**Problem x. Floating Point Number Conversion**

We’ve learned number expression and bit operation in class. In this problem, we’ll design a “new” floating-point number which is slightly different with the IEEE 754 standard, and implement the conversion between hexadecimal and our floating-point number.

Before the problem description, let me just recap on what we’ve learned so far.

1. Number expression

The built-in types of C language like int, float, double…are actually bits (a binary sequence, e.g., 0b0100100…) in memory. For example, usually, char contains 8 bits, int and float contains 32 bits, and double contains 64 bits.

1. Bit operations

We’ve learned bitwise operations and (&), or (|), not (~), xor (^) in class. The truth tables are shown in the following table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | a&b | a|b | a^b | ~a |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |

For example, assume A = 60 and B = 13 in binary format, they will be as follows.

A = 0b00111100

B = 0b00001101

A&B = 0b00001100

A|B = 0b00111101

A^B = 0b00110001

~A = 0b11000011

In this problem, you’ll find these operations are useful when extracting or modifying some bits in a number.

Now, let’s design our floating-point number. First, we’ll learn the floating-point encoding in our computer (IEEE 754 standard, <https://en.wikipedia.org/wiki/IEEE_754>).

A single-precision floating-point number is usually a 32-bit field in memory, it can be divided into 3 parts: sign, exponent and fraction. A standard float usually contains 23 fraction bits (0-22), 8 exponent bits (23-30) and 1 sign bit (31).

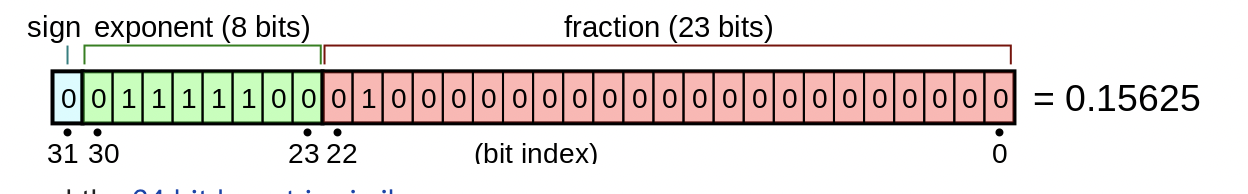
The formula of converting a floating-point expression to a decimal value (V) is:



Where E is:m



For example, let’s convert a 32-bit hexadecimal floating-point expression 0x3e200000 (in binary, 0b00111110001000000000000000000000) to decimal. The layout of this number is shown in the following figure.



Notice that the width of exponent field is 8 bits, so we can calculate E first:



And the fraction part is 01000000000000000000000binary, so “0.fraction” is 0.01binary. The expression of rational number in binary is similar to which in decimal. (e.g., 0.1binary = 0.5decimal, 0.01binary = 0.25decimal). Hence, (1 + 0.fraction)=1.01binary = 1.25decimal. Finally, the value is

V = (-1)0\*1.25\*2(124-127) = 0.15625.

For more details about standard floats, you can refer to this link: <https://en.wikipedia.org/wiki/Single-precision_floating-point_format>

But in this problem, we did a slightly modification on the standard floating-point number. Our float still holds 32 bits in memory, and the width of sign field is still 1. But the width of exponent is 7 bits, and width of fraction is 32-1-7=24bits, which is shown in the following figure

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

sign exponent fraction

In this expression, the following number can be converted to:

V = (-1)0 \* (1 + 0.001binary) \* 2(62-63) = (-1)0 \* 1.125 \* 2(62-63) = 0.5625

In this problem, you’ll receive a hexadecimal number, and you need to write a program which can convert it into a decimal number using the rules given before, then print the converted number to the screen. Each testcase contains only one hexadecimal number, and you need to print the result by this format:

printf("The input number is %f\n", result);

/\*And to simplify this problem, special cases like NaN and denorm will not occur in testcases. So, you can ignore these cases when writing your program.\*/

Sample input & output

|  |  |
| --- | --- |
| Input1 | 0x3e200000 |
| Output1 | The input number is 0.562500 |
| Input2 | 0x3e600000 |
| Output2 | The input number is 0.687500 |