DEPARTMENT OF ELECTRONICS AND COMMUNICATION

MICROPROCESSOR LABORATORY (15ECL47)
IV SEMESTER- ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY MANUAL

ACADEMIC YEAR 2017 – 2018
### Labor Resources

<table>
<thead>
<tr>
<th>Laboratory Code</th>
<th>IA Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>15ECL47</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Lecture</th>
<th>Exam</th>
</tr>
</thead>
<tbody>
<tr>
<td>01Hr Tutorial (Instructions)</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hours/Week</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 02 Hours Laboratory</td>
<td></td>
</tr>
<tr>
<td>Exam</td>
<td>03</td>
</tr>
<tr>
<td>Hours</td>
<td></td>
</tr>
</tbody>
</table>

### Laboratory Experiments:

1. **Programs involving:**

   **Data transfer instructions like:**
   - i) Byte and word data transfer in different addressing Modes
   - Block move (with and without overlap)
   - iii) Block interchange

2. **Programs involving:**

   **Arithmetic & logical operations like:**
   - i) Addition and Subtraction of multi precision nos.
   - ii) Multiplication and Division of signed and unsigned Hexadecimal nos.
   - iii) ASCII adjustment instructions
   - iv) Code conversions

3. **Programs involving:**

   **Bit manipulation instructions like checking:**
   - i) Whether given data is positive or negative
   - ii) Whether given data is odd or even
   - iii) Logical 1’s and 0’s in a given data
   - iv) 2 out 5 code
   - v) Bit wise and nibble wise palindrome

4. **Programs involving:**

   **Loop instructions like**
   - i) Arrays
     - addition/subtraction of N nos., Finding largest and smallest nos., Ascending and descending order
   - ii) Two application programs using Procedures and Macros (Subroutines)
<table>
<thead>
<tr>
<th>5</th>
<th>Programs involving</th>
</tr>
</thead>
<tbody>
<tr>
<td>String manipulation like string transfer, string reversing, searching for a string</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>Programs involving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs to use DOS interrupt INT 21h Function calls for Reading a Character from keyboard, Buffered Keyboard input, Display of character/ String on console</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>Interfacing Experiments: Experiments on interfacing 8086 with the following interfacing modules through DIO (Digital Input/Output - PCI bus compatible card / 8086 Trainer )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Matrix keyboard interfacing</td>
<td></td>
</tr>
<tr>
<td>2. Seven segment display interface</td>
<td></td>
</tr>
<tr>
<td>3. Logical controller interface</td>
<td></td>
</tr>
<tr>
<td>4. Stepper motor interface</td>
<td></td>
</tr>
<tr>
<td>5. Analog to Digital Converter Interface (8 bit)</td>
<td></td>
</tr>
<tr>
<td>6. Light dependent resistor ( LDR ), Relay and Buzzer Interface to make light operated switches</td>
<td></td>
</tr>
</tbody>
</table>

List of Experiments
<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>TITLE OF THE EXPERIMENT</th>
<th>PAGE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>INTRODUCTION TO 8086 MICROPROCESSOR</td>
<td>1-5</td>
</tr>
<tr>
<td>B</td>
<td>TUTORIALS - Creating source code</td>
<td>vi-xi</td>
</tr>
</tbody>
</table>

**PART A**

**Assembly Language Programs (ALP)**

1. Programs Involving
   Data transfer instructions

1.1 Write an ALP to move block of data without overlap 1-3
1.2 Write an ALP to move block of data with overlap 4-5
1.3 Program to interchange a block of data 6-7

2. Programs Involving
   Arithmetic & logical operations

2.1A Write an ALP to add 2 Multibyte no. 8-9
2.1B Write an ALP to subtract two Multibyte numbers 10-11
2.2A Write an ALP to multiply two 16-bit numbers 12-13
2.2B Write an ALP to divide two numbers 14-15
2.3A Write an ALP to multiply two ASCII no.s 16-17
2.4A Develop and execute an assembly language program to perform the conversion from BCD to binary 18-19
2.4B Write an ALP to convert binary to BCD 19-20
2.5A Write an ALP to find the square of a number 21-21
2.5B Write an ALP to find the cube of a number 22-22
2.5C Write an ALP to find the LCM of two 16bit numbers 23-24
2.5D Write an ALP to find the GCD of two 16bit unsigned numbers 25-26
2.5E Write an ALP to find the factorial of a given number using recursive procedure 27-28

3. Programs Involving
   Bit manipulation instructions like checking

3.1 Write an ALP to separate odd and even numbers 29-30
3.2 Write an ALP to separate positive and negative numbers 31-32
3.3 Write an ALP to find logical ones and zeros in a given data 33-33
3.4 Write an ALP to find whether the given code belongs 2 out of 5 code or not 34-35
3.5A Write an ALP to check bitwise palindrome or not 36-36
3.5B Write an ALP to check whether the given number is nibble wise palindrome or not 37-38

4. Programs Involving
   Branch/Loop instructions

4.1 Write an ALP to find largest no. from the given array 23-23
4.2 Write an ALP to find smallest no from the given array 41-41
4.3 Write an ALP to sort a given set of 16bit unsigned integers into ascending order using bubble sort algorithm 42-43

5. Programs Involving
   String manipulation
A. **INTRODUCTION TO 8086 MICROPROCESSOR**

**8086 Internal Block diagram**

8086 is a 16-bit processor having 16-bit data bus and 20-bit address bus. The block diagram of 8086 is as shown. (Refer figures 1A & 1B). This can be subdivided into two parts; the Bus Interface Unit (BIU) and Execution Unit (EU).

**Bus Interface Unit:**
The BIU consists of segment registers, an adder to generate 20 bit address and instruction prefetch queue. It is responsible for all the external bus operations like opcode fetch, mem read, mem write, I/O read/write etc. Once this address is sent OUT of BIU, the instruction and data bytes are fetched from memory and they fill a 6-byte First in First out (FIFO) queue.

**Execution Unit:**

The execution unit consists of: General purpose (scratch pad) registers AX, BX, CX and DX; Pointer registers SP (Stack Pointer) and BP (Base Pointer); index registers source index (SI) & destination index (DI) registers; the Flag register, the ALU to perform operations and a control unit with associated internal bus. The 16-bit scratch pad registers can be split into two 8-bit registers. AX ⇒ AL, AH; BX ⇒ BL, BH; CX ⇒ CL, CH; DX ⇒ DL, DH.

Figure 1A

![Execution Unit Diagram](image)

Figure 1B
Note: All registers are of size 16-bits

Different registers and their operations are listed below:

<table>
<thead>
<tr>
<th>Register</th>
<th>Uses/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>As accumulator in Word multiply &amp; Word divide operations, Word I/O operations</td>
</tr>
<tr>
<td>AL</td>
<td>As accumulator in Byte Multiply, Byte Divide, Byte I/O, translate, Decimal Arithmetic</td>
</tr>
<tr>
<td>AH</td>
<td>Byte Multiply, Byte Divide</td>
</tr>
<tr>
<td>BX</td>
<td>As Base register to hold the address of memory</td>
</tr>
<tr>
<td>CX</td>
<td>String Operations, as counter in Loops</td>
</tr>
<tr>
<td>CL</td>
<td>As counter in Variable Shift and Rotate operations</td>
</tr>
<tr>
<td>DX</td>
<td>Word Multiply, word Divide, Indirect I/O</td>
</tr>
</tbody>
</table>
### Execution of Instructions in 8086:

The microprocessor sends OUT a 20-bit physical address to the memory and fetches the first instruction of a program from the memory. Subsequent addresses are sent OUT and the queue is filled up to 6 bytes. The instructions are decoded and further data (if necessary) are fetched from
memory. After the execution of the instruction, the results may go back to memory or to the output peripheral devices as the case may be.

**8086 Flag Register format**

<table>
<thead>
<tr>
<th>BIT</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>OF</td>
<td>DF</td>
<td>IF</td>
<td>TF</td>
<td>SF</td>
<td>ZF</td>
<td>U</td>
<td>AF</td>
<td>U</td>
<td>PF</td>
<td>U</td>
<td>CF</td>
</tr>
</tbody>
</table>

**Legend:**
- **U** = UNDEFINED
- **OF** = OVERFLOW FLAG
- **DF** = SINGLE STEP TRAP FLAG
- **IF** = INTERRUPT ENABLE FLAG
- **TF** = STRING DIRECTION FLAG
- **SF** = OVERFLOW FLAG
- **ZF** = ZERO FLAG
- **AF** = AUXILIARY CARRY FLAG FOR BCD
- **CF** = CARRY FLAG
- **PF** = PARITY FLAG

(a) : CARRY FLAG – SET BY CARRY OUT OF MSB
(b) : PARITY FLAG – SET IF RESULT HAS EVEN PARITY
(c) : AUXILIARY CARRY FLAG FOR BCD
(d) : ZERO FLAG – SET IF RESULT = 0
(e) : SIGN FLAG = MSB OF RESULT
(f) : SINGLE STEP TRAP FLAG
(g) : INTERRUPT ENABLE FLAG
(h) : STRING DIRECTION FLAG
(i) : OVERFLOW FLAG
Generation of 20-bit Physical Address:

LOGICAL ADDRESS

SEGMENT REGISTER  0000

ADDER

20 BIT PHYSICAL MEMORY ADDRESS

Programming Models:
Depending on the size of the memory the user program occupies, different types of assembly language models are defined.

TINY ⇒ All data and code in one segment
SMALL ⇒ one data segment and one code segment
MEDIUM ⇒ one data segment and two or more code segments
COMPACT ⇒ one code segment and two or more data segments
LARGE ⇒ any number of data and code segments

To designate a model, we use “.MODEL” directive.
B. TUTORIALS - Creating source code

The source code consists of 8086/8088 program memories, appropriate pseudo-Opcodes and assembler directives. The first is created with a text editor and is given an extension ASM. The text editor may be any word processor (ex., EDLIN, NE) that can produce standard ASCII code.

Assembling the program
To assemble the program two assemblers are available for the IBM-PC. They are: Microsoft Macro Assembler (MASM) and Borland Turbo Assembler (TASM).

Besides doing the tedious task of producing the binary codes for the instruction statements, an assembler also allows the user to refer to data items by name rather by numerical addresses. This makes the program much more readable. In addition to program instructions, the source program contains directives to the assembler. Pseudo instructions are assembler directives entered into the source code along with the assembly language.

