**Department of Computer Science**

**lab assembly language**

**Lab Manual**

**instructor: M Usman Malik**

**Name: Faizan Munsaf**

**Roll NO: Bcsm-f19-257**

Lab # 1 An Introduction to EMU8086

Lab # 2 Using Basic Instructions

Lab # 3 Using Variables

Lab # 4 Using Basic Instructions: A Programming Exercise I

Lab # 5 Using Basic Instructions: A Programming Exercise II

Lab # 6 Sample programs

Lab # 7 Processor Status and Flag Register

Lab # 8 Sample Programs

Lab # 9 Stack and Procedure

Lab # 10 Quiz

Lab # 11 Sample Programs

Lab # 12 Sample Programs (Factorial)

Lab # 13 Sample Programs (Input and Output Decimal)

Lab # 14 Array Addressing

Lab # 15 2D array Addressing

# Lab 1 – An Introduction to EMU8086

## Objectives

In this lab, students will learn how to use EMU8086 to write and assemble programs in assembly language.

## Introduction

Emu8086 is a program that compiles the source code (assembly language) and executes it. You can watch registers, flags and memory while your program executes. Arithmetic & Logical Unit (ALU) shows the internal work of the central processor unit (CPU). Emulator runs programs on a Virtual PC; this completely blocks your program from accessing real hardware, such as hard-drives and memory, 8086 machine code is fully compatible with all next generations of Intel's microprocessors.

**Where to start?**

1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe.

2. Select "Samples" from "File" menu.

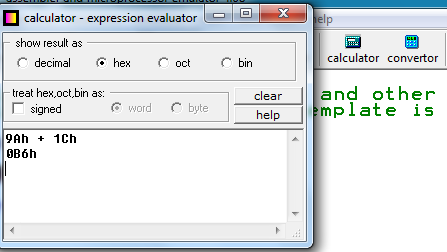
3. Click [Compile and Emulate] button (or press F5 hot key).

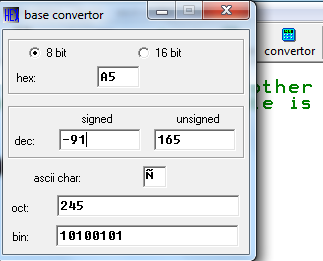
4. Click [Single Step] button (or press F8 hot key), and watch how the code is being executed.

5. Try opening other samples, all sample programs are heavily commented, so it's a great learning tool.

**TASKS**

1. Familiarize yourself with EMU8086 GUI. Explore different menu options. Learn how to use calculator and base converter. Also explore the ASCII table.





1. Press F1, to open the tutorial in browser. Read the following sections:
2. Where to Start?
3. Working with the Editor
4. How to Compile the Code
5. Assembly Language Tutorials (Numbering System, Part 1: What is Assembly Language)
6. Registers are high-speed storage locations inside the microprocessor. They are classified according to the function they performed. Each register has a name. One of the general types of register is Data Registers (that can hold data). AX (accumulator register) is a data register. Find out what other data registers can be used.
7. MOV is an instruction used in assembly language to the copy contents of source operand to destination operand. As a result, destination operands are modified but source contents remain unchanged. For example, if you execute following instruction in EMU8086, the contents will be copied in AX register.

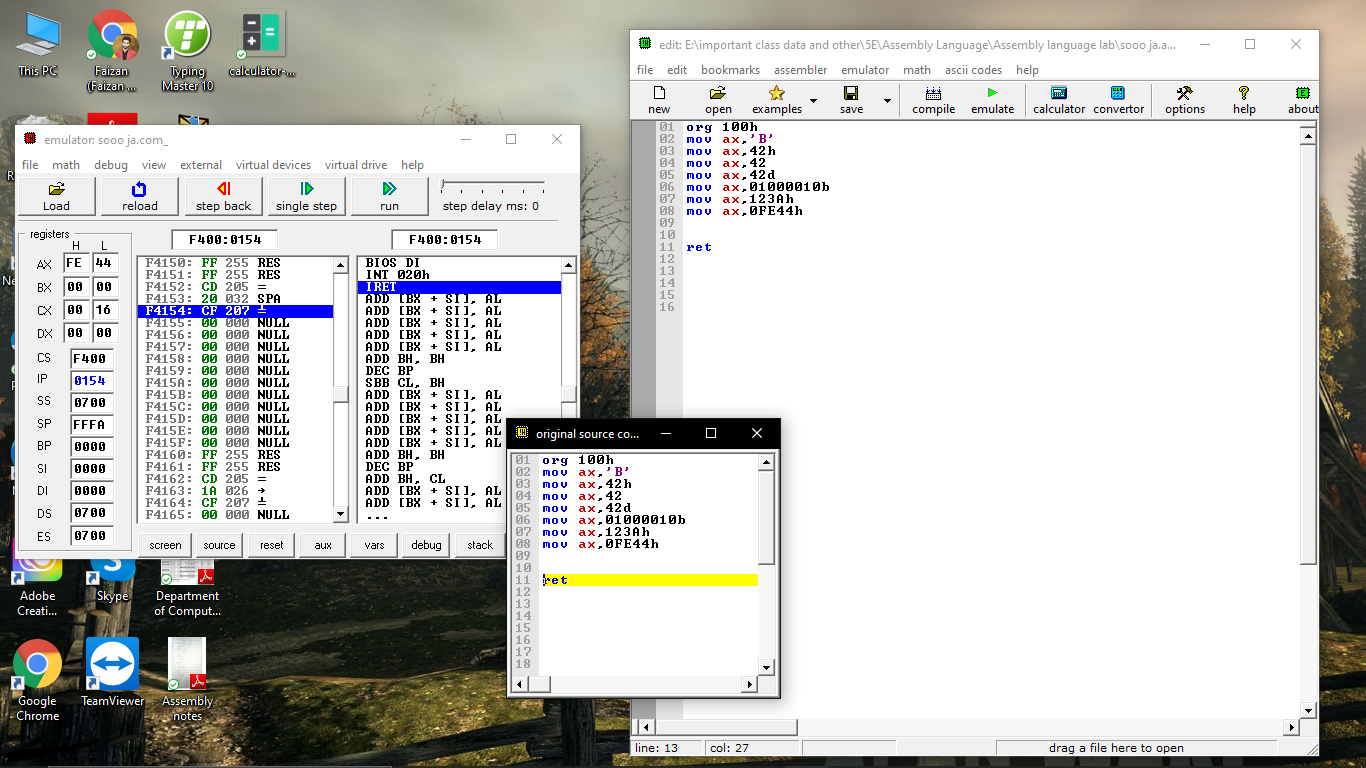
**MOV AX, 41h**

Where MOV is an assembly language instruction, AX is a data register (destination operand) and 41h is the source operand you want to copy.

Now execute the following instructions (single-step) in EMU8086 and observe the changes in contents of AX register.

1. MOV AX, ‘B’
2. MOV AX, 42h
3. MOV AX, 42
4. MOV AX, 42d
5. MOV AX, 0100 0010b
6. MOV AX, 123Ah
7. MOV AX, 0FE44h

**Solve**



# Lab 2 – Using Basic Instructions

## Objectives

In this lab, students will learn how to use instructions MOV, ADD, SUB, INC, DEC, and NEG in an assembly language program.

## Lab Tasks

**Task 1:** Read the syntax of assembly language instructions MOV, ADD, SUB, INC, DEC, and NEG (purpose, no. of operands required, and legal/illegal operands) from textbook ch#4. Also, find out all valid combinations of MOV instructions using data register and immediate values or both operands as data register.

**Task 2:** Now execute the following instructions (single-step) in EMU8086 and observe the changes in contents of destination operand. If any instruction gives error, correct that error.

1. MOV AL, 256 **= MOV AX,256**
2. MOV AX, F1ABh **=MOV AX,0F1AB**
3. MOV AX, -123
4. MOV BX, 123
5. MOV AH, 010010001b
6. MOV 1234h, BX
7. MOV DX, 33h
8. MOV CX, ‘AB’
9. MOV CH, AL
10. MOV DL, BL
11. MOV AH, BL
12. MOV AX, CL **=MOV AX,CX**

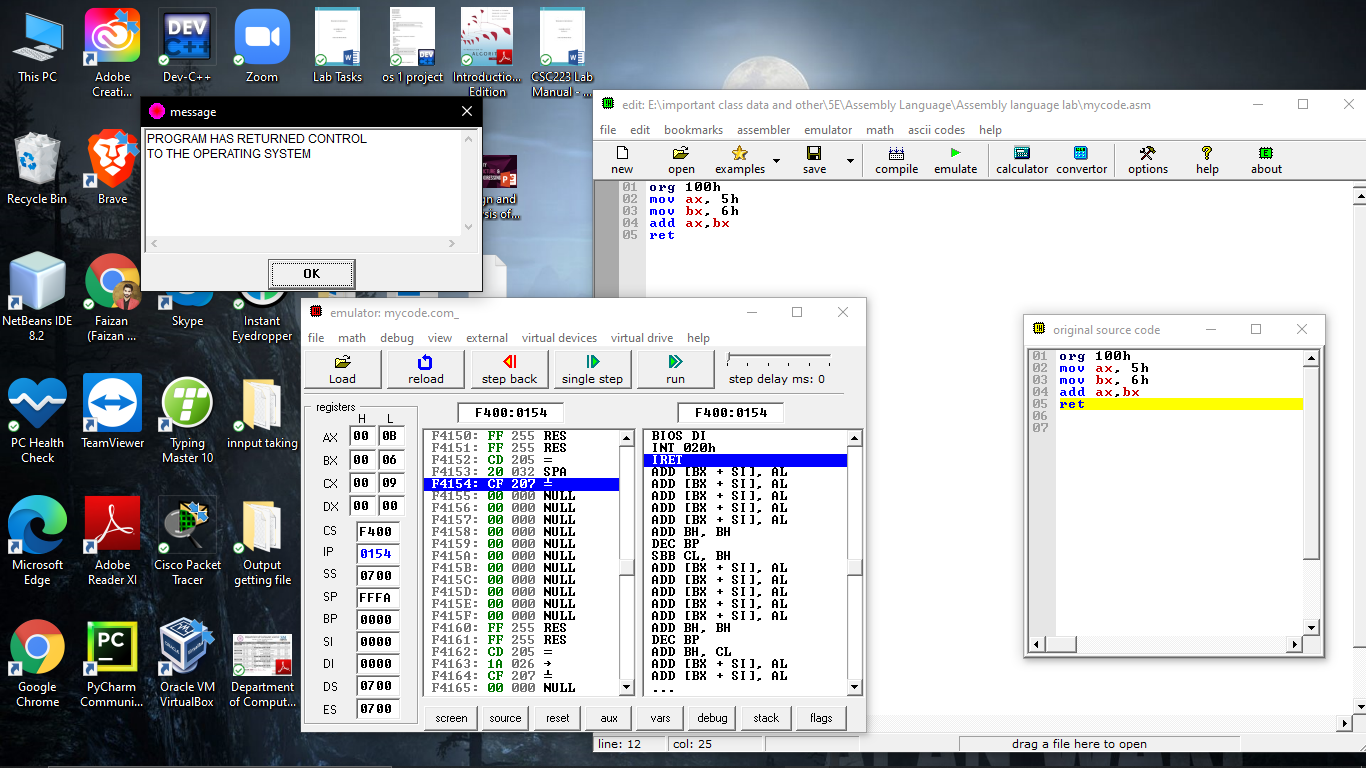
**Task 3:** For each of the following assembly language instructions, what will be the possible register value (in hex)? Assume all statements are being executed in a sequence and initially all registers contain zero.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Instructions** | **AX** | | **BX** | | **CX** | | **DX** | |
| **AH** | **AL** | **BH** | **BL** | **CH** | **CL** | **DH** | **DL** |
|  | MOV AX, 0110b | **06** | |  | |  | |  | |
|  | MOV BH, 90H |  | | **90** | |  | |  | |
|  | MOV CX, 10 |  | |  | | **0A** | |  | |
|  | MOV DX, 1234H |  | |  | | **04** | | **12 34** | |
|  | ADD DH, DL |  | |  | | **06** | | **46 34** | |
|  | ADD AX, BX | **90 06** | | **90** | | **08** | |  | |
|  | SUB CX, BX |  | | **90** | | **70 0A** | |  | |
|  | SUB AH, DL | **CC 06** | |  | |  | | **12 34** | |
|  | INC AX | **07** | |  | |  | |  | |
|  | INC BL |  | | **90 01** | |  | |  | |
|  | DEC DX |  | |  | |  | | **12 33** | |
|  | DEC CH |  | |  | | **FF 0A** | |  | |
|  | NEG CX |  | |  | | **FF F6** | |  | |
|  | NEG BL |  | | **90 00** | |  | |  | |

