

Assignment 1

Due Date: 11:59 pm, Wednesday May 28th, 2008.

Part I: SPIM Programming

Purpose: To reacquaint you with the MIPS-32 ISA and the control flow behavior captured in the ISA. This will guide your subsequent understanding of the datapath implementations and the problems/issues that must consequently be solved in high performance datapath implementations

Goal: For this assignment you will write a SPIM program to sort an array of integers. Assume that the array is stored in the data segment starting at some *known* label. Further assume that you will read the number of elements in the array from the console. Use any sorting algorithm you wish. The quality of the sorting algorithm is irrelevant. Finally, the sorting algorithm should be called from the main routine as a procedure labeled **sort**. The sorted array should be available in memory starting at any legal location. Make sure this location is labeled in an obvious way.

Warm-Up:

1. Download the example SPIM program `io.example.s` from the SPIM resources page on the class website. Execute this program and understand what it does.
2. Modify this program to read two integers from the console.
3. Further modify this program to print the larger integer to the screen.
4. Download other sample SPIM programs from the ECE 3055 class resources page and execute them. Try a few simple modifications to these programs before you starting writing your own, especially the ones dealing with I/O.
5. The TA office hours will be held in the lab – KACB 1440. This week the TA will be giving demos of the SPIM simulator during office hours for groups of students. Attendance is highly recommended. Appointments are not necessary. Specifically you should get comfortable with the following steps.
 - Learn to single the step program execution and to set breakpoints. This is invaluable for debugging.
 - Learn to read the text and data segments
 - Make sure SPIM is configured correctly
 - When you execute the program make sure the starting address in the **run** dialog box is `0x00400000`.
 - In the **Simulator** → **Settings** dialog box, make sure that delayed branches and delayed loads are disabled.

Main Assignment:

- The following steps are recommended to complete the assignment.
 - Design/pick a sorting algorithm (use the one you developed in CS 1371)

- Perform memory allocation for the data, i.e., data directives, and specify the data segment.
- Perform register allocation
- Refine the algorithm description in terms of registers and memory addresses
- Implement one block of code and test, for example the instructions that compare to numbers.
- Integrate smaller tested blocks of code for example the core loop to sort the numbers.
- Move the core loop to a procedure and test.
- Implement the register saving convention as necessary to complete the assignment.
- Make sure you turn in the components that work and identify the components that work for the TA.

Grading Guidelines

For your information here are the grading guidelines for the SPIM component of the assignment.

- Program compiles without errors (and appears on the surface to be correct) : 35pts
- Program executes correctly: (additional) 40 points
- Documentation and description of the program: 25 points
 - Have a commented a program header with your name, class, and assignment.
 - The program should well commented so much so the TA can determine what you are doing.

Submission Guidelines:

All submissions should be electronic. You can scan the worked problems below and provide PDF. Submissions must time stamped by midnight on the due date. Email to the TA at jimmy.simmons@gatech.edu.

Late assignment policy is as follows: 25%/day. Note that if you do not perform the assignments it is unlikely you will pass the class.

Part II: Problem Sets

1. Assume the data segment has the following directives.

```
newline:      .ascii "\n"
space:       .ascii " "
```

Given these directives, what does the following code segment do? You may need to make some assumptions but not many. Do not give the obvious mechanical description.

Describe the functionality.

```

        move $t1, $0
L1:     move $t2, $0
L2:     lw $a0, 0($t4)
        li $v0, 1
        syscall

        la $a0, space
        li $v0, 4
        syscall

        addi $t4, $t4, 4
        addi $t2, $t2, 1

        slt $t5, $t2, $t6
        bne $t5, $0, L2:

        la $a0, newline
        li $v0, 4
        syscall

        addi $t1, $t1, 1

        slt $t5, $t1, $t6
        bne $t5, $0, L1

```

2. Consider the scientific domain and software for manipulation large matrices. i.e., array operations. Propose the addition of two instructions to the MIPS ISA that you feel will be useful and provide a brief justification for your answer. Are there any holes (unused opcodes) in the MIPS ISA that you can use? If so what are they?