Q1 Suppose we wanted to add the auto-increment addressing mode to the MIPS ISA – e.g., lw R1, 1000(R2++). This saves an instruction every time if a load is followed by an increment of the address register. On the other hand, to implement this function, we have to increase cycle time by 5%.

Assume that the CPI would not be affected by this optimization. Meanwhile, assume that 20% of our instructions are loads, while we can apply this change to $x\%$ of all loads. What should the value of $x$ be, in order for this optimization to be helpful in improving performance?

Q2 A company has a floating-point benchmark that is considered representative of its typical application. This benchmark is used to evaluate an embedded processor that does not have a floating-point unit and must emulate each floating-point instruction by a sequence of integer instructions. On the other hand, a compatible coprocessor, which executes each floating-point instruction in hardware (i.e. no emulation is necessary), is offered to boost performance.

The following symbols are used to answer this problem:

$I$ Number of integer instructions executed on that benchmark.
$F$ Number of floating-point instructions executed on that benchmark.
$Y$ Average number of integer instructions used to emulate one floating-point instruction.
$W$ Time to execute the benchmark on the embedded processor alone.
$B$ Time to execute the benchmark on the processor/coprocessor combination.

(Part A) Given the symbols above, write an equation for the MIPS (Millions of Instructions Per Second) rating of each configuration.

(Part B) Assume $F = 9 \times 10^6$, $Y = 30$, and $W = 3$ seconds. For the configuration without the coprocessor, we measure that the processor is rated at 110 MIPS. For the processor/coprocessor combination configuration, we measure that the MIPS rating is 60. What are the values of $I$ and $B$?
Q3 This question concerns the pipeline implementation of a Reg-Mem ISA. Recall that a Reg-Mem ISA allows one of the operands of ALU instructions to come from the data memory. Therefore, a simple “add” instruction can have the following possible formats:

\[
\begin{align*}
\text{add } R1, R2, R3 \\
\text{add } R1, R2, (R3) \\
\text{add } R1, R2, 100 \\
\text{add } R1, R2, (100)
\end{align*}
\]

A simple way to implement this Reg-Mem ISA is to simply reorder the 5 stages of the traditional MIPS pipeline. Now the order of the modified 5 stages are IF ID MEM EXE WB, yet the hardware resources of each stage in this Reg-Mem pipeline are identical to the stage with the same name in MIPS pipeline.

(Part A) Because of the limitation of hardware resources in this Reg-Mem ISA, the data transfer instructions (i.e. loads and stores) can only have two possible addressing modes: register indirect addressing and absolute (direct) addressing. Compared with the MIPS pipeline, both the change of ISA (from Reg-Reg to Reg-Mem) and the change of addressing modes will have an influence in instruction counts. For different classes of instructions (i.e. load, store, ALU, branch), please compare this Reg-Mem pipeline to the MIPS pipeline in terms of instruction counts and explain the influence this Reg-Mem pipeline will have.

(Part B) Please identify the types of the forwarding paths, i.e., the starting and the ending stages of each path, that need to be introduced to this pipeline so as to minimize the performance penalty caused by data hazards. For each type of forwarding path that you think is necessary, please also provide a pair of dependent instructions that would benefit from the forwarding path.

(Part C) Assuming that all of the required forwarding paths have been added to the pipeline, for each of the code fragments listed below, please identify the number of pipeline stalls, and more importantly the position of each stall. Please show your answers by drawing pipeline diagrams using the style shown in Figure A.33 in H&P.

\[
\begin{align*}
\text{ld } R2, 0(R10) \\
\text{add } R1, R2, (R3) \\
\text{sub } R4, R5, (R1) \\
\text{st } R4, 0(R10)
\end{align*}
\]

\[
\begin{align*}
\text{add } R10, R2, (R3) \\
\text{ld } R1, 0(R10) \\
\text{sub } R4, R5, (R1) \\
\text{and } R6, R4, R6
\end{align*}
\]