CareerCon™

KARIERA IT

# Pułapki liczb zmiennoprzecinkowych

Adam Sawicki – asawicki.info

24.09.2016

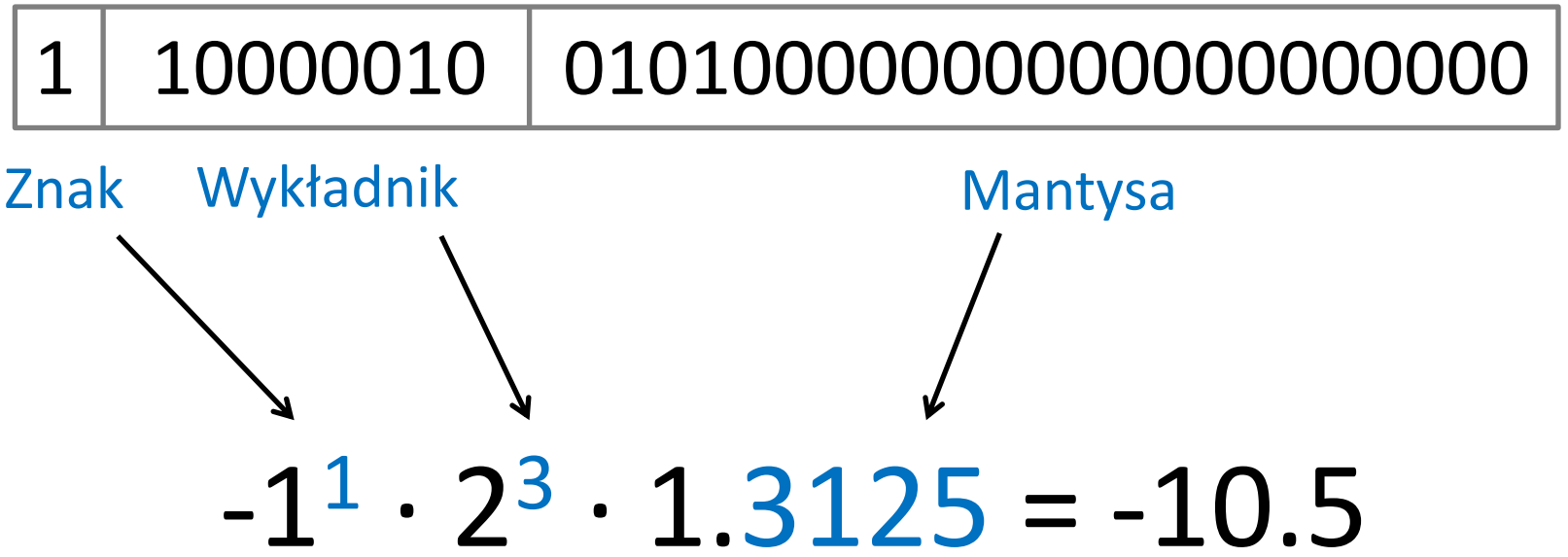
# Agenda

- Liczby zmiennoprzecinkowe
  - Budowa
  - Typy – możliwości i ograniczenia
  - Typy – w językach programowania
- Pułapki
  - Zakres
  - Precyzja
  - Nieskończone rozwinięcie
  - Liczby całkowite
  - Porównywanie
  - Wartości specjalne
- Podsumowanie
- Ciekawostka: half-float

# Liczby zmiennoprzecinkowe

- Liczby zmiennoprzecinkowe (*floating point numbers*)
  - Przybliżona reprezentacja liczb rzeczywistych
  - Mogą zawierać część całkowitą i ułamkową
  - Mogą przyjmować bardzo duże lub małe wartości
- Standard IEEE 754
  - Wspierane sprzętowo przez procesor
  - Dostępne w różnych językach programowania

# Liczby zmiennoprzecinkowe




$$-1^{1} \cdot 2^{3} \cdot 1.3125 = -10.5$$

Przydatne narzędzie: IEEE 754 Converter

# Typy – możliwości i ograniczenia

## Floating-Point Formats Cheatsheet @ asawicki.info

### Floating-Point Formats

Version 1.0 © 2013-06-13 Adam Sawicki, adam DELETE @asawicki.info, http://asawicki.info

