Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

Laboratory Report

on

EEE 3254: Sessional Based on EEE 3253

Submitted by

Al Nahian Mugdho

Roll No. 1804021

Submitted to

A. S. M. Badrudduza

Assistant Professor

Department of ETE

December 16, 2022

Contents

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 1

Introduction to IBM PC Assembly Language.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 08/10/2022

Date of Submission : 15/10/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 1

Introduction to IBM PC Assembly Language.

1.1 Objectives

The main objectives of this experiment are

- To understand assembly language syntax
- To understand Program Data
- To understand Variables.

1.2 Introduction

Every personal computer has a microprocessor that manages the machine's logical, mathematical, and control processes. For managing many functions, such as accepting keyboard input, displaying information on the screen, and performing numerous other duties, each CPU family has its own distinct set of instructions. The term "machine language instructions" is used to describe them. Processors can only understand machine language instructions, which are strings of ones and zeros. But the complexity and difficulty of machine language make it unsuitable for software development. This results in the creation of the low-level assembly language for a specific family of processors, which conveys various instructions in symbolic code and in a more understandable manner.

1.3 Required Software

1. EMU8086

1.4 Assembly Language Syntax

Assembly language programs are translated into machine language instructions by an assembler, so they must be written to conform to the assembler's specifications. Assembly language code is generally not case-sensitive. Statements make up programs, one per line.

Each statement is either an instruction that the assembler converts into machine code or an assembler directive that tells the assembler to carry out a particular task, like allocating memory for a variable or establishing a procedure. Both directives and instructions may contain up to four fields: name operation operand(a) comment

1.4.1 Name Field

The name field is used for instruction labels, procedure names, and variable names. The assembler translates names into memory addresses. Legal Names :

COUNTER1

@character

Illegal Names:

Two Words

2abc

1.4.2 Operation Field

For an instruction, the operation field contains a symbolic operation code (opcode). The assembler translates a symbolic opcode into a machine language opcode. Opcode symbols often describe the operation's function; for example, MOV, ADD, SUB.

1.4.3 Operand Field

For an instruction, the operand field specifies the data that is to be acted on by the operation. An instruction may have zero, one, or two operands. For example-

INC AX

1.4.4 Comment Field

The comment field of a statement is used by the programmer to say something about what the statement does. A semicolon marks the beginning of this field, and the assembler ignores anything typed after the semicolon. Example:

MOV CX,O ; move 0 to CX

1.5 Program Data

The processor operates only on binary data. Thus, the assembler must translate all data representations into binary numbers. However, in an assembly language program, one may express data as binary, decimal, or hex numbers, and even as characters. A binary number is represented by a bit string with either the letter "R" or "b" after it; an example would be 10108. A string of decimal digits that ends with an optional "D" or "d" is known as a decimal number. Because the assembler would be unable to determine if a symbol like "ABCH" indicates the variable name "ABCH" or the hex number ABC, a hex number must start with a decimal digit and conclude with the letter "H" or "h," for example, OABCH.

1.6 Variables

In assembly language, variables serve the same purpose as they do in high-level languages. Each variable has a data type, and the software allows it with a memory address.
One uses DB and DW to define byte variables, word variables, and arrays of bytes and words.

1.6.1 Byte Variables

The assembler directive that defines a byte variable takes the following form:
name DB; initial value
where the pseudo-DB stands for "Define Byte".

1.6.2 Word Variables

The assembler directive for defining a word variable has the following form:
name DW; initial value
where the pseudo-DW stands for "Define Word".

1.6.3 Arrays

In assembly language, an array is just a sequence of memory bytes or words. for example, to define a three-byte array called B_ARRAY, whose initial values are lOh, 20h, and 30h, we can write,

B_ARRAY DB 10H,20H,30H

1.7 Conclusions and Discussions

The experiment was to introduce assembly language. After finishing this experiment the assembly language syntax, variables and program data was learned.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 2

Experimental Study on the Processor Status and the FLAGS Register.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 08/10/2022

Date of Submission : 15/10/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	______________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 2

Experimental Study on the Processor Status and the FLAGS Register

2.1 Objectives

The main objectives of this experiment are

- To understand the Flags Register
- To understand how the instructions affect the flags.

2.2 Introduction

The microprocessor is the central unit of a computer system that performs arithmetic and logic operations, which generally include adding, subtracting, transferring numbers from one area to another, and comparing two numbers. It's often known simply as a processor, a central processing unit, or as a logic chip.

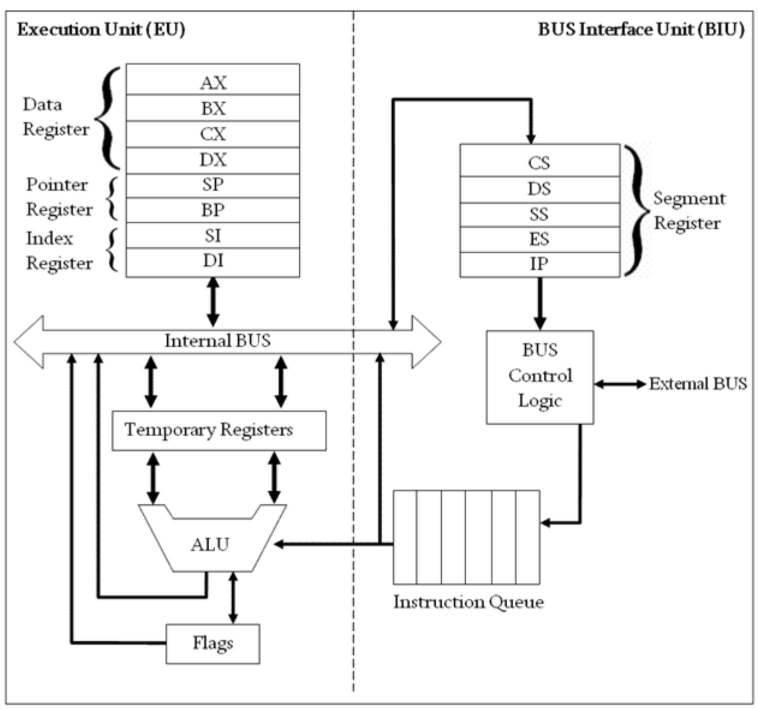


Figure 2.1: 8086 Microprocessor Internal Architecture

A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.8086 has 14 types of register- General(AX, BX, CX, DX), POINTER(SP, BP), INDEX(SI, DI), SEGMENT(CS,DS, ES, SS), INSTRUCTION(IP), FLAG(FR).FLAGS register is one of them. The purpose of the FLAGS register is to indicate the status of the microprocessor. It does this by setting up individual bits called flags. Depending upon the value of the result after any arithmetic and logical operation, the flag bits become set (1) or reset (0).8086 has a 16-bit flag register, and there are 9 valid flag bits. The flag bits are divided into two sections. The Status Flags, and the Control Flags. The status flags reflect the result of an instruction executed by the processor and the control flags enable or disable certain operations of the processor.

2.3 Required Software

- EMU8086

2.4 Problem A

Example 5.1: Make Summation of AX, BX. Where AX contains FFFFh, BX contains FFFFh

2.4.1 Steps

1. First addent was moved in AX by the command MOV
2. The Second addent was moved in BX by the command MOV
3. Finally 'ADD' command was used to operate the summation and the result was stored in AX

2.4.2 Program

Program 2.1: code for Example 5.1

```
;example 5.1

MOV AX,0FFFFH
MOV BX,0FFFFH
```

ADD AX, BX

2.4.3 Output

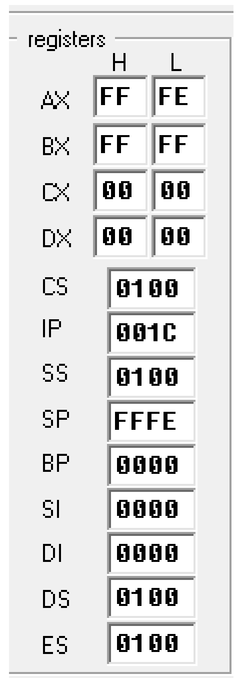


Figure 2.2: Registers After Summation of Example 5.1

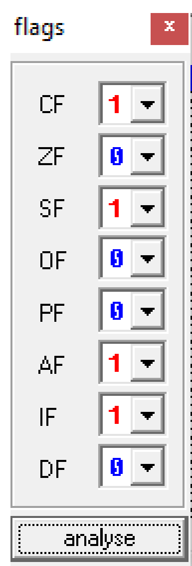


Figure 2.3: Flag Registers after Summation of Example 5.1

By observing the flag bits of the sum operation of example 5.1:

SF = 1 Because the MSB (Most Significant Bit) of the output is 1

PF = 0 Because there are 7 odd numbers of 1 bit in the low byte of the result

ZF = 0 Because the result is non-zero

CF = 1 because there is a carry-out of the MSB on addition

OF = 0 because No overflow occurred, Sign of result and operands are the same

2.5 Problem B

Example 5.2: Make Summation of AL, BL. Where AL contains 80hg, BL contains 80h

2.5.1 Steps

1. First addent was moved in AL by the command MOV
2. The Second addent was moved in BL by the command MOV
3. Finally 'ADD' command was used to operate the summation and the result was stored in AL in 8 bits

2.5.2 Program

Program 2.2: code for Example 5.2

```
;Example 5.2
MOV AL,80H
MOV BL,80H
ADD AL,BL
```

2.5.3 Output

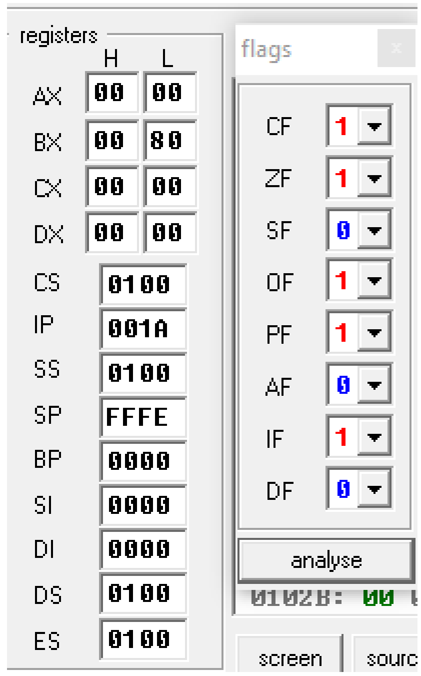


Figure 2.4: Registers and Flag bits of Example 5.2

By observing the flag bits of the sum operation of example 5.2:

SF = 0 Because the MSB (Most Significant Bit) of the output is 0

PF = 1 Because there are no odd numbers of 1 bit in the low byte of the result

ZF = 1 Because the result is zero

CF = 1 because there is a carry-out of the MSB on addition

OF = 1 because overflow occurred, Sign of result and operands are not the same

2.6 Problem C

Example 5.3 : Subtract AX,BX where AX contains 8000h and BX contains 0001h

2.6.1 Steps

1. First operand was moved in AX by the command MOV
2. The Negative operand was moved in BX by the command MOV
3. Finally 'SUB' command was used to operate the subtraction and the result was stored in AX in 16 bits

2.6.2 Program

Program 2.3: code for Example 5.3

```
;example 5.3
MOV AX,8000H
MOV BX,0001H
SUB AX,BX
```

2.6.3 Output

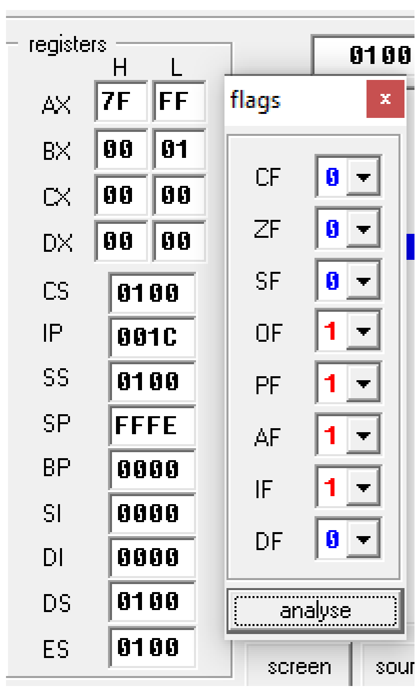


Figure 2.5: Registers and Flag bits of Example 5.3

By observing the flag bits of the sum operation of example 5.3:

SF = 0 Because the MSB (Most Significant Bit) of the output is 0

PF = 1 Because there are 8 even numbers of 1 bit in the low byte of the result

ZF = 0 Because the result is non-zero

CF = 0 because there is a smaller unsigned number is being subtracted from a larger one

OF = 1 because overflow occurred, Sign of result and operands are not the same

2.7 Problem D

Example 5.4: Use INC command, where AL contains FFh

2.7.1 Steps

1. First operand was moved in AL by the command MOV
2. The increment was done by applying 'INC' command
3. Finally the result was stored in AL in 8 bits

2.7.2 Program

Program 2.4: code for Example 5.4

```
;EXAMPLE 5.4
MOV AL,0FFH
INC AL
```

2.7.3 Output

AX =7FFFh SF = 0, PF = 1, ZF=0, CF=0

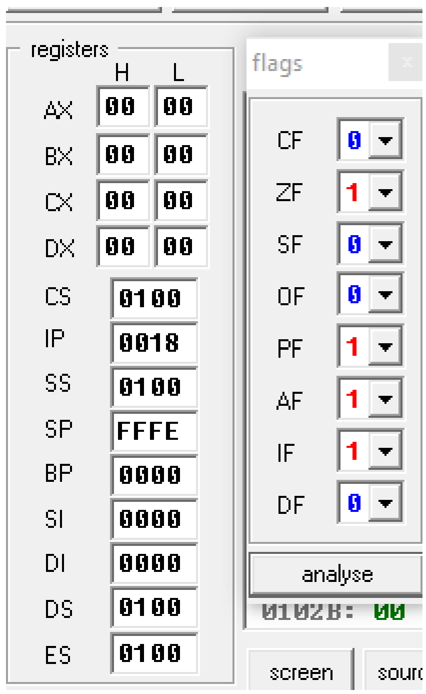


Figure 2.6: Registers and Flag bits of Example 5.4

By observing the flag bits of the sum operation of example 5.4:

SF = 0 Because the MSB (Most Significant Bit) of the output is 0

PF = 1 Because there are no even numbers of 1 bit in the low byte of the result

ZF = 1 Because the result is -zero

CF = 0 because INC command doesn't affect CF register

OF = 0 because no overflow occurred, Sign of result and operands are the same

2.8 Problem E1

Example 5.5: Use MOV command , where AX will contain -5

2.8.1 Steps

1. First operand was moved in AX by the command MOV

2. the result was stored in AX

2.8.2 Program

Program 2.5: code for Example 5.5

```
;EXAMPLE 5.5
MOV AX, -5
```

2.8.3 Output

AL =00h SF = 0, PF = 0, ZF=0

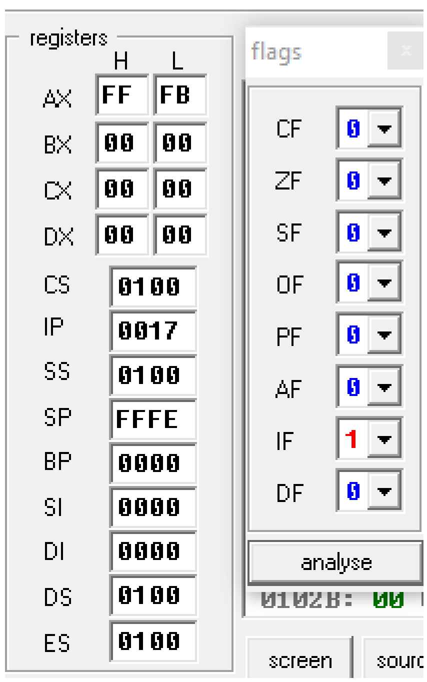


Figure 2.7: Registers and Flag bits of Example 5.5

By observing the flag bits of the sum operation of example 5.5:

SF = 0

PF = 0

ZF = 0

CF = 0

OF = 0

There is no change occurred in flag bits for the 'MOV' command

2.9 Problem E2

Example 5.6: Use of NEG AX, where AX contains 8000h

2.9.1 Steps

1. First operand was moved in AX by the command MOV

2. 1's compliment was found by using 'NEG' command

2.9.2 Program

Program 2.6: code for Example 5.6

```
MOV AX,8000H
NEG AX
```

2.9.3 Output

SF = 1, PF = 1, ZF=0 ,CF =1

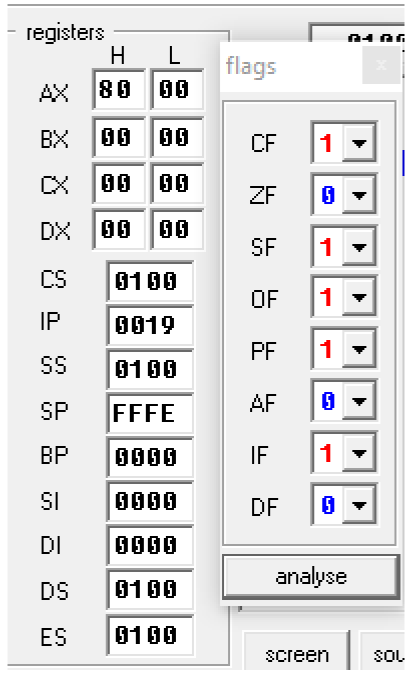


Figure 2.8: Registers and Flag bits of Example 5.6

By observing the flag bits of the sum operation of example 5.6:

SF = 1

Because the MSB (Most Significant Bit) of the output is 1 PF = 1

Because there are no 1 in low byte ZF = 0

This is a Non Zero answer CF = 1

because for NEG CF is always 1 OF = 1

signed overflow occurred

2.10 Problem F1

Exercise 1(a): Use Command ADO AX, BX where AX contains 7FFFh and BX contains QOOlh

2.10.1 Steps

1. First addent was moved in AX by the command MOV
2. The Second addent was moved in BX by the command MOV
3. Finally 'ADD' command was used to operate the summation and the result was stored

in AX

4. the result was stored in AX

2.10.2 Program

Program 2.7: code for Exercise 1

```
;EX - 2.1 (A)

MOV AX,7FFFH
MOV BX,0001H
ADD AX,BX
```

2.10.3 Output

SF = 1, PF = 1, ZF=0

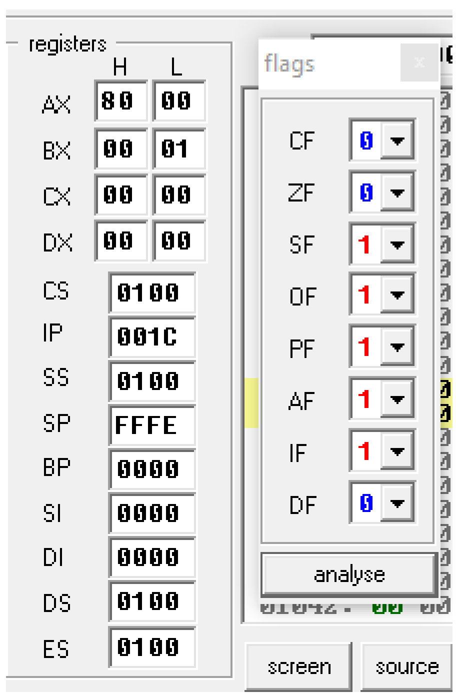


Figure 2.9: Registers and Flag bits of Exercise 1(a)

By observing the flag bits of the sum operation of exercise 1:

SF = 1

Because the MSB (Most Significant Bit) of the output is 1

PF = 1

Because there are even numbers of 1 bit in the low byte of the result

ZF = 0

It is a non zero answer

2.11 Problem F2

Exercise 1(b): Use Command SUB AL,BL where AL contains 01h and BL contains FFh

2.11.1 Steps

1. First operand was moved in AX by the command MOV
2. The Second operand was moved in BX by the command MOV
3. Finally 'SUB' command was used to operate the subtraction and the result was stored in AX
4. the result was stored in AX

2.11.2 Program

Program 2.8: code for Exercise 1

```
;example 5.1 (B)

MOV AL,01h
MOV BL,0FFh
SUB AL,BL
```

2.11.3 Output

SF = 0, PF = 0, ZF=0

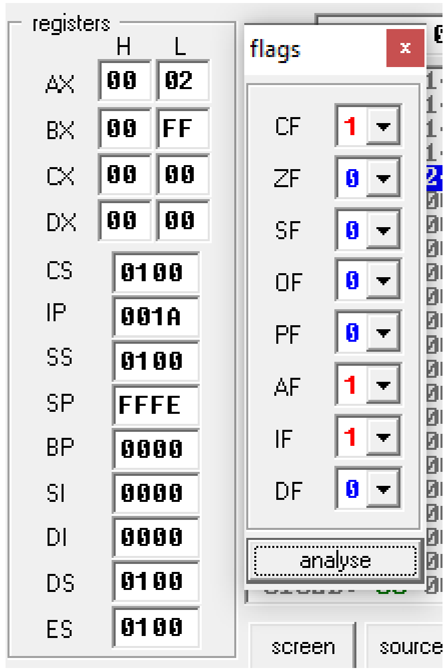


Figure 2.10: Registers and Flag bits of Exercise 1(b)

By observing the flag bits of the sum operation of exercise 1:

SF = 0

Because the MSB (Most Significant Bit) of the output is 0

PF = 0

Because there are odd numbers of 1 bit in the low byte of the result

ZF = 0

It is a zero answer

2.12 Problem F3

Exercise 1(c): DEC AL where AL contains 00h

2.12.1 Steps

1. First operand was moved in AL by the command MOV
2. DEC command is used to decrement the operand by 1
3. the result was stored in AL

2.12.2 Program

Program 2.9: code for Exercise 1

```
;e 2.1(c)

MOV AL,00H
DEC AL
```

2.12.3 Output

SF = 1, PF = 1, ZF=0

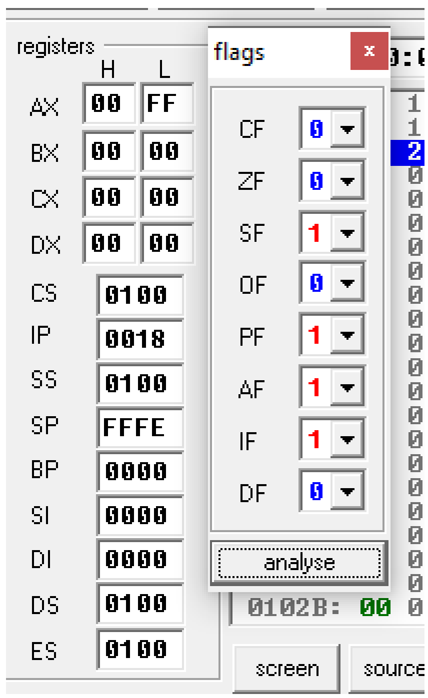


Figure 2.11: Registers and Flag bits of Exercise 1(c)

By observing the flag bits of the sum operation of exercise 1:

SF = 1

Because the MSB (Most Significant Bit) of the output is 1

PF = 1

Because there are even numbers of 1 bit in the low byte of the result

ZF = 0

It is a non zero answer

2.13 Problem F4

Exercise 1(d): NEG AL where AL contains 7Fh

2.13.1 Steps

1. First operand was moved in AL by the command MOV
2. NEG command is used to determine the NOT form or 1's Complement of AL
3. the result was stored in AL

2.13.2 Program

Program 2.10: code for Exercise 1

```
; 2.1 (D)

MOV AL,7FH
NEG AL
```

2.13.3 Output

SF = 1, PF = 1, ZF=0

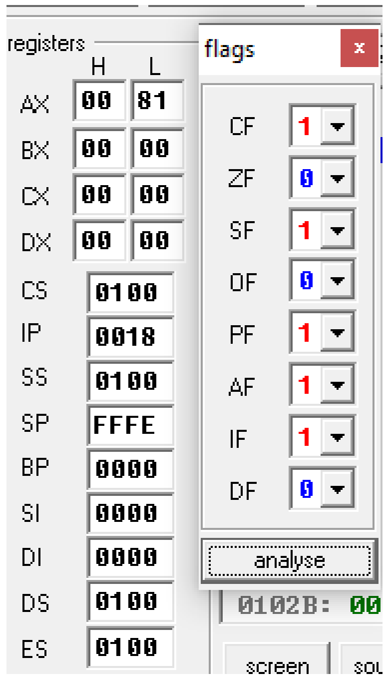


Figure 2.12: Registers and Flag bits of Exercise 1(d)

By observing the flag bits of the sum operation of exercise 1:

SF = 1

Because the MSB (Most Significant Bit) of the output is 1

PF = 1

Because there are even numbers of 1 bit in the low byte of the result

ZF = 0

It is a non zero answer

2.14 Problem F5

Exercise 1(e): XCHG AX,BX where AX contains 1ABCh and BX contains 712Ah

2.14.1 Steps

1. First operand was moved in AX by the command MOV
2. Second operand was moved in BX by the command MOV
3. XCHG interchanged the AX and BX

2.14.2 Program

Program 2.11: code for Exercise 1

```
; 2.1 (E)
MOV AX, 1ABCH
MOV BX, 712AH
XCHG AX, BX
```

2.14.3 Output

SF = 0, PF = 0, ZF=0

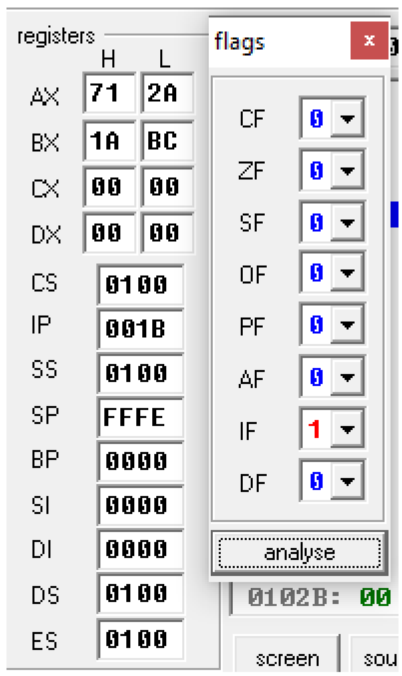


Figure 2.13: Registers and Flag bits of Exercise 1(e)

By observing the flag bits of the sum operation of exercise 1:

SF = 0 PF = 0 ZF = 0 No change occurred in the program for XCHG command

2.15 Problem F6

Exercise 1(f): ADD AL, BL where AL contains 80h and BL contains fFh

2.15.1 Steps

1. First operand was moved in AL by the command MOV
2. Second operand was moved in BL by the command MOV
3. ADD command was used to store the summation value in AL

2.15.2 Program

Program 2.12: code for Exercise 1

```
; 2.1 (F)
MOV AL,80H
MOV BL,0FFH
ADD AL,BL
```

2.15.3 Output

SF = 0, PF = 0, ZF=0

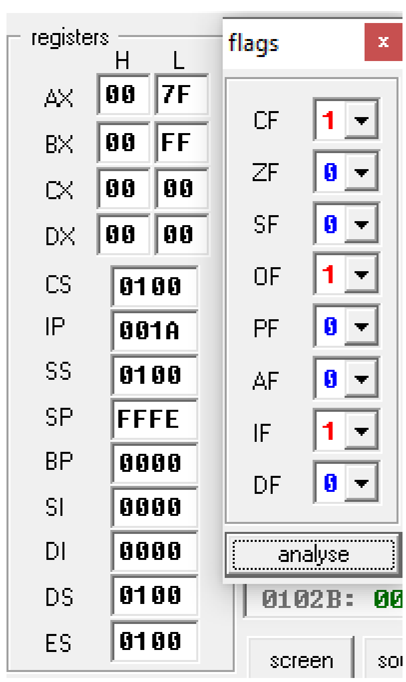


Figure 2.14: Registers and Flag bits of Exercise 1(f)

By observing the flag bits of the sum operation of exercise 1:

SF = 0

Because the MSB (Most Significant Bit) of the output is 0

PF = 0

Because there are odd numbers of 1 bit in the low byte of the result

ZF = 0

It is a non zero answer

CF = 1

There is a carry out from MSB

2.16 Problem F7

Exercise 1(g): Use Command SUB AX,BX where AX contains OOOOh and BX contains 8000h

2.16.1 Steps

1. First operand was moved in AX by the command MOV
2. The Second operand was moved in BX by the command MOV
3. Finally ’SUB’ command was used to operate the subtraction
4. the result was stored in AX

2.16.2 Program

Program 2.13: code for Exercise 1

```
; 2.1 (G)
MOV AX,0000H
MOV BX,8000H
SUB AX,BX
```

2.16.3 Output

SF = 1, PF = 1, ZF=0,CF=1

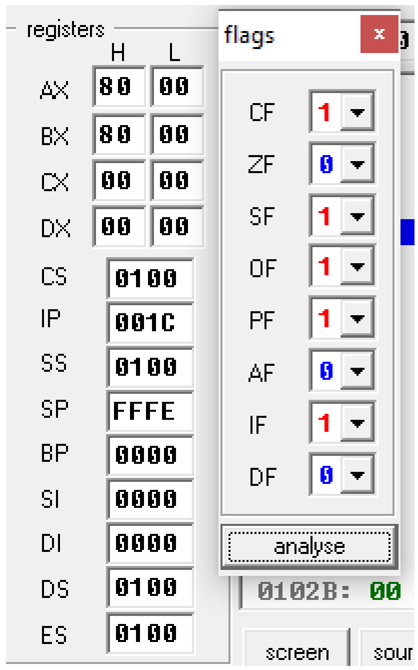


Figure 2.15: Registers and Flag bits of Exercise 1(g)

By observing the flag bits of the sum operation of exercise 1:

SF = 1

Because the MSB (Most Significant Bit) of the output is 1

PF = 1

Because there are even numbers of 1 bit in the low byte of the result

ZF = 0

It is a non zero answer

CF = 1

There is a carry out from MSB

2.17 Problem F8

Exercise 1(h): Use Command NEG AX where AX contains 0001h

2.17.1 Steps

1. First operand was moved in AX by the command MOV
2. NEG command is used to determine the NOT form or 1's Complement of AX
3. the result was stored in AX

2.17.2 Program

Program 2.14: code for Exercise 1

```
; 2.1 H

MOV AX,0001H
NEG AX
```

2.17.3 Output

SF = 0, PF = 0, ZF=0,CF=0

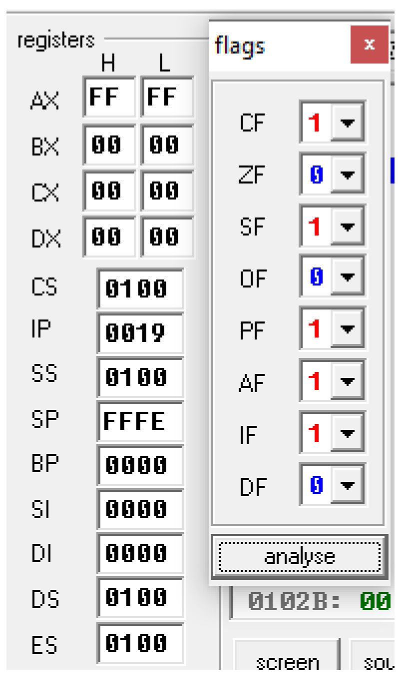


Figure 2.16: Registers and Flag bits of Exercise 1(H)

By observing the flag bits of the sum operation of exercise 1(h):

SF = 1

MSB is 1

PF = 1

There are even no of 1

ZF = 0

Non zero answer

CF = 1

There is a carry out of MSB

OF = 0

There is no Overflow

2.18 Discussion and Conclusions

After finishing the experiment, it is seen that in general each time the processor executes an instruction, the flags are altered to reflect the result. However, some instructions don't affect any of the flags or some of them remain undefined. From the above experiments - 'MOV',"XCHG" these commands have no effect on flags. "INC" and "DEC" affects all flags except CF.'ADD',SUB and NEG affects all flags

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 3

Experimental Study on String Instruction on 8086 by using EMU8086.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 15/10/2022

Date of Submission : 24/10/2022

(Teacher's Section)

Report

☐ Excellent

☐ Very Good

☐ Good

☐ Average

☐ Poor

Signature

Viva

☐ Excellent

☐ Very Good

☐ Good

☐ Average

☐ Poor

Experiment 3

Experimental Study on String Instruction on 8086 by using EMU8086

3.1 Objectives

The main objectives of this experiment are

- To learn to take input and display output
- To understand string.
- To learn the case letters

3.2 Introduction

The string is a series of data bytes or words available in memory at consecutive locations. It is either referred to as a byte string or a word string. Their memory is always allocated in sequential order. Instructions used to manipulate strings are called string manipulation instructions. Operations include storing strings in memory, loading strings from memory, comparing strings, and scanning strings for substrings. There are two categories of I/O service routines:(1)The Basic INPUT/OUTPUT System(BIOS) routines (2) DOS routines. To invoke a DOS or BIOS routine, the INT (interrupt) instruction is used.The size of code and data a program can have is determined by specifying a memory model using .MODEL directive.

