Tutorial 3
Floating-Point Numbers

Exercise 1

Convert the numbers below into their **single-precision** IEEE-754 representations. Write down the final result in its **binary form**.

1. 142
2. -64.25
3. 30.625
4. 0.3125

Exercise 2 (Homework)

Convert the numbers below into their **double-precision** IEEE-754 representations. Write down the final result in its **binary form**.

1. -256
2. 964
3. 8.19140625
4. 0.2890625

Exercise 3

Convert the **single-precision** IEEE-754 words below into their associated representations. If a representation is a number, use the base-10 following form: $k \times 2^n$ where k and n are integers (either positive or negative).

1. $1011\ 1101\ 0010\ 0000\ 0000\ 0000\ 0000\ 0000_2$
2. $0100\ 1111\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000_2$
3. $1100\ 1100\ 0101\ 0011\ 0000\ 0000\ 0000\ 0000_2$
4. $1111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
5. $0000\ 0000\ 0011\ 0000\ 0000\ 0000\ 0000\ 0000_2$

Exercise 4 (Homework)

Convert the **double-precision** IEEE-754 words below into their associated representations. If a representation is a number, use the base-10 following form: $k \times 2^n$ where k and n are integers (either positive or negative).

1. $417C\ C800\ 0000\ 0000_{16}$
2. $C808\ 0000\ 0000\ 0000_{16}$
3. $000A\ 0000\ 0000\ 0000_{16}$
4. $8000\ 0000\ 0000\ 0000_{16}$
5. $FFF0\ 0001\ 0000\ 0000_{16}$

Exercise 5

Assuming that the mantissa is **normalized**, answer the following questions for both **single- and double-precision** formats.

1. Calculate the smallest and largest absolute values of a floating-point number.
2. What is the smallest number (greater than 0) which, when added to 1, gives a different result from 1?

Exercise 6

Let us consider the following program written in the C language:

```c
#include <stdio.h>          // Required to call the printf() function.

int main()
{
    float f1, f2, f3, r;    // IEEE754 single precision for f1, f2, f3 and r.

    f1 = 1E25;              // 10^25 -> f1
    f2 = 16;                // 16    -> f2

    f3 = f1 + f2;           // f1 + f2 -> f3
    r = f3 - f1;            // f3 - f1 -> r

    printf("r = %f\n", r);  // Displays the value of r in the terminal.

    return 0;               // Ends the program.
}
```

Indication: $10^{25} \approx 2^{83}$

1. Once the program has run through, what will the value of *r* be? Explain your line of reasoning.
2. Assuming that `f1=10`^{n} where *n* is a natural number, what is the largest value of *n* that still gives a correct value of *r*?
3. Same question if *f1*, *f2*, *f3* and *r* are declared with the `double` keyword (instead of `float`)? Notice that the `double` keyword means “IEEE754 double precision”.