Contents

1 Report 4

1.1 The Base Ten System

The number system that we are used to dealing with in everyday life is based on powers of 10. In this system, an integer is represented as a sequence of digits from 0 to 9:

$$\ldots d_2 d_1 d_0$$

In the base 10 system, the value of a number is based on its position in the sequence. The right-most digit d_0 indicates ones (or 10^0), the next digit d_0 indicates tens (or 10^1), the next digit d_2 indicates hundreds (or 10^2). So, the number $d_2 d_1 d_0$ means $10^2 \cdot d_2 + 10^1 \cdot d_1 + 10^0 \cdot d_0$. This system allows us to represent integers as large as we like by adding increasing powers of ten to the left.

We can extend this system to include numbers less than 1 by the addition of a decimal point:

$$\ldots d_2 d_1 d_0 . d_{-1} d_{-2} \ldots$$

The numbers to the right of the decimal point indicate negative powers of ten. The decimal part of this number therefore means $10^{-1} \cdot d_{-1} + 10^{-2} \cdot d_{-2} + \ldots$.

Putting the decimal part together with the integer part to the left of the decimal point, we have:

$$\begin{aligned} \ldots d_2 d_1 d_0 . d_{-1} d_{-2} \ldots &= \ldots + 10^2 \cdot d_2 + 10^1 \cdot d_1 + 10^0 \cdot d_0 + 10^{-1} \cdot d_{-1} + 10^{-2} \cdot d_{-2} + \ldots \\ &= \ldots + 100 d_2 + 10 d_1 + d_0 + \frac{d_{-1}}{10} + \frac{d_{-2}}{100} + \ldots \end{aligned}$$

The dots ... here indicate that we can make the number as large as we like by adding digits to the left, and as precise as we like by adding digits to the right.

1.2 Binary Numbers

A *binary number* is based instead on powers of 2. As in the base 10 system, the value of a number is also based on its position in the sequence, and the powers used are powers of 2 rather than powers of 10. In the base 2 system, the only digits are 0 and 1. A number in this system is represented as a sequence of 1's and 0's:

$$\ldots b_2 b_1 b_0 . b_{-1} b_{-2} \ldots$$

The digits to the left of the point again represent the integer part of the number, and those to the right represent the fractional part.

$$\begin{aligned}\ldots b_2 b_1 b_0 . b_{-1} b_{-2} \ldots &= \ldots + 2^2 \cdot b_2 + 2^1 \cdot b_1 + 2^0 \cdot b_0 + 2^{-1} \cdot b_{-1} + 2^{-2} \cdot b_{-2} + \ldots \\ &= \ldots + 4b_2 + 2b_1 + b_0 + \frac{b_{-1}}{2} + \frac{b_{-2}}{4} + \ldots\end{aligned}$$

An example of a binary integer is the number $(1011)_2$. To convert this to base 10, we add up the powers of 2:

$$(1011)_2 = 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = 8 + 2 + 1 = (11)_{10}$$

The fractional part to the right of the point is dealt with in the same way, but with negative powers of 2. For example:

$$(.1101)_2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = \left(\frac{13}{16}\right)_{10} = (0.8125)_{10}$$

1.3 Scientific Notation in Base Ten

Scientific notation allows us to represent very large and very small numbers without needing to write a decimal with a large number of zeros before or after the decimal point. Instead, we write a number as a decimal between 1 and 10, multiplied by a power of 10.

To convert a number to scientific notation, the decimal point is shifted to be immediately to the right of the leading digit. The number of spaces we need to move the decimal point tell us how large the power of 10 has to be.

- If the decimal point is moved to the left, we have a positive power of 10.
- If the decimal point is moved to the right, we have a negative power of 10.

