

# EECS 100/43

## Lab 8 – Introduction to the PIC Microcontroller

### 1. Objective

In this lab, you will learn about how to use your PIC Microcontroller. This lab is the first lab for the digital component of the EE100 project. The goal of the project is to simply display the strain gauge measurement from your Strain Gauge lab on the LCD display on your PICDEM 2 Plus demo board.

### 2. Equipment

- a. Breadboard
- b. Wire cutters
- c. Wires
- d. Oscilloscope
- e. Function Generator
- f. Power supply
- g. PICDEM 2 Plus demo board
- h. MPLAB ICD 2 (In-circuit Debugger)
- i. MPLAB IDE installed on lab computer<sup>1</sup>
- j. Various connectors for power supply, function generator and oscilloscope.

### 3. Theory

#### **a. Introduction to microcontrollers [1]**

You have all heard of the Intel and AMD microprocessors. These little devices are the brains of your computers. However, there is another very important CPU (Central Processing Unit) in the digital world: the microcontroller.

Microcontroller differs from a microprocessor in many ways. First and the most important is functionality. In order for a microprocessor to be used, other components such as memory, or components for receiving and sending data must be added to it. On the other hand, microcontroller is designed with most of the components above built-in. Very few external components are needed for its application. Thus, we save the time and space needed to construct devices.

You will learn in lecture how digital electronics and systems are a beautiful abstraction of the analog world. You will also learn how you can construct computational circuits and memory units using digital electronics<sup>2</sup>. In lab, you will learn how to use a very popular microcontroller – the PIC from Microchip corporation.

---

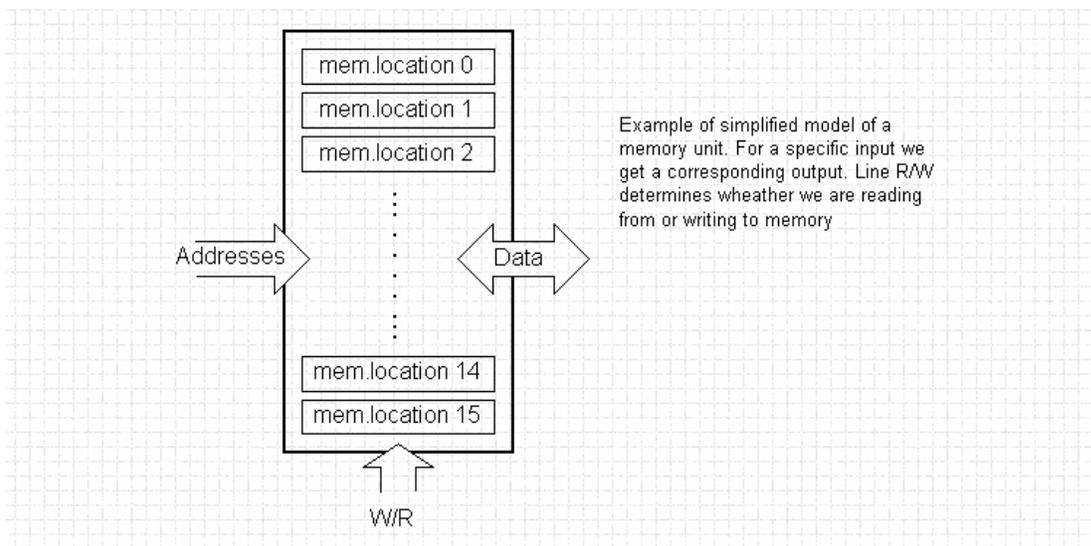
<sup>1</sup> Note: The microcontroller programming tools are installed in 140 Cory ONLY for ease of maintenance

<sup>2</sup> EE100 Summer 2007 lectures at the University of California, Berkeley

Be advised that we will view the microcontroller from a very high level perspective. That is, we will not go into the nitty-gritty details of designing microcontrollers because it is beyond the scope of this lab. However, the aim of this lab (and the project) is to give you a working idea behind PIC microcontrollers so that you may build useful circuits and continue to explore the microcontroller world on your own. We will explore the most important components of the microcontroller unit in this section.

