

3

PIC18 Development Tools

3.1 Objectives

After completing this chapter, you should be able to

- Explain the function of software tools
- Explain the basic function of hardware tools
- Use MPLAB® IDE to enter programs and build the executable code
- Use MPLAB ICD2 to perform software debugging for the PIC18 microcontroller
- Use MPLAB simulator in perform software debugging for the PIC18 microcontroller

3.2 Development Tools

Development tools for microprocessors/microcontrollers can be classified into two categories: software and hardware. Software tools include programmers' editors, assemblers, compilers, simulators, debuggers, communication programs, and integrated development environment. Hardware tools include demo boards, logic analyzers, emulators, oscilloscopes, and logic probes.

Discussing all these tools is beyond the scope of this book. However, hardware and software tools made by Microchip and a few vendors for the PIC18 microcontrollers are reviewed in this book.

3.3 Software Tools

Software tools include text editors, cross assemblers, cross compilers, simulators, source-level debuggers, and integrated development environment.

3.3.1 Text Editors

The text editor is a program that allows us to enter and edit programs and text files. Editors range from very primitive to very sophisticated. A simple editor like the Notepad bundled with Windows 98/2000/XP provides simple editing functions in four different categories: *file*, *edit*, *search*, and *option*.

The Wordpad program bundled with the Windows 98/2000/XP is another simple editor that you can use to enter your program.

The Microchip MPLAB IDE has an embedded text editor for the user to enter programs. Most of the basic text editing functions are available in the MPLAB editor.

For professional programmers, neither Notepad nor Wordpad is adequate because of their primitive functions. A programmer's editor provides additional features, such as automatic keyword completion, keyword highlighting, syntax checking, and parentheses matching. These functions can speed up the user's program entry speed dramatically. The PFE editor is a free-ware programmer's editor available from the Internet and can be downloaded from www.wintel.com or www.simtel.net.

3.3.2 Cross Assemblers and Cross Compilers

Cross assemblers and cross compilers generate executable code to be placed in ROM, EPROM, EEPROM, or flash memory of a PIC18-based product. Currently, several vendors are providing cross assemblers and cross-C compilers for the PIC18 microcontroller. At the time of this writing, Microchip, CCS, IAR, and HI-TECH provide cross compilers for the PIC18 microcontroller.

3.3.3 Simulator

A simulator allows us to run a PIC18 program without having the actual hardware. The contents of registers, internal memory, external memory, and the program source code are displayed in separate windows. The user can set breakpoints to the program and examine the program execution results. The simulator allows the user to step through the program to locate program bugs. The MPLAB IDE development software from Microchip includes simulators for all PIC16/PIC17/PIC18 devices.

3.3.4 Source-Level Debugger

A source-level debugger is a program that runs on a PC (or a workstation) and allows you to find problems in your code at the high level (such as C) or assembly language level. A source-level debugger allows you to set breakpoints at statements either at the high or assembly language level. It can execute the program from the start of the program until the breakpoint and then display the values of program variables. A source-level debugger can also trace your program statement by statement. A source-level debugger may have the option to run your program on the demo board or simulator. Like a simulator, a source-level debugger can display the contents of program variables, registers, memory locations, and program code in separate windows. With a source-level debugger, all debugging activities are done at the source level.

A source-level debugger can also invoke a simulator to perform debugging activity. The C-Spy from IAR is a source-level debugger for the PIC18 microcontrollers that uses this approach. In addition to running the program on a demo board, the debugger of the MPLAB IDE can also use simulator to debug your program.

3.3.5 Integrated Development Environment

An integrated development environment (IDE) includes everything that you need to enter, assemble, compile, link, and debug your application program. It includes every software tool mentioned earlier. A tool is invoked by clicking on the icon of the corresponding tool. The MPLAB IDE is an IDE designed for all Microchip microcontrollers. It is free and can be downloaded from Microchip's Web site at www.microchip.com. The Embedded Workbench from IAR is also an IDE for the PIC18 microcontrollers.

3.4 Hardware Tools

Numerous hardware development tools are available for the PIC18 microcontrollers. Microchip provides the following hardware development tools to support the hardware development of PIC18-based product:

1. In-circuit emulators
2. Device programmers
3. Demo boards
4. In-circuit debugger (ICD2)

3.4.1 The Nature of Debugging Activities

When an embedded product is being designed, both the hardware and the software components are being developed in parallel. Therefore, software debugging needs to be done before the final hardware is completed. Software debugging in microcontroller-based product development can be classified into a software-only approach and a hardware-assisted approach. Using a demo board with a resident monitor or simulator to test the program is the software-only approach. In the hardware-assisted approach, either the logic analyzer or the emulator is used to trace the program execution. No matter what approach is used, one needs to run the application programs in order to find out if program execution results are what one expects. When the program execution results are not what one expects, one would examine the programs to find out if there are any obvious logical errors. Many errors can be identified in this manner. However, there are some errors that cannot be easily identified by examining the program.

Many microcontroller demo boards have an onboard monitor program. A good monitor allows the user to enter commands to display and modify the contents of registers and memory locations, set breakpoints, trace program execution, and download the user program onto the demo board for execution. By using appropriate monitor commands, the user will be able to determine whether his or her program executes correctly up to the breakpoint. Instruction tracing is also available from most monitors in case the user needs to pinpoint the exact instructions that cause the error. All peripheral functionality can be actually exercised using this approach. A source-level debugger can be written to communicate with the monitor on the demo board or invoke the simulator to execute the user program. Using this approach, you need not examine the contents of memory locations in order to find out if the program execution result is correct. Instead, values of program variables are available for examination during the process of program execution. In other words, debugging activity is carried out at the *source level*.

In the hardware-assisted approach, designers use an *in-circuit emulator* (ICE) to identify program errors. The ICE needs to reconstruct instructions being executed from the data flowing on the system bus of the target prototype (including address, data, and control signals). This approach may require certain built-in hardware support from the microcontroller used in the prototype in order to identify the boundary of instructions. The ICE also allows us to set breakpoints at those locations that are likely to have errors so that program execution result can be examined. This approach is especially useful when a demo board with a resident monitor is not available. The price of an ICE usually costs much more than a demo board with a resident monitor.

In the past decade, more and more microcontrollers provide hardware support for software debugging. The IEEE 1149.1 JTAG Boundary Scan Interface was initially proposed to standardize the interface for testing newly designed integrated circuits. Many companies (e.g., 8051-variant vendor Silicon Laboratory and Atmel) also utilize this interface to provide support for software debugging and programming the on-chip flash memory or one-time-programmable EPROM. Other companies (including Motorola and Microchip) provide a serial interface (called Background Debug Module) that utilizes two to four pins to support in-circuit programming and software debugging but do not provide boundary scan test capability. These interfaces allow nonintrusive inspection of memory and register contents, support breakpoints, and allow single stepping of user programs. Tool developers can take advantage of these capabilities to implement inexpensive source-level debugger. The ICD2 from Microchip is an in-circuit debugger that utilizes this interface.

3.4.2 ICE

All debugging activities for microcontroller-based products involve running the application program. Depending on the progress of the product development, some debugging activities must be performed before the hardware product is available. An ICE allows the user to debug his or her software before the final hardware is constructed. An ICE includes a target processor module to emulate the final hardware. It can reconstruct the instruction stream being executed on the fly.

Microchip manufactures the MPLAB ICE2000 ICE. As shown in Figure 3.1, the MPLAB ICE-2000 consists of four components:

1. *Emulator pod*. This unit communicates with a PC to receive the software program to be tested.
2. *Processor module*. This module executes the instructions downloaded into the emulator pod.
3. *Device adapter*. A device adaptor consists of IC sockets and additional circuitry to allow the MCU to be connected to the target hardware.
4. *Transition socket*. This socket and the device adapter plugs into the target hardware so that signals can be exercised to the target hardware.

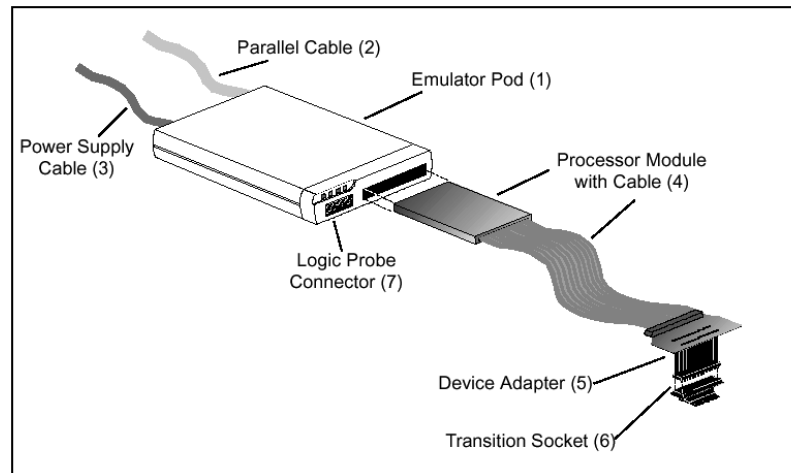


Figure 3.1 ■ MPLAB ICE2000 Emulator (reprint with permission of Microchip)

3.4.3 Device Programmer

The software program must be programmed into the on-chip ROM of the microcontroller before it can be tested. Microchip provides two programmers to meet the need of different users:

PICSTART® PLUS. PICSTART® Plus is a low-cost programmer for the PIC18 microcontrollers and other microcontrollers from Microchip. The photograph of this programmer is shown in Figure 3.2. A 40-pin socket is provided. By adding appropriate adapters, the PICSTART Plus can program PIC18 members with higher pin count. PICSTART Plus is connected to a PC through the serial port and is driven by the MPLAB IDE software.

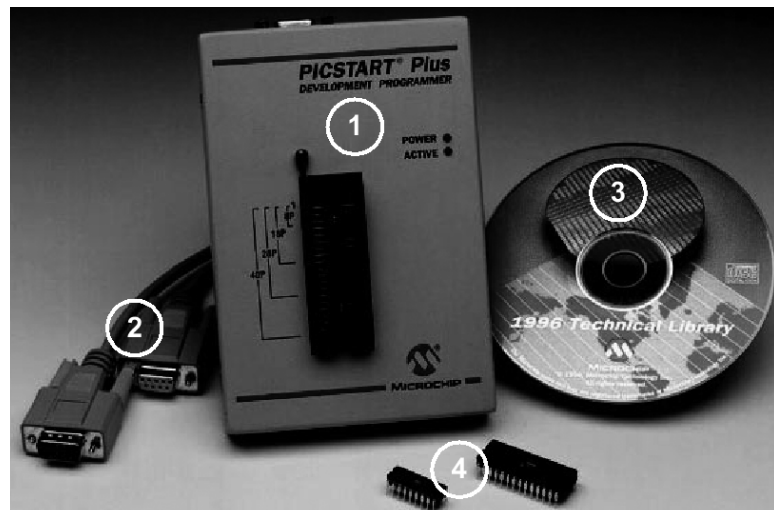


Figure 3.2 ■ PICSTART Plus programmer (reprint with permission of Microchip)

PRO MATE® II. The PRO MATE® II device programmer, shown in Figure 3.3, is another programmer made by Microchip that can program all microcontrollers from Microchip. Appropriate adapters are needed for different packages. Like the PICSTART Plus programmer, this programmer is also connected to the serial port of a PC and is controlled by the MPLAB IDE software.



Figure 3.3 ■ PRO MATE II programmer (reprint with permission of Microchip)

In addition to programming microcontrollers, PRO MATE II can also program many EPROM and EEPROM devices.

3.4.4 In-Circuit-Debugger II

In-Circuit-Debugger II (ICD2) is designed to be a low-cost solution for debugging devices that support In-Circuit Serial Programming (ICSP) protocol. These devices provide on-chip flash memory to hold application programs. To enable in-circuit debugging, the ICD2 programs a small debug executive module into the target PIC® microcontroller device (i.e., the PIC18 microcontroller in your demo board). This debug executive module will reside at the end of the program memory. The debug executive works by communicating with special on-chip functions built inside the PIC18 microcontroller.

Utilizing the in-circuit debugging capability of the Flash PIC microcontroller and Microchip's ICSP protocol, the ICD2 also acts as a programmer. It operates under MPLAB IDE, connects to an application, and runs the actual microcontroller in the design.

The ICD2 module contains the logic for debugging, programming, and control. It is connected to either a PC's serial port via a nine-pin serial cable or a USB port via a USB cable and to the PIC18 demo board or target application using a six-wire modular cable. A photograph of ICD2 is shown in Figure 3.4.

