Exploring Binary

Nine Ways to Display a Floating-Point Number

By Rick Regan February 5th, 2015

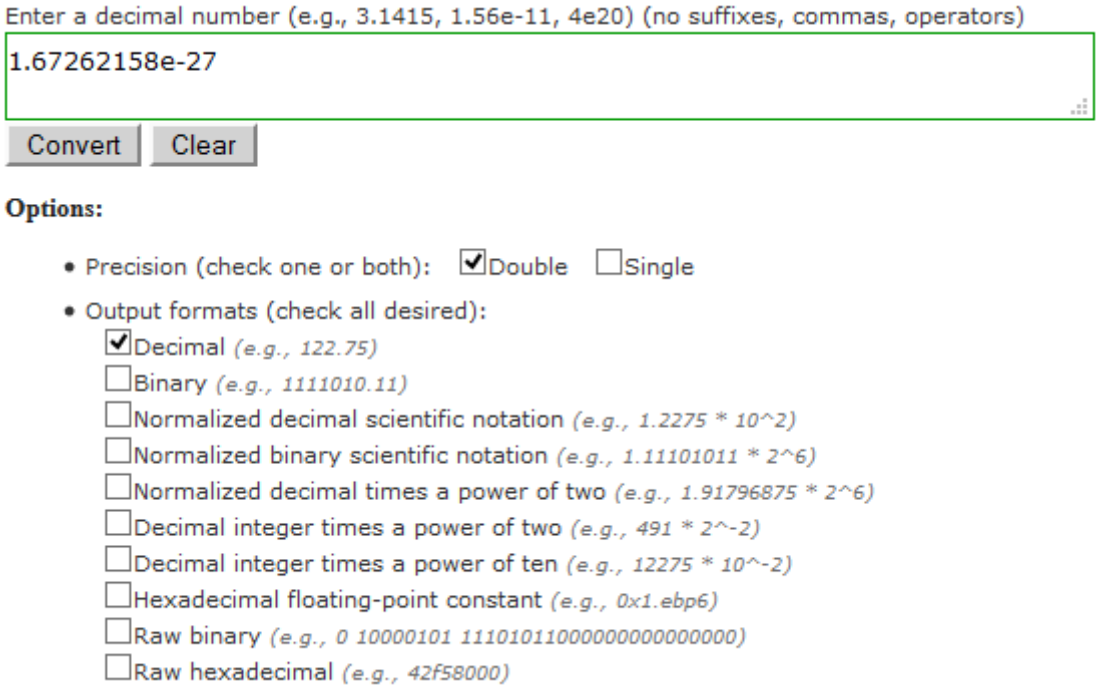
*(Updated June 22, 2015: added a **tenth** display form, "decimal integer times a power of ten".)*

In the strictest sense, converting a decimal number to binary floating-point means putting it in IEEE 754 format — a multi-byte structure composed of a sign field, an exponent field, and a significand field. Viewing it in this raw form (binary or hex) is useful, but there are other forms that are more enlightening.

I've written an online converter that takes a decimal number as input, converts it to floating-point, and then displays its exact floating-point value in ten forms (including the two raw IEEE forms). I will show examples of these forms in this article.

Example: 1.67262158e-27

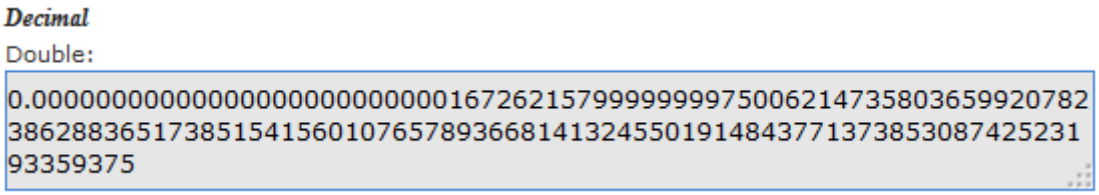
To demonstrate the ten forms of output I will use this constant, the mass of a proton in kilograms: 1.67262158e-27. (This is how it is defined in GSL, the GNU Scientific Library. But I've seen other definitions that differ in the least significant digits.) I will show its forms in double-precision floating-point.



