3.1

What is 5ED4 - 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

In this case, we can perform the subtraction in the standard way. First see that 5ED4 and 07A4 are

$$\begin{gathered} 5:5 \quad E:15 \quad D:14 \quad 4:4 \\ 0:0 \quad 7:7 \quad A:10 \quad 4:4 \end{gathered}$$

Then simply perform the subtractions

$$5ED4 - 07A4 = (5-0=5)\ (15-7=7)\ (13-10=3)\ (4-4=0) = 5730$$

3.2

What is 5ED4 - 07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

The answer does not change in this case. In sign-magnitude form the first bit indicates the sign (positive/negative), while the remaining bits determine the actual number. This means that for a 16-bit word, there is a possible numerical range from -32767 to 32767 ($\pm 2^{15}$), with 1 bit representing sign and 15 bits representing value. To confirm see the following:

$$\begin{gathered} 5ED4 \\ 5 = 0101 \quad E = 14 = 1110 \quad D = 13 = 1101 \quad 4 = 0100 \\ 5ED4 = 0101\,1110\,1101\,0100 = 24276 \end{gathered}$$

$$\begin{gathered} 07A4 \\ 0 = 0000 \quad 7 = 0111 \quad A = 10 = 1010 \quad 4 = 0100 \\ 07A4 = 0000\,0111\,1010\,0100 = 1956 \end{gathered}$$

$$\begin{gathered} 5ED4 - 07A4 \\ 5ED4 - 07A4 = 0101\,1110\,1101\,0100 - 0000\,0111\,1010\,0100 = 24276 - 1956 \\ = 5730 = 0101\,0111\,0011\,0000 = 22320 \end{gathered}$$

3.3

Convert 5ED4 into a binary number. What makes base 16 (hexadecimal) an attractive numbering system for representing values in computers?

$$5 = 0101 \quad E = 14 = 1110 \quad D = 13 = 1101 \quad 4 = 0100$$
$$5ED4 = 0101\,1110\,1101\,0100$$

Hexadecimal is an attractive numbering system for representing values in computers for a couple reasons. First, it is very simple/convenient to use with binary - every four binary digits can easily be converted to a single hexadecimal digit. Simply evaluate the value of four binary bits and replace it with a hexadecimal character (e.g. 1111 → F). Secondly, using hexadecimal allows for easy 8, 16, or 32-bit representation and is much more compact.

3.4

What is 4365 - 3412 when these values represent unsigned 12-bit octal numbers? The result should be written in octal. Show your work.

Because this is unsigned, this problem can be solved very simply with basic subtraction. One borrow step must be used (borrow from the fourth digit). See the following

$$4365 - 3412 = ((4-1) - 3 = 0)\ ((3+8) - 4 = 7)\ (6 - 1 = 5)\ (5 - 2 = 3) = 0753$$

3.5

What is 4365 - 3412 when these values represent signed 12-bit octal numbers stored in sign-magnitude format? The result should be written in octal. Show your work.

In this case, the values are signed integers (sign-magnitude format). As a result, the octal value 4365 represents a negative number because it has, when converted to binary, a sign bit of 1.

$$4365$$
$$4 = 100 \quad 3 = 011 \quad 6 = 110 \quad 5 = 101$$
$$4365 = 100\,011\,110\,101 = -245$$

$$3412$$
$$3 = 011 \quad 4 = 100 \quad 1 = 001 \quad 2 = 010$$
$$3412 = 011\,100\,001\,010 = 1802$$

$$4365 - 3412$$
$$4365 - 3412 = 100\,011\,110\,101 - 011\,100\,001\,010 = -245 - 1802$$
$$= 7777 = 111\,111\,111\,111 = -2047$$

3.6

Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185 - 122. Is there overflow, underflow, or neither?

Unsigned 8-bit values can range from 0 to 255 ($2^8 - 1$). In unsigned form, 185 is in fact 185, and 122 is 122. Thus the final result is 185 - 122 = 63. There is no overflow or underflow because 63 is within the range of 0 to 255.

3.7

Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185 + 122. Is there overflow, underflow, or neither?

In signed 8-bit sign-magnitude format, 185 and 122 are

$$185 = 1011\,1001 = -0111001 = -57$$
$$122 = 0111\,1010 = +1111010 = 122$$
$$122 - 57 = 65$$

In this case, we can simply subtract the smaller unsigned 7-bit integer from the larger unsigned 7-bit integer (122-57). The result is 65. Because this is within the range of $\pm(2^7 - 1)$, or ± 127, there is no overflow or underflow.

3.8

Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185 - 122. Is there overflow, underflow, or neither?

From Pg. 179 of the text, we see that

$$c - a = c + (-a)$$

Thus, for the substraction of signed integers, we can use the unsigned version of a and c and add in the standard fashion. The final sign of the result is positive if the minuend is greater than the subtrahend (e.g. $1 - (-1) = +2$), negative if the minuend is less than the subtrahend (e.g. $(-1) - 1 = -2$), and zero if the minuend is equal to the subtrahend (e.g. $1 - 1 = 0$). Consider the signed (sign-magnitude format) 8-bit decimal integers 185 and 122.

$$185 = 1011\,1001 = -0111001 = -57$$
$$122 = 0111\,1010 = +1111010 = 122$$

In this case, we can simply add 57 and 122. Since the minuend is less than the subtrahend (-57 < 122), the final sign is negative. Thus the final result is -179. However, this would result in an overflow because the range for signed (sign-magnitude format) 8-bit integers is $\pm(2^7 - 1)$, or ± 127. -179 is outside of this range, hence an overflow.

3.9

Assume 151 and 214 are signed 8-bit decimal integers stored in twos complement format. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.

151 and 214 in signed 8-bit 2's complement form are

$$151 = 1001\,0111 = -1101001 = -105$$
$$214 = 1101\,0111 = -0101001 = -41$$

Considering this, the result of 151 + 214 is

$$-105 + -41 = -146 \xrightarrow{saturation} -128$$

The range of a signed 8-bit integer using 2's complement is $-(2^7) - (2^7 - 1)$, or -128 to +127.

3.10

Assume 151 and 214 are signed 8-bit decimal integers stored in twos complement format. Calculate 151 - 214 using saturating arithmetic. The result should be written in decimal. Show your work.

151 and 214 in signed 8-bit 2's complement form are

$$151 = 1001\,0111 = -1101001 = -105$$
$$214 = 1101\,0111 = -0101001 = -41$$

Considering this, the result of 151 - 214 is

$$-105 - -41 = -105 + 41 = -64$$

No adjustment for saturation is required because -64 is within the range allowed by signed 8-bit integers using 2's complement (-128 to +127).

3.11

Assume 151 and 214 are unsigned 8-bit integers. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.

Because the integers are unsigned, 151 and 214 represent exactly 151 and 214. Thus, the sum of 151 and 214 is 365. However, this is adjusted to 255 for saturation because the range of unsigned 8-bit integers is 0 to 255.

3.32

Calculate (3.984375 x 10^{-1} + 3.4375 x 10^{-1}) + 1.771 x 10^3 by hand, assuming each of the values are stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

$$(0.3984375 + 0.34375) + 1771.0$$

First we need to convert to binary.

$$0.3984375$$
$$0.3984375 * 2 = 0.796875 \rightarrow 0$$
$$0.796875 * 2 = 1.59375 \rightarrow 1$$
$$0.59375 * 2 = 1.1875 \rightarrow 1$$
$$0.1875 * 2 = 0.375 \rightarrow 0$$
$$0.375 * 2 = 0.75 \rightarrow 0$$
$$0.75 * 2 = 1.5 \rightarrow 1$$
$$0.5 * 2 = 1.0 \rightarrow 1$$
$$0.3984375 = 0.0110011 = 1.10011 * 2^{-2}$$

$$0.34375$$
$$0.34375 * 2 = 0.6875 \rightarrow 0$$
$$0.6875 * 2 = 1.375 \rightarrow 1$$
$$0.375 * 2 = 0.75 \rightarrow 0$$
$$0.75 * 2 = 1.5 \rightarrow 1$$
$$0.5 * 2 = 1.0 \rightarrow 1$$
$$0.34375 = 0.01011 = 1.011 * 2^{-2}$$

$$1771.0$$
$$1771.0 = 11011101011.0 = 1.1011101011 * 2^{10}$$

Next, we can convert to 16-bit half precision floating point using a bias of 15.

$$1.10011 * 2^{-2}$$
$$S = 0E = -2 \rightarrow 15 + (-2) = 13 = 01101$$
$$M = 10011$$
$$0.3984375 = 0011011001100000$$

$$1.011 * 2^{-2}$$
$$S = 0E = -2 \rightarrow 15 + (-2) = 13 = 01101$$
$$M = 011$$
$$0.34375 = 0011010110000000$$

$$1.1011101011 * 2^{10}$$
$$S = 0E = 10 \rightarrow 15 + 10 = 25 = 11001$$
$$M = 1011101011$$
$$1771.0 = 0110011011101011$$

Next, we can carry out the addition.

$$1.10011 * 2^{-2} + 1.01100 * 2^{-2} = 10.1111 * 2^{-2} = 1.011111 * 2^{-1}$$
$$1.011111 * 2^{-1} = 0.00000000001011111 * 2^{10}$$
$$0.00000000001011111 * 2^{10} + 1.1011101011000000 * 2^{10} = 1.1011101100011111 * 10^{10}$$

Since the mantissa in 16-bit half precision IEEE754 is only 10 bits, we shorten the result to 10 bits, plus 3 extra for guard, round, and sticky bits.

$$1.1011101100011111 * 10^{10} \rightarrow 1.1011101100011 * 10^{10}$$

Here, the GRS is 011 which indicats to round down if possible. In this case, the last bit in the mantissa is 0, so nothing happens. The final result is given by the following:

$$16bit\, FP = 0110011011101100 = 1772$$

3.33

Calculate 3.984375 x10^{-1} + (3.4375 x 10^{-1} + 1.771 x 10^{3}) by hand, assuming each of the values are stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

We can reuse the conversions from question 3.32 for this problem.

$$0.3984375 + (0.34375 + 1771.0)$$

Perform the addition and cut to 13 bits (10-bit mantissa + 3 extra)...

$$0.000000000001011 * 2^{10} + 1.1011101011000000 * 2^{10} = 1.1011101100010 * 2^{10}$$
$$GRS = 010$$

Because the last bit in the mantissa is zero and GRS = 010, no rounding occurs. Perform the next addition step...

$$1.1011101100 * 2^{10} + 0.00000000000110011 * 2^{10} = 1.1011101100011$$
$$GRS = 011$$

Because the last bit in the mantissa is zero and GRS = 011, no rounding occurs. The final result is given by the following:

$$16bit\, FP = 0110011011101100 = 1772$$

3.34

Based on your answers to 3.32 and 3.33, does (3.984375 x 10^{-1} + 3.4375 x 10^{-1}) + 1.771 x 10^{3} = 3.984375 x 10^{-1} + (3.4375 x 10^{-1} + 1.771 x 10^{3})?

The two statements both result in a 16-bit floating point value of 0110011011101100, which is equal to 1772. Thus, they are indeed equivalent. According to the Wikipedia page on IEEE 754 half-precision binary floating-point format, the precision limitation on decimal values between 1024 and 2048 is fixed interval 2^0, which basically means any real number between 1024 and 2048 is represented as an integer value between 1024 and 2048.