Getting Started with MASM and Visual Studio 2019

Updated 6/28/2019

This tutorial assumes that you are using either the 7th or 8th Edition of *Assembly Language for x86 Processors*, and you are using Visual Studio 2019. We tested these pages with the Community Edition, and have every reason to assume the Enterprise Edition will work the same way.

Here's how to get started:
Right-click here to download the code examples and required libraries for the book. Unzip the downloaded file into a directory named Irvine on Drive C. Next, Right-click here to download a zip file containing a 32-bit Visual Studio 2019 project. You can extract this file into any folder on your computer. Finally, Right-click here to download a zip file containing a 64-bit Visual Studio 2019 project. You can extract this file into any folder on your computer. Now you are ready to begin the tutorials listed below.

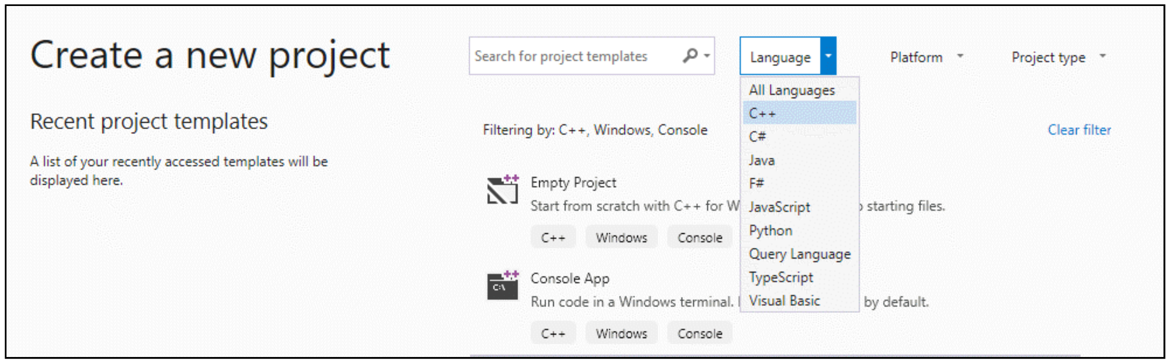
Topics:

- Tutorial: Building and running a 32-bit program
- Tutorial: Building and running a 64-bit program
- Building 16-bit programs
- Syntax highlighting in your source code
- Using the Visual Studio debugger
- EXE Programs blocked by antivirus software

Found an error in this document? Please email the author.

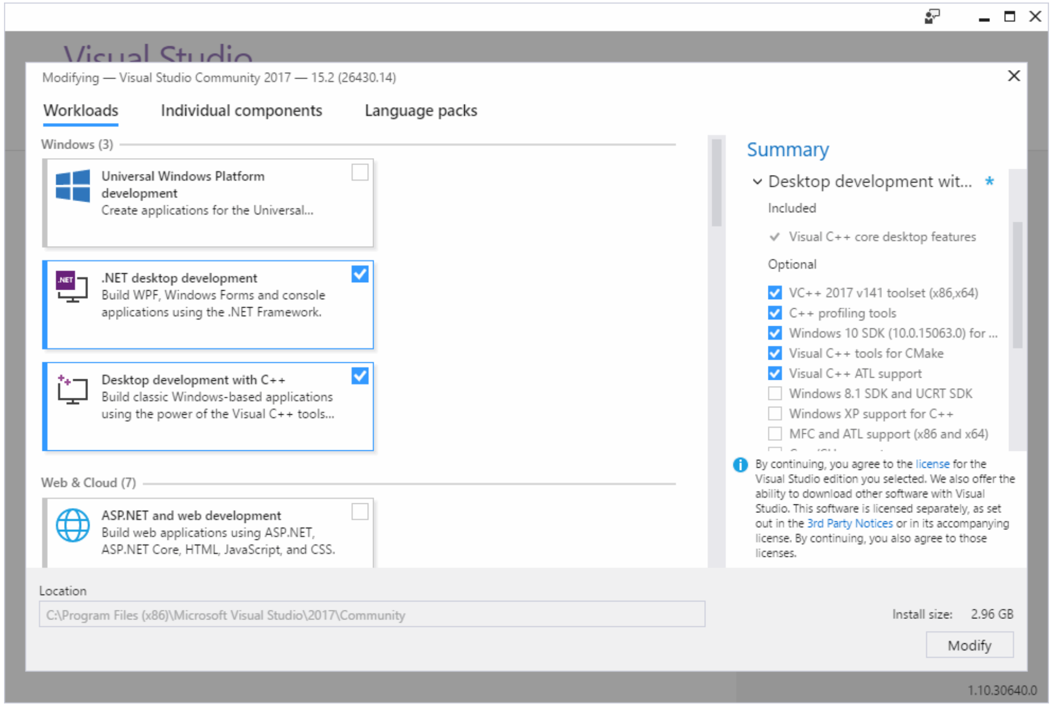
Required Setup for 32-bit Applications

After you have downloaded and installed the VS 2019 Community Edition, you may need to install the Visual C++ language option. First, let's see if it has already been installed (as often happens in college computer labs). Select File >> New >> Project from the Visual Studio menu. You will see this *Create a new project* dialog window. Look for C++ in the Language dropdown list:



Note: If you do not see Visual C++ in the list, close Visual Studio and run a separate program named the Visual Studio Installer. If your computer is in a college laboratory, your account may not have sufficient privileges to run this program, so you can ask your lab supervisor to do this.

