DIGITAL ELECTRONICS &MICROPROCESSOR LAB MANUAL

Department of Electronics & Communication Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

DIGITAL ELECTRONICS &MICROPROCESSOR LAB MANUAL

Name:_____

H.T.No:_____

Year/Semester:_____

Department of Electronics & Communication Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

VEMU Institute of Technology
Dept. of Electronics and Communication Engineering

Vision of the institute

To be one of the premier institutes for professional education producing dynamic and vibrant force of technocrats with competent skills, innovative ideas and leadership qualities to serve the society with ethical and benevolent approach.

Mission of the institute

Mission_1: To create a learning environment with state-of-the art infrastructure, well equipped laboratories, research facilities and qualified senior faculty to impart high quality technical education.

Mission_2: To facilitate the learners to inculcate competent research skills and innovative ideas by Industry-Institute Interaction.

Mission_3: To develop hard work, honesty, leadership qualities and sense of direction in learners by providing value based education.

Vision of the department

To develop as a center of excellence in the Electronics and Communication Engineering field and produce graduates with Technical Skills, Competency, Quality, and Professional Ethics to meet the challenges of the Industry and evolving Society.

Mission of the department

Mission_1: To enrich Technical Skills of students through Effective Teaching and Learning practices to exchange ideas and dissemination of knowledge.

Mission_2: To enable students to develop skill sets through adequate facilities, training on core and multidisciplinary technologies and Competency Enhancement Programs.

Mission_3: To provide training, instill creative thinking and research attitude to the students through Industry-Institute Interaction along with Professional Ethics and values.

Programme Educational Objectives (PEOs)

PEO 1: To prepare the graduates to be able to plan, analyze and provide innovative ideas to investigate complex engineering problems of industry in the field of Electronics and Communication Engineering using contemporary design and simulation tools.

PEO-2: To provide students with solid fundamentals in core and multidisciplinary domain for successful implementation of engineering products and also to pursue higher studies.

PEO-3: To inculcate learners with professional and ethical attitude, effective communication skills, teamwork skills, and an ability to relate engineering issues to broader social context at work place

Programme Outcomes(Pos)

PO_1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO_2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO_3	**Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO_4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO_5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO_6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO_7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO_8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO_9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO_10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO_11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO_12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcome(PSOs)

PSO_1	**Higher Education :** Qualify in competitive examination for pursuing higher education by applying the fundamental concepts of Electronics and Communication Engineering domains such as Analog & Digital Electronics, Signal Processing, Communication & Networking, Embeded Systems, VLSI Design and Control systems etc.,
PSO_2	**Employment**: Get employed in allied industries through their proficiency in program specific domain knowledge, Specalized software packages and Computer programming or became an entrepreneur.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

II B.Tech. I-Sem (CSE)

(20A04304P) DIGITAL ELECTRONICS &MICROPROCESSOR LAB MANUAL

COURSE OUTCOMES(CO_{S})

CO1	Design any Logic circuit using basic concepts of Boolean Algebra.
CO2	Design any Logic circuit using basic concepts of PLDs.
CO3	Design and develop any application using 8086 Microprocessor.
CO4	Design and develop any application using 8051 Microcontroller

PART A:

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. Karnaugh map 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

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

Dept. of Electronics and Communication Engineering

(20A04303P) DIGITAL ELECTRONICS &MICROPROCESSOR LAB MANUAL

II B.Tech. I-Sem (CSE)

LIST OF EXPERIMENTS TO BE CONDUCTED

HARDWARE EXPERIMENTS

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. Verification of DeMorgan's Laws.
4. Implementation of Half-Adder and Half-Subtractor.
5. Implementation of Full-Adder and Full-Subtractor.
6. Four Bit Binary Adder

MICROPROCESSORS (8086 Assembly Language Programming)

1. 8 Bit Addition and Subtraction.
2. 16 Bit Addition.
3. 8 Bit Multiplication.
4. 8 Bit Division.
5. Sorting in Ascending and Descending Orders.
6. Finding Largest and Smallest Elements from an Array.

ADVANCED EXPERIMENTS:

1. 4-Bit Binary To Gray Code Converter
2. LCM for the given data.

CONTENTS

DOS & DONTS IN LABORATORY

DO's

1. Students should be punctual and regular to the laboratory.
2. Students should come to the lab in-time with proper dress code.
3. Students should maintain discipline all the time and obey the instructions.
4. Students should carry observation and record completed in all aspects.
5. Students should be at their concerned experiment table, unnecessary moment is restricted.
6. Students should follow the indent procedure to receive and deposit the components from lab technician.
7. While doing the experiments any failure/malfunction must be reported to the faculty.
8. Students should check the connections of circuit properly before switch ON the power supply.
9. Students should verify the reading with the help of the lab instructor after completion of experiment.
10. Students must endure that all switches are in the lab OFF position, all the connections are removed.
11. At the end of practical class the apparatus should be returned to the lab technician and take back the indent slip.
12. After completing your lab session SHUTDOWN the systems, TURNOFF the power switches and arrange the chairs properly.
13. Each experiment should be written in the record note book only after getting signature from the lab in charge in the observation notebook.

DON'Ts

1. Don't eat and drink in the laboratory.
2. Don't touch electric wires.
3. Don't turn ON the circuit unless it is completed.
4. Avoid making loose connections.
5. Don't leave the lab without permission.
6. Don't bring mobiles into laboratory.
7. Do not open any irrelevant sites on computer.
8. Don't use a flash drive on computers.

SCHEME OF EVALUATION

S.No	Program	Date	Marks Awarded				Total 30(M)
			Record (10M)	Obs. (10M)	Viva (5M)	Attd. (5M)	
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	Verification of DeMorgan's Laws.						
4	Implementation of Half-Adder and Half-Subtractor.						
5	Implementation of Full-Adder and Full-Subtractor.						
6	Four Bit Binary Adder						
MICROPROCESSORS (8086 Assembly Language Programming)							
7	8 Bit Addition and Subtraction.						
8	16 Bit Addition.						
9	8 Bit Multiplication.						
10	8 Bit Division.						
11	Sorting in Ascending and Descending Orders.						
12	Finding Largest and Smallest Elements from an Array.						
ADVANCED EXPERIMENTS							
1	4-Bit Binary To Gray Code Converter						
2	LCM for the given data						

Signature of Lab In-charge

INTRODUCTION

Digital IC Trainer KIT Operation (Model No: 9002):

Specification : Digital IC Trainer KIT Operation (Model No: 9002) Specification : Digital IC Trainer Kit Model No. 9002 is available with 10 nos. of TTL compatible logic level inputs, TTL logic selectable by a toggle switch, Logic HIGH and logic LOW are displayed by LED, 10 nos of Logic Output indictors, Four crystal generated clock output of 1KHz, 100Hz, 10Hz and 1Hz. Facility for single pulse generation by a push button switch, Logic probe to check logic LOW, logic HIGH and pulse, Four seven segment displays with BCD inputs, Sockets onboard to fix the IC`s : 16pin-4nos. Built-in Power supply : 5V, 1Amp, +12V, 250mA

How to use the IC Trainer Kit? Connect the 230 volts AC power supply and switch 'ON' the Toggle switch 'ON' the left side of the Top Panel, LED will glow. Digital IC Trainer Kit is ready for use. Select the TTL IC to be used for the experiment. Insert the IC properly in the breadboard/ZIF socket (lock the ZIF by moving lever upwards), Know the biasing voltage required for different families of IC's and connect power supply voltage and ground terminals to the respective pins of the IC. Inputs such as logic clock, of different frequency, monopulse, logic levels, BCD inputs, can be selected from the patch panel.

Outputs such as LED indicator, seven segment digital display can be selected depending upon the requirement. Connect the pin connection of IC using wires as per the logic diagram, after verifying connection switch on the supply of IC Trainer Kit and verify the operation the circuit with the help of truth table.

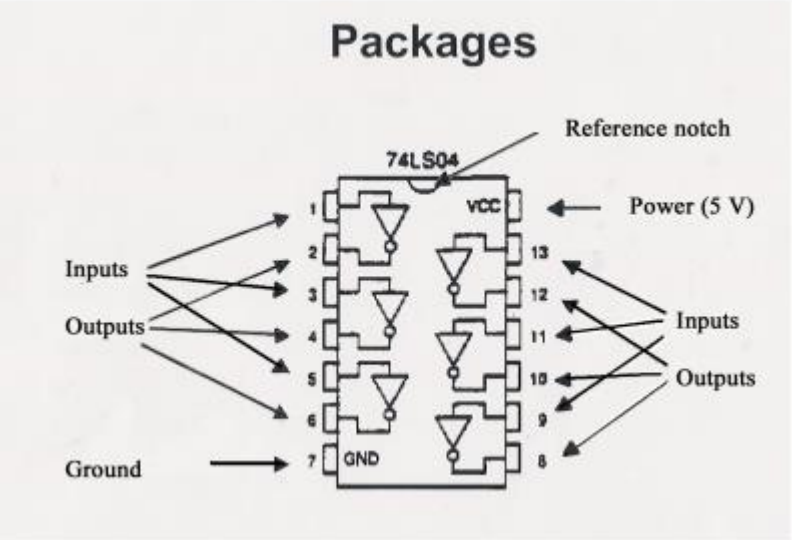
Guide to Assembling your Circuits.

In this section we describe the use of the breadboard and give basic hints about the wiring process needed to power up and interconnect your circuits. Assembling circuits on your breadboard is a fast and easy process once you get used to it. To assemble your circuit first select the chips that you need, insert them in

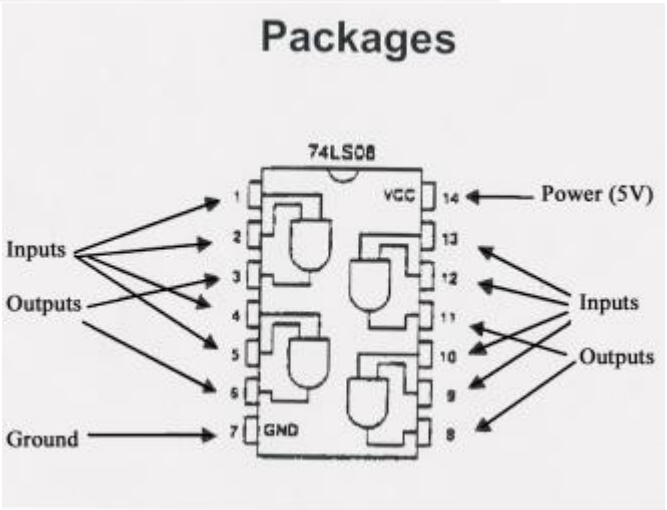
the breadboard, wire up the power and ground connections as described in the next section and next wire the logic elements according to the circuit connections that you obtained from the design process. Before you insert a chip into the breadboard, make sure it is properly oriented (see Figures), and that when you press it down the pins of the chip actually enter the holes and do not bend underneath the chip package. When wiring, be careful to hit the right hole needed in the connection, because this is one of the most common mistakes found to cause an error in your projects.

The chips or packages that will be used to build the experiments belong to the TTL logic family, and they are referred as the 74LSXX family, where the XX is a number that indicates the specific kind of gate or function. The main characteristics for some typical logic gates packages are shown in Figures

TTL Packages Description:



Inverter (NOT) gate pin distribution



AND gate pin distribution

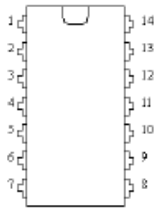
Logic Gates

Digital logic devices are the circuits that electronically perform logic operations on binary variables. binary information is represented by high and low voltage levels, which the device processes electronically. The devices that perform the simplest of the logic operations (such as AND, OR, NAND,etc.) are called gates. For example, an AND gate electronically computes the AND of the voltage encoded binary signals appearing at its inputs and presents the voltage encoded result at its output.

The digital logic circuits used in this laboratory are contained in integrated circuit (IC) packages, with generally 14 or 16 pins for electrical connections. Each IC is labeled (usually with an 74LSxx number) to identify the logic it performs. The logic diagrams and pin connections for these IC's are described in the TTL Data Book by Texas Instruments1.

