Week 2 lab activities:
developing for the PIC controller on the QL200 board and Simulator

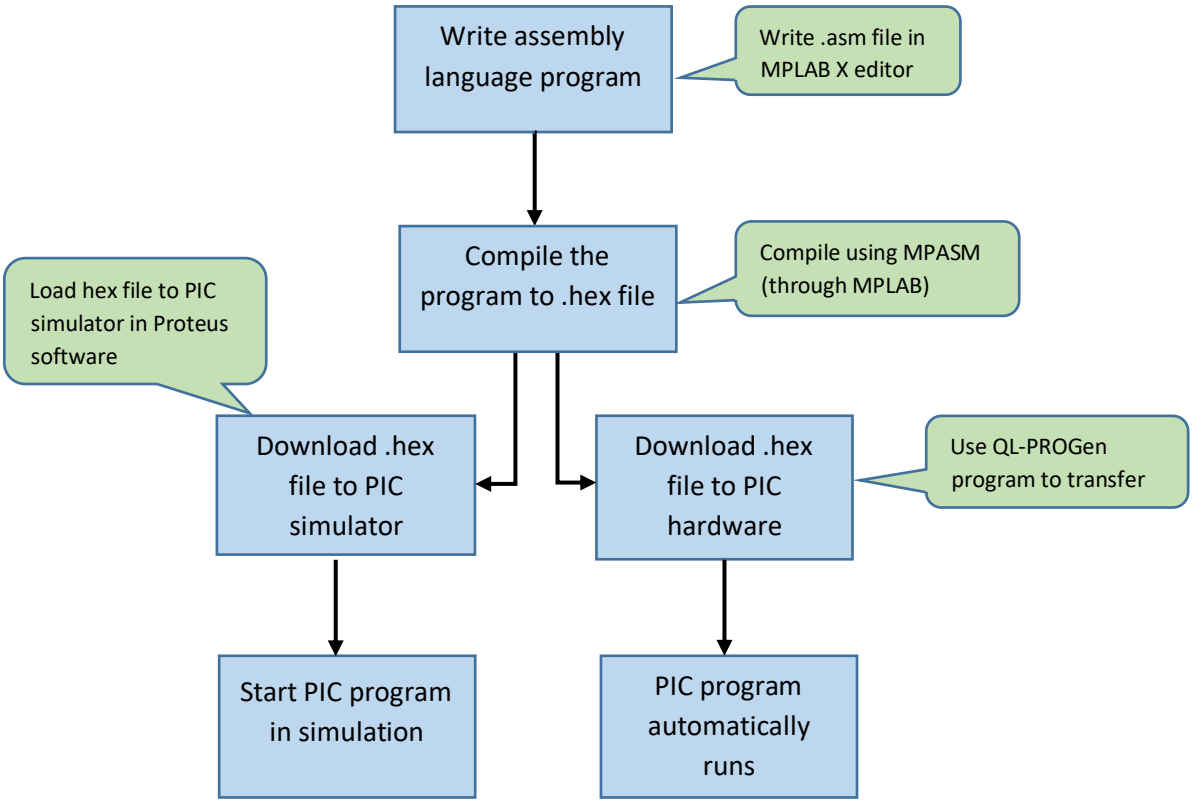
Introduction

This lab will give you basic knowledge of how to develop software to run on the PIC microcontroller. You will develop a program in assembly language on the PC and then deploy it to the PIC hardware on the QL200 development board; and to the simulator created in Proteus Virtual System Modelling (VSM).

There is a "Setup your pc for this module" guidance to help to install all the software needed for this module on Canvas/Modules/Week1.

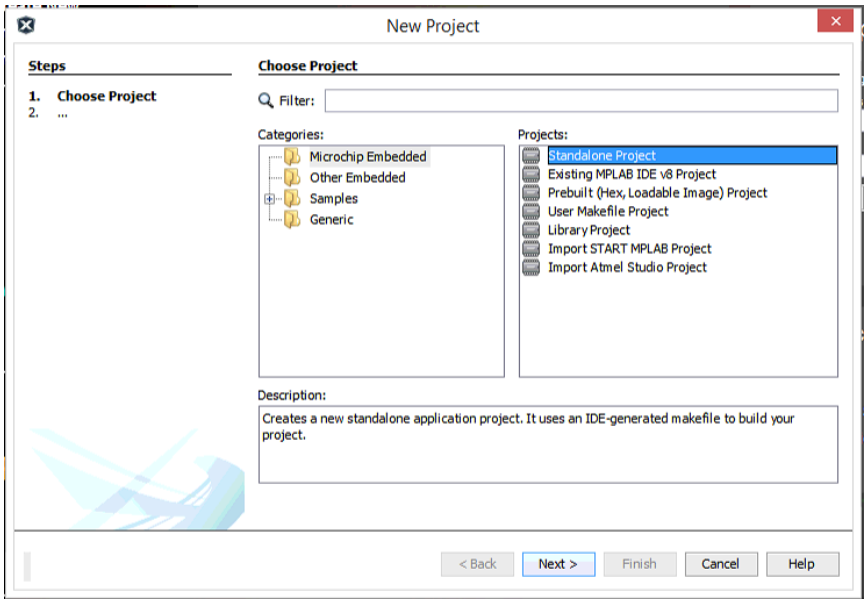
Development process

Programs for the PIC microcontroller are created in assembly language and compiled into machine code before being stored in the non-volatile memory of the device so they can be run. We will use two tools to do this. MPLAB X IDE is the integrated development environment produced by Microchip, the manufacturers of the PICmicro series of devices. You will use MPLAB X IDE to edit our source code and compile it to a .**hex** file. This hex file will then be transferred to either hardware PIC or the PIC simulator to run. In hardware PIC hardware case, it is transferred over the USB cable to the PICmicro device on the QL200 development board using the QL-PROG software. In PIC simulator case, you will need Proteus software to open the QL200 simulator file, then load the .hex file to the PIC device. The diagram below summarises this process.

