AM1: Real- Mode Programming
AM2: Protected- Mode Programming

*Note: On-the-spot evaluation may be carried out during or at the end of the experiment. Students are advised to read through this lab sheet before doing experiment. Your performance, teamwork effort, and learning attitude will count towards the marks.
AM1: Real-Mode Programming

Time: 3 hours

Components: Lab Exercises and Assessment

Objective:
- To identify the x86 assembly instructions for arithmetic operations, program flow control and string operations,
- To examine the basic assembly instructions in real-mode programming of x86 processors,
- To analyze real-mode assembly programs using EMU8086 emulator.
- To develop real-mode assembly programs for arithmetic operations.

1. Introduction

The Intel 80386, was a 32-bit microprocessor introduced by Intel in 1985. As the original implementation of the 32-bit extensions to the 8086 architecture, the 80386 instruction set, programming model, and binary encodings are still the common denominator for all 32-bit x86 processors. This is termed x86, IA-32, or the i386-architecture. The 80386 could correctly execute most code intended for earlier 16-bit x86 processors such as the 8088 and 80286 that were ubiquitous in early PCs. Following the same tradition, modern 64-bit x86 processors are able to run most programs written for older processors, all the way back to the 16-bit 8086.

The 80386 featured three operating modes: real mode, protected mode and virtual mode. The protected mode which debuted in the 286 was extended to allow the 386 to address up to 4 GB of memory. The all new virtual 8086 mode (or VM86) made it possible to run one or more real mode programs in a protected environment, although some programs were not compatible.

In this experiment, we will study the real-mode operation of the x86 processor via a software emulator known as EMU8086. Real mode is characterized by a 20-bit segmented memory address space (giving just over 1 MB of addressable memory) and unlimited direct software access to all memory and I/O addresses and peripheral hardware. Nevertheless, real mode provides no support for memory protection, multitasking, or code privilege levels.

This experiment is divided into four parts. The first part introduces the arithmetic instructions available under the real-mode operation. The second part focuses on the control flow instructions. Next, the string processing instructions are examined in the third part. The last part involves an assessment to evaluate students’ understanding of the instructions that have been discussed in this lab session. Throughout the experiment, students should refer to the documentation and tutorials available via EMU8086 to learn about the x86 assembly instructions as well as the emulator itself.
2. Experiment

Experiment 1.1
In this experiment, we will identify the x86 arithmetic instructions and examine a program which will perform the following arithmetic operations:
\[(27_{10} + 15_{10}) \times (625_{10} - 250_{10}) + 191_{10}\]

Procedure:
1. Launch the EMU8086 emulator. Choose “New” and specify “COM template”.
2. Using the assembler editor, enter the following codes:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ORG 100H</td>
</tr>
<tr>
<td>2</td>
<td>MOV AX, 27 ; Move 27_{10} to AX</td>
</tr>
<tr>
<td>3</td>
<td>ADD AX, 15 ; Add 15_{10} to AX, store result in AX</td>
</tr>
<tr>
<td>4</td>
<td>MOV BX, 625 ; Move 625_{10} to BX</td>
</tr>
<tr>
<td>5</td>
<td>SUB BX, 250 ; Subtract 250_{10} from BX, store result in BX</td>
</tr>
<tr>
<td>6</td>
<td>MUL BX ; Multiply BX to AX, store result in AX (and DX)</td>
</tr>
<tr>
<td>7</td>
<td>ADD AX, 191 ; Add 191_{10} to AX, store result in AX</td>
</tr>
<tr>
<td>8</td>
<td>RET</td>
</tr>
</tbody>
</table>

4. Single-step the program codes by pressing the “single step” button on the toolbar of the emulator window.
5. Each time after pressing the “single step” button, check and record down the contents of AX and BX registers in Table 1.1.

Experiment 1.2
In this example, we will develop a program to perform the following arithmetic operations:
\[-7_{10} \times (64_{10} - 49_{10}) + 53_{10}\]

Procedure:
1. Develop an assembly language program to perform the above arithmetic operations by only using registers AX and BX (write the program in Table 1.2).
2. Enter the program into the assembler editor in EMU8086
3. Emulate the program by pressing the “emulate” button on the toolbar.
4. In the emulator window, single-step the program codes by pressing the “single step” button on the toolbar.
5. Each time after single-stepping, observe and record down the contents of AX and BX registers in Table 1.2.
**Experiment 2.1**
In this experiment, we will program the trainer to continuously write 5AH to the memory locations starting from DS:1000H to DS:100FH. In this case, the same instructions will be repeated and the CMP and JNE instructions are used to check whether the stopping criterion is encountered (i.e. until 5AH has been written into memory location at DS:100FH).

**Procedure:**
1. Enter the following program codes into the assembler editor of EMU8086.

```
ORG 100H
START: MOV AL, 5AH ; data to be written
      MOV DI, 1000H ; starting from address DS:1000H
RPT1: MOV [DI], AL ; write data to DS:DI
      INC DI ; increment DI to next address
      CMP DI, 1010H ; is it outside the address range?
      JNE RPT1 ; if not, jump to instruction at RPT1
      RET ; else program ends
```
2. Emulate the program by pressing the “emulate” button on the toolbar.
3. In the emulator window, select “view” → “memory” on the pull down menu. A “Random Access Memory” window will pop out.
4. Type DS:1000 in the address field, click the “update” button and observe the memory contents shown in the “Random Access Memory” window.
5. Press the “run” button to execute the program and observe the memory contents shown on the “Random Access Memory”.
6. Once the program has been executed, a dialog box will pop out, stating that “PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM”.
7. Close the dialog box by pressing the “OK” button.
8. Answer the questions for Experiment 2.1 in the Answer Sheet.

