Name: _____

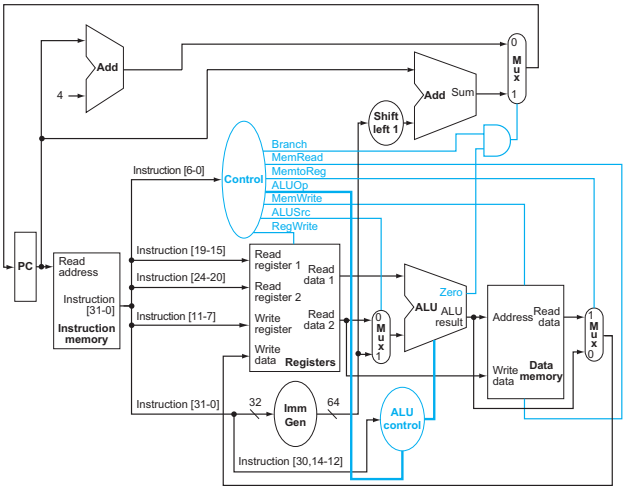
Lab for ITSC 3181, Introduction to Computer Architecture, Spring 2023

Lab #11 and #12: Logic design using Digital (https://github.com/hneemann/Digital): Design the single-cycle processor for RISC-V supporting an essential set of instructions including at least add, and, or, sub, lw, sw, and beq instructions. **Due by 04/12 and 2% of the final grade.**

From Lab 07 to Lab 12, we will build a RISC-V CPU to support an essential set of instructions including at least add, and, or, sub, lw, sw, and beq instructions. While you are encouraged to design a 32-bit CPU (each register stores a 32-bit word, and the ALU operates on 32-bit data), you are ok to design 8-bit or 16-bit CPU (each register stores a 8-/16-bit data, and the ALU operates on 8/16 bits data). Designing 8-bit/16-bit is much easier comparing with designing 32-bit, but with same design principle. But regardless of the bitwidth we use, each instruction is still encoded as a 32-bit word and the address for accessing data from memory are also 32-bit address. In this lab, we will create the processor design using the diagram from the textbook. See below. We will use the components we already designed before (Memory, Register file, register, ALU, Mux, Adder which can be created using the ALU). For other components that are needed in the diagram, please design for the lab accordingly or use the one provided by Digital. PC is just a 32-bit register.

For Control, ALU Control, Imm Gen and "Shift left 1", you can create the dummy logics that have just the exactly the same inputs, outputs, and bitwidths of the input/output of the components, and then use them in the processor design. You do not need to implement those dummy components. As the result, the CPU you created will not be functional. If want to implement the ALU control and control logics, their truth tables are already given in the textbook and lecture.

If you have not done some or any of the Lab 07 to Lab 10, you can still do this lab by creating and using dummy circuits that were supposed to be created in Lab 07 - Lab 10. For a dummy circuit, you just need to add the needed inputs and outputs, label them correctly and save them as a circuit. You can then use the dummy circuits as subcircuits in this lab.



For the instruction decoding, you will need a splitter to split a 32-bit instruction word into instruction[6-0], instruction[19-15], instruction[24-20], instruction[11-7], instruction[30], and instruction[14-12]. You will then need a splitter to combine instruction[30] and instruction[14-12] for the input to the "ALU control" logic.

1. Review lecture slides 31 – 67 of Chapter 04 (https://passlab.github.io/ITSC3181/notes/Chapter04_Processor.pdf#page=31) about the datapath and control of a single-cycle RISC-V processor design.

Tasks:

1. (5%) Create the "Control" Circuit with one input (Instruction[6-0]) and seven outputs (see the diagram) only. Make sure the bitwidth and labels of the input and output are correctly set.
2. (5%) Create the "ALU Control" Circuit with two inputs (ALUOp and Instruction[30,14-12]), and one output (ALU Control input) only. Make sure the bitwidth and labels of the inputs and output are correctly set.
3. (5%) Create the "PC" Circuit using the 32/16/8-bit register with one input (PCNext) and one output (PC). Make sure the bitwidth and labels of the input and output are correctly set. You can use the one you created before or the one provided by Digital.
4. (5%) Create the "Imm Gen" Circuit with one input (Instruction[31-0]) and one output (Imm) only. Make sure the bitwidth and labels of the input and output are correctly set. The output Imm should be the bitwidth of your choice for ALU. **NOTE**: If you are designing an 8-bit ALU, this step introduces a bug of our design since the design cause losing bits of the Imm field (12bit to 8bit). We do not have this bug for designing 32- or 16-bit ALU. You do not need to fix this bug if you design 8-bit ALU.
5. (5%) Create the "Shift left 1" Circuit with one input (Input) and one output (Output). Make sure the bitwidth and labels of the input and output are correctly set. Both input and output should be the bitwidth of your choice (32, 16 or 8). You can implement it by using two splitters.
6. (10%) Create the "Adder" Circuit using the adder you designed before or use the one provided in Digital. It should have two Inputs (A, and B) and one output (Sum) and they should be the bitwidth of your choice. The Adders are used for PC+4 and PC+offset. Since the PC is the 32-bit address and if you ALU is not 32-bit (16 or 8), you can use the least significant 16/8 bits of the PC as the input to the ALU.
7. (25%) Create the "Single-Cycle RISC-V CPU" Circuit. Then add the required the SubCircuits including PC, Instruction Memory (which is 256 x 32/16/8-bit Memory designed before), Register file, ALU, Data Memory (which is 256 x 32/16/8-bit Memory designed before), Control, two Adders, Shift left 1, Imm Gen, ALU Control, and 3 Mux (make them 32-bitwidth). Layer them out according to the diagram of the CPU design and make sure enough space are left out between components for running wires.
8. (20%) Connect wires for both datapath and control using the provided diagram
9. (10%)Add a splitter to split a 32-bit instruction word into instruction[6-0], instruction[19-15], instruction[24-20], instruction[11-7], instruction[30], and instruction[14-12], and connect those wires between the corresponding components
10. (10%) Add a splitter to combine instruction[30] and instruction[14-12] for the input to the "ALU control" logic.

Important: when designing the circuit, please make sure you follow these rules for adding, changing components, input/output and wires. These rules are applied to all the design lab tasks of the course.

1) **Each input and output of a design MUST be properly and meaningfully labeled.**
2) **Each component, input and output should be correctly configured in terms of its bitwidth and signal control width.**
3) **Keep wires and components organized and layed out according to the circuit schematics rules, a) Inputs on the left (or top), b) Outputs on right (or bottom), c) gates flow from left to right, d) Straight (not angled) wires are best, e) Wires always connect at a T junction (only 90-degree turn or connection), f) A dot where wires cross indicates a connection between the wires, and g) Wires crossing *without* a dot make no connection. While you may not need to follow all those rules for creating correct and small circuit, it**

is very important when for creating complex circuit. So, make sure you properly organize the components and wires, make them structured and look good. That will help reduce errors.

4) **For drawing straight wires using mouse, make one turn per each draw. You should not use one draw to make a connection that needs to have two or more 90-degree turns, which would create angled wires. For those wires, you have to make a wire that has a 90-degree turn, and then connect it with another wire that also make 90 turn, and so on. Try to minimize the turns as much as possible. If two wires have to be crossed, but should not be connected, make sure no dot is marked on where the wire is crossed.**

Submission: take screen shoot of your design of each task and save it to a single file of PDF format. Then submit the file on Canvas.

Your grade will be based on both the correctness of your design and the organization of the design, 60% and 40% respectively.

Task	1 (5%)	2 (5%)	3 (5%)	4 (5%)	5 (5%)	6 (10%)	7 (25%)	8 (20%)	9 (10%)	10 (10%)	Total
Correctness (components, input/output, bitwidth, connection, width of control, etc), 60%											
Organization (I/O labeled and positioned correctly, all straight wires and T junction, dot used correctly, readability etc), 40%											
Total											