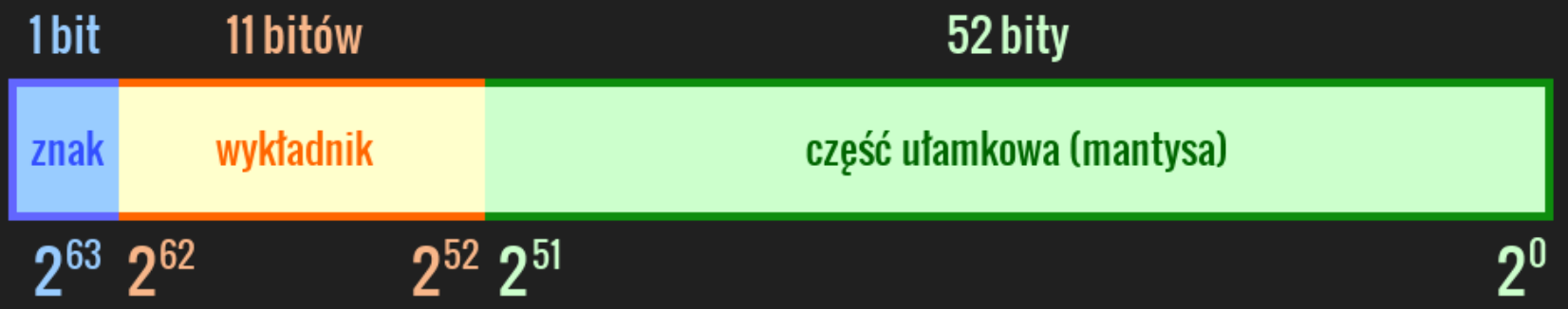
Wszystkie liczby w JavaScript to reprezentacje zmiennoprzecinkowe zapisane w systemie IEEE 754 64-bit (tzw. liczby podwójnej precyzji)

Reprezentacja zmiennoprzecinkowa IEEE 754



bit znaku (sign bit)

0 = liczba dodatnia

1 = liczba ujemna

wykładnik

2 = 1025

1 = 1024

0 = 1023

-1 = 1022

-2 = 1021

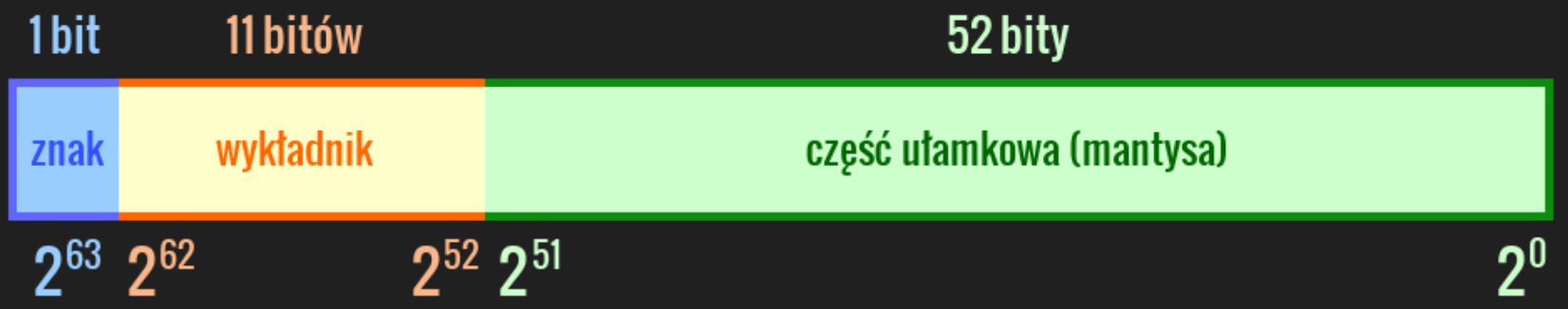
11 bitów:

1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 2047

1 1 1 1 1 1 1 1 1 1 1

$1.$ część ułamkowa $\times 2^{p}$

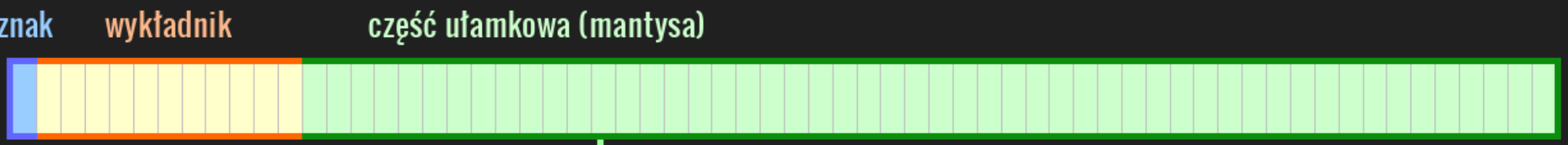
Reprezentacja zmiennoprzecinkowa IEEE 754