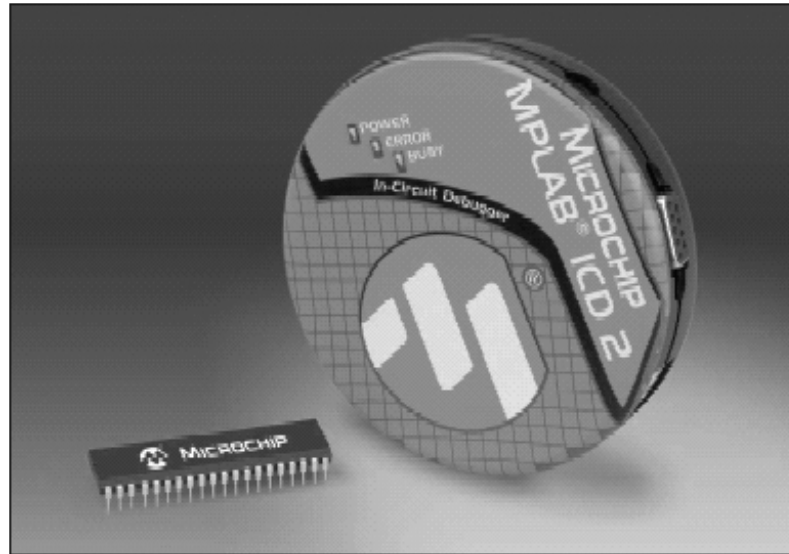


Figure 3.4 ■ Microchip ICD2 in-circuit debugger (reprint with permission of Microchip)

The module contains the software to provide serial communications to the PC and the target application or demo board and to program a supported PIC microcontroller device, all from the MPLAB IDE. The ICD2 module can provide power to the demo board/target application. The user can select VDD source on the Power tab in the ICD2 Settings dialog. If the target application draws over 200 mA, an additional power adapter must be connected to the demo board or target application. The target application also provides power to the ICD2 module only for the purpose of logic-level conversion.

The ICD2 interface cable must be plugged into a modular connector on the application circuit with the appropriate connections to the PIC microcontroller device. The interface cable carries the signals necessary to allow in-circuit debugging of the target application.

3.4.5 Demo Boards

Demo boards are useful for learning the microcontroller and testing the software before the final hardware is completed. As a learning tool, a well-designed demo board should allow the user to experiment with every peripheral function. Three demo boards will be reviewed in Section 3.8. Each of these demo boards can be used to test the programs in this book.

3.5 Using MPLAB IDE

MPLAB IDE is the center of Microchip's software development tool suite. It supports all the devices produced by Microchip that require software control. It consists of a text editor, a simulator, a cross assembler that supports all microcontrollers and digital signal processors, a simulator, and device drivers (for programmers, ICE, ICD, and ICD2) made by Microchip. A small number of third-party development tools (e.g., Hi-Tech C compiler) can also work with MPLAB IDE.

MPLAB IDE is a 32-bit window application that uses projects to manage software development tasks. A project may consist of a single or multiple files. This section will provide a step-by-step tutorial that sets up a project and gets you familiar with the debug capabilities of MPLAB IDE.

3.5.1 Getting Started with MPLAB IDE

MPLAB IDE provides the ability to do the following:

1. Create source code using the built-in editor.
2. Assemble, compile, and link source code using various language tools. An assembler, linker, and librarian come with MPLAB IDE, which also supports C compilers (MCC17 and MCC18) by Microchip. A limited number of third-party C compilers are also supported.
3. Debug the executable logic by watching program flow with the built-in simulator or in real time with the MPLAB ICE2000 emulator or MPLAB ICD2 in-circuit debugger.
4. Make timing measurements with the simulator or emulator.
5. View variables in Watch windows.
6. Program firmware (executable machine code) into devices with PICSTART® Plus or PRO MATE® II device programmers.
7. Find quick answer to questions from the MPLAB IDE online help.

To start the IDE, select Start>Programs>Microchip MPLAB IDE>MPLAB IDE from your monitor screen or simply double-click on the MPLAB IDE icon. A splash screen will display first, followed by the MPLAB IDE desktop as shown in Figure 3.5.

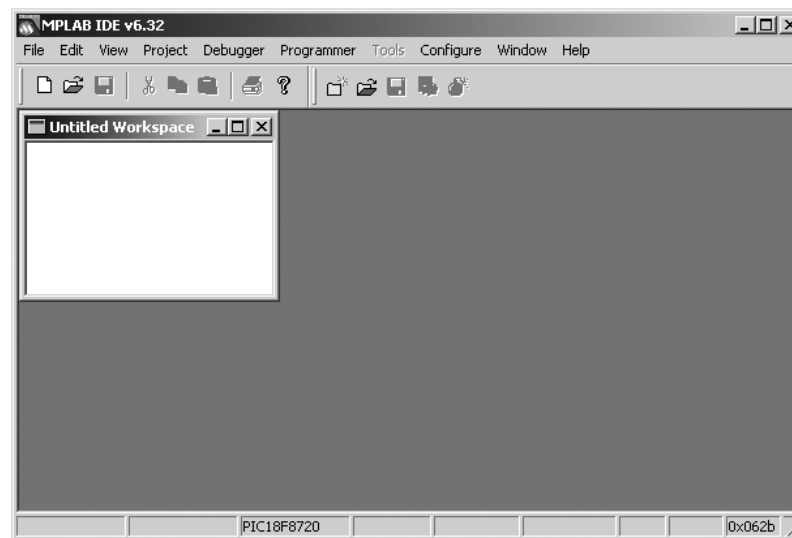


Figure 3.5 ■ MPLAB IDE desktop (reprint with permission of Microchip)

The major steps in software development using MPLAB IDE are as follows:

Step 1

Create a new project.

Step 2

Enter source files.

Step 3

Add source files into the project,

Step 4

Compile and build the executable code.

Step 5

Debug the project using the simulator, or ICD2, or ICE.

3.5.2 Creating a Simple Project

Before creating a new project, the user must configure the project and select the target device. After creating a new project, the user needs to set the language tool location and select the tool suite.

To configure the project, go to the *Configure> settings* menu and choose the Projects tab. Make sure the setup is as shown in Figure 3.6. Click on OK after making sure the setting is right.

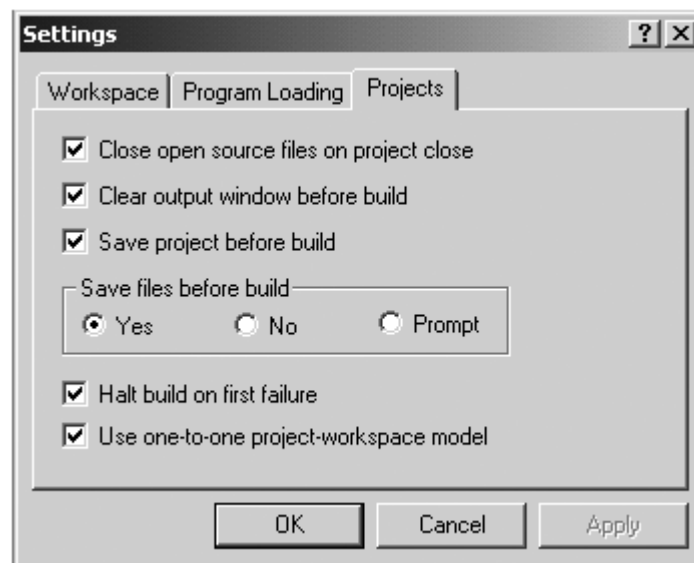


Figure 3.6 ■ Configure the project (reprint with permission of Microchip)

The next step is to make sure the right device is selected. For this tutorial, the PIC18F452 has been chosen. To select the target device, go to *Configure>Select Devices . . .* and make the selection as shown in Figure 3.7.

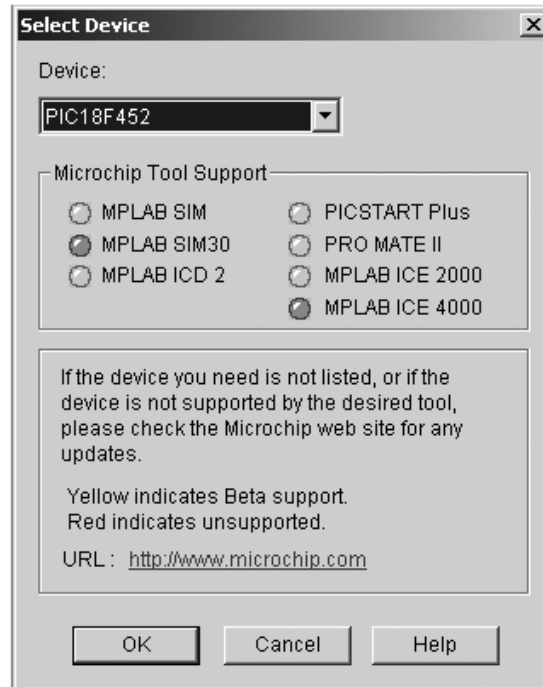


Figure 3.7 ■ Select device dialog (reprint with permission of Microchip)

After the correct target device has been selected, the user is ready to create a new project. When creating a new project, the user needs to decide where to place the project. This tutorial will assume that the user has decided to use the directory `c:\pic18\ch03` to place his or her projects. Follow these steps to create a new project:

1. Select the *Project > New* menu. The New dialog will open as shown in Figure 3.8.
2. Enter the name of the new project (e.g., `tutor1`).

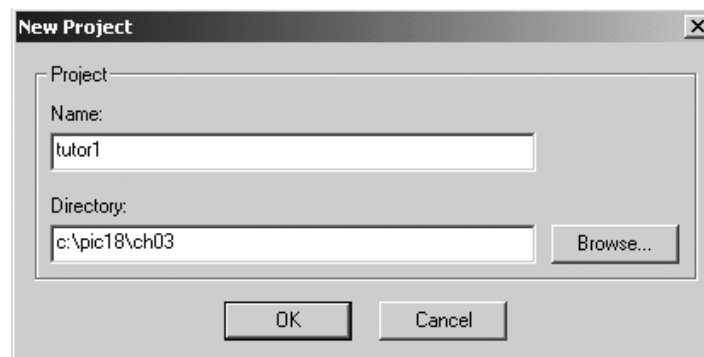


Figure 3.8 ■ New project dialog (reprint with permission of Microchip)

3. Use the Browse button to select the path to place the new project or type in the path.
4. When the Project name and Location are correct, click on OK.

After creating a new project, the user needs to make sure the language tool locations are set properly. To do this, select *Project > Set Language Tool Locations* to confirm the location of the Microchip Tool Suite. Click on MPASM Assembler (mpasmwin.exe). The full path to the MPASM Assembler executable should appear in the Location of Selected Tool text box as shown in Figure 3.9. If it is incorrect or empty, click on Browse to locate mpasmwin.exe. After the language tool location has been set up, the user can skip this step until he or she switches language tools.

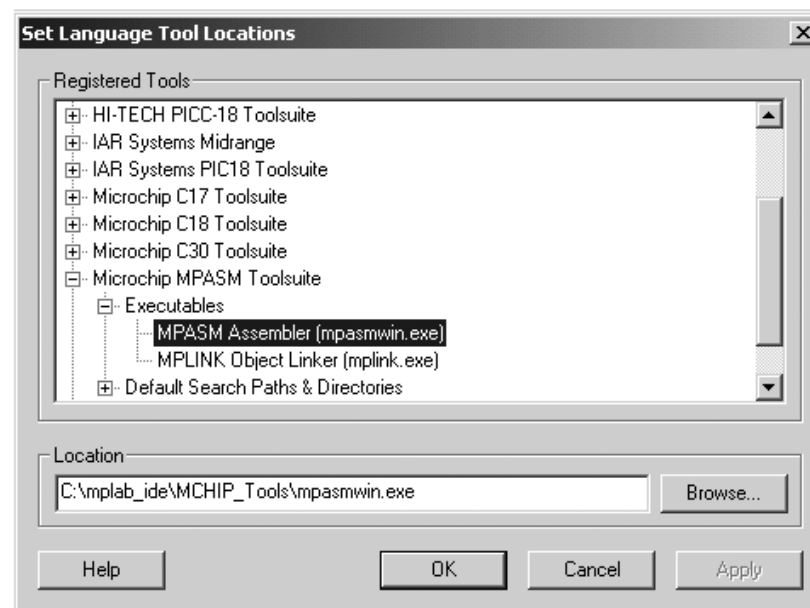


Figure 3.9 ■ Setting language tool location dialog (reprint with permission of Microchip)

Before starting entering your source code, one needs to set the language tool suite to be used in the new project. This allows the MPLAB IDE to tailor its operation more accurately with regard to context-sensitive editing and file extensions. Perform the following steps to select the language suite:

1. Select *Project > Set Language Toolsuite*.
2. For Active Toolsuite, select *Microchip MPASM Toolsuite* for this tutorial. The PICmicro® language tools will appear under Toolsuite Contents.
3. Click on OK.

