Project 3 Overview

Keith O'Neal and Danny Mahota

April 24, 2019

1 Pipeline Overview

This project simulates the implementation of a basic pipeline for the UST-3400 architecture. Pipeline refers to an architecture design that breaks commands down into separate stages. This allows some overlap in execution which results in speed up, but the trade off is an increase in the number of hazards and potential for error.

In our implementation, there are 5 stages.

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Execute (EX)
4. Memory (MEM)
5. Write Back (WB)

Cycles are simulated by reading information from an old state and writing information to a new state in buffers. There is a buffer between each of the stages listed above, in addition to a final buffer that follows the Write Back Stage.

1.1 Instruction Fetch

This stage is responsible for fetching the instruction from memory and incriminating the program counter. It passes on the instruction and pc+1 to the next stage.

1.2 Instruction Decode

This stage is responsible for interpreting the instruction and retrieving the appropriate values from the register file, as well as reading in and sign extending an immediate value if appropriate. It passes all of this information on to the execute stage in addition to the information from the IF stage.

1.3 Execute

The execute stage does all of the calculations needed. These include getting results from the alu, calculating offsets, and branch targets. Forwarding, which will be discussed more in the hazards section, is also performed entirely in this stage of execution.

1.4 Memory

The memory stage is responsible for storing and retrieving data from memory. In this implementation, sw and lw are the only commands that do this.

1.5 Write Back

The write back stage is responsible storing any necessary values back into the register file.

2 Hazards

As previously mentioned, pipeline implementations have many more hazards associated with them. In this implementation, the most significant of these were data hazards and control hazards. Structural hazards were not a concern in this project because they deal with hazards caused by hardware. Since this project only deals with a simulated version of a pipeline, it was not necessary to consider these.

2.1 Data Hazards

Data hazards refer to data that is incorrect because the value was out of date when it was originally collected in the ID stage. This was solved in our implementation by forwarding data to the EX stage whenever a data hazard was detected.

Certain instances also required a pipeline stall because a command in the EX stage needed information that had not yet been loaded from memory. These cases were solved by adding a single NOOP, a bubble, into the pipeline so that the information could be gathered before the EX stage was carried out.

2.2 Control Hazards

Control hazards deal with branches, and they occur when the wrong branch is predicted. Because the branch is not executed until the fourth stage, three other instructions have already been loaded into the pipeline. This was solved assuming the branch would not be taken and flushing the incorrect instructions whenever the branch was taken.