# MOwNiT - Arytmetyka komputerowa

Jakub Frączek

6 marca 2024

## 1 Temat ćwiczenia

Wyznaczyć doświadczalnie parametry reprezentacji liczb zmiennoprzecinkowych (float, double, long double) i porównać uzyskane wartości dla różnych architektur, systemów operacyjnych i kompilatorów. Sprawdzić, czy reprezentacje są zgodne ze standardem IEEE. Parametry do wyznaczenia:

- liczba bajtów używana do przechowywania zmiennej danego typu,
- liczba bitów na mantysę,
- liczba bitów na cechę (wliczając znak),
- "maszynowe epsilon najmniejsza liczba, dla której 1.0+e $>$ 1.0,
- występowanie i sposób reprezentacji wartości specjalnych ($\pm 0$, $\pm$Inf, NaN).

## 2 Dane techniczne

### 2.1 Procesory, systemy operacyjne, architektury oraz kompilatory

- Intel Core i5-9300H 2.40 GHz
  - Windows 11 Home x64
    - gcc 12.2.0 x64
    - tcc 0.9.27 x64
    - tcc 0.9.27 x32
- AMD Ryzen 5 5500U 2.1 GHz
  - Linux Mint 21.2 x64
    - gcc 11.4.0 x64
    - gcc 11.4.0 x32
    - tcc 0.9.27 x64
    - clang 14.0.0 x64
    - clang 14.0.0 x32
- Apple A12 Bionic 2.49GHz
  - iPadOs 17.3.1 x64
    - clang 14.0.0 x64 (zmodyfikowany; kompilacja do bytecodu -> interpretacja bytecodeu)

## 2.2 Wykorzystany język oraz biblioteki

Doświadczenie zostało przeprowadzone z wykorzystaniem języka C. Wykorzystane biblioteki to:

- float.h
- stdio.h
- float.h
- math.h
- stdlib.h

## 2.3 Funkcja do odczytu reprezentacji binarnej z pamięci

Poniżej (kod 1.) Znajduje się funkcja wykorzystywana do manualnego odczytu postaci binarnej liczby zmiennoprzecinkowej z pamięci.

```
void print_bits ( void* buffer, int bytes, unsigned int mantissa_bits){

    if(sizeof(char) != 1){

        printf("Char size != 1");
        return;

    }

    int bit_counter = 0;
    unsigned char* ptr = (unsigned char*)buffer;

    for (int i = bytes - 1; i >= 0; i -= 1) {

        for (int j = 7; j >= 0; j--) {

            printf("%d", (ptr[i] >> j) & 1);
            bit_counter += 1;

            if(bit_counter == 1 || bit_counter == 1 + (bytes * 8)
            - mantissa_bits) printf(" ");
        }

    }
    printf("\n");
}
```

Kod 1. Funkcja do odczytu reprezentacji binarnej z pamięci

## 2.4 Funkcja do wyznaczania maszynowego epsilon

Poniżej (kod 2.) Znajduje się funkcja wykorzystywana do wyznaczania maszynowego epsilon.

```
int <variable_type>_epsilon() {
    int pow = 0;
    <variable_type> eps = 1;
    while (eps + 1 != 1) {
        eps /= 2;
        --pow;
    }
```

```
    return pow + 1;
}
```

Kod 2. Funkcja do wyznaczania maszynowego epsilon

### 2.5 Funkcja do wyznaczania rozmiaru mantysy

Poniżej (kod 3.) Znajduje się funkcja wykorzystywana do wyznaczania rozmiaru mantysy.

```
int <variable_type>_mantissa(){

    <variable_type> a = 1;
    <variable_type> b = 0.5;
    <variable_type> c = a + b;
    int bits = 1;
    while (c != a) {
        bits = bits + 1;
        b = b / 2;
        c = a + b;
    }

    return bits;

}
```

Kod 3. Funkcja do wyznaczania rozmiaru mantysy

### 2.6 Opis realizacji

Szukane wartości zostały wyznaczone poprzez odczytanie stałych zawartych w bibliotece <float.h> oraz poprzez skorzystanie z powyższych funkcji.