Source: http://en.wikipedia.org

#### General Parameters

| Name | Half | Single | Double |
|---|---|---|---|
| Bytes | 2 | 4 | 8 |
| Bits = sign + exponent + significand | 16 = 1 + 5 + 10 | 32 = 1 + 8 + 23 | 64 = 1 + 11 + 52 |
| Exponent bias, range | 15, −14 … 15 | 127, −126 … 127 | 1023, −1022 … 1023 |
| Significant decimal digits | ≈ 3.311 | ≈ 7.225 (6 … 9) | ≈ 15.955 (15 … 17) |
| Minimum positive subnormal | $2^{-24} \approx 5.96 \times 10^{-8}$ | $2^{-149} \approx 1.4 \times 10^{-45}$ | $2^{-1022-52} \approx 5 \times 10^{-324}$ |
| Minimum positive normal | $2^{-14} \approx 6.10 \times 10^{-5}$ | $2^{-126} \approx 1.18 \times 10^{-38}$ | $2^{-1022} \approx 2.2250738585072014 \times 10^{-308}$ |
| Next value after 1 | $1 + 2^{-10} = 1.0009765625$ | $1 + 2^{-23} \approx 1.0000001$ | $1 + 2^{-52} \approx 1.0000000000000002$ |
| Integers represented exactly | $0 \ldots 2^{11} = 2048$ | $0 \ldots 2^{24} = 16{,}777{,}216$ | $0 \ldots 2^{53} = 9{,}007{,}199{,}254{,}740{,}992$ |
| Maximum | $(2-2^{-10}) \times 2^{15} = 65504$ | $(2-2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$ | $(1 + (1 - 2^{-52})) \times 2^{1023} \approx 1.7976931348623157 \times 10^{308}$ |

#### Example Values (hexadecimal)

| | | | |
|---|---|---|---|
| **0.0** | 0000 | 00000000 | 00000000 00000000 |
| **1.0** | 3C00 | 3F800000 | 3FF00000 00000000 |
| **-1.0** | BC00 | BF800000 | BFF00000 00000000 |
| **0.5** | 3800 | 3F000000 | 3FE00000 00000000 |
| **2.0** | 4000 | 40000000 | 40000000 00000000 |
| **Example NaN** | FFFF | FFFFFFFF | FFFFFFFF FFFFFFFF |

#### Special Values

| Exponent (binary) | Significand (binary) | Meaning |
|---|---|---|
| 000…0 | 000…0 | −0 / +0 |
| 000…0 | Non-zero | Subnormal |
| 111…1 | 000…0 | −Inf / +Inf (Infinity) |
| 111…1 | Non-zero | NaN (Not a Number) |
| Other value | Any value | Normalized value |

# Typy – możliwości i ograniczenia

Dwa najpopularniejsze: 32- i 64-bitowy

- Nazywane „pojedynczej” i „podwójnej” precyzji

| Name | Half | Single | Double |
|---|---|---|---|
| Bytes | 2 | 4 | 8 |
| Bits = sign + exponent + significand | 16 = 1 + 5 + 10 | 32 = 1 + 8 + 23 | 64 = 1 + 11 + 52 |
| Exponent bias, range | 15, −14 … 15 | 127, −126 … 127 | 1023, −1022 … 1023 |
| Significant decimal digits | ≈ 3.311 | ≈ 7.225 (6 … 9) | ≈ 15.955 (15 … 17) |
| Minimum positive subnormal | $2^{-24} \approx 5.96 \times 10^{-8}$ | $2^{-149} \approx 1.4 \times 10^{-45}$ | $2^{-1022-52} \approx 5 \times 10^{-324}$ |
| Minimum positive normal | $2^{-14} \approx 6.10 \times 10^{-5}$ | $2^{-126} \approx 1.18 \times 10^{-38}$ | $2^{-1022} \approx 2.225073858507$ |
| Next value after 1 | $1 + 2^{-10} = 1.0009765625$ | $1 + 2^{-23} \approx 1.0000001$ | $1 + 2^{-52} \approx 1.00000000000$ |
| Integers represented exactly | $0 \ldots 2^{11} = 2048$ | $0 \ldots 2^{24} = 16{,}777{,}216$ | $0 \ldots 2^{53} = 9{,}007{,}199{,}254{,}7$ |
| Maximum | $(2-2^{-10}) \times 2^{15} = 65504$ | $(2-2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$ | $(1 + (1 - 2^{-52})) \times 2^{1023} \approx 1.$ |

