ECE 2504: Introduction to Computer Engineering
Design Project 3: Introduction to PIC Assembly Language Programming

Objective
Design, code, test, and debug a simple assembly language program for the PIC microcontroller. This project will introduce and reinforce concepts related to direct and indirect addressing, subroutine calls, arithmetic operations, assemblers, and simulators.

Preparation
- Read the information on installing and getting started with the MPLAB development environment package that can be found in the 2504 section of the CEL web page: http://www.ece.vt.edu/cel/index.html.
- Install version 7.XX of MPLAB onto your personal computer using the information provided in class.
- Follow the on-screen prompts and the guidance given in the instructions found on the CEL web page.
- Read the "Getting Started with MPLAB" information on the CEL web page. You should practice the editing and assembly processes using the sample program that are described there.

Program Specification
The *Fibonacci sequence* can be defined recursively as follows:

$$F_1 = 1$$
$$F_2 = 1$$
$$F_n = F_{n-2} + F_{n-1} \text{ (for } n > 2\text{)}$$

That is, after the first two terms, the *n*th number in the sequence is equal to the sum of the preceding two terms in the sequence. Therefore, the first 12 terms of the sequence are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, and 144.

Write a program for the PIC microcontroller that calculates the value of the *N*th Fibonacci number. The value of *N* is stored in file memory location 0x20, and will be placed there prior to the start of program execution. The result, if one is calculated, should be stored in file memory location 0x22.

If the values in the PIC microcontroller's 8-bit data path are interpreted as signed integers, then any value of N greater than 11 will result in a Fibonacci number that cannot be represented using 8 bits. Write your program so that if *N* is greater than 11 (0x0B) an error flag is set. Use file memory location 0x21 as the error flag.

When a valid value for *N* is found in file memory location 0x20, all of the bits in file memory location 0x21 should be zero, i.e., the value in file memory location 0x21 should be 0x00. In this case, the resulting Fibonacci number should be stored in file memory location 0x22. When an invalid value for *N* is found in file memory location 0x20, all of the bits in file memory location 0x21 should be one, i.e., the value in file memory location 0x21 should be 0xFF. In this case, file memory location 0x22 should contain a value of zero.

Table 1 shows a set of sample inputs and outputs, as they would be observed in the file memory before and after the execution of the program:

File Memory Address	Value before execution	Value after execution	Value before execution	Value after execution	Value before execution	Value after execution	Value before execution	Value after execution
20	09	09	0C	0C	02	02	11	11
21	XX	00	XX	FF	XX	00	XX	FF
22	XX	22	XX	00	XX	01	XX	00

Table 1: Sample File Memory Contents. All values are given in hexadecimal. XX = don't care

Program Restrictions

Note that the value of *N* contained in file memory location 0x20 does not change as a result of the program's execution. Assuming that the value of *N* is used as a variable, your program must operate on a "copy" of *N* that is separate from the one contained in file memory location 0x20.

Your program should not rely on a look-up table, which seeks to match values of *N* with corresponding Fibonacci numbers that have somehow been stored in the file memory. You may handle the first two Fibonacci numbers as special cases of the sequence, but for $N > 2$, you should use a recursive algorithm to find F_n.

Write a subroutine to handle the cases where a Fibonacci number must be calculated. In cases where $N > 2$, this subroutine should handle the job of using the recursive sequence definition to obtain the desired Fibonacci number.

Designing your Program

- Originate your program starting at program memory location 0x00. Interrupts will not be used in this program, so you do not need to jump around location 0x04, the normal starting point for an interrupt service routine.
- After your program stores the appropriate value in file memory locations 0x21 and 0x22, it should enter an infinite loop that does nothing. Do not use any other method to halt or terminate the program's execution.
- You will initialize the value of *N* (file memory location 0x20) prior to executing your program. Instructions on how to do this are in given below in the appendix on using MPLAB
- A set of sample programs has been provided with this assignment. You may find them useful since they examples of how assembly programs are structured, and illustrate concepts such as direct and indirect addressing and the use of subroutines.
- Create and edit a program file of your own creation using the MPLAB environment editor (an ASCII text editor such as Notepad can also be used – Word is NOT an ASCII text editor.). The program file name must have the extension **.asm**. Your program file MUST include sufficient comments to document the overall algorithm that you are using and the operation of the code itself. See the sample program for style suggestions.

Debugging and Testing your Program

After you have designed, written, and successfully assembled your program, use MPLAB to simulate the program's execution and to verify that it works correctly. The single-step function described in the appendix is useful for debugging and testing. Once the program works, do the following test for your project submission. Instructions on how to perform these steps in MPLAB are provided in the appendix.