Some examples are:

$1234 = 1.234 \times 10^3$	decimal point moved 3 places to the left
$0.0000412 = 4.12 \times 10^{-5}$	decimal point moved 5 places to the right
$456.789 = 4.56789 \times 10^2$	decimal point moved 2 places to the left

1.4 Scientific Notation in the Binary (Base 2) System

Scientific notation works equally well in the binary (base 2) system. Just as in the base 10 system, we move the point to be immediately to the right of the leading digit, but in this case we multiply by a power of 2. Some examples are:

$1011 = 1.011 \times 2^3$	point moved 3 places to the left
$0.00000110101 = 1.10101 \times 2^{-6}$	point moved 6 places to the right
$11001.11 = 1.100111 \times 2^4$	point moved 4 places to the left

1.5 Floating Point Representation

The IEEE 754 standard defines a way to store a binary number on a computer. The standard is built on the following foundation:

- Every number is stored using the same, fixed number of binary digits, or bits (zeros and ones).
- The number of bits used depends on the computer and operating system used, but is usually either 32 or 64.
- Numbers are stored in binary scientific notation, as defined in the previous section.
- The power of 2 is referred to as the **exponent**, and the digits after the point as the **mantissa**.
- A fixed number of bits (E) are set aside to store the exponent, and a fixed number (M) to store the mantissa.
- A single bit - the **sign** bit - is used to indicate if the number stored is positive or negative. A sign bit of 0 represents a positive number, and a sign bit of 1 represents a negative number.

1.5.1 The Bias

The number stored in the bits set aside for the exponent represents a binary integer. As such, the minimum exponent value consists of all zeros, corresponding to the binary integer zero. This is not particularly useful, as it means that the smallest number we can store is 2^0 i.e. 1.0.

We would like however to be able to store both very large numbers (with a large positive exponent) and very small numbers (with a large negative exponent). To ovecome this problem, when a number in binary scientific notation is stored, a fixed integer (called the **bias**) is added to the actual exponent before it is stored.

For example, if we wanted to store the binary number 2^{-100}, we would add the bias to -100 before it is stored in the exponent. The exponent stored would therefore be -100 + bias, rather than -100. When the stored exponent is converted back to the actual exponent, the bias therefore needs to be subtracted to get back to where we started.

Putting this together, we have a formula for converting the computer representation of a number back into an actual number in binary scientific notation:

$$(-1)^s \times (1.\textbf{mantissa}) \times 2^{\textbf{exponent}-\textbf{bias}}$$

The letter 's' here denotes the value of the sign bit. Some examples are:

Actual Number		Stored Number		
Binary number	Scientific Notation	Sign	Exponent	Mantissa
1011	1.011×2^3	0	3 + bias	011 ··· 000
0.00000110101	1.10101×2^{-6}	0	-6 + bias	10101 ··· 000
11001.11	1.100111×2^4	0	4 + bias	100111 ··· 000

1.5.2 Denormalized Numbers

A last problem to be solved is how to represent the number zero in this system. The assumption that a '1.' is always present before the mantissa bits doesn't work, since the number zero doesn't start with a '1'.

To solve this problem, a stored exponent of all zeros is treated differently from all other exponents. For the case when there is a stored exponent of all zeros:

- The assumption that the mantissa is preceded by a '1.' is dropped, and we assume that the mantissa is preceded by a '0.' instead.
- The actual exponent is taken to be 1 - bias.

Such a number is referred to as **denormalized**. This gives rise to a separate formula for converting the computer representation of a denormalized number back into an actual number in binary scientific notation:

$$(-1)^s \times (0.\textbf{mantissa}) \times 2^{1-\textbf{bias}}$$

1.6 What's Required for Report 4

The second part of Report 4 involves conducting experiments in Python to determine how many bits are used for the exponent (E) and mantissa (M), and what the value of the bias (B) is.

I would recommend writing the following functions to do this:

- **find_epsilon** finds the smallest number larger than 1 that can be stored.
- **find_largest** finds the largest floating point number that can be stored.
- **find_smallest** finds the smallest floating point number that can be stored.

The technique for finding each of these numbers is essentially the same: choose a power of 2, and increase (or decrease) it until the largest (or smallest) number that can be represented in Python is found. The algorithms to do this will follow that described in class for finding the machine epsilon.

Separate marks will be given for finding each of these numbers, for inferring the values of M, E, and B from them, and for determining if denormalization is used. The main thing is to justify the approach taken and reasoning used. Marks will also be given in this report for clarity of presentation, such as appropriate division into sections, use of headings, and visual clarity of design.