**Experiment 2.2**
In this experiment, we will write a program to store a series of 10 ascending data at memory location starting from DS:2000H. In this case, \( 1_{10} \) will be written into memory location at DS:2000H, \( 2_{10} \) will be written into DS:2001H, \( 3_{10} \) will be written into DS:2002H, so on and so forth. In this case, the LOOP instruction is used to monitor the stopping criterion (i.e. 10 data have been written).

**Procedure:**
1. Enter the following program codes into the assembler editor of EMU8086.

```
ORG 100H
START: MOV AL, 1 ; first data to be written
      MOV DI, 2000H ; starting from address DS:2000H
      MOV CX, 10 ; total of 10 data to be written
RPT1: MOV [DI], AL ; write data to address DS:DI
      INC AL ; increment the data to be written next
      INC DI ; increment DI to point to next address
      LOOP RPT1 ; decrease CX, jump to RPT2 if CX=0
      RET
```
2. Emulate the program by pressing the “emulate” button on the toolbar.
3. In the emulator window, invoke the “Random Access Memory” window by following Step 3 in Experiment 2.1.
4. Type DS:2000 in the address field and click the “update” button.
5. Press the “run” button in the emulator window to execute the program and observe the memory contents shown in the “Random Access Memory” window.
6. Answer the questions for Experiment 2.2 in the Answer Sheet.

Experiment 3

The x86 microprocessors are equipped with special instructions to handle string operations. In this experiment, we will explore the use of string processing instruction in real-mode operation. The program below will prompt the user to enter a 4-digit passcode. The user passcode is then compared with the preset passcode. If the passcodes are not the same, the program will ask the user for another passcode. This will be repeated until the user has entered the matching passcode. In this case, the compare string instruction (CMPSB) is used to compare the elements of both passcodes (strings) in order to determine whether they are the same or different.

Procedure:
1. Enter the following program codes into the assembler editor of EMU8086.

```
INCLUDE EMU8086.INC
GETC MACRO P1
    MOV AH, 7
    INT 21H
    MOV [P1], AL
ENDM

ORG 100H
START:

PRINT 'ENTER PASSCODE: '
MOV CX, 4 ; 4 characters
LEA DI, USERCODE ; load address (offset) of USERCODE to DI
RPT:
    GETC DI ; get a character & store at [DI]
    INC DI ; increment DI to next address
    PUTC '*' ; print an ‘*’
    LOOP RPT ; decrement CX & jump to RPT if CX ≠ 0
    PUTC 0AH ; print a new line character
    PUTC 0DH ; print a carriage return character

MOV CX, 4 ; compare 4 characters
MOV AX, DS ; AX=DS
MOV ES, AX ; ES=AX=DS
LEA SI, PASSCODE ; load address (offset) of PASSCODE to SI
LEA DI, USERCODE ; load address (offset) of USERCODE to DI
REPE CMPSB ; repeat until CX=0 or USERCODE=PASSCODE
JE EXIT ; if USERCODE=PASSCODE, jump to EXIT
PRINTN 'WRONG PASSCODE!' ; else print error message
JMP START ; jump to START

EXIT:   PRINTN 'CORRECT PASSCODE.'
RET

PASSCODE DB '1234' ; preset passcode
USERCODE DB 4 DUP (?) ; user passcode (reserve 4 bytes)
```

2. Emulate the program by pressing the “emulate” button on the toolbar.
3. In the emulator window, select “view” → “variables” on the pull down menu. A variables window will pop out.
4. Two variables (PASSCODE and USERCODE) will be shown in the variables window.
5. For each of these variables, change the “elements” to 4 and “show as” to ascii.
6. Press the “run” button in the emulator window to execute the program.
7. An emulator screen window will pop out.
8. Enter a passcode and observe the USERCODE variable shown in the variables window. (Hints: enter a wrong passcode first before entering the correct passcode)
9. Answer the questions for Experiment 3 in the Answer Sheet.

3. References

3. EMU8086 Documentation and Tutorials.

Important Note:
Lab Assessment will be carried out during the lab session. It is the student's responsibility to complete the given tasks and report to the lab instructors for assessment.

(End)
Marking Scheme for AM1

**Experiment 1.1**
1. Complete the following table:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>AX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Using a calculator, calculate the answer for the above arithmetic operations: .......... Is it the same as the final answer in the AX register? ......................................................... [8 marks]

**Experiment 1.2**
1. Complete the following table:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>AX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Using a calculator, calculate the answer for the above arithmetic operations: .......... Is it the same as the final answer in the AX register? .........................................................

3. If Line 5 of the program codes is changed from “IMUL BL” to “MUL BL”, what is the final answer in the AX register? ................................................................. Is it correct? Why? ........................................................................................................ [12 marks]

**Experiment 2.1** [9 marks]

**Experiment 2.2** [11 marks]

**Experiment 3** [10 marks]