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

**Schedule**

* **Week 1 (today):**
  + Focus on **Continuous assignments** (RS availability and execution visibility).
  + Begin **Block 1** (instruction issue).
  + Continuous assignments and block 1 TODO should be completed by next Wednesday.
* **Week 2:**
  + Implement and refine **Block 2 (CDB update)**, **Block 3 (execution)**, and **Block 4 (cleanup)**.
  + Run full testbench and finalize waveforms.

You are welcome to work ahead.

**Provided Files**

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

**Important conventions used in this lab**

* **RS numbering:**
  + RS\_num\_of[0] = adder\_1
  + RS\_num\_of[1] = adder\_2
  + RS\_num\_of[2] = adder\_3  
    (These are the IDs that get written onto the CDB.)
* **Valid tag:**
  + valid = 6’b000000
  + If Qj or Qk = valid, that operand is ready and its value is stored in Vj/Vk.
* **Operand mapping**:
  + Operand A → Vj / Qj
  + Operand B → Vk / Qk
* **Round-robin priority:**
  + Priority\_Station, Second\_Station, Last\_Station are indices (0–2).
  + These rotate each cycle to avoid starvation.

**Block by Block Guide**

Each block represents a major feature in your ALU design. The structure below mirrors the way code comments are written — explaining what you’re building, what triggers it, and what to look for in simulation.

**Block 0 — Availability & Execution Status**

**Objective:** Provide visibility into which RSs are available or executing.

* **Outputs:**
  + available: High if any RS is free.
  + RS\_available: ID of highest-priority free RS (or no\_rs).
  + RS\_executing: ID of the RS currently executing (or no\_rs if idle).  
    **Waveform Check:** You should see these signals reflect RS allocation as instructions are issued and executed.

**Block 1 — Instruction Issue**

**Trigger:** On **negedge clock**, if issue is asserted.

**Algorithm:**

1. Check stations in priority order (Priority → Second → Last).
2. If a free RS is found:
   * Save the operation (operation[i]).
   * Mark the RS busy.
   * For Operand A:
     + If valid → store in Vj; set Qj = valid.
     + If invalid → store producing RS id into Qj.
   * For Operand B (same as above with Vk/Qk).
   * Set issued = RS\_num\_of[i].
3. If all RS busy:
   * issued = not\_issued, error = all\_rs\_busy.

**Waveform Check:** After an issue pulse, the chosen RS should update Busy, store the operation, and show correct V\*/Q\* values.

**Block 2 — Reservation Station Update from CDB**

**Trigger:** On **negedge clock**, when a CDB broadcast occurs.

**Algorithm:** For each RS:

* If Qj matches CDB\_source → update Vj and clear Qj.
* If Qk matches CDB\_source → update Vk and clear Qk.

**Waveform Check:** RSs waiting on that value should become ready after the broadcast.

**Block 3 — Begin Execution**

**Trigger:** On **posedge clock**, if unit is free.

**Algorithm:**

1. Search RSs in round-robin priority order.
2. If an RS is busy **and** both operands are valid, it can issue to the ALU.
3. Compute result based on opcode.
4. Drive results onto the CDB (with RS ID).
5. Mark unit busy until result is written back.

**Waveform Check:** See results appear on the CDB only when operands are valid.

**Block 4 — Cleanup After CDB Transmission**

**Trigger:** On **negedge CDB\_xmit** (when broadcast finishes).

**Algorithm:**

* Reset issued/error flags.
* Mark unit not busy.
* Free the RS that completed execution.

**Waveform Check:** After cleanup, the RS should return to idle state.

**Implementation Notes**

* **Tri-state CDB:** Only one RS can drive it at a time. Use conditional assignments.
* **Error Handling:** Clear issued and error each cycle.
* **Delays:** You may insert small delays (#) for adder computation to make the waveforms more realistic.

**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. alu.sv – completed module.
2. alu\_tb.sv – testbench.
3. Screenshot of waveform simulation showing correct RS and CDB behavior.

**Grading Rubric**

* **ALU + 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**

**10/08/2025** (Submit individually via Canvas).