Computer Organization – Project – Phase IV

Email: iustCompOrg+4012@gmail.com

In the phase of this project, you'll design the Load Unit using the `Data_Memory.v` included in the zip file. In the end, you are going to connect all the parts of your CPU and run a simple program on it and monitor the result. (sequential CPU)

As we mentioned before in phase I, `Core.v` file is a testbench which we designed for your CPU and you won't need to connect your CPU modules together again or write any new testbench in order to test your CPU. Code which is going to run on your CPU is included in `Instruction_Memory.v` which you've seen and worked with it before in previous phases.

1) You are going to design Load Unit using Data Memory. Include `Data_Memory.v` in `Load_Unit.v`. Data Memory has the following input signals:

`DmemWriteEnable / DmemReadEnable / DmemAddress (32 bits) / DmemWriteData (32 bits)`

And one output:

`DmemReadData (32 bits)`

Load Unit is going to load data from the data memory into the CPU according to different stages. Following signals are considered for Load Unit as inputs:

`Stage / opcode (7 bits) / bus_rs1 (32 bits) / immediate (32 bits)`

And for output:

`load_unit_output (32 bits)`

Create a Data_Memory inside your module and. Keep checking the stage in an `always` block; `if stage == 2`, assign `bus_rs1 + immediate` to a `reg` for keeping the address.

Create a wire named `memory_read_enable` and assign it according to `opcode` and `stage` counter. In the end, if `memory_read_enable == 1` then the output of your load unit will be the `memory_read_output.` Else, set the output on high-z state.

[Hint: take a look at `RISCV_CPU_Datapath.pdf`. Block diagrams will help you with designing your CPU.]

2) Now all the modules that together construct your complete CPU, are completed and connected sequentially in `Core.v`. For this part, this is a complete testbench that inputs machine code in `InstructionMemory.v` and data from `Data_Memory.v` to your core. You can use either waveforms or print statements in terminal to view the final result.

Bonus point) In the previous phases of this project, you have designed in total, 4 stages of a CPU:

Fetch / Decode & Brach Control / Execute / Write Back

Connect these stages with latches between them and pipeline you RISC-V core.

By pipelining your circuits, hazards will be raised such as data hazards and control hazards (branch hazards). There are many simple and complex ways to deal with pipeline hazards.

Implement a solution to eliminate these hazards and run the code on your core. Compare execution time of the program in your pipelined versus non-pipelined CPU.

***** Additional bonus points will be awarded to more complex or creative solutions that contribute to the performance of the CPU.

<u>Notes:</u>

Send all your assignment related files (*Top module* and *Testbench* [.v] and [.vvp] and [.vcd] files) along with a <u>detailed report</u> in [.pdf] format all in one zip file to the email address of the class.

For this assignment you can work in groups of two, each participant must submit the assignment individually with their respective name and student number.

If you have any questions regarding this assignment, feel free to contact us.

Please submit your final project in the following format:

Name_StudentNumber_Project (BillGates_12345678_Project)

Good Luck!