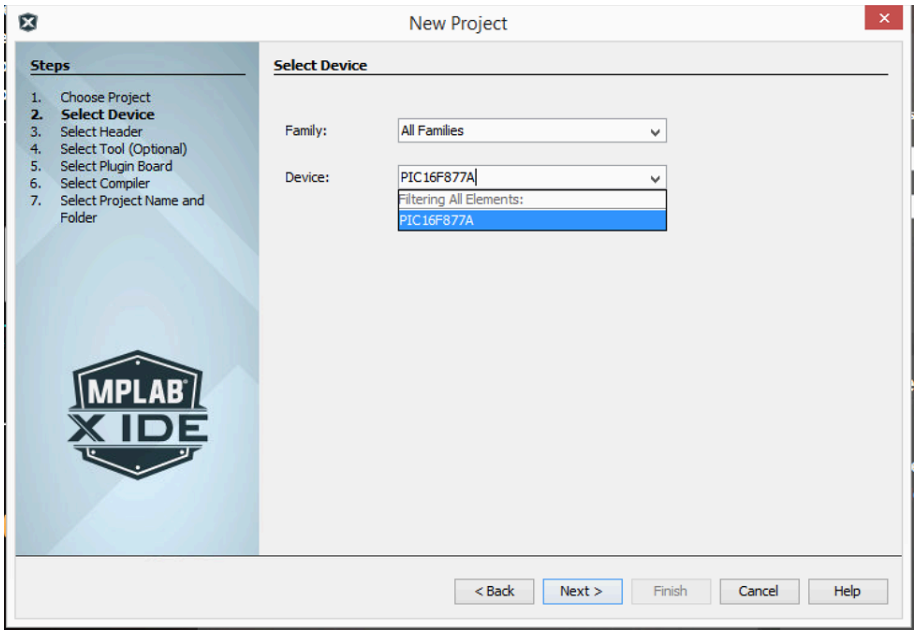


Step 1 – creating the software

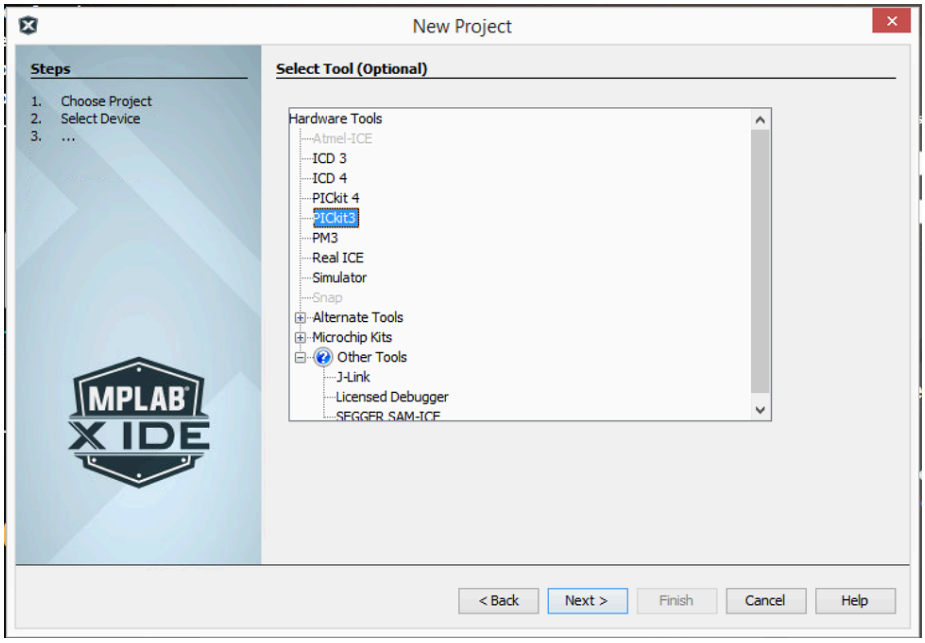
Run the MPLAB X IDE software by pressing the Windows key and searching for MPLAB X IDE. The first thing to do is to create a new project, using the Project Wizard (on the Project menu). Start the wizard by selecting **File>New Project**. Select **Standalone** Project from the dialog:



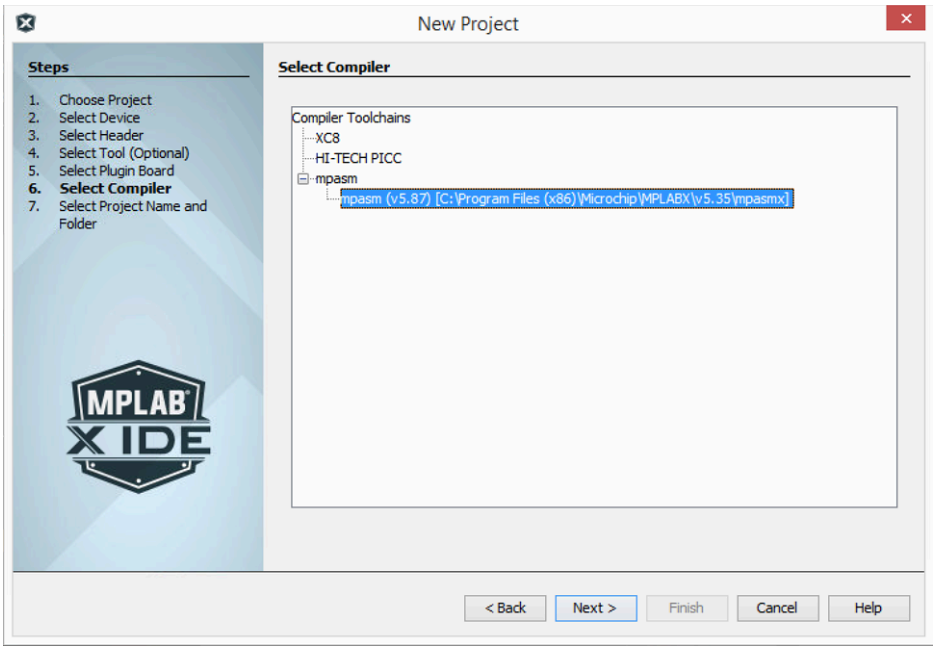
You are creating a standalone project, so select this and click **Next>**. The new project dialogue now wants to know which processor you are using. You need to select the PIC16F877A device as the target:



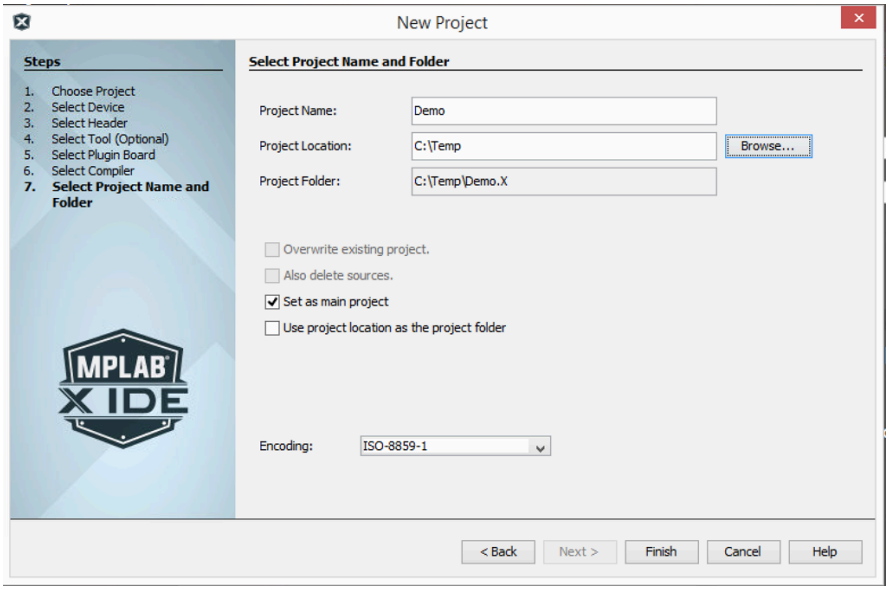
This is the chip for which you will be developing code. It's the large chip on the QL200 board. Click **Next>** and you'll see a dialog like this:



It doesn't really matter which of these tools you select, as we will be just creating an assembler file that will be loaded into the device. But I select **PICkit3** and it works for me. You do the same and click **Next>**

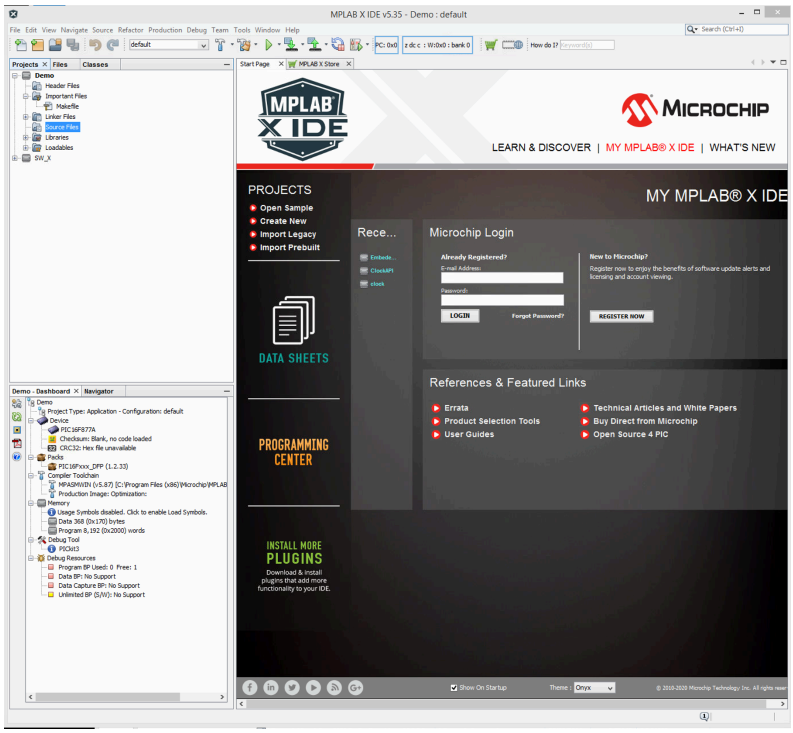


Now you need to select the compiler you are going to work with. Later we will be writing C programs but to start with we will be using the assembler. Select it as shown and click **Next>**.

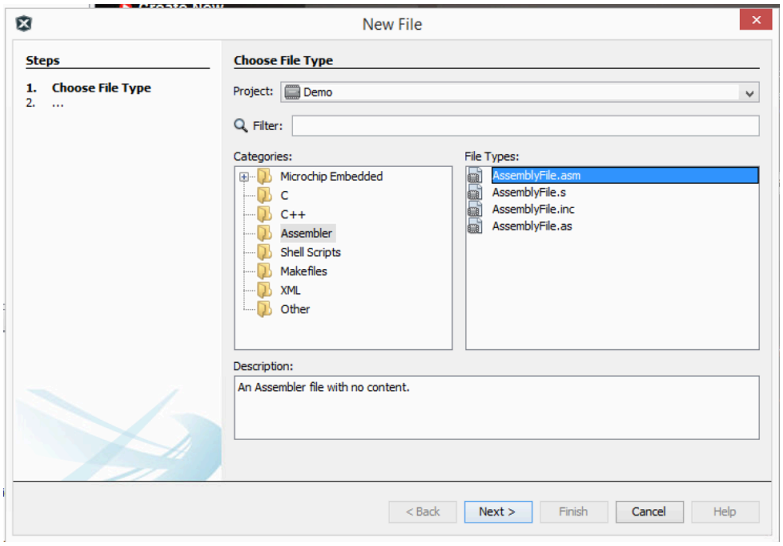


Now you need to tell MPLABX where the project is going to be stored. **The program does not like network drives, so you will have to put the project on drive C: in the Temp folder (and remember to copy it afterwards) or put it on a memory key.**

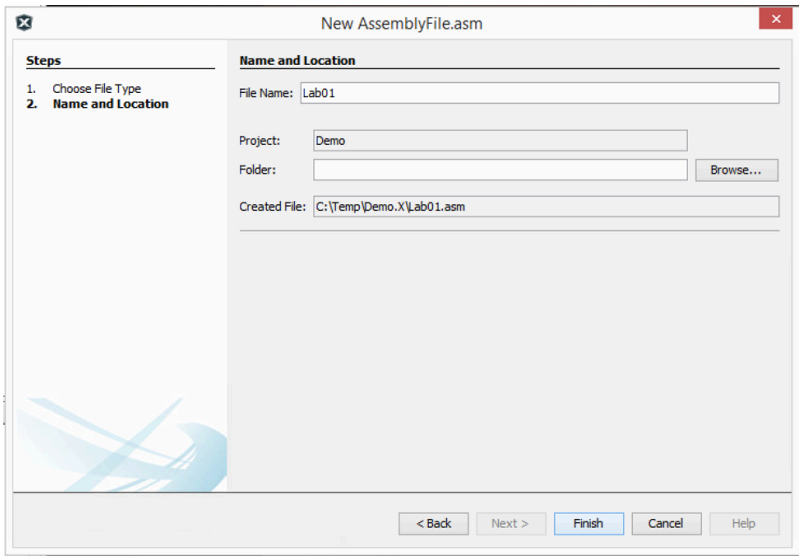
Select a sensible folder and click **Next>**