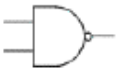
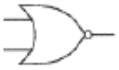
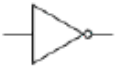
The transistor-transistor logic(TTL) IC's used in this laboratory require a 5.0 volt power supply for operation. TTL inputs require a voltage greater than 2 volts to represent a binary 1 and a voltage less than 0.8 volts to represent a binary 0.

Pin numbering is standard on IC's. Figure 1-1 illustrates the pin numbering for a 14-pin dual in-line package (DIP). With the IC oriented as shown, the numbering starts at the top left and proceeds counterclockwise around the chip:



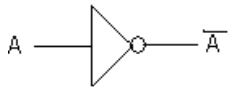
To construct circuits with IC's, a circuit board that allows easy connections to IC pins should be used. The circuit board contains rows of solder less tie points, a 5-volt power supply, a common circuit point (ground), toggle switches for input, and LEDs (light emitting diodes) for output.

LOGIC GATES AND THEIR PROPERTIES

Name	Symbol	Description
AND gate		Output is 1 only if all the inputs are 1. AND gates can have two, three, or more inputs.
NAND gate		"NOT-AND gate"--opposite of AND gate--output is 1 only if all the inputs are NOT 1. Output is only 0 when all the inputs are 1. NAND gates can have more than two inputs.
OR gate		Output is 1 if either of the inputs are 1. Output is only 0 if both of the inputs are 0. OR gates can have more than two inputs.
NOR gate		"NOT-OR gate"--opposite of OR gate--output is only 1 if both of the inputs are 0. If either of the inputs, or both the inputs, are 1, then the output is 0. NOR gates can have more than two inputs.
EX-OR gate		"Exclusive-OR gate"--output is only 1 if either of the inputs are 1, but 0 when both the inputs are 0 or when both the inputs are 1.
EX-NOR gate		"Exclusive-NOR gate"--opposite of EX-OR gate--output is only 1 when both the inputs are 0 or both the inputs are 1. Output is 0 when either of the inputs is 1.
NOT gate		Also known as INVERTER gate.

NOT GATE

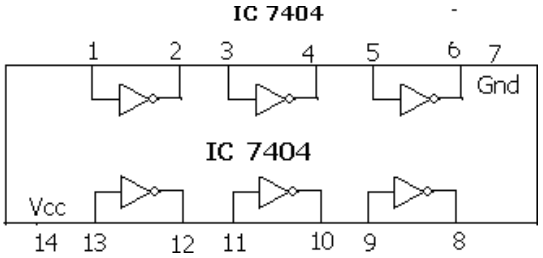
SYMBOL



TRUTH TABLE

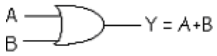
Dec	I/P (A)	O/P $\overline{(A)}$
0	0	1
1	1	0

IC 7404



OR GATE

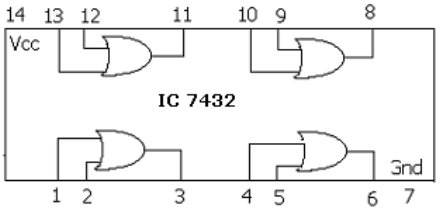
SYMBOL



TRUTH TABLE

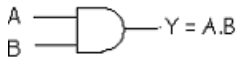
Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

IC 7432



AND GATE

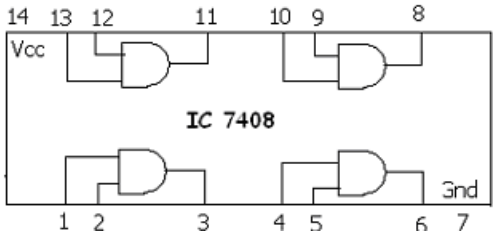
SYMBOL



TRUTH TABLE

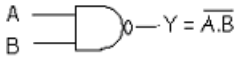
Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

IC 7408



NAND GATE

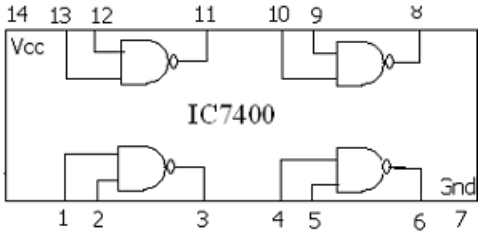
SYMBOL



TRUTH TABLE

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0

IC 7400



EXP. NO : 01 **DATE:**

LOGIC GATES

Aim: Verification of Truth Table for AND, OR, NOT, NAND, NOR and EX-OR gates.

Components Required:

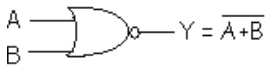
Sl. No	Name of the Gate	IC number	Qty
1	AND gate	7408	2
2	OR gate	7432	2
3	Not gate	7404	2
4	EXOR gate	7486	2
5	NAND gate	7400	2
6	NOR gate	7402	2
7	EX-NOR gate	4077	1
8	Patch chords		few
9	Trainer Kit		

THEORY:

The basic logic gates are the building blocks of more complex logic circuits. These logic gates perform the basic Boolean functions, such as AND, OR, NAND, NOR, Inversion, Exclusive-OR, Exclusive-NOR. Fig. below shows the circuit symbol, Boolean function, and truth. It is seen from the Fig that each gate has one or two binary inputs, A and B, and one binary output, C. The small circle on the output of the circuit symbols designates the logic complement. The AND, OR, NAND, and NOR gates can be extended to have more than two inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

NOR GATE

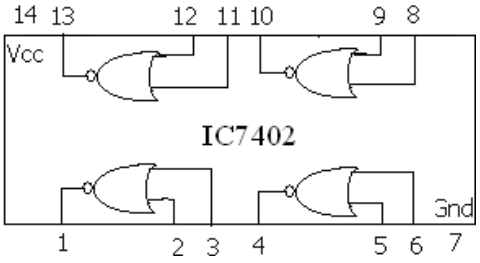
SYMBOL



TRUTH TABLE

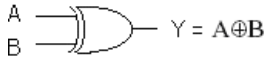
Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

IC7402



XOR GATE

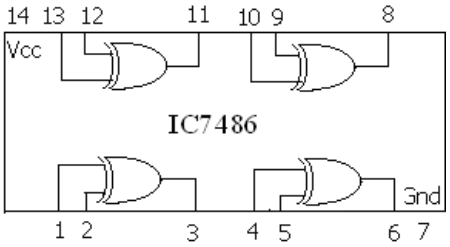
SYMBOL



TRUTH TABLE

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

IC- 7486

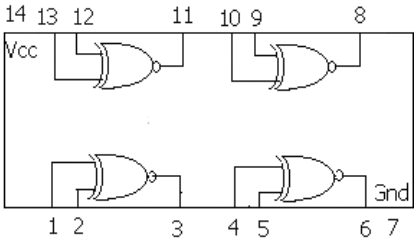


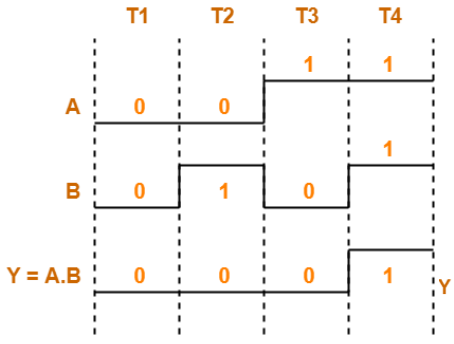
EX-NOR GATE

SYMBOL

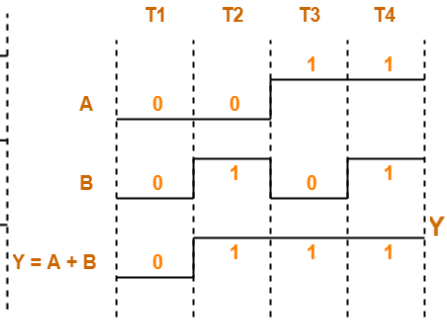
TRUTH TABLE

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1

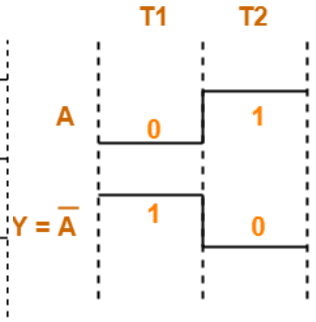




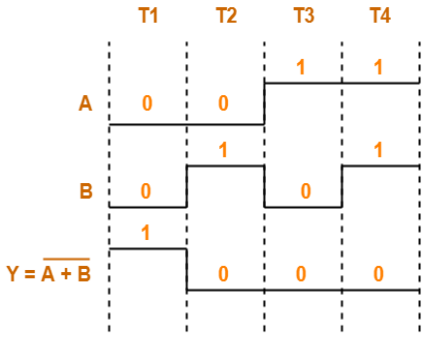
AND Gate Timing Diagram



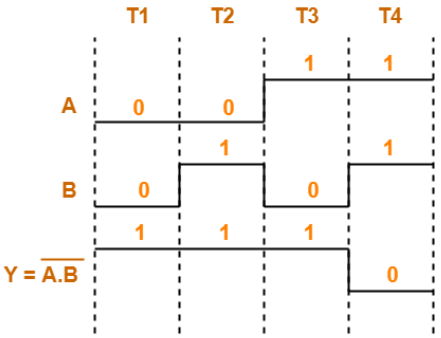
OR Gate Timing Diagram



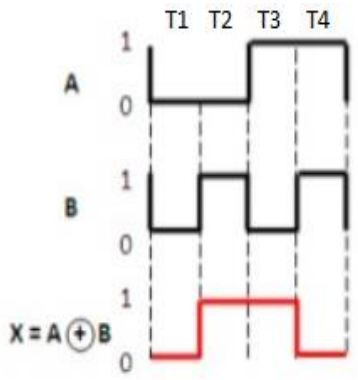
NOT Gate Timing Diagram



NOR Gate Timing Diagram



NAND Gate Timing Diagram



EX-NOR Gate Timing Diagram

PROCEDURE:

1.Connect the IC in AC bread board .

2.Connect Vcc and ground to IC.

3.Set up the circuit one by one and verify their truth table.

4.Observe the output corresponding to input combinations and enter it in truth table.

RESULT:

VIVA QUESTIONS:

1. Why NAND & NOR gates are called universal gates?

2. Realize the EX – OR gates using minimum number of NAND gates?

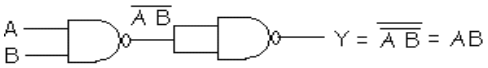
3. Give the truth table for EX-NOR and realize using NAND gates?

4. What is the principle of logic gates?

5. Which is the most commonly used logic family?

NAND gate as **AND** gate using IC7408

Logic Diagram

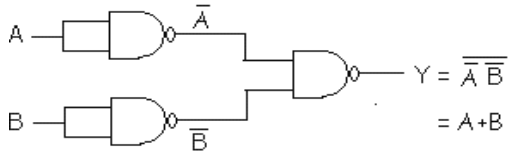


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

NAND gate as **OR** gate using IC7432

Logic Diagram



Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

NAND gate as **NOT** gate using IC7404

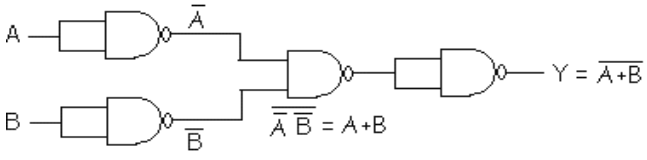
Logic Diagram

Truth Table

Dec Eq	I/P (A)	O/P $\overline{(A)}$
0	0	1
1	1	0

NAND gate as **NOR** gate using IC7402

Logic Diagram

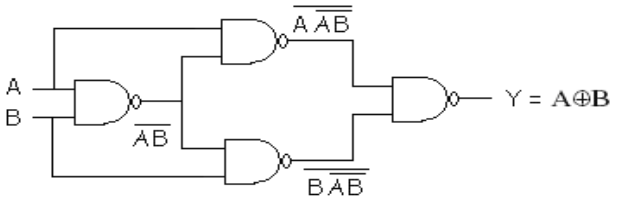


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

NAND gate as **Ex-OR** gate using IC7486

Logic Diagram



Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

$\overline{A\,\overline{A\,B}}\;\overline{B\,\overline{A\,B}} = [\overline{A} + \overline{B}]\;\overline{A\,B} = [A + B][\overline{A} + \overline{B}] = [A\,\overline{B} + \overline{A}\,B] = A \oplus B$

EXP. NO :02 **DATE:**

REALIZATION OF A LOGIC GATES USING NAND AND NOR

AIM: Realization of NOT, AND, OR, EX-OR gates with only NAND and only NOR gates.