3.3 Required Software

1. EMU8086

3.4 Problem A

Take a string and display the string

3.4.1 Steps

1. At input creating section AH =1 and INT 21h is fixed
2. At output creating section AH =2 and INT 21h is fixed
3. To create display DL = 0Dh and to create space DL = 0Ah is written

3.4.2 Program

Program 3.1: code for Program A

```
MOV AH, 1
INT 21h
MOV BL, AL

MOV AH, 2
MOV DL, 0Dh
INT 21h
MOV DL, 0Ah
INT 21h

MOV DL, BL
INT 21h
```

3.4.3 Output

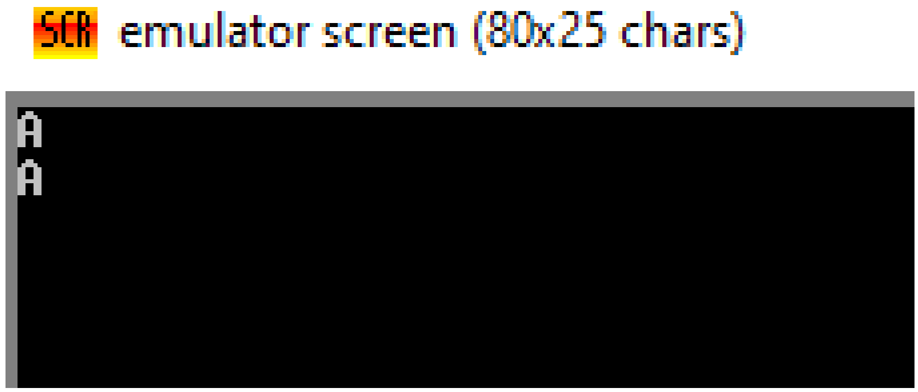


Figure 3.1: Input and Output

3.5 Problem B

Print Hello in assembly language

3.5.1 Steps

1. At first The size of the workspace is determined by the command .STACK 100h
2. Then the string that has to be printed is written with the symbol is under command MSG DB
3. The main program body is defined by MAIN PROC
4. TTo determine string AH=9 has to be written

3.5.2 Program

Program 3.2: code for Program B

```
.MODEL SMALL
.STACK 100H
.DATA

MSG DB 'HELLO$'
```

```
.CODE
MAIN PROC
        ;INITIALIZE DS
        MOV AX,@DATA
        MOV DS,AX
        ;DISPLAY MESSAGE
        LEA DX, MSG
        MOV AH,9
        INT 21H
        ;RETURN TO DOS
        MOV AH,4CH
        INT 21H
MAIN ENDP
      END MAIN
```

3.5.3 Output

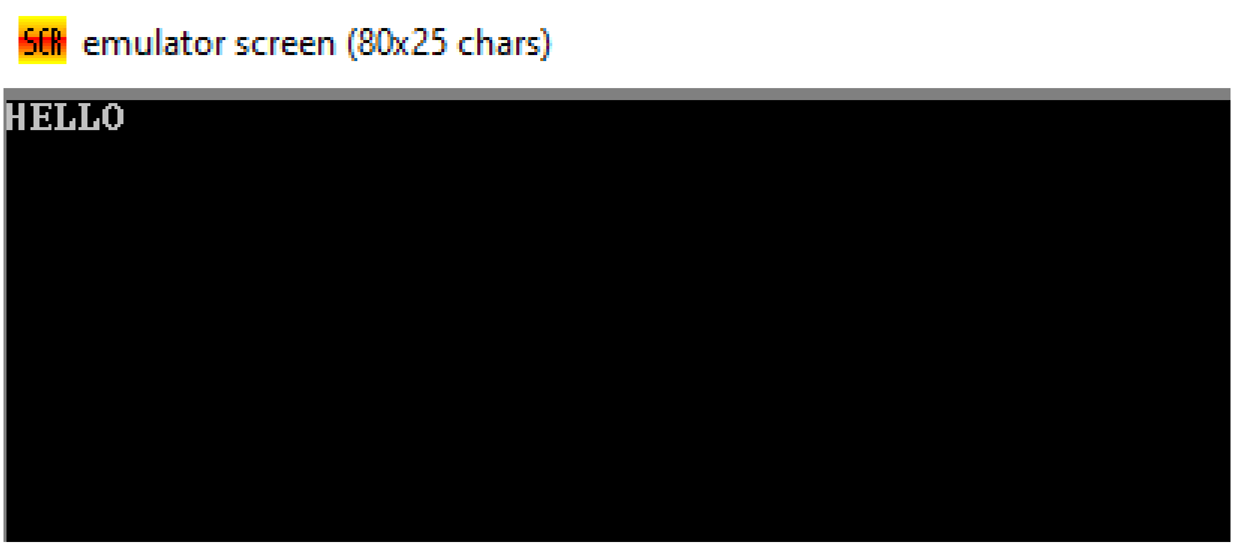


Figure 3.2: Printing String

3.6 Problem C

Take a small letter as input and give output as a capital letter

3.6.1 Steps

1. At first a small letter is taken as input

2. Then the input is substracting with 20H to get the required output

3.6.2 Program

Program 3.3: code for Program C

```
.MODEL SMALL
.STACK 100H
.DATA
MSG1 DB 'ENTER A LOWER LETTER: $'
MSG2 DB ' IN UPPER CAS , IT IS-$'
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;DISPLAY MESSAGE
    LEA DX, MSG1
    MOV AH,9
    INT 21H
    ;CHARACTER INPUT
    MOV AH,1
    INT 21H ; input a small letter
    SUB AL,20H
    MOV BL,AL
```

```
    ;New Line
    MOV AH,2
    MOV DL,0DH
    INT 21H
    MOV DL,0AH
    INT 21H
     ;DISPLAY MESSAGE2
    LEA DX, MSG2
    MOV AH,9
    INT 21H

    ;OUTPUT
    MOV AH,2
    MOV DL,BL
    INT 21H

;DOS EXIT
   MOV AH,4CH
   INT 21H
MAIN ENDP
   END MAIN
```

3.6.3 Output

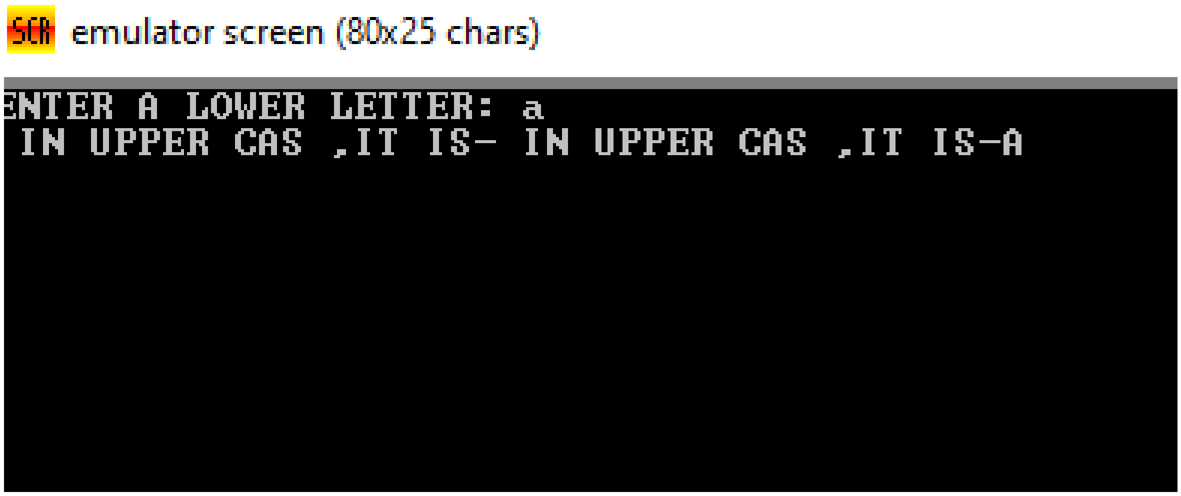


Figure 3.3: Printing String- taking Input as a Lower Case and giving output as an Upper Case

3.7 Problem D

Take a capital letter as input and give output as a small letter

3.7.1 Steps

1. At first a capital letter is taken as input
2. Then the input is added with 20H to get the required output

3.7.2 Program

Program 3.4: code for Program C

```
.MODEL SMALL
.STACK 100H
.DATA
MSG1 DB 'ENTER␣A␣Capital␣LETTER:␣$'
MSG2 DB '␣IN␣Lower␣CAS␣,IT␣IS-$'
.CODE
MAIN PROC
```

```
;INITIALIZE DS
MOV AX,@DATA
MOV DS,AX
;DISPLAY MESSAGE
LEA DX, MSG1
MOV AH,9
INT 21H
;CHARACTER INPUT
MOV AH,1
INT 21H ; input a small letter
ADD AL,20H
MOV BL,AL
;New Line
MOV AH,2
MOV DL,0DH
INT 21H
MOV DL,0AH
INT 21H
;DISPLAY MESSAGE2
LEA DX, MSG2
MOV AH,9
INT 21H
;OUTPUT
MOV AH,2
MOV DL,BL
INT 21H

;DOS EXIT
MOV AH,4CH
INT 21H
```

```
MAIN ENDP
    END MAIN
```

3.7.3 Output

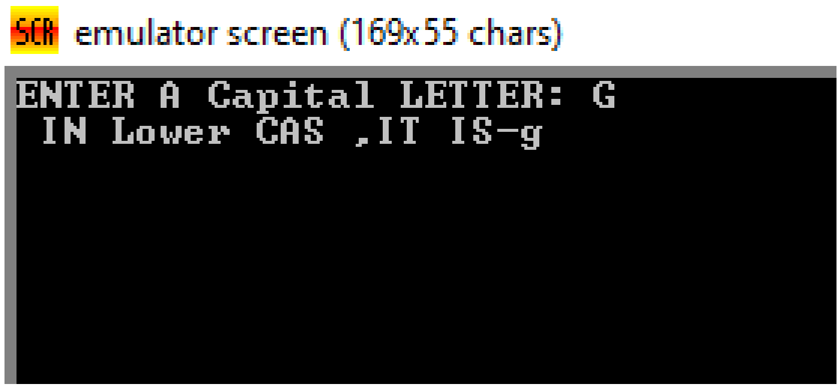


Figure 3.4: Printing String-Upper Case Input Lower case Output

3.8 Conclusions and Discussions

From the experiment, it is seen that when AH =1, the input function was indicated, and when AH=2 output function was indicated. When AH = 9, it is indicated as a string. INT 21h is used to invoke a large number of DOS functions. Again in 8086, the small letter can be converted to the capital letter or vice versa by adding or subtracting ASCII code with the position of that letter in ASCII chart.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 4

Experimental Study on Input and Output of 8086 by using EMU8086.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 24/10/2022

Date of Submission : 29/10/2022

(Teacher's Section)

Report

☐ Excellent

☐ Very Good

☐ Good

☐ Average

☐ Poor

Signature

Viva

☐ Excellent

☐ Very Good

☐ Good

☐ Average

☐ Poor

Experiment 4

Experimental Study on Input and Output of 8086 by using EMU8086

4.1 Objectives

The main objectives of this experiment are

- To learn to take two inputs and do the mathematical operations with them
- To learn compact coding with display

4.2 Introduction

There are two categories of I/O service routines:(1)The Basic INPUT/OUTPUT System(BIOS) routines and (2) DOS routines. To invoke a DOS or BIOS routine, the INT (interrupt) instruction is used. The size of code and data a program can have is determined by specifying a memory model using.MODEL directive. In an assembly language, it is not possible to accept a number with more than one digit at once or to display a number with more than one digit simultaneously. One must input user input one character at a time and print one at a time.

4.3 Required Software

1. EMU8086

4.4 Problem A

Write a program to read two decimal digits whose sum is less than 10, display them and their sum on the next line with an appropriate message

4.4.1 Steps

1. At first the code format was written
2. Then strings that were going to be used through the program was written
3. DS was initialized
4. First input was taken by using the command INT and saved to the S2 variable that was declared with the strings at step 2 and moved to BL
5. Second input was taken and saved to the S3 variable that was declared with the strings at step 2
6. these two variables were added using ADD command
7. then this sum was subtracted by 30h to get the numerical value in hexadecimal
8. finally output was shown by displaying the output window

4.4.2 Program

Program 4.1: code for Program A

```
;Chapter 4, exercise 8
;Write a program to read two deciamal digits whose sum is less than 10,
;display them and their sum on the next line with an appropriate message
.MODEL SMALL
.STACK 100H
.DATA
.CODE
S1 DB 0DH,0AH, "THE␣SUM␣OF␣"
S2 DB ?,"␣AND␣␣"
S3 DB ?,"␣IS␣␣"
S4 DB ?,"␣$"
```

```
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX

    ; INPUT 1
    MOV AH,1
    INT 21H ; input a NUMBER
    MOV S2,AL
    MOV BL,AL

    ; INPUT 2
    MOV AH,1
    INT 21H ; input aNOTHER NUMBER
    MOV S3,AL
    ADD BL,AL
    SUB BL,30H
    MOV S4,BL

    ;DISPLAY MESSAGE2
    LEA DX, S1
    MOV AH,9
    INT 21H

    ;OUTPUT
    MOV AH,2
    MOV DL,BL

   ;DOS EXIT
   MOV AH,4CH
```

```
    INT 21H
      MAIN ENDP
END MAIN
```

4.4.3 Output

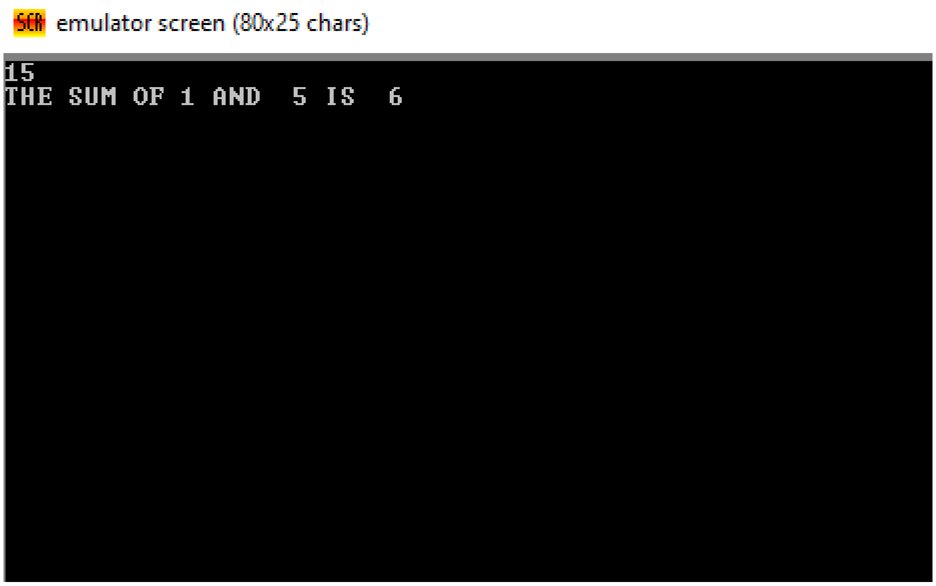


Figure 4.1: Input and Output

4.5 Problem B

Write a program to take a small letter as input and give output as a capital letter with compact coding

