Computer Systems Architecture
Assignment 3
Due: Friday, February 4

This assignment is meant to test your understanding of several hardware techniques (the scoreboard, Tomasulo's algorithm, superscalar and out-of-order execution) and a compiler technique (loop unrolling) that all increase ILP, and to enable you to gauge how and why they have different effects on performance. It should be done individually -- no teams this time.

The assignment is to do **Problem 4.14 (a), (b)**, **(e)**, **(f)**, **(g)** and **(j)**. I recommend that you work on (a), (b) and (g) at the same time and on (f) and (j) at the same time.

You'll need to make some assumptions about the various pipelines. Below are some we came up with; please use them so that you are all working under the same set of assumptions (and therefore grading is easier! ;-)). If you want to modify them, please let us know now. If you need more assumptions, indicate them clearly on your homework.

There are several different solutions, depending on how you decide to rearrange the code. You should of course be aiming for the fastest solution.

General Conventions

Whenever you need to refer to cycle numbers, label cycle #1 as the first interesting event that happens. For instance, in parts (a) and (b), the first instruction should start executing in cycle 1, since that is all we are recording. Later on when we ask for you to record the cycle in which an instruction issues, then label the first instruction as issuing in cycle 1.

Assumptions for the simple pipeline (4.14 (a), **the simple pipeline with loop unrolling** (b), **and the superscalar** (g))

Assume the DLX pipeline of Figure 3.42 with the latencies of Figure 4.2 for the FP units. The functional units are integer, FP add, and FP multiply; the FP units are fully pipelined and forwarding is used to the maximum. There is a single write port to the registers.

We found it easiest to express the answer as a table of 3 columns, where the columns were the instruction, the cycle in which it begins execution, and the number of stall cycles.

Determine and explain the speedups of (b) over (a), and (g) over (b).

Additional assumptions for the superscalar processor (4.14 (g))

Assume you have 32 FP registers that you use in even-odd pairs (this should be sufficient for a 4-way unrolling of the loop without spilling code).

The superscalar DLX looks like DLX, with loads and stores performed in the integer unit. Contrary to the superscalar DLX in the text, you should allow two independent integer operations to perform in parallel (including any two load-store combinations). However, if two memory operations (not counting instruction fetches) want to access memory at the same time, one of them has to stall for one cycle. There is no limitation on forwarding, and there is no limit on concurrency in reading/writing registers. Assume the same FP latencies as for parts (a) and (b).

Instructions are fetched two at a time. Both instructions have to leave the IF stage before 2 new instructions can be fetched. Ditto for the ID stage.

Instructions of the same type have to complete in order, but integer operations can finish ahead of FP operations.

Here we used a table of 4 columns, where the columns were the instruction, the cycle in which it issues, the cycles in which it executes, and a comment field that explains why there was a stall, if any.

Assumptions for (4.14 (e, f, j))

Assume the latencies shown in the first **two rows** of Figure 4.63 for all three subparts.

Assumptions for the scoreboard processor (4.14 (e))

The integer unit does the load-stores, and the integer unit always takes just 1 execute cycle. An instruction goes through 4 stages: issue, read operands, execute and write results (IS/RO/EX/WR). A result written in stage WR is available for the RO stage of another instruction in the same cycle. A unit is made available in the same cycle as its previous result is written in the WR stage (this allows the integer unit, clearly a bottleneck here, to have instructions issued to it without too much delay -- recall that you cannot issue if the unit is busy).

Again, we found it easiest to express the answer as a table, this time with 7 columns, where the columns were the instruction, its functional unit, the cycle in which it issues, reads operands, executes and writes a result, and a comment field that explains why there was a stall, if any. Also show the scoreboard. In fact the status of the scoreboard at the end of the second iteration is not interesting at all. So show the timings of the first iteration and the second iteration up until the 4th instruction issues, and the scoreboard at that point.

By the way, you will notice that results are committed out of order. The CDC 6600 used polling instead of interrupts and no virtual memory, so this was less of a big deal.

Assumptions for the Tomasulo solution (4.14 (f))

Assume the processor of Figure 4.8 (and ignore the statement in the book which says only 1 FP unit). Assume no limitation in the number of reservation stations, load-store buffers and the size of the instruction unit queue. An instruction goes through 3 stages: issue, execute and write register (IS/EX/WR).

In addition to the units shown in Figure 4.8, there is an integer unit with an unlimited number of reservation stations that has an EX time of 1 cycle for all operations, including load, store and branch. If more than one of these instructions is ready to execute, then instructions get priority in the following order: loads, integer ops, stores. The EX stage of loads and stores does both the effective address computation and the memory access.

If there is a conflict to access the CDB, priority is given in this order: FP multiply, FP add, integer unit. The FP add and the integer unit are considered busy if they cannot broadcast their result. (However, the FP Add is fully pipelined. so instructions might still be able to execute.)

A result broadcast at cycle *i* is read during the same cycle, i.e., an instruction dependent only on this result can start executing at cycle *i+1*.

Again here, we might have out of order completion. But 4.14 (j) will remind you of how to take care of that problem.

Show the timings for the first iteration and up until the 4th instruction issues from the second iteration, as well as the status of non-empty reservation stations and the status of registers waiting for either results or stores still pending at this point.

This time express your answer as a table of 6 columns, where the columns are the instruction, its functional unit and reservation station, the cycle in which it issues, executes and writes a result, and a comment field that explains why there was a stall, if any. Then show the Tomasulo hardware data structures mentioned above.

Which implementation performs better on this code, the scoreboard or Tomasulo's algorithm, and why?

Assumptions for the Tomasulo solution with speculative execution but in order completion (4.14 (j))

Same general assumptions as in 4.14 (f), with the architecture of Figure 4.34 extended with an integer unit. Assume the reorder buffer is large enough.

An instruction now goes through 4 stages (IS, EX, WR, Commit). Show the timings for the first iteration and up until the 4th instruction issues from the second iteration, as well as the status of non-empty reservation stations, the status of registers waiting for results and the status of the reorder buffer at the time when the `ADDD` of the second iteration is issued.

Determine and explain the speedups of (j) over (f).