You are now in MPLAB X IDE with your new project. The next thing you need to do is to add a source file. Right click **Source Files** in the file browser on the right and select **New>Others…** from the menu that appears. Then select **AssemblyFile** from the context menu that appears. If there is no assembly file present, select **Other** from this menu and then find Assembler from the New File dialog.



Select the file type as shown and click **Next>**



Now give the file a name and click **Finish**.

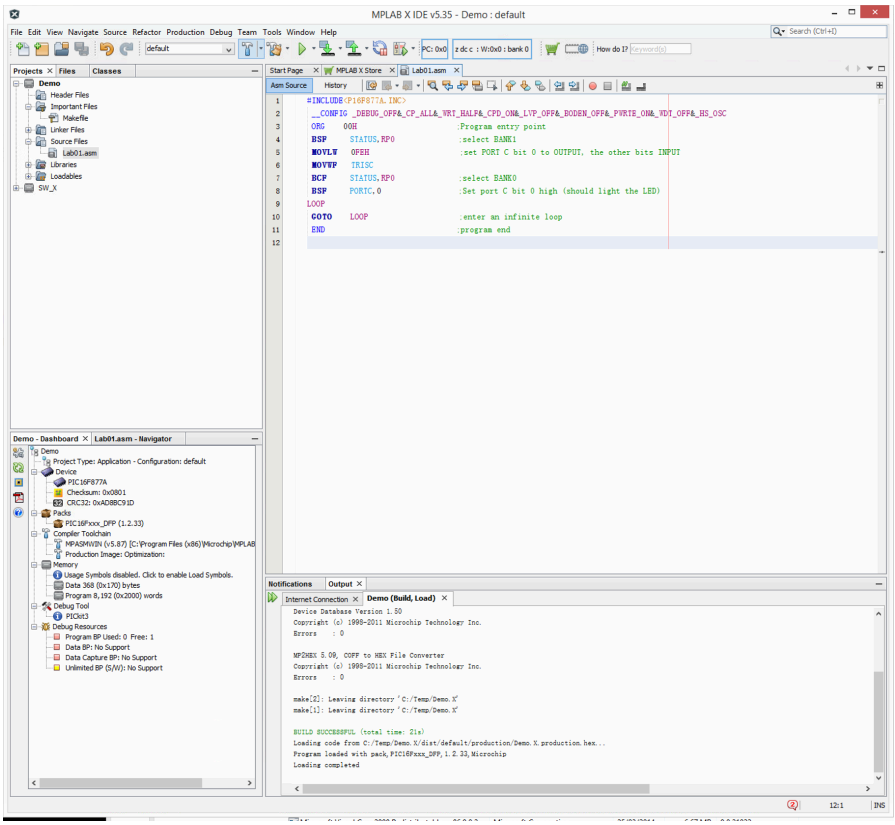
Enter the following code into your lab01.asm file:

```
#INCLUDE<P16F877A.INC>
 __CONFIG
_DEBUG_OFF&_CP_ALL&_WRT_HALF&_CPD_ON&_LVP_OFF&_BODEN_OFF&_PWRTE_ON&_WDT_OFF&_HS_OSC
 ORG     00H                     ;Program entry point
 BSF     STATUS,RP0              ;select BANK1
 MOVLW   0FEH                    ;set PORT C bit 0 to OUTPUT, the other bits INPUT
 MOVWF   TRISC
 BCF     STATUS,RP0              ;select BANK0
 BSF     PORTC,0                 ;Set port C bit 0 high (should light the LED)
LOOP
 GOTO    LOOP                    ;enter an infinite loop
```

```
END                                 ;program end
```

(Note that the __CONFIG should be on the same line as the following long line). Don't worry about entering the text after the semicolons – those are comments.

Save the file. Don't worry about what it does yet. Now, you should be able to compile this project. Press the picture of a Hammer on the toolbar. All being well, you will get an output screen like this:

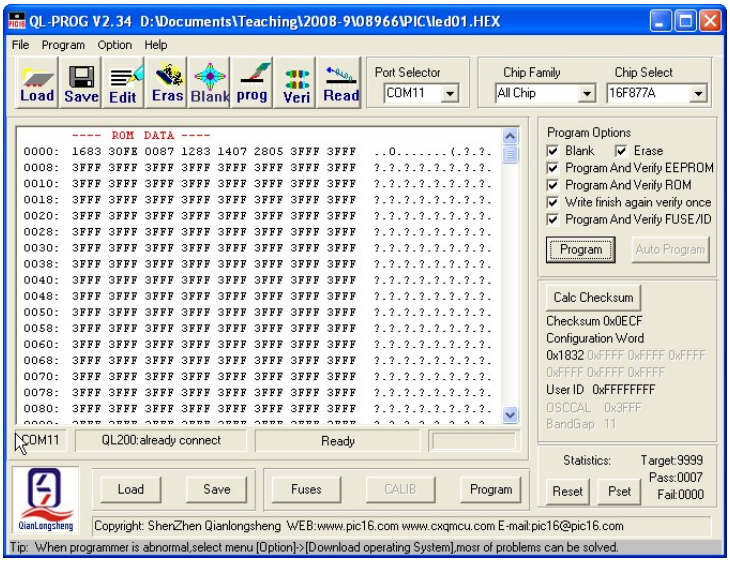


If it doesn't say 'BUILD SUCCESSFUL' at the end, then check everything and don't proceed until it does. Once the project has successfully built, you can deploy it to the hardware. (The program above is different – see if you can work out what it does.)

