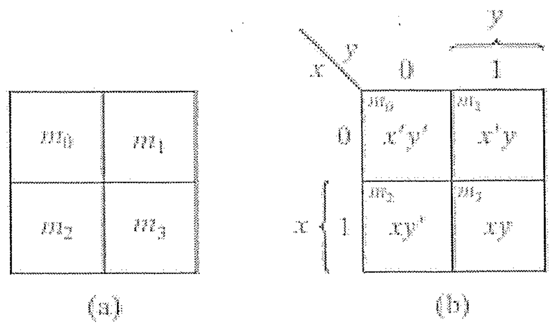
**AIM**: To minimize the given Boolean function using Karnaugh Map method and logic implementation.

**COMPONENTS REQUIRED**: Power Supply (0 – 5V), IC 7400, 7408, 7432, 7404, and 7402, Digital IC trainer kit, connecting wires.

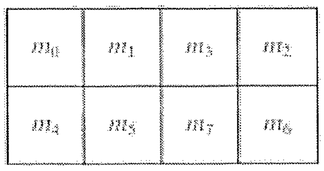
**THEORY:**

- A Karnaugh map (K-map) is a pictorial method used to minimize Boolean expressions without having to use Boolean algebra theorems and equation manipulations.
- This is a graphical technique that utilizes a sum of product (SOP) form. SOP forms combine terms that have been ANDed together that then get ORed together.
- This format lends itself to the use of De Morgan's law which allows the final result to be built with only NAND gates. The K-map is best used with logical functions with four or less input variables.
- One of the advantages of using K-maps for reduction is that it is easier to see when a circuit has been fully simplified.
- Another advantage is that using K-maps leads to a more structured process for minimization. In order to use a K-map, the truth table for a logical expression is transferred to a K-map grid.
- The grid for two, three, and four input expressions are provided in the tables below. Each cell corresponds to one row in a truth table or one given state in the logical expression.
- The order of the items in the grid is not random at all; they are set so that any adjacent cell differs in value by the change in only one variable.
- Because of this, items can be grouped together easily in rectangular blocks of two, four, and eight to find the minimal number of groupings that can cover the entire expression.
- Note that diagonal cells require that the value of more than two inputs change, and that they also do not form rectangles.

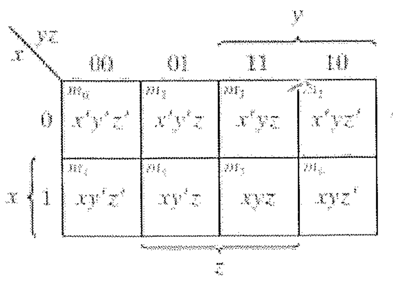
Two Variable K Map:



Three Variable K Map:



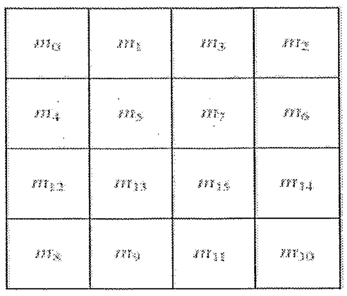
(a)



x yz
y
00 01 11 10
0 1
x z
$m_0$ $x'y'z'$ $m_1$ $x'y'z$ $m_3$ $x'yz$ $m_2$ $x'yz'$
$m_4$ $xy'z'$ $m_5$ $xy'z$ $m_7$ $xyz$ $m_6$ $xyz'$

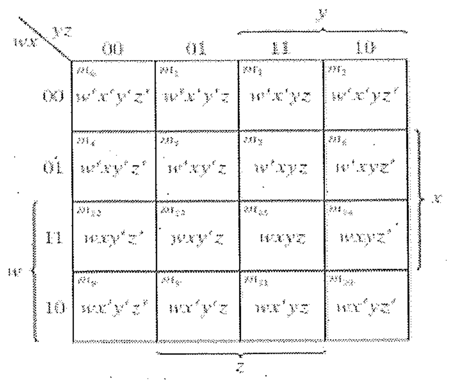
(b)

Four Variable K Map:



$m_0$ $m_1$ $m_3$ $m_2$
$m_4$ $m_5$ $m_7$ $m_6$
$m_{12}$ $m_{13}$ $m_{15}$ $m_{14}$
$m_8$ $m_9$ $m_{11}$ $m_{10}$

(a)



wx yz
y
00 01 11 10
00 01 11 10
x w z
$m_0$ $w'x'y'z'$ $m_1$ $w'x'y'z$ $m_3$ $w'x'yz$ $m_2$ $w'x'yz'$
$m_4$ $w'xy'z'$ $m_5$ $w'xy'z$ $m_7$ $w'xyz$ $m_6$ $w'xyz'$
$m_{12}$ $wxy'z'$ $m_{13}$ $wxy'z$ $m_{15}$ $wxyz$ $m_{14}$ $wxyz'$
$m_8$ $wx'y'z'$ $m_9$ $wx'y'z$ $m_{11}$ $wx'yz$ $m_{10}$ $wx'yz'$

(b)

**PROCEDURE:**

1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output.