Screenshot of 1.67262158e-27 as input to my floating-point converter (for decimal display)

1. Decimal

This form shows the floating-point number in decimal — *all its digits*:



Screenshot of 1.67262158e-27 displayed in decimal

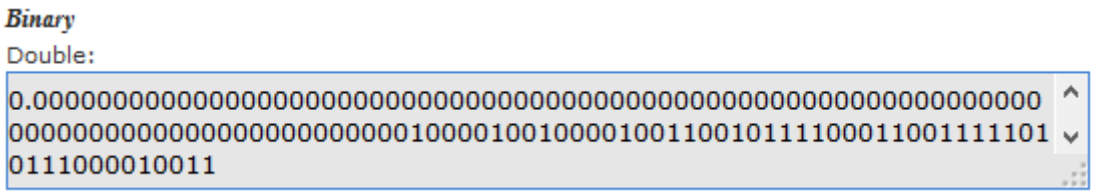
Most decimal numbers can't be represented exactly in binary floating-point, but every binary floating-point number *can* be represented exactly in decimal. By showing the number after its return trip to decimal, any "distortion" you see can be attributed to its conversion to floating-point.

The key to this form is that all decimal digits are shown. Programming languages, by default, do not do this; they round the decimal output to make it shorter. This is done for good reason, but it masks any inexactness; very often, the shortened number will match the original number (as it would in this example). Many a new programmer is tripped up by this.

As you can see, we entered a number with nine nonzero leading digits, but got one with many more (115) in return; the original number was not representable in floating-point.

2. Binary

This form shows the floating-point number in binary:

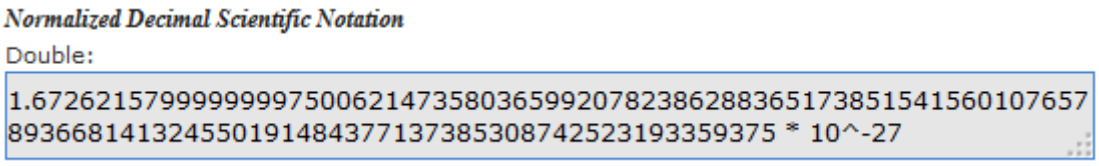


Screenshot of 1.67262158e-27 displayed in binary

Remember, this is the rounded number, in this case to 53 significant bits. It is the floating-point number written in binary, not the "pure" binary representation of the decimal input.

3. Normalized Decimal Scientific Notation

This form shows the floating-point number in decimal scientific notation (all of its nonzero leading decimal digits):

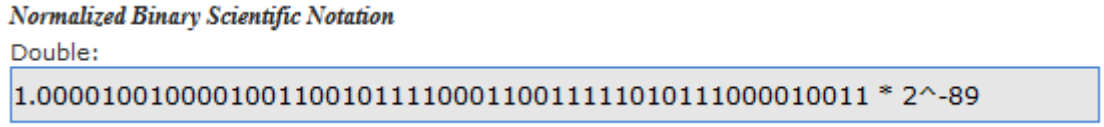


Screenshot of 1.67262158e-27 displayed in normalized decimal scientific notation

This form is useful when viewing small fractional values (as in this example), since leading zeros are omitted.

4. Normalized Binary Scientific Notation

This form shows the floating-point number in normalized binary scientific notation:

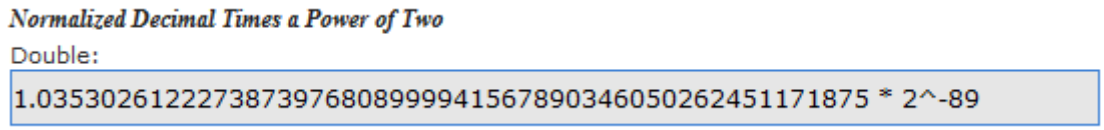


Screenshot of 1.67262158e-27 displayed in normalized binary scientific notation

This form is useful when you want to see the value of the floating-point number without the details of the IEEE 754 encoding.

5. Normalized Decimal Times a Power of Two

This form shows the floating-point number as a normalized decimal times a power of two:

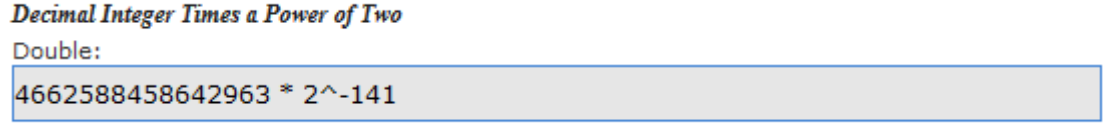


Screenshot of 1.67262158e-27 displayed as a normalized decimal times a power of two

This form is like binary scientific notation, except the significand of the floating-point number is shown in decimal. (If you convert the decimal significand to binary, it will match the binary significand bit-for-bit; the leading decimal digit is always 1, and each fractional decimal digit requires one bit.)

6. Decimal Integer Times a Power of Two

This form shows the floating-point number as a decimal integer times a power of two:



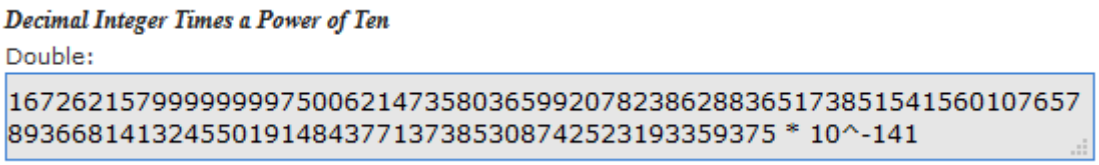
Screenshot of 1.67262158e-27 displayed as a decimal integer times a power of two

Sometimes it's useful to think of the significant bits as an integer (if the number is not an integer already), and that's where this form comes in. The decimal integer, when converted to binary, shows the exact bit pattern of the floating-point number, with trailing zeros subsumed by the power of two exponent.

This form shows that numbers with fractional parts become dyadic fractions in floating-point. Our example converts to $4662588458642963/2^{141}$.

7. Decimal Integer Times a Power of Ten

This form shows the floating-point number as a decimal integer times a power of ten:

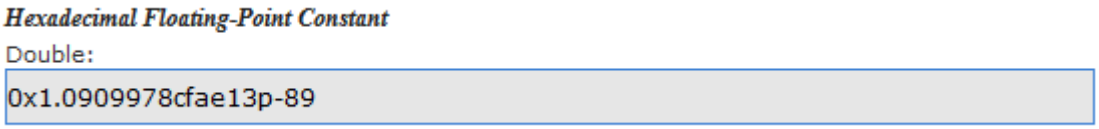


Screenshot of 1.67262158e-27 displayed as a decimal integer times a power of ten

In this form, you can display a number with a fractional part as a fraction.

8. Hexadecimal Floating-Point Constant

This form shows the floating-point number as a hexadecimal floating point constant:



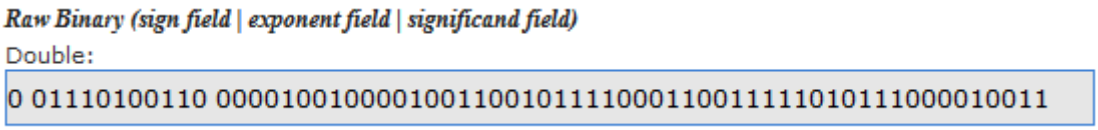
Screenshot of 1.67262158e-27 displayed as a hexadecimal floating-point constant

Hexadecimal floating point constants are used to enter correctly rounded floating-point numbers into computer programs. Decimal to floating-point conversions are done incorrectly in some implementations, so this is a way to ensure the exact value is obtained. (It is trivial to convert hex to binary.)

This form is just a level of abstraction above the raw IEEE forms. It will show subnormal numbers denormalized, with the maximum normal exponent.

9. Raw IEEE (Binary)

This form shows the floating-point number in its IEEE 754 format, in binary:

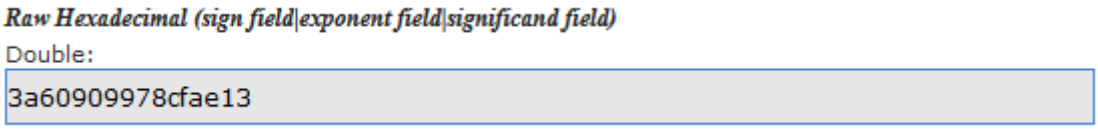


Screenshot of 1.67262158e-27 displayed in raw IEEE format (binary)

For our example double-precision number, the sign field is 0; the exponent field is the binary representation of 934, which is -89 (the actual exponent) + 1023 (the exponent bias); the significand field contains the trailing 52 significant bits (the leading bit is implied by normalization).

10. Raw IEEE (Hexadecimal)

This form shows the floating-point number in its IEEE format, in hexadecimal:



Screenshot of 1.67262158e-27 displayed in raw IEEE format (hexadecimal)

This is just the raw binary form mapped directly to hexadecimal.

People have been known to use this hexadecimal value in conjunction with type punning to initialize a floating-point variable. But initializing from a hexadecimal floating-point constant is the proper way, assuming your language supports it.

I view this as the least useful form of the ten.

Are There Other Forms?

Are there other forms you use? Please let me know.

EB

Get articles by RSS (What Is RSS?)

Get articles by e-mail

Numbers in computers / Convert to binary, Convert to decimal, Convert to hexadecimal, Decimals, Floating-point, Fractions

9 comments

1. **Adam**

   March 25, 2015 at 3:42 pm

   Thx for intresting article. I have found that form 1.6e-2 and 1.6*10^-2 are different from programmers point of view : first is a one value second is a expression. It can cause a problem because of order of operators.

2. **humpty**

   August 9, 2015 at 4:24 am

   decimal: Surely after 17 significant digits (binary64-IEEE 754), any following digits is just 'noise' ?
   Are the 18th,19th.. digits purely generated? I mean, they can't be stored anywhere.

---

3. **Rick Regan**

   August 9, 2015 at 10:18 am

   @humpty,

   Relative to the original decimal they are "noise" but those digits are really there. To see this easily, think of a decimal input that is exactly representable, like 2^{-1074}. That has 1074 places after the decimal point (751 significant digits), all of which are faithfully represented in a double.

---

4. **sean**

   October 10, 2016 at 10:51 am

   Can you clarify how you get 751 significant digits from 1074 places after the decimal point?

   Thanks.

---

5. **Rick Regan**

   October 11, 2016 at 11:15 am

   @sean,

   There are 323 leading zeros, which are not significant.

---

6. **Albert Chan**

   December 10, 2017 at 10:35 am

i suggest add 1 more way to represent double. (so make it 10)
i call it ulp representation
it return ulps away from 0
so, ulp representation = raw IEEE value for x \>= 0
for x \< 0, ulps = -(raw IEEE with top bit set to 0)

ulp (y – x) = ulp(y) – ulp(x)
ulp(x – (-x)) = 2 ulp(x)

---

7. **Rick Regan**

December 12, 2017 at 12:03 pm

@Albert,

What does this provide over representation 9, "Raw IEEE (Binary)"? Do you want the result in decimal?
Could you give some examples?

---

8. **macko**

November 18, 2018 at 1:32 pm

You did not mention the most popular notation. See how python, javascript, or java display a floating point number ("double" in C++) – i.e., they all don't display digits that would be unnecessary to identify the number when reading it back to the double (8 byte, for example) variable. See that 0.1 displays as 0.1, not 0.1000000000...

---

9. **Rick Regan**

November 18, 2018 at 2:19 pm

@macko,

Yes, you are referring to the shortest decimal string that round-trips. The ten forms in this article are those used by my floating-point converter, the intent of which is to show the underlying values. To quote what I wrote on the converter page: "in single-precision

floating-point, 0.1 becomes 0.100000001490116119384765625. If your program is printing 0.1, it is lying to you".

Comments are closed.