4.5.1 Steps

1. At first the code format was written
2. Then strings that were going to be used through the program was written
3. Then instructions for carrier and new line were added with the final string message
4. DS was initialized and the first input message string was displayed by using the LEA command

5. Input was taken with INT command and subtracted with 20H to get a capital letter in hexadecimal in ASCII code
6. Finally message was displayed in the output window

4.5.2 Program

Program 4.2: code for Program B

```
.MODEL SMALL
.STACK 100H
.DATA
S1 DB    'ENTER␣A␣LOWER␣LETTER:␣␣$'
S2 DB 0DH,0AH, 'IN␣UPPER␣CASE␣IT␣IS␣:␣'
S3 DB ?,'$'
.CODE
MAIN PROC
      ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX

    LEA DX,S1
    MOV AH,9
    INT 21H
    ; INPUT 1
    MOV AH,1
    INT 21H
    SUB AL,20H
    MOV S3,AL
    ;DISPLAY MESSAGE2
    LEA DX, S2
    MOV AH,9
```

```
INT 21H
;DOS EXIT
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

4.5.3 Output

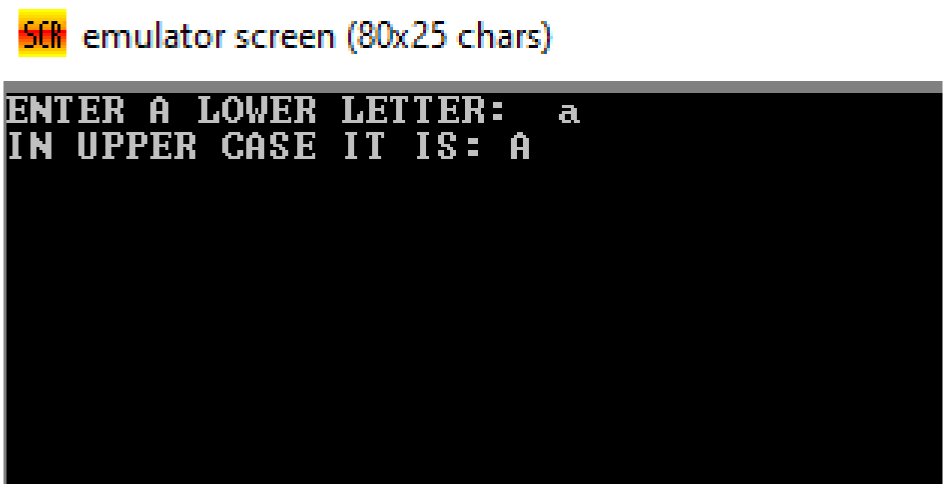


Figure 4.2: Lower case Input and Upper Case Output

4.6 Problem c

Write a program to take an capital letter as input and give output as a small letter with compact coding

4.6.1 Steps

1. At first the code format was written
2. Then strings that were going to be used through the program was written
3. Then instructions for carrier and new line were added with the final string message
4. DS was initialized and the first input message string was displayed by using the LEA command

5. Input was taken with INT command and added with 20H to get a small letter in hexadecimal in ASCII code

6. Finally message was displayed in the output window

4.6.2 Program

Program 4.3: code for Program C

```
.MODEL SMALL
.STACK 100H
.DATA
S1 DB    'ENTER␣A␣Upper␣LETTER:␣␣$'
S2 DB 0DH,0AH, 'IN␣Lower␣CASE␣IT␣IS␣:␣'
S3 DB ?,'$'
.CODE
MAIN PROC
        ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX

    LEA DX,S1
    MOV AH,9
    INT 21H
    ; INPUT 1
    MOV AH,1
    INT 21H
    ADD AL,20H
    MOV S3,AL
    ;DISPLAY MESSAGE2
    LEA DX, S2
    MOV AH,9
```

```
INT 21H
;DOS EXIT
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

4.6.3 Output

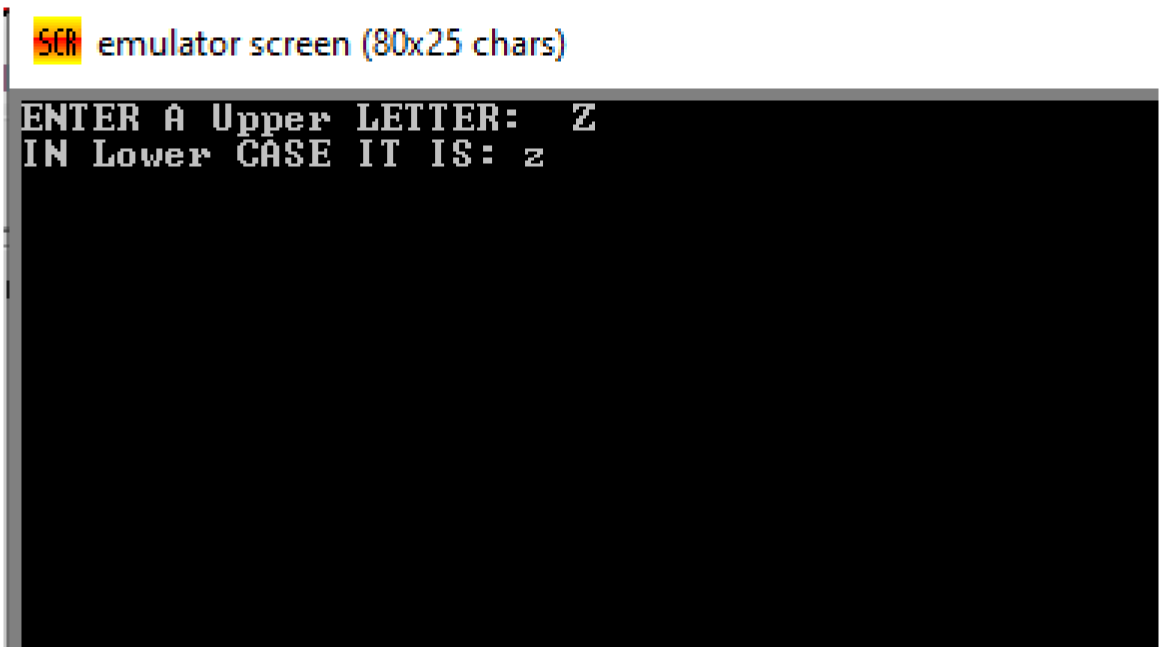


Figure 4.3: Upper case Input and Lower Case Output

4.7 Conclusions and Discussions

By observing the experiment, it is seen that a question mark was used several times while declaring string variables. It is observed that these string variables with question marks were used later to store values and display them. Again whenever it came to taking an input, the value of Ah became 1 and for output, it became 2 and for displaying string it became 9. It was also observed that Whenever doing any mathematical or case operation, it was compulsory to add or subtract the hexadecimal value of the ASCII code as the assembly language operates on ASCII code.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 5

Experimental Study on Problems based on Array, Number Conversion and Sound Output

---

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 29/10/2022

Date of Submission : 05/11/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	____________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 5

Experimental Study on Problems based on Array, Number Conversion and Sound Output

5.1 Objectives

The main objectives of this experiment are

- To learn to print array with asterisks
- To learn to take three individual inputs and display them
- To learn to make a beep sound as an output

5.2 Introduction

An array is a collection of similar data elements stored at contiguous memory locations. It is the simplest data structure where each data element can be accessed directly by only using its index number. In assembly language, there are some control characters that are used to do some specific works like making a beep sound, backspace or line break etc.The principal control characters are as follows:

no	ASCII(hex)	Symbol	function
1	7	BEL	beep(sounds a beep)
2	8	BS	backspace
3	9	HT	tab
4	A	LF	new line
5	D	CR	start of current line

On execution,AL gets the ASCII code of the control characters.

5.3 Required Software

1. EMU8086

5.4 Problem A

Write a program to (a) prompt the user, (b) read first, middle, and last initials of a person's name, and (c) display them down the left margin

5.4.1 Steps

1. At first the code format was written
2. Then strings and variables that were going to be used through the program was written
3. Inputs were taken using MOV AH,1 and INT 21h commands and stored them in S2,S3 and S4 variables
4. Outputs were displayed by using LEA DX command.

5.4.2 Program

Program 5.1: code for Program A

```
 ; exercise 9
.MODEL SMALL
.STACK 100H
.DATA
S1 DB  , 'ENTER␣Three␣initials:$␣'
P1 DB 0AH,0DH
S2 DB    ?,0AH,0DH
S3 DB ?,0AH,0DH
S4 DB ?,"$"
.CODE
MAIN PROC
    ;INITIALIZE DS
   MOV AX,@DATA
   MOV DS,AX
   ;message 1
```

```
LEA DX, S1
MOV AH,9
INT 21H
; INPUT 1
MOV AH,1
INT 21H
MOV S2 ,AL
; INPUT 2
MOV AH,1
INT 21H
MOV S3 ,AL
; INPUT 3
MOV AH,1
INT 21H
MOV S4 ,AL
;DISPLAYING OUTPUTS
LEA DX, P1
MOV AH, 9
INT 21H
;DOS EXIT
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

5.4.3 Output

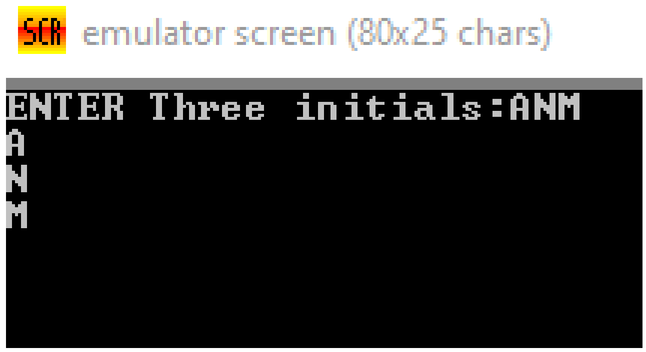


Figure 5.1: Input and Output

5.5 Problem B

Write· a program to read one of the hex digits A-F', and display it on the next line in decimal.

5.5.1 Steps

1. At first the code format was written
2. Then the strings that were used in the code were written
3. V1 variable was taken to store the output
4. As in this problem the value would be in the range from 10 to 15, in this case, 1 was common. So in the program 1 was written in a string message and 0 to 5 was printed according to the input
5. 11h was substracted from the hex value according to A to F to get 0 to 5 in decimal
6. Outputs were displayed by using LEA DX command.

5.5.2 Program

Program 5.2: code for Program B

```
;EXERCISE 10
```

```
.MODEL SMALL
.STACK 100H
.DATA
S1 DB 'ENTER␣A␣HEX␣DIGIT:␣$'
S2 DB 0DH, 0AH, 'IN␣DECIMAL␣IT␣IS␣1'
V1 DB ?, '$'
.CODE
MAIN PROC
    ;INITIALIZING
    MOV AX, @DATA
    MOV DS, AX
    ;PRINT INPUT MESSAGE
    LEA DX, S1
    MOV AH, 9
    INT 21H
    ;TAKING INPUT
    MOV AH, 1
    INT 21H
    ;SUBSTRACTING
    SUB AL, 11H ;SUBSTRACTING BY 11H
    MOV V1, AL
    ;OUTPUT DISPLAY
    LEA DX, S2
    MOV AH, 9
    INT 21H
    MOV AH, 4CH
    INT 21H
    MAIN ENDP
END MAIN
```

5.5.3 Output

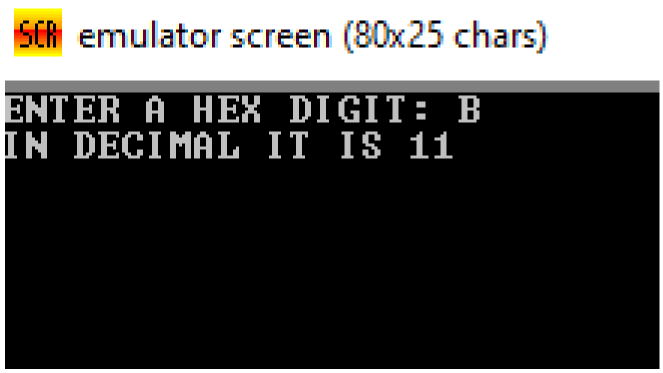


Figure 5.2: Hexadecimal Input to Decimal Output

5.6 Problem C

Write a program to display a 10 x 10 solid box of asterisks.

5.6.1 Steps

1. At first the code format was written
2. Then two separate lines of a string of asterisks were written. There were 10 asterisks in each set
3. first line of the asterisk was displayed using LEA DX command.
4. the Second line of the asterisk was displayed using LEA DX command and repeated 9 times by using INT 21h repeatedly.

5.6.2 Program

Program 5.3: code for Program C

```
;EXERCISE 11
.MODEL SMALL
.STACK 100H
.DATA
```

```
S1 DB   '**********$'
S2 DB 0AH, 0DH,'**********$',
.CODE
MAIN PROC
     ;INITIALIZING
     MOV AX, @DATA
     MOV DS, AX
     ;PRINT FIRST LINE
     LEA DX, S1
     MOV AH, 9
     INT 21H
     LEA DX, S2    ;PRINT SECOND LINE
     MOV AH, 9
     INT 21H
     ;REPEATING SECOND LINE
     INT 21H
     INT 21H
     INT 21H
     INT 21H
     INT 21H
     INT 21H
     INT 21H
     INT 21H

     MAIN ENDP
END MAIN
```

5.6.3 Output

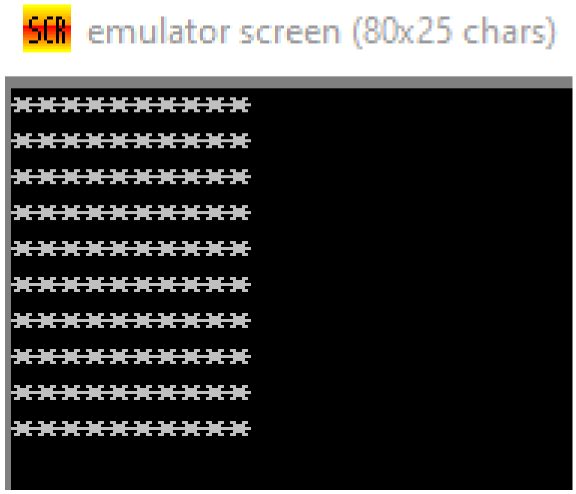


Figure 5.3: 10x10 asterisk array

5.7 Problem D

Write a program to (a) display"?", (b) read three initials,(c) display them in the middle of an 11 x 11 box of asteri5ks, and (d) beep the computer.

5.7.1 Steps

1. At first the code format was written
2. Then two lines of asterisks were written
3. In the first line 11 asterisks was written and in the second line 4 asterisks were set
4. After that three variables were taken to store three letters
5. In the program at first inputs was taken
6. then 5 lines of asterisks were printed
7. Then the line of 4 asterisks was printed and on its right inputs were printed and finally, the 4 asterisks were printed again.
8. Again 5 lines of asterisks were printed
9. Finally beep sound was made by applying MOV DL,7

5.7.2 Program

Program 5.4: code for Program D