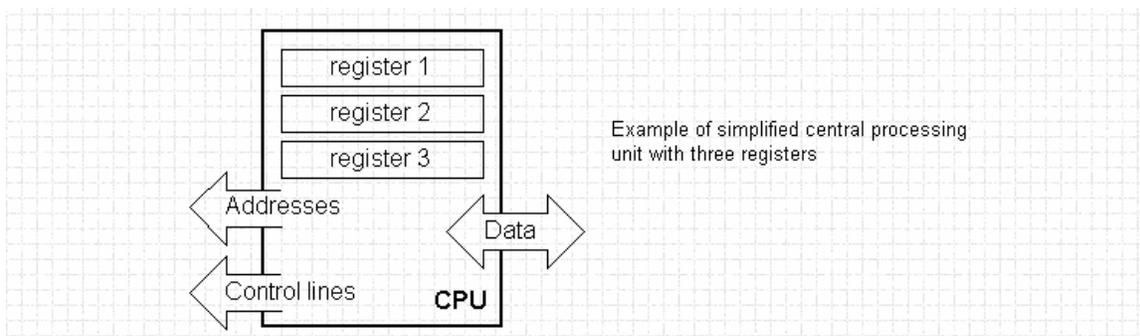
### i. The CPU

You will see how memory can be understood using the concept of digital feedback in the EE100 Summer 2007 lectures. For now, an abstract view of memory will suffice, refer to Figure 1.



**Figure 1.** Simplified view of a memory unit

You will also see how you can build simple circuits that add, subtract and compare: **the Arithmetic Logic Unit (ALU)**. The **CPU** simply consists of the **ALU** along with internal memory locations called **registers**, refer to figure 2.



**Figure 2.** A simplified CPU

## ii. The Bus

We now have two independent entities (memory and CPU). Thus any exchange of data is hindered, as well as its functionality. If, for example, we wish to add the contents of two memory locations and return the result again back to memory, we would need a connection between memory and CPU. Simply stated, we must have some "way" through data goes from one block to another. Enter the concept of a **bus**.

Physically, a bus represents a group of 8, 16, or more wires. There are two types of buses: address and data bus. The first one consists of as many lines as the amount of memory we wish to address and the other one is as wide as data, in our case 8 bits or the connection line. First one serves to transmit address from CPU memory, and the second to connect all blocks inside the microcontroller. Figure 3 shows the new situation.

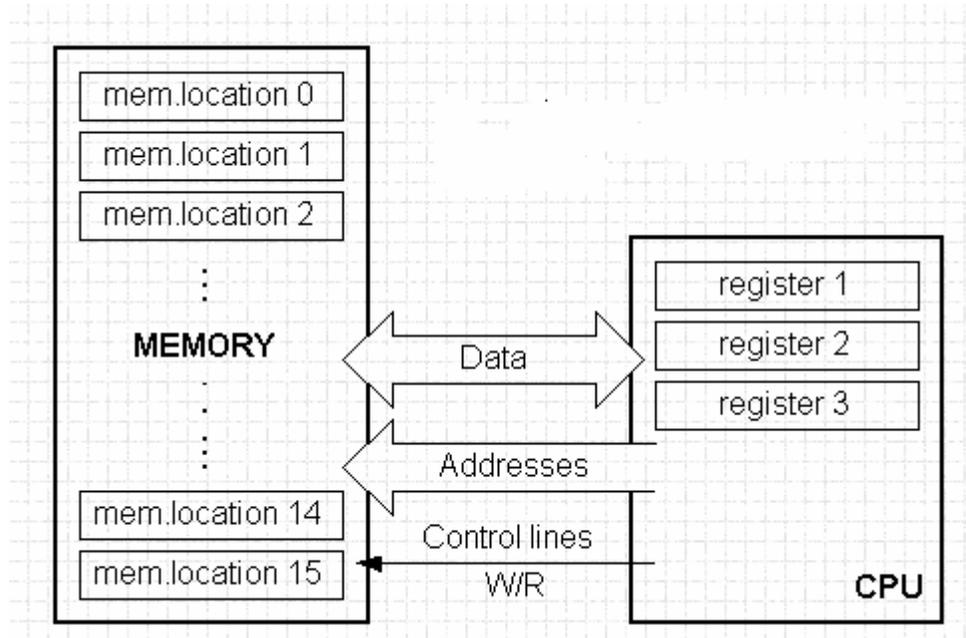
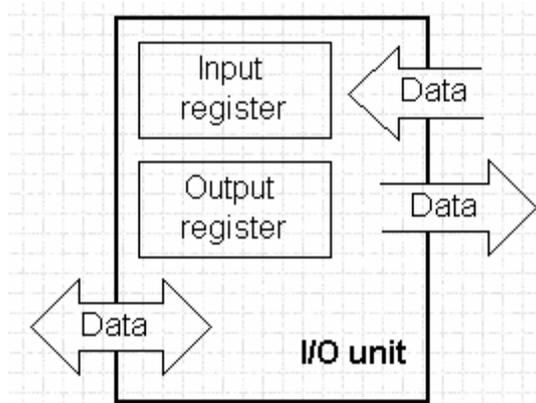


Figure 3. Connecting memory and CPU using instruction and data bus

## iii. Input Output unit

As far as functionality, the situation has improved. But a new problem has appeared: we have a unit that is capable of working by itself, but it does not have any contact with the outside world! In order to remove this deficiency, let's add a block which contains several memory locations. One end of this block is connected to the data bus and the other has connection with the output lines on the microcontroller. These output lines can be seen outside the microcontroller as output pins (just like the output pin of your op-amp). Refer to figure 4.