The dialog for selecting language suite is shown in Figure 3.10.

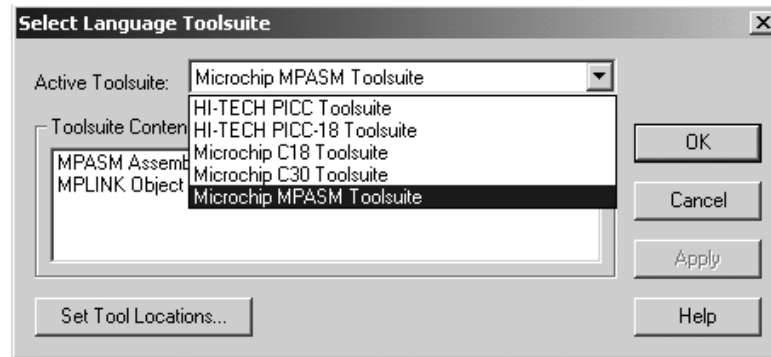


Figure 3.10 ■ Dialog for selecting language tool suite
(reprint with permission of Microchip)

3.5.3 Entering Source Code

After the new project has been configured properly, one can now enter the source code for the project.

Select *File> New*. A blank edit window will be opened in the workspace of MPLAB IDE. Enter the following program to the Edit window:

```

                                title "Finding the Number of Elements That Are a Multiple of 4"
                                #include <p18F452.inc>

ilimit    equ        0x20
count     set         0x00
loop_cnt  set         0x01
mask      equ        0x03          ; used to mask upper six bits
                                org        0x00
                                goto       start
                                org        0x08          ; high-priority interrupt service routine
                                retfie
                                org        0x18          ; low-priority interrupt service routine
                                retfie

start     clrf         count,A
                                movlw     ilimit
                                movwf     loop_cnt        ; initialize ii to N
                                movlw     upper array     ; use table pointer to point to the array
                                movwf     TBLPTRU, A      ; "
                                movlw     high array      ; "
                                movwf     TBLPTRH,A       ; "
                                movlw     low array       ; "
                                movwf     TBLPTRL,A      ; "

i_loop    movlw        mask
                                tblrd*+                ; read an array element into TABLAT
                                andwf     TABLAT,F,A
                                bnz        next          ; branch if not a multiple of 4
                                incf      count,F,A      ; increment count if it is a multiple of 4

```

```

next    decfsz    loop_cnt,F,A    ; decrement loop count
        goto     i_loop
        nop
array   db        0x00, 0x01, 0x30, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09
        db        0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13
        db        0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D
        db        0x1E, 0x1F
        end

```

This program counts the number of elements in the given array that are divisible by 4. The lowest two bits of a number divisible by 4 are 00. This program starts with the first element and checks if it is divisible by 4. It increments the variable **count** by 1 if an element is divisible by 4 until the last element is checked. The array is located in program memory.

Once the source code has been entered, the user should select *File> Save* and save the file in the project directory as **tutor1.asm**. After the user has saved the code, the program is shown with identifying colors for readability. This context-sensitive colorization is customizable. For more information about the editor, see *Help> MPLAB Editor Help*.

3.5.4 Adding Source Files to the Project

To add a source file to the project, select *Project> add Files to Project*. A dialog window as shown in Figure 3.11 will appear on the screen. Select the file that you just created and saved (tutor1.asm) and click on Open. After this, the newly inserted file should appear in the project pane under Source Files (shown in Figure 3.12). If it appears under Unclassifiable, confirm that you have selected the correct language tool suite by selecting *Project> Set Language Toolsuite*. “Microchip Toolsuite” should appear as the Active Toolsuite. If not, select it and click on OK. The file name should now appear under Source Files.

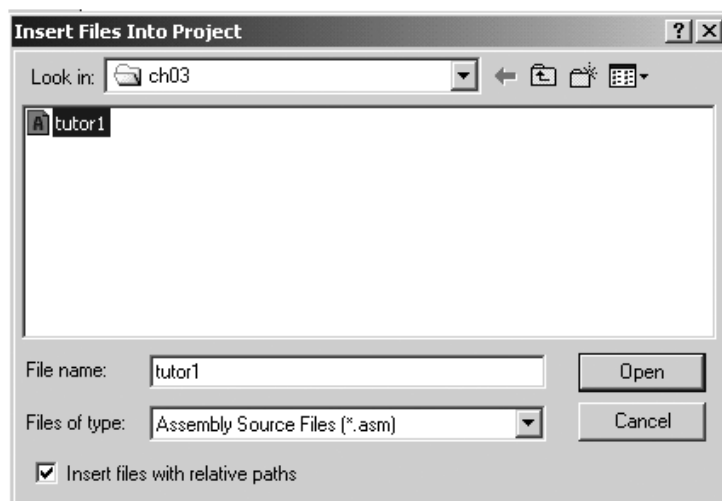


Figure 3.11 ■ Select input file dialog
(reprint with permission of Microchip)

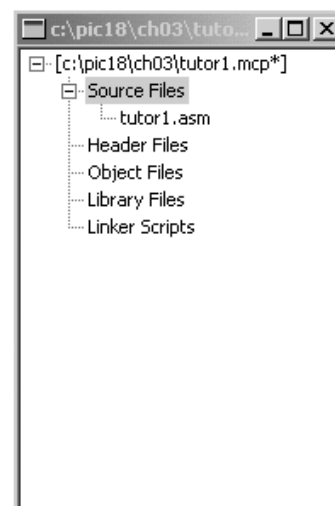


Figure 3.12 ■ Project pane
(reprint with permission of Microchip)

Save the project by selecting *Project> Save*. You can also add files and save projects by using the right mouse button in the project pane. Experiment by clicking the right mouse button while the cursor is over Source Files. Note that the menu options are different.

3.5.5 Building the Project

After adding source files into the project, it is time to build the project. This step will compile the source code using the selected language tool suite.

To build the project, select *Project> Build All* or *Project> Make* (or press function key F10). This file (tutor.asm) should assemble successfully. If this project does not build correctly, check the following items and then build the project again:

- Check the spelling and format of the program in the editor window. If the assembler reported errors in the Output window, double-click on the error in the Output window. This will indicate the corresponding line in the source code with a green arrow in the gutter of the source code window.
- Check that the correct assembler (MPASM assembler) for the PICmicro device is being used.

On a successful build, the debug files (with file name extension .cod or .cof) generated by the language tool will be loaded. This file allows one to debug using the source code and view program variables symbolically in Watch windows.

For this one-file example, a project does not seem necessary. However, the real power of projects comes when many files are to be compiled/assembled and linked to form the final executable application. Projects keep track of all this for the user.

3.5.6 Debugging the Project

After building the project, one will want to check that it is functioning the way he or she intended. To do this, one will need to select a debug tool. There are five debug tools under the *Debugger>Select Tool* menu:

1. MPLAB ICD2
2. MPLAB ICE4000
3. MPLAB SIM
4. MPLAB ICE2000
5. MPLAB SIM30

Among these five debug tools, ICD2, SIM, and ICE2000 can be used to debug PIC18 microcontroller applications. The uses of SIM and ICD2 are discussed in the next two sections. The use of MPLAB ICE2000 or the ICE4000 is not discussed because of their higher cost.

3.6 Using the MPLAB SIM in Debugging PIC18 Applications

Program simulation involves the following major operations:

- Simulator setup
- Running the code
- Viewing variables

- Using Watch windows
- Setting breakpoints
- Tracing code

3.6.1 Setting Up the Simulator

The first step for simulating the program is to select MPLAB SIM as the debugging tool. This is done by selecting *Debugger>Select Tool> MPLAB SIM*. After selecting MPLAB SIM as the debugging tool, the status bar on the bottom of the MPLAB IDE window should change to “MPLAB SIM”. Additional menu items should now appear in the Debugger menu. Additional toolbar elements would also appear on the MPLAB IDE window. This is shown in Figure 3.13.

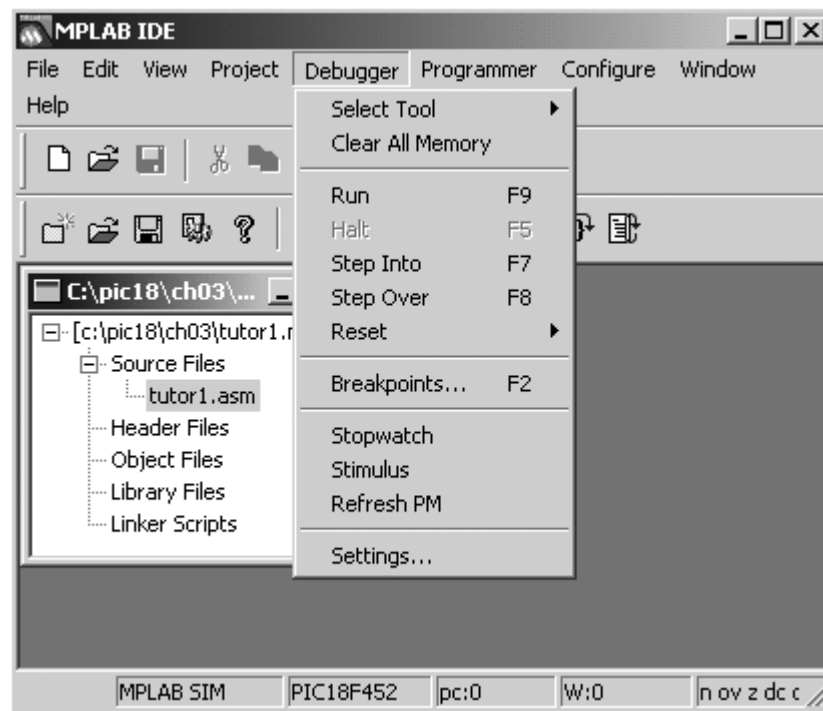
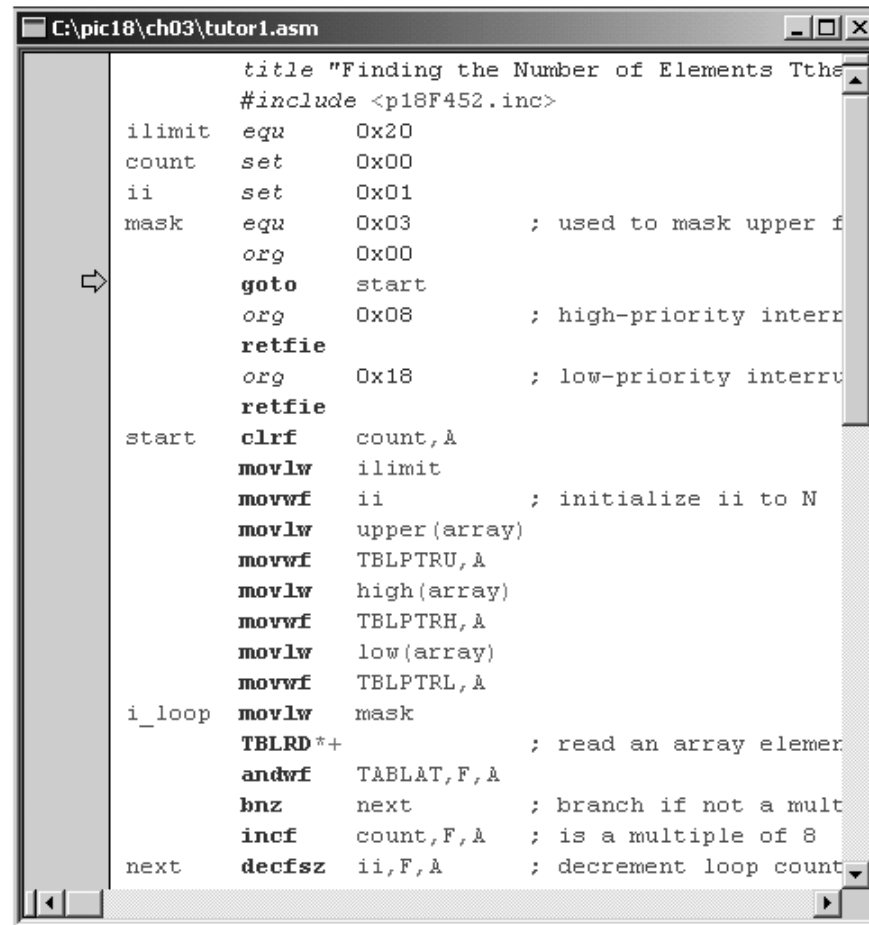


Figure 3.13 ■ MPLAB IDE window status change after selecting SIM.
Additional menu items also appear under Debugger menu.
(reprint with permission of Microchip)

3.6.2 Running Code under MPLAB SIM

Before running the program, one needs to set the program counter to the start of the program by selecting *Debugger>Reset* (or press function key F6). A green arrow should appear in the gutter of the source code window (in Figure 3.14), indicating the first source code line that will be executed.



```

C:\pic18\ch03\tutor1.asm
title "Finding the Number of Elements Tthe
#include <p18F452.inc>

ilimit equ 0x20
count set 0x00
ii set 0x01
mask equ 0x03 ; used to mask upper f
org 0x00
goto start
org 0x08 ; high-priority interr
retfie
org 0x18 ; low-priority interr
retfie
start clrf count,A
movlw ilimit
movwf ii ; initialize ii to N
movlw upper(array)
movwf TBLPTRU,A
movlw high(array)
movwf TBLPTRH,A
movlw low(array)
movwf TBLPTRL,A
i_loop movlw mask
TBLRD*+ ; read an array element
andwf T&BLAT,F,A
bnz next ; branch if not a mult
incf count,F,A ; is a multiple of 8
next decfsz ii,F,A ; decrement loop count

```

Figure 3.14 ■ Source code window after Reset (reprint with permission of Microchip)

Select *Debugger>Run* (or press function key F9) to run the program. The message "Running. . ." will appear on the status bar.

Because the simulator was not told where to stop, it will keep running. To halt the program execution, select *Debugger > Halt* (or press function key F5). The line of code where the application halted will be indicated by the green arrow.

We can also single step through the application program by selecting *Debugger > Step Into* (or press function key F7). This will execute the currently indicated line of the code and move the arrow to the next line of code that will be executed.

3.6.3 Viewing Variables

One can see the values of variables at any time by putting the mouse cursor over their names anywhere in the source file. A small output window will pop up to show the current value. A screen shot for placing mouse over the variable **count** is shown in Figure 3.15.

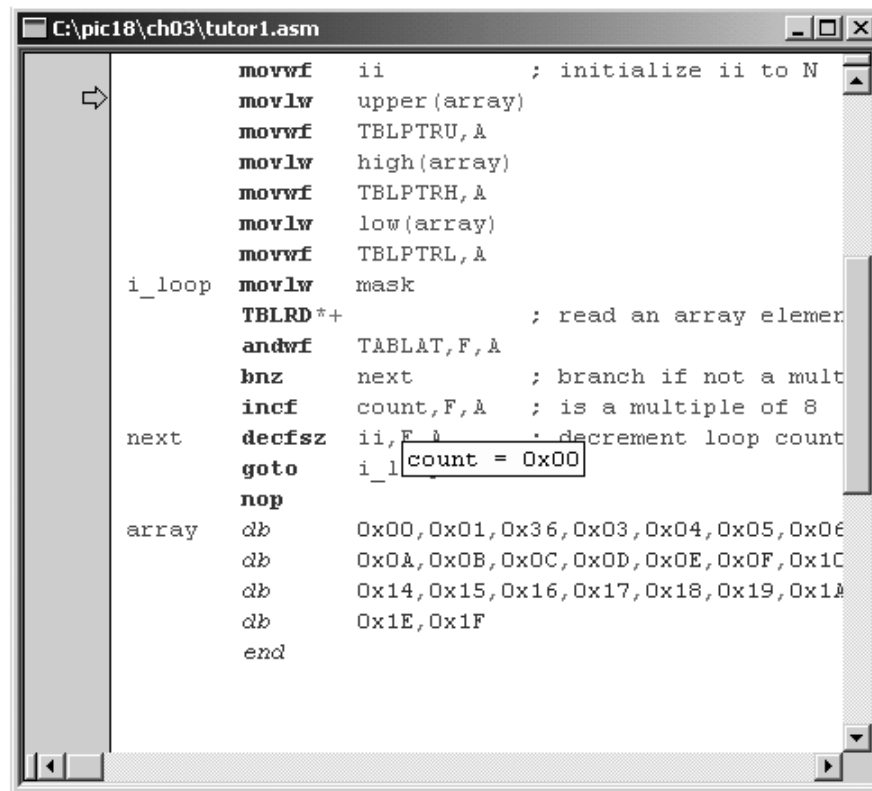


Figure 3.15 ■ Mouse over variable “count” (reprint with permission of Microchip)

3.6.4 Using Watch Window to View Variables

Often one wants to watch certain key variables all the time. Rather than floating the mouse cursor over the name each time one wants to see the value, one can open a Watch window. The Watch window will remain on the screen and show the current variable values. Watch windows can be found under the View menu.

To open a Watch window, *View> Watch*. There are two ways to add a symbol into the Watch window:

1. Click the mouse in the Watch window and then type the name of the symbol and press the Enter key.
2. Select from the symbol selection box at the top of the window and then click on **Add Symbol** to add it to the Watch window list.

One can also add any special-function register (SFR) into the Watch window list. The procedure is identical to the procedure for adding symbols.

The only symbol of interest to us is **count**. Add it to the Watch window. The resultant Watch window is shown in Figure 3.16.

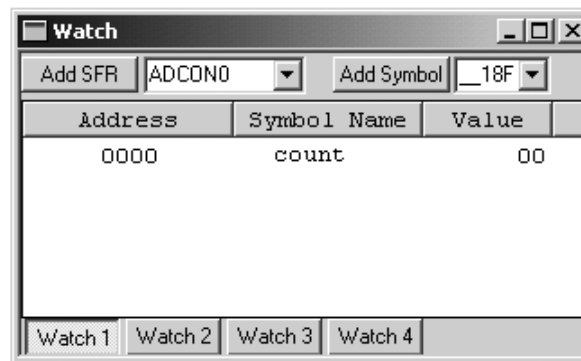


Figure 3.16 ■ Watch window for tutor1 (reprint with permission of Microchip)

Now we can watch the symbol value change as we step through the program:

1. Press function key F6 to reset your program.
2. Press function key F7 to step through the program one instruction at a time until you have stepped to the following program line:

```
incf    count,F,A    ; increment count if it is a multiple of 4
```
3. Step one more time to see the value of **count** in the Watch window change from 0 to 1. The changed value appears in red color.
4. Continue to step through the program until you reach the instruction that follows **incf count,F,A**. You will see that the value of **count** increments if a value that is divisible by 4 is reached.
5. Continue to step through the program until all the array elements have been checked. The final value of **count** is 9 for this example.

3.6.5 Setting Breakpoints

Sometimes one wants to run the program to a specific location and then halt. This is accomplished by using breakpoints. To set a breakpoint, press the right mouse button on the code line that one wants to set a breakpoint. Suppose one wants to set a breakpoint at the **nop** instruction. Press the right mouse button, and a pop-up window as shown in Figure 3.17 will appear. It is correct that Figure 3.17 appeared overlapping.

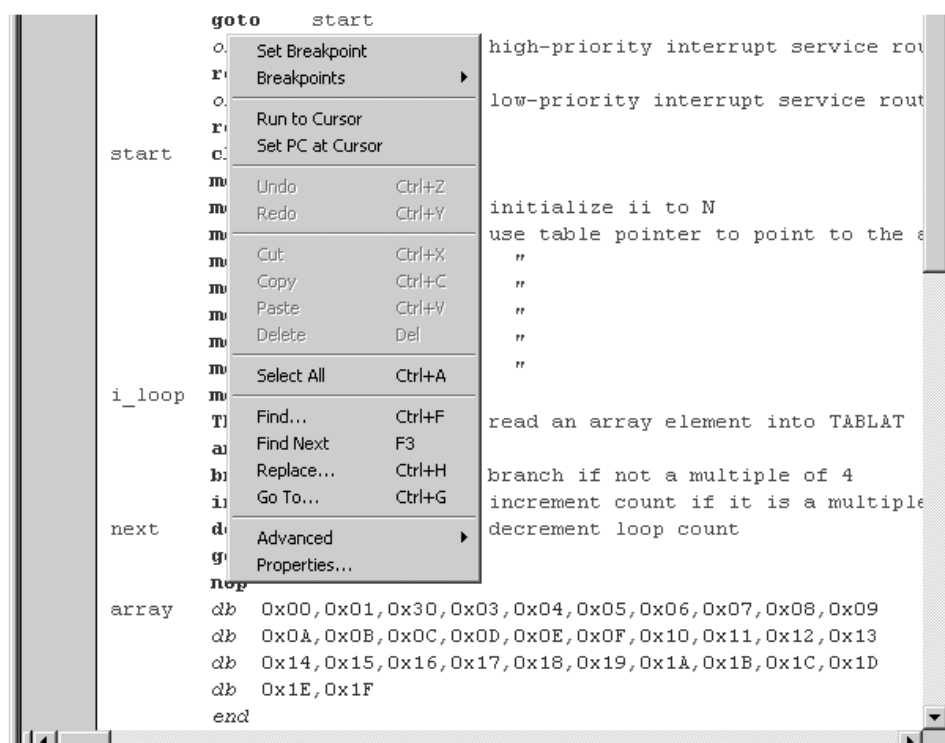


Figure 3.17 ■ A screen snapshot after pressing the right mouse button at “nop” (reprint with permission of Microchip)

From the pop-up menu that appears, select *Set Breakpoint*. A “stop sign” should appear in the gutter next to the line (shown in Figure 3.18). Before running the program, press function key F6 to reset the program.

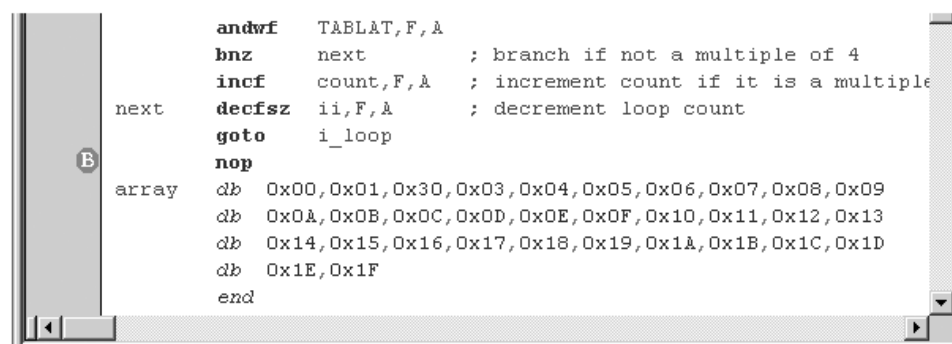


Figure 3.18 ■ Source code window - Set Breakpoint (reprint with permission of Microchip)

To run the program, select *Debugger> Run*. It should run briefly and then halt on the line at which the breakpoint was set. The screen should look like that in Figure 3.19 after the program halts.

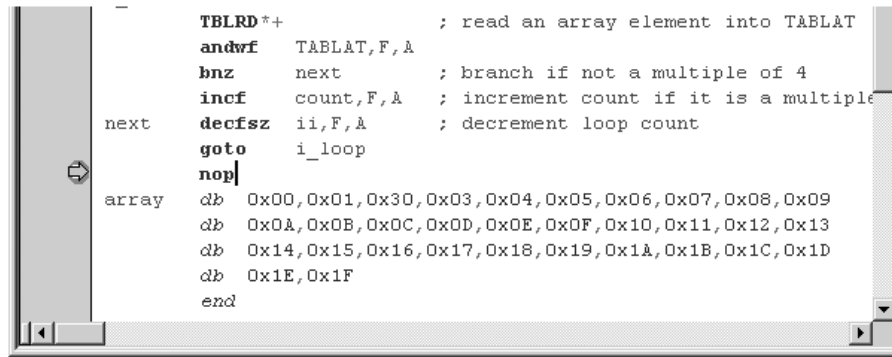


Figure 3.19 ■ Source code window—Breakpoint Halt (reprint with permission of Microchip)

3.6.6 Tracing Code

There are times that the user will have difficulty identifying the program bugs. Tracing program becomes necessary under such situations. Although single stepping through the program would allow the user to identify the program bugs, it would be easier for the simulator to capture the execution trace of many instructions all at once while the user goes through them instruction by instruction.

The Simulator Trace can be used to record the execution of the user program. The user can enable the Simulator Trace by selecting *Debugger> Settings* and choosing the Pins/Trace tab.

