CSC 3210

Computer Organization and Programming

Lab Work 4

Dr. Zulkar Nine

mnine@gsu.edu

Georgia State University

Fall 2021

Learning Objective

- Assembly Language Template
- Assembler Directives
- Writing and Debugging a program
- Watch registers and flag values

Lab Work 4 Instructions

- Lab 4: Write and run a program that can evaluate an expression (10 points)
- More details on Slide – 44 and 45

Due Date: Posted on iCollege

Disclaimer

- The process shown in these slides might not work in every single computer due to Operating system version, Microsoft Visual Studio versions and everything.
- If you find any unusual error, you can inform the instructor.
- Instructor will help you resolve the issue.

Attendance!

A review on assembly language

Program Template

in Microsoft Visual Studio

Program Template

Program title, optional →

```
; Program template (Template.asm)
```

32-bit program directives →

```
.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
```

Data section, not always needed →

```
.data
     ; declare variables here
```

Code section, always needed →

```
.code
main PROC
     ; write your code here

     INVOKE ExitProcess,0
main ENDP
END main
```

Assembler Directives

```
; AddTwo.asm - adds two 32-bit integers

.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.code
main PROC
    mov  eax,5      ; move 5 to the EAX register
    add  eax,6      ; add  6 to the EAX register

    INVOKE ExitProcess,0
main ENDP
END main
```

.386 directive, identifies this as a 32-bit program that can access 32-bit registers and addresses.

.model selects the program's memory **model (flat),** and identifies the **calling convention (named stdcall)** for procedures. Ex. Windows API

.stack aside **4096** bytes of storage for the **runtime stack**, which every program must have. Size of a memory page.

Assembler Directives

```
; AddTwo.asm - adds two 32-bit integers

.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.code
main PROC
    mov   eax,5       ; move 5 to the EAX register
    add   eax,6       ; add  6 to the EAX register

    INVOKE ExitProcess,0
main ENDP
END main
```

ExitProcess, declares **a prototype** for the ExitProcess function.
- A prototype consists of the function name, the PROTO keyword, a comma, and a list of input parameters.
- The input parameter for ExitProcess is named dwExitCode.

.**CODE** directive marks the beginning of the code area of a program, the area that contains executable instructions

The label main identifies the **program entry point** (main) and marks the address at which the program will begin to execute.

Assembler Directives

```
; AddTwo.asm - adds two 32-bit integers

.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.code
main PROC
    mov  eax,5      ; move 5 to the EAX register
    add  eax,6      ; add  6 to the EAX register

    INVOKE ExitProcess,0
main ENDP
END main
```

INVOKE Calls on, the procedure ExitProcess, passing the arguments on the stack or in registers.

ENDP directive marks the end of a procedure. Our program had a procedure named main,

end directive marks the last line to be assembled, and it identifies the program entry point (main).

Writing and Debugging

A program

Writing & Debugging a Program

Exercise 1: Write and run a program to solve the following problem:

$$\mathbf{EAX = (ECX + EBX) - EDX}$$

- Assume that all the values are 32 bit and all the registers are 32 bit as well
- Store the following decimal values into the registers -
  - 15 to ECX, 15 to EBX and 31 to EDX
  - Evaluate the above expression
  - Store the result in EAX register

- To create a project for this exercise, follow the steps provided in next slides (similar to lab 2)
- To verify the result in EAX, follow the steps in the slides titled : "Showing **registers** and **flags**"

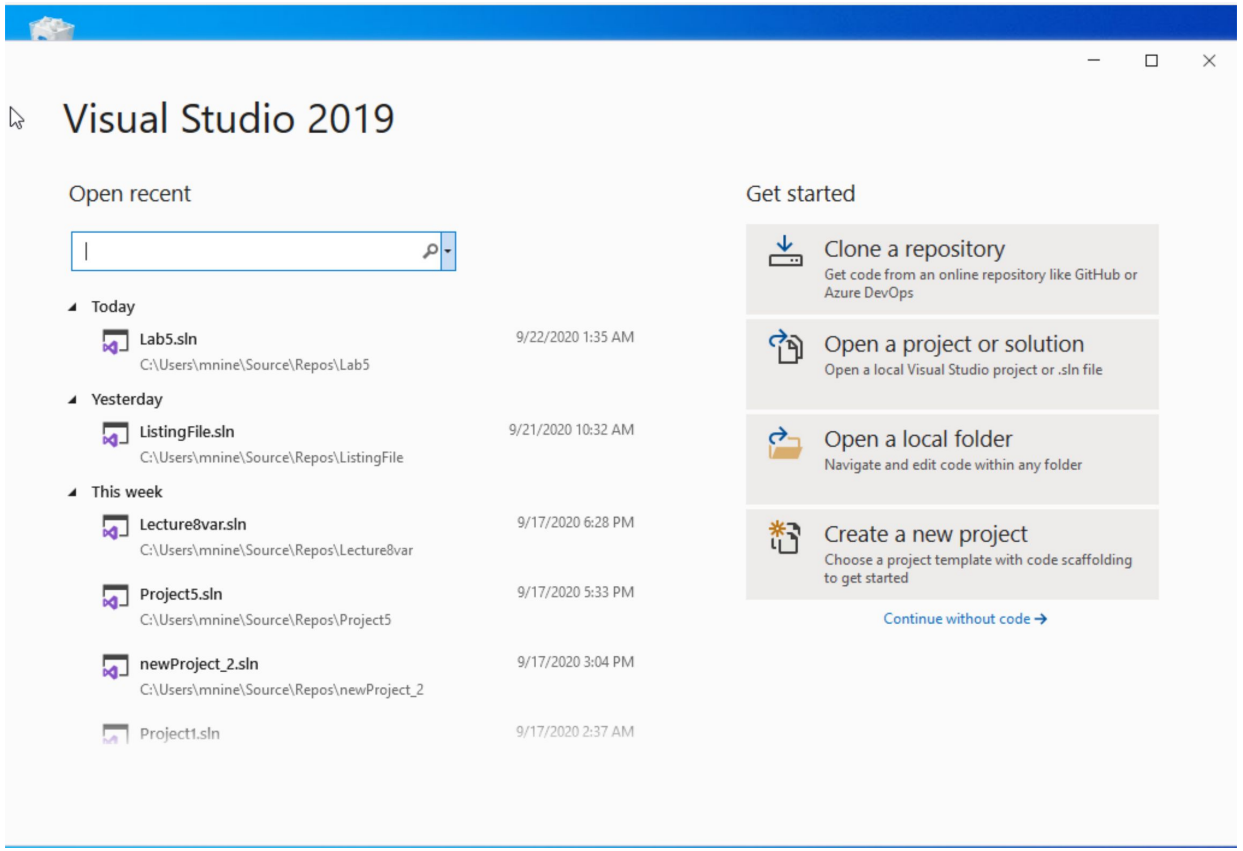
Steps to follow

- Follow the steps :
  - Step 1: Create a project
  - Step 2: Write your code to evaluate, **EAX = (ECX + EBX) – EDX**
  - Step 3: Build the project
  - Step 4: Debug the project

Step 1: Create a project (1)

(1) Start Visual Studio

(2) Click Create a new Project

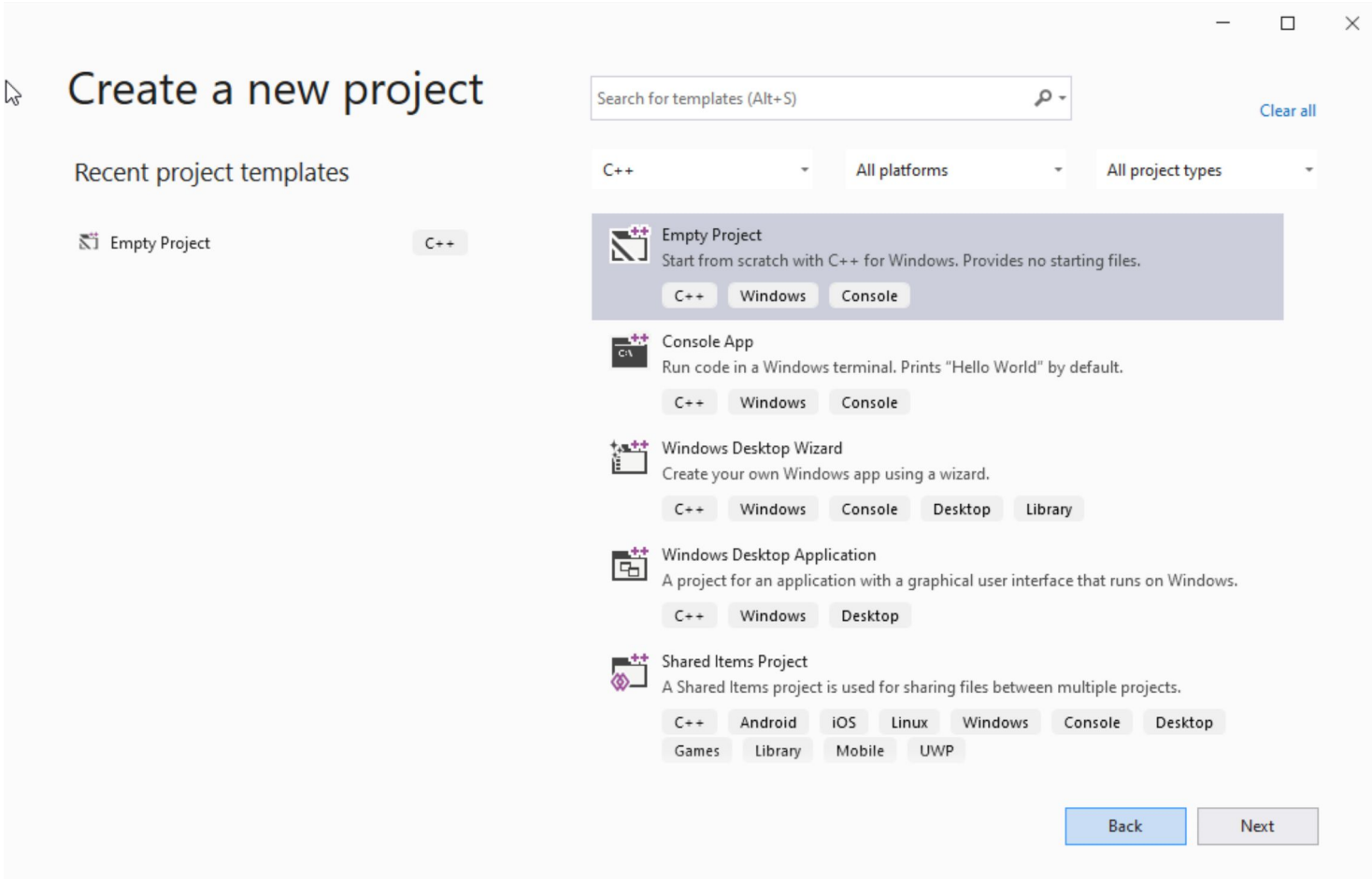


Step 1: Create a project (2)

(1) Select C++ as language

(2) Select Empty Project

(3) Click Next

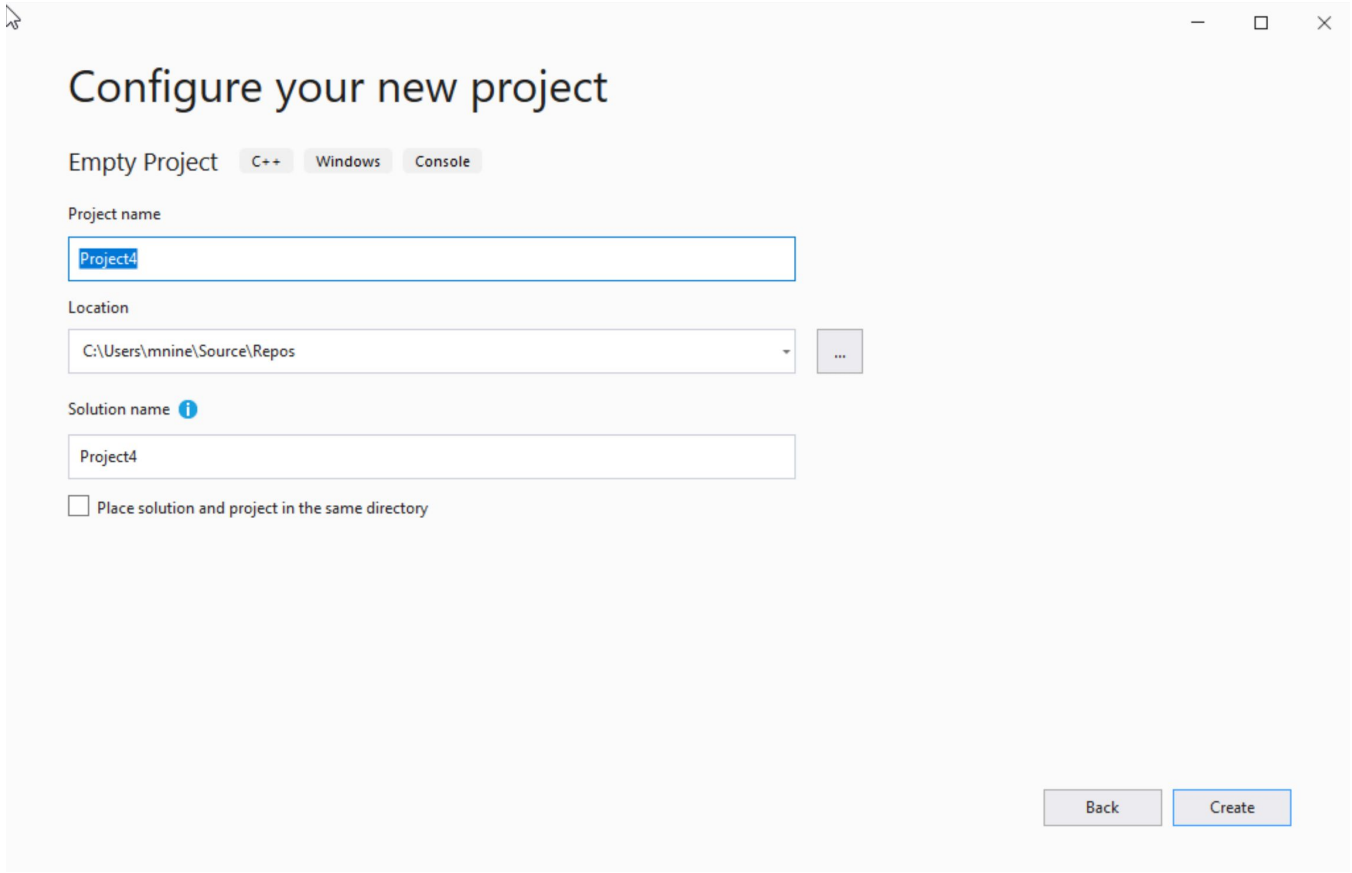


Step 1: Create a project (3)

(1) You can change the project name as you like

(1) Also you can change the project location

(2) Click Next

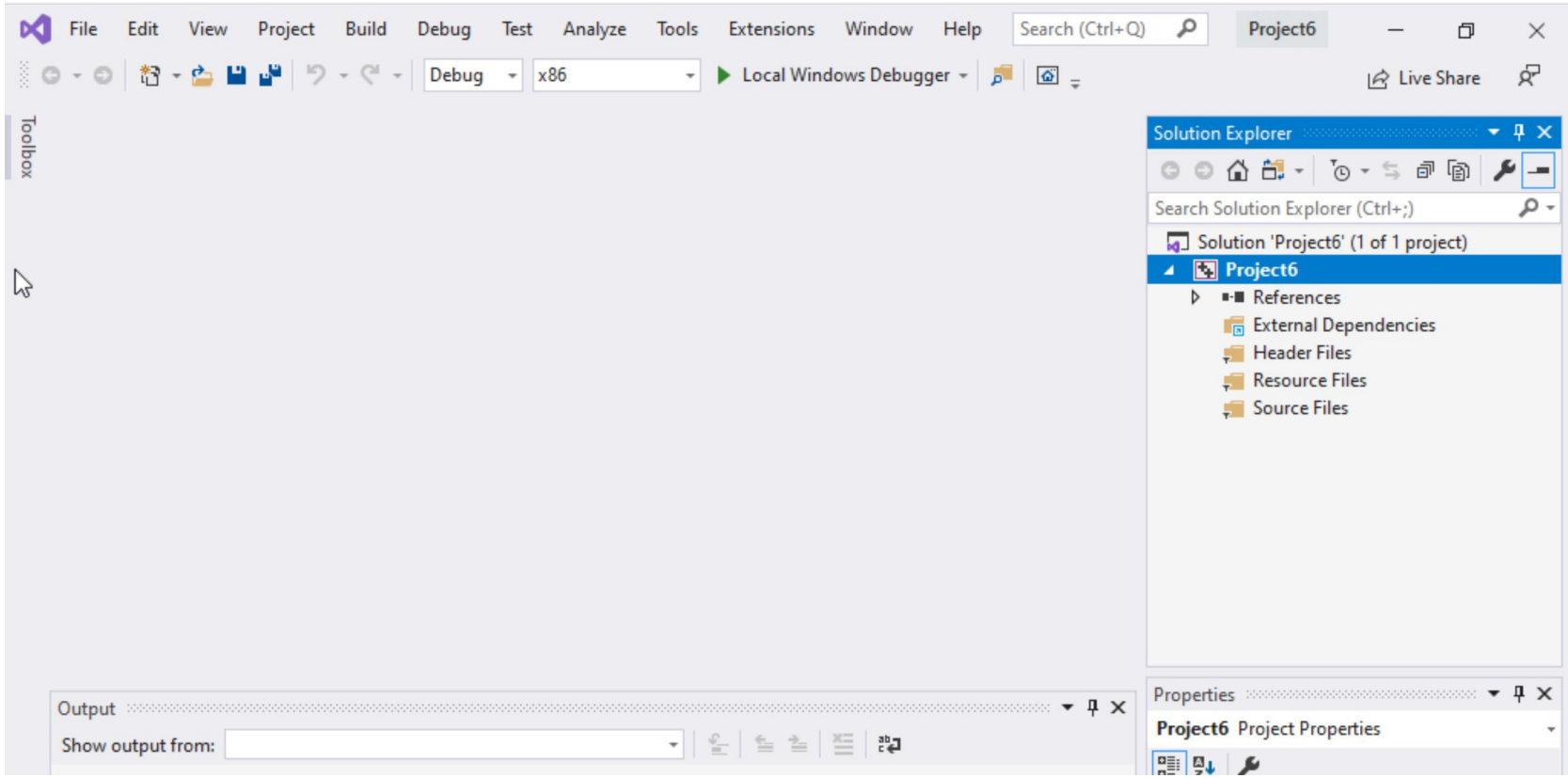


Step 1: Create a project (4)

Delete the

Following folders:

- Header files
- Resources Files, and
- Source Files



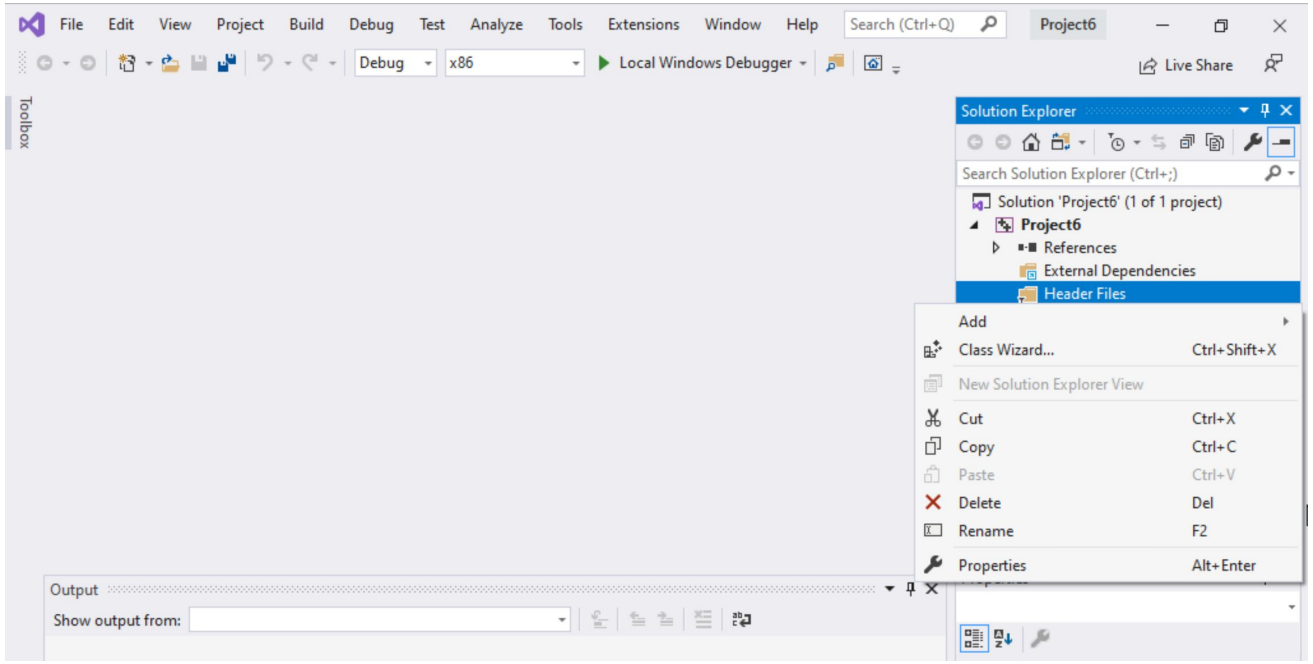
Step 1: Create a project (5)

To delete :

Select the folders

Right click on it

Select delete



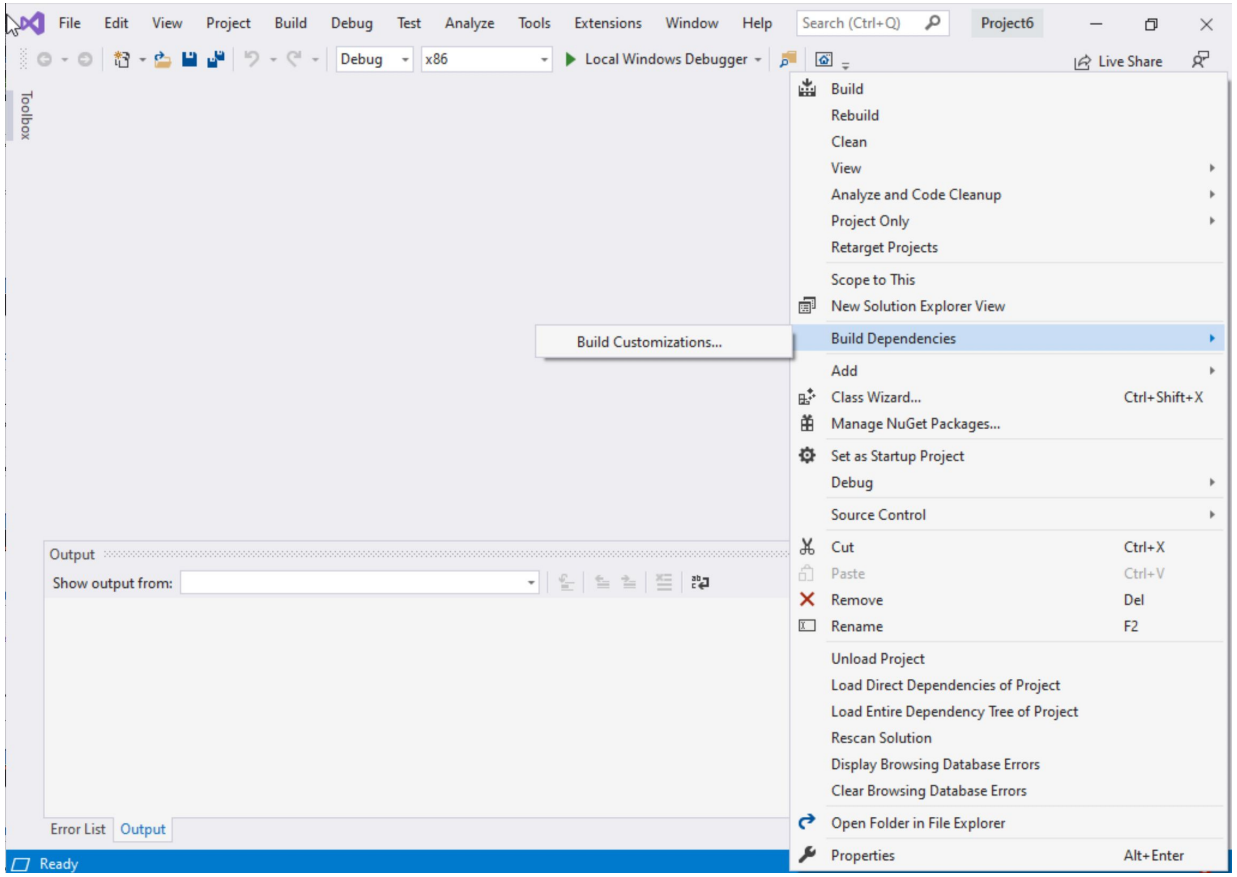
Step 1: Create a project (6)

Select Project Name on solution explorer

Right click on it

Go to Build Dependencies

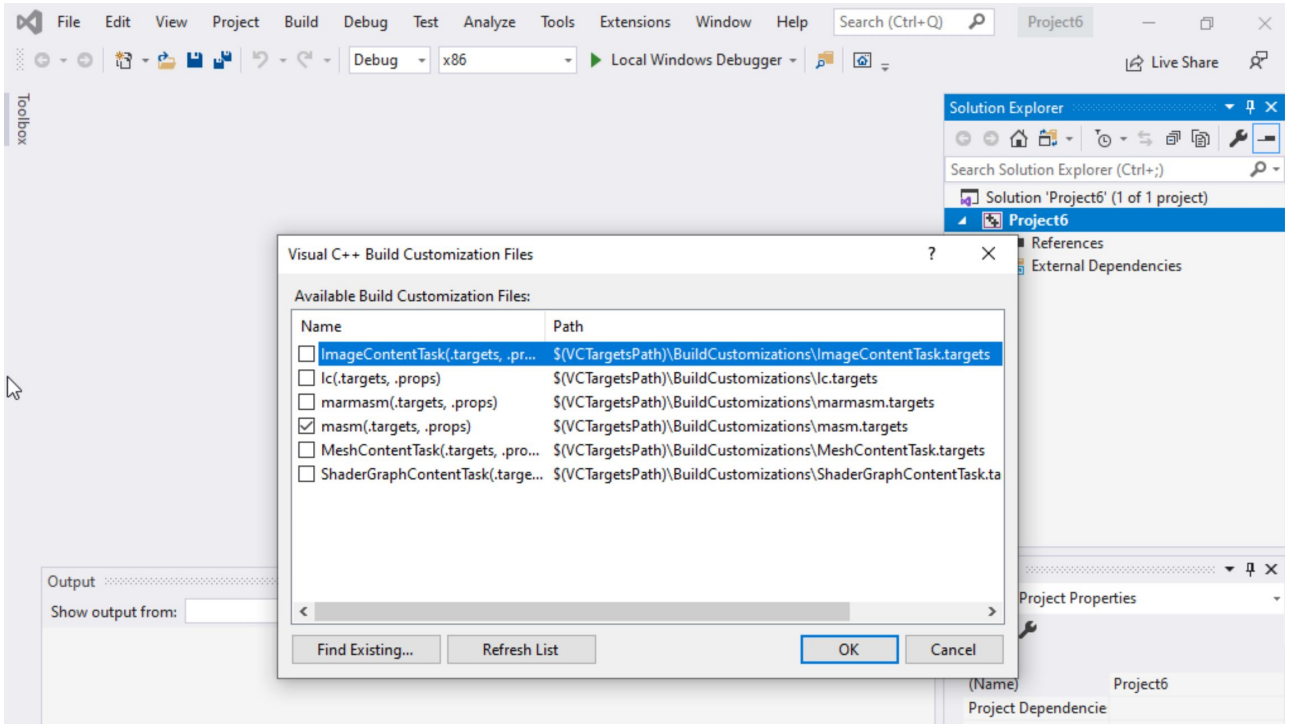
Click on Build Customizations



Step 1: Create a project (7)

Select mash(.target, .props)

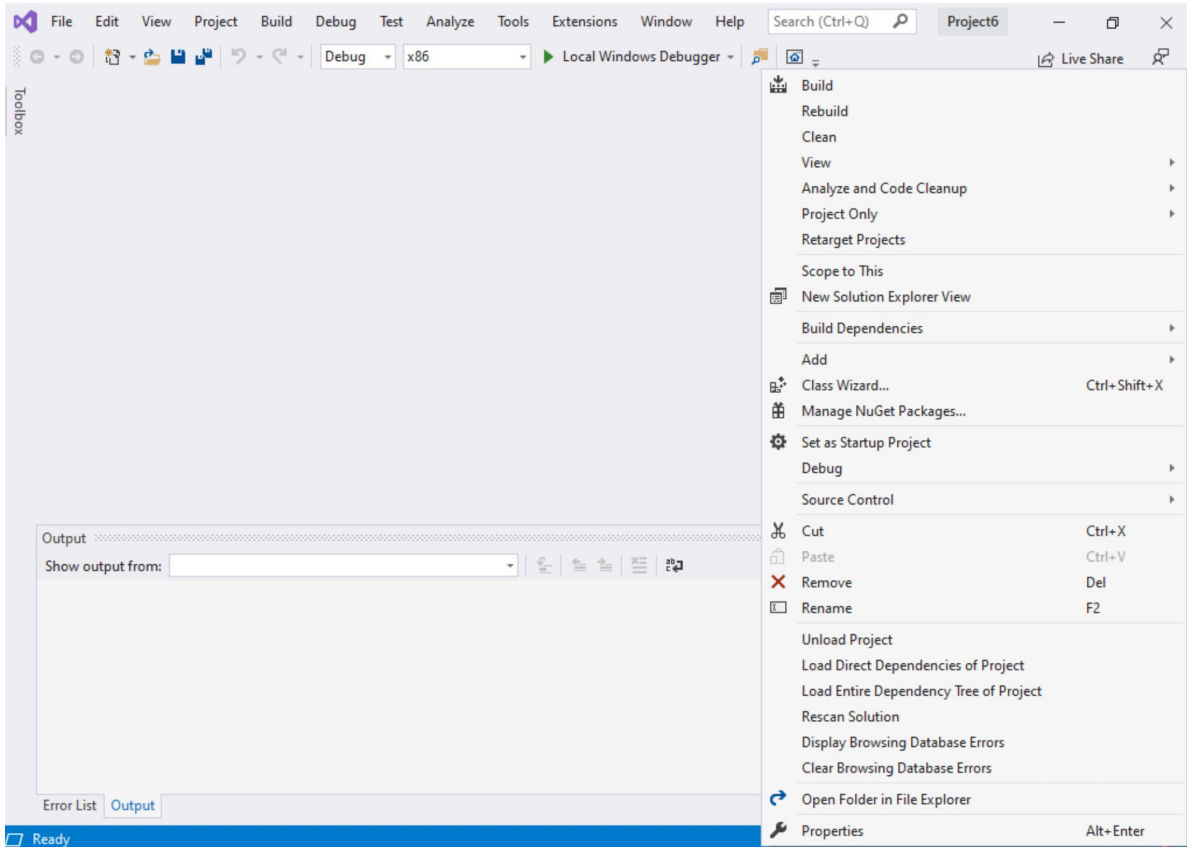
Click ok



Step 1: Create a project (8)

Right click on the Project name in the solution explorer

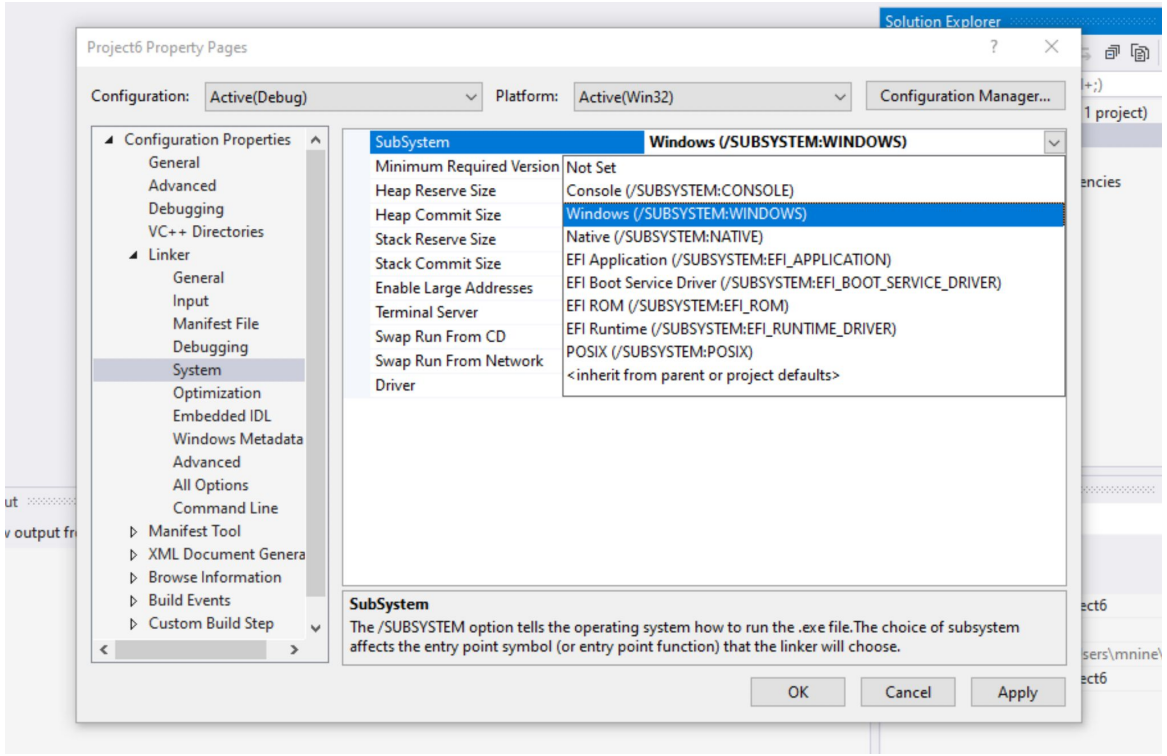
Click properties



Step 1: Create a project (9)

Select Windows(/SUBSYSTEM:WINDOWS)

Click OK



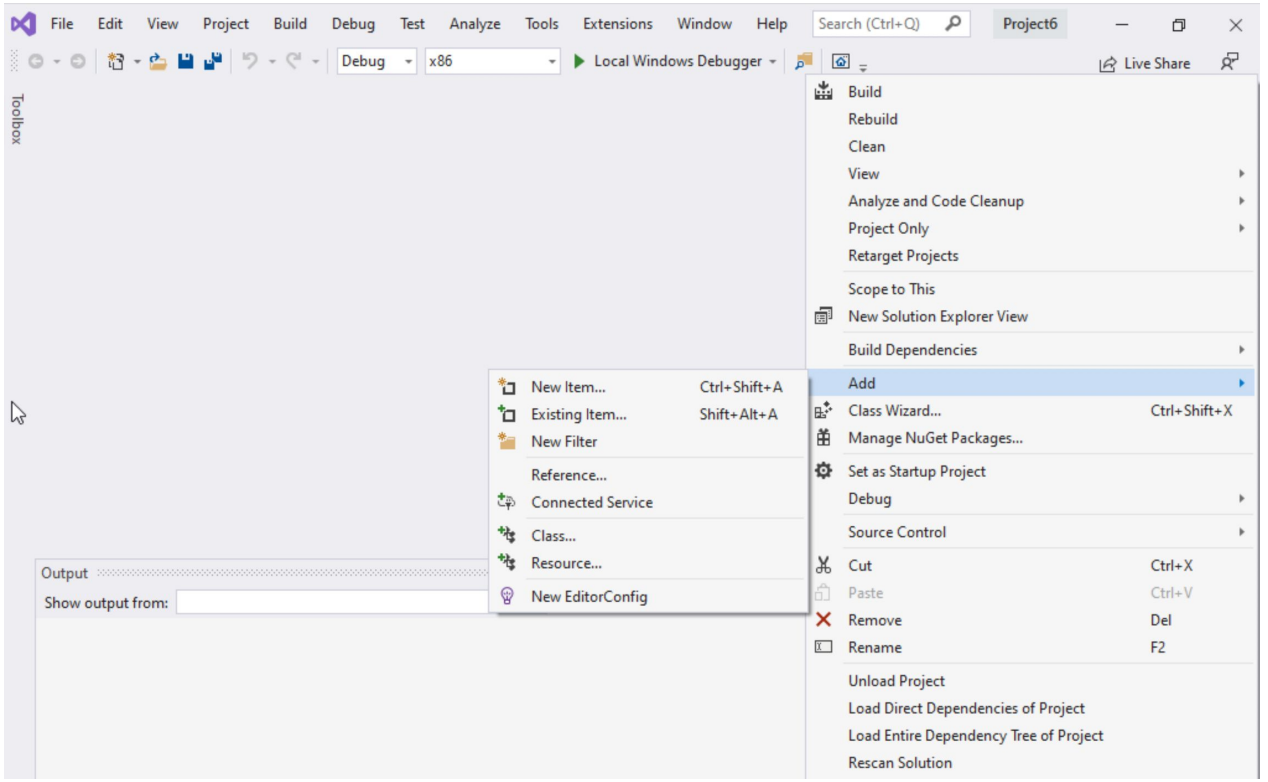
Step 1: Create a project (10)

Select Project name on solution explorer

Right click on it

Expand Add

Choose New Item

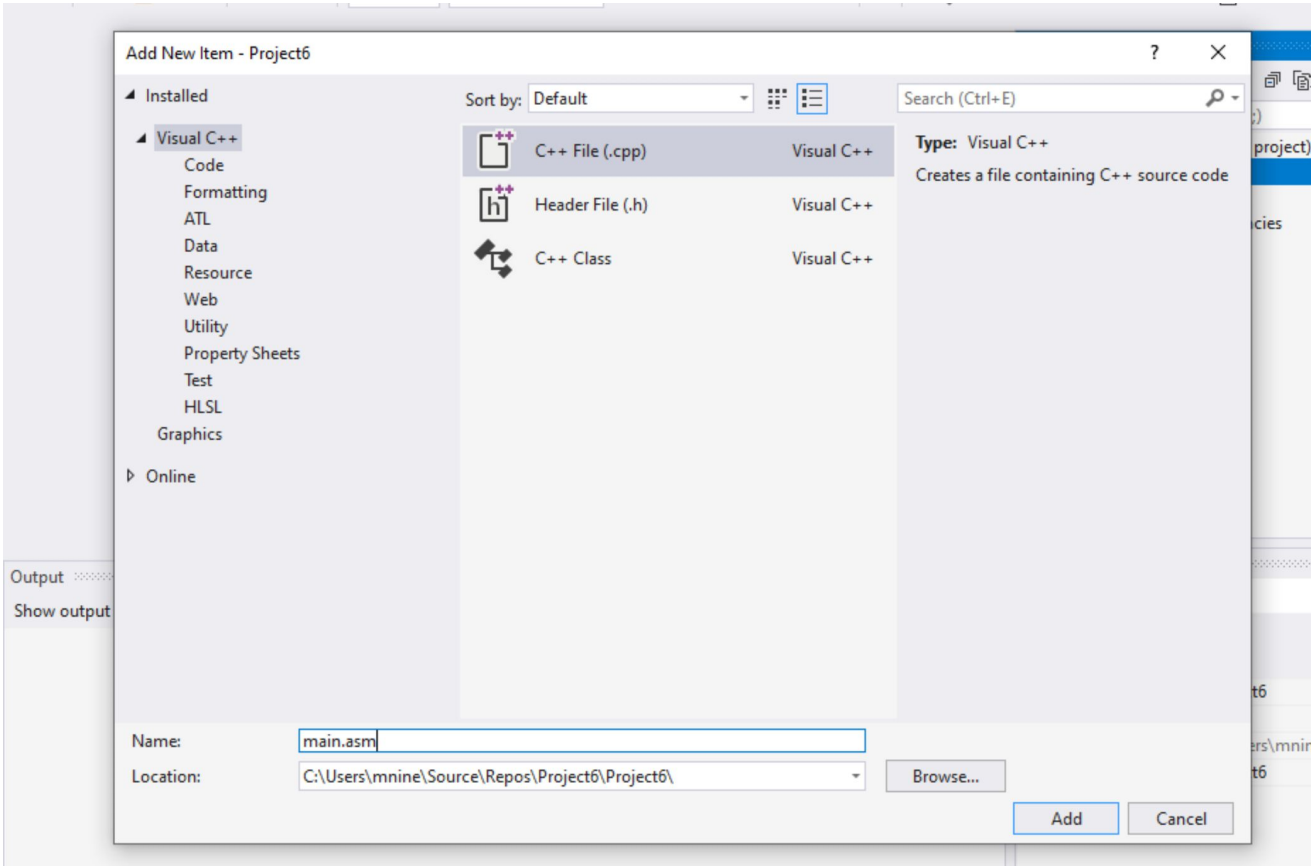


Step 1: Create a project (11)

Select C++ File(.cpp)

Name: main.asm

Click Add

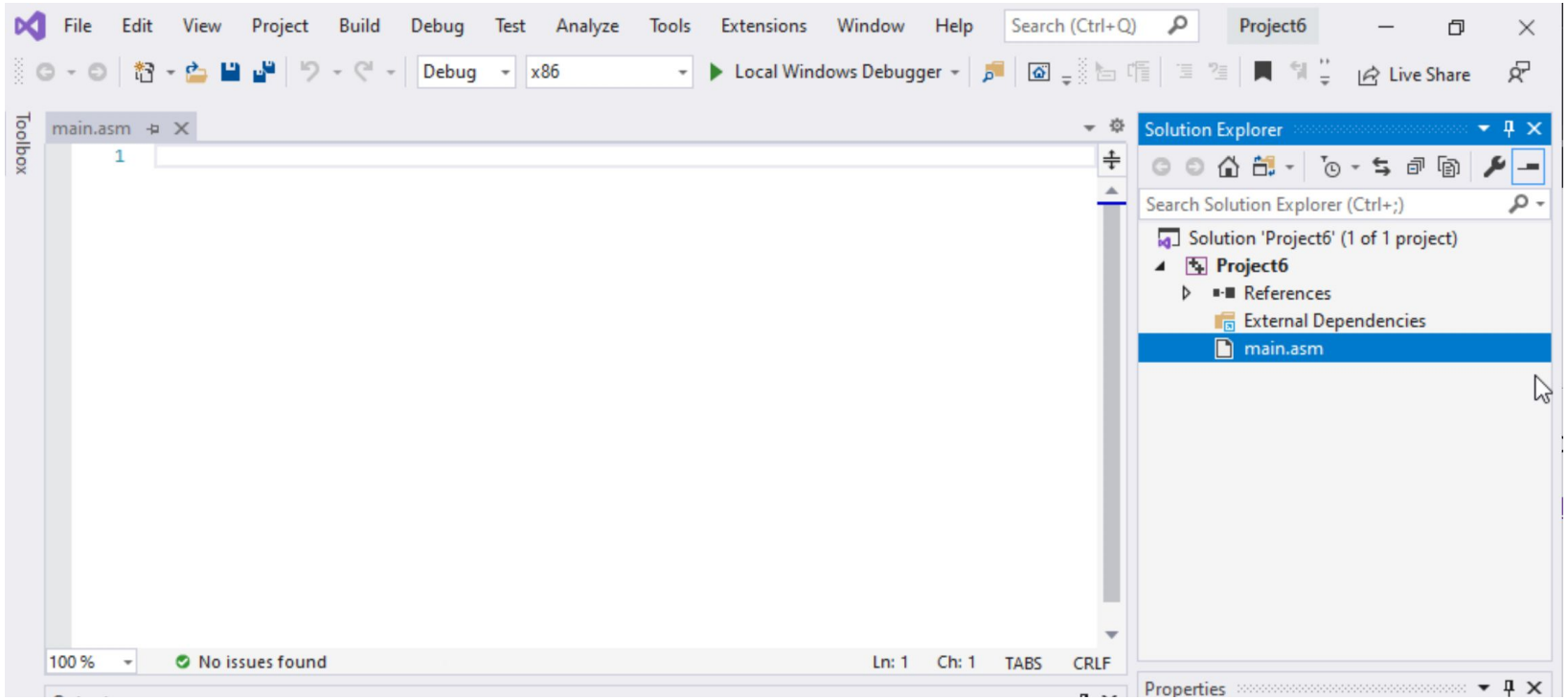


Step 1: Create a project (12)

Select main.asm

Add your code

In the main.asm File.



Step 2: Write your code to evaluate, EAX = (ECX + EBX) – EDX

4. Make the required changes:

EAX = (ECX + EBX) – EDX

- **Move to ECX the value 15**
- **Move to EBX the value 15**
- **Move to EDX the value 31**
- **Add EBX to ECX**
- **Subtract EDX from ECX (😳)**
- **Move the result in ECX to EAX**
- **Done**

Replace this

```
; AddTwo.asm - adds two 32-bit integers.
; Chapter 3 example

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.code
main proc
    mov eax,5
    add eax,6
    invoke ExitProcess,0
main endp
end main
```

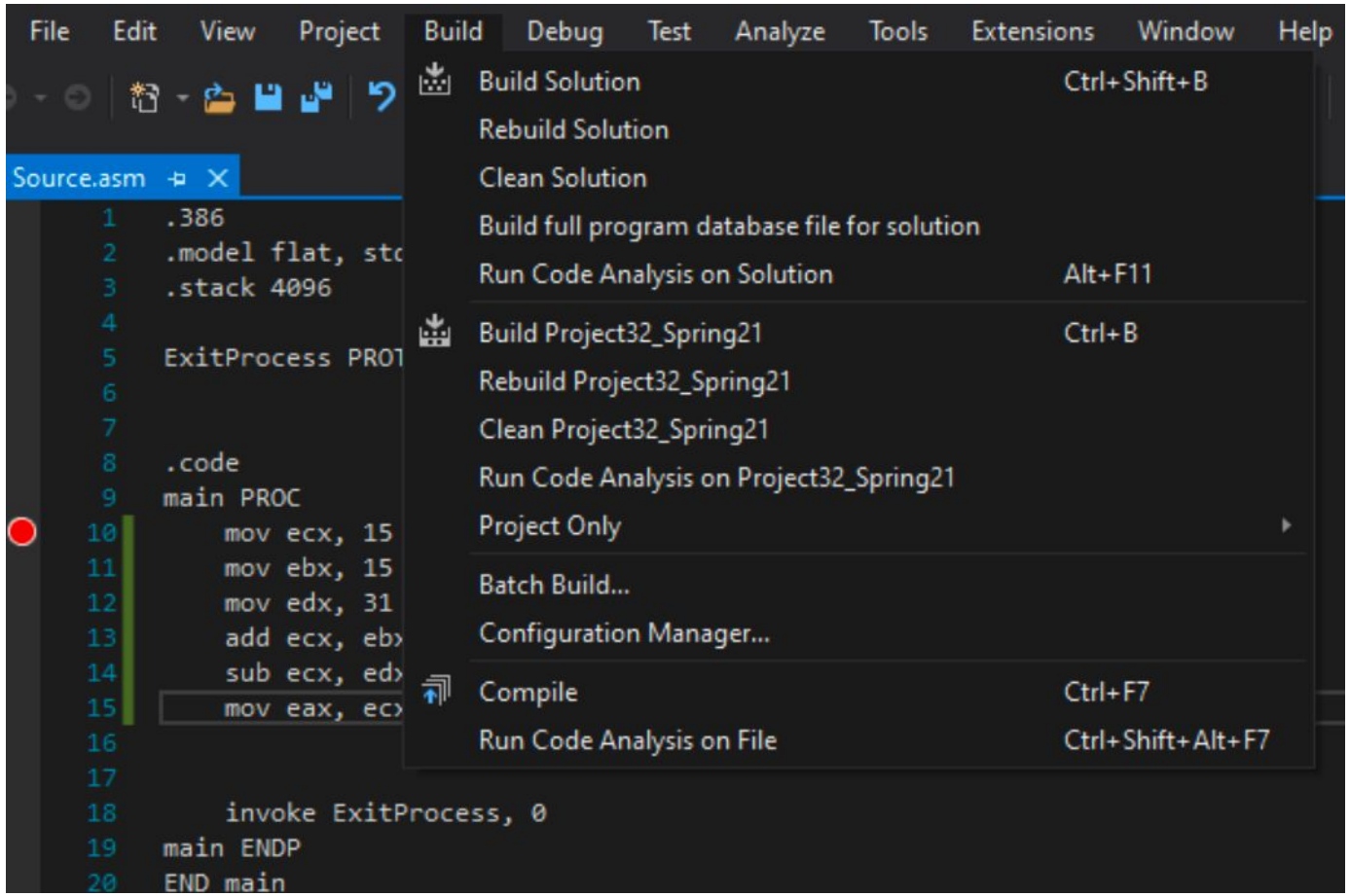
Step 2: The code

```
; AddTwo.asm - adds two 32-bit integers.
; Chapter 3 example
.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.code
main proc
    mov ecx,15
    mov ebx,15
    mov edx,31
    add ecx,ebx
    sub ecx,edx
    mov eax,ecx
    invoke ExitProcess,0
main endp
end main
```

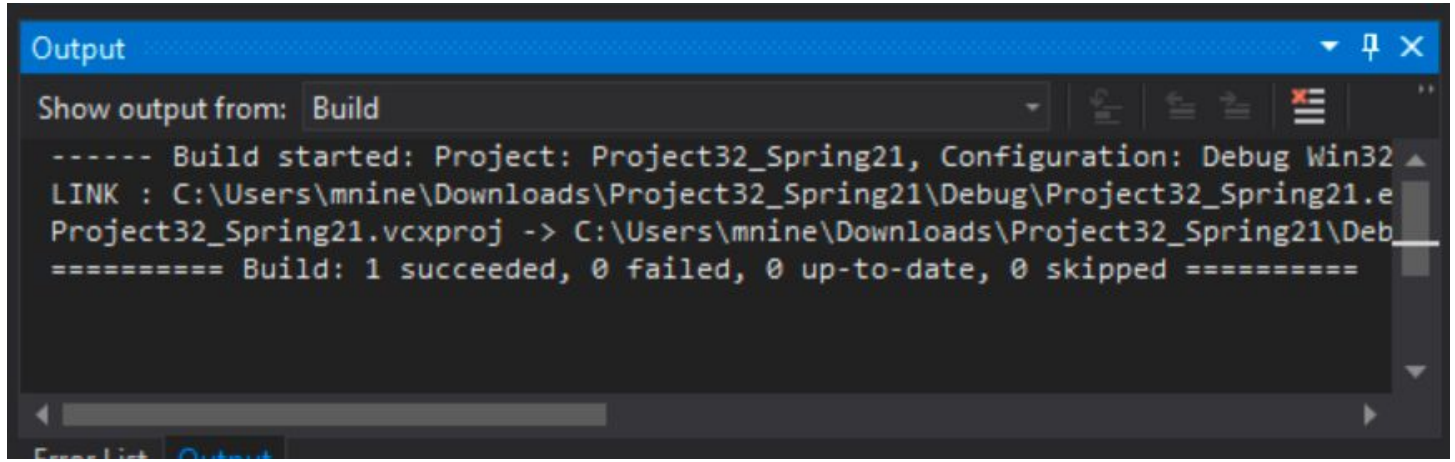
Step 3: Build the Project (1)

- Select **Build Project** from the **Build menu**.
- This will **assemble** and **link** your program and create an executable file.



Step 3: Build the Project (2)

- You should see messages like the following in your output window, indicating the build progress:

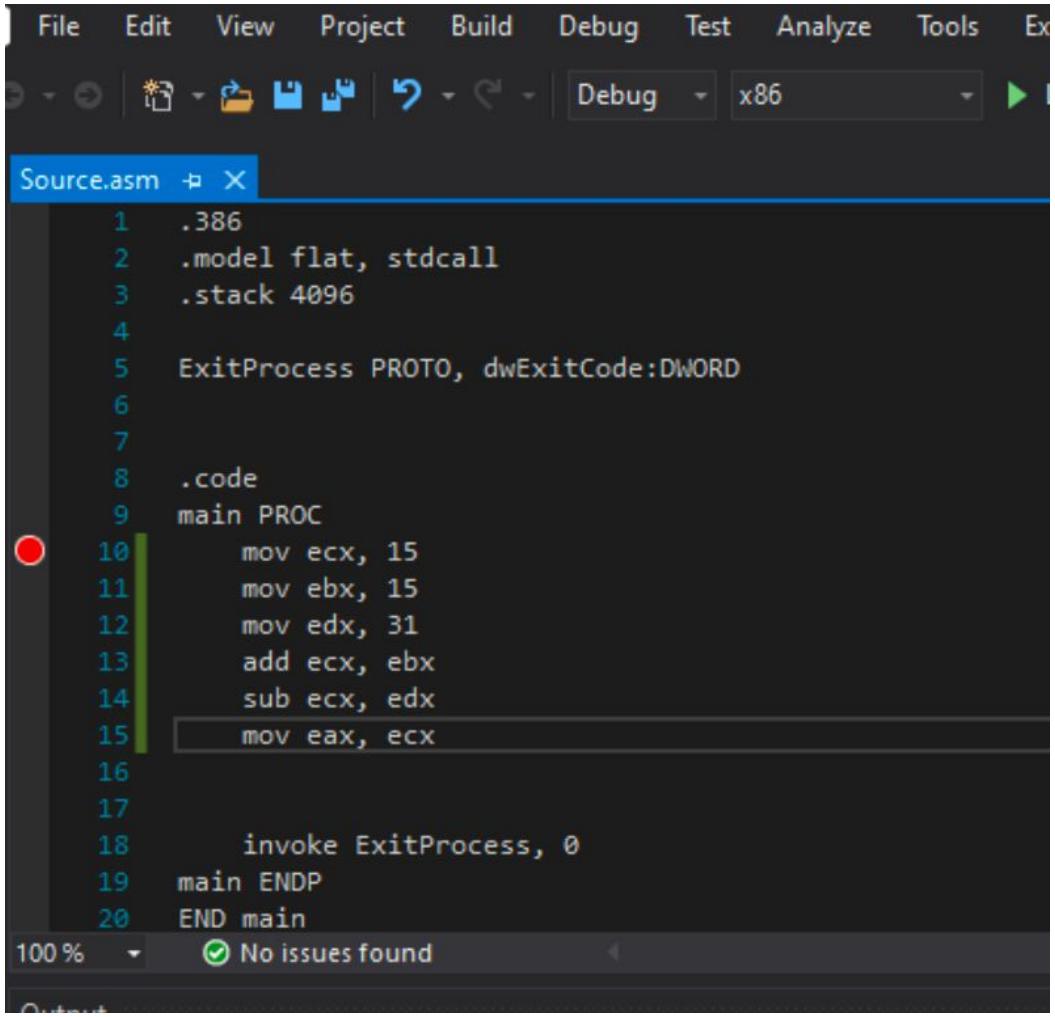


- You should see the message in the last line:

=====Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

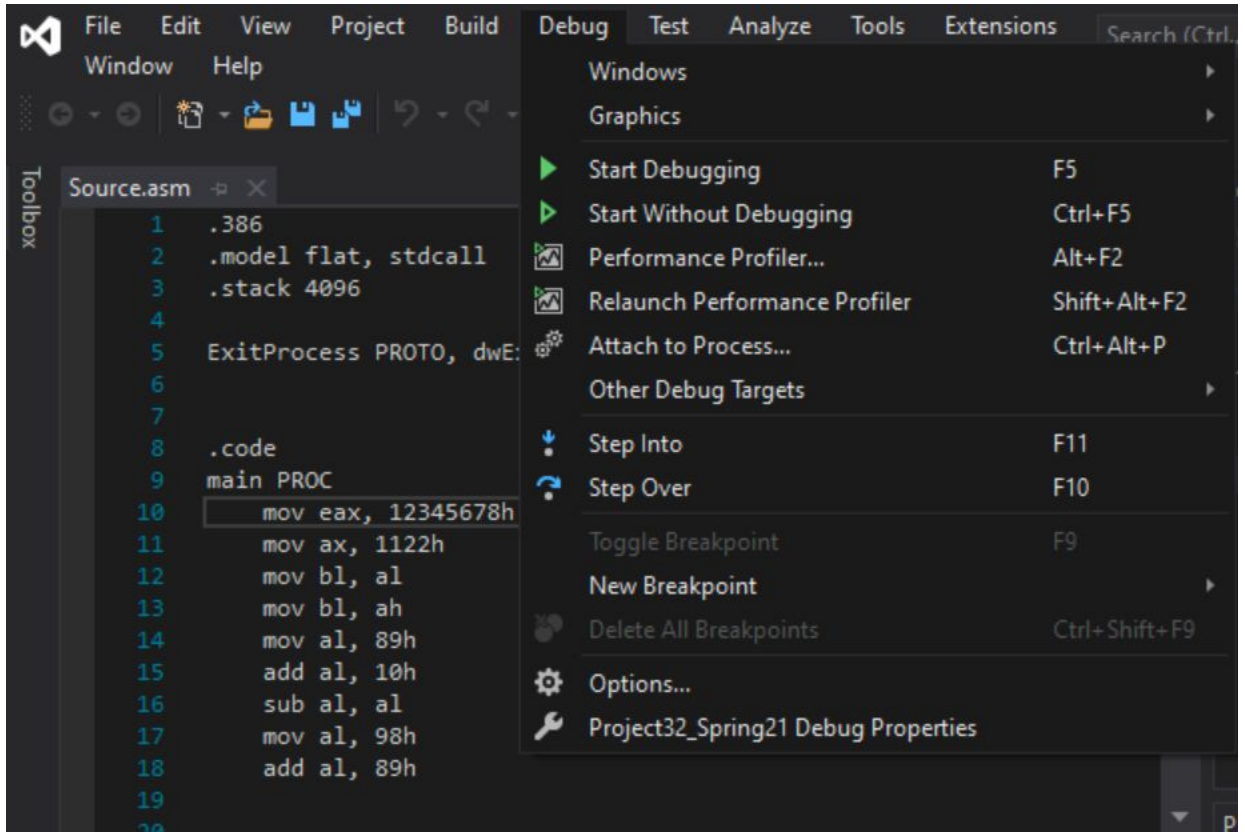
Step 4: Debug the project (1)

- Set a breakpoint first.
- **Set a breakpoint** on a program statement by clicking the mouse in **the vertical gray bar** just to the left of the code window.
- A **large red dot** will mark the breakpoint location.
- In this case, set a breakpoint at Line 10.