**RESULT:**

Thus minimization of Boolean functions using K map and implementation of logic circuits done and verified the output of simplified function successfully.

INFERENCES:

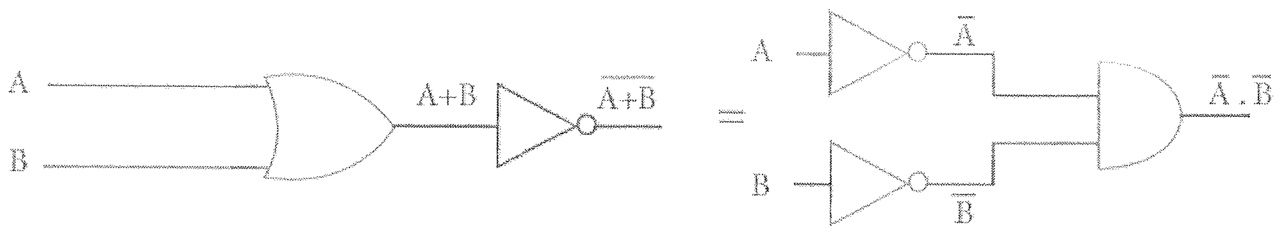
APPLICATIONS:

(To be filled by faculty)

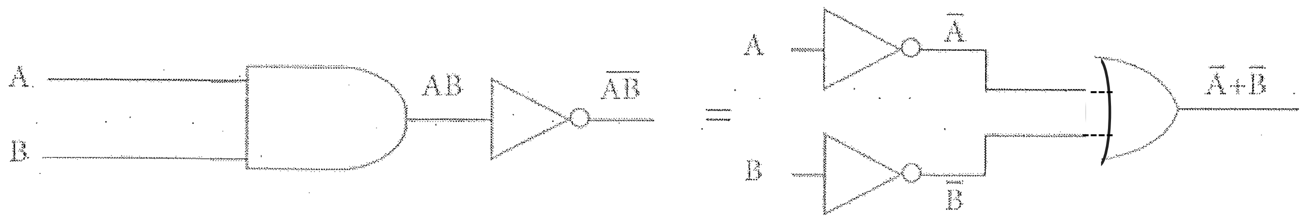
| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**CIRCUIT DIAGRAM:**

DeMorgan's first theorem :$\overline{A+B} = \bar{A} \cdot \bar{B}$



DeMorgan's Second theorem: $\overline{A \cdot B} = \bar{A} + \bar{B}$



**OBSERVATION:**

De-Morgan's first theorem,

| A | B | $\overline{A+B}$ | $\bar{A}.\bar{B}$ |
|---|---|---|---|
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |

De-Morgan's second theorem

| A | B | $\overline{A.B}$ | $\bar{A}+\bar{B}$ |
|---|---|---|---|
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |

# 4. VERIFICATION OF DE MORGAN'S LAWS

**AIM:** To verify De Morgan's first and second theorems.

**COMPONENTS REQUIRED**: Power Supply (0 – 5V), IC 7400, 7408, 7432, 7404, and 7402, Digital IC trainer kit, connecting wires.

**THEORY:**

De Morgan theorem states that

DeMorgan's first theorem :$\overline{A + B} = \bar{A} \cdot \bar{B}$

DeMorgan's Second theorem: $\overline{A \cdot B} = \bar{A} + \bar{B}$

**PROCEDURE:**

i) Verification of De Morgan's first theorem

- The connections are made for LHS $\overline{A + B}$ of the theorem as shown in the circuit diagram using appropriate ICs.
- The output is noted and tabulated for all combinations of logical inputs of the truth table.
- The same procedure is repeated for RHS $\bar{A} \cdot \bar{B}$ of the theorem.
- From the truth table, it can be shown that $\overline{A + B} = \bar{A} \cdot \bar{B}$

ii) Verification of De Morgan's second theorem

- The connections are made for LHS $\overline{A \cdot B}$ of the theorem as shown in the circuit diagram using appropriate ICs.
- The output is noted and tabulated for all combinations of logical inputs of the truth table.
- The same procedure is repeated for RHS $\bar{A} + \bar{B}$ of the theorem.
- From the truth table, it can be shown that :$\overline{A \cdot B} = \bar{A} + \bar{B}$

**RESULT:**

Thus De Morgan's first and second theorems were verified.

