Homework 7

1. **Structural Hazards:**
   - **a.** Considering your current 5-stage pipeline, describe one scenario that could cause a structural hazard in your processor.
   - b. Change your Register file or Data/Instruction memory design in order to avoid the hazard that you described in 1.a. If you've already designed your momories in that way it would not cause any structural hazard, you may skip to step 1.d.
   - c. Show the added and edited parts in your corresponding Verilog code.
   - d. Perform behavioral simulation for processor in order to show that your design works as expected by applying a small sequence of instructions that would only cause this structural hazard.
   - e. Explain your results by showing your waveform outputs and verify that structural hazard is handled.
2. **Data Hazards - Read-After-Write (RAW) Hazards:**
   - **a.** Consider all scenarios that cause RAW hazards (including memory-to-memory copies). Choose one of the hardware solutions shown in lecture slides "ITU_VLSI_II_Computer_Organization_Pipeline_2.pdf" to integrate within the appropriate places in your pipeline.
   - b. Apply stalling and/or forwarding circuitry in your design, depending on your selection.
   - c. Show the added/edited parts in your corresponding Verilog code.
   - d. Perform behavioral simulation for your processor in order to show that your design works as expected by applying a small sequence of instructions. Individually simulate each case and show that the hazards are properly handled. Clearly show and explain your results on waveform outputs.
   - e. Explain your results by showing your waveform outputs and verify that hazard is handled.
3. **Data Hazards - Load-Use Hazards:**
   - a. Consider the case where load-use hazard will occur. Apply the hardware solution shown in lecture slides "ITU_VLSI_II_Computer_Organization_Pipeline_2.pdf" within the appropriate places in your pipeline.
   - b. Add extra stalling and forwarding circuitry to appropriate places, if needed.
   - c. Show the added/edited parts in your corresponding Verilog code.
   - d. Perform behavioral simulation for your processor in order to show that your design works as expected by applying a small sequence of instructions. Individually simulate each case and show that the hazards are properly handled. Clearly show and explain your results on waveform outputs.
4. **Running an Example Program:**

An example bubble sort algorithm is provided in "https://github.com/yavuz650/RISC-V/tree/main/test/bubble_sort". Note that all files within this archive can be opened by a text editor. Contents of this archive are as follows:

- bubble_sort.c: Original algorithm written in C.
- bubble_sort.s: Compiled and disassembled version of "bubble_sort.c", done for RV32I set.
- bubble_sort.instr: Instruction memory initialization file, containing machine code level instructions from "bubble_sort.s", through its <.text> section.
- bubble_sort.data: Data memory initialization file, containing read only data from "bubble_sort.s", through its <.rodata> section.

a. Create a testbench that contains an instance of your pipelined processor, one instance of instruction memory and one instance of data memory from Homework 5. Each memory should have 128 words, with their word length set to 32-bits.
b. Connect memories to the processor within the testbench.
c. Initialize the instruction memory with "bubble_sort.instr", and data memory with "bubble_sort.data" files; using $readmemh function.
d. The program you've loaded in your instruction memory takes the unsorted integer array arr[] = {195,14,176,128} from data memory from data memory addresses 0x10C through 0x118, and writes its sorted version to the data memory addresses 0x1D4 through 0x1E0. Perform a behavioral simulation and run the program until PC value becomes 0x108.

e. Because you do not have solutions for all hazards probably your processor will not run this program successfully. Show the areas on your waveform that causes the problems.
f. Install RISCV toolchain from https://github.com/riscv-collab/riscv-gnu-toolchain.
g. Compile "bubble_sort.c" in order to produce ".s" and ".data" files.
h. Find the places to introduce NOP instruction in ".s" in order to solve the problems you noticed in 4.e.
i. Insert NOP instructions to ".s" and reproduce ".data" from this new assembly code.
j. Initialize the instruction memory with "bubble_sort.instr", and data memory with "bubble_sort.data" files; using $readmemh function.
k. Perform a behavioral simulation and run the program until PC value becomes 0x108.
l. Show that the program runs successfully.

5. OpenLane Flow:

a. Apply full OpenLane flow to the final version of your processor. Max fanout, capacitance and slew warnings can be ignored.
b. After successfully completing the flow, navigate through OpenLane reports to obtain approximate maximum clock frequency, total die area, total cell count and approximate power consumption of your design.
c. Add the report folder associated with your run to your design submission archive folder.
d. By using OpenRoad GUI interface, add post place and route layout schematic of your design to the report.

Note: Consider using ROUTING_CORES and KLAYOUT_XOR_THREADS configuration variables in your config.json file. These variables set the number of cores to participate on certain time-consuming sections of OpenLane flow, thereby decreasing the runtime. Refer to link below:

- https://openlane.readthedocs.io/en/latest/reference/configuration.html

Please write your reports as a group homework. Do not forget to add cover page and page numbers. Try to write the report regularly and well organized! You may use tables, explanations, figures, drawings, etc.