Step 2a – Deploying the code to the PICmicro (QL200 board)

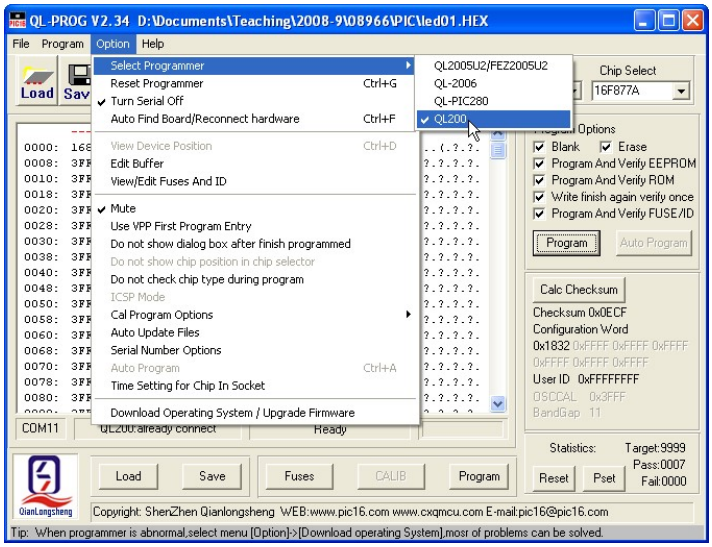
First, make sure that the QL200 board is connected to the PC via the USB lead. Then, make sure it's switched on. If it is, the 'Power' LED in the top left hand corner of the board will be on. If it isn't, switch it on with the push switch near the LED.

Now, run the QL_PROG software. You'll be presented with something like this:



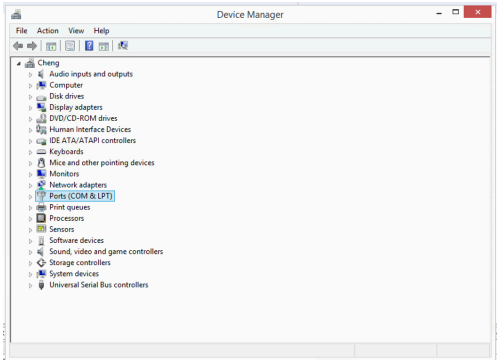
Because of the configuration of the systems, for the programmer to work it needs to be running in Administrator mode. You should all be granted with temporary administrator right during working hours of the days that have time tabled sessions. (Be mindful not to change any system files.) Please inform the demonstrator or teaching staffs if you cannot launch the program as administrator.

In the 'Chip Select' drop-down box, make sure that '16F877A' is selected. On the 'Option' menu, make sure that QL200 programmer is selected:

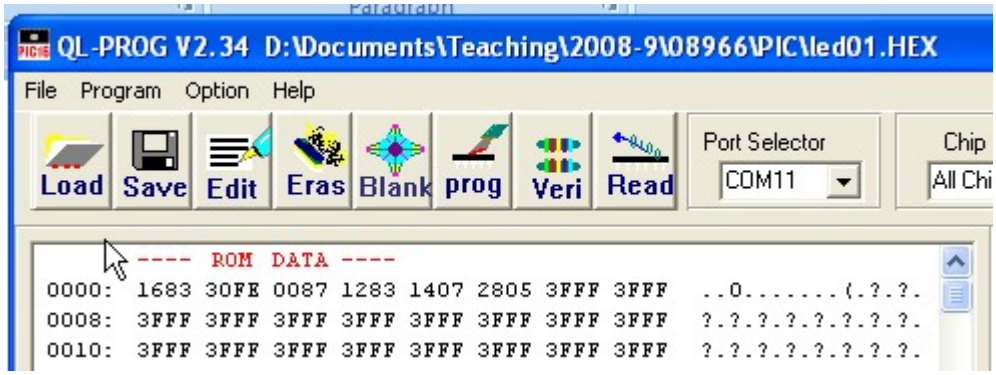


Assuming that these are OK, then the software should automatically detect the QL200 board. If it doesn't, press ctrl+F and the system will try to establish contact. If this fails, you may have to set the

COM port by hand. The COM port that is normally selected is COM3 or COM4. Or open device manager to see which serial port the QL200 has used.

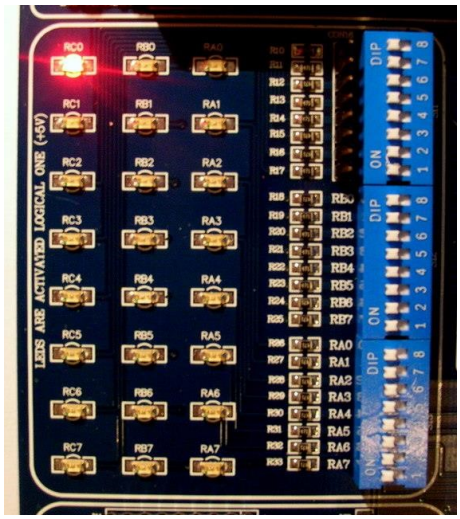


The QL-PROG software transfers the compiled program created by MPASM into the PICmicro's internal storage. To do this, press the 'Load' button, and select the 'hex' file from your MPASm project directory. You can find out where this is by checking the message at the end of the output from the build command. When this has loaded into QL-PROG, you should see this:



The hex code represents the compiled version of your program. At present it is loaded into QLPROG, but has not yet been programmed into the PICmicro. Press then 'prog' button, and after a few seconds you should get a 'programming complete' message. The program has now been loaded into tht PICmicro, and will remain there even after power is removed.

Has the program worked? Check the bank of LEDs on the left of the board. If the top left one is lit, like in the picture below, all is well. If not, check that the switches are in the positions shown.



Yes, that's all that program does. It lights a single LED, and leaves it on.

