There is a standard that describes how real values should be stored in a computer. This standard also indicates the results when two values are added or multiplied. This seems obvious, but remember that most real values cannot be represented exactly in a computer, so the result of adding two numbers might be one of several possibilities that are approximations to the sum; which possibility is selected depends upon the rounding mode.

The use of the standard by most modern computers means that computations should be much more similar when performed on different computer systems.

This standard is often called the IEEE arithmetic standard, because it was first published as IEEE 754 (IEEE is the Institute of Electrical and Electronics Engineers). It is currently published by the International Standards Organization as IEC 60559 (1989-01), *Binary floating-point arithmetic for microprocessor systems*.

### 7.1 Numerical Representations

All values are stored in a computer as a finite collection of binary digits (bits). This means that most real values cannot be represented exactly, as discussed briefly in 1.7. It also means there is a limit to the size of numbers, both real and integer, that can be represented as values of a Fortran variable.

#### Representations of Integers

Integers (whole numbers) are represented as a string of bits. The number of bits is usually a power of two, such as 4, 8, 16, 32, or 64. For most Fortran systems, default integers are stored using 32 bits. The left bit is usually reserved for the sign: 0 for positive and 1 for negative. Using a 4-bit representation as an example, the integers 0, 1, 2, and 3 would be represented as 0000, 0001, 0010, and 0011, respectively. To understand how negative numbers are represented, think of a car odometer going backward. If the odometer is 0000 and the representation is binary, going backward one yields 1111, which is the representation of $-1$. 1110 is the representation of $-2$, and so forth. Other schemes may be used to represent negative integers, but this one is the most common.

With 32 bits, the largest positive number that can be represented is a 0 followed by thirty-one 1s, which represents the integer $2^{31} - 1$ or 2,147,483,647. This is the value of the intrinsic function `huge(0)`.
Representations of Reals

To understand how real values are stored, think of them as being written in exponential form, except that the base is 2. Thus a number can be written as \( f \times 2^e \). The number is stored in three parts: the sign, the exponent \( e \), and the fraction \( f \). In IEEE standard arithmetic, the sign uses 1 bit, the exponent uses 8 bits, and the fraction uses 23 bits. Without worrying about the details, the result is that real numbers may be stored with approximately 6 decimal digits of precision and the largest value is \texttt{huge(0.0)}\), which is approximately \( 10^{38} \).

Exercises

1. Use the intrinsic function \texttt{huge} to compute and print the largest value of each of the integer kinds available on your system.

2. Use the intrinsic functions \texttt{huge} and \texttt{precision} to compute and print the largest double precision value and the precision of double precision values on your system.

3. Investigate the intrinsic functions \texttt{range}, \texttt{radix}, and \texttt{digits} by using them in a program. Determine which apply to reals and which to integers. Appendix A may contain some information of interest.

7.2 NaN and Inf

One of the interesting things about the IEEE system of arithmetic is that it includes representations for "values" that are not ordinary numbers. For example, in ordinary arithmetic, the value of \( 7.7/0.0 \) is not defined. However, \( 7.7/x \) gets larger and larger as \( x \) gets closer and closer to 0.0. This is the basis for saying that the result of \( 7.7/0.0 \) is infinity, or \texttt{Inf}. Similarly, \(-7.7/0.0\) is negative infinity or \texttt{-Inf}.

```plaintext
program inf
  implicit none
  print *, 7.7/0.0, -7.7/0.0
end program inf

+Inf -Inf
```

There is no similar argument that will produce an answer for \( 0.0/0.0 \). \( 0.0/x \) is always 0.0, except when \( x \) is 0, so 0 might be a reasonable value. However, \( x/0.0 \) is infinity unless \( x \) is 0. On the other hand, \( x/x \) is always 1.0 unless \( x \) is 0.0. Thus \( 0.0/0.0 \) is considered to be "Not a Number" or NaN.