**Figure 4.** Example of a simplified input-output (I/O) unit that provides communication with the real-world

A pin on the microcontroller chip that lets you talk to the outside world is called as a **port**. There are several types of ports – input, output or bidirectional (input and output). When working with ports, it is necessary to choose which port we need to work with and then send data to or receive data from the port. Notice also from figure 4 that working with a port is like reading and writing to a memory location. This is accomplished using **registers**. A register is a memory location mapped to a port. When you write to or read from the register, you are reading or writing from the corresponding port.

#### iv. Serial communication

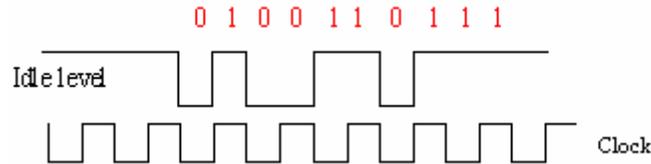
We can now communicate with the outside world, but what if we wanted to communicate kilobytes of data? Lets make a quick calculation on how many lines we would need. 1 kilobyte = 1024 bytes. 1 byte = 8 bits. Thus, we would need:  $1024 \text{ bytes} \times 8 \text{ bits/byte} = 8192 \text{ bits}$ . If a port carries 1-bit of data, then we would need 8192 ports or 8192 pins! What if we wanted to carry a megabyte (1024 kilobytes) of data? You would need 8,388,608 lines! Thus it is obvious that if you want to communicate large amounts of data with the outside world, a port is very inefficient. Ports are useful only for carry a bit of data.

In order to overcome this problem, we have to come up with a way to reduce the number of lines and not lose functionality. Suppose we are working with three lines only and one line is used for sending data, other for receiving, and the third is used as a reference line for both the input and the output. In order for this to work, we need to set a few rules for the exchange of data. These rules are called **protocols**.

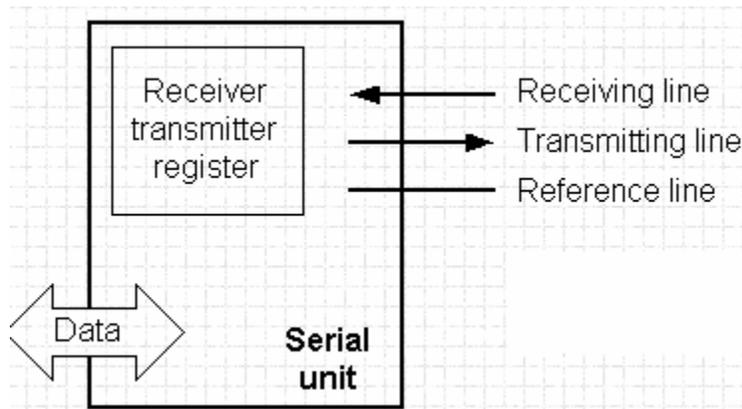
Let us suppose we have the following protocol: The logical unit "1" is set up on the transmitting line until transfer begins. Once the transfer starts, we lower the transmission line to logical "0" for a period of time (which we will designate as T), so the receiving side will know that it is receiving data. Hence, the receiving side will activate its mechanism for reception.

Now the transmitting side can start sending binary data. Let each bit stay on the line for a time period which is equal to T. In the end (after transmitting all the data) let us bring the

logical unit "1" back on the line. The protocol we've just described is called in professional literature: NRZ (Non-Return to Zero). Figure 5 illustrates this concept. Figure 6 shows a block diagram of the serial communication unit.



