CS433: Computer Architecture – Fall 2021
Homework 2
Total Points: 33 points
All students should solve all problems
Due Date: September 28, 2021 at 10:00 pm CT
(See course information slides for more details)

Directions:

- **All students must write and sign the following statement at the end of their homework submission. "I have read the honor code for this class in the course information handout and have done this homework in conformance with that code. I understand fully the penalty for violating the honor code policies for this class." No credit will be given for a submission that does not contain this signed statement.**
- **On top of the first page of your homework solution, please write your name and NETID, your partner's name and NETID, and whether you are an undergrad or grad student.**
- **Name your homework solution file as *firstname_lastname_hw2.pdf***
- **Please show all work that you used to arrive at your answer. Answers without justification will not receive credit. Errors in numerical calculations will not be penalized. Cascading errors will usually be penalized only once.**
- **See course information slides for more details.**

Problem 1 [5 points]

Consider two different machines. The first has a single cycle datapath (i.e., a single stage, non-pipelined machine) with a cycle time of 15 ns. The second is a pipelined machine with 5 pipeline stages and a cycle time of 3ns.

Part (A) [1 point]

What is the speedup of the pipelined machine versus the single cycle machine assuming there are no stalls?

Solution: The speedup is $\frac{15\ ns}{3\ ns} = 5$.

Grading: 1 point for the correct answer. No deduction for calculation errors.

Part (B) [2 points]

What is the speedup of the pipelined machine versus the single cycle machine if the pipeline stalls 1 cycle for 25% of the instructions?

Solution: $New\ CPI = 1 + .25 \times 1 = 1.25$

Since the number of instructions is the same, the speedup is $\frac{CPI_{old} \times cycle\ time_{old}}{CPI_{new} \times cycle\ time_{new}} = \frac{1 \times 15\ ns}{1.25 \times 3\ ns} = 4.$

Grading: 1 point for correct new CPI. 1 point for correct speedup equation. No deduction for calculation errors.

Part (C) [2 points]

Now consider a 4 stage pipeline machine with a cycle time of 3.1 ns. Again, assuming no stalls, is this implementation faster or slower than the original 5 stage pipeline? Explain your answer.

Solution: The 5 stage machine is faster. This is because it has a smaller cycle time, which results in a faster overall execution time (since there are no stalls, they both have the same CPI).

Grading: 1 point for correct answer. 1 point for explanation.

Problem 2 [4 points]

Consider two different 5-stage pipeline machines (IF ID EX MEM WB). The first machine resolves branches in the ID stage, uses one branch delay slot, and can fill 80% of the delay slots with useful instructions. The second machine resolves branches in the EX stage and uses a predict-not-taken scheme. Assume that the cycle times of the machines are identical. Given that 35% of the instructions are branches, 25% of branches are taken, and that stalls are due to branches alone, which machine is faster? To get any credit, you must justify your answer.

Solution: For the first machine, a cycle is wasted every time a delay slot can't be filled i.e. for 20% of the branches. Thus, $CPI = 1 + 0.35 \times .20 \times 1 = 1.07.$

For the second machine, two cycles are wasted due to the unknown target address for 25% of the branches. Thus, $CPI = 1 + .35 \times .25 \times 2 = 1.175.$

Therefore, the first machine is faster.

Grading: 2 points for the CPI of the first machine. 2 points for the CPI of the second machine. No deduction for calculation errors.

Problem 3 [10 points]

Consider the following loop.

$loop$:

1. $ADDI\ R2,\ R2,\ \#1$
2. $LD\ R4,\ 0(R3)$
3. $LD\ R5,\ 4(R3)$
4. $ADD\ R6,\ R4,\ R5$
5. $MUL\ R4,\ R6,\ R7$
6. $SUBI\ R3,\ R3,\ \#8$
7. $BNEZ\ R2,\ loop$
8. $ADD\ R11,\ R12,\ R13$

Part (A) [4 points]

Identify all data dependencies (potential data hazards) in the given code snippet. Assume the loop takes exactly one iteration to complete. Specify if the data dependence is RAW, WAW or WAR.

Solution:

1) 1 7 (RAW)
2) 2 4 (RAW)
3) 2 5 (WAW)
4) 2 6 (WAR)
5) 3 4 (RAW)
6) 3 6 (WAR)
7) 4 5 (RAW)
8) 4 5 (WAR)

NOTE: There is no WAR or RAW from 7 1 because the loop only executes for a single iteration.

Grading: ½ point for identifying a dependence correctly per pair of instructions. ½ point for identifying all such dependences. -½ point for RAR.

Part (B) [2 points]

Assume a 5-stage pipeline (IF ID EX MEM WB) without any forwarding or bypassing hardware, but with support for a register read and write in the same cycle. Also assume that branches are resolved in the ID stage and handled by stalling the pipeline. All stages take 1 cycle. Again, the loop takes one iteration to complete. Which dependencies from part (a) cause stalls? How many cycles does the loop take to execute?

Solution: Stalls are caused by dependencies 5 and 7 above. The loop takes 17 cycles. The code would take 12 cycles if there were no stalls (5 cycles for the first instruction, then 1 cycle each for the next 7 instructions). The stalls from dependencies 5 and 7 add two cycles each, and the branch flush adds another cycle. Total = 12 + 2 + 2 + 1 = 17.