# Typy – w językach programowania

| Język | 32-bit Float | 64-bit Float |
|---|---|---|
| C, C++ * | float | double |
| C# | float | double |
| Java | float | double |
| Delphi/Object Pascal | Single | Double |
| SQL | REAL | FLOAT |
| Python * | | float |
| PHP * | | float |
| JavaScript ** | | Number |
| Lua ** | | number |

* Zazwyczaj – zależnie od implementacji
** Jedyny dostępny typ liczbowy, nie ma typów całkowitych

# Zakres

Ograniczony zakres możliwych do zapisania wartości

- Wartość najbliższa zero
- Wartość największa/najmniejsza
- Jest szeroki, rzadko stanowi problem

| Name | Single | Double |
|---|---|---|
| Bytes | 4 | 8 |
| Bits = sign + exponent + significand | 32 = 1 + 8 + 23 | 64 = 1 + 11 + 52 |
| Exponent bias, range | 127, −126 … 127 | 1023, −1022 … 1023 |
| Significant decimal digits | ≈ 7.225 (6 … 9) | ≈ 15.955 (15 … 17) |
| Minimum positive subnormal | $2^{-149} \approx 1.4 \times 10^{-45}$ | $2^{-1022-52} \approx 5 \times 10^{-324}$ |
| Minimum positive normal | $2^{-126} \approx 1.18 \times 10^{-38}$ | $2^{-1022} \approx 2.2250738585072014 \times 10^{-308}$ |
| Next value after 1 | $1 + 2^{-23} \approx 1.0000001$ | $1 + 2^{-52} \approx 1.0000000000000002$ |
| Integers represented exactly | $0 \ldots 2^{24} = 16{,}777{,}216$ | $0 \ldots 2^{53} = 9{,}007{,}199{,}254{,}740{,}992$ |
| Maximum | $(2-2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$ | $(1 + (1 - 2^{-52})) \times 2^{1023} \approx 1.7976931348623157 \times 10^{308}$ |

# Precyzja

Precyzja określana jest liczbą cyfr znaczących

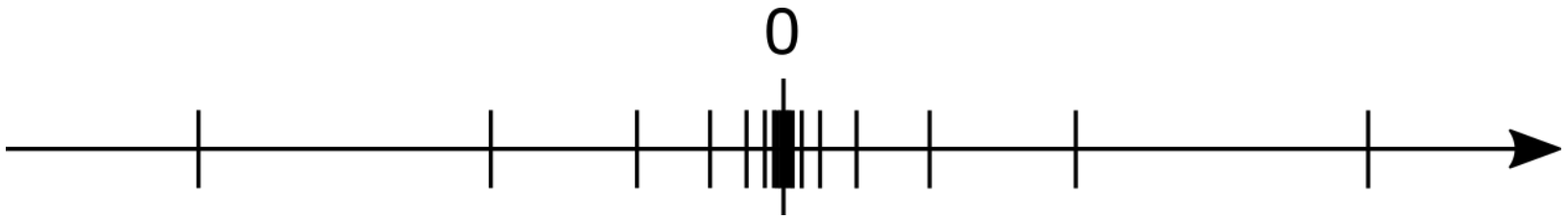
- Binarnych – to liczba bitów mantysy + 1
- Dziesiętnych – podawana w przybliżeniu

| Name | Single | Double |
|---|---|---|
| Bytes | 4 | 8 |
| Bits = sign + exponent + significand | 32 = 1 + 8 + 23 | 64 = 1 + 11 + 52 |
| Exponent bias, range | 127, −126 … 127 | 1023, −1022 … 1023 |
| Significant decimal digits | ≈ 7.225 (6 … 9) | ≈ 15.955 (15 … 17) |
| Minimum positive subnormal | $2^{-149} \approx 1.4 \times 10^{-45}$ | $2^{-1022-52} \approx 5 \times 10^{-324}$ |
| Minimum positive normal | $2^{-126} \approx 1.18 \times 10^{-38}$ | $2^{-1022} \approx 2.2250738585072014 \times 10^{-308}$ |
| Next value after 1 | $1 + 2^{-23} \approx 1.0000001$ | $1 + 2^{-52} \approx 1.0000000000000002$ |
| Integers represented exactly | $0 \ldots 2^{24} = 16{,}777{,}216$ | $0 \ldots 2^{53} = 9{,}007{,}199{,}254{,}740{,}992$ |
| Maximum | $(2-2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$ | $(1 + (1 - 2^{-52})) \times 2^{1023} \approx 1.7976931348623157 \times 10^{308}$ |

