Your task is to implement double-precision floating-point \textbf{FADD} and \textbf{FMUL} functions in MIPS using only integer instructions and integer registers. You are not supposed to use ANY floating-point instructions or registers for this Lab assignment.

1. Assume that the data section of your code contains two integer arrays \textit{A} and \textit{B}, each which is of size 2 words, and their addresses are given in registers \texttt{a0} and \texttt{a1}. Register \texttt{a0} (or \texttt{a1}) points to the most significant byte of the most significant word of \textit{A} (or \textit{B}). For example, if \textit{A} is the 64-bit vector written from the most significant to the least significant bytes as: \textit{A}_7\textit{A}_6\textit{A}_5\textit{A}_4 \textit{A}_3\textit{A}_2\textit{A}_1\textit{A}_0, then address in register \texttt{a0} is the address of the byte \textit{A}_7.

\begin{tabular}{|c|c|c|c|c|}
  \hline
  \texttt{a0} & \texttt{A}_7 & \texttt{A}_6 & \texttt{A}_5 & \texttt{A}_4 \\
  \hline
  \quad & \texttt{A}_3 & \texttt{A}_2 & \texttt{A}_1 & \texttt{A}_0 \\
  \hline
\end{tabular}

However, you will interpret these 8 bytes as a double-precision floating-point number. The most significant bit of the most significant byte (i.e., \textit{A}_7) contains the sign bit of the floating-point number, while the following bits are the 11-bit (biased) exponent and 52-bit fraction, illustrated as follows:

\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
  \hline
  \multicolumn{1}{|c|}{1} & \multicolumn{11}{|c|}{11} & \multicolumn{1}{|c|}{20} & \multicolumn{1}{|c|}{2} \\
  \hline
  \texttt{a0} \rightarrow & S & E'_{10}E'_9E'_8\cdots E'_{1}E'_0 & F_1 & F_2 & F_3 & \cdots & F_{19} & F_{20} \\
  & F_{21} & F_{22} & F_{23} & F_{24} & \cdots & \cdots & \cdots & F_{51} & F_{52} \\
  \hline
\end{tabular}

In other words \texttt{a0} contains the address of the (most significant) byte which contains the following 8 bits: \textit{S}E'_{10}E'_9E'_8\cdots E'_{1}E'_0. The next 3 bytes (24 bits) in the top word contain the bits \textit{E}'_3E'_2E'_1E'_0 \textit{F}_1\textit{F}_2\textit{F}_3 \cdots \textit{F}_{20}. The next word (addressed by \texttt{a0}+4) contains the remaining 32 bits of the fraction: \textit{F}_{21}\textit{F}_{22}\textit{F}_{23} \cdots \textit{F}_{51}\textit{F}_{52}. Therefore, the value of the double-precision floating point number is

\[ (-1)^S \times 2^{E'_{10} - 1023} \times (1.F_{1}F_{2} \cdots F_{52}) \]

2. Perform the double-precision floating-point addition operation \textit{A} + \textit{B} assuming \textit{A} and \textit{B} are IEEE double-precision floating-point numbers. Place the resulting floating-point number as a 2-word binary vector in the memory location \textit{C} whose address is given in \texttt{a2}, according to the representation above.
3. Perform the double-precision floating-point multiplication operation $A \times B$ assuming $A$ and $B$ are IEEE 754 double-precision floating-point numbers. Place the resulting floating-point number as a 2-word binary vector in the memory location $D$ whose address is given in $a3$, according to the representation above.

4. You are expected to implement Rounding with $R$ and $S$ bits for both addition and multiplication operations. The usage of the $S$ bit implies that the fractions that are right in the middle are rounded down (implying $S = 0$) and those that are above the middle are rounded up (implying $S = 1$). For example, 23.5000 is rounded down to 23.0 (because it means $S$ was zero), while 23.5001 is rounded up to 24.0 (because it means $S$ was one).

**Deliverables**
Submit both your .asm file and your lab report into the dropbox. Included in your lab report should be your name, perm, issues with your code, and screenshots of the simulator states (data section of SPIM before/after execution).

**Grading**
Your code will be run on a number of test cases using SPIM. Your grade will be based on the number of tests passed/failed.