CMPE 12L Lab 5 - Spring 2014
Arduino/PIC32

Due: Mar 2, 2014
100 Points (40 Report, 60 Work)

1 Background

In this lab assignment you are asked to create two assembly program using MIPS assembly language. Your first program will be to create a delay function that will accept an argument as number of milliseconds. You will then configure a light emitting diode (LED) on the Uno32 board to turn on and off according to duration of your delay subroutine. For your second program, you will use a stack to reverse the order of characters in a word. You will be given a string of characters. Your program will reverse the characters in the words but preserve the order of the words.

2 Prerequisites

- Read through this **entire** lab assignment.
- Review the lecture notes on the PIC32 architecture.
- Read about MIPS assembly: `http://en.wikibooks.org/wiki/MIPS_Assembly`
- Read section 12.1-12.2 of Sec12_IO.pdf in eCommons
- Read section 3.3 of Sec03_Memory.pdf in eCommons
- Look at Table 3-31 on page 75 of PIC32_6114H.pdf in eCommons
- Chapters 1 and 2 are included in eCommons for reference: Sec01_Intro.pdf, Sec02_CPU.pdf

3 Tutor/TA Review

Your lab tutor/TA will cover the following items in the first portion of the first lab:

- PIC32 register organization (`http://www.johnloomis.org/microchip/pic32/cpu/registers.html`)
- Downloading/setting up the lab5.zip which contains template files and starting batch files. (Need to edit paths to MPIDE.)
- What's required

4 Assignment

This assignment has 4 parts. You should finish first part 1 quite quickly, but the other three parts may take significant time. Parts 2 requires writing your first MIPS assembly program. Part 3 requires learning about the PIC32/Uno32 hardware to control an LED on the board. Part 4 requires writing some significant assembly code.

4.1 Preliminary

Before you start, make sure you can compile/assemble the "Hello, world" serial port example and see the results in TeraTerm. You need to understand this to even start the next portions of the lab. This is also the place where your lab TAs/tutors can help the most. Once you are debugging your own program, they won't be as much of a help since your bugs are specific to your program!

4.2 Instruction Timing

In the first part of this assignment, you are to write a function with the label "mydelay" in lab5.s. The function should take a single argument from register a0 that is the number of milliseconds (ms) to delay before returning. Functions like this can be used to synchronize events external to the chip by waiting variable amounts of time.

In doing this, you should do the following:

- Use serial print statements before and after a call to your delay function so you can estimate "wall clock" time using a watch (or your phone).
- Your code should convert the time (ms) into a number of loop iterations. This might require using a loop-within-a-loop to get the whole range.
- Calculate the total number of instructions in your function.
- Calculate the instructions per cycle (IPC) given that the clock frequency is 80 MHz (cycles per second).
- Calibrate "mydelay" by using at least 2 delay data points such as 1000ms and 5000ms. (HINT: You can add instructions inside the loop or change the loop bounds.)
- COMMENT YOUR CODE!!

4.3 Input/Output

In this section, you should implement create a program that will blink LED 5 every 1s.

Input/output in our microcontroller is based on device registers mapped to certain addresses. Specifically, LED 5 is connected to pin 43 on the board. This is pin 58 on the microcontroller which is digital IO port F. Specifically, bit 0 (RF0) is the bit that controls the LED. This information is available in the Uno32 document from Digilent (chipKIT-Uno32-RevC_rm.pdf) and the PIC32mx reference guide from Microchip (PIC32_61143H.pdf). Both are in the class resources on eCommons.

The port F address starts at 0xBF88_6140 and is shown in Table 4-31 of PIC32_61143H.pdf. There are 4 registers that let us configure the input/output: TRISF (0xBF88_6140), PORTF (0xBF88_6150), LATF (0xBF88_6160), and ODCF (0xBF88_6170) which are described in Section 12.1-12.2 of the PIC32 family reference manual in eCommons (Sec12_IO.pdf). The document

describes all the ports as TRISx where x can be A to G for each of the IO ports (A to G) at different addresses.

Each of the above registers actually has four addresses that perform different functions. The base register can be read or written to change all of the bits at once. Three additional registers are used to modify one bit at a time: clear (offset 4), set (offset 8), and invert (offset 12). The ones in a mask specify which bits to clear, set or invert. So, for example, if you want to set bit 0 of the TRISF register, you can write a binary mask of 0x1 to address 0xBF88_6148 (0xBF88_6140 + 8).

For this lab, you will only need to use the TRISF register and PORTF register. First configure PORTF so that it will be an output port by setting TRISF register bit 0 to a 0 (active low). Once you have TRISF bit 0 set to a 0, you can now write data to bit 0 of PORTF. Setting PORTF bit 0 to a 1 will turn on LED 5 and setting to a 0 will turn off LED 5.

4.4 String Reversal

In the last part of this assignment, you will use the stack to reverse a string. Your program should behave as follows:

- Print a string stored in memory
- Read each character of the string until you reach a space or a null character
- If the character is not a space or null character, push it in the stack.
- If the character is a space or a null character, then reverse the string by popping each character, storing it in a temporary array, and printing it. If the character was a null character, end the program after printing the word.
- Go back to step 2

The program should reverse the characters in a word but preserve the order of the words. The reversed words should be separated by a new line. For example if your string is "Hello world", your program should output the following:

Hello world
olleH
dlrow

The file lab5b.s contains a starter code that you can use for this part of the lab. In there contains a variable msg, which contains the string you will reverse, and a variable tmp, which is an array of 10 address spaces that will hold the reversed word to be printed. A subroutine called initTmp is also included to clear the content of your temporary array.

5 PIC32 Assembly

5.1 Reference Material

There should be coverage from the lectures as you complete this lab, but in the meantime, you can also look at this web page to see detailed information on the MIPS assembly architecture and assembly language:

http://en.wikibooks.org/wiki/MIPS_Assembly.

There is also an extra resource at:

http://www.johnloomis.org/microchip/pic32/resources.html

Some of the items refer to the "MPLAB IDE" and "MPLAB Simulator" but we are using "MPIDE" which is different.

There is a ton of information on PIC32 here:

http://www.microchip.com/pic32

Specifically, if you click "Resources" and "Training" there are extra videos. The 4 minute video on the execution pipeline and the 18 minute architecture overview are both good, but are not necessary for this assignment. If you click "Data Sheets" on the left, the PIC32MX3/4 Data Sheet has the complete information on the hardware. We also provide some of these documents in the Resources section on eCommons.

5.2 Program Structure

Our assembly files will be a bit more complicated than before. We provide a template program called lab5.s that contains the skeleton code for a function "myprog" that is called from the main.cpp program after initializing the board. **You should not create extra assembly files other than what is required (lab5a.s and lab5b.s) and you should not modify main.cpp.**

There are a few things of importance in this program structure:

- .global myprog
  This puts the symbol "myprog" into the symbol table of the object so that external code (specifically main.cpp) can call the function.

- .text
  This specifies the "instruction" part of your program.

- .set noreorder
  This tells the assembler not to reorder instructions in a "smart" way to maximize performance.

- .ent myprog
  This specifies where the entry to myprog starts.

- myprog:
  This is the label of the address for myprog. Note that labels here require a colon after them.

- .end myprog
  This specifies the end of your program.

- .data
  This specifies the "data" part of your program.

6 Debugging over Serial

In the last assignments, we used the LC-3 simulator and could easily step through our program and see the state of the processor. Now, however, the processor is running and we cannot see what is going on inside. How do we debug it? One solution would be to use a simulator as before. While this may be useful for early bugs, we always need to find late bugs on the hardware itself.

Most of these debugging capabilities are implemented using JTAG (Joint Test Action Group) which is the short name for a serial link between a host computer and the chip you are debugging. This serial link allows input to be sent to the chip and output to be read from it. In the extreme case, a special debug language will allow you to step through the program on the actual chip. This is often known as "In Circuit Emulation".

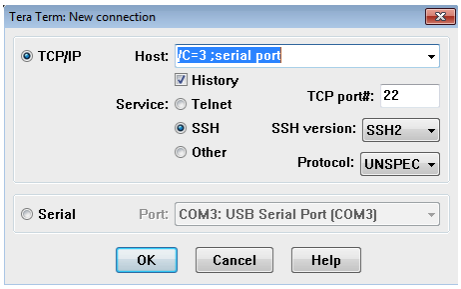
In our case, we are going to use some Arduino serial utilities that enable simple input/output. These are actually "emulated" serial routines that use the USB cable and they are automatically included in the MPIDE GUI (see the example DigitalReadSerial). We include these same routines in the main.cpp file and a library core.a. We initialize the board and setup the serial port, "Serial.begin(9600);". We can follow the same examples if we wanted to call the serial port routines in our main.cpp, but how do we call this from our assembly routines?

If you look at core.a using pic32-nm, the symbol for the Serial.println() method is _ZN5Print7printlnEPKc. In order to call this, we need to load two values into registers first. Register a0 should contain the address of the C++ object Serial and register a1 should contain the address of the ASCII string to display. For example, we can do this using the pseudo-op load address (la) to formulate the "Hello, world!" program from lab 3:

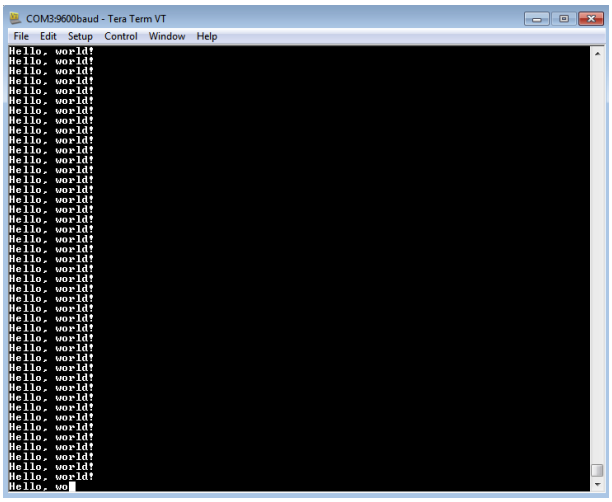
```
/* load the address of the serial object */
la      $a0,Serial
/* load the address of the ASCII serial message */
la  $a1,Hello
loop:
/* call the function */
jal     _ZN5Print7printlnEPKc
/* fill the branch delay slot */
nop
/* infinite loop */
j loop
Hello:
.ascii "Hello, world!\0"
```

Note that the function name is funny "_ZN5Print7printlnEPKc". This is a side-effect of C++ to uniquely name over-riden functions. This is beyond the scope of this class, but extra reading is specified at the end of this lab for those interested.

Once we add this code, assemble our program, and upload it to the PIC32, we can start a terminal program (TeraTerm) to communicate between the host and the board. In TeraTerm, the serial port should be configured as COM4 as shown in Figure 1(a). By default, it should use 9600 baud. If you do this, while running the above program, you should see the serial output as shown in Figure 1(b).



(a) TeraTerm configuration should select COM 3 (/C=3).



(b) TeraTerm serial port monitor.

Figure 1: TeraTerm usage examples.

7 Format

Your program must be a text file with assembly code that assembles with no errors.

8 Lab Submission

Your lab will be submitted via your eCommons account. Please log in to eCommons using your UCSC account and attach the following files to your "Lab 5" assignment submission:

- lab5a.s which contains the Blink implementation
- lab5b.s which contains the string reversal implementation
- lab5_report.pdf

Note that the final report must be submitted in PDF format.

Make sure to confirm that your assignment is SAVED and SUBMITTED before the deadline. You may resubmit your assignment an unlimited number of times up until the due date.

8.1 Check-off

For this lab, as with most labs, you will need to demonstrate your lab when it is finished to the TA or tutor and get it signed off. You will also need to submit your lab files using eCommons.

8.2 Grading template

This is a suggested grading rubrik. It is also a good general guideline before submitting your lab to check off these points.

8.2.1 Requirements

☐ (30 pts) myDelay subroutine

- Does LED 5 properly turn on and off? (5 pts)
- Is a delay of 1000ms reasonably accurate? (5 pts)
- Has the function been calibrated for other delays? (10 pts)
- Does the delay work for the full range of 32-bit inputs by using multiple loops? (10 pts)

☐ (30 pts) String Reversal

- Can they easily recompile to display different strings? (5 pts)
- Does the program properly reads characters from the string and recognize the space character? (5 pts)
- Is the stack being used correctly? (10 pts)
- Does the program properly outputs the characters in reverse but preserves the order of the words? (10 pts)

☐ (5 pts) Extra Credit

- Does it recognize multiple space characters and reverses only alphnumeric characters(5 pts)

8.2.2 Lab write-up requirements

In the lab write-up, we will be looking for the following things. The lab report is worth 40 points. We do not break down the point values; instead, we will assess the lab report as a whole while looking for the following content in the report.

Along with the usual items, you should answer the following questions:

- 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 a code that will configure PORTF.
- What did you have to do to calibrate your delay function? How did you calculate the number of times you have to loop?
- What happens if you put a serial output in your delay function/loop? How does this impact the delay?
- What happens if you forget to put a nop after your branch?
- 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?