Computer Architecture
Multicycle Processor
CPSC 321 Computer Architecture
Implementing MIPS–Lite Processors in Verilog (100 Points)
Due Date: 12/08/2003, at 1200 hrs, via `turnin`.
Work in groups of up to four students

1 Objective

In this project you will build a full-blown multi-cycle MIPS processor. You need to verify that they execute a given subset of the MIPS instruction set.

This alternate project is intended for teams that did not succeed implementing a near-complete version of a single-cycle processor. You need to get permission of your instructor to do this alternate project.

2 Multi-Cycle MIPS Processor

This part requires you to develop a multi-cycle processor. The specification of functionality for the datapath is identical to that of the single-cycle processor. The main difference is that the datapath has to be multi-cycle, as we discussed in class and in [Chapter 5].

2.1 Supported Instructions

Build a multi-cycle datapath that implements the following subset of MIPS instructions.

Type	Instructions
arithmetic (unsigned)	`addu, subu, addiu`
arithmetic (signed)	`add, sub, addi`
logical	`and, andi, or, ori, xor, xori`
shift	`sll, sra, srl`
compare	`slt, slti, sltu`
control	`beq, bne, bgez, bltz, j, jr, jal`
data transfer	`lw, sw, lui`

You have to implement test programs for your machine as you did for the single-cycle processor.

2.2 Verilog Controller for the Multi-Cycle Datapath

Build the Control Unit (CU) for the multi-cycle datapath. You may use any behavioral Verilog statement available to develop the CU. The Verilog code that specifies the behavior of your CU must be as much structured and clearly written as possible. Use the Verilog `‘define SYMBOL code_to_substitute` as much as possible and specify upper bounds for units as `parameter n = ...`. Pay attention to the time that you compute the "next state" for your FSM. We can tell the correct next state only after all current input is available. Make sure that you implement the current state transition at the falling edge of the current clock. Outputs (control signals) should be computed only after the input is valid and stable. Remember that control signals that enable the loading of storage elements must be *explicitly* de-asserted if this storage must not be modified at the current clock step.

Turn in a copy of the state transition diagram, your transition matrix, and output matrix for the control unit, along with processor schematics, diagnostic programs, and a simulation log that shows the correct execution of the model. You need to specify all control signals that drive the datapath and all conditions the datapath generates at each clock step. You need to mention if your FSM is Moore or Mealy machine.

3 Deliverables for Submission

In summary, your project needs to accomplish the following parts.

1. [**Code Deliverables**] Provide the following parts for the multi-cycle MIPS-Lite processor.
   (a) The 32–bit register file.
   (b) The 32–bit ALU.
   (c) The completed multi-cycle data-path.
   (d) Turn in a copy of the state transition diagrams, your transition matrix, and output matrix for the multi-cycle control unit, and processor schematic.
   (e) You need to specify all control signals that drive the datapath and all conditions the datapath generates. You need to mention if your FSM is Moore or Mealy machine.
   (f) Provide the diagnostic program(s), and a simulation log that shows the correct execution of and test part for your processor.

2. [**Demonstration**] Demonstrate the execution of your project to your TA. You need to provide all test programs, as well as, the expected results. Print descriptive messages (use `$display(...)`) concerning which task your model is performing at each time.

   During demonstration your models are expected to execute a number of test MIPS programs.