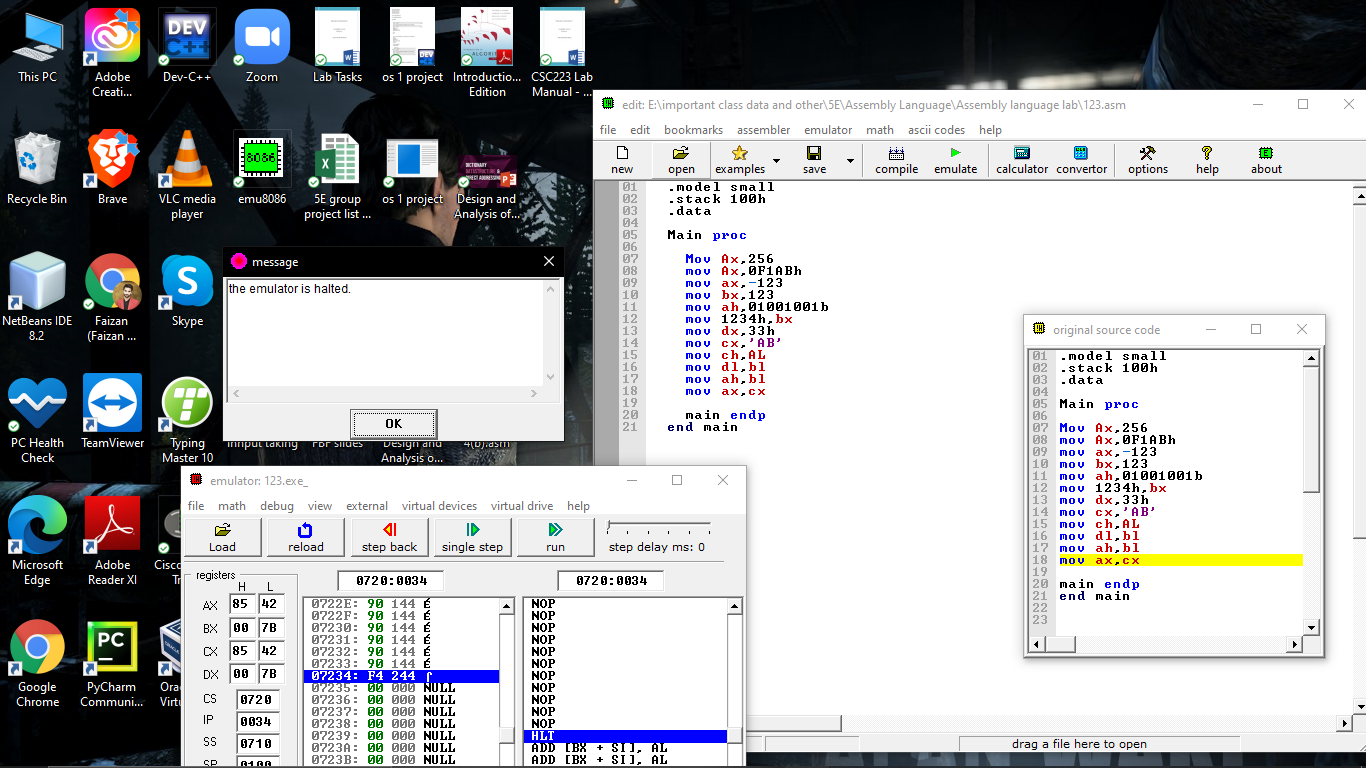
**Task 4:** Execute following statements (single-step) in EMU8086. Observe changes in the contents of destination operand.

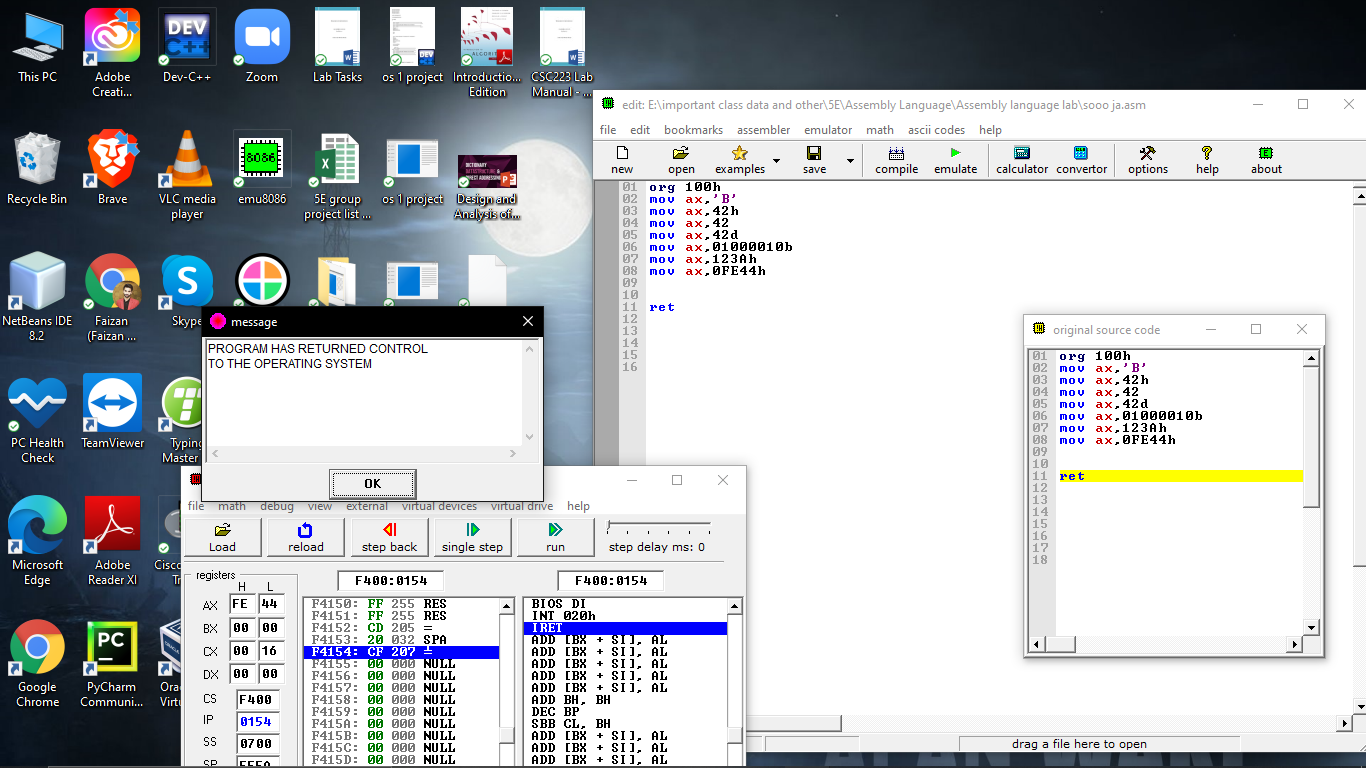
|  |  |  |
| --- | --- | --- |
| **No.** | **Code Snippet** | **Result** |
|  | MOV AX, 4D12h  MOV BX, 1C21h  ADD AX, BX | AX | AH = 69 | AL = 33 |  BX | BH = 1C | BL = 21 |  CX | CL= 09 |
|  | MOV AX, 4D12h  MOV BX, 1C21h  SUB AX, BX | AX | AH = 30 | AL = F1 |  BX | BH = 1C | BL = 21 |  CX | CL= 09 |
|  | MOV AL, 25  MOV BH, 1Ch  ADD AL, BH | AX | AH = 00 | AL = 35 |  BX | BH = 1C | BL = 00 |  CX | CL= 07 |
|  | MOV CL, 10000011b  MOV CH, 10010010b  SUB CH, CL | CX | CL= 07 |
|  | MOV AX, 80h  ADD AX, 01 | AX | AH = 00 | AL = 81 |  CX | CL= 07 |
|  | MOV AL, 80h  ADD AL, 01 | AX | AH = 00 | AL = 81 |  CX | CL= 05 |
|  | MOV AL, 0A5h  MOV BL, 12h  ADD AL, BL  MOV DH, BL  SUB DH, 10 | AX | AH = 00 | AL = B7 |  BX | BH = 00 | BL = 12 |  CX | CL= 0C  DX | DH= 08 |

