Problem 1: [25 pts] Appearing below are what are supposed to be pipeline execution diagrams (PEDs) of code fragments executing on the illustrated implementation. The PEDs are incorrect.

(a) Correct the PEDs.

Correct the PED below.

\[
\text{add } r1, r2, r3 \quad \text{IF ID EX ME WB} \\
\text{lwr3, 0(r1)} \quad \text{IF ID } \rightarrow \text{ EX ME WB}
\]

Correct the PED below.

\[
\text{lwr3, 0(r1)} \quad \text{IF ID EX ME WB} \\
\text{add } r4, r3, r5 \quad \text{IF ID } \rightarrow \text{ EX ME WB} \\
\text{sub } r6, r7, r8 \quad \text{IF ID EX ME WB}
\]

Correct the PED below.

\[
\text{# Cycle} \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \\
\text{beq } r1, r1 \quad \text{TARG} \quad \text{IF ID EX ME WB} \quad \text{# Branch is taken.} \\
\text{xor } r5, r6, r7 \quad \text{IF IDx} \\
\text{add } r8, r9, r10 \quad \text{IFFx} \\
\text{TARG: } \\
\text{sub } r2, r3, r4 \quad \text{IF ID EX ME WB} \\
\text{# Cycle} \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\]
(b) Appearing below are more PEDs which are not correct for the illustrated implementation. This time modify the implementation so that the executions are correct. Only make necessary changes.

- Delete a bypass path by showing an \( \times \) at the mux input where it ends.
- Do not delete or add more hardware than is necessary.

Modify the implementation so that the execution below is correct.

\[
\text{add } r1, r2, r3 \quad \text{IF ID EX ME WB}
\]
\[
\text{sub } r3, r1, r5 \quad \text{IF ID ---\rightarrow EX ME WB}
\]

Modify the implementation so that the execution below is correct.

\[
\text{lw } r1, 0(r2) \quad \text{IF ID EX ME WB}
\]
\[
\text{sw } r1, 0(r3) \quad \text{IF ID EX ME WB}
\]
Problem 2: [25 pts] The implementation below is based on the solution to Homework 2 Problem 2 in which a bypass was added for \texttt{bltz} instructions.

\textit{Use Next Page for solution.}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{Diagram of the processor pipeline stages.}
\end{figure}

\textit{Use Next Page for solution.}

\(a\) The implementation can only bypass values in \texttt{EX} to a \texttt{bltz}. Modify the implementation on the next page so that values can be bypassed from both \texttt{EX} and \texttt{ME}. With these changes the two fragments below should run without a stall and of course bypass the correct value.

\begin{verbatim}
add r1, r2, r3
sub r4, r5, r6
bltz r1, TARG
add r1, r2, r3
sub r1, r1, r6
bltz r1, TARG
\end{verbatim}

\(b\) A bypass from \texttt{EX} isn’t possible for the code fragment below, and a bypass from \texttt{ME} is problematic too. On the next page add logic to generate a stall signal for these situations (load/\texttt{bltz} dependencies) and connect it to the word \texttt{STALL} in the upper-right of the diagram. Notice that there is an \texttt{is Load} logic block in \texttt{ID}.

\begin{verbatim}
lw r1, 0(r2)
bltz r1, TARG
\end{verbatim}

\(c\) Explain why it would not be a good idea to bypass the load value to the \texttt{bltz} when the load is in \texttt{ME}.

\begin{itemize}
\item \textbf{Bypassing load from ME not a good idea because:}
\end{itemize}
Problem 2, continued:

- Modify implementation so `bltz` can bypass from EX and ME.
- Logic to generate stall signal for `bltz` dependent on load.
- Answer part c.
Problem 3: [12 pts] Answer each question below.

(a) Each code fragment below writes register $f30$ with the sum $f2 + 4720$.

# Plan A

```assembly
addi $t0, $0, 4720
mtc1 $t0, $f17
cvt.s.w $f16, $f17
add.s $f30, $f2, $f16
```

# Plan B

```assembly
lui $t0, 0x4593
ori $t0, $t0, 0x8000
mtc1 $t0, $f16
add.s $f30, $f2, $f16
```

☐ What is the difference between `mtc1` and `cvt`?

☐ Why doesn’t Plan B need a `cvt`?

(b) All MIPS integer instructions have their source register numbers in the `rs` and, if needed, `rt` fields. But the destination register number can be found in either the `rt` or `rd` fields.

☐ How does limiting integer sources to `rs` and `rt` reduce cost and improve performance?

☐ Why isn’t performance hurt by having the destination in either `rt` or `rd`?
Problem 4: [28 pts] Answer each question below.

(a) The statement below omits an important reason why customers can be kept by companies that manage an ISA and implementation as two different things.

*By separating the ISA from the implementation we can keep our customers by offering them a faster implementation when they are ready to buy a new system.*

☐ What is the important reason that has been omitted?

(b) To use profiling to improve performance a program is compiled twice.

☐ What is done between the first and second compilation?

☐ Why does the program need to be compiled a second time?

☐ Suppose that taken branches have a penalty. Show how profiling helps.
Problem 4, continued:

(c) Consider an instruction such as \texttt{add (r1), r2, 4(r3)}. What about it makes it unsuitable for a RISC ISA? Explain why it would be difficult to implement in our pipelined design.

- \texttt{add (r1), r2, 4(r3)} unsuitable for RISC because:

- It would be difficult to implement because:

(d) When we compared the un-optimized and optimized versions of the \( \pi \) program we found that the optimized version had many fewer load and store instructions. Why?

- The optimized \( \pi \) program had fewer loads and stores because:

(e) A tester preparing a run of the SPECcpu suite is responsible for compiling the benchmarks. Why does that make SPECcpu results interesting to computer engineers?

- Tester compilation makes SPECcpu interesting to computer engineers because:
Problem 5: [10 pts] Answer the following questions about bypass paths.

(a) Consider the two statements below about bypasses in implementations like our five-stage MIPS running typical programs.

   A: Compiler scheduling makes bypass paths unnecessary.

   Explain why the statement above is wrong.

   B: Bypass paths make compiler scheduling unnecessary.

   Explain why the statement above is wrong.

(b) Consider the two statements again as it applies to our MIPS implementation, but this time running a special set of programs. We plan to design an implementation for this set of programs. For these programs the two statements are true! Note: The original exam did not mention the new implementation, and it had an “or both” option below.

   For such programs should we eliminate bypass paths or compiler scheduling?

   Explain.