1. Perform the following calculations (unsigned and infinite number of bits/digits):

\[
\begin{align*}
11 \quad (a) \quad 11 & \quad 01_2 \quad \bar{1} \quad \overline{1} \quad 02_2 \\
111101102 & \quad x0010002 \\
+01100011_2 & \quad 01111012 \\
\hline
101011001 & \quad 0010111
\end{align*}
\]

2. Represent the following decimal numbers in binary using 16-bit signed magnitude, one's complement, and two's complement:

<table>
<thead>
<tr>
<th>decimal #</th>
<th>signed magnitude 16-bits</th>
<th>one's complement 16-bits</th>
<th>two's complement 16-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>92_{10}</td>
<td>0000000001 01100</td>
<td>0000000001 011100</td>
<td>0000000001 011100</td>
</tr>
<tr>
<td>-40_{10}</td>
<td>10000000000 01100</td>
<td>1111111110 011110</td>
<td>1111111110 011100</td>
</tr>
</tbody>
</table>

3. Using 16-bits what is the range of values for each of the following representations: (Leave your answer as an equation contain powers of 2.)

a) unsigned integers:

\[
0 - \left(2^{15} - 1\right)
\]

b) signed integers using two's complement:

\[
-2^{15} + 0 + \left(2^{15} - 1\right)
\]

4. What decimal (base 10) value is represented by the 32-bit signed, two’s complement value FFFF A7E9_{16}? (The 32-bits two’s complement value is shown as a hexadecimal so I did not need to write a 32-bit binary number.)

\[
\begin{align*}
\text{Value:} & \quad F \quad F \quad F \quad F \quad A \quad 7 \quad E \quad 9 \\
\text{Binary:} & \quad 1111 \quad 1111 \quad 1111 \quad 1010 \quad 0111 \quad 1110 \quad 1001 \\
\text{Signed:} & \quad 0 \quad 0 \quad 10110000010110 \quad \text{flip bits} \\
\text{Add 1:} & \quad 0 \quad 01011000010111 \\
\text{Value:} & \quad -(16384 + 4096 + 2048 + 16 + 4 + 2 + 1) = -2255_{10}
\end{align*}
\]

5. Use Booth's algorithm to calculate the 14-bit product of 11001011 x 11011012.

\[
\begin{align*}
& 11001011 \quad \text{(27)} \\
& 0010010 \quad \text{(10)} \\
& 00111011 = +27 \\
& 00100111 = \pm 19 \\
& -27 \times 19 = 513 \quad \text{desired answer}
\end{align*}
\]
Q5.

**Multiplicand, M**

\[
\begin{array}{c}
1100101 \\
0011010
\end{array}
\]

(e) Flip bits

\[+1\] add 1

- **M**

\[
0011011
\]

**Product**

\[
00000000
\]

**Multiplier**

\[
1101101
\]

**Previous bit**

\[
0
\]

\[
0
\]

\[
1 \text{ start run}
\]

\[
\rightarrow
\]

\[
\text{start run}
\]

\[
\text{end run}
\]

\[
0
\]

\[
0
\]

\[
\text{middle of run}
\]

\[
\text{end run}
\]

\[
0
\]

\[
\text{middle of run}
\]

\[
0
\]

\[
\text{start run}
\]

\[
0
\]

\[
\text{middle of run}
\]

\[
0
\]

\[
\text{start run}
\]

\[
0
\]

\[
\text{middle of run}
\]

\[
0
\]

\[
+513
\]

\[
0000100
\]

\[
00000001
\]

512 256 128 64 32 16 8 4 2 1
6. Convert -179.375_{10} to its 32-bit IEEE-754 floating point representation.

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

\[
1\text{.}011001101\times2^{+7} + 1.127 = 134
\]

7. Suppose A, B and C are normalized 32-bit IEEE 754 floating point variables with A having a real value of 1.1011_{2} \times 2^{60} and B having a real value of 1.011_{2} \times 2^{20}. After the high-level language assignment statement "C = A+B", why is C's value equal to A's value and not the \textit{mathematically} correct sum?

(A's normalized 32-bit IEEE 754 representation would be: 0 10111011 101100000000000000000000)

(B's normalized 32-bit IEEE 754 representation would be: 0 10010011 010000000000000000000000)

To align the radix/decimal point for addition, we need to shift B's value 60 - 20 = 40 places:

\[
A: \begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
\end{array} \times 2^{60}
\]

\[
B: \begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
\end{array} \times 2^{60}
\]

\[
\text{Correct sum:} \begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array} \times 2^{60}
\]

Since only 23 bits of mantissa can be saved, B's contribution to the sum is lost.

8. For the same values of A and B in question 7, would the high-level language assignment statement "C = A+B" assign C the \textit{mathematically} correct sum if A, B and C were used the 64-bit IEEE 754 floating point format? (explain your answer)

Yes, the 64-bit format stores a 52-bit mantissa which can hold all the bits of the correct sum.