ECE552 Lab 1
Performance Measurements
Jay Mohile, Leo Zhang

I. INTRODUCTION

This lab provides an introduction to developing performance metrics using the simple-sim simulator. A method is designed to detect, on each of two processor architectures, the presence of stalls caused by RAW hazards. Both processors are in-order multi-cycle, but differ primarily in their support for forwarding/bypassing.

II. MICROBENCHMARK

A microbenchmark was developed in C (-O2 optimization) only to profile performance for the first processor. During preparation, it was identified that only two stall-causing conditions exist:

1. A write immediately followed by a read. (2 cycle stall)
2. A write, followed by an unrelated instruction, followed by a read. (1 cycle stall)

Note that the specific types of read/write are not important for this particular processor. Since it possesses no bypassing capability, all writes become available during write-back, and must be consumed during decode.

The microbenchmark consists of two loops, included below, which exercise the 2-cycle and 1-cycle stall respectively. We expect each loop to trigger a 2-cycle stall each iteration (due to the while's set + compare expansion). Additionally, each test-case causes an intentional 1/2-cycle stall per iteration, respectively. With N iterations, we therefore expect 3N 2-cycle stalls, and 1N 1-cycle ones. Ignoring overhead, this is what we see.

```c
// Force a two-cycle stall on each iteration.
// This is caused by an immediate data access.
while (a < X) {
  a += 1; // WRITE
  // In the for loop case, GCC was smart enough to add the loop-increment here,
  // Which turned this into a 1-cycle stall.
  b += a; // READ

  RAW_BUFFER();
}
```

```c
// Force a 1-cycle stall on each iteration.
// This is caused by a data access separated by one instruction.
while (c < X) {
  c += 1; // WRITE
  asm("nop");
  d += c; // READ

  RAW_BUFFER();
}
```

Figure 1. 2-cycle stall (left) and 1-cycle stall (right) inducing loops.
Both loops have an additional 2-cycle stall each iteration, due to the "while less-than" check,
Which expands to a set+compare (RAW). For completeness, this is annotated more fully in Appendix A.

III. RESULTS

The overall goal of this was to use the developed metrics to evaluate CPI of a known benchmark, GCC. For both processors, we could calculate CPI and slowdown using the following relationship. This is based on the fact that 1-cycle and 2-cycle stalls raise the CPI as such, proportionally to their presence.

$$CPI = 1 + \frac{1 * N_{hazard,1\ cycle} + 2 * N_{hazard,2\ cycle}}{N_{instrs,dynamic}} \qquad slowdown = \frac{CPI - 1}{1}$$

Based on this analysis, we calculate the following results (full statistics in Appendix B).

Q1: CPI = 1.6642, slowdown = 66.42% **Q2:** CPI = 1.3903, slowdown = 39.03%

As expected, the pipelined processor (Q2) decreases our CPI significantly.

IV. ATTRIBUTION

Jay: Prelab, Microbenchmark, Simple-Sim, Report
Leo: Prelab, Report

APPENDIX. A

Description of assembled microbenchmark (excerpts).

Q1 Microbenchmark Loop

```
$L4:
  addu  $4,$4,1     # Increment (write) “a”, $4
  addu  $6,$6,$4    # Increment “b”, $6, using “a” (read)
 #APP
  nop               # Space to isolate above test from loop condition.
  nop
  nop
 #NO_APP
  slt $2,$4,$5      # Loop condition. Expands to two instrs,
  bne $2,$0,$L4     # which causes an addition RAW.
```

Q2 Microbenchmark Loop

```
$L8:
  addu  $3,$3,1     # Increment (write) “c”, $3
 #APP
  nop               # An unrelated instr, to reduce RAW to 1-cycle.
 #NO_APP
  addu  $7,$7,$3    # Increment “d”, $7, using “c” (read)
 #APP
  nop               # Rest same as first loop.
  nop
  nop
 #NO_APP
  slt $2,$3,$5
  bne $2,$0,$L8
```

APPENDIX. B

Full simulation results for the GCC benchmark.

```
sim: ** simulation statistics **
sim_num_insn              279373007 # total number of instructions
executed
sim_num_refs              109106589 # total number of loads and stores
executed
sim_elapsed_time                  7 # total simulation time in seconds
sim_inst_rate          39910429.5714 # simulation speed (in insts/sec)
sim_num_RAW_hazard_2cycle_q1     88182314 # total number of 2-cycle RAW
hazards (q1)
sim_num_RAW_hazard_1cycle_q1      9206516 # total number of 1-cycle RAW
hazards (q1)
sim_num_RAW_hazard_q1      97388830 # total number of RAW hazards (q1)
sim_num_RAW_hazard_2cycle_q2     20126394 # total number of 2-cycle RAW
hazards (q2)
sim_num_RAW_hazard_1cycle_q2     68796288 # total number of 1-cycle RAW
hazards (q2)
sim_num_RAW_hazard_q2      88922682 # total number of RAW hazards (q2)
CPI_from_RAW_hazard_q1       1.6642 # CPI from RAW hazard (q1)
CPI_from_RAW_hazard_q2       1.3903 # CPI from RAW hazard (q2)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                2166768 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment
base
ld_data_size                 264644 # program init'ed `.data' and
uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest
address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address
address
ld_target_big_endian              0 # target executable endian-ness, non-
zero if big endian
mem.page_count                  875 # total number of pages allocated
mem.page_mem                  3500k # total size of memory pages allocated
mem.ptab_misses                 894 # total first level page table misses
mem.ptab_accesses        1341120003 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```