**ELC 5313: ADVANCED Computer Architecture**

**Fall 2025**

**Lab 2: ALU with Reservation Stations**

**Introduction**

In this lab, you will implement a small portion of Tomasulo’s algorithm: an adder with reservation stations (RS). Reservation stations allow instructions to wait until all operands are available before executing, supporting out-of-order execution.

Unlike a simple ALU that immediately processes two inputs, this design:

* Supports multiple instructions waiting in reservation stations.
* Handles invalid operands (representing future results from other RS).
* Communicates results via a Common Data Bus (CDB).

Key concepts:

* Reservation Stations (RS): Hold operations until operands are ready.
* CDB (Common Data Bus): Transmits results from execution units to waiting instructions.
* Valid/Invalid Operands: Operands can be immediately available or pending from another RS.
* Tri-state buses: Allow multiple sources to safely share a common bus.

You will work with 32-bit signed operands and three reservation stations for the adder unit. The ALU supports addition, subtraction, and basic logic operations (AND, OR, XOR, NOT).

**Scope**

This lab introduces:

* Implementing reservation station logic in SystemVerilog.
* Handling instruction issue, execution, and CDB write-back.
* Using a provided testbench to simulate multiple instructions.
* Observing timing diagrams in Vivado to confirm correct behavior.

You will:

1. Complete the adder.sv module with reservation station logic.
2. Run simulations using the provided testbench.
3. Observe instruction flow: issue → execute → write-back.

**Provided Files**

* alu.sv – template for the adders module.
* alu\_tb.sv – testbench to verify your implementation. Students must use this testbench.

**Module: alu.sv**

This module should implement the following:

1. Accept arithmetic operation requests:  
   alu\_add, alu\_sub, alu\_and, alu\_or, alu\_not, alu\_xor.
2. Assign each operation to a reservation station (RS) if one is available.
3. Track which RSs are busy and which are waiting for operands.
4. Execute operations once operands are ready.
5. Communicate results back via the CDB.

**Reservation Station (RS) Logic**

**1. RS Slots**

* 3 stations: adder\_1, adder\_2, adder\_3.
* Each RS tracks:
  + Vj and Vk: operand values (if ready).
  + Qj and Qk: producing RS of operands (if not ready).
  + Busy: whether the station currently holds an operation.

**2. Issuing Operations**

On issue signal:

1. Check for the first available RS (Busy = 0).
2. Store operation and operands in the selected RS:
   * If operand is valid (A\_invalid = 0 or B\_invalid = 0), store in Vj/Vk and mark Qj/Qk as valid.
   * If operand is not valid (A\_invalid = 1 or B\_invalid = 1), store the producing RS in Qj/Qk.
3. Set issued to indicate which RS was used.

If all RSs are busy, set error = all\_rs\_busy.

**3. Execution**

* Each clock cycle, check if a RS has valid operands (Qj = Qk = valid) and if the adder unit is free.
* If ready, execute the operation (alu\_add, alu\_sub, etc.) and mark the unit as busy.
* Use #delay to simulate computation time.
* Output the result on CDB\_data and indicate the RS that produced it (CDB\_source).

**4. CDB Update**

* On the negative edge of the clock, update all RSs waiting for a value:
  + If CDB\_source matches Qj or Qk, copy CDB\_data to Vj or Vk and mark the operand as valid.
* This allows stations to dynamically receive operands as other operations complete.

**5. Priority & Starvation Prevention**

* Cycle RSs using mod3counter in a round-robin order:  
  Priority\_Station → Second\_Station → Last\_Station.
* Execute operations in this order to ensure fairness.
* Start from the file alu.sv located in ELC5313-Lab.

**Implementation Notes**

* Integration: Ensure adder.sv and mod3counter.sv are in the same project.
* Tri-state CDB: Only one RS can write at a time. Use conditional assignments for CDB\_data, CDB\_source, and CDB\_write.
* Error Handling: Reset issued and error at each clock cycle.
* Simulation Delays: Use #delay for adder computation time. Optionally, use #update\_delay for updating RS signals.

**Testing**

* Use the provided testbench (alu\_tb.sv) in ELC5313-Lab.
* Verify that:
  1. Multiple operations are issued correctly.
  2. Operations execute only when operands are ready.
  3. CDB outputs the correct results.
  4. RS allocation and priority cycling work correctly.
  5. The error flag is set if all RSs are busy.

**Deliverables**

1. adder.sv – completed module.
2. adders\_test.sv – testbench.
3. Screenshot of waveform simulation showing correct RS and CDB behavior.

**Grading Rubric**

* **Adder + RS implementation** (60 pts)
  + Correct handling of issue, execution, and CDB write-back.
* **Testbench correctness** (10 pts)
  + Easy points. Proper simulation of multiple instructions and CDB timing.
* **Successful simulation & waveform** (20 pts)
  + Timing diagram shows correct instruction progression and results. An example of what your waveform should look like is below.

A screenshot of a computer

AI-generated content may be incorrect.

* **Code clarity & comments** (10 pts)
  + Descriptive names and comments explaining design choices.

**Due Date**

**9/24/2025** (Submit individually via Canvas).