Once the program written completely, it can be assembled to obtain the OBJ file by executing MASM. The assembly language program file name should be mentioned along with the command.

MASM<file name.ASM>

The <file name.ASM> file that contains the assembly language program is assembled. The assembler generates error messages if there are any error (Syntax errors).

These errors are listed along with the line number. If there are no errors then .OBJ file is created. To obtain the .EXE file the user has to LINK the .OBJ file.
LINK <file name>; or TLINK <file name>;

If a file is smaller than 64K bytes it, can be converted from an execution file to a command file (.COM). The command file is slightly different from an execution file (.EXE).

In a command file the program must be originated at location 100H before it can execute. This means that the program must be no longer than (64K-100H) in length. The command file requires less space in memory than the equivalent execution file. The system loads .COM file off the disk into the computer memory more quickly than the execution file. To create a .COM file from a .EXE file, we need the EXE2BIN converter EXE2BIN converts .EXE file to .COM or binary file.

Example: EXE2BIN <filename><file name.com>

The <filename> with an EXE extension is converted to <filename> with .com extension with the above command.

**Test and Debug**

The executable program can be run under DOS or DEBUG. As a thumb rule a program under DOS only when there is no error or it produces some not visible or audible result. If the program result is stored in registers or in memory, the result is visible. Hence it should be run using DEBUG or TD (Turbo Debugger) or code-view only. .EXE file can be loaded into memory using DEBUG.

Example: DEBUG<filename.EXE>

Using DEBUG it is possible to find the bugs in the program. After loading it into the memory it is possible to check and correct the errors using different commands in DEBUG. Some of the commands are as follows:

- **G-GO**
  Format: G[offset][, offset]
  Action: Executes a program starting at the current location offset values are temporary breakpoints. Upon encounter of a breakpoint instruction the processor stops and displays registers and flag contents.

- **T – TRACE**
  Format: T [Instruction count]
  Action: Executes one or more instructions and displays register and flag values for each of them. Example: T: Executes only the next instructions
  T5: Executes the next 5 instructions

- **P- PTRACE**
  Format: P [instruction count]
  Action: Same as Trace, but treats subroutine calls, interrupts, loop instructions, and repeat String instructions as a single instruction

- **Q-QUIT**
  Format: Q
  Action: Exists to dos.
N-Name the program
  Format: N <filename>
  Action: Name the program

W-Write the file to disk
  Format: W
  Action: Bytes the starting from the memory location whose address is provided by IP addresses and written as a .COM file to the disk. The number of bytes that are to be stored is indicated by the contents of the CX Register. The name of the file is to be specified by means of the N command prior to executing the W command.

R-Register
  Format: R <register file name>
  Action: The contents of register are displayed additionally, the register content can replace by the value entered by the user. If no register name is provided, the contents of all the register are displayed

A-Assemble
  Format: A<CS: offset>
  Action: This command allows us to enter the assembler mnemonics directly.

U- Unassemble
  Format: U<CS: offset>
  Action: This command lists a program from the memory. The memory start location is specified by CS: offset.

L-Load
  Format: L[address][drive][first sector][number]
  Action: Reads sectors from the disk into memory. The memory start address is provided in the command

E-Enter
  Format: E<address> [list]
  Action: It enables us to change the contents of the specified memory location.
  List is an optional data that has to be entered.

A program can be written and debugged using the following additional techniques.

U.1. Very carefully define them program to solve the problem in hand and work out the best algorithm you can.
U.2. If the program consists of several parts, write, test and debug each part individually and then include parts one at a time.
U.3. If a program or program section does not work, first recheck the algorithm to make sure it really does what you want it to. You might have someone else look at it also.
U.4. If the algorithm seems correct, check to make sure that you have used the correct instructions to implement the algorithm. Work out on paper the effect that a series of instructions will have on some sample data. These predictions on paper can later be compared with the actual results producer when the program section runs.
If you don’t find a problem in the algorithm or the program instruction use debugger to help you localize the problem. Use single step or trace for short program sections. For longer programs use breakpoints. This is often a faster technique to narrow the source of the problem down to a small region.

**Program Development**

The first step to develop a program is to know “What do I really want this program to do?” As you think about the problem, it is good idea to write down exactly what you want the program to do and the order in which you want the program to do it. At this point, no program statement is written but just the operation in general terms.

Flowcharts are graphic shapes to represent different types of program operations. The specific operation desired is written by means of graphic symbols. Flowcharts are generally used for simple programs or program sections.

Steps to convert an algorithm to assembly language:

1. Set up and declare the data structure for the algorithm you are working with.
2. Write down the instructions required for initialization at the start of the code section.
3. Determine the instructions required to implement the major actions taken in the algorithm, and decide how data must be positioned for these instructions.
4. Insert the instructions required to get the data in correct position.

**Assembler Instruction Format**

The general format of an assembler instruction is

Label: Opcode & Operand, Mnemonic Operand, Operand; comments

The inclusion of spaces between label Opcode, operands, mnemonics and comments are arbitrary, except that at least one space must be inserted if no space would lead to anambiguity (e.g. between the mnemonic and first operand). There can be no spaces within a mnemonic or identifier and spaces within string constants or comments will be included as space characters. Each statement in program consists of fields.

Label: It is an identifier that is assigned the address of the first byte of the instruction in which it appears. The presence of a label in an instruction is optional, but, if present, the label provides a symbolic name that can be used in branch instruction to branch to the instruction. If there is no label, then the colon must not be entered. All labels begin with a letter or one of the following special character: @, $,’ – or?. A label may be any length from 1 to 35 characters. A label appears in a program to identify the name of memory location for storing data and for other purposes.

Opcode and Operands: The Opcode field is designed to hold the instruction Opcode. To the right of Opcode field is the operand field, which contains information used by the Opcode.

Mnemonic: All instructions must contain a mnemonic. The mnemonic specifies the operation to be executed.

Operand: The presence of the operands depends on the instruction. Some instructions have no operands; some have one operand, and some two. If there are two operands, they are separated by a comma.

Comments: The comment field is for commenting the program and may contain any combination of characters. It is optional and if it is deleted the semicolon may also be deleted. A comment may appear on a line by itself provided that the first character on the line is a semicolon.
Program Format and assembler Directives

The typical assembler program construct for 8086/8088:

The MODEL directive selects a standard memory model for the assembly language program. A memory model may be thought of a standard blue print or configuration, which determines the way segments are linked together. Each memory model has a different set of restrictions as to the maximum space available for code and data. But the most important thing to know about model is that they affect the way that subroutines and data may be reached by program.

This table summarizes the different types of models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Description (Memory Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny</td>
<td>Code and Data combined must be &lt;=64K</td>
</tr>
<tr>
<td>Small</td>
<td>Code &lt;=64K; Data &lt;=64K</td>
</tr>
<tr>
<td>Mediu</td>
<td>Data &lt;=64K; Code any size</td>
</tr>
<tr>
<td>Compa</td>
<td>Code &lt;=64K; Data any size</td>
</tr>
<tr>
<td>Large</td>
<td>Both code and data may be &gt;64K</td>
</tr>
<tr>
<td>Huge</td>
<td>same as the large model, except that arrays may be Large than 64k</td>
</tr>
</tbody>
</table>

A program running under DOS is divided into 3 primary segments (point to by CS) contains program code; the data segment (pointed to by DS) contains the program variables, the stack segment (pointed to by SS) contains the program stack.

" .DATA" directive (line 2) indicates the start of the data segment. It contains the program variables.

" .CODE" directive (line k) indicates the start of the code segment. The end directive (line n) indicates the end of the program file.

Another program construct for 8086/8088
User can use code view to debug the program by following the steps given below:

- Write the program in a file with .ASM extension using an editor [PRETEXT Editor which saves it in ASCII].
  
  **Ex:** `EDIT TEST1.ASM`

- Assemble the program using the command MASM/ZI file name;
  
  **Ex:** `MASM TEST1.ASM`

- Link the program using the command LINK/CO file name;
  
  **Ex:** `LINK TEST1.OBJ`

- To debug use `DEBUG FILENAME.EXE`

F1 – Step by step,  F2 – Step by Procedure,  F4 - Help

CMD > MO A ON

Switch between DOS screen and AFDEBUG screen using F6

Note: F1, F2, F4, F6 are Function Keys in Keyboard

All the command of debug can be used to display the program. You have an advantage to see the result of the program typing the variable name, instead of using dump command. The variable name is provided using "?".
Experiment No.1.1.                                                                                       Date:

AN ALP TO MOVE A BLOCK OF DATA WITHOUT OVERLAP

Aim:
To Write an ALP to Move a Block of Data without Overlap

Software Required:
Masm 16 Bit

Algorithm:
1. Define block of data
2. Save memory for block transfer as block2
3. Load block1 into SI
4. Load block2 into DI
5. Initialize counter
6. Move first data into DI
7. Repeat step 6 until counter is zero
8. End

Program:

.MODEL SMALL
.DATA

BLK1 DB 01,02,03,04,05,06,07,08,09,0AH BLK2 DB 10
DUP (?)
COUNT DW 0AH

.CODE

MOV AX, @DATA
MOV DS, AX
MOV ES, AX
MOV SI, OFFSET BLK1;
MOV DI, OFFSET BLK2
MOV CX, COUNT
AGAIN: CLD

REP MOVS
MOV

Pre Viva Questions:

1. List all the modern microprocessor
2. Name some 16 bit Processor (8086, 80286, 80386L, EX)
3. Name some 32 bit processors (80386DX, 80486, PENTIUM OVERDRIVE)
4. Name some 64 bit processor (Pentium, Pentium pro, Pentium II, Xeon, Pentium III, and Pentium IV)
5. List the address bus width and the memory size of all the processor
The Block Of Data Defined In The Program Is Moved To Destination Without Overlap And Output Is Verified.

**Conclusion**

Output Is Verified For Different Bytes Of Data And Is Successfully Moved From Default Source Address To Verification And Validation.

The Block Of Data Defined In The Program Is Moved From Source To Destination Without Overlap Successfully.

---

**Result:**

The Block Of Data Defined In The Program Is Moved From Source To Destination Without Overlap Successfully.

**Verification And Validation:**

Output Is Verified For Different Bytes Of Data And Is Successfully Moved From Default Source Address To Destination Address Without Overlap.

