Problem 1: Recall that the MIPS-I multiply instruction reads two integer registers and writes the product into registers hi and lo. To use the product the values of lo and hi (if needed) have to be moved to integer registers, done using a move from instruction such as `mflo`. In this problem these instructions will be added to the implementation below.

Consider an integer multiply unit that consists of two stages, Y1 and Y2. The inputs to Y1 are the 32-bit multiplier and multiplicand, and the output of Y2 is the 64-bit product. Unit Y1 has three 32-bit outputs named s0, s1, and s2; unit Y2 has 3 32-bit inputs of the same name. As one would guess, the data from the s0 output of Y1 should be sent to the s0 input to Y2, likewise for s1 and s2.

As with other functional units, such as the ALU, inputs to Y1 and Y2 must be stable near the beginning of the clock cycle and the outputs must be stable near the end of the clock cycle. There is enough time to put a multiplexer before the inputs, or after the outputs (but not both).

Solve the two parts below together. That is, the hardware for part (b) might take advantage of the hardware for part (b) and vice versa.

(a) Add the datapath hardware needed to implement the `mtlo`, `mthi`, `mflo`, and `mfhi` instructions. Both the ALU and the integer multiply unit have an operation to pass either input to its output unchanged. That is, let \( x = a + b \) and \( x = a \& b \), the ALU can also perform a pass-a operation, that is, \( x = a \) and a pass-b operation, \( x = b \). The integer multiply unit also has pass-a and pass-b operations.

- Put the hi and lo registers in the ID stage.
- Do not write the hi and lo registers earlier than the ME stage.
• As always, cost is a criteria.
• Bypass paths will be added in the parts below.

(b) Add the datapath hardware needed to implement the \texttt{mult} instruction. That is, put the Y1 and Y2 units in the appropriate stages, and connect them to the appropriate pipeline latch registers (adding new ones where necessary).
• Don’t add new bypass paths, but take advantage of what is available.

(c) Show the execution of the code below on your hardware so far. That is, your hardware should not have any new bypass paths, but existing bypass paths in the implementation can be used.

\begin{verbatim}
sub r2, r6, r7
mult r1, r2
mflo r3
add r4, r3, r5
\end{verbatim}

(d) Add bypass paths so that the code below (which is the same as in the previous part) can execute without a stall. Assume that an additional multiplexer delay is tolerable.

\begin{verbatim}
sub r2, r6, r7
mult r1, r2
mflo r3
add r4, r3, r5
\end{verbatim}

\textbf{Problem 2}: Continue to consider the implementation of the MIPS-I \texttt{mult} instruction. If MIPS designers thought that an integer multiply unit could be built with two stages they might not have used special registers, \texttt{hi} and \texttt{lo}, for the product.

\begin{enumerate}
\item Show how the pipeline would look if the multiply unit had three stages, Y1, Y2, and Y3. There is no need to add bypass paths for this part.
\item Explain why there is much less of a need for the \texttt{hi} and \texttt{lo} registers with a two-stage multiply unit (the first problem) than with a three-stage unit (this problem).
\end{enumerate}