## 3 Czym jest IEEE

Instytut Inżynierów Elektryków i Elektroników. Organizacja typu non-profit skupiająca osoby zawodowo związane z elektrycznością i elektroniką, a także pokrewnymi dziedzinami. Jednym z podstawowych jej zadań jest ustalanie standardów dla urządzeń elektronicznych, w tym standardów dla urządzeń i formatów komputerowych.

## 4 Standard IEEE 754

Jest to standard reprezentacji binarnej i operacji na liczbach zmiennoprzecinkowych, implementowany powszechnie w procesorach i oprogramowaniu obliczeniowym.

### 4.1 Zapis liczb zmiennoprzecinkowych

$$L = (-1)^{znak} * mantysa * 2^{cecha}$$

- znak - 0 oznacza liczbę dodatnią, 1 ujemną
- mantysa - wpływa na precyzję
- cecha - wpływa na zakres

### 4.2 Normalizacja mantysy

Polega na zwiększaniu potęgi wykładnika i przesuwaniu przecinka w lewo, aż liczba będzie należeć do przedziału [1, 2).

## 4.3 Przesunięcie wykładnika

Bias jest to przesunięcie pozwalające na zapis ujemnego wykładnika. Typowe wartości przesunięcia to:

- 127 w formacie 32-bitowym,
- 1023 w formacie 64-bitowym,
- 16383 w formacie 80-bitowym.

## 4.4 Liczby znormalizowane

$$L = (-1)^{znak} * 1.mantysa * 2^{cecha-bias}$$

Mantysa ma wartość z przedziału [1,2) - pierwszy bit ma zawsze wartość 1 (ukryta 1), więc nie trzeba go zapisywać (jest on tam "domyślnie").

Przykład liczby znormalizowanej (Kod 4.):

```
float number1 = 3.9;
print_bits(&number1, sizeof(float), FLT_MANT_DIG);

output: 0 10000000 11110011001100110011010
```

Kod 4. Uruchomiony z wykorzystaniem: Intel Core i5-9300H 2.40 GHz, Windows 11 Home x64, gcc 12.2.0

## 4.5 Liczby zdenormalizowane

$$L = (-1)^{znak} * 0.mantysa * 2^{-bias+1}$$

Przykład liczby zdenormalizowanej (Kod 5.):

```
float number2 = 1e-39;
    print_bits(&number2, sizeof(float), FLT_MANT_DIG);

output: 0 00000000 00010101110001110011000
```

Kod 5. Uruchomiony z wykorzystaniem: Intel Core i5-9300H 2.40 GHz, Windows 11 Home x64, gcc 12.2.0

Jeżeli cecha składa się z samych 0 to liczba jest zdenormalizowana i mantysa nie posiada "domyślnego"bitu. Pozwala to na reprezentacje liczb bliskich 0, które bez denormalizacji byłyby 0.

## 4.6 Parametry reprezentacji liczb zmiennoprzecinkowych

Poniżej (w tabeli 1.) przedstawione są parametry reprezentacji liczb zmiennoprzecinkowych zawarte w standardzie IEEE 754.

| Nazwa | Typ | Znak | Cecha | Mantysa | Maszynowe epsilon |
|---|---|---|---|---|---|
| pojedynczej precyzji | 32 bity | 1 bit | 8 bitów | 23 bity | 1.19e-07 |
| podwójnej precyzji | 64 bity | 1 bit | 11 bitów | 52 bity | 2.22e-16 |
| poczwórnej precyzji | 128 bitów | 1 bit | 15 bitów | 112 bitów | 1.93e-34 |