**Conclusion:**

The Block Of Data Defined In The Program Is Moved To Destination Without Overlap And Output Is Verified.
Post Viva Questions:

1. The Memory Map Of Any Ibm Compatible Pc Consists Of Three Main Parts, Name Them [Transient Memory Area, System Area, Extended Memory System]
2. The First I Mb Of The Memory Area Is Called As .............. (Real Memory Area)
4. The System Area Contain Programs In .............Memory(Rom)
5. What Are The Main Two Parts Of 8086 Internal Architecture.(Biu,Eu)
6. Name The Registers In Biu (Cs, Ds, Es, Ss, Ip)
Experiment No.1.2.  

Write An Alp To Move Block Of Data With Overlap

Aim:
To Write An Alp To Move Block Of Data With Overlap

Software Required:
Masm 16 Bit

Algorithm:
1. Define block of data
2. Reserve memory for block transfer as block2
3. Move block1 address to SI
4. Move block2 address to DI
5. Initialize counter
6. Point DI to block+ n
7. Move block1 data to block2
8. Repeat step 7 until counter is zero
9. End

Program:

```
.MODEL SMALL
.DATA
    BLK1 DB 01,02,03,04,05,06,07,08,09,0AH
    BLK2 DB 10 DUP (?)
.CODE
    MOV AX, @DATA ; MOV THE STARTING ADDRESS
    MOV DS, AX
    MOV ES, AX
    MOV SI, OFFSET BLK1 ; SET POINTER REG TO BLK1
    MOV DI, OFFSET BLK2 ; SET POINTER REG TO BLK2
    MOV CX, 0AH ; SET COUNTER
    ADD SI, 0009H
    ADD DI, 0004H
    MOV AL, [SI]
    MOV [DI], AL
    DEC SI
    DEC DI
    DEC CL ; DECREMENT COUNTER
    JNZ AGAIN ; TO END PROGRAM
    MOV AH, 4CH
    INT 21H
END
```

Pre Viva Questions:

1. Name the registers in EU. (AX, BX, CX, DX, SP, BP, SI, DI)
2. Name the flag registers in 8086. (O, D, I, T, S, Z, A, P, C)
3. How is the real memory segmented?
4. What is the advantage of segmentation?
5. Name the default segment and offset register combinations.
Output:
Before Execution

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>E</td>
<td>4</td>
<td>C</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>E</td>
<td>4</td>
<td>C</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Result:
The Block Of Data Defined In The Program Is Moved From Source To Destination With Overlap Successfully.

Verification And Validation:
Output Is Verified For Different Bytes Of Data And Is Successfully Moved From Default Source Address To Destination Address With Overlap.

Conclusion:
The Block Of Data Defined In The Program Is Moved To Destination With Overlap And Output Is Verified.

Post Viva Questions:

1. What is the relocatable program.
2. Name the three main addressing modes in 8086.
3. Name the data addressing modes. And the program addressing modes. Give examples
4. Explain MOV AL, ‘A’, MOV AX, NUMBER, MOV [BP], DL, MOV CH,[1000], MOV[BX+SI],SP, MOV ARRAY[SI],BL, MOV DH,[BX+DI+10H]
Experiment No.1.3.                                                                                      Date:

Program To Interchange A Block Of Data

Aim:                                                                                                 Date:
To Program To Interchange A Block Of Data

Software Required:                                                                                     Date:
Masm 16 Bit

Algorithm:                                                                                               Date:
1. Define two sets of data.                                                                                    Date:
2. Load address of src to SI                                                                                   Date:
3. Load address of dst to DI                                                                                   Date:
4. Initialize counter                                                                                        Date:
5. Interchange data in src and dst                                                                             Date:
6. Repeat step 5 until counter = 0.                                                                              Date:
7. End                                                                                                        Date:

Program:                                                                                                    Date:
 .MODEL SMALL                                                                                                  Date:
 .DATA                                                                                                         Date:
    SRC DB 10H,20H,30H,40H,50h                                                                                      Date:
    DST DB 06,07,08,09,0AH COUNT                                                                                   Date:
    EQU 05                                                                                                         Date:
 .CODE                                                                                                         Date:
    MOV AX, @DATA ; INITIALIZE THE DATA REGISTER                                                                 Date:
    MOV DS, AX                                                                                                   Date:
    LEA SI, SRC                                                                                                 Date:
    LEA DI, DST                                                                                                 Date:
    MOV CL, COUNT ; INITIALIZE THE COUNTER                                                                     Date:
    BACK:                                                                                                         Date:
        MOV AL, [SI]                                                                                              Date:
        MOV BL, [DI]                                                                                              Date:
        MOV [SI], BL ; INTERCHANGE THE DATA                                                                        Date:
        MOV [DI], AL                                                                                              Date:
        INC SI                                                                                                     Date:
        INC DI                                                                                                     Date:
        DEC CL                                                                                                     Date:
        JNZ BACK                                                                                                   Date:
        MOV AH, 4CH                                                                                               Date:
        INT 21H                                                                                                    Date:
        END                                                                                                         Date:

Pre Viva Questions:                                                                                         Date:
1. Name the programme memory addressing modes. (Direct, relative, indirect)                                   Date:
2. What is an intersegment and intrasegment jump?                                                            Date:
3. Differentiate near and short jumps (+_32k and +127to_128 bytes)                                           Date:
4. Differentiate near and far jumps.                                                                         Date:
### OUTPUT:
#### BEFORE EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### AFTER EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>30</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Result:
Program Is Executed Without Errors And The Output Is Verified

### Verification And Validation:
Output Is Verified And Is Found Correct

### Conclusion:
The Blocks Of Data Defined In The Program Is Interchanged And Output Is Verified

### Post Viva Questions:

1. Differentiate push and pop instructions.
2. Explain PUSH word ptr [BX], POP F.
3. Explain JMP TABLE[BX]
4. Explain the following : ASSUME,DB,DD,DW,DQ,END
Write An Alp To Add 2 Multibyte No.s

Aim:
To Write An Alp To Add 2 Multibyte No.s

Software Required:
Masm 16 Bit

Algorithm:
1. Initialize the MSBs of sum to 0
2. Get the first number.
3. Add the second number to the first number.
4. If there is any carry, increment MSBs of sum by 1.
5. Store LSBs of sum.
6. Store MSBs of sum.

Program:

```
.MODEL SMALL
.DATA
N1 DQ 122334455667788H ; FIRST NUMBER
N2 DQ 122334455667788H ; SECOND NUMBER
SUM DT ?
.CODE
MOV AX, @DATA ; INITIALIZE THE DATA REGISTER
MOV DS, AX
LEA SI, N1 ; POINTER TO FIRST NUMBER
LEA DI, N2 ; POINTER TO SECOND NUMBER
LEA BX, SUM
MOV CL, 04H ; COUNTER FOUR WORD
CLC
BACK:
  MOV AX, [SI] ; MOVE FIRST WORD
  ADC AX, [DI]
  MOV [BX], AX
  INC SI
  INC SI
  INC DI
  INC DI
  INC BX
  INC BX
  DEC CL
  JNZ BACK ; REPEAT UNTIL COUNTER BECOMES ZERO
  JNC OVER
  MOV AX, 0001H
  MOV [BX], AX
OVER:
  MOV AH, 4CH
  INT 21H
END
```

Pre Viva Questions:

1. Give the opcode format for 8086 instructions. (op(1-2b),(mode,reg,rem),(displacement-0-2b))
2. Explain how the string instructions are executed.
   3. List some string instructions
   4. Explain the significance of REP Prefix.

---

Post Viva Questions:

1. Explain XCHG, LAHF, SAHF, XLAT
2. What are the two types of I/O addressing modes. (fixed port, variable port)
3. What do you mean by segment override prefix.
4. Explain the following directives. NEAR, FAR, BYTE PTR, ORG, OFFSET, ORG
   Differentiate END, ENDP, ENDM

---

Result:
Program Is Executed Without Errors And The Output Is Verified

Verification And Validation:
Output Is Verified And Is Found Correct

Conclusion:
The Addition Of Two Multibye Data Is Done And The Output Is Verified
Experiment No.2.1.B.  

Write An Alp To Subtract Two Multibyte Numbers

Aim:
To Write An Alp To Subtract Two Multibyte Numbers

Software Required:
MASM 16 BIT

Algorithm :
1. Initialize the MSBs of difference to 0
2. Get the first number
3. Subtract the second number from the first number.
4. If there is any borrow, increment MSBs of difference by 1.
5. Store LSBs of difference
6. Store MSBs of difference

Program:

.MODEL SMALL
.DATA
N1 DQ 12233455667788H ; FIRST NUMBER
N2 DQ 111111111111111H ; SECOND NUMBER
RESULT DT ?
.CODE
MOV AX, @DATA ; INITIALIZE THE DATA REGISTER
MOV DS, AX
LEA SI, N1 ; POINTER TO FIRST NUMBER
LEA DI, N2 ; POINTER TO SECOND NUMBER
LEA BX, RESULT
MOV CX, 04H ; COUNTER FOUR WORD
CLC
MOV AX, [SI] ; MOVE FIRST WORD
SBB AX, [DI]
MOV [BX], AX
INC SI
INC SI ; MOVE SI, DI CONTENTS
INC DI
INC DI
INC BX ; INCREMENT BX TO STORE RESULTS
INC BX
LOOP BACK
MOV AH, 4CH
INT 21H
END

---------------------------------------------------------------------

OUTPUT:
BEFORE EXECUTION
---------------------------------------------------------------------

AFTER EXECUTION
Result:
Program Is Executed Without Errors And The Output Is Verified

Verification And Validation:
Output Is Verified And Is Found Correct

Conclusion:
The Subtraction Of Two Multibyte Data Is Done And The Output Is Verified
Experiment No.2.2.A

Write An Alp To Multiply Two 16-Bit Numbers

Aim:
To Write An Alp To Multiply Two 16-Bit Numbers

Software Required:
Masm 16 Bit

Algorithm:
1. Get The Multiplier.
2. Get The Multiplicand
3. Initialize The Product To 0.
4. Product = Product + Multiplicand
5. Decrement The Multiplier By 1
6. If Multiplicand Is Not Equal To 0, Repeat From Step (D) Otherwise Store The Product.