As shown in Figure 3.20, there are two check boxes to control how the Simulator Trace collects data. When only the top box is checked, the simulator collects data when the simulator is in Run mode, it collects data until the user halts at a breakpoint or manually stops the simula-

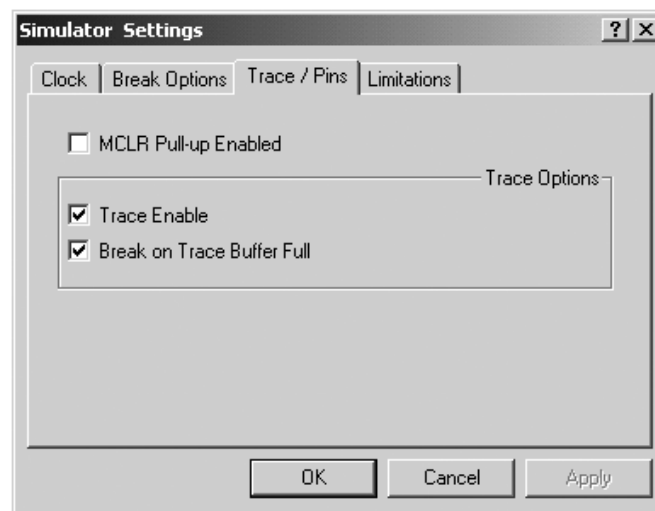


Figure 3.20 ■ Simulation Trace enable (reprint with permission of Microchip)

tor. It will show the last 8192 cycles collected. This mode is useful if the user wants to see the record of instructions leading up to a breakpoint.

If the second button is also checked, the trace memory will collect 8192 cycles of data and then stop collecting and halt the user application at a breakpoint. This mode is useful for seeing the record of instructions after the user presses run.

After enabling Simulator Trace, make sure that the program is reset before running the program. Since *tutor1.asm* is a short program, the simulator will halt at the breakpoint. Select *View> Simulator Trace* to view the simulation trace. The trace display is shown in Figure 3.21. The trace display shows a time stamp at every cycle, and the data that were read or written into file registers will be captured and displayed.

Line	Addr	Op	Label	Instruction	SA	SD	DA	DD	Cycles	Probe 7	Probe 6	Probe 5	Probe 4	Probe 3	Probe 2	Probe 1	Probe 0
0	0000	EF0D		GOTO 0x1a	----	--	----	--	0000000000	0	0	0	0	0	0	0	0
1	0002	F000		NOP	----	--	----	--	0000000001	0	0	0	0	0	0	0	0
2	001A	6A00	start	CLRF 0, 0	----	--	0000	00	0000000002	0	0	0	0	0	0	0	0
3	001C	0E20		MOVLW 0x20	----	--	0FE8	20	0000000003	0	0	0	0	0	0	0	0
4	001E	6E01		MOVWF 0x1, 0	----	--	0001	20	0000000004	0	0	0	0	0	0	0	0
5	0020	0E00		MOVLW 0	----	--	0FE8	00	0000000005	0	0	0	0	0	0	0	0
6	0022	6EF8		MOVWF 0xff8, 0	----	--	0FF8	00	0000000006	0	0	0	0	0	0	0	0
7	0024	0E00		MOVLW 0	----	--	0FE8	00	0000000007	0	0	0	0	0	0	0	0
8	0026	6EF7		MOVWF 0xff7, 0	----	--	0FF7	00	0000000008	0	0	0	0	0	0	0	0
9	0028	0E3E		MOVLW 0x3e	----	--	0FE8	3E	0000000009	0	0	0	0	0	0	0	0
10	002A	6EF6		MOVWF 0xff6, 0	----	--	0FF6	3E	000000000A	0	0	0	0	0	0	0	0
11	002C	0E03	i_loop	MOVLW 0x3	----	--	0FE8	03	000000000B	0	0	0	0	0	0	0	0
12	002E	0009		TBLRD*+	----	--	0FF5	00	000000000C	0	0	0	0	0	0	0	0
13	0030	16F5		ANDWF 0xff5, 0	0FF5	00	0FF5	00	000000000E	0	0	0	0	0	0	0	0
14	0032	E101		BNZ 0x36	----	--	----	--	000000000F	0	0	0	0	0	0	0	0
15	0034	2A00		INCF 0, 0x1, 0	0000	00	0000	01	0000000010	0	0	0	0	0	0	0	0
16	0036	2E01	next	DECFSZ 0x1, 0x0001	20	0001	1F	0000000011	0	0	0	0	0	0	0	0	0
17	0038	EF16		GOTO 0x2c	----	--	----	--	0000000012	0	0	0	0	0	0	0	0
18	003A	F000		NOP	----	--	----	--	0000000013	0	0	0	0	0	0	0	0
19	002C	0E03	i_loop	MOVLW 0x3	----	--	0FE8	03	0000000014	0	0	0	0	0	0	0	0
20	002E	0009		TBLRD*+	----	--	0FF5	01	0000000015	0	0	0	0	0	0	0	0
21	0030	16F5		ANDWF 0xff5, 0	0FF5	01	0FF5	01	0000000017	0	0	0	0	0	0	0	0
22	0032	E101		BNZ 0x36	----	--	----	--	0000000018	0	0	0	0	0	0	0	0
23	0036	2E01	next	DECFSZ 0x1, 0x0001	1F	0001	1E	000000001A	0	0	0	0	0	0	0	0	0
24	0038	EF16		GOTO 0x2c	----	--	----	--	000000001B	0	0	0	0	0	0	0	0
25	003A	F000		NOP	----	--	----	--	000000001C	0	0	0	0	0	0	0	0

Figure 3.21 ■ Simulation trace display (reprint with permission of Microchip)

There are 18 columns in the trace display. The meanings of these columns are as follows:

- Line. Decimal cycle number from start of trace session
- Addr. Program address of instruction
- Op. Numeric op code of instruction
- Label. Symbolic label of instruction, if known
- Instruction. Disassembled instruction
- SA. Source address, the register address of read operation
- SD. Source data, the data read from the register
- DA. Destination address, the register address of the destination
- DD. Destination data, the data written into the destination register
- Cycles. Time before the execution of an instruction, from reset
- *n* Probe (*n* = 7, 0). Irrelevant in simulator, used in MAPLAB ICE2000 emulator only (each *n* value is in one column)

If there is any dash in the row for these values, it means that the operation did not access any file register for this instruction. The column with the label of “Time” is the time stamp. This can be used to measure the execution time of routines. The time is calculated on the basis of the clock frequency entered in the *Debugger> Settings Clock* tab.

In line 15 of Figure 3.21, both the source register and the destination register are the same register (at 0x00, the count value). The value of count is 1 after the first array element (0x00) is checked because 0x00 is a multiple of 4. You can scroll down the Simulator Trace and find that the final value of count is 9.

If the user puts the cursor over the top row of the trace display where the column headings are listed and presses the right mouse button, a configuration dialog will pop up as shown in Figure 3.22. All the checked items will appear in the trace window. The user can uncheck columns to reduce clutter if the user is not interested in the data in those columns. The entries labeled as Probe 7, Probe 6, and so on are for the MPLAB ICE2000 emulator trace and are not relevant to the simulator. They should be unchecked.

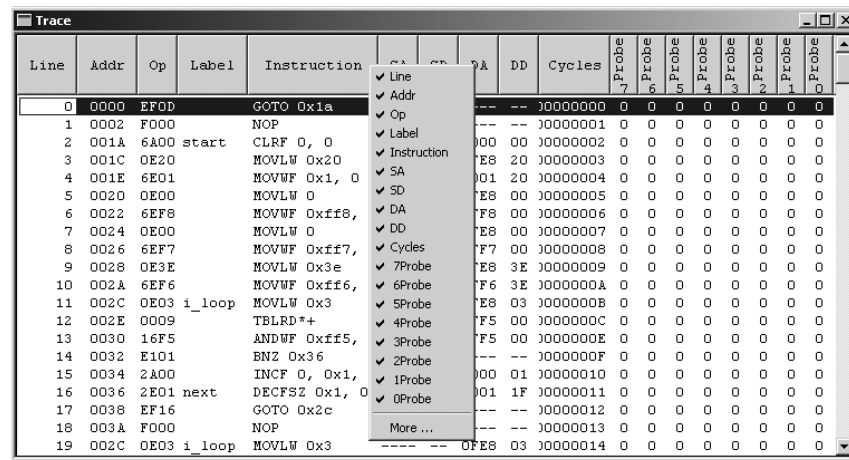


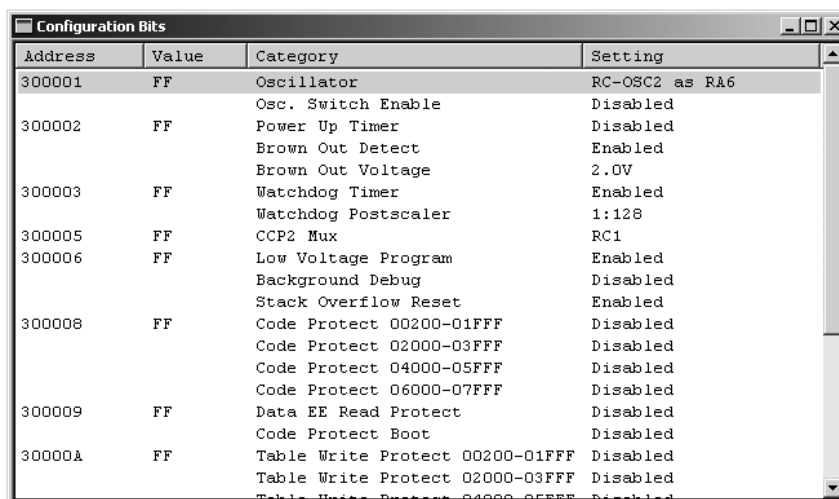
Figure 3.22 ■ Configure Simulation Trace (reprint with permission of Microchip)

3.6.7 Advanced Simulator Options

There are other characteristics of the simulator that can be configured from the MPLAB IDE dialogs. Normally, the default condition of the configuration bits has the Watch Dog Timer (WDT) enabled. This will cause the simulator to reset when the internal WDT times out.

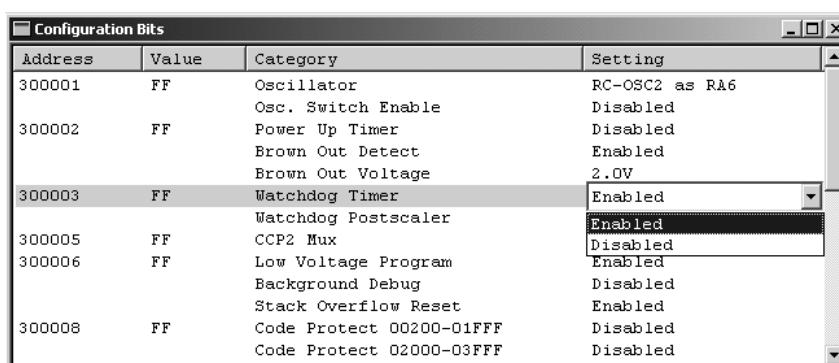
Unless one wants to test the functioning of the WDT, one would be better off by disabling the WDT. To disable the WDT, one needs to bring up the Configuration Bits dialog by selecting *Configure> Configuration Bits*. The Configure Bits dialog is shown in Figure 3.23.

Click on the line that contains Watchdog Timer, and then a selection box will appear. Scroll down to select *Disabled* to prevent the WDT from causing the program to reset. The WDT selection box is shown in Figure 3.24. If the user uses ICD2 to perform software debugging, then *Low Voltage Program* should also be disabled, whereas *Background Debug* should be enabled.



Address	Value	Category	Setting
300001	FF	Oscillator	RC-OSC2 as RA6
		Osc. Switch Enable	Disabled
300002	FF	Power Up Timer	Disabled
		Brown Out Detect	Enabled
		Brown Out Voltage	2.0V
300003	FF	Watchdog Timer	Enabled
		Watchdog Postscaler	1:128
300005	FF	CCP2 Mux	RC1
300006	FF	Low Voltage Program	Enabled
		Background Debug	Disabled
		Stack Overflow Reset	Enabled
300008	FF	Code Protect 00200-01FFF	Disabled
		Code Protect 02000-03FFF	Disabled
		Code Protect 04000-05FFF	Disabled
		Code Protect 06000-07FFF	Disabled
300009	FF	Data EE Read Protect	Disabled
		Code Protect Boot	Disabled
30000A	FF	Table Write Protect 00200-01FFF	Disabled
		Table Write Protect 02000-03FFF	Disabled
		Table Write Protect 04000-05FFF	Disabled

Figure 3.23 ■ Configuration Bits dialog (reprint with permission of Microchip)



Address	Value	Category	Setting
300001	FF	Oscillator	RC-OSC2 as RA6
		Osc. Switch Enable	Disabled
300002	FF	Power Up Timer	Disabled
		Brown Out Detect	Enabled
		Brown Out Voltage	2.0V
300003	FF	Watchdog Timer	Enabled
		Watchdog Postscaler	Enabled
300005	FF	CCP2 Mux	Disabled
300006	FF	Low Voltage Program	Enabled
		Background Debug	Disabled
		Stack Overflow Reset	Enabled
300008	FF	Code Protect 00200-01FFF	Disabled
		Code Protect 02000-03FFF	Disabled

Figure 3.24 ■ Disable the WDT timer (reprint with permission of Microchip)

3.7 Using the MPLAB ICD2

ICD2 can be connected to the PC via one of the COM ports or USB ports. The USB port connection is preferred because it provides much faster data transfer between the PC and the ICD2.

