**Lab03: Getting familiar with ASSEMBLY**

**本次实验的题目3,5,6需要提交答案**

窗体顶端

**1. Using Disassembly Window to inspect assembly code in Visual Studio 2017**

Open any C++ solution created in Visual Studio 2017. We use Hello.c as the example program in the following figure. Set a breakpoint on the first line of method main and start to debug your program. The execution will pause at where you set the break point and wait for further interaction. Now click Debug on the *Menu*, *Menu > Windows > Disassembly.* Then a new disassembly window will pop up with tab “Disassembly”.

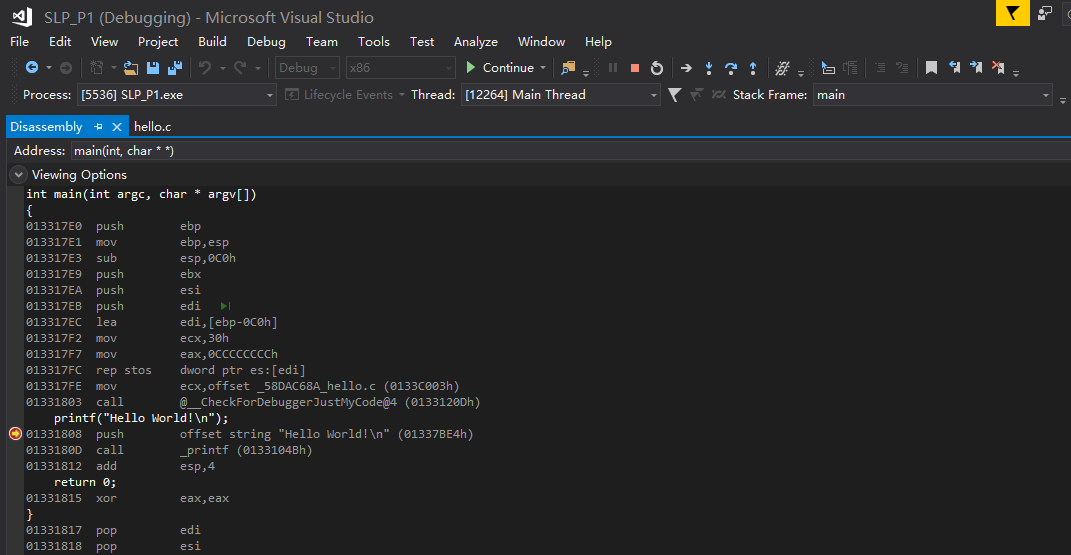


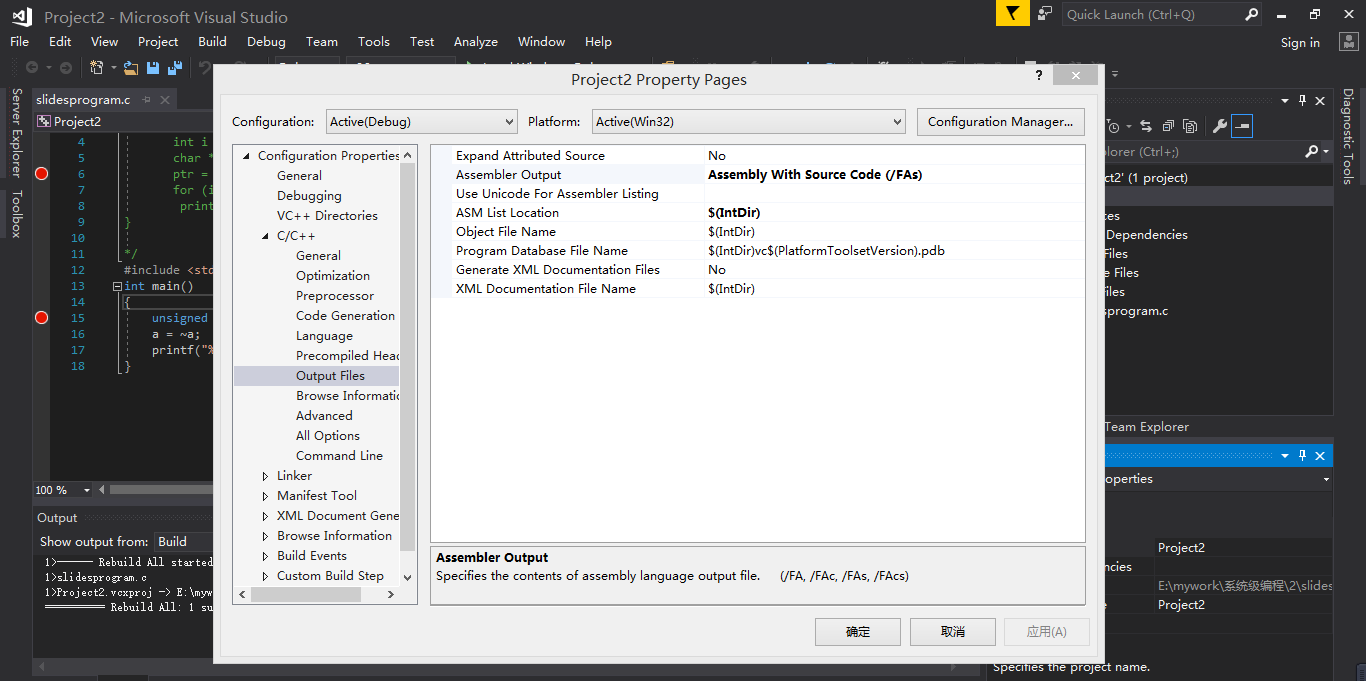
Figure 1 Open Disassembly Window

You can press F10 to execute one assembly instruction or F5 from current break point to the next one.

**2. Generate \*.asm file from your C or C++ program**

Right click on your project, then select *Properties* from the pop-up menu. Properties> *Configuration properties > C/C++/ >Output Files > Assembler Output*, select one from the list(You’d better select *Assembly With Source Code* for convenient.). After rebuild your program, you can find file \*.asm in the Debug folder and it can be opened by text editor.

Now create a C/C++ project, or open an existing one, try to generate an assembly file from your source code.



**3. Understand the basic structures of assembly instructions**

You have already learned how to inspect assembly code of your C or C++ program in topic 1 and 2. Now choose one of them and complete the following practice.

This practice is designed to help you understand the basic structure of assembly code segments transformed from high level language C or C++, meanwhile you can prepare for the challenge of Binary Bomb lab.

Please download ***practice2.c***, inspect each c program in the disassembly window following the instructions and answer all the questions. You are expected to write down all the answers in a word file studentid\_studentname\_lab03.doc.

**4. MASM Assembly in Visual Studio 2017**

Practice how to create an assembly program in Visual Studio 2017.

Note: MASM in Visual Studio don’t support DOS interrupt.

#### Step 1: Create a clean project

*File | New | Project…*Expand the ‘Visual C++‘ tree, Select ‘Windows Desktop‘|‘Windows Desktop Wizard‘, name it as “asm1”.

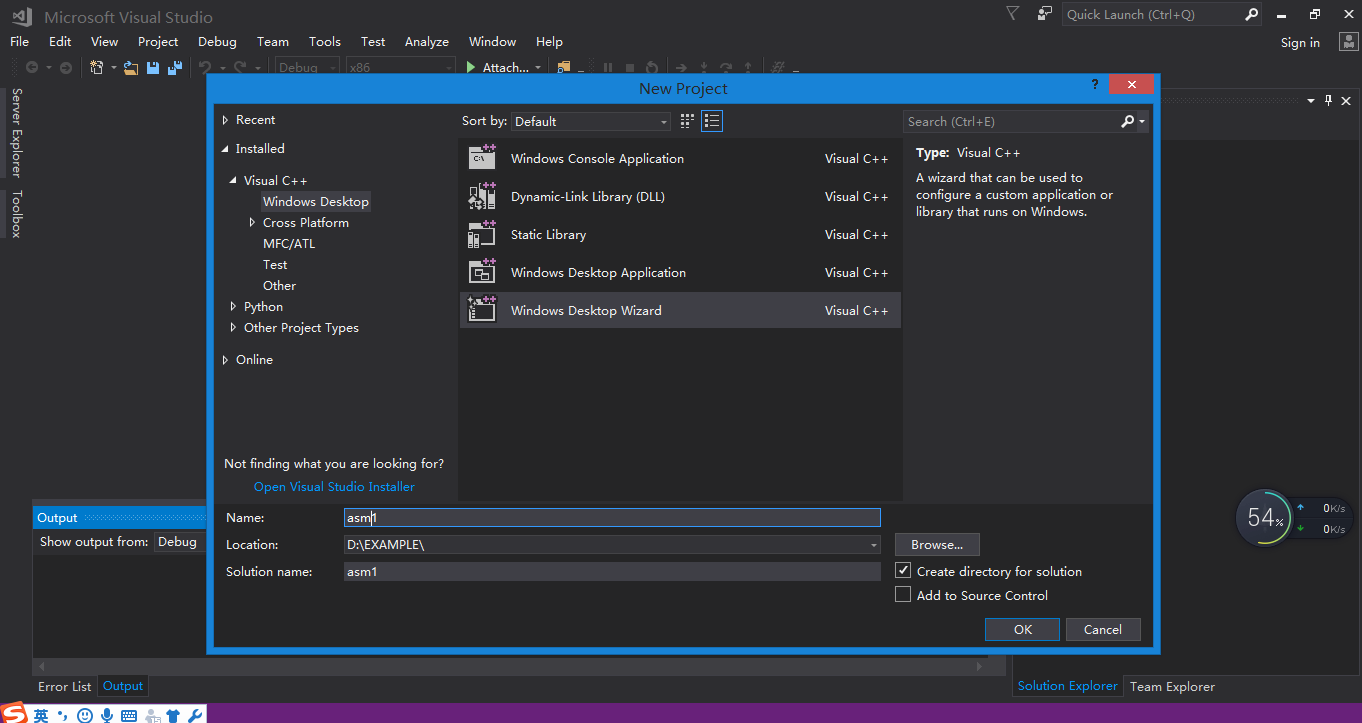


Figure 1 Create a Project

In the next pop-up window, select “Empty Project” and unselect “Security Development Lifecycle (SDL) checks”.

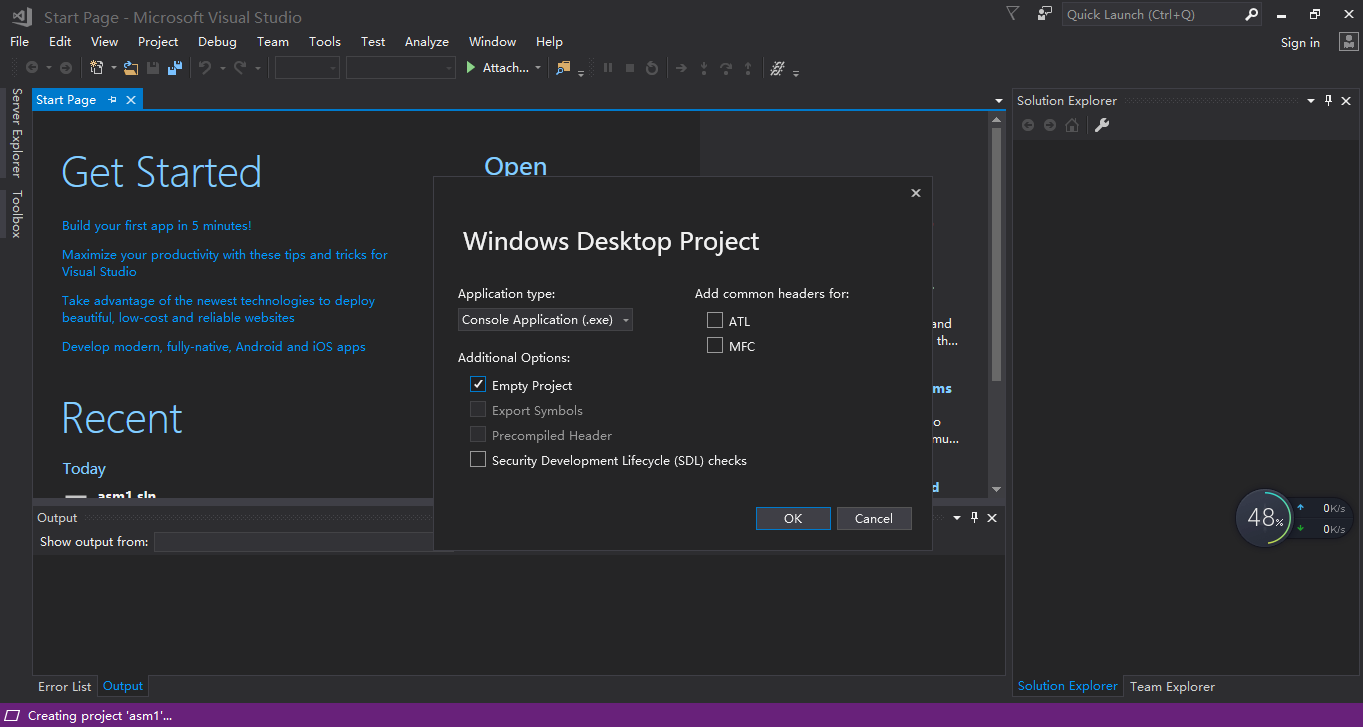


Figure 2 create an empty project

**Step 2: Acquire the MASM options.**

Now right click on the Project “asm1” in the Solution Explorer and select ‘*Build Customizations…*‘|*’Build Dependencies’.*

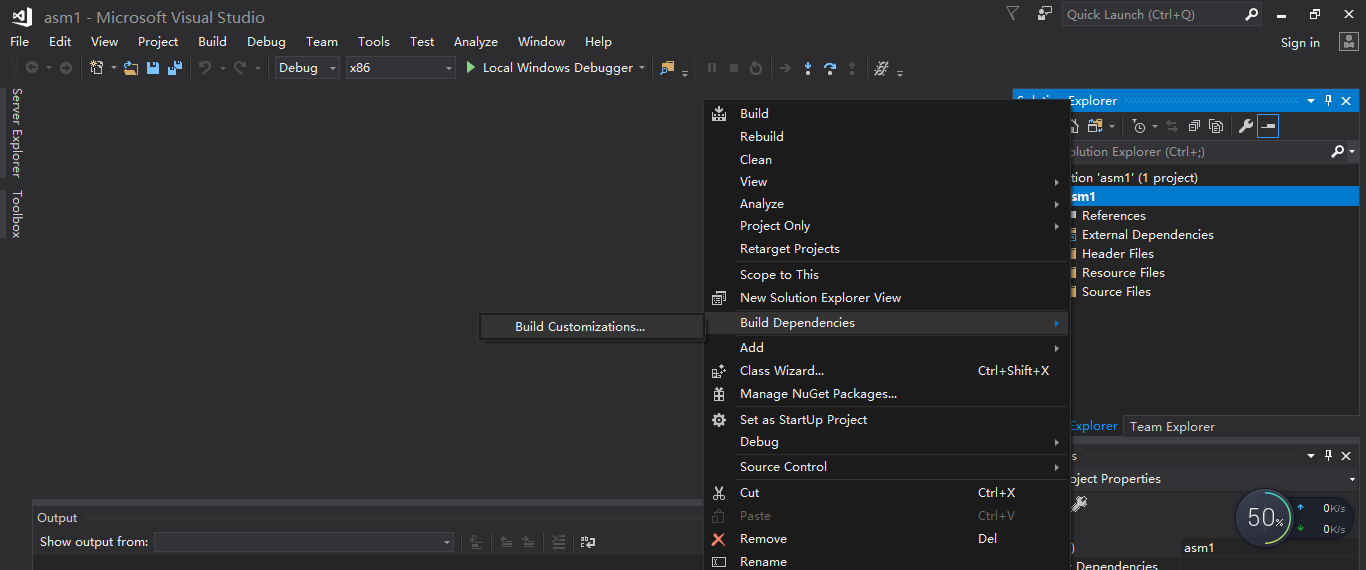


Figure 3 Set build dependence

In the following pop-up window, tick the ‘*masm*‘ box and say OK.

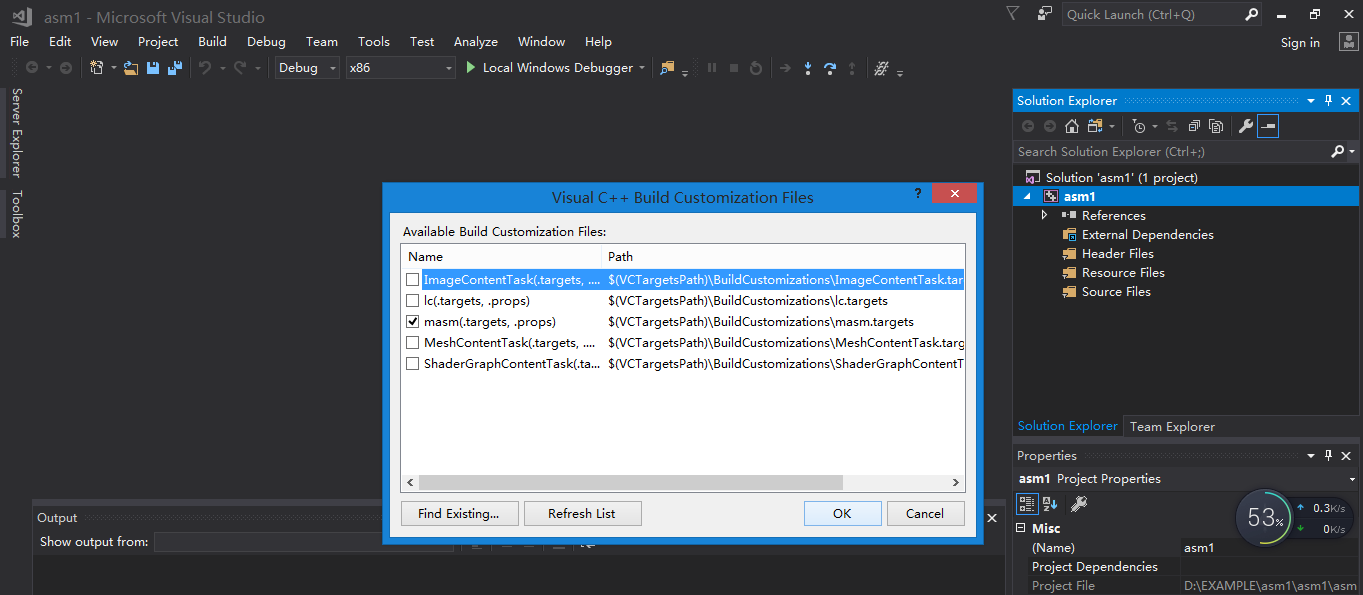


Figure 4 Select ‘masm’

**Step 3: Add ASM file to project.**

Add a new file to the project with the .asm extension by right clicking on the Source File in the Solution Explorer and selecting ‘*Add | New Item…*‘ then ‘*C++ File*‘.

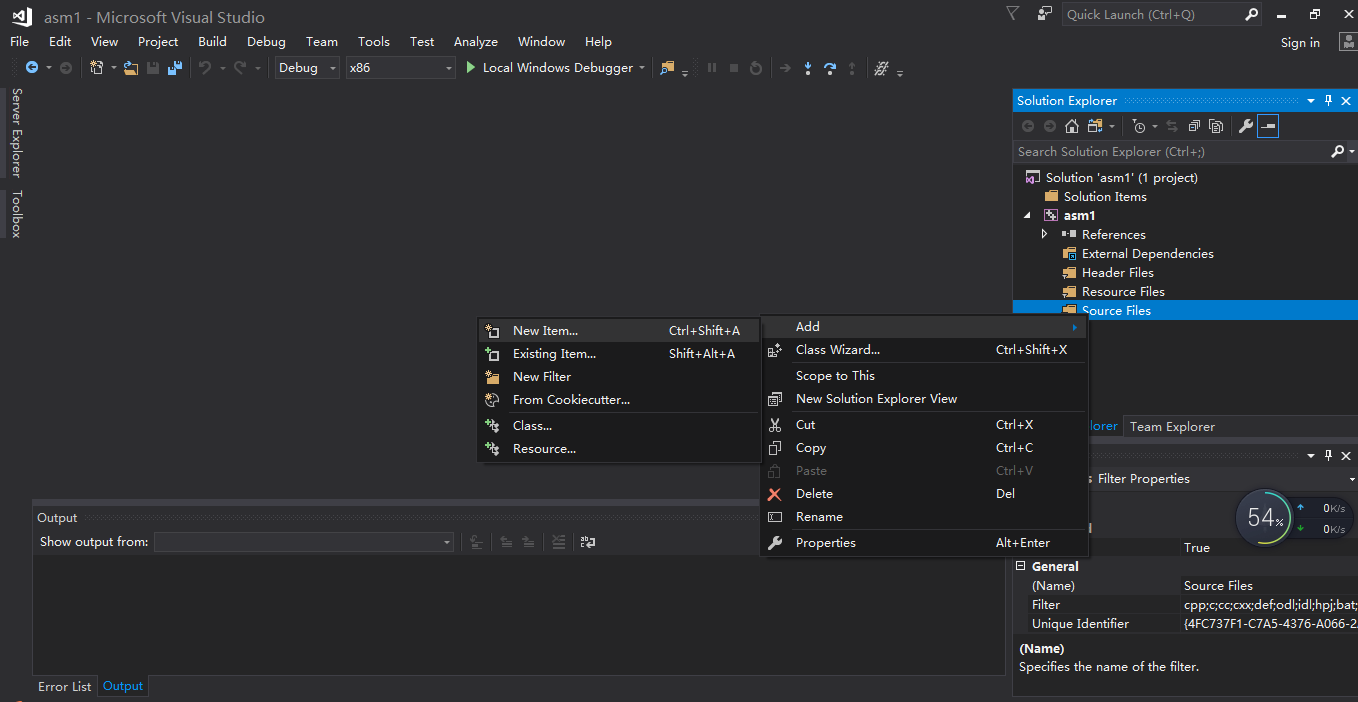


Figure 5 Add new item to the project

Enter a filename ending with .asm (e.g. *test.asm*). Say OK. Now you can edit your assembly program.

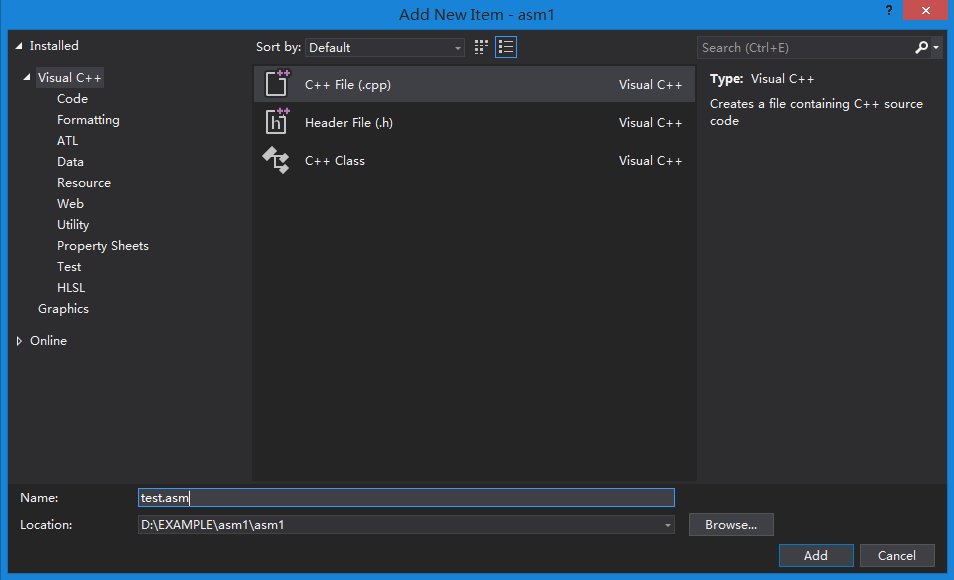


Figure 6 Add ASM file to the project

**Step 4: Edit your program.**

|  |
| --- |
| .386 ;instruction set  .model flat,stdcall ;model  .code ;this is a code segment  main proc ; define main function  mov eax,1  mov ebx,2  add eax,ebx  ret ;return  main endp ;end of main  end ;end of program |

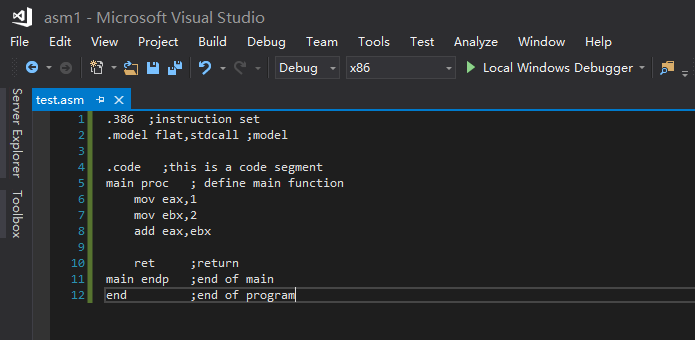


Figure 7 An example program

**Step 5: Configure the linker**

There are a few critical things to set up in the Linker options in order to get it to work. Now right click on the Project “asm1” in the Solution Explorer and select *Configuration Properties > Linker > System> SubSystem*, set it to “Console (/SUBSYSTEM:CONSOLE)”.

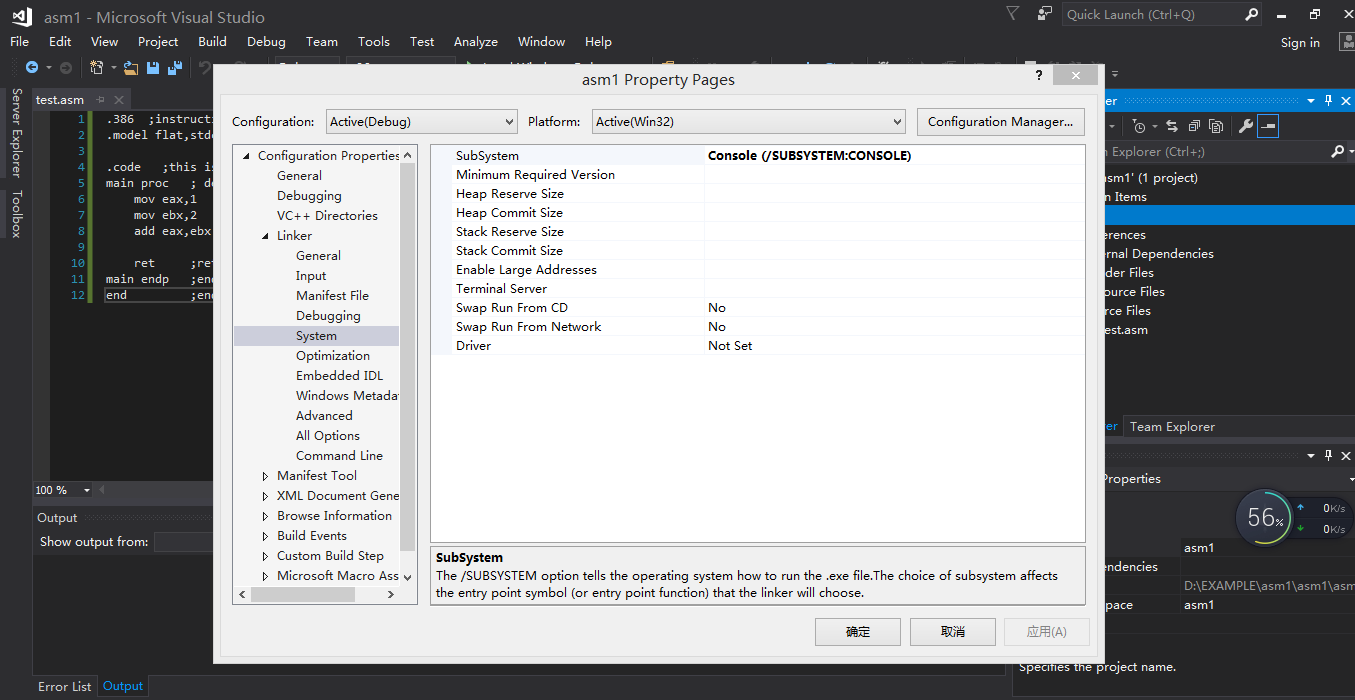


Figure 8 Configure subsystem

Instead of using main, you can give your method any name (e.g. nicetry). Set the entry point to the name of your main method(function) (e.g. nicetry). *Configuration Properties > Linker > Advanced > EntryPoint*

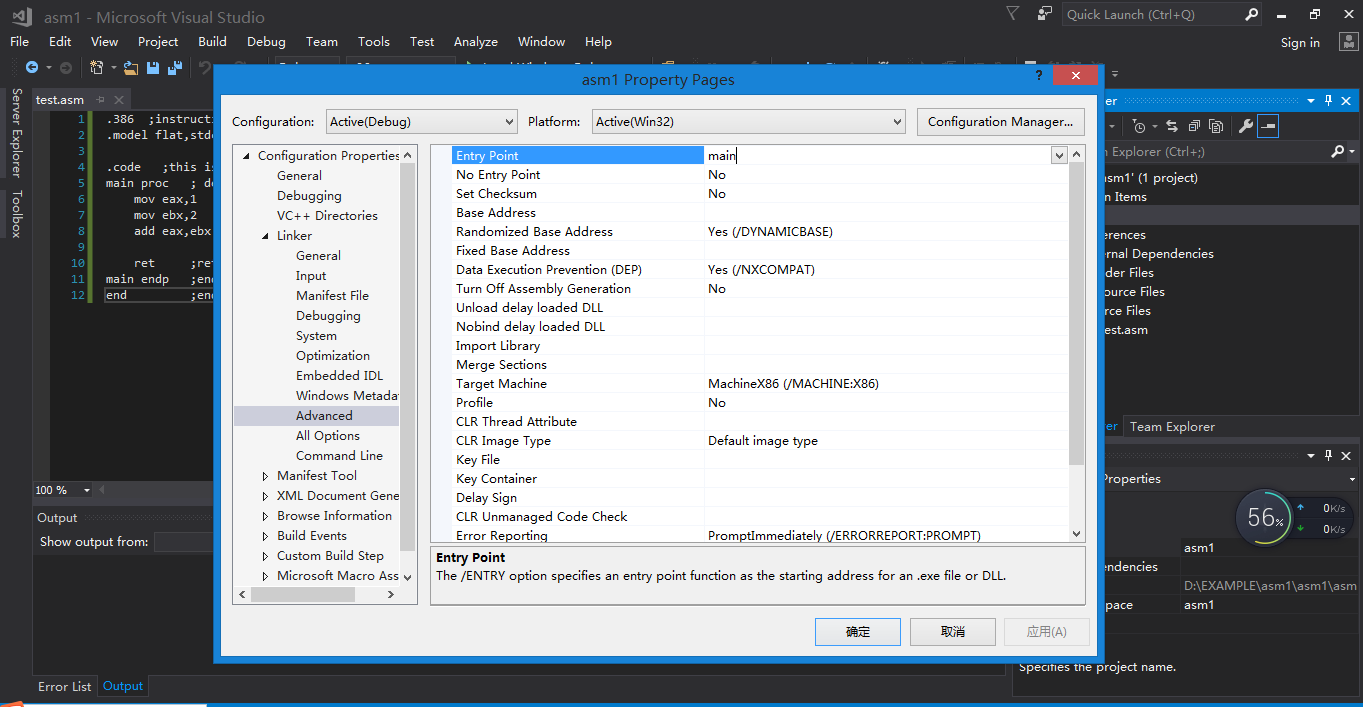


Figure 9 Configure entry point

**Step 6: Set break points and debug it**

Now you can set break points and step through your code much as you are used to doing with C++.Remember that you can only set break points on the instruction lines, not on directive lines. Click on the left most column to set the break point on a certain line, then start debugging the program and it will stop execute on that line.

You can watch the values of registers by “Registers” window. *Menu > Debug > Windows > Registers*

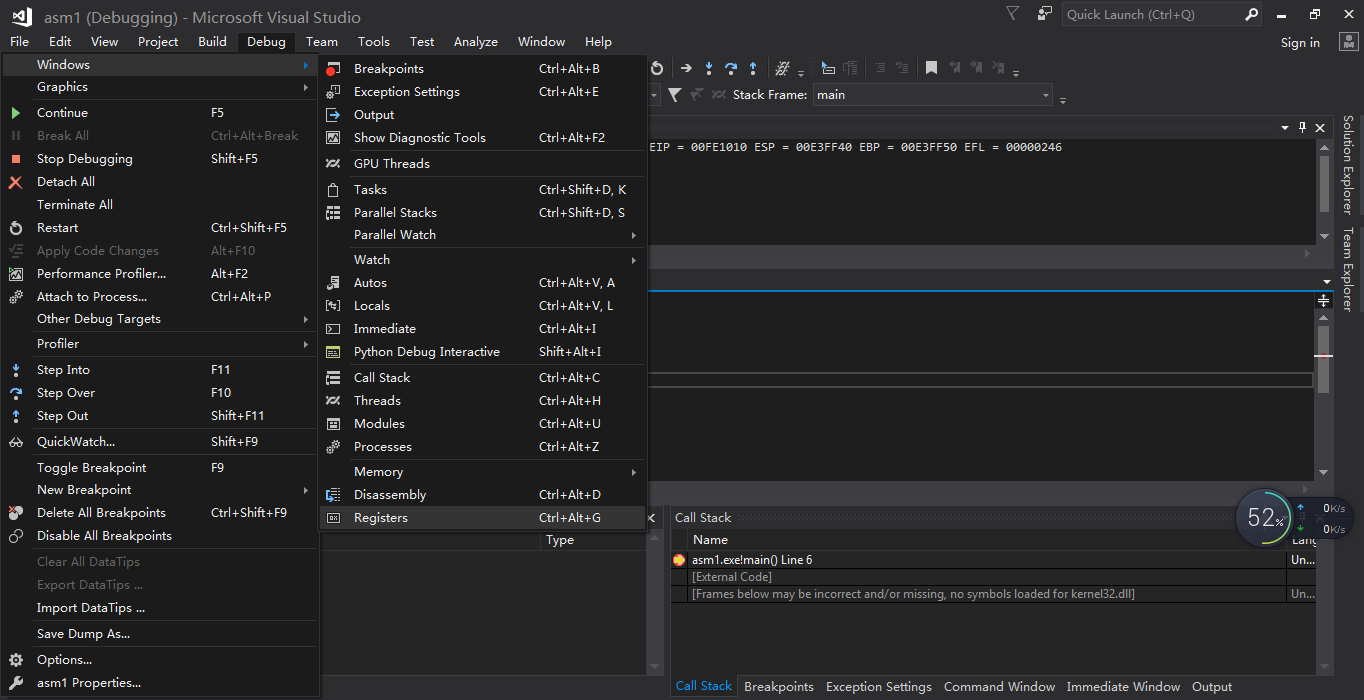


Figure 10 Display registers window

Press F1 to execute instruction one by one and F5 from current break point to the next one. The values in the register window may be white/black or red. Red signifies that that value has changed by the instruction just executed.

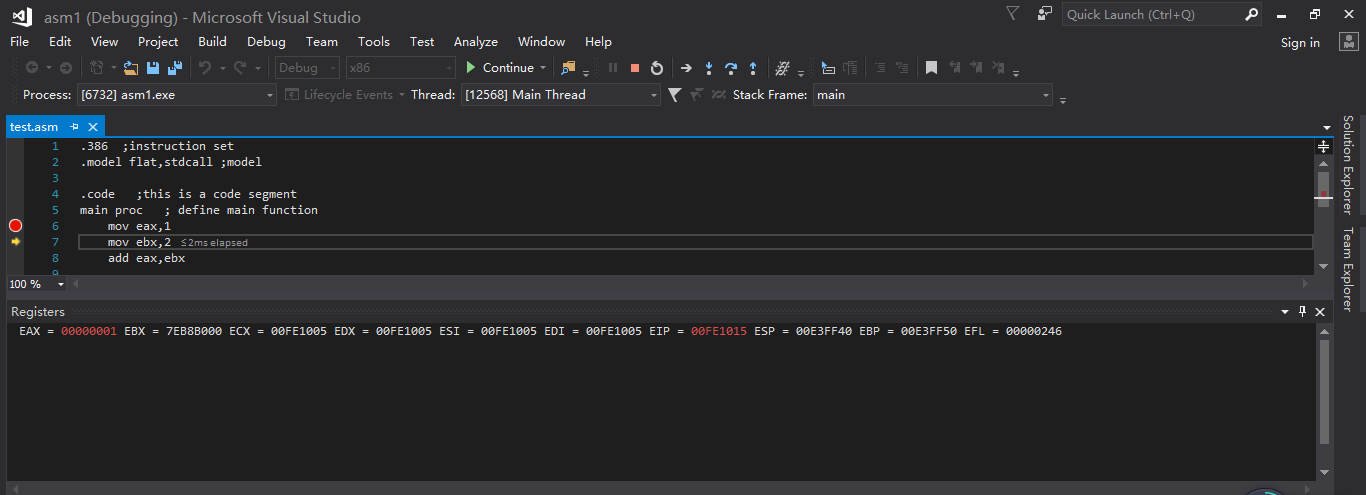


Figure 11 Watch values of registers

**5. Write your own assembly program and practice transferring parameters between functions**

窗体底端

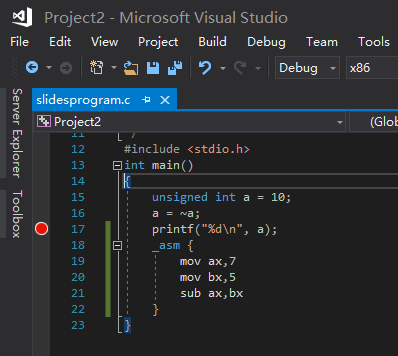
Given the following program segment, make up the missing parts. In the program, function ***main*** will call function ***xaddy*** to add two numbers (summand and addend) which are saved in registers ecx and edx. These two parameters (summand and addend) should be transferred from the caller ***main*** to the callee ***xaddy*** by stack before the function ***xaddy*** is called.The sum should be saved in register eax, since values can be returned to the caller by registers.

|  |
| --- |
| .386 ;instruction set  .model flat,stdcall ;model  .code ;this is a code segment  main proc ; define main function  mov ecx,1 ; summand  mov edx,2 ; addend        ret ;return  main endp ;end of main  xaddy proc    xaddy endp  end ;end of program |

You are expected to write down the answers in a word file studentid\_studentname\_lab03.doc.

**6. Insert assembly segment into a C or C++ program**

You are also allowed to insert a segment of assembly code into your current C or C++ program in Visual Studio C++. Open an existing C or C++ project or create a new one. Within the C or C++ code, at where you want to insert the assembly code, type \_asm{ }. Write down all your assembly code within the { } then compile the project.



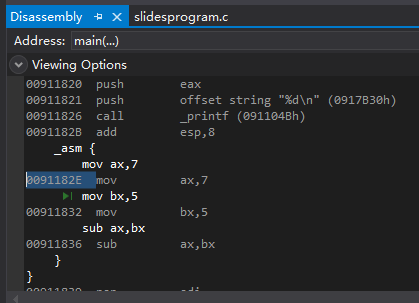
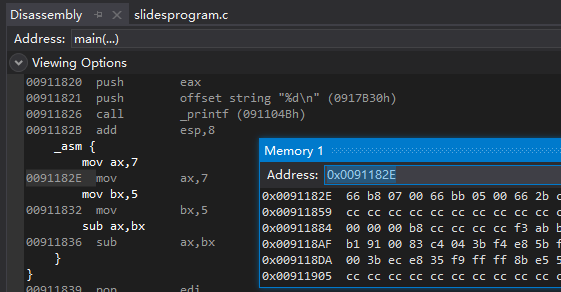
This is an easy way to test whether your assembly code segment is correct. You can also get the corresponding machine code of a certain assembly code by:

1. Inspect the Disassembly Window, get the memory address of the machine code
2. Open Memory Window by Debug>Windows>Memory>Memory 1
3. Type in the memory address to redirect memory window.
4. Since “mov ax,7” is saved at 0x0091182E and “mov bx,5” at 0x00911832, it indicates the machine code of “mov ax,7” is 4 bytes long, from 0x0091182E to 0x00911831.
5. So the machine code is 00 07 b8 66.
6. After you have learned all above, try to decode the machine code for the following assembly instructions and You are expected to write down the answers in a word file studentid\_studentname\_lab03.doc.

mov EAX, 0xdeadbeaf

push 0X0028FE60

ret

**7. Know more about assembly IDE**

Using DOSBOX & MASM: https://blog.csdn.net/xyisv/article/details/69062382

**8. 实验报告提交**

Sumbit a report named studentid\_studentname\_lab03.doc which containing answers to practice3 “3. Understand the basic structures of assembly instructions” and practice5 “5. Write your own assembly program and practice transferring parameters between functions”.