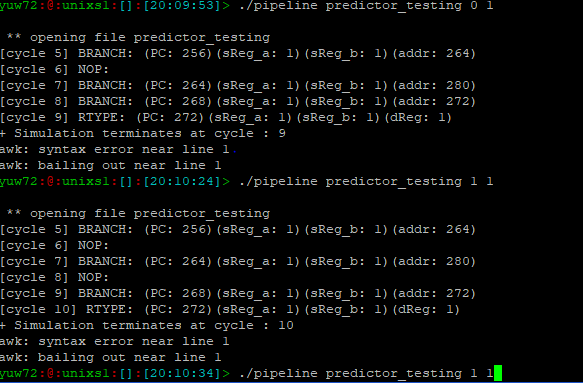
This writeup demonstrates the testing example of data hazard for load (lw) instruction and control hazard for predictions of both not taken and taken. The testing file is called “predictor testing”. As the picture shows, the above one is “not taken”. Although the PC is not as large as the one provided in sample.tr, it is still working as long as the bit 4 to bit 9 are the same. Cycle five thinks it is not taken, but it is taken, so there is a NOP. Cycle 7 and cycle 8 predicts not taken, and it predicts correctly, so there are no NOPs. Thus, it can approve my branch “not taken” code is correct.

The lower one is “branch predictor”. I intentionally create a conflict miss so that we can visualize whether the NOPs are inserted correctly. As it shows, the conflict miss is caught and also, the branch instruction in cycle 9 predicts correctly after adjusting its value in hash table. Thus, we can prove the branch predictor is written correctly.



The data hazard is shown below. We create a testing file called “data\_hazard\_testing” file. It contains all sorts of possibility that the pipeline may encounter. For simplicity, we made PC number starting from 0, and make sReg\_a, sReg\_b and dReg from 1-6. Address doesn’t matter in the data hazard detection. As you can see, from cycle 5 to cycle 15, the data hazard comes from data dependency between sReg\_a or sReg\_b and dReg. From cycle 16 to cycle 23, data dependency is between sReg\_a and dReg.