# Lab 3 – Using Variable

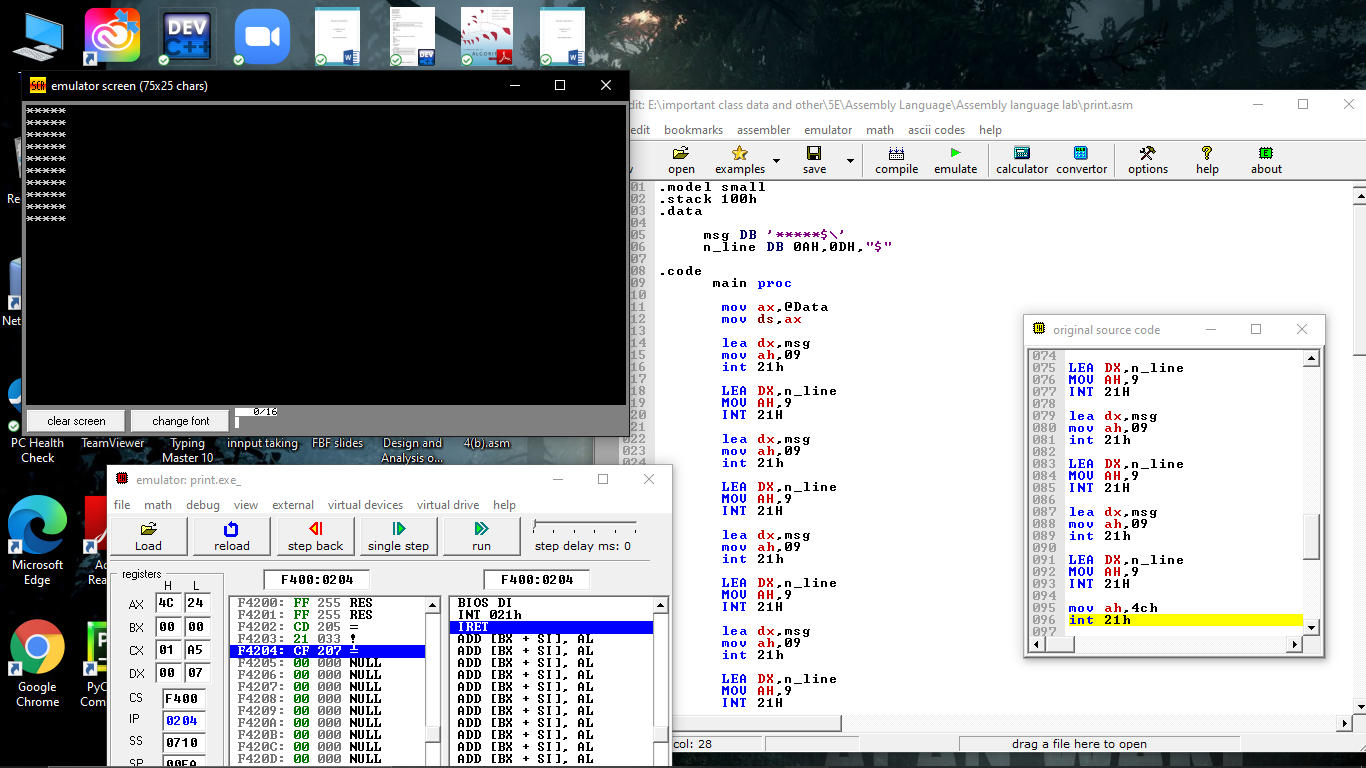


# Lab 4 – Using Basic Instructions: A Programming Exercise I



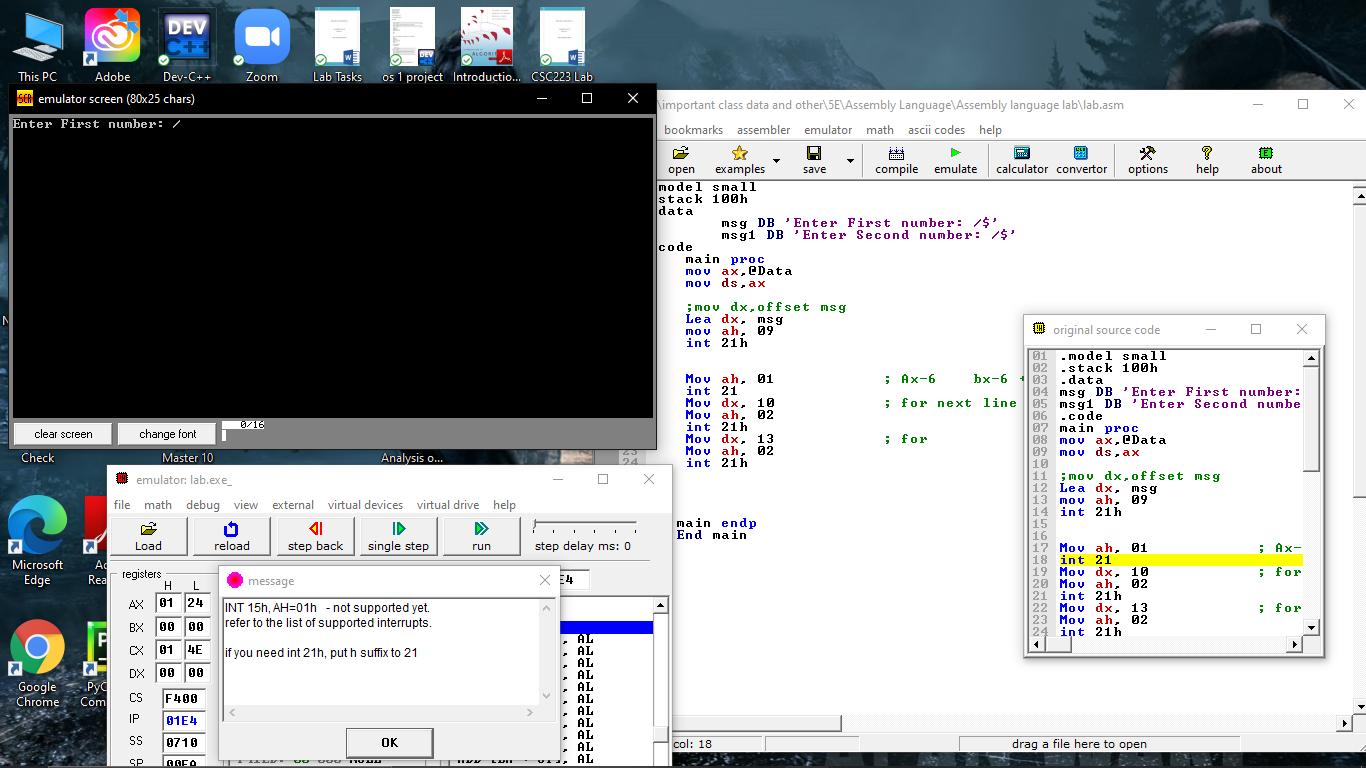


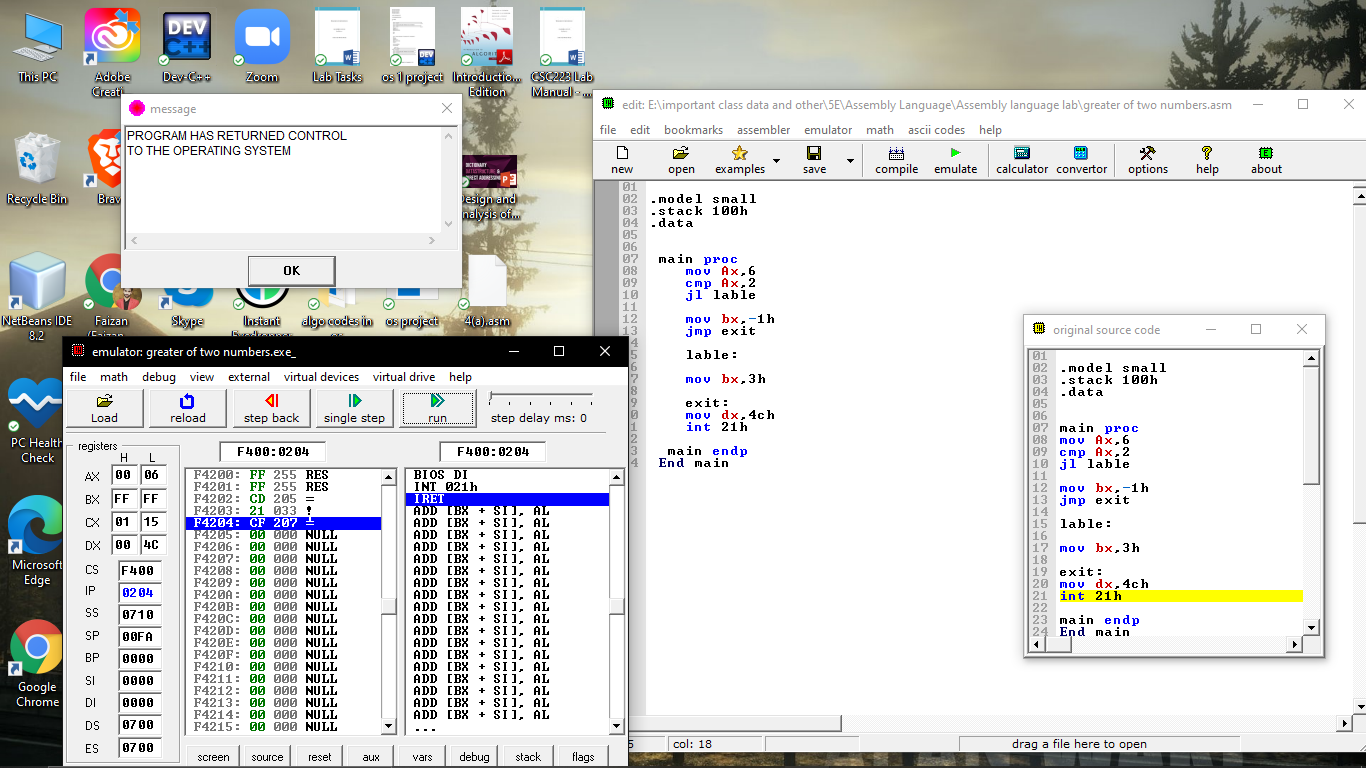
# Lab 5 – Using Basic Instructions: A Programming Exercise II



