Hui Shi Li

Partner: Sarah Borland

Section #6

TA: Daphne Gorman

3/15/2015

Lab 5

Introduction: The purpose of this lab is to understand how to write a function in assembly that will control the blink rate of lighting the LEDs and create an interrupt that will blink all LEDs for 0.5 seconds.

Procedure:

Part 1: Following the tutorial, we learned how to use MPLAB to build and debug a project.

Part 2: We looked at the execution memory to trace the printf statement to find how many levels deep the call is.

Part 3: We first pushed the return address onto the stack, in order to return after the printf statement. Then we load the address of the string into a0, the first argument of the function and load the address of the counter variable into t1. Next we load the word of the address in t1(counter) into a1, the second argument of the function. After we jump and link to printf to print the string. We increment the counter by one and pop off the stack, to get the return address and return.

Part 4: In order to light the LEDs with the correct delay. First we printed out the values of PORTD(status of the switches) for each switch combination. Then we created a linear fit and figured that we can calculate the delay with the following equation: DELAY_IN_SECONDS = PORTD_VALUE/256 To get the PORTD value, we "anded" the actual value by the mask xF00 to get only bits 8 through 11, which are the switches. Once we calculated the DELAY_IN_SECONDS, we then multiply that value by the clock time which is 80000 to get the actual delay. To return the actual delay, we stored it in v0.

Part 5:

Before setting up the interrupts, we check if the timer has been set(first bit of T1CON). If it has then we skip setting up the interrupts, otherwise we continue with doing the following:

- clearing T1CON- clear the value in T1CON using 0xFFFF
- setting the prescalar(bits 4-5) using 0x0030
- clearing the count- use xFFFF to clear TMR1
- set PR1 value
- set the interrupt priority T1IP(bits 0-2) using 0x0004
- clear prior interrupt using 0x0010
- enable the interrupt using 0x0010
- turn on the timer(first bit of T1CON) using 0x8000

*Note: We use +4 offset to clear and +8 offset to set/enable

Then in the ISR, we clear the interrupt flag(IFS0) and set the first 8 bits of the LEDs(PORTE).

Results: A lot of the mistakes that we made in this lab was due to not fully understanding what the lab was asking us to do. In part 3, we thought that we had to implement the printing of the "Hello World" string in C. But then found out that we had to implement it in the assembly code. Part 3 correctly outputs "Hello World!" along with the counter that is incremented each time it is printed.

Part 4 changes the delay between the LED lights depending on the state of the input switches. If no switches are on, there is no delay(constant moving of LEDs). If all switches are on, the delay is 15 seconds and each LED will be moving every 2 seconds.

Part 5 makes the LEDs blink every 0.5 seconds while still maintaining the delay as in part 4 with the switch input.

Discussion:

1) "li $t0, 0xBF886110" is an instruction in assembly. It loads the immediate value in the address 0xBF886110 into the temporary register, t0.
2) "mask == mask << 1" which is a left shifts the mask by 1 gets compiled into:
   LW v0, 16(S8)
   SLL v0, 16(S8)
   SW v0, 16(s8)
3) The LEDs are on before executing "PORTECLR == 0x00FF" because we enable the LEDs by setting the TRISE register using TRISECLR = 0x00FF.
4) The memory ranges are from 0x80000_0000 to 0x80000_3FFF. The type of memory is data memory.
5) The memory ranges are from 0x9D00_0000 to 0x9D01_FFFF. The type of memory is program memory.
6) 62 instructions * 32 bits = 1984 bits of memory
   17 characters * 8 bits + 32 bits * 1 word = 168 bits for data
7) I think the program does in increase in size when in debug mode because it has reserved memory and resources to perform the checks on the code.
8) To read from an input switch, we first load the address of PORTD. Then we load the word of that address, to get the value stored at that address.
9) 3 levels deep. The functions are printf, vfprintf and fputc. The addresses are 0x9D00054C, 0x9D000550 and 0x9D000554. The arguments are addiu sp, sp, -32, addiu sp, sp -104 and lui v0, 12(a1).
10) We saved the ra register, return address onto the stack. We stored the number of times it has been called in a variable called Counter.
11) We output the PORTD values to the screen and created a linear fit from these values to find an equation, where x is PORTD value and y is the delay in seconds. A delay of 0 has all the LEDs constantly moving.
12) The ISR has to be as short as possible because additional instructions will make the time less accurate.

Conclusion: This lab taught me how interrupts work and how the stack works. I also learned how to combine C and assembly code. Overall, this lab took a lot of trial and error especially with part 5. It also took a lot of understanding of what you need to do, as in part 3 where we implemented it in the C code rather than in assembly.