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

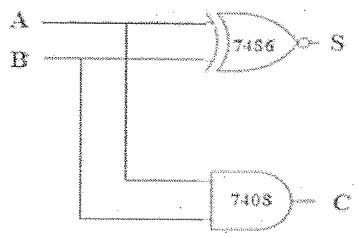
| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## HALF-ADDER:

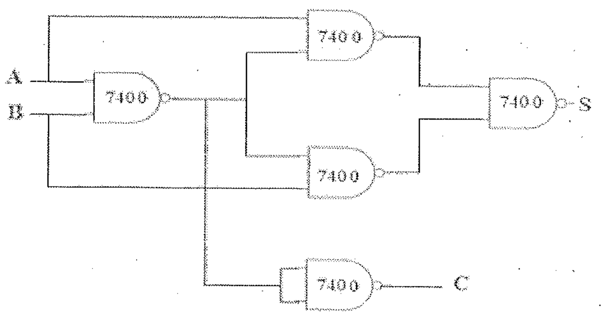
TRUTH TABLE

| INPUTS | | OUTPUTS | |
|---|---|---|---|
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Basic gates



NAND gates

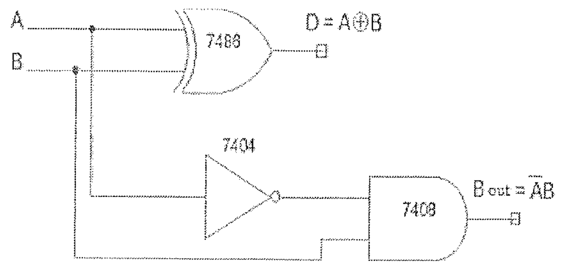


## HALF-SUBTRACTOR

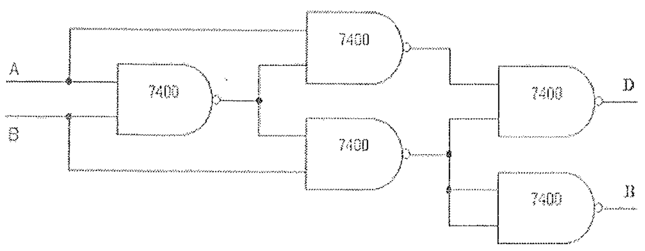
TRUTH TABLE

| INPUTS | | OUTPUTS | |
|---|---|---|---|
| A | B | D | $B_{out}$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Logic circuit using XOR and Basic gates:



Logic circuit using NAND gates:



# 5. HALF ADDER AND HALF SUBTRACTOR

**AIM:** To design, implement and verification of the truth tables of Half adder and Half subtractor circuits.

**COMPONENTS REQUIRED:** Logic trainer kit, NAND gates (IC 7400), XOR gates (IC 7486), AND gates (IC 7408), wires

**THEORY:**

Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

$$S = A \oplus B \qquad C = A\,B$$

Half Subtractor: A combinational logic circuit that performs the subtraction of two data bits, A and B, is called a half-subtractor. Subtracting a single-bit binary value B from another A (i.e. A -B) produces a difference bit D and a borrow out bit B-out. The Boolean functions describing the half Subtractor are:

$$D = A \oplus B \qquad Bout = A'\,B$$

**PROCEDURE:**

- Check the components for their working.
- Insert the appropriate IC into the IC base.
- Make connections as shown in the circuit diagram.
- Verify the Truth Table and observe the outputs.

**RESULT:**

Thus the half adder and half subtractor were implemented and truth tables verified.

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## FULL ADDER:

**Logic expressions:**

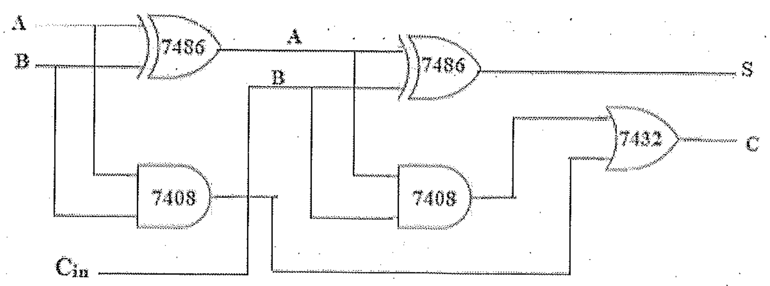
| A \ $BC_{IN}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 |  | 1 |  | 1 |
| 1 | 1 |  | 1 |  |