Components Required:

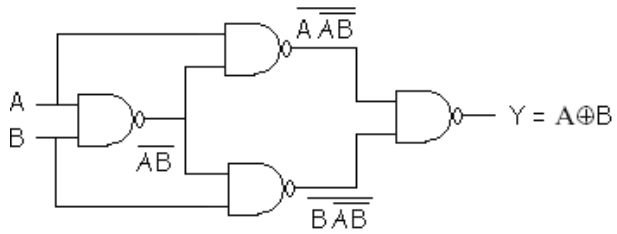
Sl. No	Name of the gate	IC number	Qty
1	AND gate	7408	2
2	OR gate	7432	2
3	NOT gate	7404	2
4	EX-ORgate	7486	2
5	NAND gate	7400	2
6	NOR gate	7402	2
7	EX-NOR gate	4077	1
8	Patch chords		few
9	Trainer Kit		

THEORY:

The basic logic gates are the building blocks of more complex logic circuits. These logic gates perform the basic Boolean functions, such as AND, OR, NAND, NOR, Inversion, Exclusive-OR, Exclusive-NOR. Fig. below shows the circuit symbol, Boolean function, and truth. It is seen from the Fig that each gate has one or two binary inputs, A and B, and one binary output, C. The small circle on the output of the circuit symbols designates the logic complement. The AND, OR, NAND, and NOR gates can be extended to have more than two inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

NAND gate as **Ex-NOR** gate using IC4077

Logic Diagram



$\overline{A\overline{AB}}\ \overline{B\overline{AB}} = [\overline{A}+\overline{B}]\ \overline{AB} = [A+B][\overline{A}+\overline{B}]$

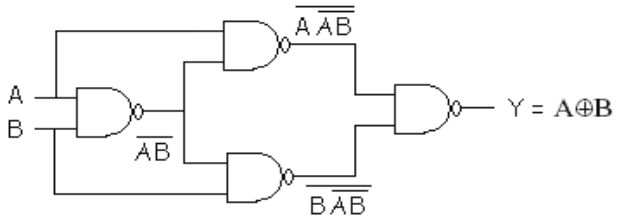
$= [A\overline{B}+\overline{A}B] \quad = A \oplus B$

Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

NAND gate as **Ex-OR** gate using IC7486

Logic Diagram



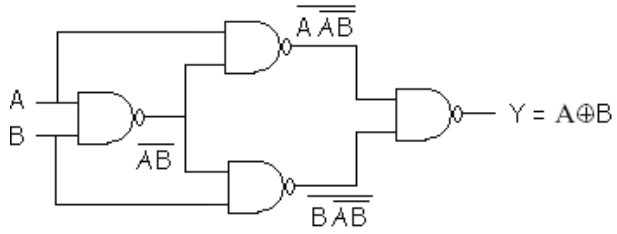
$\overline{A\overline{AB}}\ \overline{B\overline{AB}} = [\overline{A}+\overline{B}]\ \overline{AB} = [A+B][\overline{A}+\overline{B}] = [A\overline{B}+\overline{A}B] = A \oplus B$

Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

NAND gate as **Ex-NOR** gate using IC4077

Logic Diagram



$\overline{A\overline{AB}}\ \overline{B\overline{AB}} = [\overline{A}+\overline{B}]\ \overline{AB} = [A+B][\overline{A}+\overline{B}]$

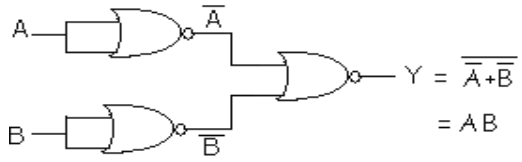
$= [A\overline{B}+\overline{A}B] \quad = A \oplus B$

Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

NOR gate as **AND** gate using IC7408

Logic Diagram

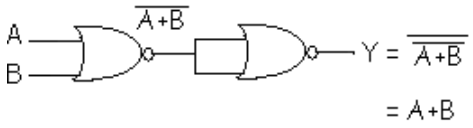


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

NOR gate as **OR** gate using IC7432

Logic Diagram

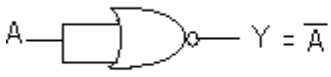


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

NOR gate as **NOT** gate using IC7404

Logic Diagram

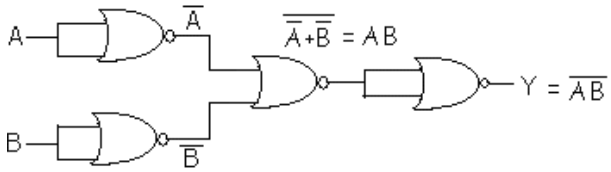


Truth Table

Dec Eq	I/P (A)	O/P $\overline{(A)}$
0	0	1
1	1	0

NOR gate as **NAND** gate using IC7400

Logic Diagram

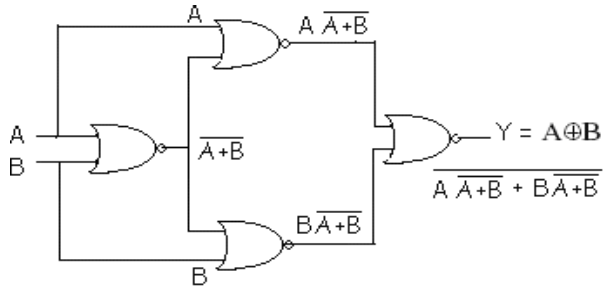


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0

NOR gate as **Ex-NOR** gate using IC4077

Logic Diagram

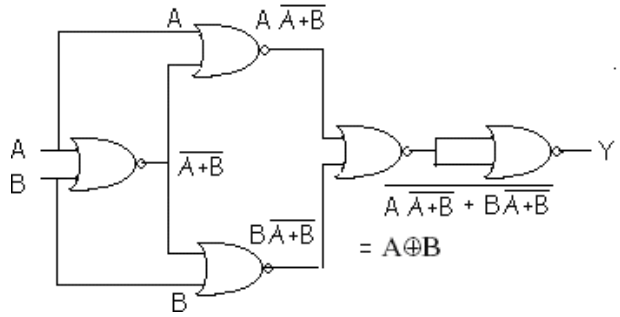


Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

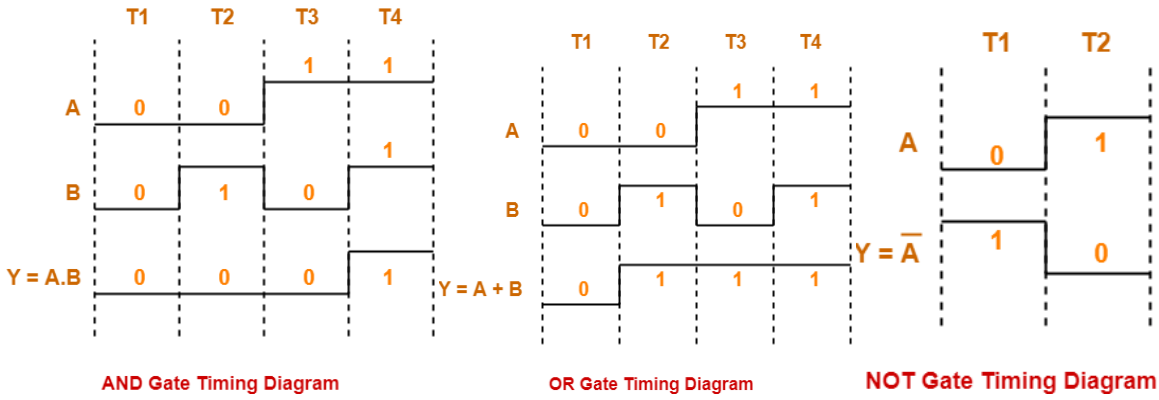
NOR gate as **Ex-OR** gate using IC7486

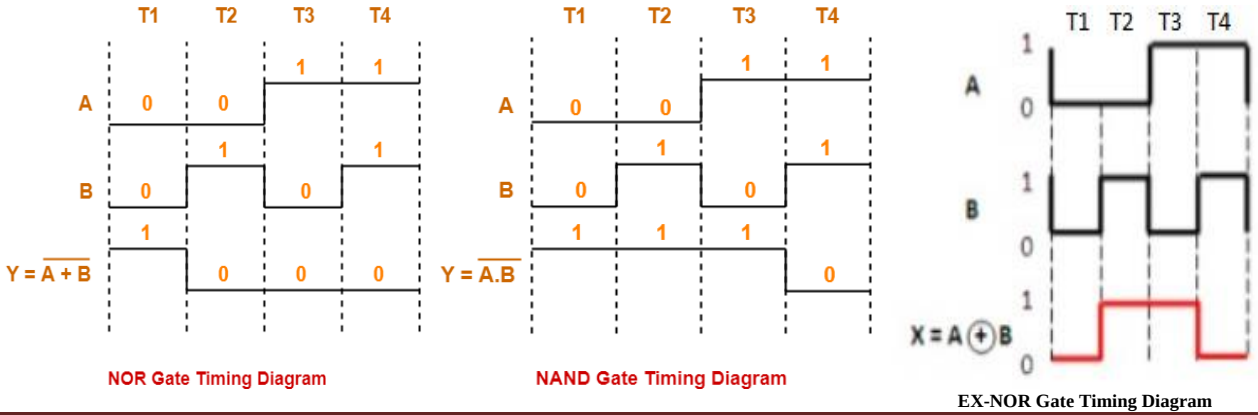
Logic Diagram



Truth Table

Dec Eq	Inputs		Output
	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1





PROCEDURE:

1.Connect the IC in AC bread board .

2.Connect Vcc and ground to IC.

3.Set up the circuit one by one and verify their truth table.

4.Observe the output corresponding to input combinations and enter it in truth table.

RESULT:

VIVA QUESTIONS:

1) What are the different methods to obtain minimal expression?

2) What is a Min term and Max term?

3) What is meant by canonical representation?

4) What is K-map? Why is it used?

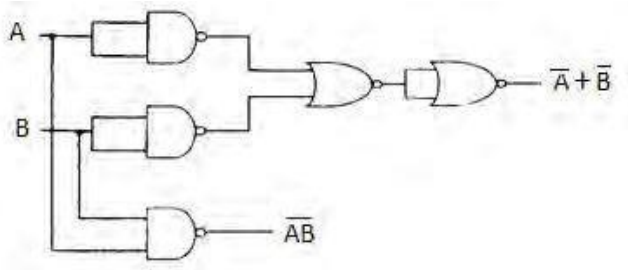
5) What are universal gates?

a) $\overline{A\,B} = \overline{A} + \overline{B}$

TRUTH TABLE:

A	B	$\overline{AB}$	$\overline{A}+\overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

CIRCUIT DIAGRAM:

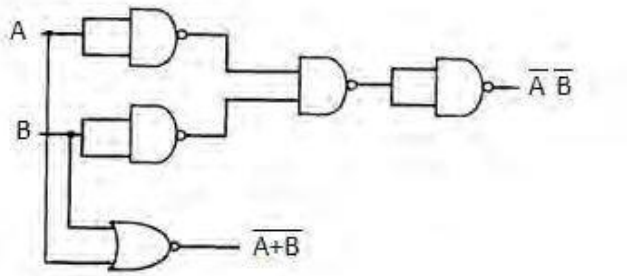


b) $\overline{A + B} = \overline{A}\,\overline{B}$

TRUTH TABLE:

A	B	$\overline{A+B}$	$\overline{A}\,\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

CIRCUIT DIAGRAM:



EXP. NO :03 **DATE:**

DE-MORGAN'S THEOREM

AIM: To verify De-Morgan's theorem for two variables

COMPONENTS REQUIRED: IC Trainer kit, IC 7400, IC 7402

THEORY:

1. De Morgan theorem states that
   a) AB'=A'+B'
   b) (A+B)'=A'.B'

   De-Morgan's theorem is highly useful to simplify the Boolean expression
2. Gates NAND and NOR are known as universal gates, because any logic gates or Boolean expression can be realized by either NAND or NOR gate alone. Each product term in the SOP expression is called minterm and each sum term in the POS expression is called maxterm.SOP expression can be economically realized using NAND gates and POS expression can be economically realized using NOR gates.

