15-348: Embedded Systems, Fall 2022
Homework 2: Programmers Assemble
Due: Tuesday, August 30th, 2022 by 10:00pm

Part	Quantity
Launchpad	1
LED	4
220Ω Resistor	4
Momentary, Push-button switch	1
8-bit DIP Switch	1

Introduction

This is a programming homework. You must submit your code to Gradescope by **Tuesday, August 30th, 2022 by 10:00pm**. Only submit the files indicated below. You should also demo your code to the one of the course staff members on or before Thursday, September 1st during one of the available slots. You can only demo once. More instructions about the demos will be announced during class and on Piazza.

Learning Objectives:

Familiarize yourself with assembly language.

Getting Started:

1. For each task, create a new project in CCS, as explained in the tutorial on the website.
2. Submit only the source code files that you modified (i.e., not the entire project, binaries, etc).

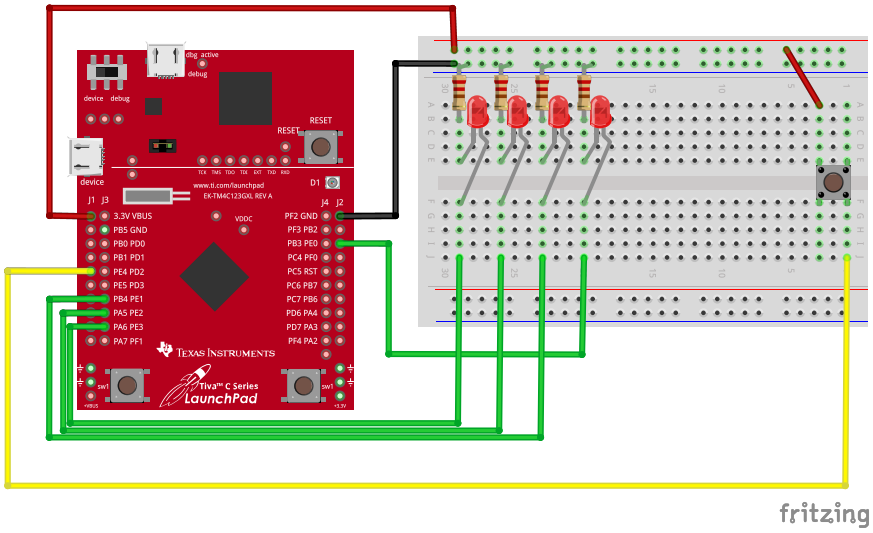
Some Important Notes:

1. You should read the entire assignment before starting. This will help you predict what your final circuit will look like, what sort of functionality your software will need, etc.
2. We expect that your implementation exactly matches the specifications indicated below.
3. Although we might be lenient with unspecified behaviors, you should justify your implementation choices.
4. If you are unsure about the specifications, you should ask questions on Piazza or directly to the course staff.
5. Reading, understanding, and eliciting the requirements and specifications is also part of the assignment.

Task 1 (30 pts)

1. Create a new project in CCS and add the following starter code:
   https://github.com/CMUQ-15-348/hw2-part1-startercode
   (You should not have a `main.c` in your project, instead you should have `main.s`.)

2. Build the circuit shown in the diagram below:



3. Configure Ports **F0** and **F4** to be inputs to enable the onboard push buttons. **Port F** is special, you will need to set bits 0 and 4 on the **GPIOPUR** register for port F in order to enable the pull-up resistors. These buttons are referred to as push button **SW1** and push button **SW2** respectively in the write up.

4. Configure ports **E0**, **E1**, **E2**, **E3** to be **outputs**.

5. Configure port **E4** to be input with a pull-down resistor.

6. Connect ports **E0**, **E1**, **E2**, and **E3** to four LEDs on the breadboard.

7. Connect port **E4** to the provided push button (in the breadboard). This external button is referred to as **SW3**.

8. Implement the following functionality in **assembly**:

   (a) Every time **SW1** is pressed and then released, increment a counter. (For the purpose of describing the algorithm we will call this counter N.)

   (b) When **SW2** is pressed, display the sum of all numbers from **1 to N, inclusive** using the four LEDs. Each LED represents one bit of this sum and the four LEDs combined form a four-bit number. (For example, if N=3, then the binary representation of 6 $(1 + 2 + 3)$ should be displayed on the four LEDs as 0110.)

   (c) If the sum cannot be represented by four bits, then turn on all four LEDs.

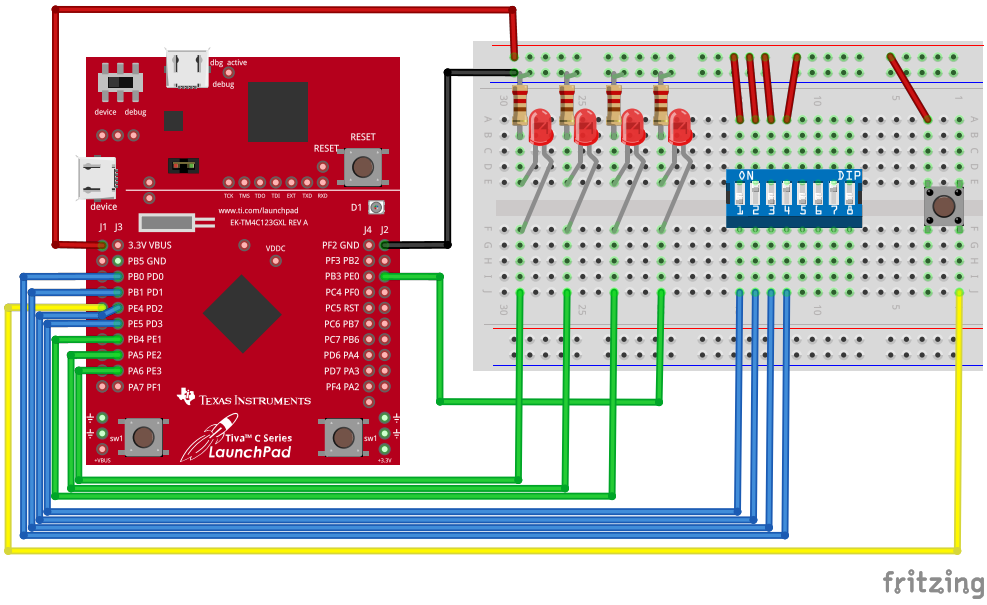
   (d) If **Push Button 3** is pressed and released, turn off all four LEDs and set **N** to 0. **SW1** and **SW2** should then work the same as before.

Task 2 (20 pts)

1. Create a new project in CCS and add the following starter code:
   https://github.com/CMUQ-15-348/hw2-part2-startercode
   (You should not have a `main.c` in your project, instead you should have `main.s`.)

2. Do steps 1-7 of Task 1 in order to setup the LEDs and the push buttons.

3. Modify your circuit by adding a DIP switch as follows:



4. Configure port **D0, D1, D2, D3** to be inputs with pull-down resistors.

5. Connect pins 1-4 of the provided DIP switch to PORT D0, D1, D2, D3, respectively.

6. Write an **assembly program** to input a list of numbers as follows:

   (a) Each time **SW1** is pressed and released, read the current value of **N** and add it to the list. Display the binary representation of **N** with the LEDs like in the previous task. This confirms that the number has been read and added to the list.

   (b) When **SW2** is pressed and then released, clear all LEDs.

   (c) Repeat the procedure above to build a list of 4-bit numbers in memory.

   (d) When **Push Button 3** is pressed, compute the sum of all numbers in the list and display the result using the LEDs. If the sum cannot be represented by four bits, then turn on all four LEDs. Clear the list and **SW1** and **SW2** should then work the same as before.

Please note that even though this task could be performed without actually storing a list of numbers (for example, you could keep a running sum in step 6a), you **must** solve it by storing, and then summing, a list of numbers in memory. (The goal is to force you to use loops and memory addressing in assembly.)

Additional Guidance

General Tips

- Work incrementally, testing each small piece of functionality as you go. For example, your first step might be simply seeing if you can toggle an output pin from 1 to 0 and see the voltage change on the multimeter.
- If you spend too long debugging something, and can't find the problem, start over. Rewire your circuit and rewrite the code from scratch. It shouldn't take you that long the second time, and you may be surprised to find that it works.

The Starter Code

- The provided starter code includes some fields for common port registers, but it might not include everything you need. If you find you need another register, you should add it yourself. You can find the correct memory-mapped addresses in `15348.h` from the previous assignment.
- There are some sample variable declarations. They are provided to show you how to declare variables, not because each of them is required for the assignment.
- There are subtle differences between the assembly language used in the course textbook and the assembly language used by Code Composer Studio in this class. For an overview of the differences, please see https://users.ece.utexas.edu/~valvano/arm/ConvertKeilCCS.pdf, from the author of the textbook.

Debouncing in Assembly

- You will need to debounce your switch inputs. If you use a simple, delay-based debouncing method, then we encourage you to write a helper function that delays for the appropriate amount of time. Look through `timer.c` from previous labs for some ideas of how to do this, but you should try to understand what that code does so that you can make something simpler. Porting all of those functions, as is, to assembly will be needlessly time-consuming. Making one function that incorporates the most important ideas is much better.

Connecting LEDs

- Connect the long end of the LED to the pin from the Launchpad, and the short end of the LED to the 220Ω resistor that is connected to ground.
- LEDs (Light Emitting Diodes), like all diodes, only allow current to flow in one direction. This means that if you connect the LED in reverse, it will not light up.
- DO NOT try to test the LEDs by connecting them directly to the 3.3V pin, it will burn the LED and may damage the Launchpad.

Connecting the Push Button

The button is connected to 3.3v when pressed and is floating when not pressed. In this configuration, you must use the internal pull-down resistor to ensure that the signal is LOW when the button is not pressed.