



### Practical 01

Title:- Know the digital Lab IC Tester, Multimeter, Bread Board, Trainer Kit.

Aim:- Describe the basic component of digital Lab.

#### Theory:-

##### 1. Digital Lab IC Tester:-

An IC tester is a specialized device used to:-

- Identify unknown ICs
- Verify IC functionality
- Check pin configurations
- Detect shorts or opens in IC pins.

IC testers usually have a library of known IC types and can automatically identify the IC by applying a series of test signals. They may also have additional features like:

- Pinout display
- logic function testing
- Supply voltage testing

##### 2. Multimeter:-

A multimeter is a versatile measuring instrument used to measure electrical parameters including:-

- Voltage (DC/AC)

- Current ( DC / AC )

- Resistance

- Continuity

- Diode testing ( in some models )

- Capacitance ( in some models )

Multimeters can be analog or digital, with digital multimeters being more common. They usually have:

- Multiple measurement ranges.

- Auto-ranging or manual ranging options.

- Auto-hold & storage capacities

- Data hold & temperature measurement features like temperature measurement

- Additional features like insulation testing ( in some models )

frequency, or insulation frequency, or insulation testing ( in some models )

### 3. Breadboard:-

A breadboard is a rectangular board with rows of holes & columns used to:-

- Prototype digital circuits

- Build temporary circuits

- Test and experiment with circuit designs.

Breadboard typically have:

- Rows of connected holes ( strips ) for power and ground.

- Columns of connected holes ( strips ) for signal connections

- Separate areas for ICs, resistors, capacitors, & other components.

- Jumper wires for connecting components.

#### 4. Trainer Kit:-

A Trainer Kit is a self-contained unit that includes:

- A breadboard or experimental circuit
- Power supply (fixed or adjustable)
- Components like:
  - LEDs
  - Resistors
  - Capacitors
  - IC sockets
  - Function generators
  - Oscilloscopes
  - Logic analyzers

- Sometimes additional features like:

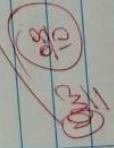
- Jumper wires
- Sometimes additional features like:
  - Function generators
  - Oscilloscopes
  - Logic analyzers

~~Trainer kits are designed for hand-on experimentation, learning, and teaching digital electronics and circuit design. They often come with instructional materials & exercise~~

~~These components form the core of a digital lab, allowing users to design, test, & experiment with digital circuits of electronics.~~

- In Result:- After studying the basic component of digital lab:
  - Understand the functions & uses of each component.
  - Design, test, & experiment with digital circuits & electronics.
  - Develop hands-on skills in prototyping & trouble shooting knowledge in digital electronics.
  - Gain practical knowledge in digital electronics and circuit design.

Conclusion: Hence, we studied the functions & uses of digital lab components.



### Practical No. 02

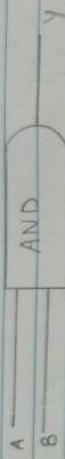
- Aim:- Study of basic gate IC.
- Objectives:-
  - 1. Draw Symbol
  - 2. Draw pin diagram
  - 3. Draw truth table
  - 4. Write equation.

:- Required IC's →	1. 7408	AND
	2. 7432	OR
	3. 7404	NOT
	4. 7486	EX-OR
	5. 74266	EX-NOR
	6. 7400	NAND
	7. 7402	NOR

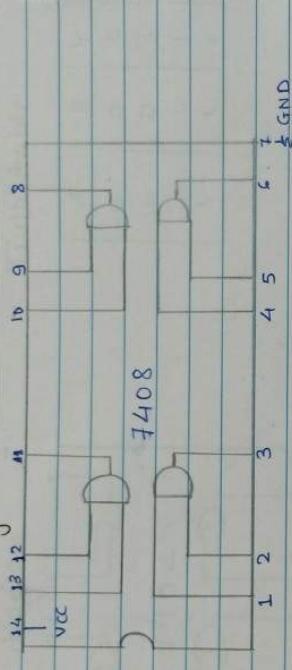
:- Theory :- There are total 3 basic gates (AND, OR, NOT), 2 derived gates (EX-OR, EX-NOR) & 2 universal gates (NAND, NOR).

- 1. AND gate :- IC 7408.
  - It is logic gate chip with two inputs & one output.
  - If all outputs are high, then only results are high.

1. Symbol :-



2. Pin diagram :-



3. Truth Table:-

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

4. Equation:-

$$Y = A \cdot B$$
$$= A \times B$$

2. OR gate :- IC 7432.

- It is logic gate chip with two inputs & one output.
- If any one input is high then result is high.

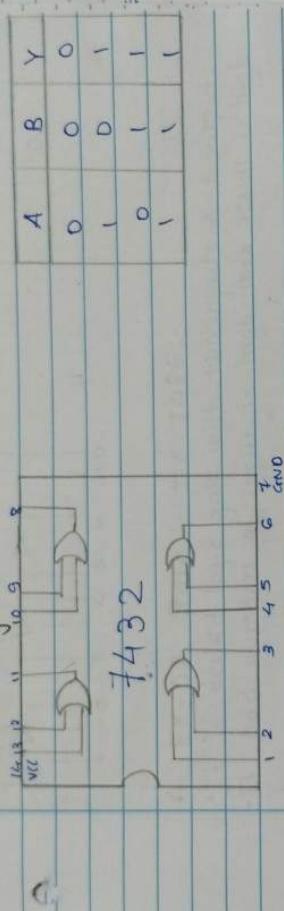
1. Symbol:-



2. Equation:-

$$\begin{aligned}Y &= A \text{ OR } B \text{ OR } \dots \text{ OR } N \\&\downarrow \\Y &= A + B + \dots + N \\&\therefore Y = A + B.\end{aligned}$$

3. Pin diagram



4. Truth table.

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

3. NOT gate:- IC 7404

• It is logic gate chip with single input & single output.

• Result is always compliment of input.

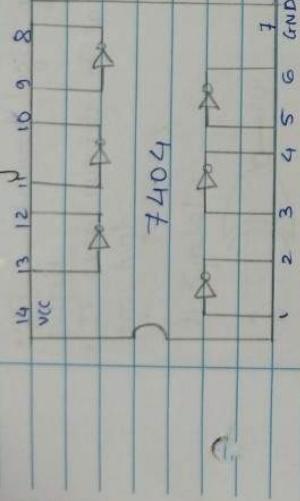
1. Symbol:-



2. Equation:-

$$Y = \bar{A}$$

### 3. Pin diagram



### 4. Truth Table

A	Y
0	1
1	0

### 4. Ex-OR Gate :- IC 7486 :-

- It is denoted logic gate with two inputs A & B output.
- If odd numbers of input is high then result is high.

### 1. Symbol :-

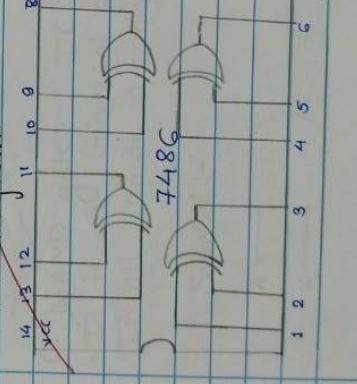


### 2. Equation :-

$$Y = A \oplus B$$

$$= AB' + A'B$$

### 3. Pin diagram :-



### 4. Truth Table

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

5. Ex-NOR Gate:- IC 74266:
- It is derived logic gate having two inputs & one output.
  - If even number of inputs are high the only results are high.

1. Symbol:-

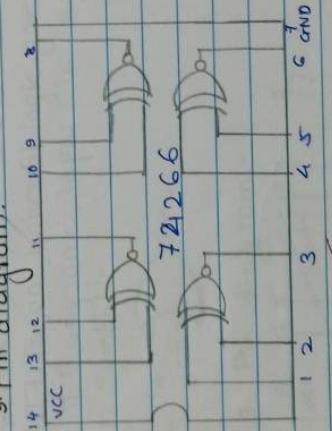


2. Equation:-

$$\gamma = A \oplus B$$

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

3. Pin diagram:-

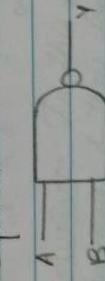


4. Truth table:-

	A	B	$\gamma$
0	0	0	1
1	0	0	0
0	1	0	0
1	1	1	1

6. NAND gate:- IC : 7400
- It is an universal gate with two inputs & 1 output
  - If all inputs are high, then only result is low.