PROCEDURE:

1.Connect the IC in AC bread board .

2.Connect Vcc and ground to IC.

3.Set up the circuit one by one and verify their truth table.

4.Observe the output corresponding to input combinations and enter it in truth table.

RESULT:

VIVA QUESTIONS:

1.What is the use of De Morgan's theorem?

2.What are the importance of de Morgan's theorems in Boolean algebra?

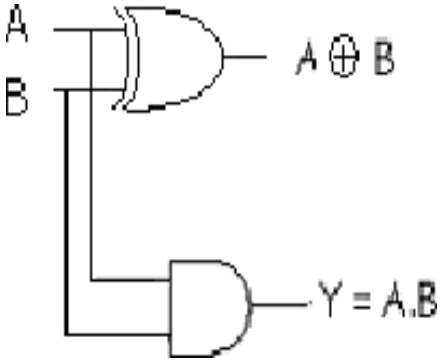
3.What are DeMorgan's theorems prove algebraically the DeMorgan's theorems?

4.How do you remember DeMorgan's law?

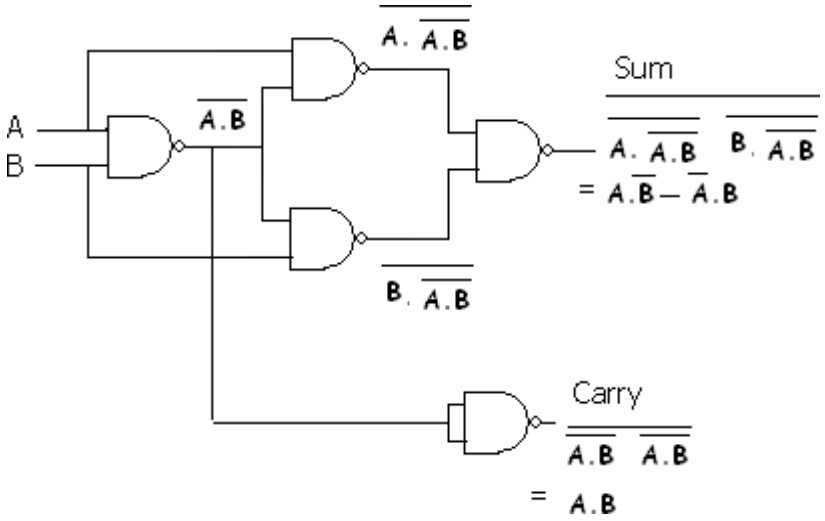
5. What are combinational logic gates?

Half Adder Using Basic Gates

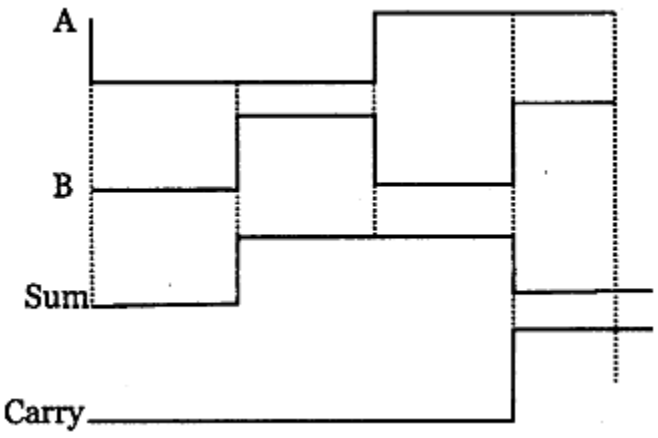
Truth Table				
DEC EQ	INPUTS		OUTPUTS	
	A	B	SUM	CARRY
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1



Half Adder Using NAND Gates



Timing Diagram:



EXP. NO : 04 **DATE:**

HALF ADDER & HALF SUBTRACTOR

Aim: Realization of half adder and half Subtractor using logic gates.

Components Required:

Sl. No	Name of the Component	IC number	Qty
1	AND gate	7408	1
2	OR gate	7432	1
3	Not gate	7404	1
4	EXOR gate	7486	3
5	NAND gate	7400	3
6	NOR gate	7402	3
7	Patch chords		Few
8	Trainer Kit		

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:

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

Half Subtractor: Subtracting a single-bit binary value B from another A (i.e. A -B) producesa difference bit D and a borrow out bit B-out. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half- Subtractor are:

$S = A \oplus B$ $C = A' B$

PROCEDURE:

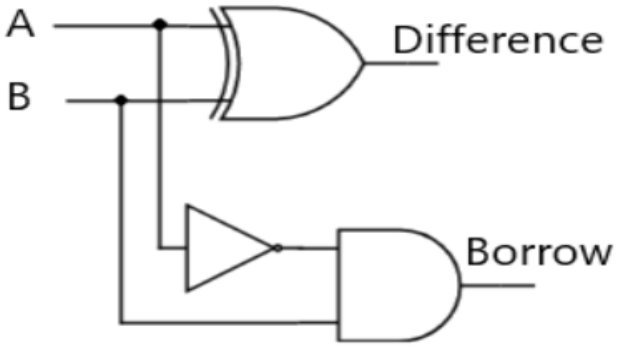
1. Obtain the Boolean Expressions for half adder and half subtractor (sum & Carry) by writing the truth table and simplifying with the help of K-map.
2. Make the connections as shown in the logic diagram.
3. Apply different combinations of inputs according to the truth table and verify the outputs.
4. Repeat the above procedure for all the circuit diagrams.

Half Subtractor Using Basic Gates

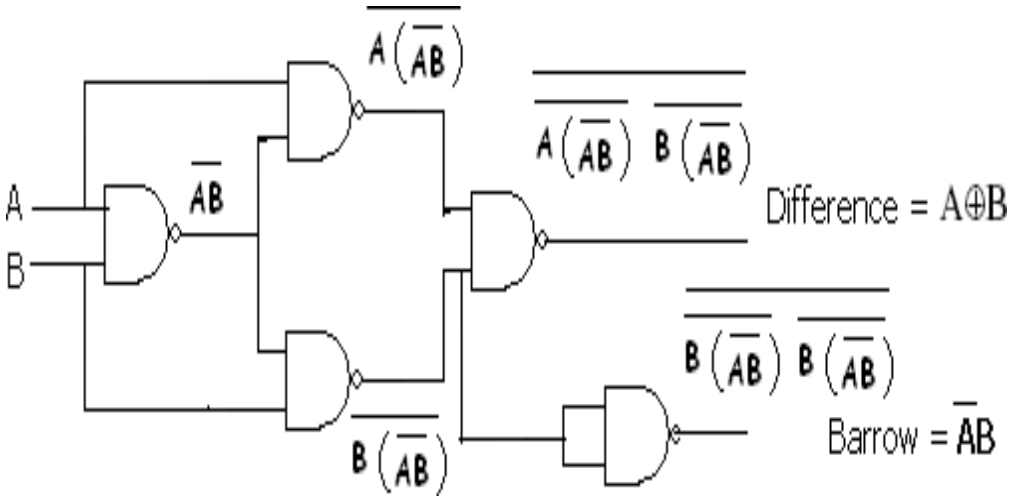
Truth Table

Dec Eq	INPUTS		OUTPUTS	
	A	B	Diff	Barrow
0	0	0	0	0
1	0	1	1	1
2	1	0	1	0
3	1	1	0	0

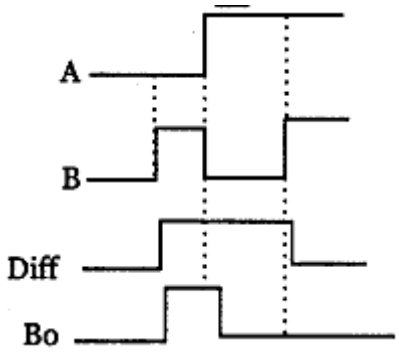
Circuit Diagram



Using NAND gates



Timing Diagram:



RESULT:

VIVA QUESTIONS:

1)What is a half adder?

2)What are the applications of adders?

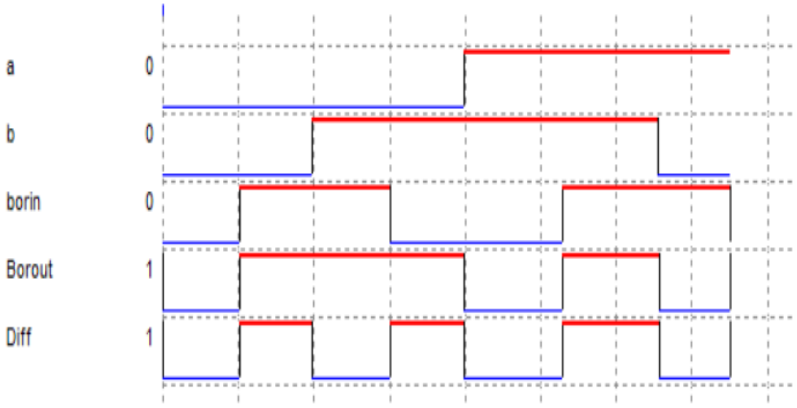
3)What is a half subtractor?

4) What are the applications of subtractors?

5) Realize a full adder using two half adders?

Truth Table

Dec Equi	Inputs			Outputs	
	A	B	Bin	Diff	Borrow
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1



$Diff = \bar{A}\,\bar{B}\,Bin + \bar{A}\,B\,\overline{Bin} + A\,\bar{B}\,\overline{Bin} + A\,B\,Bin$

$= (\bar{A}\,\bar{B} + A\,B)Bin + (\bar{A}\,B + A\,\bar{B})\,\overline{Bin}$

$= (A \odot B)\,Bin + (A \oplus B)\overline{Bin}$

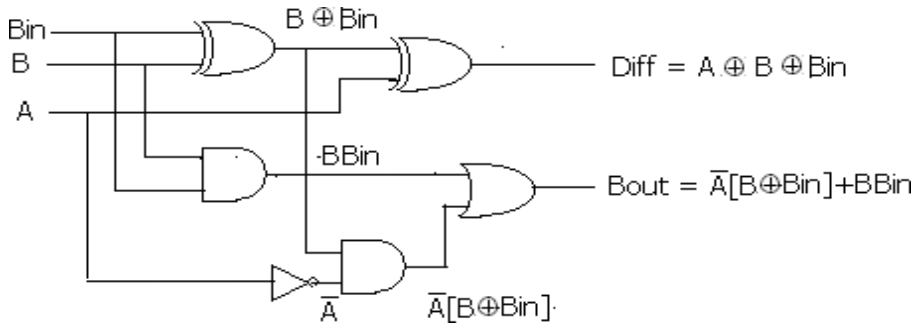
$Diff = A \oplus B \oplus Bin$

$Bout = \bar{A}\,\bar{B}\,Bin + \bar{A}\,B\,\overline{Bin} + \bar{A}\,B\,Bin + A\,B\,Bin$

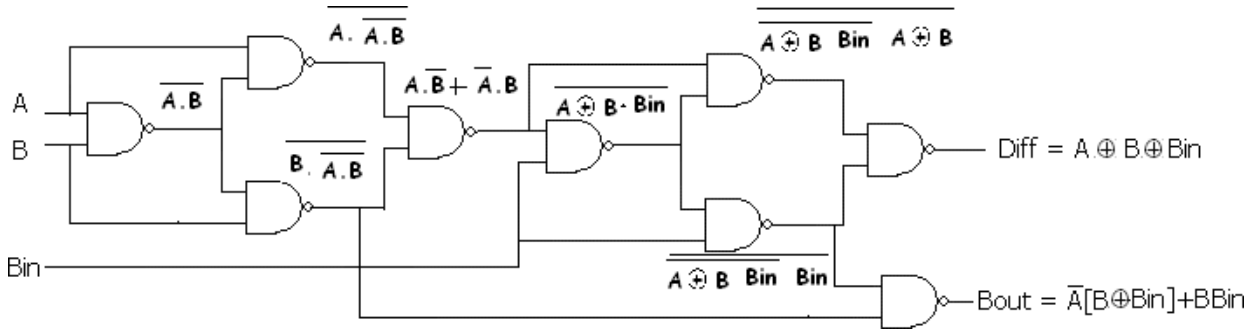
$= \bar{A}\,[\bar{B}\,Bin + B\overline{Bin}] + [A + \bar{A}]\,B\,Bin$

$Bout = \bar{A}\,(B \oplus Bin) + B\,Bin$

Logic Diagram



Using NAND Gates



EXP. NO : 05 **DATE:**

FULL ADDER & FULL SUBTRACTOR

Aim: Realization of full adder and full Subtractor using logic gates.

Components Required:

Sl. No	Name of the Component	IC number	Qty
1	AND gate	7408	1
2	OR gate	7432	1
3	Not gate	7404	1
4	EXOR gate	7486	3
5	NAND gate	7400	3
6	NOR gate	7402	3
7	Patch chords		Few
8	Trainer Kit		

THEORY:

Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit, Cin , is called a full-adder. The Boolean functions describing the full-adder are:

$S = (x \oplus y) \oplus Cin$ $C = xy + Cin\ (x \oplus y)$

Full Subtractor: Subtracting two single-bit binary values, B, Cin from a single-bit value A produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-subtracter are:

$D = (x \oplus y) \oplus Cin$ $Br = A'B + A'(Cin) + B(Cin)$

Procedure:

1.Obtain the Boolean Expressions for full adder and full subtractor (sum & Carry) by writing the truth table and simplifying with the help of K-map.

2.Make the connections as shown in the logic diagram.

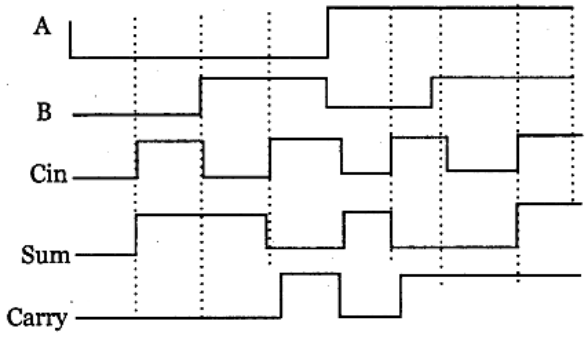
3.Apply different combinations of inputs according to the truth table and verify the outputs.

4.Repeat the above procedure for all the circuit diagrams.

Full Adder:

Truth Table

Dec Eq	Inputs			Outputs	
	A	B	Cin	Sum	Carry
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1



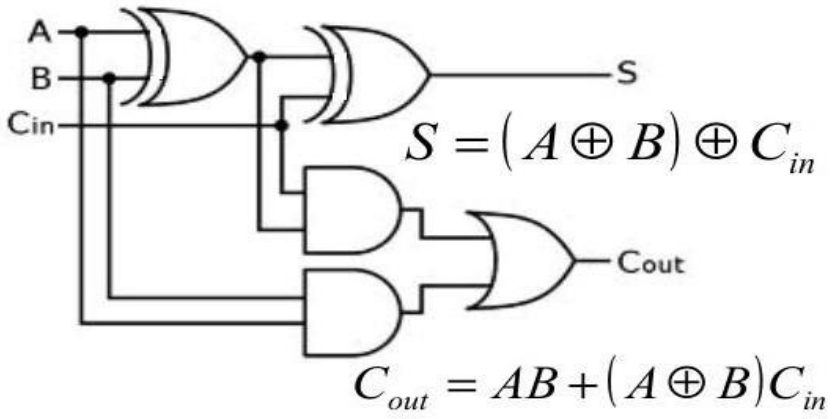
$SUM = \bar{A}\bar{B}Cin + AB\bar{C}in + A\bar{B}\bar{C}in$

$SUM = Cin\,(\bar{A}\bar{B} + AB) + \bar{Cin}\,(\bar{A}B + A\bar{B})$

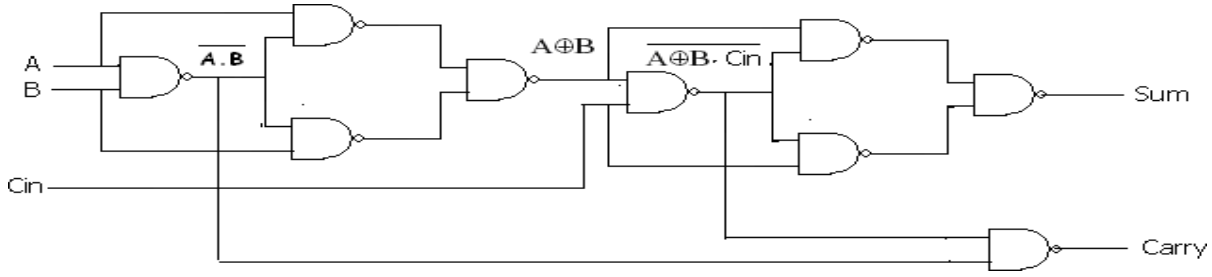
$SUM = Cin\ XOR\ (A\ XOR\ B)$

or $Cout = Cin\ (A\ XOR\ B) + AB$

Logic Diagram Using Basic Gates



Using NAND Gates:



RESULT:

VIVA QUESTIONS:

1)What is a full adder?

2)What is a full subtractor?

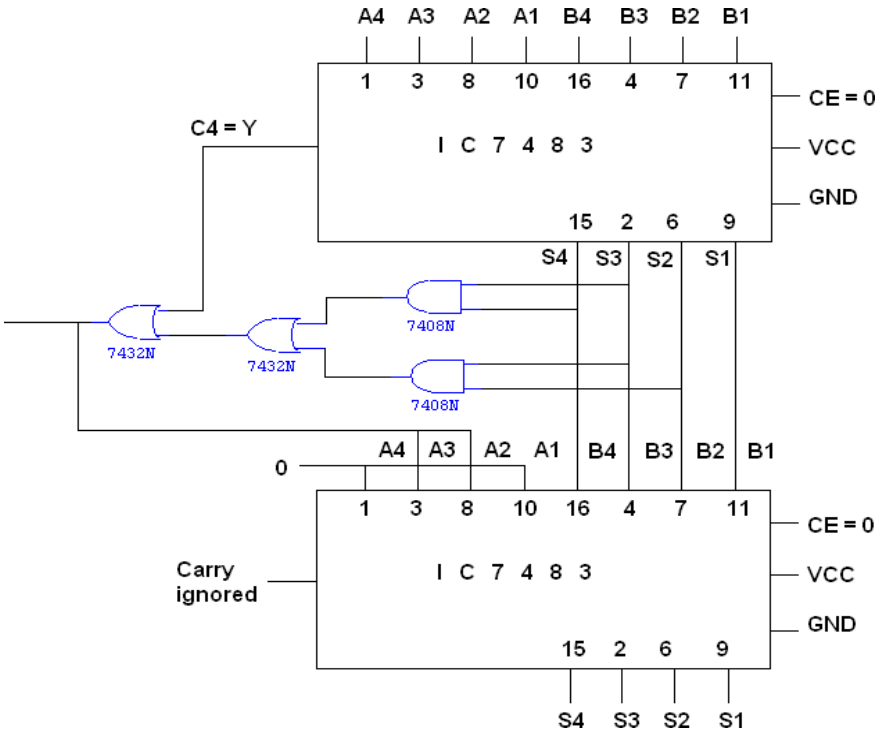
3)Obtain the minimal expression for above circuits?

4)Realize a full adder using two half adders?

5) Realize a full subtractors using two half subtractors?

LOGIC DIAGRAM:

BCD ADDERS:



TRUTH TABLE:

Input Data A				Input Data B				Addition					Subtraction				
A4	A3	A2	A1	B4	B3	B2	B1	C	S4	S3	S2	S1	B	D4	D3	D2	D1
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1	0	1	0	0	1	1	1	1
1	0	1	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1

EXP. NO : 06 **DATE:**

FOUR BIT BINARY ADDER

AIM:

To design and implement 4-bit adder and subtractor using IC 7483.

APPARATUS REQUIRED:

SI.No.	COMPONENT	SPECIFICATION	QTY.
1.	IC	IC 7483	1
2.	EX-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	40

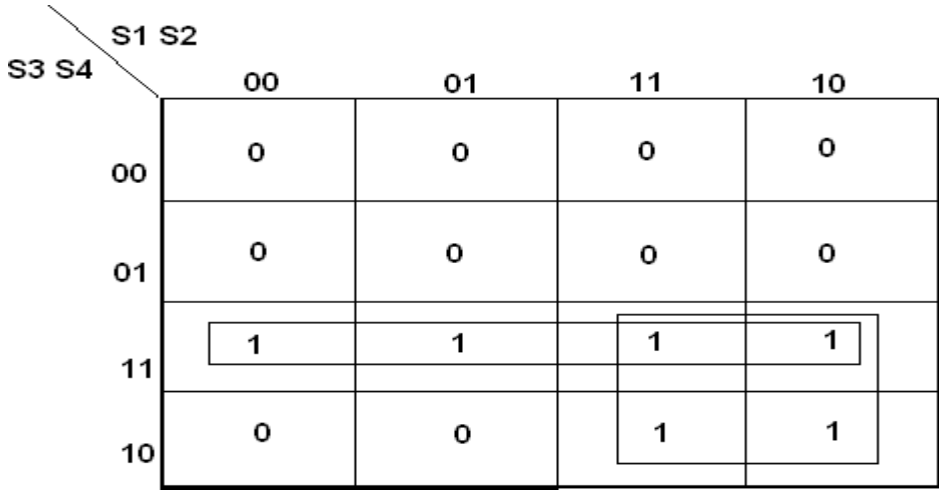
THEORY:

4 BIT BINARY ADDER:

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of next full adder in chain. The augends bits of 'A' and the addend bits of 'B' are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bits. The carries are connected in chain through the full adder. The input carry to the adder is C_0 and it ripples through the full adder to the output carry C_4.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and should appear in the form listed in the columns.ABCD adder that adds 2 BCD digits and produce a sum digit in BCD. The 2 decimal digits, together with the input carry, are first added in the top 4 bit adder to produce the binary sum.

K MAP



Y=S4(S3+S2)

TRUTH TABLE:

BCD SUM				CARRY
S4	S3	S2	S1	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

PROCEDURE:

1.Connect the IC in AC bread board .

2.Connect Vcc and ground to IC.

3.Set up the circuit one by one and verify their truth table.

4.Observe the output corresponding to input combinations and entered it in truth table.

RESULT:

VIVA QUESTIONS:

1.What does a 4-bit adder do?

2.What is an 8 bit adder?

3.What are the two types of basic adder circuits?

4.What is the function of parallel adder?

5.Which IC is used as 4-bit binary adder?

8086 MICROPROCESSOR

INTRODUCTION TO MASM PROGRAMMING

The microprocessor development system consists of a set of hardware and software tools. The hardware of development systems usually contains a standard PC (Personal Computer), printer and an emulator. The software tools are also called program development tools and they are Editor, Assembler, and Library builder, Linker, Debugger and Simulator. These software tools can be run on the PC in order to write, assemble, debug, modify and test the assembly language programs.

EDITOR (TEXT EDITOR):

The Editor is software tool which, when run on a PC, allow the user to type/enter and modify the assembly language program. The editor provides a set of commands for insertion, deletion, modifications of letters, characters, statements, etc., The main faction of an editor is to help the user to constrict the assembly language program in the right format. The program created using editor is known as source program and usually it is saved with file extension "ASM".

ASSEMBLER:

The assembler is a software tool which run on a PC, converts the assembly language program to machine language program. Several types of assemblers are available and they are one pass assembler, two pass assembler, macro assembler, cross assembler, resident assembler and Meta assembler.