$S = A \oplus B \oplus C_{IN}$

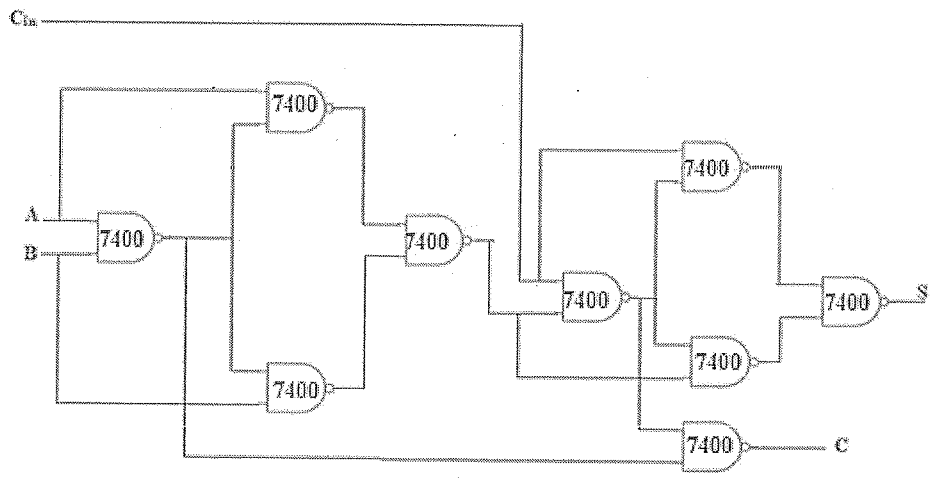
| A \ $BC_{IN}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 |  |  | 1 |  |
| 1 |  | 1 | 1 | 1 |

$C_{OUT} = AB + BC_{IN} + AC_{in}$

**BASIC GATES**



**NAND GATES**



Observations:

| A | B | C | SUM | CARRY |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 6. FULL ADDER AND FULL SUBTRACTOR

**AIM:** To design, implement and verification of the truth tables of full adder and full subtractor circuits.

**COMPONENTS REQUIRED:** Logic trainer kit, NAND gates (IC 7400), XOR gates (IC 7486), AND gates (IC 7408), wires.

**THEORY:**

**Full Adder:**

Full Adder is a logical circuit, which performs addition of three bits (i.e. addition of two bits with previous carry) and provides an output with a Sum and Carry. It can be built using 2-halfadders and an OR gate.

**TRUTH TABLE:**

**Full adder:**

| INPUT | | | OUTPUTS | |
|---|---|---|---|---|
| A | B | $C_{IN}$ | S (Sum) | $C_{OUT}$ (Carry) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Full Subtractor**: It is a combinational logic circuit, which performs Subtraction of three bits and provides an output with a Difference and Borrow, It can be built using 2-half Subtractors and an OR gate.

| INPUTS | | | OUTPUTS | |
|---|---|---|---|---|
| A | B | $B_{IN}$ | D (Difference) | $B_{OUT}$ (Borrow) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

FULL SUBTRACTOR:

**Logic Expressions:**

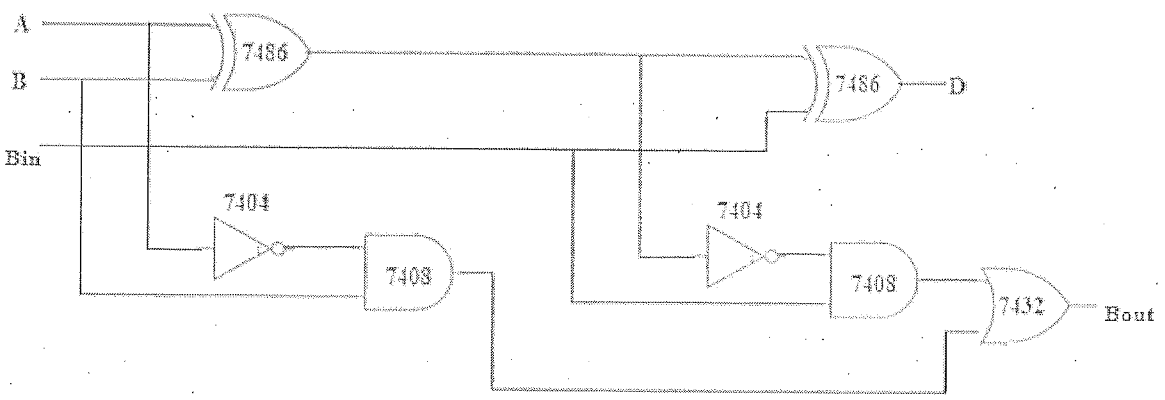
| A \ B $B_{IN}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 |  | 1 |  | 1 |
| 1 | 1 |  | 1 |  |

