Lab3: Out-of-Order Pipeline with In-Order Retirement

This is an individual assignment. You can discuss this assignment with other classmates but you should code your assignment individually. You are NOT allowed to see the code of (or show your code to) other students.

OBJECTIVE

The objective of the third programming assignment is to do a performance evaluation of an out-of-order machine. In particular, you will equip the code to do register renaming, implement different scheduling algorithms (in order and out-of-order), and use a ROB to maintain a precise state. You will initially implement a 1-wide pipeline and later extend it to a 2-wide superscalar pipeline.

PROBLEM DESCRIPTION

The seven stage out-of-order pipeline that we discussed in class is shown in the Figure above. It consists of FETCH (FE), Instruction Decode (ID), Rename (RN), Schedule (SC), Execute (EX), Broadcast (BCAST) and Commit (COMMIT) stages. The newly added units include the Register Alias Table (RAT), Reservation Station (REST), and the ReOrder Buffer (ROB). More details about these individual units and newly added stages can be found in the powerpoint slide provided on T-square. To simplify this assignment, we will assume perfect branch prediction. We will ignore the cc_read and cc_write operations generated by the instructions (otherwise, you would need to rename the cc register also). We will also assume that memory instructions (LD/ST) in the pipeline do not conflict with each other, to avoid the complexity of the hardware associated with memory disambiguation. We will assume that all instructions, except LD, incur a latency of 1 cycle to execute. The latency of the LD instructions can be varied using configuration parameters.

We will use a trace driven simulator that is strictly meant for doing timing simulation. To keep the framework simple, we will not be doing any functional simulation -- which means the trace records that is fed to the pipelined machine does not contain any data values, and your pipeline will not track any data values (in REG, Memory, REST, ROB, PC) either. Furthermore, the traces only contain the committed path instructions. The purpose of our simulation is to figure out how many clock cycles it takes to execute the given instruction stream, for a variety of different machines such as different scheduling policies, LD latencies, and pipeline width.

You will be provided with a trace reader, as well as a pipeline machine for which the fetch() decode() and exe() stage are already filled for you. Your job is to do the following:

Part A (1 point): Create the functionality for the three newly added units RAT, REST, ROB. We have already created the *.h files for these objects, and your job is to fill the corresponding functions in the *.cpp file. The objective of this part is simply to create the three objects and compile them independently (using g++ -c filename.cpp). Note that you will not be able to debug these structures just yet, as you will need to have a working pipeline for testing. So, for this part, as long as you fill the *.cpp and submit the three files than can compile without error you will receive 1 point. You will be able to change these files for Part B. Note that if you do not do this part with seriousness, you may find Part B impossibly hard to finish within a few days!

Part B (7 Points for ECE6100, 9 points for ECE4100): For a 1-wide machine, integrate the created objects in your pipeline and populate the four functions of the pipeline: rename(), schedule(), broadcast() and commit(). You will implement two scheduling policies: in-order and out-of-order (oldest ready first)

What experiments to run:
B.1 Schedule in-order, load latency=1 cycle
B.2 Schedule ooo-oldest first, load latency = 1 cycle
B.3 Schedule in-order, load latency=4 cycle
B.4 Schedule ooo-oldest first, load latency=4 cycle

Part C (2 points): Extend your ooo machine to be 2-wide superscalar. Run the same four experiments as in part B, but for a 2-wide machine.

WHAT to SUBMIT (on T-square):

A. For Part A, submit the tarball or src.A as src.A.tar.gz (note you are allowed to change only the *.cpp files and not the *.h files).
B. For Part B and Part C, you can tarball the src.BC directory and upload as src.BC.tar.gz. We will also provide a script to generate the report to be submitted.

Note for ECE4100 students: You are not required to do Part C ; however, you can still choose to do it for Extra Credit worth 2 points.

REFERENCE MACHINE:
ECE STUDENTS: We will use **ecelinsrv7.ece.gatech.edu** as the reference machine for this course. (http://www.ece-help.gatech.edu/labs/unix/names.html).

Before submitting your code ensure that your code compiles on this machine, and generates the desired output (without any extra printf statements). Please follow the submission instructions. If you do not follow the submission file names, you will not receive the full credit.

Reference simulator:
Here is a link to the executable & the results.

https://drive.google.com/open?id=0B7LNjwui0KH_WVdKcW9KOXBBdjg

This has been compiled on ecelinsrv7. You can run the executable in the usual way:
./sim -pipewidth 1 -schedpolicy 0 -loadlatency 1 ./traces/bzip2.ptr.gz

Make sure you change permissions if necessary (chmod 744 ./sim)

FAQ:

1. **What are the convention changes from Lab2 to Lab3**

We will use a struct called inst_info to track both the ISA state (sr1,sr2,dr) as well as microarchitectural state (srtag, drtag, sr1ready, sr2ready) as well as simulation metadata (inst_num). You can use inst_num to estimate the age of the instruction. Furthermore, we will use "-1" to denote invalid values or when a value is not needed. For example, if sr1reg=-1 it means sr1reg is not needed. This simplifies the number of variables the inst info needs to carry throughout the pipeline.

2. **Why are we not simulating condition code?**

It will significantly increase the complexity of the pipeline as you would need to rename the cc for each instruction that does a a cc_write. So, in essence an instruction would have two sources and two destinations (destreg and cc). To keep the simulation model tractable (so that students can finish this assignment in a couple of weeks), we will ignore cc read and cc write (in fact, we are ignoring branch instructions altogether, and they are treated as "OP_OTHER" instructions).