Tutorial T21
Toh Zhen Yu, Nicholas
A0201406Y

1.
a. $X' = B^n - X$
b.
Let the number be d digits long. Then there are d digits and any number B^d or larger will overflow
By substituting the formula from part a, $n' = B^d - n$
$n' + n = B^d - n + n = B^d = 0 \ (mod \ B^d)$
c.
first we find the positive of each number by padding 0s in front
$123 = (0123)_{10s}$
$456 = (0456)_{10s}$
We are done for 123
To negate 456,
$-456 \ (mod \ 10^5) = 10^4 - 456 \ (mod \ 10^5) = (9544)_{10s}$
d.
$123 - 456 = 123 + 456' = (0123)_{10s} + (9544)_{10s} = (9667)_{10s}$

2.
a.
Firstly, the MSB signifies the sign. Secondly, the range of positive number that can be represented is close to the range of negative numbers that can be represented. Equivalently, there are an approximately equal amount of positive and negative numbers that can be represented.
2^{n-1} negative numbers can be represented and have 0 as the MSB.
$2^{n-1} - 1$ positive numbers can be represented and have 1 as the MSB.
In a 3 bit system, (000) to (011) represent -4 to -1 respectively, (100) represents 0, and (101) to (111) represent +1 to +3 respectively.
b.
Smallest:
Sign = (1) (positive)
Exponent = (000) (smallest possible)
Mantissa = (0000) (smallest possible)
Then this number in decimal is $(1.0000)_2 \times 2^{(000)_2 - 4} = 1 \times 2^{-4} = 0.0625$

Largest:
Sign = (1) (positive)
Exponent = (111) (largest possible)
Mantissa = (1111) (largest possible)
Then this number in decimal is $(1.1111)_2 \times 2^{(111)_2 - 4} = 1.9375 \times 2^3 = 15.5$

c.
X=4.4
Approximation to 6sf in binary: $(100.011)_2 = (1.00011)_2 \times 2^2$
Sign: 0 (positive)
Exponent: $2 + 4 = 6 = (110)_2$
Mantissa: $(0010)_2$
Number in binary: $0b01100010$
Number in hex: $0x62$

Y=-0.7
Approximation to 6sf in binary: $-(0.10110)_2 =-(1.0110)_2 \times 2^{-1}$
Sign: 1 (negative)
Exponent: $-1+4=3=(011$
Mantissa: $(0110)_2$
Number in binary: $0b10110110$
Number in hex: $0xB6$

d.
Converting X back to decimal:
$-(1.0010)_2 \times 2^2$ = 4.5
Absolute error = 4.5-4.4 = 0.1
Fractional error = 0.1 / 4.4 = 1/44 ≈ 0.0227

Converting Y back to decimal:
$-(1.0110)_2 \times 2^{-1}$ = -0.6875
Absolute error = -0.6875 - (-0.7) = 0.0125
Fractional error = 0.0125 / 0.7 = 1/56 ≈ 0.0179

Neither number is represented perfectly.

e.
Accuracy of Y is better, because its absolute and fractional error are lower compared to X

3.
a.
Table 1: Component Sizes

variable	Size in bytes
t1.num1	4
t1.ch	1
t1.num2	4

Table 2: Component Addresses

variable	Address
&t1.num1	000000000061FE10
&t1.ch	000000000061FE14
&t1.num2	000000000061FE18

Table 3: Array Element Addresses

variable	Address
&t2[0]	000000000061FDC0.
&t2[1]	000000000061FDCC.
&t2[2]	000000000061FDD8.

b.
They differ by 4 bytes. Even though the size of t1.ch is 1 byte, the next piece of data is stored 4 bytes away due to word alignment.

c.
```
        add $t0, $zero, $zero       # $t0 represents our counter i, initialized to 0
        addi $t1, $zero, 6          # $t1 holds the value 6 (to exit the loop)
        add $t2, $zero, $s0         # $t2 holds the address of t2[i]
loop:   beq $t0, $t1, exit
        lw $t3, 8($s1)              # load result.num2 into $t3
        lw $t4, 8($t2)              # load t2[i].num2 into $t4
        add $t3, $t3, $t4           # add t2[i].num2 to result.num2
        sw $t3, 8($s1)              # store the updated result.num2
        addi $t0, $t0, 1            # increment i
        addi $t2, $t2, 12           # update address of t2[i] (strength reduction to avoid multiply)
        j loop
exit:
```

4.
a.
```
        addi $1, $0, 0          # use $t0 as counter because $1 is reserved
        add $3, $20, $1           # $3 stores the address of the current word
loop:   lw $4, 0($3)                # loads arr[i] into $4
        srl $5, $4, 16
        sll $6, $4, 16
        or $4, $5, $6
        sw $4, 0($3)                # writes updated value to arr[i]
        addi $1, $1, 1              # increment counter
        addi $3, $3, 4          # update address of arr[i]
        slt $5, $1, $21
        bne $5, $zero, loop         # loops if $t3<$t21 ie i < array size
```

b.
The loop will execute 5 times. There are 9 instructions in the loop. Hence 2+5*9=47 instructions will be executed

c.
The program swaps the first two and last two bytes of each integer in the array