One Pass Assembler: In the one pass assembler source code is processed only once, and we can use only backward reference.

Two Pass Assembler: Most of the popularly used assemblers are two pass assembler. In two pass assembler, the first pass is made through source code for the purpose of assigning an address to all the labels and to store this information in a symbol table. The second pass is made to actually translate the source code into machine code.

Some examples of assemblers are TASM (Borland's Turbo Assembler), MASM (Microsoft Macro Assembler), ASM86 (INTEL'S 8086 Assembler), etc,.

TASM:

The Turbo Assembler (TASM) mainly PC-targeted assembler package was Borland's offering in the X86 assembler programming tool market. As one would expect, TASM worked well with Borland's high-level language compilers for the PC, such as Turbo Pascal, Turbo Basic and Turbo C. Along with the rest of the Turbo suite, Turbo Assembler is no longer maintained.

The Turbo Assembler package came bundled with the linker Turbo Linker, and was

interoperable with the Turbo Debugger. For compatibility with the common Microsoft Macro Assembler (MASM), TASM was able to assemble such source code files via its MASM mode. It also had an ideal mode that enabled a few enhancements.

The effective execution of a program in assembly language we need the following

1. MASM assembler
2. NE (Norton's Editor) editor (or) Edlin editor
3. Linker
4. Debug utility of DOS

How to use TASM:

Install the specified TASM software on PC with DOS operating system. The program implementation and its execution are illustrated in four stages, there are

1. Editing of program
2. Assembling the program
3. Linking the program
4. Debugging and execution of the program

C:\ tasm>edit (file name).asm ↵

>tasm (file name).asm ↵ **Assembling**

Turbo assembler version 3.2 copy right (C) 1988, 1992 Borland International
Assembling file : (file name)
Error messages : None
Warning messages : None
Passes : 1

PROGRAM:

Remaining memory : 392K

C:\ tasm>tlink (file name).obj ↵ **Turbo Linking**

Turbo link version 5.1 copy right (C) 1992 Borland International **the Program**

C:\ tasm>debug (file name).exe ↵ **Debugging & Operation**

- r ↵ **of the**

- p ↵ **Program**

- q ↵

DEBUG COMMANDS:

Command	Command Character & Syntax	Description
Assembler	- A [address]	Assembles the instructions at a particular address.
Quit	- q	Quits from debug
Compare	- C range address	Compares two memory ranges
Display	- D range	Displays the contents of memory
Enter	- E address [List]	Enters new or modifies old memory contents
Fill	- F range list	Fills in a range of memory
Go	- G – address	Executes a program in memory
Hex	- H V_1V_2	Adds & Subtracts two hex values (V_1 & V_2)
Load	- L [address] [drive]	Load disk data into memory
Trace	- T	Traces disk data into memory
Un assemble	- U	Un assembles hex bytes into assembler instructions

MASM:

The Microsoft Macro Assembler (abbreviated MASM) is an x86 high-level assembler for DOS and Microsoft Windows. Currently it is the most popular x86 assembler. It supports a wide variety of macro facilities and structured programming idioms, including high-level functions for looping and procedures. Later versions added the capability of producing programs for Windows. MASM is one of the few Microsoft development tools that target 16-bit, 32-bit and 64-bit platforms. Earlier versions were MS-DOS applications. Versions 5.1 and 6.0 were OS/2 applications and later versions were Win32 console applications. Versions 6.1 and 6.11 included Phar Lap's TNT DOS extender so that MASM could run in MS-DOS.

MASM can be used along with a link program to structure the codes generated by MASM in the form of an executable file. This assembler reads the source program as its inputs and provides an object file. The link accepts the object file produced by this MASM assembler as input and produces an EXE file. The effective execution of a program in assembly language we need the following

1. MASM assembler
2. NE (Norton's Editor) editor (or) Edlin editor
3. Linker
4. Debug utility of DOS

LIBRARY BUILDER:

The library builder is used to create library files which are collection of procedures of frequently used functions.

The input to library builder is a set of assembled object of program modules/procedures.

The library builder combines the program modules/procedures into a single file known as library file and it is saved with file extension ".LIB". Some examples of library builder are Microsoft's LIB Borland's TLIB, etc,.

LINKER:

The linker is a software tool which is used to combine releasable object files of program modules and library functions into a single executable file.

The linker also generates a link map file which contains the address information about the linked files. Some examples of linkers Microsoft's linker LINK, Borland's Turbo linker TLINK, etc,.

DEBUGGER:

The debugger is a software tool that allows the execution of a grogram in single step or break-point mode under the control of user. The process of locating and correcting the errors in a program using a debugger is known as debugging.

The debugger tools can help the user to isolate a problem in the program. Once the problem/errors are identified, the algorithm can be modified. Then the user can the editor to correct the source program, reassemble the corrected source program, relink and run the program again.

SIMULATOR:

The simulator is a program which can be run on the development system (Personal computer) to simulate the operations of the newly designed system. Some of the operations that can be simulated are given below.

- Execute a program and display result.
- Single step execution of a person.
- Break – point execution of a program.
- Display the contents of register/memory.

EMULATOR:

An emulator is a combination of hardware and software. It is usually used to test and debug the hardware and software of a newly designed microprocessor based system. The emulator has a multi core cable which connects the PC of development system and the newly designed hardware of microprocessor system.

VARIABLES AND CONSTRAINTS USED IN ASSEMBLERS:

Variables: The variables are symbols (or terms) used in assembly language program statements in order to represent variable data and address. While running a program, a value has to be attached to each variable in the program. The advantage of using variables is that the value of the variable can be dynamically varied while running program.

Rules of Framing Variable names:

1. The variable name can have any of the following characters. A to Z a to z, 0 to 9
@ , _ (underscore).
2. The first character in the variable name should be an alphabet (A to Z or a to z) or an underscore.
3. The length of variable name depends on assembler and normally the maximum length of variable name is 32 characters.
4. The variable name are case insensitive. Therefore the assembler do not distinguish between the upper and lower case letters/alphabets.

Constraints: The decimal, binary or hexadecimal number used to represent the data address in assembly language program statement is called constants or numerical constants. When constants are used to represent the address the address the address/data then their values are fixed and cannot be changed while running a program. The binary, hexadecimal and decimal constants can be differentiated by placing a specific alphabet at the end of the constant.

Example of Valid Constant:

1011 --------- Decimal (BCD) constant

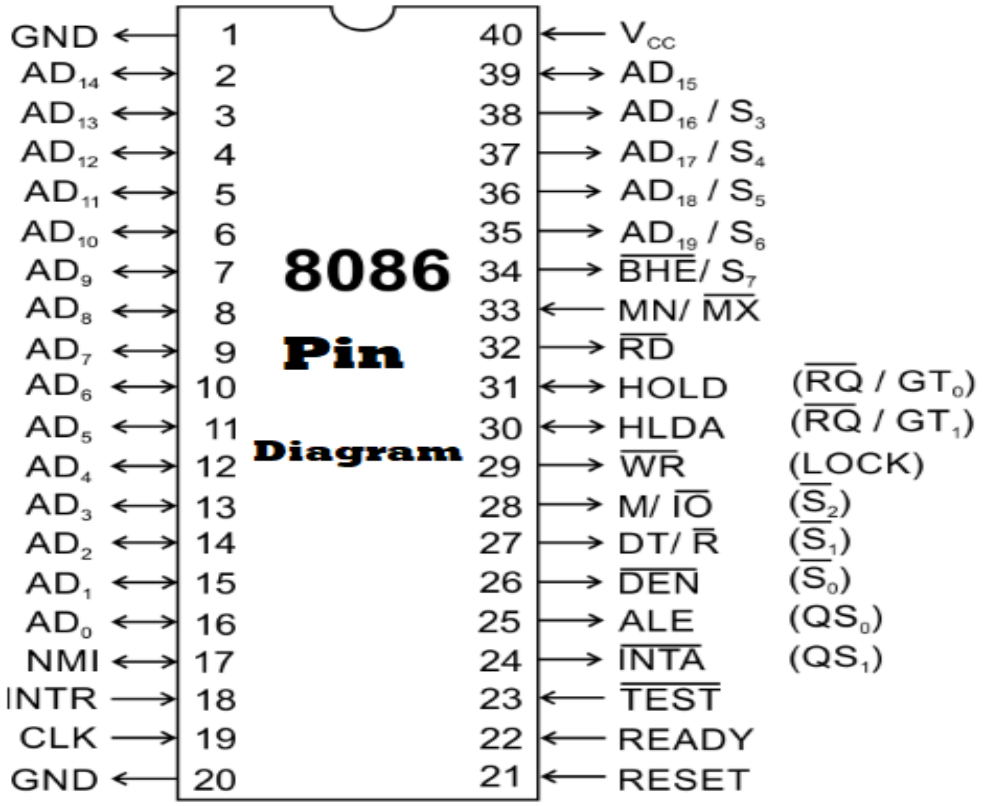
1060 D ------ Decimal constant

Examples of invalid Constant:

1131 B ------ The character 3 should not be used in binary constant.

0E2 ---------- The character H at the end of hexadecimal number in missing.

PIN DIAGRAM:


8086
Pin
Diagram
GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 VCC
39 AD15
38 AD16 / S3
37 AD17 / S4
36 AD18 / S5
35 AD19 / S6
34 BHE/ S7
33 MN/ MX
32 RD
31 HOLD (RQ / GT0)
30 HLDA (RQ / GT1)
29 WR (LOCK)
28 M/ IO (S2)
27 DT/ R (S1)
26 DEN (S0)
25 ALE (QS0)
24 INTA (QS1)
23 TEST
22 READY
21 RESET

8-BITADDITION:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**CX,0000H**	; it loads 0000H to CX
			MOV	**SI,3000H**	;it loads offset address3000 to SI
			MOV	**AL,[SI]**	;it moves the contents of [SI] to AL` register
			INC	**SI**	; it increments SI register by 1
			MOV	**BL,[SI]**	;it moves the contents of [SI] to BL register
			ADD	**AL,BL**	;add AL,BL
			JNC	**L1**	;Jump if no carry
			INC	**CX**	;increment CX
		L1	**INC**	**SI**	; it increments SI register by 1
			MOV	**[SI],AL**	;it moves the contents of AL` register to[SI]
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],CL**	;it moves the contents of CL register to [SI]
			INT	**03**	;it ends the program

OBERVATION TABLE 8-BITADDITION

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

EXP.NO: 01 **DATE:**

ADDITION AND SUBTRACTION OF TWO 8 BIT NUMBERS

AIM: To Write an ALP for the addition and subtraction of two 8 bit numbers using TASM software.

APPARATUS REQUIRED:

i) PC
ii) TASM software

ALGORITHM:

FLOWCHART:

PROGRAM ADDITION:

```
.MODEL SMALL
.STACK 100
.DATA
        OPR1  DB  0F8H
        OPR2  DB  67H
        RES DB  2  DUP(?),'$'
.CODE
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  AL,OPR1
        MOV  BL,OPR2
        ADD  AL,BL
        MOV  RES,AL
        MOV AL, 00H
        RCL   AL,01
        MOV  [RES+1],AL
        MOV  AH,09H
        MOV  DX,OFFSET RES
        INT  21H
        MOV  AH,4CH
        INT  21H
END
```

RESULT:.

