Getting Started with MASM and Visual Studio 2013

Adapted from Kip Irvine

This tutorial assumes that you are using Visual Studio 2013 (including Visual Studio 2013 Express
for Windows Desktop, and Visual Studio Community 2013 edition) to work with the Microsoft assembler. Visual Studio 2013 Express and Visual Studio Community 2013 may be downloaded from Microsoft.com. *(Note: the directions shown here are **not** designed for use with "Visual Studio Express 2013 for Windows" That product is designed for creating Windows Store apps in Visual Basic and C#.)*

Topics:

Required Setup for 32-bit Applications

First, you must install Visual Studio and select the C++ language configuration option the firs t time you run it. All versions of Visual Studio include the Microsoft Assembler (MASM) version 12.0. You can verify that the Microsoft Assembler is installed by looking for the file **ml.exe** in th e \vc\bin folder of your Visual Studio installation directory, such as c:\Program Files\Microsoft Visual Studio 12.0\vc\bin.

Setting up Visual Studio

You will only have to do these steps the first time you use Visual Studio.

Add the *Start Without Debugging* command to the Debug menu

It's very useful to run programs without having to debug them. To do that, you will want to add a new command to the Debug menu: Start Without Debugging. Here's how to do it:

1. From the Tools, menu, select *Customize*.
2. Select the *Commands* tab.
3. Select Menu bar (radio button).
4. Click the *Add Command* button.
5. Select *Debug* from the Categories list.
6. Select *Start Without Debugging* in the right-hand list box.
7. Click the OK button.
8. Click the Close button.

In fact, you can use the same sequence to customize any of the menus and toolbars in Visual Studio.

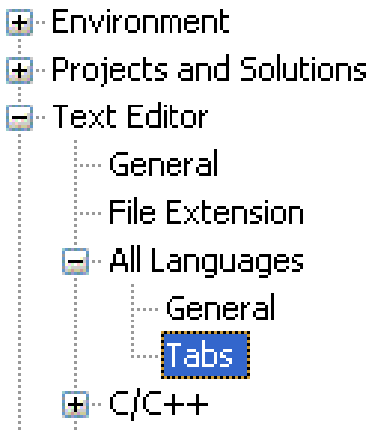
Select the C++ Configuration

(Skip this topic if you installed Visual Studio Express.) Visual Studio Professional, Ultimate, and Premium editions support multiple programming languages and application types. The C++ programming language configuration most closely matches that of assembly language programming, so we suggest the following steps:

1. Select Tools | Import and Export Settings from the menu
2. Select the "Import selected environment settings" radio button
3. Select the "No, just import..." radio button
4. Select "Visual C++" from the Default Settings List and click the Next button
5. Click the Finish button, then click the Close button
6. Notice the tabs on the left and right sides of the Visual Studio workspace. Close the Server Explorer, Toolbox, and Properties tabs. Use the mouse to drag the Solution Explorer tool window to the right side of the workspace. You can also select other tabs at the bottom of this window, such as "Class View", "Property Manager", and "Team Explorer", and close them. They will not be used in the future. If you need to bring back the Solution Explorer window at any time in the future, select View from the menu, and locate Solution Explorer in the list of views.

Set the Tab Size to 5

(This is an optional step.) Start Visual Studio, and select **Options** from the **Tools** menu. Select **Text Editor**, Select **All Languages**, and select **Tabs**. Optionally, you may want to select the **Insert spaces** radio button:



Set the Tab Size and Indent Size to 5.

Tutorial: Building a 32-Bit Assembly Language Program

Now you're ready to open and build your first 32-bit project.

Opening a Project

Visual Studio requires assembly language source files to belong to a *project*, which is a kind of container. A project holds configuration information such as the locations of the assembler, linker, and required libraries. A project has its own folder, and it holds the names and locations of all files belonging to it. We have created a sample project folder in the zipfile for Lab 4, and its name is *Project32*.

Do the following steps, in order:

1. Copy the **Lab4** folder to a location on your hard drive that permits you to read and write files. You can use a USB drive, although Visual Studio may run a little more slowly when it creates temporary files during the build process.
2. Start Visual Studio.
3. To begin, open our sample Visual Studio project file by selecting **File/Open/Project** from the Visual Studio menu.
4. Navigate to the **Lab4/Project32** folder and select the file named **Project.sln**.
5. Once the project has been opened, you will see the project name in the Solution Explorer window. If there are any files with .asm file extensions in the Solution Explorer window, select and delete them now.
6. Next, you will add an existing source code file to the project: In the Solution Explorer window, right-click on **Project**, select **Add**, select **Existing Item**, navigate to the "**Lab4/**" folder, select **AddTwo.asm**, and click the **Add** button to close the dialog window. (You can use this sequence of commands in the future to add any asm file to a project.) You should now see the AddTwo.asm file in the Solution Explorer window.
7. Next, open the AddTwo.asm for editing by double-clicking its filename in the Solution Explorer window.

You should see the following program in the editor window:

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

.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
```

In the future, you can use this file as a starting point to create new programs by copying it and renaming the copy in the Solution Explorer window.

Build the Program

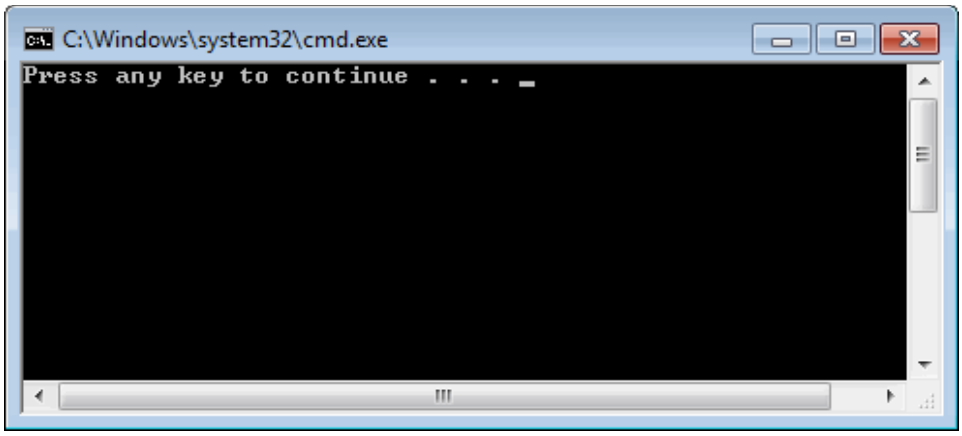
Now you will build (assemble and link) the sample program. Select **Build Project** from the Build menu. In the Output window for Visual Studio at the bottom of the screen, you should see messages similar to the following, indicating the build progress:

```
1>------ Build started: Project: Project32, Configuration: Debug Win32 ------
1>  Assembling ..\..\..\..\51_Fall2015\ICS 51\Labs\Lab4\AddTwo.asm...
1>  Project32.vcxproj -> c:\users\shannon\documents\visual studio
2013\Projects\Project32\Debug\Project32.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

If you do not see these messages, the project has probably not been modified since it was last built. No problem--just select **Rebuild Project** from the Build menu.

Run the Program

Select **Start without Debugging** from the Debug menu. The following console window should appear, although your window will be larger than the one shown here:



The "Press any key to continue..." message is automatically generated by Visual Studio.

Congratulations, you have just run your first Assembly Language program!

Press any key to close the Console window.

TIP: When you assembled and linked the project, a file named **Project.exe** was created inside the project's \Debug folder. This file executes when you run the project. You can execute Project.exe by double-clicking its name inside Windows Explorer, but it will just flash on the screen and disappear. That is because Windows Explorer does not pause the display before closing the command window. On the other hand, you can open a Command prompt window, move to the Debug directory, and run Project.exe by typing "Project" (without the quotes). You will need to do some reading on Windows shell commands if you plan to use the command line.

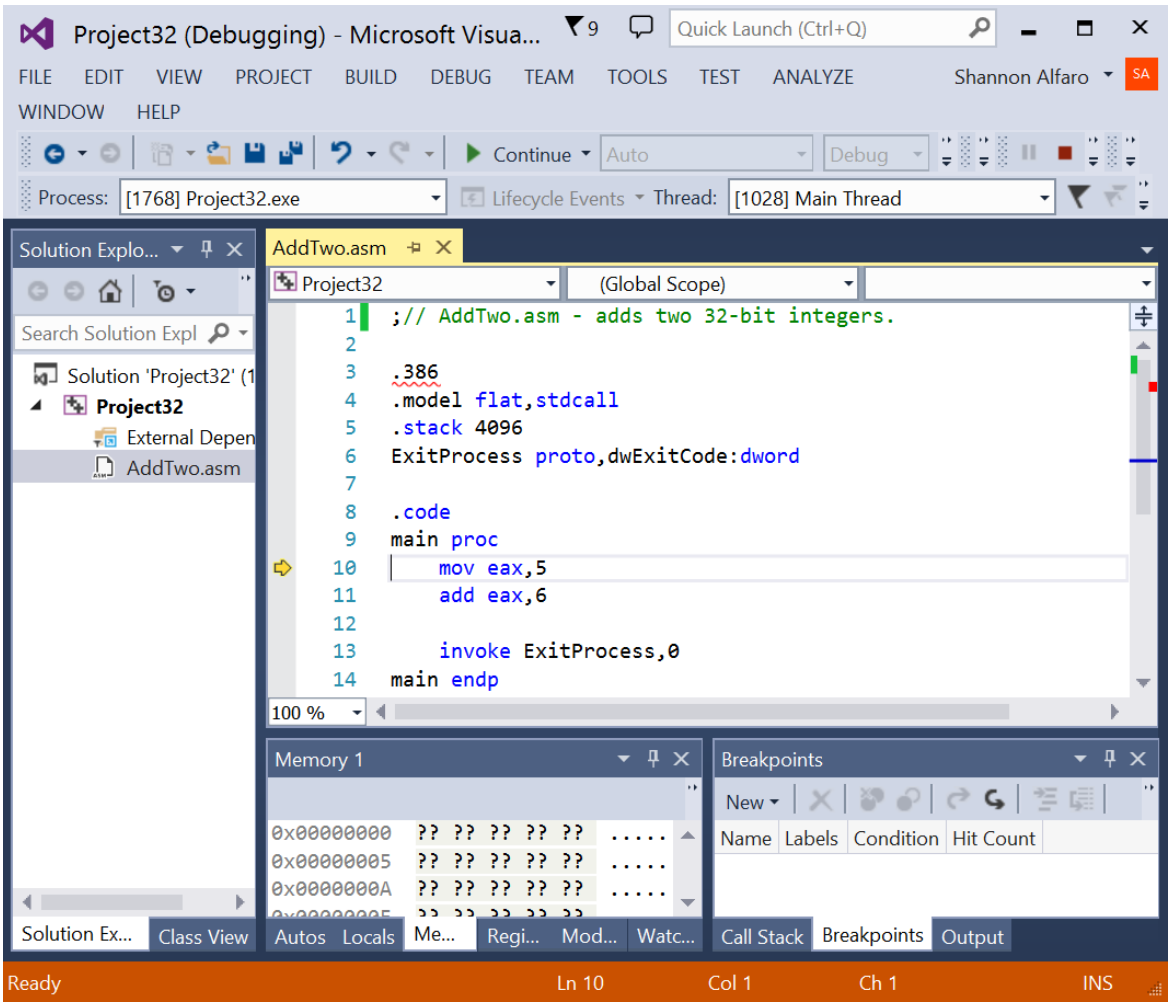
Any time you want to remove a source file from the Visual Studio window, right-click its filename and select **Remove**. The file will not be deleted from the file system. On the other hand, if you want to delete the file, select it and press the Del key.

Step 5: Running the Sample Program in Debug Mode

In this step, you set a breakpoint inside the sample program. Then you use the Visual Studio debugger to step through the program's execution one statement at a time.

1. Make sure the ASM source code file is open in the editor window.
2. To begin stepping through your program in Debug mode, press the F10 key.
3. A yellow arrow should appear next to the first program statement. The arrow indicates that the statement is next to be executed.

4. Press the F10 key (called *Step Over*) to execute the current statement. Continue pressing F10 until the program is about to execute the **invoke** statement.
5. A small black window icon should appear on your Windows status bar. Open it and look at the contents of the Command window. The window should be blank because this program does not display any output.
6. Press F10 one more time to end the program.



Registers

If you want to display the CPU registers, do the following: Start debugging the program, then select *Windows* from the *Debug* menu. Select *Registers* from the drop-down list. The Registers window may appear at the bottom of the workspace, as a tab highlighted in yellow. Use the mouse to drag the window to the right side of the work area. Right click inside the Registers window and check the item *Flags* to enable the display of CPU status flags.

You can interrupt a debugging session at any time by selecting *Stop Debugging* from the Debug menu. You can do the same by clicking the maroon-colored square button on the toolbar. To remove a breakpoint from the program, click on its red dot to make it disappear.

Setting a BreakPoint

If you set a breakpoint in a program, you can use the debugger to execute the program a full speed (more or less) until it reaches the breakpoint. At that point, the debugger drops into single-step mode.

1. In our sample program, click the mouse along the border to the left of the **mov eax,5** statement. A large red dot should appear in the margin.
2. Select *Start Debugging* from the Debug menu. The program should run, and pause on the line with the breakpoint, showing the same Yellow arrow as before.
3. Press F10 until the program finishes.

You can remove a breakpoint by clicking its red dot with the mouse. Take a few minutes to experiment with the Debug menu commands. Set more breakpoints and run the program again. For the time being, you can use the F11 key to step through the program in the same way the F10 key did.

Return to top

Using the Microsoft Visual Studio 2013 Debugger

Adapted from Kip Irvine

This tutorial explains how to use the Microsoft Visual Studio 2013 Debugger to debug 32-bit assembly language programs running in Protected mode. Specifically, you will learn how
to perform the following tasks:

- Step through your program, viewing the source code
- Set breakpoints in your code
- View CPU registers and flags
- View a disassembly of your program
- Watch the values of program variables
- View the runtime stack

- Display blocks of memory

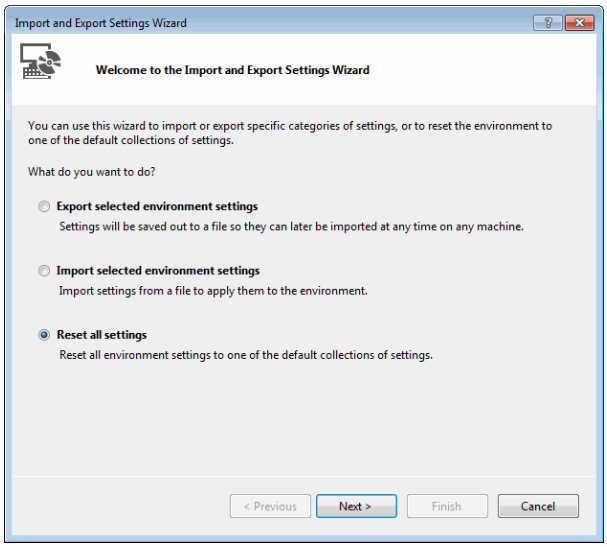
For additional information about debugging in Visual Studio 2013, visit
http://msdn.microsoft.com
and search for *Debugging in Visual Studio*.

