Richard Wu | wuricha8 | 1008078296
Christian Vedtofte | vedtofte | 1012872125

ECE552 Lab 1 Report

This report states our answers to questions 1 and 2 of section 3 of the lab assignment and how our microbenchmark validated the correctness of our code changes to sim-safe.c.

Q1) We only have internal forwarding available to us, so the earliest a register value can be ready for a dependent instruction is during the writeback (WB) stage. The longest stall would occur if the dependent instruction is the immediate next one, as the forwarding would occur and cause a stall of 2-cycles (since ID and WB are 2 stages apart). A 1-cycle stall occurs if the dependent instruction is 2 instructions away.

Q2) We have full forwarding and bypassing available to us. Only memory or ALU instructions can cause a dependency, since they write to the register file or memory. For memory instructions, such as a load, values are ready at the end of the memory (ME) stage; for ALU instructions, values are ready at the end of the execute (EX2) stage. The longest stall would occur between a load and an immediate dependent instruction, as we must stall for 2-cycles before we can use the WX bypass (data used at start of EX1 for the dependent instruction); if the dependent instruction is 2 instructions away, only 1 stall is needed. A dependency between an ALU instruction and another instruction would cause a 1-cycle stall; we use the MX bypass. All other scenarios would not cause a stall due to appropriate bypassing.

The slowdown is calculated as such:

$$slowdown = 100\%(1 - speedup)$$

$$speedup = \frac{T_{ideal}}{T_{real}} = \frac{IC \times CPI_{ideal} \times T_c}{IC \times CPI_{real} \times T_c} = \frac{1}{CPI_{real}}$$

$$CPI_{real} = 1 + (\frac{1}{IC})(1 \times \#1\ cycle\ stalls + 2 \times \#2\ cycle\ stalls)$$

Our microbenchmark runs 6 loops with different tests for 1-6 million iterations respectively, so effects of initialization can be neglected. We can count the number of 1- and 2-cycle stalls within the loops, from the assembly breakdown of the C-code, and approximate the total IC as the instructions run in the loops. Then the CPI can be calculated from the formula above. All the scenarios described earlier are included within the microbenchmark. See Analysis with mbq1.c for our CPI results on the benchmark. Our microbenchmark compilation command is:

```
/cad2/ece552f/compiler/bin/ssbig-na-sstrix-gcc mbq1.c -O0 -o mbq1
```

Our sim-safe CPI results for the gcc.eio trace are:

Q1) 1.6642, slowdown = 39.91%

Q2) 1.3902, slowdown = 2.81%