Zapiszmy liczbę:

16.7

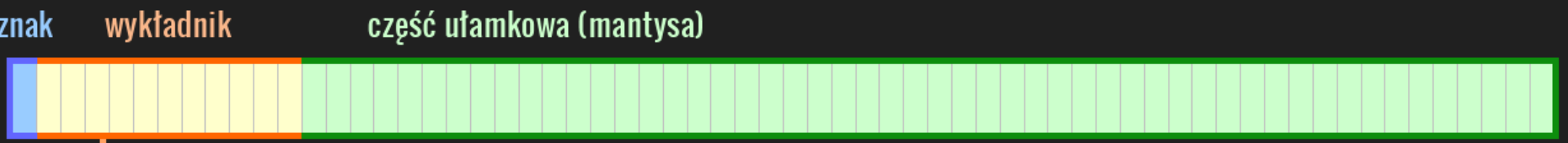
1. część ułamkowa $\times 2^{p}$



część ułamkową zakodujemy w mantysy

$$16.7 = 1.\boxed{04375} \times 2^{4}$$

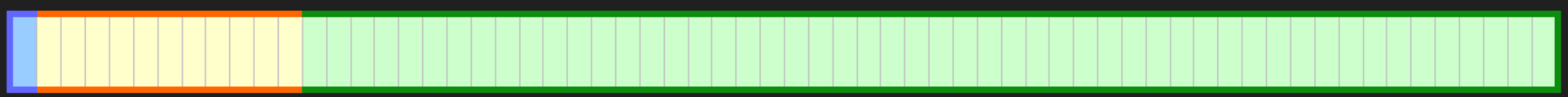
Zapis liczby: $1.$ [część ułamkowa] $\times 2^{p}$



tutaj zakodujemy wykładnik, pamiętając jednak, iż 0 = 1023

$$16.7 = 1.04375 \times 2^{4}$$

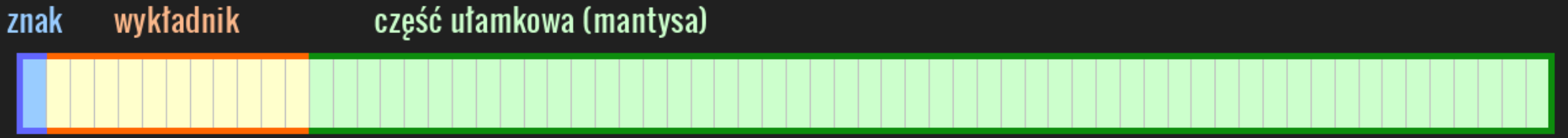
Zapis liczby: $1.$ część ułamkowa $\times 2^{p}$



$$16.7 : 2^{[1]} = 8.35$$
$$8.35 : 2^{[2]} = 4.175$$
$$4.175 : 2^{[3]} = 2.0875$$
$$2.0875 : 2^{[4]} = 1.04375$$

$$16.7 = 1.04375 \times 2^{4}$$

Zapis liczby: $1.$ część ułamkowa $\times 2^{p}$

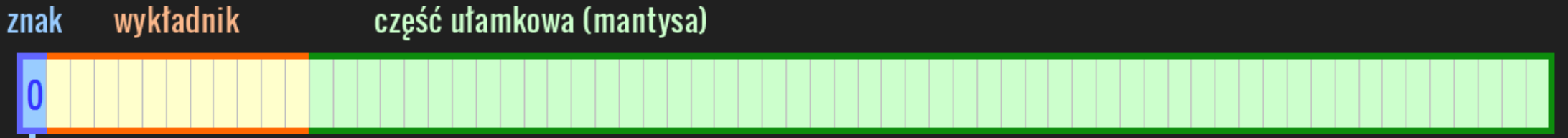


Alternatywny sposób wyznaczenia wykładnika:

$$\log_2(16.7) = 4.06$$

$$16.7 = 1.04375 \times 2^4$$

Zapis liczby: $1.$ część ułamkowa $\times 2^p$



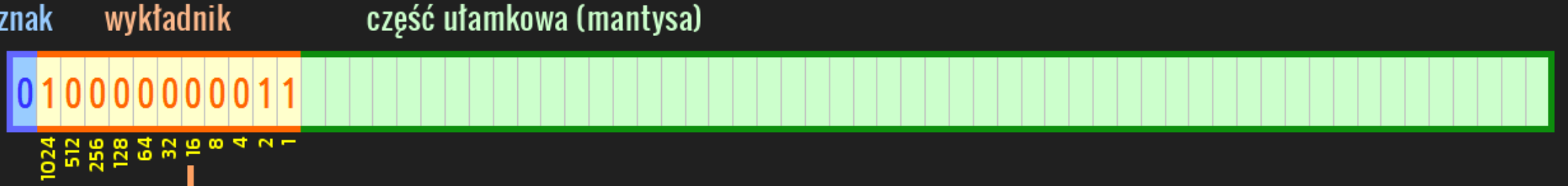
bit znaku
(sign bit)

0 = liczba dodatnia

1 = liczba ujemna

$$16.7 = 1.04375 \times 2^{4}$$

Zapis liczby: $1.$ [część ułamkowa] $\times 2^{p}$

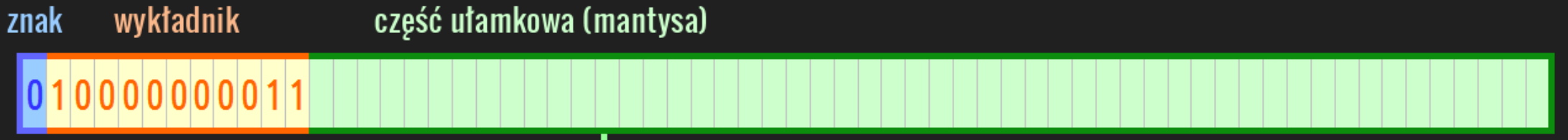


wykładnik u nas równy: 4

1023	wykładnik zerowy
1024	wykładnik = 1
1025	wykładnik = 2
1026	wykładnik = 3
1027	wykładnik = 4

$$16.7 = 1.04375 \times 2^{4}$$

Zapis liczby: $1.$ część ułamkowa $\times 2^{p}$



część ułamkową zakodujemy w mantysy

postępując od bitu
o największej wadze
mnożymy część ułamkową ×2

$$16.7 = 1.\boxed{04375} \times 2^{4}$$

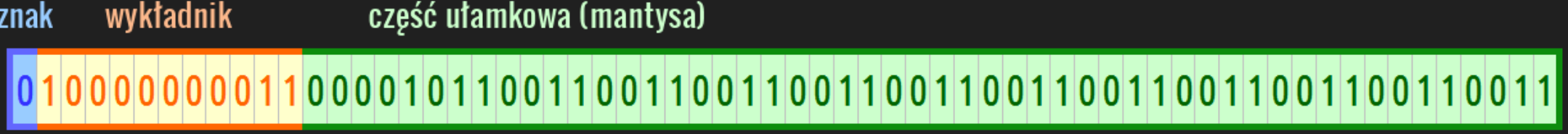
Zapis liczby: $1.$ **część ułamkowa** $\times 2^{p}$

Część ułamkowa	Wartość x 2	Przed przecinkiem
0.04375	0.0875	0
0.0875	0.175	0
0.175	0.35	0
0.35	0.7	0
0.7	1.4	1

Część ułamkowa	Wartość x 2	Przed przecinkiem
0.4	0.8	0
0.8	1.6	1
0.6	1.2	1
0.2	0.4	0
0.4		

$$16.7 = 1.04375 \times 2^{4}$$

Zapis liczby: $1.$ [część ułamkowa] $\times 2^{p}$



Ostatecznie liczba 16.7 przyjęła postać:

0100000000110000101100110011001100110011001100110011001100110011

$$16.7 = 1.04375 \times 2^{4}$$

Zapis liczby: $1.$ **część ułamkowa** $\times 2^{p}$

Jak wielką liczbą całkowitą możemy się posługiwać?

W języku C++ maksymalny czterobajtowy int (ze znakiem) ma następującą wartość:

2147483647

Natomiast w JavaScript liczby całkowite to reprezentacje zmiennoprzecinkowe zapisane w standardzie IEEE 754

Czy w przypadku liczb całkowitych także wystąpią niedokładności (na przykład na 17 miejscu po przecinku)?