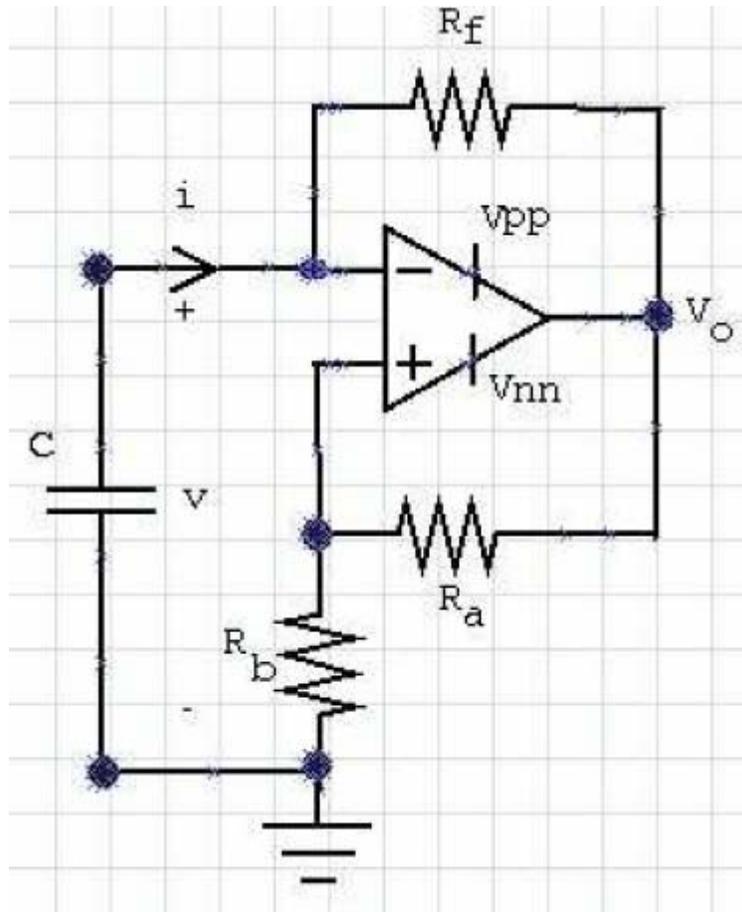
**Figure 5.** NRZ protocol. Notice that there is a clock which tells us how long a logic 1 and logic 0 are. We will discuss clocks in the next section



**Figure 6.** The Serial communication unit.

### v. Timer unit

We have seen in the previous section that we need a clock signal as a reference. You have seen how you can generate a square wave using a simple op-amp circuit. Figure 7 shows the capacitive square wave oscillator you are already familiar with.



**Figure 7.** Square-wave generator. Output  $V_o$  will oscillate between  $V_{pp}$  and  $V_{nn}$ .

However, there are more efficient methods to generate oscillators. We will not go into these techniques. It is sufficient to understand that the basic unit of the timer block is a free running counter whose numeric value increments by one in even intervals. Therefore, if we read the value of this counter at  $T_1$  and then read it again at  $T_2$ , the difference would indicate how much time has elapsed.

#### vi. Watchdog Timer

Unlike a regular PC, your microcontroller will be functioning in time critical applications (like aircraft control) and/or in remote in-accessible areas (like seismic sensors). Hence, if your microcontroller crashes then there is probably no one to push the reset button (unlike a PC).

To overcome this obstacle, designers introduced the concept of a “Watchdog Timer”. This block is in fact another free-run counter where our program will write a zero every time it executes correctly. If the program gets "stuck" due to some external condition, zero will not be written and the Watchdog timer will reset the microcontroller upon achieving its maximum value. This will result in executing the program again, and correctly this time around. The Watchdog timer is thus an important element of every

microcontroller. It helps the system recover from errors without human supervision. Notice that the Watchdog timer is **NOT** a substitute for correct program functionality. It is assumed that the program is functioning correctly. Figure 8 shows a block diagram of the Watchdog timer.

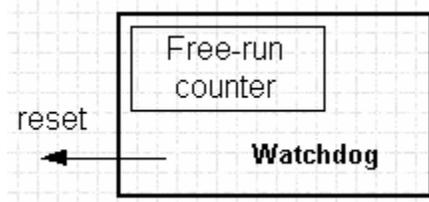


Figure 8. The Watchdog timer

### vii. Analog to Digital (A/D) converter

As external signals are usually substantially different from the ones that microcontroller can understand (zero and one), they have to be converted into a pattern which can be comprehended by a microcontroller. This task is performed by a block for analog to digital conversion or by an ADC. Figure 9 shows a block diagram of the A/D.

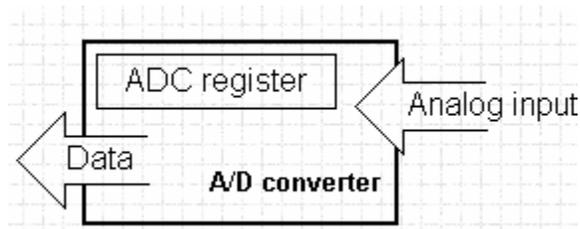


Figure 9. Block diagram of the A/D

### viii. The Microcontroller Block Diagram

Now we are ready to assemble our microcontroller based on the building blocks discussed above. Refer to figure 10.