INPUT: **1** OPR1 = 0F8H
OPR2 = 67H

OUTPUT: RES = 015FH

INPUT: **2** OPR1 =
OPR2 =

OUTPUT: RES =

8-BIT SUBTRACTION:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**CX,0000H**	; it loads 0000H to CX
			MOV	**SI,3000H**	;it loads offset address3000 to SI
			MOV	**AL,[SI]**	;it moves the contents of [SI] to AL` register
			INC	**SI**	; it increments SI register by 1
			MOV	**BL,[SI]**	;it moves the contents of [SI] to BL register
			SUB	**AL,BL**	;Sub AL,BL
			JNC	**L1**	;Jump if no carry
			INC	**CX**	;increment CX
		L1	**INC**	**SI**	; it increments SI register by 1
			MOV	**[SI],AL**	;it moves the contents of AL` register to[SI]
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],CL**	;it moves the contents of CL register to [SI]
			INT	**03**	;it ends the program

OBERVATION TABLE 8-BIT SUBTRACTION

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

PROGRAM OF SUBTRACTION:

```
.MODEL SMALL
.STACK 100
.DATA
        OPR1  DB  0F8H
        OPR2  DB  67H
        RES  DB  2  DUP(?),'$'
.CODE
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  AL,OPR1
        MOV  BL,OPR2
        SUB   AL,BL
        MOV  RES,AL
        MOV AL, 00H
        RCL   AL,01
        MOV  [RES+1],AL
        MOV  AH,09H
        MOV  DX,OFFSET RES
        INT  21H
        MOV  AH,4CH
        INT  21H
END
```

ALGORITHM:

FLOWCHART:

RESULT:.

INPUT: 1 OPR1 = 0F8H
OPR2 = 67H

OUTPUT: 2 RES = 91H

INPUT: 2 OPR1 =
OPR2 =

OUTPUT: RES =

CONCLUSION:

VIVA QUESTIONS:

1.What is the function of LXI H, 8000 H instruction?

2. How you can store a data in a memory location?

3.How you can read a data from a memory location?

4. What are flags available in 8085?

5. What is the function of RESET key of a 8085 microprocessor kit

16-BITADDITION:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**CX,0000H**	; it loads 0000H to CX
			MOV	**SI,3000H**	;it loads offset address3000 to SI
			MOV	**AX,[SI]**	;it moves the contents of [SI] to AX` register
			ADD	**SI,02**	; it increments SI register by 2
			MOV	**BX,[SI]**	;it moves the contents of [SI] to BX register
			ADD	**AX,BX**	;add AX,BX
			JNC	**L1**	;Jump if no carry
			INC	**CX**	;increment CX
		L1	**ADD**	**SI,02**	; it increments SI register by 2
			MOV	**[SI],AX**	;it moves the contents of AX register to[SI]
			ADD	**SI,02**	; it increments SI register by 2
			MOV	**[SI],CX**	;it moves the contents of CX register to [SI]
			INT	**03**	;it ends the program

OBERVATION TABLE 16-BITADDITION

MEMORY LOCATION	DATA	MEMORY LOCATION	DATA

EXP. NO: 02 **DATE:**

ADDITION OF TWO 16 BIT NUMBERS

AIM: To write an ALP for the addition of two 16 bit numbers using TASM software.

APPARATUS REQUIRED:

i) PC
ii) TASM software

ALGORITHM:

FLOWCHART:

PROGRAM:

```
.MODEL SMALL
.STACK 100
.DATA
        OPR1  DW  8888H
        OPR2  DW  6666H
        RES  DW  3  DUP(?),'$'
.CODE
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  AX,OPR1
        MOV  BX,OPR2
        ADD  AX,BX
        MOV  RES,AX
        MOV AL, 00H
        RCL   AL,01H
        MOV  [RES+2],AX
        MOV  AH,09H
        MOV  DX,OFFSET RES
        INT  21H
        MOV  AH,4CH
        INT  21H
END
```

RESULT:

INPUT: 1 OPR1 = 8888 H
OPR2 = 6666 H

OUTPUT: RES = 00EEEE H

INPUT: 2 OPR1 =
OPR2 =

OUTPUT: RES =

CONCLUSION:

VIVA QUESTIONS:

1)How many bit 8086 microprocessor is?

2)What is the size of data bus of 8086?

3)What is the size of address bus of 8086?

4) What is the max memory addressing capacity of 8086?

5) Which are the basic parts of 8086?

8 BIT MULTIPLICATION:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**AX,0000H**	; it loads 0000H to AX
			MOV	**SI,3000H**	it loads offset address3000 to SI
			MOV	**AL,[SI]**	;it moves the contents of [SI] to AL` register
			INC	**SI**	; it increments SI register by 1
			MOV	**BL,[SI]**	;it moves the contents of [SI] to BL register
			MUL	**BL**	;multiply BL
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],AL**	;it moves the contents of AL register to[SI]
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],AH**	;it moves the contents of AH register to[SI]
			INT	**03**	it ends the program

OBERVATION TABLE:

8 BIT MULTIPLICATION:

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

EXP. NO: 03 **DATE:**

MULTIPLICATION OF TWO 8 BIT NUMBERS

AIM: To write an ALP for the multiplication of two 8 bit numbers using TASM software.

APPARATUS REQUIRED:

i) PC
ii) TASM software

ALGORITHM:

FLOWCHART:

PROGRAM:

```
.MODEL SMALL
.STACK 100
.DATA
        OPR1  DB  05H
        OPR2  DB  03H
        RES  DB  2  DUP(?),'$'
.CODE
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  AL,OPR1
        MOV  BL,OPR2
        MUL  BL
        MOV  RES,AL
       MOV [RES+1],AH
        MOV  AH,09H
        MOV  DX,OFFSET RES
        INT  21H
        MOV  AH,4CH
        INT  21H
END
```

RESULT:-

INPUT: 1 OPR1 = 05H
OPR2 = 03H

OUTPUT: RES = 000FH

INPUT: 2 OPR1 =
OPR2 =

OUTPUT: RES =

CONCLUSION:

VIVA QUESTIONS:

1.Define bit, byte and word

2.What is the function of BIU ?

3.What is the function of EU?

4.What is the maximum size of segment in 8086 microprocessor?

5.What is general purpose registers in 8086?

8 BIT DIVISION:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**AX,0000H**	; it loads 0000H to AX
			MOV	**SI,3000H**	it loads offset address3000 to SI
			MOV	**AL,[SI]**	;it moves the contents of [SI] to AL` register
			INC	**SI**	; it increments SI register by 1
			MOV	**BL,[SI]**	;it moves the contents of [SI] to BL register
			DIV	**BL**	;div BL
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],AL**	;it moves the contents of AL register to[SI]
			INC	**SI**	; it increments SI register by 1
			MOV	**[SI],AH**	;it moves the contents of AH register to[SI]
			INT	**03**	it ends the program

OBERVATION TABLE:

8 BIT DIVISION:

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

EXP. NO : 04 **DATE:**

DIVISION OF TWO 8 BIT NUMBERS

AIM: To write an ALP for the division of two 8 bit numbers using TASM software.

APPARATUS REQUIRED:

i) PC
ii) TASM software

ALGORITHM:

FLOWCHART:

PROGRAM:

```
.MODEL SMALL
.STACK 100
.DATA
        OPR1  DB  31H
        OPR2  DB  02H
        RES  DB  2  DUP(?),'$'
.CODE
        MOV  AX,@DATA
        MOV  DS,AX
       XOR AX,AX
        MOV  AL,OPR1
        MOV  BL,OPR2
        DIV  BL
        MOV  RES,AL
        MOV  [RES+1],AH
        MOV  AH,09H
        MOV  DX,OFFSET RES
        INT  21H
        MOV  AH,4CH
        INT  21H
END
```

RESULT:

INPUT: 1 OPR1 = 31 H
OPR2 = 02 H

OUTPUT: QUE = 18H
REM = 01 H

INPUT: 2 OPR1 =
OPR2 =

OUTPUT: QUE =
REM =

CONCLUSION:

VIVA QUESTIONS:

1. Which is the default pointer for CS/ES?

2. What do you mean by directives?

3. What does int 21h signify?

4. Which are the registers present in 8086?

5. What is the size of flag register?

ASCENDING ORDER:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
		L3	**MOV**	**DX,0000H**	;Clear DX register
			MOV	**AX,0000H**	;clear AX register
			MOV	**CX,0000H**	;clear CX register
			MOV	**SI,5000**	;Assign address to the SI register
			MOV	**CX,[SI]**	;move the content of SI to CX register
			ADD	**SI,02**	;Increment SI register by 02
		L2	**MOV**	**AX,[SI]**	;move the content of SI to AX
			ADD	**SI,02**	;Increment SI by 02 register
			CMP	**AX,[SI]**	;compare the content of AX&SI registers
			JC	**L1**	;Jump the specific location if there is carry
			XCHG	**AX,[SI]**	;Exchange the content of AX,SI register
			SUB	**[SI],02**	;Decrement SI register by 02
			XCHG	**AX,[SI]**	;Exchange the content of AX,SI register
			ADD	**SI,02**	;Increment SI register by 02
			MOV	**DX,0001**	;Initialize the DX register
				CX	
		L1	DEC		;Decrement CX register
			JNZ	**L2**	;Jump to specific location if there is no zero
			Dec	**DX**	;Decrement DX register
			JZ	**L3**	;Jump to specific location if there is zero
			INT	**03**	;terminate the program

EXP. NO : 05 **DATE:**

PROGRAM FOR SORTING AN ARRAY FOR 8086
A.ASCENDING ORDER

AIM: To write an assembly language program to sorting of numbers in an ascending in a given series by using MASM software.

APPARATUS REQUIRED:

i). 8086microprocessor kit -1/ MASM software
ii).FPS(+5V) -1

ALGORITHM:

FLOWCHART:

PROGRAM:

```
.MODEL SMALL
.STACK 100
.DATA

LIST   DB   56H,12H,72H,32H,13H
       COUNT   EQU   ($-LIST)
.CODE

        MOV   AX,@DATA
        MOV   DS,AX
        MOV   CX,COUNT
        MOV   DX,CX

AGAIN: MOV   SI,OFFSET LIST
       MOV   CX,DX

BACK : MOV   AL,[SI]
       INC   SI
       CMP   AL,[SI]
       JC   NEXT
       XCHG   [SI],AL
       DEC   SI
       MOV   [SI],AL
       INC   SI

NEXT  :LOOP   BACK
       DEC   DX
       JNZ   AGAIN
       MOV   AH,09H
       MOV   DX,OFFSET LIST
       INT 21H

       MOV AH,4CH
       INT 21H
       END
```

B.DESCENDING ORDER

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
		L3	**MOV**	**DX,0000H**	;Clear DX register
			MOV	**AX,0000H**	;clear AX register
			MOV	**CX,0000H**	;clear CX register
			MOV	**SI,5000**	;Assign address to the SI register
			MOV	**CX,[SI]**	;move the content of SI to CX register
			ADD	**SI,02**	;Increment SI register by 02
		L2	**MOV**	**AX,[SI]**	;move the content of SI to AX
			ADD	**SI,02**	;Increment SI by 02 register
			CMP	**AX,[SI]**	;compare the content of AX&SI registers
			JNC	**L1**	;Jump the specific location if there is no carry
			XCHG	**AX,[SI]**	;Exchange the content of AX,SI register
			SUB	**[SI],02**	;Decrement SI register by 02
			XCHG	**AX,[SI]**	;Exchange the content of AX,SI register
			ADD	**SI,02**	;Increment SI register by 02
				DX,0001	
			MOV	**CX**	;Initialize the DX register
		L1	DEC		;Decrement CX register
			JNZ	**L2**	;Jump to specific location if there is no zero
			Dec	**DX**	;Decrement DX register
			JZ	**L3**	;Jump to specific location if there is zero
			INT	**03**	;terminate the program

B.DESCENDING ORDER

AIM: To write an assembly language program to sorting of numbers in an descending in a given series by using MASM software.

APPARATUS REQUIRED:

i). 8086microprocessor kit -1/ MASM software
ii).FPS(+5V) -1

ALGORITHM:

FLOWCHART:

PROGRAM:

```
.MODEL SMALL
.STACK 100
.DATA

LIST   DB   56H,12H,72H,32H,13H
       COUNT   EQU   ($-LIST)
.CODE

        MOV   AX,@DATA
       MOV   DS,AX
       MOV   CX,COUNT
       MOV   DX,CX

AGAIN:   MOV   SI,OFFSET LIST
       MOV   CX,DX

BACK :    MOV   AL,[SI]
       INC   SI
       CMP   AL,[SI]
       JNC   NEXT
       XCHG   [SI],AL
       DEC   SI
       MOV   [SI],AL
       INC   SI

NEXT  :LOOP   BACK
       DEC   DX
       JNZ   AGAIN
       MOV   AH,09H
       MOV   DX,OFFSET LIST
       INT 21H

       MOV AH,4CH
       INT 21H
       END
```

OBERVATION TABLE:

ASCENDING ORDER

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

OBERVATION TABLE:

DECENDING ORDER

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

RESULT:

CONCLUSION:

.

