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
- Capacitance (in some models)

Multimeters can be anlog or digital, with digital multimeters being more common. They usually have:
- Multiple measurement ranges
- Auto-ranging or manual ranging options.
- Data hold & storage capacities
- Additional features like temperature measurement 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
- Seperate 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 area
- Power supply (fixed or adjustable)
- Components like:
  - LED's
  - Switches
  - Resistors
  - Capacitors
  - IC sockets
  - 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 & 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
- Gain practical knowledge in digital electronics and circuit design.

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

08/10

Practical No. 02

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

⊹ Required IC's →

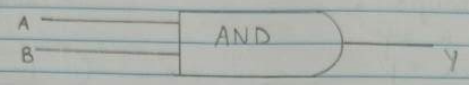
1.	7408	AND
2.	7432	OR
3.	7404	NOT
4.	7486	EX-OR
5.	74266	EX-NOR
6.	7400	NAND
7.	7402	NOR

⊹ Theory:-

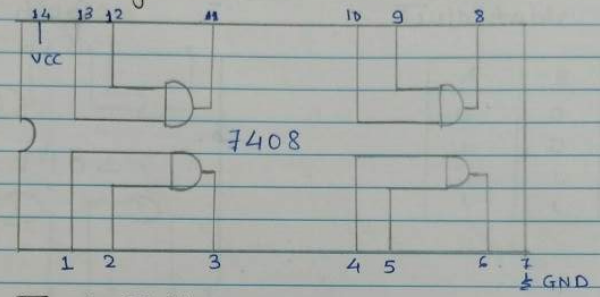
There are total 3 basic gates (AND, OR, NOT), 2 derived gates (EX-OR, EX-NOR) & 2 universal gates (NAND, NOR).

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 :-

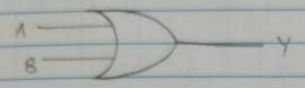
$Y = A \cdot B$

$= A \times B$

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. Symol:-



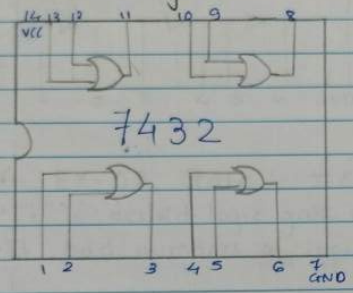
2. Equation:-

$Y = A \text{ OR } B \text{ OR } \ldots N$

$Y = A + B + \ldots + N$

$\therefore Y = A + B.$

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.

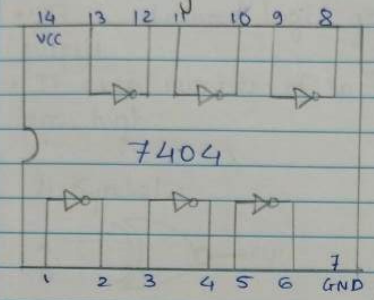
- It is logic gate chip with single input & single output.
- Result is always compliment of input.

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 derived logic gate with two inputs & 1 output.
- If odd numbers of input is high then result is high.

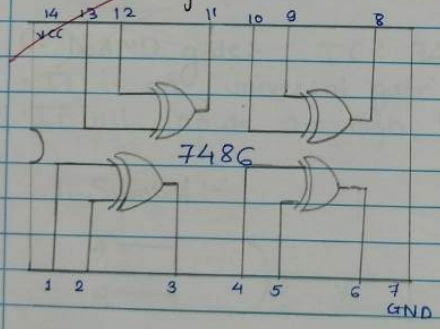
1. Symbol:-

2. Equation:-

$Y = A \oplus B$

$= A'B + AB'$

3. Pin diagram:-



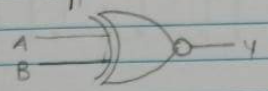
4. Truth Table.

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:-

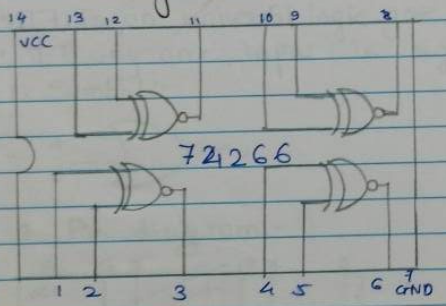


2. Equation.

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

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

3. Pin diagram:-



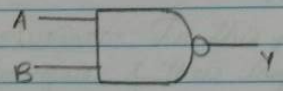
4. Truth table.

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

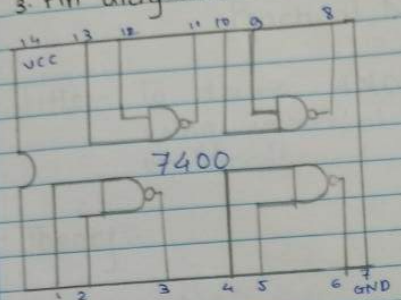
1. Symbol:-



2. Equation:-

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

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

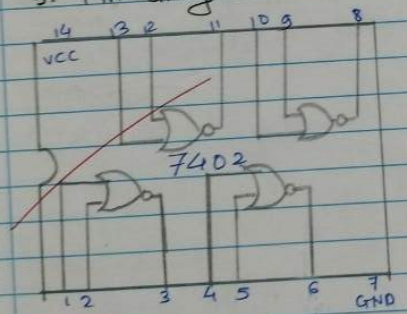
- It is an universal logic gate with two inputs & 1 output.
- If any one input is high then result is low.