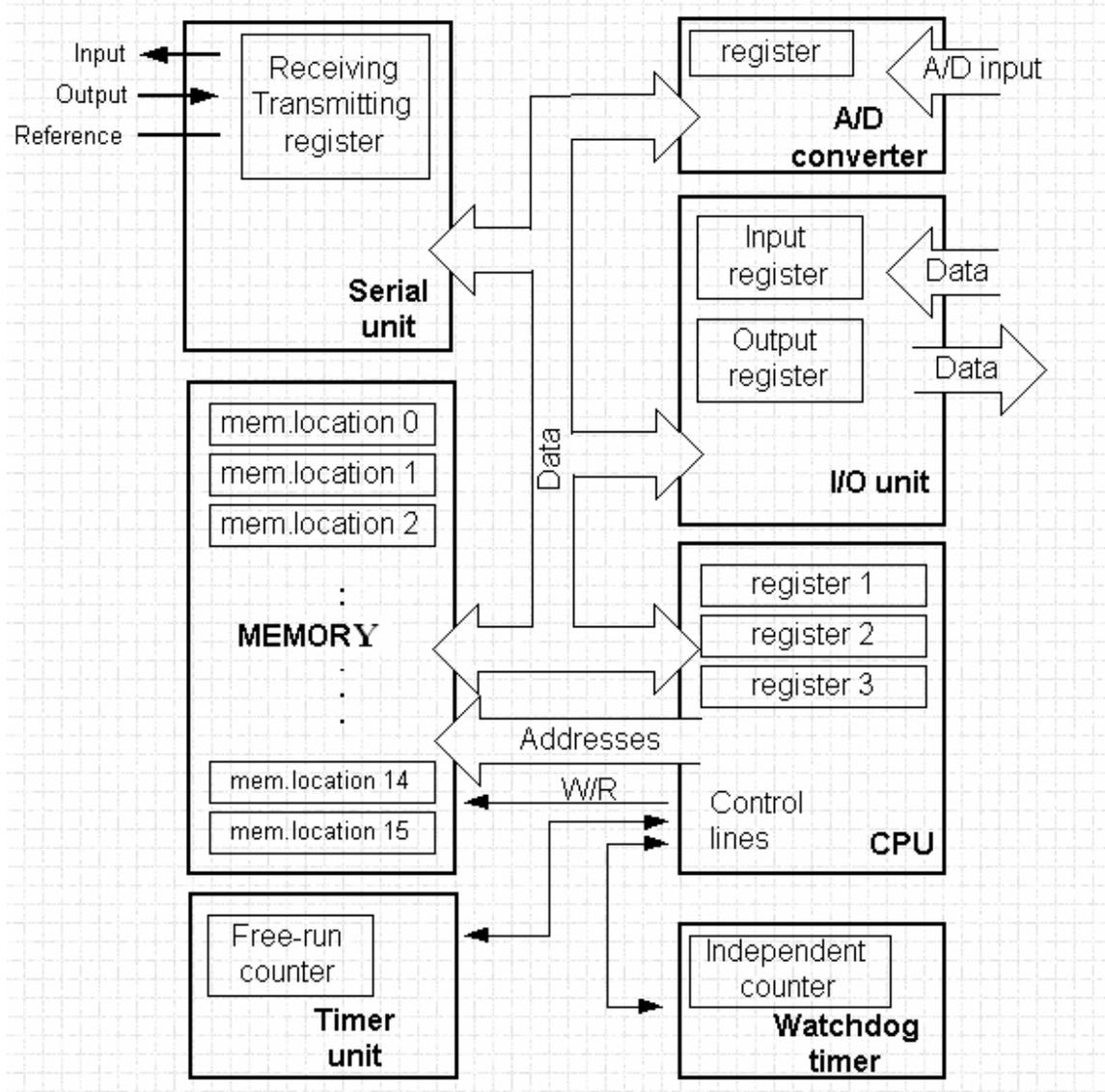


Figure 10. Microcontroller block diagram

### ix. Programming a Microcontroller

Now that we have the hardware in place, we need to program the microcontroller to accomplish a task. First, we will write a simple program in **pseudo-code**. By pseudo-code we mean this program is not written using any programming language (like C). Refer to the code below.

```
START
REGISTER 1 = MEMORY LOCATION A
REGISTER 2 = MEMORY LOCATION B
PORT A = REGISTER 1 + REGISTER 2
END
```

The program above simply adds two memory locations and puts their sum on port A. The first line of the program moves the contents of memory location A into a CPU register. As we need the other data as well, we will move it into register 2. The next instruction tells the CPU to add the contents of those registers and send it to port A, so the sum would be visible to the outside world.

Of course we program in the real world using languages like “C”. However, programming a microcontroller is more complex than programming on the PC. This is because of the fundamental issue in microcontroller programming: you are writing a program that is going to run on a completely different system. This process is called **cross-compiling**. Hence you need sophisticated tools (hardware and software) to help you program a microcontroller. In the next section, we will deal with the hardware – the PICDEM 2 Plus development board.

### **b. The PICDEM 2 Plus development board [2]**

In this section, we will give an overview of the PICDEM 2 plus development board, shown in figure 11.



