Jason Ou (jaou 1385128)
Partner: Vanessa Putnam
CMPE 12L
11 MARCH 2015
TA: Daphne Gorman
Section: 6

*NOTE: I finished the lab early and had it checked off already the week before Monday, and I did most of the write up as well, but forgot to turn it in on time since I was studying for finals, if possible, can I get deducted late just for the write-up and not for overall of this lab?

Lab Write-Up #5

OVERVIEW/INTRODUCTION

The purpose of this lab is to learn how to configure the processor to understand time in human terms by interfacing with the outside world. We will use a new type of function in this assignment called an interrupt to trigger the execution of an Interrupt Service Routine (ISR).

METHODOLOGY/PROCEDURES

PART A: TUTORIAL

We have to read through and experiment with the options in the entire tutorial, then we run the demo.c program and answer some of the initial questions at the end of the tutorial.

PART B: DEBUGGING

We have to trace through the execution memory of the C program and observe what happens. To view the addresses and return addresses that are used in the jal and jr instruction, we have to look at the CPU registers. In the CPU register, as we trace through the C program, we can see that the call of the program is three levels deep.

PART C: SUBROUTINES IN MIPS

We have to write our own function that will return a constant 0x80000 for the delay loop in the demo.c program and print out the number of times that it has been called to the serial output. We have to combine both the demo.c file with a .s file that will allow the demo.c file to call a function from the .s file which will execute the MIPS code and then return back to the demo.c program.

PART D: I/O IN MIPS

In this section, we have to "tune" the delay values using the input switches. Depending on the value of the switches, we have to return a value of (1 second delay) * (switch value). To do this, we read from PORTD and then bitmask the value of PORTD with 0x0F00 and shift that value to the right 8 times to get the value of the switches. From there, we multiply that value with the value of (1 second) and then return that delay back to the demo.c file.

PART E: INTERRUPTS

We have to create a program that will make all of the LEDs blink with a 0.5 second delay between turning on and off. To do this, we have to enable interrupts and then create a subroutine that will then turn on and off the LEDs all at the same time. To set the delay, we have to configure PR1 with the time that we want. This part is used for learning how to configure an ISR.

RESULTS

PART A: TUTORIAL

The tutorial taught me how to run the demo.c program, do simple debugging, set breakpoints, and other MIPS instructions. I also learned the basic layout of a MIPS .s file.

PART B: DEBUGGING

Jason Ou (jaou 1385128)
Partner: Vanessa Putnam
CMPE 12L
11 MARCH 2015
TA: Daphne Gorman
Section: 6

The functions are printf at address 9D00_470C then vprintf at address 9D00_0000, and finally vfprintf at address 9D00_3868. By looking at the disassembly code of the C program, we can see that there are three arguments and the arguments are ADDIU SP, SP, -32 → ADDIU SP, SP, -104 → LHU V0, 12(A1).

PART C: SUBTOUTINES IN MIPS

The output looked like "Hello, World! 0" → "Hello, World! 1" → "Hello, World! 2" and etc. In the end we got the demo.c file call from the lab5.s file. We had to call from the .s file by using "extern in getDelay();" and then create a getDelay() function in the .s file. Using a stack, we were able to store return addresses into it so that we could call from .s file and then return back to the demo.c file.

PART D: I/O IN MIPS

We used PORTD to read in the input values of the switches. From there, we were able to use these values to determine the speed our LEDs should be after we bitmask and shift the value of the PORTD input to the right. Depending on the value of the switch, we then multiply the value of the switch with a constant equal to the delay of 1 second and then return that value to create a delay for the LEDs. In the end we got it to work and it resulted with the bigger the value of the switches getting turned on, the slower the LEDs moved.

PART E: INTERRUPTS

In the end, we got the configurations for the interrupt to work correctly and initially the lab wanted a delay of 4 seconds, but the value for the delay of four seconds was too larger, so we had to change the delay to 0.5 seconds. We ended up creating a program that would blink all of the LEDs on and off with a delay of 0.5 seconds between each change in LED value. All we had to do was configure a function to enable interrupts and then create a subroutine to enable or disable all of the LEDs.