1. Reset the processor.
2. Choose a **valid** value for N and initialize file memory location 0x20 with that value. This is the only file register location that you should initialize. Print a copy of the file register window. On the copy, highlight the values in file registers 0x20, 0x21, and 0x22. Label this copy as "Before Test X," where X represents the test number. You need not test the program for all possible cases of N, but you should test enough cases to show that your program meets the specification requirements. Submit each view of the file register window with your lab report.
3. Use the trace feature of MPLAB to highlight the execution of a small section of the code, such as the calculation subroutine. The trace shows that the Fibonacci number is being calculated in the correct recursive manner. You need only perform this step for the **first** test that you generate.
4. Execute the program.
5. Examine the contents of file register locations 0x21 and 0x22 to verify that the error flag was set appropriately and that the Fibonacci number was calculated correctly. Print a copy of the file register window, and highlight the values in locations 0x20, 0x21, and 0x22. Label this copy as "After Test X," where X represents the test number and submit it with your lab report. Each "Before Test X" view should have a corresponding "After Test X" view.
6. Print a copy of the trace memory window and verify that the simulator has traced the desired instructions. Turn in the trace output with your lab report.
7. For each test, print a copy of the Special Function Registers window. Label it as corresponding to the appropriate test case turn it in with your lab report.

If your program did not operate correctly, reset the simulation and execute your program again using the single-step or trace debugging tools to find your errors. When you are confident that your program works correctly, take your program on a floppy disk to the CEL. Use MPLAB on any available CEL computer to demonstrate your program's operation to one of the GTAs that are on duty. Use the attached validation form to do this.

Submission Requirements

You must prepare a concise project report using the guidelines given in the ECE 2504 Laboratory Manual. Your report should include the following items in the order listed below. Securely staple your report together. Do not use any type of binder, notebook, or folder.

- Cover sheet (provided at the end of this assignment)

- Body of report
  - Statement of design requirements
  - Brief description of your program's design
  - Summary of any problems encountered with the design, tools, or testing
  - Conclusions

- Printed documentation of your results, including the following:
  - **Listing file** for your program (the **.lst** file created during the assembly step)
  - **Trace memory window**, annotated as necessary, for the **first** test case
  - Highlighted **before** and **after** file memory windows for each test case

- Completed validation sheet

Honor Code

You must comply with Virginia Tech's Honor System. The program and report must be your own work. You may ask other students general questions about the PIC instruction set and how MPLAB works. You are not allowed to ask anyone except your instructor or a CEL GTA questions about the design of the program. The program design, coding, debugging, and testing must be your own work. It is an honor code violation to share your design with another person or to copy another person's design either as a paper design or as a computer file.

Appendix: Notes on Using MPLAB

This appendix provides information to help you complete Design Project 3. It discusses installing MPLAB, creating a project, writing and assembling your program, and using the simulator to set file register values, print the file register window, single-step through your program, and create and print program traces.

The information in this appendix was adapted largely from information originally created by Scott Midkiff. It should now conform to the features of MPLAB version 7.XX

MPLAB Installation

Microchip MPLAB IDE version 7.00 is available at the CEL web site (http://www.ece.vt.edu/cel) in the ECE 2504 section. Version 7.10 is available at the Microchip web site (http://www.microchip.com). Do not use a version of MPLAB older than version 7.00. Download and install MPLAB as follows.

- Use your web browser to retrieve the installation file. Save the file to your computer's disk. The file is called **mplab700.zip** on the CEL web site. You will need to use your Virginia Tech PID and password to retrieve the file from the CEL web site. The file is called **mplab710.zip** on the Microchip web site.

- Once you have downloaded the installation file to your computer, unzip it using WinZip or other utility. Begin installation by double clicking on the unzipped file. The default options suggested during the install process should work.

Creating a Project and Assembling a Program Using the MPLAB IDE

- Before starting to work with MPLAB, read the first two chapters of the MPLAB Quick Start Guide. Chapter 2 is a simple tutorial on the use of the IDE. It may make reference to features of version 6.XX. These features should be the same in version 7.XX

- Section 2.2 of the "Quick Start Guide" instructs you to input a 25 to 30-line program. Instead, enter the following short program. Note that there is an intentional assembly language error in the third line. We will fix it later.

```
; Sample program to use while learning MPLAB

        org 0x00

        clearw          ; This is an error that we will fix.
loop    addlw   0x01
        movwf   0x20
        goto    loop
        end
```

- Save your program code. Note that MPLAB restricts file names to a maximum of 62 characters. Saving your file multiple levels down into your "Documents and Settings" directory may cause problems. Consider using a top-level MPLAB-files directory such as "C:\mplabfiles" and save your work to that directory. Make sure you save your file with the **.asm** extension.

- The next thing you need to do is create a project. Follow the steps discussed in Section 2.3 of the "Quick Start Guide." Make sure you select **16F84** as the device you will be using (the step shown in Figure 2-3 in the guide). The active toolset should be Microchip MPASM Toolsuite, as shown in Figure 2-4. (Creating a project in MPLAB is not always necessary. If you choose not to create a project, you must use the "Configure-Select Device" to select the 16F84 microcontroller chip and then use the "Project-Select Language Toolset" menu to specify the Microchip MPASM Toolsuite and its location in your computer's file system.)

- Once you have created your project, you can build it. This is discussed in Section 2.4 of the Quick Start Guide. Select **Project > Build All** from the menu or type **Ctrl+F10** as a short cut.

- Your program should assemble, but the build output message should report the following error:

```
Warning [207] C:\mplabfiles\programname.ASM 3 : Found label after column 1. (clearw)
```

Your directory and file name may be different.

- To fix the error, first double click on the warning message. Your source code window will be displayed again. A green arrow will point to the line containing the problem. In this case, it is the "clearw" statement. Since the line of text does not start in column 1, the assembler wants to interpret its contents as an assembly instruction. But since the word in question does not correspond to any assembly instruction, the assembler assumes that it is a label that has been moved out of column 1. In reality, it's a syntax error. Many such warnings will occur when you make syntax errors that involve assembly instructions. Correct the error by replacing **clearw** with the instruction **clrw**.

- After fixing the error, go through the **Build All** step again and observe that the program assembles with no errors or warning messages.

Simulating Program Execution

You will not run your program on a real PIC processor but, instead, will use the simulator built into MPLAB, as described in this section.

- Once your project has been successfully built, you can simulate its execution by running the MPLAB SIM simulator. Simulation and debugging are discussed in Sections 2.5 and 2.6 of the Quick Start Guide.

- Make sure you select the MPLAB SIM, as shown in Figure 2-11 of the guide.

- Before simulating your program, open the File Registers and Special Function Registers windows using the **View** > **File Registers** and **View > Special Function Registers** menu options. These two windows display the contents of the file memory, and of various named register locations, such as W, STATUS, and FSR.

- Single-step through the execution of the program using the **Debugger** > **Step Into** menu option, the **F7** key, or the **Step Into** button on the tool bar. Single-stepping through your program allows you to observe the execution of your program as it executes each individual instruction. Single-step through 8 to 10 instruction executions. Observe the contents of the W register in the Special Function Registers window and location 0x20 in the File Registers window. You should see register W being incremented, followed by the current value of W being written to location 0x20 in the register file.

- If you want to execute your program again, you can reset the processor (which resets the Program Counter but not most File Register contents) by selecting the **Debugger** > **Reset** > **Processor Reset** menu option, typing **F6**, or clicking the **Reset** button on the tool bar.

- Once you think your Project 3 program is working correctly, you can run the program by selecting the **Debugger** > **Run** menu option. You can also do this by pressing **F9,** or by clicking the **Run** button on the tool bar. You can stop the simulation by selecting the **Debugger** > **Halt** menu option, by pressing **F5**, or by clicking on the **Halt** button on the tool bar. Alternately, you can run your program with animation by selecting the **Debugger** > **Animate** menu option, or by clicking on the **Animate** button on the tool bar. In this mode, the green arrow in your code window will point to each instruction as it executes so that you can attempt to follow the execution. However, the simulation moves very fast, which limits the value of the animation feature. The **Animate** feature may also be useful in watching changing values in either the File Memory or the Special Function Registers.

To test your program for Project 3, you must initialize register file location 0x20 before you execute the program. Use the File Registers window to do this. Click on the location that you want to change. Type over the current value with the new value and then press the Enter key, the Tab key, or an arrow key. The value is stored and the next memory location is selected. Similar data initializations can be done in the Special Function Registers window. You should not need to do that for this project.

Tracing Program Execution

You are required to trace the program's execution. To trace program execution, you must have already successfully assembled your source file. Create the program trace by performing the following procedure.

Reset the processor by selecting the **Debugger > Reset > Processor Reset** menu option, by typing **F6**, or by clicking the Reset button on the tool bar. Scroll the source file window until you see the instruction that you want to trace last. You might choose the return statement that ends the calculation subroutine. Left-click on the instruction to move the source window cursor to that line. The green arrow will not move. Right-click on the instruction in the source code window and select **Run to Cursor** from the pop-up menu. The program will execute and stop at the selected instruction.

