Performance Drops

5-Stage pipeline with no forwarding, I found the number of

- 1-cycle RAW stalls: 9977846 (oneRAW)
- 2-cycle RAW stalls: 88158277 (twoRAW)
- And took note of the total number of instructions executed: 279373007 (sumInst)

An used the formulas:

- CPI = 1 + (oneRAW + 2 * twoRAW) / sumInst = 1.6668
- Slowdown = (1 - 1 / CPI) * 100% = **40.0% slowdown**

6-Stage pipeline with full forwarding, I found the number of

- 1-cycle load-to-use (LTU) stalls: 11228299 (oneLTU)
- 2-cycle LTU stalls: 20126394 (twoLTU)
- 1-cycle RAW (but not LTU) stalls: 57726317 (oneRAW)

An used the formulas:

- CPI = 1 + (oneLTU + oneRAW + 2 * twoCycle) / sumInst = 1.3909
- Slowdown = (1 - 1 / CPI) * 100% = **28.1% slowdown**

Microbenchmark

I used two strategies to make this easier for me, I set the **-O0 compilation flag** and used the "register int" type as much as I needed to. For my own sake I not only wrote a benchmark for Q1 (5-stage pipeline), I also extended the benchmark for Q2 (6-stage pipeline).

For each line of C code I added the corresponding assembly instruction as a comment, then (if applicable) a description of the hazard, what register caused it, and the corresponding case that it tests. I tested for 8 cases I found relevant:

- Q1: (Case 1) 1-cycle RAW, (2) 2-cycle RAW, (3) 0-cycle RAW b/c cascading stall
- Q2: (4) 1-cycle RAW but not LTU, (5) 0-cycle RAW b/c 2nd instruction is store, (6) 1-cycle LTU, (7) 2-cycle LTU, (8) 0-cycle LTU b/c cascading stall

I built on my microbenchmark until it tested each of the relevant cases at least one. When I was done, I recorded the following frequency of hazards types: (1)2, (2)4, (3)2, (4)1, (5)1, (6)1, (7)2, (8)1. This means that with 10000 loop iterations I should have around:

- Q1: 20000 1-cycle RAW, 40000 2-cycle RAW (hazards)
  I had: 20091 1-cycle RAW, 40854 2-cycle RAW
- Q2: 10000 1-cycle RAW not LTU, 10000 1-cycle LTU, 20000 2-cycle LTU
  I had: 10735 1-cycle RAW not LTU, 10035 1-cycle LTU, 20091 2-cycle LTU

Note: I used a relatively low number of loop iterations because when I used more than 40000, unwanted instructions were added to the assembly. However, when I used 30000 loop iterations the difference between the theoretical numbers and the exact numbers were the same. This proves that the difference is due to overhead.