(If you run the VS installer, select the *Desktop development with C++* button in the installer window, look at the Summary list on the right side to verify that VC++ is selected, and click the Modify button in the lower right corner of the window.)



Modifying — Visual Studio Community 2017 — 15.2 (26430.14)

Windows (3)

- [ ] Universal Windows Platform development — Create applications for the Universal...
- [✓] .NET desktop development — Build WPF, Windows Forms and console applications using the .NET Framework.
- [✓] Desktop development with C++ — Build classic Windows-based applications using the power of the Visual C++ tools...

Web & Cloud (7)

- [ ] ASP.NET and web development — Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and CSS.

Summary

Desktop development wit... *

Included

- ✓ Visual C++ core desktop features

Optional

- [✓] VC++ 2017 v141 toolset (x86,x64)
- [✓] C++ profiling tools
- [✓] Windows 10 SDK (10.0.15063.0) for ...
- [✓] Visual C++ tools for CMake
- [✓] Visual C++ ATL support
- [ ] Windows 8.1 SDK and UCRT SDK
- [ ] Windows XP support for C++
- [ ] MFC and ATL support (x86 and x64)

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Location: C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

Install size: 2.96 GB

The Visual C++ language includes the Microsoft Assembler (MASM). To verify that MASM is installed, open a Windows Explorer window and look for the file named **ml.exe** in the Visual Studio installation directory, such as *C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.xx.xxxx\bin\HostX64\x86*. (The "x" characters above indicate digits in the version number of your current VS installation.)

The Book's Example Programs

At the top of this document, we explained how to download the file named Irvine.zip and extract it into the C:\Irvine folder. Unless you have some objection to using that location, do not alter the path. (Note to lab administrators: you can designate c:\Irvine directory as read-only.).

The folllowing files should appear in the c:\Irvine directory:

Filename	**Description**
b16.asm, b32.asm	Blank templates for 16-bit and 32-bit assembly language source files

GraphWin.inc	Include file for writing Windows applications
Irvine16.inc	Include file used with the Irvine16 link library (16-bit applications)
Irvine16.lib	16-bit link function library used with this book
Irvine32.inc	Include file used with the Irvine32 link library (32-bit applications)
Irvine32.lib	Irvine's 32-bit link library
Kernel32.lib	32-bit link library for Windows API
Link16.exe	16-bit Microsoft linker
Macros.inc	Irvine's macro include file (see Chapter 10)
make16_vs2019.bat	Visual Studio 2019 batch file for building 16-bit applications
SmallWin.inc	Small-sized include file containing MS-Windows definitions, used by Irvine32.inc
User32.lib	MS-Windows basic I/O link library
VirtualKeys.inc	Keyboard code definitions file, used by Irvine32.inc

A subdirectory named **Examples** will contain all the example programs shown in the book, source code for the book's 16-, 32-, and 64-bit libraries, and two sample projects for earlier versions of Visual Studio.

Setting up Visual Studio

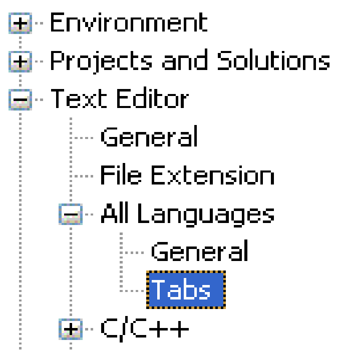
Select the C++ Configuration

Visual Studio supports 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. (Optionally, you can use the mouse to drag the Solution Explorer tool window to the right side of the workspace.) If you should accidentally close the Solution Explorer window in the future, you can bring it back: select View from the menu, and locate Solution Explorer in the list of views.

Optional Step: Set the tab indent size

Start Visual Studio and select **Options** from the **Tools** menu. Select and expand the **Text Editor** item, select **All Languages**, and select **Tabs**. Optionally, you may want to select the **Insert spaces** radio button:



I prefer to set the Tab Size and Indent Size values to 5.

Tutorial: Building and running a 32-bit 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.

If you have not already done so,Right-click here to download a zip file containing an up-to-date Visual Studio 2019 project that has been configured for assembly language. After downloading this file, un-zip it into your working directory. It contains a sample asm test file named AddTwo.asm.

Follow these steps:

1. Start Visual Studio.
2. Open our sample Visual Studio project file by selecting **File/Open/Project** from the Visual Studio menu.
3. Navigate to your working folder where you unzipped our project file, and select the file named **Project.sln**.
4. Once the project has been opened, you will see the project name in Visual Studio's *Solution Explorer* window. You should also see an assembly language source file in the project named AddTwo.asm. Double-click the file name to open it in the editor.

You should see the following program in the editor window:

```
; 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
```

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.

Adding a File to a Project: If you need to add an .asm file to an open project, do the following: (1) Right-click the project name in the Visual Studio window, select Add, select Existing Item. (2) In the *Add Existing Item* dialog window, browse to the location of the file you want to add, select the filename, and click the Add button to close the dialog 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: Project, Configuration: Debug Win32 ------
1>Assembling AddTwo.asm...
1>Project.vcxproj -> ...\Project32_VS2019\Debug\Project.exe
========== Build: 1 succeeded, 0 failed, 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 in Debug Mode

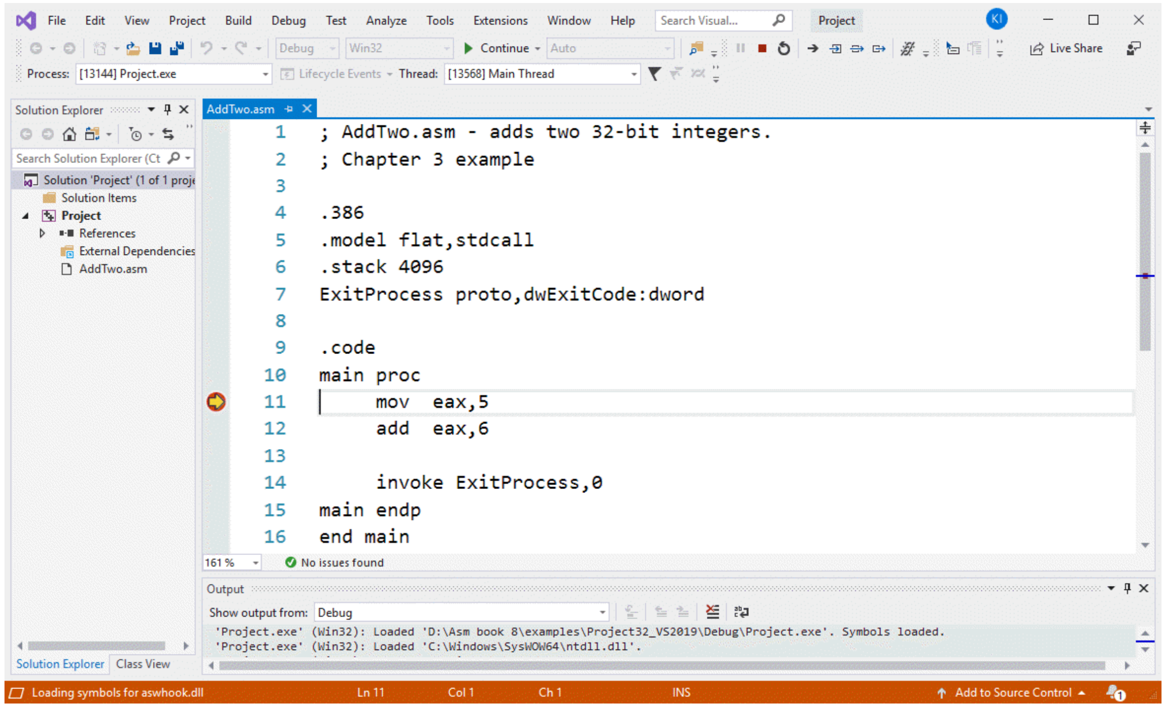
The easiest way to run your first program is to use the debugger. First, you must set a breakpoint. When you set a breakpoint in a program, you can use the debugger to execute the program a full speed (more or less) until it reaches the breakpoint. At that point, the debugger drops into single-step mode. Here's how to do it:

1. Make sure the ASM source code file is open in the editor window.
2. Click the mouse along the border to the left of the **mov eax,5** statement. A large red dot should appear in the margin.

3. Select *Start Debugging* from the Debug menu. The program should run and pause on the line with the breakpoint. (Optionally, you can close the Diagnostic Tools, Autos, and Call Stack windows.)
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 either your Windows desktop or status bar. The window should be blank because this program does not display any output.
6. Press F10 one more time to end the program.

You can remove a breakpoint by clicking its dot with the mouse. Take a few minutes to experiment with the Debug menu commands. Set more breakpoints and run the program again.

Here's what your program will look like when paused at the breakpoint:



```
; 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
```

```
'Project.exe' (Win32): Loaded 'D:\Asm book 8\examples\Project32_VS2019\Debug\Project.exe'. Symbols loaded.
'Project.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ntdll.dll'.
```

Running a program from the Command Prompt: 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 any EXE by double-clicking its name inside Windows Explorer, but it will often 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.

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.

Registers

Soon you will want to display CPU registers when debugging your programs. Here's how to make them visible: First, under the Tools >> Options menu, select Debbuging in the left panel, and select *Enable address-level debugging*. Next, set a breakpoint in your source code on an executable statement, run your program in Debug mode, select *Windows* from the *Debug* menu, and then select *Registers* from the drop-down list.

If you do not see the Registers command in the Debug >> Windows drop-down menu (which seems to be the case for the VS2019 Community Edition, there is a way to add a Registers command button to your Debug toolbar. Here's how to do it:

1. While not debugging, select *Customize* from the Tools menu.
2. Click the *Commands* tab, select the Toolbar tab, and select *Debug* from the list of toolbars.
3. Click the *Add Command* button. In the *Categories* list, select *Debug*.
4. Select *Registers* from the list of commands, click the OK button to close the dialog window.
5. Click the *Close* button to close the Customize dialog. You should now see a new button on the Debug toolbar that looks like a small rectangle containing "0X" when you begin debugging a program.