1. Symbol:-

2. Equation:-

$Y = \overline{A + B}$

3. Pin diagram:-



4. Truth table.

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

09/10

Conclusion:- In this practical we learnt about various logic gates & their logical eq^{n}s & also pin dia with their truth tables.

Practical No. 03.

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

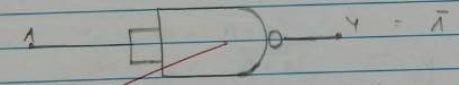
Theory:-

Deriving NOT gate usin NAND gate:-

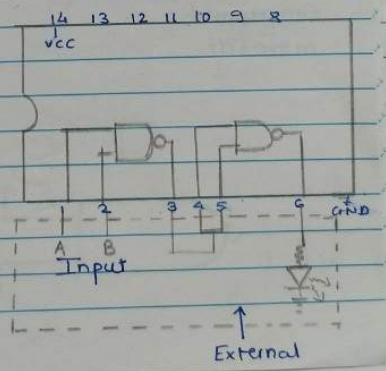
A. Procedure:-

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	$A \cdot B$	$\bar{A} \cdot \bar{B}$	$A \cdot B$
0	0	0	0	1
0	1	0	0	1
1	0	0	0	1
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 & then connecting with one more NAND gate.

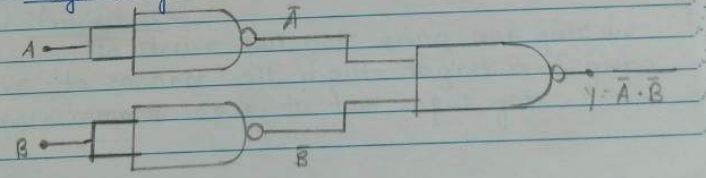
Boolean expression for OR gate:-

$\therefore Y = A + B$

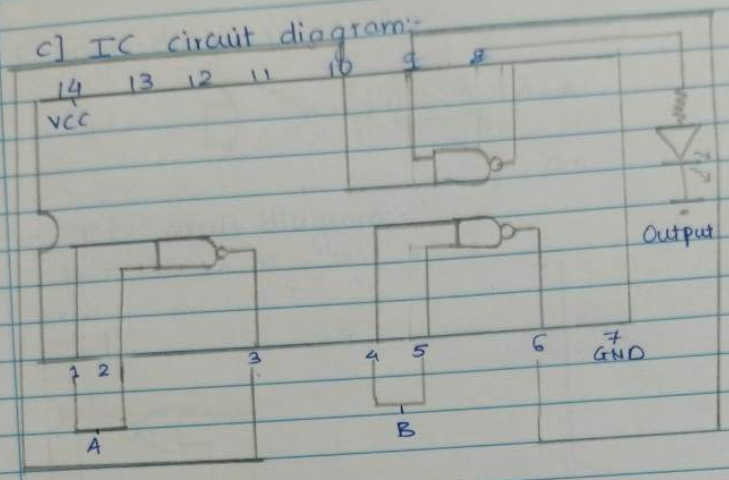
$\therefore Y = \overline{\bar{A} + \bar{B}}$

$\therefore Y = \overline{\bar{A} \cdot \bar{B}}$ ... by demorgan's theorem.

B]. Logic diagram:-



c] IC circuit diagram:-



D] Truth table:-

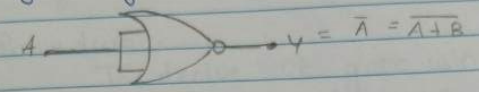
Input		Output				
A	B	A+B	$\bar{A}$	$\bar{B}$	$\bar{A}\cdot\bar{B}$	$\overline{\bar{A}\cdot\bar{B}}$
0	0	0	1	1	1	0
0	1	1	1	0	0	1
1	0	1	0	1	0	1
1	1	1	0	0	0	1

3. Deriving NOT gate using NOR gate:-

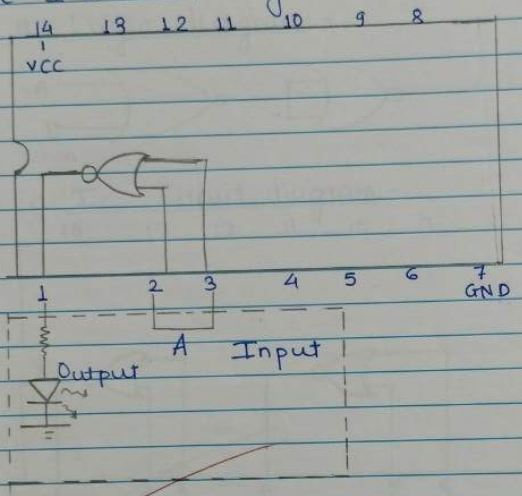
A) Procedure:-

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

B. logic diagram:-



C. I C circuit diagram:-



D. Truth table:-

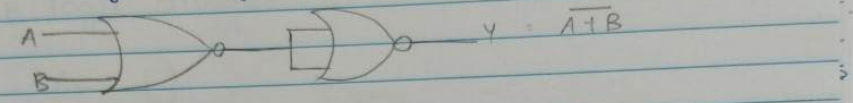
Input		Output		
A	B	$\bar{A}$	A+B	$\overline{A+B}$
0	0	1	0	1
1	1	0	1	0

5. Deriving OR gate using NOR gate:

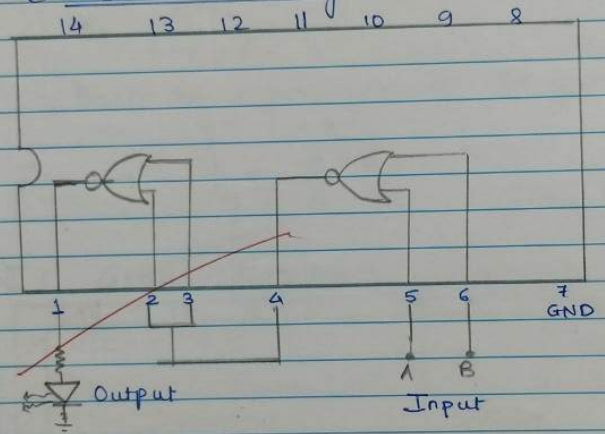
1. Procedure:-

To derive OR gate using NOR gate, we have to invert output of NOR gate.

