Project #4
EGCP 4210

Develop a simulation of a dynamically scheduled processor which executes a subset of the MIPS instruction set. You will work in teams of 2.

Specific instructions / discussion:

a. Your simulation should be of a single-issue architecture which employs Tomasulo's algorithm to provide dynamic scheduling (of integer operations), performs control speculation using a reorder buffer, and provides forwarding of memory operations. Your simulation should be done at about the granularity of the architecture shown in Figure 3.29.
b. The simulation interface should be similar to Project #3.
c. This simulation will be more abstract than Project #3, in which you had to simulate individual control signals. I envision this project to be an object-oriented design, with objects similar to the objects shown in Figure 3.29. Communication between objects should be with simple message passing (method calls). For example, to issue an instruction to the integer functional unit, you can use have the Issue object make a call something like integerALU.addInstruction(inst). Perhaps this would be proceeded by the Issue object making calls to the Reorder Buffer and/or Register objects to obtain the necessary values/virtual regs to be sent as part of the instruction sent to the integerALU. How an object responds to a message does not have to emulate the hardware in detail. In general, I expect that your objects will model the hardware fairly well (e.g., the Reorder Buffer object should have fields similar to what the hardware would have), but you do not have to model internal operations of objects exactly like the hardware.
d. You do not have to concern yourself with a memory hierarchy; rather, you can just assume 100% hit rate in cache and that all memory operations can be done in 1 clock cycle.
e. You do not need to handle any exceptions. Your architecture should have a ReorderBuffer which would be able to support them, but I do not expect you to generate or handle exceptions.
f. You must handle memory operations per the model we discussed in class. Stores should not actually update memory until commit. Loads should not execute unless all preceding stores have resolved their address and are known not to conflict. You should forward data from preceding stores to the loads, and not access memory. Note that in the case of forwarding the load buffer still needs to broadcast the result on the CDB, so anyone needing the value will be able to latch it in.

g. Your architecture will be ineffective without some form of branch prediction. I would like you to make this as simple as possible, but be able to have some predictions to support speculation. For example, you might make a simple BTB which stores target addresses, and just assume it can all happen in the IF stage with 0 cycle penalty for correct predictions. Rather than emulating a cache structure like a real BTB, you can just declare an array of 50 branch addresses, and just kind of store all branches that arrive without worrying about tags and cache lines and all that. Your branch execution unit can be separate or part of the integer unit. It should update the BTB with the actual result of the branch, and be able to flush the ReorderBuffer and all other necessary objects on a mispredict. A mispredicted branch can be handled as soon as it is discovered or when the instruction tries to retire.
h. The primary outputs of the simulation will be a count of clock cycles required, the number of instructions issued, and the number of instructions retired. You might also want to keep some other statistics like what was the largest number of instruction ever in the Reorder Buffer or how many issues stalls you had due to lack of reservation stations.
i. You may want to consider the use of a log file, since the data generated by the simulation could be sizable. You might consider a verbose/quiet mode also to manage the amount of data.
j. All data to run a program must start in memory. You cannot assume any values will be in register at the beginning of the program.
k. Your simulation should be developed in Java.

Required for turn-in:

a. Listings of your code.
b. Instructions for the use of your simulator.
c. A discussion of the simulation; this will likely require 3-4 pages. In particular, discuss why certain things are modeled the way they are in your simulation: e.g., how you handle branch prediction and control speculation, or how you handle memory operations. Clearly state assumptions; e.g., when a reservation station has all operands, does execution start that clock cycle or the next.
d. Your report should include a diagram of the hardware you are simulating (ala Fig 3.29).
e. An report on the results from running your two programs, including correctness, clock cycles required, number of stall cycles, and any problems and/or lessons learned from the simulation.