To view the trace, choose **View > Simulator Trace** menu option. Once you have the trace, you can resume execution of your program by running the program in any of the ways indicated above. Halt execution of the program in any of the ways indicated above.

To print the trace file, choose the **View > Simulator Trace** menu. Select and maximize the window. Make sure that it contains the lines of code that you wanted to trace. Choose the **File** > **Print** menu option. After you have set the appropriate print settings, click the **OK** button.

ECE 2504: Introduction to Computer Engineering
Design Project 3: Introduction to PIC Assembly Language Programming

Student Name: ____ Student Number: ___________________________

GTA: The student has written a PIC assembly language program that takes a value *N* stored in file memory location 0x20, and computes the *N*th Fibonacci number. For your convenience, here are the corresponding values for *N* and F_n:

N	Decimal	1	2	3	4	5	6	7	8	9	10	11
	Hexadecimal	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**
F_n	Decimal	1	1	2	3	5	8	13	21	34	55	89
	Hexadecimal	**01**	**01**	**02**	**03**	**05**	**08**	**0D**	**15**	**22**	**37**	**59**

The table shows **valid values** for *N*, since those values of *N* values for F_n that can be expressed in 8-bit signed 2's complement representation. If *N* is greater than 11 (0x0B), then *N* is an **invalid value**. Regardless of whether *N* is valid or invalid, the value of *N* stored in file memory location 0x20 **should not change** as a result of program execution.

If file memory location 0x20 contains a **valid value** of *N* prior to program execution, then file memory location 0x21 will contain the value 0x00 and file memory location 0x22 will contain the value of F_n that corresponds to *N* following execution of the program. If file memory location 0x20 contains an **invalid value** of *N* prior to program execution, then file memory location 0x21 will contain the value 0xFF and file memory location 0x22 will contain the value 0x00 following execution.

Have the students perform the operations shown below **under your direction**. You should choose the values that the student uses for *N*. Record the result of each execution. Use the file memory window to observe the register file contents. The student should have MPLAB running, and his program should be loaded into the simulation environment when the validation begins.

Operation	Memory Address		
Enter a **valid value** for N into file memory location 0x20. Do this in the file registers window. Record the contents of the indicated file memory locations **prior** to executing the program.	0x20	0x21	0x22
Execute the program by selecting **Debugger** > **Run** from the menu. Halt program execution by selecting **Debugger** > **Halt**. Record the contents of the indicated file memory locations **after** you halt execution.	0x20	0x21	0x22
Reset the processor by selecting **Debugger** > **Reset** > **Processor Reset** (**F6**). Do **not** do a system reset. Enter an **invalid** value for N into file memory location 0x20. Record the contents of the indicated file memory locations **prior** to executing the program.	0x20	0x21	0x22
Execute the program. After halting program execution, record the contents of the indicated file memory locations **after** executing the program.	0x20	0x21	0x22
Reset the processor. Enter a **valid** value for N into file memory location 0x20. The value should be different from the first one used. Record the contents of the indicated file memory locations **prior** to executing the program.	0x20	0x21	0x22
Execute the program. After halting program execution, record the contents of the indicated file memory locations **after** executing the program.	0x20	0x21	0x22

Additional Comments: (Please comment if something did not work, or worked in a manner that seemed strange to you.)

GTA Printed Name and Signature: ____

Validation Date: ____

ECE 2504: Introduction to Computer Engineering
Design Project 3: Introduction to PIC Assembly Language Programming

Student Name: ____

Student Number: ____

Date Submitted: ____

Pledge: I have neither given nor received unauthorized assistance on this assignment.

Signed: ____

Project Grading to be completed by GTA or Instructor:

Grading: The design project will be graded on a 100-point basis, as shown below:

Manner of Presentation (25 points)
- Completed Cover Sheet with Name, Student ID, and Signed pledge (5)
- Report Organization: Clear and concise presentation of information (10)
- Mechanics: Spelling and grammar (10 points)

Design Discussion (25 points)
- Statement of design requirements (5)
- Description of your design approach (10)
- Design summary and conclusions, including descriptions of any problems encountered with the design, software tools, and program testing (10)

Design Documentation (25 points)
- Documented listing file (*.lst) (10)
- Trace window with annotations (where appropriate) (5)
- Before and after file memory windows with annotations (where appropriate) (10)

Functionality Based on Completed Validation Form (25 points)

Grade: ____________ (of 100)