DISCUSSION

1. **Is an instruction or pseudo-operation in the assembly? Explain.**

    - "li $t0, 0xBF886110" is an instruction in assembly. "li" directly translates to a machine instruction.

2. **What instruction does the "mask == mask << 1;" get compiled into?**

    - Disassembly Code :
    - LW V0, 16(S8)
    - SLL V0, V0, 1
    - SW V0, 16(S8)
    - This is a bitwise shift to the left by one.

3. **Which LEDs are on before executing "PORTECLR == 0x00FF"? Why do you think this is?**

    - Before executing "PORTECLR = 0x00FF", we execute "TRISECLR = 0x00FF" which will enable LED outputs 0-7 by setting TRISE register. You need to enable the LEDs first before you can use them later on. Similar to initiating a variable.

Jason Ou (jaou 1385128)
Partner: Vanessa Putnam
CMPE 12L
11 MARCH 2015
TA: Daphne Gorman
Section: 6

4. **In what memory ranges does the program store your data such as the mask in the demo program? What type of memory is this?**

    - The memory ranges from 0x80000_0000 to 0x80000_3FFF and this is the range for Data Addresses. The type of memory is virtual memory.

5. **In what memory ranges does the program store your instructions? What type of memory is this?**

    - The memory ranges from 0x9D00_0000 to 0x9D01_FFFF and this is the range for Memory Flash. The type of memory is virtual memory.

6. **How much memory do the instructions in your final program occupy? How much data does it use?**

    - 54 Instructions * 32 bits = 1728 bits of instruction memory.
    - (17 characters * 8 bits) + (3 words * 32 bits) = 232 bits of data being used.

7. **Does the program or data size change when it is in debug mode? Why do you think this is?**

    - The program/data does change in size when it is in debug mode. This is because while the program is in debugging mode and the PIC32 is plugged in, there is more for the program to manage while debugging. Therefore, data size increases and less program memory is used as the actual controls are moved onto the PIC32.

8. **How do you read from an input switch?**

    - To read from an input switch, we load in the address of PORTD and then load word of the value stored in the address of PORTD.

9. **In part 2: How many levels deep do the calls go? What functions? What addresses? What are the arguments at each level?**

    - The calls go 3 levels deep.
    - The functions are printf → vfprintf → fputc.
    - The addresses are 9D00_470C → 9D00_0000 → 9D00_3868.
    - The arguments are addiu sp, sp, -32 → addiu sp, sp, -104 → lhu v0, 12(a1).

10. **In part 3: What registers did you save and why? Where do you store the number of times that the function has been called?**

    - We used $s0 to $s1 to save values to be loaded onto the stack
    - $sp was used to save to the stack pointer to get the current location of the pointer address
    - $ra was used to save accurate return address to be used to return to demo.c

Jason Ou (jaou 1385128)
Partner: Vanessa Putnam
CMPE 12L
11 MARCH 2015
TA: Daphne Gorman
Section: 6

- $t0-$t7 was used to store temporary variables that were used for quick access
- $a0 to $a1 were used to save argument variables to use with printf

11. **In part 4: How did you compute your delay numbers and what are they? What does a delay of 0 look like?**

    - To compute the delay numbers, we used trial and error, guessing and checking. We tested a variety of different numbers and timed the LEDs while the PIC32 ran. A delay of 0 would result with all the LEDs flashing on, while as the delay increased, the LEDs turned on slower.

12. **In part 5: Why do ISRs in particular need to be as short as possible?**

    - ISRs in particular need to be short as possible to ensure worse case clumping of ISR executions doesn't overload your CPU.

CONCLUSION

With Lab 5, we were able to gain the knowledge of learning how to use and program using the PIC32. We learned to use function calls, returns, IO, and interrupts. We also learned how to program in C and MIPs. We learned how to enable and manipulate hardware by configuring the PIC32 using the MIPS IDE. We now gained a better understanding of how to use a higher level language beyond assembly.