# Computer Architecture

## Bonus (5 integer points)

**Due:** Last Section.

* Group assignment (students <= 3).
* Points colored with Red are mandatory to collect the Bonus points.

**Introduction**

The goal of this project is to further introduce you to the details of modeling digital systems in Verilog while enforcing the concepts of processor design that you have been learning in class. You will be given a basic model of the Single-Cycle Processor. You will simulate this model, extend its design to perform additional instructions, and simulate your final circuit.

### Part 1 – Building and Simulating the Processor Model

Follow these steps to download the processor model and build it in Verilogger.

1. Download Single Cycle Project from the following link (Download project)
2. Download Verilogger from the following link (Download Verilogger )
3. Start Verilogger. Go the timing diagram window and press the "Add clock" button to create a clock to drive the simulation. Fill in the clock name **clk** and click the "OK" button (this will create a period of 100ns).
4. Use the "Project->Add File(s)" command to add the source files you downloaded in Step 1 to your project (you can add all of the files at once). Save the project using the "Project->Save As" command.
5. Compile the simulation using the "build" button. Check the "errors" window to see if there are any errors or warnings (there shouldn't be).
6. The top-level module in the processor design is called **mips_single**. It only has two input ports: **clk**, and **reset**. To view internal signals, go to the project window, clock on the "+" next to the the mips_single entry, and click on the the "+" next to the "signals" submenu. You can now add "interesting" signals to the simulation trace by right-clicking on a listing and selecting "watch connection" from the popup menu. Do this to trace the **pc**, **pc_next**, and **instr** signals.
7. Go to the timing diagram window and set the reset signal HIGH for one clock signal, then LOW for several following clock cycles. Click the "Run" button to run the simulation and observe what happens to the signals on the timing diagram. You should notice several different instructions appearing on the instr trace, while pc is incremented by 4 at the end of each clock cycle.
8. Trace additional signals to examine what is happening in the datapath as each instruction executes. Note that you can examine the source code of the rom32 module to see what each instruction is. Experiment by placing different instructions 2 in the rom32 model and examine the simulation to verify that different instructions execute properly.

## Part 2 – Extending the Processor Design

In this part of the project, you will modify the processor to implement the addi and bne instructions. In addition, you will implement the jump (j) instruction as described in the class. Note that this should be possible just by modifying the datapath and controller Verilog files – it should not be necessary to change any of the "low-level" modules (except for rom32 to test the instructions). For your own sanity, it would be a good idea to do these modifications one at a time!

1. Modify the datapath and controller to implement the addi (add immediate) instruction. Modify the instruction memory to test this instruction and verify that it executes correctly. Comment your changes to explain the modifications that you made. Simulate the execution of an addi instruction in your modified processor and verify that it operates correctly.
2. Modify the datapath and controller to implement the bne (branch if not equal) instruction (in addition to executing the existing beq instruction). Modify the instruction memory to test this instruction and verify that it executes correctly. Comment your changes to explain the modifications that you made. Simulate the execution of an addi instruction in your modified processor and verify that it operates correctly.
3. Modify the datapath and controller to implement the j (jump) instruction. Modify the instruction memory to test this instruction and verify that it executes correctly. Comment your changes to explain the modifications that you made. Simulate the execution of an addi instruction in your modified processor and verify that it operates correctly.
4. Hand in a report of your modified code. Use a highlighting pen to show where your modifications were made. Also hand in printouts of timing diagrams for each of the instruction simulations.

## Part 3 – Simulating the Modified Processor

1. Write a short (8 instructions or less) assembly-language program that tests the new instructions that you have added to the processor design. Include a loop in your code that executes at least four times.
2. Convert your assembly-language program to binary either by hand or using the assembler built into the SPIM simulator. Modify the rom32 model to store the instructions of your program.
3. Simulate your program to show that it operates properly. Print a timing diagram and annotate it to explain what your program is doing at different times during its execution.
4. Hand in a report of your program as written in assembly language, a listing of your modified rom32 module, and your annotated timing diagram.
5. Finally, indicate your estimate of the amount of time you spent on this project, what you felt was most valuable and least valuable about this project, and any suggestions you might have for improving this project in the future.