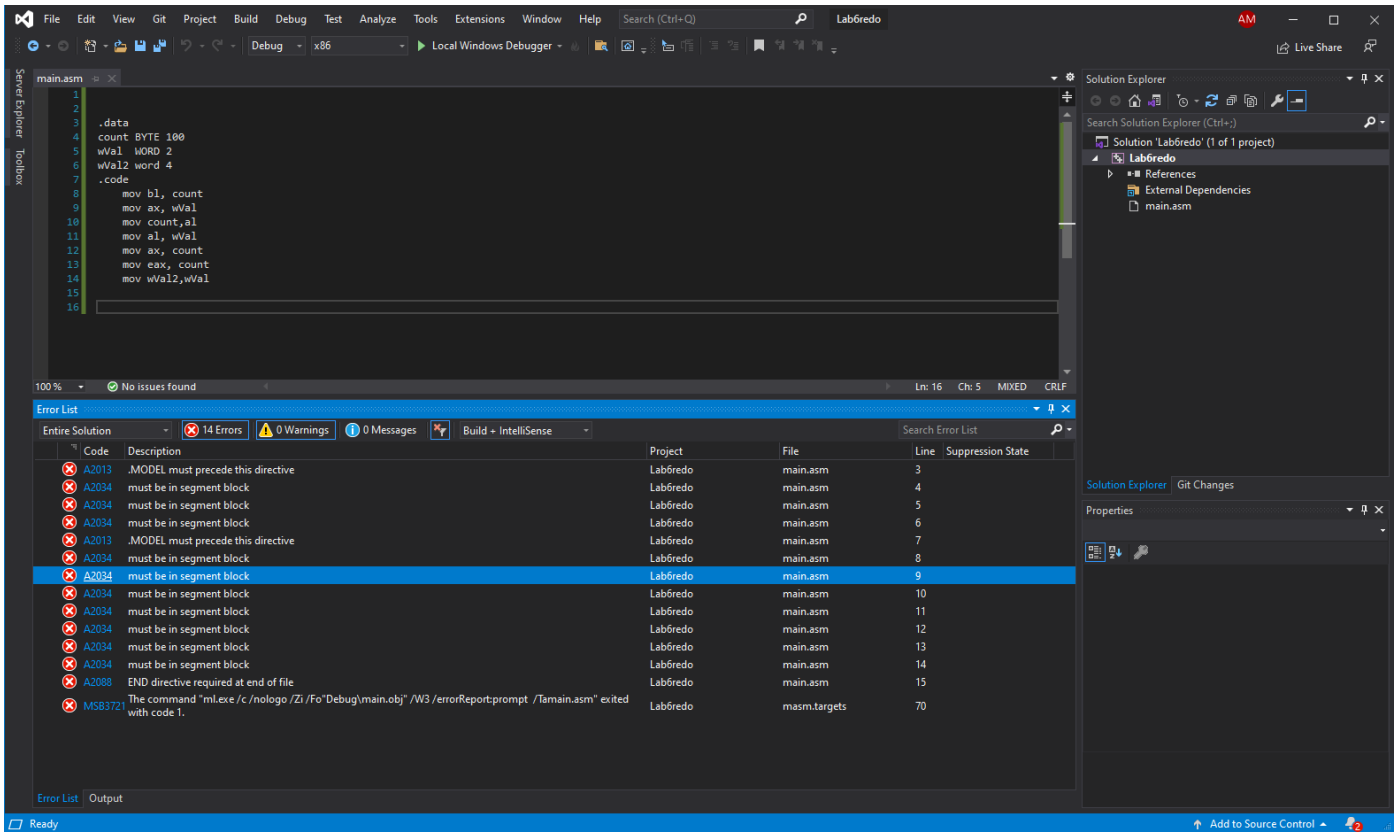
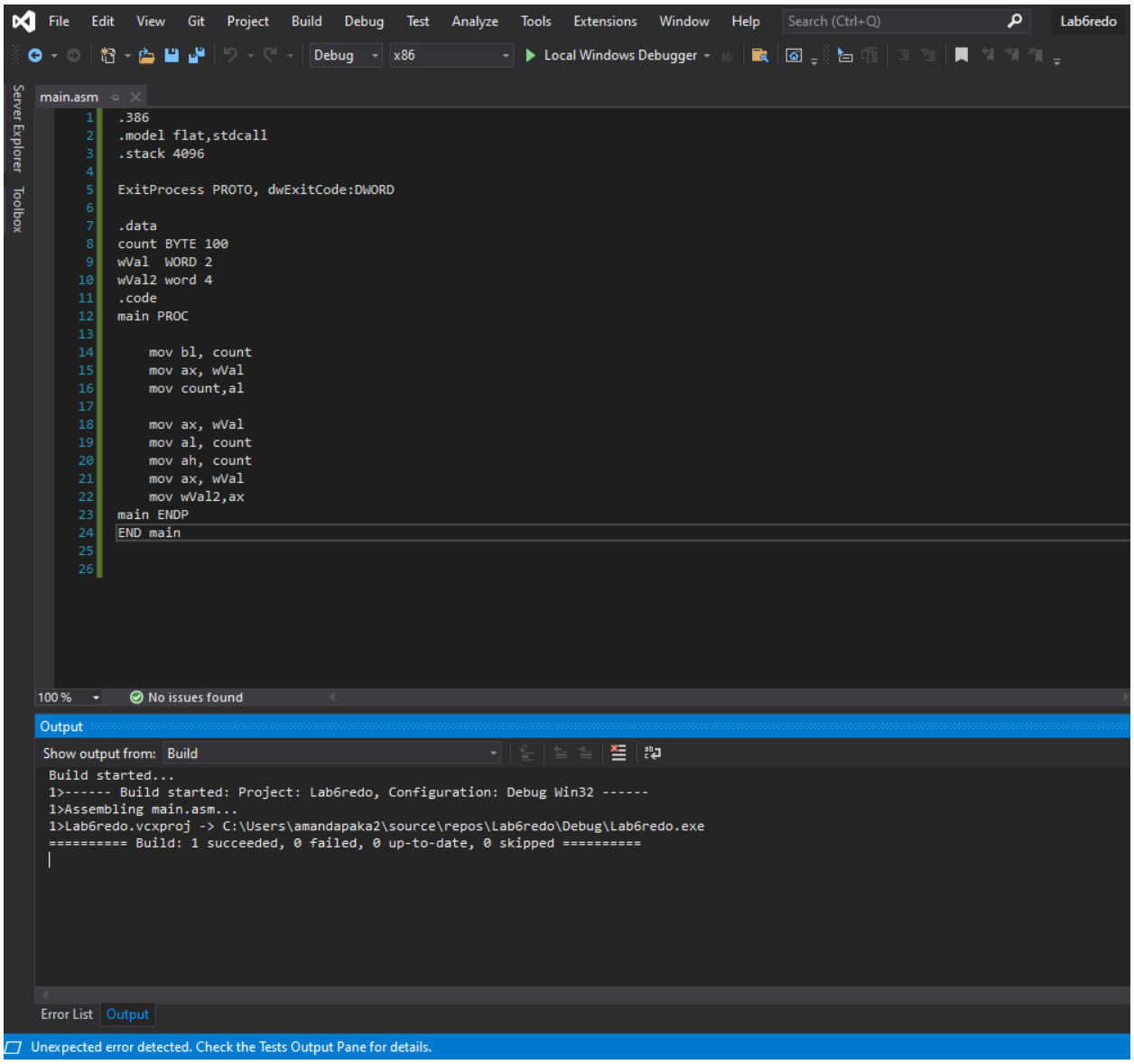
Lab Part A with errors



Fixed errors


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab6redo
Debug
x86
Local Windows Debugger
Server Explorer
Toolbox
main.asm
.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.data
count BYTE 100
wVal WORD 2
wVal2 word 4
.code
main PROC
mov bl, count
mov ax, wVal
mov count,al
mov ax, wVal
mov al, count
mov ah, count
mov ax, wVal
mov wVal2,ax
main ENDP
END main
100 %
No issues found
Output
Show output from: Build
Build started...
1>------ Build started: Project: Lab6redo, Configuration: Debug Win32 ------
1>Assembling main.asm...
1>Lab6redo.vcxproj -> C:\Users\amandapaka2\source\repos\Lab6redo\Debug\Lab6redo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Error List
Output
Unexpected error detected. Check the Tests Output Pane for details.

Lab 6(b)

Debug through each line of instructions.

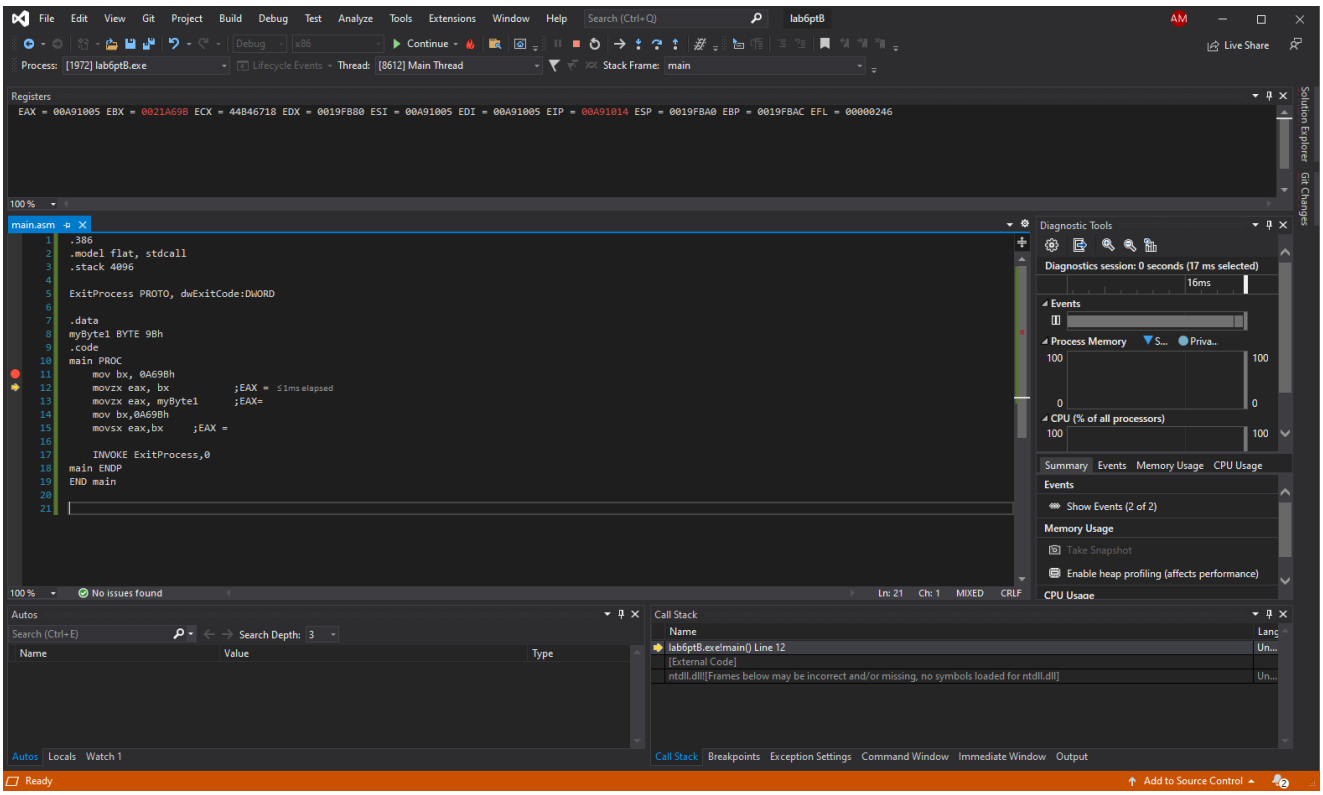
Take a screenshot that includes code and a register window.

Record the register content and explain the register contents.

Line number: 11

Instruction: mov bx, 0A69Bh

Register values: EBX: 0021A69B



Screenshot:

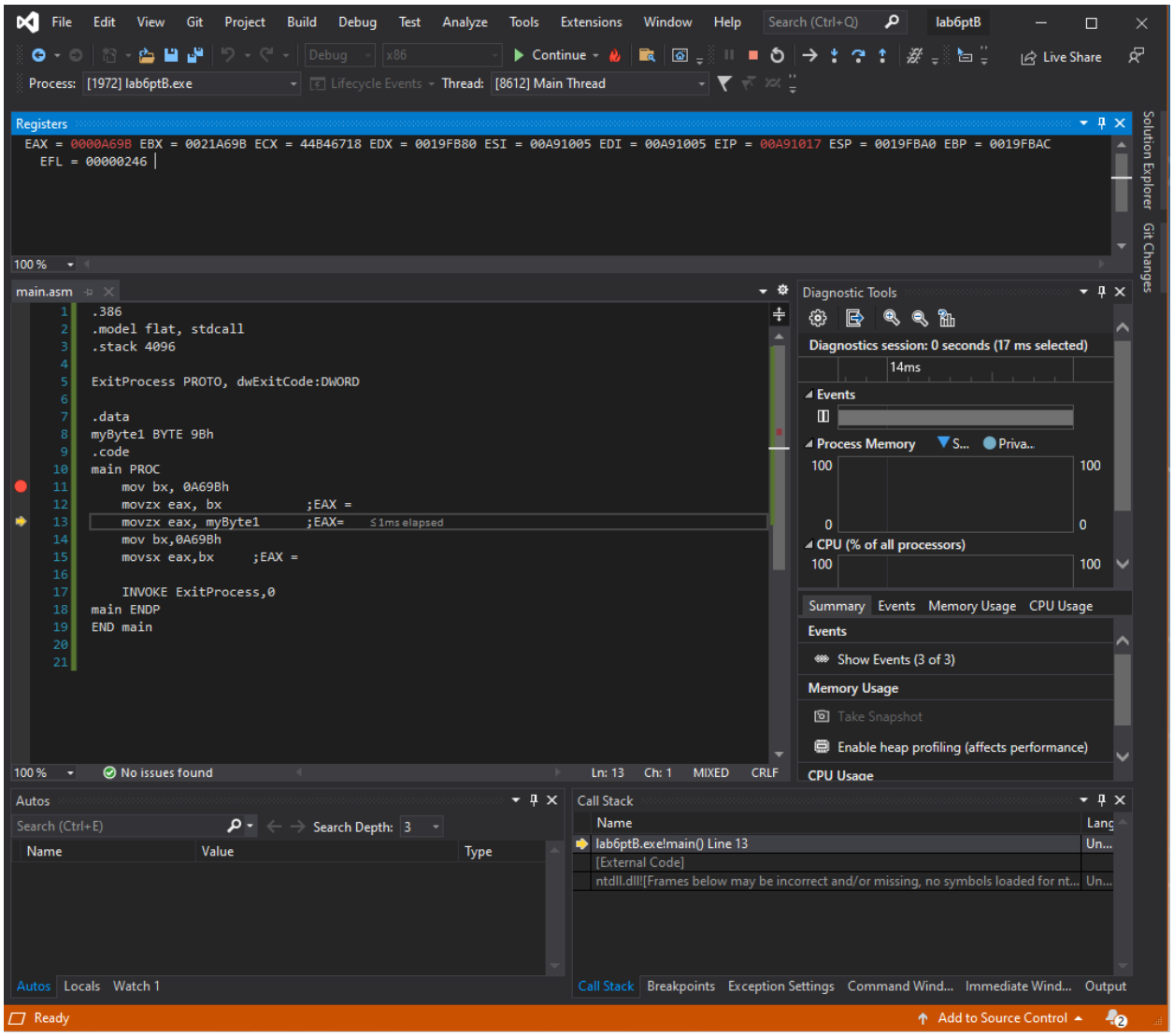
Explanation: when we do mov bx, 0A69Bh we are moving the value '0A69Bh' to the EBX register, and when we execute it we can see that the EBX register registers the value A69Bh and it doesn't take 0 into account because it's 0 as a normal value and doesn't add it into the EBX register.

Line number: 12

Instruction: movzx eax, bx

Register values: EAX: 0000A69B

Screenshot:



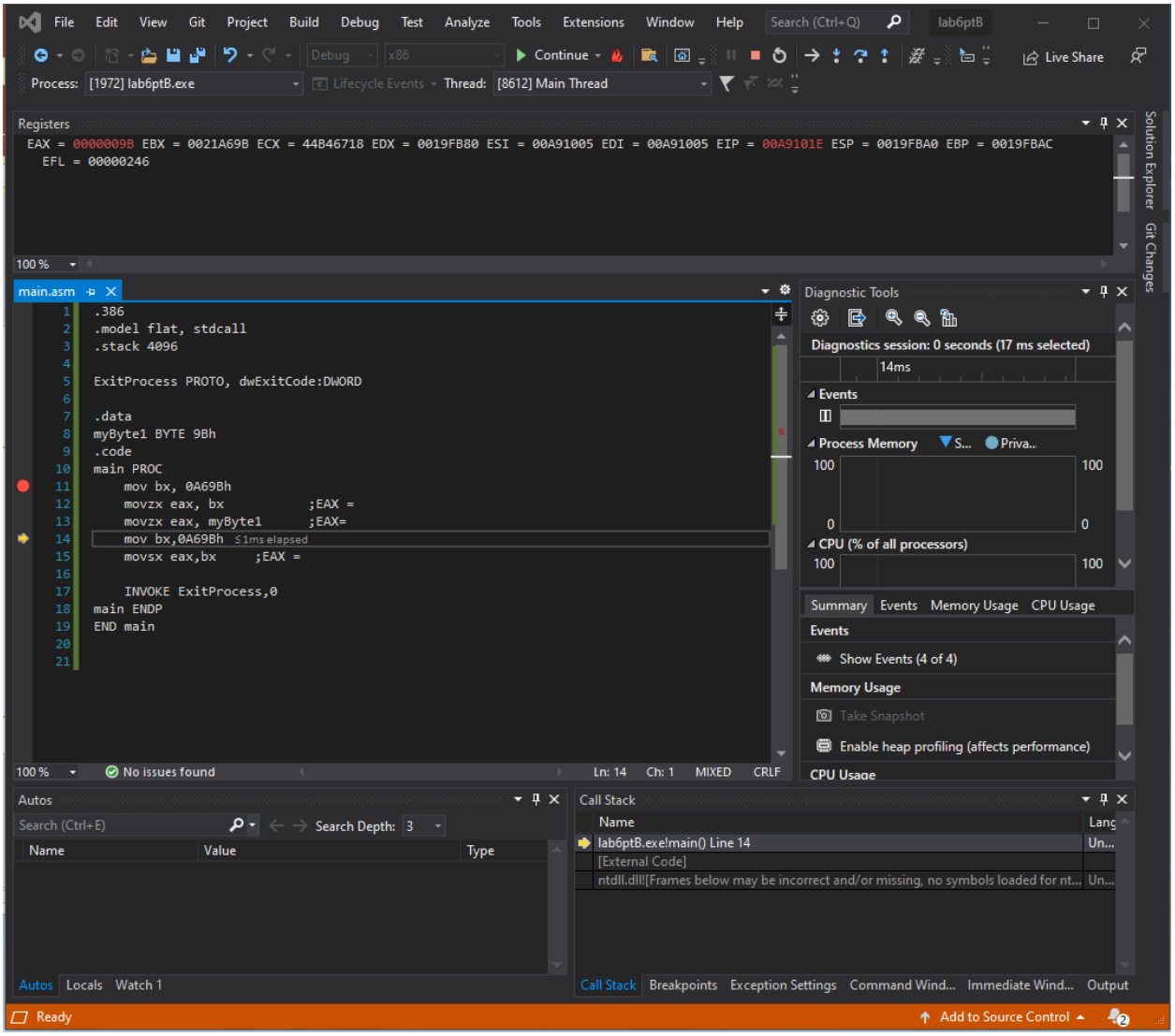
Explanation: In the line above we can see that movzx reads the contents of the register. The bx is 16 bit and EAX is 32 bit since there is a size difference we use movzx to get around the size error, so basically movzx stores 0's in the EAX register and add's the 16 bit value from bx towards the end of the EAX register. That's the reason why we can see the 0's in the beginning and the x values at the end.

Line number: 13

Instruction: movzx eax, myByte1

Register values: EAX: 0000009B

Screenshot:



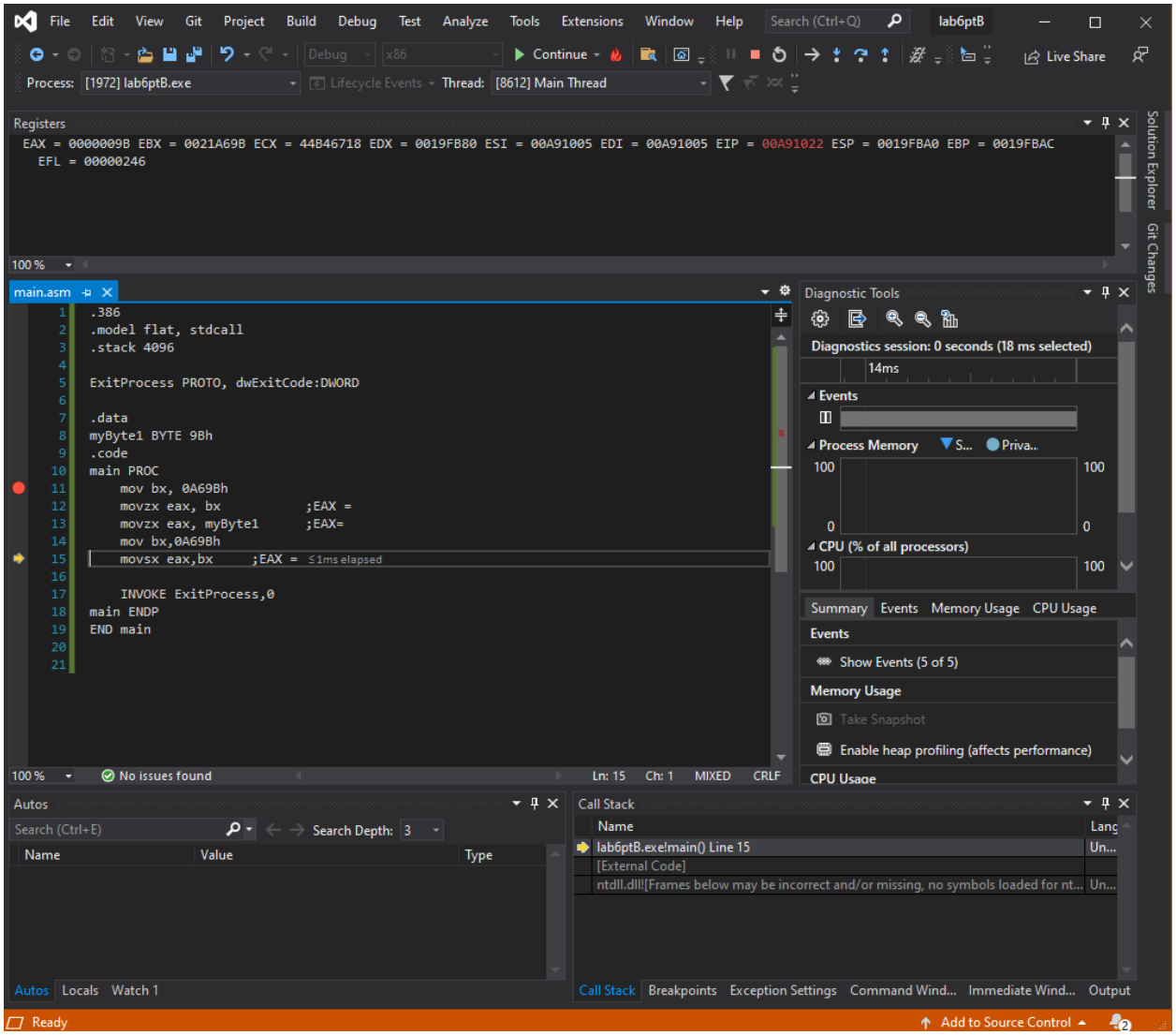
Explanation: movzx basically adds 0's to the eax register and then adds the value of myByte1, which is 9Bh to the eax register and the 'h' is not shown in the eax register because we add 'h' or other extension to make the computer understand the type of value we are putting in, like if it is a hexadecimal, decimal or octal value. So movzx does the same process as the above explanation line but this time with different variables and values.

Line number: 14

Instruction: mov bx,0A69Bh

Register values:

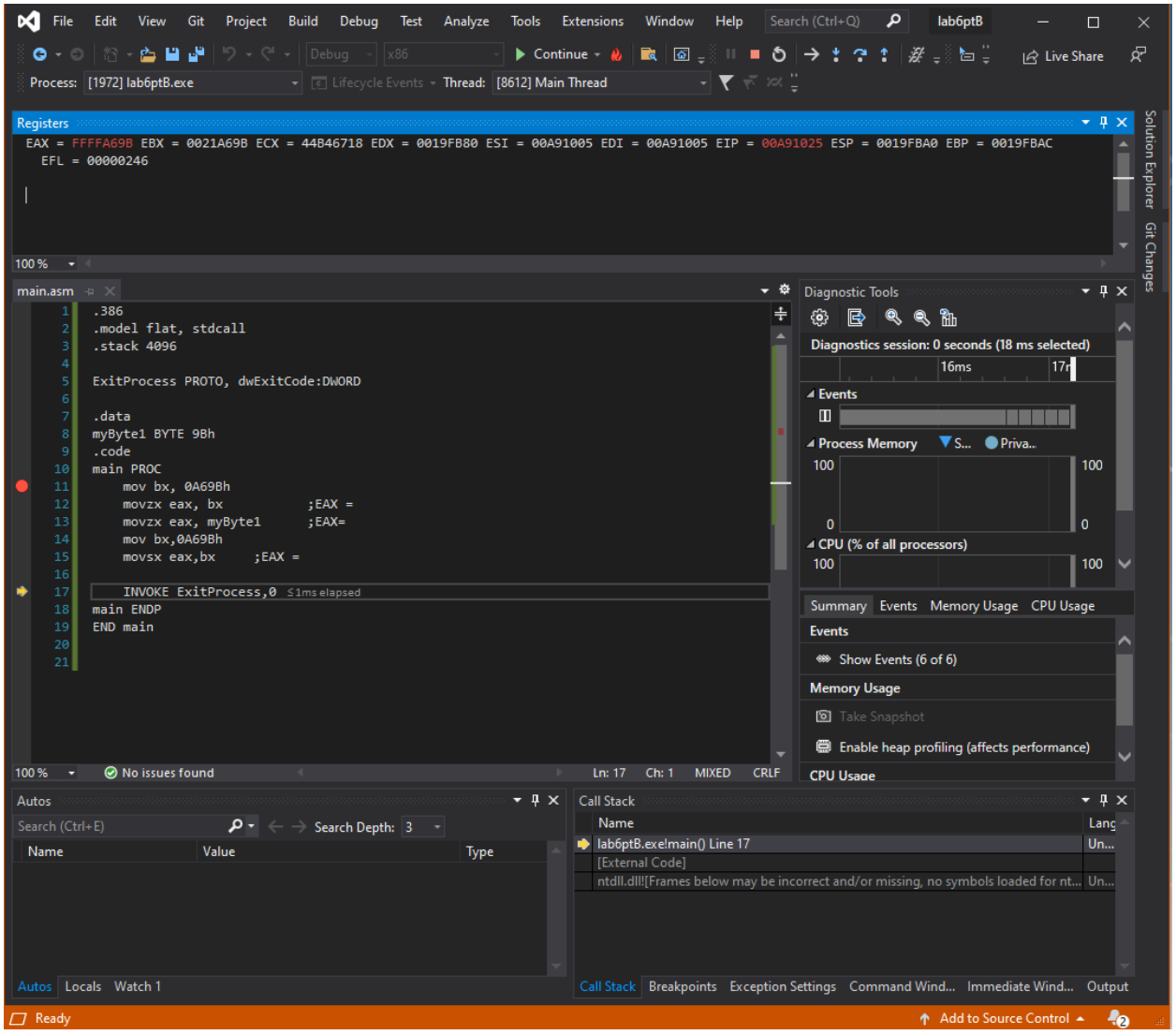
Screenshot:



Explanation: this instruction is moving the hexadecimal value 0A69Bh into the bx memory variable. This mov instruction only updates the BX register which is part of the EBX register. So when we look at the registers and EBX register we can see that EBX register is updated with the value ‘A69B’ and it starts with a 0 so that the computer doesn’t treating the starting ’A’ as a variable, so adding 0 makes the computer process ‘A69B’ as a whole value than just a variable and there is ‘h’ towards the end which indicates that its an hexadecimal number.

Line number: 15
Instruction: movsx eax,bx
Register values: EAX: FFFFA69B

Screenshot:

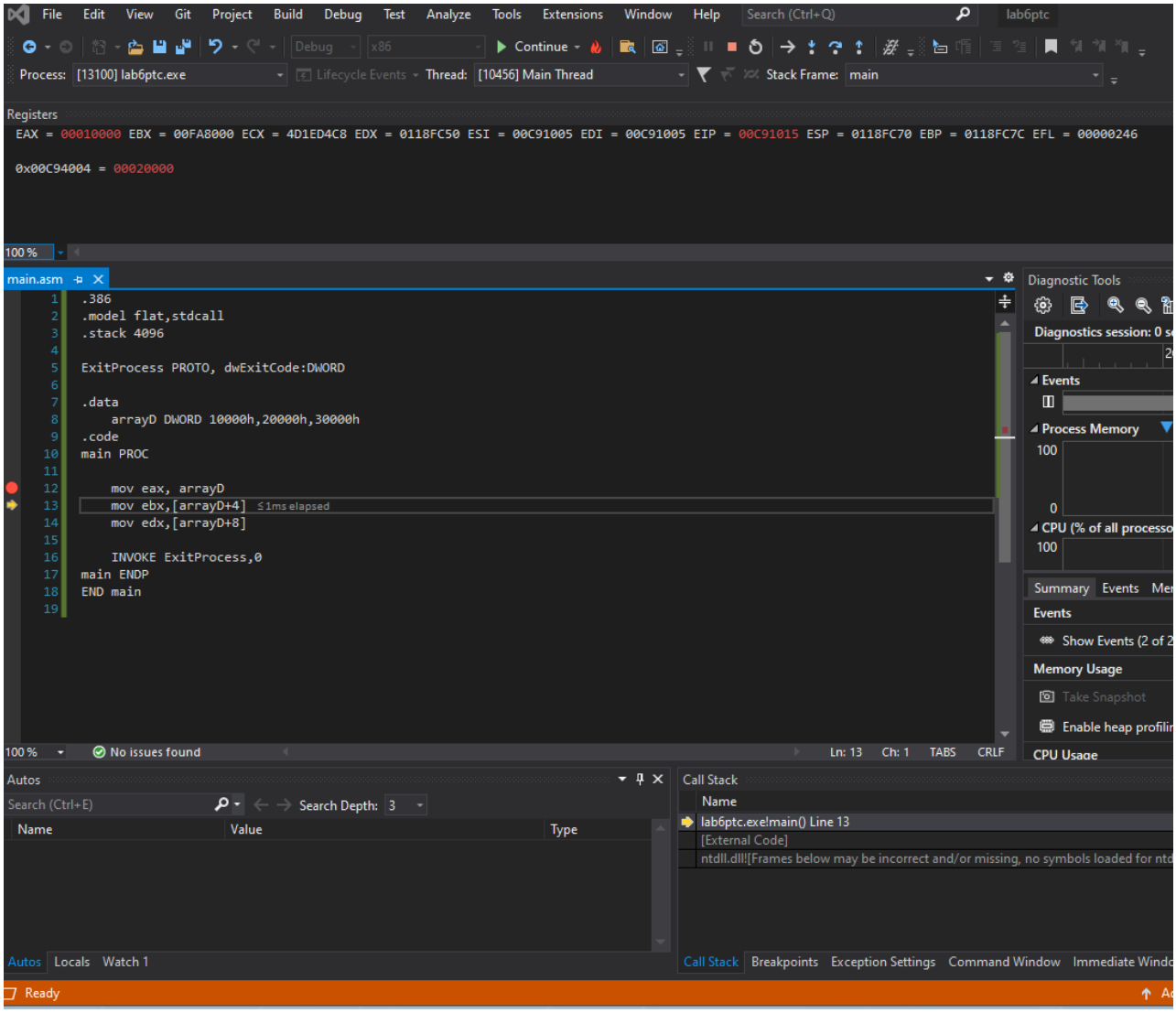


Explanation: movsx moves all the contains from bx to eax registers, it moves the value ‘A69B’ to the eax register and we can see that instead of 0’s now it’s F’s, which means that its a negative number. From the EAX register output we can see that the value ‘A69B’ is a signed number because it is being extended with F’s, if its not a signed number then it will be extended with 0’s.

Lab 6(c)

Debug through each line of instructions.
Take screenshot that includes code and register window.
Record the register content.
and explain the register contents.

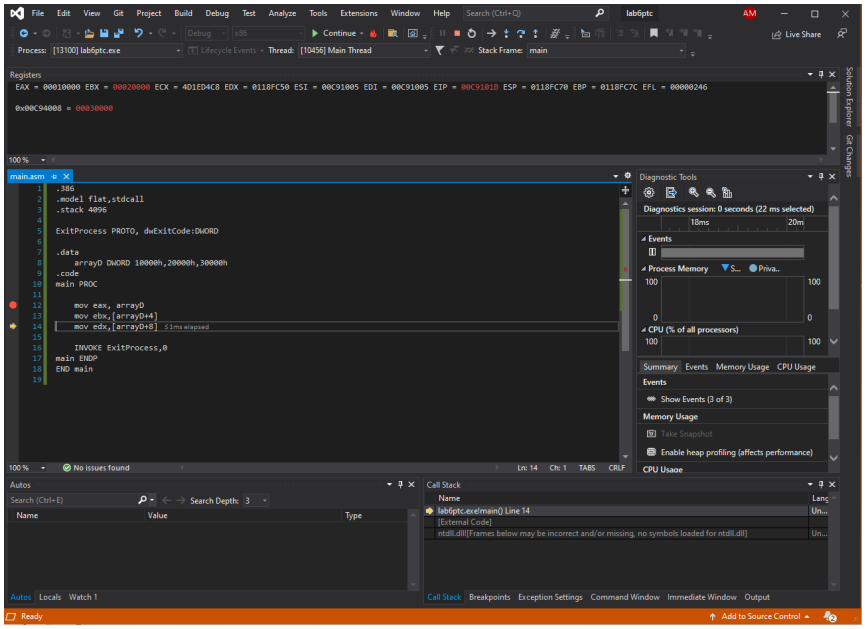
Line number: 12
Instruction: mov eax, arrayD
Register values: EAX = 00010000
Screenshot:



Explanation: Here we will be moving the first value of arrayD to the eax register using the mov command. This command moves the first item in the array, because it is just displayed as ‘mov

eax, arrayD’ which automatically considers it as the first one and then moves it to the eax register.

Line number: 13
Instruction: mov ebx,[arrayD+4]
Register values: EBX = 00020000
Screenshot:



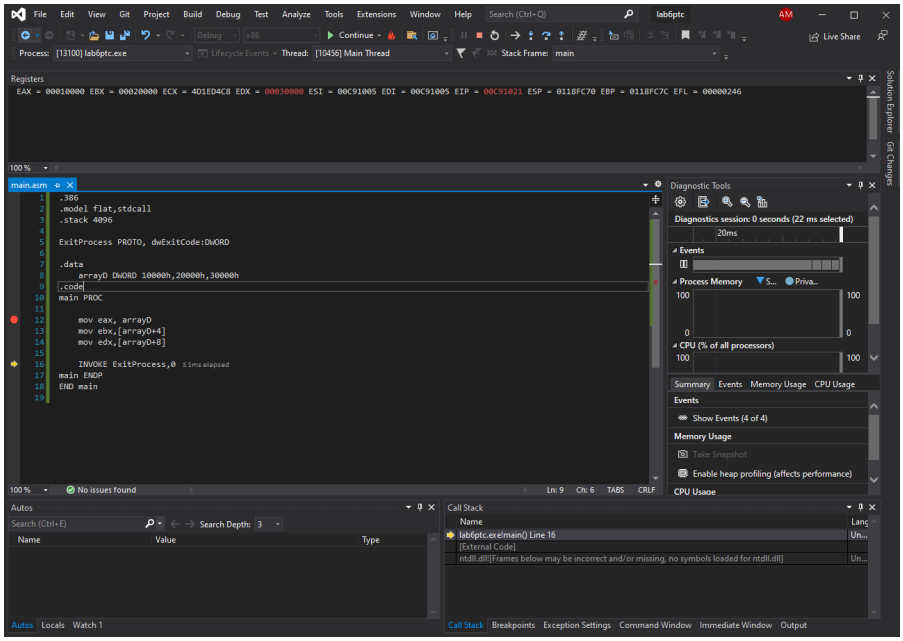
Explanation: In the above command we are able to access the eight byte in the array by adding 4 to the arradyD, which registers the value ‘20000’ to the EBX register. So when we do [arrayD+4] it basically looks for the value that is in the eight byte because since its a DWORD its 4 byte long and since we are looking at the second element of the array, it would be 8 bytes long and when we declare ‘[arrayD+4]’ it looks for the number in the 8 byte long array list, which is 20000, which it then moves it into the EBX register and stores it there.

Line number: 14

Instruction: mov edx,[array+8]

Register values: EDX = 00030000

Screenshot:



Explanation: This command is doing the same as the one before. In this instruction we can see that the 12 bytes long value in the array is 30000 and this byte can be accessed by adding 8 to it and then the [array+8] is done, it basically gives the value of 30000, which is then stored in the edx register using the mov operation.

Lab 6(d)

Create a new project to run the following program.
Declare an array in the data segment: arrayB WORD 1,2,3,4
Write code to Rearrange the array as follows: 4,3,1,2

Add the screenshot of your code here.