Course Name:	**Microprocessors and Microcontrollers Laboratory**	**Semester:**	**IV**
Date of Performance:	**29 / 01 / 2024**	**Batch No:**	**A - 2**
Faculty Name:	**Kirti Sawlani**	**Roll No:**	**16014022050**
Faculty Sign & Date:		**Grade / Marks:**	**___ / 25**

Experiment No.: 1
Title: Multiplication of 32 - bit numbers

Aim and Objective of the Experiment:

Write an 8086 based Assembly Language Program to multiply two 32-bit numbers stored in the data segment and store the result back in the data segment.

COs to be achieved:

CO2: Develop 8086 based assembly language programs.

Theory:

To study basic instructions and addressing modes of 8086. Understand assembler directives and concept of data and code segment.

This experiment covers following instructions groups.

a. Data transfer
b. Arithmetic (Multiply instructions)

Stepwise-Procedure:

1. Open EMU8086 and write your ASM code in the empty workspace.
2. Click on emulate button and it should open the emulator window.
3. You can run the code using single step execution and monitor the internal registers / flags.

Algorithm / Flowchart (example shown):

29/01/24

MPMC : Experiment 1

→ Algorithm / Flowchart : (INPUT : A, B, C, D : 16-bit integers)

1. Load the data segment address into DS.
2. Multiply B by D and store the result in res
3. Multiply A by D, add the result to res, and propagate carry.
4. Multiply C by B, add the result to res, and propagate carry.
5. Multiply C by A, add the result to res, and propagate carry.
6. Terminate program.

Assembly Language Program:

Code:

```
; MPMC LAB Experiment 1 | 16014022050

; Data segment declaration
data segment
    A dw  0FFFFh          ; Declare word-sized variable A initialized to FFFFh
    B dw  0FFFFh          ; Declare word-sized variable B initialized to FFFFh
    C dw  0FFFFh          ; Declare word-sized variable C initialized to FFFFh
    D dw  0FFFFh          ; Declare word-sized variable D initialized to FFFFh
```

```
    res dw  4 dup(0)     ; Reserve a doubleword array 'res' of 4 elements,
initialized to 0

data ends

; Code segment declaration
code segment
    assume ds: data, cs: code ; Set up data segment and code segment registers

start:
    mov ax, data           ; Load the data segment address into AX register
    mov ds, ax             ; Move the data segment address into the DS register

    ; Multiply B by D and store the result in 'res'
    mov ax, B
    mul D
    mov res, ax
    mov res+2, dx

    ; Multiply A by D and add the result to 'res'
    mov ax, A
    mul D
    add res+2, ax
    adc res+4, dx
    adc res+6, 0

    ; Multiply C by B and add the result to 'res'
    mov ax, C
    mul B
    add res+2, ax
    adc res+4, dx
    adc res+6, 0

    ; Multiply C by A and add the result to 'res'
    mov ax, C
    mul A
    add res+4, ax
    adc res+6, dx

    mov ah, 4Ch            ; Load the function number 4Ch (terminate process) into
AH
    int 21h                ; Call DOS interrupt 21h to terminate the program
```

```
code ends
end start
```

Listing of Code:

EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

noname.exe_ -- emu8086 assembler version: 4.08

[06-02-2024 -- 10:25:51]

```
===============================================================================
[LINE]     LOC: MACHINE CODE                          SOURCE
===============================================================================



[   1]        :
[   2]        :                                 ; MPMC LAB Experiment 1 | 16014022050
[   3]        :
[   4]        :                                 ; Data segment declaration
[   5]        :                                 data segment
[   6]    0000: FF FF                                     A dw  0FFFFh         ; Declare word-sized variable A
initialized to FFFFh
[   7]    0002: FF FF                                     B dw  0FFFFh         ; Declare word-sized variable B
initialized to FFFFh
[   8]    0004: FF FF                                     C dw  0FFFFh         ; Declare word-sized variable C
initialized to FFFFh
[   9]    0006: FF FF                                     D dw  0FFFFh         ; Declare word-sized variable D
initialized to FFFFh
[  10]        :
[  11]    0008: 00 00 00 00 00 00 00 00           res dw  4 dup(0)     ; Reserve a doubleword array
'res' of 4 elements, initialized to 0
[  12]        :
[  13]        :                                 data ends
[  14]        :
[  15]        :                                 ; Code segment declaration
[  16]        :                                 code segment
[  17]        :                                         assume ds: data, cs: code ; Set up data segment and code
segment registers
[  18]        :
```

```
[ 19]    0010:                          start:
[ 20]    0010: B8 00 00                          mov ax, data          ; Load the data segment address into
AX register
[ 21]    0013: 8E D8                             mov ds, ax            ; Move the data segment address into
the DS register
[ 22]        :
[ 23]        :                          ; Multiply B by D and store the result in 'res'
[ 24]    0015: A1 02 00                          mov ax, B
[ 25]    0018: F7 26 06 00                       mul D
[ 26]    001C: A3 08 00                          mov res, ax
[ 27]    001F: 89 16 0A 00                       mov res+2, dx
[ 28]        :
[ 29]        :                          ; Multiply A by D and add the result to 'res'
[ 30]    0023: A1 00 00                          mov ax, A
[ 31]    0026: F7 26 06 00                       mul D
[ 32]    002A: 01 06 0A 00                       add res+2, ax
[ 33]    002E: 11 16 0C 00                       adc res+4, dx
[ 34]    0032: 83 16 0E 00 00                    adc res+6, 0
[ 35]        :
[ 36]        :                          ; Multiply C by B and add the result to 'res'
[ 37]    0037: A1 04 00                          mov ax, C
[ 38]    003A: F7 26 02 00                       mul B
[ 39]    003E: 01 06 0A 00                       add res+2, ax
[ 40]    0042: 11 16 0C 00                       adc res+4, dx
[ 41]    0046: 83 16 0E 00 00                    adc res+6, 0
[ 42]        :
[ 43]        :                          ; Multiply C by A and add the result to 'res'
[ 44]    004B: A1 04 00                          mov ax, C
[ 45]    004E: F7 26 00 00                       mul A
[ 46]    0052: 01 06 0C 00                       add res+4, ax
[ 47]    0056: 11 16 0E 00                       adc res+6, dx
[ 48]        :
[ 49]    005A: B4 4C                             mov ah, 4Ch           ; Load the function number 4Ch
(terminate process) into AH
[ 50]    005C: CD 21                             int 21h               ; Call DOS interrupt 21h to terminate the
program
[ 51]        :
[ 52]        :                          code ends
[ 53]        :                          end start
[ 54]        :
[ 55]        :
```