```
;Exercise 12
.MODEL SMALL
.STACK 100h
.DATA
    M1 DB 'ENTER␣THREE␣INPUTS:$'
    M3 DB 0AH,0DH,'*␣*␣*␣*␣*␣*␣*␣*␣*␣*␣*$'
    M4 DB 0AH,0DH,'*␣*␣*␣*␣'
    S1 DB ?,'␣'
    S2 DB ?,'␣'
    S3 DB ?,'␣'
    M5 DB '*␣*␣*␣*$'
.CODE
MAIN PROC
    ;INITIALISING
    MOV AX,@DATA
    MOV DS,AX
    ;DISPLAYING first message
    LEA DX,M1
    MOV AH,9
    INT 21h
    ;GETTING THE inputs
    MOV AH,1
    INT 21h
    MOV S1,AL
    MOV AH,1
    INT 21h
    MOV S2,AL
```

```
MOV AH,1
INT 21h
MOV S3 ,AL
;DISPLAYING A STRING
LEA DX,M3
MOV AH,9
INT 21h
INT 21h
INT 21h
INT 21h
INT 21h
LEA DX,M4; displaying M4 to M5 including S1,S2,S3
MOV AH,9
INT 21h
LEA DX,M3
MOV AH,9
INT 21h
INT 21h
INT 21h
INT 21h
INT 21h
;GETTING A BEEP SOUND
MOV AH,2
MOV DL,7
INT 21h
;EXIT TO DOS
MOV AH,4Ch
INT 21h
MAIN ENDP
END MAIN
```

5.7.3 Output

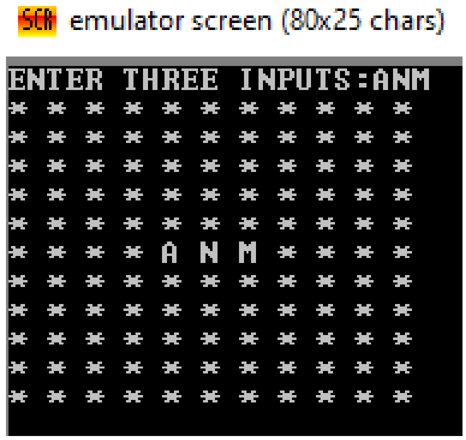


Figure 5.4: 11x11 asterisk array

5.8 Conclusions and Discussions

The Experiment was about four different problems of assembly language. From the first problem, it was observed that carrier space and new line command didn't work well with the first message line string. For that reason, carrier space and new line command were used as separate lines. Then for the second problem, the input was substracted by 11H to get the hexadecimal value for the required capital letter. For the third problem to create an asterisk array, INT 21h command was used several times to print the same line for the asterisk to create the array. The same procedure was applied to create the next problem's array. And to make a beep sound, the value 7 was moved to DL. While writing asterisks it was ensured that there was no additional space inside the strings.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 6

Experimental Study on Problems based on Multiplication and Division

---

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 05/11/2022

Date of Submission : 12/11/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	____________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 6

Experimental Study on Problems based on Multiplication and Division

6.1 Objectives

The main objectives of this experiment are

- To learn to work with multiplication commands
- To learn to work with division command

6.2 Introduction

It is possible to multiply binary data using two distinct instructions. The MUL (Multiply) instruction handles unsigned data, whereas the IMUL instruction handles signed data (Integer Multiply). On the Carry and Overflow flags, both commands have an effect. The multiplicand in both cases will be in an accumulator depending on the size of the multiplicand and multiplier, and the final product will also be placed in two registers depending on the size of the operands.
A quotient and a remainder are the two results of the division operation. Since the product is stored in double-length registers during multiplication, overflow does not happen. Overflow could happen in the division scenario, though. If overflow happens, the processor generates an interrupt. For unsigned data, the DIV (Divide) instruction is used, whereas for signed data, the IDIV (Integer Divide) instruction is used. The dividend is stored in a bucket. The operands for both instructions can be 8-bit, 16-bit, or 32-bit.

6.3 Required Software

1. EMU8086

6.4 Problem A

Suppose AX contains 1 and.BX contains FFFFh, Do multiplication

6.4.1 Steps

1. At first the code format was written
2. For signed multiplication 'IMUL' command was used and for unsigned multiplication 'MUL' was used

6.4.2 Program

Program 6.1: code for Program A(unsigned)

```
;EXM 9.1 (unsigned)
MOV AX,1
MOV BX,0FFFFH
MUL BX
```

Program 6.2: code for Program A(signed)

```
;EXM 9.1
MOV AX,1
MOV BX,0FFFFH
IMUL BX
```

6.4.3 Output

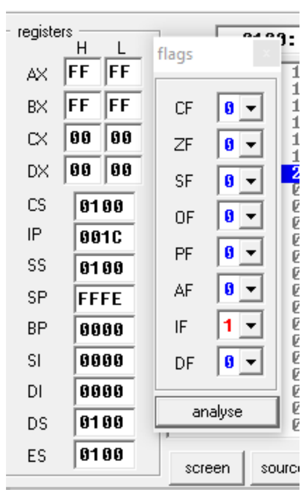


Figure 6.1: Unsigned Multiplication

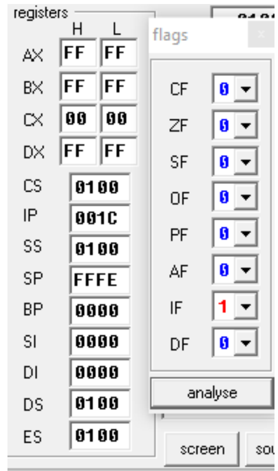


Figure 6.2: Signed Multiplication

6.5 Problem B

Suppose AX contains FFFFh and BX contains FFFFh.Do Multiplication

6.5.1 Steps

1. At first the code format was written
2. For signed multiplication 'IMUL' command was used and for unsigned multiplication 'MUL' was used

6.5.2 Program

Program 6.3: code for Program B(unsigned)

```
;EXM 9.2(unsigned)
MOV AX,0FFFFH
MOV BX,0FFFFH
MUL BX
```

Program 6.4: code for Program B(Signed)

```
;EXM 9.2
MOV AX,0FFFFH
MOV BX,0FFFFH
```

IMUL BX

6.5.3 Output

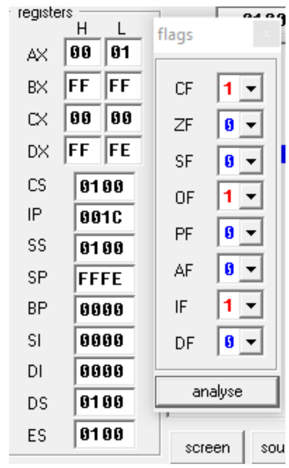


Figure 6.3: Unsigned Multiplication

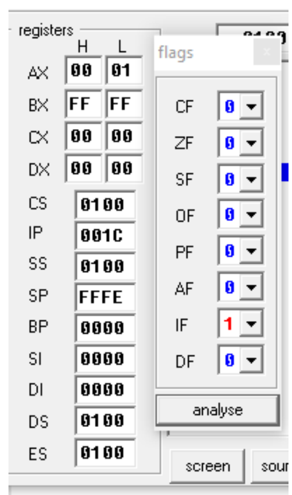


Figure 6.4: Signed Multiplication

6.6 Problem C

Suppose AX contains OFFFh: DO multiplication

6.6.1 Steps

1. At first the code format was written

2. For signed multiplication 'IMUL' command was used and for unsigned multiplication 'MUL' was used

6.6.2 Program

Program 6.5: code for Program C(unsigned)

```
;9.3 (Unsigned)
MOV AX,0FFFH
MUL AX
```

Program 6.6: code for Program C(Signed)

```
;9.3 (signed)
MOV AX,0FFFH
IMUL AX
```

6.6.3 Output

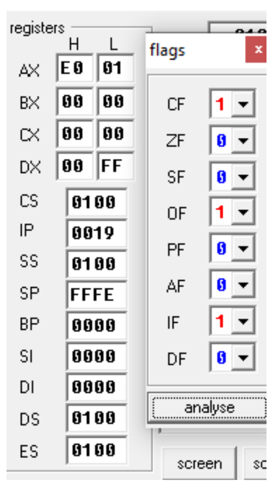


Figure 6.5: Unsigned Multiplication

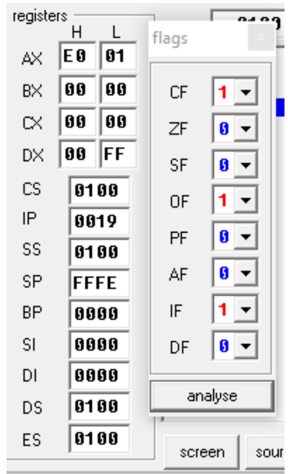


Figure 6.6: Signed Multiplication

6.7 Problem D

Suppose AL contains 80h and BL contains FFh

6.7.1 Steps

1. At first the code format was written
2. For signed multiplication 'IMUL' command was used and for unsigned multiplication 'MUL' was used

6.7.2 Program

Program 6.7: code for Program D(unsigned)

```
;EXM 9.4(Unsigned)
MOV AX,0100H
MOV CX,0FFFFH
MUL CX
```

Program 6.8: code for Program D(Signed)

```
;EXM 9.4(signed)
MOV AX,0100H
MOV CX,0FFFFH
```

IMUL CX

6.7.3 Output

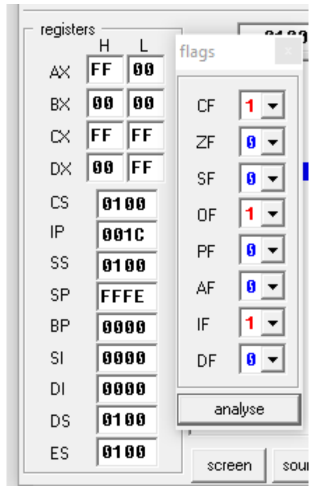


Figure 6.7: Unsigned Multiplication

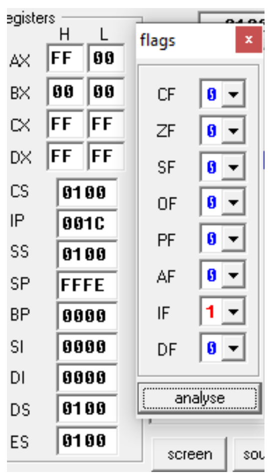


Figure 6.8: Signed Multiplication

6.8 Problem E

Suppose AL contains 80h and BL contains FFh:

6.8.1 Steps

1. At first the code format was written

2. For signed multiplication 'IMUL' command was used and for unsigned multiplication 'MUL' was used

6.8.2 Program

Program 6.9: code for Program E(unsigned)

```
;EXM 9.5
MOV AL,80H
MOV BL,0FFH
MUL BL
```

Program 6.10: code for Program E(Signed)

```
;EXM 9.5
MOV AL,80H
MOV BL,0FFH
IMUL BL
```

6.8.3 Output

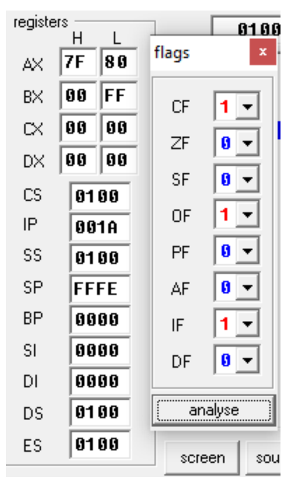


Figure 6.9: Unsigned Multiplication

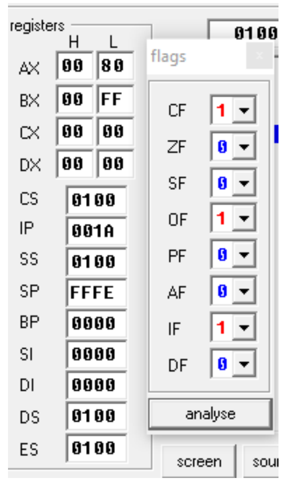


Figure 6.10: Signed Multiplication

6.9 Problem F

Translate the high-level language assignment statement A = 5 x A - 12 x B into assembly code. Let A and B be word variables, and suppose there is no overflow. Use IMUL for multiplication.

6.9.1 Steps

1. At first the code format was written
2. then 2 variables were taken and filled with 3 and 1 respectively
3. Then 5 was taken in AX and multiplied with 3 using the 'IMUL' command and the same process was used with 12 to multiply it with 1

6.9.2 Program

Program 6.11: code for Program F

```
.MODEL SMALL
.STACK 100H
.DATA

A DW 3
```

```
B DW 1

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

    MOV AX,5
    IMUL A
    MOV A,AX

    MOV AX,12
    IMUL B
    SUB A,AX
    MOV CX,A

  ;DOS EXIT
    MOV AH,4CH
    INT 21H
      MAIN ENDP
END MAIN
```

6.9.3 Output

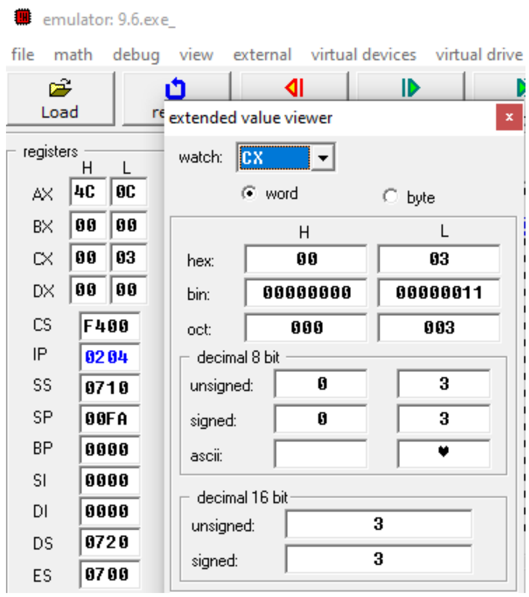


Figure 6.11: Signed Multiplication, 5x3-12x1=3

6.10 Problem G

Write a procedure FACTORIAL that will compute N! for a positive integer N. The procedure should receive N in CX and return N! in AX. Suppose that overflow does not occur.

6.10.1 Steps

1. At first the code format was written
2. 5 was taken in AX
3. 4 was taken in CX as loop count
4. 'LABEL' loop was continued for 4 loops
5. finally the output of 5 factorial was stored in AX

6.10.2 Program

Program 6.12: code for Program G

```
;9.7
.MODEL SMALL
```

```
.STACK 100H
.DATA
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;OPERATION
    MOV AX,5  ;TAKE 5  SO 5! WILL BE 5X4X3X2X1=120
    MOV CX,4  ; 4 LOOPS IS NEEDED FOR 5!
    MOV BX,AX
    SUB BX,1   ;5-1 =4

    LABEL:
    MUL BX   ; MULTIPLY 5 WITH 4,3,2,1
    SUB BX,1
    LOOP LABEL

END MAIN
```

6.10.3 Output

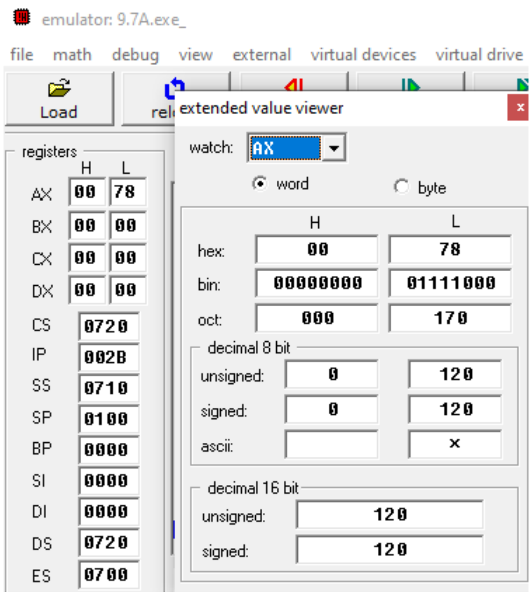


Figure 6.12: FACTORIAL OF 5

6.11 Problem H

Supp0se DX contains OOOOh, AX contains OOOSh, and BX contains 0002h. Do Division

6.11.1 Steps

1. At first the code format was written
2. for the signed division 'IDIV' was used and for the unsigned division 'DIV' was used.

6.11.2 Program

Program 6.13: code for Program H