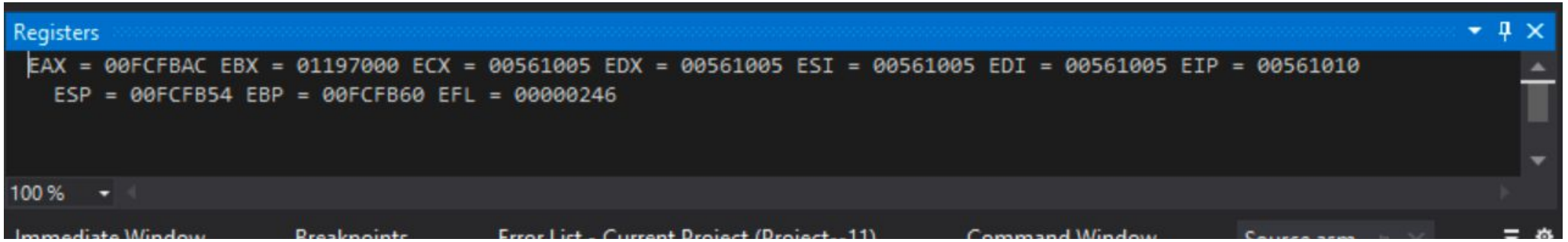
Step 4: Debug the project (2)

- **Run** the Program by selecting **Start Debugging** from the **Debug menu.**



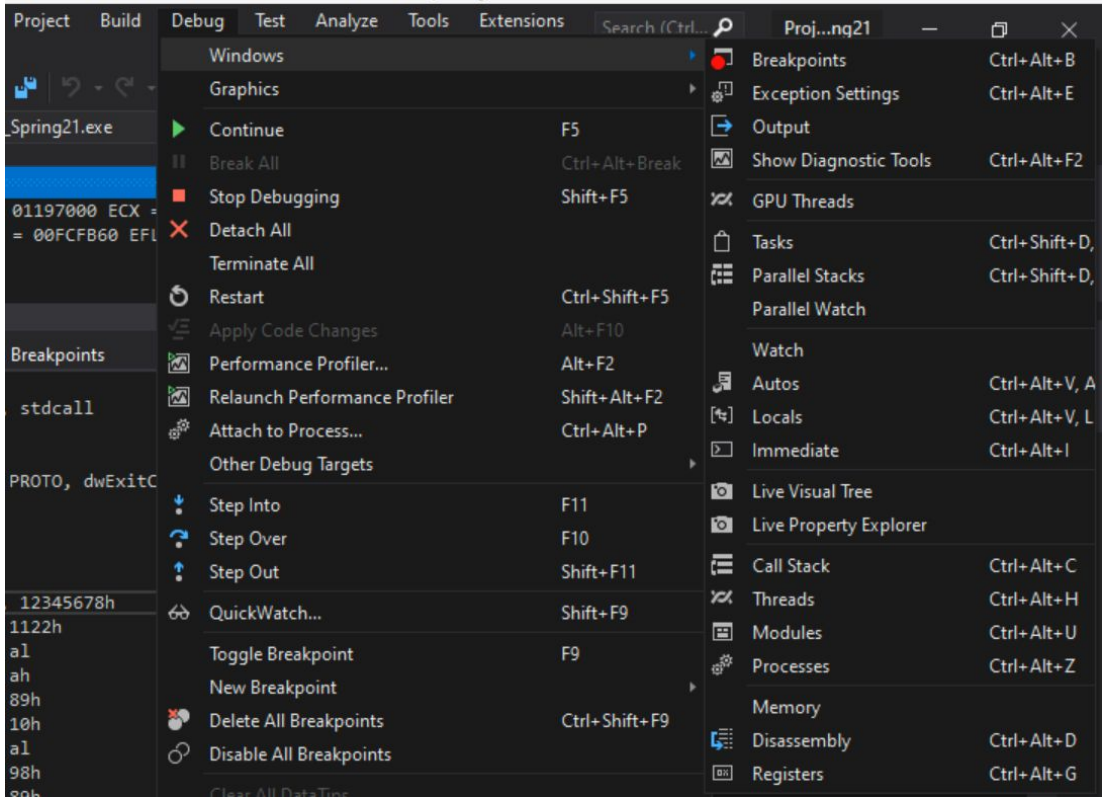
Step 4: Debug the project (3)

- You should be able to see the register window :



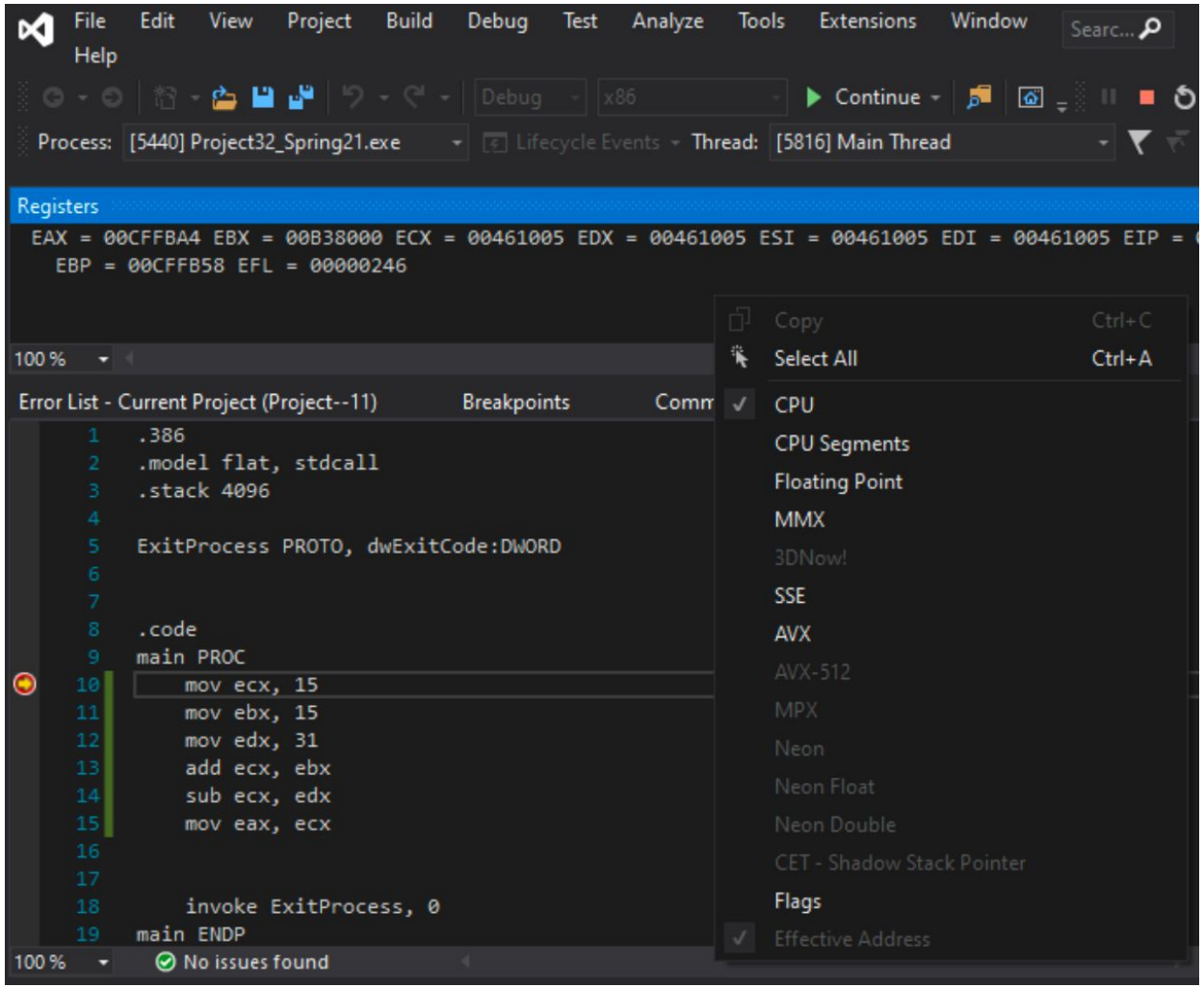
Step 4: Debug the project (4)

- If you don't see the register window: Go to : Debug ->Windows->registers.



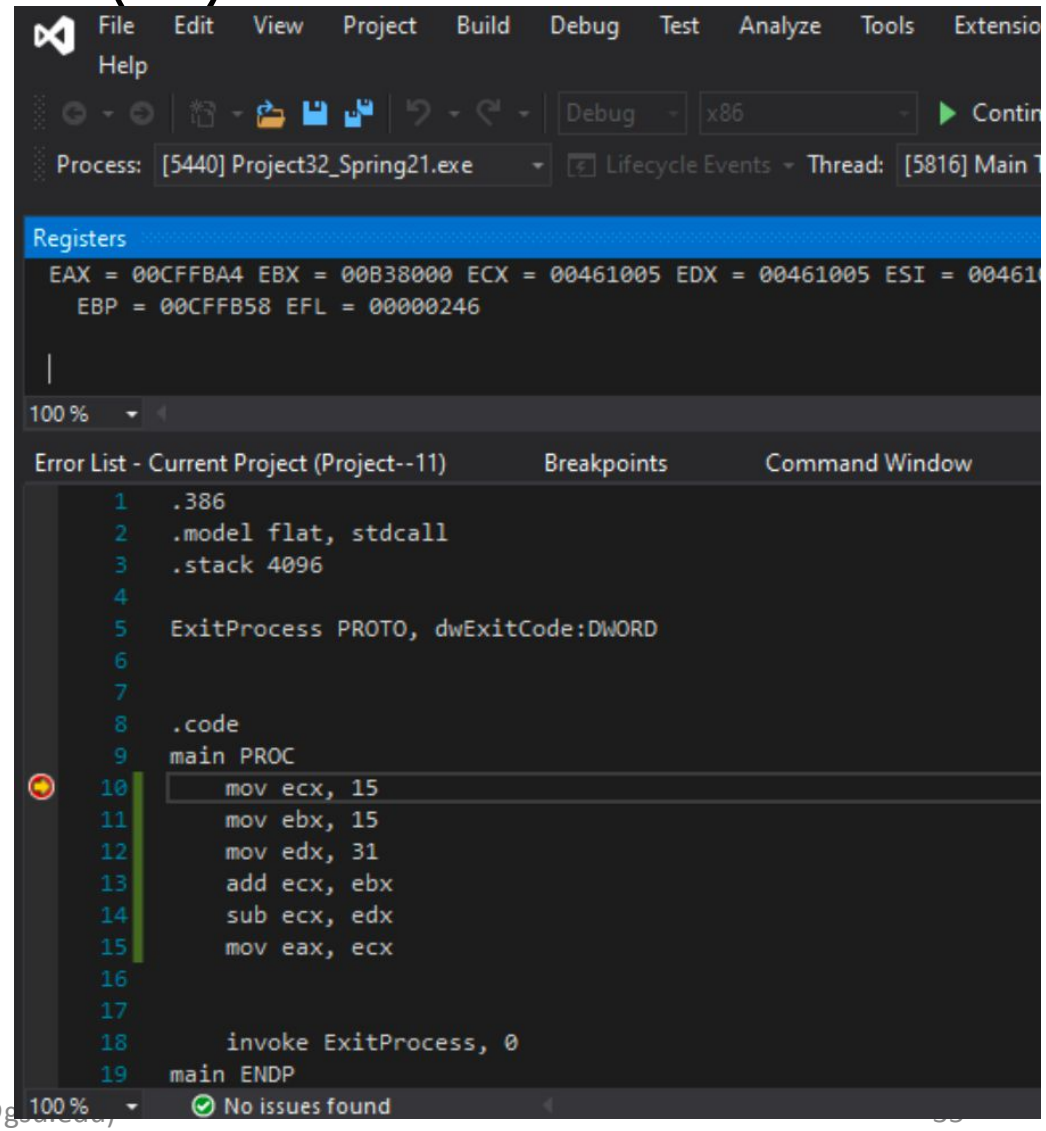
Step 4: Debug the project (5)

- Right click on the register window
- Turn on the EFLAGS in the register window.