The Registers window may appear docked to the top of the workspace, but you may find it helpful to float the window on top of your workspace. Just grab the window header with the mouse and pull it to the center area. You will also want to display the CPU flags. To do that, right click inside the Registers window and check the word *Flags* from the popup menu.

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 a program, click its red dot to make it disappear.

A reminder, you might want to review our tutorial: Using the Visual Studio debugger

Building and Running Other Programs

Suppose you want to run another example program, or possibly create your own program. You can remove the existing assembly language file from the Solution Explorer window and insert a new .asm file into the project.

- To remove a program from a project without deleting the file, right-click its name in the *Solution Explorer window*. In the context menu, select **Remove**. If you change your mind and decide to add it back to the project, right-click in the same window, select **Add,** select **Existing item,** and select the file you want to add.

Adding a File to a Project

An easy way to add an assembly language source file to an open project is to drag its filename with the mouse from a Windows Explorer window onto the name of your project in the Solution Explorer window. The physical file will not be copied--the project only holds a reference to the file's location. Try this now:

1. Remove the AddTwo.asm file from your project.
2. Add a reference to the file Examples\ch03\AddTwoSum.asm to the project.
3. Build and run the project.

Copying a Source File

One way to make a copy of an existing source code file is to use Windows Explorer to copy the file into your project directory. Then, right-click the project name in Solution Explorer, select Add, select Existing Item, and select the filename.

Return to top

Tutorial: Building and running a 64-bit program

In this tutorial, we will show you how to assemble, link, and run a sample 64-bit program. We assume you have already completed our tutorial entitled *Building a 32-Bit Assembly Language Program*.

Do the following steps, in order:

1. Right-click here to download the Project64_VS2019.zip file and unzip it into your working directory.
2. In Visual Studio 2019, select Open Project from the File menu, navigate to the Project64_VS2019 folder, and select the file named **Project.sln**.
3. You are about to add an existing source code file to the project. To do that, right-click on **Project** in the Solution Explorer window, select **Add**, select **Existing Item**, navigate to the book's Examples\ch03\64 bit" folder, select **AddTwoSum_64.asm**, and click the **Add** button to close the dialog window.
4. Open the AddTwoSum_64.asm file for editing by double-clicking its filename in the Solution Explorer window.

You should see the following program in the editor window:

```
; AddTwoSum_64.asm - Chapter 3 example.

ExitProcess proto

.data
sum qword 0

.code
main proc
```

```
   mov  rax,5
   add  rax,6
   mov  sum,rax

   mov  ecx,0
   call ExitProcess

main endp
end
```

(Notice that the program's entry point is the main procedure. If you wish to use a different name for your startup procedure in your own programs, you can modify this option by selecting Properties from the Project menu, and then selecting Linker / Advanced / Entry Point.)

