1. Assuming 4-bit numbers, perform the following additions:
   a) for unsigned numbers: \( 4_{10} + 6_{10} \) \( 9_{10} + 10_{10} \)

   b) for signed numbers: \( 4_{10} + (-6_{10}) \) \( (-4_{10}) + (-6_{10}) \) \( 4_{10} + 6_{10} \)

2. For 4-bit unsigned numbers, when do we have overflow and get the wrong result during addition? (Hint: think about the carry bits into and/or out of the most-significant bit)

3. a) For 4-bit signed numbers, complete the following table about signed overflow:

<table>
<thead>
<tr>
<th>Sign of Operands for addition</th>
<th>Expected Sign of Result</th>
<th>Wrong Sign of Result (indicates overflow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand 1</td>
<td>Operand 2</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>These two rows cannot cause signed overflow in addition</td>
</tr>
<tr>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

b) For 4-bit signed numbers, when do we have overflow and get the wrong result during addition? (Hint: think about the carry bits into and/or out of the most-significant bit)
IEEE 754 Standard Floating Point Representation

<table>
<thead>
<tr>
<th>Single Precision</th>
<th>Double Precision</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>Mantissa</td>
<td>Exponent</td>
</tr>
<tr>
<td>1-254</td>
<td>any value</td>
<td>1-2046</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>2,047</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td>2,047</td>
</tr>
</tbody>
</table>

4) Convert the value 23.625_{10} to its binary representation.

\[ \begin{array}{cccccccc}
64 & 32 & 16 & 8 & 4 & 2 & 1 & .5 & .25 & .125 & .0625 \\
\end{array} \]

5) Normalize the above value so that the most significant 1 is immediately to the left of the radix point. Include the corresponding exponent value to indicate the motion of the radix point.

\[ \begin{array}{cccc}
1. & \text{ } & \text{ } & \times 2 \\
\end{array} \]

6) Write the corresponding 32-bit IEEE 754 floating point representation for 23.625_{10}.
7) Write the corresponding 64-bit IEEE 754 floating point representation for 23.625_{10}.

8) What would be the smallest positive normalized 32-bit IEEE 754 floating point value?

9) The smallest positive denormalized 32-bit IEEE 754 floating point value has representation of

\[
\begin{array}{|c|c|c|}
\hline
\text{Sign bit} & \text{Exponent (bias 127)} & \text{23-bit Mantissa} \\
\hline
0 & + & 000000000000000000000000001 \\
1 & - & 0000000000000000000000000101 \\
\hline
\end{array}
\]

What value would it represent?

\[2 \times 2^{x} \]

10) What would be the representation for the largest positive denormalized 32-bit IEEE 754 floating point?

\[
\begin{array}{|c|c|c|}
\hline
\text{Sign bit} & \text{Exponent (bias 127)} & \text{23-bit Mantissa (for denormalized values, leading 0 not stored)} \\
\hline
0 & + & 00000000000000000000000000001 \\
1 & - & 00000000000000000000000000000 \\
\hline
\end{array}
\]
11) How would you add two IEEE 754 floating point numbers?

12) How would you multiply two IEEE 754 floating point numbers?

13) Consider adding 1.011 x 2^{40} and 1.01 x 2^{5}.
   a) How many places does the second number's mantissa get shifted?

   b) After we add these two numbers and store the results back into a 32-bit IEEE 754 value, what would be the result?