```
;EXM 9.8
MOV AX,0005H


MOV BX,0002H
DIV BX
```

6.11.3 Output

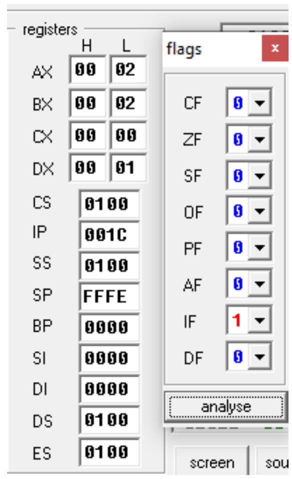


Figure 6.13: Division

6.12 Problem I

6.12.1 Steps

1. At first the code format was written
2. for the signed division 'IDIV' was used and for the unsigned division 'DIV' was used.

6.12.2 Program

Program 6.14: code for Program I

```
;EXM 9.9
MOV AX,0005H
MOV BX,0FFFEH
IDIV BX
```

6.12.3 Output

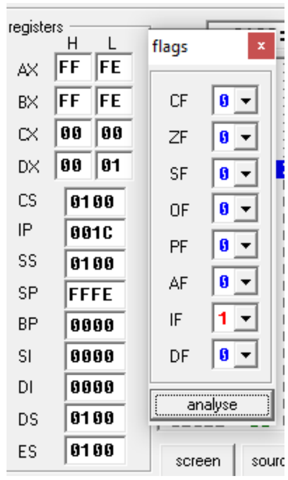


Figure 6.14: Division

6.13 Problem J

Suppose DX contains FFFFh, AX contains FFFBh, and BX contains 0002.

6.13.1 Steps

1. At first the code format was written
2. for the signed division 'IDIV' was used and for the unsigned division 'DIV' was used.

6.13.2 Program

Program 6.15: code for Program j (Signed)

```
;EXAMPLE 9.10
MOV DX, 0FFFFH
MOV AX, 0FFFBH
MOV BX, 0002H
;FOR SIGNED DIVISION
IDIV BX
```

Program 6.16: code for Program j (Unsigned)

```
;EXAMPLE 9.10
MOV DX, 0FFFFH
```

MOV AX, 0FFFBH

MOV BX, 0002H

;FOR UNSIGNED DIVISION

DIV BX

6.13.3 Output

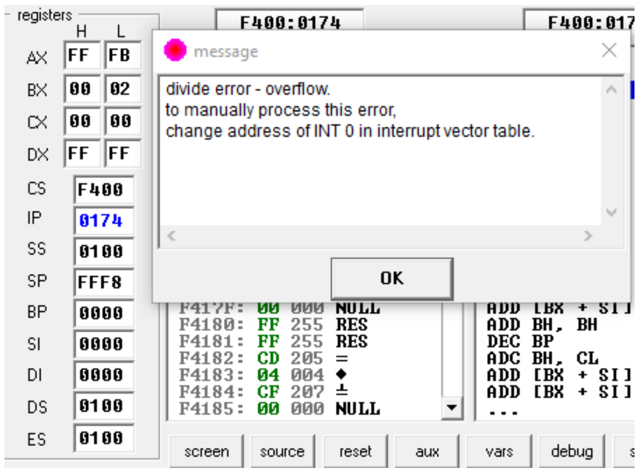


Figure 6.15: Unsigned Division

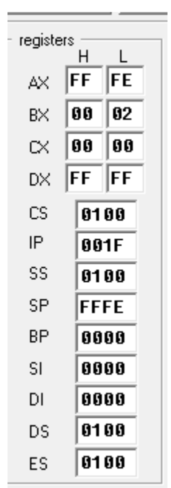


Figure 6.16: signed Division

6.14 Problem K

Divide -1250 by 7

6.14.1 Steps

1. At first the code format was written
2. CWD command was used to extend sign to DX

6.14.2 Program

Program 6.17: code for Program K

```
MOV AX, -1250
CWD
MOV BX, 7
IDIV BX
```

6.14.3 Output

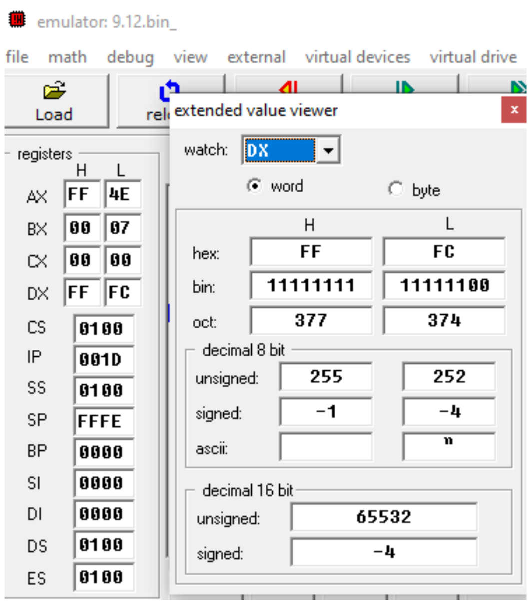


Figure 6.17: Division with Sign Extension

6.15 Problem L

-Divide the signed value of the byte variable XBYTE by -7.

6.15.1 Steps

1. At first the code format was written
2. A variable Xbyte was taken and filled with value
3. CBW was used to extend sign in byte division in AH

6.15.2 Program

Program 6.18: code for Program L

```
;EX 9.13
.MODEL SMALL
.STACK 100H
.DATA
 XBYTE DB 0E8H
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    MOV AL,XBYTE
    CBW
    MOV BL,-7
    IDIV BL
    MAIN ENDP
END MAIN
```

6.15.3 Output

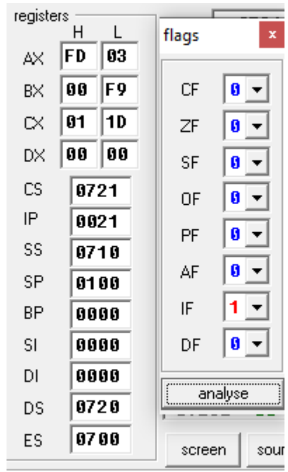


Figure 6.18: Division with Sign Extension USING CBW

6.16 Conclusions and Discussions

The experiment was about multiplication and division. From the experiments, it was seen that for multiplication operation there was a change in flag registers but for division operation, there was no change in flag registers. In word division, the answer was stored in AX and the remainder was stored in DX.And in byte form multiplication answer was stored in AX and in word form multiplication answer was stored in DX.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 7

Experimental Study on Flow Control Instructions

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 12/11/2022

Date of Submission : 26/11/2022

(Teacher's Section)

Report	Signature	Viva
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good		☐ Good
☐ Average		☐ Average
☐ Poor		☐ Poor

Experiment 7

Experimental Study on Flow Control Instructions

7.1 Objectives

The main objectives of this experiment are

- To learn to work with Flow Control Instructions
- To learn to work with Jump command

7.2 Introduction

Conditions in assembly language control the execution of loops and branches. The program evaluates the conditional instruction and executes certain instructions based on the evaluation. The CMP and JMP instructions implement conditional instructions.The CMP instruction takes two operands and subtracts one from the other, then sets O, S, Z, A, P, and C flags accordingly. CMP discards the result of subtraction and leaves the operands unaffected.The JMP instruction transfers the program control to a new set of instructions indicated by the label in the instruction. The conditional jump instructions evaluate if the condition is satisfied through flags, then jump to the label indicated in the instruction. Some Jump Instructions are the following-

Instruction	Description	Data Type	Flags
JE/JZ	Jump equal/Jump zero	Signed/Unsigned	ZF
JNE/JNZ	Jump not equal/Jump not zero	Signed/Unsigned	ZF
JG/JNLE	Jump greater/Jump not less or equal	Signed	OF, SF, ZF
JGE/JNL	Jump greater or equal /Jump not less	Signed	OF, SF
JL/JNGE	Jump less /Jump not greater or equal	Signed	OF, SF
JAE/JNB	Jump above or equal /Jump not below	Unsigned	CF

7.3 Required Software

1. EMU8086

7.4 Problem A

suppose AX and BX contain signed numbers. Write some code to put the biggest one in cx

7.4.1 Steps

1. At first the code format was written
2. First signed number was put in AX , then moved to CX
3. Second signed number was put in BX
4. BX and CX was compared
5. CX<=BX then then the program would terminate
6. otherwise the the biggest item would be in CX

7.4.2 Program

Program 7.1: code for Program 6.1

```
;EX 6.1
;suppose AX aml BX contain signcd numbers. Write some
;code to put the biggest one in cx..
.MODEL SMALL
.STACK 100H
.DATA
A DB ?
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;OPERATION
    MOV AX,   0001H   ;255
```

```
MOV CX,AX
MOV BX,0002H  ;128

CMP BX,CX
 JLE NEXT  ;IF A<=B
MOV CX,BX



    NEXT:
;DOS EXIT
    MOV AH,  4CH
    INT 21H
   MAIN ENDP
END MAIN
```

7.4.3 Output

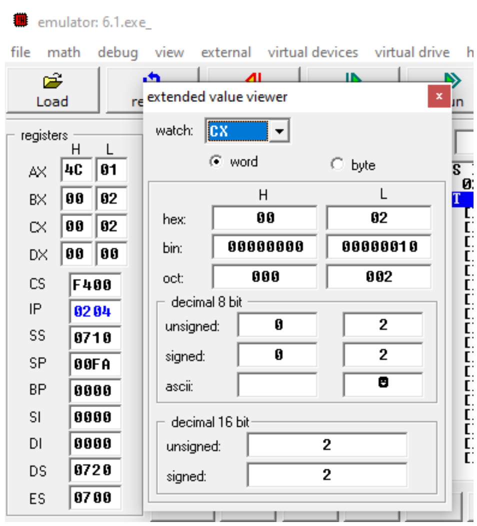


Figure 7.1: Output of 6.1

7.5 Problem B

Replace.the number in AX by its absolute value

7.5.1 Steps

1. At first the code format was written
2. First signed number was put in AX
3. AX is compared with 0
4. if AX is not less than 0 then the program will end
5. if Ax is less than 0 then the item will be made Negative

7.5.2 Program

Program 7.2: code for Program 6.2

```
;EX 6.2

.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;OPERATION
    MOV AX,0FFFFH    ;-1
    CMP AX,0
    JNL END_IF    ;IF AX>0
    NEG AX
    MOV CX,AX

    END_IF:
```

```
      MOV BX,AX
  ;DOS EXIT
         MOV AH, 4CH
         INT 21H
       MAIN ENDP
END MAIN
```

7.5.3 Output

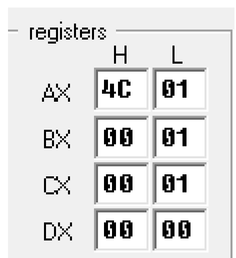


Figure 7.2: Output of 6.2

7.6 Problem C

Display the one that comes first in the character sequence

7.6.1 Steps

1. At first the code format was written
2. A and B two variables were declared
3. A and B were compared, the first item was displayed

7.6.2 Program

Program 7.3: code for Program 6.3

```
;6.3
;Suppose AL and BL contain extended ASCII characters.
;Display the one that comes first in the character sequence.
```

```
.MODEL SMALL
.STACK 100H
.DATA
M1 DB 'TWO␣ASCII␣CHARACTERS:␣$'
A DB 0
B DB 0
M2 DB 0AH,0DH,'␣FIRST:␣$'
.CODE
MAIN PROC
     MOV AX,@DATA
     MOV DS,AX

     LEA DX,M1
     MOV AH,9
     INT 21H

     MOV AH,1
     INT 21H
     MOV A,AL
     MOV AH,1
     INT 21H
     MOV B,AL

     LEA DX,M2
     MOV AH,9
     INT 21H

     MOV BL,B
     CMP A,BL
     JL DISPLAY
```

```
JMP ELS

DISPLAY:
    MOV AH,2
    MOV DL,A
    INT 21H
    MOV AH,4CH
    INT 21H
ELS:
    MOV AH,2
    MOV DL,B
    INT 21H
    MOV AH,4CH
    INT 21H

MAIN ENDP
END MAIN
```

7.6.3 Output

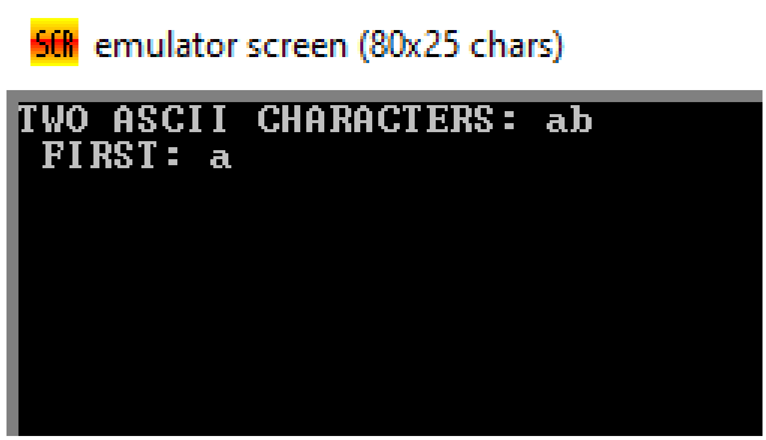


Figure 7.3: Output of 6.3

7.7 Problem D

If AX contains a negative number, put −1 In BX; if AX ; contains 0, put O In BX; if AX contains a positive number, put 1 In BX

7.7.1 Steps

1. At first the code format was written
2. Input was taken in AX
3. Ax was compared with 0
4. if Ax was greater than 0, Positive JUMP occurred, and BX contains 1 ,if AX contains less than 0 then NEGATIVE jump occurred and BX contains −1 otherwise BX contains 0

7.7.2 Program

Program 7.4: code for Program 6.4

```
;EX 6.4
;If AX contains a negative number, put -1 In BX; if AX
;contains 0, put O In BX; if AX contains a positive number, put 1 In BX
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
    ;INITIALIZE DS
   MOV AX,@DATA
   MOV DS,AX
    ;OPERATION
   MOV AX,0
   CMP AX,0
    JG POSITIVE
```

```
        JL  NEGATIVE
       MOV  BX, 0
       JMP  END_IF
        POSITIVE:
       MOV  BX, 1
       JMP  END_IF
        NEGATIVE:
       MOV  BX, -1
       JMP  END_IF
    ;DOS  EXIT
   END_IF:
           MOV  AH,   4CH
           INT  21H
         MAIN  ENDP
END  MAIN
```

7.7.3 Output

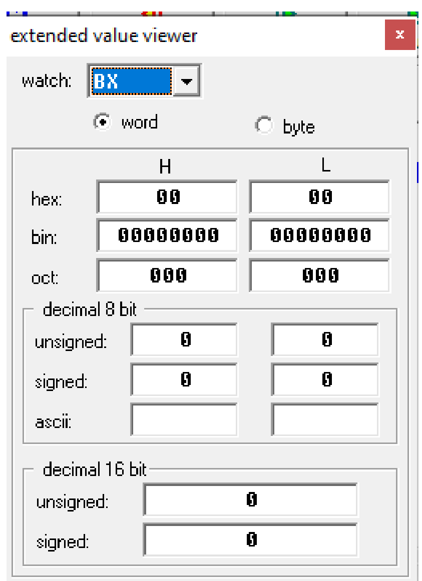


Figure 7.4: Output of 6.4

7.8 Problem E

If AL contains 1 or 3 , display " o "; if AL contains 2 or 4, display " e "

7.8.1 Steps

1. At first the code format was written
2. 1,3,2,4 was compared with AL
3. if 1 and 3 was matched with AL 'O' will be printed otherwise 'E" will be printed.

7.8.2 Program

Program 7.5: code for Program 6.5