Tabela 1. Parametry reprezentacji liczb zmiennoprzecinkowych

### 4.7 Specjalne przypadki

- +0 - wszystkie bity są zerami
- -0 - bit znaku jest ustawiony, reszta jest zerami
- NaN - ustawione wszystkie bity wykładnika, mantysa różna od 0
- +inf - bit znaku jest zerem, ustawione wszystkie bity wykładnika, mantysa równa 0
- -inf - bit znaku jest ustawiony, ustawione wszystkie bity wykładnika, mantysa równa 0

## 5 Wyniki

### 5.1 Float

Dla każdego przeprowadzonego testu, na wszystkich konfiguracjach opisanych w sekcji 2.1 otrzymany wynik został zaprezentowany w tabeli 2.

| Liczba bitów | Liczba bitów mantysy | Liczba bitów cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 32 | 24 | 8 | 1.192093e-07 |

Tabela 2. Wyniki testów przeprowadzonych dla zmiennej float

### 5.2 Double

Dla każdego przeprowadzonego testu, na wszystkich konfiguracjach opisanych w sekcji 2.1 otrzymany wynik został zaprezentowany w tabeli 3.

| Liczba bitów | Liczba bitów mantysy | Liczba bitów cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 64 | 53 | 11 | 2.22045e-16 |

Tabela 3. Wyniki testów przeprowadzonych dla zmiennej double

### 5.3 Long double

Dla tego typu zmiennej wyniki okazały się być bardzo zróżnicowane. Zauważyłem też, że w niektórych przypadkach wartości podane w stałych zawartych w "float.h"różnią się od rzeczywistych. Wartość w nawiasie odzwierciedla niepoprawną wartość odczytaną ze stałej, bądź przy użyciu funckji sizeof().

#### 5.3.1 Wynik I

Dla:

- Intel Core i 5 Intel Core i5-9300H 2.40 GHz; Windows 11 Home x64; gcc 12.2.0 x64
- AMD Ryzen 5 5500U 2.1 GHz; Linux Mint 21.2 x64; gcc 11.4.0 x64
- AMD Ryzen 5 5500U 2.1 GHz; Linux Mint 21.2 x64; tcc 0.9.27 x64
- AMD Ryzen 5 5500U 2.1 GHz; Linux Mint 21.2 x64; clang 14.0.0 x64

Otrzymałem (tabela 4.):

| Liczba bitów | Bity mantysy | Bity cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 80 (128) | 64 | 16 (64) | 1.084202e-19 |

Tabela 4. Wybrane wyniki dla zmiennej long double

Liczba w nawiasie w kolumnie "Liczba bitów"oznacza liczbę otrzymaną poprzez użycie funkcji sizeof(), jednak po odczytaniu wartości z pamięci okazuje się, że znajduje się tam, aż 48 zaalokowanych, jednak nie używanych bitów. Najprawdopodobniej dzieje się tak najprawdopodobiej ze względu na proces zwany "bit aligmnent"(obiekt jest "8 bits aligned"jeśli rozmiar jego pamięci jest wielokrotnością 8. Niektóre procesory czytają tylko pamięć, która jest taką liczbą, a dla innych przyspiesza to czas odczytu).

### 5.3.2 Wynik II

Dla:

- Intel Core i 5 Intel Core i5-9300H 2.40 GHz; Windows 11 Home x64; tcc 0.9.27 x64
- Intel Core i 5 Intel Core i5-9300H 2.40 GHz; Windows 11 Home x64; tcc 0.9.27 x32

Otrzymałem (tabela 5):

| Liczba bitów | Bity mantysy | Bity cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 64 | 53 (64) | 11(0) | 2.22045e-016 (1.084202e-019) |

Tabela 5. Wybrane wyniki dla zmiennej long double

W tym wypadku, w stałej LDBL_MANT_DIG znajdowała się niepoprawna wartość reprezentująca ilość bitów mantysy, po zweryfikowaniu okazało się, że jest ich 53, a nie 64; co byłoby z resztą nie możliwe przy rozmiarze long double wynoszącym 64 bity. Zła była także wartość maszynowego epsilon.

