UNIT 3

PROGRAMMING CONCEPTS
Introduction

- This chapter develops the idea of mixing C/C++ and assembly language.
- This text uses Microsoft Visual C/C++ Express, but programs can often be adapted to any version of C/C++, as long as it is standard ANSI (American National Standards Institute) format C/C++.
- If you want, you can use C/C++ to enter and execute all the programming applications in this text.
Chapter Objectives

Upon completion of this chapter, you will be able to:

- Use assembly language in _asm blocks within C/C+.
- Learn the rules that apply to mixed language software development.
- Use common C/C++ data and structures with assembly language.
Chapter Objectives

Upon completion of this chapter, you will be able to:

• Use both the 16-bit (DOS) interface and the 32-bit (Microsoft Windows) interface with assembly language code.

• Use assembly language objects with C/C++ programs.
1. Using Assembly Language With C++ For 16-bit DOS Applications

- Use software to build 16-bit applications when attempting any programs in this section.
  - if you build a 32-bit application and attempt the DOS INT 21H function, the program will crash.

- To build a 16-bit **DOS** application, you will need the legacy 16-bit compiler.

- Programs are generated using Notepad or DOS Edit.
Basic Rules and Simple Programs

- Before assembly language code can be placed in a C/C++ program, some rules must be learned.
- Example 7–1 shows how to place assembly code inside an assembly language block within a short C/C++ program.
- Assembly code in this example is placed in the _asm block.
- Labels are used as illustrated by the label big.
Extremely important to use lowercase characters for any inline assembly code.
  ◦ if you use uppercase, you will find some assembly language commands and registers are reserved or defined words in C/C++

Example 7–1 reads one character from the console keyboard, and then filters it through assembly language so that only the numbers 0 through 9 are sent back to the video display.

It shows how to set up and use some simple programming constructs in C/C++. 
• Note that AX, BX, CX, DX, and ES registers are never used by Microsoft C/C++.
  ◦ these might be considered *scratchpad* registers, available to use with assembly language
• If you wish to use any of the other registers, make sure that you save them with a PUSH before they are used and restore them with a POP afterwards.
  ◦ if you fail to save registers used by a program, the program may not function correctly and can crash the computer
What Cannot Be Used from MASM Inside an _asm Block

- MASM contains features, such as conditional commands (.IF, .WHILE, .REPEAT, etc.)
- The inline assembler does not include the conditional commands from MASM.
  - nor does it include the MACRO feature found in the assembler
- Data allocation with the inline assembler is handled by C.
Using Character Strings

• Example 7–3 illustrates a simple program that uses a character string defined with C and displays it so that each word is listed on a separate line.
  ◦ notice the blend of both C statements and assembly language statements

• The WHILE statement repeats the assembly language commands until the null (00H) is discovered at the end of the character string.
For each space, the program displays a carriage return/line feed combination.
This causes each word in the string to be displayed on a separate line.
Using Data Structures

- Example 7–5 uses a data structure to store names, ages, and salaries.
- It then displays each of the entries by using a few assembly language procedures.
- The string procedure displays a character string, no carriage return/line feed combination is displayed—instead, a space is displayed.
- The Crlf procedure displays a carriage return/line feed combination.
  - the Numb procedure displays the integer
An Example of a Mixed-Language Program

- Example 7–6 shows how the program can do some operations in assembly language and some in C language.
- The assembly language portions of the program are the Dispn procedure that displays an integer and the Readnum procedure, which reads an integer.
- The program makes no attempt to detect or correct errors.
2. Using Assembly Language With Visual C/C++ For 32-bit Applications

- 32-bit applications are written with Microsoft Visual C/C++ Express for Windows and 16-bit applications using Microsoft C++ for DOS.

- Visual C/C++ Express for Windows is more common today.

- Visual C/C++ Express cannot easily call DOS functions such as INT 21H.
Figure 7–1 The new project screen selection of a WIN32 console application.
Directly Addressing I/O Ports

- If a program is written that must access an actual port number, we can use console I/O commands such as the `_inp(port)` command to input byte data, and the `_outp(port,byte_data)` command to output byte data.

- When writing software for the personal computer, it is rare to directly address an I/O port, but when software is written for an embedded system, we often directly address an I/O port.
• I/O ports may not be accessed in Windows environment if you are using Windows NT, Windows 2000, Windows XP, or Windows Vista.

• The only way to access the I/O ports in these modern operating systems is to develop a kernel driver.
  ◦ at this point in the text it would not be practical to develop such a driver

• In Windows 95 or 98, you can use inp and outp instructions in C/C++ to access I/O ports.
Developing a Visual C++ Application for Windows

- The Microsoft Foundation Classes (MFC) is a collection of classes that allows us to use the Windows interface without a great deal of difficulty.
- The MFC has been renamed to the common language runtime (CLR) in Visual C++ Express.
• To create a Visual C++ form-based application, start Visual C++ Express and click on Create Project near the upper left corner of the start screen.

• Figure 7–2 illustrates what is displayed when the CLR Windows Forms application type is selected under Visual C++ Express Projects.

• Enter a name for the project and select an appropriate path for the project, then click on OK.
Figure 7–2 Starting a C++ program for Windows in Visual C++ Express.
After a few moments the design screen should appear as in Figure 7–3.

In the middle section is the form created by this application.

To test the application, as it appears, find the green arrow located somewhere above the form and below the Windows menu bar at the top of the screen and click on it to compile, link, and execute the dialog application.

You have just created and tested your very first Visual C++ Express application.
Figure 7–3 Design window screen shot.
In the screen shot in Figure 7–3, several items are important to program creation and development.

- The right margin contains a Properties window, with the properties of the form; the left margin contains Solution Explorer.

- The tabs, at the bottom of Solution Explorer window, allow other views to be displayed such as a class view and so forth in this area.

- The tabs at the bottom of the Properties window allow the classes, properties, dynamic help, or output to be displayed in this window.
To create a simple application, select the toolbox by clicking on Tools at the top of the screen or by opening the View dropdown menu and selecting Toolbox from the list.

Click on the button control near the top of the toolbox, which selects the button.

Now move the mouse pointer (do not drag the button) over to the dialog application in the middle of the screen and draw, by left-clicking and resizing the button near the center.

See Figure 7–4.
Figure 7–4 A button control placed on the form.
• Once the button is placed on the screen, an event handler must be added so pressing or clicking the button can be handled.

• The event handlers are selected by going to the Properties window and clicking on the yellow lightning bolt.

• Make sure that the item selected for events is the button1 object.

• To switch back to the Properties window from the event window, click on the icon just to the left of the lightning bolt.
Locate the Click event (should be the first event) and then double-click on the textbox to the right to install the event handler for Click.

The view will now switch to the code view and change the location of the button click software.

The software currently in view is the button1_Click function, which is called when the user clicks on the button.

This procedure is illustrated in Example 7–8.
• We can modify the application to a more complex application as shown in Figure 7–5.
• The caption on the button has been changed to the word “Convert.”
• To return to the design screen, select the tab at the top of the program window that is labeled Form1.h[design]*.
• When in the Design window, change the caption on button1 by clicking the button and finding the Text property from the properties of button1 in the Properties window.
• Change the Text property to “Convert.”
• In Figure 7–5 there are three Label and three textbox controls in the illustration below and to the left of the Convert button.
• These controls are located in the toolbox.
• Draw them on the screen in approximately the same place as in Figure 7–5.
• The labels are changed in properties for each label control.
• Change the text for each label as indicated.
Figure 7–5 The first application.
• The goal is to display any number entered in the Decimal Number box as a number with any radix (number base) as selected by the number entered in the Radix box.

• The result appears in the Result box when the Convert button is clicked.

• To switch to the program view, click on the Form1.h tab at the top of the Design window.

• To obtain the value from an edit control, use Text property to obtain a string version of the number.
• The string must be converted to an integer.
• The Convert class provided in C++ performs conversion from/to most data types.
  ◦ the Convert class member function ToInt32 is used to transform the string into an integer
• The difficult portion of this example is the conversion from base 10 to any number base.
  ◦ will only function correctly if the number entered into textbox1 is an integer
• If a letter or anything else is entered, the program will crash and display an error.
- Windows does not use ASCII code, it uses Unicode so the Char (16-bit Unicode) is needed in place of the 8-bit char.
- A 0x30 is added to each digit to convert to ASCII code in the example.
- Horner’s algorithm:
  - divide the number by the desired radix
  - save the remainder and replace the number with the quotient
  - repeat steps 1 and 2 until the quotient is zero
• Since this is an assembly language text, the Convert class is not going to be used.
  ◦ the function is quite large

• To see how large, you can put a breakpoint in the software to the left of a Convert function by clicking on the gray bar to the left of the line of code.
  ◦ a brown circle, a \textbf{breakpoint}, will appear

• If you run the program, it will break (stop) at this point and enter the debugging mode so it can be viewed in assembly language form.
To display the disassembled code, run the program until it breaks, and then go to the Debug menu and select Windows. In the Windows menu, near the bottom, find “Disassembly.”

The registers can also be displayed to step through a program in assembly language.
The main problem with using inline assembly code is that the code cannot be placed into a Windows-managed forms application in a managed class.

In order to use the assembler, the function must be placed before the managed class in order for it to compile.

Therefore, in the project properties, Common Runtime Support must also be changed to /clr from the default setting of /clr:pure so it will compile successfully.
• Refer to Figure 7–6 for a screen shot of how to change Common Language Runtime support to /clr.

• A **managed** program runs under the virtual machine called .net and an **unmanaged** application operated in the native mode of the computer.

• The inline assembler generates native code for the microprocessor so it must be unmanaged and reside before the managed class in a program.
Figure 7–6 Changing to /clr for assembly language.
Example 7–13 illustrates how to replace part of the Horner’s algorithm with assembly code in a function called Adjust.

The adjust function tests the number for 9, and if it’s greater than 9, it adds $0x07$ and then $0x30$ to convert it to ASCII, which is returned.

