**LAB # 3**

**To EXPLAIN and Show the Output of Arithmetic Instructions using EMU8086**

**Software Tool**

**Objectives**

* + To explain and show the output of ADD, SUB, MUL, and DIV Instruction using emu8086 software tool.
  + To explain and show the output of INC, DEC, and NEG Instruction using emu8086 software tool.

# Pre-Lab Exercise

Read the details given below in order to comprehend the basic operation of Arithmetic instructions. Arithmetic instructions incudes addition, subtraction, multiplication and division. Study in detail and become familiar with the various ways and combinations in which these instructions can be used.

# ADD Instruction

ADD instruction adds values exiting in two registers, register and memory, immediate and memory, immediate and register. To add two values existing in AL and BH register, ADD is used as:

**ADD AL, BH**

This instruction adds the contents of AL and BH registers, leaving sum in AL and BH unchanged. ADD is applied, for example, on CX and DX in the same way. Consider CX is containing 18 and DX containing 25 before executing the following instruction.

**ADD CX, DX**

After this instruction is executed, CX is left with 43, DX with 25 and CF with 0.

Any 8-bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with ADD. For 16-bit addition, ADD can be applied on **AX, BX, CX, DX, SI** and **DI.** You are encouraged to check the validity of ADD instruction for another 16-bit registers yourself.

# ADC Instruction

**ADC**, pronounced as Add with Carry, instruction adds the content of operands along with value in carry flag that exists in it at the instant instruction **ADC** is executed. Consider **SI** contains **15678**, **DI** contains **325** and **CF** is set. After executing following instruction,

**ADC SI, DI**

**SI** is left with **16004** (**3E84H**), **DI** with **325 (145H**) and **CF** is cleared as the latest result does not produce overflow condition.

# SUB Instruction

SUB instruction subtracts the contents of operand2 from operand1 and leaves the difference in operand1, with operand2 unchanged.

**SUB operand1, operand2**

Consider content of AH are to be subtracted from content of DL then SUB can be used as,

**SUB DL, AH**

For 16-bit subtraction,

**SUB DX, DI**

**SUB BX, AX**

Any 8-bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with SUB. For 16-bit subtraction, SUB can be applied on **AX, BX, CX, DX, SI** and **DI**. You are encouraged to check the validity of SUB instruction for another 16-bit registers yourself.

# SBB Instruction

SBB, pronounced as Subtract with Borrow, subtracts the contents of operand2 and CF from the content of operand1 and leaves the result in operand1. Operand2 is left unchanged, and CF depends upon whether most significant bit in operand1 required a borrow bit or not. If borrow was required CF is set, otherwise cleared. Format of SBB is,

**SBB operand1, operand2**

Consider CF is set, BX = 67ABH and DX = 100H before executing following instruction.

**SBB BX, DX**

After execution, BX is left with 66AAH, DX with 100H and CF is cleared.

# INC Instruction

Increment instruction INC adds 1 to the contents of its operand. It can be applied on any 8-bit register (**AH, AL, BH, BL, CH, CL, DH, DL**). For 16-bit increment, INC can be applied on AX, BX, CX, DX, SI and DI. You are encouraged to check the validity of INC instruction for other 16-bit registers yourself.

Consider AL contains 97H before execution of following intrusion.

**INC AL**

After execution, AL is left with 98H.

# DEC Instruction

Decrement instruction DEC subtracts 1 from the contents of its operand. It can be applied on any 8- bit register (AH, AL, BH, BL, CH, CL, DH, DL). For 16-bit decrement, DEC can be applied on AX, BX, CX, DX, SI and DI. You are encouraged to check the validity of DEC instruction for other 16- bit registers yourself.

Consider **DI contains 55968** before execution of following intrusion.

**DEC DI ; after execution,**

**DI is left with 55967.**

**NEG Instruction**

**NEG instruction** is single operand instruction that leaves the operand with **2’s complement** of its original data. For example, register AX contains 12656 (3170H). After executing NEG on AX, new contents of AX will be CE90H which is -12656. NEG can be applied on 8-bit registers, 16-bit registers or memory location. Consider,

**NEG AX**

**MUL Instruction**

MUL carry out multiplication on two operands in 8086-88 CPU. MUL is a single operand instruction whereas other operand is assumed to be in a specified register. MUL can be applied on two 8-bit values producing 16-bit result, and on 16-bit values producing 32-bit result.

For 8-bit multiplication, one operand is assumed in AL register whereas other is the part of instruction. For example, in order to multiply 230 with 165, one of these values must be in AL register. Other operand can be in any 8-bit register or in memory location. Consider following code.

**MOV AL, 230**

**MOV BL, 165**

**MUL BL**

After executing these three instructions, 16-bit result (37950 in this case) will be found in AX register, while content of BL is left unchanged.

For 16-bit multiplication, one operand is assumed in AX register whereas other is the part of instruction. For example, in order to multiply 22330 with 10365, one of these values must be in AX register. Another operand can be in any 16-bit register. Consider following code.

**MOV AX, 22330**

**MOV BX, 10365**

**MUL BX**

After executing these three instructions, 32-bit result (231450450 in this case) will be found in DX- AX registers. DX register contains most significant 16 bits (from bit 16 to bit 31) and AX contains least significant 16 bits (from bit 0 to bit 15). Content of BX are left unchanged.

# DIV Instruction

To do division in 8086-88, DIV instruction is provided by the instruction set of 8086-88 CPU. Like MUL, DIV can be done on 8-bit data and on 16-bit data. In 8-bit division, dividend (numerator) is stored in AX register while 8-bit divisor (denominator) is stored in 8-bit register or in memory. After executing DIV, 8-bit quotient moves in AL while 8-bit remainder moves in AH.

Consider the following code that divides 2334 by 167.

**MOV AX, 2334**

**MOV BH, 167**

**DIV BH**

After executing above three instructions, AL contains DH (13 decimal, the quotient) and AH contains A3H (163 decimal, the remainder).

In 16-bit division, 32-bit dividend (numerator) is stored in DX-AX registers such that most significant 16 bits are in DX and least significant 16 bits are in AX. 16-bit divisor (denominator) is stored in a register. After executing DIV, 16-bit quotient moves in AX while 16-bit remainder moves in DX.

Consider the following code that divides 235634 (39872H) by 45667 (B263H).

|  |  |  |
| --- | --- | --- |
| **MOV** | **AX,** | **9872H** |
| **MOV** | **DX,** | **3H** |
| **MOV** | **BX,** | **0B263H** |
| **DIV** | **BX** |  |

After executing above three instructions, AX contains 5H and DX contains 1C83H.

# Note:

An interrupt is occurred if division is achieving such that ‘divided by zero’ is carried out or a very large number is divided by a very small number that cause quotient greater than the range of register.

# In-Lab Exercise

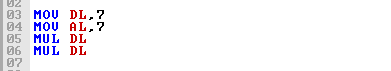
**Task 1: Fill the table after compiling the code**

Type in the following program in editor of EMU8086 and fill in the table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **AH** | **AL** | **BH** | **BL** | **DH** | **DL** | **CF** |
| **MOV AX,1456** | 05 | B0 | 00 | 00 | 00 | 00 | 0 |
| **MOV BX,238** | 05 | B0 | 00 | EE | 00 | 00 | 0 |
| **ADD AX, BX** | 06 | 9E | 00 | EE | 00 | 00 | 0 |
| **SUB AX,134** | 06 | 18 | 00 | EE | 00 | 00 | 0 |
| **MOV AL,33H** | 06 | 33 | 00 | EE | 00 | 00 | 0 |
| **MOV AH,54H** | 54 | 33 | 00 | EE | 00 | 00 | 0 |
| **MUL AH** | 10 | BC | 00 | EE | 00 | 00 | 1 |
| **ADC AH, AL** | CD | BC | 00 | EE | 00 | 00 | 0 |
| **MOV BH,14** | CD | BC | 0E | EE | 00 | 00 | 0 |
| **MUL BH** | 0A | 48 | 0E | EE | 00 | 00 | 1 |
| **DIV BH** | 00 | BC | 0E | EE | 00 | 00 | 1 |

## Task 2: Write an assembly language code to compute the cube of a number

Write an assembly language program that cube the 8-bit number found in DL. Load DL with 7 initially, and make sure that your result is a 16-bit number.

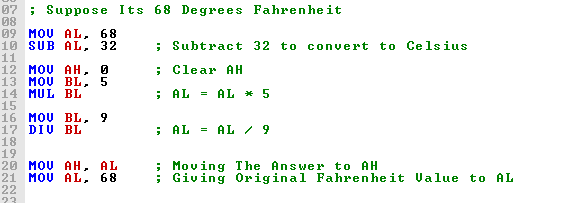


## Task 3: Write an assembly code that converts the Fahrenheit reading of temperature into the equivalent Celsius reading

Write an assembly language program that converts the Fahrenheit reading of temperature into the equivalent Celsius reading. Celsius reading is found in AH register and Fahrenheit reading is to be stored in AL register.

Hint: Formula for conversion is as follows





**AX Holds ‘14’ which is 20C and AL Holds ‘44’ which is 68F**

A screenshot of a computer

Description automatically generated

# Rubric for Lab Assessment

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

**Instructor Signature:**  **Date:**