# LAB # 8

# To Explain and Show the Output of BIOS Interrupt Programming using EMU8086 Software Tool

## Objectives

* To explain the working of BIOS interrupt programming and EXE Program Structure using assembly language program.
* To show the String and Character on output screen using BIOS Interrupt programming.

## Pre-Lab Exercise

Read the details given below in order to comprehend the basic operations of BIOS programming and different routines. Study in detail and become familiar with the various ways and in which these instructions and routines can be used.

### Memory Models

Machine language programs consist of code, data and stack. Each part occupies a memory segment. The same organization is reflected in an assembly language program. This time the code data and stack are structured as program segments. Each program segment is translated into a memory segment by the assembler.

The size of code and data a program can have been determined by specifying a memory model using the. MODEL directive. The syntax is

**. MODEL** memory model the most frequently used memory models are.

Model Description

**SMALL** **CODE** in one segment **DATA** in one segment

**MEDIUM** **CODE** in more than one segment **DATA** in one segment

C**OMPACT** **CODE** in one segment **DATA** in more than one segment

**LARGE** **CODE** and DATA in more than one segment no array larger than 64K bytes

**HUGE** **CODE** in more than one segment **DATA** in more than one segment

arrays may be larger than 64K bytes Memory model must be specified before any segment definition.

Program structure for .EXE files

The general form of a SMALL model program is given below. Unless there is a lot of code or data, the appropriate model is SMALL. When this program is assembled, assembler will generate executable file with extension .EXE.

**. MODEL SMALL**

**. STACK 100h**

**.DATA**

; Variables and constants are defined here

**. CODE**

### MAIN PROC

; Instructions will be written here MAIN ENDP

; Procedures other than MAIN will be written here END MAIN

The END directive in last line of the program specifies the entry point of code. The code will start its execution from start of MAIN procedure. The name after END directive specifies the name of procedure that must be executed first.

### DOS I/O service routines

CPU communicates with peripherals through I/O registers called I/O ports. There are two instructions, **IN** and **OUT**, that access the ports directly. These instructions are used when a program needs to directly communicate with peripherals like when fast I/O is needed. However most, most applications programs do not use **IN** and **OUT** because it’s much easier to program I/O with the service routines provided by manufacturer.

There are two categories of I/O service routines.

* BIOS routines and
* DOS routines

BIOS routines are stored in ROM and communicate directly with the I/O ports.

The DOS routines use BIOS routines to execute direct I/O operation. They can carry out the more complex tasks; for example, printing a character string, getting input from keyboard etc.

### INT instruction

To invoke DOS or BIOS routine, the INT (interrupt) instruction is used. It has the format

**INT interrupt number**

Where interrupt number is a number that specifies a routine. For example, INT 16h invokes a BIOS routine that performs keyboard input. In this Lab you will use a particular DOS routine, INT 21h. INT 21h may be used to invoke a large number of DOS functions. A particular function is requested by placing function number in the AH register and invoking INT 21h. Some of the commonly used interrupt numbers are given below Table 5-1: Commonly used interrupt numbers

|  |  |
| --- | --- |
| **AH**  **value (Hex)** | **Function** |
| **1** | Character input (with echo) |
| **2** | Character output |
| **7** | Character input (without echo) |
| **9** | String Output |
| **A** | String Input |
| **2A** | Get System Date |
| **2C** | Get System Time |
| **4C** | return control to the operating system (stop program). |

**Read a character input from keyboard**

To get input from keyboard execute the following instructions

MOV AH, 1 ; input key function

INT 21h ; get ASCII code in AL

When a character key is pressed, AL will contain its ASCII code. If any other key is pressed, such as an arrow key or F1-F10 and so on, AL will contain 0.

### Display a character on screen

To display a character 'A' on screen execute the following instructions, actually DL must contain the ASCII code of character that is to be displayed. In this example DL will contain ASCII code of 'A'.