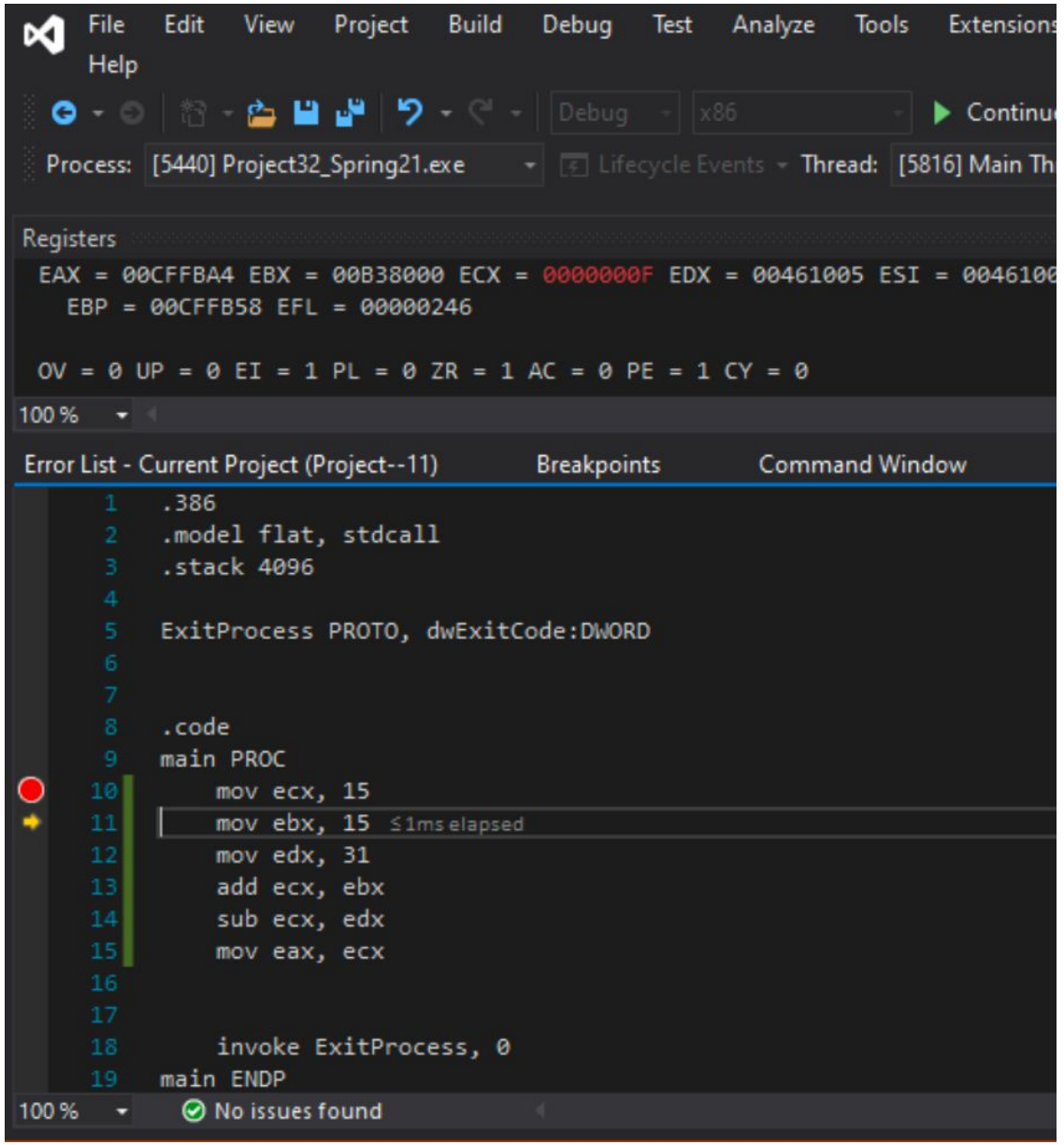
Step 4: Debug the project (6)

- Now the red dot (breakpoint) has a yellow pointer inside of it now.
- That means code execution halts at line 10 now.



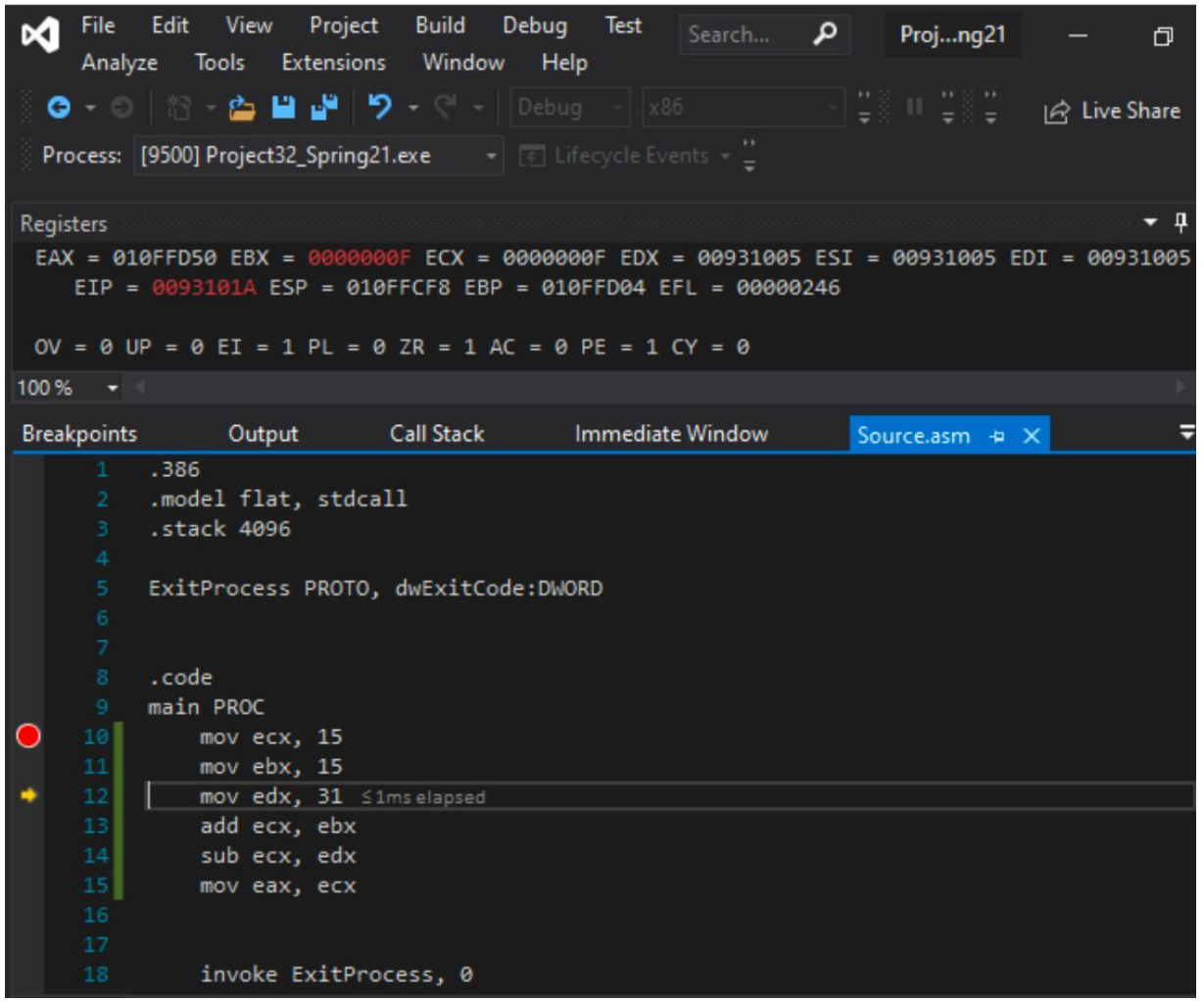
Debug the Project

- To execute line 10, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 11. That means line 10 is executed.
- Check the ECX register content.
- You can see the hexadecimal representation of decimal 15.



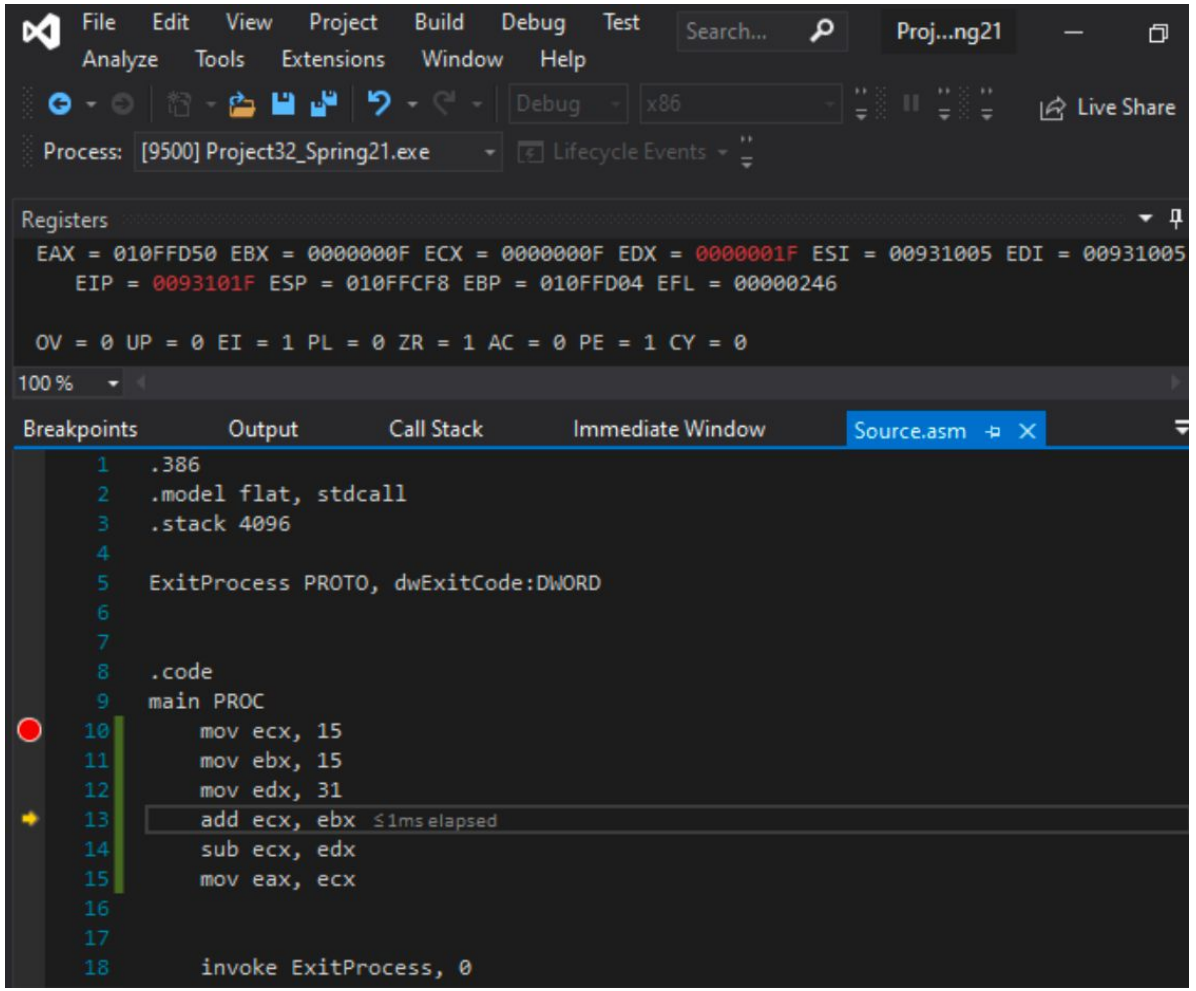
Debug the Project

- To execute line 11, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 12. That means line 11 is executed.
- Check the ECX register content.
- You can see the hexadecimal representation of decimal 15.



