EEC 170

Lab 2 – Intro to RISC-V and Venus Simulator

**Purpose**

The remaining labs in this course will focus on assembly-level programming. This type of programming is common in bare metal embedded systems where programs are run without an operating system. Speed-critical applications may also require developers to program small portions of their code to be optimized for their target architecture.

**Tools**

[Venus RISC-V simulator](https://venus.cs61c.org/)

**Resources**

*Computer Organization and Design* chapter 2

Venus RISC-V simulator Wiki (access from the Venus simulator’s “Venus” tab)

**Deliverables**

One lab report document containing properly formatted RISC-V assembly code, concise descriptions of all requested results and any requested screen captures. To avoid loss of progress, write all assembly code in a local text editor and copy into the Venus simulator.

**Procedure**

**Part I: Basic Arithmetic and Logic Operations (5 pts)**



Code Block 1: Part 1 C-Code

1. Convert the above code into RISC-V assembly.



1. Simulate your code.
   1. Navigate to the Venus webpage linked above.
   2. Go to the Editor section and paste your assembly code.
   3. Navigate to the Simulator section, click “Assemble & Simulate from Editor”.
   4. Clicking an instruction will highlight it red. This inserts a breakpoint and tells the simulator to pause simulation. Insert a breakpoint at the last instruction.
   5. Either run or step through program depending on debugging needs.
2. Capture the register file state after your code finishes executing.
   1. Turn off the “Dark Mode” setting by navigating to the “Venus” tab.
   2. Navigate back to the Venus Simulator and capture the register outputs in three screen captures (x0-x11, x12-x22, x23-x31).
   3. Place the screen captures in your report. You may copy the object shown below into your report, double-click the object and replace each image with your register screen capture.
3. Format and place your assembly code in your report document.
4. In your report, briefly describe how the register contents of step 4 are consistent with the assembly code written in step 1.

**Part II: Loading, storing, and initializing data (5 pts)**

1. The “.data” keyword of a RISC-V assembly code can be used to place data directly into the data segment of memory. The following example shows this method of initializing data into memory:



Code Block 2: Populating the Data Segment

* 1. Paste this code into the Venus RISC-V editor. Do not add in any code yet but navigate to the simulator and select “Assemble & Simulate…”.
  2. Navigate to “Memory” and use the “Jump to” drop-down menu to locate the data populated in the code. Screen capture the memory region with the data and explain why the data is located where you found it.

1. Data saved in the data segment may be referred to symbolically using a symbol (i.e. data\_example). Write an assembly program that performs the following operations.
   1. Load each word of “data\_example” into arbitrary registers.
   2. Sum all the elements of “data\_example”, and place in a new register.
   3. Store the result in the address above the location of the last element of “data\_example.”



1. Paste the assembly code into the Venus Editor.
2. Assemble, simulate, and run the code. Be sure to place a breakpoint at the last instruction.
3. Screen capture the memory region where the result of part 2c was stored as well as the complete register state as shown in Part I 4c. Be sure to briefly describe your results.
4. Place your assembly code into your report document.

**Part III: Decision-making instructions (10 pts)**

1. Convert the following C-code into RISC-V assembly. Hint: you may populate your array in memory using a data segment as described above:



Code Block 3: C-Code for Decision Making



1. Simulate your code, screen capture the ending register status, and briefly describe your results. Place the assembly code into your report.

