# Lecture 2. Floating Point Representation

In computation, we will use base 2.

## Half-Precision

16-bit precision (IEEE standard, “half-precision”) is represented this way (see <https://en.wikipedia.org/wiki/Half-precision_floating-point_format> ) :

* One bit is used for the sign, where 1 == negative
* Five bits are used for the exponent. “00000” is reserved to mean 0, and “11111” is reserved for (infinity), leaving 30 exponents which can be represented ().
* This leaves 11 bits for the significand.

Eleven digits for the significand? Numbers are given a normalized representation, so that they are (mostly) unique. Since this is base 2, that means the first digit in any numeric representation is a “1”. So we only need to store 10 digits.

In the IEEE standard, exponents are not represented in literal base 2, they are stored with an offset. So the smallest exponent “00001” (a decimal 1 represented in base 2) is (minus 14 base 10), and the largest exponent, “11110” (a literal ) is . So exponents are stored with an offset of . This means the exponent 0 is represented as “01111”.

The smallest positive number which can be represented in half precision is therefore

* “0 00001 0000000001” (the smallest possible significand with the largest possible negative exponent will be the closest we can get to zero). If our numbers are not normalized this is .
* If our numbers are normalized, as in the IEEE standard, the smallest positive number is which would be represented “0 00001 0000000000”. (Keep in mind the implied leading “1” in the significand.)
* The smallest positive 16 representation can be displayed in Julia\_v1 with
  + floatmin(Float16) for a decimal representation
  + bitstring(floatmin(Float16)) for a binary representation

The largest normalized non-infinite number is just . In Julia\_v1 try

* floatmax(Float16)
* bitstring(floatmax(Float16)) to see “0 11110 1111111111”

## Single Precision

A 32-bit representation is “single precision” (four byte floats). (See <https://en.wikipedia.org/wiki/Single-precision_floating-point_format> .) Here

* 1 digit is given to the sign
* 8 digits are given to the exponent
* 23 digits are stored to represent a normalized 24 digit significand

Similar to half precision floating point (fp) numbers, the exponents “00000000” and “11111111” are reserved for zero and infinity, respectively. The remaining binary integers represent the signed exponents from -126 to 127, so exponents can be thought of as having an offset of . In other words, the digits “00000001” can be understood as an exponent as .

Zero is represented twice, once as 0, once as -0.

The smallest possible positive value is in base 10, represented by “0 00000001 00000000000000000000000” (remember the silent leading “1”). Were our representation not normalized the smallest possible positive value would be “0 00000001 00000000000000000000001” or .

The largest possible representable normalized number, an integer, is . This is “0 11111110 11111111111111111111111”.

## Double Precision

See <https://en.wikipedia.org/wiki/Double-precision_floating-point_format> ,

This representation uses

* 1 bit sign
* 11 bits as exponent
* 52 stored bits as a 53 bit normalized significand

There are possible exponents (two being reserved for zero and infinity). The binary integers are offset by 1023, to represent to 1023.

The smallest positive number in this representation is . This is the number reported by Julia\_v1 from floatmin(Float64).

The smallest non-normalized (subnormal) number is . This is the number reported by Julia\_v1 from floatnext(Float64(0.0)) .

The largest number in this representation is . This is the number reported by Julia\_v1 from floatmax(Float64). A line of code that verifies this is (2.0^53-1)\*2.0^(1023-52)==floatmax(Float64). (Note here that it is important to represent the 2 as “2.0” for the calculation to be reported with floating point numbers rather than integers!)