Program:

.CODE
START:

MOV AX, @DATA
MOV DS, AX
MOV AX, MULTIPLICAND
MUL MULTIPLIER
MOV PRODUCT, AX
MOV PRODUCT+2, DX
MOV AH, 4CH
INT 21H
END START

==============================================================================

OUTPUT:
BEFORE EXECUTION

==============================================================================

DS:0000 1 0 0 E 4 C 2 00 F 0 F 0 0 0 0 0
DS:0010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0020 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0030 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0040 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

AFTER EXECUTION

==============================================================================

DS:0000 1 0 0 E 4 C 2 00 F 0 F 0 0 F 0 0
DS:0010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0020 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Result:
Program Is Executed Without Errors And The Output Is Verified

Verification And Validation:
Output Is Verified And Is Found Correct

Conclusion:
The Multiplication Of Two 16 Bit Data Is Done And The Output Is Verified
Aim:
To Write An Alp To Divide Two Numbers

Software Required:
Masm 16 Bit

Algorithm:
1. Get the dividend
2. Get the divisor
3. Initialize the quotient to 0.
4. Dividend = dividend – divisor
5. If the divisor is greater, store the quotient. Go to step g.
6. If dividend is greater, quotient = quotient + 1. Repeat from step (d)
7. Store the dividend value as remainder.

Program:

.MODEL SMALL
.DATA
    W1 DW 02222H
    W2 DW 1111H
    Q DW ?
    R DW ?
.CODE
    MOV AX, @DATA
    MOV DS, AX
    ; INITIALIZE DATA SEGMENT
    MOV AX, W1
    ; GET DIVIDEND
    MOV BX, W2
    ; GET DIVISOR
    DIV BX
    ; DIVIDE
    MOV Q, AX
    ; STORE QUOIENT
    MOV R, DX
    ; STORE REMAINDER
    MOV AH, 4CH
    INT 21H
    END
    ; END PROGRAM

OUTPUT:
BEFORE EXECUTION

| DS:0000 | 0 | 8 | 1 | 1 | 0 | E | 4 | C | 2 | 0 | 2 | 2 | 1 | 1 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

AFTER EXECUTION

| DS:0000 | 0 | 8 | 1 | 1 | 0 | E | 4 | C | 2 | 0 | 2 | 2 | 1 | 1 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
Result:
Program Is Executed Without Errors And The Output Is Verified

Verification And Validation:
Output Is Verified And Is Found Correct

Conclusion:
The Division Of Two Numbers Is Done And The Output Is Verified
Aim:
To Write An Alp To Multiply Two Ascii No.S

Software Required:
Masm 16 Bit

Program:

.MODEL SMALL
.STACK 100
.DATA
NUM1 DB "4" ; NUMBER 1 (SINGLE DIGIT)
NUM2 DB "9" ; NUMBER 2 (SINGLE DIGIT)
PRODUCT DB 00, 00 ; MEMORY FOR PRODUCT
.CODE
MOV AX, @DATA
MOV DS, AX ; INITIALIZE DATA SEGMENT
MOV DL, NUM1 ; GET NUMBER 1
AND DL, 0FH ; MASK THE HIGHER NIBBLE TO GET ONLY NUMBER
MOV AL, NUM2 ; GET NUMBER 2
AND AL, 0FH
MUL DL ; MULTIPLY TWO NUMBER
AAM ; CONVERT IT IN TO ASCII FORMAT OR
AL, 30H
MOV PRODUCT, AL ; SAVE THE LOWER DIGIT
OR AH, 30H
MOV PRODUCT+1, AH ; SAVE THE HIGHER
MOV AH, 4CH
INT 21H
END ; END PROGRAM

OUTPUT:
BEFORE EXECUTION

| DS:0000 | 0 | E | 4 | C | 2 | 0 | 3 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

AFTER EXECUTION

| DS:0000 | 0 | E | 4 | C | 2 | 0 | 3 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
**Result:**
Program Is Executed Without Errors And The Output Is Verified

**Verification And Validation:**
Output Is Verified And Is Found Correct

**Conclusion:**
The Multiplication Of Two Ascii Data Is Done And The Output Is Verified
EXPERIMENT NO.2.4.A. DEVELOP AND EXECUTE AND ASSEMBLY LANGUAGE PROGRAM TO PERFORM THE CONVERSION FROM BCD TO BINARY

AIM: TO DEVELOP AND EXECUTE AND ASSEMBLY LANGUAGE PROGRAM TO PERFORM THE CONVERSION FROM BCD TO BINARY

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```assembly
.MODEL SMALL
.DATA
    BCD_INPUT DB 61H ; BCD
    NUMBER IN.VALUE DB (?)
.CODE
    MOV AX, @DATA
    MOV DS, AX ; INITIALIZE DATA SEGMENT
    MOV AL, BCD_INPUT
    MOV BL, AL ; MOVE NUMBER TO AL
    AND AL, 0FH
    AND CL, 0FH
    MOV CL, 04H
    ROR AL, CL
    MOV BH, 0AH
    MUL BH
    ADD AL, BL
    MOV IN.VALUE, AL ; STORE THE BINARY EQUIVALENT
    NUMBER IN.VALUE, AH
    INT 21H
    END ; END PROGRAM
```

=================================================================================

OUTPUT:

BEFORE EXECUTION

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>DS:0000</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>DS:0000</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE CONVERSION OF NUMBER FROM BCD TO BINARY IS DONE AND THE OUTPUT IS VERIFIED
EXPERIMENT NO.2.4.B. WRITE AN ALP TO CONVERT BINARY TO BCD

AIM: TO WRITE AN ALP TO CONVERT BINARY TO BCD

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
BIN DB 0FFH ; BINARY INPUT
BCD DB 2 DUP (0) ; STORE BCD VALUE
.CODE
BACK:
NEXT2:
NEXT1:
BIN DB 00H ; BINARY INPUT
BCD DB 2 DUP (0) ; STORE BCD VALUE
MOV AX, @DATA
MOV DS, AX ; INITIALIZE DATA SEGMENT
MOV AL, BIN ; MOVE BINARY NUMBER INTO AL REGISTER
MOV BL, AL ; MOVE NUMBER TO AL REGISTER
MOV CX, 0000H ; CLEAR CX REGISTER
CONTENT CMP AL, CL
JE NEXT1
MOV AL, 00H
INC CL ; INCREMENT CL REGISTER
ADD AL, 01H
DAA ; DECIMAL ADJUST AFTER ADDITION
JNC NEXT2
PUSH AX
MOV AL, 00H
ADD AL, 00H
DAA
ADD CH, AL
POP AX
NEXT2:
CMP BL, CL
JNZ BACK
NEXT1:
MOV BCD, AL ; STORE THE BCD INPUT VALUE
MOV BCD+1, CH
MOV AH, 4CH
INT 21H
END ; END PROGRAM

=================================================================================

OUTPUT:
BEFORE EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>C</td>
<td>2</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>C</td>
<td>2</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

=================================================================================================
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE CONVERSION OF NUMBER FROM BINARY TO BCD IS DONE AND THE OUTPUT IS VERIFIED
EXPERIMENT NO. 2.5.A. WRITE AN ALP TO FIND THE SQUARE OF A NUMBER

AIM: TO WRITE AN ALP TO FIND THE SQUARE OF A NUMBER

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```asm
.MODEL SMALL
.STACK
.DATA
  X DB 08H ; NUMBER TO BE SQUARED
  SQR DW (?) ; LOCATION TO STORE NUMBER

.CODE
  MOV AX, @DATA ; INITIALIZE DATA SEGMENT
  MOV DS, AX
  MOV AL, X
  MUL AL
  MOV SQR, AX ; SQUARE THE NUMBER
  MOV AH, 4CH
  INT 21H
END ; END PROGRAM
```

OUTPUT:

BEFORE EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE SQUARE OF THE GIVEN NUMBER IS FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 2.5.B. WRITE AN ALP TO FIND THE CUBE OF A NUMBER