$D = A \oplus B \oplus B_{IN}$

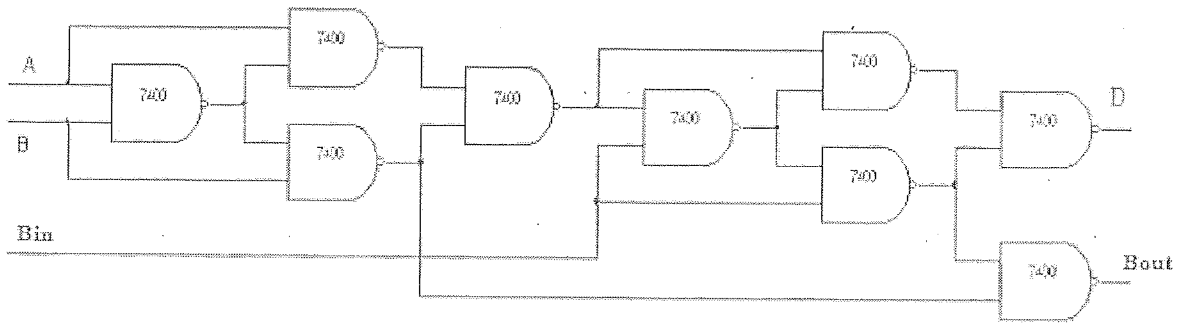
| A \ B $B_{IN}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 |  | 1 |  |  |
| 1 | 1 | 1 | 1 |  |

$B_{OUT} = A'B + A'B_{IN} + BinB$

**Logic circuit using Basic gates and XOR gates:**



Logic circuit using NAND Gates:



OBSERVATIONS:

| A | B | C | DIFFERENCE | BORROW |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**PROCEDURE:**

- Check the components for their working.
- Insert the appropriate IC into the IC base.
- Make connections as shown in the circuit diagram.
- Verify the Truth Table and observe the outputs.

**RESULT:**

Thus the half adder and half subtractor were implemented and truth tables verified.

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**Circuit implementation:**

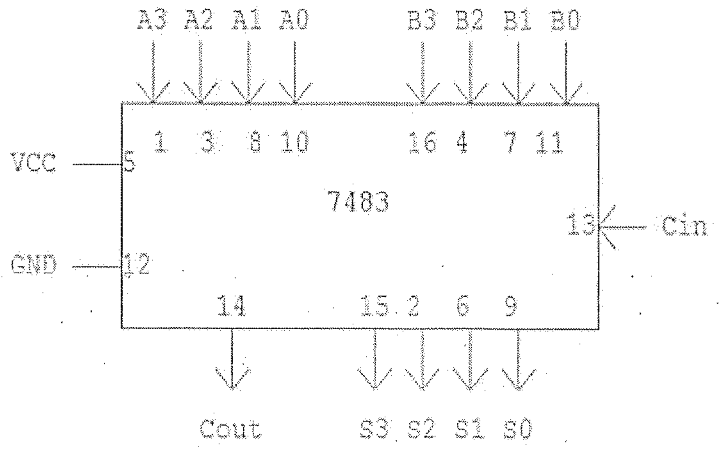
**4 Bit Binary Adder:**

An Example: 7+2= 9

7 is realized at $A_3$ $A_2$ $A_1$ $A_0$ = 0111

2 is realized at $B_3$ $B_2$ $B_1$ $B_0$ = 0010

Sum = 1001



**Truth table:**

| Inputs A | | | | | Inputs B | | | | | Sum | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A4 | A3 | A2 | A1 | | B4 | B3 | B2 | B1 | | S4 | S3 | S2 | S1 |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

# 7. PARALLEL ADDER / SUBTRACTOR

**AIM:** To design and verify the operation of four(4) bit binary adder and Subtractor.

**COMPONENTS REQUIRED:** IC 7483, IC 7486 trainer kit, patch cords.

**THEORY:**

Parallel adders are ripple carry type in which the carry output of each full adder stage is connected to the carry input of the next higher order stage.

The 7483 IC is a 4-bit parallel adder used to perform subtraction as well, the two 4-bit binary numbers $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$ are added to give a sum $S_4S_3S_2S_1$ and carry (Cout) when carry in ($C_{IN}$) and S is 0.The 1's compliment subtraction is carried out when carry out (Cout) connected to Cin and S= 1, The 2's compliment subtraction is carried out when carry in ($C_{IN}$) and S is 1.

**PROCEDURE:**

1. Connections are made as shown in the logic diagram using the pin details of the gates and IC 7483.

2. Connect Vcc & GND to respective pins of ICs

3. Switch on the Trainer kit.

4. Apply the inputs and observe the outputs.

5. Compare the practical value with the theoretical value.

**RESULT:**

Thus the operation of Parallel adder and Parallel Subtractor were studied and verified using IC 7483.