MOV AH, 2 ; display character function

MOV DL,'A' ; character is ‘A’

INT 21h ; display character

If DL contains ASCII code of any of the control characters, INT 21h instruction causes control function to be executed. ASCII codes for principal control characters are as follows.

The above code will print 'A' on the screen and cursor will move to next position on the same line. To move cursor to new line we need to execute two control functions i.e. carriage return (CR) and line feed (LF).

### Display a string on screen

To display a string execute the following instructions, DX register must contain the starting offset address of string that is to be displayed

LEA DX, MSG ; get offset address of MSG

MOV AH, 9 ; string display function

INT 21h ; display MSG

Where MSG is string defined in data segment as

MSG DB "Hello World!",'$'

The last character of string must be '$', in order to indicate the end of string.

### Read a string input

Following lines of code will input a string from keyboard and save it to a buffer whose offset address must exist in register DX and first location of buffer must contain size of buffer. This function does not allow entering more characters than the specified buffer size.

LEA DX, buffer ; get offset address of buffer

MOV AH, 0A ; input string function

INT 21h ; input string

When above code executes, first byte of buffer will contain size of buffer as specified in buffer declaration, 2nd byte will contain actual number of characters read.

' $' must be appended at end of buffer to print the contents of buffer as string.

Buffer of size 10 can be defined as:

buffer DB 10, 10 dup (' ')

## In-Lab Exercise

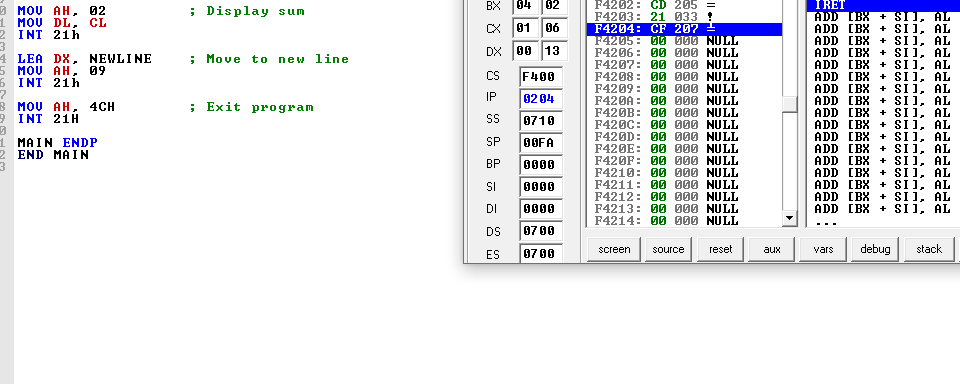
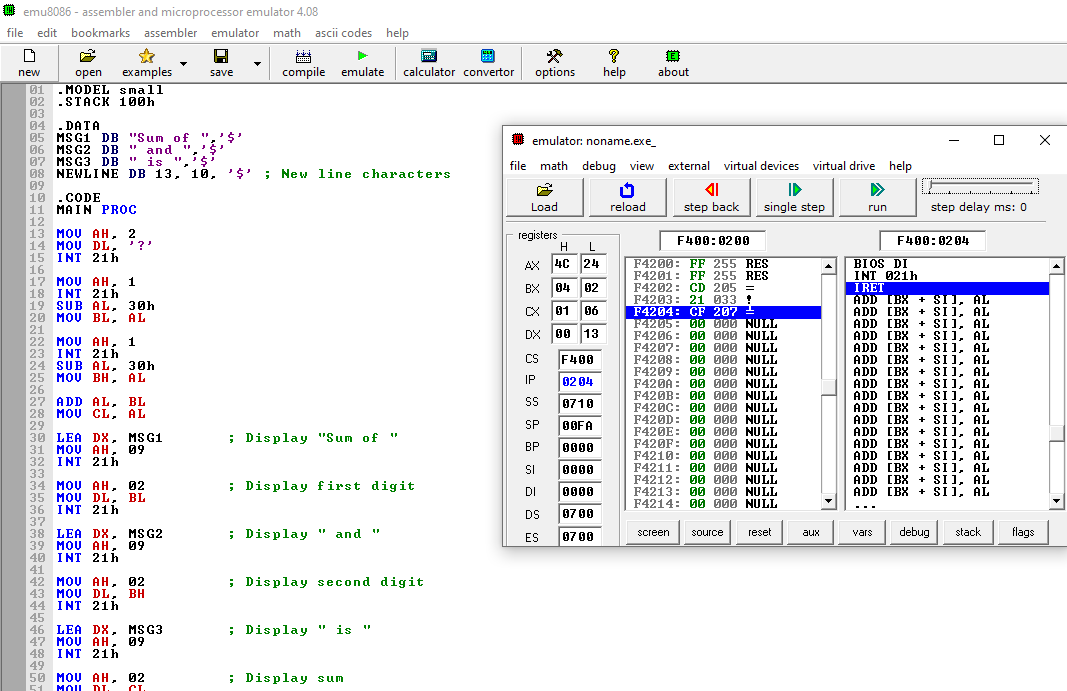
**Task 1:** Write an assembly language program to Display “?” Read two decimal digits between 0 to 4

