Lesson 3
First MPLAB Project

Overview

Introduction

In this lesson, we will assemble a short MPLAB program to validate that the installation is working correctly.

In this section

The following is a list of topics in this section:

Project Folder

Introduction

Before we can start to write programs, it's handy to set up a folder where we can keep our project information.

Constraints on folders

As we experiment with the PIC we would like a way to keep our projects organized. I like to have a Projects folder for all my projects, so that the entire projects folder can be quickly and easily backed up.

It's also nice to have several categories of projects. It might be nice to have a PIC projects folder, a SPICE projects folder, and so on. Within each of those we may want to have several further categories.

However, as we play with various ham related software, we often find ourselves using software which wasn't developed with the latest and greatest of development software. Software developed by hams may not have been developed or tested with the ordinary user in mind. Indeed, even the MPLAB IDE has some constraints that, while perfectly acceptable for developers, might not be obvious to ordinary users.

When developing a folder structure for these kinds of projects, the following rules should be followed:

- The entire path length should be kept below 51 characters. Fewer characters will give you more flexibility in file names
- The entire path should consist of folders with names 8 characters or less
- The entire path should contain no blanks

Personally, my laptop has a folder tree like:

```
C:\Projects
      \PIC
            \Lesson1
            \Lesson2
            …
      \SPICE
            …
```

This allows me to keep files organized while avoiding long path names that could cause me problems with some software.

A Warning to XP users

In Windows XP, the default is for documents to be placed in 'My Documents'. In general, this will not work with MPLAB. 'My Documents' is preceded by C:\Documents and Settings*(username)*\. The result is that, depending on the length of your user name, you are left with only a few characters to use beneath My Documents.

Further, depending on your configuration, you may need to adjust the privileges on the folder you create. Be sure you have complete privileges to the project folder.

Setting Up a Project

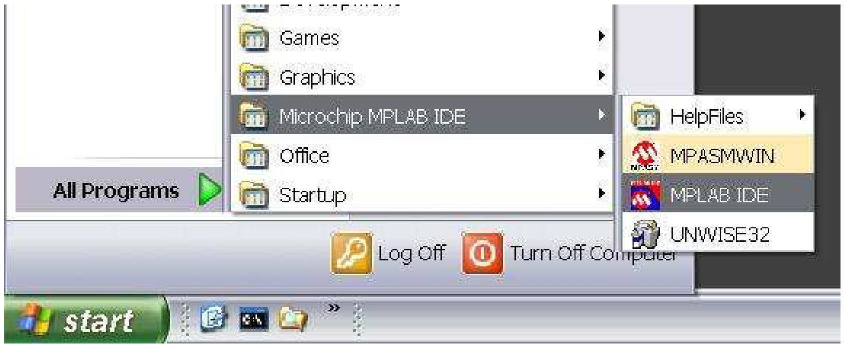
Introduction

Before we can assemble a program, we need to set up a ***Project***. This is an MPLAB construct that collects all the various files that are associated with a particular PIC program.

Launch MPLAB

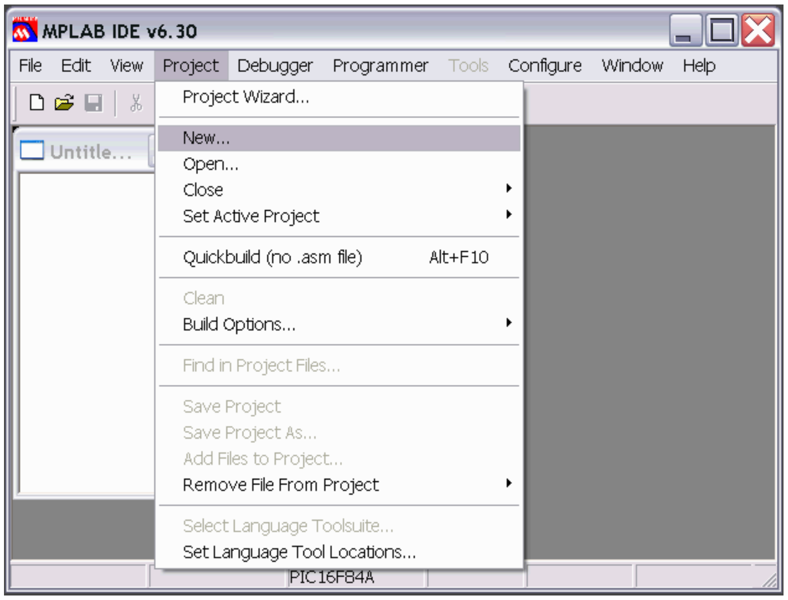
Begin by starting the MPLAB program. If you chose to have an icon installed on your desktop, double-click the bright red MPLAB icon:

Alternatively, select the IDE from the menu:



Create the Project

From the MPLAB menu, select ‘`Project->New...`’:



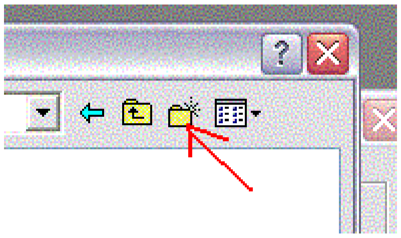
Continued on next page

Setting Up a Project, Continued

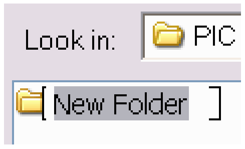
Create the Project (continued)

In the New Project dialog that appears, type 'Test' in the 'Project Name' box.

Click on the 'Browse…' button, and navigate to the folder you created as the '**Root of all Projects**' folder. Click on the New Folder icon:

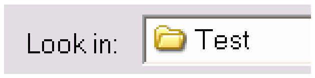


A new folder will appear, named, appropriately enough, New Folder:



Type 'Test' over the folder name.

Double click the icon so that Test appears in 'Look In:'



and click the 'Select' button.

Selecting the Processor

On the workspace menu, select `Configure->Select Device`…

On the drop down in the dialog box, select PIC16F84A. Click OK.

This sets the default chip type for the project. We will override this with a specific directive, but choosing it now will prevent an unnecessary message.

Continued on next page

Setting Up a Project, Continued

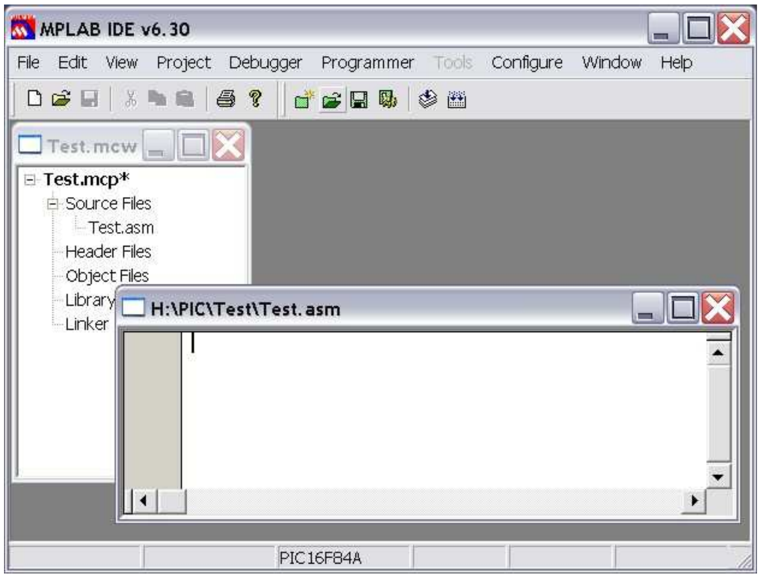
Adding a file to the project

Select ’`File->New`’. A blank window will appear.

Select ‘`File->Save As…`’, a file Save As dialog will appear. Be sure that the current folder is your Test folder. Type Test.asm into the ‘File name:’ box and click the ‘Save’ button.

Select ‘`Project->Add Files to Project…`’. Double-click Test.asm.

Test.asm will appear under `Source Files` in the Test.mcw window:



Select ‘`Project->Save Project’`. Notice that the asterisk next to Test.mcp disappears, indicating that there are no unsaved changes to the project.

Entering a Program

Introduction

Before we can test whether the assembler works, we need to have a program to assemble. In this section, we will enter a program.

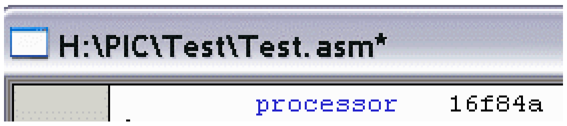
Entering Text

The assembler recognizes 3 columns. The columns are separated by any number and combination of spaces and tabs. Generally, it's more convenient to use tabs as this make it easier to keep the columns lined up. The assembler really doesn't care about that, but it's nice for us humans.

Type in:

```
        <tab><tab>processor<tab>16f84a<enter>
```

It should look like:



H:\PIC\Test\Test.asm*

```
processor    16f84a
```

The IDE helps us along by color coding things that it recognizes. In this case, it recognized the word `processor` as an assembler directive, and coded it blue. The processor directive informs the assembler that this code is targeted at the 16f84a. Now it knows things like the memory size, number of timers, etc.

Now add:

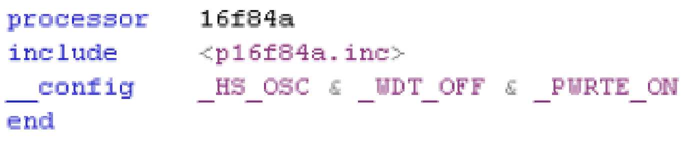
```
        <tab><tab>include<tab><tab><p16f84a.inc><enter>
```

This tells the assembler to include a file which contains definitions for the various assets of the 16f84a. This, in turn, allows us to do things like reference port A as PORTA rather than H'05'.

Finally, let's add:

```
        <tab><tab>__config<tab>_HS_OSC & _WDT_OFF & _PWRTE_ON<enter>
        <tab><tab>end<enter>
```

(Notice 2 underbars before config). Our program should look like:



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

Almost every program we write will include these four lines, or lines very similar. Click on the floppy disk icon (or select `File->Save`) to save the program.

Also notice that the assembler directives (and opcodes) are colored blue. If you type in a directive and it isn't blue, you probably have a typo.

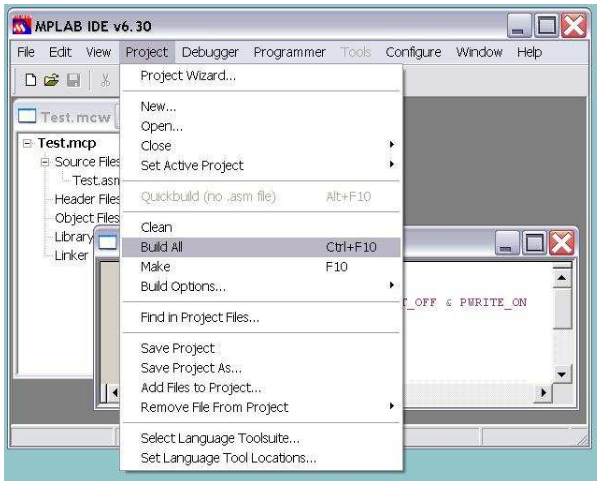
Assembling the Program

Introduction

Remember way back in Lesson 1 we said that an assembler converts text that we can (more or less) read into binary data for the computer? Well, now we are about to do that.

Assembling

To perform the assembly, we can select `Build All` from the `Project` menu:

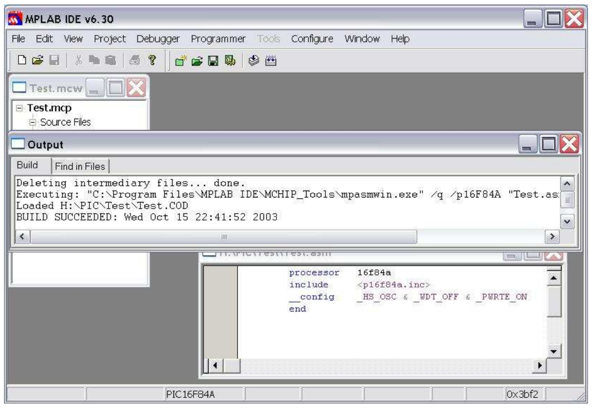


Or we can click on the Build All toolbar button:

Or we can press the F10 key while holding down the Ctrl key.

A new window will pop up with the assembly results. With a little luck, the last line will say:

```
BUILD SUCCEEDED
```



```
Deleting intermediary files... done.
Executing: "C:\Program Files\MPLAB IDE\MCHIP_Tools\mpasmwin.exe" /q /p16F84A "Test.as
Loaded H:\PIC\Test\Test.COD
BUILD SUCCEEDED: Wed Oct 15 22:41:52 2003
```

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

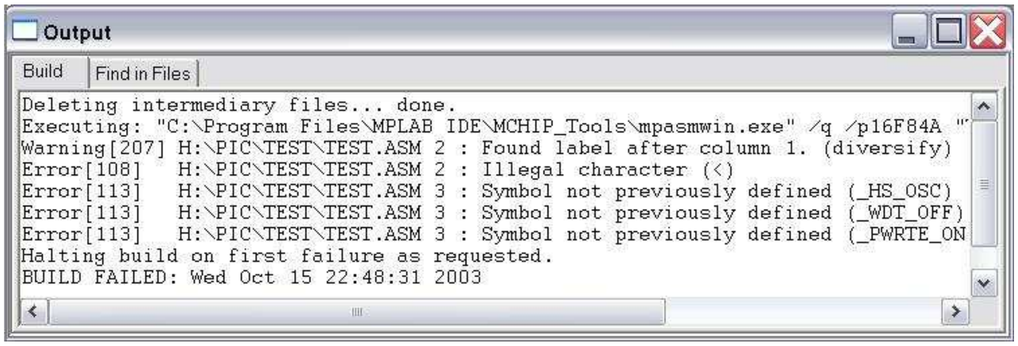
Continued on next page

Assembling the Program, Continued

What if I did something wrong?

Of course, there's a possibility that we had a typo or something. The assembler, of course, will tell us. Just to see this behavior, try changing the word `include` to something else, say, `diversify`.

Assemble the program and see:



```
Deleting intermediary files... done.
Executing: "C:\Program Files\MPLAB IDE\MCHIP_Tools\mpasmwin.exe" /q /p16F84A "
Warning[207] H:\PIC\TEST\TEST.ASM 2 : Found label after column 1. (diversify)
Error[108]   H:\PIC\TEST\TEST.ASM 2 : Illegal character (<)
Error[113]   H:\PIC\TEST\TEST.ASM 3 : Symbol not previously defined (_HS_OSC)
Error[113]   H:\PIC\TEST\TEST.ASM 3 : Symbol not previously defined (_WDT_OFF)
Error[113]   H:\PIC\TEST\TEST.ASM 3 : Symbol not previously defined (_PWRTE_ON
Halting build on first failure as requested.
BUILD FAILED: Wed Oct 15 22:48:31 2003
```

Wow, there's a lot of stuff there.

First of all, once the assembler sees one error, it can get pretty confused. In this case, it wasn't even an error at all, just a warning 'Found label after column 1.' What's it thinking?

Well, sometimes you need to get yourself into the assembler's head. It found the word 'diversify', and it knew it was neither an instruction nor a directive. If it's neither of those, it *must* be a label. But it didn't start in column one like all good labels should, so it warned us about that typo.

Well now, the next thing after a label should be an instruction. But the '<' character isn't allowed in an instruction, so it flags it as an illegal character. Now things just go from bad to worse.

Whenever you are debugging a program, always start with the first message first. There's a good chance that the later messages, even if they are on different lines, are a result of the first.

In this case, the next 3 errors are on line 3. The 3 symbols were never defined! Well, no, because we never read the include file since the assembler didn't know when we said diversify we really meant include.

So What Did That Mean

Introduction	We wrote and assembled a little program with four lines, but we never really explained what most of those lines meant. We'll skip `processor`, because we already talked about that one.
include	The include directive tells the assembler to read another file as if it were typed into the current file. Within the directory we installed MPLAB, there is a folder containing a number of these files, and that is where the assembler looks first. The most common use for an include file is to describe a number of constants. In this case, there are a pile of constants defined in p16f84a.inc, and they are used frequently. They include, as we mentioned earlier, things like the addresses of some of the special registers and the values of various bits in the configuration word (see below) and other special registers.
end	The end directive is pretty simple. It simply marks the end of the source file.
__config	The __config directive is the most interesting of those we have entered so far. The PIC has a configuration word that sets various behaviors, and the __config directive sets the value for that configuration word. Each bit in the configuration word has a special meaning. The '&' character between the various symbols we entered does a bitwise AND operation between the various symbols. The symbols are defined such that we can AND them in order to set the right combination of features into the configuration word. `_HS_OSC` set the bits to use the high speed crystal oscillator. Other choices were `_XT_OSC` (crystal oscillator), `_RC_OSC` (RC oscillator), and `_LP_OSC` (low power crystal oscillator). `_WDT_OFF` set the watchdog timer off. The processor contains a timer that can interrupt us every so often, or cause us to wake up if we have put the processor to sleep. Normally we don't use this feature, but if we did, we would set this value to `_WDT_ON`. `_PWRTE_ON` told the processor that we want the power up timer enabled. This basically prevents the PIC from executing any instructions until 72 milliseconds have gone by. This gives our external circuitry a chance to stabilize before our program actually starts. We could have said `_PWRTE_OFF` if we wanted to be Johnny-on-the-spot when the power came up. The final thing we may have done in the configuration word is to set `_CP_ON`. This would prevent our code from being read back out of the device. This security feature is generally not used in amateur applications since it causes a number of other complications.

Wrap Up

Summary	In this lesson, we entered a small program into the IDE. The program actually did nothing, but it was a program. We assembled it, and even inserted an error so we could see the assembler's behavior when we got an error. Finally, we talked about what each of the four lines do.
Coming Up	In the next lesson, we will explore many of the PIC instructions, and start to get an understanding of how we use the file registers and the working register.