**4 Bit Binary Subtractor:**

Subtraction is carried out by adding 2's complement of the subtrahend. Example: 8 – 3 = 5 (0101)

| | |
|---|---|
| 8 is realized at $A_3$ $A_2$ $A_1$ $A_0$ | = 1000 |
| 3 is realized at $B_3$ $B_2$ $B_1$ $B_0$ through X-OR gates | = 0011 |
| Output of X-OR gate is 1's complement of 3 | = 1100 |
| 2's Complement can be obtained by adding Cin | =1 |

Therefore

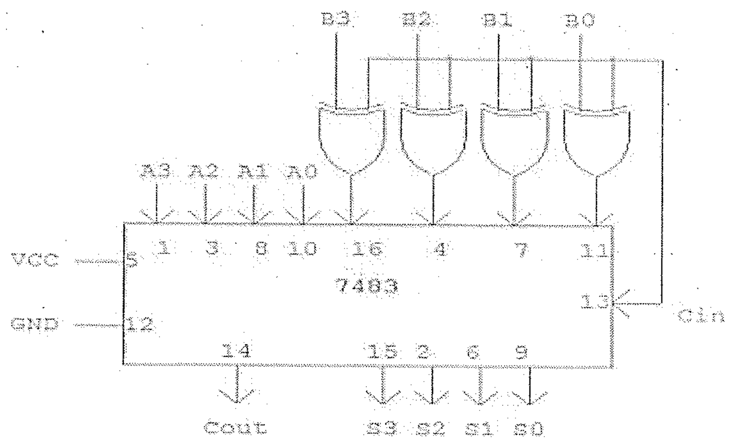
Cin = 1

$A_3$ $A_2$ $A_1$ $A_0$ = 1 0 0 0

$B_3$ $B_2$ $B_1$ $B_0$ = 1 1 0 0

$S_3$ S2 S1 $S_0$ = 0 1 0 1

Cout = 1 (Ignored)



**Truth table:**

| Inputs A | | | | Inputs B | | | | Difference | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A4 | A3 | A2 | A1 | B4 | B3 | B2 | B1 | S4 | S3 | S2 | S1 |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

# MICROPROCESSORS

## 8086

## Assembly Language Programming

38,a

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 1a. 8 BIT ADDITION

**AIM:** To write an assembly language program for Addition and subtraction of two 8-bit numbers.

APPARATUS: MASM software, PC

PROGRAM:

8 Bit addtion:

```
          ASSUME CS:CODE,  DS:DATA

          DATA SEGMENT

            A DB ?
            B DB ?
            RES DB ?
            C  DB ?

          DATA  ENDS

          CODE SEGMENT
          START:

              MOV  AX,DATA
              MOV  DS,AX
              MOV  AL,A
              MOV  BL,B
              ADD  AL,BL
              MOV  RES,AL
              MOV C,1H
              JC DOWN
              MOV C,0H
      DOWN:  INT  03H
              CODE  ENDS
              END  START
              END
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 1b. 8 BIT SUBTRACTION

**AIM:** To write an assembly language program for subtraction of two 8-bit numbers.

APPARATUS: MASM software, PC

PROGRAM:

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
  A DB ?
  B DB ?
  RES  DB ?
DATA  ENDS

CODE SEGMENT
START:
  MOV  AX,DATA
  MOV  DS,AX
  MOV  AL,A
  MOV  BL,B
  SUB  AL,BL
  MOV  RES,AL
  INT  03H
  CODE  ENDS
  END  START
  END
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 2a. 16 BIT ADDITION

**AIM**: To write an assembly language program for Addition of two 16-bit numbers.

APPARATUS: MASM software, PC

PROGRAM:

```
                  ASSUME CS:CODE,  DS:DATA

                  DATA SEGMENT
                    A DW ?
                    B DW ?
                    RES  DW ?
                    C   DB ?
                  DATA  ENDS

                  CODE SEGMENT
                  START:
                      MOV  AX,DATA
                      MOV  DS,AX
                      MOV  AX,A
                      MOV  BX,B
                      ADD  AX,BX
                      MOV  RES,AX
                      MOV C,1H
                      JC DOWN
                      MOV C,0H
             DOWN:  INT  03H
                      CODE  ENDS
                      END  START
                      END
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# 2b. 16 BIT SUBTRACTION

**AIM:** To write an assembly language program for subtraction of two 16-bit numbers.

APPARATUS: MASM software, PC

PROGRAM:

```
ASSUME CS:CODE,  DS:DATA

DATA SEGMENT
  A DW ?
  B DW ?
RES  DW ?
DATA  ENDS

CODE SEGMENT
START:
  MOV  AX,DATA
  MOV  DS,AX
  MOV  AX,A
  MOV  BX,B
  SUB  AX,BX
  MOV  RES,AX
  INT  03H
  CODE  ENDS
  END  START
  END
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 3. BCD ADDITION

**AIM:** To write an assembly language program for addition of two 8 bit BCD numbers.

APPARATUS: MASM software, PC

**PROGRAM:**

```
        ASSUME CS: CODE,DS:DATA

        DATA SEGMENT

                A DB ?
                B DB ?
                RES DB ?
                C DB ?

        DATA ENDS

        CODE SEGMENT
        START:

                MOV AX,DATA
                 MOV DS,AX
                 MOV AL, A
                 MOV BL, B
                 ADD AL, BL
                 DAA
                 MOV RES,AL
                 MOV C,1H
                 JC DOWN
                 MOV C,0H
       DOWN:     INT  03H
                 CODE ENDS
                 END START
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| | | |
|---|---|---|
| Category | Marks awarded | Remarks: |
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 4. BCD SUBTRACTION

**AIM:** To write an assembly language program for subtraction of two 8 bit BCD numbers.

APPARATUS: MASM software, PC

**PROGRAM:**

```
ASSUME CS: CODE,DS:DATA

DATA SEGMENT

        A DB ?
        B DB ?
        RES DB ?

DATA ENDS

CODE SEGMENT
START:
        MOV AX,DATA
          MOV DS,AX
          MOV AL, A
           MOV BL, B
          SUB  AL, BL
          DAS
          MOV RES,AL
          INT 3
          CODE ENDS
          END START
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# 5. 8 BIT MULTIPLICATION

**AIM:** To write an assembly language program for multiplication of two 8-bit unsigned binary numbers.

APPARATUS: MASM software, PC

PROGRAM:

```
ASSUME CS:CODE,  DS:DATA

DATA SEGMENT
  A DB ?
  B DB ?
RES  DW ?
DATA  ENDS

CODE SEGMENT
START:
  MOV  AX,DATA
  MOV  DS,AX
  MOV  AL,A
  MOV  BL,B
  MUL BL
  MOV  RES,AX
  INT  03H
  CODE  ENDS
  END  START
  END
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

**RESULT:**

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 6.8 BIT DIVISION

AIM: To write an assembly language program for division of two 8 bit numbers.

APPARATUS: MASM software, PC

PROGRAM:

```
          ASSUME CS:CODE,  DS:DATA

          DATA SEGMENT

            A DW ?
            B DB ?
            RES DW ?

          DATA  ENDS

          CODE SEGMENT

          START:

            MOV  AX,DATA
            MOV  DS,AX
            MOV  AX,A
            MOV  BL,B
            DIV  BL
            MOV  RES,AX
            INT  03H
            CODE  ENDS
            END  START
            END
```

OUTPUT:

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |

RESULT:

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# 7. SEARCHING FOR AN ELEMENT IN AN ARRAY

**AIM:** To write an assembly level language for searching an element in an array.

**APPARATUS:** MASM Software, PC

**PROGRAM:**

```
            ASSUME CS: CODE,DS:DATA

            DATA SEGMENT
            LIST DB 19H, 99H, 45H, 46H, 54H
            COUNT  EQU  05H
            SEARCH DB 45H
            RES DB ?
            DATA ENDS

            CODE SEGMENT
            START:
            MOV AX,DATA
            MOV DS, AX
            LEA SI, LIST
            XOR CH,CH
            MOV CL, COUNT
            MOV AL, SEARCH
        UP: CMP AL, [SI]
            JZ SKIP
            JMP DOWN
    SKIP:   INC CH
 DOWN:      INC SI
            DEC CL
            JNZ UP
            MOV RES,CH
            INT 03H
            CODE ENDS
            END START
```

**OUTPUT:**

| INPUT | | OUTPUT |
|---|---|---|
| **Address of the memory location** | **DATA** | |
| | | |
| | | |
| | | |
| | | |
| | | |

**RESULT:**

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 8a. STRING SORTING (Ascending Order)

**AIM**: To write an assembly language program for arranging a string in ascending order.

**APPARATUS**: MASM Software, PC

**PROGRAM:**

```
ASSUME CS: CODE, DS: DATA

        DATA SEGMENT
                LIST DW 9234H,1325H,4119H,5B02H
                COUNT EQU 04H
        DATA ENDS

        CODE SEGMENT
        START:
                    MOV AX, DATA
                    MOV DS, AX
                    MOV DX, COUNT-1
        AGAIN0:     MOV CX, DX
                    MOV SI, OFFSET LIST
        AGAIN1:     MOV AX, [SI]
                    CMP AX, [ SI+2]
                    JC DOWN
                    XCHG [SI+2], AX
                    XCHG [SI], AX
        DOWN:       ADD SI,02H
                    LOOP AGAIN1
                    DEC DX
                    JNZ AGAIN0
                    INT 03
                    CODE ENDS
                    END START
```

