Key to 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
   - **S = 0**
   - $|142| = 142 = 1000\ 1110_2$
   - $142 = (1.000111)_2 \times 2^7$
     M = 0001110…0₂ and e = 7
   - E = e + bias = 7 + 127 = 6 + 128
     E = 1000 0110₂
   - **142 → 0 10000110 00011100000000000000000**

2. -64.25
   - **S = 1**
   - $0.25 \times 2 = \mathbf{0}.5$
     $0.5 \times 2 = \mathbf{1}$
     $|-64{,}25| = 64.25 = 100\ 0000.01_2$
   - $64.25 = (1.00000001)_2 \times 2^6$
     M = 000000010…0₂ and e = 6
   - E = e + bias = 6 + 127 = 5 + 128
     E = 1000 0101₂
   - **-64.25 → 1 10000101 00000001000000000000000**

3. 30.625
   - **S = 0**
   - $0.625 \times 2 = \mathbf{1}.25$
     $0.25 \times 2 = \mathbf{0}.5$
     $0.5 \times 2 = \mathbf{1}$
     $|30.625| = 30.625 = 1\ 1110.101_2$
   - $30.625 = (1.1110101)_2 \times 2^4$
     M = 11101010…0₂ and e = 4
   - E = e + bias = 4 + 127 = 3 + 128
     E = 1000 0011₂
   - **30.625 → 0 10000011 11101010000000000000000**

4. 0.3125
   - **S = 0**
   - 0.3125 × 2 = **0**.625
     0.625 × 2 = **1**.25
     0.25 × 2 = **0**.5
     0.5 × 2 = **1**
     $|0.3125| = 0.3125 = 0.0101_2$
   - $0.3125 = (1.01)_2 \times 2^{-2}$
     M = 010…0₂ and e = -2
   - E = e + bias = -2 + 127
     E = 0111 1101₂
   - **0.3125 → 0 01111101 01000000000000000000000**

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
   - **S = 1**
   - $|-256| = 256 = 1\ 0000\ 0000_2$
   - $256 = (1.0)_2 \times 2^8$
     M = 00…0₂ and e = 8
   - E = e + bias = 8 + 1,023 = 7 + 1,024
     E = 100 0000 0111₂
   - **-256 → 1 10000000111 00……0**

2. 964
   - **S = 0**
   - $|964| = 964 = 11\ 1100\ 0100_2$
   - $964 = (1.1110001)_2 \times 2^9$
     M = 11100010…0₂ and e = 9
   - E = e + bias = 9 + 1,023 = 8 + 1,024
     E = 100 0000 1000₂
   - **964 → 0 10000001000 11100010……0**

3. 8.19140625
   - **S = 0**
   - 0.19140625 × 2 = **0**.3828125
     0.3828125 × 2 = **0**.765625
     0.765625 × 2 = **1**.53125
     0.53125 × 2 = **1**.0625
     0.0625 × 2 = **0**.125
     0.125 × 2 = **0**.25
     0.25 × 2 = **0**.5
     0.5 × 2 = **1**
     $|8.19140625| = 8.19140625 = 1000.0011\ 0001_2$
   - $8.19140625 = (1.00000110001)_2 \times 2^3$
     M = 000001100010…0$_2$ and e = 3
   - E = e + bias = 3 + 1,023 = 2 + 1,024
     E = 100 0000 0010$_2$
   - **8.19140625 → 0 10000000010 000001100010……0**

4. 0.2890625
   - **S = 0**
   - 0.2890625 × 2 = **0**.578125
     0.578125 × 2 = **1**.15625
     0.15625 × 2 = **0**.3125
     0.3125 × 2 = **0**.625
     0.625 × 2 = **1**.25
     0.25 × 2 = **0**.5
     0.5 × 2 = **1**
     $|0.2890625| = 0.2890625 = 0.0100\ 101_2$
   - $0.2890625 = (1.00101)_2 \times 2^{-2}$
     M = 001010…0$_2$ and e = -2
   - E = e + bias = -2 + 1,023
     E = 011 1111 1101$_2$
   - **0.2890625 → 0 01111111101 001010……0**

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. $1011\ 1101\ 0010\ 0000\ 0000\ 0000\ 0000\ 0000_2$
   - S = 1 → **negative**
   - e = E - bias = $0111\ 1010_2$ - 127
     e = 122 - 127
     e = -5
   - **m** = $(1.M)_2$ = $\mathbf{(1.01)_2}$
   - $-m \times 2^e = -(1.01)_2 \times 2^{-5}$
   - $= -(101)_2 \times 2^{-7}$
     $\mathbf{= -5 \times 2^{-7}}$

2. $0100\ 1111\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000_2$
   - S = 0 → **positive**
   - e = E - bias = $1001\ 1110_2$ - 127
     e = 158 - 127
     e = 31
   - **m** = $(1.M)_2$ = $\mathbf{(1.001)_2}$
   - $+m \times 2^e = (1.001)_2 \times 2^{31}$
   - $= (1001)_2 \times 2^{28}$
     $\mathbf{= 9 \times 2^{28}}$

3. $1100\ 1100\ 0101\ 0011\ 0000\ 0000\ 0000\ 0000_2$
   - S = 1 → **negative**
   - e = E - bias = $1001\ 1000_2$ - 127
     e = 152 - 127
     e = 25
   - **m** = $(1.M)_2$ = $\mathbf{(1.1010011)_2}$
   - $-m \times 2^e = -(1.1010011)_2 \times 2^{25}$
   - $= -(11010011)_2 \times 2^{18}$
     $\mathbf{= -211 \times 2^{18}}$

4. $1111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
   - S = 1, E = 1…1 and M = 0…0 → $-\infty$

5. $0000\ 0000\ 0011\ 0000\ 0000\ 0000\ 0000\ 0000_2$
   - E = 0…0 and M ≠ 0…0 → **denormalized mantissa**
   - S = 0 → **positive**
   - m = $(0.M)_2 = (0.011)_2$
   - $+m \times 2^{1-bias} = (0.011)_2 \times 2^{-126} = (11)_2 \times 2^{-129}$
     $\mathbf{= 3 \times 2^{-129}}$

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}
   = 0100 0001 0111 1100 1100 1000 00……0
   - S = 0 → **positive**
   - e = E - bias = 100 0001 0111_2 - 1,023 = 1,047 - 1,023
     e = 24
   - **m** = $(1.M)_2$ = **$(1.110011001)_2$**
   - $+m \times 2^e = (1.110011001)_2 \times 2^{24}$
   - $= (1110011001)_2 \times 2^{15}$
     $= 921 \times 2^{15}$

2. C808 0000 0000 0000_{16}
   = 1100 1000 0000 1000 00……0
   - S = 1 → **negative**
   - e = E - bias = 100 1000 0000_2 - 1,023 = 1,152 - 1,023
     e = 129
   - **m** = $(1.M)_2$ = **$(1.1)_2$**
   - $-m \times 2^e = -(1.1)_2 \times 2^{129} = -(11)_2 \times 2^{128}$
   - **$= -3 \times 2^{128}$**

3. 000A 0000 0000 0000_{16}
   = 0000 0000 0000 1010 00……0
   - E = 0…0 and M ≠ 0…0 → **denormalized mantissa**
   - S = 0 → **positive**
   - e = 1 - bias = 1 - 1,023 = **-1,022**
   - **m** = $(0.M)_2$ = **$(0.101)_2$**
   - $m \times 2^e = (0.101)_2 \times 2^{-1,022} = (101)_2 \times 2^{-1,025}$
   - **$= 5 \times 2^{-1025}$**

4. 8000 0000 0000 0000_{16}
   = 1000 0000 0000 0000……0
   - S = 1, E = 0…0 and M = 0…0 → **-0**

5. FFF0 0001 0000 0000_{16}
   = 1111 1111 1111 0000 0000 0000 0001 0000……0
   - E = 1…1 and M ≠ 0…0 → **NaN**

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.

- **Single precision**

  - **Smallest value**

    $Min_{single} = m_{min} \times 2^{emin}$

    $m_{min} = (1.0)_2 = 1$

    $e_{min} = E_{min} - bias$ with $E_{min} = 1$

    $e_{min} = 1 - 127 = -126$

    $\mathbf{Min_{single} = 2^{-126} \approx 1.1755 \times 10^{-38}}$

  - **Largest value**

    $Max_{single} = m_{max} \times 2^{emax}$

    $m_{max} = (1.M_{max})_2 = 1 + (0.M_{max})_2 = 1 + M_{max} \times 2^{-23}$ with $M_{max} = 2^{23} - 1$

    $m_{max} = 1 + (2^{23} - 1) \times 2^{-23} = 1 + 1 - 2^{-23} = 2 - 2^{-23} = 2 \times (1 - 2^{-24})$

    $e_{max} = E_{max} - bias$ with $E_{max} = (2^8 - 1) - 1 = 254$

    $e_{max} = 254 - 127 = 127$

    $Max_{single} = 2 \times (1 - 2^{-24}) \times 2^{127}$

    $\mathbf{Max_{single} = (1 - 2^{-24}) \times 2^{128} \approx 3.4028 \times 10^{38}}$

- **Double precision**

  - **Smallest value**

    $Min_{double} = m_{min} \times 2^{emin}$

    $m_{min} = (1.0)_2 = 1$

    $e_{min} = E_{min} - bias$ with $E_{min} = 1$

    $e_{min} = 1 - 1{,}023 = -1{,}022$

    $\mathbf{Min_{double} = 2^{-1{,}022} \approx 2.2251 \times 10^{-308}}$

  - **Largest Value**

    $Max_{double} = m_{max} \times 2^{emax}$

    $m_{max} = (1.M_{max})_2 = 1 + (0.M_{max})_2 = 1 + M_{max} \times 2^{-52}$ with $M_{max} = 2^{52} - 1$

    $m_{max} = 1 + (2^{52} - 1) \times 2^{-52} = 1 + 1 - 2^{-52} = 2 - 2^{-52} = 2 \times (1 - 2^{-53})$

    $e_{max} = E_{max} - bias$ with $E_{max} = (2^{11} - 1) - 1 = 2{,}046$

    $e_{max} = 2{,}046 - 1{,}023 = 1{,}023$

    $Max_{double} = 2 \times (1 - 2^{-53}) \times 2^{1{,}023}$

    $\mathbf{Max_{double} = (1 - 2^{-53}) \times 2^{1{,}024} \approx 1.7977 \times 10^{308}}$

2. What is the smallest number (greater than 0) which, when added to 1, gives a different result from 1?

- **Single precision**

  We set: $\mathbf{1 = m \times 2^e}$ with $m = \mathbf{(1.0)_2}$ and $e = \mathbf{0}$.
  The encoded mantissa M contains 23 zeros.

