CS161 - Homework 3
DUE 11:59 PM Monday, May 27, 2019 (Please Submit to iLearn)!

1. Consider the following loop.

```
loop:       lw   r1, 0(r1)
            and  r1, r1, r2
            lw   r1, 0(r1)
            lw   r1, 0(r1)
            beq  r1, r0, loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards, we can run `lw` right after `beq`), branches resolve in the MEM stage, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

1.1 Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration, but also the tail end of the second iteration).

1.2 How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

2. This exercise is intended to help you understand the relationship between control hazards and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
                lw   r2, 0(r1)
label1:         beq  r2, r0, label2    # not taken once, then taken
                lw   r3, 0(r2)
                beq  r3, r0, label1    # taken
                add  r1, r3, r1
label2:         sw   r1, 0(r2)
```

2.1 Draw the pipeline execution diagram for this code, assuming branches execute in the ID stage (means resolve in the EX stage). Assume there is a branch target buffer so that we know the branch target address during the fetch stage.

2.2 Draw the pipeline execution diagram for this code, assuming that branches execute in the EX stage (means resolve in the MEM stage). Assume there is NO branch target buffer.

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%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

3.1 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.

3.2 Repeat 3.1 for the "always-not-taken" predictor.

3.3 Repeat 3.1 for the 2-bit predictor.

4. This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: `T, NT, T, T, NT`

4.1 What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

4.2 What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the strongly predict not taken?

4.3 What is the accuracy of the two-bit predictor if this pattern is repeated forever?