**OUTPUT:**

| INPUT | | OUTPUT | |
|---|---|---|---|
| MEMORY LOCATION | DATA | MEMORY LOCATION | DATA |
| | | | |
| | | | |
| | | | |
| | | | |

**RESULT:**

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 8b. STRING SORTING (Descending order)

**AIM:** To write an assembly language program for arranging a string in descending order.

**APPARATUS:** MASM Software, PC

**PROGRAM:**

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT
        LIST DW 9234H,1325H,4119H,5B02H
        COUNT EQU 04H
DATA ENDS

CODE SEGMENT
START:
             MOV AX, DATA
             MOV DS, AX
             MOV DX, COUNT-1
AGAIN0:      MOV CX, DX
             MOV SI, OFFSET LIST
AGAIN1:      MOV AX, [SI]
             CMP AX, [ SI+2]
             JNC DOWN
             XCHG [SI+2], AX
             XCHG [SI], AX
 DOWN:       ADD SI,02H
             LOOP AGAIN1
             DEC DX
             JNZ AGAIN0
             INT 03
             CODE ENDS
             END START
```

**OUTPUT:**

| INPUT | | OUTPUT | |
|---|---|---|---|
| MEMORY LOCATION | DATA | MEMORY LOCATION | DATA |
| | | | |
| | | | |
| | | | |
| | | | |

**RESULT:**

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 9a. FINDING LARGEST ELEMENT IN AN ARRAY

**AIM:** - To write an assembly language program for finding the largest element in an array.

**APPARATUS:** MASM SOFTWARE, PC

**PROGRAM:**

```
                ASSUME CS: CODE, DS: DATA

                DATA SEGMENT
                LIST DB  47H, 50H, 0FFH, 0CDH, 12H
                SMALLEST DB ?
                DATA ENDS

                CODE SEGMENT
                START:
                MOV AX, DATA
                MOV DS, AX

                MOV CL,04H
                MOV SI, OFFSET LIST
                MOV AL,[SI]
           UP:  INC SI
                CMP AL,[SI]
                JNC DOWN
                MOV AL,[SI]
        DOWN:   DEC CL
                JNZ UP
                MOV SMALLEST,AL
                INT 3H
                CODE ENDS
                END START
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**RESULT:**

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## 9b. FINDING SMALLEST ELEMENT IN AN ARRAY

**AIM:** - To write an assembly language program for finding the smallest element in an array.

**APPARATUS:** MASM SOFTWARE, PC

**PROGRAM:**

```
            ASSUME CS: CODE, DS: DATA

            DATA SEGMENT
            LIST DB  47H, 50H, 0FFH, 0CDH, 12H
            SMALLEST DB ?
            DATA ENDS

            CODE SEGMENT
            START:
            MOV AX, DATA
            MOV DS, AX
            MOV CL,04H
            MOV SI, OFFSET LIST
            MOV AL,[SI]
      UP:   INC SI
            CMP AL,[SI]
            JC DOWN
            MOV AL,[SI]
    DOWN:   DEC CL
            JNZ UP
            MOV SMALLEST,AL
            INT 3H
            CODE ENDS
            END START
```

**OUTPUT:**

| Address of an operand | Example – 1 | Example - 2 |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

**RESULT:**

## INFERENCES:

## APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |

**THEORETICAL CALCULATIONS:**

| Address | OP Code | Label | Mnem..nics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# 10. MOVE BLOCK

**AIM:** - To write an assembly language program to move a bock of data bytes from one memory location to another for 8086 microprocessor

**APPARATUS:** MASM SOFTWARE, PC

PROGRAM:

```
ASSUME  CS:CODE,DS:DATA,ES:EXTRA

DATA SEGMENT
SRC DB "MICROPROCESSOR"
DATA ENDS

EXTRA SEGMENT
DST DB 0EH DUP(?)
EXTRA ENDS

CODE SEGMENT
START:
        MOV AX,DATA
        MOV DS,AX
        MOV AX,EXTRA
        MOV ES,AX
        LEA SI,SRC
        LEA DI,DST
        MOV CL,0EH
        CLD
        REP MOVSB
        INT 03H
        CODE ENDS
        END START
```

**OUTPUT:**

| INPUT | | OUTPUT | |
|---|---|---|---|
| **Address of the memory location** | **DATA** | **Address of the memory location** | **DATA** |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**RESULT:**

## THEORETICAL CALCULATIONS:

| Address | OP Code | Label | Mnemonics | Operands | Comment |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

INFERENCES:

APPLICATIONS:

(To be filled by faculty)

| Category | Marks awarded | Remarks: |
|---|---|---|
| Experiment Conduction(10) | | |
| Regularity(5) | | |
| Calculations(5) | | |
| Inference(5) | | |
| Viva Voce(5) | | Date of Submission: |
| Total (30): | | Signature of the faculty: |