### 5.3.3 Wynik III

Dla:

- Apple A12 Bionic 2.49GHz; iPadOs 17.3.1 x64; clang 14.0.0 x64 (Zmodyfikowany)

Otrzymałem (tabela 6.):

| Liczba bitów | Bity mantysy | Bity cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 64 | 53 | 11 | 2.22045e-016 |

Tabela 6. Wybrane wyniki dla zmiennej long double

Tutaj wynik jest podobny jak powyżej, z tą różnicą, że wszystkie wartości w "float.h"były zgodne z rzeczywistością.

### 5.3.4 Wynik IV

Dla:

- AMD Ryzen 5 5500U 2.1 GHz; Linux Mint 21.2 x64; gcc 11.4.0 x32
- AMD Ryzen 5 5500U 2.1 GHz; Linux Mint 21.2 x64; clang 14.0.0 x32

Otrzymałem (tabela 7.):

| Liczba bitów | Bity mantysy | Bity cechy (wliczając znak) | Maszynowe epsilon |
|---|---|---|---|
| 96 | 64 | 32 | 1.084202e-19 |

Tabela 7. Wybrane wyniki dla zmiennej long double

W tym wypadku mamy do czynienia z największą liczbą bitów przeznaczonych do zapisu zmiennej long double.

## 5.4 Wartości specjalne

Wartości specjalne w prawie każdym przypadku były przechowywane w identyczny sposób.

### 5.4.1 +0

Na wszystkich konfiguracjach w postaci:

- bit znaku: "0"
- mantysa: same "0"
- cecha: same "0"

### 5.4.2 -0

Na wszystkich konfiguracjach w postaci:

- bit znaku: "1"
- mantysa: same "0"
- cecha: same "0"

### 5.4.3 +Inf

Na wszystkich konfiguracjach w postaci:

- bit znaku: "0"
- mantysa: same "1"
- cecha: same "0"

### 5.4.4 -Inf

Na wszystkich konfiguracjach w postaci:

- bit znaku: "1"
- mantysa: same "1"
- cecha: same "0"

## 5.5 NaN

Na konfiguracjach z kompilatorem tcc:

- bit znaku: "1"
- mantysa: same "1"
- cecha: jedna "1", reszta "0"

Na konfiguracjach bez kompilatora tcc:

- bit znaku: "0"
- mantysa: same "1"
- cecha: jedna "1", reszta "0"

# 6 Wnioski

- Po przeprowadzonej analizie można stwierdzić, że wszystkie przetestowane przeze mnie konfiguracje są zgodne ze standardem IEEE 754. Jeśli chodzi long double to nie jest on w nim przewidziany, więc jego implementacje się różnią.
- Po porzeczytaniu danych z sekcji 5.3.2 oraz 5.3.3 można dojść do wniosku, że typ long double został tam zrealizowany jako zwykły double.
- Nie należy polegać na typie danych long double, gdyż implementacja znacząco się różni w zależności od procesora, systemu oraz kompilatora.
- Dodając ":i386"do nazwy pakietu instalowanego przy pomocy unixowego apt można uzyskać jego 32 bitową wersję (w przypadku korzystania z 64 bitowego sytemu).
- System iPadOs nie pozwala na uruchamianie kodu niepochodzącego z dedykowanego sklepu, stąd kompilator clang został zmodyfikowany (przez dewelopera, który udostępnił go w App Store), tak by kod był kompilowany do bytecode'u na następnie interpretowany, co pozwoliło obejść restrykcje.

# 7 Źródła


- https://pl.wikipedia.org/wiki/Liczba_zmiennoprzecinkowa
- https://pl.wikipedia.org/wiki/Kod_z_przesuni%C4%99ciem
- https://en.wikipedia.org/wiki/Machine_epsilon
- https://stackoverflow.com/questions/2846914/what-is-meant-by-memory-is-8-bytes-aligned
- https://en.wikipedia.org/wiki/IEEE_754
- https://askubuntu.com/questions/29665/how-do-i-apt-get-a-32-bit-package-on-a-64-bit-installation