===

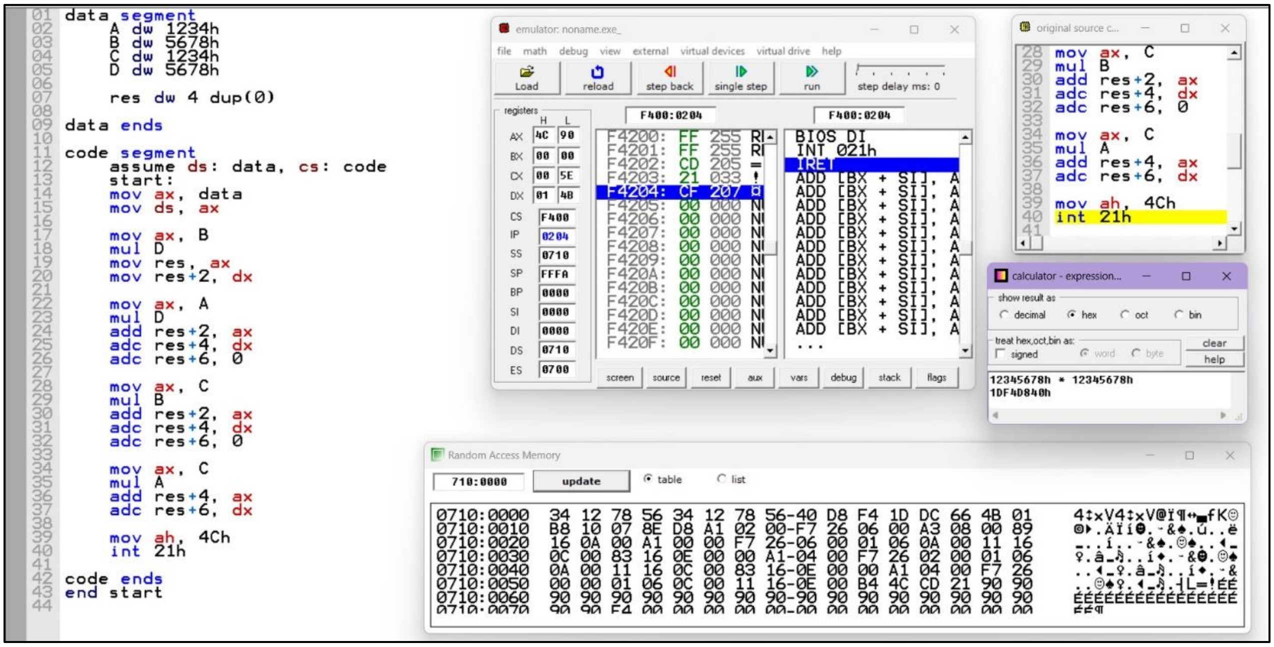
EXE HEADER - bytes from 0000 to 01FF inclusive.

0000: 4D - exe signature (M)
0001: 5A - exe signature (Z)
0002: 5E - bytes on last page (l.byte)
0003: 00 - bytes on last page (h.byte)
0004: 02 - 512 byte pages in file (l.byte)
0005: 00 - 512 byte pages in file (h.byte)
0006: 01 - relocations (l.byte)
0007: 00 - relocations (h.byte)
0008: 20 - paragraphs in header (l.byte)
0009: 00 - paragraphs in header (h.byte)
000A: 00 - minimum memory (l.byte)
000B: 00 - minimum memory (h.byte)
000C: FF - maximum memory (l.byte)
000D: FF - maximum memory (h.byte)
000E: 00 - SS - stack segment (l.byte)
000F: 00 - SS - stack segment (h.byte)
0010: 00 - SP - stack pointer (l.byte)
0011: 00 - SP - stack pointer (h.byte)
0012: B8 - check sum (l.byte)
0013: 69 - check sum (h.byte)
0014: 00 - IP - instruction pointer (l.byte)
0015: 00 - IP - instruction pointer (h.byte)
0016: 01 - CS - code segment (l.byte)
0017: 00 - CS - code segment (h.byte)
0018: 1E - relocation table adress (l.byte)
0019: 00 - relocation table adress (h.byte)
001A: 00 - overlay number (l.byte)
001B: 00 - overlay number (h.byte)
001C: 01 - signature (l.byte)
001D: 00 - signature (h.byte)
001E: 01 - relocation table - offset inside segment (l.byte)
001F: 00 - relocation table - offset inside segment (h.byte)
0020: 01 - relocation table - segment anchor (l.byte)
0021: 00 - relocation table - segment anchor (h.byte)
0022 to 01FF - reserved relocation area (00)

