Review. Possibilities for binary representations of real numbers:

- fixed-point number: $\pm x.y$ with x and y of a certain number of bits
- floating-point number: $\pm 1.x \cdot 2^y$ with x and y of a certain number of bits
  IEEE 754, single precision: 32 bit (1 bit for sign, 23 bit for significand x, 8 bit for exponent y)
  IEEE 754, double precision: 64 bit (1 bit for sign, 52 bit for significand x, 11 bit for exponent y)

Example 16. Represent -0.375 as a single precision floating-point number according to IEEE 754.

Solution. $-0.375 = -\frac{3}{8} = -3 \cdot 2^{-3} = \boxed{-}\underbrace{1.\boxed{1}}_{\text{binary}} \cdot 2^{-2}$

The exponent -2 gets stored as $-2+127 = \boxed{0111,1101}$. (Recall that the bias $2^7-1=127$ is being added to the exponents. Also, it helps to keep in mind that $127 = (0111,1111)_2$.)

Overall, -0.375 is stored as $\boxed{1}\boxed{0111,1101}\boxed{1000,0000,\ldots}$.

Example 17. Represent $-1/7$ as a single precision floating-point number according to IEEE 754.

Solution. To write $1/7$ in binary, we repeatedly multiply with 2:

- $2 \cdot 1/7 = \boxed{0} + 2/7$
- $2 \cdot 2/7 = \boxed{0} + 4/7$
- $2 \cdot 4/7 = \boxed{1} + 1/7$ and now things repeat...

Hence, $1/7 = (0.001\cdots)_2$ and the final three digits 001 repeat: $1/7 = (0.001001001\cdots)_2$. Hence:
$-\frac{1}{7} = \boxed{-}\underbrace{1.\boxed{001001\ldots}}_{\text{binary}} \cdot 2^{-3}$

The exponent -3 gets stored as $-3+127 = \boxed{0111,1100}$.

Overall, $-1/7$ is stored as $\boxed{1}\boxed{0111,1100}\boxed{0010,0100,\ldots}$.

Example 18. What is the largest single precision floating-point number according to IEEE 754?

Technical detail. The exponent $(1111,1111)_2 = 2^8-1 = 255$ is reserved for special numbers (such as infinities or "NaN"). Hence, the largest exponent corresponds to $(1111,1110)_2 = 254$.

Solution. The largest single precision floating-point number is

$$\boxed{1}\boxed{1111,1110}\boxed{1111,1111,\ldots} = +1.111\ldots \cdot 2^{127} \approx 2^{128} = 2^{2^7} \approx 3.4 \cdot 10^{38}.$$

Here, we used that $(1111,1110)_2 = 2^8-2 = 254$ so that the actual exponent is $254-127=127$.

For comparison. The largest 32 bit (signed) integer is $2^{31}-1 \approx 2.1 \cdot 10^9$ (the exponent is $31 = 32-1$ to account for using 1 bit to store the sign). You might find this surprisingly small. And, indeed, 32 bit is not enough to address all memory locations in modern systems which is why the step to 64 bit was necessary.

Double precision. Likewise, the largest double precision floating-point number is

$$\boxed{1}\boxed{111,1111,1110}\boxed{1111,1111,\ldots} = +1.111\ldots \cdot 2^{1023} \approx 2^{1024} = 2^{2^{10}} \approx 1.8 \cdot 10^{308}.$$

On the other hand, the largest 64 bit (signed) integer is $2^{63}-1 \approx 9.2 \cdot 10^{18}$.

Armin Straub
straub@southalabama.edu

Interlude: Representing negative integers

In our discussion of IEEE 754, we have seen two ways of storing signed numbers using r bits:

- **sign-magnitude:** One bit is used for the sign, the other bits for the absolute value.
  Typically, the most significant bit is set to 0 for positive numbers and 1 for negative numbers.
  This is what happens in IEEE 754 for the most significant bit.
- **offset binary (or biased representation):** Instead of storing the signed number n, we store $n+b$ with b called the bias.
  Often the bias is chosen to be $b=2^{r-1}$.
  This is what happens in IEEE 754 for the exponent (however, the bias is chosen as $b=2^{r-1}-1$).

(Perhaps) surprisingly, neither of these is how signed integers are most commonly stored:

- **two's complement:** The most significant bit is counted as -2^{r-1} instead of as 2^{r-1}.
  Two's complement is used by nearly all modern CPUs.
  For instance, using 4 bits, the number 3 is stored as 0011 (2^1+2^0), while -3 is stored as 1101 ($-2^3+2^2+2^0$). Similarly, 5 is stored as 0101, while -5 is stored as 1011.
  To negate a number n in this representation, we invert all its bits and then add 1.
  As a result, adding a number to its negative produces all ones plus 1 (using r bits in the usual way, all ones corresponds to 2^r-1 so that adding 1 results in 2^r; which is where the name comes from).
  Important. At the level of bits, addition in the two's complement representation works exactly as addition of unsigned integers. For instance, consider the addition $0010+1011=1101$: interpreted as unsigned integers, this is $2+11=13$; alternatively, interpreted as signed integers (using two's complement), this is $2+(-5)=-3$. Likewise, the same is true for multiplication.
  Advanced comment. Two's complement makes particular sense when we interpret it in terms of modular arithmetic (ignore this comment if you are not familiar with this). Namely, if using r bits, all numbers are interpreted as residues modulo 2^r (recall that -1 and 2^r-1 represent the same residue; similarly, 2^{r-1} and -2^{r-1} are the same modulo 2^r). Instead of representing the residues by $0,1,2,\ldots,2^r-1$, we then make the choice to represent them by $0,1,2,\ldots,-3,-2,-1$. Since we can compute with residues as with ordinary numbers, this explains why, using two's complement, addition and multiplication work the same as if the numbers are interpreted as unsigned (assuming that no overflow occurs).
  Comment. Two's complement can also be interpreted as follows: instead of storing the signed number n, we store $n+2^r$, where we only use the r least significant bits (thus throwing away the bit with value 2^r). Apart from this last bit (pun intended!), this is like offset binary.

There are yet further possibilities that are used in practice, most notably:

- **ones' complement:** A positive number n is stored as usual with the most significant bit set to 0, while its negative $-n$ is stored by inverting all bits.
  For instance, using 4 bits, the number 5 is stored as 0101, while -5 is stored as 1010.
  As a result, adding a number to its negative produces all ones (hence the name).

https://en.wikipedia.org/wiki/Signed_number_representations

Example 19. Which of the above four representations has more than one representation of 0?

Solution. In the sign-magnitude as well as in the ones' complement representation, we have a $+0$ and a -0.

Example 20. Express -25 in binary using the two's complement representation with 6 bits.

Solution. Since $25=(011001)_2$, -25 is represented by 100111 (invert all bits, then add 1).
Alternatively, note that $-25=-2^5+7$ and $7=(111)_2$ to arrive at the same representation.
As another alternative, we store $-25+2^6=7$, resulting in the same representation.