COSE222: Computer Architecture
Design Lab #4

Due: Nov 25, 2024 (Monday) 11:59pm on Blackboard

Total score: 30

In the previous design lab, you were requested to design the datapath elements: instruction memory, register file, ALU, and data memory. In this lab you are requested to complete the single-cycle processor design using the designed datapath elements. The single-cycle processor in this lab will support just the following instructions: `lw, sw, add,` *`addi,`* `sub,` *`xor, xori,`* `or,` *`ori,`* `and,` *`andi,`* `beq,` *`bne, blt, bge`*. Note that our processor will support 3 more branch instructions (`bne, blt, bge`), xor instruction (`xor`), and I-type arithmetic/logical instructions (`addi, xori, ori, andi`) compared to the processor in the textbook.

1. Single-cycle processor design

As you already designed the major datapath elements in the design lab #2, you can just instantiate your design blocks in the top design of the single-cycle processor. We also provide the complete designs for the instruction memory, register file, ALU, and data memory. If you failed to complete the design lab #2, you can use the provided design files. (Please check if the modules you designed for lab #2 are correct.) The required interfaces are also provided in the incomplete top design of the single-cycle processor. In the provided top design of the single-cycle processor, you are to include the following items.

(1) Simple datapath elements such as MUXes, a shifter, and an immediate generator

(2) The main control unit as described in the textbook plus more branch instructions and I-type arithmetic/logical instructions

(3) The ALU control unit as described in the textbook plus supports for more instructions

(4) Connecting datapath elements and control signals

The instruction memory already includes the following RISC-V assembly code.

Assembly code	Binary data in imem
`start:  lw x4, 0(x0)`	`00000000000000000010001000000011`
`        lw x5, 8(x0)`	`00000000100000000010001010000011`
`        lw x6, 16(x0)`	`00000001000000000010001100000011`
`T1:     add x7, x5, x4`	`00000000010000101000001110110011`
`        sub x8, x5, x4`	`01000000010000101000010000110011`
`        and x9, x5, x4`	`00000000010000101111010010110011`
`        or x10, x5, x4`	`00000000010000101110010100110011`
`        xor x11, x5, x4`	`00000000010000101100010110110011`
`        addi x27, x4, 1`	`00000000000100100000110110010011`
`        addi x28, x4, -7`	`11111111100100100000111000010011`
`        andi x29, x4, 15`	`00000000111100100111111010010011`
`        ori x30, x4, 16`	`00000001000000100110111100010011`
`        xori x31, x4, 15`	`00000000111100100100111110010011`
`        beq x7, x8, T1`	`11111100100000111000110011100011`
`        sw x7, 24(x0)`	`00000000011100000010110000100011`
`        lw x11, 24(x0)`	`00000001100000000010010110000011`

`bne x7, x11, T1`	`11111100101100111001011011100011`
`T2:     lw x4, 32(x0)`	`00000010000000000010001000000011`
`lw x5, 40(x0)`	`00000010100000000010001010000011`
`bge x4, x5, T2`	`11111110010100100101110011100011`
`add x4, x4, x5`	`00000000010100100000001000110011`
`blt x5, x4, T1`	`11111010010000101100110011100011`
`sub x4, x4, x5`	`01000000010100100000001000110011`

You can find the more detailed instructions in the incomplete top design file (single_cycle_cpu.sv). Please complete the single-cycle processor design. Verify your design using Verilator.

2. What to do

(a) Complete the top design of the single-cycle processor (single_cycle_cpu.sv).

(b) You should use "tb_single_cycle_cpu.cpp" as the testbench of the single-cycle processor. See the below commands for compiling and verifying the single-cycle processor using Verilator. If you execute the generated binary, you will find the generated "report.txt" file. You can see the generated waveforms by reading "wave.vcd" using gtkwave.

```
$ verilator -Wall --trace --cc single_cycle_cpu.sv imem.sv regfile.sv
alu.sv dmem.sv --exe tb_single_cycle_cpu.cpp
$ make -C obj_dir -f Vsingle_cycle_cpu.mk Vsingle_cycle_cpu
$ ./obj_dir/Vsingle_cycle_cpu
```

If you encounter error messages with the provided testbench file with the new version of Verilator, you can use the alternative testbench file, "tb_single_cycle_cpu_new.cpp". You can ignore warning messages created by Verilator.

(c) Compress your all the design files (imem.sv, regfile.sv, alu.sv, dmem.sv, and single_cycle_cpu.sv), the testbench (tb_single_cycle_cpu.cpp), and the report file (report.txt) in **one zip file**. You must name your zip file as "`FirstName_LastName.zip`". (e.g. `Gildong_Hong.zip` for Gildong Hong) If your submission file does not meet this rule, we will reduce 1 point from your score. ☹️