# Precyzja

Precyzja jest względna

- Różnice między sąsiednimi możliwymi do zapisania wartościami są tym większe, im większa jest wartość liczbowa.
- Wartości te rozmieszczone są „gęściej” w okolicy zera.




```
float x = 1111.111111111111f;
printf("%.16g\n", x);
```

→ `1111.111083984375`

# Precyzja – przykład

```
#include <Windows.h>
#include <stdio.h>

int main() {
    float poczatek = (float)timeGetTime() / 1000.0f;
    DlugaOperacja();
    float koniec = (float)timeGetTime() / 1000.0f;
    printf("DlugaOperacja trwala: %g s\n", koniec - poczatek);
}
```

timeGetTime zwraca czas od startu systemu, w milisekundach.

- Ile wynosi precyzja wyniku?
- Co jest źle w tym przykładzie? Jak to naprawić?

# Precyzja – przykład

Jest to przykład tzw. *catastrophic cancellation*

- Zmienne poczatek i koniec mogą mieć duże wartości.
- Niewielka różnica między nimi może być niedokładna przez ograniczenia pracyzji.

| Czas od startu systemu | Precyzja zmiennej |
|---|---|
| 1 s | 119 ns |
| 10 000 s (3 godziny) | 1 ms |
| 100 000 s (1 dzień) | 7.81 ms |

# Precyzja – przykład

Rozwiązaniem jest zachować czasy w ich natywnym typie, a dopiero ich różnicę zamienić na typ float.

```
int main() {
    DWORD poczatek = timeGetTime();
    DlugaOperacja();
    DWORD koniec = timeGetTime();
    float czasTrwania = (float)(koniec - poczatek) / 1000.0f;
    printf("DlugaOperacja trwala: %g s\n", czasTrwania);
}
```

Podobnie precyzyjny czas z funkcji QueryPerformanceCounter warto pozostawić jako 64-bitowy integer.

# Nieskończone rozwinięcie

Liczba mająca skończone rozwinięcie w zapisie dziesiętnym niekoniecznie ma takie w zapisie binarnym.

| Liczba | System dziesiętny | System binarny |
|---|---|---|
| 1 + 1/2 | 1.5 | 1.1 |
| π | 3.1415927… | 11.001001… |
| 1/10 | 0.1 | 0.0001101… (!) |

# Liczby całkowite

Wartości całkowite są w typach zmiennoprzecinkowych reprezentowane dokładnie…

- Operacje na nich (dodawanie, odejmowanie, mnożenie) także dają dokładny wynik.
- Dzięki temu można ich używać zamiast liczb całkowitych (jak w JavaScript, Lua).

```
float a = 256, b = 13;
printf("%g\n", a * b);
```



```
3328
```

# Liczby całkowite

…jednak tylko do pewnej wartości maksymalnej!

- Powyżej tej wartości zaczynają „przeskakiwać” co 2, potem co 4 itd.
- Zakres dokładnych liczb całkowitych w X-bitowym float jest mniejszy, niż w X-bitowym integer.
- Zakres dokładnych liczb całkowitych w 64-bitowym float obejmuje cały zakres 32-bitowych integer.

| Name | Single | Double |
|---|---|---|
| Bytes | 4 | 8 |
| Bits = sign + exponent + significand | 32 = 1 + 8 + 23 | 64 = 1 + 11 + 52 |
| Exponent bias, range | 127, −126 … 127 | 1023, −1022 … 1023 |
| Significant decimal digits | ≈ 7.225 (6 … 9) | ≈ 15.955 (15 … 17) |
| Minimum positive subnormal | $2^{-149} \approx 1.4 \times 10^{-45}$ | $2^{-1022-52} \approx 5 \times 10^{-324}$ |
| Minimum positive normal | $2^{-126} \approx 1.18 \times 10^{-38}$ | $2^{-1022} \approx 2.2250738585072014 \times 10^{-308}$ |
| Next value after 1 | $1 + 2^{-23} \approx 1.0000001$ | $1 + 2^{-52} \approx 1.0000000000000002$ |
| Integers represented exactly | $0 \ldots 2^{24} = 16{,}777{,}216$ | $0 \ldots 2^{53} = 9{,}007{,}199{,}254{,}740{,}992$ |
| Maximum | $(2-2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$ | $(1 + (1 - 2^{-52})) \times 2^{1023} \approx 1.7976931348623157 \times 10^{308}$ |

# Porównywanie

Wyników obliczeń nie należy porównywać operatorem == ani !=

- Niedokładności na ostatnich miejscach po przecinku mogą spowodować, że liczby nie będą identyczne.

```
#include <stdio.h>

int main() {
    float a = 1.0f / 10.0f;
    float b = 1.0f - 0.9f;
    printf("a=%g, b=%g\n", a, b);
    if (a == b)
        printf("Zgadza sie.\n");
    else
        printf("Nie zgadza sie!\n");
}
```



```
a=0.1, b=0.1
Nie zgadza sie!
```

# Porównywanie

```c
#include <stdio.h>

int main() {
    float a = 1.0f / 10.0f;
    float b = 1.0f - 0.9f;
    printf("a=%g, b=%g\n", a, b);
    if (a == b)
        printf("Zgadza sie.\n");
    else
        printf("Nie zgadza sie!\n");
}
```

a = 0x3dcccccd ≈ 0.100000001

b = 0x3dccccd0 ≈ 0.100000024

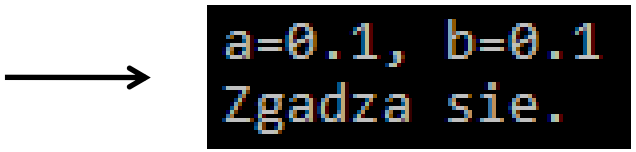
# Porównywanie

Rozwiązaniem jest porównywanie z pewnym małym marginesem ±ε

- Ile powinien wynosić? To trudne pytanie. Zależy od rzędu wielkości wyniku i spodziewanych błędów.

```
#include <stdio.h>
#include <math.h>

int main() {
    float a = 1.0f / 10.0f;
    float b = 1.0f - 0.9f;
    printf("a=%g, b=%g\n", a, b);
    if (fabsf(b - a) < 0.00001f)
        printf("Zgadza sie.\n");
    else
        printf("Nie zgadza sie!\n");
}
```



# Wartości specjalne

Wartość zmiennoprzecinkowa może być jednego z kilku rodzajów, oprócz wartości normalnej (*normal*):

- Zero
  - Są dwa zera, zależnie od bitu znaku: +0 i -0.
  - Są sobie równe (+0 == -0), więc nie trzeba się nimi zajmować.
- Wartość zdenormalizowana (*denormal*, *subnormal*)
  - Pozwala zapisać wartość jeszcze mniejszą (bliższą zero), niż normalna.
  - Po prostu działa – nie trzeba się nią zajmować.

# Wartości specjalne

- INF – nieskończoność (od *Infinity*)
  - Również ma dwie wartości: +INF i -INF.
  - Oznacza przepełnienie ponad maksymalną wartość lub matematyczną nieskończoność ∞.
  - Drukowana jako „1.#INF” lub „inf”.
- NaN – „nie liczba” (od *Not a Number*)
  - Wartość niemożliwa do określenia, błąd obliczeń – np. wynik niedozwolonej operacji.
  - Drukowana jako „1.#IND” (od *indeterminate* – nieokreślony) lub „nan” itp.