**Figure 11.** The PICDEM 2 plus development board (PICDEM 2 plus demo board).

This board has at its core a P18F4520 microcontroller. The board also has a host of other peripherals to ease interfacing to the microcontroller and programming the microcontroller. The most important of these are covered below.

#### **i. Board Hardware Features ([2], p. 10)**

The PICDEM 2 plus development board has the following hardware features (refer to figure 12):

1. 18, 28 and 40-pin DIP sockets. Although three sockets are provided, only one device may be used at a time. In your case you have a PIC18F4520 in the 40-pin DIP socket.
2. On-board +5V regulator for direct input from 9V, 100 mA AC/DC wall adapter or 9V battery, or hooks for a +5V, 100 mA regulated DC supply.
3. Serial (RS-232) socket and associated hardware for direct connection to an RS-232 interface.
4. In-Circuit Debugger (ICD) connector.
5. 5 K $\Omega$  potentiometer for devices with analog inputs.
6. Three push button switches for external stimulus and Reset.
7. Power-on indicator LED.
8. Four LEDs connected to PORTB.
9. Jumper J6 to disconnect LEDs from PORTB.
10. 4 MHz canned crystal oscillator.
11. Unpopulated holes provided for crystal connection.
12. 32.768 kHz crystal for Timer1 clock operation.
13. Jumper J7 to disconnect on-board RC oscillator (approximately 2 MHz).
14. 32K x 8 Serial EEPROM.
15. LCD display.
16. Piezo buzzer.
17. Prototype area for user hardware.
18. Microchip TC74 thermal sensor.

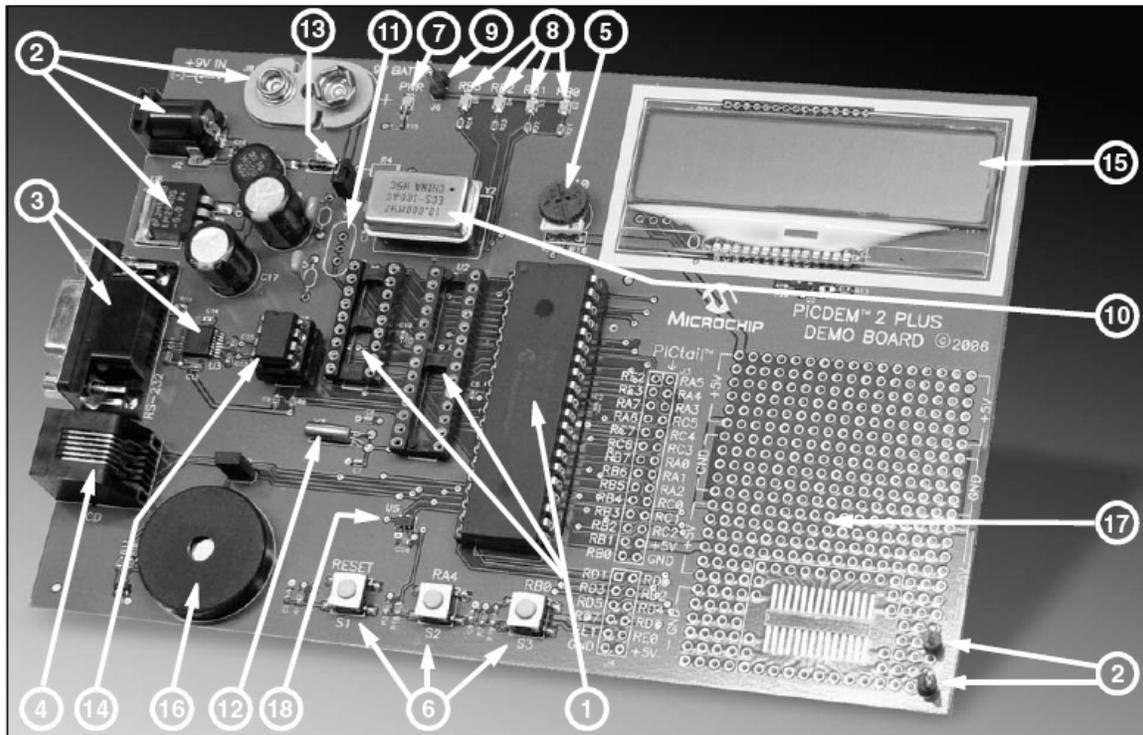


Figure 12. PICDEM 2 Plus hardware

You should relate the board components above to the high-level microcontroller blocks you learned about in section (a). For example, the serial hardware on the board helps a computer interface to the microcontroller's serial unit. For a detailed description of the board hardware, refer to pp. 15 and 16 in [2].

