T 1

1. Convert these decimal numbers to 8-bit 2's complement numbers:

- -144

$$-114 = \sum_{i=0}^{7} b_i \times 2^i$$

Since -114 is negative, we can convert the magnitude of -144 first ,which is actually 114.Because 144 is even and positive, so we can inffer that $b_0 = 0$, $b_7 = 0$ .After that we subtract 0 from both side of the equation ,and than divide both side of equation by 2 :

$$57 = \sum_{i=1}^{6} b_i \times 2^{i-1}$$

Obviously 57 is odd,so we can inffer that $b_1 = 1$,and than we subtract 1 form both side of the equation and divide by 2:

$$28 = \sum_{i=2}^{6} b_i \times 2^{i-2}$$

According to the previous steps,we can inffer that $b_2 = 0$,and:

$$14 = \sum_{i=3}^{6} b_i \times 2^{i-3}$$

Countinue to calculate and we will get $b_3 = 0$ and:

$$7 = \sum_{i=4}^{6} b_i \times 2^{i-4}$$

Finally we get $b_4 = 1, b_5 = 1, b_6 = 1$, so

$$114 = 01110010$$

than we add 1 to the complement of 01110010:

$$-144 = 10001110$$

- $+81$

$$+81 = \sum_{i=0}^{7} b_i \times 2^i$$

Same as the last work,we get:

$$+81 = 01010001$$

T 2

1. What's the smallest and largest number that can be represented by an 8-bit 2's complement number? (Answer in decimal)

- The biggist number is:

$$2^7 - 1 = 127$$

  And the smallest number is:

$$-2^8 = -128$$

2. Try to determine the range that an N-bit 2's complement number can represent. (Answer in decimal)

- The range that an **N-bit** 2's complement number is:

$$-2^N \leqslant \textbf{The number} \leqslant +2^N - 1$$

T 3

1. Is there a negative integer that has identical 2's complement representation and original code in binary (8-bit)? If so, what is it? (Answer in decimal)

- -64 has identical 2's complement representation and originalcode in binary (8-bit).

$$X = -\left(1 \times 2^7\right) + X$$
$$X = -\left(2^6\right)$$
$$X = -64$$

T 4

1. Under what circumstances will the program print $a < b$ while actually $a \geqslant b$?

- This happens when integer overflow occurs. For example we define $a = \text{INT\_MAX}, b = -1$, the program will print $a < b$ but actually $a \geqslant b$.

2. What if we change the code to the following? (Also, the numbers given are guaranteed to be in the range of unsigned int .)

- The code will accurately compare the `unsigned int` values a and b and print "$a < b$" if a is less than b, and "a $\geqslant$ b" if a is greater than or equal to b.

T 5

1. Write the decimal equivalents for the IEEE floating point number below.

$$0\,10001011\,00000000001000000001000$$

- $$\begin{aligned}
&a_{31} = 0 \Rightarrow X \geqslant 0 \\
&a_{23} \sim a_{30} = 10001011\,(\text{binary}) = 139 \Rightarrow \text{The Exponent bit is:}\, 139 - 127 = 12 \\
&a_0 \sim a_{22} = 00000000001000000001000 = 1.00000000001000000001000 \\
&\Rightarrow X = +1.00000000001000000001000\,(\text{binary}) \times 2^{12} = 4098.003906
\end{aligned}$$

T 6

1. What is the smallest number that can be represented in IEEE floating point format with 32 bits regardless of infinity? What about the smallest positive number? (Answer in binary)

- Sign Bit: 1 (for negative)
  Exponent Bits: 11111110
  Mantissa Bits (all 1):

  $$\begin{aligned}
  11111111111111111111111 &= 1.11111111111111111111111 \\
  &= 2 - 0.00000000000000000000001
  \end{aligned}$$

  Which is $-1 \times (2 - 2^{-23}) \times 2^{127} = -3.4028 \times 10^{38}$

- Sign Bit: 0 (for positive)
  Exponent Bits : 00000000
  Mantissa Bits:

  $$\begin{aligned}
  00000000000000000000001 &= 0.00000000000000000000001 \\
  &= 2^{-23}
  \end{aligned}$$

  Which is $(-1)^0 \times 2^{-23} \times 2^{-126} = 1.4013 \times 10^{-45}$

T 7

1. Can you list all the integers whose IEEE floating point representations are exactly the same as their 2's complement integer representations? (Answer in decimal)

- The numbers are:

1. -834214802 2's Complement = 11001110010001101110010001101110 = IEEE
2. 0 2's Complement = 00000000000000000000000000000000 = IEEE
3. 1318926965 2's Complement = 01001110100111010011101001110101 = IEEE

The code of the program to solve this task will be attached to the end of the document.

T 8

1. The code below uses three XOR operation to swap two integers. void swap(int *a, int *b)

(1) Fill in the blanks to complete the code.

- The code is below:

```c
void swap(int *a, int *b) {
    *a = *a ^ *b;
    *b = *a ^ *b;
    *a = *a ^ *b;
}
```

(2) Is there anything wrong to use the swap function in the sorting function below? If so, how can you fix it?

- In the given sort function, we are passing pointers to **swap(a + i, a + min)**;. The swap function we've defined expects pointers to integers (integers' addresses), but we are passing pointers to pointers to integers. To fix this, we need to dereference the pointers we're passing to **swap**:

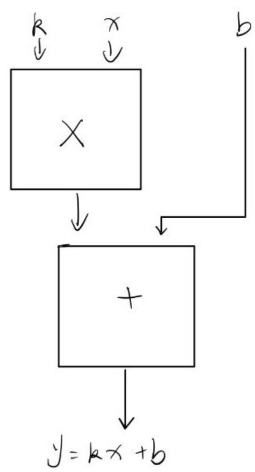
```c
void sort(int *a, int n) {
    // sort a[0] ~ a[n - 1]
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i; j < n; j++) {
```

```
                if (a[j] < a[min]) {
                    min = j;
                }
            }
            swap(&a[i], &a[min]); // Fix
        }
    }
```

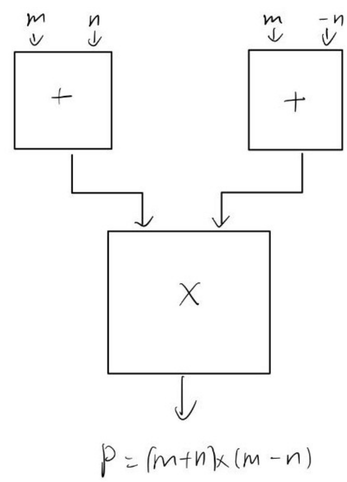
T 9

1. You're required to draw circuits that can calculate the following expressions:

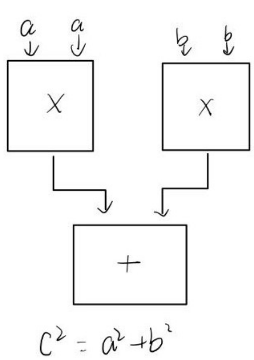
- $y = kx + b$



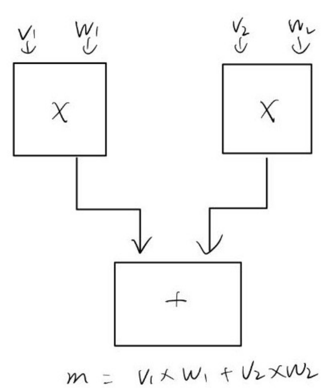
- $p = (m + n)(m - n)$



- $c^2 = a^2 + b^2$



- $m = v_1 \times w_1 + v_2 \times w_2$



T 10

1. How many bits do we need to represent a single character?

- The number of bits needed to represent a single character depends on the total number of characters we want to represent. In this case, we want to represent 26 uppercase letters (A-Z), 26 lowercase letters (a-z), 10 digits (0-9), space, and ".". That's a total of $(26 + 26 + 10 + 2 = 64)$ characters.To represent a single character using binary, we would need to use **at least 6 bits** because $2^6 = 64$, which is enough to represent 64 unique characters.

2. How many bits do we need to represent a string of characters?

- To represent a string of characters, we would need to multiply the number of bits per character by the length of the string. If we use 6 bits per character, a string of N characters would require $6 \times \mathrm{N} = 6\mathrm{N}$ bits.

3. Assume that we use 0 to represent A , 1 to represent B , and so on. So we use 63 to represent '.' What is the binary representation of 'Hello World.' ?

- In this case ,we can inffer that 'Hello World.' is represented by 12 numbers , which is :

7(H) 30(e) 37(l) 37(l) 40(o) 62('space')

22(W) 40(o) 43(r) 37(l) 29(d) 63('.')

Then we code these numbers to 6-bit binary:

000111 011110 100101 100101 101000 111110 010110 101000 101011 100101 011101 111111

Code for T7

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

union IntToBinary {
    int integer;
    unsigned char bytes[sizeof(int)];
};

union FloatToBinary {
    float floating_point;
    unsigned int binary;
};

char* floatToBinary(float num) {
    static char binary[sizeof(float) * 8 + 1]; // +1 for '\0'
    union FloatToBinary converter;
    converter.floating_point = num;

    int index = 0;
    for (int i = sizeof(unsigned int) - 1; i >= 0; i--) {
        for (int j = 7; j >= 0; j--) {
            binary[index++] = ((converter.binary >> (i * 8 + j
               )) & 1) + '0';
        }
    }
    binary[index] = '\0'; // the end

    return binary;
}

char* intToBinary(int num) {
    static char binary[sizeof(int) * 8 + 1]; // +1 for '\0'
    union IntToBinary converter;
    converter.integer = num;

    int index = 0;
    for (int i = sizeof(int) - 1; i >= 0; i--) {
```

```c
            for (int j = 7; j >= 0; j--) {
                binary[index++] = ((converter.bytes[i] >> j) & 1) + '0';
            }
        }
        binary[index] = '\0'; // the end

        return binary;
    }

    int compareArrays(const char* arr1, const char* arr2, int size) {
        return memcmp(arr1, arr2, size) == 0; // using memcmp to compare
    }

    int main() {
        int num;
        char *F , *In;

        for(num = INT_MIN ; num <= INT_MAX-1 ; num++){
            F = floatToBinary((float)num);
            In = intToBinary((int)num);

            if(compareArrays(F,In,32)){
                printf("\nThe number is %d ,2's Complent: %s ,IEEE: %s\n" , num , In, F);
            }
        }

        if(floatToBinary((float)INT_MAX) == intToBinary(INT_MAX)){
            printf("%d" , INT_MAX);
        }

        return 0;
    }
```