VIVA QUESTIONS:

1. Give the concept of Jump with return and jump with non return.

2. What are the flags that are effected by compare statement?

3. What is the Significance of inserting label in programming.

4. What is the Significance of int 03h.

5. What is the purpose of offset?

SMALLEST NUMBER:

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**CX,0000H**	;Clear CX register
			MOV	**AX,0000H**	;clear AX register
			MOV	**BX,0000H**	;clear BX register
			MOV	**SI,5000**	;Assign address to the SI register
			MOV	**CX,[SI]**	;move the content of SI to CX register
			ADD	**SI,02**	;Increment SI register by 02
			MOV	**AX,[SI]**	;move the content of SI to AX
			DEC	**CX**	;Decrement CX register
		L2	**ADD**	**SI,02**	;Increment SI by 02 register
			MOV	**BX,[SI]**	;Move content of SI to BX
			CMP	**AX,BX**	;compare the content of AX&BX registers
			JB	**L1**	;Jump if barrow
			MOV	**AX,BX**	;Move content of BX to AX
		L1	**DEC**	**CX**	;Decrement CX register
			JNZ	**L2**	;jump if no zero
			ADD	**SI,02**	;Increment SI register by 02
			MOV	**[SI],AX**	; move AX content to SI register
			INT	**03**	;Terminate the program

OBERVATION TABLE:

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

EXP. NO : 06 **DATE:**

SMALLEST/LARGEST NUMBER IN AN ARRAY

AIM: To write an assembly language program to Smallest/Largest number in a given series by using MASM software.

APPARATUS REQUIRED:

i). 8086microprocessor kit -1/ MASM software
ii).FPS(+5V) -1

ALGORITHM:

FLOWCHART:

B.LARGEST NUMBER

Address Field	Opcode Field	Label Field	Mnemonic Field	Operand Field	Comment Field
			MOV	**CX,0000H**	;Clear CX register
			MOV	**AX,0000H**	;clear AX register
			MOV	**BX,0000H**	;clear BX register
			MOV	**SI,5000**	;Assign address to the SI register
			MOV	**CX,[SI]**	;move the content of SI to CX register
			ADD	**SI,02**	;Increment SI register by 02
			MOV	**AX,[SI]**	;move the content of SI to AX
			DEC	**CX**	;Decrement CX register
		L2	**ADD**	**SI,02**	;Increment SI by 02 register
			MOV	**BX,[SI]**	;Move content of SI to BX
			CMP	**AX,BX**	;compare the content of AX&BX registers
			JNC	**L1**	;Jump no carry
			MOV	**AX,BX**	;Move content of BX to AX
		L1	**DEC**	**CX**	;Decrement CX register
			JNZ	**L2**	;jump if no zero
			ADD	**SI,02**	;Increment SI register by 02
			MOV	**[SI],AX**	; move AX content to SI register
			INT	**03**	;Terminate the program

OBERVATION TABLE:

INPUT		OUTPUT	
MEMORY LOCATION	**DATA**	**MEMORY LOCATION**	**DATA**

PROGRAM FOR LARGEST/SMALLEST NUMBER

PROGRAM:

```
      . MODEL SMALL
      . STACK
      . DATA
LIST DB 05H,06H,03H,02H,09H,"$'
.CODE
      MOV AX,@DATA
      MOV DS,AX
      MOV CL,04H
      MOV AL,00H
      MOV SI,OFFSET LIST
      MOV AL,[SI]
  L1: CMP AL,[SI+1]
      JNC L                        ; Instead of JNC (JC for smallest)
      XCHG AL,[SI+1]
  L: INC SI
      DEC CL
      JNZ L1
      MOV SI,3000H
      MOV [SI],AL
      INT 03H
CODE ENDS
end start
```

RESULT:

INPUT: 1 LIST = 05h,06h,03h,02h,09h

OUTPUT: 1 AL=09h

INPUT: 2 LIST = 05h,06h,03h,02h,09h

OUTPUT: 2 AL=02h

RESULT:

CONCLUSION:

.

VIVA QUESTIONS:

1. What are the loop instructions supported by 8086?

2. What are the conditional instructions?

3. What is the use of destination index(DI)?

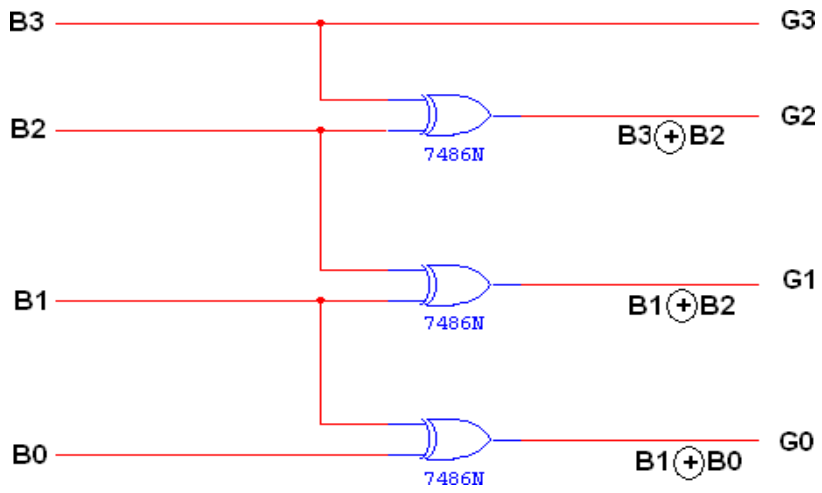
4. Write the addressing modes of 8086

5. Write some data transfer instructions in 8086

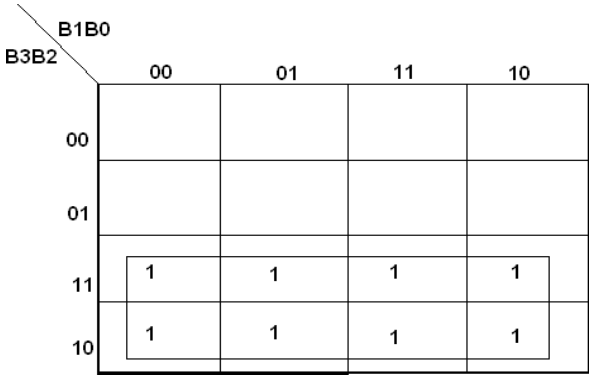
ADVANCED EXPERIMENTS

LOGIC DIAGRAM:

BINARY TO GRAY CODE CONVERTOR

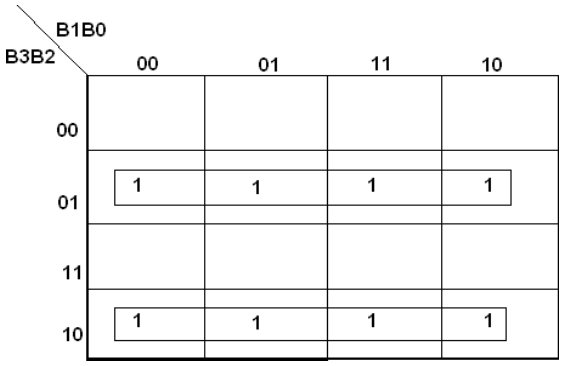


K-Map for G3:



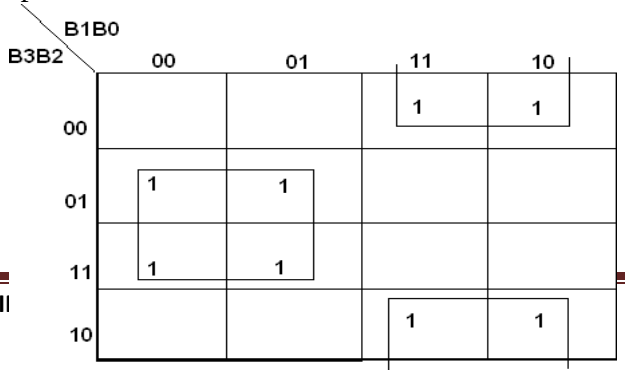
$G_3 = B_3$

K-Map for G2:

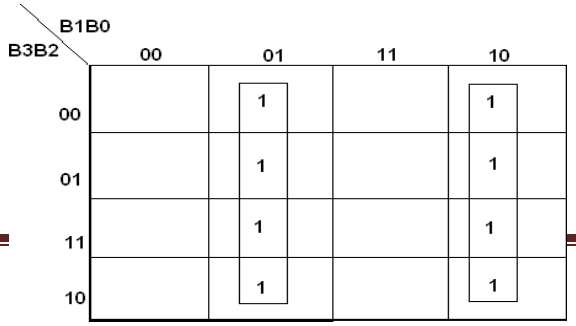


G2 = B3⊕B2

K-Map for G_1:



K-Map for G0:



G0 = B1⊕B0

EXP. NO : **DATE:**

4-BIT BINARY TO GRAY CODE CONVERTER

To design and implement 4-bit Binary to gray code converter

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
	$G1 = B1 \oplus B2$		
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	35

THEORY:

The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that makes the two systems compatible even though each uses different binary code.

The bit combination assigned to binary code to gray code. Since each code uses four bits to represent a decimal digit. There are four inputs and four outputs. Gray code is a non-weighted code.

The input variable are designated as B3, B2, B1, B0 and the output variables are designated as C3, C2, C1, Co. from the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. To convert from binary code to Excess-3 code, the input lines must supply the bit combination of elements as specified by code and the output lines generate the corresponding bit combination of code. Each one of the four maps represents one of the four outputs of the circuit as a function of the four

TRUTH TABLE:

Binary input				Gray code output			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**1**	**0**	**0**	**0**	**1**	**1**
0	**0**	**1**	**1**	**0**	**0**	**1**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**0**
0	**1**	**0**	**1**	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**0**	**1**	**0**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**1**	**1**	**0**	**0**
1	**0**	**0**	**1**	**1**	**1**	**0**	**1**
1	**0**	**1**	**0**	**1**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**1**	**1**	**0**
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**
1	**1**	**0**	**1**	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**

derived by the maps. These are various other possibilities for a logic diagram thatimplements this circuit. Now the OR gate whose output is C+D has been used to implement partially each of three outputs.

PROCEDURE:

(i) Connections were given as per circuit diagram.

(ii) Logical inputs were given as per truth table

(iii) Observe the logical output and verify with the truth tables.

RESULTS:

CONCLUSION:

VIVAQUESTIONS:

1.How do you convert 4 bit binary to gray code?

2.How many valid inputs will a 4 bit binary to Gray code converter have?

3.What is Gray code explain with example?

4.What is gray to binary code conversion?

5.What is gray code?

EXP NO: **Date:**

LCM FOR THE GIVEN DATA

AIM: To write an Assembly Language Program to find LCM for the given data using 8086.

APPARATUS REQUIRED:

(i) 8086 Microprocessor Kit
(ii) TASM Software/Win86E
(iii) FPS (+5V)
(iv) PC
(v) USB Cable

ALGORITHM:

PROGRAM:

ASM code:

```
        . Model small
        . Stack
        . Data
        Num1  DW 0005h
        Num2  DW 0002h
        Ans  DW  ?
        . Code
        Mov  AX, @data
        Mov  DS,AX
        Mov  AX,Num1
       Mov  BX,Num2
        Mov  dX,0000h
Next:  Push  AX
        Push DX
        Div  BX
        Cmp  DX,0000h
        JE  LAST
        POP  DX
        POP AX
        Add AX,Num1
        JNC Next
LAST:  Pop Ans+2
       Pop Ans
       Mov AH,4ch
        Int  21h
        End
```

INPUT:

OUTPUT:

RESULT:

CONCLUSION:

VIVA QUESTIONS:

1. What is stack?

2. Which interrupt has highest priority?

3. What is an Interrupt?

4. What is a compiler?

5. What is meant by Maskable Interrupts?