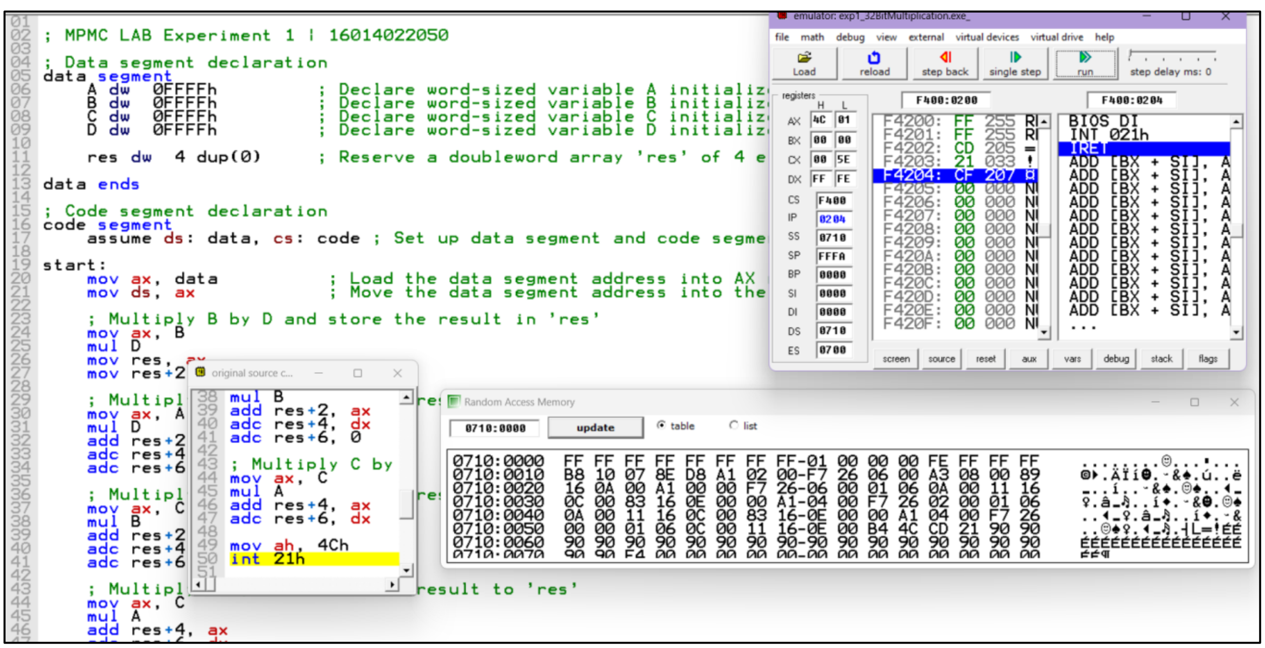
===

Output Screenshots:

1. Multiplication of 1234H × 5678H:



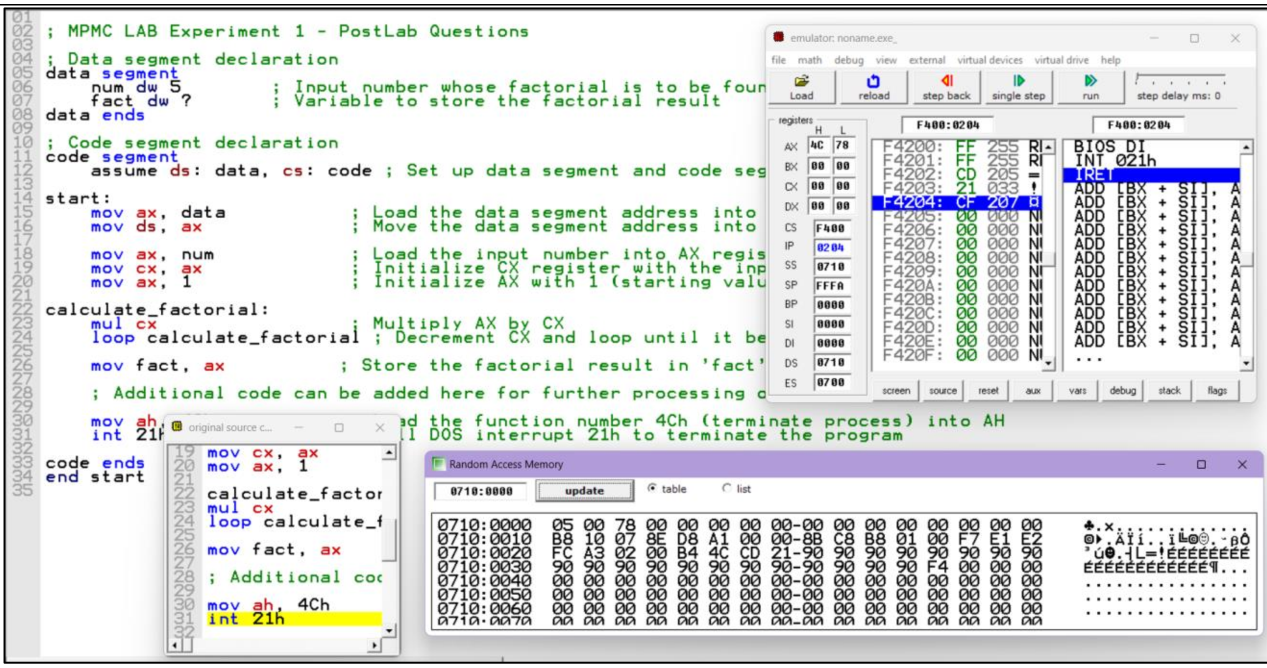
2. Multiplication of FFFFH × FFFFH:



Post Lab Subjective / Objective type Questions:

1. **Write an 8086 based ALP to find the factorial of a number in data segment and store the result back in data segment.**

```
; MPMC LAB Experiment 1 - PostLab Questions

; Data segment declaration
data segment
    num dw 5         ; Input number whose factorial is to be found (example: 5)
    fact dw ?        ; Variable to store the factorial result
data ends

; Code segment declaration
code segment
    assume ds: data, cs: code ; Set up data segment and code segment registers

start:
    mov ax, data           ; Load the data segment address into AX register
    mov ds, ax             ; Move the data segment address into the DS register

    mov ax, num            ; Load the input number into AX register
    mov cx, ax             ; Initialize CX register with the input number
    mov ax, 1              ; Initialize AX with 1 (starting value for factorial)

calculate_factorial:
    mul cx                 ; Multiply AX by CX
    loop calculate_factorial ; Decrement CX and loop until it becomes zero

    mov fact, ax          ; Store the factorial result in 'fact'

    ; Additional code can be added here for further processing or output

    mov ah, 4Ch            ; Load the function number 4Ch (terminate process) into
AH
    int 21h                ; Call DOS interrupt 21h to terminate the program

code ends
end start
```



2. What is the output of the following instruction?

AX = 37D7H, BH = 151 decimal

DIV BH

The DIV instruction in x86 assembly performs unsigned division. The instruction divides the 32-bit value in the DX:AX register pair by the specified operand (register or memory).

In the given instruction DIV BH, AX = 37D7H (hex) and BH = 151 (decimal). The quotient is stored in AX, and the remainder is stored in DX.

Performing the division: 37D7H ÷ 151 = 24 (quotient), remainder 58.

So, after the execution of the DIV BH instruction, AX will be 24 (decimal) and DX will be 58.

3. What is the difference between MUL and IMUL? Explain with example.

MUL (Unsigned Multiply): It is used for unsigned multiplication. The operands and result are treated as unsigned integers. The result is twice the size of the operands.

```
mov ax, 5    ; Operand 1
mov bx, 6    ; Operand 2
mul bx       ; Result in DX:AX
```

IMUL (Signed Multiply): It is used for signed multiplication. It considers the operands and result as signed integers. The result is twice the size of the operands, and it may include a sign extension.

```
mov ax, -5   ; Operand 1 (signed)
mov bx, 6    ; Operand 2 (signed)
imul bx      ; Result in DX:AX (signed)
```

Example of MUL and IMUL with signed operands:

```
mov ax, -5    ; Operand 1 (signed)
mov bx, 6     ; Operand 2 (signed)

mul bx        ; Unsigned multiplication, result in DX:AX
; AX = FFF6H, DX = FFFFFFEC (signed)

imul bx       ; Signed multiplication, result in DX:AX
; AX = FFEC H, DX = FFFF FFEC H
```

In the case of signed multiplication (IMUL), the result is sign-extended to fill the doubleword.

Conclusion:

In conclusion, this experiment involved writing an 8086 assembly language program to multiply two 32-bit numbers stored in the data segment and subsequently store the result back in the data segment. Through this exercise, the fundamental concepts of memory management, arithmetic operations, and data manipulation in assembly language programming were reinforced.

Signature of faculty in-charge with Date: