Introduction to Numerical Analysis I
Handout 2

1 Computer Representation of Numbers

1.1 The integers

An integer can be expressed as (w.l.o.g., $n > 0$)

$$n = \sum_{k=0}^{p} s_k b^k,$$

where $0 \leq s_k < b$ are digits.

- The common case is $b = 10$, i.e. decimal (decimal-base) representation.
- On computers numbers are represented in binary base, $b = 2$,
- the other useful bases are octal ($b = 8$), and
- the hexadecimal ($b = 16$).

To reduce ambiguity, the basis b is added in a subscript of the number, e.g. $10_{16} = 16_{10} = 18_8$.

1.1.1 Basis change

			$b=2$			
$b=8$	$b=10$	$b=16$	s_3	s_2	s_1	s_0
0	0	0	0	0	0	0
1	1	1	0	0	0	1
2	2	2	0	0	1	0
3	3	3	0	0	1	1
4	4	4	0	1	0	0
5	5	5	0	1	0	1
6	6	6	0	1	1	0
7	7	7	0	1	1	1
10	8	8	1	0	0	0
11	9	9	1	0	0	1
12	10	A	1	0	1	0
13	11	B	1	0	1	1
14	12	C	1	1	0	0
15	13	D	1	1	0	1
16	14	E	1	1	1	0
17	15	F	1	1	1	1

Table 1: Octal, Decimal, Hexadecimal and Binary numbers

- **From binary to octal**: Separate the binary number into triples sing a space or a comma (add leading zeroes if required), then and convert the value of the tuple using the Table 1 to the octal base.
- **From binary to hexadecimal**: Separate the binary number into quadruples using a space or a comma (add leading zeroes if required), then convert the value of the tuple using the Table 1 to the hexadecimal base.
- **From the octal/hexadecimal to the binary basis**: Similar to the above (just an opposite direction).
- **From octal to hexadecimal (or vise verse)** Change from octal to binary then from binary to hexadecimal (or vise verse).

Algorithm 1: Change from base-10 to base-2 for integers

Input: decimal integer n
Output: binary number $s_p s_{p-1} ... s_1 s_0$
j=0
while $n \neq 0$ **do**
  $s_j = ReminderOf(n/2)$ // $s_j \in \{0, 1\}$
  $n = IntegralPartOf(n/2)$
  j=j+1
end

1.2 The Real Numbers

A real number can be expressed in binary basis as (w.l.o.g., consider $0 < f < 1$, i.e. $f = 0.s_1...s_p$)

$$f = \sum_{k=1}^{p} s_k 2^{-k}$$

Example 1.1. Assume $m = 5$. The smallest fraction under this assumption is $\varepsilon = 0.00001_2$. Let us look into the previous and the next numbers to the number $0.11001_2 = \frac{25}{32}$ from example **??**, by adding and subtracting ε to it:

$$0.11001_2 + \varepsilon = 0.11010_2 =$$
$$= 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} = \tfrac{13}{15}$$

$$0.11001_2 - \varepsilon = 0.11000_2 =$$
$$= 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} = \tfrac{3}{4}$$

Note: we found that for $m = 5$, the fraction $\frac{25}{32}$ represents all real numbers in interval $\left(\frac{3}{4}, \frac{13}{15}\right)$. *Thus, we figured that not all real numbers can be represented on computers, therefore they are approximated.*

1.2.1 Basis change

Algorithm 2: Change from base-10 to base-2 for real numbers in $[0, 1)$

Input: real number $\alpha \in [0, 1)$, basis 2
Output: binary number $0.s_0 s_1 ... s_{p-1} s_p$
j=1
while $\alpha \neq 0$ *and* $j <= m$ /* num of bits */ **do**
  $s_j = IntegralPartOf(2 \cdot \alpha)$ // $s_j \in [0, 2)$
  $\alpha = ReminderOf(2 \cdot \alpha)$
  j=j+1
end

Example 1.2. In Table 2 one see example of translation of 0.1_{10} into binary basis (for $m = 5$, the resulting number reads from top to bottom of the integral part column: 0.00011. **Note**: A number, with

2α	Remainder	Integral part
0.1 * 2	0.2	0
0.2 * 2	0.4	0
0.4 * 2	0.8	0
0.8 * 2	0.6	1
0.6 * 2	0.2	1

Table 2: Example of changing from decimal to binary for real numbers

finite decimal representation have infinite representation in binary. Another reason for approximation of real numbers.

1.3 The Floating Point Representation

The standard representation of real numbers is floating-point representation. This representation is closely related to so called scientific notation, e.g.

$$+6132.789 = +0.6132789e4 = +0.6132789 \times 10^4$$

but in binary basis, i.e. the exponential part would be powers of 2. Thus, the real number in Floating Point representation would be given by

$$r = (-1)^{sign} \cdot 2^e \cdot \mathbf{M}$$

where the exponent $-2^n - 2 < e < 2^n - 1$ is a signed number (n is number of bits) and the mantissa $\mathbf{M}$ represent the most significant digits of r. The range of mantissa is

$$\frac{1}{2} \leq \mathbf{M} < 1.$$

The lower bound mean that the most significant bit will be 1, which is designed to increase the accuracy, since more bits can be used for significant digits. For example $(1.s_1 s_2 ...)_2 \times 2^e$ instead of $(0.01 s_1 s_2 ...)_2 \times 2^e + 2$. Note that the part of '1.' doesn't have to use any space in memory, it could be implied.

The standard today is 32 or 64 bits in the following form

Precision	Total	sign	e	range of e	**M**
single	32	1	8	$-126 : 127$	23
double	64	1	11	$-1022 : 1023$	52

Example 1.3. In this example we see several numbers represented in 64 bit double precision floating point representation:
$\frac{8}{3} = (-1)^0 \cdot 2^2 \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{32} + ... + \frac{1}{2^{52}}\right) =$

$$0|\overbrace{0|0...010}^{11\,bit}|\overbrace{(1.)\,01...0101}^{52\,bit}$$

The smallest (positive) number:

$$0|\overbrace{1|1...11}^{11\,bit}|\overbrace{(1.)\,00...0}^{52\,bit} = +2^{-1022}.\frac{1}{2} = 2^{-1023} \approx 10^{-307}$$

The biggest number: $0|\overbrace{0|1...11}^{11\,bit}|\overbrace{(1.)\,11...11}^{52\,bit} =$

$$2^{1023} \sum_{n=1}^{52} 2^{-n} = 2^{1023}\left(\frac{2^{-52} - 1}{1/2 - 1} - 1\right) = 2^{1023} \approx 10^{307}$$

Definition 1.4. (Overflow & Underflow)
The term **underflow** is a condition where the result of a calculation is a smaller number than the computer can actually store in memory.

The term **overflow** is a condition where the result of a calculation is a greater number than the computer can actually store in memory.

Example 1.5.

- The overflow is easy to show in integers, for example if the numbers are represented by $m = 5$ bits, then if one adds the biggest number to another (even the smaller one) the result will be $11111_2 + 00001_2 = 1|00000_2$. The computer will end with 00000_2 and 'carry' flag.
- In example 1.3 we wound that on a computer with double precision the smallest number is $\nu = 2^{-1023}$, the underflow happens for $\nu/2$.

  Note: You may not see the underflow using the example above. This is due to the **subnormal numbers** representation. The trick is in violating the rule of "no leading zeros" in mantissa. This way, the significant digits lost gradually.