Brandon Rullamas – brullama

Lab section: 4 TTH TA: Shea Ellerson

Due: 5/27/2013

Lab Partner: Trieste (Fizzy) Devlin

Title:

Lab 5: Arduino/PIC32

Purpose:

The purpose of this lab is to learn about the Arduino, MIPS assembly language, and PIC32 by creating two programs: creating a program that will take in an argument as a number of milliseconds which will make a LED on the Uno32 board blink on and off, and a program that uses the stack and reverses the order of the characters in a string.

Procedure:

The procedure was relatively simple, as it was broken down into four steps as outlined by the lab pdf. The first step was to successfully compile a “Hello, world” serial port example in TeraTerm to understand how programs compile and how we would later be compiling with our own programs. This didn’t require a lot of work from us, as most of the work was done by following the TA’s steps and listening to what she said.

The second step was where work actually began, despite it being relatively simple. The first part was to make a function, myDelay, which would take in an argument of milliseconds and translate that to a blinking LED on the Uno32 board. This was done quite trivially as the TA and tutor helped outline a lot of what had to be done for the algorithm, such as using two loops to correctly get the number of milliseconds needed. The first loop was to loop the amount of times that the argument specified (in this case, 40,000, because 1 second was needed. Details of this are in algorithms and other data). The second loop was to define a millisecond based on how long it takes for each instruction to be performed. With these two combined, the function myDelay was made.

The third step was to link the algorithm of myDelay to a LED on the Uno32 board. The specifics of the LED in relation to the board was described well in the pdf, cutting down on any work that required finding out how to link the two. By using the TRISF and PORTF registers that are mapped to certain addresses, the LED would be able to turn on by simply making PORTF 0 or 1, which would correspond to off or on for the LED.

The last step, and the step that proved the most challenging, was to create a program using the stack to reverse a string. Although the algorithm behind the string was relatively simple and that the idea of the program itself can be easily executed in any high-level language, the fact that the program had to be in MIPS made the assignment very difficult. Despite it not working in our version, here is what would be done: First, the string and temporary array would be initialized. Then, each character of the string would be read until the string contained a space or null character (null characters are found at the end of strings, typically). If this was not the case, the character was pushed into the stack and counters were incremented. If it was the case, then the characters in the stack would be popped and stored into the temporary array that was initialized at the beginning of the program and then printed. A newline would then be printed following the world and then it would be repeated until a null character is found, in which the program would end.

Algorithms and other data:

There were two main pieces of ‘data’ or algorithms that were required for this lab. The first piece would be for the second part of the lab where the myDelay function had to be created. This is associated with creating the millisecond, as a computer cannot define what a millisecond is. In order to get around this, the instructions per cycle had to be calculated given that the clock frequency is 80 MHZ. Because it takes 80,000 cycles for one instruction to reach a millisecond, and there were two instructions in the loop to create a millisecond, only 40,000 cycles had to be made for the outer loop to define a thousand milliseconds, or one second.

The second piece of data required would be in the fourth part of the lab, where push and pop are needed. The main concepts of push and pop are discussed in CMPS 12B, but that class has already been taken, so the main difficulty was applying the concepts to a different scenario. A stack follows a LIFO data structure, which stands for last in, first out. This applies to the two main concepts that go with stacks: push and pop. A push adds things into the stack, and a pop removes things from the stack. This allows us to create a reverse order of words that the program dictates due to the LIFO structure behind the stack. For example, if you push “Hello” into a stack, the stack looks like so:

O

L

L

E

H

Because the “h” is at the beginning of the word, it is the first character to be read in and consequently pushed into the stack. This puts it at the bottom of the stack. This is repeated until a space is found in the string, which in this case of the program would be “Hello world”. So, it would stop right before the “world”. The stack would then be popped and stored into a temporary array that would print the reversed word, as so:

```
O    L    L    E    H

L    L    E    H

L    E    H

E    H

H
```

The first iteration of this would leave the array empty, with the second iteration giving the array [O], and then [O][L], [O][L][L], [O][L][L][E], and finally [O][L][L][E][H]. By printing out this temporary array, you would get a reversed string.

What went wrong or what were the challenges?

The main problem with this lab was not even the program itself or the algorithms behind it. The program was in compiling the lab and trying to test our code to see if we were able to successfully achieve what we were trying to do. As a result, the code for lab5b.s is simply a shot in the dark as to what we think the finished project would look like, as we were unable to compile the code and test it for accuracy. This was due to the compiler complaining about two things: a str operand that should have easily been passed (this could have been bypassed by commenting out the code, but we thought it was crucial to the program), and the compiler complaining about missing pic32 files. The reason that this was a challenge was that the compiler was complaining about the str operand in the lab5b.o file, not the lab5b.s file. The missing pic32 files were also strange due to the fact that we had no problem compiling and running lab5a.s so that we could test it. Many different attacks were set forth on the program to try and fix this problem, but with the looming midterm on the next day, it was hard to focus on such a random problem.

Other information:

When you configured PORTF bit 0 to be an output port you wrote a 1 on bit 0 of TRISF clear register. If we want to configure PORTF bits 3 and 4 to be an input port, what do we need to do? Include the code that will configure PORTF.

To configure PORTF bits 3 and 4 to be an input port, we would need to set TRISF register bits 3 and 4 to a 0. Once that is done, bits 3 and 4 of PORTF can be written. The code for this is:

```
li      $t9, 0x0        # li = pseudo op to load an immediate value into a register, 0 => $t9

li      $t8, 0xbf886140       # load address of TRISF into $t8
```

```
sw      $t9, 4($t8)       # store $t9 into address defined by $t8 plus an offset of 4
```

What did you have to do to calibrate your delay function? How did you calculate the number of times you have to loop?

To calibrate our delay function, timing was simply done once the function was made to ensure that the LED was blinking on and off every one second. In order to figure out the values that would ensure that the LED blinks at the appropriate time, the instructions per cycle algorithm and data information was used, which is outlined in the "Algorithms and other data" section. In short, because we were using an 80MHz clock frequency, and there were two instructions in the loop to define a millisecond, we needed 40,000 instructions (80Mhz/2).

What happens if you put a serial output in your delay function/loop? How does this impact the delay?

If a serial output is put in the delay function/loop, it will have two main results. First, the program will print out whatever is in the serial output thousands of times due to it passing that instruction every time the loop executes. The second result ties in with the second question, as it will increase the delay due to there being 3 instructions instead of 2. This will cause 120,000 instructions to be executed instead of 80,000, which translates to there being a second and a half delay instead of just a second.

What happens if you forget to put a nop after a branch?

If you forget to put a nop after a branch, the commands itself will be messed up as the branch needs the nop to properly execute. It could also have other effects, such as in the myDelay program, where the amount of instructions is key. In this case, it could decrease the amount of instructions per cycle which would make the blinking go faster than expected.

If we are reading a sequence of integers instead of a string of characters, what necessary changes do you need in your program to make it work?

In order to read in a sequence of integers instead of a string of characters, the program would need to identify the integers as characters instead of integers. This can be done through ascii values or by creating a string out of the integers, which would then operate the same way that the current program would.

Conclusion:

This lab was difficult simply because it involved learning a whole new system (Arduino) and all of the processes that came with it as well as trying to study for the midterm that was quickly soon after. The algorithms behind each of the programs and learning how to connect the board to the program was not as bad as it seemed at first, but trying to get past the whole new software and trying to learn how to successfully compile and run the code proved to be the biggest challenge. This lab served its purpose in showing how MIPS works as well as how the Arduino communicates with the computer, though.

Extra:

The algorithms and tips given by the TAs were helpful. This lab made me appreciate high level languages a lot more due to how more simple this lab would have been through the use of a language like Python or Java as well as using the compilers that are associated with each one, such as using Eclipse with Java, which would have made this entire process easier.