  Let us have a look at the addition below:

```
  2⁰ 2⁻¹                      2⁻²³
  ↑  ↑                        ↑
  1.00000000000000000000000 00000…      ← Number 1
+ 0.00000000000000000000000 101010…     ← Small number
= 1.00000000000000000000000 101010…     ← Result
    └──────────────────────┘
      23-bit encoded mantissa
```

  If the encoded mantissa of the result is 0, the encoded result will be identical to the encoded number 1. The smallest number that makes a difference between these two encoded numbers is $\mathbf{2^{-23}}$.

- **Double precision**

  With the same line of reasoning, the smallest number is $\mathbf{2^{-52}}$.

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.

   - We set: $\mathbf{f1 = (1.M1)_2 \times 2^{e1}}$
     From the indication, it can be deduced that $\mathbf{f1 \approx (1.0)_2 \times 2^{83}}$ with **e1 = 83** and ***M1* = 0…0**

   - We set: $\mathbf{f3 = (1.M3)_2 \times 2^{e3}}$

   - Let us have a look at the addition below:

```
  2⁸³2⁸²                        2⁶⁰     2⁴
  ↑ ↑                           ↑       ↑
  1.00000000000000000000000000……0000… ×2⁸³ → f1 = 10²⁵ ≈ 2⁸³
+ 0.00000000000000000000000000……0100… ×2⁸³ → f2 = 16 = 2⁴
= 1.00000000000000000000000000……0100… ×2⁸³ → f3 = 2⁸³ + 2⁴
    └──────────────────────────┘
                M3
```

     It can be noticed that the smallest value of *f2* that can modify *M3* is 2^{60}. Since *f2* = 16, the variables *f1* and *f3* will be encoded in the same way. That is to say: *f1* = *f3*.

     The addition ***f3* = *f1* + *f2*** becomes ***f3* = *f1***.
     The subtraction ***r* = *f3* - *f1*** becomes ***r* = *f3* - *f3* = 0**.

     Once the program has run through, the value of *r* will be 0.

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*?

   - The difference between *f1* and *f2* must be small enough so that *f3* will be affected by the addition. Since *f2* = 16, the largest weight of the least significant bit of *M1* must be 2^4, because if this weight is greater than or equal to 2^5, *f2* will be too small and the addition will not affect *f3*.

```
  2²⁸                       2⁵
  ↑                         ↑
  1.00000000000000000000000000… ×2²⁸ → f1 = 2²⁸
+ 0.00000000000000000000000100… ×2²⁸ → f2 = 16 = 2⁴
= 1.00000000000000000000000100… ×2²⁸ → f3 = 2²⁸ + 2⁴
    └──────────────────────┘
               M3
```

   - Therefore, the variable *f1* must be less than 2^{28} so that the addition will affect *f3*.

- Which gives:
  $f1 < 2^{28}$
  $10^n < 2^{28}$
  $n < \text{Log}(2^{28})$
  $n < 8.42$

  $\mathbf{n_{max} = 8}$

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”.

   With the same line of reasoning:
   $f1 < 2^{5+52}$
   $10^n < 2^{57}$
   $n < \text{Log}(2^{57})$
   $n < 17.15$

   $\mathbf{n_{max} = 17}$

Handling integer and floating-point variables is not as easy as it seems. Overflows are common pitfalls and can elude the attention of any skilled programmer. Precision matters should never be underestimated.

About the Ariane 5 space rocket

Here is a famous example of how a tiny programming error can turn into a huge disaster. After being launched for the very first time on 4 June 1996, Ariane 5 exploded only 40 seconds after taking off from the Kourou space center in French Guiana. The financial loss was estimated at about $500 million. The CNES (Centre National d’Études Spatiales) and the ESA (European Space Agency) immediately launched an enquiry. One month later, a panel of international experts reported unequivocally that the explosion had been caused by a software error.

The INS or Inertial Navigation System was to blame. This part of the software, derived from the Ariane 4 launcher, had not been adapted to the highest horizontal speed of Ariane 5. As a result, when the 64-bit floating-point number containing the horizontal speed was converted into a 16-bit integer, an overflow occurred and an exception was generated. Unfortunately, as there was no special routine to deal with it, the standard routine ran by default and the software shut down altogether.

Since then, Ariane’s software designers have established a defensive programming plan, which has been recognised as a standard in the field.

Jean-Christophe Arnulfo, *Métier Développeur*, Paris, Dunod, 2003