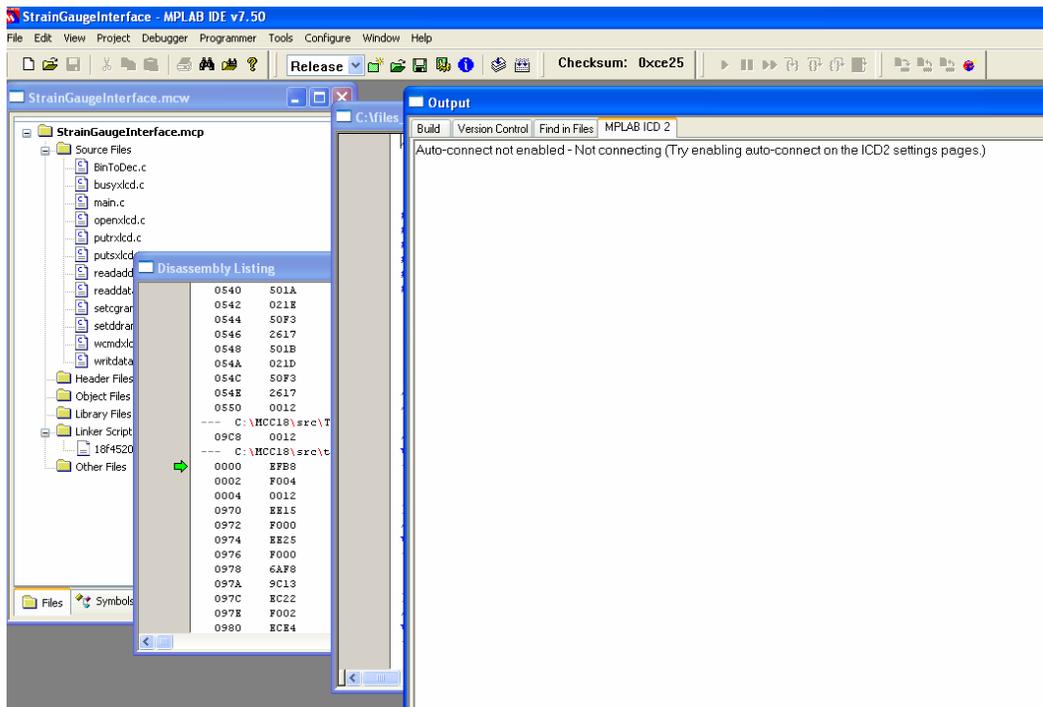
Now that you have an idea of the hardware platform, it is time to play with the actual board.

### **c. Introduction to MPLAB IDE and MPLAB C18 C Compiler**

In this section, you download the digital code to your microcontroller board. You will use the ICD2 (In-Circuit Debugger) module to do so. Note that since you do not cover digital systems etc. in great detail, we have designed all the code for you. The digital portion of your project simply requires you to change a few bits to instruct the microcontroller to read from a different A/D – the goal of the final lab. For now, you will just “burn” the code onto the PIC:

1. Download the StrainGauge project folder from the EE100 labs homepage. Save this to your U: (network) drive. **DO NOT SAVE THIS ONTO YOUR DESKTOP. FILES SAVED ONTO YOUR DESKTOP ARE AUTOMATICALLY DELETED WHEN YOU LOG OFF!!**

2. Navigate to your folder and double-click on  StrainGaugeInterface.mcp icon. A window similar to figure 13 will pop up.



**Figure 13.** MPLAB IDE has opened your Strain Gauge project

3. Now, double click on main.c in the StrainGaugeInterface.mew window. The main program should pop up. Look through it. Find the line that displays VOLTMETER. Replace VOLTMETER with EE100 ROCKS!
4. You should debug the program before downloading it to the board. To do this: Click on Debugger→Select Tool. Make sure MPLAB ICD 2 is selected. Refer to figure 14.

