Homework #2

TA in charge: Wonyoung Lee

(E-mail: wy_lee@kaist.ac.kr)

Place to submit: The submission box in front of N1 #922

Due date: Nov. 20th (Tue.), 2018, 23:59:00)

Extended due date: Nov. 21th (Wed.), 2018, 23:59:00)

(*. **After the due date**, there will be a 50% penalty on your score)

(*. **After the extended due date**, submission is not allowed.)

- Solve the below problems, and write a report with the answers.
- Both English and Korean are fine for your report.
- Do not submit your homework on KLMS; Please submit it in the submission box.

1. In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
I1:   or r1, r2, r3
I2:   or r2, r1, r4
I3:   or r1, r1, r2
```

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
150ps	200ps	180ps

a. Indicate every dependence that incurs hazards.
   (Answer example: Dependence on register **r??** from **I??** to **I??**)
b. Assume there is no forwarding in this pipelined processor. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.
   (Answer example:

```
Ix:   XX r1, r2, r3
   NOP           // Delay to avoid data hazard on rXX from Ix
Iy:   YY r2, r1, r4
   NOP           // Delay to avoid data hazard on rXX from Iy
Iz:   XX r1, r1, r2
```

c. Assume there is full forwarding. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.
d. What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?
e. Add the minimum number of **nop** instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).
f. What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

2. In this exercise, we examine how resource hazards, control hazards, and Instruction Set Architecture (ISA) design can affect pipelined execution. Problems in this exercise refer to the following fragment of MIPS code:

```
sw r16,12(r6)
lw r16,8(r6)
beq r5,r4,Label         # Assume r5!=r4
add r5,r1,r4
slt r5,r15,r4
```

   a. For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If we only have one memory (for both instructions and data), there is a structural hazard every time we need to fetch an instruction in the same cycle in which another instruction accesses data. To guarantee forward progress, this hazard must always be resolved in favor of the instruction that accesses data. What is the total execution time of this instruction sequence in the 5-stage pipeline that only has one memory?
   b. For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. As a result, **MEM** and **EX** stages can be overlapped and the pipeline has only 4 stages. Change this code to accommodate this changed ISA. Assuming this change does not affect clock cycle time, what speedup is achieved in this instruction sequence?
   c. Assuming stall-on-branch and no delay slots, what speedup is achieved on this code if branch outcomes are determined in the **ID** stage, relative to the execution where branch outcomes are determined in the **EX** stage?

Assume that individual pipeline stages have the following latencies:

IF	ID	EX	MEM	WB
160ps	100ps	120ps	150ps	80ps

d. Given these pipeline stage latencies, repeat the speedup calculation from 2.b, but take into account the (possible) change in clock cycle time. When **EX** and **MEM** are done in a single stage, most of their work can be done in parallel. As a result, the resulting **EX/MEM** stage has a latency that is the original **MEM** stage latency plus 10 ps needed for the work that could not be done in parallel.

e. Given these pipeline stage latencies, repeat the speedup calculation from 2.c, taking into account the (possible) change in clock cycle time. Assume that the latency **ID** stage increases by 100% and the latency of the **EX** stage decreases by 20 ps when branch outcome resolution is moved from **EX** to **ID**.

f. Assuming stall-on-branch and no delay slots, what is the new clock cycle time and execution time of this instruction sequence if **beq** address computation is moved to the **MEM** stage? What is the speedup from this change? Assume that the latency of the **EX** stage is reduced by 20 ps and the latency of the **MEM** stage is unchanged when branch outcome resolution is moved from **EX** to **MEM**.

3. The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-Type	**BEQ**	**JMP**	**LW**	**SW**
40%	15%	15%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	**Always-Not-Taken**	**2-Bit**
35%	65%	90%

   a. Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the Always-Taken predictor? Assume that branch outcomes are determined in the **EX** stage, that there are no data hazards, and that no delay slots are used.
   b. Repeat 3.a for the "Always-Not-Taken" predictor.
   c. Repeat 3.a for the 2-Bit predictor.
   d. With the 2-Bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaces a branch instruction with an ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.
   e. With the 2-Bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.
   f. Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-Bit predictor on the remaining 20% of the branch instructions?