Laboratory Exercise #3-1
Arithmetic Operations

Target Course Outcome:
CO2: Designs a microprocessor-based firmware integrating a microprocessor with supporting peripherals and devices.

Objectives:
To use the different arithmetic and logic operations in creating assembly language programs.

Tools Required:
Emu8086 Emulator

Part 1:

Arithmetic Operation

The 8086-instruction set supports the following arithmetic operations:

- Addition/Addition with carry
- Subtraction
- Multiplication
- Division

ADD – ADD Destination, Source
ADC – ADC Destination, Source

These instructions add a number from some *source* to a number in some *destination* and put the result in the specified destination. The ADC also adds the status of the carry flag to the result. The source may be an immediate number, a register, or a memory location. The destination may be a register or a memory location. The source and the destination in an instruction cannot both be memory locations. The source and the destination must be of the same type (bytes or words). If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with 0's before adding. Flags affected: AF, CF, OF, SF, ZF.

Example:
```
ADD AL, 74H     ; Add immediate number 74H to content of AL. Result in AL
ADC CL, BL       ;Add content of BL plus carry status to content of CL(CL = CL+BL+Carry Flag)
ADD DX, BX      ; Add content of BX to content of DX
ADD DX, [SI]    ;Add word from memory at offset [SI] in DS to content of DX
```

SUB – SUB Destination, Source
SBB – SBB Destination, Source

These instructions subtract the number in some *source* from the number in some *destination* and put the result in the destination. The SBB instruction also subtracts the content of carry flag from the destination. The source may be an immediate number, a register or memory location. The destination can also be a register or a memory location. However, the source and the destination cannot both be memory location. The source and the destination must both be of the same type (bytes or words). If you want to subtract a byte from a word, you must first move the byte to a word location such as a 16-bit register and fill the upper byte of the word with 0's. Flags affected: AF, CF, OF, PF, SF, ZF.

Example:

```
SUB CX, BX,     ;CX – BX; Result in CX
SBB CH, AL      ; Subtract content of AL and content of CF from content of CH.
```

MUL – MUL Source

This instruction multiplies an *unsigned* byte in some *source* with an *unsigned* byte in AL register or an unsigned word in some *source* with an unsigned word in AX register. The source can be a register or a memory location. When a byte is multiplied by the content of AL, the result (product) is put in AX. When a word is multiplied by the content of AX, the result is put in DX and AX registers. If the most significant byte of a 16-bit result or the most significant word of a 32-bit result is 0, CF and OF will both be 0's. AF, PF, SF and ZF are undefined after a MUL instruction.

If you want to multiply a byte with a word, you must first move the byte to a word location such as an extended register and fill the upper byte of the word with all 0's. You cannot use the CBW instruction for this, because the CBW instruction fills the upper byte with copies of the most significant bit of the lower byte.

Example:

```
MUL BL          ; Multiply AL with BH; result in AX
MUL CX          ;Multiply AX with CX; result high word in DX, low word in AX
```

IMUL – IMUL Source

This instruction multiplies a *signed* byte from *source* with a *signed* byte in AL or a *signed* word from some *source* with a *signed* word in AX. The source can be a register or a memory location. When a byte from source is multiplied with content of AL, the signed result (product) will be put in AX. When a word from source is multiplied by AX, the result is put in DX and AX. If the magnitude of the product does not require all the bits of the destination, the unused byte / word will be filled with copies of the sign bit. If the upper byte of a 16-bit result or the upper word of a 32-bit result contains only copies of the sign bit (all 0's or all 1's), then CF and the OF will both be 0; If it contains a part of the product, CF and OF will both be 1. AF, PF, SF and ZF are undefined after IMUL.

If you want to multiply a signed byte with a signed word, you must first move the byte into a word location and fill the upper byte of the word with copies of the sign bit. If you move the byte into AL, you can use the CBW instruction to do this.

Example:

```
IMUL BH         ;Multiply signed byte in AL with signed byte in BH; result in AX.
IMUL DX         ; Multiply AX times AX; result in DX and AX
```

DIV – DIV Source
This instruction is used to divide an *unsigned* word by a byte or to divide an *unsigned* double word (32 bits) by a word. When a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain the 8-bit quotient, and AH will contain the 8-bit remainder. When a double word is divided by a word, the most significant word of the double word must be in DX, and the least significant word of the double word must be in AX. After the division, AX will contain the 16-bit quotient and DX will contain the 16-bit remainder. If an attempt is made to divide by 0 or if the quotient is too large to fit in the destination (greater than FFH / FFFFH), the 8086 will generate a type 0 interrupt. All flags are undefined after a DIV instruction.

If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0's. Likewise, if you want to divide a word by another word, then put the dividend word in AX and fill DX with all 0's.

Example:

```
DIV BL          ;Divide word in AX by byte in BL; Quotient in AL, remainder in AH
DIV CX          ; Divide down word in DX and AX by word in CX; Quotient in AX, and remainder
                  in DX.
```

IDIV – IDIV Source
This instruction is used to divide a *signed* word by a *signed* byte, or to divide a *signed* double word by a *signed* word. When dividing a signed word by a signed byte, the word must be in the AX register. The divisor can be in an 8-bit register or a memory location. After the division, AL will contain the signed quotient, and AH will contain the signed remainder. The sign of the remainder will be the same as the sign of the dividend. If an attempt is made to divide by 0, the quotient is greater than 127 (7FH) or less than –127 (81H), the 8086 will automatically generate a type 0 interrupt.

When dividing a signed double word by a signed word, the most significant word of the dividend (numerator) must be in the DX register, and the least significant word of the dividend must be in the AX register. The divisor can be in any other 16-bit register or memory location. After the division, AX will contain a signed 16-bit quotient, and DX will contain a signed 16-bit remainder. The sign of the remainder will be the same as the sign of the dividend. Again, if an attempt is made to divide by 0, the quotient is greater than +32,767 (7FFFH) or less than –32,767 (8001H), the 8086 will automatically generate a type 0 interrupt. All flags are undefined after an IDIV.

If you want to divide a signed byte by a signed byte, you must first put the dividend byte in AL and sign-extend AL into AH. The CBW instruction can be used for this purpose. Likewise, if you want to divide a signed word by a signed word, you must put the dividend word in AX and extend the sign of AX to all the bits of DX. The CWD instruction can be used for this purpose.

Example:

```
IDIV BL         ; Signed word in AX/signed byte in BL
IDIV CX         ; Signed word AX in DX and/signed word in CX
```

Part II
Write a program to implement the different arithmetic operations.

Activity #1
Using registers and immediate value or immediate data .

Procedure:

1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe
2. Create a new file and label it as Exp3-1.asm
3. In the editor window, write the following instructions.

```
ORG 100H
    MOV AL, F0H
    MOV BL, 20H
    ADC AX, BX
    MOV CL, AL
    SBB CL, BL
    ADC CL, BL
    MUL BL
    MUL CX
    IMUL BX
    MOV AX,BX
    DIV DL
RET
```

4. Press the Compile and Emulate button or press F5 key on your keyboard. The emulator window should open with this program loaded, click the Single Step button and watch the register values.
5. Write the value of the registers in Canvas after each instruction execution as seen on the emulator window.

Copyright Information

Change log:

Date	Version	Author	Changes
August 7, 2020	1.0	Rosana J. Ferolin	Initial Draft
Sep. 3, 2025	2.0	Marlowe Edgar C. Burce	Corrected arithmetic instruction syntax errors, revised code and procedures