After building the project, the user can select ICD2 to debug his or her program. This choice is preferred when the hardware, such as a PIC18 demo board, is available. ICD2 can be selected by selecting *Debugger> Select Tool> MPLAB ICD2*.

Before using the ICD2 to debug the application, the user must make sure that his or her target hardware or demo board is connected to the ICD2 using a modular connector. The shape of the modular connector is shown in Figure 3.25.

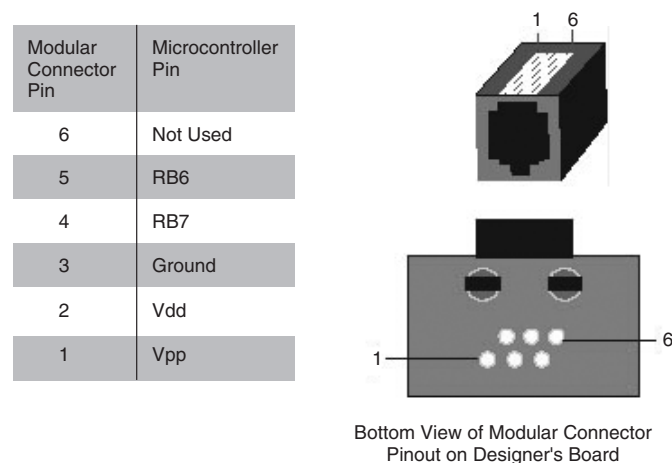


Figure 3.25 ■ MPLAB ICD2 modular connector (reprint with permission of Microchip)

3.7.1 ICD2 Settings

ICD2 needs to be set up properly before it can be used. Select *Debugger> Settings* (or *Programmer> Settings*) to configure ICD2. A setting window as shown in Figure 3.26 will appear on the screen. Figure 3.26 shows the status of the ICD programmer. The status of ICD2 is connected and it is automatically connected at startup.

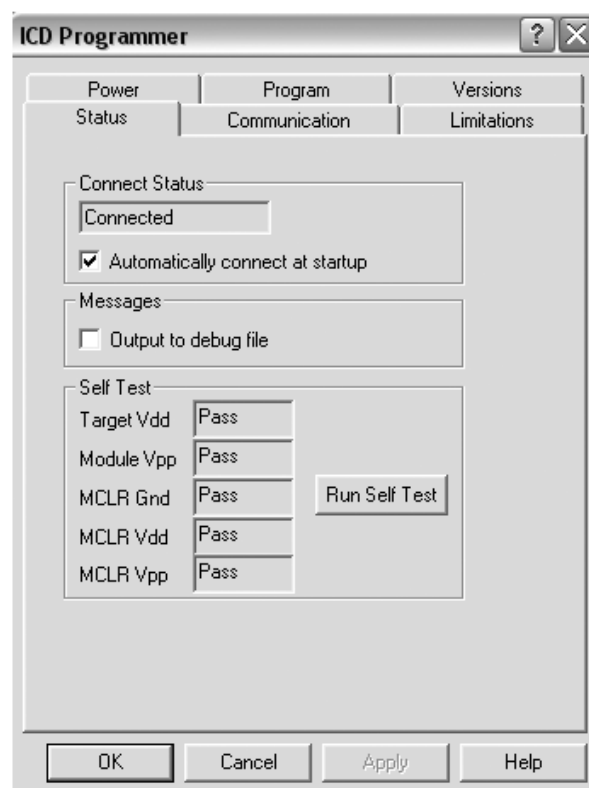


Figure 3.26 ■ ICD2 set up window (reprint with permission of Microchip)

The user must make sure that every setting is correct. There are three settings that need to be set properly: power, communication, and program. Click on the Power tab, and the screen will change to that in Figure 3.27. Notice that the Power Target Circuit From ICD2 setting is not selected. This is not desirable because ICD2 does not have enough power to drive the target hardware or demo board.

Click on the Communication tab shown in Figure 3.26, and the available settings will be made visible as shown in Figure 3.28. Choose USB if the PC has an available USB port. Otherwise, choose one of the COM ports. When selecting a COM port, the user will also need to set the baud rate. There are two baud rates to choose from: 19200 and 57600. Try the higher data rate. If it is not working properly, then switch to 19200.

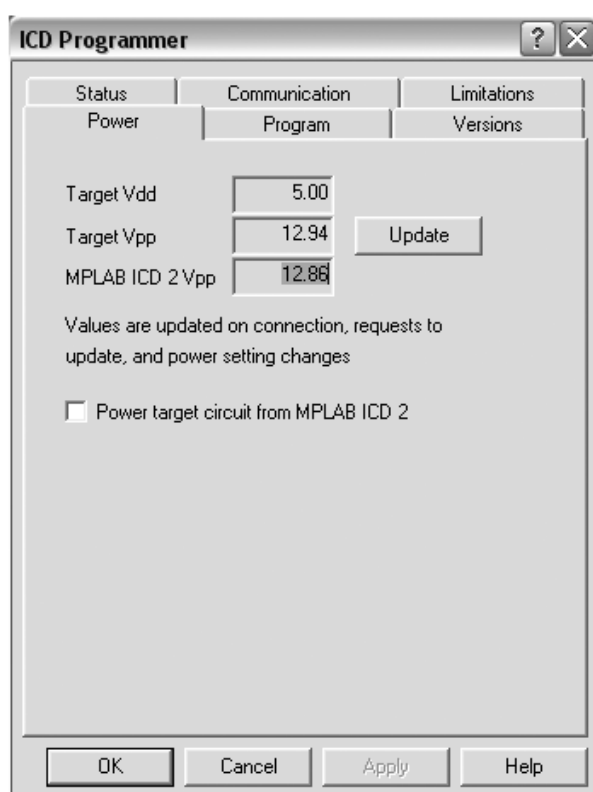


Figure 3.27 ■ ICD2 power status and setting
(reprint with permission of Microchip)

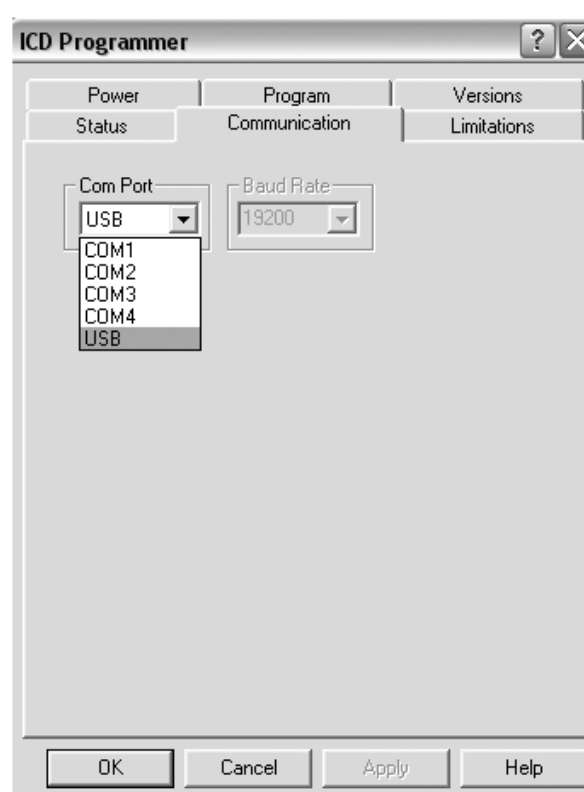


Figure 3.28 ■ Communication settings for ICD2
(reprint with permission of Microchip)

The last setting to be made is **Program**. Click on the Program tab, and its setting will be brought out as shown in Figure 3.29. The default settings are acceptable for this tutorial.

The remaining two tabs (Limitations and Versions) are for information purpose. The user should know the limitations of ICD2. Click on the Limitations tab, and its contents are shown in Figure 3.30.

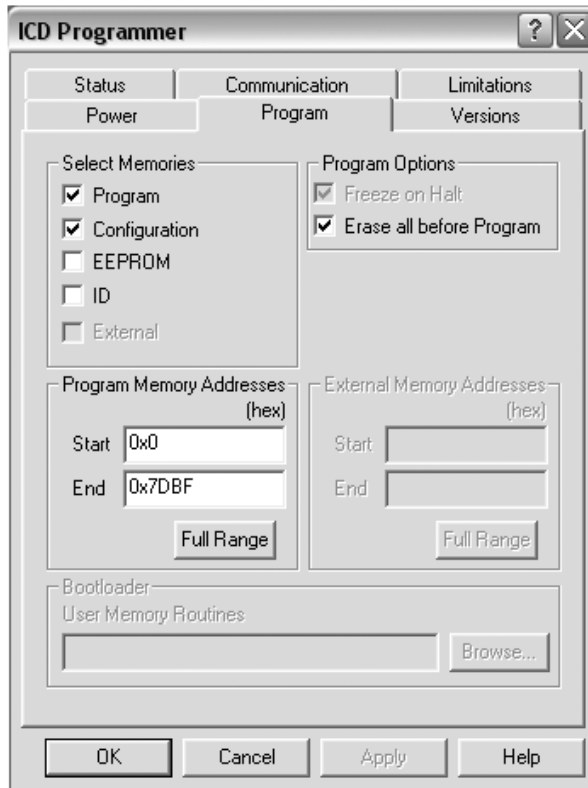


Figure 3.29 ■ ICD2 program setting
(reprint with permission of Microchip)

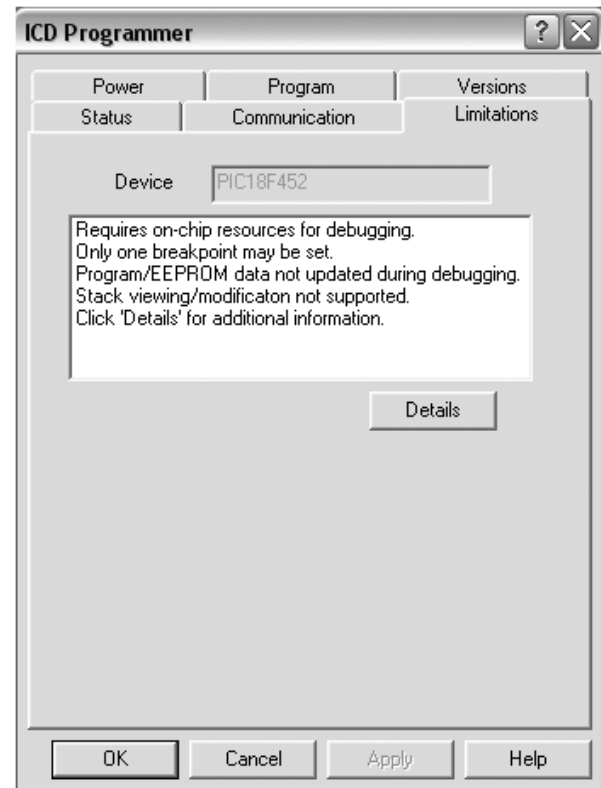


Figure 3.30 ■ ICD2 limitations (reprint with permission of Microchip)

ICD2 will use certain program memory and special- function registers (SFRs) on the target microcontroller unit. By clicking on the Details tab, the user can look for device specific limitations.

After completing the settings of ICD2, the user is ready to debug his or her application on the target hardware with the help of ICD2. Most of the debug techniques applicable to MPLAB SIM are also applicable to ICD2.

Here we use a program that computes the sum of integers from 1 to n to illustrate the process. The program is as follows:

```
#include <p18F452.inc>
n      equ      D'100'
sum_hi set      0x01      ; high byte of sum
```

```

sum_lo    set      0x00      ; low byte of sum
i          set      0x02      ; loop index i
          org      0x00      ; reset vector
          goto     start
          org      0x08
          retfie
          org      0x18
          retfie
start      clrf     sum_hi,A   ; initialize sum to 0
          clrf     sum_lo,A   ; “
          clrf     i,A        ; initialize i to 0
          incf     i,F,A      ; i starts from 1
sum_lp     movlw    n         ; compare i with n and skip if greater than
          cpfsgt   i,A
          goto     add_lp     ; perform addition when i ≤ 50
          goto     done       ; it is done when i > 50
add_lp     movf     i,W,A      ; place i in WREG
          addwf    sum_lo,F,A  ; add i to sum_lo
          movlw    0
          addwfc   sum_hi,F,A  ; add carry to sum_hi
          incf     i,F,A      ; increment loop index i by 1
          goto     sum_lp
done       nop
          end

```

Perform the following steps to enter and build the project:

1. Follow the procedure described in Section 3.5.3 to enter the source code (call it tutor2.asm).
2. Follow the procedure described in Section 3.5.4 to add source code to the project (call the project tutor2).
3. Follow the procedure described in Section 3.5.5 to build the project.

After building the project, perform the following steps:

1. Configure the ICD2 debugger properly.
2. Before debugging the program using ICD2, the target device needs be programmed.
Program the hex file (generated by the assembler) into the demo board by selecting the Program command under the Debugger menu.
3. Set up a watch window and add variables **sum_hi** and **sum_lo** into the Watch window.
4. Set a breakpoint at the statement of **done nop**.
5. Reset ICD2 by pressing function key F6.
6. Run the program (the program execution will be halted at the breakpoint).

The resultant screen on the MPLAB IDE window is shown in Figure 3.31. The Watch window displays the sum as a hex value 13BA (equivalent to decimal 5050) and is correct.

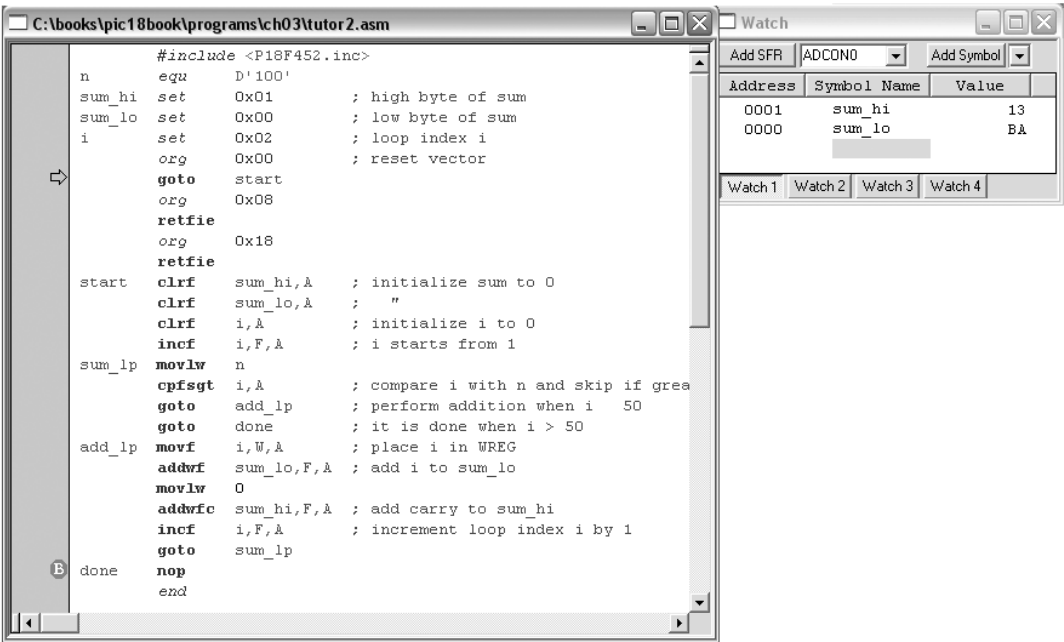


Figure 3.31 ■ Program for computing the sum of integers from 1 to 100 and the watch window (reprint with permission of Microchip)

If the program is not running correctly, then the user will need to use the techniques described in Sections 3.6.2 to 3.6.6 to debug the program. The user can combine the use of breakpoints, single stepping, and code tracing to identify the errors in the program.

3.8 Demo Boards from Shuan-Shizu Enterprise

Shuan-Shizu Enterprise has designed three PIC18 demo boards for learning the PIC18 microcontroller. The SSE452, the SSE8720, and the SSE8680 use the PIC18F452, the PIC18F8720, and the PIC18F8680, respectively, as their microcontrollers. Product information about these demo boards can be found at the Web site at www.evb.com.tw or by e-mail at Vincent-fan@umail.hinet.net. The features of these demo boards are described in the following sections.

3.8.1 SSE452 Demo Board

The main design feature of this demo board is that it allows the user to easily switch microcontrollers (see Figure 3.32). The SSE452 allows the user to specify the option of adding 28-pin and/or 40-pin ZIF sockets for the microcontroller. This allows the user to experiment with different 28-pin or 40-pin PIC18 microcontrollers.

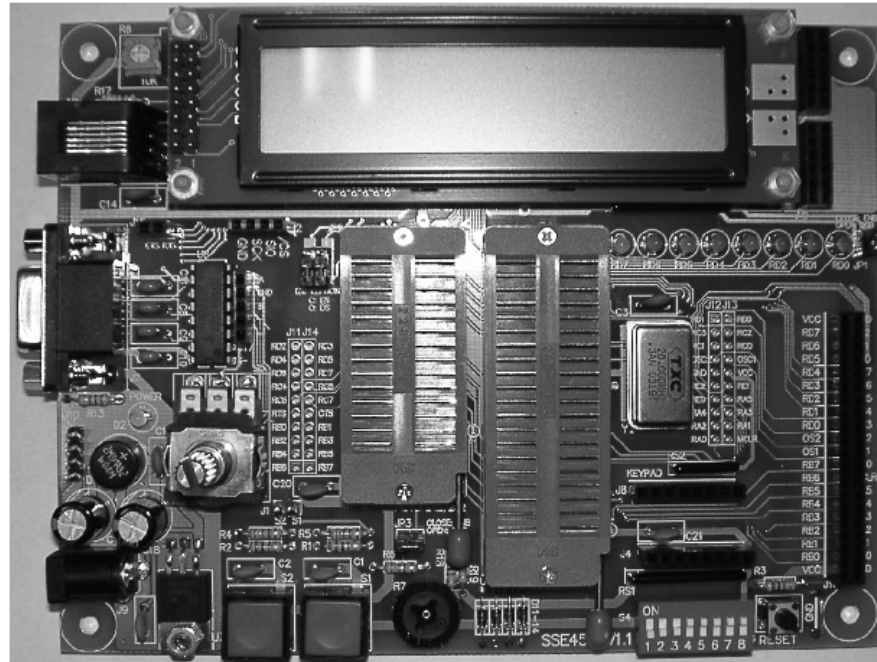


Figure 3.32 ■ Photo of the SSE452 demo board (reprint with permission of Shuan-Shizu Enterprise)

In addition to the PIC18F452 on-chip peripheral functions, the SSE452 adds the following features:

1. One PCB suitable for any 28-pin or 40-pin PIC18 microcontroller
2. High-current sink/source: 25 mA
3. One RS-232 connector
4. Two debounced push-button switches (can be used as external interrupt sources)
5. One 8-bit DIP switch for digital input
6. One 4 × 4 keypad connector for interfacing with 16-key keypad
7. One rotary encoder with push button for optional input
8. One TC77 temperature sensor with SPI interface
9. One EEPROM (24LC04B) with I²C interface
10. One 2 × 20 bus expansion port to make signals available to end user
11. One potentiometer for exercising the A/D function
12. Optional devices: 2 × 20 character LCD, 48/28-pin ZIF socket
13. Digital signals with frequency from 1 Hz up to 8 MHz for exercising timer functions
14. ICD2 connector

3.8.2 SSE8720 Demo Board

This demo board uses the PIC18F8720 as its microcontroller and was designed for those users who need more I/O pins and more on-chip flash program memory (see Figure 3.33). The features of the SSE8720 are as follows:

1. Digital signals with frequency from 1 Hz up to 16 MHz for experimenting with timer functions
2. ICD2 connector for debugging
3. DB9 connector provides EIA232 interface to connect to USART1
4. Four debounced switches (connected to RB0/INT0, RB1/INT1, RB2/INT2, and RB3/INT3) and one reset button
5. One potentiometer connected to RA0/AN0 pin for evaluating the A/D converter
6. One 8-bit DIP switch (Port F)
7. Eight LEDs
8. One 2×20 LCD
9. On-board 5-V regulator
10. One EEPROM with I²C interface
11. An SPI-compatible (four-wire) digital temperature sensor TC72
12. An 8- Ω speaker (driven through an NPN transistor) connected to RC2/CCP1 pin
13. One Dallas DS1306 SPI-compatible real-time clock chip
14. Two 2×20 connectors for accessing microcontroller signals

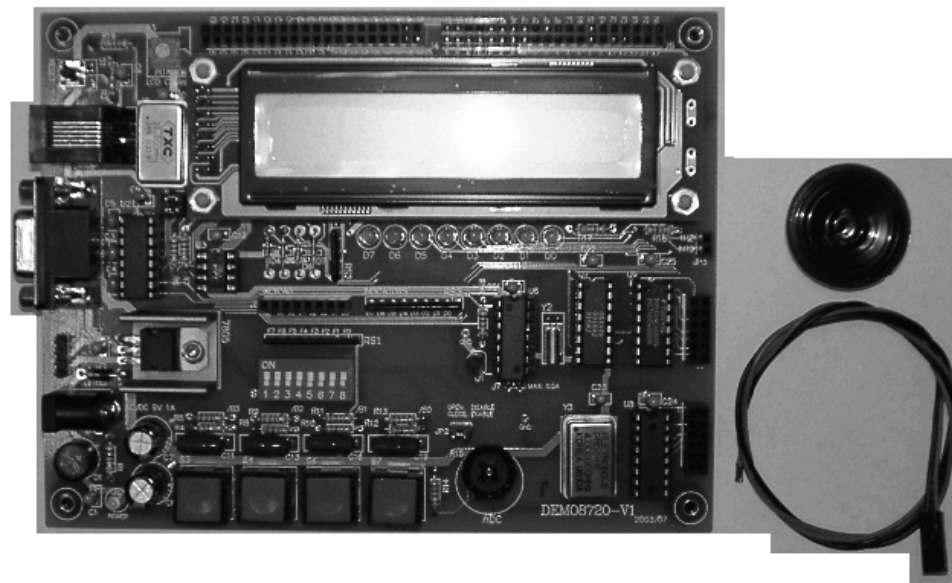


Figure 3.33 ■ SSE8720 demo board with speaker (reprint with permission of Shuan-Shizu Enterprise)

3.8.3 The SSE8680 Demo Board

The SSE8680 is designed for those users who are interested in experimenting with the CAN network (see Figure 3.34). Since this board uses the 80-pin PIC18F8680, it is also suitable for those applications that require many I/O pins. The CAN network is widely used in automotive and control applications. In addition to the on-chip peripheral functions of the PIC18F8680 microcontroller, the SSE8680 demo board has the following features:

1. Digital signals with frequency ranging from 1 Hz up to 16 MHz for experimenting with timer functions
2. ICD2 connector for debugging
3. DB9 connector provides EIA232 interface to connect to USART1
4. Four debounced switches (connected to RB0/INT0, RB1/INT1, RB2/INT2, and RB3/INT3) and one reset button (not connected to any pin)
5. One potentiometer connected to RA0/AN0 pin for evaluating the A/D converter
6. One 8-bit DIP switch (Port F)
7. Eight LEDs
8. One 2 × 20 LCD

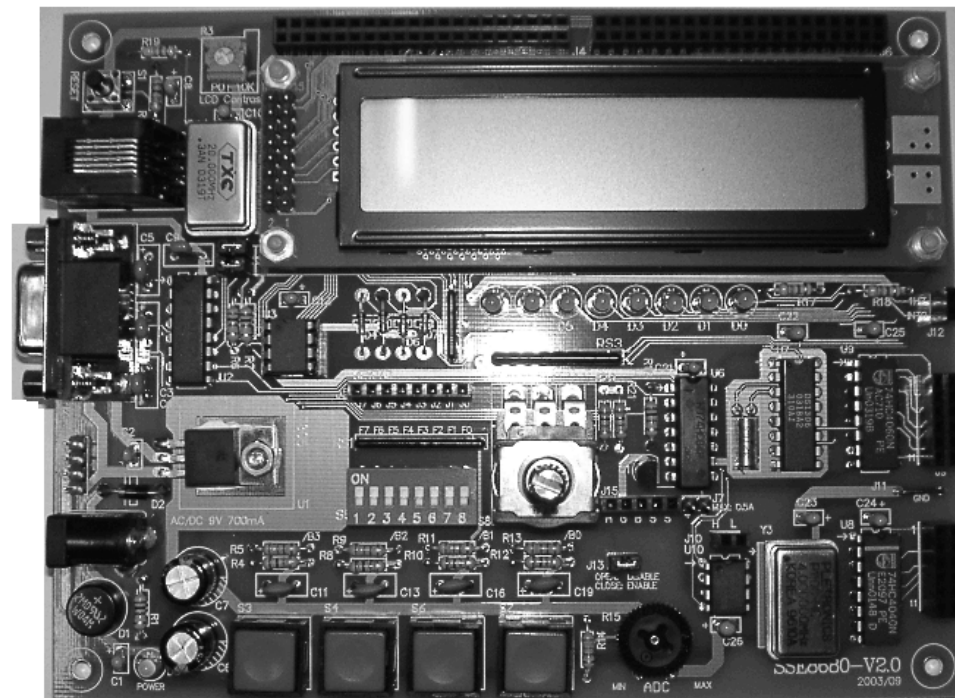


Figure 3.34 ■ SSE8680 demo board with LCD kit (reprint with permission of Shuan-Shizu Enterprise)

9. On-board 5-V regulator
10. One EEPROM with I²C interface
11. An SPI-compatible (four-wire) digital temperature sensor TC72
12. An 8- Ω speaker (driven through an NPN transistor) connected to RC2/CCP1 pin
13. One Dallas DS1306 SPI-compatible real-time clock chip
14. Two 2 \times 20 connectors for accessing microcontroller signals
15. MCP2551 CAN transceiver
16. Rotary encoder

3.8.4 Debug Monitor

A debug monitor is being developed for these demo boards. The debug monitor will allow the user to download the hex file (created by MPLAB® IDE) onto the demo board without using the ICD2 in-circuit debugger. In addition, this monitor will allow the user to set breakpoints, single step the program, and display and modify the contents of register values. The monitor will also be able to display the program downloaded into the demo board. (It needs a disassembler to do this.). The beta version of the monitor is functioning. The stable version of the monitor should be available by the time this book is published.

