Lista 3

tags: ASK

#Zadanie 2

Zmiennopozycyjna reprezentacja liczby: $(-1)^S 2^E M$, gdzie E = EXP - bias.

16 bitowe liczby zmiennopozycyjne w standardzie IEEE 754-2008:

- S <- 1 bit
- EXP <- 5
- M <- 10 (wiodąca jedynka jest w domyśle)
- bias = 15

0.15625_{10} = 0.00101_{2} $=_f$ $1.01 * 2^{-3}$

($EXP = E + bias = -3 + 15 = 12 = 1100_2$)

Kodując dostajemy:

Liczba	S	EXP	M
$1.01 * 2^{-3}$	0	01100	0100000000

Połowiczna precyzja (16 bit)

- min: 0 00001 0000000000
- max: 0 11110 1111111111

Zakres: $[6.1 * 10^{-5}, 65504]$ (jeśli zdenormalizowane to minimum wynosi $5.96*10^{-8}$)

Pojedyncza precyzja (32 bit)

- min: 0 00000001 00000000000000000000000
- max: 0 00000000 11111111111111111111111

Zakres: $[1.17510^{-38}, 3.410^{38}]$ (jeśli zdenormalizowane to minimum wynosi $1.4*10^{-45}$)

Zadanie 3

Reprezentacja mantysy w terminach $guard$, $round$, $sticky$

M = 1.BBB..BGRXXX..X liczba bitów B + 1 to rozmiar mantysy, gdzie G(guard) to ostatni bit, który zochowujemy przy zaokrąglaniu. R(round) to pierwszy tracony bit (połówka). X(sticky) to każdy kolejny bit po R.

(bias = 15)

Liczba	S	EXP	M	E
0.3984375	0	01101	1001100000	-2
0.34375	0	01101	0110000000	-2
1771	0	11001	1011101011	10

Dodając od lewej 0.3984375 + 0.34375 $1.1001100000 * 2^{-2} + 1.0110000000 * 2^{-2}$ = $10.1111100000 * 2^{-2}$ = $1.01111100000*2^{-1}$

+1771 $1.01111100000 2^{-1}$ + $1.1011101011 * 2^{10}$ = $0.0000000000101111 2^{10}$ + $1.1011101011 * 2^{10}$ = $1.10111010111011111*2^{10}$

$round$ i $sticky$ to 1, więc zaokrąglamy w górę $1.1011101011 1011111 2^{10} =_r 1.1011101110 2^{10}$ = 1772_{10}

Dodając od prawej 0.34375 + 1771 $1.0110000000 * 2^{-2}$ + $1.1011101011 * 2^{10}$ = $0.000000000001011 * 2^{10}$ + $1.1011101011 * 2^{10}$ = $1.101110101101011 * 2^{10}$ $=r$ $1771{10}$

+0.3984375 $0.0000000000011001 * 2^{10}$ + $1.1011101011 * 2^{10}$ = $1.1011101011011001 * 2^{10}$ $=r$ $1771{10}$

Zadanie 4

```
 int32_t x, y, z;
dx = (double)x;
dy = (double)y;
dz = (double)z;

//Double dokładnie reprezentuje każdą zmienną typu int32_t
```

- ***(float)x == (float)dx*** Wyrażenie zawsze oblicza się do prawdy. Castowanie do $float$ będzie się wiązać z taką samą utratą bitów.
- ***dx - dy == (double)(x-y)*** W operacji x-y może wystąpić zjawisko $overflow$ jeśli odejmujemy liczby z zakresu int32_t (np. x = INT32_MIN, y = 1). Natomiast dla dx-dy wartość wyliczy się poprawnie.
- ***(dx + dy) + dz == dx + (dy + dz)*** Wyrażenie zawsze oblicza się do prawdy. Wyniki tych działań zmieszczą się w reprezentacji double.
- ***(dx * dy) * dz == dx * (dy * dz)*** Wyrażenie może obliczyć sie do fałszu, ponieważ nie jesteśmy w stanie reprezentować dokładnie wyników mnożenia dowolnych dwóch liczb z zakresu typu int, czyli będziemy wykonywać zaokrąglenia. Przykładowo dla wartości $x = 6, y = 2^{23} + 3 + 1, z = 2^{30}$.

Od lewej ``` 6 x 8 388 611

50 331 666 binarnie: 1100000000000000000000010010 <- mieści się w mantysie

```

```

50 331 666 x 1 073 741 825 = 54 043 214 906 130 450 binarnie: 110000000000000000000001001000001100000000000000000000010|010

```

Od prawej
```

1 073 741 825 x 8 388 611 = 9 007 202 484 355 075 binarnie: 1000000000000000000000110000001000000000000000000000001|1 Zaokrąglamy do parzystej: 10000000000000000000001100000010000000000000000000000010 1000000000000000000000110000001000000000000000000000010_2 x 110_2

11000000000000000000010010000011000000000000000000000011|00

```





- ***dx / dx == dz / dz***
Dla x = 0, y = 1 lewa strona oblicza się do $NaN$, a prawa nie.
```