ECE 1110
Introduction to Computer Organization & Architecture

Due date: October 4rd, 2021

We have seen how microprogramming can be used to implement a RISC-V processor. In this assignment, you will add microcode to an existing microcode implementation of RISC-V. This assignment requires thought and preparation. Do *not* leave this until the last moment!

In order to complete this assignment you will need tools. Here's what you'll need:

1. A functioning C++ compiler
2. A copy of a tool called `Verilator`. This compiles a language called `SystemVerilog` into C++. This C++ code is then compiled and executed to simulate the machine.
3. A copy of the latest and greatest Python, `python3.10`. You'll need this for the microassembler.

Here's how it all works: The machine is described in a Hardware Description Language (HDL) called `SystemVerilog`. `SystemVerilog` is the latest iteration of `Verilog`, an HDL like `VHDL`. A zoom video will describe how the machine functions. The machine uses an assembled version of the microcode. A microcode assembler, written in Python3.10, can output either a mapping of labels to microcode locations (for use in dispatch) or a hexadecimal output of the microcode.

Here's the full assignment: The C idiom for copying a string is:

```
while (*dp++ = *sp++);
```

1. Write a RISC-V assembly language subroutine that copies a string. Assume that the strings are statically allocated with the source address in register `a0` and the destination in register `a1`. Hint: Code this up using `rars` first to debug your algorithm. But you can't just use this program in the raw machine because of address ranges. So you'll have to re-code it. Run this code under Verilator by putting the hex binary in program.hex — count the number of microcode cycles.

2. We are going to add a new type of instruction: "A-type", where the register hold *addresses*. So, then `STRCPY rd, rs1` copies the string starting at the memory location held in `rs1` to the memory location held in `rd`. Change the microcode to add a string copy instruction. Here's what you'll need to do:

   (a) Add microcode at the label `EXTRA` (at the end of the `microcode.uas` file) to implement the string copy instruction. Read the microcode for the LB and SB instructions carefully. Make sure you understand what *every* field is doing. Assemble the microcode with `uas.py` and save in `ucode.hex`.

   (b) I have provided a binary to test your new microcode. You can find it in `strcpy-test.hex`. It copies a string starting at 0x30 to 0x40.

   (c) Run the simulation. Verify that it works.

   (d) Count the number of microcycles and compare with the assembly language version in part (a).

   (e) C style strings are terminated with a zero ('\0') but that is not the only way to implement strings. They can also be implemented by storing the length in the first byte and not terminated with a zero. From your standpoint as a microcode writer, what is the difference between the two?