CS204 - Computer Architecture
Practise Lab - 1.2
Getting Started with RISC - V using Venus

Date: 20th Jan 2025. Not Graded!

Deadline: Complete all the tasks at the earliest, by 24th Jan 2025.

General instructions:

1. Read through all the tasks thoroughly first.
2. Work on the tasks in the order specified. They are designed to help you get started step-by-step.
3. Try to finish all the tasks by 24th Jan 2025 11.59AM. During the **tutorial hour on 24th Jan 2025**, we will discuss any specific queries you have but you should have tried each of the tasks at least once before that.

==========

For the first few labs we will use a simple web based RISC-V simulator, namely, *Venus*. Link - `https://venus.cs61c.org/`. Thanks to Mr. Keyhan Vakil (UC Berkeley) for creating such an elegant and efficient online educational instruction simulator.

Venus executes in your browser and there is no need to install any tools on your system. You can also save that web page to have a local copy of the simulator to work offline. *Venus* supports RV32IM - basic integer and multiplication functions.

In this lab, you'll get acquainted to Venus and explore few basic machine instructions in the RISC-V instruction set.

Task 1: Start by pasting the following assembly program (you can also save it for reuse) into *Venus*, in the Editor pane:

```
# Sample code 1
# A minimal RISC-V assembler example
# Note:  '#' is used to comment any line
addi x11, x0, 2
addi x12, x0, 3
add x13, x11, x12
```

Task 2: Change to the 'Simulator' tab and step through the code with the `Step` button. Observe how the registers $x1$, $x2$, and $x3$ change after each instruction. *Adding immediate values to register $x0$, which is always 0, is one way to load constants (immediate values) into registers.*

In the 'Simulator' tab, you can also observe the Program Counter (PC) and the Machine Code of each of the instructions. *Venus* is acting as a RISC-V assembler here. You can also observe the default values of Registers (`x0-x31`) on the right side pane.

Loading immediate values is so basic to get a program started that RISC-V defines a pseudo instruction, `li`, as a shortcut. "What is a pseudo instruction?" will be discussed in class

in-detail, for now, use them directly. Enter the following code into the Editor and switch to the Simulator pane.

```
# Sample code 2
# Use of pseudo instructions to load immediate values
li x11, 2 # This instruction is same as addi x1, x0, 2
li x12, 3
add x13, x11, x12
```

You will notice that your code is listed under 'Original Code', but the final RISC-V instructions are listed under 'Basic Code'. You can also notice that those instructions (in 'Basic Code') are the very same as the ones you have entered in Sample code 1.

Task 3: Extend your program with a handful of more instructions to explore the functions of the *Venus* simulator. You can clear the registers and the program counter (PC) by pressing `Reset`. Step through your program with `Step` or run you program to completion with `Run`.

Another important concept is a `Breakpoint`. You can set a `Breakpoint` by clicking on the instruction in the 'Simulator' tab. A `Breakpoint` is marked by coloring it red. Another click on that instruction will clear the `Breakpoint`. With a `Breakpoint` set, you can `Run` the program until it reaches the `Breakpoint`. There you can explore/cross-check/validate values in the registers of your interest. Its a very handy tool for debugging your program.

Task 4: Computing with ALU Instructions
Only a handful of instructions in the arithmetic logic unit (ALU) are used to compute. Operations for ALU instructions are provided in registers (source operands) and the result is put into a register (destination operand) as well. Locate all integer ALU instructions of RISC-V Reference Data sheet and use them in the simulator.

Task 5: Use the shift and bit-wise operations to complete the below task.
Write RISC-V code that extracts bits 16 down to 11 from register x5 and uses the value of this field to replace bits 31 down to 26 in register x6 without changing the other bits of registers x5 or x6. (Be sure to test your code using x5 = 0 and x6 = 0xffffffff. Doing so may reveal a common oversight.)

Task 6: Write RISC-V code to perform bit-wise 'NOT' operation on a value.

Task 7: Check pseudo instructions *neg* and *not*. Is the result of both the instructions same? Try them on a register (say, x5) containing the value ABH.

Task 8: Observing the RISC-V 'fields' in the machine code
Over the last few classes we looked various fields of the R-format, I-format and S-format instructions. Looking at the RISC-V Reference Data sheet you'll realize that many of the instructions are encoding in one of these three formats. Let's study them closely.
Include instructions from each of these formats in Editor pane and note the 'Machine Code'

in Simulator pane. Pick one assembly instruction at a time, look at its machine instruction and divide the machine instruction into its various fields.
(a) Notice the fields for operands.
(b) Look at the opcode, funct3 and funct6/7 fields and compare them with the RISC-V Reference Data sheet (Page 2).
(c) If you are using an immediate value, observe how it is interpreted in arithematic and load instructions. How will you know this?
(d) For S-format instructions, observe how the immediate value is represented in two fields.
(e) For shift instructions, try giving the value of the shift to be more than 6-bits and observe the result.

Task 9: Once you are finished with *all* the above tasks, upload a single text file (name it "YourName_YourRollNo_learnings_lab2.txt") documenting your learnings and interesting aspects that you have noticed today. Write point-wise statements and not paragraphs. Upload the file by **Monday, 27th Jan 2025, 11.55AM**. We will go through your responses and tailor the next lab to address any common difficulties you have faced.