Grading: 1 point for listing dependencies, 1 point for computing number of cycles.

Part (C) [2 points]

Assume that the pipeline now supports full forwarding and bypassing. Furthermore, branches are handled as predicted-not-taken. As before, the loop takes one iteration to complete. Which dependencies from part (a) still cause stalls and why? How many cycles does the loop take to execute now?

Solution: Dependency 5 still causes a stall because the LD instruction doesn't load the required data till the end of its MEM cycle, whereas the MUL instruction needs that data in the beginning of that particular cycle.
The loop takes 13 cycles. The code would take 12 cycles without stalls (same as described above), and the stall from dependency 5 adds 1 cycle (forwarding reduces the number of stall cycles from 2 to 1). Therefore, total cycles = 12 + 1 = 13.

Grading: 1 point for listing dependency and reason, 1 point for computing number of cycles.

Part (D) [2 points]

If the pipeline from part (c) instead uses a branch delay slot, how would you schedule the instructions in the loop to minimize stalls? For this part, assume the *loop takes multiple iterations to complete*. Explain your answer.

Solution: Place/reschedule instruction 6 after instruction 7. Since instruction 6 must always execute, the branch delay slot will always have a useful instruction, regardless of the direction of the branch. Furthermore, instruction 6 will not cause stalls in instructions 2 and 3 of the next cycle because of the forwarding path.

Another option is to place/reschedule instruction 5 after instruction 7. Instruction 5 is independent of instruction 6, so moving it does not violate any dependencies. Moreover, instruction 5 must always execute, so the branch delay slot will always have a useful instruction, regardless of the direction of the branch. Finally, putting instruction 5 in the branch delay slot will not cause stalls in instructions 2 or 3 of the next cycle.

Grading: 1 point for correct answer. 1 point for explanation.

Problem 4 [14 points]

For this problem, we will explore a pipeline for a register-memory architecture. The architecture has two instruction formats: a register-register format and a register-memory format. In the register-memory format, one of the operands for an ALU instruction could come from memory.

There is a single memory-addressing mode (offset + base register). The only non-branch register-memory instructions available have the format:

Op Rdest, Rsrc1, Rsrc2

or

Op Rdest, Rsrc1, MEM

where Op is one of the following: Add, Subtract, And, Or, Load (in which case Rsrc1 is ignored), or Store. Rsrc1, Rsrc2, and Rdest are registers. MEM is a (base register, offset) pair.

Branches compare two registers and, depending on the outcome of the comparison, move to a target address. The target address can be specified as a PC-relative offset or in a register (with no offset). Assume that the pipeline structure of the machine is as follows:

IF RF ALU1 MEM ALU2 WB

The first ALU stage is used for effective address calculation for memory references and branches. The second ALU stage is used for operations and branch comparison. RF is both decode and register-fetch stage. Assume that when a register read and a register write of the same register occur in the same cycle, the write data is forwarded.

Part (A) [4 points]

Find the number of adders, counting any adder or incrementor, needed to minimize the number of structural hazards. Justify why you need this number of adders.

Solution: We need three adders – one for each of the two ALUs and one to increment the PC.

Grading: 1 point for each adder, 1 bonus point for the correct answer. Take off ½ a point if additional number of adders/incrementers are provided along with the correct answer.

Part (B) [4 points]

Find the number of register read and write ports and memory read and write ports needed to minimize the number of structural hazards. Justify why you need this number of ports for the register file and memory.

Solution: The register file is used in two pipeline stages; we will have to sum the ports required by both stages to find out how many ports we must have to avoid structural hazards. In the RF stage, we need up to three reads due to the branch instructions which can have three register operands. In the WB stage, we need one write. Therefore, we need three read ports and one write port for the register file.

Memory is accessed in two stages, IF for reading the next instruction from memory and MEM, which can either read or write to memory. So we need two read ports and one write port for memory.

Grading: 2 points for register file ports (½ point for each port), 2 points for memory ports (½ point for each port, ½ point bonus for correct answer). Take off ½ point if additional number of ports are provided along with correct answer.

Part (C) [3 points]

Will data forwarding from the ALU2 stage to any of ALU1, MEM, or ALU2 stages reduce or avoid stalls? Explain your answer for each stage.

Solution: The result of ALU2 could be used in the ALU1 stage or the ALU2 stage, and so forwarding to those stages is beneficial. There are instances where ALU2 to MEM forwarding is required to avoid a stall. E.g.,

ADD R1 R2 R3
Some other Instruction
STORE R1 0(R4)

If there is forwarding from ALU2 to MEM then we will avoid a stall.

Grading: 1 point for each stage.

Part (D) [3 points]

Will data forwarding from the MEM stage to any of ALU1, MEM, or ALU2 stages reduce or avoid stalls? Explain your answer for each stage.

Solution: The result of a memory access could be used in the ALU1 stage, and so forwarding to the ALU1 stage is beneficial. Forwarding to ALU2 is not needed since ALU2 comes after MEM. MEM to MEM forwarding is also required. E.g.,

LOAD R1 0(R2)
STORE R1 0(R3)

Grading: 1 point for each stage.