UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE12/L: COMPUTER SYSTEMS AND ASSEMBLY LANGUAGE

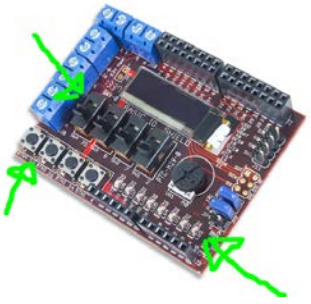
Lab 5: Introduction to Uno32 and MIPS
Worth 60 points (50 lab + 10 report)

Lab Objectives:

Now that you've gotten a feel for assembly language programming, it is time to learn an assembly language that is used on real hardware in the real world. This lab will introduce you to the basics of MIPS and running code on an embedded processor. We will do so by writing code, in MIPS assembly, that will allow a user to control a a row of LEDs by physically pressing buttons and flipping switches.

Using Hardware:

For the remainder of this course, we will be using an embedded processor to run our code. You will receive a CMPE12 Lab Kit, which consists of 3 main parts:

Uno32 Development board: This printed circuit board consists of a kind of processor called a PIC32, along with various other circuit components to power the PIC32, control its clock speed, protect it from high voltages, and some other things. The Uno32 was designed to be an alternative to Arduino boards. It is made by chipKit (and the PIC32 is made by Microchip).	
Uno32 I/O Shield: A "shield" is a circuit board that snaps on top of another board. This particular shield contains hardware that lets you interact with the PIC32 on the Uno32 board below. In particular, it carries 8 LEDs, 4 buttons, and 4 switches. It has some other stuff we don't use in this course: A potentiometer, an OLED display, some current drivers, and an external memory chip. Like the Uno32 that it rides on, the IO shield is made by chipKit.	

PicKit 3 Programmer: Getting your code into the processor's memory is fairly complicated, and requires some special circuitry. In this course, we will use the PicKit 3 to put our code onto the processor. The PicKit 3 also has the ability to control the processor's instruction cycle, allowing us to use runtime debugging techniques like setting breakpoints and stepping through our code. The PicKit 3 is made by Microchip.	

Your CE12 lab kit has two USB cords, both of which must be connected to your computer for the lab kit to function as intended. One USB cord is connected to the Uno32 board. It carries power to the board, and also carries the serial messages from the PIC32 that you will read in CoolTerm. The other USB cord is connected to the PicKit3. This cord carries power to the PicKit3, and also allows MPLABX to control the PicKit3 during programming and debugging.

If for some reason you cannot pick up / drop off your lab kit in class, you may pick one up from Baskin Engineering Lab Support directly. BELS is located in the basement of Baskin Engineering, on the west side of the building. If you've never been to BELS, the easiest way to find them is to enter through the truck loading entrance. BELS is right next to the print shop. (here it is on Google street view: https://tinyurl.com/lk6vele)

Part 1: Hello World

In this lab, we will use the MPLABX Integrated Development Environment to write and assemble code, and then load it onto a processor. We will use the "CoolTerm" terminal program to read serial output from the processor.

Use the "getting started guide" on how to set up a new assembly project in MPLABX. Once you've gotten the "Hello World" message to display on CoolTerm, you can continue to work on the rest of the lab.

Part 2: Mapping Switches, Buttons, and LEDs to Pins

In this part of the lab, you will write code that checks the buttons and switches on the IO shield to see if they are pressed or set, and then light some of the LEDs to reflect the state of the buttons and switches. When your code is complete, the user will be able to turn LEDs on and off by pressing the buttons and flipping the switches.

A microprocessor interacts with the outside world by reading voltages on some of its pins, and outputting voltages on other pins. In code, these interactions are controlled by "Special Function Registers" or "SFRs." These are special locations in memory that are connected to specialized circuits (called "peripherals") that read from or write to the SFRs. In this course, you'll mostly work with two kinds of SFRs: Direction registers (TRISA, TRISB, etc) and Port registers (PORTA, PORTB, etc).

First, you need to make a table to work out what buttons/switches connect to what pins, and which registers are used to interact with those pins. This is called a "Pin Map." It should look a bit like this:

Hardware Component:	Uno32 Pin	PIC32 Pin	PIC32 Direction Register and Bit	PIC32 IO Port Register and Bit
LED 1	33	PMD7/RE7	TRISE, bit 0	PORTE, bit 0
…	…	…	…	…
Switch 1	2	IC1/RTCC/INT1/RD8	TRISD, bit 8	PORTD, bit 8
…	...	…	…	…
Button 1	…	…	…	…
…	...	…	…	…

Your table should have 16 rows: 8 LEDs, 4 buttons, and 4 switches.

Filling out this table requires getting into the hardware documentation. Google for it, or check Piazza. You will need:

- Uno32 IO Shield manual
    - This will let you figure out how hardware components connect to pins on the Uno32 board.
- Uno32 manual
    - This will help figure out which Uno32 pins connect to which PIC32 pins. (Uno32 pins are numbered 1-36, somewhat arbitrarily, so this isn't as straightforward as you might hope).
    - PIC32 pins have names that describe the various peripherals that pin can interact with, so if you see a name like "IC1/RTCC/INT1/RD8" that means that the pin can connect to the Interrupt Capture #1 (IC1) peripheral, the Real Time Calendar and Clock (RTCC) peripheral, Interrupt #1 (INT1), or the 8th pin of I/O poRt D (RD8).
    - In this lab, we only use the I/O functions, so don't worry if you don't know what the rest of it means! That is, we only care about the "Rxy" part of pin names.
- IO shield schematic

    - Not necessary, but can be very handy if you know/learn some of the electrical symbols
- PIC32 Family Reference Manual Chapter 12
    - Discusses the mechanics of how IO ports work
    - Be sure you undertstand how use the TRISx registers to control whether an IO pin is an input or an output. Be sure you understand how to use PORTx registers to read the input on a pin, or to control the output on that pin.
    - Also read about SET, CLR, and INV registers, which can be used to change individual pins in a TRISx or PORTx register without affecting the other pins in that port.