Without a monitor, one needs the MPLAB IDE, a demo board, and an ICD2 to experiment with the hardware. With the monitor, one needs only the MPLAB IDE and a demo board in order to experiment with the hardware peripheral functions.

3.9 Summary

Hardware and software development tools are essential for learning the features of the microcontroller and developing microcontroller-based products. This chapter provides a brief review of most of the development tools from Microchip and also the demo boards made by Shuan-Shizu Enterprise.

Undoubtedly, the MPLAB® IDE is the most important software development tool from Microchip. This package consists of the following:

- Assemblers for all microcontrollers manufactured by Microchip
- MPLINK linker
- Simulators for all microcontrollers manufactured by Microchip
- Control programs for several hardware tools, such as MPLAB ICD2 debugger and other debugging hardware made by Microchip
- An IDE that combines all the software development tools and allows the user to perform development work from program entry until simulation without leaving the same environment

All microcontrollers provide certain features to support software debugging. Many microcontrollers utilize the JTAG interface to support software debugging in addition to performing chip-testing function. The PIC18 microcontroller provides the ICSP protocol to support software debugging and on-chip flash memory programming.

Tutorials on the use of MPLAB IDE, ICD2, and demo boards are provided at the end of this chapter.

3.10 Lab Exercises and Assignments

L3.1 Start the MPLAB IDE program and enter the following program as a text file with the file name progL1.asm:

```

        #include    <p18F452.inc>
        radix       dec
sum_hi   set         0x01
sum_lo   set         0x00
lp_cnt   set         0x02
kk       equ        50
        org         0x00
        goto        start
        org         0x08
        retfie
        org         0x18
        retfie
start    movlw       kk
        movwf       lp_cnt
        clrf        sum_hi,A
        clrf        sum_lo,A
        movlw       upper array
        movwf       TBLPTRU,A
        movlw       high array
        movwf       TBLPTRH,A
        movlw       low array
        movwf       TBLPTRL,A
loop     tblrd*+
        btfsc       TABLAT,0,A
        goto        next
        movf        TABLAT,W,A
        addwfc      sum_lo,F,A
        clrf        WREG,A
        addwfc      sum_hi,F,A
next     decfsz      lp_cnt,F,A
        goto        loop
        nop
array    db          01,02,03,04,05,06,07,08,09,10
        db          11,12,13,14,15,16,17,18,19,20
        db          21,22,23,24,25,26,27,28,29,30
        db          31,32,33,34,35,36,37,38,39,40
        db          41,42,43,44,45,46,47,48,49,50
        end

```

Perform the following operations:

1. Create a new project called **progL1** in the same directory where the program **progL1.asm** is stored.
2. Configure MPLAB IDE properly.
3. Add the source code file progL1.asm into the project.

4. Build the project.
5. Select MPLAB SIM as the debug tool and perform appropriate configuration as described in Sections 3.6.1 to 3.6.7.
6. Set a breakpoint at the last instruction (**nop**).
7. Open a Watch window and enter symbols **sum_hi** and **sum_lo** into the window.
8. Reset the project by pressing the function key F6.
9. Run the program by pressing the function key F9.
10. Check the values of symbols **sum_hi** and **sum_lo**.
11. View the simulation trace to identify errors. You should not see any error if you type everything correctly.

Can you figure out what this program is doing?

L3.2 Enter the following program as a text file and name it progL2.asm:

```

#include <p18F452.inc>
radix      dec
ar_cnt     equ      30                ; array count
lp_cnt     set      0x00              ; loop count symbol
buffer     set      0x010
           org      0x00
           goto     start
           org      0x08
           retfie
           org      0x18
           retfie
start      movlw     upper array        ; set TBLPTR as the array pointer
           movwf     TBLPTRU,A          ; "
           movlw     high array         ; "
           movwf     TBLPTRH,A          ; "
           movlw     low array          ; "
           movwf     TBLPTRL,A          ; "
           lfsr      FSR0,buffer        ; use FSR0 as a pointer to buffer
           movlw     ar_cnt
           movwf     lp_cnt            ; set up loop count value
loop1      tblrd*+
           movff     TABLAT,POSTINCo    ; move from table latch to buffer in data memory
           decfsz    lp_cnt, F,A        ; decrement lp_cnt and skip if zero
           goto      loop1
           lfsr      FSR0,buffer        ; set FSR0 to point to the first array element
           lfsr      FSR1,buffer+ar_cnt-1 ; set FSR1 to point to the last array element
           movlw     ar_cnt/2
           movwf     lp_cnt            ; set loop count
loop2      movf      INDF0,W            ; copy array element
           movff     INDF1,POSTINC0     ; store and increment pointer FSR0
           movwf     POSTDEC1          ; store and decrement pointer FSR1
           decfsz    lp_cnt,F,A
           goto      loop2
           nop

```

```

forever    goto    forever
array      db      1,2,3,4,5,6,7,8,9,10
           db      11,12,13,14,15,16,17,18,19,20
           db      21,22,23,24,25,26,27,28,29,30
           end

```

Perform the following operations:

1. Create a new project called **progL2** in the same directory where the file progL2.asm is saved.
2. Add the file **progL2.asm** into the project.
3. Build the project.
4. Select ICD2 as your debug tool.
5. Make sure that ICD2 is configured properly.
6. Program the hex code into the microcontroller on your demo board. It takes a few seconds for the programming to be completed. You will see the message "Programming Target. . ." at the left bottom of the MPLAB IDE window. When programming is done, this message will disappear.
7. Open the Program Memory window by selecting *View>Program Memory*. Scroll the program memory so that program lines 44 to 60 can be seen on the window as shown in Figure L3.1. Look at the column with the title Opcode. The numbers from 1 to 30 can be seen starting from line 46. How does the Program Memory window store the data?

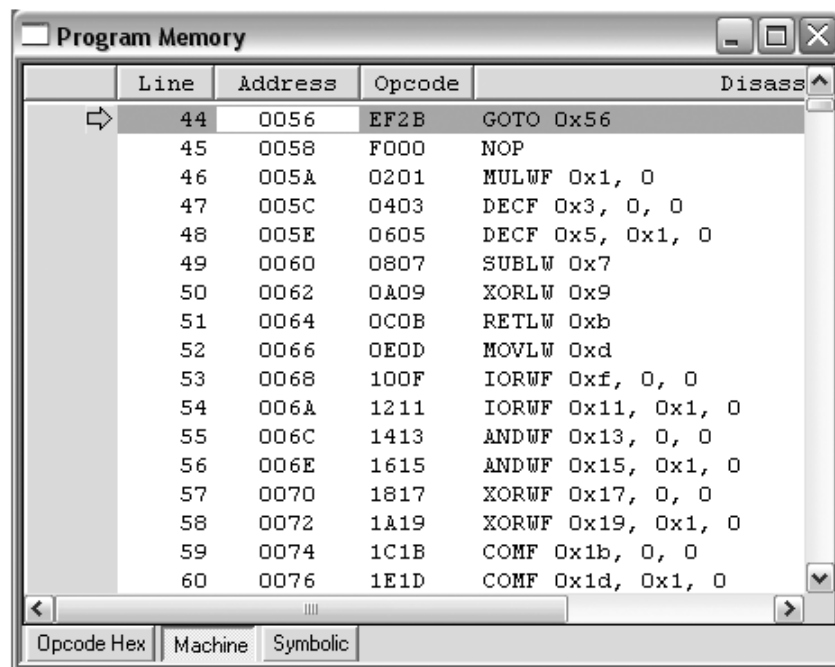
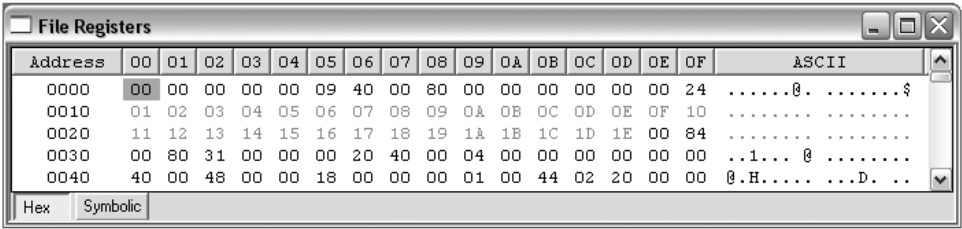


Figure L3.1 ■ Snap shot of program memory window for lab exercise L3.2 (reprint with permission of Microchip)

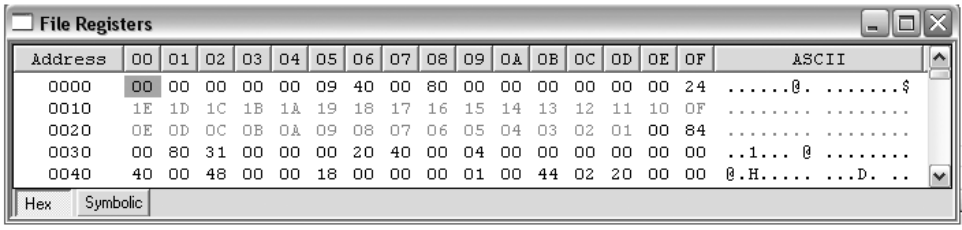
- 8. Open the File Register window. Resize and scroll so that the data memory locations 0x00 to 0x40 can be seen on the screen.
- 9. Set a breakpoint at line 30 and run the program until it halts at the breakpoint. The contents of the File Registers window should change to that in Figure L3.2.
- 10. Set a new breakpoint at line 42 (the last **nop** instruction) and rerun the program. The contents of the File Registers window should change to that in Figure L3.3.



The screenshot shows the 'File Registers' window with a table of memory addresses and their contents. The 'Hex' tab is selected. The table has columns for Address, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, and ASCII. The data is as follows:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	00	00	00	00	09	40	00	80	00	00	00	00	00	00	24@.\$
0010	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
0020	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	00	84
0030	00	80	31	00	00	00	20	40	00	04	00	00	00	00	00	00	..1... @
0040	40	00	48	00	00	18	00	00	00	01	00	44	02	20	00	00	@.H.....D. ..

Figure L3.2 ■ Contents of file registers at the first breakpoint (reprint with permission of Microchip)



The screenshot shows the 'File Registers' window with a table of memory addresses and their contents. The 'Hex' tab is selected. The data is as follows:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	00	00	00	00	09	40	00	80	00	00	00	00	00	00	24@.\$
0010	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	0F
0020	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00	84
0030	00	80	31	00	00	00	20	40	00	04	00	00	00	00	00	00	..1... @
0040	40	00	48	00	00	18	00	00	00	01	00	44	02	20	00	00	@.H.....D. ..

Figure L3.3 ■ File register contents when program halts at the second breakpoint (reprint with permission of Microchip)

What operation is performed by this program?

L3.3 Write a program to compute the average of an array of 32 8-bit elements. The array is stored immediately after your program with the name of **array1**. Store the sum (two bytes) and average in data memory 0x00–0x01 and 0x02, respectively. Use the simulator to simulate the program. Hint: Divide-by-32 can be implemented by shifting to the right by five places.

L3.4 Write a program to count the number of elements that are greater than 30 in an array of n 8-bit elements using the **for i = n1 to n2 do** looping construct. Debug the program using the ICD2 and the demo board. Open a Watch window to view the result.