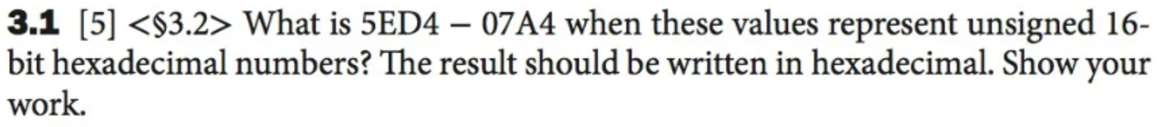
# Written Assignment 3

Name: Aina Azman

Net_ID: 457-464-051



**3.1** [5] <§3.2> What is 5ED4 − 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

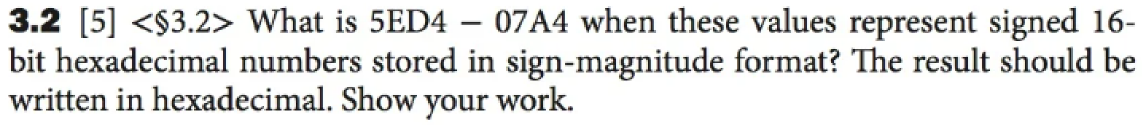
- Below is the conversion from hexadecimal to decimal:

$$5ED4 = 24276$$
$$07A4 = 1956$$

- Then we minus the equation in decimals and convert the decimal into hexadecimal:

$$24276 - 1956 = 22320$$
$$22320 = 0x5730$$

- Hence, $5ED4 - 07A4 = 5730$



**3.2** [5] <§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.

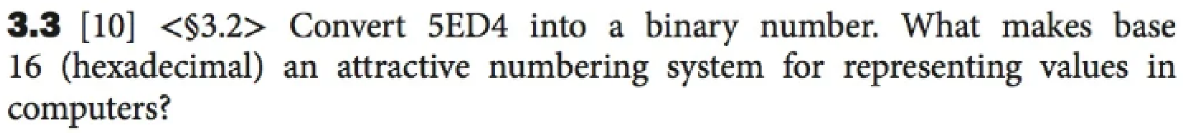
- Below is the conversion from hexadecimal to binary:

$$5ED4 = 0101111011010100$$
$$07A4 = 0000011110100100$$

- Subtracting the equation:

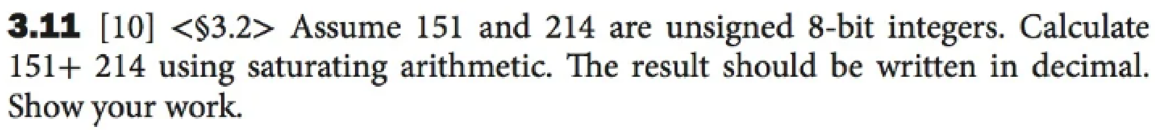
$$\begin{array}{r} 0101\ 1110\ 1101\ 0100 \\ -\ 0000\ 0111\ 1010\ 0100 \\ \hline 0101\ 0111\ 0011\ 0000 \end{array}$$

- Convert to hexadecimal: $0101011100110000 = 5730$
- Hence the answer is 5730.



**3.3** [10] <§3.2> Convert 5ED4 into a binary number. What makes base 16 (hexadecimal) an attractive numbering system for representing values in computers?

- Convert 5ED4 into a binary number: $0101\ 1110\ 1101\ 0100.$
- Base 16 (hexadecimal) is an attractive numbering system for representing values in computers as:
  - It is compatible with binary, which means each hexadecimal digit corresponds to a unique combination of four binary digits, making it easier to convert the two systems.
  - All machines have bits in multiples of 4 or 8, which makes it easy to denote or represent the value in hexadecimal.



**3.11** [10] <§3.2> 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.

- The maximum decimal value of unsigned 8-bit integers is 255 (0b1111 1111).
- Converting decimal to binary:

$$
\begin{array}{r|l l}
2 & 151 & 1 \\
 & 75 & 1 \\
 & 37 & 1 \\
 & 18 & 0 \\
 & 9 & 1 \\
 & 4 & 0 \\
 & 2 & 0 \\
 & & 1
\end{array}
\uparrow
\qquad
151_{10} = 1001\ 0111_2
$$

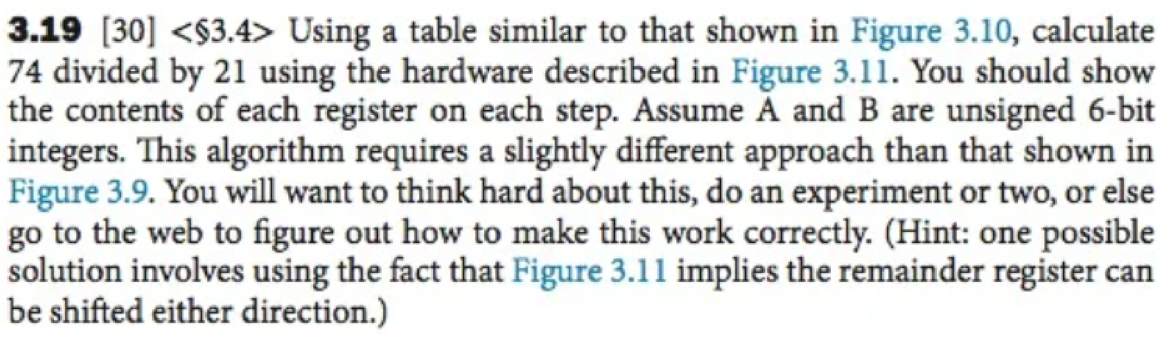
$$
\begin{array}{r|l l}
0 & 214 & 0 \\
 & 107 & 1 \\
 & 53 & 1 \\
 & 26 & 0 \\
 & 13 & 1 \\
 & 6 & 0 \\
 & 3 & 1 \\
 & & 1
\end{array}
\uparrow
\qquad
214_{10} = 1101\ 0110_2
$$

- Adding the two binary operands and convert into decimal:

$$
\begin{array}{r r r}
 & & \overset{1}{\phantom{0}} \\
 & 1001 & 0111 \\
+ & 1101 & 0110 \\
\hline
1 & 0110 & 1101 \\
\hline
\end{array}
= (1 \times 2^8) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^0)
$$

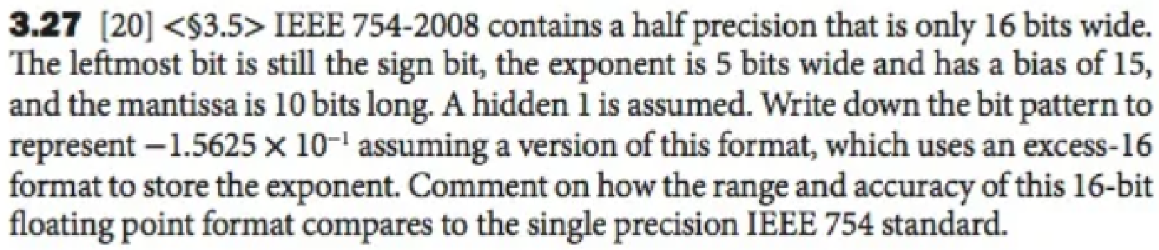
$$= 365_{10}$$

- Since the range of unsigned 8-bits integers is 255, the result of saturating math is the largest 8-bits positive integer 255.


**3.19** [30] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.11. You should show the contents of each register on each step. Assume A and B are unsigned 6-bit integers. This algorithm requires a slightly different approach than that shown in Figure 3.9. You will want to think hard about this, do an experiment or two, or else go to the web to figure out how to make this work correctly. (Hint: one possible solution involves using the fact that Figure 3.11 implies the remainder register can be shifted either direction.)

- Since in decimal, the 74 would have been more than 6 bits, we assume that the number given are in octal.
- Assumes that Register A holds the dividend, 74 and in binary would be 111100.
- Assumes that Register B holds the divisor, 21 and in binary would be 010001.

| Iteration | Step | Divisor | Remainder |
|---|---|---|---|
| 0 | Initial values | 010 001 | 000000 111100 |
| | Shift the remainder left | 010 001 | 000001 111000 |
| 1 | Rem - Div | 010 001 | 110000 111000 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 000011 110000 |
| 2 | Rem - Div | 010 001 | 110010 111000 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 000111 100000 |
| 3 | Rem - Div | 010 001 | 110110 100000 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 001111 000000 |
| 4 | Rem - Div | 010 001 | 111110 000000 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 011110 000000 |
| 5 | Rem - Div | 010 001 | 001101 000000 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 011010 000001 |
| 6 | Rem - Div | 010 001 | 001001 000001 |
| | Rem< 0⇒ +div , LSL R, R=0 | 010 001 | 010010 000011 |
| Last | Shift right left half of the remainder | 010 001 | 001001 000011 |

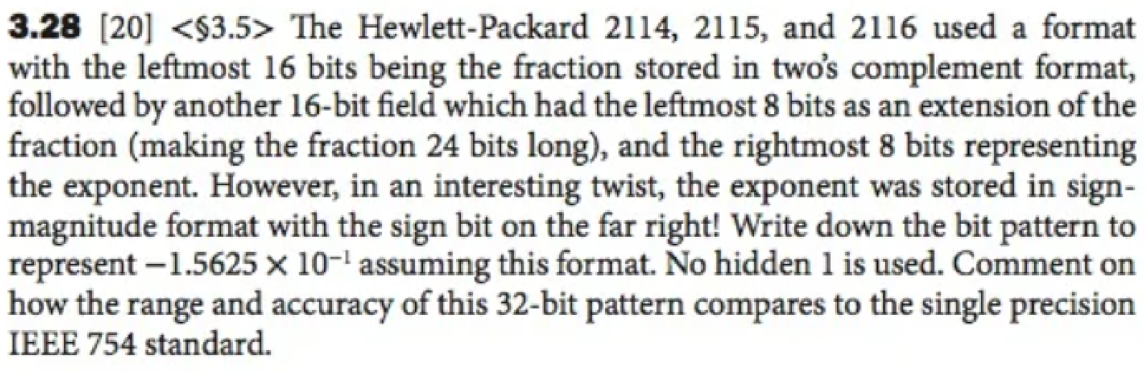

**3.27** [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent $-1.5625 \times 10^{-1}$ assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

- Converting base 10 to base 2, we get that $(-1.5625 \times 10^{-1})_{ten} = -0.00101_{two} = -1.01 \times 2^{-3}$.
- Sign bit = 1
- 10 bit fractional field = 0100000000
- 5 bit exponential field = exponent + bias = $-3 + 15 = 12 = -1100$
- Hence the 16 bit IEEE 754-2008 of $-1.5625 \times 10^{-1}$ is

$$1 \;\; 01100 \;\; 0100000000$$

Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

- The half-precision format can represent numbers with a smaller range compared to single-precision. In half-precision, the range is limited due to the smaller number of bits allocated for the exponent and mantissa. Single-precision offers a wider range of exponents and a larger mantissa, allowing it to represent much larger and smaller numbers.
- The accuracy of the half-precision format is lower compared to single-precision. With only 10 bits for the mantissa, it can represent a smaller number of significant digits compared to the 23 bits in single-precision. This means that half-precision numbers will have less precision and may suffer from rounding errors more frequently than single-precision numbers.



**3.28** [20] <§3.5> The Hewlett-Packard 2114, 2115, and 2116 used a format with the leftmost 16 bits being the fraction stored in two's complement format, followed by another 16-bit field which had the leftmost 8 bits as an extension of the fraction (making the fraction 24 bits long), and the rightmost 8 bits representing the exponent. However, in an interesting twist, the exponent was stored in sign-magnitude format with the sign bit on the far right! Write down the bit pattern to represent $-1.5625 \times 10^{-1}$ assuming this format. No hidden 1 is used. Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard.

- Converting base 10 to base 2, we get that $(-1.5625 \times 10^{-1})_{ten} = -0.00101_{two}$
- There is no hidden 1. Hence, $-0.00101_{two} = (-0.101 \times 2^{-2})_{two}$.
- So, the fraction is 0101 and leading zero must be written to apply two's complement = $0101\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$.
- Since the number is negative, two's complement is applied = $1010\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 11111$.

$$\begin{array}{c} 0101\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000 \\ 1010\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 11111 \\ +0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0001 \\ -\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,- \\ 1011\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000 \end{array}$$

- The exponent is -2. Hence, $0000\ \ 0101$ where the rightmost bit indicates the negative sign, while another represents 2 = 0000 010.
- Therefore, the bit pattern of $-1.5625 \times 10^{-1}$ is

$$1011\ \ 0000\ \ 0000\ \ 0000\ \mid\ 0000\ \ 0000\ \ 0000\ \ 0101$$

<u>Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard</u>

- It uses 24 bits for the fraction and two’s complement. Hence the accuracy is slightly better than IEEE 754 single-precision format as IEE 754single-precision uses 24 bits for fraction, one being sign bit, which lowers the range number by 1.
- Their ranges are almost the same, using exactly the same argument as before.
- This is because the exponents take 8 bits in both formats, so the slight difference in fractions means that the range of IEEE 754 single-precision format is slightly smaller.