

Homework #2 – Assembly Programming

Due Monday, February 13 at 5:00pm

You must do all work individually, and you must submit your assignment electronically via Sakai.

All submitted code will be tested for suspicious similarities to other code, and the test will uncover cheating, even if it is “hidden” (by reordering code, by renaming variables, etc.).

Directions:

- For short-answer questions, submit your answers in PDF format as a file called <NetID>-hw2.pdf. **Word documents will not be accepted.**
- For programming questions, submit your source file using the filename specified in the question.
- **You must submit your work electronically via assignments section of Sakai.** Submission contents 4 files: <NetID>-hw2.pdf, fibtimes2.s, recurse.s, HoopStats.s.

MIPS Instruction Set

- 1) [5 points] What MIPS instruction is this? 0000 0011 0011 0010 1000 0000 0010 0010
- 2) [5] What is the binary representation of this instruction? sw \$r7, 12(\$r4)

Compiling C Code to MIPS Assembly

3) [10 points total] In Homework 1 Q3 (“Compiling and Testing C Code”), you observed how changing the level of compiler optimization altered execution time. In this question, we will examine how that works.

The computer used in Homework 1 was a Duke Linux system, which is a PC based on the Intel x86 64-bit architecture, so the compiler generated instructions in that ISA for that CPU. In this question, we want to examine the resulting assembly language code, but we aren’t learning Intel x86 assembly language, so we’ll need a compiler that produces MIPS code instead. [Such a tool has been developed for you here.](#) This web-based tool acts a front end to a g++ compiler which has been set to produce MIPS code. Further, it’s been set up to show us the assembly language code (.s file) rather than build an executable binary.

A small piece of the program from Homework 1, **prog_part.c**, has been included in the starter kit and linked on Sakai->Resource->Homework Resources. Compile this code to assembly language with

optimization disabled (-O0) and set to maximum (-O3). Locate the `get_random` function (labeled “_Z10get_randomv” or similar) in the two versions of the code to answer the following questions:

- (a) [1] How many total instructions (not dot-prefixed directives like `.frame`) are in each of the two versions?
- (b) [1] How many memory accesses (loads and stores) were in each of the two versions?
- (c) [1] Which version of the code uses more registers?
- (d) [7] Based on the above, what are some general strategies you suspect the optimizer takes to improve performance?

Assembly Language Programming in MIPS

Directions and Rules

For the MIPS programming questions, use the QtSpim simulator that you used in Recitation #4. Please note that the TAs will be grading your assembly programs using SPIM. If you use some other MIPS simulator (e.g., MARS), there is NO guarantee that what works on that simulator will also work on SPIM. We will grade strictly based on what works when your program runs on SPIM.

Note: Do not try to use the web-based MIPS C compiler to “automate” the programming questions below. In addition to being academically dishonest, it also won’t work very well for the technical reasons listed on the web-based tool itself.

Testing Your Code

As in Homework #1, we provide a self-test tool to help you validate your code. The tool is provided in Sakai->Resource->Homework 2-> `homework2-kit.tgz`. You can drag your `.s` file into the tool directory and run “`./hw2test.py`” as in HW1, which will give you some information on which test cases you pass/fail. Please run the self-tester on the Duke Linux machines (login.oit.duke.edu) to avoid unexpected errors.

Before testing on the Duke Linux machines, you can also manually provide input and visually check the output of your program in the QtSpim console against the expected output for a given test. Test cases and expected value are provided in `/tests` directory. Note that some test cases from HW1 no longer apply, and have been eliminated.

Each of your programs should prompt for input and display output via the **QtSpim Console window**.

If you want to execute a manual test, type in the **Input** listed in the tables associated with each program when prompted. After you have run your program, your program's output should match the **expected output** from the file indicated. For problem (c), the input comes from the file listed in the **Input file** column. Each line in the file should be typed in individually per prompt as instructed in problem (c).

Alternatively, you can simply upload your (`.s`) files into the self-test tool directory and test there.

Similarity to Programming in Homework #1

These programming questions are almost the same as those from Homework 1 with the following key differences:

- In HW1, input came from command line arguments. In this assignment, input is typed into the console.
- Because the program is now prompting for input interactively, your program will output prompts before reading values. We intend to use an automated tool to assist with grading, so **please end all your user prompts in a colon.** (e.g. "Please enter an integer:").

The Programming Tasks

4) [20] Write a MIPS program called fibtimes2.s that prints out the first N Fibonacci numbers multiplied by 2 (as in Homework #1, when you coded this in C).

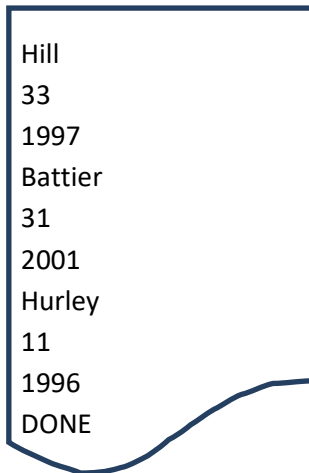
You will upload fibtimes2.s into Sakai (via Assignments).

5) [40] Write a MIPS program called recurse.s that computes $f(N)$, where N is an integer greater than zero that is input to the program. $f(N) = 3*(N-1) + [2*f(N-1)] - 1$. The base case is $f(0)=5$. Your code must be recursive, and it must follow proper MIPS calling conventions. **The key aspect of this program is to teach you how to obey calling conventions → code that is not recursive or that does not follow MIPS calling conventions will be severely penalized!**

You will upload recurse.s into Sakai (via Assignments).

6) [50] Write a MIPS program called HoopsStats.s that is similar to the C program you wrote in Homework #1. However, instead of reading in a file, your assembly program will read in lines of input as strings. That is, each line will be read in as its own input (using spim's syscall support for reading in inputs of different formats). The file is a series of player stats, where each player entry is 3 input lines long. The first line is the player's last name, the second line is his jersey number (an int), and the third line is his year of graduation (an int). The last line you read is the string "DONE".

For example:



```
Hill
33
1997
Battier
31
2001
Hurley
11
1996
DONE
```

IMPORTANT: There is no constraint on the number of players, so you may not just allocate space for, say, 10 basketball player records; you must accommodate an arbitrary number of players. You must allocate space on the heap for this data. **Code that does not accommodate an arbitrary number of players will be penalized!** Furthermore, you may not read all of the lines to first find out how many players there are and *then* do a single dynamic allocation of heap space; instead, you must dynamically allocate memory on-the-fly as you read the lines. To perform dynamic allocation in MIPS assembly, I recommend looking at: http://chortle.ccsu.edu/assemblytutorial/Chapter-33/ass33_4.html

Your program should output a number of lines equal to the number of players, and each line is the player's name and his jersey number. The lines should be sorted in ascending order of graduation year (with ties broken alphabetically by last name). For example, the output for the sample statsfile above should be:

```
Hurley 11
Hill 33
Battier 31
```

You will upload HoopStats.s into Sakai (via Assignments).