Build the Program

Select **Build Project** from the Build menu. You should see text written to Visual Studio's output window like the following:

```
1>------ Build started: Project: Project, Configuration: Debug x64 ------
1>  Assembling AddTwoSum_64.asm...
1>  Project64_VS2019.vcxproj -> ...\Project64_VS2019\x64\Debug\Project.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.

You use the same Visual Studio commands to run and debug 64-bit programs as you would for 32-bit programs.

Return to top

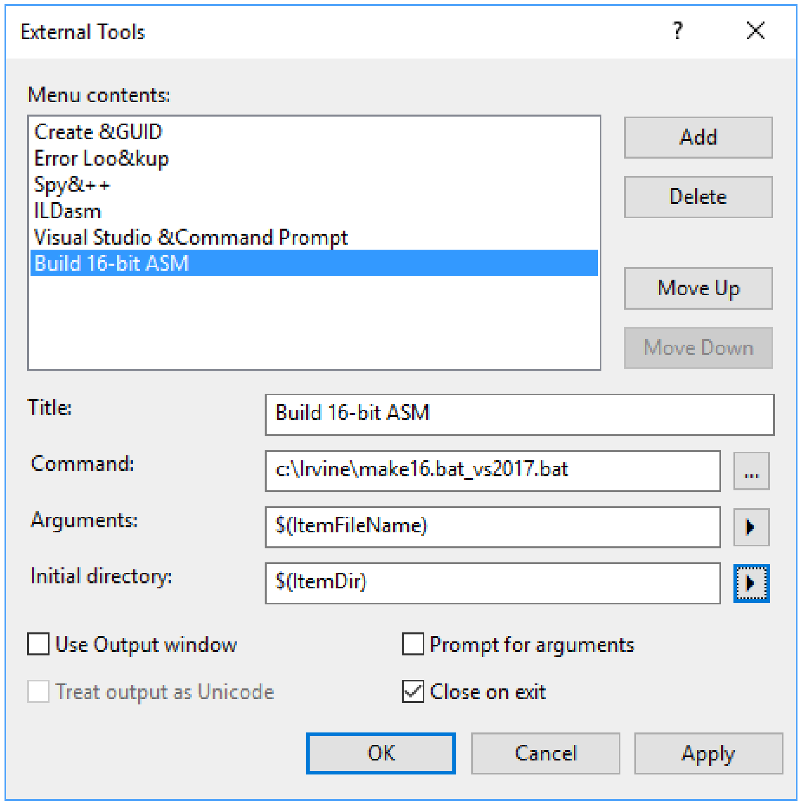
Building 16-bit programs (Chapters 14-17)

Only Chapters 14 through 17 require you to build 16-bit applications. Except for a few exceptions, which are noted in the book, your 16-bit applications will run under the 32-bit versions of Windows (Windows XP, Windows Vista, Windows 7).

If you're interested in running 16-bit programs under 64-bit Windows, you will need to enable a feature named NTVDM.) Click here to read a web site with instructions on how to do this. Another alternative you may wish to explore is to install a virtual machine (using a free program named VirtualBox from Oracle) and install 32-bit Windows on the virtual machine.

The book's example programs in Chapters 1-13 have been successfully tested in 32-bit Windows 7,8, and 10. On the other hand, many programs in Chapters 14-17 will not run in any Microsoft OS later than Windows 98, because they rely on direct access to hardware and system memory. You cannot directly run 16-bit applications in any 64-bit version of Windows.

If you plan to build 16-bit applications, you need to add two new commands to the Visual Studio Tools menu. To add a command, select **External Tools** from the Tools menu. The following dialog will appear, although many of the items in your list on the left side will be missing. Download the batch file here (rename the .txt extension to .bat after downloading): make16_vs2019.txt.



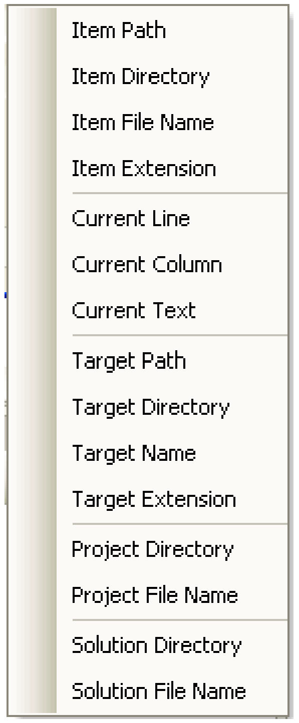
Menu contents:
Create &GUID
Error Loo&kup
Spy&++
ILDasm
Visual Studio &Command Prompt
Build 16-bit ASM

Title: Build 16-bit ASM
Command: c:\Irvine\make16.bat_vs2017.bat
Arguments: $(ItemFileName)
Initial directory: $(ItemDir)

☐ Use Output window
☐ Prompt for arguments
☐ Treat output as Unicode
☑ Close on exit

Step 1: Create the Build 16-bit ASM Command

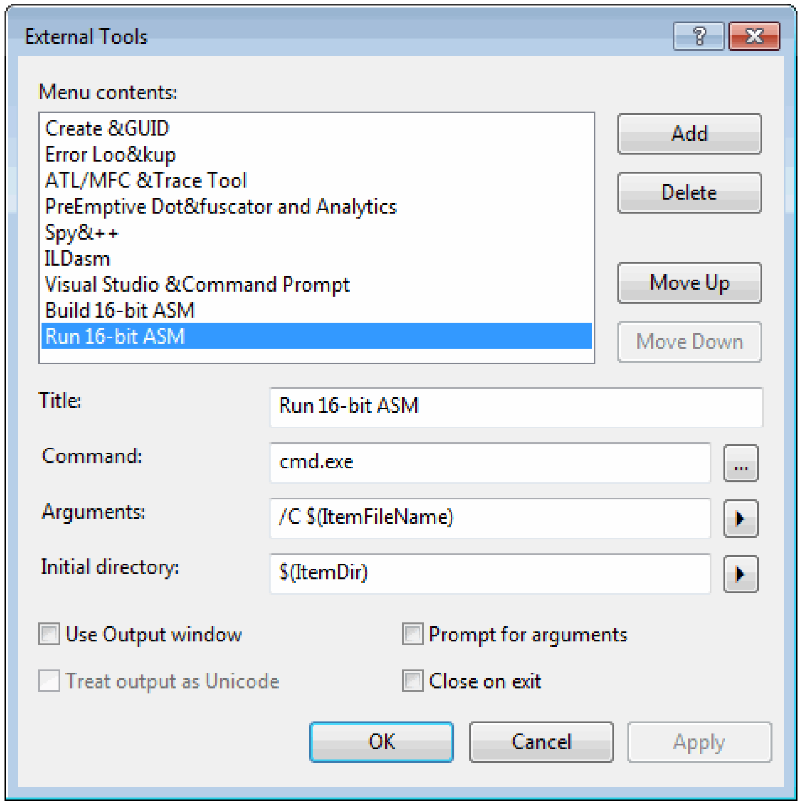
Click the **Add** button and fill in the Title, Command, Arguments, and Initial directory fields as shown in the screen snapshot. If you click the buttons with arrows on the right side of the Arguments and Initial directory fields, a convenient list appears. You can select an item without having to worry about spelling:



Click the **Apply** button to save the command.

Step 2: Create the Run 16-bit ASM Command

Click the Add button again, and create a new command named **Run 16-bit ASM**:



Create &GUID
Error Loo&kup
ATL/MFC &Trace Tool
PreEmptive Dot&fuscator and Analytics
Spy&++
ILDasm
Visual Studio &Command Prompt
Build 16-bit ASM
Run 16-bit ASM

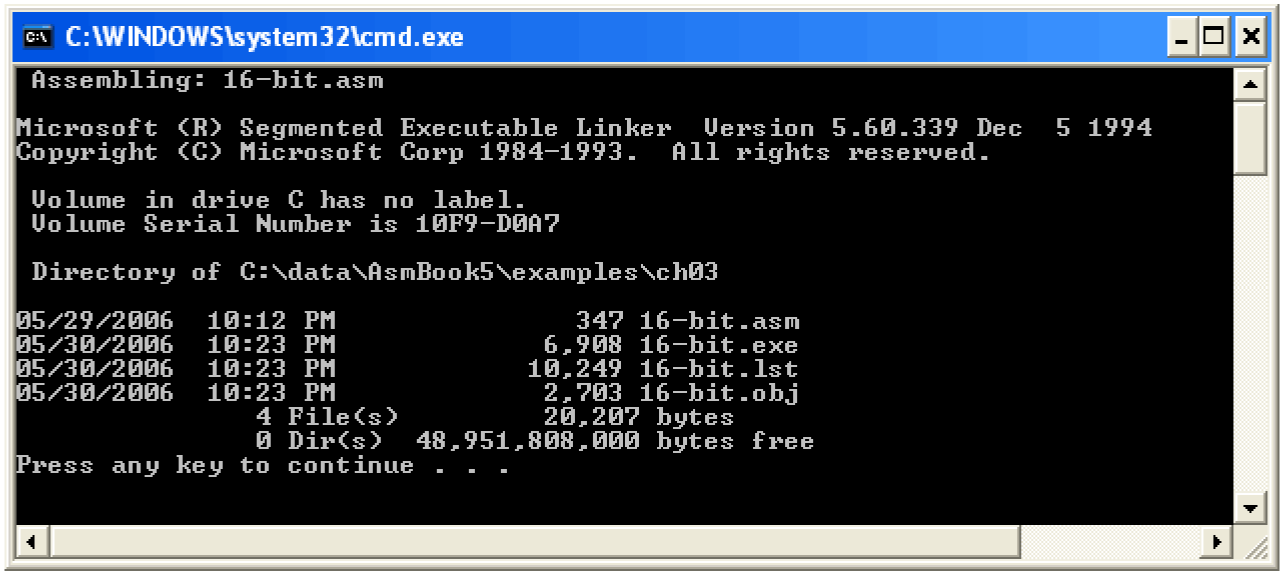
Title: Run 16-bit ASM
Command: cmd.exe
Arguments: /C $(ItemFileName)
Initial directory: $(ItemDir)

Use Output window
Prompt for arguments
Treat output as Unicode
Close on exit

Uncheck the "Close on exit" option and click the OK button to save the command and close the External Tools dialog.

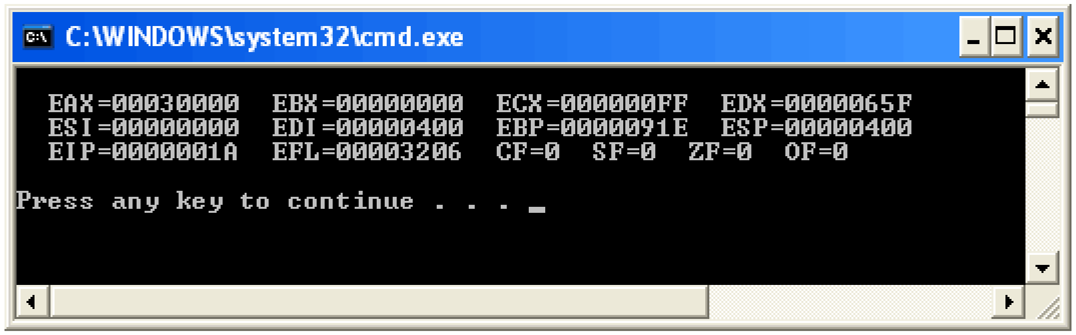
Testing Your new 16-Bit Commands

To test your new 16-bit commands, close any Visual Studio project that happens to be open. Then, select File | Open | File from the menu and choose the file named **16-bit.asm** from the ch03 folder in the book's example programs. Select **Build 16-bit ASM** from the Tools menu. The following command window should appear, showing the successful execution of the assembler and linker, followed by a listing of all files related to this program:



```
 Assembling: 16-bit.asm

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

 Volume in drive C has no label.
 Volume Serial Number is 10F9-D0A7

 Directory of C:\data\AsmBook5\examples\ch03

05/29/2006  10:12 PM                347 16-bit.asm
05/30/2006  10:23 PM              6,908 16-bit.exe
05/30/2006  10:23 PM             10,249 16-bit.lst
05/30/2006  10:23 PM              2,703 16-bit.obj
               4 File(s)         20,207 bytes
               0 Dir(s)  48,951,808,000 bytes free
Press any key to continue . . .
```

Press a key to close the window. Next, you will run the program. Select **Run 16-bit ASM** from the Tools menu. The following window will appear, although the contents of all registers except EAX will be different:



```
  EAX=00030000  EBX=00000000  ECX=000000FF  EDX=0000065F
  ESI=00000000  EDI=00000400  EBP=0000091E  ESP=00000400
  EIP=0000001A  EFL=00003206  CF=0  SF=0  ZF=0  OF=0

Press any key to continue . . . _
```

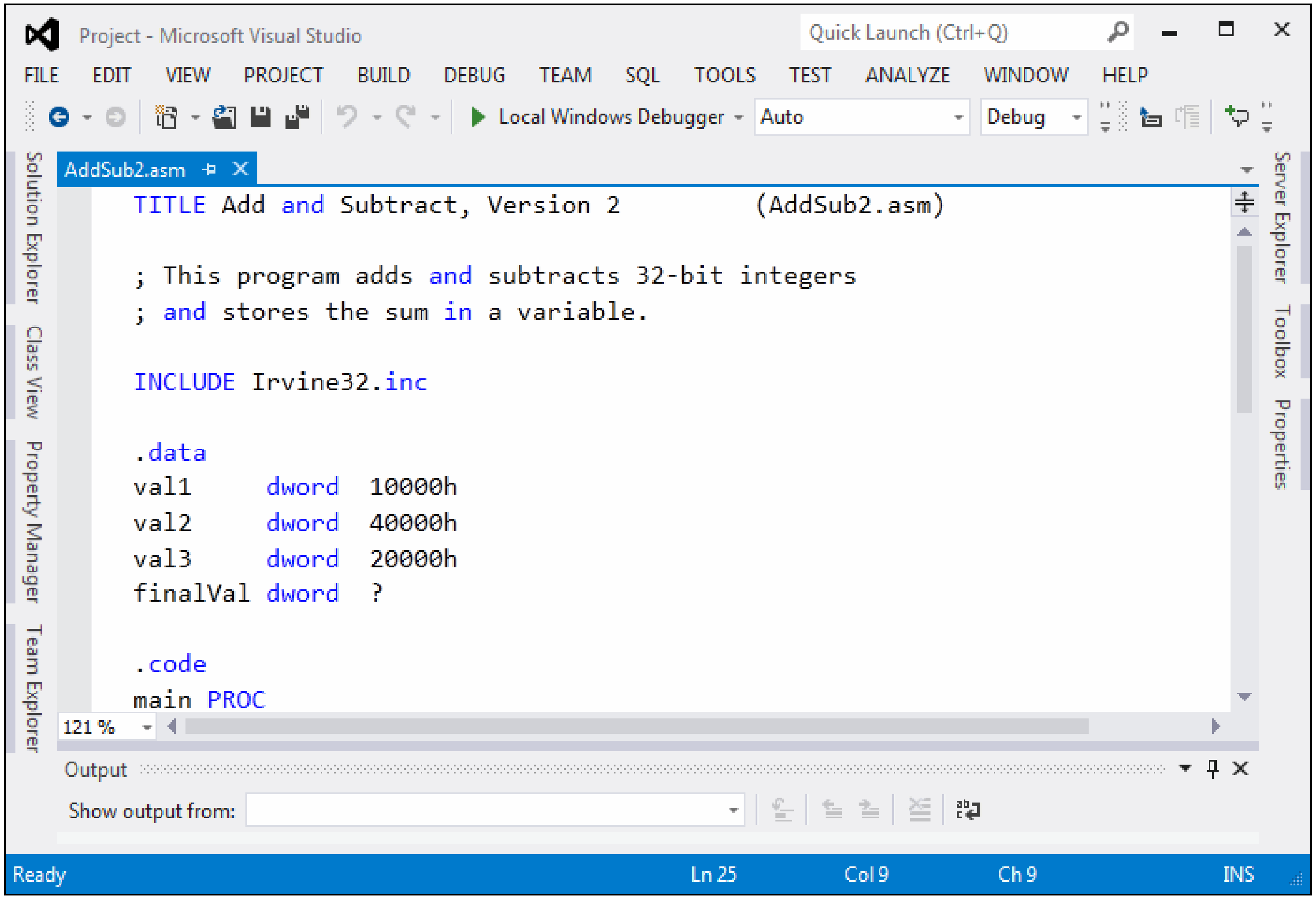
Press a key to close the window.

You have completed the setup for building and running 16-bit assembly language programs.

Return to top

Syntax highlighting in your source code

When a text editor uses syntax highlighting, language keywords, strings, and other elements appear in different colors. Visual Studio highlights MASM reserved words and strings, as shown in the following example:



```
TITLE Add and Subtract, Version 2         (AddSub2.asm)

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

INCLUDE Irvine32.inc

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

.code
main PROC
```

This won't happen automatically, but you can create a syntax definition file named Usertype.dat that contains MASM keywords. Then when Visual Studio starts, it reads the syntax file and highlights MASM keywords.

If you decide to use Visual Studio's built-in MASM syntax highlighter, here are the required steps to set it up:

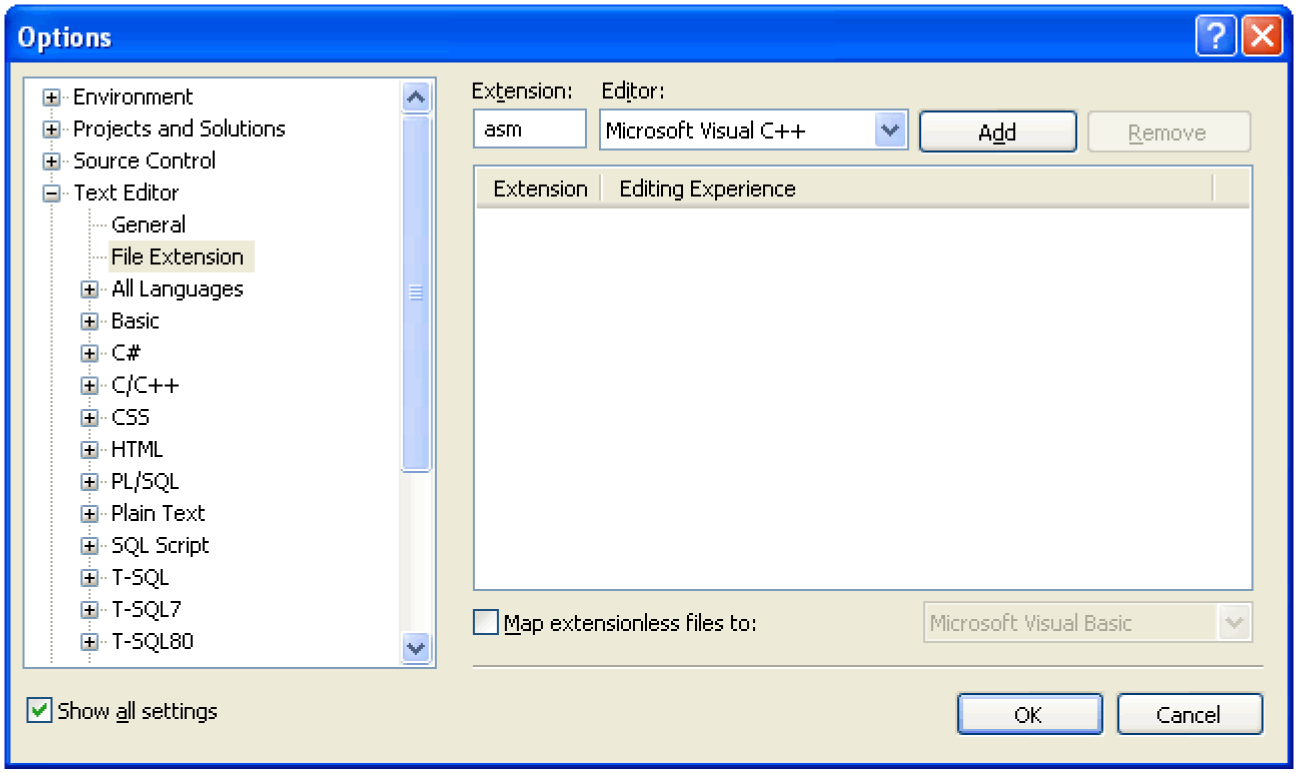
1) Download this Usertype.dat file (enclosed in a ZIP file) given here to a folder in which you have read/write permissions. Extract it from the zip archive.

2) Close Visual Studio.

3) Copy Usertype.dat to the C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE folder.

Windows will display a confirmation dialog before copying the file.

4) Open Visual Studio, select **Options** from the Tools menu, select **Text Editor**, and select **File Extension**. On the right side of the dialog (shown below), enter **asm** as the extension, select **Microsoft Visual C++** from the Editor list, and click the **Add** button. Click the **OK** button to save your changes.



Extension: asm
Editor: Microsoft Visual C++
Map extensionless files to: Microsoft Visual Basic
[✓] Show all settings

Open your project and display an ASM file. You should see syntax highlighting in the editor. There is a glitch in the highlighting--assembly language comment line starts start with a semicolon, which C++ doesn't recognize. But this is a simple workaround: add an extra // right after the semicolon, like this, which will cause the comments to appear in their usual green color:

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

Return to top

Was your program's EXE file blocked by an Antivirus scanner?

Antivirus scanner software has improved greatly in recent years, and so have the number of viruses (one website reports 50,000 at present). Because of this, your computer's antivirus scanner may report a false positive when you build your program, and refuse to let you run it. There are a few workarounds: (1) You can add your project's bin/debug folder into an exclusion list in your antivirus configuration. This is my preferred approach, and it nearly always works. (2) You can suspend your realtime antivirus scanner software, but this will leave you open to malware for a short time. If you choose this option, be sure to temporarily disconnect your computer from the Internet.

Return to top