Lesson 4
Fun with W and F

Overview

Introduction	This lesson introduces the first few PIC instructions.
	The following topics are introduced in this lesson:

Writing Programs

Introduction

As we said way back in Lesson 1, we use an assembler to help translate mnemonics for the instructions and memory locations into the ones and zeroes that the processor needs to do its thing.

In this lesson, we will do a number of experiments using some of the more basic instructions in the PIC. These instructions manipulate the working register (W) and the file register (F).

Setting up the first project

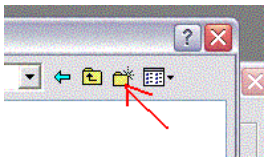
Before we can start to write, we need to have a project for the IDE.

Begin by starting the MPLab.

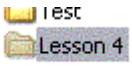
Select Project➔New… from the menu and a dialog box with 2 edit controls will appear. In the upper box, type "Lesson 4a" (without the quotes).

Click on the "Browse…" button on the lower right of the dialog.

Navigate to the "root of all projects" folder that you created in Lesson 3 and click on the "Create Folder" icon (a picture of a folder with a star in the upper right).



A new folder will appear named "New Folder" and the name will be highlighted, ready for editing. Type "Lesson 4" and then double-click on the folder icon.



Check that 'Lesson 4' appears in the top of the dialog then click on the 'Select' button.

Click on OK.

Continued on next page

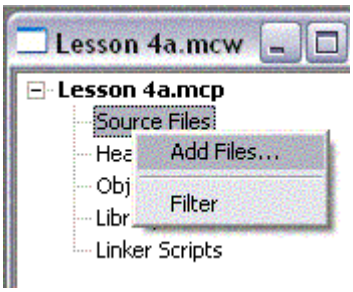
Writing Programs, Continued

Adding files to the project

OK, now we have a project, but it has nothing in it. We need to have at least one assembler source file to type in.

Select 'File➔New' from the menu. A new window will appear. Select File➔Save and type 'Lesson 4a.asm'. Click Save.

In the project window is a sub-window that lists the different types of files. Right-click 'Source Files' and select 'Add Files…':



A file open dialog will appear. Double-click Lesson 4a.asm. The name will be added to the Lesson 4a.mcw window and the title of the blank window will change from Untitled to the name of your file.

Also notice the asterisk in the title bar of the Lesson 4a.mcw window. This means that the project hasn't been saved. Select 'Project➔Save Project' from the main menu.

We now have an empty project, ready for us to go to work.

Our first Program

Introduction

The project consists of a number of files. If you look in the Lesson 4 folder with Windows Explorer, you will see 3 files at this point. The Lesson 4a.asm file is the one we are currently interested in. The Lesson 4a.mcp file contains the actual project information, that is, what files make up this project. The Lesson 4a.mcw file is the 'Workspace' file. This file remembers what windows are open in our workspace. In the future, if you double-click on the mcw file, the MPLab will open with all the windows where you last left them.

Basic stuff

There are a few things you need in every program. Might as well get them in the file now.

When entering data into the MPLab assembler, there are 3 columns of interest. The columns are separated by whitespace (tabs and spaces). How much whitespace is entirely up to us. We can use a single space, or 10 tabs, really doesn't matter to the assembler. Personally, I like to use 2 tabs. This makes the columns line up without thinking much about it, and it allows a reasonable length for identifiers.

The first column is anything that starts in column 1. The assembler assumes that this is a ***label*** that we will reference somewhere in our program.

The second column contains the ***opcode***. This is the instruction that tells the PIC what we want it to do.

The third column is the ***operand***. This is the thing we want the PIC to do something to.

Besides instructions, there can be assembler directives. These don't end up as instructions in the PIC, instead, they tell the assembler things we want it to know.

We need 3 directives in any program:

```
processor   16f84a
include    <p16f83a.inc>
end
```

It's also a good idea to include the configuration word. We will talk about this one in more detail, but for now, type in the following:

```
<tab><tab>processor<tab>16f84a<enter>
<tab><tab>include<tab><tab><p16f84a.inc><enter>
<tab><tab>__config<tab>_HS_OSC & _WDT_OFF & _PWRTE_ON<enter>
<tab><tab>end<enter>
```

Continued on next page

Our first Program, Continued

Basic Stuff (continued)	The `processor` directive tells the assembler which type of PIC we are using. The `include` directive tells the assembler to include a file which contains definitions for a number of symbols relevant to that processor. The `__config` tells the processor that we will be using a crystal (`_HS_OSC`), we want the watchdog timer turned off (`_WDT_OFF`) and we want the power-up timer enabled (`_PWRTE_ON`). Select 'File➔Save' to save your work.
Assembling the program	OK, so far, the program doesn't do anything … there are no instructions. But we can check for typos by assembling the program. From the main menu, select 'Project➔Build All'. We will get a new window with a bunch of junk, but the last line should say: BUILD SUCCEEDED

Adding some instructions

Introduction

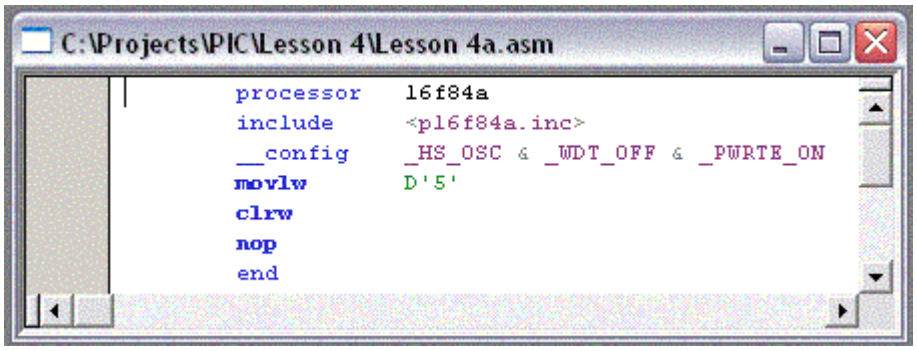
Now that we have the basic skeleton for all programs, we can go ahead and work over the actual instructions for our program. In this lesson, we aren't going to do a lot useful. Our point here is to get to understand how some of the basic instructions work.

At this point, you may find it useful to find the file for the quick reference card, and print out the page titled '14-Bit Core Instruction Set'. Throughout this course we will be referring to this page. There are other parts of the card that are interesting, but this particular page is the one that will get dog-eared.

Our first instructions

We are going to begin with the simplest of instructions. When we enter instructions, we place them after the `__config` directive and before the `end` directive. For our experiments right now, we need a `nop` instruction right before end. This is the simplest of instructions, it does nothing!

Let's add 2 more instructions before our `nop`, a `movlw D'5'` and a `CLRW` instruction. These instructions move the number 5 into the W register, then clear it. Our program should now look like this:



Assembling the program

As before, select 'Project➔Build All'. With a little luck, you should get the friendly 'BUILD SUCCEEDED'. You can also select the Build All toolbar button:

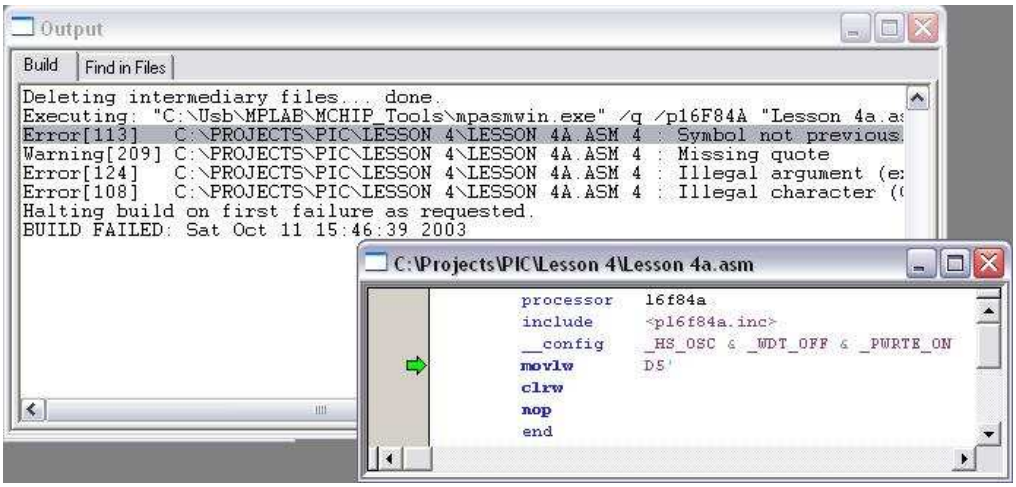
or, simply hold down the Ctrl key and press F10.