AIM: TO WRITE AN ALP TO FIND THE CUBE OF A NUMBER

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```assembly
.MODEL SMALL
.DATA
X DB 02H ; NUMBER TO BE SQUARED
CUB DW (?) ; LOCATION TO STORE NUMBER
.CODE
MOV AX, @DATA ; INITIALIZE DATA SEGMENT
MOV DS, AX
MOV AL, X ; STORE THE NUMBER IN AL REGISTER
MUL AL
MOV BL, AL
MOV AL, X
MUL BL
MOV CUB, AX ; SQUARE THE NUMBER
MOV CUB+2, Dx
MOV AH, 4CH
INT 21H
END ; END PROGRAM
```

OUTPUT:

BEFORE EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>F</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>E</td>
<td>4</td>
<td>C</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>F</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>E</td>
<td>4</td>
<td>C</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE CUBE OF THE GIVEN NUMBER IS FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 2.5.C. WRITE AN ALP TO FIND THE LCM OF TWO 16BIT NUMBERS

AIM: TO WRITE AN ALP TO FIND THE LCM OF TWO 16BIT NUMBERS

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
  VALUE DW 0005H, 000FH ; INITIALIZE DATA MEMORY LOCATIONS FOR THE OPERANDS
  LCM DW 2 DUP (?) ; AND THE CALCULATED RESULT
.CODE
  MOV AX, @DATA ; INITIALIZE DATA SEGMENT
  MOV DS, AX
  MOV DX, 0000H ; CLEAR DX REGISTER
  MOV AX, VALUE ; LOAD THE FIRST NUMBER
  MOV BX, VALUE+2 ; LOAD THE SECOND
  AGAIN NUMBER:
  PUSH AX ; SAVE BOTH THE NUMBER ON TOP OF THE STACK
  PUSH DX
  DIV BX ; DIVIDE FIRST NUMBER BY THE SECOND
  CMP DX, 0000H ; IS THERE A NUMBER?
  JE EXIT ; NO, TERMINATE THE PROGRAM
  POP DX ; YES, POP THE DATA STORED
  POP AX
  ADD AX, VALUE ; ADD THE FIRST NUMBER TO THE CONTENTS OF AX
  JNC NOINCDX ; IF THE RESULT IS GREATER THAN 16-BITS INCREMENT
  INC DX ; DX REGISTER
  NOINCDX:
  JMP AGAIN ; REPEAT TILL THE REMAINDER IS ZERO
  EXIT:
  POP LCM+2 ; POP THE LCM VALUE FROM THE TOP OF THE STACK
  POP LCM
  MOV AH, 4CH
  INT 21H
  END ; END PROGRAM

OUTPUT:

BEFORE EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0010</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0020</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0030</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0040</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0010</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0020</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0030</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS:0040</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE LCM OF TWO GIVEN NUMBERS IS FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 2.5.D. WRITE AN ALP TO FIND THE GCD OF TWO 16BIT UNSIGNED NUMBERS

AIM: TO WRITE AN ALP TO FIND THE GCD OF TWO 16BIT UNSIGNED NUMBERS

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```
.MODEL SMALL
.DATA
    NUM1 DW 0005H ; INITIALIZE DATA
    NUM2 DW 000FH
    GCD DW (?) ; INITIALIZE MEMORY FOR THE RESULT
.CODE
    AGAIN:
    MOV AX, @DATA ; INITIALIZE DATA SEGMENT
    MOV DS, AX
    MOV AX, NUM1 ; LOAD THE FIRST NUMBER
    MOV BX, NUM2 ; LOAD THE SECOND NUMBER
    CMP AX, BX ; ARE THEY EQUAL?
    JE EXIT ; YES, SAVE THE GCD
    JB EXCH ; NO, IS AX<BX? ELSE YES, EXCHANGE THE NUMBERS
    BACK:
    MOV DX, 0000H ; CHECK WHETHER AX IS DIVISIBLE BY BX
    CMP DX, 0000H ; IS THERE A NUMBER?
    JE EXIT ; YES, SAVE GCD
    MOV AX, DX ; MOVE THE REMAINDER AS NUM1 DATA
    JMP AGAIN ; REPEAT THE PROCEDURE TILL THERE IS NO REMAINDER
    EXCH
    XCHG AX, BX ; LOAD HIGHER NUMBER IN AX AND
    JMP BACK ; LOWER NUMBER IN DX AND CONTINUE
    : MOV GCD, BX ; SAVE THE GCD NUMBER
    MOV AH, 4CH
    EXIT:
    INT 21H
    END ; END PROGRAM

OUTPUT:
BEFORE EXECUTION

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GCD OF TWO GIVEN NUMBERS IS FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 2.5.E. WRITE AN ALP TO FIND THE FACTORIAL OF A GIVEN NUMBER USING RECURSIVE PROCEDURE

AIM: TO WRITE AN ALP TO FIND THE FACTORIAL OF A GIVEN NUMBER USING RECURSIVE PROCEDURE

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
   NUM DW 8
   RESULT DW (?) ; INITIALIZE MEMORY FOR THE RESULT

.CODE

    ; INITIALIZE MAIN PROCEDURE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, 01
    MOV CX, NUM
    CMP CX, 00
    JE LOOP1
    MOV BX, CX
    CALL FACT
    LOOP1:
    MOV RESULT, AX

    ; INITIALIZE PROGRAM
    MOV AH, 4CH
    INT 21H

    ; END MAIN PROCEDURE
    MAIN ENDP

    ; INITIALIZE FACT PROCEDURE
    CMP BX, 01
    JZ LOOP2
    PUSH BX
    DEC BX
    CALL FACT
    LOOP2:
    MOV AX, 01

    ; END FACT PROCEDURE
    FACT ENDP

    ; END PROGRAM
    END

OUTPUT:
BEFORE EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE FACTORIAL OF A GIVEN NUMBER IS FOUND AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Explain XCHG, LAHF, SAHF, XLAT
2. What are the two types of I/O addressing modes. (fixed port, variable port)
3. What do you mean by segment override prefix.
4. Explain the following directives. NEAR,FAR,BYTE PTR,ORG,OFFSET,ORG
Differentiate END, ENDP, ENDM
EXPERIMENT NO. 3.1. WRITE AN ALP TO SEPARATE ODD AND EVEN NUMBERS

AIM: TO WRITE AN ALP TO SEPARATE ODD AND EVEN NUMBERS

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```asm
.MODEL SMALL
.DATA
   ARR_EVEN DB 10 DUP (?)
   ARR_ODD DB 10 DUP (?)

.CODE
   MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
   MOV DS, AX
   MOV CL, 0AH ; INITIALIZE THE COUNTER
   XOR DI, DI ; INITIALIZE THE ODD POINTER
   XOR SI, SI ; INITIALIZE THE EVEN POINTER
   LEA BP, ARRAY
   MOV AL, DS:[BP] ; GET THE NUMBER
   TEST AL, 01H ; MASK ALL BITS EXCEPT LSB
   JZ NEXT ; IF LSB = 0 GOT TO NEXT

   LEA BX, ARR_ODD
   MOV [BX+DI], AL ; INCREMENT THE ODD POINTER JMP SKIP

   LEA BX, ARR_EVEN

   NEXT
   MOV [BX+SI], AL ; INCREMENT THE EVEN POINTER
   INC BP ; INCREMENT ARRAY BASE POINTER
   LOOP BACK ; DECREMENT THE COUNTER

   SKIP: INT 21H

END ; END PROGRAM
```

PRE VIVA QUESTIONS:

11. Differentiate PROC AND
2. What are the two basic formats used by assemblers. E. 3. Where are they used.
   (Models, full segment definition)
4. Explain ADD BYTE PTR (.model tiny (64kb), .model small(128 kb), .model huge.
5. Explain ADD BYTE PTR [DI], 3, SBB BYTE PTR [DI], 5, CMP[DI], CH IMUL
   BYTE PTR [BX], IDIV SI, CWD, CBW,
   DAA, (ONLY ON AL), AAA, AAD, AAM, AAS.
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE ODD AND EVEN NUMBERS ARE SEPERATED AND OUTPUT IS VERIFIED
EXPERIMENT NO.3.2. WRITE AN ALP TO SEPARATE POSITIVE AND NEGATIVE NUMBERS

AIM: TO WRITE AN ALP TO SEPARATE POSITIVE AND NEGATIVE NUMBERS

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```assembly
.MODEL SMALL
.DATA
NEGI DB 10 DUP (?)
POSI DB 10 DUP (?)

.CODE
BACK:
NEXT:
SKIP:

; NEGI DB 10 DUP (?)
; POSI DB 10 DUP (?)

MOV AX, @DATA
; INITIALIZE THE DATA SEGMENT
MOV DS, AX
MOV CL, 0AH
; INITIALIZE THE COUNTER
XOR DI, DI
; INITIALIZE THE POINTER FOR NEGATIVE NUMBER
XOR SI, SI
; INITIALIZE THE POINTER FOR POSITIVE NUMBER
LEA BP, ARRAY

TEST AL, 80H
; MASK ALL BITS EXCEPT MSB
JZ NEXT
; IF LSB = 0 GOT TO NEXT
LEA BX, NEG
MOV [BX+DI], AL
INC DI
; INCREMENT THE NEGATIVE
JMP SKIP

LEA BX, POSI
MOV [BX+SI], AL
INC SI
; INCREMENT THE POSITIVE POINTER
INC BP
; INCREMENT ARRAY BASE POINTER
LOOP BACK
; DECREMENT THE

.Skip:
COUNTER MOV AH, 4CH
INT 21H
END
; END PROGRAM
```

OUTPUT:

BEFORE EXECUTION

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
DS:0000 1 6 F 8 E 6 9 A 9 7 0 0 0 0 0 0
DS:0010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0020 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0030 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0040 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

AFTER EXECUTION

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
DS:0000 1 6 F 8 E 6 9 A 9 7 E 8 E 9 A 9
DS:0010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0020 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0030 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS:0040 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED
VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE POSITIVE AND NEGATIVE NUMBERS ARE SEPERATED AND OUTPUT IS VERIFIED
EXPERIMENT NO. 3.3. WRITE AN ALP TO FIND LOGICAL ONES AND ZEROS IN A GIVEN DATA

AIM: TO WRITE AN ALP TO FIND LOGICAL ONES AND ZEROS IN A GIVEN DATA

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
    NUM DB 0FAH
    ONES DB 0
    ZEROS DB 0
.CODE
.START:
    MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
    MOV DS, AX
    MOV AL, NUM ; GET BYTE
    MOV CX, 08H ; SET COUNTER

    BACK:
    ROR AL, 1 ; MOVE MSB IN CARRY
    JNC ZERINC ; CHECK BYTE FOR 0 AND 1
    INC ONES ; IF 1, INCREMENT ONE COUNT
    JMP NEXT

    ZERINC:
    INC ZEROS ; IF 0, INCREMENT ZERO COUNTER

    NEXT:
    DEC CX ; REPEAT UNIT CX = 0
    JMP BACK

    MOV AH, 4CH
    INT 21H
.END START

END ; END PROGRAM

=================================================================================

OUTPUT:
BEFORE EXECUTION

|   |   |   |   |   |   |   |   |   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ds:0000| 2 | 0 | F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0010| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0020| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0030| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0040| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

AFTER EXECUTION

|   |   |   |   |   |   |   |   |   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ds:0000| 2 | 0 | F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0010| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0020| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0030| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ds:0040| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

=================================================================================

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE NUMBER OF ONES AND ZEROS IN A GIVEN DATA ARE FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 3.4. WRITE AN ALP TO FIND WHETHER THE GIVEN CODE BELONGS 2 OUT OF 5 CODE OR NOT

AIM: TO WRITE AN ALP TO FIND WHETHER THE GIVEN CODE BELONGS 2 OUT OF 5 CODE OR NOT

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.CODE
.MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
.MOV DS, AX
.MOV AL, N
.MOV BL, AL
.AND AL, 0E0H
.JNZ NOT_CODE
.MOV BL, 00H
.MOV AL, N
.MOV CX, 0005H

BACK: ROR AL, 1

: JNC SKIP
.INC BL

DEC CX

SKIP:
.JNZ BACK
.CMP BL, 02 JNZ NOT_CODE
.MOV DX, OFFSET MSG2
.MOV AH, 09
.INT 21H
.JMP EXIT

NOT_CODE:
.MOV DX, OFFSET MSG3
.MOV AH, 09
.INT 21H

EXIT: MOV AH, 4CH
.INT 21H
.END ; END PROGRAM

==========================================================================

OUTPUT:

;C:\8086> ENTER THE FILE NAME

; YOUR CODE IS 2 OUT OF 5 CODE

==========================================================================
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN NUMBER IS 2 OUT OF 5 CODE AND THE OUTPUT IS VERIFIED
3.5. A  WRITE AN ALP TO CHECK BITWISE PALINDROME OR NOT

AIM: TO WRITE AN ALP TO CHECK BITWISE PALINDROME OR NOT

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.SECTION  100

; PRINTSTRING MACRO MSG
; MOV AH, 09H ; MACRO TO DISPLAY THE MESSAGE
; MOV DX, OFFSET MSG
; INT 21H
; ENDM

.DATA
; NUM DB 0FFH
; TABLE DB 81H, 42H, 24H, 18H
; MSG1 DB 'THE NUMBER EXHIBITS BITWISE PALINDROME:$'
; MSG2 DB 'THE NUMBER DOES NOT EXHIBITS BITWISE PALINDROME:

.CODE

L1: NEXT:

; MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
MOV DS, AX
LEA SI, TABLE
MOV CX, 0004H ; SET COUNTER
XOR AX, AX ; CLEAR AX REGISTER

MOV AL, NUM
AND AL, [SI] ; JPE
JPE NEXT
; PRINTSTRING MSG2 ; DISPLAY MESSAGE 2
PRINTSTRING MSG2
JMP SKIP

INC SI ; INCREMENT POINTER
JMP L1

DEC CX ; DECREMENT
; COUNTER JNZ L1
; PRINTSTRING MSG1 ; DISPLAY MESSAGE 1
PRINTSTRING MSG1

MOV AH, 0CH
INT 21H

.Skip: END ; END PROGRAM

========================================================================

OUTPUT:

;C:\8086> ENTER THE FILE NAME

; THE NUMBER EXHIBITS BITWISE PALINDROME

========================================================================

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN NUMBER EXHIBITS BITWISE PALINDROME
3.5.B WRITE AN ALP TO CHECK WHETHER THE GIVEN NUMBER IS NIBBLEWISE PALINDROME OR NOT

AIM: TO WRITE AN ALP TO CHECK WHETHER THE GIVEN NUMBER IS NIBBLEWISE PALINDROME OR NOT

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
DAT DW 8989H
TEMP DW 0
MSG1 DB 10,13,'THE NUMBER IS NIBBLEWISE PALINDROME:$'
MSG2 DB 10,13,'THE NUMBER IS NOT A NIBBLEWISE PALINDROME:$'
.CODE
START:
    MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
    MOV DS, AX
    MOV DX, DAT ; GET THE WORD
    MOV BX, DX ; MAKE A COPY OF THE WORD
    MOV CH, 02H ; INITIALIZE ROTATION COUNTER
    MOV CL, 04H ; INITIALIZE ITERATION COUNTER
    ROL DX, CL ; INITIALIZE ROTATION COUNTER
    AND DX, 0FH
    MOV AX, BX
    AND BX, 0FH
    CMP BX, DX ; IF NO CARRY SKIP TO THE NEXT INSTRUCTION
    JNZ TER
    MOV BX, AX
    ROR BX, CL ; RESTORE THE CONTENTS OF BX
    MOV DX, TEMP
    ROR BX, TEMP
    CMP BX, DX
    JNZ TER
    MOV AH, 09H
    LEA DX, MSG1
    INT 21H
    JMP LAST
TER:
    MOV AH, 09H
    LEA DX, MSG2 ; SET POINTER TO MESSAGE 2
    INT 21H
LAST:
    MOV AH, 4CH
    INT 21H
    END START
END ; END PROGRAM

=================================================================================
OUTPUT:
;C:\8086> ENTER THE FILE NAME
;THE NUMBER IS NOT A NIBBLEWISE PALINDROME

=================================================================================
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN NUMBER IS NOT A NIBLEWISE PALINDROME AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Name the logical instructions. How can we invert number (XOR WITH 1s)

2. Differentiate TEST and CMP, and NOT& NEG, SAR & SHR, RCL & ROL, SCAS & CMPS, REPE SCASB & REPNE & SCASB

3. Which are the flags affected. JA(Z=0 C=0), JB(C=0), JG (Z=0 S=0), JLE( Z=1 S<>0)
   LOOP, LOOPNE, LOOPE LOOPZ

4. Differentiate NEAR & FAR CALL, NEAR RET & FAR RET
EXPERIMENT NO. 4.1. WRITE AN ALP TO FIND LARGEST NO FROM THE GIVEN ARRAY

AIM: TO WRITE AN ALP TO FIND LARGEST NO FROM THE GIVEN ARRAY

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.STACK 100
.DATA
NUM DB 12H, 37H, 01H, 36H, 76H ; INITIALISE DATA
SMALL DB (?) ; TO STORE LARGEST NUM
.CODE
MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
MOV DS, AX
MOV CL, 05H ; SET COUNTER
MOV AL, 00H
LEA SI, NUM ; POINTER TO NUMBER
LOOP1:
CMP AL, [SI] ; COMPARE 1ST AND 2ND NUMBER
JNC LOOP2
MOV AL, [SI]
INC SI
DEC CL
JNZ LOOP1
MOV SMALL, AL
MOV AH, 4CH
INT 21H
END ; END PROGRAM

PRE VIVA QUESTIONS:

1. Explain, maskable, non maskable, vectored, non vectored, software & Hardware Interrupts.
2. What are interrupt vectors. (4 byte no. stored in the first 1024 bytes of memory. There are 256 interrupt vectors. Each vector contains value of CS & IP, 32 vectors are reserved for present and future. 32 to 255 are available for users.
3. Name the interrupt instructions. (INT, INT0, INT3)
4. Give significance of INT0, INT3.
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE LARGEST NUMBER IN THE GIVEN ARRAY IS 76 AND OUTPUT IS VERIFIED
EXPERIMENT NO. 4.2. WRITE AN ALP TO FIND SMALLEST NO FROM THE GIVEN ARRAY

AIM: TO WRITE AN ALP TO FIND SMALLEST NO FROM THE GIVEN ARRAY

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.CODE
LOOP1
LOOP2
INC SI
DEC SI
JC LOOP1
MOV SMALL, AL
MOV AH, 4CH
INT 21H
END

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE SMALLEST IN THE GIVEN NUMBER IS 01 AND OUTPUT IS VERIFIED
EXPERIMENT NO. 4.3. WRITE AN ALP TO SORT A GIVEN SET OF 16BIT UNSIGNED INTEGERS INTO ASCENDING ORDER USING BUBBLE SORT ALGORITHM

AIM: TO WRITE AN ALP TO SORT A GIVEN SET OF 16BIT UNSIGNED INTEGERS INTO ASCENDING ORDER USING BUBBLE SORT ALGORITHM

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
    A DB 23H, 45H, 55H, 22H, 64H ; INITIALISE DATA
    SIZE DW ($) - A ; CALCULATE SIZE OF NUMBERS
.CODE

    MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
    MOV DS, AX
