

CSC 3210

Computer Organization and Programming

Lab Work 3

Dr. Zulkar Nine

mnine@gsu.edu

Georgia State University

Summer 2021

Lab Work 3 Instructions

- Lab 3(a): Load and build an existing project (3 points)
- Lab 3(b): Debug the Project (5 points)
- Lab 3(c): Math Problems (2 points)

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!

Lab3(a)

Load and Build an existing project

in Microsoft Visual Studio

Lab 3(a) Instructions

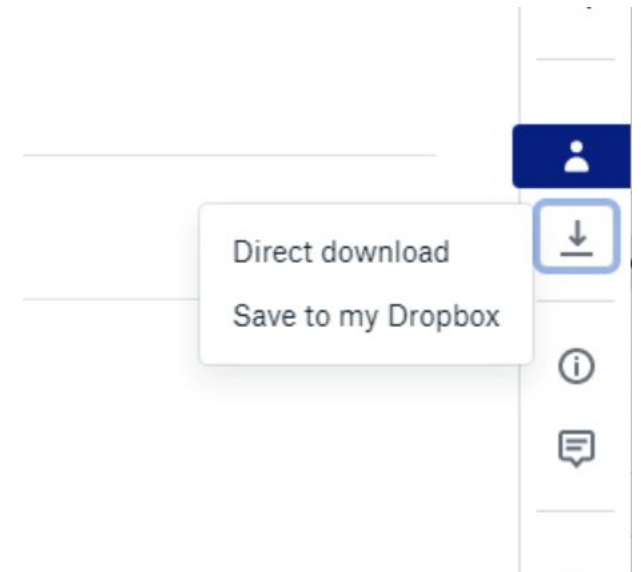
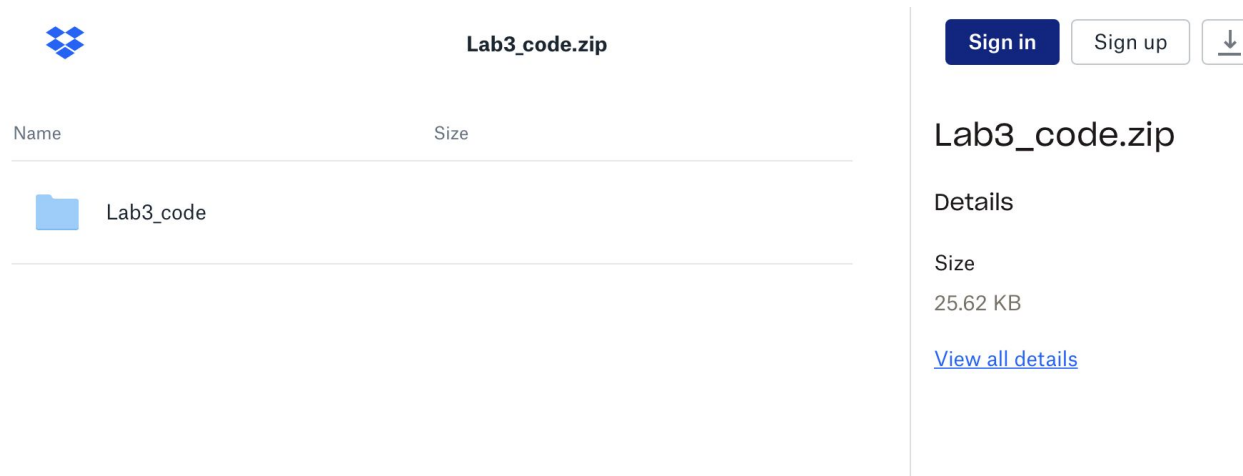
- Follow the instructions to build the project.
- Take screenshot/screenshots showing the code, and the output window. The output window must show that the project was built successfully.
- Submit to the iCollege.

Load an existing project

- In this lab, you load an existing project.
- Follow the steps :
 - **Step 1:** Download the project
 - **Step 2:** Load the project into Microsoft Visual Studio
 - **Step 3:** Build the project

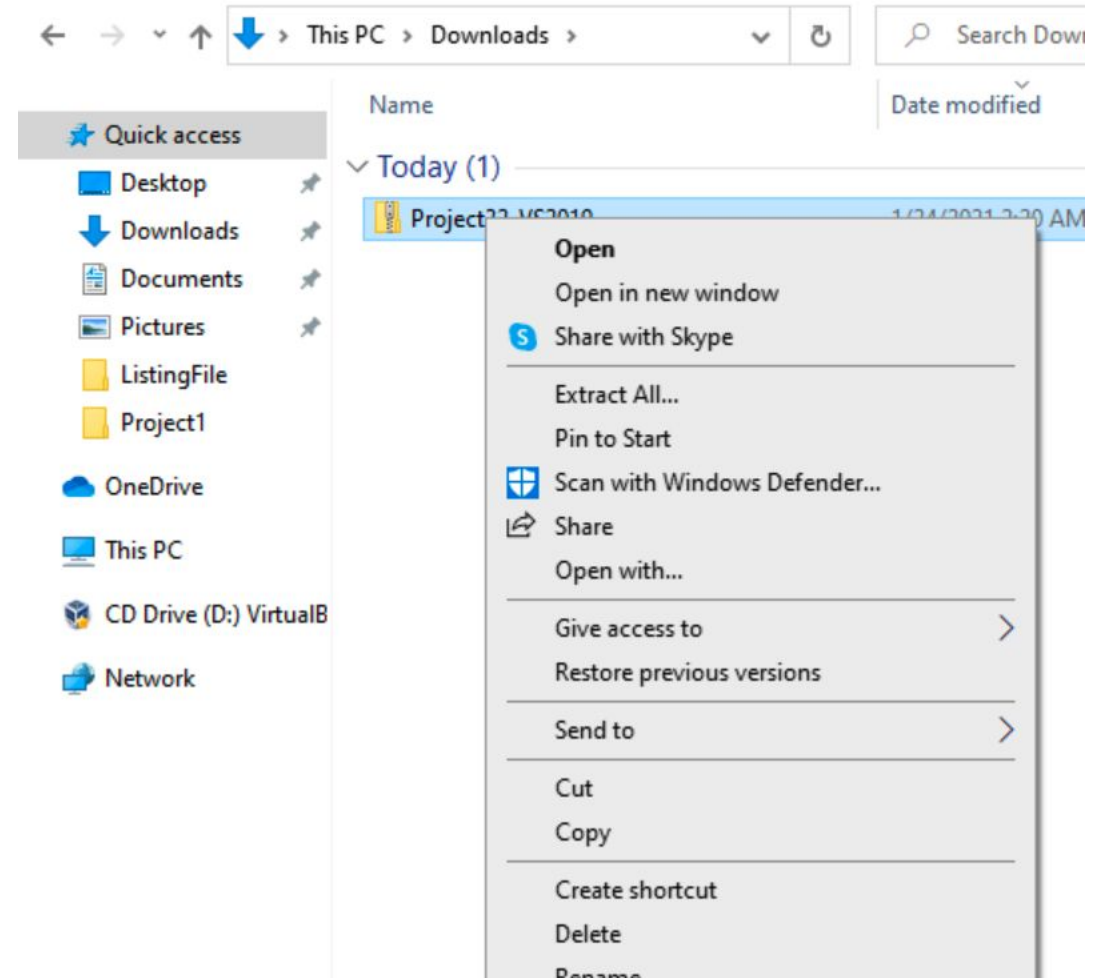
Step 1: Download the Project (1)

- Now download a project from the link:
- <https://tinyurl.com/lab3codefall21>
- Click Direct Download
- Most likely it is downloaded in your 'Downloads' folder.



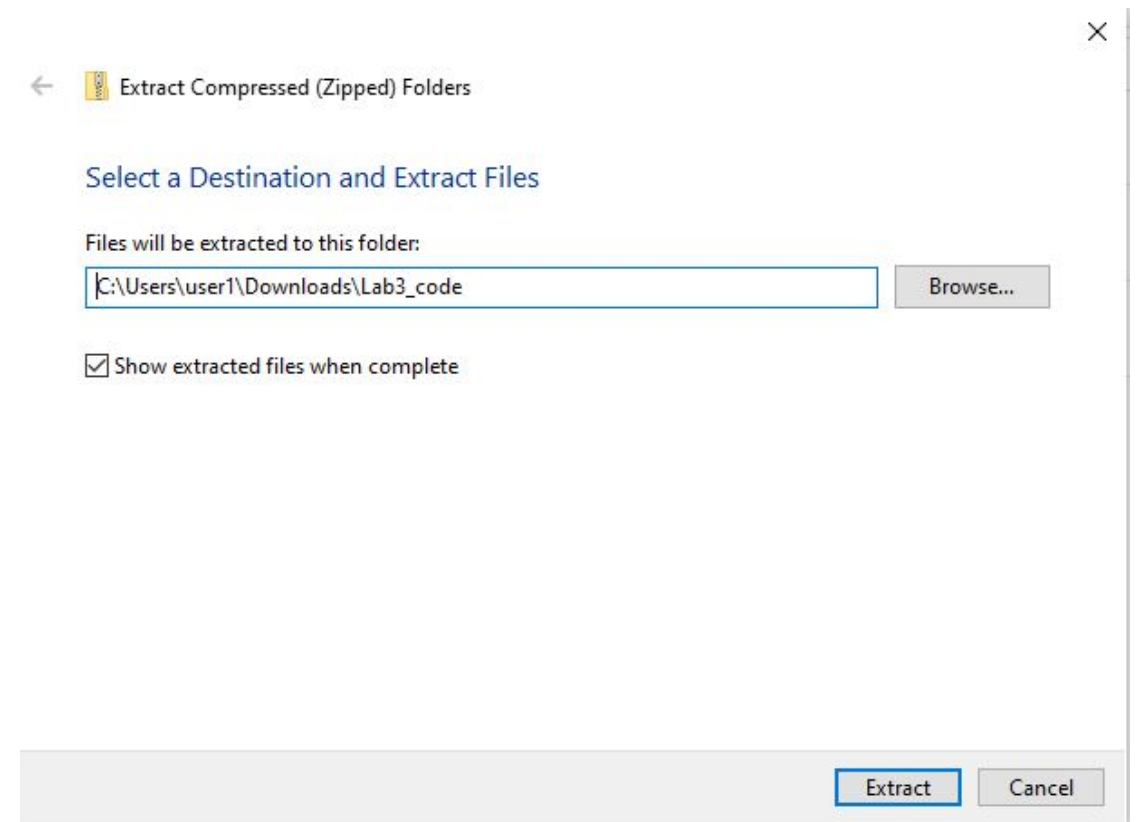
Step 1: Download the project (2)

- Go to the folder where it is downloaded
- Unzip the downloaded file
 - Right click on the zip file
 - Click Extract All...



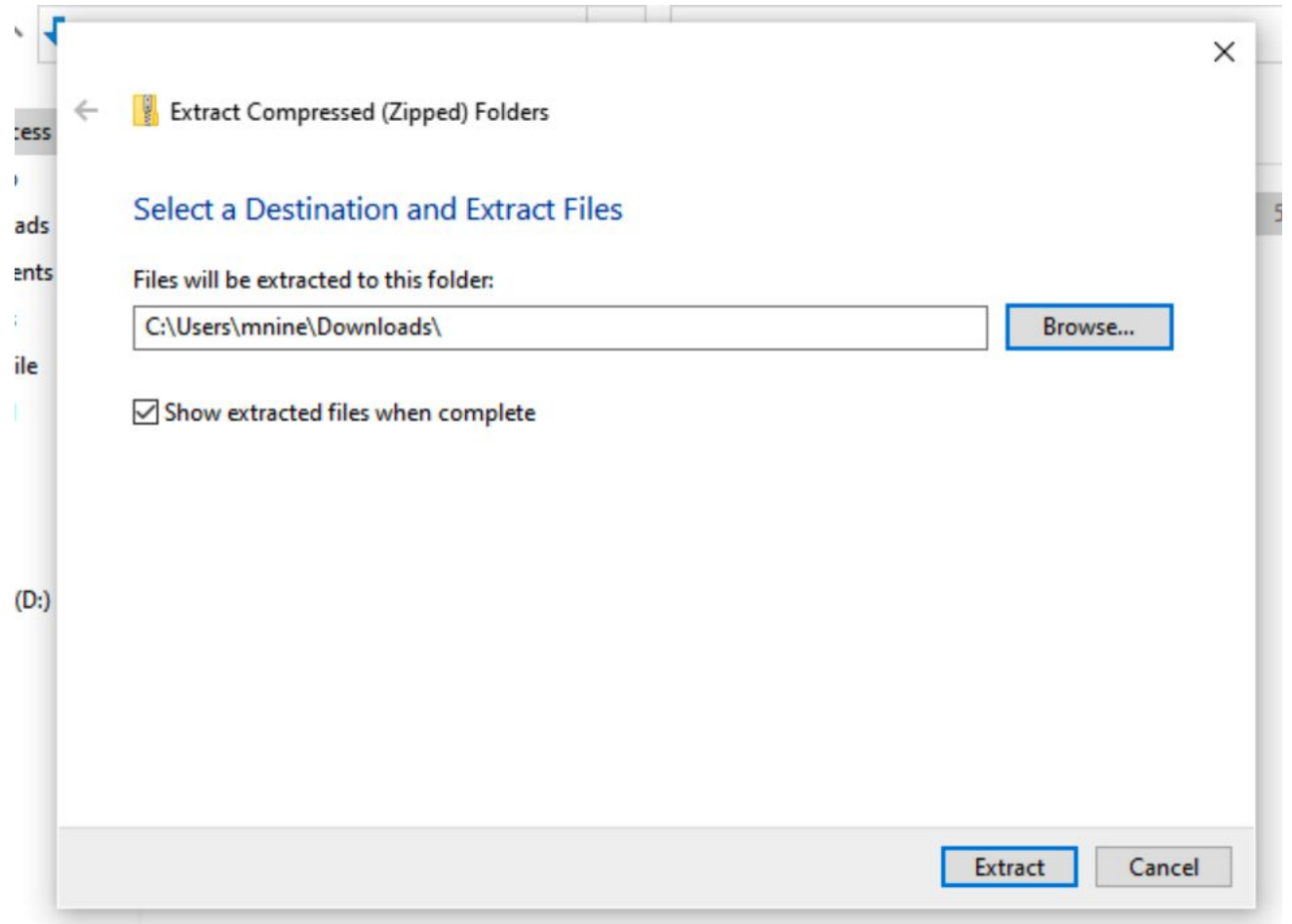
Step 1: Download the project (3)

Remove “Lab3_code” from the path.
(keep the forward slash at the end)



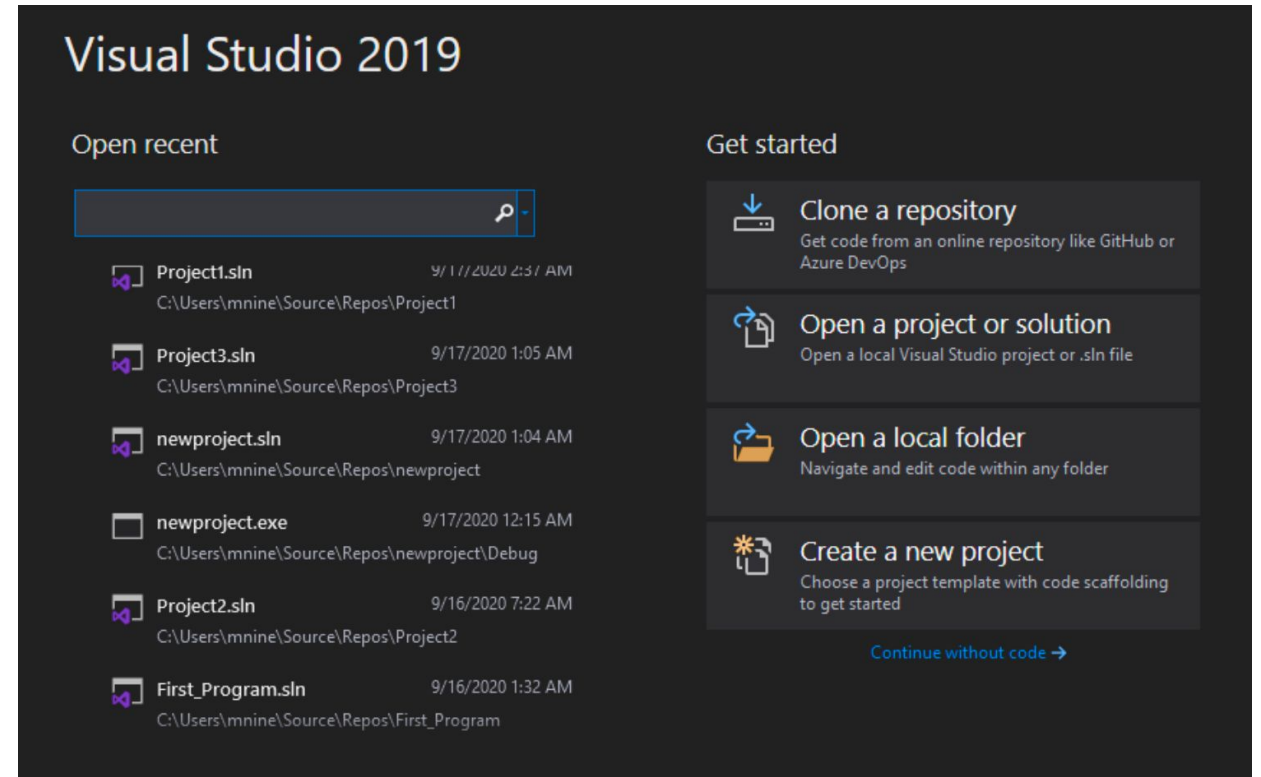
Step 1: Download the project (4)

- The path should look like
- Click Extract button
- It will extract the files in a folder named –
 Lab3_code
in your current directory.



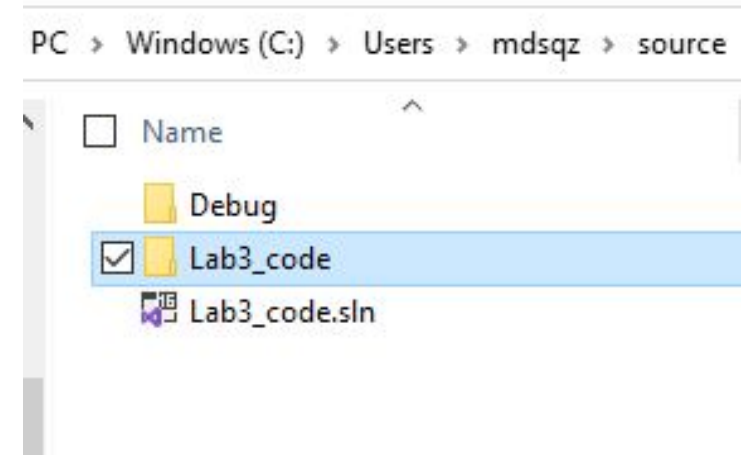
Step 2: Load the project into Visual Studio (1)

- Open Microsoft Visual Studio
- Click “Open a project or solution”



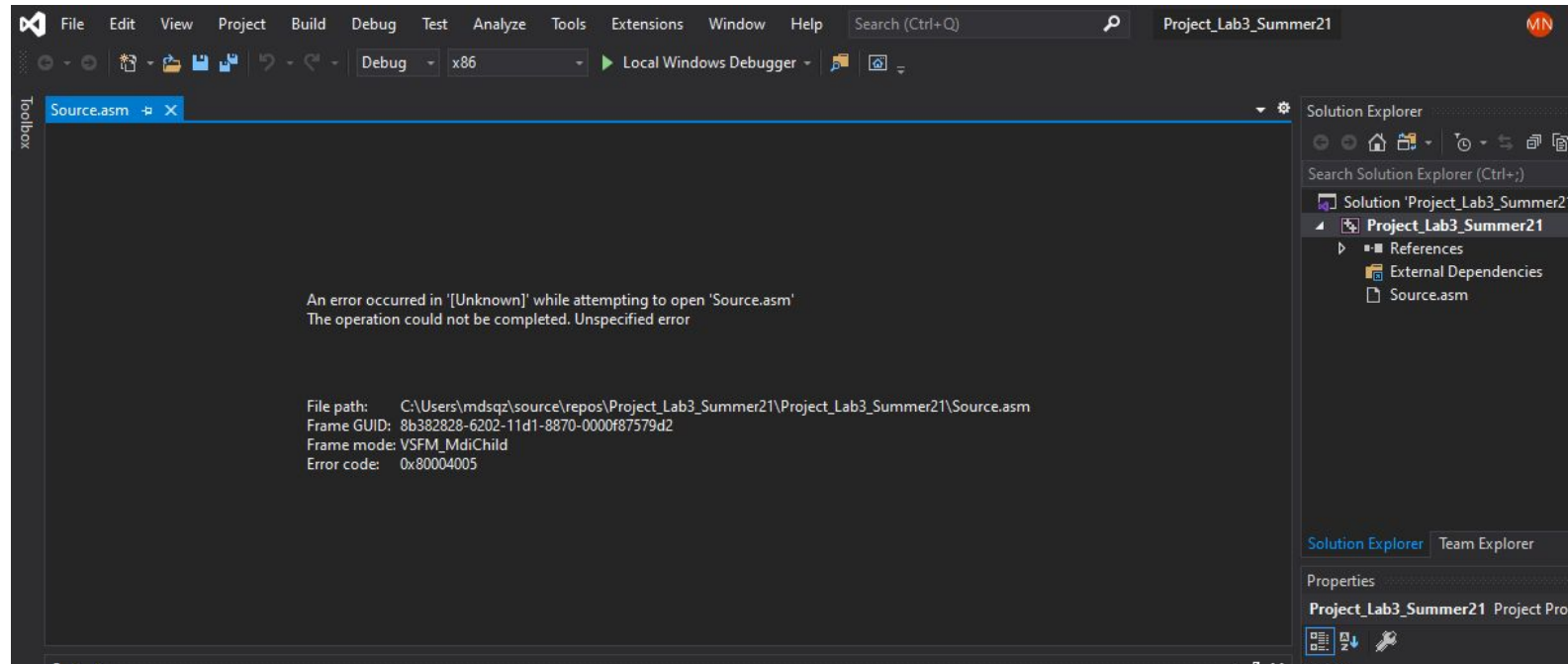
Step 2: Load the project into Visual Studio (2)

- Browse to the directory where you saved the extracted project “Lab3_code”.
- Go inside the folder “Lab3_code”.
- Select ‘Lab3_code.sln’ file.
- Click Open button at the bottom right corner.
- You might see a security warning.
- If you see one : click OK.
- The project will be loaded in you Microsoft Visual Studio.



Step 2: Load the project into Visual Studio (2)

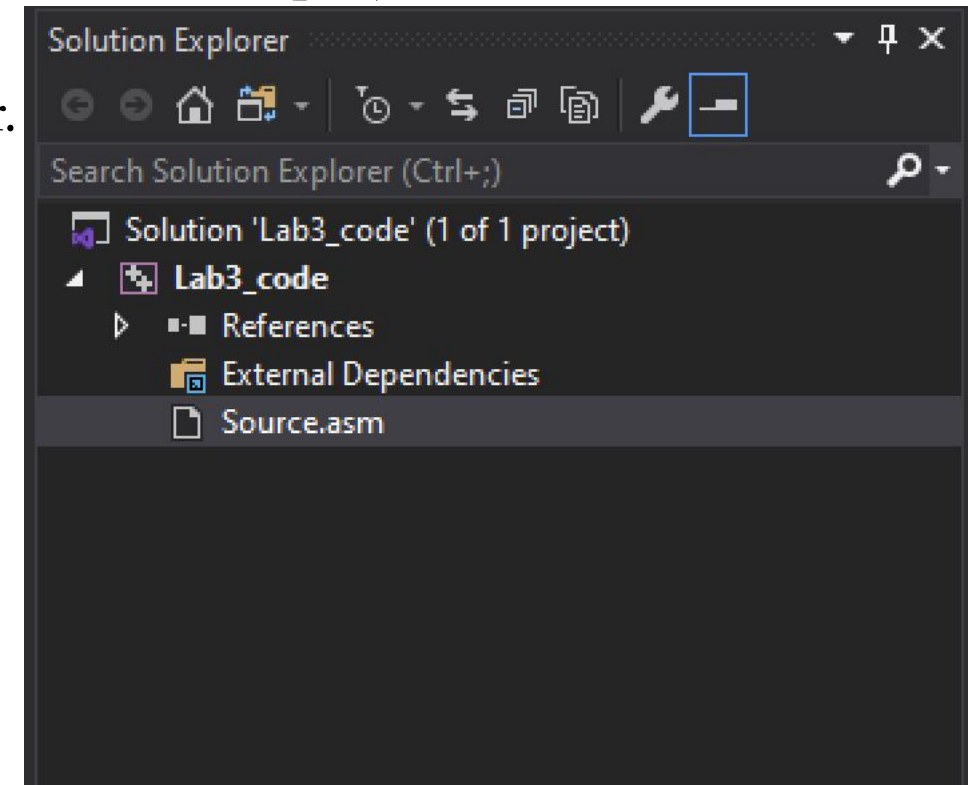
If you see the following error!



Close the Source.asm file by clicking 'X'
Then follow the next slide

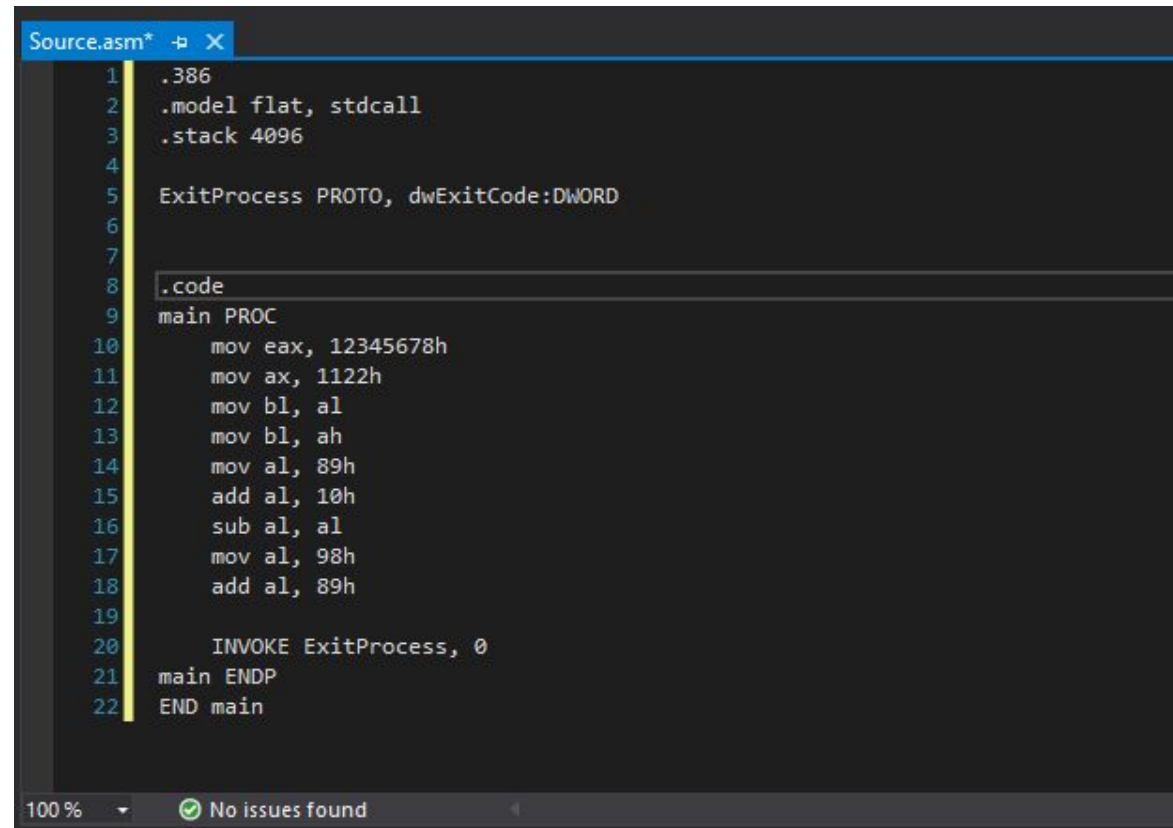
Step 2: Load the project into Visual Studio (3)

- Once the project has been opened, you will see the project name in the Solution Explorer window.
 - You should also see an **assembly language source file** in the project named **Source.asm**.
 - **Double-click** the file name to open it in the editor.



Step 2: Load the project into Visual Studio (4)

- You should see the **source code in the editor window**



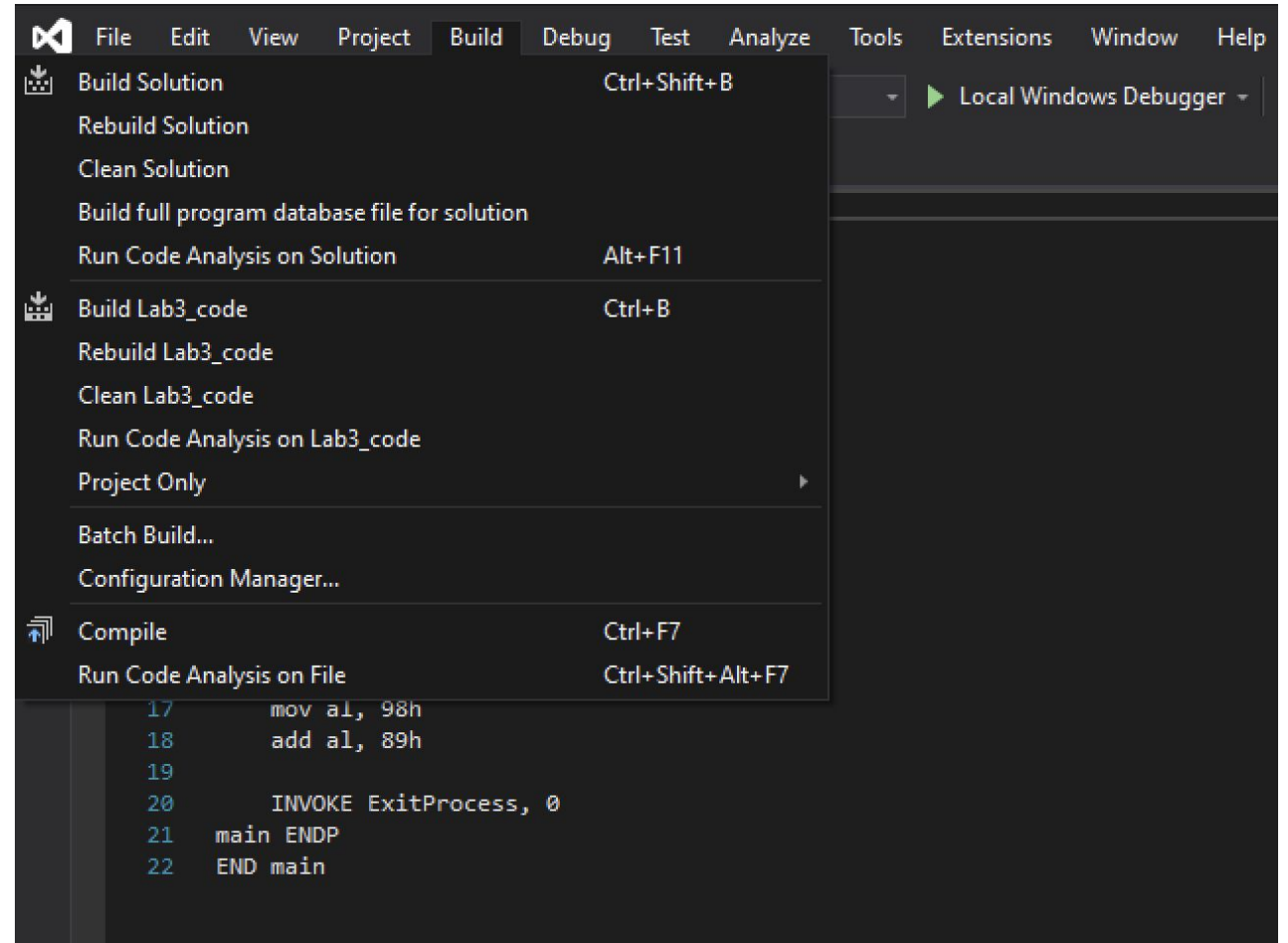
The screenshot shows a Visual Studio editor window with a dark theme. The title bar at the top reads "Source.asm*". The editor contains assembly code with line numbers 1 through 22 on the left. The code is as follows:

```
1  .386
2  .model flat, stdcall
3  .stack 4096
4
5  ExitProcess PROTO, dwExitCode:DWORD
6
7
8  .code
9  main PROC
10     mov eax, 12345678h
11     mov ax, 1122h
12     mov bl, al
13     mov bl, ah
14     mov al, 89h
15     add al, 10h
16     sub al, al
17     mov al, 98h
18     add al, 89h
19
20     INVOKE ExitProcess, 0
21 main ENDP
22 END main
```

At the bottom of the window, there is a status bar showing "100 %" and a green checkmark icon followed by the text "No issues found".

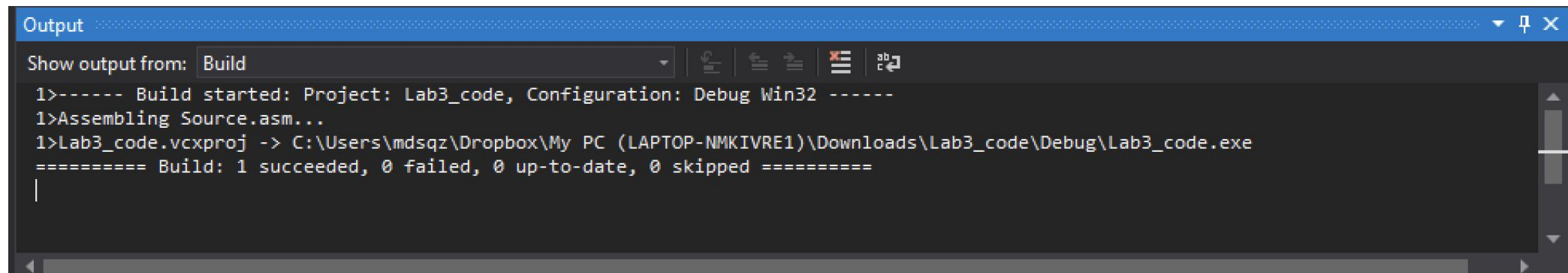
Step 3: Build the Project (1)

- 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:

A screenshot of the Visual Studio Output window. The title bar is blue and says "Output". Below the title bar is a dropdown menu set to "Build" and several icons. The main area is a dark gray text box with white text showing the build process. The text includes: "1>----- Build started: Project: Lab3_code, Configuration: Debug Win32 -----", "1>Assembling Source.asm...", "1>Lab3_code.vcxproj -> C:\Users\mdsqz\Dropbox\My PC (LAPTOP-NMKIVRE1)\Downloads\Lab3_code\Debug\Lab3_code.exe", and "==== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====". There is a vertical scrollbar on the right side of the text box.

```
Output
Show output from: Build
1>----- Build started: Project: Lab3_code, Configuration: Debug Win32 -----
1>Assembling Source.asm...
1>Lab3_code.vcxproj -> C:\Users\mdsqz\Dropbox\My PC (LAPTOP-NMKIVRE1)\Downloads\Lab3_code\Debug\Lab3_code.exe
==== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

- You should see the message in the last line:
====Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

Lab 3(b)

Debug the project code

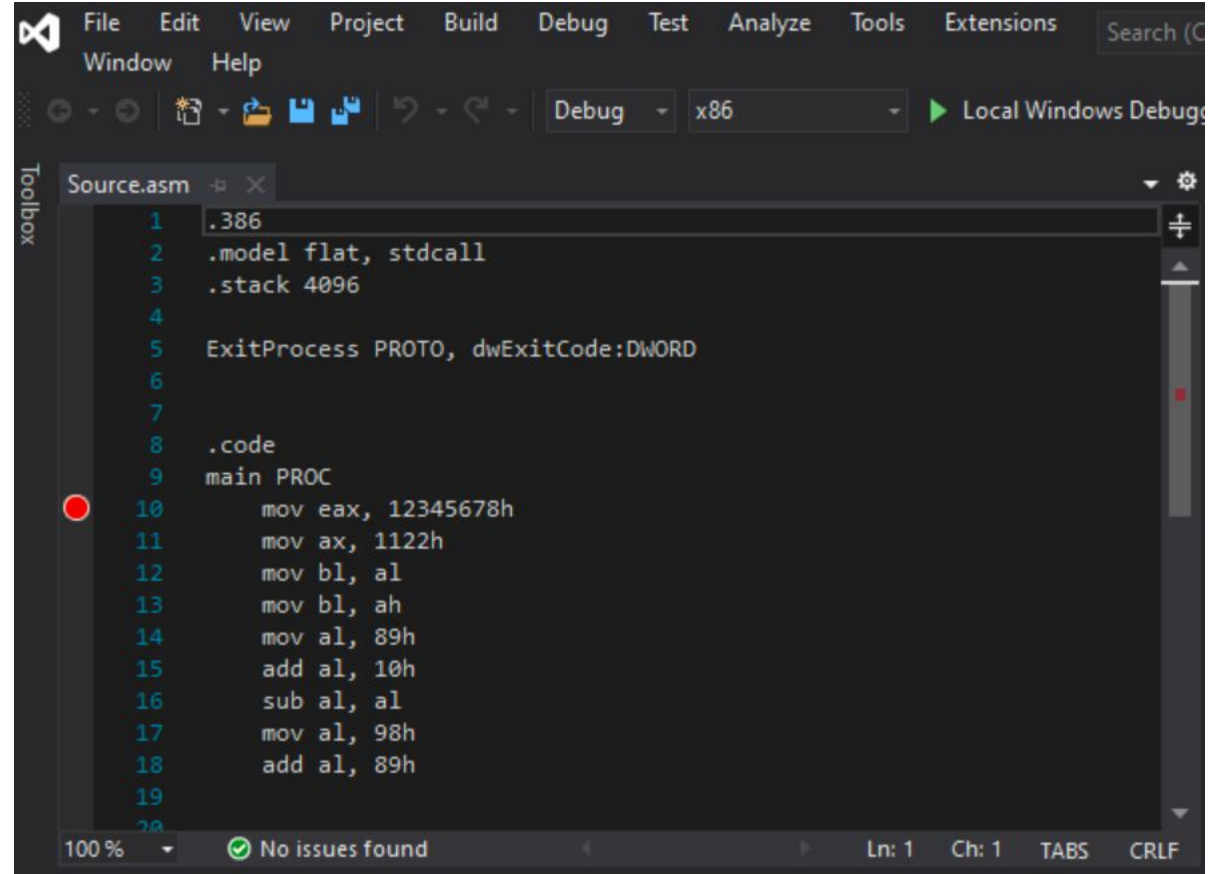
And record the register and flag contents

Lab 3(b) Submission Instructions

- An **answer sheet** is provided with this lab for Lab 3(b).
- You need to debug the code
- Stop for each instruction and record the register content in the answer sheet. Then explain the register content.
- Submit your answer sheet to the iCollege.

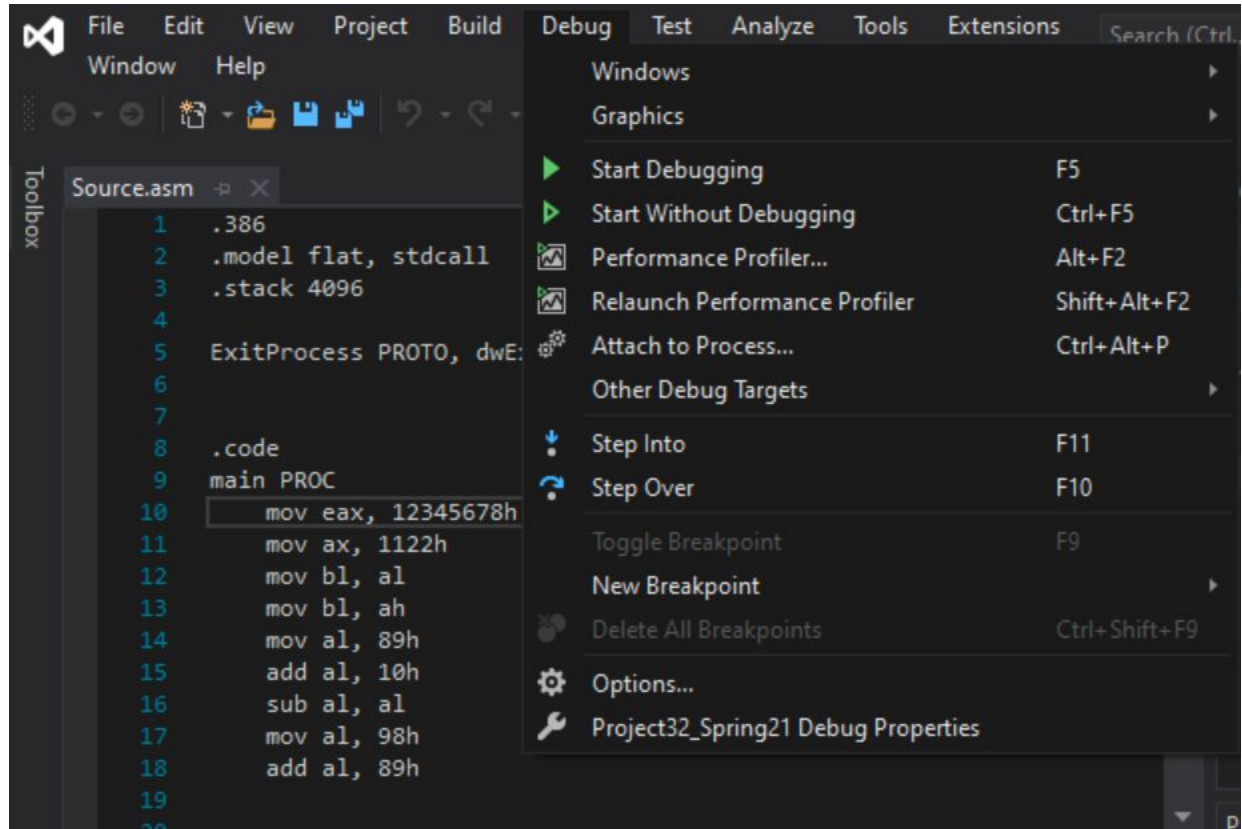
Debug the Project

- 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**.



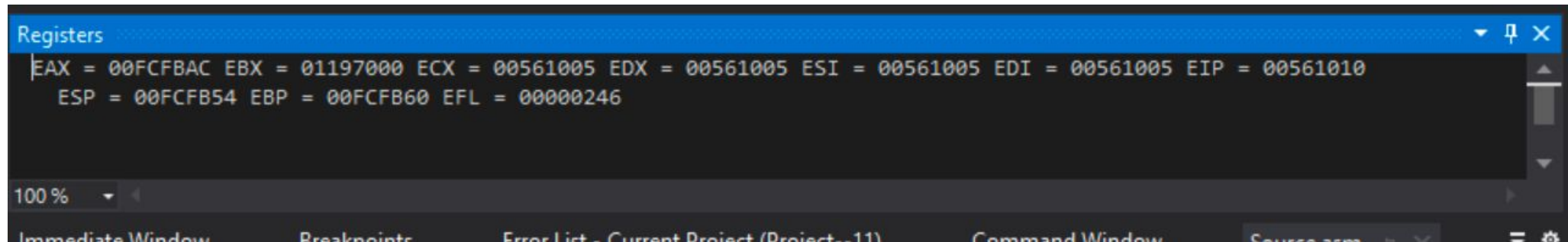
Debug your code

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



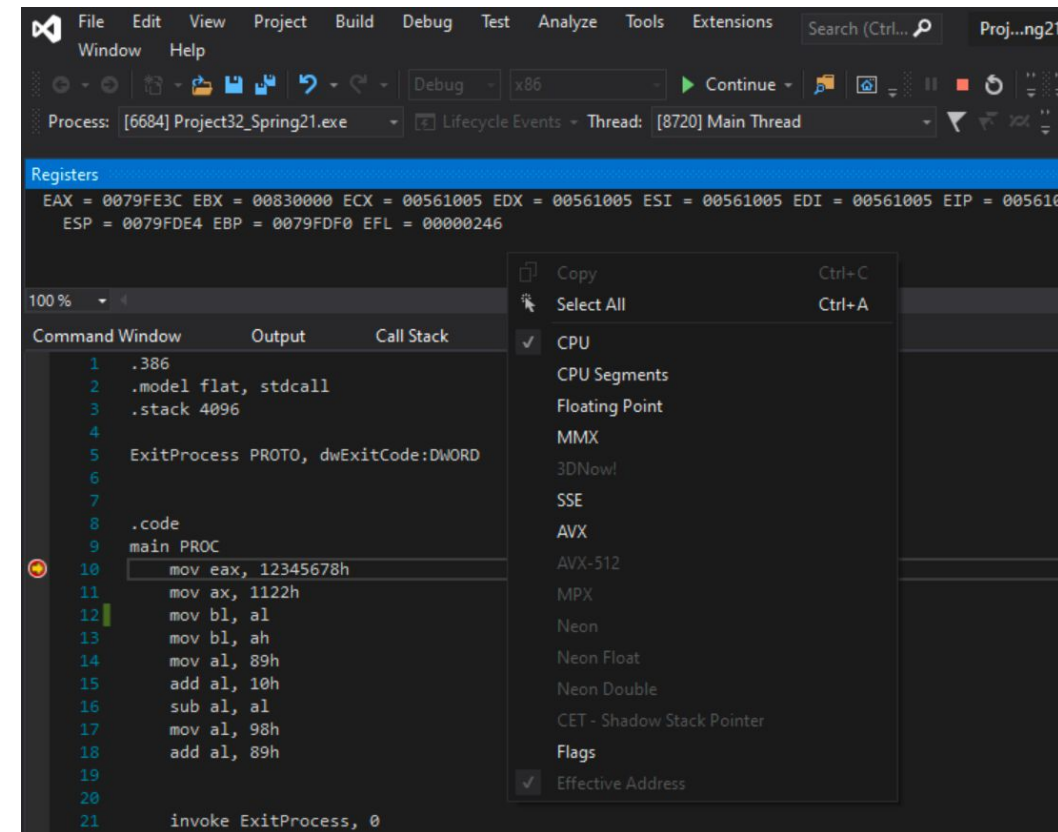
Debug the Project

- You should be able to see the register window :



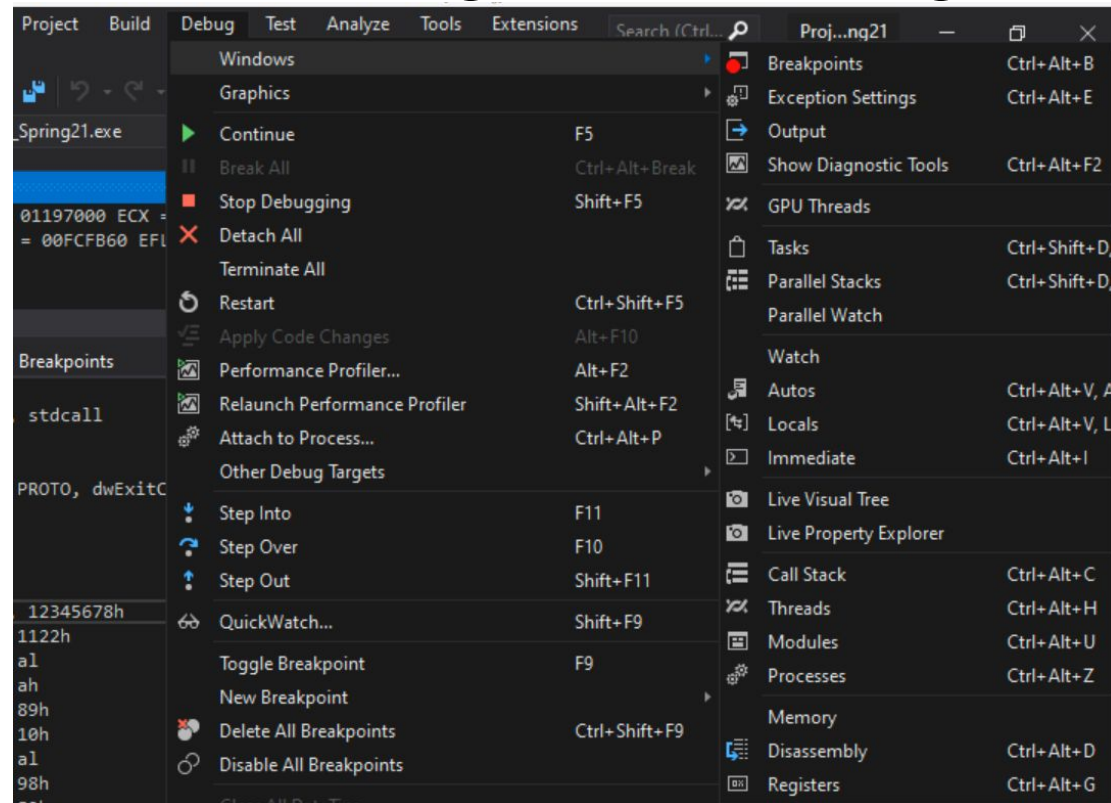
Debug the project

- Turn on the EFLAGS in the register window.
- Right click on the register window
- Then check the flag



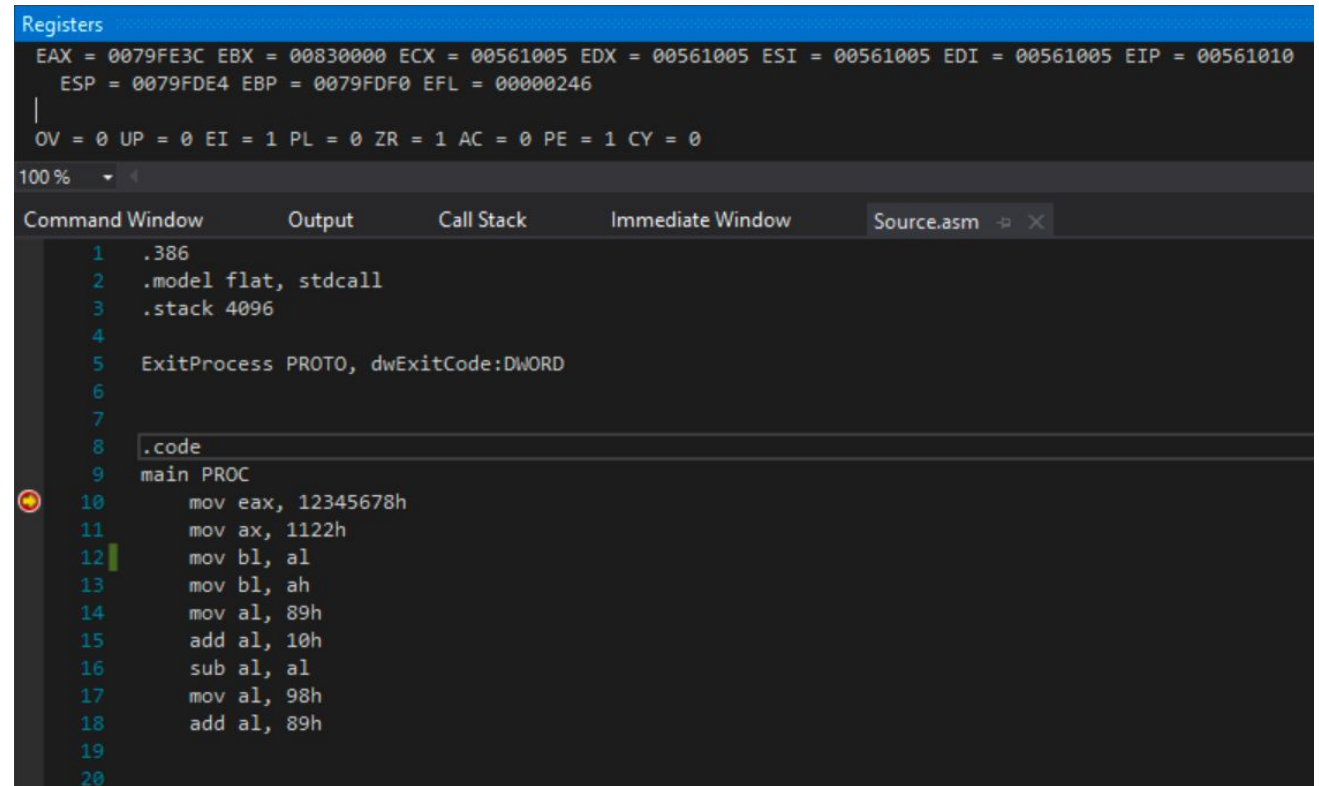
Debug the Project

- You must be inside debug mode:
 - If you are not in debug mode : Go to Debug -> Start debugging
- If you don't see the register window: Go to : Debug -> Windows->registers.



Debug your code

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



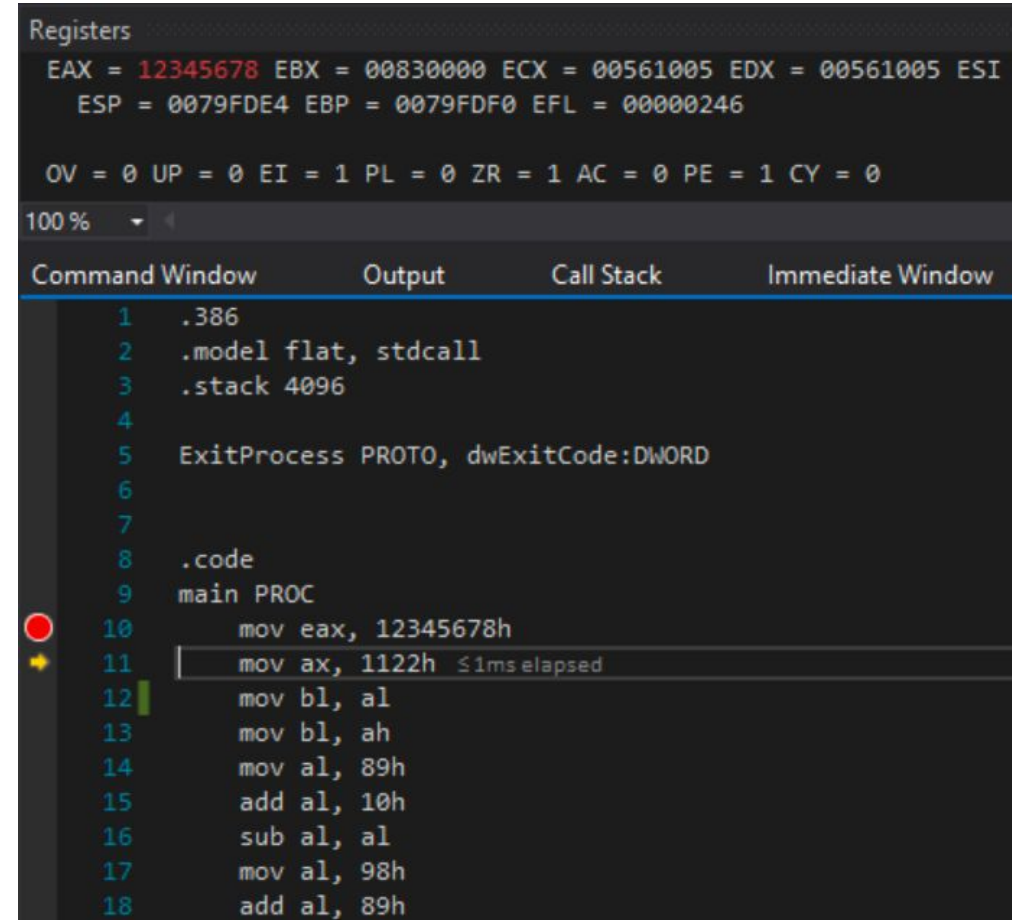
The screenshot shows a debugger interface with a dark theme. At the top, the 'Registers' window displays various CPU registers and their values: EAX = 0079FE3C, EBX = 00830000, ECX = 00561005, EDX = 00561005, ESI = 00561005, EDI = 00561005, EIP = 00561010, ESP = 0079FDE4, EBP = 0079FDF0, EFL = 00000246. Below this, a status bar shows flags: OV = 0, UP = 0, EI = 1, PL = 0, ZR = 1, AC = 0, PE = 1, CY = 0. The main window is titled 'Source.asm' and shows assembly code. A red dot with a yellow pointer is positioned next to line 10, indicating a breakpoint. The code includes directives like .386, .model flat, stdcall, .stack 4096, and a procedure named main PROC. The instructions in the main PROC are: mov eax, 12345678h, mov ax, 1122h, mov bl, al, mov bl, ah, mov al, 89h, add al, 10h, sub al, al, mov al, 98h, and add al, 89h.

```
Registers
EAX = 0079FE3C EBX = 00830000 ECX = 00561005 EDX = 00561005 ESI = 00561005 EDI = 00561005 EIP = 00561010
ESP = 0079FDE4 EBP = 0079FDF0 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

100 %
Command Window Output Call Stack Immediate Window Source.asm
1 .386
2 .model flat, stdcall
3 .stack 4096
4
5 ExitProcess PROTO, dwExitCode:DWORD
6
7
8 .code
9 main PROC
10 mov eax, 12345678h
11 mov ax, 1122h
12 mov bl, al
13 mov bl, ah
14 mov al, 89h
15 add al, 10h
16 sub al, al
17 mov al, 98h
18 add al, 89h
19
20
```

Debug the Project

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



The screenshot shows a debugger interface with a dark theme. At the top, the 'Registers' window displays the following values: EAX = 12345678, EBX = 00830000, ECX = 00561005, EDX = 00561005, ESI = 0079FDE4, ESP = 0079FDF0, and EFL = 00000246. Below the registers, status flags are shown: OV = 0, UP = 0, EI = 1, PL = 0, ZR = 1, AC = 0, PE = 1, and CY = 0. The main window is divided into four panes: 'Command Window', 'Output', 'Call Stack', and 'Immediate Window'. The 'Command Window' pane is active and shows assembly code with line numbers 1 through 18. Line 10, 'mov eax, 12345678', is highlighted with a yellow background. A red circle and a yellow arrow point to line 10. Line 11, 'mov ax, 1122h', is also highlighted. The code continues with 'mov bl, al', 'mov bl, ah', 'mov al, 89h', 'add al, 10h', 'sub al, al', 'mov al, 98h', and 'add al, 89h'.

```
Registers
EAX = 12345678 EBX = 00830000 ECX = 00561005 EDX = 00561005 ESI
ESP = 0079FDE4 EBP = 0079FDF0 EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

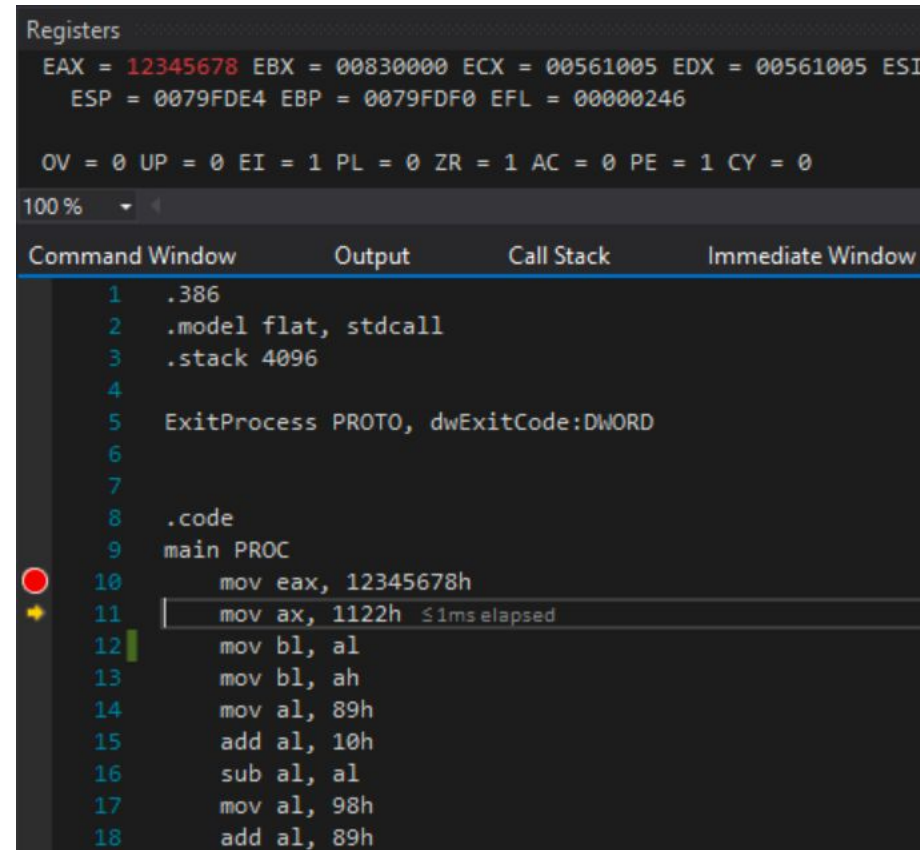
100 %
Command Window Output Call Stack Immediate Window
1 .386
2 .model flat, stdcall
3 .stack 4096
4
5 ExitProcess PROTO, dwExitCode:DWORD
6
7
8 .code
9 main PROC
10 mov eax, 12345678
11 mov ax, 1122h ≤1ms elapsed
12 mov bl, al
13 mov bl, ah
14 mov al, 89h
15 add al, 10h
16 sub al, al
17 mov al, 98h
18 add al, 89h
```

mov Instruction

- mov instruction has two inputs – first one is destination and second one is source.
 - mov **destination, source**
 - mov instruction copy the source content to the destination.
 - It is more **like assignment operation** in high level language.
- In the Code, Line 10
 - Mov eax, 12345678h
 - Here 12345678 is a hexadecimal number. Radix ‘h’ after the number indicates that it is a hex.
 - This can be converted to 32-bit binary number. (**Try convert it to binary**)
 - Mov instruction stores that value to **EAX register that is also 32-bit register.**
- After executing line 10, you can see the content change in **EAX register.**

mov Instruction

Eax contains 12345678
(all the register values
are in hexadecimal)



The screenshot shows a debugger window with the following content:

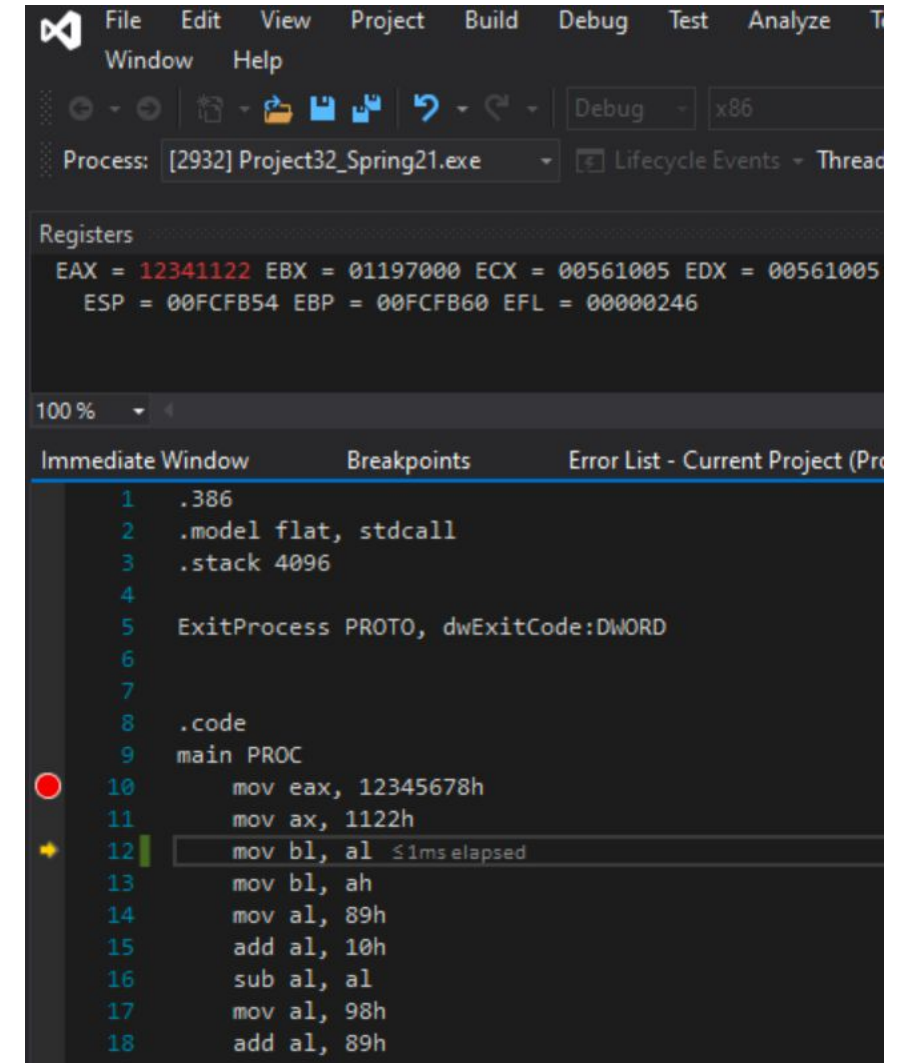
Registers
EAX = 12345678 EBX = 00830000 ECX = 00561005 EDX = 00561005 ESI
ESP = 0079FDE4 EBP = 0079FDF0 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

100 %

Command Window	Output	Call Stack	Immediate Window
1	.386		
2	.model flat, stdcall		
3	.stack 4096		
4			
5	ExitProcess PROTO, dwExitCode:DWORD		
6			
7			
8	.code		
9	main PROC		
10	mov eax, 12345678h		
11	mov ax, 1122h ≤1ms elapsed		
12	mov bl, al		
13	mov bl, ah		
14	mov al, 89h		
15	add al, 10h		
16	sub al, al		
17	mov al, 98h		
18	add al, 89h		

Debug : Line 11

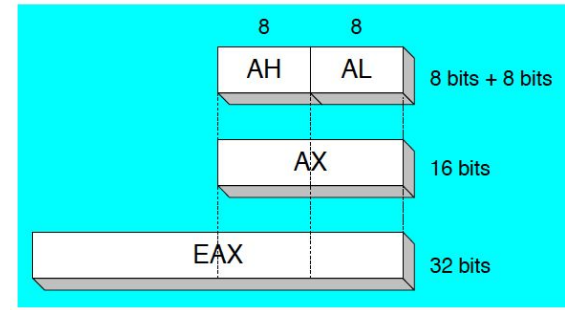
- To execute line 11, Select 'Step over' from debug menu
 - You can also use shortcut : Fn+F10
- Mov ax, 1122h
 - Here 1122 is a hex and it is 16 bit.
 - AX is also a 16-bit register.
- After executing the mov instruction,
 - Only AX part (lower) of EAX register updated.
 - The upper part is unchanged.



```
File Edit View Project Build Debug Test Analyze
Window Help
Process: [2932] Project32_Spring21.exe Lifecycle Events Thread
Registers
EAX = 12341122 EBX = 01197000 ECX = 00561005 EDX = 00561005
ESP = 00FCFB54 EBP = 00FCFB60 EFL = 00000246
100 %
Immediate Window Breakpoints Error List - Current Project (Pro
1 .386
2 .model flat, stdcall
3 .stack 4096
4
5 ExitProcess PROTO, dwExitCode:DWORD
6
7
8 .code
9 main PROC
10 mov eax, 12345678h
11 mov ax, 1122h
12 mov bl, al ≤1ms elapsed
13 mov bl, ah
14 mov al, 89h
15 add al, 10h
16 sub al, al
17 mov al, 98h
18 add al, 89h
```


Debug: Line 12

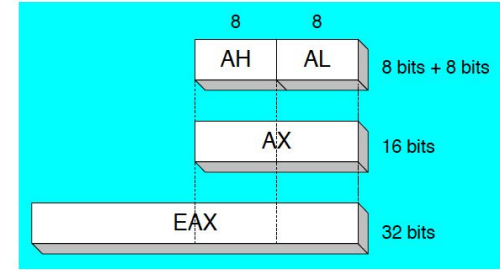
- To execute line 12, Select 'Step over' from debug menu
 - You can also use shortcut : Fn+F10
- Mov bl, al
 - Here the content of AL register is moved to the BL register.
 - AL register is the lower 8-bit of the EAX register.
 - It contains 22 now.
 - 22 is a hex
 - BL is the lower 8 bit of EBX register
 - BL is the destination here
- After executing this line, BL contains 22 as well. However, rest of the EBX register remains unchanged with previous garbage values.



The screenshot shows a debugger window with the following components:

- Registers:** EAX = 12341122, EBX = 00830022, ECX = 00561005, EDX = 00561005, ESP = 0079FDE4, EBP = 0079FDF0, EFL = 00000246. Below this, status flags are shown: OV = 0, UP = 0, EI = 1, PL = 0, ZR = 1, AC = 0, PE = 1, CY = 0.
- Command Window:** Shows assembly code with line numbers 1 through 18. Line 12 is highlighted with a green bar and a red dot in the left margin, indicating it is the current instruction being executed or about to be executed. Line 13 is highlighted with a yellow bar and a yellow arrow in the left margin, indicating it is the next instruction to be executed.
- Output:** Shows the output of the execution, including the instruction 'mov bl, al' and the elapsed time '≤ 1ms elapsed'.
- Call Stack:** Shows the current call stack, including 'ExitProcess' and 'main PROC'.
- Immediate Window:** Shows the immediate values of the registers, including 'EAX = 12341122' and 'EBX = 00830022'.

Debug: Line 13



- To execute line 13, Select 'Step over' from debug menu
 - You can also use shortcut : Fn+F10
- Mov bl, ah
 - Here the content of AH register is moved to the BL register.
 - AH register is the upper 8-bit of the **AX register**.
 - It contains 11 now.
 - 11 is a hex
 - BL is the lower 8 bit of EBX register
 - BL is the destination here
- After executing this line, BL contains 11 as well. However, rest of the EBX register remains unchanged with previous garbage values.

```
Registers
EAX = 12341122 EBX = 00830011 ECX = 00561005 EDX = 00561005
ESP = 0079FDE4 EBP = 0079FDF0 EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
100 %
Command Window Output Call Stack Immediate Window
1 .386
2 .model flat, stdcall
3 .stack 4096
4
5 ExitProcess PROTO, dwExitCode:DWORD
6
7
8 .code
9 main PROC
10 mov eax, 12345678h
11 mov ax, 1122h
12 mov bl, al
13 mov bl, ah
14 mov al, 89h ≤1ms elapsed
15 add al, 10h
16 sub al, al
17 mov al, 98h
18 add al, 89h
19
```


Add and Sub instruction

- **add** instruction has two inputs – first one is destination and second one is source.
 - add destination, source
 - add instruction adds the source content to the destination and store the result in destination.
- For example,
 - add al, 12h
 - Adds 12 to the content of AL register and store the result in AL register
 - It is more like : $al = al + 12$ in a high level language
- **sub** instruction is similar to add instruction
 - Instead of addition it performs subtraction.

Submission : Lab 3(b)

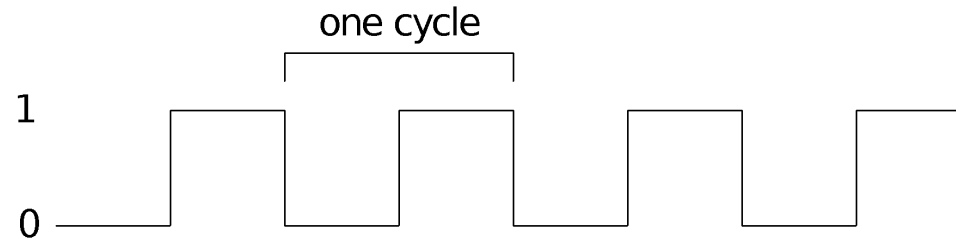
- Debug the rest of the code one line at a time.
- There is an **answer sheet** provided with the lab.
- After executing each line record the register value in the answer sheet.
- Then also provide short explanation for the changes.
- Submit the filled out answer sheet at iCollege.

Processor Clock

A review

Clock

- Each processor has a built in clock to synchronize the internal operations.
- Each CPU operation synchronized by an internal clock pulsing at constant rate.



- A **clock** is a sequence of **1's** and **0's**
- **Clock Cycle:** a 0 and 1 produce a clock cycle
- **Frequency:** The number of cycles happens per second
- Unit : Hz = 1 cycle per second

Clock

- Clock period = time length of a clock cycle

$$\text{Clock period} = \frac{1}{\text{Clock frequency}}$$

- A CPU has a clock frequency 1 GHz. What is the clock period?
 - $\text{Clock period} = \frac{1}{10^9} = 1 \text{ nanosecond}$
- Let's say an instruction, takes 40 clock cycle to execute in your 1 GHz processor. What is the actual time it takes to execute the instruction?
 - Clock period = 1 ns, so each cycle takes 1 ns to finish
 - 40 clock cycle takes = $40 * 1 \text{ ns} = 40 \text{ ns}$ to finish
- So the instruction takes 40 ns to execute.

An example

- Suppose a program contains 1 billion instructions to execute on a processor running on 2 GHz. The instructions takes 3 clock cycles to execute. What is the execution time of the program?
- Answer:
 - 1 billion instructions each takes 3 clock cycles
 - Total clock cycle for the program = 1 billion * 3 cycles = $3 * 10^9$ cycles.
 - Given, Processor Frequency = 2 GHz
 - Processor produces $2 * 10^9$ cycles in 1 second.
 - $2 * 10^9$ cycles takes 1 second
 - $3 * 10^9$ cycles takes $\frac{3 * 10^9}{2 * 10^9}$ second = 1.5 seconds.

Clock per Instruction (CPI)

- Is an effective average.
- It is the average number of clocks required by the instructions in a program.
- In a program 30% instructions takes 4 clock cycles and the rest of the instructions takes 1 clock cycles.
- $\text{CPI} = 0.3 * 4 + 0.7 * 1 = 1.9$ clocks per instruction.

Million Instructions Per Second

- **Step 1:** Perform Divide operation between no. of instructions and Execution time.
- **Step 2:** Perform Divide operation between that variable and 1 million for finding millions of instructions per second.
- For example,
 - if a computer completed 2 million instructions in 0.10 seconds
 - $2 \text{ million} / 0.10 = 20 \text{ million}$.
 - No of MIPS = $20 \text{ million} / 1 \text{ million}$
 - = 20

Lab 3(c)

Some math Problems

Submit the problems

Lab 3(c) : Submission

- Solve the Problems provided in slide 45 and 46.
- You can do your work in a text editor (Microsoft word, open office, etc.)
- Or you can do it in a piece of paper, then scan or take a picture of the paper.
- Convert them into pdf and submit in the iCollege.

Problem 1

- Suppose a program contains 500 million instructions to execute on a processor running on 2.2 GHz. Half of the instructions takes 3 clock cycles to execute, where rest of the instructions take 10 clock cycle. What is the execution time of the program?

Problem 2

- A processor is 20 MIPS. If you run a program on that processor and the program takes 30 seconds to finish. How many instructions are there in this program?