Continued on next page

Adding some instructions, Continued

Suppose there was an error

If we had a typo, this can cause the assembler to get confused and give us a lot of error messages. Don't be concerned if you see a long list of messages. If we left off one of the quotes around the 5 we might see something like this:



Double-clicking the error message will cause MPLab to put a green arrow left of the offending line. It's always good to look at the first error first. The remaining messages could be a result of the first. In this case, they are all on the same line, but sometimes an error on one line causes another line to be in error, so correct the first error first.

Let's see what happens

Once we get the program to assemble correctly, we want to see whether it does what we expect.

From the main menu, select 'Debugger➔Select Tool➔MPLAB SIM'. Now select 'Debugger➔Reset➔Processor Reset F6'

Notice at the bottom of the window is says 'pc:0' and 'W:0'. This says that the program counter is pointing at the first address in program memory, zero, and that the working register, W, contains a zero.

Select 'Debugger➔Step Into F7'. Several things happen. First, the green arrow moves down one line in our program. At the bottom of the window, it now says, pc:0x1 and W:0x5. The 0x business is a way of warning us that the numbers we are looking at are in hexadecimal. The program counter has incremented by one, as we would expect, and the W register contains a 5, which is what we told it to do with the `movlw D'5'` instruction.

Now press F7 (or select 'Debugger➔Step Into F7' again). The green arrow moves yet again, the bottom of the screen changes telling us that we have incremented the program counter one more time, and have cleared the W register, just like we told it to do.

Continued on next page

Adding some instructions, Continued

Some more playing with the simulator

Now add a few more lines so our program looks like this:

```
                processor    16f84a
                include      <p16f84a.inc>
                __config     _HS_OSC & _WDT_OFF & _PWRTE_ON
Spot1           equ          H'30'
                movlw        D'5'
                movwf        Spot1
                clrw
                clrf         Spot1
                nop
                end
```

Assemble the program, and select View➔File Registers. Arrange the windows so you can see both the program source and the file register window.

Select 'Debugger➔Clear Memory➔File Registers' and reset the processor (F6). Now as we press F7, there are several things to watch. On the first F7, besides the pc and w changing at the bottom of the screen as before, notice that location 0x02 in the file register also changed to a 0x01. This is because the low 8 bits of the program counter are mapped into location 0x02 of the file register.

The next time we press F7, besides 0x02 of the file register, 0x30 also changes. This is because we used that location to store our value `Spot1`. If we don't want to remember where we put things when we are debugging, we can click on the 'Symbolic' tab of the file register display. When we scroll down to 0x30 we can see the name, Spot1, on the right.

Press F7 again and our W register again goes to zero, and yet again and that zero gets stored in Spot1.

Let's do some Arithmetic

OK, so we've loaded a number into both the working register and the file registers. Now let's do a little something with those values.

Change our program yet again to look like this:

```
                processor    16f84a
                include            <p16f84a.inc>
                __config     _HS_OSC & _WDT_OFF & _PWRTE_ON
Spot1           equ          H'30'
Spot2           equ          H'31'
                movlw        D'5'
                movwf        Spot1
                movlw        D'2'
                addwf        Spot1,W
                movwf        Spot2
                movlw        D'3'
                subwf        Spot2,W
                movwf        Spot1
                clrw
                clrf         Spot1
                nop
                end
```

Continued on next page

Adding some instructions, Continued

Let's do some Arithmetic (continued)

Now as we step through the program, we will see us storing the 5 in Spot1 like before, but then we will load a 2 into the W register, and add it to Spot1, then store the result in Spot2. Next, we will move a 3 into the W register, subtract that from Spot2, and store the result in Spot1.

Notice the '`,W`' on the add and subtract instructions. These instructions can store the result either into the W register, or the original memory location.

Helping to understand our program

Introduction

So far, we've worried only about the specific instructions that make up the program. As we develop programs, they can get to be a little long. We need some aid in understanding the program, especially when we come back to it after being away a few days, or weeks.

Comments

The assembler allows us to put comments in our code. Whenever the assembler encounters a semicolon, everything after that is ignored. The assembler also allows us to have lines that are entirely blank, which can help us with readability.

The following assembly is exactly equivalent to what we had before: