EE108B Lab Assignment #1

MIPS Assembly Programming

Due: Tuesday, January 31, 2012

1. Introduction

Throughout these labs, you will be designing a processor that can execute programs written in MIPS assembly. Once the hardware support for software has been built, a large variety of tasks can be performed without having to constantly modify the hardware design as in EE108A. To demonstrate how software can perform tasks previously done with hardware, you will be implementing in MIPS assembly the classic game of Pong.

2. Requirements

We will provide the implementation of a MIPS processor, as well as controllers for the VGA monitor and Sega gamepad. You must write a MIPS program for single-player Pong and demo the emulated version.

IMPORTANT: Please implement Lab 1 on the **myth** machines (do **NOT** use corn or the Packard 129 lab computers). To access these computers, you should either go to the actual cluster (located in the basement of Gates) or ssh to the cluster from Terminal/Command Line like this:

ssh SUNETID@myth.stanford.edu

where SUNETID is your SUNETID.

There will be one paddle at the left edge of the screen which can be moved vertically using the Sega gamepad. Meanwhile, a ball will be bouncing off the edges of the screen, and the user must move the paddle so that it prevents the ball from reaching the left edge of the screen. The game ends when the ball reaches the left edge, and a new game can be started by pressing one of the buttons on the gamepad. If you complete these minimum requirements, you may choose to implement any of the following extensions:

-Vary the speed of the ball based on where it hits the paddle
- Have paddles on multiple sides, so the goal is to prevent the ball from hitting any edge
-Increase the speed of the ball as the game progresses
- Implement the game Breakout

3. Implementation Details

Writing the Program
A skeleton file, pong_skeleton.s, has been provided on the course website. It initializes some constants (for example, the screen dimensions and colors) and provides an outline of what needs to be added.

It is highly suggested that you follow this guide. Also, you *must* comment your code. Note that the instruction memory is limited to 256 words, while the data memory is limited to 64.

We were able to create a working game with under 100 lines of Assembly, so there should not be any problems with space if you are careful. However, the 256-instruction limit may prevent you from implementing all of the extensions.

The following instructions are *not* supported in our processor implementation so should not be used in writing your program:

Divide/Multiply:	`div divu mult multu`
Branch:	`bczt bczf bgezal bltzal`
Load:	`lb lbu lh lhu lwcz lwl lwr`
Store:	`sb sh swcz swl swr`
Move:	`mfhi mflo mthi mtlo mfcz mtcz`

Handling I/O Through Memory-Mapping
In lab 1, there is one memory-mapped I/O device: the VGA monitor (with address 0xFF).

The following instruction will draw a specified color at the specified coordinate:

```
sw  $s0,  0xFF($0)
```

The color and coordinate are specified in a single register as follows:

31-19	18-16	15-8	7-0
0	3-bit color	x coord	y coord

The 3-bit color consists of one bit each of red, green, and blue.

The VGA monitor has been divided up into a grid of 40 x 30 squares. So, while the X and Y coordinates have been allotted 8 bits each (for a maximum value of 255), the maximum values in this game are 39 and 29 respectively. Because of this lower resolution, coloring the coordinate (X, Y) will actually color a small square instead of an individual pixel. (0,0) is the top left corner of the screen.

Using the XSPIM emulator for Pong

The starter code does nothing beyond draw a single red dot on the screen (the ball).

You can test your implementation by using xspim and a perl script, pong.pl, which are provided from the zip file on lab website (you may need to make xspim executable with the command: chmod a+x xspim). Make sure you copy trap.handler wherever you put xspim and pong.pl. From your AFS account, type the following.

```
perl pong.pl &   // This should open up a small graphics display
./xspim  -file your_pong.s &
```

Note that, while you are free to design and test your implementation on any machine you like, the lab will be graded on the myth machines, so we suggest you use those.

Once the xspim window has appeared, click on run (if these windows do not appear, you may need to pass the –Y flag to your ssh command.) This displays the ball bouncing from the paddle in the small display (provided that your implementation is correct). Note that you will have to write extra code to move the paddle on its own since you do not have a way of moving it externally. Finally, to exit from xspim and pong.pl, you may have to FORCE EXIT on both windows (click on the top left corner to see this option).
Note that you cannot use the lower addresses in xspim since that region is reserved for instruction memory. We recommend using the stack pointer, $sp, which is already initialized by xspim to point to the highest memory location.

For Lab 1, since there is no user input, please make it so that the paddle tracks the ball. Note that this means that the game will play itself, and that it should never lose since the paddle should always be tracking the ball.

4. Submission

You must demo your program in lab on **1/31/12 in office hours (7-9 PM)**. By **11:59 PM**, submit your report to the course website (details TBA). A template report will be posted soon. Failure to use the soon-to-be-posted format rules will be penalized. Note that you only need A2, not A1 or A3. Also, you don't need to report any performance statistics for Lab 1.

- Also, submit your pong assembly code in your PDF report. Your program should be well-organized and **commented**. We will be looking to see that you used the MIPS conventions regarding register usage and function calls and returns (see book).
- In addition, in your report answer the following question: What are the advantages and disadvantages of the softw approach to this specific project? Give specific examples of things that are more and less efficient in the software implementation. Even if you did not implement any extensions, think about what types of extensions would be easier to do with the software approach, and what types would be easier to do with the hardware approach.

Grading:

Here is the rubric that will be used in grading your lab:

Lab Total is 15 points.

Report is 5 points:
- +1 pts general format per handout
- +2 pts sw vs. hw trade-off
- +2 design discussion

Code is 5 points:
- +1 pts Turning it in :)
- +2 pts MIPS conventions
- +2 pts commenting code

Demo is 5 points:
- +5 pts works
- +1 per extension