Step 2b – Deploying the code to the PIC simulator (Using Proteus VSM)

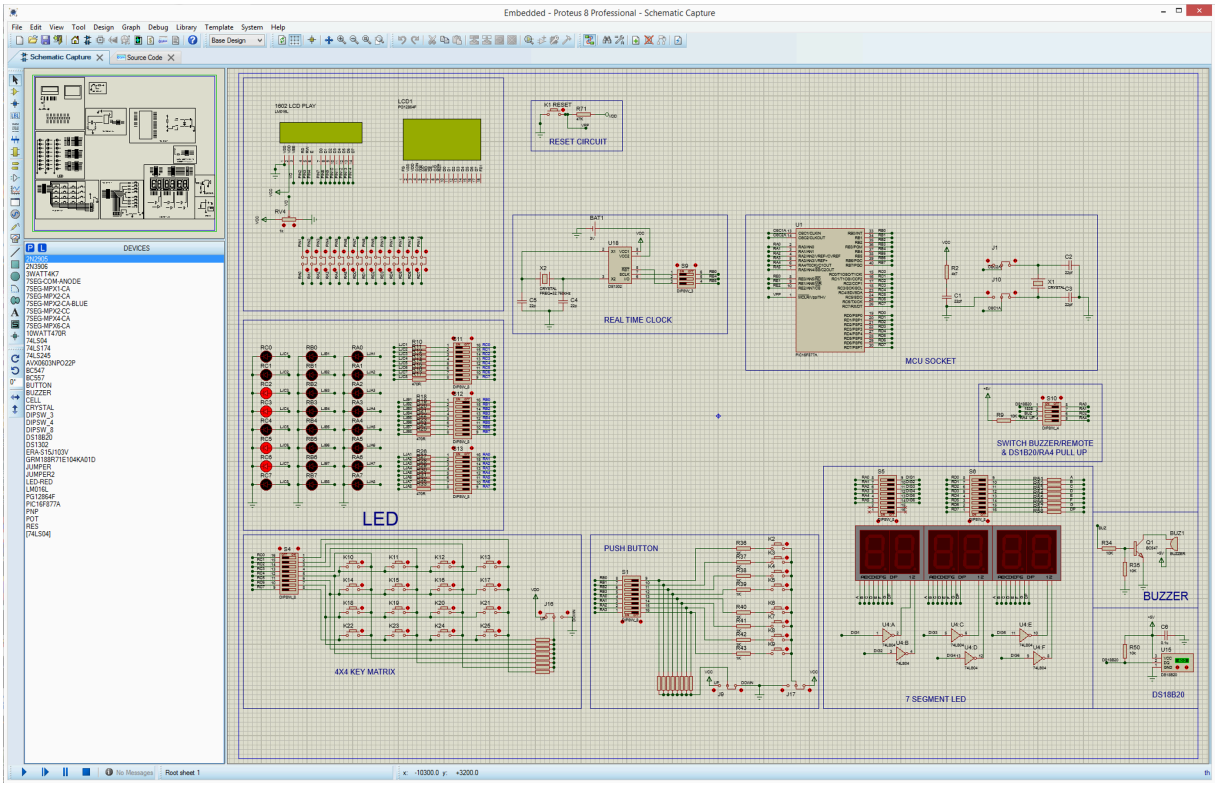
Before you start, download the simulator file QL200_Simulator from Canvas site and save it to your home drive.

Now run the Proteus 8 Professional software by pressing the Windows key and searching for Proteus 8 Professional, or look for this 

icon on the desktop. This is a commercial software, we have bought licenses for PIC processor and VSM. When prompted for user name and password, use the information provided to you on Canvas home page. You can download the software on your own PC and use the same credential. Be aware that we have one licence for each student, so don't try to access using multiple devices.

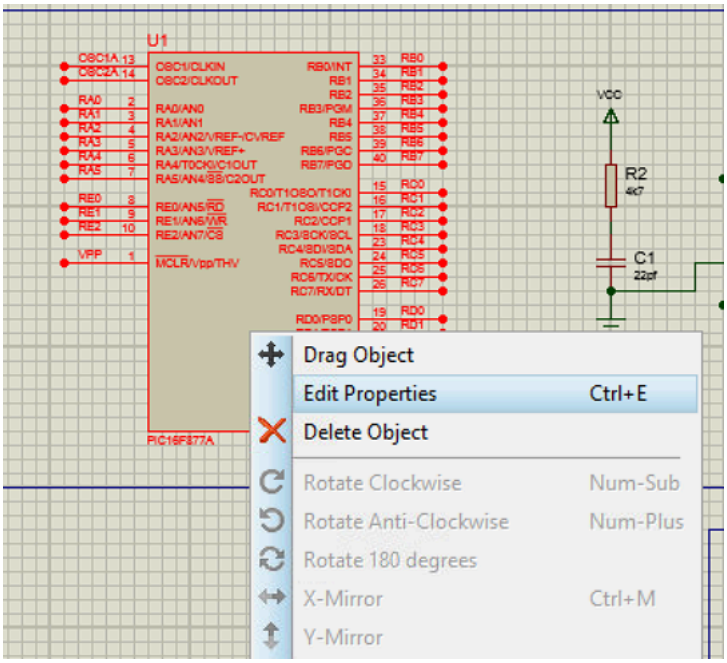
Once Proteus is open, chose **File>Open project** from the main menu. in the opened dialog, locate the QL200_Simulator file you've just downloaded, and click "Open" button.

Then you should be presented with a window similar as below (it might be slightly different as I am updating the design over time).

