CS/COE 0447 Fall 2009
Lab 1: Introduction to MIPS
Due Date: September 17, 2009

To get credit on this lab, attend recitation on 9/11. Each of you should submit your own solution, according to these instructions: http://www.cs.pitt.edu/~sab104/teaching/cs447/submission.html. You may collaborate with your partner, but each person must turn in their own copy of the lab, with the name of their partner. The lab is due on 9/17 before midnight.

Part 1: Getting started with MIPS assembly language

Launch the MARS simulator (if it's not installed in your computer or you want to run it somewhere else, you can download it from http://www.cs.missouristate.edu/MARS/). Click on the Edit tab on the top left (in this pane, you'll see your assembly language program. Well, after you've created it). Go to File and click on New, to start a new program.

Let's start with a simple program. Start the program with:

```
.text
```

This says that the following are program instructions (and not, e.g., data). Let's do the following computation:

```
$t1 = 3 + 5 + 8
```

First, let's put 3 in $t1:

```
addi $t1,$zero,3
```

This says to take what is stored in register $zero, add 3 to it, and put the result in register $t1. $zero ALWAYS contains 0, so this puts 0 + 3 into register $t1.

Before you can go further, MIPS needs you to save the file. Now, go to the Run tab and click on assemble.

The Text segment shows you:

Address -- where this instruction is stored in memory
Code -- the machine code of the instruction
Basic -- [not too helpful at this point]
Source -- the original instruction you typed in

The Data Segment in the middle of the page shows you the contents of the part of memory where data is stored. We haven't put any data in memory, so the values here are all 0. The bottom window gives messages from the simulator. Any error messages will be displayed here.

Click on Go (the green arrow) to execute the machine code (technically, "simulate" it). Look at $t1 in the panel on the right.
Ok, let's continue writing the program (to get back to your program, click on Edit).
Now we want to add 5 to our running sum:

```
addi $t1, $t1, 5
```

And then we want to add 8:

```
addi $t1, $t1, 8
```

Assemble the program again, and run it. You will see that $t1 contains 0x00000010. The 0x just means that the number following it is in hex.

- **Question 1: What decimal number is 0x00000010?**

Assemble the program again (you need to assemble it again to be able to run it again) . Click Step under Run (or the triangle with the 1). This will step through your instructions one by one. As each instruction executes, you will see registers and memory being updated.

- **Question 2: What is the value of the program counter before the first instruction is executed? (the program counter is the register labeled "pc")**
- **Question 3: What is the value of the program counter after the first instruction is executed?**
- **Question 4: What are the values of the program counter and register $t1 after the second instruction is executed?**
- **Question 5: What are the values of the program counter and register $t1 after the third instruction is executed?**
- **Question 6: By how much is the program counter incremented after each instruction is executed?**

A bit is a 0 or 1.
A byte is 8 bits.
A word is 4 bytes.
In a MIPS architecture, each byte has its own address (the byte is the "addressable unit").

- **Question 7: How big are instructions in MIPS?**
  Number of bits:
  Number of bytes:
  Number of words:

This shows a good way to learn assembly language. Enter and assemble instructions; this will show you the machine code that is produced. Then, step through execution so you can see the effects of the individual instructions.

Part 2: Memory

Before working with memory, you need to be able to add hex numbers. Here is a simple example of adding 2 hex numbers:

```
   1010
+  001A
  --------
   102A
```

Keep in mind that this is a very simple example, since we are not dealing with carries (we'll learn how to do that later in the course
Here is how a human does this. Note that numbers without the 0x are decimal:

Digit 0: 0x0 + 0xA = 0 + 10 = 10 = 0xA
Digit 1: 0x1 + 0x1 = 1 + 1 = 2 = 0x1
Digit 2: 0x0 + 0x0 = 0 + 0 = 0 = 0x0
Digit 3: 0x1 + 0x0 = 1 + 0 = 1 = 0x1

Now, you try one:

- **Question 8:**

  ```
     1000000
  + 000002C
    ------------
  ```

Now, let's look at memory. Start by entering the following into the simulator:

```
.data
.word 0x25,0x3,16,25,3,0x44,0x33,0x22,44,33,22
```

".data" is an assembler directive (instruction to the assembler) that tells the assembler we are in the data segment of memory. ".word" is an assembler directive that tells the assembler to store the following into subsequent words in memory. The size of each value is 4 bytes. The 0x values are in hex. The other values are in decimal. The data segment of memory begins at address 0x10010000.

Before assembling your program, try to figure out what hex values will be stored at which memory locations by the above directives (show the correct number of digits! 8 hex digits = 32 bits = 1 word). You won't lose credit for incorrect answers here before you check the assembler --- you just need to take a stab at it.

	Address	Hex Value
0x25		
0x3		
16		
25		
3		
0x44		
0x33		
0x22		
44		
33		
22		

Now assemble the above code, and look at memory. Let's figure out what you are looking at. The + values across the top are in hex (even though the 0x is missing --- they left it off to fit more on the screen).

- **Question 9: Does each box shown in the data segment window represent a byte or a word? Please explain.**

Note: in MIPS, each byte of memory does have its own address. It's just that MARS does not show an address for every byte, to fit more on the display.

Now that you understand what you are looking at, fill this out correctly.

- **Question 10:**

	Address	**Hex Value**
0x25		
0x3		
16		
25		
3		
0x44		
0x33		
0x22		
44		
33		
22		