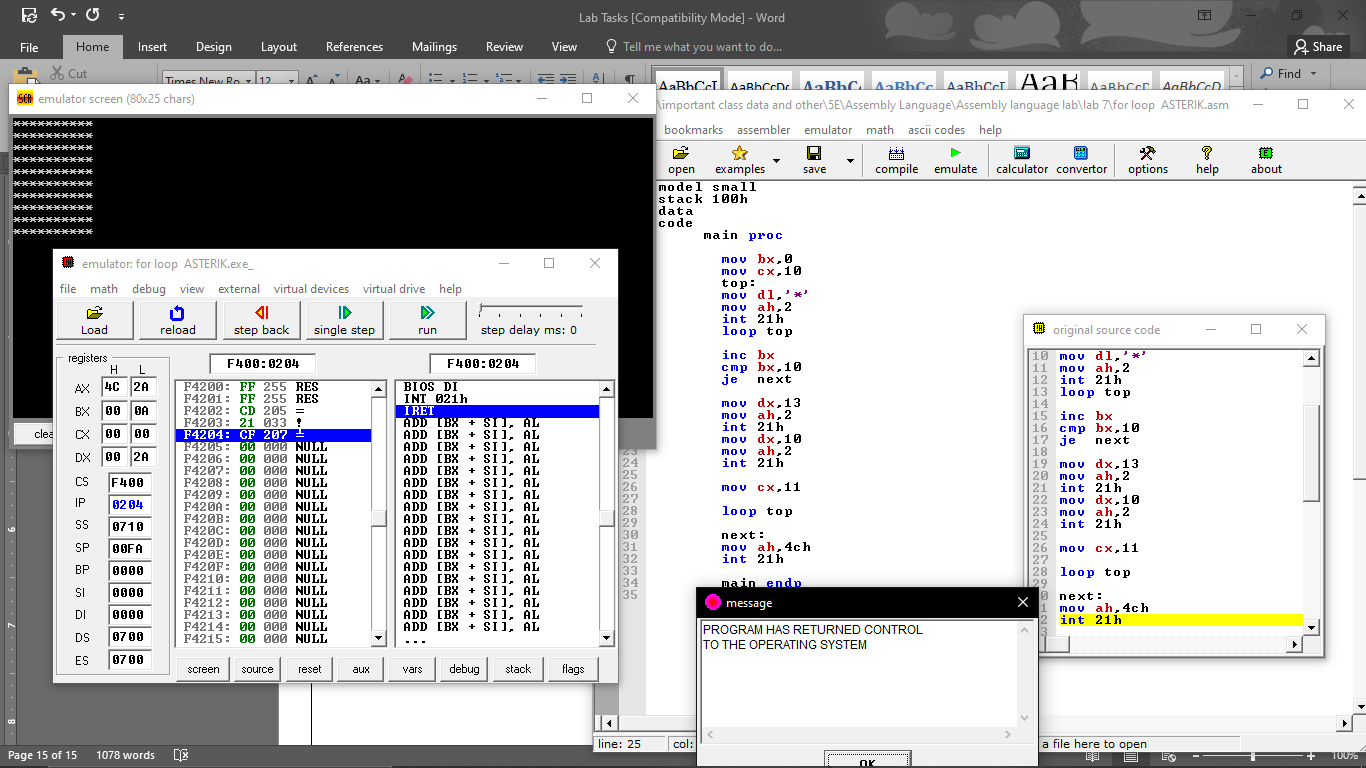


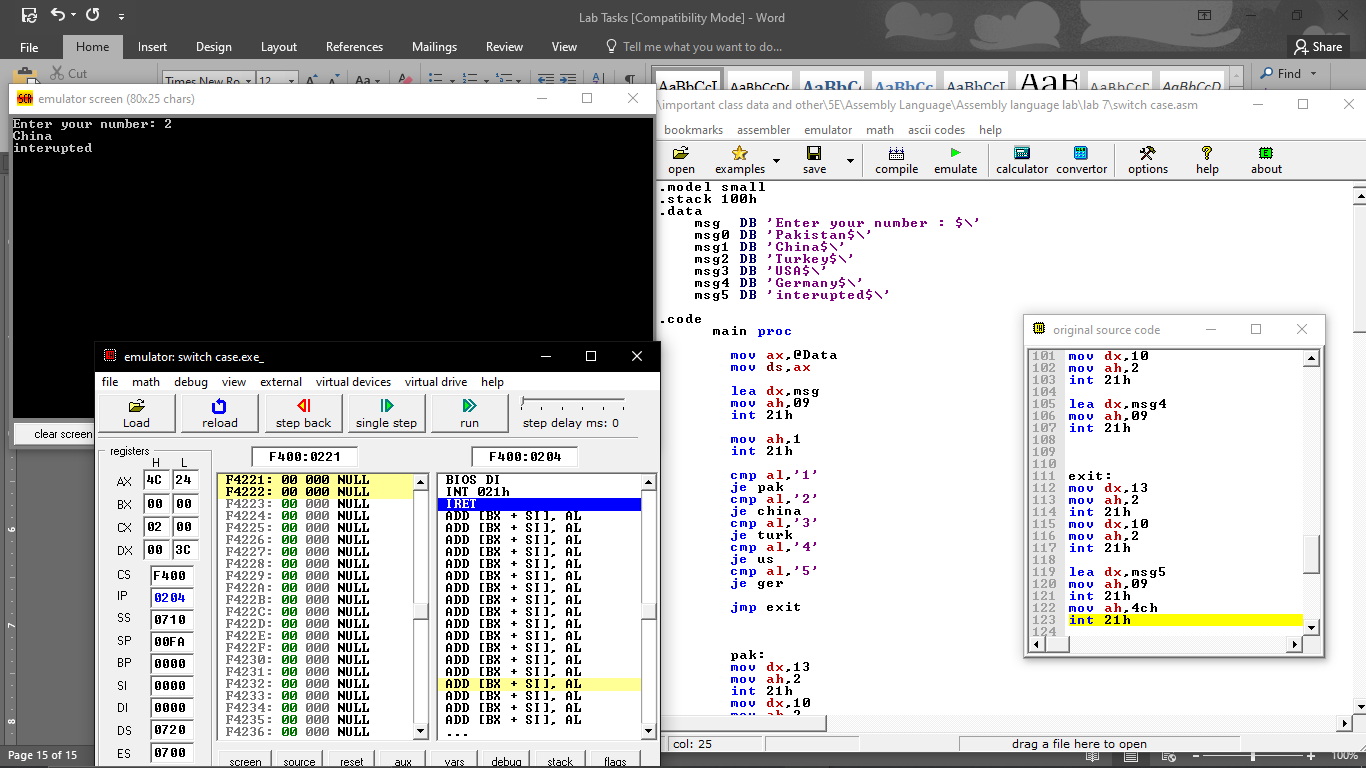
# Lab 6 – Sample programs



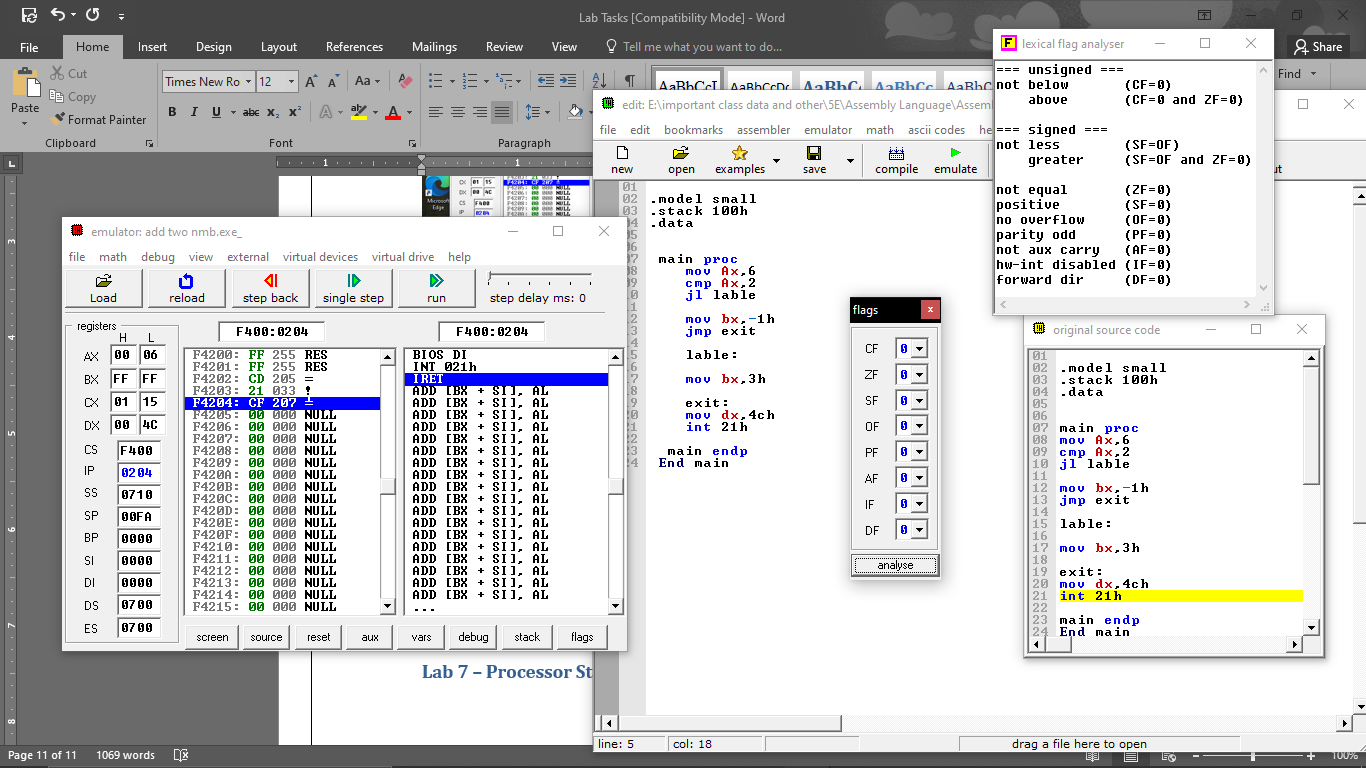


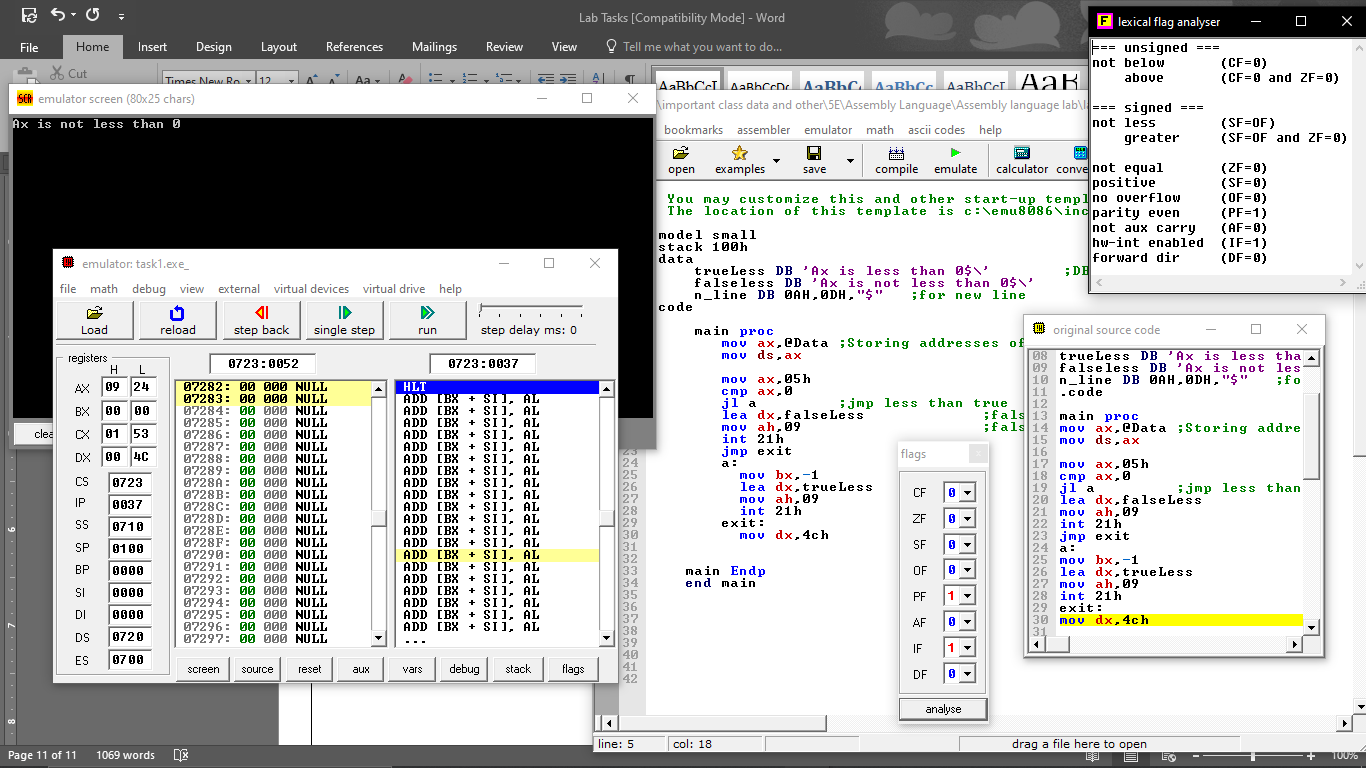


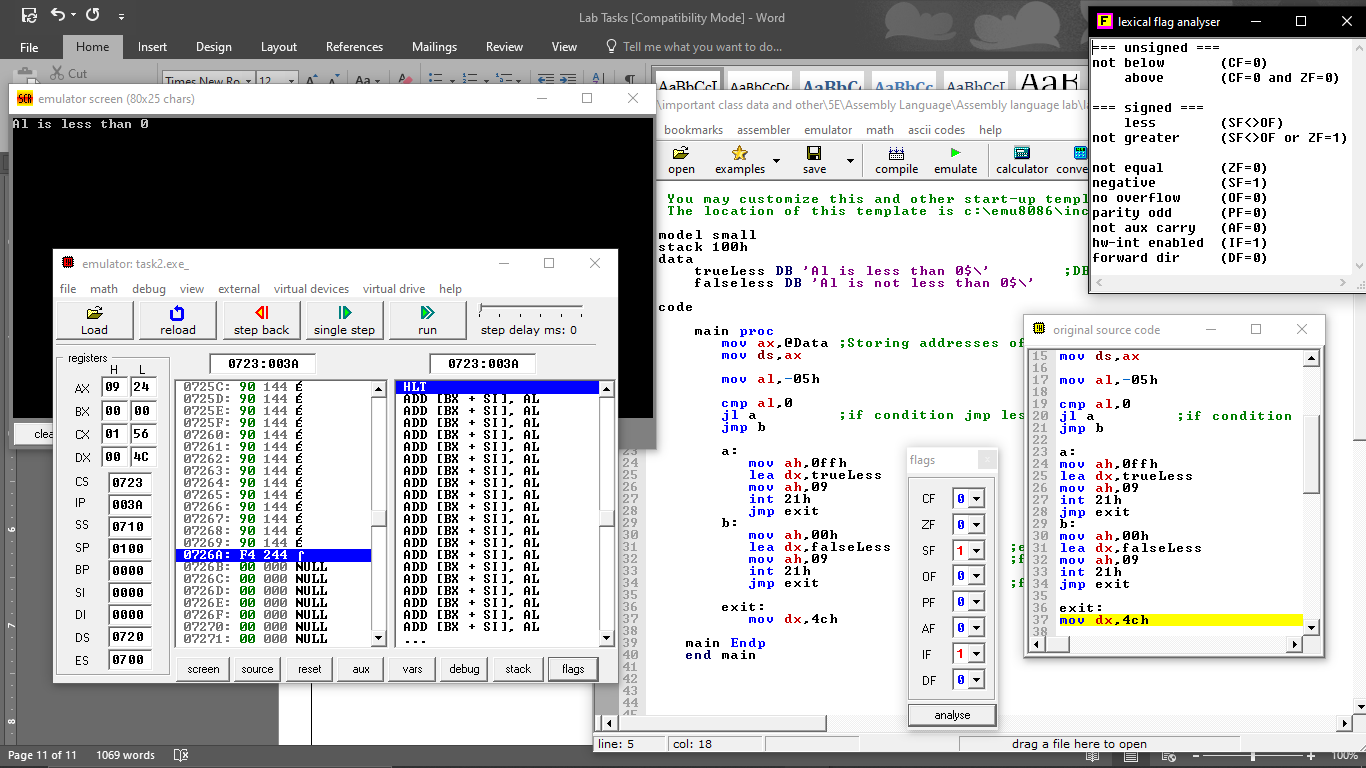


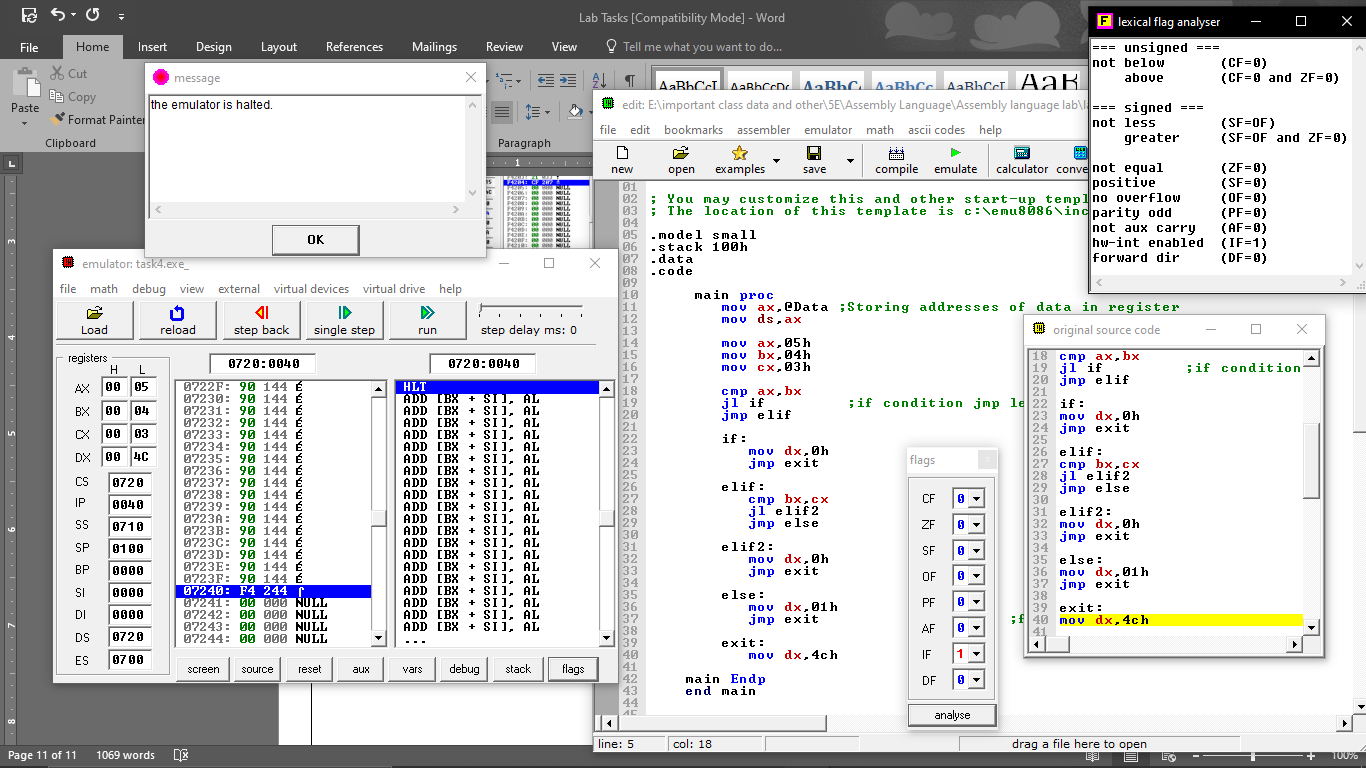


# Lab 7 – Processor Status and Flag Register

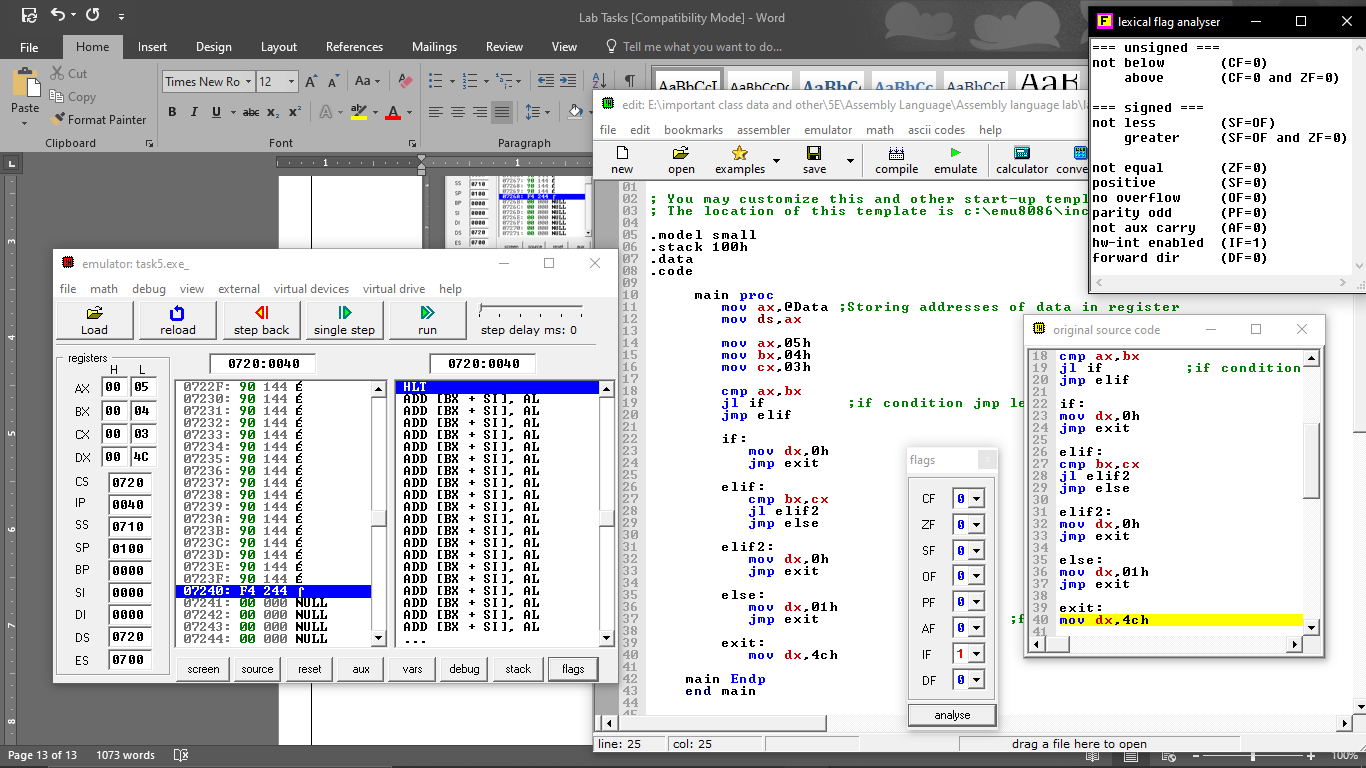




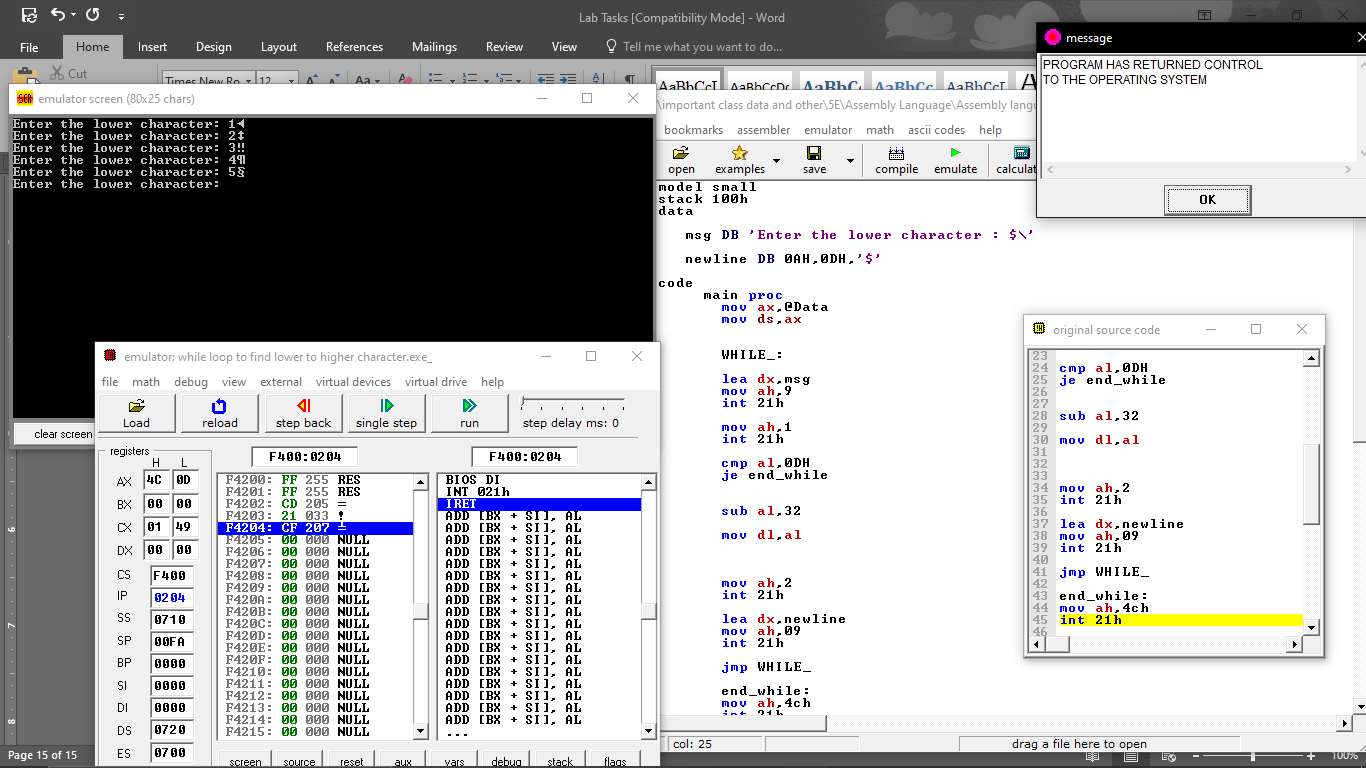




# Lab 8 – Sample Programs

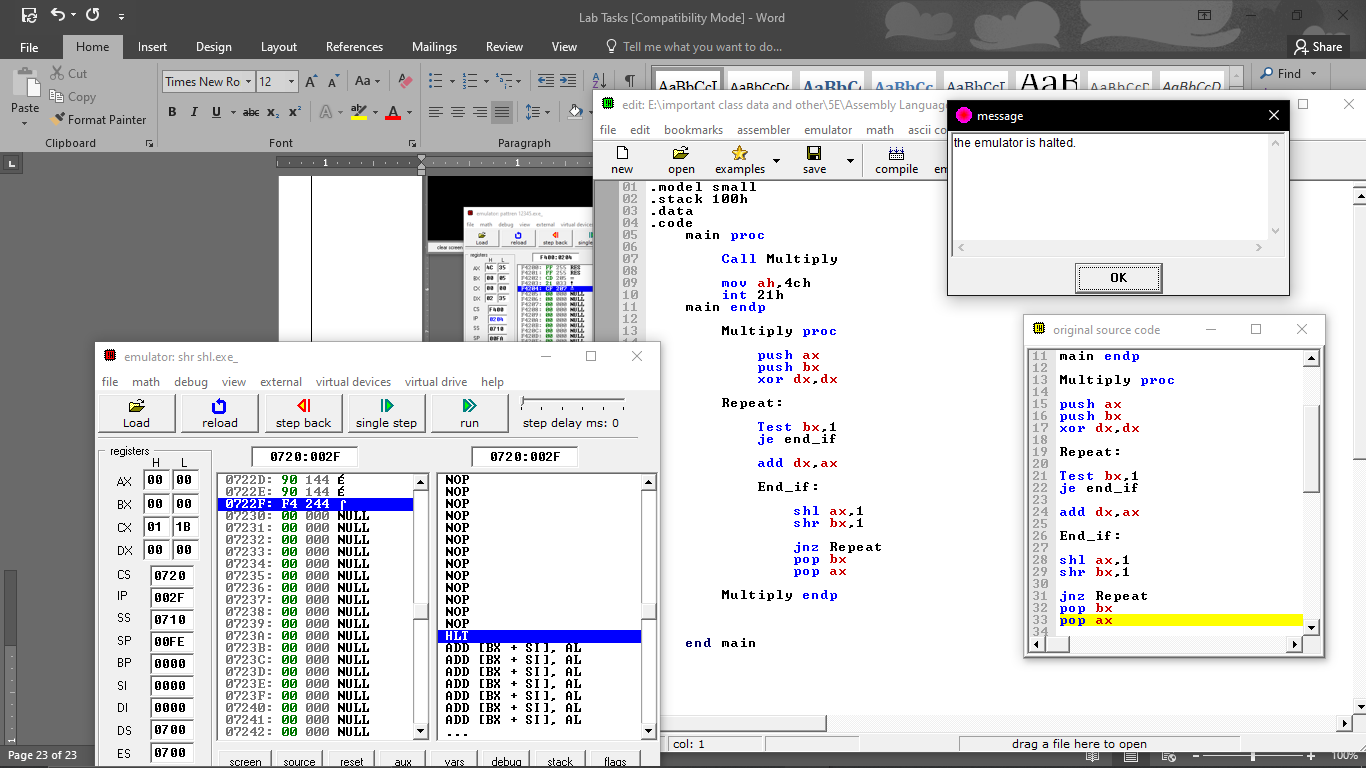


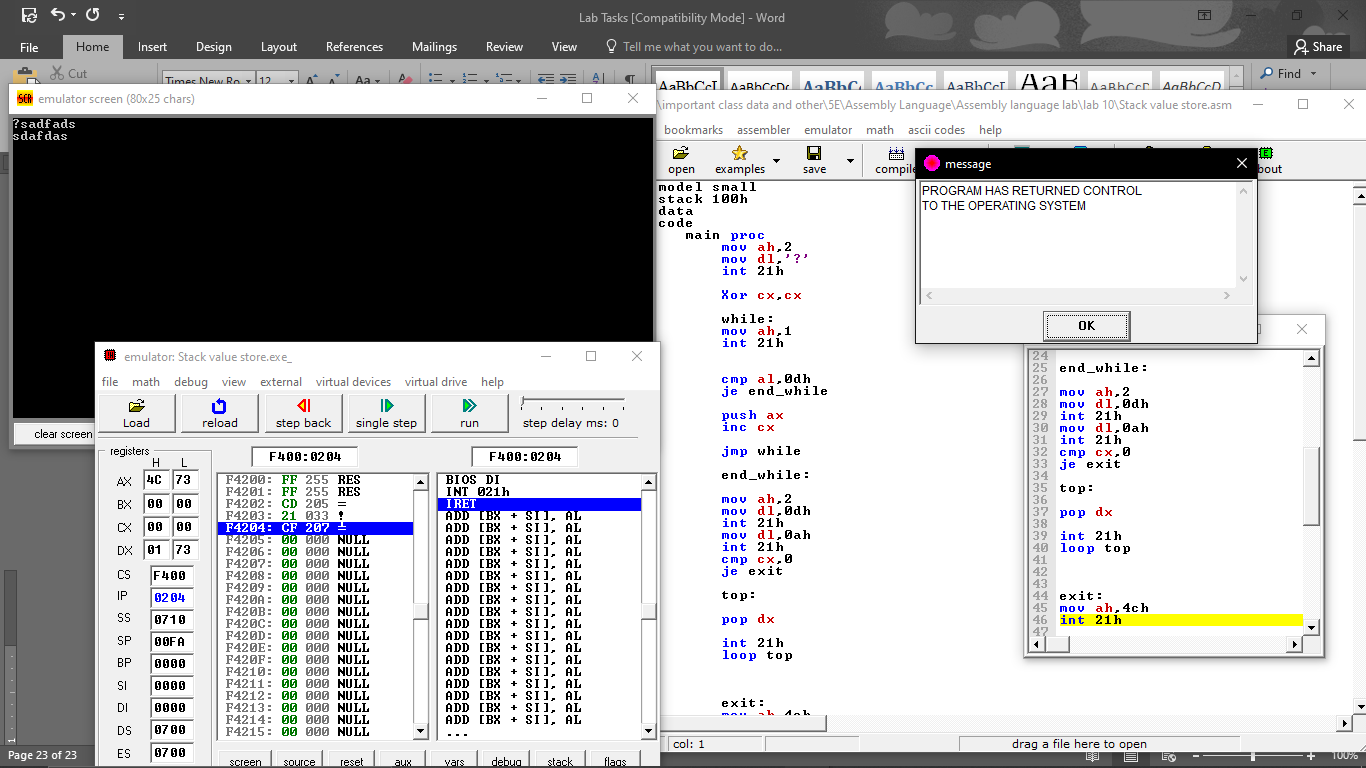




# Lab 9 – Stack and Procedure

**Stack**





# Lab 10 – Quiz

**Quiz**

.model small

.stack 100h

.data

.code

main proc

mov cx,1

top:

mov dx,13

mov ah,2

int 21h

mov dx,10

mov ah,2

int 21h

mov bx,1

top1:

mov dx,cx

add dx,48

mov ah,2

int 21h

inc bx

cmp bx,cx

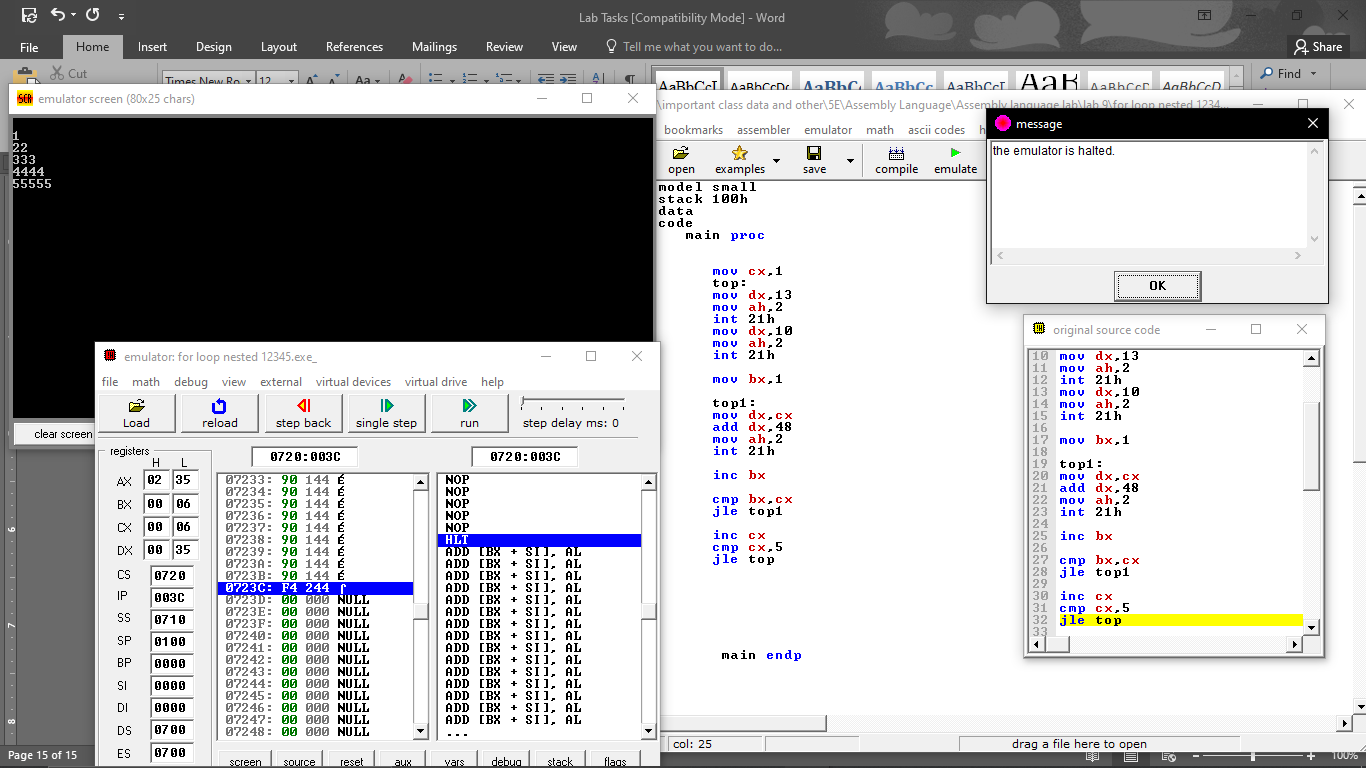
jle top1

inc cx

cmp cx,5

jle top

main endp



.model small

.stack 100h

.data

.code

main proc

mov ax,49

mov cx,1

top:

mov dx,ax

mov ah,2

int 21h

loop top

mov dx,13

mov ah,2

int 21h

mov dx,10

mov ah,2

int 21h

add ax,40

mov cx,2

top1:

mov dx,ax

mov ah,2

int 21h

loop top1

mov dx,13

mov ah,2

int 21h

mov dx,10

mov ah,2

int 21h

add ax,41

mov cx,3

top2:

mov dx,ax

mov ah,2

int 21h

loop top2

mov dx,13

mov ah,2

int 21h

mov dx,10

mov ah,2

int 21h

add ax,42

mov cx,4

top3:

mov dx,ax

mov ah,2

int 21h

loop top3

mov dx,13

mov ah,2

int 21h

mov dx,10

mov ah,2

int 21h

add ax,43

mov cx,5

top4:

mov dx,ax

mov ah,2

int 21h

loop top4

mov bx,5

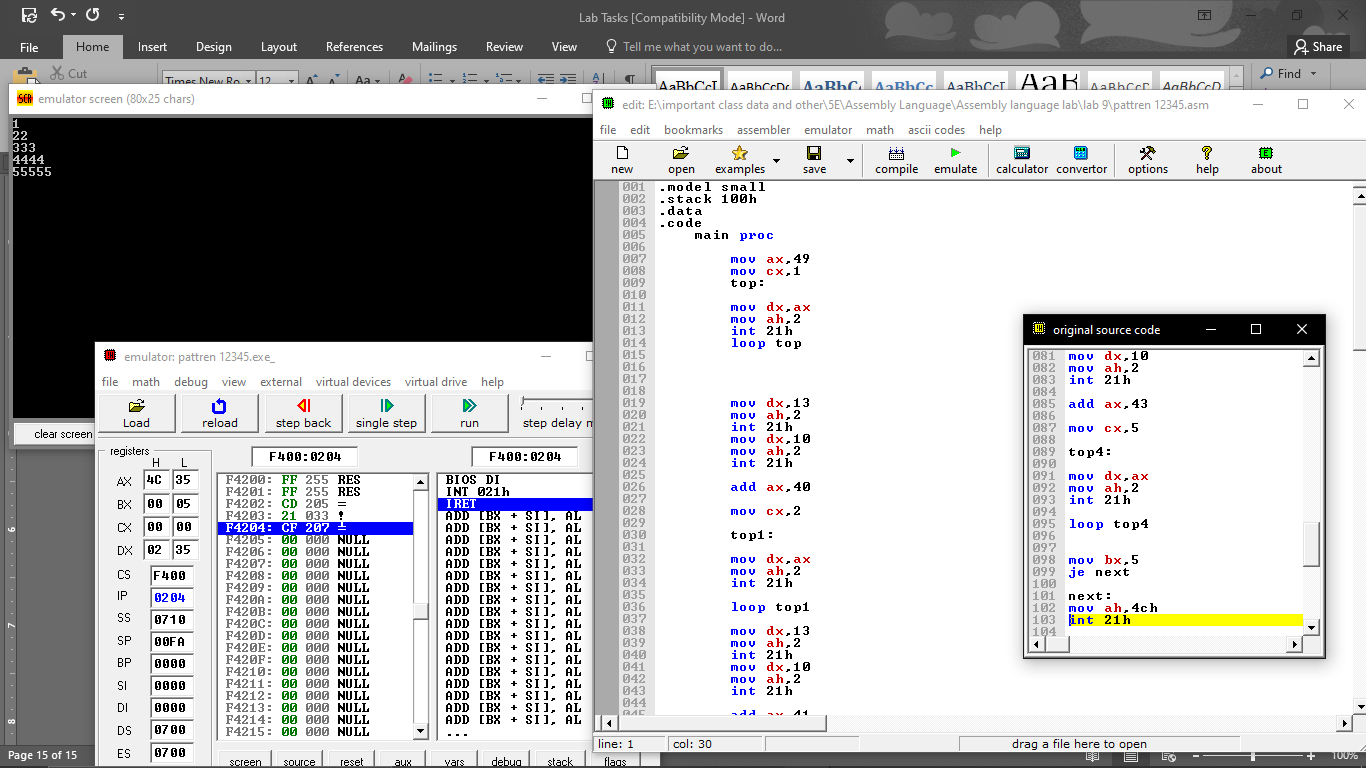
je next

next:

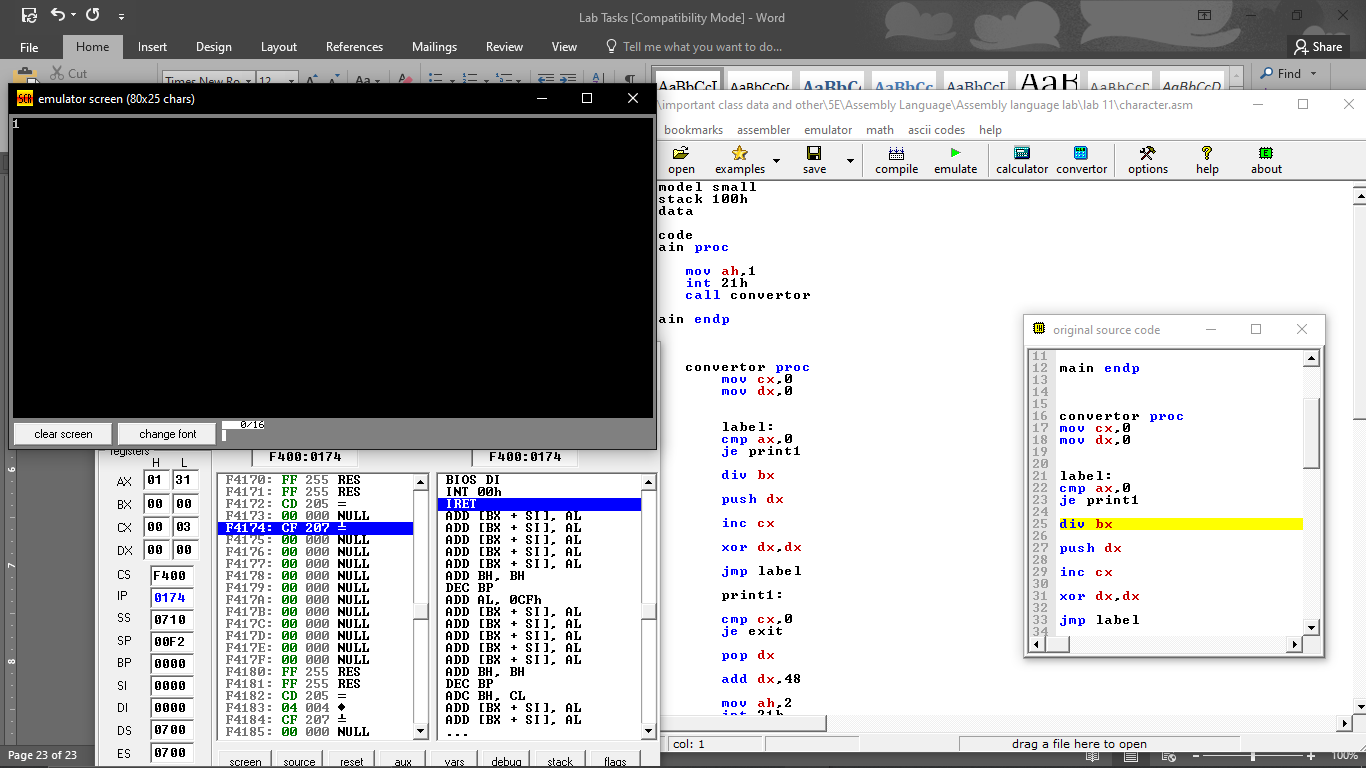
mov ah,4ch

int 21h

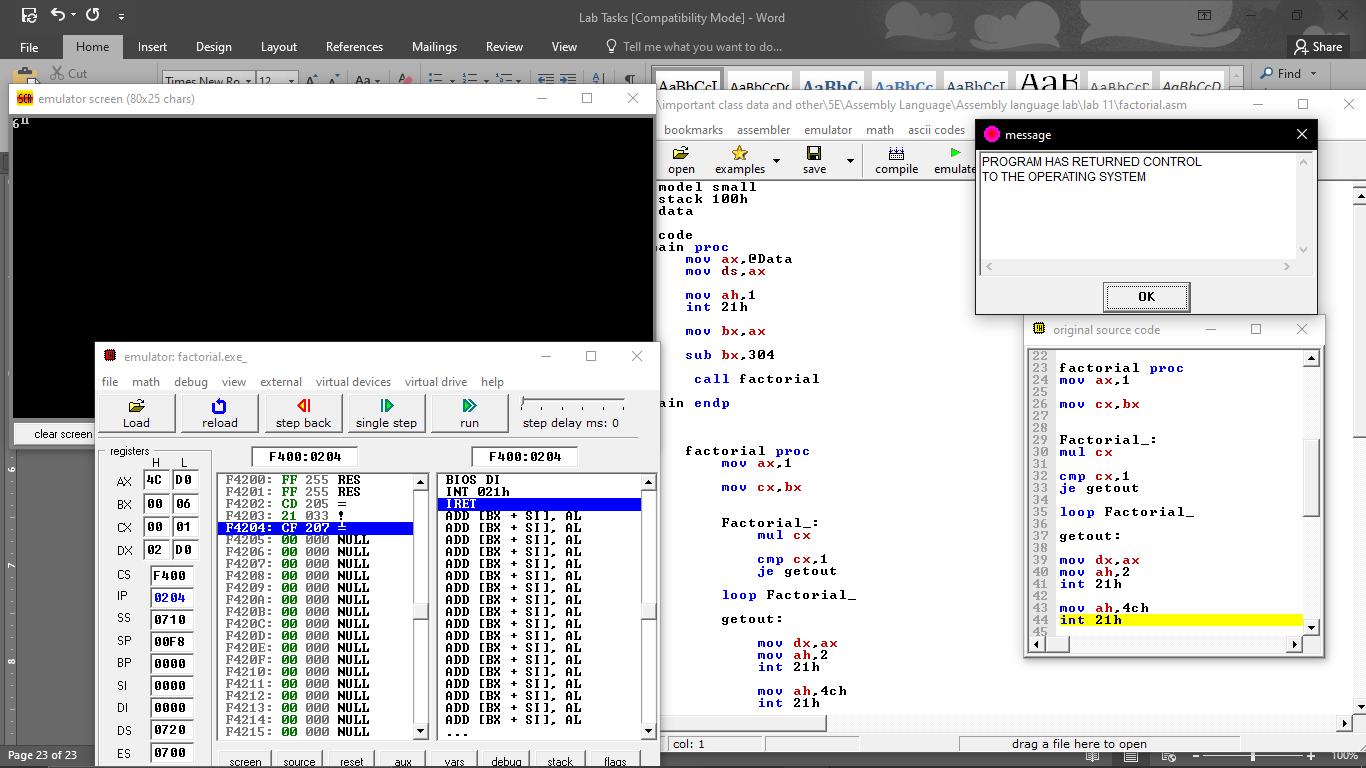
main endp



# Lab 11 – Sample Programs



# Lab 12 – Sample Programs (Factorial)



# Lab 13 – Sample Programs

.model small

.stack 100h

.data

.code

main proc

call indec

push ax

mov ah,2

mov dl,0dh

int 21h

mov dl,0ah

int 21h

pop ax

call outdec

mov ah,4ch

int 21h

main endp

indec proc

push bx

push cx

push dx

Begin:

mov ah,2

mov dl,'?'

