CSC 462/562 Computer Architecture
Homework #3 answer key

1. See separate spreadsheet

2. See separate spreadsheet. The original code had 28 cycles of stalls and the branch delay (29 total added cycles). The new code only requires stalls between the fiv.d and fsd (22 stalls for the fiv.d latency and 1 cycle to prevent it and the fsd entering the MEM stage at the same time) with the branch delay slot being filled, so only 23 total cycles.

3. See separate spreadsheet.

4. Assume not taken: no penalty for conditional branches not taken (24% * 30%), 2 cycle penalty for all unconditional branches (5%) and 4 cycle penalty for all conditional branches taken (24% * 70%), or .05 * 2 + .24 * .70 * 4 = .77 cycle of penalties per instruction.

   Assume taken: 4 cycle of any branch not taken (24% * 30%), 2 cycle of penalty for all taken branches (5% + 24% * 70%), or .24 * .30 * 4 + (.05 + .24 * .70) * 2 = .72.

   Assuming taken is slightly better in this case.

5. We improved the 5-stage pipeline by moving branch decisions from EX (with the MUX changing the PC in MEM) to ID. This allowed us to reduce the branch penalty from 3 cycles to 1 and then we try to let the compiler schedule the branch delay slot to remove all penalties. Why not do something similar for the MIPS R4000?
   a. Because the MIPS R4000 is superpipelined, the instruction is not fetched until the end of the IS stage, so we can’t compute the PC+Offset or decode the instruction earlier than the RF stage.
   b. If the branch condition is being computed by an earlier instruction, it adds 1 cycle of RAW hazard stall because the computed condition occurs in the EX stage and is now needed in the RF stage, and if the branch condition is being loaded, it adds 1 cycle of RAW hazard stall because the load is completed in the DS stage and now needed in the RF stage.
   c. Branch penalties are now 2 cycles (assuming no RAW hazard stalls from moving branch decisions earlier in the pipeline) if the branch is taken and 0 cycles if the branch is not taken. 70% of a single branch delay can be filled but the other branch delay slot is not filled. Branches are taken 100% of the time for the unconditional branches and 75% of the time for conditional branches. This gives us a branch delay per instruction of .04 * .70 * 1 + .04 * .30 * 2 + .20 * .75 * .70 * 1 + .20 * .75 * .30 * 2 = .247, so a CPI of 1.25

6.
   a. The breakpoint is fetched with the instruction, so determined in the ID stage.
   b. Because this type of interrupt is asynchronous, it could arise in any of the 5 stages. We might assume though that the CPU only checks for interrupts in one stage, the WB stage.

c. IF: any instruction fetch, no other exception should arise, so this would be any type of memory violation
   ID: any illegal op code but for no other reason
   EX: div x3, x2, x0 (x0 = 0, can’t have this for a denominator)
   MEM: any illegal memory access for a load or store as in lw x3, 0(x0)
   WB: none