B. Logic diagram:-



C. Ic Circuit diagram:-



D. Truth table:-

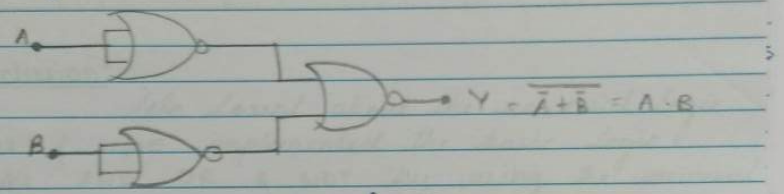
Input		Output.		
A	B	A + B	$\overline{A+B}$	$\overline{\overline{A+B}}$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

6. Derive AND gate using NOR gate:-

A. Procedure:-

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

B. logic diagram:-

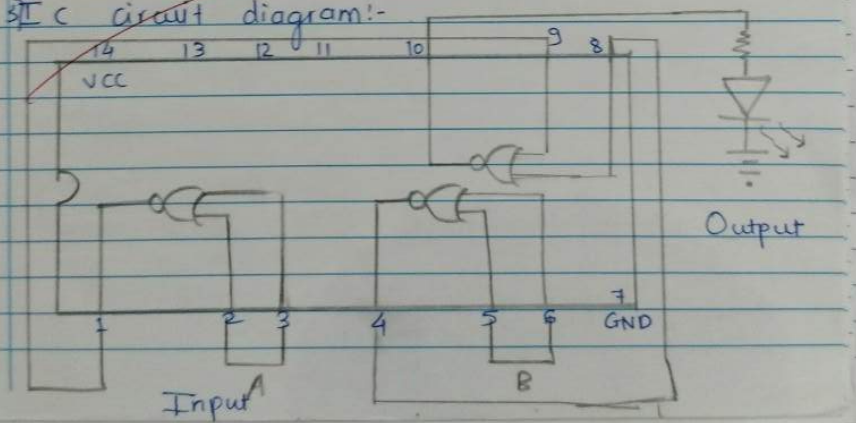


Boolean expression for OR gate:-

$$\therefore Y = \frac{A+B}{\bar{A} \cdot \bar{B}}$$

$$Y = \bar{A} + \bar{B} \quad \text{... using demorgan's theorm}$$

C. IC circuit diagram:-



D] Truth table:-

Input		Output				
A	B	$A \cdot B$	$\bar{A}$	$\bar{B}$	$\bar{A}+\bar{B}$	$\overline{\bar{A}+\bar{B}}$
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 & also implemented the basic logic gates AND, OR & NOT by using the universal gates NAND & NOR only.

09/10

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.

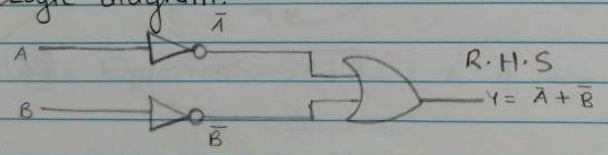
In the equation form they are:-

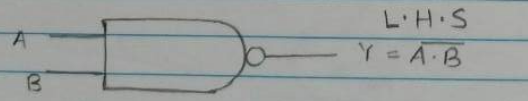
1. $\overline{AB} = \bar{A} + \bar{B}$: The complement of product is equal to sum of complements.
2. $\overline{A+B} = \bar{A} \cdot \bar{B}$: The complement of sum is equal to product of complements.

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

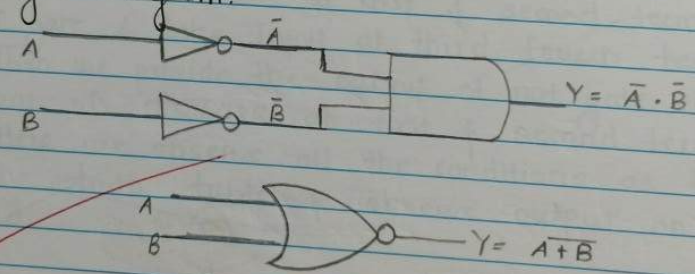
iv. Truth table :-

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

2. Demorgan's Second Theorem:-

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

ii. Logic diagram:-



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

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

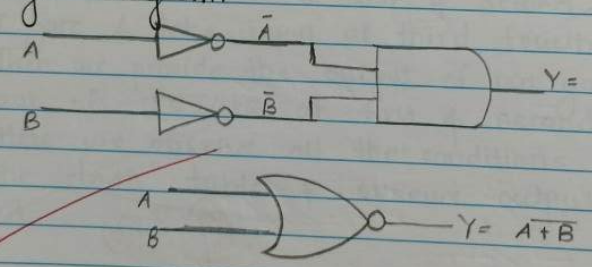
iv. Truth table :-

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

2. Demorgan's Second Theorem:-

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

ii. Logic diagram:-



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

iv. Truth table:-

L.H.S			R.H.S		
A	B	$\bar{A}+\bar{B}$	$\bar{A}$	$\bar{B}$	$\bar{A}\cdot\bar{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	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 IC7486 at first & second terminal of IC
  - Here, we observe all the conditions as follows in the truth table & observe output on bread-board.