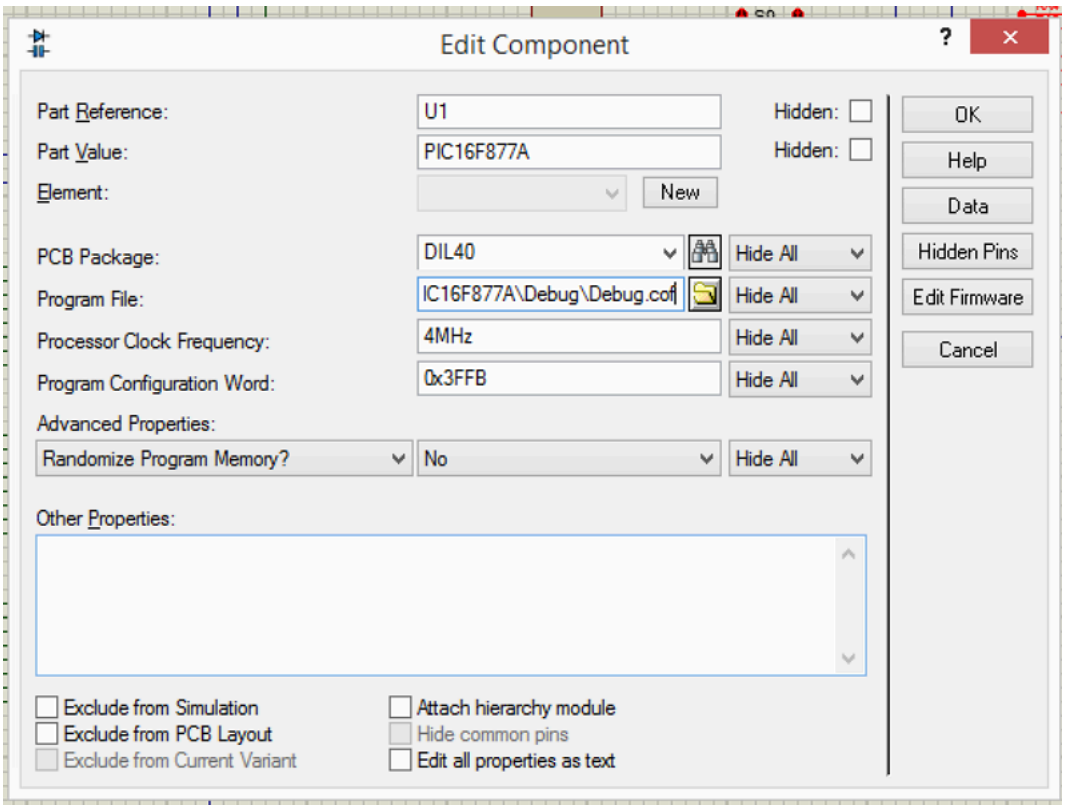


This is the QL200 schematic circuit diagram I created, which have the same components and connections as the QL200 hardware you used in Step 2a.

Now, locate the MCU Socket block, and right click PIC16F877A which is U1 in the diagram. Select **Edit Properties** from the pop up menu.



In the opened properties dialog, change the Processor clock frequency to 4MHz if it's a different value.



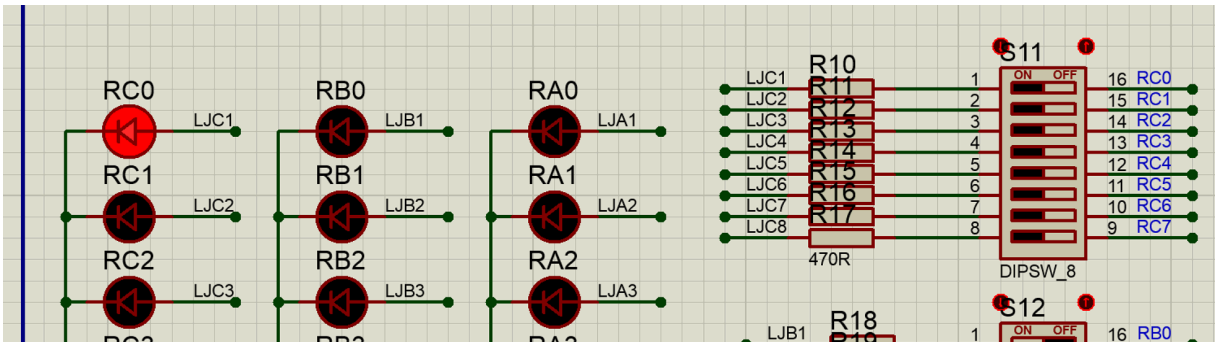
Click the 

open folder icon, and locate the hex file from your MPLAB X IDE MPASm project directory. If you have followed the same path in previous steps, it should be < C:\Temp\Demo.X\dist\default\production\Demo.X.production.hex >. You can find out where this is by checking the message at the end of the output from the build command. When this is done, click OK button on the top right of the dialog window.

Now the program has been loaded and we're ready to rock and roll.

To run the simulation, either click the little triangle 

on the bottom left corner, or F12. The square icon is to stop simulation. If the top left LED (shown as RC0) is lit, like in the picture below, all is well. If not, check that the switches are in the positions shown. (You can click the individual switch to switch it on or off, or you can click the little up and down arrows near the switch to switch all switches to on or off).



Yes, that's all that program does. It lights a single LED, and leaves it on. It should be the same as you did in Step 2a.

There are abundant online help files on how to use Proteus, simply search the functions you need using your favourite search engine. Here are some tips,

- Click the mouse middle button to move the view; click it one more time to exit the mode.
- Scroll the mouse middle button to zoom in or out.
- Click  icon to zoom to view the entire sheet, and reposition to the centre.
- Don't delete or change any connections of the simulator unless you know what you're doing; if you accidentally changed anything, download a fresh copy of the project and open the new one.
- You may create your own simulation circuit diagrams, but that's out of scope of this module.
- The system clock of the simulator runs slower than the hardware board.(in my very old PC, this ratio is about 30~40 times). If you might need to adjust the delay.

The initial program

Now let's have a look at how it does it.

The program you have just written looked like this:

```
#INCLUDE<P16F877A.INC>
__CONFIG
_DEBUG_OFF&_CP_ALL&_WRT_HALF&_CPD_ON&_LVP_OFF&_BODEN_OFF&_PWRTE_ON&_WDT_OFF&_HS_OSC
 ORG    00H                      ;Program entry point
 BSF     STATUS,RP0              ;select BANK1
 MOVLW   0FEH                    ;set PORT C bit 0 to OUTPUT, the other bits INPUT
 MOVWF   TRISC
 BCF     STATUS,RP0              ;select BANK0
 BSF     PORTC,0                 ;Set port C bit 0 high (should light the LED)
LOOP
 GOTO    LOOP                    ;enter an infinite loop
END                             ;program end
```

At the beginning of the file is a #INCLUDE line, which simply includes a standard file defining a number of symbols to make life easier when programming this particular processor. You don't need to include this file, but doing so allows you to use meaningful names like TRISC instead of the numbers they represent (0x0087, if you are interested).

The __CONFIG line is a necessary directive which sets up a number of properties of the PIC. It's important to include this line, and to leave it unchanged. PICs are designed to be reprogrammable, but can also have a single program permanently burned into them, preventing reprogramming. The config line is where that would be specified. For now, just include the line exactly as above in all your programs.

The program proper begins with the line 'ORG 00H'. All this line does is to set the memory location of the start of the program. In this case, we are telling the assembler to put the program at the top of memory, where it will run automatically when the system starts up.

