

Converting it back into the decimal equivalent form: 1.11111011100000000000000 x 2141

Exponent = 141 – 127 = 14

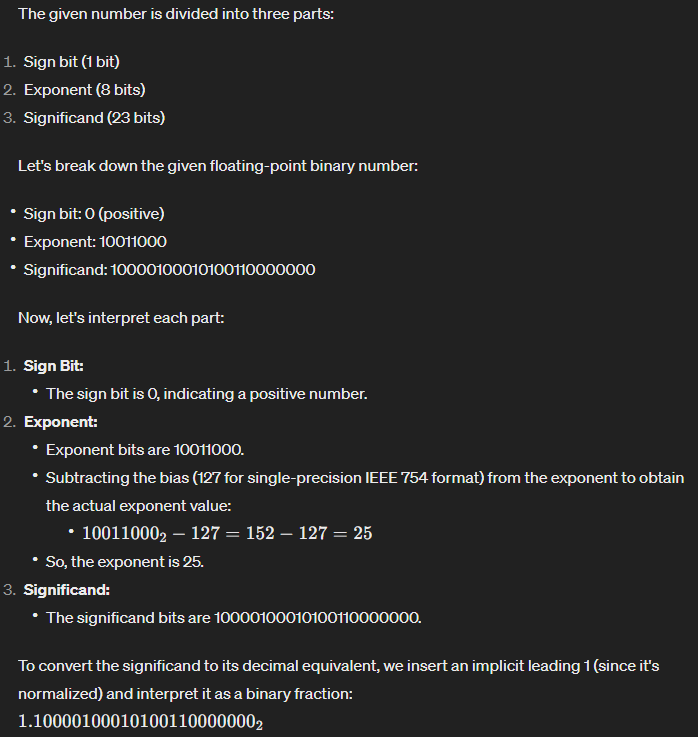
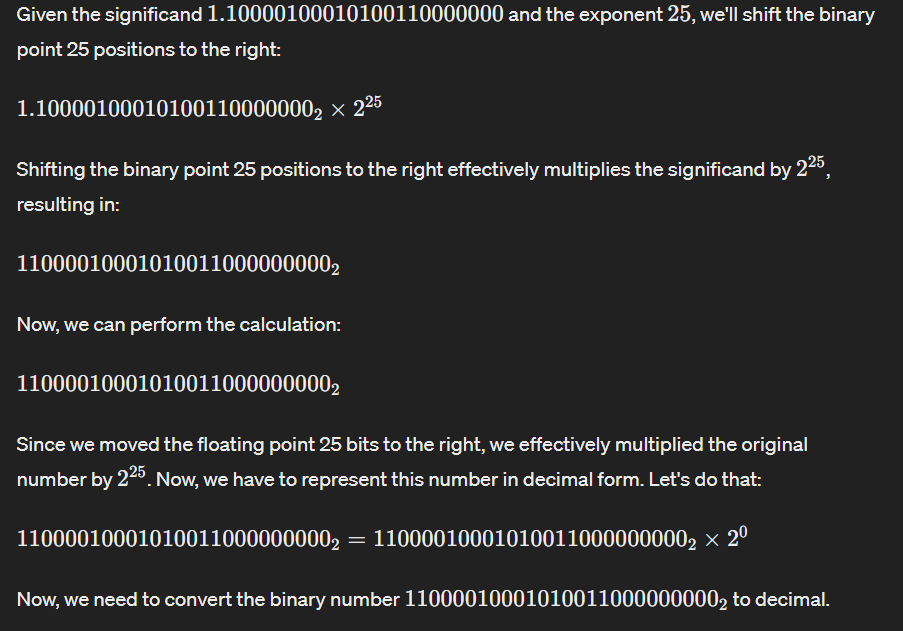
1.11111011100000000000000 x 214 = (111111011100000)2 = (32480)₁₀

**--Determine the binary value of the following floating-point binary number:   
0 10011000 10000100010100110000000**

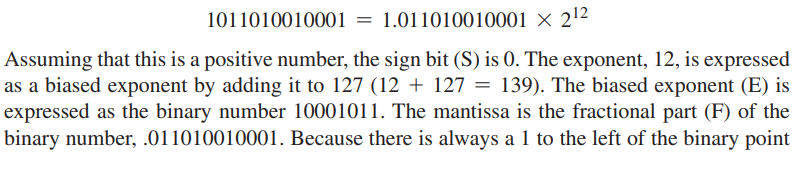
Sign Bit indicates the number is positive.

Exponent = 10011000 = 152 – 127 = 25

1.10000100010100110000000 x 225 = (11000010001010011000000000)₂ = (1 × 2²⁵) + (1 × 2²⁴) + (0 × 2²³) + (0 × 2²²) + (0 × 2²¹) + (0 × 2²⁰) + (1 × 2¹⁹) + (0 × 2¹⁸) + (0 × 2¹⁷) + (0 × 2¹⁶) + (1 × 2¹⁵) + (0 × 2¹⁴) + (1 × 2¹³) + (0 × 2¹²) + (0 × 2¹¹) + (1 × 2¹⁰) + (1 × 2⁹) + (0 × 2⁸) + (0 × 2⁷) + (0 × 2⁶) + (0 × 2⁵) + (0 × 2⁴) + (0 × 2³) + (0 × 2²) + (0 × 2¹) + (0 × 2⁰) = (50898432)₁₀

(11000010001010011000000000)2 = 29 + 210 + 213 + 215 + 219 + 224 + 225 = (50898432)10

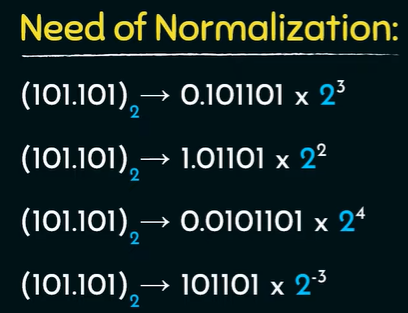


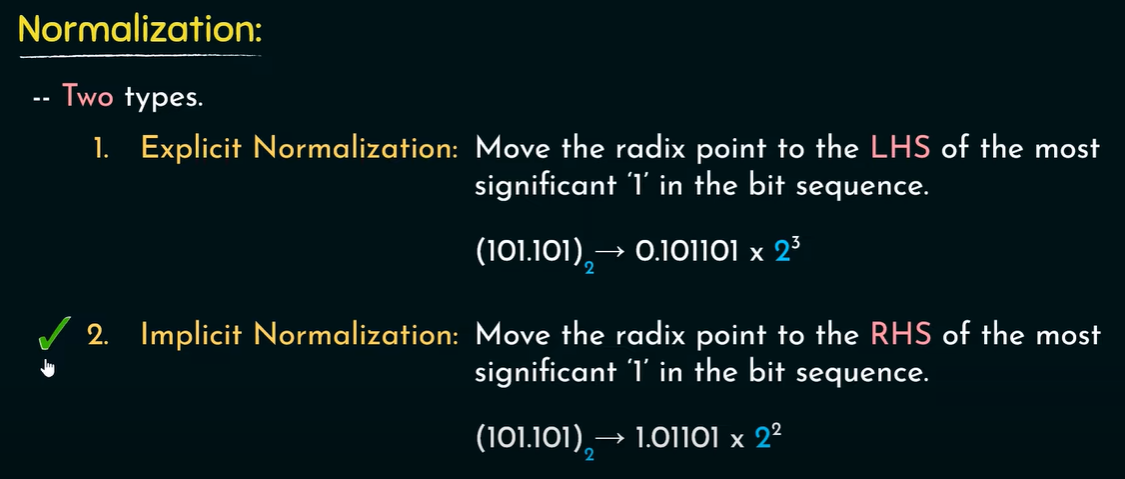
A screenshot of a computer

Description automatically generated

**-----------------THE END-----------------**

General Knowledge👇





Implicit normalization is better with respect to precision.

A screen shot of a black background

Description automatically generated

A screenshot of a blackboard with numbers and numbers

Description automatically generated

A screen shot of a graph

Description automatically generated

Since comparators cannot handle signed and unsigned numbers both, basically they only with unsigned numbers.

A screenshot of a computer

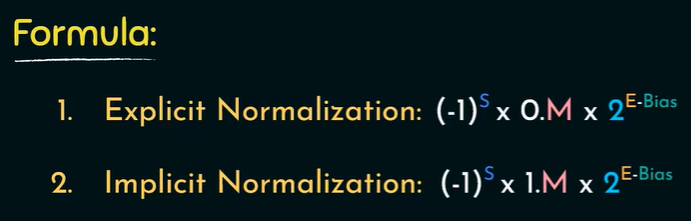
Description automatically generated

We converted the range into an unsigned form.

A screenshot of a computer

Description automatically generated

**Biasing in 10-bit memory space**

****

A screenshot of a computer

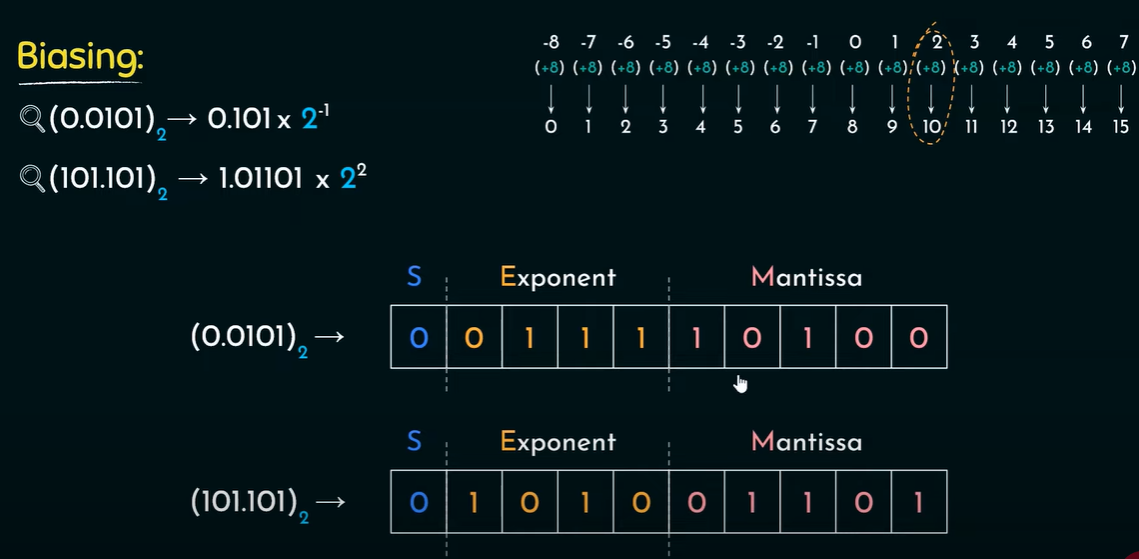
Description automatically generated

Perform explicit normalization on the first value and implicit normalization on the second value.

A screenshot of a computer

Description automatically generated

This is how the first value (0.0101)2 is going to be saved in the memory using explicit normalization.



This is how the second value (101.101)2 is going to be saved in the 10-bit memory using implicit normalization.