By placing the assembly code before the managed class, it is available to the entire application and it executes in unmanaged or native mode.
3. Mixed Assembly And C++ Objects

- The inline assembler is limited because it cannot use MACRO sequences and conditional program flow directives.
- In some cases, it is better to develop assembly language modules that are then linked with C++ for more flexibility.
- This is especially true if the application is being developed by a team of programmers.
Linking Assembly Language with Visual C++

- Example 7–15 illustrates a flat model procedure that will be linked to a C++ program.
- We denote that the assembly module is a C++ module by using the letter C after the word flat in the model statement.
- The linkage specified by the letter C is the same for the C or C++ languages.
• The flat model allows assembly language software to be any length up to 2G bytes.
• The .586 switch appears before the model statement, which causes the assembler to generate code that functions in the protected 32-bit mode.
• The Reverse procedure, shown in Example 7–15, accepts a character string from a C++ program, reverses its order, and returns to the C++ program.
• The assembly language module can have any name and can contain more than one procedure, as long as each procedure contains a PUBLIC statement defining the name of the procedure as public.

• Any parameters that are transferred in the C++ program and the assembly language program are indicated with the backslash following the name of the procedure.
• Example 7–16 illustrates a C++ language program for DOS console applications that uses the Reverse assembly language procedure.

• The EXTERN statement is used to indicate that an external procedure called Reverse is to be used in the C++ program.

• The name of the procedure is case-sensitive, so make sure that it is spelled the same in both the assembly language module and the C++ language module.
• The EXTERN statement shows external assembly language procedure transfers a character string to the procedure and returns no data.

• If data are returned from the procedure, data are returned as a value in register EAX for bytes, words, or doublewords.

• If floating-point numbers are returned, they must be returned on the floating-point coprocessor stack.

• If a pointer is returned, it must be in EAX.
• Once both the C++ program and the assembly language program are written, the Visual C++ development system must be set up to link the two together.

• For linking and assembly, we will assume that the assembly language module is called Reverse.txt (you cannot add an .asm extension file to the file list for inclusion into a project, so just use the .txt extension and add a .txt file) and the C++ language module is called Main.cpp.
• Both modules are stored in the C:\PROJECT\MINE directory or some other directory of your choosing.
• After the modules are placed in the same project workspace, the Programmer’s Workbench program is used to edit both assembly language and C++ language modules.
Figure 7–7 Using the assembler to assemble a module in Visual C++. 

![Image of Reverse.txt Property Pages dialog box in Visual C++](image.png)
• At last, you can execute the program.
• Click on the green arrow. You should see two lines of ASCII text data displayed.
• The first line is in correct forward order and the second is in reverse order.
• Although this is a trivial application, it does illustrate how to create and link C++ language with assembly language.
Adding New Assembly Language Instructions to C/C++ Programs

- From time to time, as new microprocessors are introduced by Intel, new assembly language instructions are also introduced.
- These can’t be used in C++ unless you develop a macro to include them.
- An example is the CPUID assembly language instruction.
  - will function in an _asm block within C++ because the inline assembler does not recognize it.
Another group of newer instructions includes the MMX and SEC instructions.

These are also recognized, but in order to illustrate how a new instruction is added that is not in the assembler, we show the technique.

For example, the machine code for the CPUId instruction is 0FA2.

This 2-byte instruction can be defined as a C++ macro, as illustrated in Example 7–19.
To use the new macro in a C++ program, all we need to type is CPUID.

The _emit macro stores the byte that follows it in the program.

**Example 7–19:**

```c
#define CPUID _asm _emit 0x0f _asm _emit 0xa2
```
SUMMARY

- The inline assembler is used to insert short, limited assembly language sequences into a C++ program.
- The main limitation of the inline assembler is that it cannot use macro sequences or conditional program flow instructions.
- Two versions of C++ are available. One for 16-bit DOS con-sol-e applications and the other for 32-bit Windows applications.
SUMMARY

• The type chosen for an application depends on the environment, but in most cases programmers today use Windows and the 32-bit Visual Express version.

• The 16-bit assembly language applications use the DOS INT 21H commands to access devices in the system. The 32-bit assembly language applications cannot efficiently or easily access the DOS INT 21H function calls even though many are available.
SUMMARY

• The most flexible and often-used method of interfacing assembly language in a C++ program is through separate assembly language modules.

• The only difference is that these separate assembly language modules must be defined by using the C directive following the .model statement to define the module linkage as C/C++ compatible.
SUMMARY

• The PUBLIC statement is used in an assembly language module to indicate that the procedure name is public and available to use with another module.

• External parameters are defined in an assembly language module by using the name of the procedure in the PROC statement. Parameters return through the EAX register to the calling C/C++ procedure from the assembly language procedure.
Assembly language modules are declared external to the C++ program by using the extern directive. If the extern directive is followed by the letter C, the directive is used in a C/C++ language program.
SUMMARY

• When using Visual Studio, we can instruct it to assemble an assembly language module by clicking on Properties for the module and adding the assembler language program (ml /c /Cx /coff Filename.txt) and output file as an object file (File-name.obj) in the Custom Build step for the module.

• Assembly language modules can contain many procedures, but can never contain programs using the .startup directive.