

Project 2, CSci 330, Fall 2006

Due: Oct 4, 2:10pm. Value: 60 pts.

Note: *You may work with another student on this project if you please; if so, you should submit your project jointly.*

Submit your code solution (the OutOfOrder class only) by e-mail to burch@grendel.hendrix.edu. While you are free to include your analysis as an attachment also, I will only grade your analysis if submitted on paper.

In class, we saw how dynamic multiple-issue architectures such as the Intel Core Microarchitecture will reorder user programs in order to make maximum use of the available pipelines. Such out-of-order execution can be useful even for architectures containing only one pipeline.

As an example where out-of-order execution can be helpful, consider the following instruction sequence.

```
        andi    $t0, $a0, 1
        beq     $t0, $zero, even
        :
even:   addi    $v0, $v0, 1
```

In the same clock cycle, the `andi` instruction proceeds to the EX stage while the `beq` enters the ID stage. Recall that after accounting for data and control hazards, the branch decision will be made in the pipeline's ID stage. Thus, the `beq` instruction will need to stall while the EX stage performs its computation of `beq`.

A CPU could avoid this stall this by reordering the execution so that the `addi` instruction starts down the pipeline before the `beq` instruction does. (Of course, if the branch ends up not being taken, then the `addi` instruction will have to be canceled. In this case, there is neither loss nor gain for starting `addi` early.)

For this project, you should investigate this issue by modifying a pipeline simulation program and writing a short paper comparing its performance. Your comparison should include numerical comparison(s); it should also describe difficulties that you experienced during the development process that an implementor should be concerned about. Though I will not evaluate the paper based on length, it is possible to do a good job with this paper within two pages.

The Assignments section of the course Web page contains an existing pipeline simulator, written in Java, which will be very useful for developing your simulation. (After writing this myself, I feel confident in asserting that you don't want to write this on your own.) After creating a Java project within Eclipse, download the ZIP file into the project directory. Then perform the following at the shell prompt.

```
unix% cd eclipse-work/ProjectName
unix% unzip proj2.zip
unix% rm proj2.zip
```

The MIPS simulation does not do delay slots; however, it does perform the other workarounds, including register forwarding and branch decisions in the ID stage. The MIPS simulation supports very few instructions, but enough to do simulations using the provided `3n+1.asm` program. I may well provide more versatile versions later in the week.

As distributed, the program uses a regular in-order pipeline; this is set up near the end of the `Main` class, where you see it create an `InOrder` object. For your out-of-order simulation, you should edit the `OutOfOrder` class and edit this line in `Main` to use your code instead.

You will find several static methods in the `Decode` class useful for determining instructions. It is better to use these methods than to duplicate them yourself, since it will allow you to use any more versatile versions distributed later.

When you do out-of-order execution, you can feel free to write it so that it does only one inversion at a time. That is, it never skips more than one instruction in the ordering. You can try to be more flexible, but it may end up being more complicated, and it isn't likely to be helpful.

Of course, inversions would only occur if the previous instruction (call it A) issued into the pipeline will modify register d , and the instruction (call it B) that would proceed it will use register d . However, if the following instruction C (which is being considered to be issued before B, even though it belongs after B) uses d also, there is little point in issuing C first. C should *not* be issued if it uses the register (if any) that B modifies. Although it's possible to get around it, I would also recommend not issuing C if it modifies the same register than B modifies.

After getting past the last paragraph, you will also have to worry about what happens when a branch is mispredicted. In this case, up to two instructions may have to be aborted. In particular, in the above scenario, if B is a branch instruction whose destination is mispredicted, then C (which is after B in the pipeline although it belongs before) and D, the instruction that is issued just after B, will both have to be aborted. (If you want to get fancy, then actually if it happens that B branches to D, then D would not need to be aborted.)