Once you've made your mapping table, test it! This is a good opportunity to get familiar with MIPS code, and to experiment with MPLABX's powerful debugger. See if you write code that can turn on every other LED. See if you can use the debugger to watch PORTD change as you mess with the switches.

Part 3: Polling Switches and Buttons

Once you are comfortable reading buttons and turning on LEDs, let's write an embedded program.

First, read through this section, and write some pseudocode or a flowchart for your Part 3 code.

Once your pseudocode or flowchart is done, open your MPLABX project and make a new copy of "basefiles.s". Name it "Lab5_Part3.s". MPLABX won't compile a project with two files that have ".main" directives in them, so right-click your original "basefiles.s" and select "do not include in build."

Every embedded program has a "main loop" which runs continuously. This program is no exception. In each iteration of your part 3 main loop, your code will read the buttons and switches (checking an input regularly is called "polling"). After reading the input devices, it will light a pattern of LEDs to match.

Your code should:

- On initialization:
    - Print a greeting message
    - Change the contents of appropriate direction registers so that:
        - pins connected to buttons and switches are configured as inputs
        - pins connected to LEDs are configured as outputs
- In the main loop:
    - Read the value of each of the 4 switches (SW1 – SW4) and turn on a corresponding LED (LD1-LD4) if the corresponding switch is "on".
    - Read the state of the 4 buttons (BTN1 – BTN4) and turn on a corresponding LED (LD5-LD8) if the corresponding button is being pressed.

There are a couple of strategies you can use to map the buttons and switches to the LEDs. One strategy is to handle each button/LED pair separately, using branches to handle the LED appropriately. This is conceptually simple, but tedious and long.

Another strategy is to use Boolean logic and left- and right-shifting to read the important parts of the PORT registers and combine them into a single register, which you can then write to the LED port.

Part 4:

Read through this section and make flowcharts for MYDELAY and your main code.

Once you've done that, add a new file called "Lab5_Part4.s" and de-activate the other ".s" files.

In this part you will learn about hardware-based delays in executing your code, and writing a subroutine to create a software-based delay. Your code will eventually create a pattern on the LED bar in which a lit LED "bounces" between the ends of the display. The period of the bounce will be controlled by the four switches. (The pattern should look like a Cylon eye: https://www.youtube.com/watch?v=faukADr0_6g)

Start by writing a subroutine with the following specification:

MYDELAY:

- MYDELAY:
  - Inputs: $a0 contains a positive binary integer.
  - Outputs: None
  - MYDELAY will do nothing for a time approximately equal to $1/16^{th}$ of a second times the value in $a0. So, if $a0 contains 3, MYDELAY will "waste" about $3/16^{th}$s of a second. If $a0 contains 0, MYDELAY will return very quickly.
  - MYDELAY is callsafe, and has no side effects.

You will need to write a loop that does nothing besides increment (or decrement) a counter, and check that counter. To handle the multiplication, you can use the MULT instruction, or else use a nested loop.

To figure out the number of times to iterate your loop, consider that the processor speed of the PIC32 is 8MHz, so each instruction takes (1/8e6) seconds. Therefore, you can solve the following equation:

Total Time = # of iterations of loop * # of instructions in loop * (1/8e6) seconds

Once you've written and tested MYDELAY, write some main code to do the following:

- When the program initializes:
  - Print a greeting message
  - Light up exactly one LED
- As the main loop iterates:
  - Read in the value of the 4 switches.
  - Use the 4 switches to set a value for $a0 that is between 1 and 16. So, if all four switches are "off", $a0 should contain 1. If switches 1 and 2 are "on", then $a0 should contain 4.

    - Call MYDELAY to delay for $a0 sixteenths of a second.
    - Advance the lit LED by 1. If the lit LED is at the end of the LED bar, change the direction along which the LED advances.

The code should continue to bounce the LED back and forth forever. At its fastest, the LED should take about 1 second to make a round trip. At its slowest, the LED should move about 1 time per second (we aren't too exact about this, so you don't need a stopwatch to test your code!) The LED's speed should change immediately after changing the switches.

Deliverables:

Submit the following to Canvas:

- Lab5.zip
    - Contains your MPLABX project (the entire folder called "Lab5.X")
    - In Lab5.X, there are at least 2 .s files:
        - Lab5_Part3.s
        - Lab5_Part4.s
- Lab5_report.txt

Have the following available for review during checkoff:

- A pin mapping table
- Pseudocode or a flowchart for part 3
- Two flowcharts for part 4:
    - A flowchart for MYDELAY
    - A flowchart for your main loop

During check-off, you will demonstrate your lab to the TA/Tutor. Be prepared to answer questions about your code's design and testing.

As usual, every document you submit to canvas or present during grading should have your name, date, lab number, section letter, and TA name.

Point Breakdown:

10 pts: Correctly reads in the 4 switches and 4 buttons

10 pts: Correctly displays the the buttons and switches to 8 LEDs

5 pts: Reads in the 4 switches and stores a corresponding number between 1 and 16 into $a0

10 pts: Has a MYDELAY procedure which takes $a0 as an argument and delays for approximately $a0/16 seconds.

5 pts: uses the MYDELAY procedure to control the timing the main loop

5 pts: Does a “bounce” pattern

5 pts: Code is readable and well-commented

10 pts: Lab Report