Q1  Scheduling branch delay slots can improve performance. Assume an instruction execution pipeline has a single branch delay slot, while the branch outcome is determined in the second stage.

(Part A) For a delayed branch instruction, what is the penalty for each branch delay slot scheduling scheme (a total number 3 scheduling schemes have been listed in your textbook) if the branch is taken and if it is not taken, respectively? What condition, if any, must be satisfied to ensure correct execution in each case?

(Part B) A `cancel-if-not-taken` branch instruction does not execute the instruction in the delay slot if the branch is not taken. Thus, a compiler need not be as conservative when filling the delay slot.

Using this type of branch, for each branch delay slot scheduling scheme, what is the penalty if the branch is taken and if it is not taken, respectively? What condition, if any, must be satisfied to ensure correct execution in each case?

(Part C) Assume that an instruction set has a delayed branch and a `cancel-if-not-taken` branch. For each branch delay slot scheduling scheme, which of the two branch instruction is superior? Please clearly state your reasoning.

Q2  This question concerns the latency of memory accesses in the classical 5-stage MIPS pipeline described in your textbook. One important design concept of pipeline architectures is the balance of pipe stages, as the cycle time is limited by the longest pipeline stage. Because the accesses to the instruction and data memory typically require more time than the function of `decode`, `execute`, and `write back`, your design team is trading off between the following two pipeline implementations:

1. A 5-stage pipeline `IF ID EXE MEM WB` with a cycle time of 1.5ns.
2. A 7-stage pipeline `IF1 IF2 ID EXE MEM1 MEM2 WB` with a cycle time of 0.9ns.
(Part A) Assume both pipeline implementations have full data forwarding. For the following pairs of instructions with true data dependences (either through a register or a memory location), what is the number of stalls needed to be inserted in between (assume initially each pair of instruction is scheduled back to back)? Please analyze your answers for both the 5-stage and the 7-stage pipeline.

- Arithmetic ⇒ arithmetic
- Load ⇒ arithmetic
- Arithmetic ⇒ store
- Store ⇒ load
- Load ⇒ load

(Part B) Your design team has decided to go with the 7-stage pipeline implementation. However, in order to save hardware, the team also decided to cut down on additional forwarding paths. Consequently, only the following forwarding paths exist in the 7-stage pipeline:

- EXE/MEM1 ⇒ ID/EXE
- MEM2/WB ⇒ ID/EXE

The following loop is going to be executed on this 7-stage pipeline. Assume the base addresses for X[ ] and Y[ ] are stored in R11, R12, respectively. Furthermore, each array element takes 4 bytes in storage, and R0 is always zero.

```
ADDI R10, R0, #100
L:  LD R1, 0(R11)  //X[i]
    LD R2, 0(R12)  //Y[i]
    ADD R3, R1, R1 //2X[i]
    ADD R4, R3, R1 //3X[i]
    ADD R5, R2, R3 //2X[i]+Y[i]
    SD R4, 0(R11)
    SD R5, 0(R12)
    ADDI R11, R11, #4
    ADDI R12, R12, #4
    SUBI R10, R10, #1
    BNEZ R10, L
```

Assume the loop is composed of 100 iterations. The loop branch, which is resolved in the EXE stage, is predicted to be always taken. Furthermore, there are no structure hazards in the pipeline. For the code presented above, you are asked to identify the position of all stalls, the reason for each stall (i.e., the dependent instruction that causes the stall), as well as the steady-state CPI of the code.

(Part C) Reschedule the code in (Part B) to minimize the total number of stalls. Write your optimized code below and specify the position of all stalls as well as the reason for each stall. Compared with the original code, how much speedup does your rescheduled code achieve? (Assume the branch is predicted to be always taken.)
Q3 An important issue in current highly accurate branch predictors is that most of the predictors suffer from non-trivial amounts of latency, which degrades the performance enhancement attained by high accuracy. This question asks you to explore the accuracy-latency trade-off for the most advanced branch prediction schemes.

Consider a 6 stage pipeline, **IF1 IF2 ID EXE MEM WB**, in which the instruction fetch is split into two stages, IF1 and IF2. An aggressive branch predictor is accessed during the IF1 stage. However, the predictor takes 1 cycle to make a decision, that is, the prediction is available at the end of the **IF2** stage. Therefore, the fetch continues from the **fall through path** until the ID stage. Then the processor will choose the path predicted by the branch predictor. During the **EXE** stage, the branch condition is definitively resolved.

(Part A) Assume that **all** the branch instructions encountered are guaranteed to be found in the Branch Target Buffer (BTB), which is accessed in the **IF1** stage. The prediction accuracy of the aggressive branch predictor is 90%, and 60% of all the branches in the benchmark are **taken**. Please compute the average number of stall cycles per branch.

(Part B) In your design team, one engineer suggests that before the prediction of the aggressive branch predictor is available, instead of just fetching from the fall through path, a simple predictor with reduced accuracy but less latency can be used to generate a prediction. Therefore, he proposes to use a hybrid prediction scheme: for each branch, the prediction is first given by the simple predictor which can generate a result in the **IF1** stage. Then the subsequent instructions following the path of this branch are fetched in the next cycle. Two cycles later, in the **ID** stage, a new prediction is also given by the aggressive predictor for the **same** branch. If this new prediction disagrees with the prediction given by the simple predictor, the path following the new prediction is chosen and the previous fetched path is canceled.

Assume the simple predictor has a prediction accuracy of 80%, while the accuracy of the aggressive predictor is still 90%. Please compute the average number of stall cycles per branch.

(Part C) Assume the prediction accuracy of the simple predictor is **x**, while the accuracy of the aggressive predictor is 92%. What should the value of **x** be, in order for the hybrid predictor to be superior to the simple predictor? For partial credit, you may want to use two expressions of **x** to represent the average number of stall cycles per branch for the simple predictor and for the hybrid predictor, respectively.