    MOV BX, SIZE ; THE NO. OF DATA BYTES IS INITIALIZE IN BX

    DEC BX

    OUTLOOP:
        MOV CX, BX ; SAVE COUNTER IN CX REGISTER
        MOV SI, 00 ; INITIALISE POINTER

    INLOOP:
        MOV AL, A[SI] ; LOAD THE DATA INTO AL POINTED BY SI
        INC SI ; INCREMENT THE POINTER
        CMP AL, A[SI] ; IS CONTENT OF AL<SI POINTED
        JB NEXT ; YES, GO NEXT
        XCHG AL, A[SI] ; NO, EXCHANGE TWO DATA
        MOV A[SI - 1], AL ; MOVE TILL END OF MEMORY

    NEXT LOOP:
        INLOOP DEC
        BX
        JNZ OUTLOOP
        MOV AH, 4CH
        INT 21H

    END ; END PROGRAM

OUTPUT:
BEFORE EXECUTION

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DS:0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DS:0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN NUMBERS ARE ARRANGED IN ASCENDING ORDER AND THE OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Give the significance of IRET instruction how is it different from RET.
2. (Like far RET retrieves 6 bytes from stack, two for IP, two for CS and two for flags.)
3. Explain the operation of real mode interrupt.
4. Explain the protected mode interrupt.
5. Explain how the interrupt flag bit IF and TF are used during an interrupt
6. Name the hardware and soft ware interrupt of 8086, explain about them. (NMI, INTR are hardware interrupts. INT, INT0, INT3, BOYND, are the software interrupts)
EXPERIMENT NO.5.1. WRITE AN ALP TO TRANSFER OF A STRING IN FORWARD DIRECTION

AIM: TO WRITE AN ALP TO TRANSFER OF A STRING IN FORWARD DIRECTION

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
SRC DB ">CITY ENGINEERING COLLEGE" DST DB 25 DUP(?)

.CODE
MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
MOV DS, AX
MOV ES, AX
LEA SI, SRC
LEA DI, DST
MOV CX, 19H
CLD ; CLEAR THE DIRECTION FLAG
REP MOVSB ; TRANSFER THE STRING BYTE TILL CX=0
MOV AH, 4CH ; TERMINATE THE PROGRAM
INT 21H
END ; END PROGRAM

PRE VIVA QUESTIONS:

1. How can you expand the interrupt structure. (using 74LS 244 7 more interrupts can accommodated. Daisy chained interrupt is better as it requires only one interrupt vector.)
2. Give a general description of 8259 interrupt controller.
3. Explain the above pins of 8086 TEST, READY, RESET, BHE/S7, MN/MX, ALE, DT/R, DEN, HOLD, ILDA, SO, RO/GT1, LOCK, QS1-QS0.
4. Name the maximum mode pins.
5. Name the minimum mode pins.

OUTPUT:
BEFORE EXECUTION

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

AFTER EXECUTION

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

=================================================================================

OUTPUT:
BEFORE EXECUTION

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

AFTER EXECUTION

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN STRING IS TRANSFERRED IN FORWARD DIRECTION
EXPERIMENT NO. 5.2. WRITE AN ALP TO REVERSE STRING

AIM: TO WRITE AN ALP TO REVERSE STRING

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
    X DB "AKANAK" ; GIVEN STRING
    Z DW (Z-X) ; STRING LENGTH
    Y DB (Z-X) DUP (?),'S' ; REVISED STRING

.CODE
    MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
    MOV DS, AX
    LEA SI, Z-1 ; POINTER TO LAST CHARACTER
    LEA DI, Y ; POINTER TO REVERSE
    CHARACTER MOV CX, Z

    L1: MOV AL, [SI]
        MOV [DI], AL
        DEC SI
        INC DI
        DEC CX
        JNZ L1
    LEA DX, Y ; DISPLAY THE REVERSED STRING ON THE SCREEN
    MOV AH, 4CH
    TERMINATE THE
    DISPLAY THE REVERSED STRING ON THE SCREEN
    MOV AH, 4CH
    TERMINATE THE
    PROGRAM INT 21H
    END

OUTPUT:
BEFORE EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>C</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AFTER EXECUTION

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS:0000</td>
<td>C</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0010</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0030</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DS:0040</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN STRING IS REVERSED AND OUTPUT IS VERIFIED
POST VIVA QUESTIONS:

1. Differentiate between MACRO and PROCEDURE.
2. What are the conditional statements used in a MACRO. (REPEAT, WHILE)
3. What are the different methods of reading the keyboard using DOS function calls.
4. How can we use XLAT instruction for look up tables.
5. What are the two methods of interfacing I/O (memory mapped I/O and I/O mapped I/O)
EXPERIMENT NO. 6.1. WRITE AN ALP TO SEARCH A CHARACTER IN A STRING

AIM: TO WRITE AN ALP TO SEARCH A CHARACTER IN A STRING

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.STACK 100
.DATA
STRING DB "COLLEGE"
CHARACTER DB 'E'
RESULT DB (?)
COUNT EQU 07H

.CODE
MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
MOV DS, AX
MOV CX, COUNT ; INITIALIZE COUNTER
LEA SI, STRING
MOV AL, CHARACTER ; LOAD THE CHARACTER TO BE SEARCHED

BACK: CMP AL, [SI] ; COMPARE EACH CHARACTER OF STRING TO THE CHARACTER:
: JE STROBE1
INC SI
DEC CX
JNZ BACK
JMP STROBE

STROBE1:
MOV AL, 01H
MOV RESULT, AL
JMP LAST

STROBE:
MOV AL, 00H
MOV RESULT, AL

LAST: MOV AH, 4CH ; TERMINATE THE
PROGRAM INT 21H
END ; END PROGRAM

PRE VIVA QUESTIONS:

1. Name the difference between 8086, 8088.
2. Name the difference between 8085 and 8086.
3. Name the types of memory used in microprocessor based system.
4. What is the function of the 8288 controller
5. What are the various signals in a RAM and ROM memories.
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN STRING IS SEARCHED AND FOUND AND OUTPUT IS VERIFIED
EXPERIMENT NO. 6.2. WRITE AN ALP TO GIVEN STRING IS PALINDROME OR NOT

AIM: TO WRITE AN ALP TO GIVEN STRING IS PALINDROME OR NOT

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
    X DB "RACECAR" ; GIVEN STRING
    Z DW (Z-X)  ; LENGTH OF STRING
    Y DB (Z-X) DUP(?) ; STORE REVERSED STRING
    M1 DB "NOT PALINDROME","$"
    M2 DB "PALINDROME","$"

.CODE
    MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT
    MOV DS, AX
    MOV ES, AX
    LEA SI, Z-1 ; POINTER TO LAST CHARACTER IN STRING:
    LEA DI, Y ; POINTER TO REVERSED STRING
    MOV CX, Z ; COUNTER
    LOC1: MOV AL, [SI] ; MOV A FIRST CHARACTER
    MOV [DI], AL
    DEC SI
    INC DI
    DEC CX
    JNZ LOC1
    LEA DX, M1
    JNZ LOC2
    LEA SI, X
    LEA DI, Y
    MOV CX, Z
    CLD ; CLEAR THE DIRECTION FLAG
    REPE CMPSB ; COMPARE THE STRING BYTE
    JE PALIN
    LEA DX, M1

LOC2: MOV AH, 09H
    INT 21H
    MOV AH, 4CH
    INT 21H

PALIN: LEA DX, M2
    JMP LOC2
    END

OUTPUT:

;C:\8086> ENTER THE FILE NAME

;PALINDROME

====================================================================
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE GIVEN STRING IS A PALINDROME AND THE OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Name the following. 8255, 8155, 8259, 8253, 8257, 8251
2. Give the format of control word register.
3. Explain the PPI you know.
4. Explain the modes of 8255.
5. Explain the basic function of 8279.
6. How are the delays obtained in a microprocessor based system.
7. What is an arithmetic coprocessor, What are its functions. (multiply, devide, ad, subtract, square root, calculate partial tangent, partial arctangent and logarithms)
EXPERIMENT NO.7.1. WRITE AN ALP TO READ A CHARACTER FROM KEYBOARD

AIM: TO WRITE AN ALP TO READ A CHARACTER FROM KEYBOARD

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.CODE
.CODE
.MODEL SMALL
.MOVE DS, AX
MOV AX, @DATA ; INITIALIZE THE ADDRESS OF DATA
MOV DS, AX ; SEGMENT IN DS
BACK:
MOV AH, 01H ; LOAD FUNCTION NUMBER
INT 21H ; CALL DOS INTERRUPT
CMP AH, '0' ; DISPLAY THE KEYS UNTIL 0 KEY IS PRESSED
JZ LAST
JMP BACK
LAST:
MOV AH, 4CH ; TERMINATE THE
INT 21H ; PROGRAM INT 21H
END ; END PROGRAM

=================================================================================
OUTPUT:

;C:\TEST>ENTER THE FILE NAME AND TYPE KEYS, PRESS ZERO TO EXIT THE PROGRAM

=================================================================================
7.2. WRITE AN ALP TO READ BUFFERED INPUT FROM THE KEYBOARD USING DOS INTERRUPTS

MODEL SMALL
DATA
MSG DB "KEYBOARD WITH BUFFER: ", ' $' ; MESSAGE FOR THE INPUT
BUFF DB 25
DB 00
DB 25 DUP (?)
CODE
MOV AX, @DATA ; INITIALIZE THE ADDRESS OF DATA
MOV DS, AX ; SEGMENT IN DS
MOV AH, 09H
MOV DX, OFFSET MSG ; FUNCTION TO DISPLAY
INT 21H
MOV AH, 0AH
MOV DX, OFFSET BUFF ; FUNCTION TO TAKE BUFFERED DATA
INT 21H
MOV AH, 4CH ; TERMINATE THE
PROGRAM INT 21H
END ; END PROGRAM

PRE VIVA QUESTIONS:

1. What is the clock frequency of the 8086.
2. How are the address and data buses are separated.

OUTPUT:

; C:\8086> ENTER THE FILE NAME
; KEYBOARD WITH BUFFER: CITY ENGINEERING COLLEGE

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE KEYBOARD FUNCTIONS ARE EXECUTED AND OUTPUT IS VERIFIED
7.3. WRITE AN ALP TO DISPLAY SINGLE CHARACTER

**AIM:** TO WRITE AN ALP TO DISPLAY SINGLE CHARACTER

**SOFTWARE REQUIRED:** MASM 16 BIT

**PROGRAM:**

```assembly
.MODEL SMALL
.CODE
    MOV AH, 02H ; CALL DISPLAY CHARACTER FUNCTION
    MOV DL, 'S' ; MOVE THE CHARACTER TO DL
    INT 21H ; TERMINATE THE
    MOV AH, 4CH ; PROGRAM INT 21H
    END ; END PROGRAM
```

OUTPUT:

;C:\8086> ENTER THE FILE NAME DIRECTLY

7.4. WRITE AN ALP TO DISPLAY STRING ON CONSOLE

**AIM:** TO WRITE AN ALP TO DISPLAY STRING ON CONSOLE

**SOFTWARE REQUIRED:** MASM 16 BIT

**PROGRAM:**

```assembly
.MODEL SMALL
.DATA
    MSG DB 10, 13, "CITY ENGINEERING COLLEGE", 'S'
.CODE
    MOV AX, @DATA ; INITIALISE DS REGISTER
    MOV DS, AX
    LEA DX, MSG ; LOAD EFFECTIVE ADDRESS
    MOV AH, 09H
    INT 21H
    MOV AH, 4CH ; TERMINATE THE
    PROGRAM INT 21H
    END ; END PROGRAM
```

OUTPUT:

;C:\8086> ENTER THE FILE NAME

;CITY ENGINEERING COLLEGE

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED
VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE STRING CHARACTER IS DISPLAYED AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. What do you mean by modular programming, how is it accomplished in 8086.
2. what are libraries.
3. Differentiate between MACRO and PROCEDURE.
4. What are the conditional statements used in a MACRO. (REPEAT, WHILE)
5. What are the different methods of reading the keyboard using DOS function calls.
6. How can we use XLAT instruction for look up tables.
EXPERIMENT NO. 8.1. SCAN 4*4 KEYBOARD FOR KEY CLOSURE AND DISPLAY THE CORRESPONDING KEY CODE

AIM: TO SCAN 4*4 KEYBOARD FOR KEY CLOSURE AND DISPLAY THE CORRESPONDING KEY CODE

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

INITDS MACRO
    MOV AX, @DATA
    MOV DX, AX
ENDM

INIT8255 MACRO
    MOV AL, CW
    MOV DX, CR
    OUT DX, AL
ENDM

INPA MACRO
    MOV DX, PA
    IN AL, DX
ENDM

OUTPC MACRO
    MOV DX, PC
    OUT DX, AL
ENDM

DISPLAY MACRO MSG
    LEA DX, MSG
    MOV AH, 09H
    INT 21H
ENDM

PRINT MACRO NUM
    MOV AL, NUM
    AAM
    MOV BX, AX
    MOV BX, 3030H
    MOV DL, BL
    MOV AH, 02H
    INIT 21H
    MOV DL, BH
    MOV AH, 02H
    INT 21H
ENDM

EXIT MACRO
    MOV AH, 4CH
    INT 21H
ENDM
.MODEL SMALL
.DATA
PA EQU 0D400H ; PORT A : INPUT PORT
PC EQU 0D402H ; PORT C : OUTPUT PORT
CR EQU 0D403H
CW EQU 90H
MSG1 DB 10, 13, 'ROW NO $'
MSG2 DB 10, 13, 'COL NO $'
MSG3 DB 10, 13, 'CODE OF THE KEY PRESSED $'
ROW DB 0
COL DB 0
KEY DB 0

.CODE
INITDS
INIT8255 CALL
SCAN
DISPLAY
MSG1 PRINT
ROW DISPLAY
MSG2 PRINT
COL DISPLAY
MSG3 PRINT
KEY EXIT

SCAN PROC
START:
   MOV BH, 80H
   MOV ROW, 00H
   MOV COL, 00H
   MOV KEY, 00H
   MOV BL, 03H
NXTROW:
   ROL BH, 01H
   MOV AL, BH
   OUT PC
   MOV CX, 08H
   IN PA
NXTCOL:
   ROR AL, 01H JC
   QUIT
   INC KEY
   INC COL
   LOOP NXTCOL
   INC ROW
   MOV COL, 00H
   DEC BL
   JMP
   NXTROW
   JMP START
QUIT:  RET
SCAN ENDP
END

PRE VIVA QUESTIONS:

1. How does IN and OUT instruction work?
2. What do you mean by control word of 8255 and how do you calculate?
3. What is the port size supported by 8255?
4. How many ports we can be accessed on interfacing 8255?

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE PROGRAM IS EXECUTED AND KEYBOARD IS SCANNED AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Explain different modes of operation of 8255.
2. How do you switch between ports while programming?
3. In 8x3 keyboard interface, which port points to X axis and which one to Y axis?
4. How is each key numbered in 8x3 Keyboard interface?
5. How to find the position of a bit in a byte data?
6. How to perform arithmetic operation using 8x3 keyboard interface?
EXPERIMENT NO. 8.2. PROGRAM FOR SEVEN SEGMENT LED DISPLAY THROUGH 8255 (PCI BASED)

AIM: TO WRITE A PROGRAM FOR SEVEN SEGMENT LED DISPLAY THROUGH 8255 (PCI BASED)

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.CODE
MOV AX, @DATA
MOV DS, AX
MOV AL, 80H
MOV DX, CR
OUT DX, AL
MOV CX, 02H
MOV DI, 50
LEA SI, FIRE
CALL DISPLAY
DEC DI
JNZ DISP1
MOV DI, 50
LEA SI, HELP
CALL DISPLAY
DEC DI
JNZ DISP2
LOOP AGAIN
MOV AH, 4CH
INT 21H

DISPLAY PROC
MOV AH, 0
BACK:
MOVAL, AH
: MOV DX, PORTC OUT
DX, AL LODSB
MOV DX, PORTA OUT DX,
AL CALL
DELAY INC AH
CMP AH, 6
JNZ BACK
RET
DISPLAY ENDP

DELAY PROC
PUSH BX
PUSH CX
MOV BX, 0FFH
LOOP2: LOOP1:
MOV CX, LOOP1 DEC
          0FFH  BX
    JNZ
L    LOOP2
O    POP CX
O    POP BX
P    RET
    DELAY ENDP
END

PRE VIVA QUESTIONS:

1. What is the control work for the 7 segment display?
2. How do you calculate the 7 segment code?
3. How do you identify each 7 segment module in the interface kit?
4. What is the relevance of delay between each character display?
5. How does XLAT instruction work?

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE 7 SEGMENT DISPLAY IS PROGRAMMED SUCCESSFULLY AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:

1. Value stored in Port C is pointing to what?
2. Value sent through Port A is displayed in which 7 segment?
3. Explain the programming logic of content flashing alternatively.
4. Explain the programming logic of content in rolling fashion.
5. Explain the programming logic of content in bi-directional rolling fashion.
6. Explain the logic of converting a hexadecimal value to decimal equivalent.
EXPERIMENT NO. 8.3.A. READS STATUS OF 8 INPUT FROM THE LOGIC CONTROLLER INTERFACE AND DISPLAY COMPLEMENT OF INPUT ON THE SAME INTERFACE;
"AND GATE OUTPUT"

AIM: TO READS STATUS OF 8 INPUTS FROM THE LOGIC CONTROLLER INTERFACE AND DISPLAY COMPLEMENT OF INPUT ON THE SAME INTERFACE

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```
.MODEL SMALL
.DATA
    CR EQU 0D403H
    PA EQU 0D400H ; PORT A : OUTPUT PORT
    PB EQU 0D401H
    PC EQU 0D402H ; PORT C : INPUT PORT

.CODE
    MOV AX, @DATA
    MOV DS, AX

    MOV AL, 8AH
    MOV DX, CR
    OUT DX, AL

    MOV DX, PB
    IN AL, DX
    MOV BL, AL

    MOV DX, PC
    IN AL, DX

    AND AL, BL
    MOV DX, PA
    OUT DX, AL

    MOV AH, 4CH
    INT 21H

DELAY PROC NEAR
    PUSH CX
    PUSH BX
    MOV BX, 01000H

B2:
    MOVCX, 01000H

B1:
    LOOP B1
    DEC BX
    JNZ B2
    POP BX
    POP CX
    RET

DELAY ENDP
END
```

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE LOGIC CONTROLLER IS PROGRAMMED SUCCESSFULLY AND OUTPUT IS VERIFIED
EXPERIMENT NO.8.3.B. READS STATUS OF 8 INPUT FROM THE LOGIC CONTROLLER INTERFACE AND DISPLAY COMPLEMENT OF INPUT ON THE SAME INTERFACE

AIM: TO READS STATUS OF 8 INPUTS FROM THE LOGIC CONTROLLER INTERFACE AND DISPLAY COMPLEMENT OF INPUT ON THE SAME INTERFACE

"RING COUNTER"

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

.MODEL SMALL
.DATA
CR EQU 0D403H
PA EQU 0D400H ; PORT A : OUTPUT PORT
PB EQU 0D401H
PC EQU 0D402H
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AL, 80H
MOV DX, CR
OUT DX, AL

MOV AL, 01H
MOV CX, 0AH
MOV DX, PA
OUT DX, AL
CALL DELAY
ROR AL, 01
LOOP BACK

MOV AH, 4CH
INT 21H

DELAY PROC NEAR
PUSH CX
PUSH BX
MOV BX, 0FFFFH
B2:
MOV CX, 0FFFFH
B1:
LOOP B1
DEC BX
JNZ B1
POP BX
POP CX
RET
DELAY ENDP
END
PRE VIVA QUESTIONS:
1. Value stored in Port C is pointing to what?
2. Value sent through Port A is displayed in which 7 segment?
3. Explain the programming logic of content flashing alternatively.
4. Explain the programming logic of content in rolling fashion.
5. Explain the programming logic of content in bi-directional rolling fashion.
6. Explain the logic of converting a hexadecimal value to decimal equivalent.

RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE LOGIC CONTROLLER IS PROGRAMMED SUCCESSFULLY AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:
1. Explain the logic of programming logic controller.
2. Can I make B port as output port and display information?
3. What is the relevance of delay in the program?
4. Explain the programming logic of Johnson's counter
EXPERIMENT NO. 8.4. PROGRAM TO ROTATE THE STEPPER MOTOR IN CLOCK-WISE DIRECTION (8 STEPS)

AIM: TO PROGRAM TO ROTATE THE STEPPER MOTOR IN CLOCK-WISE DIRECTION (8 STEPS)

SOFTWARE REQUIRED: MASM 16 BIT

PROGRAM:

```
MODEL SMALL
.DATA
    CR EQU 0E803H
    PA EQU 0E800H
    PB EQU 0E801H
    PC EQU 0E802H
    ; PORT C : OUTPUT PORT

.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, 80H
    MOV DX, CR
    OUT DX, AL

    MOV AL, 88H
    MOV CX, 200
    MOV DX, PC
    OUT DX, AL
    CALL DELAY
    ROR AL, 1
    LOOP BACK

    MOV AH, 4CH
    INT 21H

DELAY PROC NEAR
    PUSH CX
    PUSH BX
    MOV BX, 01FFFH
    B2:
    MOV CX, 1FFFH
    B1:
    LOOP B1
    DEC BX
    JNZ B2
    POP BX
    POP CX
    RET

DELAY ENDP
END
```

PRE VIVA QUESTIONS:

1. Explain the internals of a stepper motor.

2. Explain the programming logic of a stepper motor.

3. How do you initiate a clock-wise rotation in stepper motor? What is logic in sending the value to port?

4. How do you initiate anti clock-wise rotation?
RESULT: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

VERIFICATION AND VALIDATION: OUTPUT IS VERIFIED AND IS FOUND CORRECT

CONCLUSION: THE STEPPER MOTOR IS PROGRAMMED SUCCESSFULLY AND OUTPUT IS VERIFIED

POST VIVA QUESTIONS:
1. How do you initiate anti clock-wise rotation?
2. What is relevance of delay in stepper motor?
3. Mention few application of stepper motor.
4. How many ports we can be accessed on interfacing 8255?
5. Explain different modes of operation of 8255.
6. How do you switch between ports while programming?
KEYBOARD CUM CALCULATOR INTERFACE CARD

26pin FRC male connector

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td></td>
</tr>
<tr>
<td>PA3</td>
<td></td>
</tr>
<tr>
<td>PA4</td>
<td></td>
</tr>
<tr>
<td>PA5</td>
<td></td>
</tr>
<tr>
<td>PA6</td>
<td></td>
</tr>
<tr>
<td>PA7</td>
<td></td>
</tr>
<tr>
<td>PC0</td>
<td></td>
</tr>
<tr>
<td>PC1</td>
<td></td>
</tr>
<tr>
<td>PC2</td>
<td></td>
</tr>
<tr>
<td>PC3</td>
<td></td>
</tr>
<tr>
<td>PC4</td>
<td></td>
</tr>
<tr>
<td>PC5</td>
<td></td>
</tr>
<tr>
<td>PC6</td>
<td></td>
</tr>
<tr>
<td>PC7</td>
<td></td>
</tr>
</tbody>
</table>

7406N
U1A
R5
LED1

U1B
180Ω
R6
LED2

U1C
180Ω
R6
LED3

U1D
180Ω
R9
LED4

U1E
180Ω
R9
LED5

U1F
180Ω
R10
LED6

U2A
180Ω
R11
LED7

U2B
180Ω
R12
LED8

PC0-PC3
Return lines
PC4-PC7
Scan lines

8 LED OUTPUTS:
LEDs driven from PB0-PB7 through 7406 open collector inverters

16 KEYS KEYBOARD:
Scan lines: PC0-PC3
Return lines: PC4-PC7
8 Digit Display Board

Digit 1 - 8 Common Anode Display

26pin FRC male connector

PA0  1  PA1
PA2  3  PA3
PA4  5  PA5
PA6  7  PA7
PC0  9  PC1
PC2 11  PC3
PC4 13  PC5
PC6 15  PC7
PB0 17  PB1
PB2 19  PB3
PB4 21  PB5
PB6 23  PB7

GND  VCC

B  2  ~
  1  Q9

R19  1

U1  1kΩ

1kΩ

B  3  ~
  1  Q11

R21  1

1kΩ

PN2907

A  1

R23  1

1kΩ

PN2907

A  1

R24  1

1kΩ

BC547BP

1kΩ

Q1

1kΩ

Q4

1kΩ

Q5

1kΩ

Q6

1kΩ

Q7

1kΩ

Q8

1kΩ

Q9

1kΩ

Q10

1kΩ

Q11

1kΩ

Q13

1kΩ

Q14

1kΩ

Q15

1kΩ

Q16

1kΩ