int 21h

xor bx,bx

xor cx,cx

mov ah,1

int 21h

cmp al,'-'

je minus

cmp al,'+'

je plus

jmp repeat2

minus:

mov cx,1

plus:

int 21h

repeat2:

cmp al,'0'

jnge not\_digit

cmp al,'9'

jnle not\_digit

and ax,000fh

push ax

mov ax,10

mul bx

pop bx

add bx,ax

mov ah,1

int 21h

cmp al,0dh

jne repeat2

mov ax,bx

or cx,cx

je exit

neg ax

exit:

pop dx

pop cx

pop bx

ret

not\_digit:

mov ah,2

mov dl,0dh

int 21h

mov dl,0ah

int 21h

jmp begin

indec endp

outdec proc

push ax

push bx

push cx

push dx

or ax,ax

jge end\_if1

push ax

mov dl,'\_'

mov ah,2

int 21h

pop ax

neg ax

end\_if1:

xor cx,cx

mov bx,10d

repeat1:

xor dx,dx

div bx

push dx

inc cx

or ax,ax

jne repeat1

mov ah,2

print\_loop:

pop dx

or dl,30h

int 21h

loop print\_loop

pop dx

pop cx

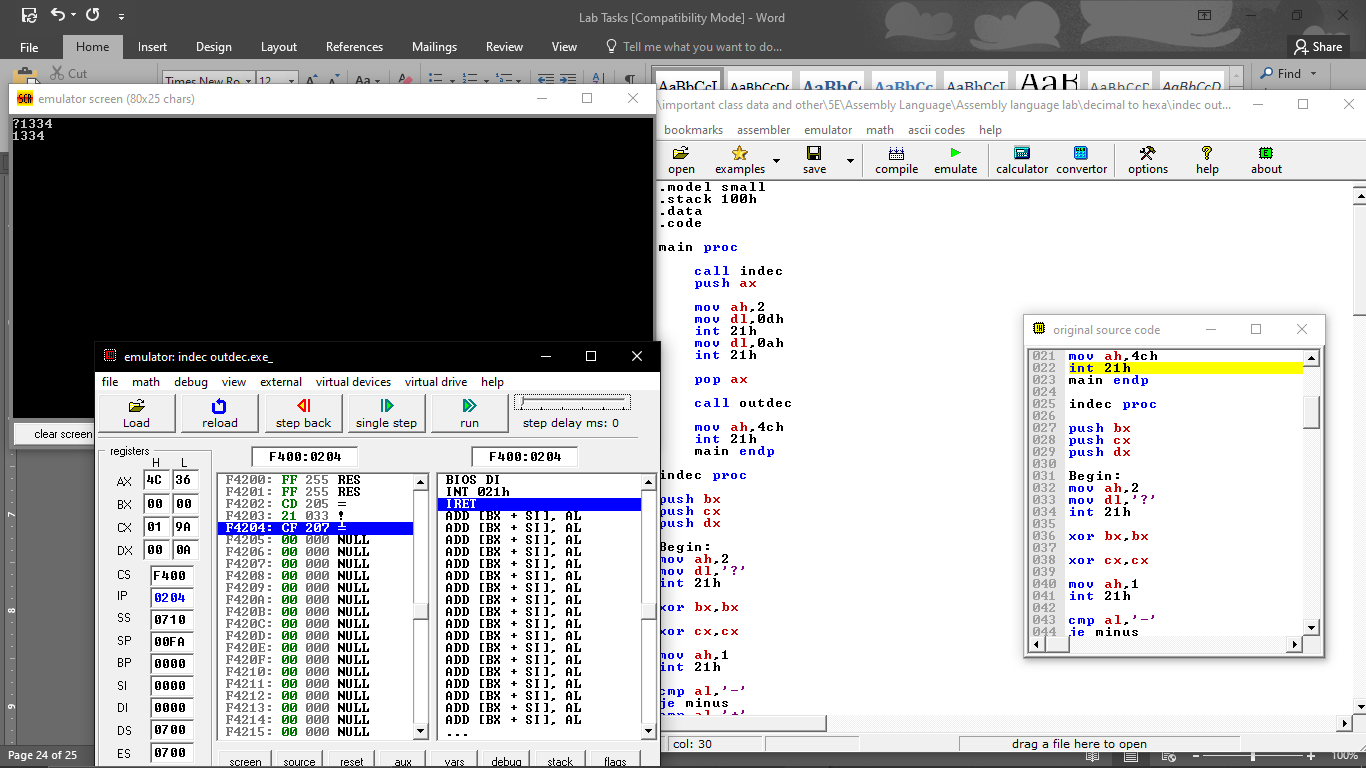
pop bx

pop ax

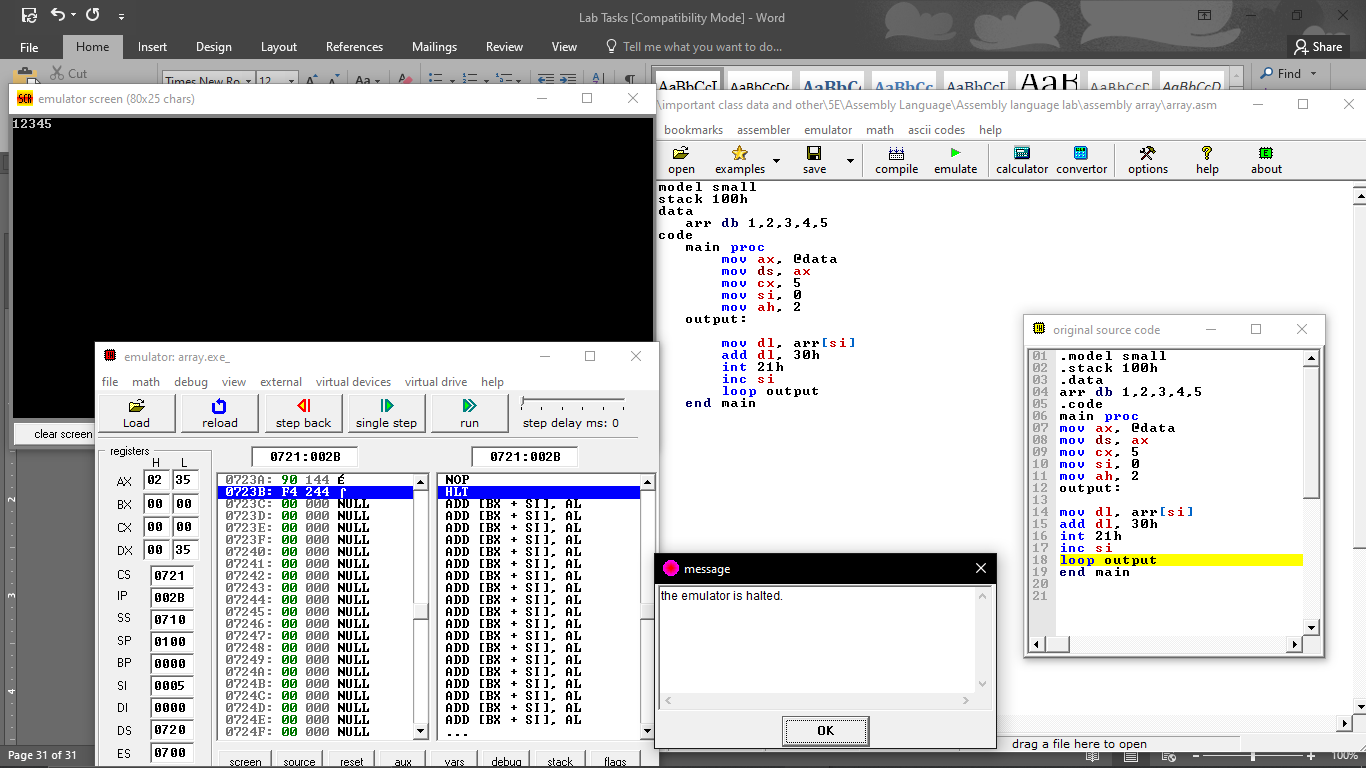
ret

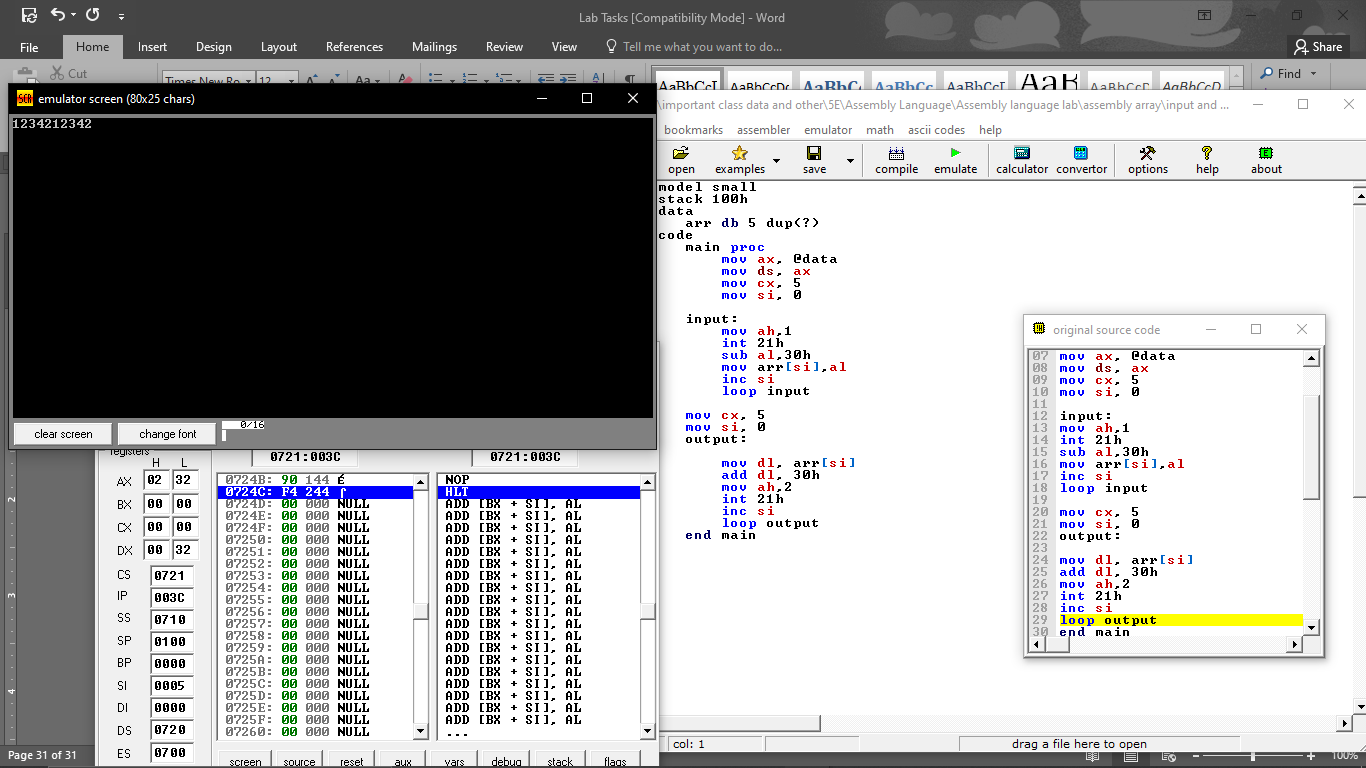
outdec endp

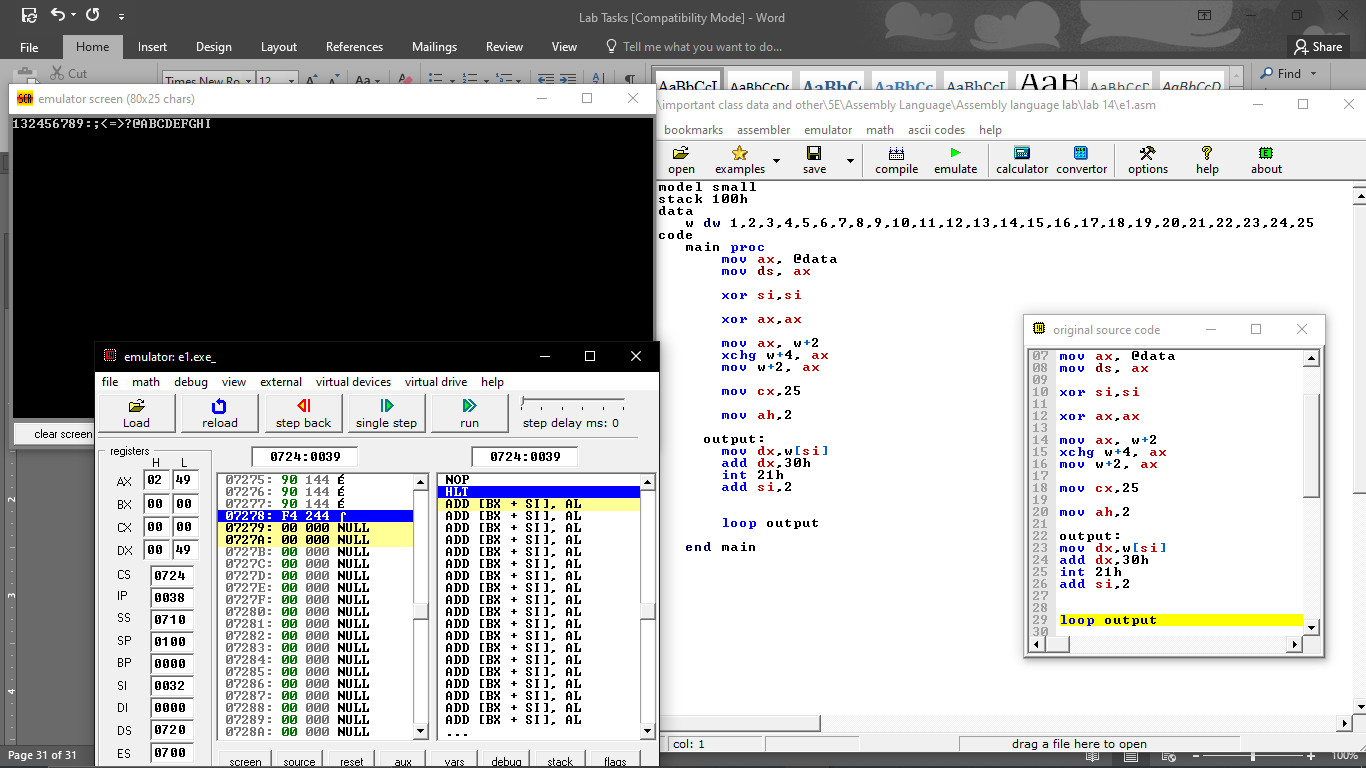
end main

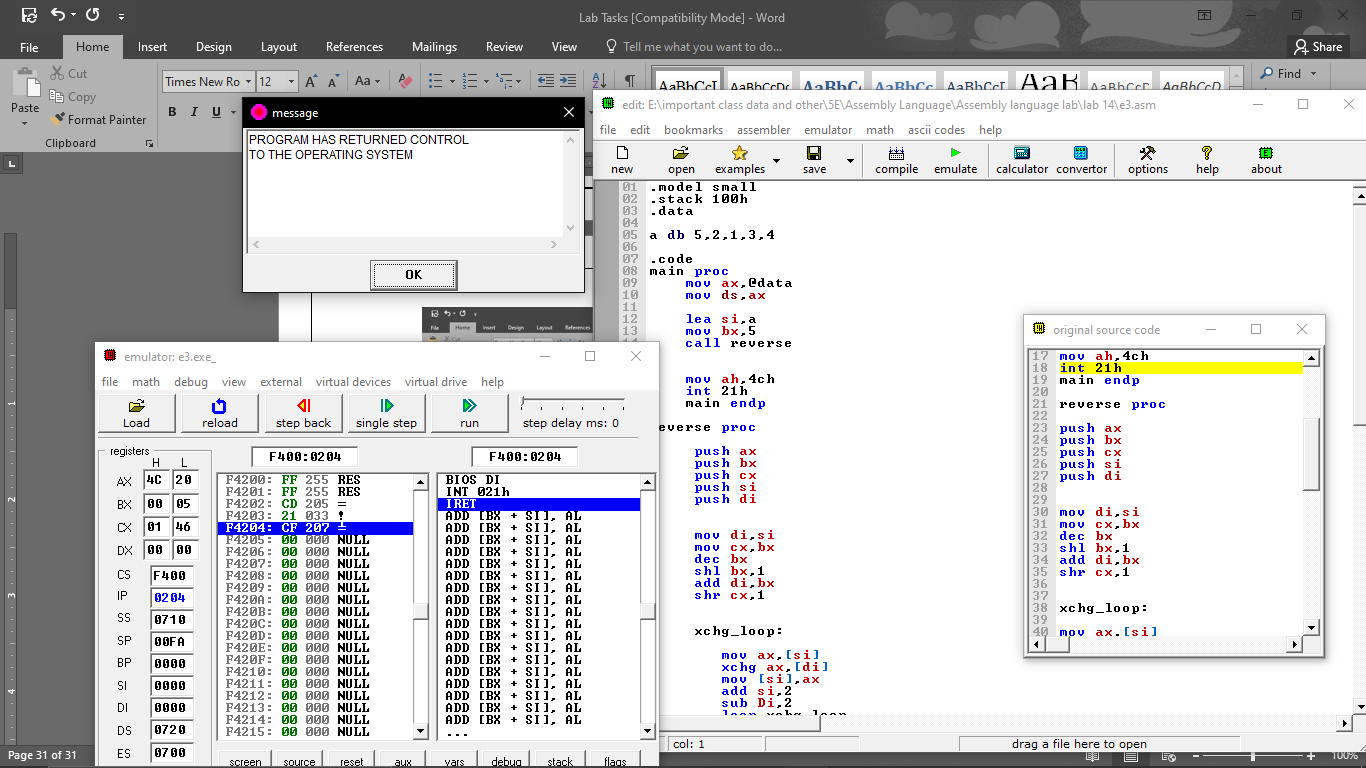


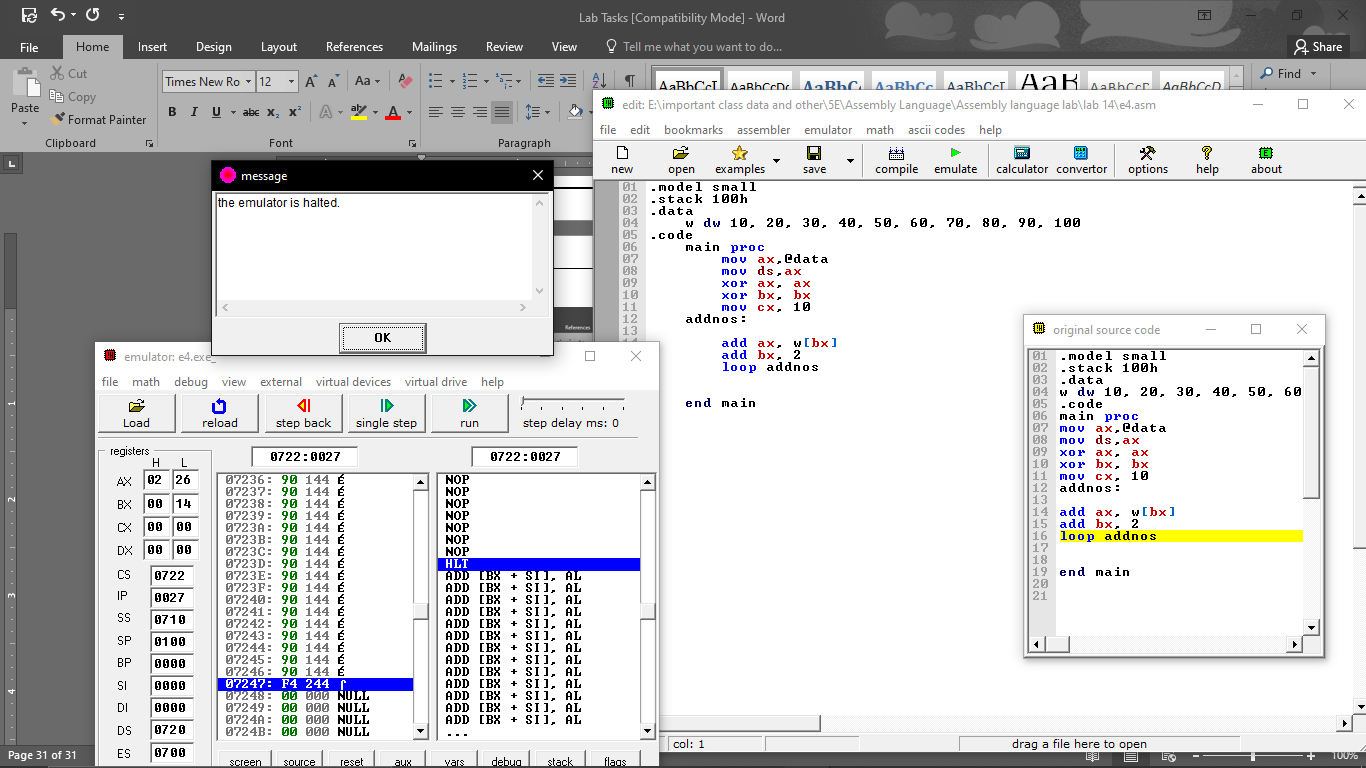
# Lab 14 – Array Addressing

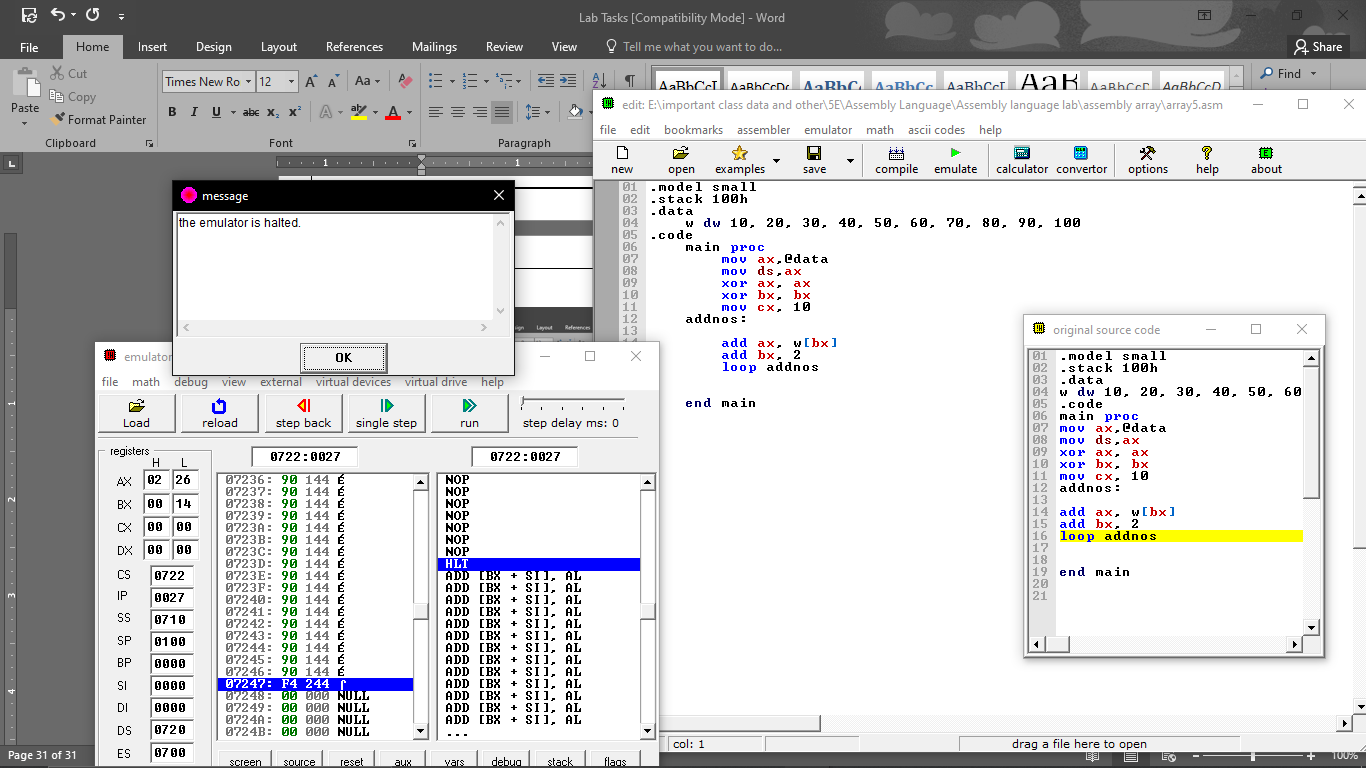












# Lab 15 – 2D Array Addressing