Select this link if you're having trouble getting the debugger to recognize debugging information.

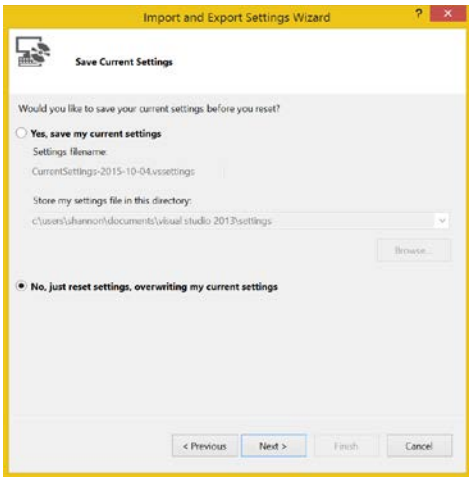
Configure Visual Studio for Debugging

Visual Studio has a set of different configurations that make it easy to create a variety of project
types (Visual Basic, ASP.NET, etc.). In order to get the most out of debugging assembly languagee programs, you need to select an appropriate configuration. From the Tools menu in
Visual Studio, select I*mport and Export Settings*. (In Visual Studio Express, select *Settings* from
the Tools menu, then select *Import and Export Settings*.)

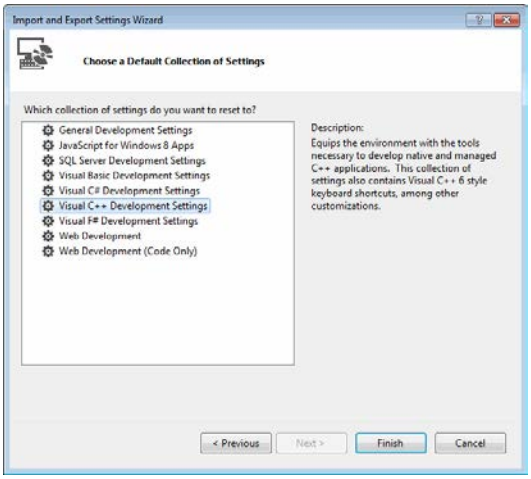
In the following dialog, select *Reset all settings* and click the *Next* button.



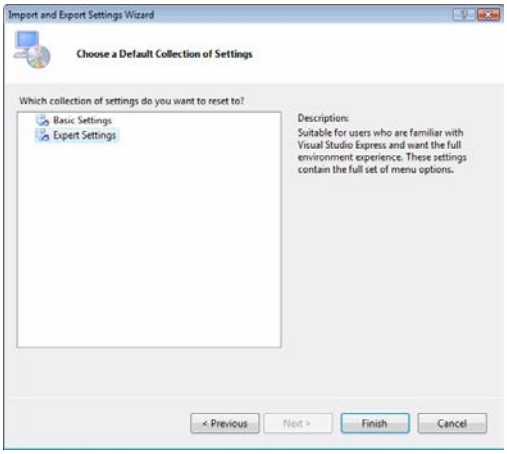
In the next pane, you have the option of saving your current settings. This would be useful if you want to return to your current settings later. Select an option and click the *Next* button.



In the following pane (from Visual Studio Professional), select the Visual C++ settings and click the *Finish* button.



In Visual Studio Express you will see the following two options. Select *Expert Settings* and click the *Finish* button:



Once your settings have been reset, click the *Close* button to close the wizard.

Note: In Visual Studio Express, you can switch between Basic Settings and Expert Settings by
selecting *Settings* from the Tools menu. The Expert Settings option in Visual Studio Express is
required if you want to display Registers, Memory, and Disassembly windows.

Once you begin working with the debugger, you can fine-tune the arrangement of the Visual Studio menus and windows, and save your specific settings for later recall.

Open a Visual Studio Project file

In this step-by-step tutorial, you will open the sample project and insert an example program
named AddTwo.asm.

1. Go to the **Lab4\Project32** folder in the disk directory that contains the lab.
2. Double-click the file named **Project.sln**.
3. In The Solution Explorer window, right-click any existing files with an extension of .asm
   and select **Remove**. When prompted by a dialog window, click the **Remove** button.
4. Right-click Project in the same window, select **Add**, and select **Existing Item**.
5. Browse back one level to the **Lab4** folder, and select the file named **AddTwo.asm**.
   It will be added to the Solution Explorer window.
6. Select **Save** All from the **File** menu to save your project settings.
7. Double-click the AddTwo.asm filename in Solution Explorer to open it in the editor:

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

.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
```

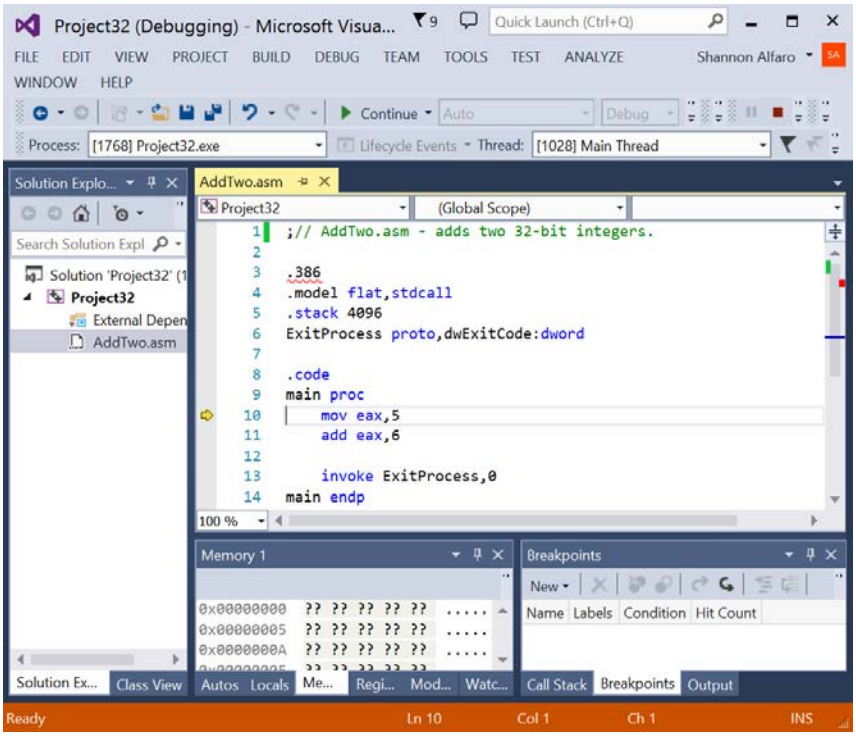
Build the project by selecting **Build Solution** from the **Build** menu. In the output window, you should see text similar to that shown below. Extra lines may be added:

```
1>------ Build started: Project: Project32, Configuration: Debug
Win32 ------
1>  Assembling ..\..\..\..\51_Fall2015\ICS
51\Labs\Lab4\AddTwo.asm...
1>  Project32.vcxproj -> c:\users\shannon\documents\visual
studio 2013\Projects\Project32\Debug\Project32.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
==========
```

The important thing to notice is the last line, which indicates that the build operation succeeded.

Start the Debugger

Press the **F10 key** to start the debugger. Your environment should appear like this:



Don't worry if some of the windows shown here do not appear on your screen. You can open and
close them at will. First however, let's take a tour:

Source Window

The **Source** window displays the program's source file. Note that the first **MOV** statement has a
yellow arrow next to it. The arrow always indicates the next statement that's about to execute.

Tip: You can remove any debugger window by right-clicking on its Tab and selecting **Hide**. You can open any debugging window by selecting **Windows** from the **Debug** menu. You can make a window collapse when it is not used by clicking the Auto Hide icon in its title bar. The icon looks like a push pin. If the pin points sideways, the window will collapse automatically.

Adjust the Debugging Support Windows

This is a good time to remove panels that you will not use. In the lower part of the debugger
window, select and remove all but the Watch and Output panels. On the side panel, remove all
but the Solution Explorer window. Also, from the Debug menu, select Windows, and select Registers.

You may want to remove some of the extra toolbars from the top of the window. If so, select Toolbars from the View menu, and un-check any toolbars you want to remove. You can also remove individual buttons from a toolbar: Select the dropdown arrow on the right side of the toolbar, and select *Add or Remove Buttons*.

Moving and Undocking Windows

You can move any window by dragging its titlebar with the mouse. You can dock it (position it) by dropping it on any side of your workspace. For example, I like to put the Registers and Watch windows on the right side of the work area. If, however, you have more than one window positioned directly on top of each other, dragging the title bar will move all windows at once. If you want to select only one window from a cluster of windows, drag its

tab (on the bottom) to the new location. The other windows in the cluster will not be affected.

You can undock a window by dragging it with the mouse to any location on your desktop. It can be completely outside the Visual Studio window.

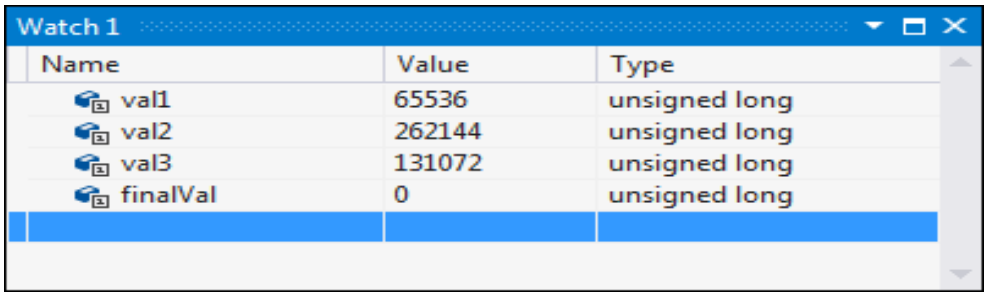
Watch Window

For this part of the tutorial, you will need use the AddSubTwo.asm program in the **Lab4** folder. Please add it to your Solution Explorer window, and remove the AddTwo.asm program. Here is a partial program listing:

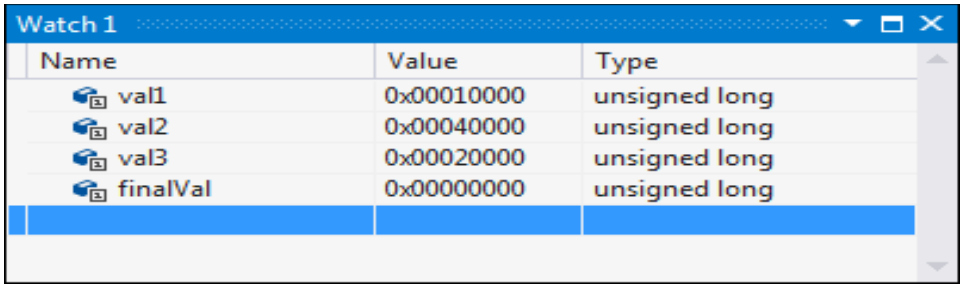
```
; This program adds and subtracts 32-bit integers
; and stores the sum in a variable.

.data
val1     dword  10000h
val2     dword  40000h
val3     dword  20000h
finalVal dword  ?

.code
main PROC
        mov     eax,val1              ; start with 10000h
        add     eax,val2              ; add 40000h
        sub     eax,val3              ; subtract 20000h
        mov     finalVal,eax          ; store the result (30000h)

        invoke  ExitProcess
main ENDP
‘/;
END main
```

If at some later time the watch window disappears, you can get it back again. While currently debugging a program, select *Windows* from the Debug menu, and select *Watch 1.* A Watch window is like an electronic spreadsheet that displays the names and values of selected variables. As you step through a program, you can see variables in this window change value. Currently the window is empty, but you can drag any program variable into the window with your mouse.

Next, you will display the value of some variables in the Watch window. Drag the **val1**, **val2**, **val3,** and **finalVal** variables into the Watch window and note their current values:



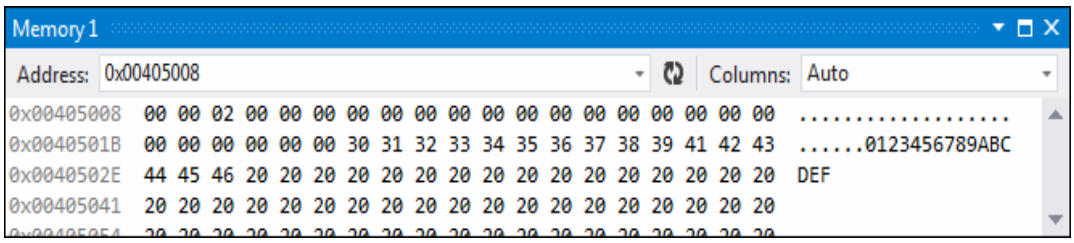
The values are currently displayed in decimal, so select hexadecimal format by right-clicking on the watch window and selecting **Hexadecimal Display** from the popup menu. Following is a sample:



Memory Window

From the Debug menu, welect *Windows,* select *Memory,* and select *Memory 1*. The Memory1 window displays a raw dump of memory in either hexadecimal or decimal. It is particularly useful when working with array variables. Since we don't have any arrays in the program, let's display the value of **val3**. Next to the Address label, type: **&val3**

The & before the variable name means to interpret the variable name as an address. In assembly language, all labels are implied addresses. Variable names are case-sensitive in the debugger. The Memory window displays a series of individual memory bytes, beginning at the address of **val3**. Right-click on the window, and select *4-byte Integer* as the display format. Along the left side of the window is shown the address of the first value in each line:

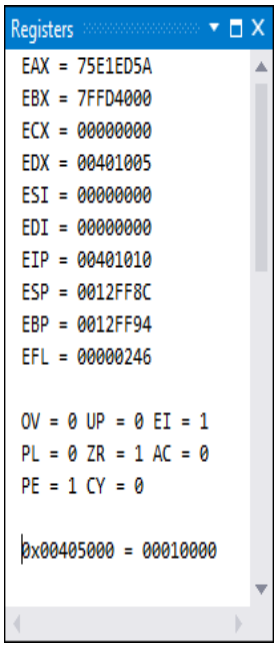


In our sample, **val3** is located at address 00405008, although your addresses might be different. You can also type a hexadecmal constant into the Address field. Use the C-language format, where the digits are preceded by "0x". For example: 0x0040400C.

Note: The memory Windows are only available if address-level debugging is enabled in the Tools / Options / Debugging settings, available by clicking on the Tools menu in Visual Studio, and then selecting Options.

Register Window

You can display the Register window by selecting *Windows* from the Debug menu, and selecting *Register*. The Register window displays the contents of the CPU registers. The flag values do not show by default, but you can add them in by right-clicking and selecting *Flags:*

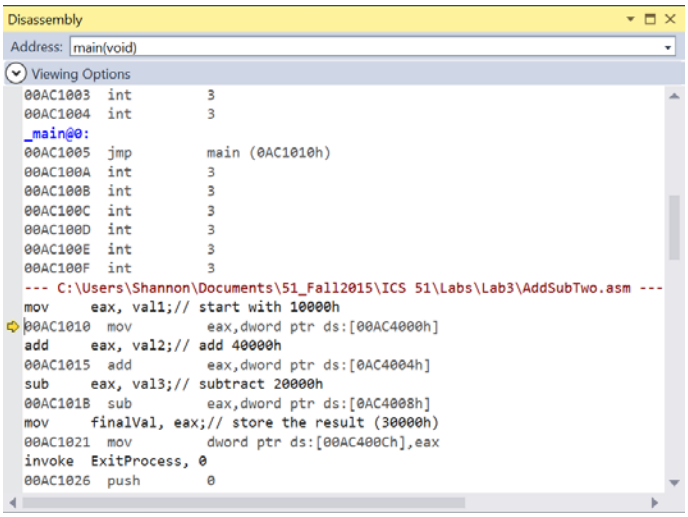


Key to flag abbreviations

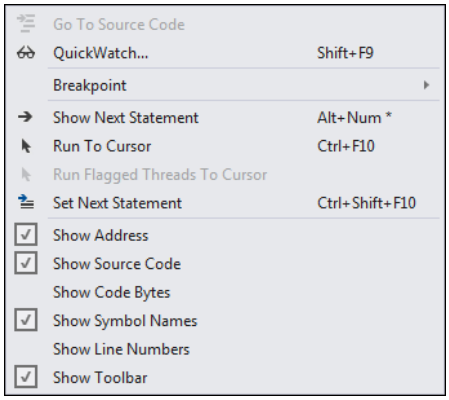
Flag Name	Abbreviation
Overflow	OV
Direction	UP
Interrupt	EI
Sign	PL
Zero	ZR
Aux Carry	AC
Parity	PE
Carry	CY

Disassembly Window

Select *Windows* from the Debug menu, and select *Disassembly*. The Disassembly window displays a raw listing of the program's code. In some programs, it will show code generated by macros or by high-level MASM directives (such as .IF).



If you right-click inside the Disassembly window, the popup context menu offers different viewing options:



Here is a sample in which *Show Source Code* is disabled, and both *Show address* and *Show symbol names* options are enabled:

```
--- C:\Users\Shannon\Documents\51_Fall2015\ICS 51\Labs\Lab4\AddSubTwo.asm
------
_main@0:
00AC1010  mov         eax,dword ptr ds:[00AC4000h]
00AC1015  add         eax,dword ptr ds:[0AC4004h]
00AC101B  sub         eax,dword ptr ds:[0AC4008h]
00AC1021  mov         dword ptr ds:[00AC400Ch],eax
00AC1026  push        0

00AC1028  call        _ExitProcess@4 (0AC1034h)
```

The offset of each variable apears next to its name.

Stepping Through the Program's Execution

Next, you will begin stepping through the program's execution, one line at a time. Press the F10 several times and watch the yellow arrow in the Source window move from statement to statement. Notice the following as you step through the program:

- Individual register names (in the *Register* window) turn red, indicating that they have been modified. The AddSub2 program modifies the EAX and EIP registers quite a bit, as you can see.
- Variables in the Watch window turn red when they are modified.
- Memory locations in the Memory window turn red when they are modified.

When you reach the **exit** statement and press F10, the debugger halts and you return to editing mode.

Step Into Command (F11)

Another way to step through a program is to use the **Step Into** (F11) command. It steps down into procedure calls. In contrast, the F10 key just executes procedure calls without tracing into the procedure code.

Stopping and Restarting

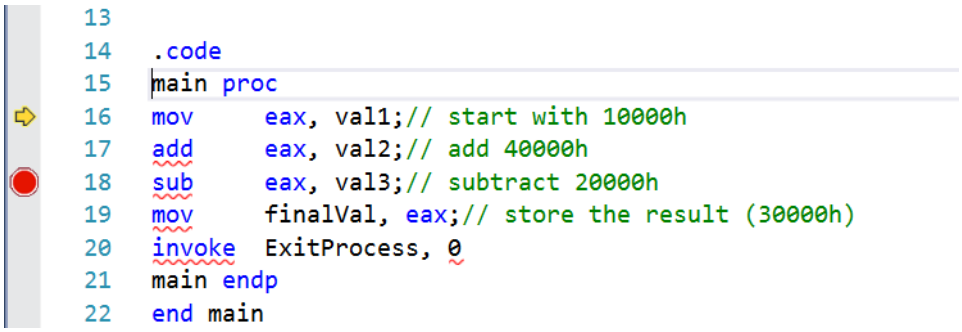
It's easy to either stop or restart your program inside the debugger while you're in the process of stepping through a program:

- To restart the program, select *Restart* from the Debug menu. The program is reloaded into memory, so any changes previously made to variables in the Watch window are undone. Also, you have to retype the name of your variable in the Memory window, because it resets itself to a default address.
- To stop debugging in the middle of a program, select *Stop Debugging* from the Debug menu. You can also use the icon containing a small square on the Debug toolbar.

Toggle Breakpoint (F9)

You can set a breakpoint on any executable line in your source program by either clicking the mouse in the left margin, or pressing the F9 key. This can be done before you start the debugger, or while you are stepping through the program. If you press F9 when the cursor is on a line containing a breakpoint, the breakpoint is removed. The following image shows

code containing a breakpoint:



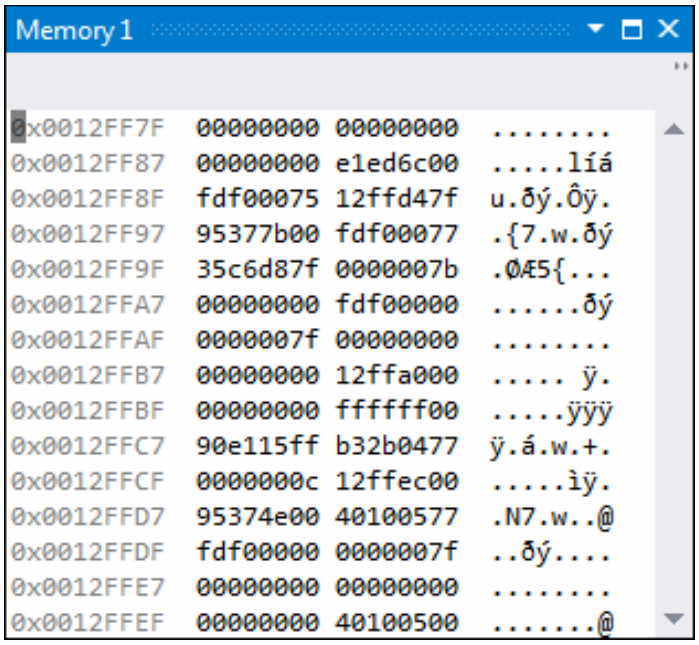
When you set a breakpoint, you can run your program at full speed up to the line containing the first breakpoint. The debugger will pause on the line, and wait for you to either step through the code from that point, or run the program to the next breakpoint.

- Use the F5 key (Menu: Debug / Run) to start a program and run to the first breakpoint.
- When at a breakpoint, use the F5 key (Menu: Debug/Continue) to continue execution to the next breakpoint.

Tip: Be sure to stop the debugger before trying to modify and re-assemble your program's source code. Otherwise, the linker will refuse to assemble your EXE file, indicating that it's currently in use

Viewing the Runtime Stack

The runtime stack will become more important as you begin to write programs that contain procedures other than main. Each time you call a procedure, you place the program's return address on the stack. To view the runtime stack while debugging, display a memory window at the address specified in the ESP register. In the following image, the stack shows a return address (00401025) stored at the memory location point to by the ESP (stack pointer) register:

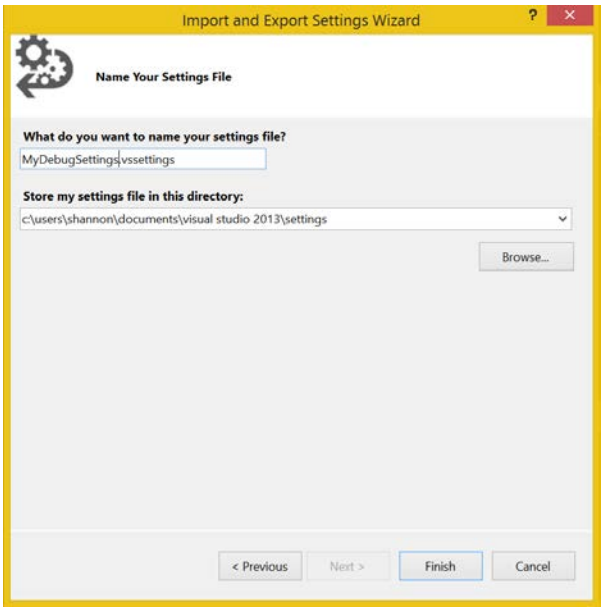


When you display the stack, right-click the memory window and select *4-byte Integer* as the display format.

Saving Your Visual Studio Setup

After configuring Visual Studio to your debugging and editing preferences, you would be wise to save your setup. Here are the steps:

1. From the Tools menu, select *Import and Export Settings*.
2. In the next wizard step, select *Export selected environment settings*.
3. In the next step, you select which settings to save. Leave the default selections unchanged.
4. In the last panel, select a name that you're likely to remember for the settings file and click the *Finish* button. Here is a sample of this step (change 2012 to 2013 in your copy):



If you ever want to return Visual Studio to the settings you've saved, just run the Import and Export Settings Wizard again.