```
;EX 6.5
;If AL contains I or 3, display "o"; if AL contains 2 or 4, display "e".
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
    ;INITIALIZE DS
   MOV AX,@DATA
   MOV DS,AX
    ;OPERATION
   MOV AH,1
   INT 21H
   SUB AL, 30H
   CMP AL, 2
   JE EVEN
   CMP AL,4
   JE EVEN
```

```
    CMP AL,1
    JE ODD
    CMP AL,3
    JE ODD
    JMP END_IF

EVEN:
    MOV CL,'e'
    JMP END_IF

ODD:
   MOV CL,'o'
   JMP END_IF
 ;DOS EXIT
END_IF:
        MOV AH, 2
        MOV DL, 0DH
        INT 21H
        MOV DL, 0AH
        INT 21H
        MOV DL, CL
        MOV AH, 2
        INT 21H
        MOV AH, 4CH
        INT 21H

      MAIN ENDP
END MAIN
```

7.8.3 Output

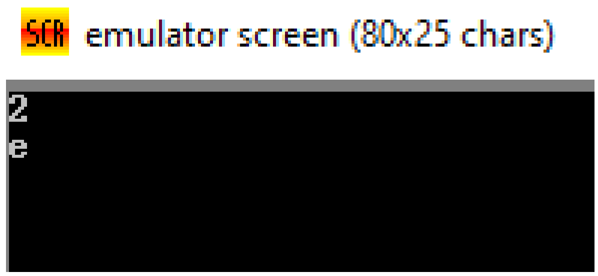


Figure 7.5: Output of 6.5

7.9 Problem F

Read a character, and if it's an uppercase letter, display it

7.9.1 Steps

1. At first the code format was written
2. an input was taken
3. if the input is in the range of A to Z ascii characters then it would be displayed

7.9.2 Program

Program 7.6: code for Program 6.6

```
;EX 6.6
;Read a character, and if it's an uppercase letter, display it.
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;OPERATION
```

```
MOV AH, 1
INT 21H
CMP AL, 'A'
JNGE END_IF
CMP AL, 'Z'
JNLE END_IF
MOV DL, AL
MOV AH, 2
INT 21H
;DOS EXIT
END_IF:
MOV AH, 4CH
INT 21H
MAIN ENDP
END MAIN
```

7.9.3 Output

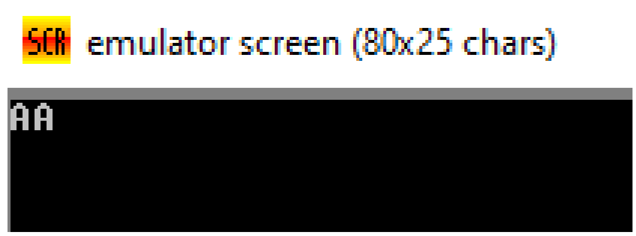


Figure 7.6: Output of 6.6

7.10 Problem G

Read a character. If it ' s " y " or "Y " , display it ; otherwise terminate the program

7.10.1 Steps

1. At first the code format was written
2. a string was displayed which gives instructions to get an input

3. the input was compared with 'y' and 'Y'

4. then the output data is displayed with a combination of string

7.10.2 Program

Program 7.7: code for Program 6.7

```
;6.7
;Read a character. If it's "y" or "Y", display it; otherwise terminate the program
.MODEL SMALL
.STACK 100H
.DATA
M1 DB 'Read: $'
M2 DB 0AH,0DH, 'If The letter is y or Y Then: '
A DB 0,'$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    LEA DX,M1
    MOV AH,9
    INT 21H

    MOV AH,1
    INT 21H
    MOV A,AL

    CMP A,'y'
    JE DISPLAY
    CMP A,'Y'
```

```
    JE  DISPLAY
    JMP  EXIT

    DISPLAY:
        LEA  DX,M2
        MOV  AH,9
        INT  21H
        JMP  EXIT
    EXIT:
        MOV  AH,4CH
        INT  21H
MAIN  ENDP
END  MAIN
```

7.10.3 Output

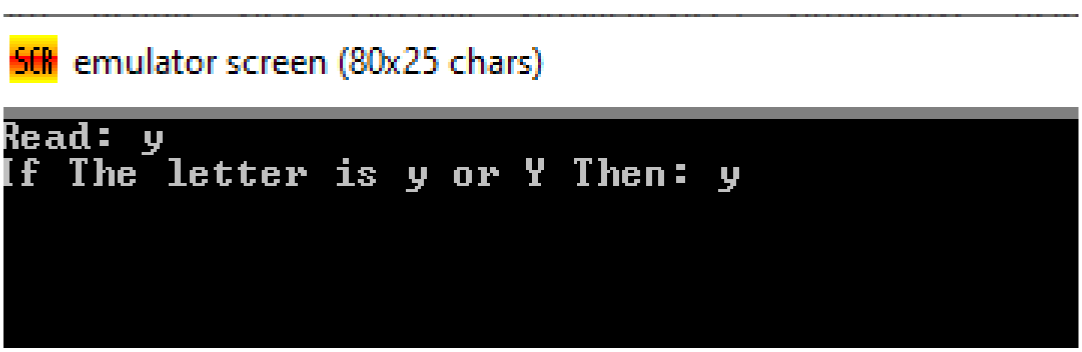


Figure 7.7: Output of 6.7

7.11 Problem H

Write a count-controlled loop to display a row of 80 stars

7.11.1 Steps

1. At first the code format was written

2. the input is taken and this data is stored in a memory byte and some procedures are taken to get the desired output.

3. the ASCII code of * was stored in M3 and CL was stored with 80D to write a loop for 80 times.

7.11.2 Program

Program 7.8: code for Program 6.8

```
;6.8
;Write a count-controlled loop to display a row of 80 stars.

.MODEL SMALL
.STACK 100H
.DATA
M1 DB '80 STARS IN A ROW: '
M2 DB 0AH,0DH,'$'
M3 DB '*$'
.CODE
MAIN PROC
     MOV AX,@DATA
     MOV DS,AX
     MOV CL,80D
     LEA DX,M1
     MOV AH,9
     INT 21H
     CMP CL,0
     JNE DISPLAY
     DISPLAY:
          MOV AH,2
          MOV DL,M3
```

```
        INT  21H
        JMP  THEN
    THEN:
        DEC  CL
        CMP  CL, 0
        JE   EXIT
        JMP  DISPLAY
    EXIT:
        MOV  AH, 4CH
        INT  21H
MAIN ENDP
END MAIN
```

7.11.3 Output

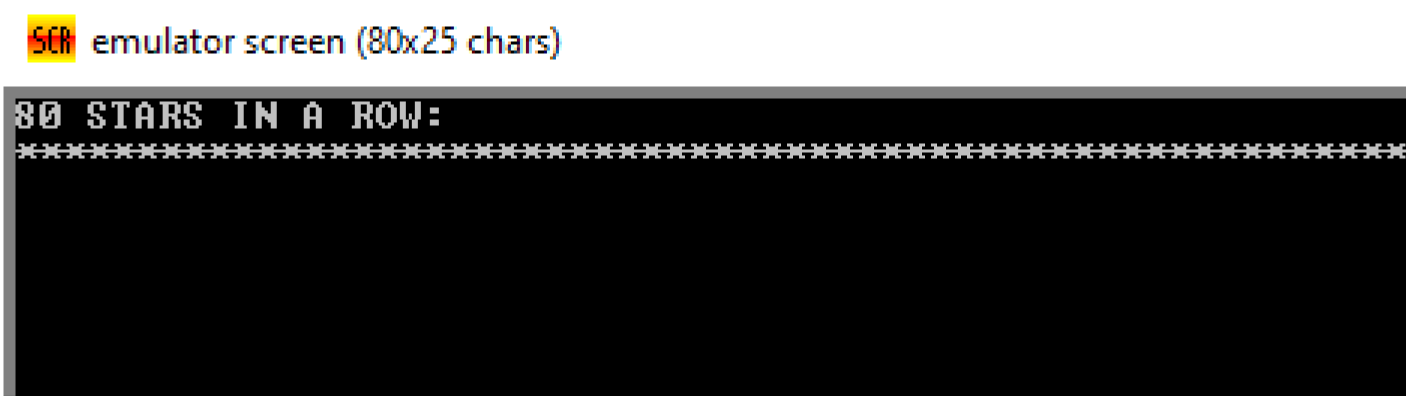


Figure 7.8: Output of 6.8

7.12 Problem I

Write some code to count the number of characters In; n input line

7.12.1 Steps

1. At first the code format was written

2. After that the input is taken by a loop and through this loop the count of the input is taken and this data is stored in a memory byte

3. the output data is displayed by computing the number of characters.

7.12.2 Program

Program 7.9: code for Program 6.9

```
;6.9
;Write some code to count the number of charaters In; n  input line.
.MODEL SMALL
.STACK 100H
.DATA
M1 DB 'THE NUMBER OF DIGIT IS : '
M2 DB 0
M3 DB '$'
M4 DB 'ENTER SERIES OF CHARACTERS: '
M5 DB 0AH,0DH, '$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    LEA DX,M4
    MOV AH,9
    INT 21H
    MOV CL,0
    MOV AH,1
    INT 21H
    WHILE_:
        CMP AL,0DH
        JE EXIT
```

```
        INC  CL
        INT  21H
        JMP  WHILE_
    EXIT:
        ADD  CL,30H
        MOV  M2,CL
        LEA  DX,M1
        MOV  AH,9
        INT  21H
MAIN ENDP
END MAIN
```

7.12.3 Output

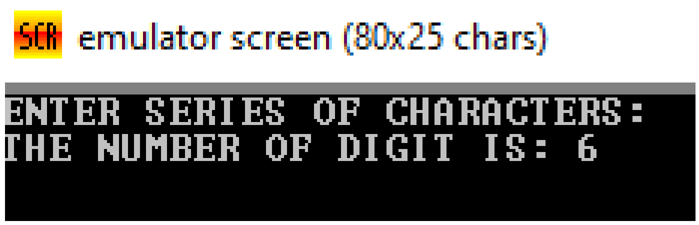


Figure 7.9: Output of 6.9

7.13 Problem J

Write some code to read characters until a blank is read

7.13.1 Steps

1. At first the code format was written
2. displaying a string that gives instructions to get an input
3. a REPEAT loop was created to Compare input with ' '.If a blank is detected then the output will be shown

7.13.2 Program

Program 7.10: code for Program 6.10

```
;6.10
;Write some code to read characters until a blank is read
.MODEL SMALL
.STACK 100H
.DATA
M1 DB 'SERIES␣OF␣CHARACTERS:'
M2 DB 0AH,0DH,'$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    LEA DX,M1
    MOV AH,9
    INT 21H
    MOV AH,1
    REPEAT:
        INT 21H
        CMP AL,'␣'
        JNE REPEAT
MAIN ENDP
END MAIN
```

7.13.3 Output

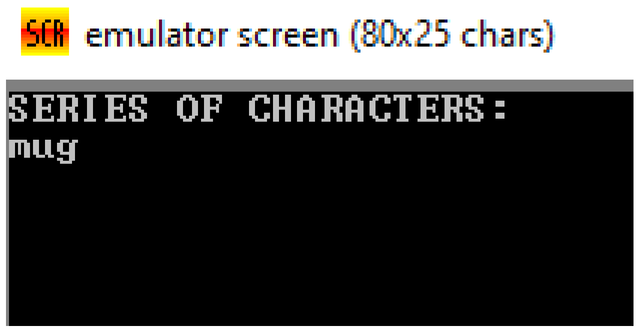


Figure 7.10: Output of 6.10

7.14 Conclusions and Discussions

This experiment demonstrated the operation of flow control instructions and how to use them while writing 8086 assembly code.

The algorithm of the flow control instruction reveals the true structure by mixing jump syntax. The events of the 8086 go off without a hitch. As a result, the output of the operations as well as the status of the registers were found. So, the experiment was successful.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 9

Experimental Study on Input Storing Using PUSH and POP

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 26/11/2022

Date of Submission : 03/12/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	____________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 8

Experimental Study on Input Storing Using PUSH and POP

8.1 Objectives

The main objectives of this experiment are

- To learn to take single inputs with multiple characters
- To learn to the addition of two single inputs with multiple characters
- To learn to convert Celcius to Fahrenheit

8.2 Introduction

The 8086 uses a simple stack in memory for the storage of temporary data. It also uses this stack to store the return addresses when it enters a new procedure. All values on the stack are 16-bit words. The registers that manage the stack are SS, SP and BP.

- SS denotes the segment of the stack
- SP (stack pointer) points to the last element on top of the stack
- BP (base pointer) points to the bottom of the stack. This is used to set up and manage information on the stack during a procedure call.

There are a few commands which allow the programmer to store and retrieve values from the stack such as -The PUSH/POP instructions.PUSH decrements the SP register (by 2) and copies a value onto the top of the stack. POP retrieves the value from the top of the stack and stores it in the destination, then increments the SP register (by 2). PUSH and POP can be used to save and restore the values of registers when the register needs to be used for some other function temporarily.

8.3 Required Software

1. EMU8086

8.4 Problem A

Take two or more inputs to display.

8.4.1 Steps

1. At first the code format was written
2. Then two variable P and R are taken
3. 0 was moved in BX
4. Input was taken continuously until a 'SPACE' is found
5. input was substracted by 30h to get exact decimal value
6. NUM operation was applied to make the input together
7. then by applying PUSH and POP operation input was printed

8.4.2 Program

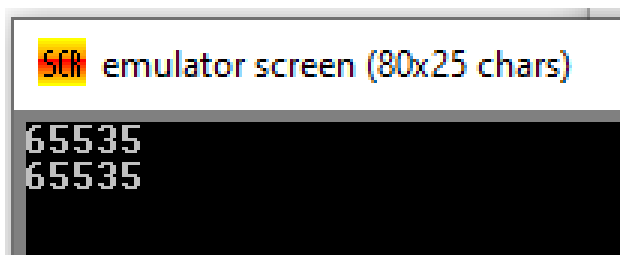
Program 8.1: code for taking two or more inputs

```
.DATA
P DW 10
R DW 0
.CODE
MAIN PROC
     ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
     ;OPERATION
    MOV BX,0
INPUT:
    MOV AH,1
```

```
        INT  21H
        CMP  AL,13D
        JNE NUM
        JMP  STOP
NUM:
        SUB  AL,30H
        MOV CL,AL
        MOV CH,0
        MOV AX,BX
        MUL  P
        ADD  AX,CX
        MOV  BX,AX
        JMP  INPUT
STOP:
        MOV DX,0
        MOV AX,BX
        DIV  P
        INC  R
        PUSH  DX
        CMP  AX,0
        JE  S1
        MOV BX,AX
        JMP  STOP

S1:
        MOV CX,R
        MOV AH,2
        MOV DL,0DH
        INT  21H
        MOV DL,0AH
```

```
        INT  21H
S2:
        POP DX
        ADD DL,30H
        MOV AH,2
        INT  21H
        LOOP  S2
MAIN ENDP
END MAIN
```

8.4.3 Output



8.5 Problem B

Add two multiple inputs

8.5.1 Steps

1. At first the code format was written
2. Then two variable P and R are taken
3. 0 was moved in BX
4. Input was taken continuously until a 'SPACE' is found
5. input was substracted by 30h to get the exact decimal value
6. NUM operation was applied to make the input together

7. then by applying PUSH and POP operation input was printed
8. same procedure was mae to enter another input
9. addition of two inputs were done by using ADD command

8.5.2 Program

Program 8.2: code for addition of two multiple inputs

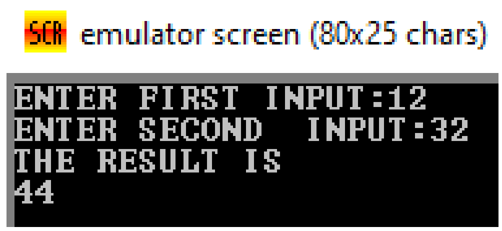
```
.DATA
P DW 10
R DW 0
X DW 0
MSG1 DB "ENTER FIRST INPUT:$"
MSG2  DB 0DH,0AH, 'ENTER SECOND  INPUT:$'
RESULT DB 0DH,0AH, "THE RESULT IS$ "
.CODE
MAIN PROC
    ;INITIALIZE DS
    MOV AX,@DATA
    MOV DS,AX
    ;OPERATION
    LEA DX,MSG1
    MOV AH,9
    INT 21H
    MOV BX,0
    MOV BX,0
INPUT1:
    MOV AH,1
    INT 21H
    CMP AL,13D
    JNE NUM1
```