The next line 'BSF STATUS, RP0' is the first actual instruction in the program. The BSF instruction sets a bit in a particular register. In this case, the register is the STATUS register, and the bit being set is RP0. STATUS and RP0 are symbols defined in the included file P16F877A.INC, and are simply readable ways of writing the numbers 0x03 and 0x05 respectively. We could have written the instruction BSF 3,5 instead. The purpose of the instruction is to set the RP0 bit in the STATUS register, which has the effect of selecting bank 1 of registers for subsequent operations. We need to do this, because we are going to want to use the TRISC register, which is in bank 1.

MOVLW 0FEH means 'load the literal value 0xFE into the W register'. The W or working register (often called the accumulator' in other microporcessors) is the register commonly used for interim results of calculations, or very temporary storage. Think of it as being like the display on a simple calculator. It holds one value until it can be used for something.

MOVWF TRISC means 'write the value in the W register in to the TRISC register'. This takes the value that we have just loaded into W and writes it to the special purpose register TRISC. TRISC is not just a memory location, but controls the direction of the signal wires of port C of the chip. Each bit represents a single I/O line. If the bit is high, the line is an input to the chip. If it is low, the line is an output. Setting the value to 0xFE, as we have done, means that port C bit 0 is an output, and the other seven bits are inputs.

BCF STATUS,RP0 clears the status bit that we set earlier. This has the effect of switching back to register bank 0. We need to do this because we want to write data to port C, which is in Bank 0.

BSF PORTC,0 is another bit-setting instruction which makes bit 0 of PORTC high. Because we have defined bit0 of TRISC low, PORTC bit 0 represents an output from the chip, which means that the voltage on the wire connected to that pin on the chip is raised. As it happens, this wire is connected to a LED on the QL200 board, so LED RC0 lights up.

At this point, the program has reached its objective – we have lit a LED. Now, for safety's sake, we put the program in an infinite loop. If we don't, the processor will continue trying to sequentially execute instructions from memory – but we have not explicitly set the contents of memory beyond the end of our program, so it's just random data, or possibly the remains of the previous program.

The LOOP label is just that – a label. It acts simply as a referent for the assembler, so that subsequent instructions can use its memory address.

The final instruction, GOTO LOOP, simply causes the processor to transfer its instruction pointer to the memory address represented by LOOP, and hence to loop round indefinitely.

The END directive informs the compiler that this is the end of the code. It's not a PIC instruction, which is why we need the infinite loop to effectively terminate the program.

Program 2 – Counting in binary

Here's some new code:

```
#INCLUDE<P16F877A.INC>
__CONFIG
_DEBUG_OFF&_CP_ALL&_WRT_HALF&_CPD_ON&_LVP_OFF&_BODEN_OFF&_PWRTE_ON&_WDT_OFF&_HS_OSC

; Predefined values
outer_delay   equ   0ffh            inner_delay   equ   0afh

 org      00h
main
 clrf     PORTC
 banksel  TRISC
 clrf     TRISC
 clrf     STATUS

loop
 incf     PORTC
 call     delay
 goto     loop

; Delay loops delay
 movlw    outer_delay
 movwf    30h outer_loop

 movlw    inner_delay
 movwf    31h
inner_loop
 decfsz   31h,1
 goto     inner_loop

 decfsz   30h,1
 goto    outer_loop
 return
 end
```

Some of this is familiar, but there are a few new bits. Firstly, we've defined two symbols (outer_delay and inner_delay), so that we can simply use the words instead of number further on. It doesn't actually simplify our code much in this instance, it's simply to show how it's done. In the initial part of the program, we set the TRISC register so that all 8 bits of port C are outputs. The interesting stuff starts at the label 'loop'. The first new instruction is incf. This instruction adds 1 to the value stored in the named register. In this case, PORTC is used, which happens to be the I/O port connected to the row of LEDs. This results in a pattern of lights showing the binary representation of the number in PORTC.

The next instruction is 'call delay'. The call instruction jumps to a subroutine, by changing the execution point of the processor to the memory location specified, in this case to the label 'loop'. It also pushes the current execution location on to a stack, so that when the subroutine is complete, popping the value off the stack will return execution to the next instruction after the call. Calls can be nested up to eight deep, because the stack only contains eight locations.

After the call has returned, the goto instruction means that the program loops indefinitely. When you run the program, you'll see a changing pattern of lights, representing repeated counting in binary from 0 to 255.

The delay subroutine is included to slow down the process to a speed you can actually see. Try removing the 'call delay' instruction, and it will appear that all the LEDs are permanently lit. They are not, it's just that the changes are happening too fast for you to perceive. The delay subroutine contains two nested loops, which simply take time to execute and thereby slow down the whole process. There are two loops, because a single loop based on a single byte counter can only execute 256 times. Nesting two loops means a delay of 256x256 iterations, which gives a delay big enough to be useful.

The delay loop only introduces one new instruction: decfsz. This means 'decrement the value in register f, and if the resulting value is zero, skip the next instruction'. So, let's look at the inner loop:

```
 movlw    inner_delay
 movwf    31h
inner_loop
 decfsz   31h,1
goto      inner_loop
```

First, we load the previously-defined inner_delay value into the W register. Then, we write that value into register location 0x31 (which is part of the range of memory registers free for general use). The next instruction is the decfsz 31h,1. The first parameter of this instruction is the memory location to be decremented, the second specifies whether the result is placed into the W register (if 0) or back into the memory location (if 1). In this case, we're writing the value back to the memory location. Once the decrement has been performed, the instruction checks the value to see if it is zero. If it is not, execution proceed s to the next instruction, which is goto inner_loop. If the zero value is reached, however, the next instruction is skipped, so our loop exits (back to the outer loop). When both loops are finished, the return instruction causes execution to jump back to the main program.

Program 3 – your code

Using the knowledge you have gained from the previous two examples, write a program which causes a single light to appear to move over the set of LEDs. Use all three columns (PORTA, PORTB and PORTC) if you wish.

You will probably need to use other instructions for this. Consult the datasheet, pages 159 onwards for the full instruction set.

Other resources

The canvas site contains the full specification sheet for the PIC16F877A chip, which includes full details of the hardware and the instruction set. NOTE: the main PIC data sheet has a missing page which leaves out several instructions! Read the errata sheet as well.

Details of the hardware board are also on the canvas site.