Creating the IEEE Representation

The passage below describes how to create the IEEE representation of a floating-point number.

The first step is to normalize the sign bit, exponent, and significand fields. This means that the significand must be adjusted so that the leading bit is 1.

For example, the significand 1.1011 is normalized by shifting the bits one position to the left, giving the normalized significand 11.011.

The next step is to encode the sign bit, exponent, and significand fields.

The **sign bit** is encoded as a single bit, with 0 representing a positive number and 1 representing a negative number.

The **exponent** is encoded as an 8-bit unsigned integer with a bias of 127. This means that the number's actual exponent must be added to 127 to get the biased exponent.

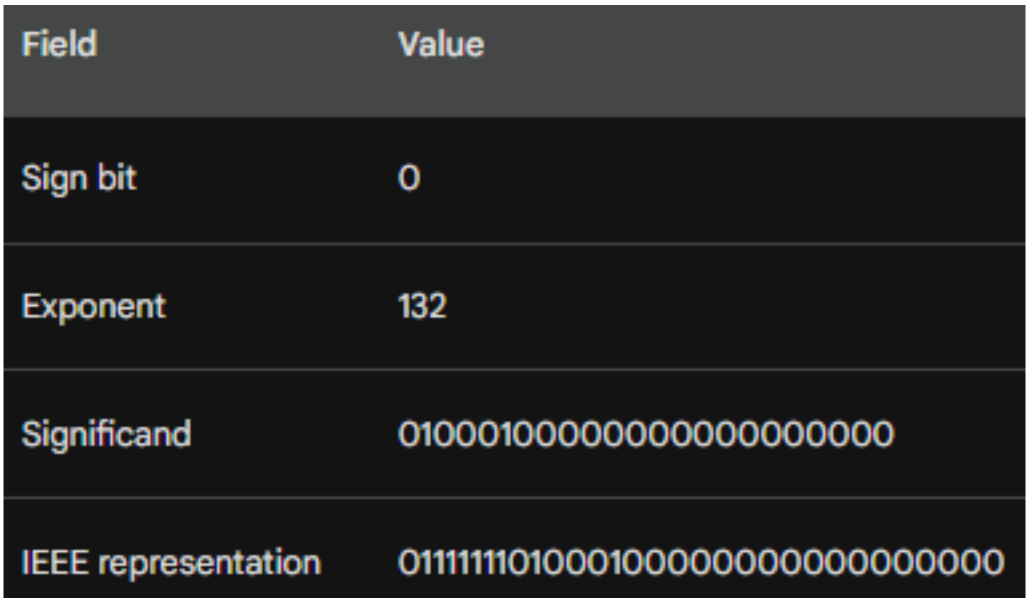
For example, the actual **exponent 5** is **encoded as** the **biased exponent 132.**

The **significand** is encoded as a 23-bit unsigned integer. The leading bit of the normalized significand is not explicitly encoded, since it is always 1.

This means that the significand 11.011 is encoded as the 23-bit integer 01000100000000000000000.

Once the sign bit, exponent, and significand fields have been encoded, they can be combined to form the complete binary IEEE short real.

The following table shows an example of how to create the IEEE representation of the floating-point number 1.101 * 2^{20}:

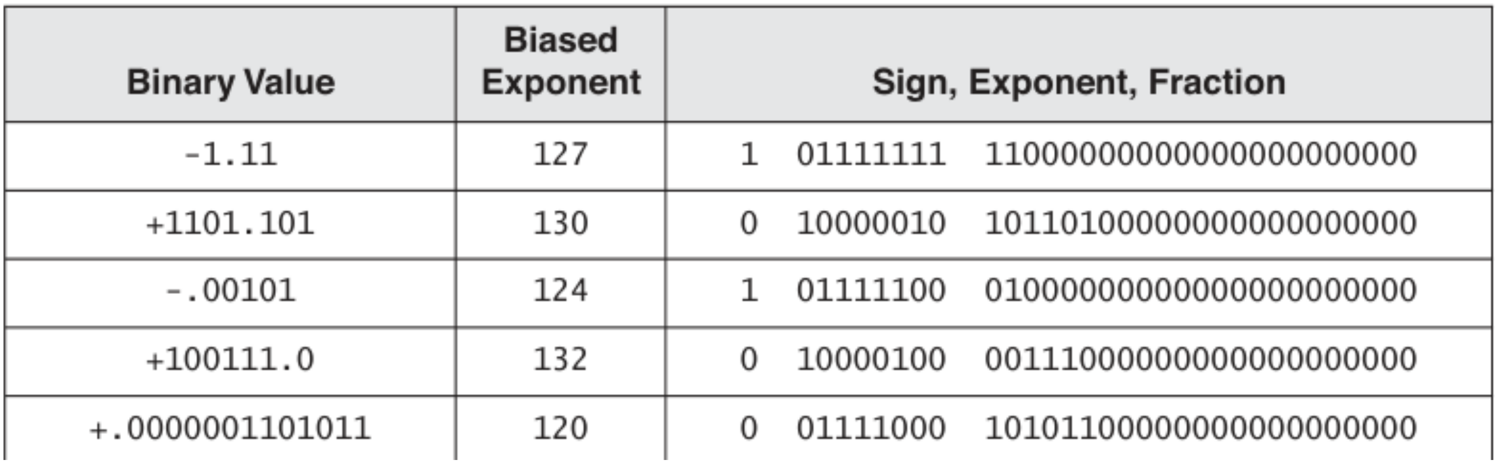


Field	Value
Sign bit	0
Exponent	132
Significand	01000100000000000000000000
IEEE representation	0111111101000100000000000000000000

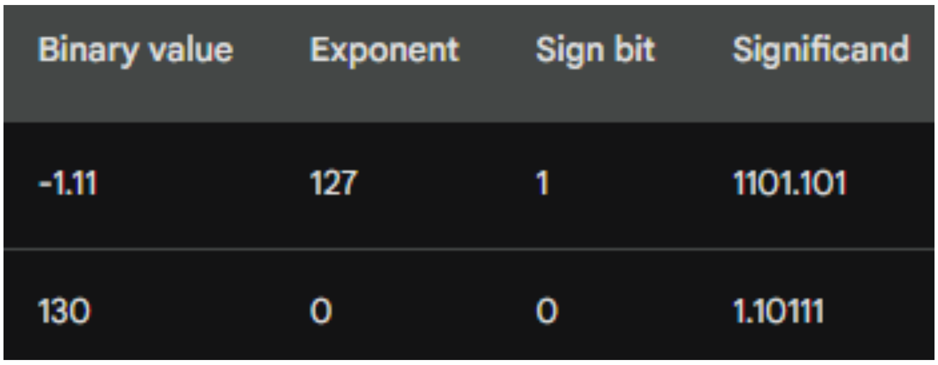
The IEEE representation of a floating-point number can be used to store and manipulate the number in a computer.

IEEE floating-point numbers are widely used in a variety of applications, including scientific computing, graphics, and gaming.

The table you sent shows the binary value, exponent, sign bit, and significand of several different IEEE floating-point numbers. Here is a concise explanation of each row:



Binary Value	Biased Exponent	Sign, Exponent, Fraction
-1.11	127	1 01111111 11000000000000000000000
+1101.101	130	0 10000010 10110100000000000000000
-.00101	124	1 01111100 01000000000000000000000
+100111.0	132	0 10000100 00111000000000000000000
+.0000001101011	120	0 01111000 10101100000000000000000



Binary value	Exponent	Sign bit	Significand
-1.11	127	1	1101.101
130	0	0	1.10111

The first row shows the IEEE representation of the floating-point number -1.11. The sign bit is 1, which indicates that the number is negative. The exponent is 127, which is the largest possible exponent for a normalized floating-point number. The significand is 1101.101, which is the binary representation of the fraction 3/4.

The second row shows the IEEE representation of the floating-point number 1.10111. The sign bit is 0, which indicates that the number is positive. The exponent is 0, which means that the significand is not multiplied by 2. The significand is 1.10111, which is the binary representation of the fraction 23/16.

The other rows in the table show the IEEE representations of other floating-point numbers. The last row in the table shows the smallest possible normalized floating-point number.

The IEEE specification includes several real-number and non-number encodings, including:

Positive and negative zero:
These are two different representations of the number zero.

Denormalized finite numbers:
These are numbers that are very close to zero. They are represented using a special format that allows them to be represented with a wider range of exponents.

Normalized finite numbers:
These are all the other nonzero finite numbers that can be represented in floating-point format. They are represented using a format that allows them to be represented with a high degree of accuracy.

Positive and negative infinity:
These are representations of the concepts of positive and negative infinity.

Non-numeric values (NaN, known as Not a Number):
These are used to represent results of invalid floating-point operations.

Indefinite numbers:
These are used by the floating-point unit (FPU) as responses to some invalid floating-point operations.

Normalized and denormalized numbers

Normalized finite numbers are all the nonzero finite values that can be encoded in a normalized real number between zero and infinity. Normalized numbers have a leading bit of 1 in the significand. This allows for the greatest possible precision in representing the number.

Denormalized finite numbers are numbers that are very close to zero. They are represented using a special format that allows them to be represented with a wider range of exponents. Denormalized numbers have a leading bit of 0 in the significand. This means that some precision is lost in representing the number, but it allows for a wider range of numbers to be represented.

In the example you provided, the number:

$1.01011110000000000001111 \times 2^{-129}$

Is too small to be represented as a normalized number.

The FPU therefore denormalizes the number by shifting the binary point left 1 bit at a time until the exponent reaches a valid range.

$1.01011110000000000001111 \times 2^{-129}$ -> $0.10101111000000000000111 \times 2^{-128}$
$0.10101111000000000000111 \times 2^{-128}$ -> $0.01010111100000000000011 \times 2^{-127}$
$0.01010111100000000000011 \times 2^{-127}$ -> $0.00101011110000000000001 \times 2^{-126}$

After three denormalization steps, the exponent is now within the valid range. The denormalized number is now represented as $0.00101011110000000000001 \times 2^{-126}$.

This results in some loss of precision in the significand, but it allows the number to be represented.

Denormalized numbers are important because they allow the FPU to represent a wider range of numbers than it would otherwise be able to. This is important for many applications, such as scientific computing and graphics.

So, to finish off:

Converting IEEE single-precision values to decimal

The following are the steps to convert an IEEE single-precision value to decimal:

- Check the most significant bit (MSB) to determine the sign of the number. If the MSB is 1, the number is negative. If the MSB is 0, the number is positive.
- Extract the next 8 bits to get the exponent. Subtract 127 from the exponent to get the unbiased exponent. Convert the unbiased exponent to decimal.
- Extract the next 23 bits to get the significand. Add a leading 1 to the significand bits. Ignore any trailing zeros.
- Create a floating-point binary number using the significand, the sign determined in step 1, and the exponent calculated in step 2.
- Denormalize the binary number produced in step 3. To do this, shift the binary point the number of places equal to the value of the exponent. If the exponent is positive, shift the binary point to the right. If the exponent is negative, shift the binary point to the left.
- Convert the denormalized binary number to decimal using weighted positional notation.

Example:

Convert the following IEEE single-precision value to decimal:

0 10000010 01011000000000000000000

- The MSB is 0, so the number is positive.
- The unbiased exponent is 00000011, which is equal to decimal 3.
- The significand is 01011000000000000000000, which is equal to 1.01011 binary.
- The floating-point binary number is 1.01011 * 2^3.
- The denormalized binary number is 1010.11.
- The decimal value is 10 3/4, or 10.75.