1. Exercise 3.2 but convert to single precision floating point number:

\(-2047\) \(_{\text{ten}} = -0111 1111 1111 \two_{\text{two}} = -1.11 1111 1111 \two_{\text{two}} \times 2^{10}

Sign = 1
Exponent = 10 + 127 = 137 \(_{\text{ten}} = 1000 1001 \two_{\text{two}}
Fraction = .11 1111 1111 \two_{\text{two}}
1 10001001 11111111110000000000000

2. Exercise 3.5 but convert from single precision floating point number:

Exponent = 1111 1111 \two_{\text{two}} = \text{MAX}

Fraction \neq 0
So NaN

3. Exercise 3.30

Bit pattern = 1010 1101 0000 0000 0000 0000 0000 0010

a. Two’s complement integer = -0101 0010 1110 1111 1111 1111 1111 1110 \two_{\text{two}}
   = -1 391 460 350 \text{ten}

b. Unsigned integer = 2 903 506 946 \text{ten}

c. Single precision floating point number:
   Sign = 1 = negative
   Exponent = 0101 1010 \rightarrow 90 - 127 = -37 \text{ten}
   Significand = 1.001 0000 0000 0000 0000 0010 = 9437186 \times 2^{-23}
   Hence the decimal number is \rightarrow -9437186 \times 2^{-60} = -8185454.051 \times 10^{-18}
   = -8.185454051 \times 10^{-12}

d. Opcode = 101011 = sw ; 01000 = $t0; 10000 = $s0; 0000 0000 0000 0010 = 2 \text{ten}
   Hence the instruction is sw $s0, 2 ($t0)

4. Exercise 3.37

\(20\) \(_{\text{ten}} = 0001 0100 \two_{\text{two}} = +1.01 \two_{\text{two}} \times 2^{4}

Single precision bit pattern:
Sign = 0
Exponent = $4 + 127 = 131_{\text{ten}} = 1000 0011$
Fraction = .010 0000 0000 0000 0000 0000
Hence 0 1000 0011 010 0000 0000 0000 0000 0000
Double precision bit pattern:
Sign = 0

Exponent = $4 + 1023 = 1027_{\text{ten}} = 100 0000 0011$
Fraction = .010000…0
Hence 0 100 0000 0011 0100 0000 0000 0000 0000 0000 0000 0000 0000

5. Exercise 3.39

0.1 $\times 2 = 0.2$ | 0 # Generate 0 and continue
0.2 $\times 2 = 0.4$ | 0 # Generate 0 and continue
0.4 $\times 2 = 0.8$ | 0 # Generate 0 and continue
0.8 $\times 2 = 1.6$ | 1 # Generate 1 and continue with the rest
0.6 $\times 2 = 1.2$ | 1 # Generate 1 and continue with the rest
0.2 $\times 2 = 0.4$ | 0 # Generate 0 and continue
0.4 $\times 2 = 0.8$ | 0 # Generate 0 and continue
0.8 $\times 2 = 1.6$ | 1 # Generate 1 and continue with the rest
0.6 $\times 2 = 1.2$ | 1 # Generate 1 and continue with the rest
...

The reason why the process seems to continue endlessly is that it does. The number 1/10, which makes a perfectly reasonable decimal fraction, is a repeating fraction in binary, just as the fraction 1/3 is a repeating fraction in decimal. (It repeats in binary as well.) We cannot represent this exactly as a floating point number. The closest we can come in 23 bits is 00 1100 1100 1100 1100 1100

0.1 = 0.0001100110011001100110011… = 1.1001100110011001100110011…$^{\times 2}$

(Note: “0011” pattern repeats).
Bit pattern:
Sign = 0, Fraction = .1001100110011001100110011…
Single exponent = $-4 + 127 = 123 = 0111 1011$
Double exponent = $-4 + 1023 = 1019 = 0111 1111 1011$

Single precision bit pattern:
0 0111 1011 100 1100 1100 1100 1100 1100 trunc
0 0111 1011 100 1100 1100 1100 1100 1100 round
Double precision bit pattern:
0 011 1111 1011 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 trunc
0 011 1111 1011 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1011 round

6. Exercise 3.42

a. $X + Y$
$X = 0 10001101 101 1000 0000 0000 0000 0000$
\( \text{exponent is } 141 - 127 = 14 \text{, sign is } + \text{ (positive), significand is } 1.1011_{\text{two}} \)

\[
Y = 1\ 01111101\ 110\ 0000\ 0000\ 0000\ 0000\ 0000
\]

\( \text{exponent is } 125 - 127 = -2 \text{, sign is } - \text{ (negative), significand is } 1.11_{\text{two}} \)

Hence \( +1.1011 \times 2^{14} + -1.11 \times 2^{-2} \)

Sum of significands (unsigned numbers):

\[
-1.1011\ 0000\ 0000\ 0000\ 0000\ 0000

-0.0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000

-1.1010\ 1111\ 1111\ 1110\ 0100\ 0000
\]

no need to normalize (already in normalized form).

Hence result is \( 0\ 1000\ 1101\ 101\ 0111\ 1111\ 1111\ 0010\ 0000 \)

b. Please solve it yourself.
7. Produce a truth table for a logic design of an overflow detection circuit in an ALU.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Sign (Operand A)</th>
<th>Sign (Operand B)</th>
<th>Sign (Result)</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>A + B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A + B</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A - B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A - B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>