Northwestern University
CompEng 361 - Fall 2023
Lab 2 - Single Cycle CPU

In this project, you will work in groups of two to design a single cycle RISC-V CPU which implements the majority of the RV32I Base Instruction Set. You will eventually use this in the final lab to implement a pipelined processor. Follow the RISC-V documentation links on Canvas to learn the instruction encodings and functionality. The specific instructions that you must support are:

```
lui, auipc
jal, jalr, beq, bne, blt, bge, bltu, bgeu
lb, lh, lw, lbu, lhu, sb, sh, sw
addi, slti, xori, ori, andi, slli, srli, srai,
add, sub, sll, slt, sltu, xor, srl, sra, or, and
```

The single cycle processor will be implemented in Verilog (your choice of behavioral or structural) and must have the following interface and port list:

```
module SingleCycleCPU(halt, clk, rst);
      output halt;
      input clk, rst;
```

The halt line should be asserted if and only if the cpu encounters an illegal/unsupported instruction or there is a memory alignment error (e.g. effective address for a lh is not an address which is a multiple of two, attempt to fetch from an address which is not a multiple of four). At that point, your cpu should not execute any more instructions or update any more system state. The testbench that we have supplied (more on this later) will at that point exit the simulation and dump system state.

Your single cycle CPU design should instantiate three library modules. We provide the implementation for these modules. You should NOT modify them. They are:

```
module InstMem(Addr, Size, DataOut, CLK);
module DataMem(Addr, Size, DataIn, DataOut, WEN, CLK);
module RegFile(AddrA, DataOutA, AddrB, DataOutB,
               AddrW, DataInW, WenW, CLK);
```

They implement idealized versions of instruction/data memory and a register file. Reads are combinational and writes are synchronous.

We supply a testbench that you must use. It will read instructions/data from the file “mem_in.hex” at startup and write contents of the data memory to “mem_out.hex”. The register file will correspondingly read from “regs_in.hex” and write to “regs_out.hex”. Do not modify the input/output behavior of the test bench. You can assemble your own test programs and copy them into the files previously discussed. You can look at the memory and register output to verify that your design works correctly. We encourage use of waveform viewers as you debug your design.

We have provided some template code to get you started. Here are a few important notes:

- Please do NOT change the interface to the module. It must not deviate from what is posted above.
- Your solution should be able to compile and run correctly with unmodified testbench and library files.
- Your solution MUST be entirely in Verilog (no Chisel or System Verilog)
- Your solution should be self-contained in a single Verilog source file without use of any external source files beyond the ones supplied.
- You should feel free to use either structural or behavioral code to implement your design, but we strongly recommend the former.

You must devise your own testing programs. Make sure the module is thoroughly tested. Try to think of corner cases and make sure they are appropriately handled.

You should turn in a single Verilog file with the following format:

```
<group-name>_lab2.v
```

Do not include testbenches, library files, test programs, or other supporting files.

Note that this is a group assignment. Turn in one submission per group.