09/10

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

Practical No.05

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

Theory:-

- $y = \bar{A}.\bar{B}.\bar{C}.\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + A\bar{B}CD.$

Answer:-

This equation is in SOP form.

$\therefore \bar{A}\bar{B}\bar{C}\bar{D} = m_1$ $\quad \therefore \bar{A}\bar{B}\bar{C}D = m_5$

$\therefore \bar{A}BCD = m_7$ $\quad \therefore \bar{A}BC\bar{D} = m_6$

$\therefore AB\bar{C}\bar{D} = m_{12}$ $\quad \therefore AB\bar{C}D = m_{13}$

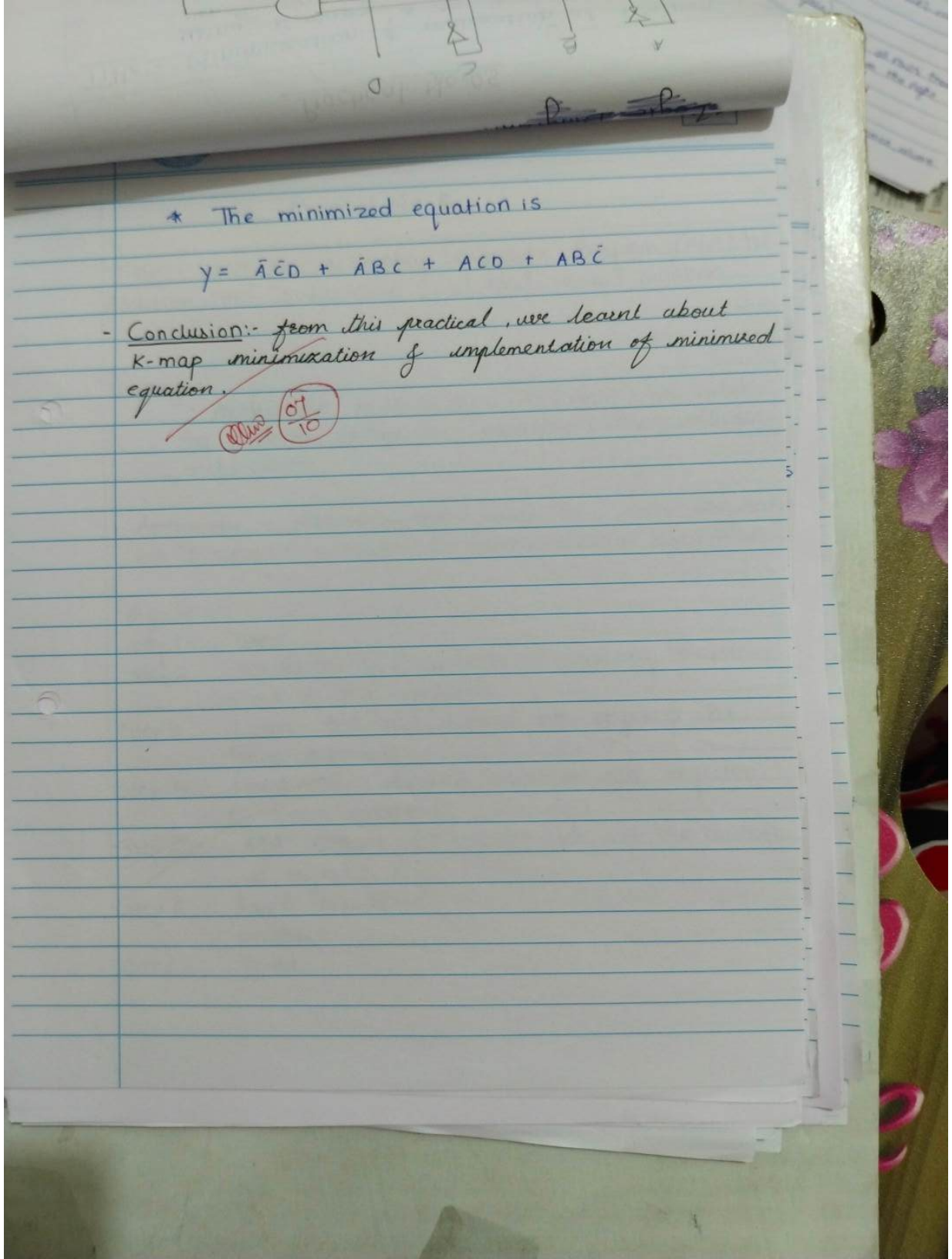
$\therefore ABCD = m_{15}$ $\quad \therefore A\bar{B}CD = m_{11}$

The given equation is min-terms is.

$y = \Sigma m(1, 5, 6, 7, 11, 12, 13, 15)$

* K-map for given equation is:-

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

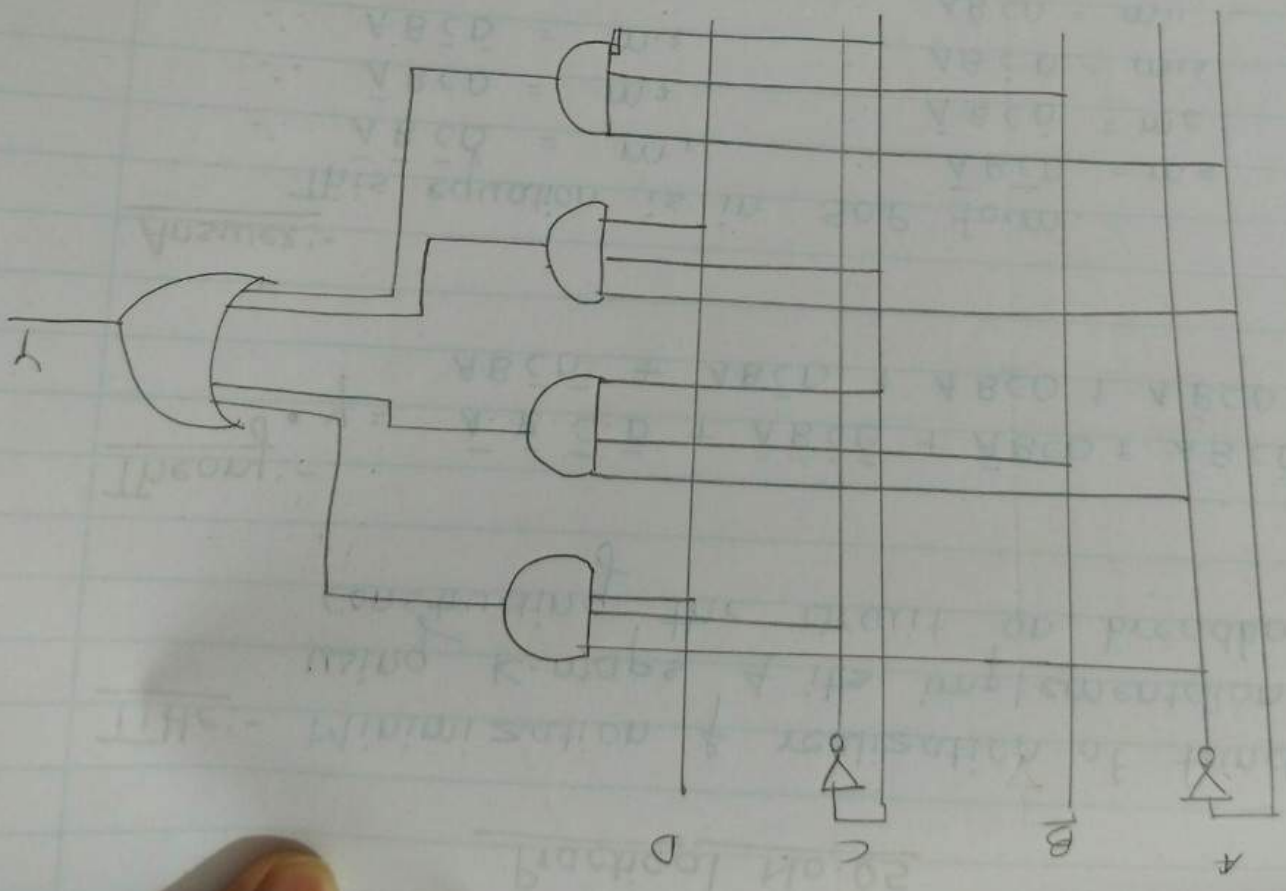


* The minimized equation is

$$Y = \bar{A}\bar{C}D + \bar{A}BC + ACD + AB\bar{C}$$

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

07/10



inimized equation is

* The minimized equation is

$$Y = \bar{A}\bar{C}D + \bar{A}BC + ACD + AB\bar{C}$$

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

07/10

Practical No. 5

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 substration.
- Arithmetic:- ADD instructions adds the data destination & sourse operand & stores data in destination.

-Algorithm:

step1: Start
step2: 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.

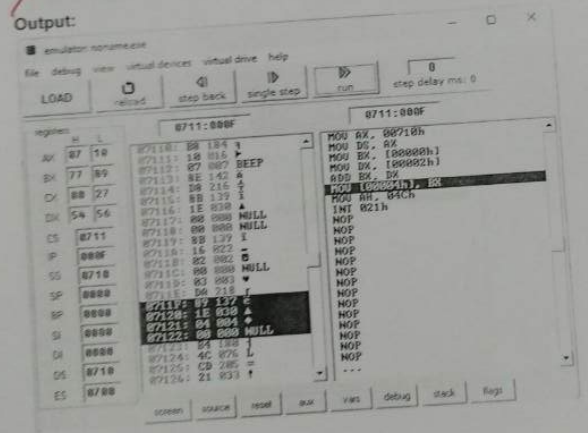
08/10

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:

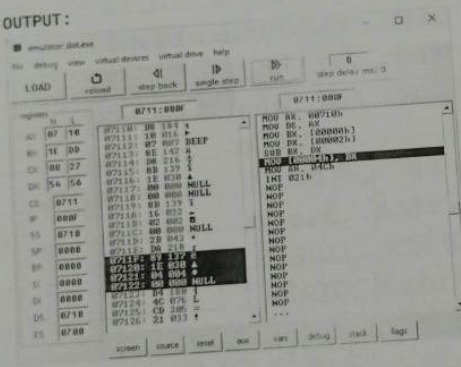


PROGRAM TO SUBTRACT TWO 16-BIT NUMBERS

CODE:

```
.MODEL SMALL
.DATA
    N2   DW 7333H          ; initialized and assigned value to N2.
    N3   DW 5456H          ; initialized and assigned value to N3.
    DIFF DW ?              ; initialized var variable.

.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
        SUB BX, DX      ; SUBTRACTING BH to DH
        MOV DIFF, BX    ; moving BH value to DIFF variable.

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

OUTPUT:



Practical No. 07

Title:- Write an Assembly language Program (ALP) 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: Intialize the contents by memory.
Step 3: load the first number of into Ax register
Step 4: load the second number into Bx register.
Step 5: Multiply Bx with Ax
Step 6: More data from Ax register to RESULT variables.
Step 7: More higher word of multiplication to higher word of RESURT variables.
Step 8: End.

• Division: DIV instruction performs division of two operands.

Algorithm:

Step 2: start
Step 3: Intialize 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: Hence we had computer practical on assembly language program for multiplication & division

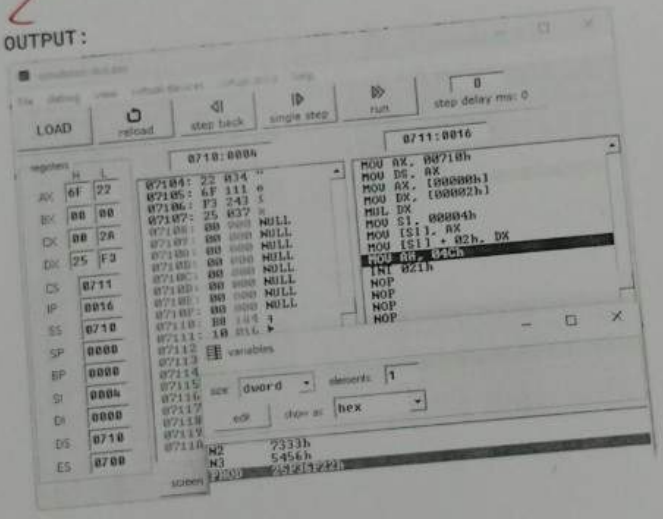
08/10

PROGRAM TO MULTIPLY TWO 16-BIT NUMBERS

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
        MOV DS, AX         ; Initialize DS

        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:

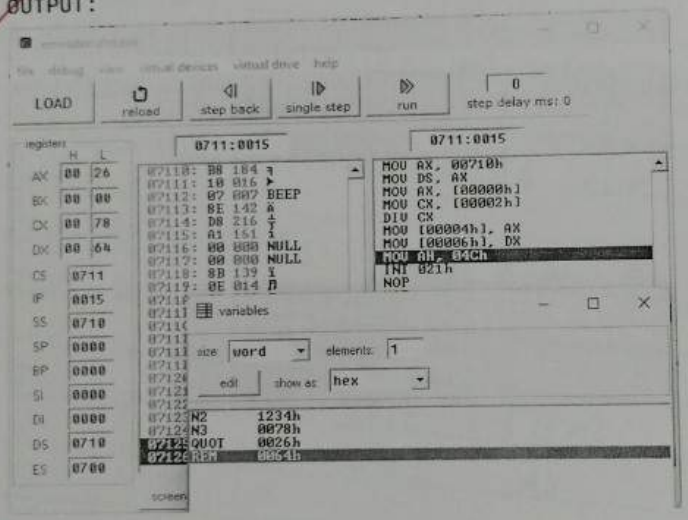


PROGRAM TO DIVIDE TWO 16-BIT NUMBERS

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 (ALP) to sum of given series of number. (8 bit / 16 bit)

Theory:-

1) INC :-

- Syntax: INC Destination.
- Destination will be register or memory location & 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 form content of operand
- Immidiate data can note be operand of this instruction.
- Example:

```
INC [BX]
INC AX
```

2) DEC :-

- Syntax: DEC destination.
- Destination will be register or memory location & it will be 8bit or 16 bit.
- 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 lable.

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

Example:
```
INZ UP
INZ continue.
```

- Algorithm:

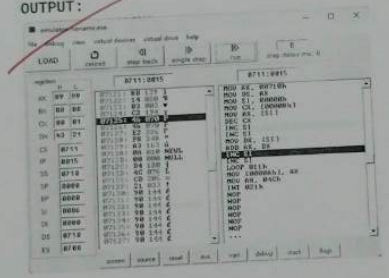
Step 1: Start
Step 2: Initialize variable numlist & count with datatype DBI define bytes for 8bits.
Step 3: Initidlize data segments.
Step 4: Move the offset of NUMLIST to CL registe
Step 5: Move the content of COUNT to CL register.
Step 6: Move 0000H to Ax & Bx register & result store in AX.
Step 7: Add content of Bx & Ax & result. [si] into BL.
Step 8: Increment si by 1 82 for 8 16 bit.
Step 9: Decrement CL by 1
Step 10: Ends.

PROGRAM TO ADD A GIVEN SERIES OF 16-BIT NUMBERS

CODE:

```
.MODEL SMALL
.DATA
    ARRAY  DW 2333H, 1111h, 1234h, 4321h  ; initialize array
    LENGTH DW 0004H   ; length of the array
    SUM    DW ?       ; define sum
.CODE
    MAIN PROC
        MOV AX, @DATA
        MOV DS, AX      ; Initialize DS

        LEA SI, ARRAY   ; store the base address of array in SI
        MOV CX, LENGTH  ; store length of array in Counter

        MOV AX, [SI]    ; Load first element in AX
        DEC CX          ; Decrement Counter
        INC SI          ; Increment element pointer
        INC SI

        NEXT:
            MOV DX, [SI] ; MOV next element in DX
            ADD AX, DX   ; ADD dx to previous sum
            INC SI       ; increment element pointer
            INC SI
        LOOP NEXT        ; Loop until all variables are added