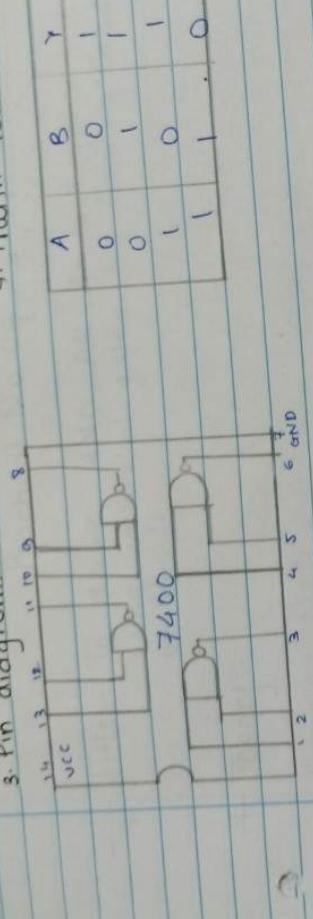
1. Symbol :-



2. Equation:-

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

3. Pin diagram:-



4. Truth table

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

7. NOR Gate :- IC 7402

- It is an universal logic gate with two inputs & 1 output.
- If any one input is high then result is low.

2. Equation:-

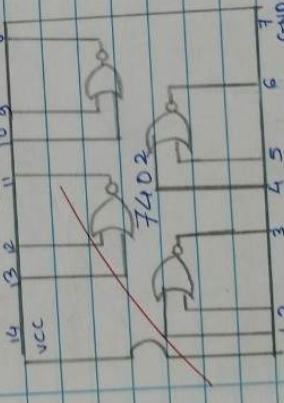
$$Y = \overline{A + B}$$

1. Symbol:-



4. Truth table.

3. Pin diagram:-



7

6

5

4

3

2

1

8

7

6

5

4

3

2

1

14

VCC

15

VCC

13

VCC

10

VCC

9

VCC

8

VCC

7

VCC

6

VCC

5

VCC

4

VCC

3

VCC

2

VCC

1

VCC

Conclusion:- In this practical we learnt about various logic gates & their logical eq's & also find out their truth tables.

### Practical No. 03.

A. Title:- To derive AND, OR, & NOT gates using universal gates by forming circuits on breadboard

B. Theory:-

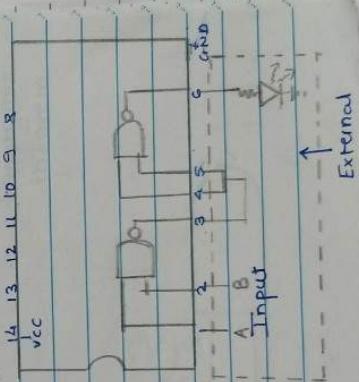
Deriving NOT gate using NAND gate:-

A. Procedure:- A NOT gate (inverter) can be made from NAND gate by connecting all of the inputs together & creating, in effect, a single common input for a two-input gate.

B. Logic diagram:-



C. Circuit diagram:-



D) Truth table:-

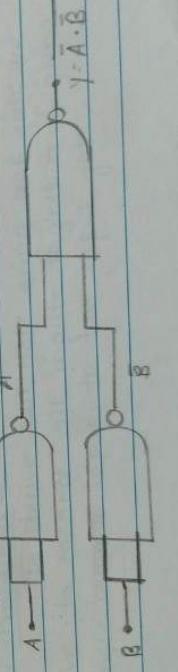
Input		Output	
A	B	$A \cdot B$	$\bar{A} \cdot \bar{B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

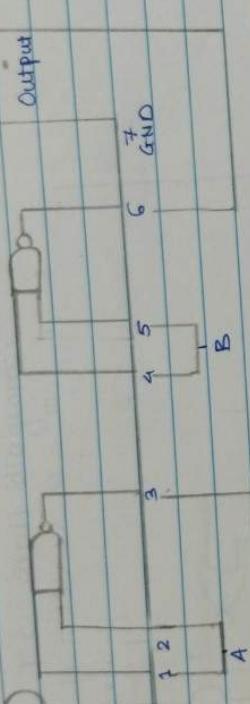
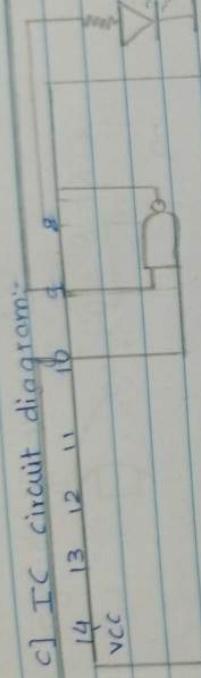
2. Deriving OR gate using NAND gate.

A. Procedure:- OR gate using NAND gate is derived by inverting both inputs of then connecting with one more NAND gate.  
Boolean expression for OR gate:-

$$\therefore Y = \overline{\overline{A+B}} \quad \therefore Y = \overline{\overline{A} \cdot \overline{B}} \quad \text{by deMorgan's theorem}$$

B) Logic diagram:-





3. Deriving NOT gate using NOR gate:-

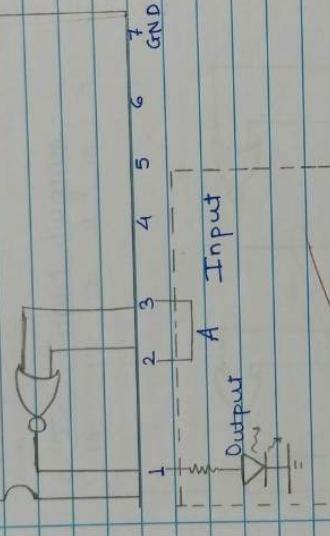
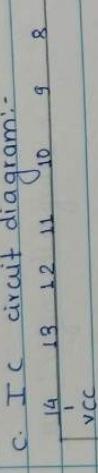
a) Procedure:-

To derive NOT gate using NOR gate we have to connect all inputs to either of giving single common input to two input gate.

B. logic diagram:-



C. Ic circuit diagram:-



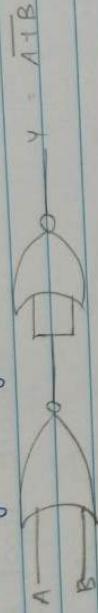
D. Truth table:-

Input		Output	
A	B	$\bar{A}$	$A + B$
0	0	1	0
1	1	0	1

5. Deriving OR gate using NOR gate:

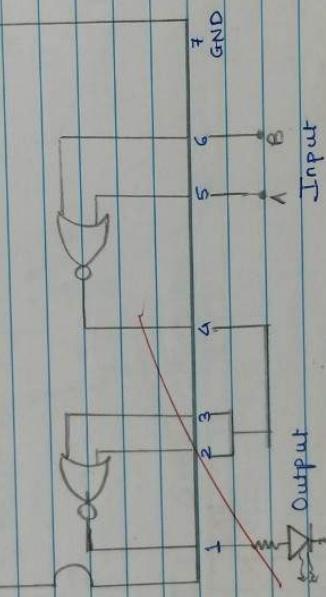
A. Procedure:-  
To derive OR gate using NOR gate, we have to invert output of NOR gate.

B. Logic diagram:-



C. IC Circuit diagram:-

14 13 12 11 10 9 8



D. Truth table:-

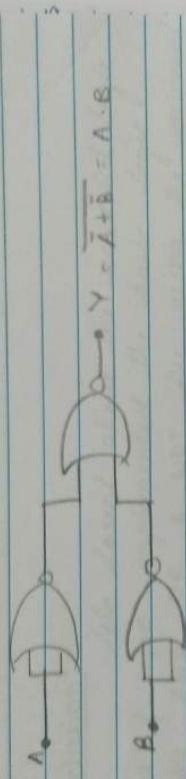
Input	Output	Output	Output	Output
A	B	$A+B$	$\bar{A}+\bar{B}$	$\bar{A}+B$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

C. Drive AND gate using NOR gate

A. Procedure:-

To derive AND gate using NOR gate  
we have to take inverted inputs of A & B of  
then have to give output to one more NOR gate

B. logic diagram:-



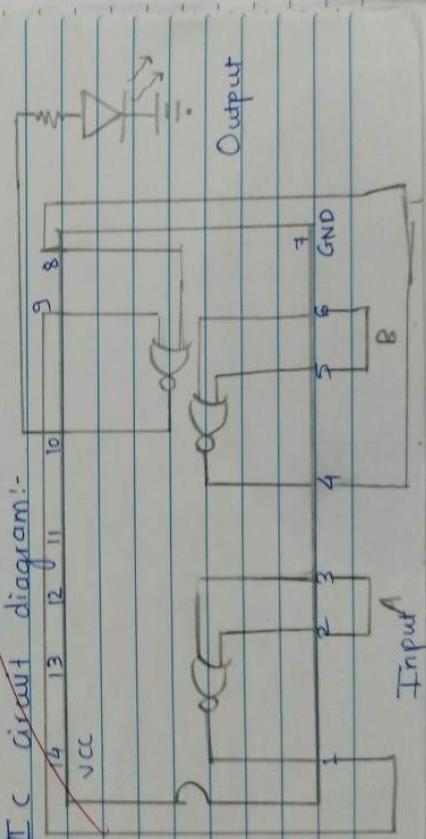
Boolean expression for OR gate:-

$$\therefore Y = A \uparrow B$$

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

... using de morgan's theorem

~~D~~ C Circuit diagram:-



### Q1 Truth table:

Input		Output				
A	B	$A \cdot B$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	0	1	1	1	0
0	1	0	1	0	1	0
1	0	0	0	1	1	0
1	1	1	0	0	0	1

Conclusion: We learnt about the universal logic gates of above implemented the basic logic gates AND, OR & NOT by using the universal gates NAND & NOR only.

Q1  
Ans  
Date

## Practical No. 04

- Title:- Verify demorgan's theorem by forming the circuit on board.

- Theory:-

- Demorgan's Theorem :- Demorgan suggested two theorems that form an important part of Boolean Algebra.

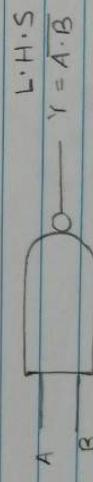
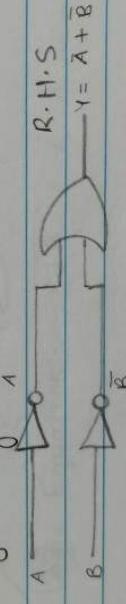
In the equation form they are:-

1.  $\overline{AB} = \overline{A} + \overline{B}$  : The complement of product is equal to sum of complements.
2.  $\overline{A+B} = \overline{A} \cdot \overline{B}$  : The complement of sum is equal to product of complements.

- 1. Demorgan's first theorem:-

- i. Statement form:- The compliment of product is equal to sum of the complement.

- ii. Logic diagram:-



iii. Logic Equation:-  $\overline{A \cdot B} = \bar{A} + \bar{B}$

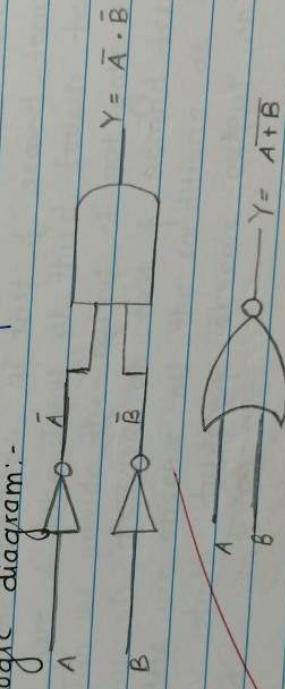
iv. Truth table :-

		L.H.S		R.H.S	
		A	B	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$
0	0	0	0	1	1
0	1	0	1	1	1
1	0	1	0	0	0
1	1	0	0	0	0

2. De Morgan's Second Theorem:-

- i. Statement form:- The complement of sum is equal to the product of complements.

ii. Logic diagram:-



iii. Logic Equation:-  $\overline{A+B} = \bar{A} \cdot \bar{B}$

iii. Logic Equation:-  $A \cdot \bar{B} = \bar{A} + \bar{B}$

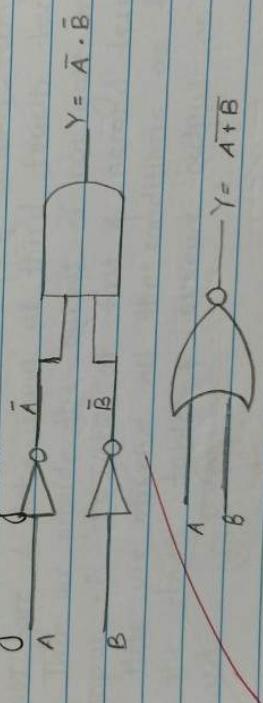
iv. Truth table :-

L.H.S				R.H.S			
A	B	$\bar{A} \cdot \bar{B}$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$	$\bar{A} \cdot \bar{B}$	
0	0	1	1	1	1	1	
0	1	1	1	0	1	0	
1	0	1	0	1	1	1	
1	1	0	0	0	0	0	

2. De Morgan's Second Theorem:-

- i. Statement form:- The complement of sum is equal to the product of complements.

ii. Logic diagram:-



iii. Logic Equation:-  $\bar{A} + \bar{B} = \bar{A} \cdot \bar{B}$

iv. Truth table:-

L.H.S		R.H.S	
A	B	$\bar{A} + \bar{B}$	$\bar{A}$
0	0	1	1
0	1	0	0
1	0	0	1
1	1	0	0

Thus, with the help of truth table, demorgan's second theorem is proven.

Procedure of working:-

- First we mount IC 7400 on breadboard.
- Now, we give input to A & B as 1st & 2nd terminal of IC.
- Then from first & second terminal then we provide it to the IC 7404 at first & second terminal of NOT gate & take input at third, fourth terminal.
- Then we provide the output of not gate to the input of IC 7486 at first & second terminal of IC.
- Here, we observe all the conditions as follows in the truth table & observe output on breadboard. ~~Now~~ ~~Now~~ ~~Now~~ ~~Now~~ ~~Now~~

- Conclusion:- Here, we learned how to mount IC & observe the output. we proved demorgan's theorem.

### Practical No.05

Title:- Minimization of realization of function using K-maps & its implementation by constructing the circuit on breadboard.

Theory:-

$$Y = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} + A \cdot B \cdot \bar{C} \cdot \bar{D}$$

Answer:-

This equation is in SOP form.

$$\begin{aligned}\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} &= m_1 & \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} &= m_5 \\ \therefore \bar{A} \cdot B \cdot C \cdot D &= m_2 & \therefore \bar{A} \cdot B \cdot C \cdot \bar{D} &= m_6 \\ \therefore A \cdot B \cdot \bar{C} \cdot \bar{D} &= m_{12} & \therefore A \cdot B \cdot \bar{C} \cdot D &= m_{13} \\ \therefore A \cdot B \cdot C \cdot D &= m_{14} & \therefore A \cdot \bar{B} \cdot C \cdot \bar{D} &= m_{15}\end{aligned}$$

The given equation is min-terms is,

$$Y = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$$

\* K-map for given equation is:-

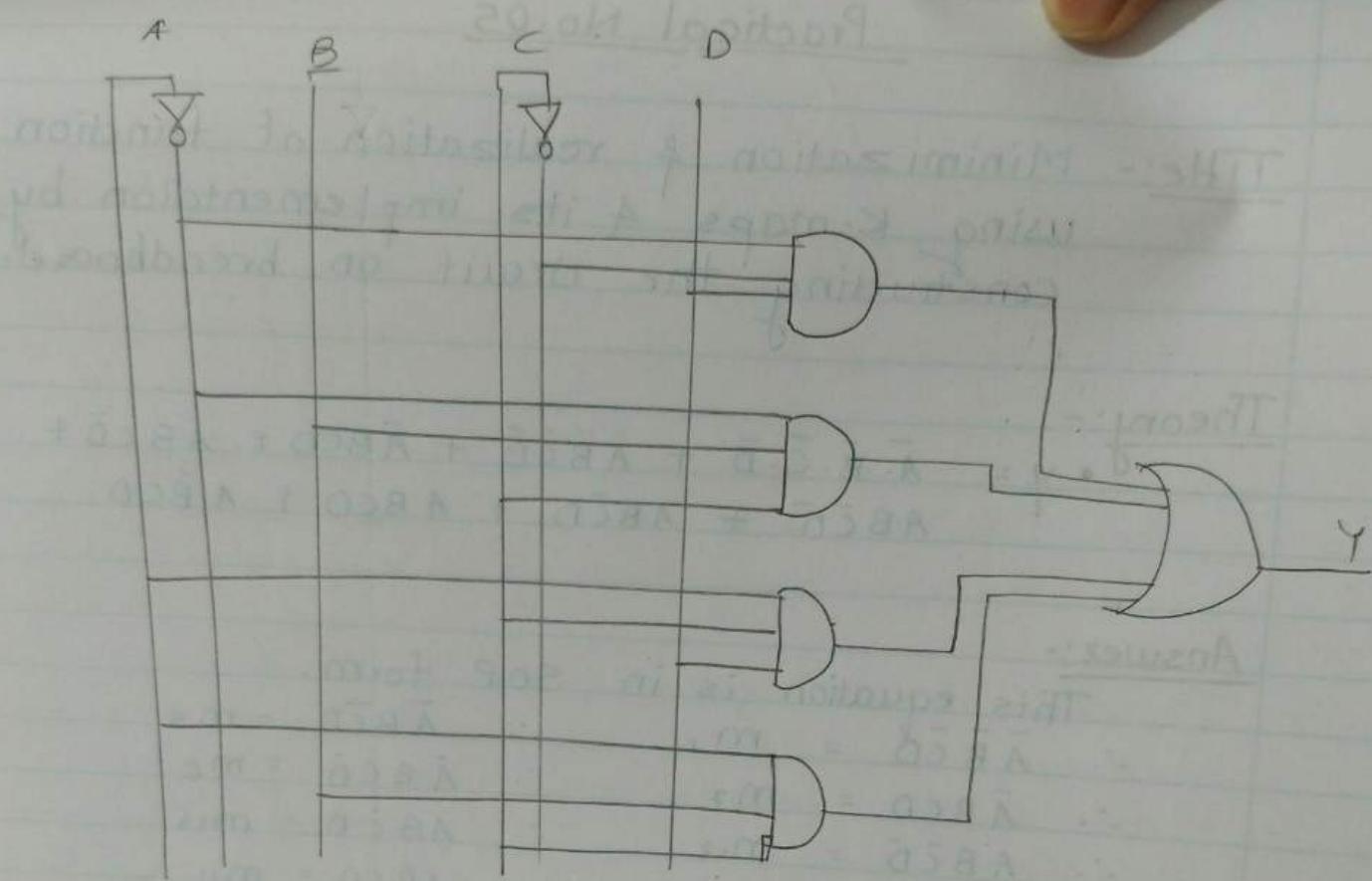
00	00	01	11	10
01	0	1	0	0
11	1	1	1	1
10	0	0	1	0

\* The minimized equation is

$$y = \bar{A}\bar{C}D + \bar{A}B\bar{C} + ACD + ABC$$

- Conclusion: From this practical, we learnt about K-map minimization of implementation of minimized equation.

or  
or



$A_2$	$A_1$	$B_2$	$B_1$	$C_2$	$C_1$	$D_2$	$D_1$	$Y$
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	1	0	0	0	1	0	0	1
0	1	0	1	0	1	0	1	1
1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	1	1	1
1	1	0	0	1	1	0	0	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	0	1	0	0
1	1	1	0	0	0	1	1	1
1	1	1	1	1	0	1	1	1

\* The minimized equation is

$$Y = \bar{A}\bar{C}D + \bar{A}B\bar{C} + ACD + ABC$$

- Conclusion:- from this practical, we learnt about K-map minimization & implementation of minimized equation.

Ques  
Q1/10



### Practical No. 6

Title:- Write an Assembly language Program (ALP) for Addition and subtraction of two 16-bit numbers.

#### Theory:-

- Arithmetic instructions (ADD, SUB) are used to perform arithmetic operations like addition & substraction.
- Arithmetic :- ADD instructions adds the data destination & source operand & stores data in destination.

#### -Algorithm:

- Step 1 : Start
- Step 2 : Initialize the contents of memory location in AX & DS registers.
- Step 3 : Load the first number into register BX from memory.
- Step 4 : Load the second number into register DX from memory.
- Step 5 : Add content of register DX with the content of register BX.
- Step 6 : Load the result from BX register to sum variable.
- Step 7 : Ends.

**Subtraction:** SUB instruction subtracts data in source operand from data in destination operand & then store result back to destination operand.

-Algorithm:

- Step 1 : Start
- Step 2 : Initialize the contents of memory location in AX & DS register.
- Step 3 : Load the first number into BX register contents
- Step 4 : Load the second number in DX registers from num2 variable.
- Step 5 : Subtract the content of DX from BX register's contents
- Step 6 : Load the result value from BX register to SUBT variable.
- Step 7 : End.

~~Conclusion :- We learnt program of addition & subtraction in assembly language.~~

~~Ques  
Ans  
0 1 0 1 0~~

PROGRAM TO ADD TWO 16-BIT NUMBERS

**Code:**

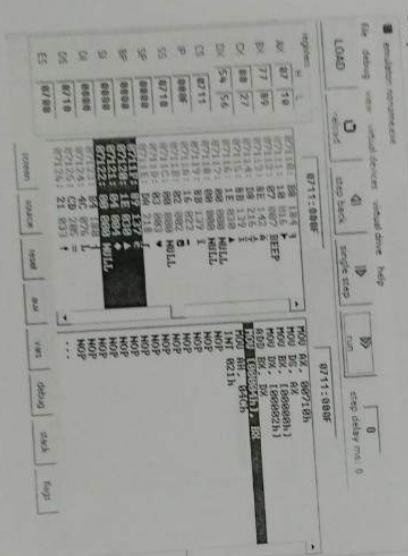
```
.MODEL SMALL
.DATA
N2 DW 2333H
N3 DW 5456H
SUM DW ?
.CODE
MAIN PROC
MOV AX, @DATA
MOV DS, AX ; Initialize DS
MOV BX, N2 ; moving N2 value in BH register
MOV DX, N3 ; moving N3 value in DS register
ADD BX, DX ; adding BH to DH and storing value in BH
MOV SUM, BX ; moving BH value to SUM variable.
```

MOV AH, 4CH

INT 21h

MAIN ENDP

**Output:**



initialized and assigned value to N2.  
initialized and assigned value to N3.  
initialized sum variable.

CODE

CODE:

INITIALISING AND OPERATING  
SYSTEMS FOR VARIOUS  
DIFF DW ?

.CODE  
MAIN PROC

```
MOV AX, @DATA  
MOV DS, AX ; Initialize DS
```

: moving H2 value in BH register

MOV BX, N2 ; moving N3 value in DS register.  
MOV DX, N3 ; SUBTRACTING BH to DH

MOV DIFF, BX

~~MOV AH, 4CH~~

MAIN ENDP

**OUTPUT:**



### Practical No. 07

Title:- Write an Assembly language Program(MLP) to  
Multiplication & division of 8 bit / 16-bit / 32-  
bit signed / unsigned numbers.

- Theory :- Arithmetic instructions (MUL, DIV) are used  
to perform arithmetic operations like multiplication  
& division.
- Multiplication: MUL instruction is used for the  
multiplication of unsigned numbers.

- Algorithm:

- Step 1: Start
- Step 2: Initialize the contents by memory Ax register
- Step 3: Load the first number of into Ax register
- Step 4: Load the second number into Bx  
register.
- Step 5: Multiply Ax with Ax
- Step 6: Move data from Ax register to  
RESULT variables.
- Step 7: More higher word of multiplication to  
higher word of RESULT variables
- Step 8 : End.



- Division : DIV instruction performs division of two operands.

Algorithm :

- Step 1 : start
- Step 2 : Initialize the contents of memory location in AX & DS register.
- Step 3 : locate the divisor into BX register
- Step 4 : Divide AX by BX register.
- Step 5 : Load the data in QUOTIENT & REMINDER.
- Step 6 : End.

Conclusion: ~~Once we had computer practical on assembly language program for multiplication of divisor.~~

OB  
TO

```

.CODE
.MODEL SMALL
.DATA
    N2 DW 7333H ; initialized and assigned value to n2.
    N3 DW 5456H ; initialized and assigned value to n3.
    PROD DD ? ; initialized PROD variable.

.CODE
MAIN PROC
    MOV AX, @DATA ; Initialize DS
    MOV DS, AX

    MOV AX, N2 ; moving N2 value in AX register
    MOV DX, N3 ; moving N3 value in DX register
    MUL DX ; MULTIPLY AX with DX, answer stored in DX:AX

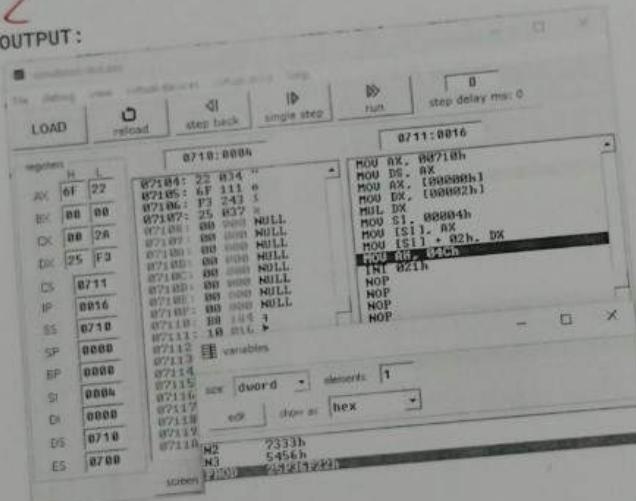
    LEA SI, PROD
    MOV [SI], AX
    MOV [SI + 02h], DX

    MOV AH, 4CH
    INT 21h

MAIN ENDP

```

### **OUTPUT:**



PROGRAM TO DIVIDE TWO 16-BIT NUMBERS

CODE:

```
.MODEL SMALL
.DATA
    N2 DW 1234H ; initialized and assigned value to n2.
    N3 DW 0078H ; initialized and assigned value to n3.
    QUOT DW ?    ; initialized PROD variable.
    REM DW ?

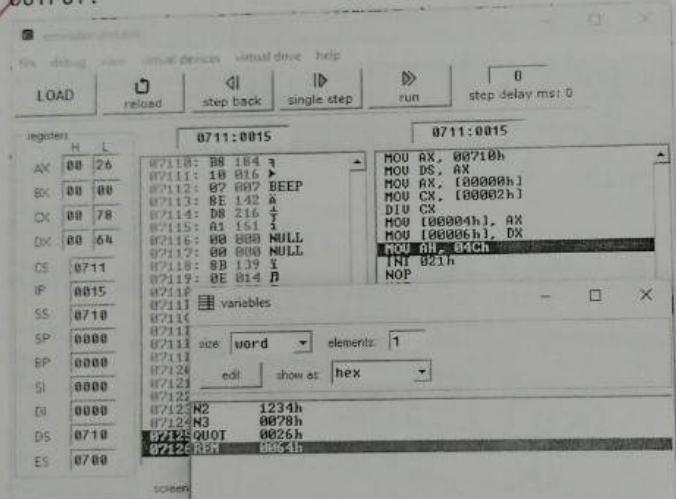
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX ; Initialize DS

    MOV AX, N2 ; moving N2 value in AX register
    MOV CX, N3 ; moving N3 value in CX register
    DIV CX ; divide AX with CX, answer stored in DX:AX

    MOV QUOT, AX
    MOV REM, DX

    MOV AH, 4CH
    INT 21h
MAIN ENDP
```

OUTPUT:





### Practical No.08

Title: Write Assembly language program (Any) to sum of given series of number (8bit / 16bit)

#### Theory:-

##### 1) INC :-

- Syntax: INC Destination
- Destination will be register or memory location if it will be 8bit or 16bit.
- This instruction is increment content of destination by 1
- All the conditional flag are affected except carry flag.
- This instruction adds one from content of operand
- Immediate data can not be operand of this instruction.

Example: INC (BX)

INC AX

##### 2) DEC :-

- Syntax: DEC destination
- Destination will be register or memory location if it will be 8bit or 16bit.
- This instruction is decrement content of dest. by 1.



- This instruction subtracts 1 from operands.
- All this conditions flags are affected except carry flag.

• Example:  
DEC CX

DEC [1615H]

3) INZ:

- Syntax: INZ label.
- This instruction helps in branching the program.
- Jumps if previous conditions is not zero.

Example: INZ UP

INZ continue.

• Algorithm:

- Step 1: Start
- Step 2: Initialize variable numlist & count with datatype DBT define bytes for 8bits.
- Step 3: Initialize data segments.
- Step 4: Move the offset of NUMLIST to CL register.
- Step 5: Move the content of COUNT to a register.
- Step 6: Move 0000H to AX & BX register & result store in AX.
- Step 7: Add content of BX & AX & result.
- Step 8: Increment si by 182 for 8 16 bit.
- Step 9: Decrement cl by 1
- Step 10: Ends.



PROGRAM TO ADD A GIVEN SERIES OF 8-BIT NUMBERS

**CODE:**

.MODEL SMALL

.DATA  
ARRAY DB 31H, 11H, 3CH, 2DH ; Initialize array

LENGTH DB 0AH ; length of the array

SUM DB ? ; define sum

**CODE**

MAIN PROC

MOV AX, @DATA

MOV DS, AX ; Initialize DS

MOV CL, LENGTH ; store length of the array in Counter

MOV AL, 00H

; initialize AX with 0

LEA SI, ARRAY ; store the base address of the array in SI

MOV DL, [SI] ; MOV next element in DX

ADD AL, DL ; ADD dx to the previous sum

INC SI ; increment element pointer

LOOP NEXT ; Loop until all variables are added

MOV SUM, AL ; moving total to SUM variable

MOV AH, 4Ch ; Exit the program and return control to OS

INT 21h

MAIN ENDP

OUTPUT:

DATA	0	4	8	C	X
array	31H	11H	3CH	2DH	
length	0AH				
sum	00H				
0	00H				
1	00H				
2	00H				
3	00H				
4	00H				
5	00H				
6	00H				
7	00H				
8	00H				
9	00H				
10	00H				
11	00H				
12	00H				
13	00H				
14	00H				
15	00H				
16	00H				
17	00H				
18	00H				
19	00H				
20	00H				
21	00H				
22	00H				
23	00H				
24	00H				
25	00H				
26	00H				
27	00H				
28	00H				
29	00H				
30	00H				
31	00H				
32	00H				
33	00H				
34	00H				
35	00H				
36	00H				
37	00H				
38	00H				
39	00H				
40	00H				
41	00H				
42	00H				
43	00H				
44	00H				
45	00H				
46	00H				
47	00H				
48	00H				
49	00H				
50	00H				
51	00H				
52	00H				
53	00H				
54	00H				
55	00H				
56	00H				
57	00H				
58	00H				
59	00H				
60	00H				
61	00H				
62	00H				
63	00H				
64	00H				
65	00H				
66	00H				
67	00H				
68	00H				
69	00H				
70	00H				
71	00H				
72	00H				
73	00H				
74	00H				
75	00H				
76	00H				
77	00H				
78	00H				
79	00H				
80	00H				
81	00H				
82	00H				
83	00H				
84	00H				
85	00H				
86	00H				
87	00H				
88	00H				
89	00H				
90	00H				
91	00H				
92	00H				
93	00H				
94	00H				
95	00H				
96	00H				
97	00H				
98	00H				
99	00H				
100	00H				
101	00H				
102	00H				
103	00H				
104	00H				
105	00H				
106	00H				
107	00H				
108	00H				
109	00H				
110	00H				
111	00H				
112	00H				
113	00H				
114	00H				
115	00H				
116	00H				
117	00H				
118	00H				
119	00H				
120	00H				
121	00H				
122	00H				
123	00H				
124	00H				
125	00H				
126	00H				
127	00H				
128	00H				
129	00H				
130	00H				
131	00H				
132	00H				
133	00H				
134	00H				
135	00H				
136	00H				
137	00H				
138	00H				
139	00H				
140	00H				
141	00H				
142	00H				
143	00H				
144	00H				
145	00H				
146	00H				
147	00H				
148	00H				
149	00H				
150	00H				
151	00H				
152	00H				
153	00H				
154	00H				
155	00H				
156	00H				
157	00H				
158	00H				
159	00H				
160	00H				
161	00H				
162	00H				
163	00H				
164	00H				
165	00H				
166	00H				
167	00H				
168	00H				
169	00H				
170	00H				
171	00H				
172	00H				
173	00H				
174	00H				
175	00H				
176	00H				
177	00H				
178	00H				
179	00H				
180	00H				
181	00H				
182	00H				
183	00H				
184	00H				
185	00H				
186	00H				
187	00H				
188	00H				
189	00H				
190	00H				
191	00H				
192	00H				
193	00H				
194	00H				
195	00H				
196	00H				
197	00H				
198	00H				
199	00H				
200	00H				
201	00H				
202	00H				
203	00H				
204	00H				
205	00H				
206	00H				
207	00H				
208	00H				
209	00H				
210	00H				
211	00H				
212	00H				
213	00H				
214	00H				
215	00H				
216	00H				
217	00H				
218	00H				
219	00H				
220	00H				
221	00H				
222	00H				
223	00H				
224	00H				
225	00H				
226	00H				
227	00H				
228	00H				
229	00H				
230	00H				
231	00H				
232	00H				
233	00H				
234	00H				
235	00H				
236	00H				
237	00H				
238	00H				
239	00H				
240	00H				
241	00H				
242	00H				
243	00H				
244	00H				
245	00H				
246	00H				
247	00H				
248	00H				
249	00H				
250	00H				
251	00H				
252	00H				
253	00H				
254	00H				
255	00H				
256	00H				
257	00H				
258	00H				
259	00H				
260	00H				
261	00H				
262	00H				
263	00H				
264	00H				
265	00H				
266	00H				
267	00H				
268	00H				
269	00H				
270	00H				
271	00H				
272	00H				
273	00H				
274	00H				
275	00H				
276	00H				
277	00H				
278	00H				
279	00H				
280	00H				
281	00H		</		

## Practical No. 09.

Title:- Write an Assembly language program (A CO) to find the smallest & greatest number from the given series.

### Theory:-

- Searching:

A program of finding the required element from a set of data or any array i.e. smallest element, largest element, select particular element even or odd no.

- CMP Instruction:

Used to compare two numeric data fields one or both of which are contained a register or memory.

Algorithm for finding smallest 8-bit no:

~~Step1: Initialize data smallest segment, byte counter memory pointer to read number from array.~~

~~Step2: Read no. from array.~~

~~Step3: Increment memory pointer to read next no.~~

~~Step4: Decrement byte counters.~~

~~Step5: [compare two numbers]~~

~~if no < next no then;~~

~~perform Step 6.~~

~~Step6: Increment memory pointer to add next~~

~~Step7: Decrement counter by 1.~~

- Step 8: If byte counter is not equal then  
perform steps.  
Step 9: Store Smallest number.  
Step 10: End.

Algorithm for finding largest no. form 8-bit.

- Step 1: Initialize data segment, byte counter memory pointer.  
Step 2: Read no. from array.  
Step 3: Increment memory pointer to next variable.  
Step 4: Decrement byte counter.  
Step 5: Compare two no. if no > next no. then Perform step 6.  
Step 6: Replace no. with next no.  
Step 7: increment memory pointer to read next number from array.  
Step 8: Decrement counter by 1.  
Step 9: If byte counter is not equal then perform steps.  
Step 10: Store largest no.  
Step 11: End.

Conclusion: Once, we learnt smallest & greatest no. from the given series.

Ques 68/10

## PROGRAM TO FIND THE SMALLEST NUMBER IN THE GIVEN SERIES OF NUMBERS

### CODE:

```

model small
data
    ARRAYLIST DB 10h, 30h, 40h, 50h, 20h ; Define an Array
    MAX    DB 20h ; Variable to store the max value in memory
    LENGTH DB 05h ; Length of the array

code
MAIN PROC
    MOV AX, 0000h
    MOV DS, AX      ; Initialise the DS

    MOV AX, 0000h ; Initialize register to null
    MOV BX, 0000h
    MOV CX, 0000h

    MOV CL, LENGTH ; Store Length in CL

    MOV SI, OFFSET ARRAYLIST ; Load base address of array list
    MOV AL, [SI]           ; store first element of array in AL

UP: ; declare block UP, for the loop
    MOV BL, [SI] ; Get ith element in BL
    CMP AL, BL ; Compare AL > BL
    JG LESSER ; If AL > BL, then go to LESSER block

CONTINUE: ; continue from here after store max
    INC SI ; Increment SI to point to the next element in arraylist
LOOP UP
    MOV MIN, AL

    MOV AH, 4Ch ; Clear registers, terminate program and return the control to OS
    INT 21h

LESSER: ; JUMP here if AL > BL
    MOV AL, BL ; Store BL in AL, as BL is current MIN
    JMP CONTINUE ; Continue from where we jumped

MAIN ENDP

```

### OUTPUT:

register	0711:0015
AX	00 10 0712F 80 138 E
BX	00 00 0712F 80 138 E
CX	00 05 0712F 80 138 E
SI	00 00 0712F 80 138 E
DI	0711 0712F 80 138 E
BP	0017 0712F 80 138 E
SP	0710 0712F 80 138 E
IP	0000 0712F 80 138 E
CS	0000 0712F 80 138 E
DS	0000 0712F 80 138 E
ES	0000 0712F 80 138 E
FS	0000 0712F 80 138 E
GS	0000 0712F 80 138 E

PROGRAM TO FIND THE GREATEST NUMBER IN THE GIVEN SERIES OF NUMBERS

**CODE:**

```

model small
data
    ARRAYLIST DB 10h, 30h, 40h, 50h, 20h ; Define an Array
    MAX      DB 00h ; Variable to store the max value in memory
    LENGTH   DB 05h ; Length of the array
code
MAIN PROC
    MOV AX, @data
    MOV DS, AX ; Initialise the DS
    MOV AX, 0000h ; Initialize register to null
    MOV BX, 0000h
    MOV CX, 0000h
    MOV CL, LENGTH ; Store Length in CL
    MOV SI, OFFSET ARRAYLIST ; Load base address of array list
    MOV AL, [SI] ; store first element of array in AL

    UP: ; declare block UP, for the loop
        MOV BL, [SI] ; Get 1st element in BL
        CMP AL, BL ; Compare AL < BL
        JL GREATER ; If AL < BL, then go to GREATER block
    CONTINUE: ; continue from here after store max
        INC SI ; Increment SI to point to the next element in arraylist
    LOOP UP
    MOV MAX, AL ; Store the Max value in the MAX variable (in memory)
    MOV AH, 4Ch ; Clear registers, terminate program and return the control to OS
    INT 21h
    GREATER: ; JUMP here if AL < BL
        MOV AL, BL ; Store BL in AL, as BL is current MAX
        JMP CONTINUE ; Continue from where we jumped
MAIN ENDP

```

**OUTPUT:**

Registers	H	L	Address	Op-Code	Mnemonic	Operands
AX	00	50	0711:0024	471100	MOV	BL, [SI]
BX	00	50		071101	CMP	BL, BL
CX	00	0F		071102	JL	R27h
DX	00	00		071103	INT	21h
SI	0711	F7		071104	LOOP	R17h
				071105	MOV	10000000h, AL
				071106	MOV	AL, R17h
				071107	INT	21h
				071108	MOV	AL, BL
				071109	INT	21h
				07110A	MOV	AL, R17h
				07110B	MOV	AL, R17h
				07110C	MOV	AL, R17h
				07110D	MOV	AL, R17h
				07110E	MOV	AL, R17h
				07110F	MOV	AL, R17h
				071110	MOV	AL, R17h
				071111	MOV	AL, R17h
				071112	MOV	AL, R17h
				071113	MOV	AL, R17h
				071114	MOV	AL, R17h
				071115	MOV	AL, R17h
				071116	MOV	AL, R17h
				071117	MOV	AL, R17h
				071118	MOV	AL, R17h
				071119	MOV	AL, R17h
				07111A	MOV	AL, R17h
				07111B	MOV	AL, R17h
				07111C	MOV	AL, R17h
				07111D	MOV	AL, R17h
				07111E	MOV	AL, R17h
				07111F	MOV	AL, R17h
				071120	MOV	AL, R17h
				071121	MOV	AL, R17h
				071122	MOV	AL, R17h
				071123	MOV	AL, R17h
				071124	MOV	AL, R17h
				071125	MOV	AL, R17h
				071126	MOV	AL, R17h
				071127	MOV	AL, R17h
				071128	MOV	AL, R17h
				071129	MOV	AL, R17h
				07112A	MOV	AL, R17h
				07112B	MOV	AL, R17h
				07112C	MOV	AL, R17h
				07112D	MOV	AL, R17h
				07112E	MOV	AL, R17h
				07112F	MOV	AL, R17h
				071130	MOV	AL, R17h
				071131	MOV	AL, R17h
				071132	MOV	AL, R17h
				071133	MOV	AL, R17h
				071134	MOV	AL, R17h
				071135	MOV	AL, R17h
				071136	MOV	AL, R17h
				071137	MOV	AL, R17h
				071138	MOV	AL, R17h
				071139	MOV	AL, R17h
				07113A	MOV	AL, R17h
				07113B	MOV	AL, R17h
				07113C	MOV	AL, R17h
				07113D	MOV	AL, R17h
				07113E	MOV	AL, R17h
				07113F	MOV	AL, R17h
				071140	MOV	AL, R17h
				071141	MOV	AL, R17h
				071142	MOV	AL, R17h
				071143	MOV	AL, R17h
				071144	MOV	AL, R17h
				071145	MOV	AL, R17h
				071146	MOV	AL, R17h
				071147	MOV	AL, R17h
				071148	MOV	AL, R17h
				071149	MOV	AL, R17h
				07114A	MOV	AL, R17h
				07114B	MOV	AL, R17h
				07114C	MOV	AL, R17h
				07114D	MOV	AL, R17h
				07114E	MOV	AL, R17h
				07114F	MOV	AL, R17h
				071150	MOV	AL, R17h
				071151	MOV	AL, R17h
				071152	MOV	AL, R17h
				071153	MOV	AL, R17h
				071154	MOV	AL, R17h
				071155	MOV	AL, R17h
				071156	MOV	AL, R17h
				071157	MOV	AL, R17h
				071158	MOV	AL, R17h
				071159	MOV	AL, R17h
				07115A	MOV	AL, R17h
				07115B	MOV	AL, R17h
				07115C	MOV	AL, R17h
				07115D	MOV	AL, R17h
				07115E	MOV	AL, R17h
				07115F	MOV	AL, R17h
				071160	MOV	AL, R17h
				071161	MOV	AL, R17h
				071162	MOV	AL, R17h
				071163	MOV	AL, R17h
				071164	MOV	AL, R17h
				071165	MOV	AL, R17h
				071166	MOV	AL, R17h
				071167	MOV	AL, R17h
				071168	MOV	AL, R17h
				071169	MOV	AL, R17h
				07116A	MOV	AL, R17h
				07116B	MOV	AL, R17h
				07116C	MOV	AL, R17h
				07116D	MOV	AL, R17h
				07116E	MOV	AL, R17h
				07116F	MOV	AL, R17h
				071170	MOV	AL, R17h
				071171	MOV	AL, R17h
				071172	MOV	AL, R17h
				071173	MOV	AL, R17h
				071174	MOV	AL, R17h
				071175	MOV	AL, R17h
				071176	MOV	AL, R17h
				071177	MOV	AL, R17h
				071178	MOV	AL, R17h
				071179	MOV	AL, R17h
				07117A	MOV	AL, R17h
				07117B	MOV	AL, R17h
				07117C	MOV	AL, R17h
				07117D	MOV	AL, R17h
				07117E	MOV	AL, R17h
				07117F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
				07118A	MOV	AL, R17h
				07118B	MOV	AL, R17h
				07118C	MOV	AL, R17h
				07118D	MOV	AL, R17h
				07118E	MOV	AL, R17h
				07118F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
				07118A	MOV	AL, R17h
				07118B	MOV	AL, R17h
				07118C	MOV	AL, R17h
				07118D	MOV	AL, R17h
				07118E	MOV	AL, R17h
				07118F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
				07118A	MOV	AL, R17h
				07118B	MOV	AL, R17h
				07118C	MOV	AL, R17h
				07118D	MOV	AL, R17h
				07118E	MOV	AL, R17h
				07118F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
				07118A	MOV	AL, R17h
				07118B	MOV	AL, R17h
				07118C	MOV	AL, R17h
				07118D	MOV	AL, R17h
				07118E	MOV	AL, R17h
				07118F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
				07118A	MOV	AL, R17h
				07118B	MOV	AL, R17h
				07118C	MOV	AL, R17h
				07118D	MOV	AL, R17h
				07118E	MOV	AL, R17h
				07118F	MOV	AL, R17h
				071180	MOV	AL, R17h
				071181	MOV	AL, R17h
				071182	MOV	AL, R17h
				071183	MOV	AL, R17h
				071184	MOV	AL, R17h
				071185	MOV	AL, R17h
				071186	MOV	AL, R17h
				071187	MOV	AL, R17h
				071188	MOV	AL, R17h
				071189	MOV	AL, R17h
			</			



## Practical No. 10

Title: Write an Assembly language program (ALP) to arrange the given numbers in ascending & descending order.

### Theory:-

- Sorting:- In this we are going to bubble sort method to sort the elements of series in both ascending & descending orders.

- Algorithm for ascending order:-

Step 1:- Start.

Step 2:- Initialize data segment.

Step 3:- Set BX = CX = -1.

Step 4:- Load base address of array to SI.

Step 5:- Load I<sup>th</sup> & (I<sup>th</sup> + 1) element to AX & DX.

Step 6:- Compare I<sup>th</sup> & (I<sup>th</sup> + 1) element.

Step 7:- Swap if 1<sup>st</sup> is greater.

Step 8:- Move back values in memory.

Step 9:- Increment Pointer.

Step 10:- Loop (+0 swap) till bx is 0

Step 11:- Traverse to 'n'.

Step 12:- Exit.

- Algorithm for descending order:-

Step 1:- Initialize data segment.



- Step 2:- Set BX = CX = -1  
Step 3:- Load base address of array in SI.  
Step 4:- Load  $i^{th}$  &  $(i^{th} + 1)$  element in AX & DX.  
Step 5:- Compare  $i^{th}$  &  $(i^{th} + 1)$  element.  
Step 6:- Swap if  $i^{th}$  element is lesser.  
Step 7:- Move back value in memory.  
Step 8:- Increment pointer.  
Step 9:- Loop (to swap loop) till BX is 0.  
Step 10:- Traverse for 'n' times.  
Step 11:- Exit.

Conclusion:- In this practical, I came to know about arranging numbers in ascending/descending orders.

Ques  
Ans  
09/10

PROGRAM TO ARRANGE A SERIES OF GIVEN NUMBER IN ASCENDING ORDER

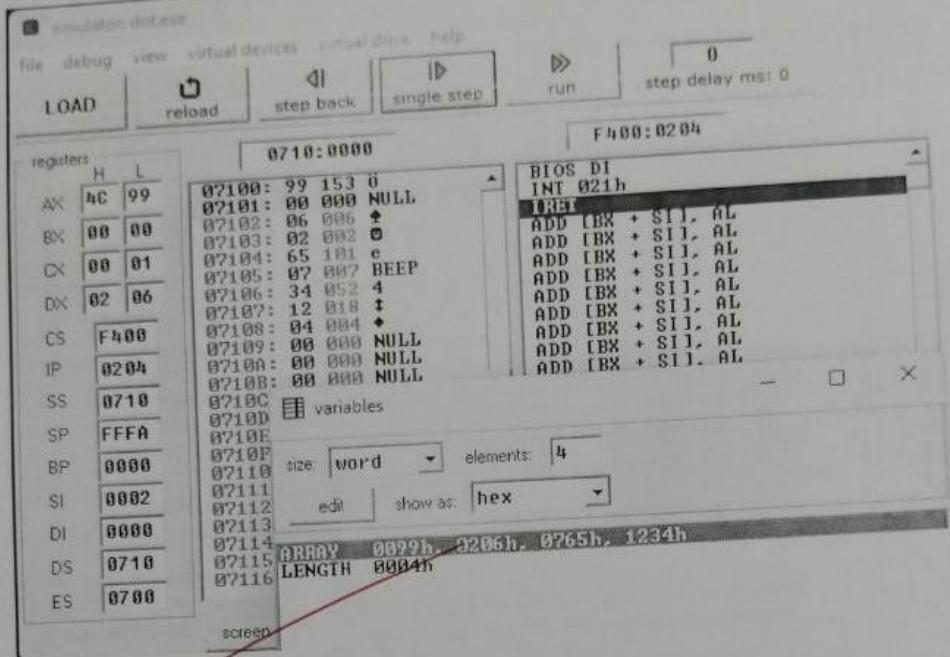
CODE:

```
.MODEL SMALL
.DATA
    ARRAY DW 0765h, 123Ah, 0009h, 0206h
    LENGTH DW 0004h
```

CODE

```
MAIN PROC
    MOV AX, BDATA
    MOV DS, AX      ; Initialize DS
    MOV CX, LENGTH ; USING BUBBLE SORT ALGORITHM
    TRAVERSE:
        MOV BX, CX      ; SET bx = cx - 1
        DEC BX
        JZ END
        LEA SI, ARRAY   ; load base address of ARRAY in SI
        SWAPLOOP:
            MOV AX, [SI]
            MOV DX, [SI + 02h] ; Load ith and (i+1)th element in AX, and DX
            CMP AX, DX      ; Compare ith and (i+1)th element
            JG SWAP          ; swap if ith element is greater
        CONTINUE:
            MOV [SI], AX      ; Move back the values in memory
            MOV [SI + 02h], DX
            INC SI           ; Increment Pointer
            DEC BX
            JNZ SWAPLOOP    ; loop (to SWAPLOOP) till bx is 0
            LOOP TRAVERSE    ; traverse for n (length of array) times
        END:
            MOV AH, 4CH
            INT 21h
        SWAP:
            XCHG AX, DX      ; USE XCHG Instruction to swap
            JMP CONTINUE
MAIN ENDP
```

**OUTPUT:**



PROGRAM TO ARRANGE A SERIES OF GIVEN NUMBER IN DESCENDING ORDER

CODE:

.MODEL SMALL

DATA  
ARRAY DW 0755h, 1234h, 0099h, 0206h  
LENGTH DW 0004h

MAIN PROC

MOV AX, 0DATA

MOV DS, AX ; Initialize DS

MOV CX, LENGTH

TRAVERSE:

JZ END

DEC BX

MOV BX, CX

LEA SI, ARRAY

MOV DX, [SI + 02h]

SWAP:

MOV AX, [SI]

MOV DX, [SI + 02h]

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

MAIN ENDP

DMP CONTINUE

XCHG AX, DX

SWAP:

END:

LOOP TRAVERSE

DEC BX

INC SI

INC SI

MOV [SI + 02h], DX

CONTINUE:

JL SWAP

COMP AX, DX

; Compare i-th and (i+1)th elements

; swap if i-th element is lesser

SWAP:

MOV AX, [SI]

MOV DX, [SI + 02h]

; load i-th and (i+1)th elements in AX, and DX

SWAP:

LEA SI, ARRAY

; load base address of ARRAY in SI

JZ END

DEC BX

MOV BX, CX

TRAVERSE:

MOV CX, LENGTH

CODE:

MAIN PROC

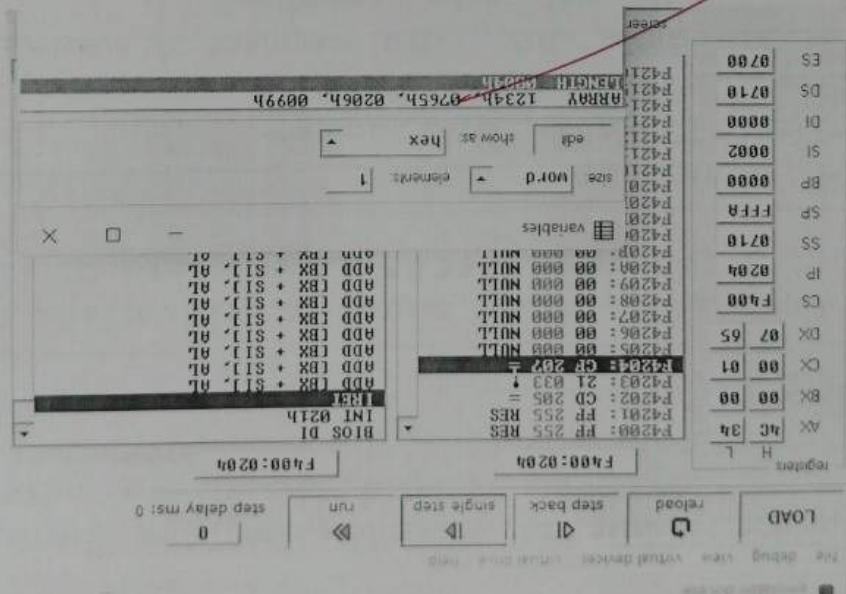
MOV AX, 0DATA

MOV DS, AX ; Initialize DS

DATA

ARRAY DW 0755h, 1234h, 0099h, 0206h

LENGTH DW 0004h



OUTPUT:



### Practical No. 11.

Title:- Write an Assembly language Program (ALP) to string related descending order.

#### Theory:-

- i) Copy String from source to destination  
Instruction: REP - repeat until 0.

#### Algorithm:

Step 1:- Initialize data segment.

Step 2:- Take Source string.

Step 3:- Take an extra segment & overlap it with data segment.

Step 4:- Load base address of destination DI

Step 5:- Load base address of Source to SI

Step 6:- Add length to counter variable.

Step 7:- Repeat instruction MOV SB.  
until CX is 0.

Step 8:- end.

#### ii) Reverse a string:

Instruction: Push:- pushes element into the stack.  
Pops:- Pops element out of stack.

#### Algorithm:

Step 1: Initialize data Segment.



### Practical No. 11.

Title:- Write an Assembly language Program (ALP) to string related descending order.

#### Theory:-

- i) Copy string from source to destination.  
Instruction: REP - repeat until 0.

#### Algorithm:

Step 1:- Initialize data segment.

Step 2:- Take source string.

Step 3:- Take an extra segment & overlap it with data segment.

Step 4:- Load base address of destination DI

Step 5:- Load base address of source to SI

Step 6:- Add length to counter variable.

Step 7:- Repeat instruction MOV SB.

until CX is 0.

Step 8:- end.

- ii) Reverse a string:

Instruction: Push:- pushes element into the stack.

Pops:- Pops element out of stack.

#### Algorithm:

Step 1: Initialize data Segment.



- Step 2: Take base address of string & load to SI  
Step 3: push every characters of string to stack.  
Step 4: POP characters out of stack.  
Step 5: End.

Conclusion: ~~In this practical, I came to know about string handling assembly language.~~

Ques 08/10