# Wartości specjalne – test

```c
#include <stdio.h>
#include <math.h>
int main() {
    double zero = 0.0;
    double dwa  = 2.0;
    printf(" 2 / 0 = %g\n", dwa  / zero);
    printf("-2 / 0 = %g\n", -dwa / zero);
    printf(" 0 / 0 = %g\n", zero / zero);
    printf("log(0)   = %g\n", log(zero));
    printf("log(-2)  = %g\n", log(-dwa));
    printf("sqrt(-2) = %g\n", sqrt(-dwa));
    double inf = dwa  / zero;
    double nan = zero / zero;
    printf(" 2 + INF = %g\n", dwa  + inf);
    printf(" 2 * INF = %g\n", dwa  * inf);
    printf("-2 * INF = %g\n", -dwa * inf);
    printf(" 0 * INF = %g\n", zero * inf);
    printf(" 2 / INF = %g\n", dwa  / inf);
    printf(" 0 / INF = %g\n", zero / inf);
    printf("INF + INF = %g\n", inf + inf);
    printf("INF - INF = %g\n", inf - inf);
    printf("INF * INF = %g\n", inf * inf);
    printf("INF / INF = %g\n", inf / inf);
    printf("2 + NaN = %g\n", dwa + nan);
    printf("2 * NaN = %g\n", dwa * nan);
    printf("2 / NaN = %g\n", dwa / nan);

    printf(" INF > 2 = %s\n" , inf > dwa ? "true" : "false");
    printf("-INF < 2 = %s\n", -inf < dwa ? "true" : "false");
    printf("2   == NaN = %s\n",  dwa == nan ? "true" : "false");
    printf("NaN == NaN = %s\n", -nan == nan ? "true" : "false");
}
```

# Wartości specjalne – test

```
 2 / 0 = inf
-2 / 0 = -inf
 0 / 0 = -nan(ind)
log(0)   = -inf
log(-2)  = -nan(ind)
sqrt(-2) = -nan(ind)
 2 + INF = inf
 2 * INF = inf
-2 * INF = -inf
 0 * INF = -nan(ind)
 2 / INF = 0
 0 / INF = 0
INF + INF = inf
INF - INF = -nan(ind)
INF * INF = inf
INF / INF = -nan(ind)
2 + NaN = -nan(ind)
2 * NaN = -nan(ind)
2 / NaN = -nan(ind)
 INF > 2 = true
-INF < 2 = true
2   == NaN = false
NaN == NaN = false
```

# Wartości specjalne

- INF i NaN zachowują się zgodnie z zasadami matematyki
  - Każda operacja z NaN w wyniku daje NaN
  - Każde porównanie z NaN daje false (nawet z samym sobą)
- Teoretycznie możnaby je wykorzystywać
  - Wykrywać? Jawnie przypisywać?
- W praktyce oznaczają błąd obliczeń
  - Niespodziewane zero, liczba ujemna, bardzo duża lub bardzo mała
  - Należy im zapobiegać

# Podsumowanie

- Liczby zmiennoprzecinkowe są przybliżeniem liczb rzeczywistych
  - Mają złożoną budowę
  - Mają ograniczony zakres i precyzję
- Warto być świadomym ich cech i ograniczeń
  - Nie każdą liczbę da się zapisać dokładnie
  - Im większa wartość, tym większy bezwzględny błąd
  - Wyniki obliczeń mogą się różnić od spodziewanych na dalszych miejscach po przecinku
  - Nie należy ich porównywać operatorem == ani !=, ale $\pm\varepsilon$
  - Wartości specjalne INF, NaN oznaczają błąd obliczeń

# Ciekawostka: half-float

- Typ zmiennoprzecinkowy 16-bitowy – precyzji „połówkowej”
- Bardzo ograniczone możliwości
  - Zakres: maksimum to 65504
  - Precyzja: ok. 3 cyfr dziesiętnych
  - Wciąż większy zakres i precyzja, niż bajt 0…255
- Brak wsparcia sprzętowego w CPU
  - Wsparcie sprzętowe w nowych GPU
  - Wykorzystywany w grafice do zapisywania kolorów RGB

| Name | Half |
|---|---|
| Bytes | 2 |
| Bits = sign + exponent + significand | 16 = 1 + 5 + 10 |
| Exponent bias, range | 15, −14 … 15 |
| Significant decimal digits | ≈ 3.311 |
| Minimum positive subnormal | $2^{-24} \approx 5.96 \times 10^{-8}$ |
| Minimum positive normal | $2^{-14} \approx 6.10 \times 10^{-5}$ |
| Next value after 1 | $1 + 2^{-10} = 1.0009765625$ |
| Integers represented exactly | $0 \ldots 2^{11} = 2048$ |
| Maximum | $(2-2^{-10}) \times 2^{15} = 65504$ |

Pytania?