CS2610: Computer Organization and Architecture Lab
Assignment #9

Submission deadline: 15th April 2019

Superscalar Processor Simulator: Extend the scalar pipelined processor design simulator, designed as part of Assignment #8, to support *2*-wide superscalar processor that can fetch *2* instructions per cycle, decode *2* instructions per cycle, issue *2* instructions per cycle, and commit *2* instructions per cycle. Add all the necessary components to support multi-instruction issue, out-of-order execution, in-order commit, operand forwarding, load bypassing, load forwarding, structural hazards handling, and false dependency elimination. Assume that each functional unit has a set (configurable parameter) of reservation stations. Note that source operands can be available either from the register file or from reservation stations.
In order to support superscalar processing, consider the following changes to the scalar pipeline processor:

- Split the **Instruction Decode Cycle** (ID) into **Decode stage** and **Dispatch stage**, where the **Dispatch stage** can perform both register read as well as instruction dispatch to reservation stations.
- **Execute** (EX) and **Memory access** (MEM) stages are combined together as a multi-cycle **Execute** stage (refer to the *superscalar organization*, given in slide #2 of Lectures 36-38).
- The **Write-Back** (WB) stage is split into **Instruction Completion** stage and **Instruction Retire** stage to deal with ALU/LD and ST instructions, respectively (the role of these two stages is discussed in Slide #7 of Lectures 36-38).

Assumptions:

- The simulator simulates the same set of instructions as was consider for the scalar pipelined processor design.
- When both *load* and *store* requests come to the data cache at the same time, *load* requests will be given priority.

Design the simulator in such a way that user can specify the number of entries in the reservation station, re-order buffer, store buffer, and the latency for each ALU operation. Given a program code (will be supplied at the evaluation time), the simulator has to compute the total number of cycles required to complete the execution of the code and the final result of the computation.