Display the result as per the following format

?24

The sum of 2 and 4 is 6

Hint: INT 21h always read input values as ASCII characters

****

**Task 2:** Write an assembly language program to convert the case of a string

Write a complete assembly program that converts a user-input lower-case string into corresponding upper-case string.

Sample execution:

Input: hello world

Output: HELLO WORLD

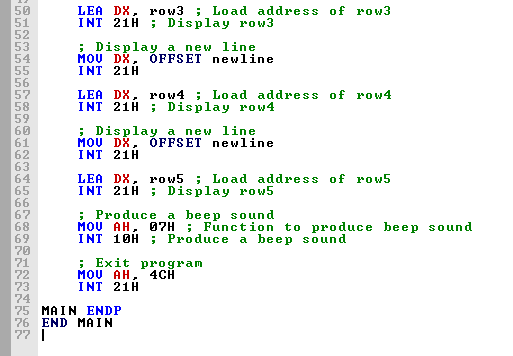
A screenshot of a computer

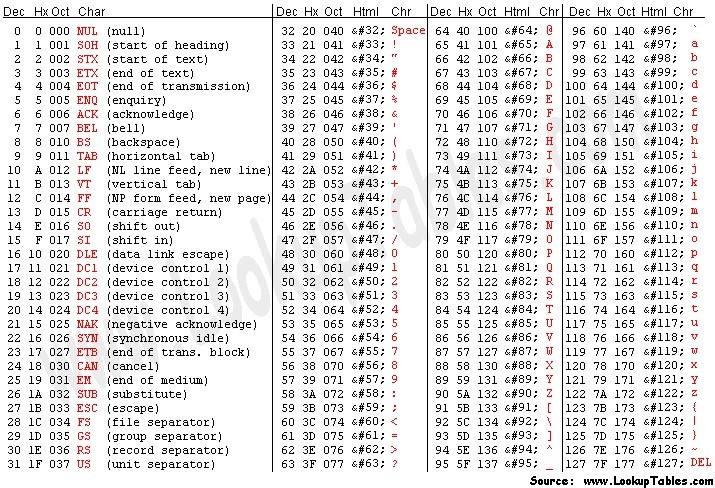
Description automatically generatedA screenshot of a computer

Description automatically generated

**Task 3:** An Assembly Language Program to display “?” read three initials display them in the middle of an 11\*11 box of asterisks and beep the computer

A screenshot of a computer

Description automatically generated

Table 5-2: ASCII Table

## Rubric for Lab Assessment

|  |  |  |  |
| --- | --- | --- | --- |
| The student performance for the assigned task during the lab session was: | | | |
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 |  |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 |  |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 |  |
| Worst | The student did not complete assigned tasks. | 1 |  |

Instructor Signature: Date: