Project 2
CSCI 540
Advance Computer Architecture
Jordan Manier

OoO pipeline

Out-of-order execution, also known as dynamic scheduling, is a technique used to get back some of the wasted execution bandwidth. With out-of-order execution, the processor would issue each of the instructions in program order, and then enter a new pipeline stage called "read operands" during which instructions whose operands are available would move to the execution stage, regardless of their order in the program. The term *issue* could be redefined at this point to mean "issue and read operands."

- **Project2.cpp**

Written in c++ is my source code for the LC. Also Included is the branch prediction fragment. This code is used to read the .MC file produced by the compiling the .ASM.

- **a.c**

The a.c file was provided to the class. This is used to convert valid assembly code for the LC into machine code.

Simulator Operation

OoO pipelining is structured in a 7 stage design. It consists of both dynamic scheduling and basic branch predication using a Branch Table and 2-bit state machine for predictions. The pipelines in conducted:

- Register renaming
- Instruction window
- Enhanced issue logic
- Reservation stations
- Load/store queue
- Score-boarding
- Common data bus

- **The Test Cases**

nand_test

I implemented this test case to make sure the nand test works properly. We get the correct value of the nand when executed.

memory_test

This test consists of a for loop in which regi1 is loaded from memory 11, reg 2 is added with the number in reg 1 and stored to reg 2, and then that is stored to mem 11. Basically, this loop is one big data hazard. We expect that the data will be forwarded when possible and that the system will stall in order to prevent corruption. Additionally, this test tests the sw instruction. When running this program through the simulator we see that data us appropriately forward added between the add and sw instructions resulting in no stall between those instructions. Additionally, the system does stall (and forwards data) between lw and add thus preventing

that hazard. Additionally, we see that the branch prediction correctly predicts the branch all but two times like in the previous test because they are similar.

noop_test

This test was designed to test basic functionality of the simulator it also tests the noop instruction.