Nie! Liczby całkowite aż do 2^{53} są reprezentowane dokładnie!

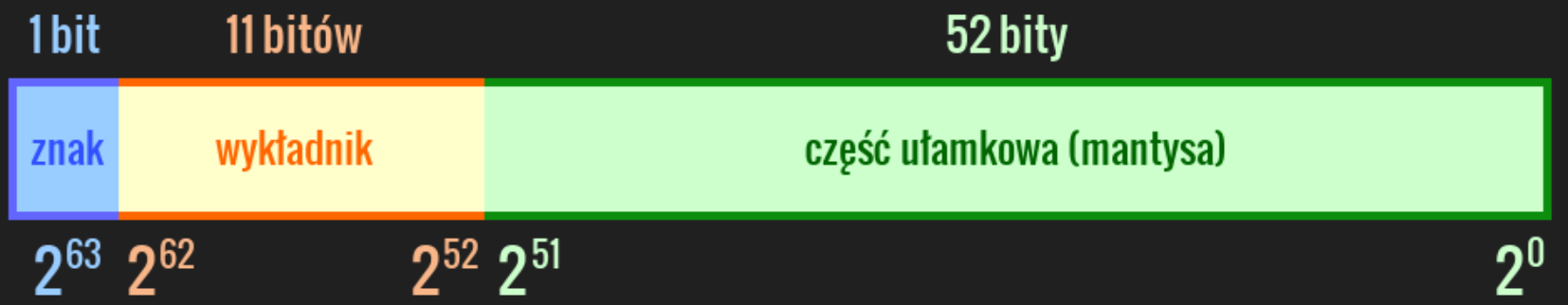
Kiedy tylko można, używa się tzw. natywnych intów, to znaczy rodzimych dla danej architektury

Maksymalny int obsługiwany w JavaScript

$$2^{53} = 9007199254740992$$

to taka liczba całkowita, która sama jest poprawnie reprezentowana w JS (IEEE 754), jak również każda liczba mniejsza od niej jest poprawnie obsługiwana

Dlaczego największy obsługiwany int to 2^{53}?



Mantysa zajmuje 52 bity, więc może 2^{52}
albo $2^{53}-1$ (następna waga minus jeden)

Nie! Reprezentacja zmiennoprzecinkowa IEEE 754 nie składa się przecież tylko z mantysy – to nasz umysł szuka prostej formuły

ZAOKRĄGLANIE

1) Math.round( ) zaokrąglenie do najbliższej wartości
   0..4 zaokrąglenie w dół, 5..9 zaokrąglenie w górę
2) Math.floor( ) zaokrąglenie zawsze w dół
   floor = ang. podłoga
3) Math.ceil( ) zaokrąglenie zawsze w górę
   ceiling = ang. sufit
4) Math.trunc( ) usunięcie części ułamkowej
   truncate = ang. okroić, przyciąć

Sposoby zaokrąglania do 2 miejsc po przecinku

1) Z użyciem funkcji round( )

```
x = Math.round(x * 100) / 100;
```

2) Z użyciem funkcji toFixed( )

```
x = x.toFixed(2);
```

3) Z użyciem własnego prototypu funkcji

```
Number.prototype.round = function(miejsc)
{
    return +(Math.round(this + "e+" + miejsc)  + "e-" + miejsc);
}
```

```
x = x.round(2);
```

atraktor Lorenza

Efekt motyla (ang. butterfly effect) - efekt wrażliwości rozwiązań na małe zaburzenie parametrów (powodujące złożone przyczyny). Nazwa wzięła się od kształtu tzw. atraktora Lorentza albo od anegdoty o trzepocie skrzydeł motyla, wywołującego burzę piaskową

Obiekt Math - wybrane stałe matematyczne

Math.E stała Eulera 2.718281828459045

Math.PI wartość liczby π 3.141592653589793

Math.SQRT2 wartość $\sqrt{2}$ 1.4142135623730951

Math.LN2 watość logarytmu ln(2) 0.6931471805599453

Obiekt Math - funkcje trygonometryczne

Math.sin(x) sinus z liczby x (podanej w radianach)
Math.asin(x) arcus sinus z liczby x (podanej w radianach)

Math.cos(x) cosinus z liczby x (podanej w radianach)
Math.acos(x) arcus cosinus z liczby x (podanej w radianach)

Math.tan(x) tangens z liczby x (podanej w radianach)
Math.atan(x) arcus tangens z liczby x (podanej w radianach)