Floating Point Binary Representation

These data conversions were dealt with in the first chapters notes.

Floating-point binary representation is a method for encoding decimal numbers using three primary components: a **sign**, a **significand (or mantissa)**, and an **exponent**. These components work together to represent numbers like **-1.23154×10^5** in a binary format.

• **Sign:** The sign indicates whether the number is positive or negative.
In the example **-1.23154×10^5**, the sign is negative.

• **Significand (Mantissa):** The significand is the main part of the number and contains its value. It's similar to the decimal part of a real number. In the example, the significand is 1.23154.

• **Exponent:** The exponent indicates the order of magnitude by which the significand should be multiplied.

It determines whether the number is very large or very small. In this case, the exponent is 5, which means the significant 1.23154 should be multiplied by 10^5 to get the final value.

This representation is commonly used in computer systems for storing and processing real numbers with both integer and fractional parts. It allows for a wide range of values, from very small to very large, and is essential for scientific and engineering calculations.

----

x86 processors employ the IEEE 754-1985 standard for Binary Floating-Point Arithmetic, which defines **three distinct binary floating-point storage formats.**

These formats are designed to represent real numbers with varying precision and characteristics. Here, we'll focus on the single-precision format, as it's the most common and widely used.

Single-Precision Format (32 bits)

In the single-precision format, 32 bits are used to represent a floating-point number. These bits are organized with the most significant bit (MSB) on the left, and they are stored in memory in little-endian order, meaning that the least significant bit (LSB) is located at the starting memory address.

The Sign Bit: The leftmost bit, or the bit at the very start, is the sign bit. If this bit is set to 1, it indicates that the number is negative, whereas if it's set to 0, the number is positive. It's worth noting that zero is considered a positive number in this context.

The Significand (Mantissa): The significand, also known as the mantissa, constitutes the part of the number that holds the value. It contains both the integer and fractional components of the number. In this representation, the significand consists of 23 bits for the fractional part.

The Exponent: In the floating-point number expressed as m * be, 'm' represents the significand (or mantissa), 'b' stands for the base, and 'e' represents the exponent. The exponent part plays a vital role in determining the scale or magnitude of the number.

In essence, the significand (mantissa) represents the decimal digits on both sides of the decimal point, and this can be seen as an extension of the concept of weighted positional notation used in various numbering systems like binary, decimal, and hexadecimal.

Just like the integers to the left of the decimal point have positive powers of 10, the numbers to the right side of the decimal point have negative powers of 10.

To illustrate this with a decimal number, consider 123.154. It can be represented as a sum, with each digit's position contributing based on its exponent:

$123.154 = \mathbf{(1 * 10^{2}) + (2 * 10^{1}) + (3 * 10^{0}) + (1 * 10^{-1}) + (5 * 10^{-2}) + (4 * 10^{-3})}$

This concept applies similarly to the binary floating-point format, where the significand holds the binary digits to the left and right of the binary point (analogous to the decimal point).

It allows representation of a wide range of real numbers, from very small to very large, making it an essential part of computer systems and scientific calculations.

In summary, the IEEE single-precision floating-point format, as used in x86 processors, provides a standardized way to represent real numbers using a combination of a sign bit, significand, and exponent.

This format allows for accurate and flexible representation of numbers across a wide range of magnitudes.

Double precision format and Double Extended

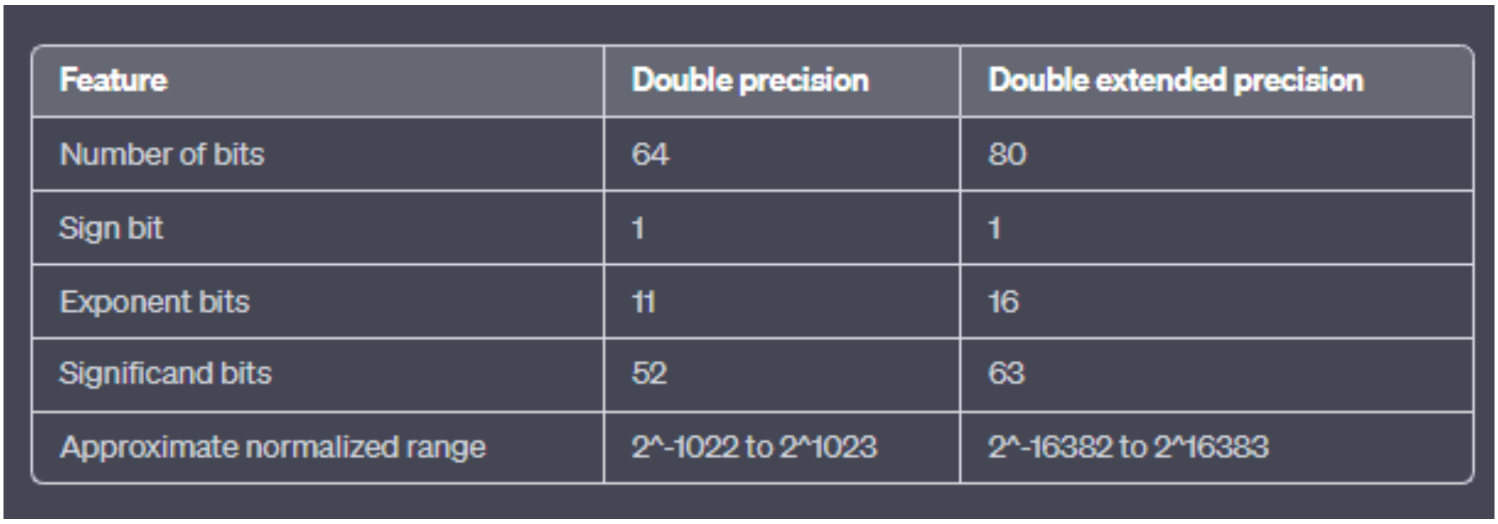
Double precision is a floating-point number format that uses **64 bits** to represent a number. This allows for a wider range of values and greater precision than single precision, which uses 32 bits.

It's the most common floating-point format used in scientific computing and other applications where high precision is required.

Double extended precision is a floating-point format that uses **80 bits** to represent a number. This provides even greater range and precision than double precision.

However, double extended precision is not as widely supported as double precision, so it is only used in applications where the extra precision is essential.

Here is a table comparing double precision and double extended precision:



Feature	Double precision	Double extended precision
Number of bits	64	80
Sign bit	1	1
Exponent bits	11	16
Significand bits	52	63
Approximate normalized range	2^-1022 to 2^1023	2^-16382 to 2^16383

The single-precision floating-point format is a 32-bit format that consists of three parts:

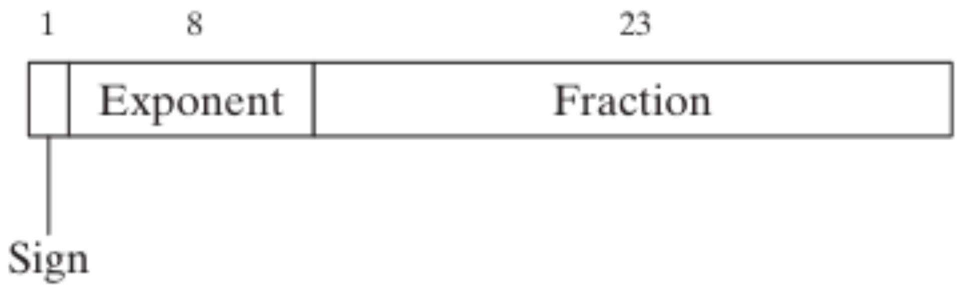
Sign bit: A single bit that indicates whether the number is positive or negative.

Exponent: An 8-bit field that represents the power of 2 by which the significand is multiplied.

Significand: A 23-bit field that represents the fractional part of the number.

The **sign bit is the most significant bit of the floating-point number**. The exponent is stored next, followed by the significand.

The following diagram shows the layout of the single-precision floating-point format:



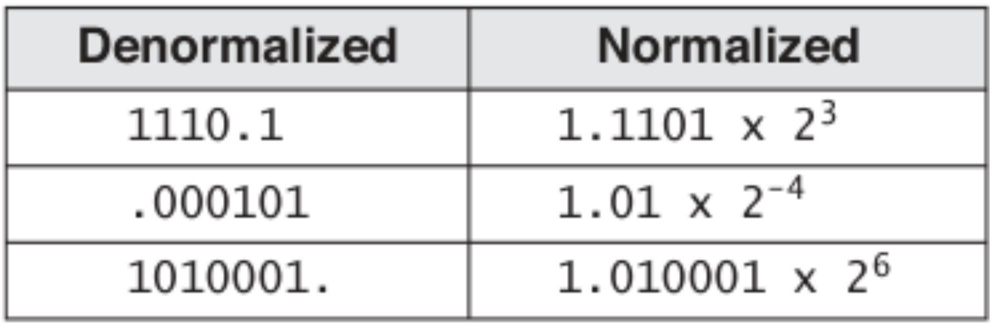
```
+---------------+-----------------+-----------------+-----------------------------+
| Sign bit (1 bit) | Exponent (8 bits) | Significand (23 bits) | Padding (0 bits) |
+---------------+-----------------+-----------------+-----------------------------+
```

The padding bits are zeros that are not used in the floating-point representation.

Value	Approximate range	Explanation
Normalized	2^-126 to 2^127	Range for typical non-zero values.
Denormalized	2^-149 to 2^-126	Range for very small values close to 0.
Special values	NaN, +/-Infinity	Represents Not-a-Number and Infinity.

Normalized numbers are the most common type of floating-point number. They have a significand that is between 1 and 2 (not including 2).

Denormalized numbers are numbers that have a significand that is less than 1. Denormalized values are used to represent very small numbers close to zero.



Denormalized	Normalized
1110.1	1.1101×2^{3}
.000101	1.01×2^{-4}
1010001.	1.010001×2^{6}

Special values are used to represent NaN (Not a Number) and infinity. **NaN (Not-a-Number)** represents undefined or unrepresentable values. **+/-Infinity** represents positive and negative infinity, often used to represent overflow and underflow situations in calculations.

The single-precision floating-point format is a widely used format for representing floating-point numbers. It is used in many different types of applications, including scientific computing, graphics, and gaming.

Sign

The sign bit of a floating-point number indicates whether the number is positive or negative. A sign bit of 1 means the number is negative, and a sign bit of 0 means the number is positive. Zero is considered positive.

Significand

The significand of a floating-point number is the fractional part of the number. It is represented by a sequence of digits to the left and right of the decimal point.

The digits to the left of the decimal point have positive exponents, and the digits to the right of the decimal point have negative exponents.

For example, the decimal floating-point number 123.154 can be represented as follows:

$$123.154 = (1 * 10^{2}) + (2 * 10^{1}) + (3 * 10^{0}) + (1 * 10^{-1}) + (5 * 10^{-2}) + (4 * 10^{-3})$$

The significand of this number is 123.154.

In binary floating-point numbers, the significand is represented by a sequence of 1s and 0s to the right of the binary point.

The first bit of the significand is always 1, and the remaining bits represent the fractional part of the number.

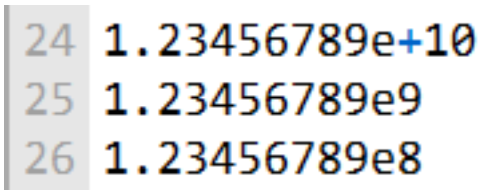
For example, the binary floating-point number 11.1011 can be represented as follows:

$$11.1011 = (1 * 2^{1}) + (1 * 2^{0}) + (1 * 2^{-1}) + (0 * 2^{-2}) + (1 * 2^{-3}) + (1 * 2^{-4})$$

The significand of this number is 1.1011.

Floating-point numbers can represent a very wide range of values, from very small numbers to very large numbers. This is because the significand and exponent can be adjusted to represent different values.

For example, the following floating-point numbers represent the same value:



```
1.23456789e+10
1.23456789e9
1.23456789e8
```

The significand is the same in all three numbers, but the exponent is different. The exponent indicates how many times the significand is multiplied by 2.

In the first number, the significand is multiplied by 2 ten times. In the second number, the significand is multiplied by 2 nine times.

And in the third number, the significand is multiplied by 2 eight times.

The precision of the significand is the number of bits that are used to represent the fractional part of the number.

In the IEEE double-precision format, the significand has 53 bits. This means that the significand can represent up to 53 bits of precision.

However, the entire continuum of real numbers cannot be represented in any floating-point format having a finite number of bits.

This is because there are an infinite number of real numbers, but only a finite number of bits can be used to represent a number.

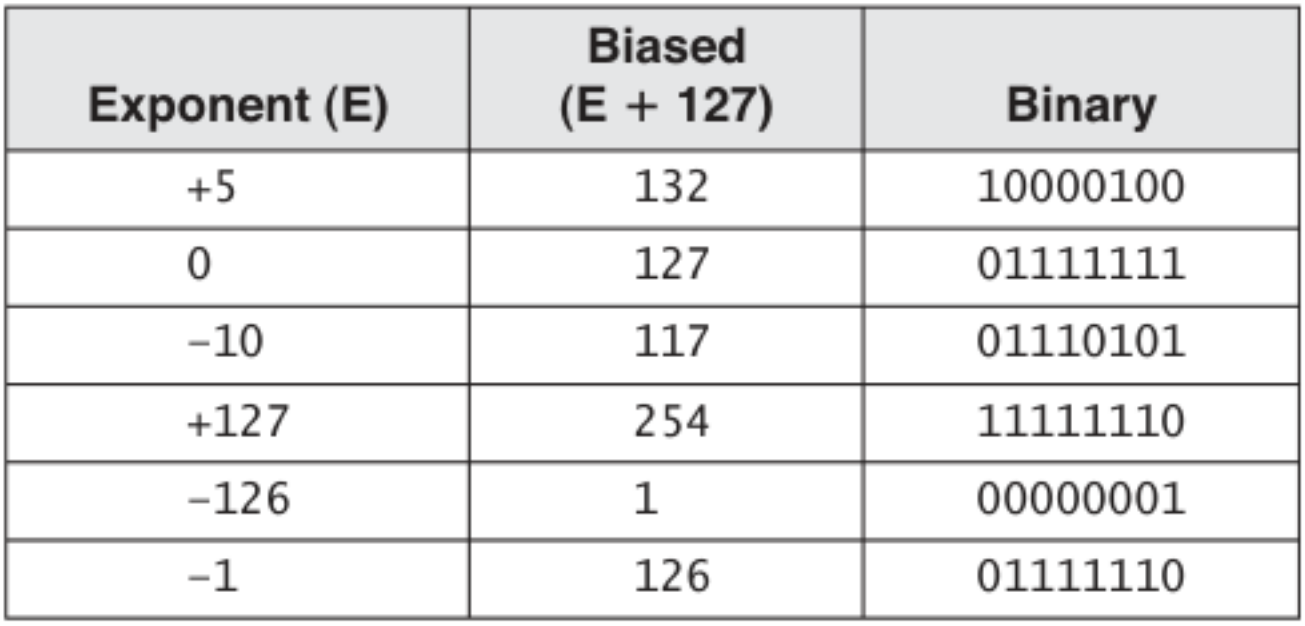
For example, the binary value 1.11111 requires a 54-bit significand to be represented exactly. However, the IEEE double-precision format only has a 53-bit significand.

This means that the number 1.11111 cannot be represented exactly in the IEEE double-precision format.

Exponents

The exponent in a floating-point number indicates how many times the significand is multiplied by 2. For example, the binary value 1.1011 * 2^5 means that the significand, 1.1011, is multiplied by 2 five times.

The biased exponent is always positive, between 1 and 254. The actual exponent range is from -126 to 127. The range was chosen so the smallest possible exponent's reciprocal cannot cause an overflow.



Exponent (E)	Biased (E + 127)	Binary
+5	132	10000100
0	127	01111111
–10	117	01110101
+127	254	11111110
–126	1	00000001
–1	126	01111110

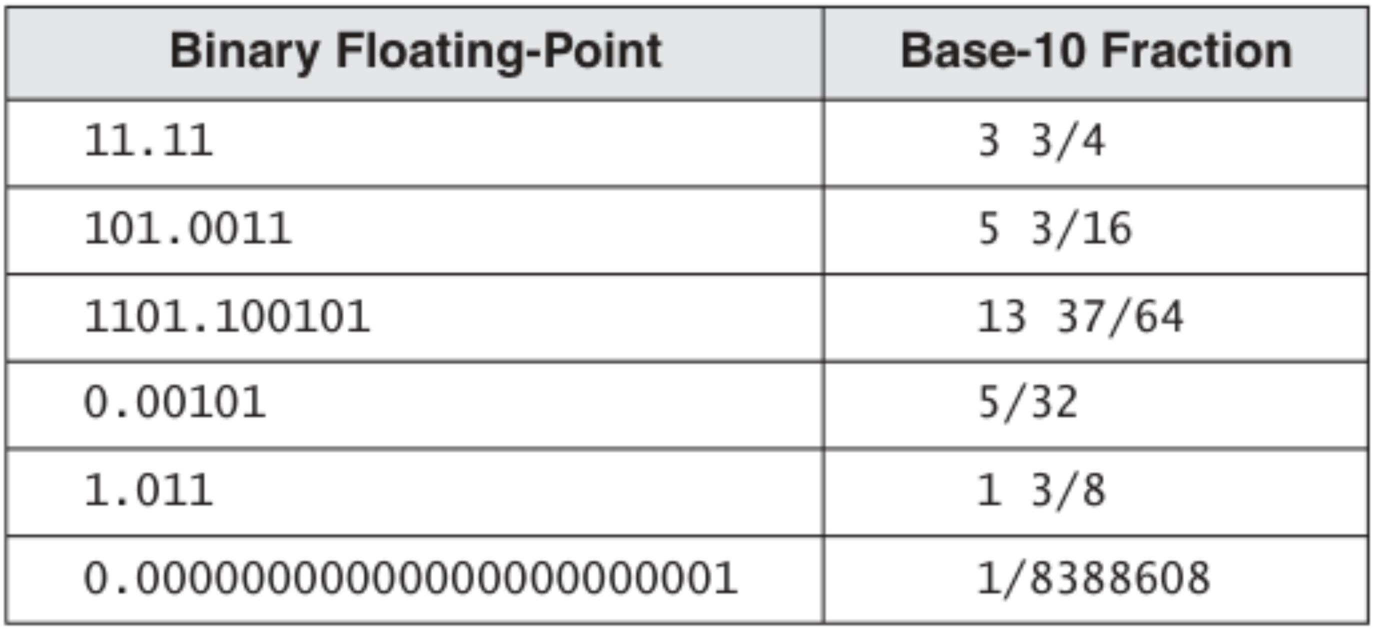
Single precision exponents are stored as 8-bit unsigned integers with a **bias of 127.** This means that the number's actual exponent must be added to 127 to get the biased exponent.

For example, the binary value 1.1011 * 2^5 has an actual exponent of 5. The biased exponent is therefore 127 + 5 = 132.

Translating Binary Floating-Point to Fractions

The table you sent shows examples of how to translate binary floating-point numbers to base-10 fractions.

The first column of the table shows the binary floating-point number. The second column shows the base-10 fraction. The third column shows the decimal value of the base-10 fraction.



Binary Floating-Point	Base-10 Fraction
11.11	3 3/4
101.0011	5 3/16
1101.100101	13 37/64
0.00101	5/32
1.011	1 3/8
0.00000000000000000000001	1/8388608

Here is an explanation of each row in the table:

Row 1:

Binary floating-point number: 11.11
Base-10 fraction: 3 3/4
Decimal value: 3.75

Row 2:

Binary floating-point number: 101.0011
Base-10 fraction: 5 3/16
Decimal value: 5.1875

Row 3:

Binary floating-point number: 1101.100101
Base-10 fraction: 13 37/64
Decimal value: 13.58125

Row 4:

Binary floating-point number: 0.00101
Base-10 fraction: 5/32
Decimal value: 0.15625

Row 5:

Binary floating-point number: 1.011
Base-10 fraction: 1 3/8
Decimal value: 1.375

Row 6:

Binary floating-point number: 0.00000000000000000000001
Base-10 fraction: 1/8388608
Decimal value: 0.00000011920928955078125

To translate a binary floating-point number to a base-10 fraction, we can use the following steps:

- Convert the significand to a decimal number.
- Multiply the significand by the base 2 raised to the power of the exponent.
- If the sign bit is 1, the result is negative. Otherwise, the result is positive.

For example, to translate the binary floating-point number 11.11 to a base-10 fraction, we would follow these steps:

- Convert the significand 1.11 to a decimal number. 1.11 = 1 + 1/2 + 1/4 = 7/4

- Multiply the significand by the base 2 raised to the power of the exponent. 7/4 * 2^1 = 7/2

- If the sign bit is 1, the result is negative. Otherwise, the result is positive. The sign bit is 0, so the result is positive.

- Therefore, the base-10 fraction of the binary floating-point number 11.11 is 7/2.

To calculate the value of a floating-point number, we simply multiply the significand by the base 2 raised to the power of the exponent. For example, the following floating-point number:

1.1011 * 2^3

The exponent in the floating-point number 1.1011 * 2^3 is 3. This means that the significand, 1.1011, is multiplied by 2 three times.

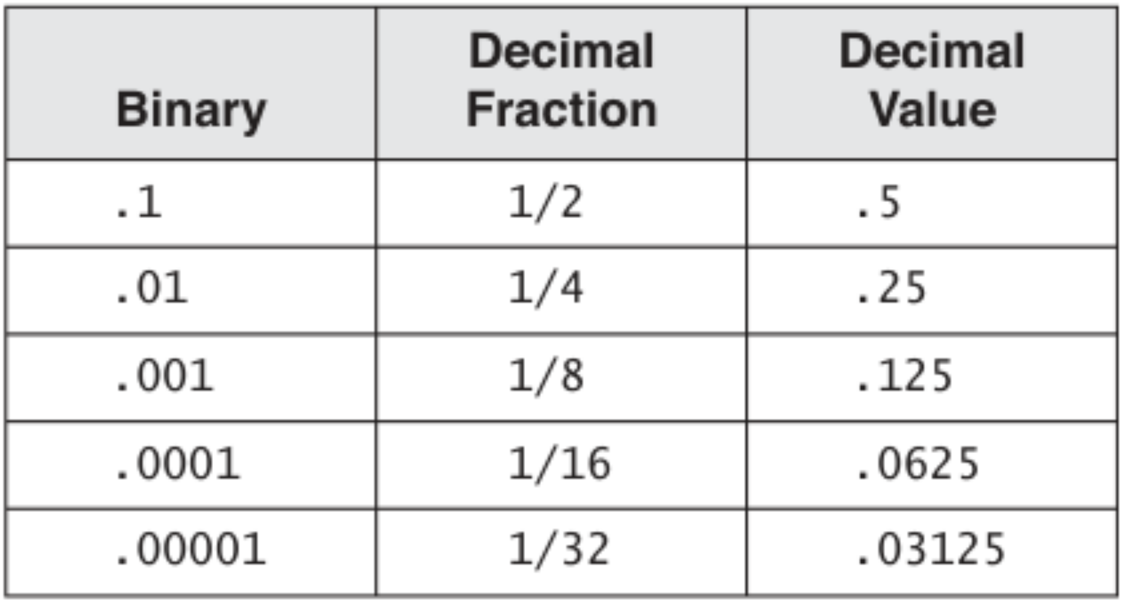
The exponent in a floating-point number indicates how many times the significand is multiplied by the base of the number system. In this case, the base is 2, because the number is in binary.

The exponent in this floating-point number was chosen so that the value of the number would be between 1 and 2.

If the exponent had been smaller, the value of the number would have been less than 1. If the exponent had been larger, the value of the number would have been greater than 2.

Floating-point numbers are used to represent a wide range of values, from very small numbers to very large numbers. The exponent allows us to do this by scaling the significand up or down.

<u>Let's discuss another table:</u>



Binary	Decimal Fraction	Decimal Value
.1	1/2	.5
.01	1/4	.25
.001	1/8	.125
.0001	1/16	.0625
.00001	1/32	.03125

To convert from binary to decimal, we multiply each digit of the binary fraction by its corresponding power of two, and then add the results together.

For example, to convert the binary fraction 0.1011 to decimal, we would do the following:

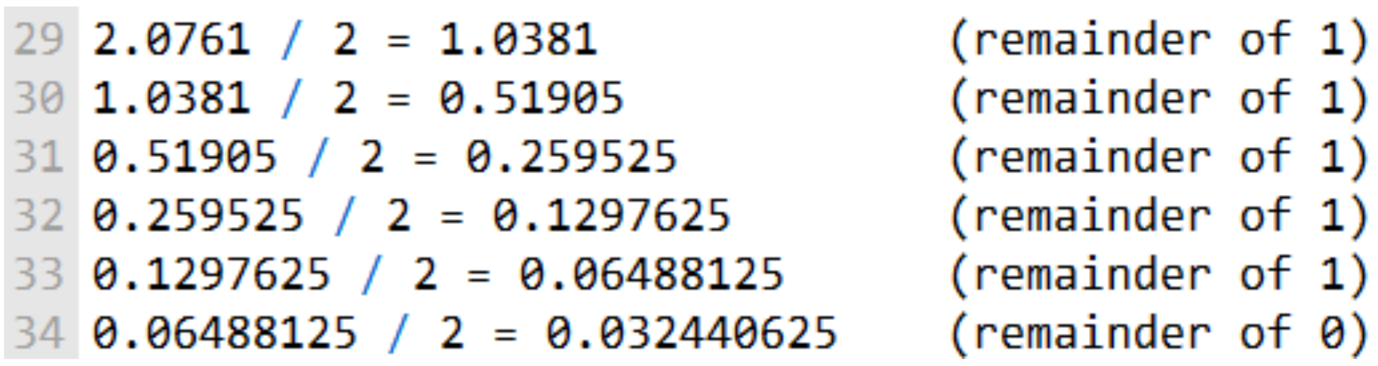
$$\mathbf{0.1011 * 2^{-3} + 0.1011 * 2^{-2} + 0.1011 * 2^{-1} + 0.1011 * 2^{-0}}$$

$$\mathbf{= 0.125 + 0.25 + 0.5 + 1.1011}$$

= 2.0761

To convert from decimal to binary, we repeatedly divide the decimal number by two, and write down the remainder at each step.

The remainders, in reverse order, give the binary representation of the number. For example, to convert the decimal number 2.0761 to binary, we would do the following:



```
2.0761 / 2 = 1.0381                  (remainder of 1)
1.0381 / 2 = 0.51905                 (remainder of 1)
0.51905 / 2 = 0.259525               (remainder of 1)
0.259525 / 2 = 0.1297625             (remainder of 1)
0.1297625 / 2 = 0.06488125           (remainder of 1)
0.06488125 / 2 = 0.032440625         (remainder of 0)
```

The binary representation of 2.0761 is therefore 1.0111.

Floating-point numbers can represent a much wider range of values by using more bits.