CSE 351 Two's Complement/Floating-Point Practice Worksheet

1 Exercises

1.1 Decimal to Two's Complement Binary

Convert the following decimal numbers to 8-bit two's complement binary. Record the result in binary and hex.

1.1.1 -39

```
Convert to binary: 0b100111
Pad to 7 bits:     0b00100111
Invert the bits:   0b11011000
Add 1:                    1
------------------------------
            0xD9 = 0b11011001
```

1.1.2 127

```
Convert to binary: 0b1111111
Pad to 7 bits:     0b01111111
------------------------------
            0x7F = 0b01111111
```

1.1.3 -69

```
Convert to binary: 0b1000101
Pad to 7 bits:     0b01000101
Invert the bits:   0b10111010
Add 1:                    1
------------------------------
            0xBB = 0b10111011
```

1.1.4 104

```
Convert to binary: 0b1101000
Pad to 7 bits:     0b01101000
------------------------------
            0x68 = 0b01101000
```

1.2 Two's Complement Math

Compute the following 8-bit two's complement sums. Note if the solution has carryout, overflow, or if the sum is correct.

1.2.1 -39 + 92

```
1  1     1  1              (carry)
   1  1  0  1  1  0  0  1  (-39)
+  0  1  0  1  1  1  0  0  (92)
--------------------------
   0  0  1  1  0  1  0  1
Carryout, no overflow, sum is correct
```

1.2.2 127 + 1

```
   1  1  1  1  1  1  1     (carry)
   0  1  1  1  1  1  1  1  (127)
+  0  0  0  0  0  0  0  1  (1)
--------------------------
   1  0  0  0  0  0  0  0
No carryout, overflow, sum is incorrect
```

1.2.3 104 + 45

```
   1  1     1              (carry)
   0  1  1  0  1  0  0  0  (104)
+  0  0  1  0  1  1  0  1  (45)
--------------------------
   1  0  0  1  0  1  0  1
No carryout, overflow, sum is incorrect
```

1.2.4 -103 + -69 = -172

```
1     1  1  1     1  1     (carry)
   1  0  0  1  1  0  0  1  (-103)
+  1  0  1  1  1  0  1  1  (-69)
--------------------------
   0  1  0  1  0  1  0  0
Carryout, overflow, sum is incorrect
```

1.3 Decimal to Floating-Point Binary

Convert the following decimal numbers to 32-bit floating-point binary numbers. Record the result in binary and hex.

1.3.1 1313.3125

```
Sign bit: 0
1313 = 0b10100100001
0.3125 = 0b0.0101
1313.3125 = 0b10100100001.0101
Normalize: 1.01001000010101 * 2^10
Mantissa: 01001000010101000000000
Exponent: 10 + 127 = 137 = 10001001
Ans: 0 10001001 01001000010101000000000 = 0x44A42A00
```

1.3.2 0.1015625

```
Sign bit: 0
0.1015625 = 0b0.0001101
Normalize: 1.101 * 2^-4
Mantissa: 10100000000000000000000
Exponent: -4 + 127 = 123 = 01111011
Ans: 0 01111011 10100000000000000000000 = 0x3DD00000
```

1.4 Floating-point Math

Compute the following floating-point sum:
`1313.3125 + 0.1015625`

```
1313.3125 = 1.01001000010101 * 2^10
0.1015625 = 1.101 * 2^-4

normalize to same exponent:
101001000010101.0     * 2^-4
              1.1010  * 2^-4
-----------------------------
101001000010110.1010  * 2^-4
renormalized: 1.010010000101101010 * 2^10
new mantissa: 01001000010110101
exponent: 10 + 127 = 137 = 10001001
Ans: 0 10001001 01001000010110101000000 = 0x44A42D40
```

Compute the following floating-point product:
`1313.3125 * 0.1015625`

```
1313.3125 = 1.01001000010101 * 2^10
0.1015625 = 1.101 * 2^-4
New Sign: 0 ^ 0 = 0
Temp Exp: 10 + -4 = 6
New Mant: 1.01001000010101
          *           1.101
          ------------------
         111 1    111111    (carry row)
            101001000010101
           0000000000000000
          10100100001010100
         101001000010101000
         -------------------
        1000010101100010001
Result: 10.00010101100010001
Adjusted Mant: 1.000010101100010001
Adjusted Exp: 7 + 127 = 134 = 10000110
Final Mant: 00001010110001000100000
Ans: 0 10000110 00001010110001000100000
```