```
    JMP FIRST
NUM1:
    SUB AL,30H
    MOV CL,AL
    MOV CH,0
    MOV AX,BX
    MUL P
    ADD AX,CX
    MOV BX,AX
    JMP INPUT1

FIRST:
    MOV X,BX

LEA DX,MSG2
MOV AH,9
INT 21H
MOV BX,0

INPUT2:
    MOV AH,1
    INT 21H
    CMP AL,13D
    JNE NUM2
    JMP SECOND
NUM2:
    SUB AL,30H
    MOV CL,AL
    MOV CH,0
    MOV AX,BX
```

```
        MUL P
        ADD AX,CX
        MOV BX,AX
        JMP INPUT2

SECOND:
        ADD BX,X

LEA DX,RESULT
MOV AH,9
INT 21H

STOP:
        MOV DX,0
        MOV AX,BX
        DIV P
        INC R
        PUSH DX
        CMP AX,0
        JE S1
        MOV BX,AX
        JMP STOP

S1:
        MOV CX,R
        MOV AH,2
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H
```

```
S2:
    POP DX
    ADD DL,30H
    MOV AH,2
    INT 21H
    LOOP S2
MAIN ENDP
END MAIN
```

8.5.3 Output



8.6 Problem C

Take a celcius input and display the output as Fahrenheit

8.6.1 Steps

1. At first the code format was written
2. Then two variable P and R are taken
3. 0 was moved in BX
4. Input was taken continuously until a 'SPACE' is found
5. input was substracted by 30h to get exact decimal value
6. NUM operation was applied to make the input together
7. then by applying PUSH and POP operation input was printed
8. the equation of Fahrenheit conversion was applied to the input to get the output

8.6.2 Program

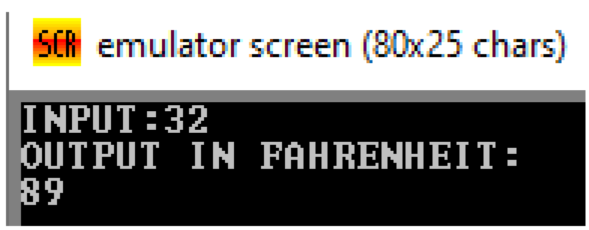
Program 8.3: code for Fahrenheit showing

```
.MODEL SMALL
.STACK 100H

.DATA
P DW 10
R DW 0
K DW 9
J DW 5
MSG DB 'INPUT:$'
MSG2 DB 0DH,0AH,'OUTPUT IN FAHRENHEIT:$'
.CODE
MAIN PROC
 MOV AX,@DATA
 MOV DS,AX
 MOV BX,0
LEA DX,MSG
MOV AH,9
INT 21H
INPUT:
     MOV AH,1
     INT 21H
     CMP AL,13D
     JNE NUM
     JMP FAR
NUM:
     SUB AL,30H
     MOV CL,AL
```

```
    MOV CH,0
    MOV AX,BX
    MUL P
    ADD AX,CX
    MOV BX,AX
    JMP INPUT


FAR:
    MOV AX,BX
    MUL K
    DIV J
    ADD AX,32
    MOV BX,AX

LEA DX,MSG2
MOV AH,9
INT 21H

STOP:
  MOV DX,0
  MOV AX,BX
  DIV P
  INC R
  PUSH DX
  CMP AX,0
  JE S1
  MOV BX,AX
  JMP STOP
```

```
S1:
   MOV CX,R
   MOV AH,2
   MOV DL,0DH
   INT 21H
   MOV DL,0AH
   INT 21H
S2:
   POP DX
   ADD DL,30H
   MOV AH,2
   INT 21H
   LOOP S2

   MAIN ENDP
END MAIN
```

8.6.3 Output



8.7 Conclusions and Discussions

The experiment is about advanced operation using POP and PUSH. By using POP and PUSH it was seen that single input of multiple characters was taken. This is the method to insert this type of input using assembly language. The experiment was successful

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 9

Experimental Study on Displaying and Reversing String Operations

---

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 03/12/2022

Date of Submission : 10/12/2022

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	____________________	☐ Good
☐ Average	Signature	☐ Average
☐ Poor		☐ Poor

Experiment 9

Experimental Study on Displaying and Reversing String Operations

9.1 Objectives

The main objectives of this experiment are

- To learn to read string inputs and print it
- To learn to read multiple string inputs and reverse it
- To learn to read string inputs and use backspace to change it

9.2 Introduction

A "string" is typically just a group of letters that can be used as a literal constant or a particular variable. In the 8086 assembly language, a memory string is just a string of bytes. or a word cloud. Numerous string operations exist, such as copying one string into another, storing a character in a string, alphabetically comparing strings of characters, and querying a string for a certain byte or word. Moving, storing, and processing have no impact on the flag bits. loading operations, but they are changed by comparison or scanning procedures. two bytes and word operations are carried out in this. They make memory-to-memory operations possible, as well as provide automatic pointer register updating.

9.3 Required Software

1. EMU8086

9.4 Problem A

Read Name and Display it

9.4.1 Steps

1. At first the code format was written

2. Then an empty address S1 , a variable R and a string was declared

3. DI and SI was declared

4. CLD was used to increment and to clear DF

5. Input was taken under label 'INPUT' . For each input R was incremented by 1 and this loop continued untill 'ENTER' was inserted.

6. when 'ENTER' was pressed program jumped to DISPLAY loop

7. In 'DISPLAY' loop the string that was declared before was printed

8. Finally 'TOP' loop was declared to print the outputs.

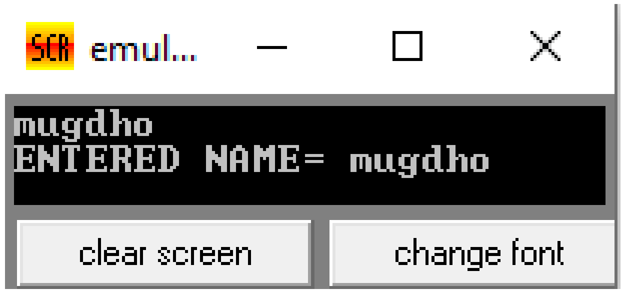
9.4.2 Program

Program 9.1: code for reading name and displaying it

```
.MODEL SMALL
.STACK 100H
.DATA
S1 DB 10 DUP(?)
R DW 0
S2 DB 0AH,0DH, 'ENTERED␣NAME=␣$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV ES,AX
    MOV DS,AX

    LEA DI,S1
    CLD

INPUT:
```

```
        MOV AH,1
        INT 21H
        INC R
        STOSB
        CMP AL,0DH
        JNE INPUT
        JMP DISPLAY

DISPLAY:
        MOV AH,9
        LEA DX,S2
        INT 21H
        MOV CX,R
TOP:
        LEA DI,S1
        CLD
        LODSB
        MOV AH,2
        MOV DL,AL
        INT 21H
        LOOP TOP

    MAIN ENDP
END MAIN
```

9.4.3 Output



9.5 Problem B

Read string and Reverse it

9.5.1 Steps

1. At first the code format was written
2. Then an empty address S1 , a variable R and a string was declared
3. DI and SI was declared
4. CLD was used to increment and to clear DF
5. Input was taken under label ’INPUT’ . For each input R was incremented by 1 and this loop continued untill ’ENTER’ was inserted.
6. when ’ENTER’ was pressed program jumped to DISPLAY loop
7. In ’DISPLAY’ loop the string that was declared before was printed
8. Finally ’TOP’ loop was declared to print the outputs.
9. PUSH and POP operation was done in order to reversely print the output.

9.5.2 Program

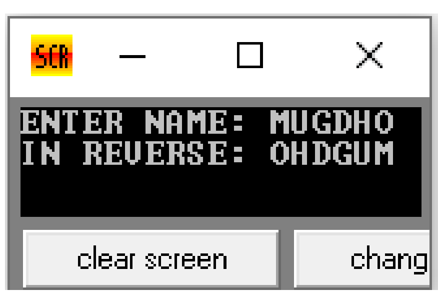
Program 9.2: code for Reverse String

```
.MODEL SMALL
.STACK 100H
.DATA
```

```
M1 DB '␣ENTER␣NAME:␣$'
M2 DB 0AH,0DH,'IN␣REVERSE␣:␣'
M3 DB "$"
S1 DB 10,""

.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    LEA DX,M1
    MOV AH,9
    INT 21H
    LEA DI,S1
    CLD

    MOV BX,0

    INPUT:
        MOV AH,1
        INT 21H
        CMP AL,0DH
        JNE STORE
        LEA DX,M2
        MOV AH,9
        INT 21H
        LEA SI,S1+BX-1
        STD
        MOV CX,BX
        JMP DISPLAY
```

```
    STORE:
        STOSB
        INC BX
        JMP INPUT

    DISPLAY:
        LODSB
        MOV AH,2
        MOV DL,AL
        INT 21H
        LEA DX,M3
        MOV AH,9
        INT 21H
        LOOP DISPLAY

    EXIT:
        MOV AH,4CH
        INT 21H
MAIN ENDP
END MAIN
```

9.5.3 Output



9.6 Problem C

Read String and Use backspace to change previous input

9.6.1 Steps

1. At first the code format was written
2. Then an empty address STRING was declared
3. DS and ES was declared
4. CLD was used to increment and to clear DF
5. Input was taken under label 'INPUT' . 'Backspace' was compared with input.
6. when 'Backspace' was pressed program jumped to FIX loop
7. FIX loop used to return to previous input and let the user to change it

9.6.2 Program

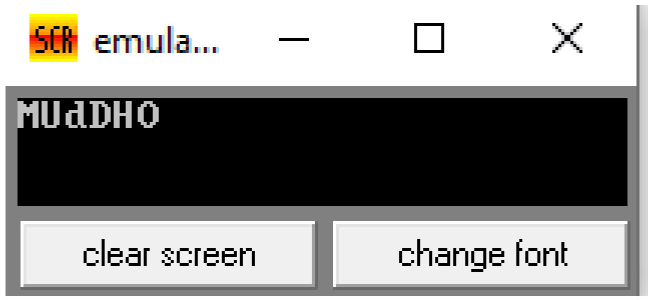
Program 9.3: code for changing previous input using backspace

```
.MODEL SMALL
.STACK 100H
.DATA
STRING DB 100 DUP (0)
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS, AX
    MOV ES, AX
    MOV BX, 0
    LEA DI, STRING ;STRING ADDRESS IS COPIRD INTO DI
    MOV AH,1
```

```
INT 21H
INPUT:
    CMP AL, 13D
    JE NEXT1
    CMP AL, 8H;COMPARING WITH THE BACKSPACE
    JE FIX
    CLD
    STOSB
    INC BX
    INT 21H
    JMP INPUT
FIX:
      DEC DI
      DEC BX
      INT 21H ;AGAIN TAKING INPUT
      JMP INPUT
NEXT1:
    LEA SI, STRING   ;COPING THE ADDRESS OF STRING INTO SI
    MOV CX, BX
    MOV AH, 2
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
    MOV AH, 2
NEXT:
      LODSB
      MOV DL, AL
      INT 21H
      LOOP NEXT
```

```
MOV AH, 4CH
INT 21H
MAIN ENDP
END MAIN
```

9.6.3 Output



9.7 Conclusions and Discussions

The experiment was about various string operations. When writing the code, many string operations were employed. The user entered the name, which was then saved using the STOSB command. then used the LODSB command to load it for display. A string was also kept by the application until the next line was displayed after typing a carriage return. the backspace key was pressed. Upon pressing, the string's preceding character was deleted. The loading process displayed the name as entered, starting at the beginning of the string, and from the end of the string to reverse the order of the name.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

EEE 3254: Sessional Based on EEE 3253

Experiment No. 10

Experimental Study on Writing Specific strings from a Group of Strings

---

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

A. S. M. Badrudduza

Assistant Professor

Dept. of ETE, RUET

Date of Experiment : 10/12/2022

Date of Submission : 17/12/2022

(Teacher's Section)

Report

- ☐ Excellent
- ☐ Very Good
- ☐ Good
- ☐ Average
- ☐ Poor

____

Signature

Viva

- ☐ Excellent
- ☐ Very Good
- ☐ Good
- ☐ Average
- ☐ Poor

Experiment 10

Experimental Study on Writing Specific strings from a Group of Strings

10.1 Objectives

The main objectives of this experiment are

- To learn to write specific strings from a group of strings

10.2 Introduction

A "string" is typically just a group of letters that can be used as a literal constant or a particular variable. In the 8086 assembly language, a memory string is just a string of bytes. or a word cloud. Numerous string operations exist, such as copying one string into another, storing a character in a string, alphabetically comparing strings of characters, and querying a string for a certain byte or word. Moving, storing, and processing have no impact on the flag bits. loading operations, but they are changed by comparison or scanning procedures. two bytes and word operations are carried out in this. They make memory-to-memory operations possible, as well as provide automatic pointer register updating.

10.3 Required Software

1. EMU8086

10.4 Problem A

Read a group of strings and filter output "ARGENTINA"

10.4.1 Steps

1. At first the code format was written
2. Then an empty address S1, a variable R, and a string were declared
3. DI and SI was declared

4. CLD was used to increment and clear DF
5. Input was taken under the label 'INPUT'. For each input R was incremented by 1 and this loop continued until 'ENTER' was inserted.
6. when 'ENTER' was pressed program jumped to the DISPLAY loop
7. Finally 'TOP' loop was declared to print the outputs.

10.4.2 Program

Program 10.1: code for reading string and filtering "ARGENTINA"

```
;Fillter ARGENTINA

.MODEL SMALL
.STACK 100H
.DATA
STRING DB 100 DUP (0)
CLR DB 0DH,0AH,'$'

.CODE
MAIN PROC

    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    MOV BX,0
    LEA DI,STRING
    CLD

INPUT:
    MOV AH,1
```

```
        INT 21H
        INC BX
        CMP AL,13D
        JE DISPLAY
        STOSB
        JMP INPUT

DISPLAY:
        LEA DX,CLR
        MOV AH,9
        INT 21H
        LEA SI,STRING
        CLD
        MOV AH,2
        MOV CX,BX

TOP:
        LODSB
        CMP AL,'A'
        JNGE EN
        CMP AL,'Z'
        JNLE EN
        MOV DL,AL
        MOV AH,2
        INT 21H
        CMP CX,0
        JL EXIT
        LOOP TOP

EN:
```

```
    CMP  CX, 0
    JL   EXIT
    DEC  CX
    JMP  TOP

EXIT:
    MOV  AH, 4CH
    INT  21H
     MAIN ENDP
END MAIN
```

10.4.3 Output

10.5 Conclusions and Discussions

The experiment was about filtering string operations. When writing the code, many string operations were employed. The user entered the name, which was then saved using the STOSB command. then used the LODSB command to load it for display. A string was also kept by the application until the next line was displayed after typing a carriage return. the backspace key was pressed. Upon pressing, the string's preceding character was deleted. The loading process displayed the name as entered, starting at the beginning of the string, and from the end of the string to reverse the order of the name.