        MOV SUM, AX      ; moving total to SUM variable
        MOV AH, 4Ch      ; Exit the program and return control to OS
        INT 21h
    MAIN ENDP
```

OUTPUT:

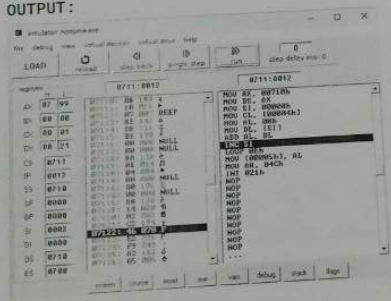


PROGRAM TO ADD A GIVEN SERIES OF 8-BIT NUMBERS

CODE:

```
.MODEL SMALL
.DATA
    ARRAY  DB 33H, 11h, 34h, 21h  ; initialize array
    LENGTH DB 04H   ; length of the array
    SUM    DB ?       ; define sum

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

        LEA SI, ARRAY   ; store the base address of the array in SI
        MOV CL, LENGTH  ; store length of the array in Counter

        MOV AL, 00h     ; initialize AX with 0
        NEXT:
            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:



Practical No. 09

Title:- Write an Assembly language program (ALP) 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: intialize 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 preform steps.
Step 10: Store largest no.
Step 11: End.

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

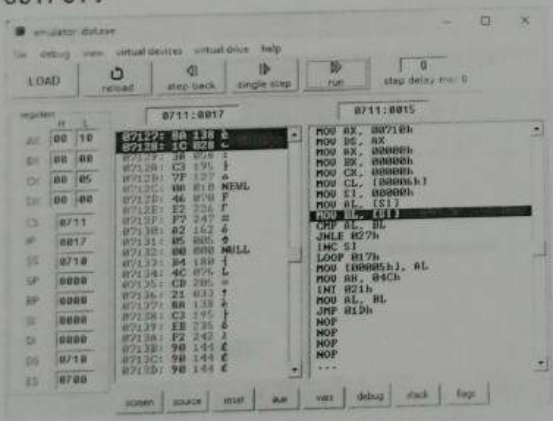
08/10

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
    MIN       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 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 ; Contunue from where we jumped
    MAIN ENDP
```

OUTPUT:

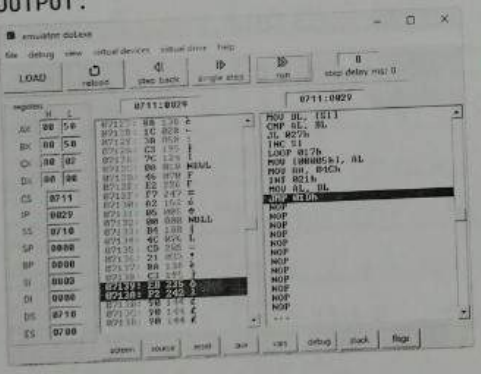


PROGRAM TO FIND THE GREATEST NUMBER IN THE GIVEN SERIES OF NUMBERS

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 ith 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 varaible (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 ; Contunue from where we jumped
   MAIN ENDP
```

OUTPUT:



Practical No. 10

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

Theory:-

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

- **Algorithm for ascending order:-**
  Step 1:- Start.
  Step 2:- Intialize data segment.
  Step 3:- Set BX = CX = -1.
  Step 4:- load base address of array to SI.
  Step 5:- load I^{th} & $(I^{th}+1)$ element to AX & DX.
  Step 6:- Compare I^{th} & $(I^{th}+1)$ element.
  Step 7:- swap is 1st is greater.
  Step 8:- Move back values in memory.
  Step 9:- Increment Pointer.
  Step 10:- loop (to swap) till bx is 0
  Step 11:- Traverse to 'n'
  Step 12:- Exit.

- **Algorithm for decending order:-**
  Step 1:- intialize 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.

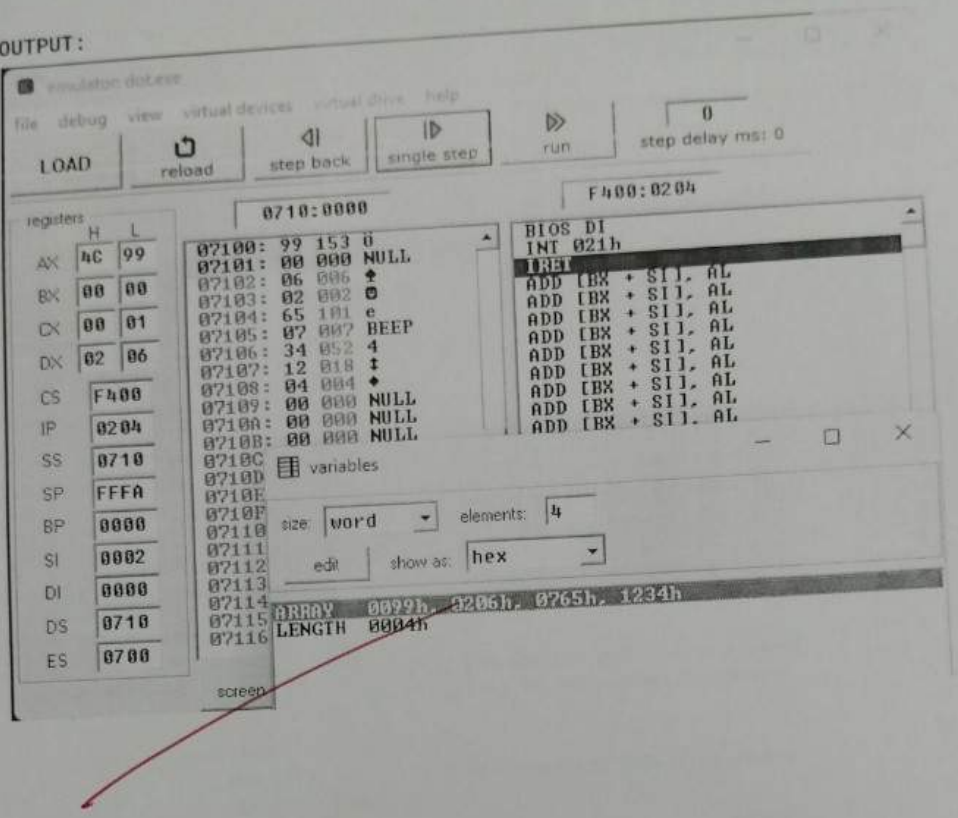
09/10

PROGRAM TO ARRANGE A SERIES OF GIVEN NUMBER IN ASCENDING ORDER

CODE:

```
.MODEL SMALL
.DATA
    ARRAY  DW 0765h, 1234h, 0009h, 0206h
    LENGTH DW 0004h

.CODE
    MAIN PROC
        MOV AX, @DATA
        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
                INC SI
                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 :

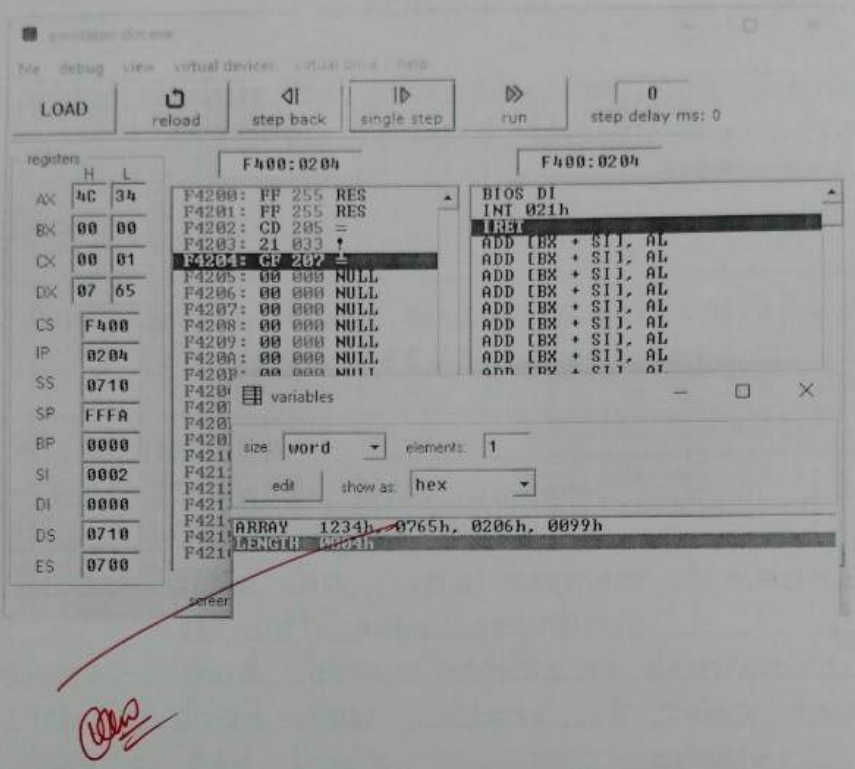

emulator dot exe
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 4C 99
BX 00 00
CX 00 01
DX 02 06
CS F400
IP 0204
SS 0710
SP FFFA
BP 0000
SI 0002
DI 0000
DS 0710
ES 0700
0710:0000
07100: 99 153 Ö
07101: 00 000 NULL
07102: 06 006
07103: 02 002
07104: 65 101 e
07105: 07 007 BEEP
07106: 34 052 4
07107: 12 018
07108: 04 004
07109: 00 000 NULL
0710A: 00 000 NULL
0710B: 00 000 NULL
0710C
0710D
0710E
0710F
07110
07111
07112
07113
07114
07115
07116
screen
F400:0204
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
variables
size: word
elements: 4
edit
show as: hex
ARRAY 0099h, 0206h, 0765h, 1234h
LENGTH 0004h

PROGRAM TO ARRANGE A SERIES OF GIVEN NUMBER IN DESCENDING ORDER

CODE:

```
.MODEL SMALL
.DATA
    ARRAY  DW 0765h, 1234h, 0099h, 0206h
    LENGTH DW 0004h

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

        MOV CX, LENGTH
        TRAVERSE:
            MOV BX, CX
            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
                JL SWAP                 ; swap if ith element is lesser

                CONTINUE:
                    MOV [SI], AX        ; Move back the values in memory
                    MOV [SI + 02h], DX

                INC SI                  ; Increment Pointer
                INC SI
                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 :


file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX 4C 34
BX 00 00
CX 00 01
DX 07 65
CS F400
IP 0204
SS 0710
SP FFFA
BP 0000
SI 0002
DI 0000
DS 0710
ES 0700
F400:0204
F400:0204
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ≟
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
variables
size word
elements 1
edit
show as hex
ARRAY 1234h, 0765h, 0206h, 0099h
LENGTH 0004h

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:- Repead 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 if 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:- Repead 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 if stack.

- Algorithm:
Step 1: Initialize data Segment.

Step2: Take base address of string & load to SI
Step 3: push every characters if 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.

08/10