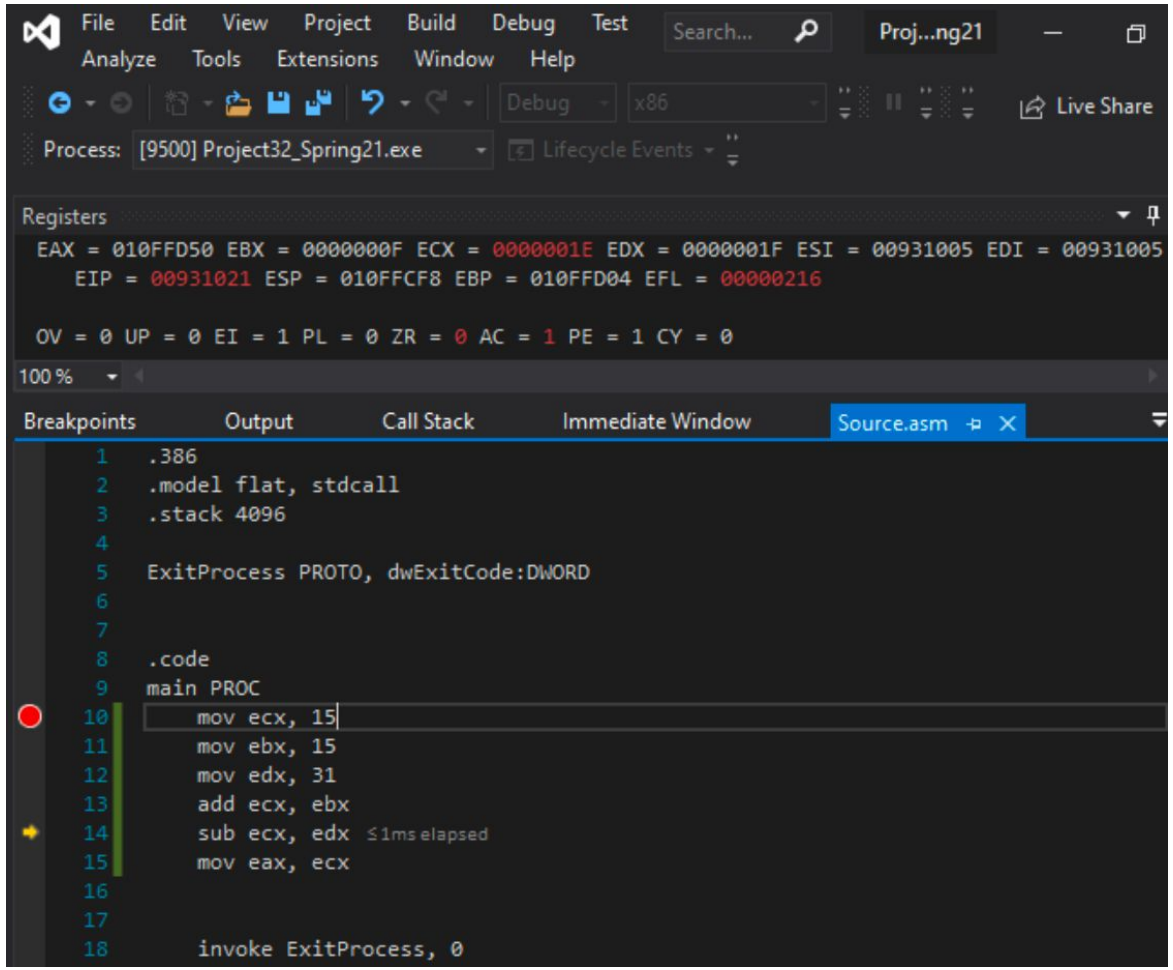
Debug the Project

- To execute line 12, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 13. That means line 12 is executed.
- Check the EDX register content.
- You can see the hexadecimal representation of decimal 31.



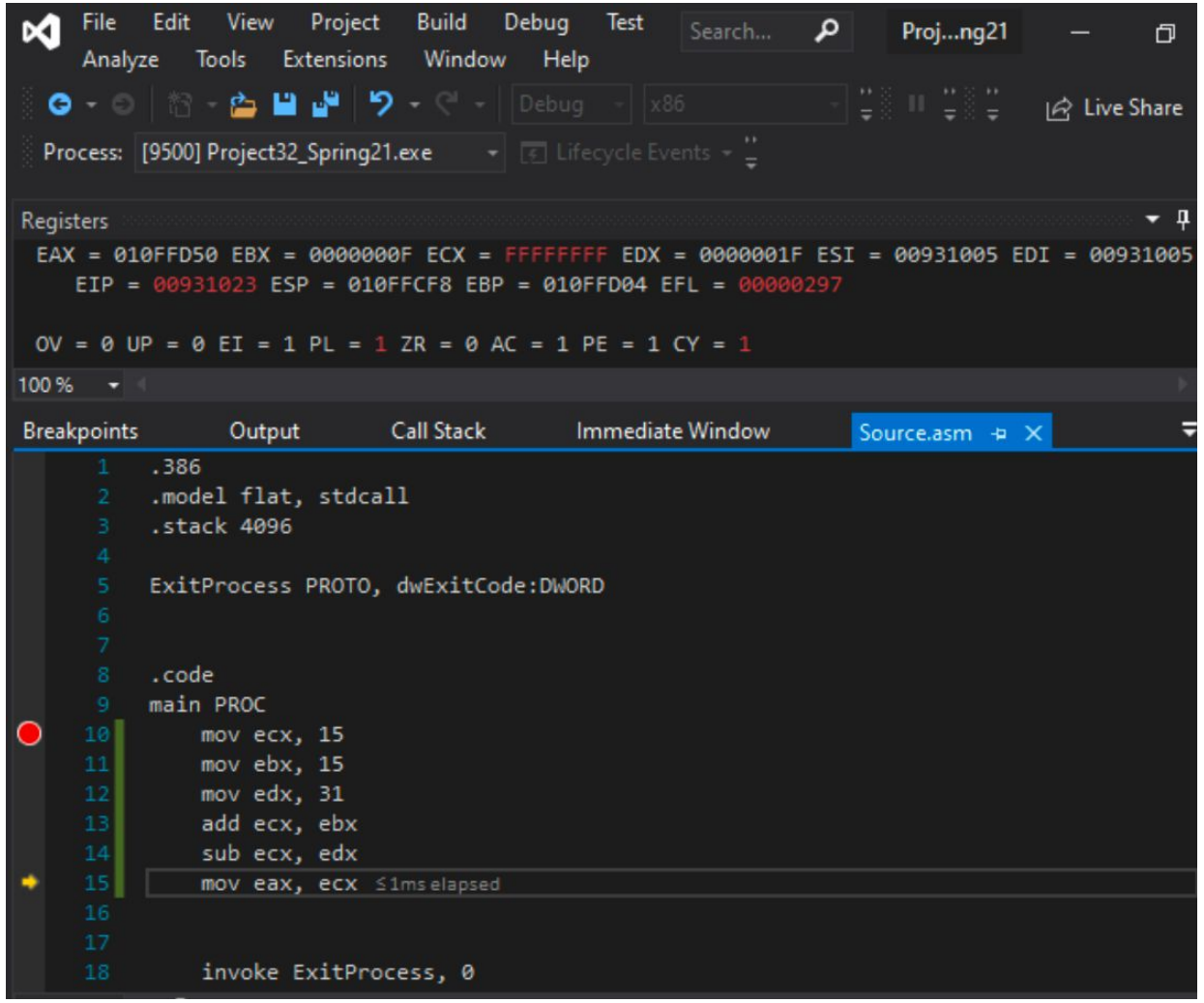
Debug the Project

- To execute line 13, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 14. That means line 13 is executed.
- Check the ECX register content.
- Add instruction performs (15+15) and stores the result in ECX in hexadecimal, 1E.



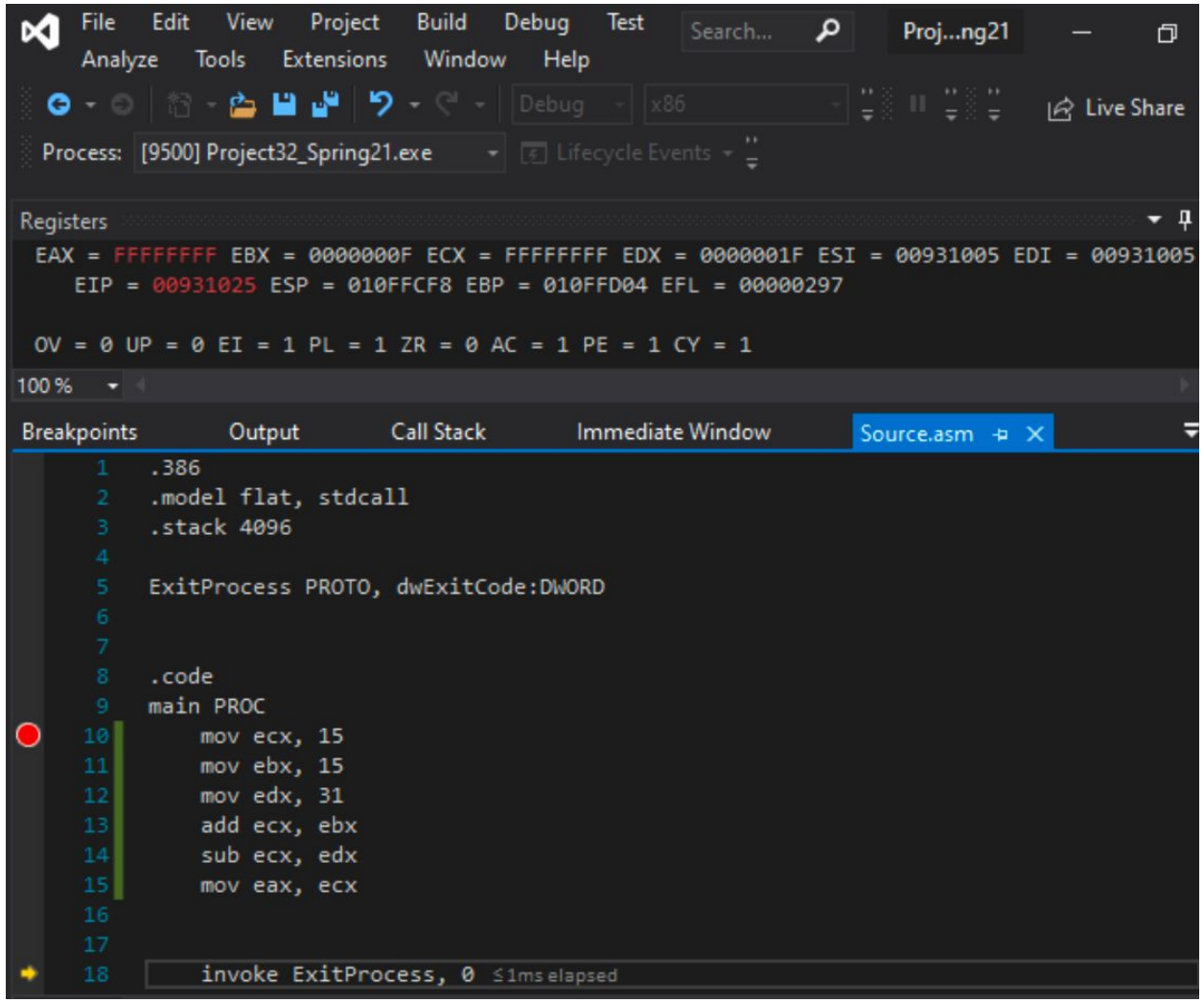
Debug the Project

- To execute line 14, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 15. That means line 14 is executed.
- Check the ECX register content and the flags.
- Sub instruction subtracts edx from ecx.
- That is (30-31) in decimal.
- Result = -1
- You can see the result in ECX = FFFFFFFF
- That is 2's complement of -1 in 32-bit.



Debug the Project

- To execute line 15, Select 'Step over' from debug menu
  - OR You can also use shortcut : Fn+F10
- Now the yellow pointer moved to line 16. That means line 15 is executed.
- Check the EAX register content.



Debug the Project

- After you reach the ”INVOKE ExitProcess, 0”
  - Do not click Step over or Fn+F10
- What is the status of the sign flag (PL) ?

Lab 4

Submission Instruction

Problem 1

- Assume that you have 8-bit registers.
- Write and run a program to evaluate the following expression:

  AL = (AL – DL) + CL - BL
  - Store the following decimal values into the registers
  - Store 245 (decimal) to AL register
  - Store 41 (decimal) to BL register
  - Store 11 (decimal) to CL register
  - Store 215 (decimal) to DL register
  - Evaluate the above expression
  - Result should be at AL register at the end

Submission Instruction

- There is an answer sheet attached to the lab (similar to lab-3b)
- Debug through each line of your code.
  - Execute the instruction
  - Take a screenshot of the code and register window
  - Record the line number, instruction, Register values in the answer sheet.
  - Also add the screenshot
  - Then explain the register contents. (similar to lab-3b)