HW #2

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.

```
  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.

$\because$ 5ED4 > 0, 07A4 > 0

$\therefore$ 5ED4 − 07A4 = 5730

3.4 [5] <§3.2> What is 4365 – 3412 when these values represent unsigned 12-bit octal numbers? The result should be written in octal. Show your work.

```
  4365
- 3412
------
  0753
```

3.5 [5] <§3.2> 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.

$\because$ 4365 < 0, 3412 > 0 $\therefore$ 4365 − 3412 < 0

−4365 = 0365

```
  0365
+ 3412
------
  3777
```

$\therefore$ 4365 − 3412 = 7777

3.6 [5] <§3.2> Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185–122. Is there overflow, underflow, or neither?

unsigned 8-bit integer range [0,255]
185-122 = 63 ∈ [0,255]
∴ there is neither overflow nor underflow

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.

饱和算法，如果溢出则取最大值/最小值

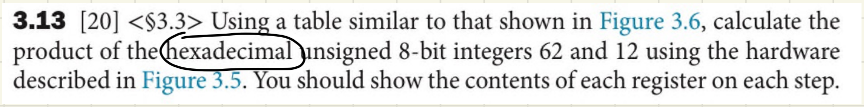
151+214 = 365 > 255
∴ the result is 255

3.12 [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the octal unsigned 6-bit integers 62 and 12 using the hardware described in Figure 3.3. You should show the contents of each register on each step.

110010 × 001010

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	001010	000000110010	000000000000
1	1: 0 ⇒ no operation	001010	000000110010	000000000000
	2: sll Mcand	001010	000001100100	000000000000
	3: srl Mplier	000101	000001100100	000000000000
2	1a: 1 ⇒ Prod = Prod + Mcand	000101	000001100100	000001100100
	2: sll Mcand	000101	000011001000	000001100100
	3: srl Mer	000010	000011001000	000001100100
3	1: 0 ⇒ no operation	000010	000011001000	000001100100
	2: sll Mcand	000010	000110010000	000001100100
	3: srl Mer	000001	000110010000	000001100100
4	1a: 1 ⇒ Prod = Prod + Mcand	000001	000110010000	000111110100
	2: sll Mcand	000001	001100100000	000111110100
	3 srl Mer	000000	001100100000	000111110100

5	1 : 0 ⇒ no operation	000000	0011 00100000	00011111 0100
	2 : sll Mcand	000000	011 001000000	00011111 0100
	3 : srl Mer	000000	011 001000000	00011111 0100
6	1 : 0 ⇒ no operation	000000	011 001000000	00011111 0100
	2 : sll Mcand	000000	11 0010000000	00011111 0100
	3 : srl Mer	000000	11 0010000000	00011111 0100


3.13 [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

0110 0010 × 0001 0010

Iteration	Step	Multiplicand	Product
0	Initial Value	01100010	0000000000010010
1	1 : 0 ⇒ no operation	01100010	00000000 00010010
	2 : srl Product	01100010	00000000 00001001
2	1a : 1 ⇒ Product[15:8] = Mcand + Product[15:8]	01100010	0110 0010 0000 1001
	2 : srl Product	01100010	0011000100000100
3	1 : 0 ⇒ no operation	01100010	0011000100000100
	2 : srl Product	01100010	0001100010000010
4	1 : 0 ⇒ no operation	01100010	0001100010000010
	2 : srl Product	01100010	0000110001000001
5	1a : 1 ⇒ Product[15:8] = Mcand + Product[15:8]	01100010	0110111001000001
	2 : srl Product	01100010	0011011100100000
6	1 : 0 ⇒ no operation	01100010	0011011100100000
	2 : srl Product	01100010	0001101110010000
7	1 : 0 ⇒ no operation	01100010	0001101110010000
	2 : srl Product	01100010	0000110111001000
8	1 : 0 ⇒ no operation	01100010	0000110111001000
	2 : srl Product	01100010	0000011011100100

3.18 [20] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.8. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers.

111100÷010001

Iteration	Step	Quotient	Divisor	Remainder
0	Initial Value	000000	010001000000	000000111100
1	1: Rem = Rem−Div	000000	010001000000	101111111100
	2b: Rem<0 ⇒ +Div, SLL Q, $Q_0=0$	000000	010001000000	000000111100
	3: SRA Div	000000	001000100000	000000111100
2	1: Rem = Rem−Div	000000	001000100000	110111111100
	2b: Rem<0 ⇒ +Div, SLL Q, $Q_0=0$	000000	001000100000	000000111100
	3: SRA Div	000000	000100010000	000000111100
3	1: Rem = Rem−Div	000000	000100010000	111100101100
	2b: Rem<0 ⇒ +Div, SLL Q, $Q_0=0$	000000	000100010000	000000111100
	3: SRA Div	000000	000010001000	000000111100
4	1: Rem = Rem−Div	000000	000010001000	111110110100
	2b: Rem<0 ⇒ +Div, SLL Q, $Q_0=0$	000000	000010001000	000000111100
	3: SRA Div	000000	000001000100	000000111100
5	1: Rem = Rem−Div	000000	000001000100	111111111000
	2b: Rem<0 ⇒ +Div, SLL Q, $Q_0=0$	000000	000001000100	000000111100
	3: SRA Div	000000	000000100010	000000111100
6	1: Rem = Rem−Div	000000	000000100010	000000011010
	2a: Rem>0 ⇒ SLL Q, $Q_0=1$	000001	000000100010	000000011010
	3: SRA Div	000001	000000010001	000000011010
7	1: Rem = Rem−Div	000000	000000010001	000000001001
	2a: Rem>0 ⇒ SLL Q, $Q_0=1$	000011	000000010001	000000001001
	3: SRA Div	$\boxed{000011}$	000000001000	$\boxed{000000001001}$

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.)

111100 ÷ 010001

Iteration	Step	Divisor	Remainder
0	1: Initial Value	010001	000000111100
	2: SLL Rem	010001	$00000111100\underline{\underline{0}}$
1	1: Rem[11:6] = Rem[11:6] − Div	010001	110000111000
	2b: Rem<0 ⇒ Div+, SLL Rem. Rem0=0	010001	$0000111100\underline{\underline{0}}0$
2	1: Rem[11:6] = Rem[11:6] − Div	010001	110010110000
	2b: Rem<0 ⇒ Div+, SLL Rem. Rem0=0	010001	$000111100\underline{\underline{0}}00$
3	1: Rem[11:6] = Rem[11:6] − Div	010001	110110100000
	2b: Rem<0 ⇒ Div+, SLL Rem. Rem0=0	010001	$00111100\underline{\underline{0}}000$ 101111
4	1: Rem[11:6] = Rem[11:6] − Div	010001	111110000000
	2b: Rem<0 ⇒ Div+, SLL Rem. Rem0=0	010001	$0111100\underline{\underline{0}}0000$
5	1: Rem[11:6] = Rem[11:6] − Div	010001	$0011010\underline{\underline{0}}0000$
	2a: Rem>0 ⇒ SLL Rem. Rem0=1	010001	$011010\underline{\underline{0}}00001$
6	1: Rem[11:6] = Rem[11:6] − Div	010001	$001001\underline{\underline{0}}00001$
	2a: Rem>0 ⇒ SLL Rem. Rem0=1	010001	$01001\underline{\underline{0}}000011$
	SRL Rem[11:6]	010001	$\boxed{001001}\boxed{000011}$

3.22 [10] <§3.5> What decimal number does the bit pattern 0 × 0C000000 represent if it is a floating point number? Use the IEEE 754 standard.

0000 1100 0000 ··· 0

$exp = (00011000)_2 - (127)_{10} = (-103)_{10}$

$\therefore$ the floating number is 1.0×2^{-103}

3.23 [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

$(63.25)_{10} = (111111.01)_2 = 1.1111101 \times 2^5$

$5 + 127 = 132 = (10000100)_2$

0 10000100 11111010000000000000000

3.26 [20] <§3.5> Write down the binary bit pattern to represent -1.5625×10^{-1} assuming a format similar to that employed by the DEC PDP-8 (the leftmost 12 bits are the exponent stored as a two's complement number, and the rightmost 24 bits are the fraction stored as a two's complement number). No hidden 1 is used. Comment on how the range and accuracy of this 36-bit pattern compares to the single and double precision IEEE 754 standards.

pattern	exponent range	fraction accuracy
PDP-8	[-2048, 2047]	23
single precision	[-126, 127]	23
double precision	[-1022, 1023]	52

$-1.5625 \times 10^{-1} = -0.15625 = -0.00101 \times 2^0 = -0.101 \times 2^{-2}$

$\therefore$ the binary bit pattern is $\underbrace{1\cdots}_{11bit}1010110\underbrace{\cdots}_{20\ bit}0$

3.29 [20] <§3.5> Calculate the sum of 2.6125×10^1 and $4.150390625 \times 10^{-1}$ by hand, assuming A and B are stored in the 16-bit half precision described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps. $\rightarrow 1+5+10$

$2.6125 \times 10^1 = 26.125 = (11010.001)_2 = 1.1010001 \times 2^4$

presentation: $\underline{0}\ \underline{10011}\ \underline{1010001000}$

$4.150390625 \times 10^{-1} = 1.1010100111 \times 2^{-2}$

presentation: $\underline{0}\ \underline{01100}\ \underline{1010100111}$

① alignment $1.10101001 \times 2^{-2} = 0.0000011010100111 \times 2^4$

② sum

$$\begin{array}{r} \overset{g\,r\,s}{} \\ 0.0000011010101\boxed{1} \\ +\ 1.1010001000000 \\ \hline 1.1010100010101 \end{array}$$

③ round 1.1010100011
to the nearest even

④ result $\underline{0}\ \underline{10011}\ \underline{1010100011}$

3.41 [10] <§3.5> Using the IEEE 754 floating point format, write down the bit pattern that would represent $-1/4$. Can you represent $-1/4$ exactly?

$-\frac{1}{4} = -0.25 = (-0.01)_2 = -1.0 \times 2^{-2}$

$\underline{1}\ \underline{01111101}\ \underbrace{0 \cdots 0}_{23\text{bit}}$ represent $-1/4$ exactly