CSCI 424
Computer Architecture
Fall 2016

Homework #2

Distributed: September 08, 2016
Due: September 20, 2016 via Blackboard by 11:00pm
Points: 100

Instructions:

a. No collaboration is allowed.
b. Upload the PDF as <CS424_HW2_lastname_firstname>.pdf (e.g. Student Last Name = Bob, Student First Name = Alice, then submit CS424_HW2_Bob_Alice.pdf).
c. Attached code files should follow similar naming conventions.

Part #1: Getting started with MARS (10 pts)

- The goal is to learn to use the MARS simulator, and then answer some simple questions.

- Download the MARS program from:
  http://courses.missouristate.edu/kenvollmar/mars/
  This is version 4.5. It's a Java program, so it should run anywhere. The tutorial at:
  http://courses.missouristate.edu/KenVollmar/MARS/tutorial.htm
  (The second item: MARS tutorial -- 20 pg. handout in three phases) is pretty good, with lots of screenshots. Try that first, before starting MARS. Then start MARS, and go through the tutorial again.

- To start MARS, just double-click on its icon.
  Let's start with an absolute minimal program - it starts and ends, and does nothing else. Comments begin with # and go to the end of the line. Assembler directives begin with a dot, and are usually tab-indented.

- Start MARS, click on the New File button (farthest left, or menu File/New), then type the program into the editor window. The default file name is mips1.asm. You will probably need to resize the MARS window and its panels, to increase the size of the Edit panel. Be sure that the Registers panel shows all three columns.

- We will start by looking at the MIPS code in **HW2_Version1.txt** file and running it on MARS. Don't take a shortcut by using copy-and-paste. You will miss the information that MARS displays while you type.

- Save the file as version1.mips (fourth button, or File/Save as ...). The usual filename would be like version1.asm.

- Assemble the program. You should see:

  ```
  Assemble: assembling ../version1.mips
  Assemble: operation completed successfully.
  ```

  in the Mars Messages panel at the bottom. Of course, your directory name will be different.

- Run the program. You should see:

  ```
  Go: running version1.mips
  Go: execution completed successfully.
  ```

  in the Mars Messages panel, and

  ```
  -- program is finished running --
  ```

  in the Run I/O panel.

- The program is displayed in the Text Segment panel, and the global variables are displayed in the Data Segment, except that there aren't any.

- You can set breakpoints in the program by checking the Bkpt box at the left of each instruction. More of that later.

- Note that after the addi instruction is executed, $v0 will be loaded by 10.

- The syscall instruction (system call) initiates an operating system action, depending on the value in register $v0 when it is executed. Syscall 10 is like the exit() function in C or C++.

- Note that the Program Counter register started as 0x00400000 and ended as 0x00400008, which is one instruction past the last one in the program.

- We will now start by looking at the MIPS code in **HW2_Version2.txt**, the ubiquitous Hello World program. We also added some function definitions, which make it easier to deal with system calls. Note the appearance of the second .text directive, so the new code isn't treated as data.

- We will now start by looking at the MIPS code in **HW2_PH_COD.txt**. Now we need to verify that you actually can use MARS successfully.
  1. When you run the program, what is printed?
  2. What is the value in register $t7 (in decimal) when the program ends?
  3. How many times the line 16 of program "addu $t0, $t6, 2" is executed?
  4. Set a breakpoint for the instruction at line 12 of the assembler source code. Run the program again; it should stop at the breakpoint. Now execute that one instruction. Which registers have changed as a result of executing that one instruction? You might need to continue past the breakpoint several times to see what's going on.

- Put the answers to these four questions in a PDF file. You don't need to attach any code files for this part.

Part #2 (40 pts)

- Here is the starter program, **Homework2.asm**. You should copy-and-paste it into the MARS editor, and then save the file as your own, for editing. ALIGN the code to be successfully assembled. Spend some time trying to see what it does (it's not intended to be difficult).

- The parts that need to be changed in main are between the comment lines **Your part starts here** and **Your part ends here**. You should also add more lines to the register assignment table, to describe what you did. Additional comments will probably be helpful.

- Here are the Mars Messages and Run I/O panels, after assembling and running the starter program:

```
Assemble: assembling /.../Homework2.asm
Assemble: operation completed successfully.

Go: running Homework2.asm
Go: execution completed successfully.

CSCI 424 Homework 2
```

```
Student’s First Name
Student’s Last Name
0 2
1 2
2 2
3 2
4 2
5 2
6 2
7 2
8 2
9 2
10 2
11 2
12 2
13 2
14 2
15 2
16 2
All done!

-- program is finished running --
```

- Your output should be the same, except for the names, and the value of n that is printed (it's a constant 2 in the starter version).

- OK, here's what you need to do for the assignment. The variable n is the total number of 1s in the binary representation of variable i assuming 32-bit integers. Here are some examples:

i (decimal)	i (binary)	n (decimal)
0	00000000000000000000000000000000	0
1	00000000000000000000000000000001	1
2	00000000000000000000000000000010	1
3	00000000000000000000000000000011	2
4	00000000000000000000000000000100	1
5	00000000000000000000000000000101	2
6	00000000000000000000000000000110	2

It's going to be a lot easier if you sketch the solution in C, and then rewrite it into

MIPS assembler, inserting the C version as a comment.

- Submit your program file (.asm or .mips) via Blackboard. Please specify your name in the title of your submission (e.g. CS424_HW2_P2_Bob_Alice.asm).

Part #3 (50 pts)

- Using MARS, write a program that determines if an ASCII string entered into the console is a palindrome. A palindrome is a string that is the same forwards and backwards. For example, the word *radar* is a palindrome, but the word *Radar* is not. The string of ASCII characters is input into a single line on the console followed by a new line (carriage return) with code like:

```
inpt: addi $v0, $zero, 8 #sys call code for read_str
syscall #read it
```

- The read_string input command (system call code 8) reads up to n-1 ASCII characters into a buffer and terminates the string with a null byte. The string is left in a buffer in memory whose address is contained in $a0 and its length is recorded in $a1. Your program should determine if the string is a palindrome and then output the following to the console:

```
The string <string> <is | is not> a palindrome
```

- Console output is accomplished by placing the ASCII characters to be output in memory at the address contained in $a0 and issuing a print_string output command (system call code 4) as follows:

```
oupt: addi $v0, $zero, 4 #sys call code for print_str
syscall #print it
```

- You may assume the input string will be no more than ten characters. The only illegal string character is a carriage return.

- Submit your program file (.asm or .mips) via Blackboard. Please specify your name in the title of your submission (e.g. CS424_HW2_P3_Bob_Alice.asm).

Note:

In addition to MIPS instructions covered in the class, it is a good idea to lookup other MIPS instructions given in the textbook (Appendix A10). For example, la (load address), li (load immediate), branch instructions (e.g. bgt, bne, beq).