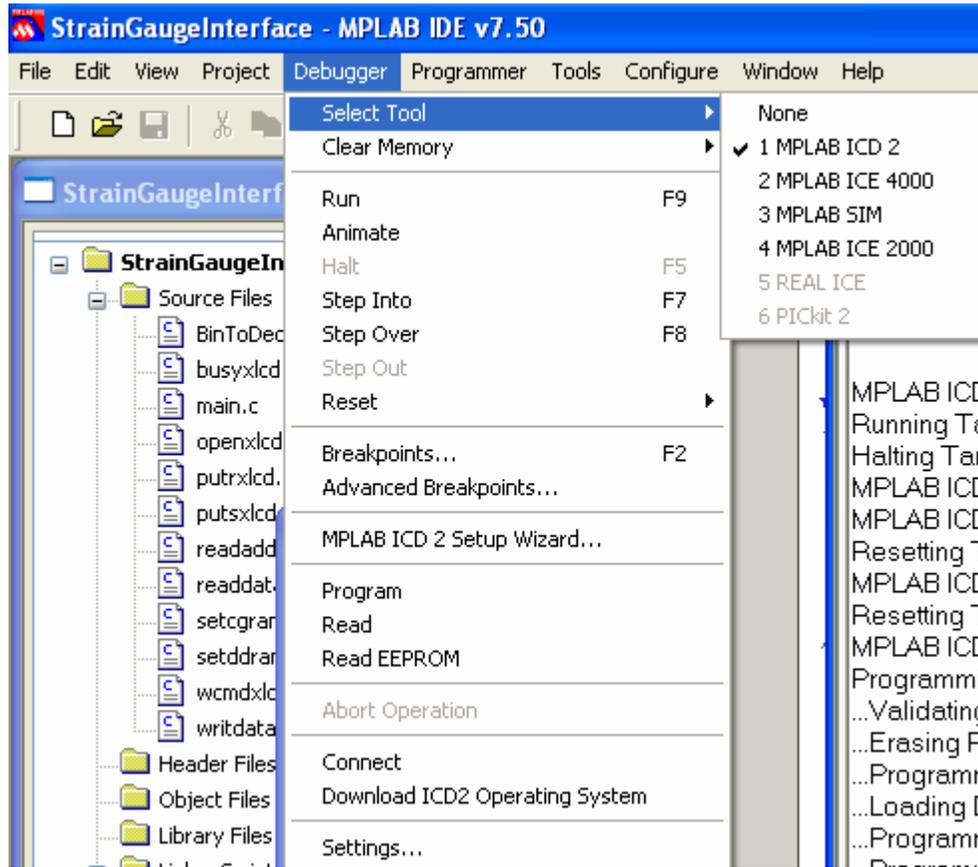


Figure 14. Make sure MPLAB ICD 2 is selected as the debugger

5. The debugger should ask you to download an operating system to the board. Defaults are OK for everything. Before you connect to the board, **MAKE SURE THE BOARD IS POWERED ON AND CONNECTED TO THE ICD2 DEBUGGER. MAKE SURE THE DEBUGGER IS CONNECTED TO THE COMPUTER (check with a TA).** Next, connect to the board: Debugger→Connect.
6. Build your program. Select Project→Build All
7. Program the board with your code: Debugger→Program.
8. Run the program: Debugger→Run.
9. If all goes well you should see the LCD screen display “EE100 ROCKS!” and a voltage reading on the second line. Use the pot (RA0) on the PICDEM 2 plus board to adjust the voltage value (from 0.00 to 5.01 V).

10. If you are satisfied that your program is working, you can “permanently” download code to EEPROM<sup>3</sup>. First, halt debug mode: Debugger→Halt.
11. Select Debugger→Select Tool→None. You cannot use the programmer and the debugger at the same time.
12. You pretty much repeat the same process for programming as debugging: Select Programmer→Select Tool→MPLAB ICD 2.
13. Select Programmer→Connect
14. Select Programmer→Program
15. Select Programmer→Release from Reset.
16. Power off the board and power-on again. Your code will start running again.

---

<sup>3</sup> EEPROM: Electrically Erasable Programmable Read Only Memory

**4. PRELAB**      **NAME:** \_\_\_\_\_ **/SECTION** \_\_\_\_\_

Please turn in **INDIVIDUAL COPIES** of the prelab. They are due **10 MINUTES** after start of lab, **NO EXCEPTIONS!**

**a. TASK 1: THOROUGHLY READ THIS DOCUMENT!**  
**(YES, THIS PRELAB IS A GIMME)**

**PRELAB COMPLETE:** \_\_\_\_\_ **(TA CHECKOFF)**

**5. REPORT**      **NAME(S):** \_\_\_\_\_ **/SECTION** \_\_\_\_\_

**a. TASK 1:** PICDEM 2 Plus LCD displays EE100 ROCKS!    TA Checkoff: \_\_\_\_\_

**b. TASK 2:** Program still functional after power cycle. TA Checkoff: \_\_\_\_\_

**TURN IN ONE REPORT PER GROUP AT THE END OF YOUR LAB SESSION.  
THERE IS NO TAKE HOME REPORT.**

## 6. REFERENCES

- [1] PIC Microcontrollers: Chapter 1 – Introduction to Microcontrollers. Online at:  
[http://www.mikroe.com/en/books/picbook/1\\_chapter.htm](http://www.mikroe.com/en/books/picbook/1_chapter.htm). Last Accessed: July 8<sup>th</sup> 2007
- [2] PICDEM 2 Plus Board User's Guide. PDF format. Microchip corporation,  
<http://www.microchip.com>

## 7. REVISION HISTORY

Date/Author	Revision Comments
Summer 2007/Bharathwaj Muthuswamy	Typed up source documentation, organized lab report, typed up solutions.