Laboratory 4

Objective

The objective of this laboratory is to develop a structural model for the KURM data path and a behavioral model of the KURM controller and to integrate the two components to model the CPU.

Problem 1

Design and implement a structural model for the KURM data path. Use your behavioral register file and ALU models from Laboratory 3 in conjunction with other components from the KURM library to implement your system. No behavioral VHDL is allowed in the data path description other than the models for KURM components you have been provided. The only exception is that you may design components to calculate offset values for `BNE`, `LW`, `SW` and the `JMP` operation. Note that efficiency in these components is of utmost importance.

Your data path should combine instruction and data memory fetch and store into a single memory access subsystem. Use multiplexers to select and route data to the memory access subsystem and route retrieved data appropriately. Note that handling output can be achieved largely using load-enables for various components and should not require the use of a demultiplexer.

Your data path should be implemented as a single entity with a structural architecture using as many structural components as necessary. Document all control settings and outputs from the data path. Remember that you must include the system clock to trigger data storage elements.

Note that, in the design of the data path, you should use behavioral models for both the register file and the ALU. You may modify your behavioral models as necessary to suite your particular design. It is unlikely that your register file will require modification, but the control signals to the ALU may be reconfigured to more effectively interpret instructions.

Problem 2

Design and implement a behavioral model for the KURM controller that implements KURM instructions. Your controller should be implemented behaviorally using case statements and/or if statements in a VHDL process. Note that each instruction may involve different numbers of clock cycles. Make certain that you draw a state transition diagram and systematically approach your design.

Your controller should be implemented as a single entity with a behavioral architecture implementing the controller. Document all inputs and outputs defined for the controller. Your controller should change states on each rising edge of the clock pulse.

Problem 3

Integrate your controller and data path to define a structural architecture for the KURM. Use the entities from Laboratories 1 and 3 to define a structural architecture for KURM. This architecture should use the same entity model developed for Laboratory 2. Modifications may be made to correct errors from Laboratory 2 if necessary.

Problem 4

Test your structural KURM implementation using a test bench that compares execution with the instruction interpreter from Laboratory 2. Your test bench should execute both models in parallel on the same program inputs. Outputs from the two CPU models should be compared when appropriate to assure correctness of the models. Note that timing differences will certainly be present in your models. Some output differences may exist as well, but primary data outputs should be equivalent between the models.

Optional Problem

Rather than implementing a behavioral controller, implement a controller that uses a ROM to implement the controller state machine. Your ROM should decode an instruction input and current state input into control signals and a next state value. Store the state in a register and use it along with the instruction being executed to access the appropriate ROM location.