LICZBY ZMIENNOPRZECINKOWE

Liczby zmiennoprzecinkowe są komputerową reprezentacją liczb rzeczywistych zapisanych w formie wykładniczej (naukowej). Aby uprościć arytmetykę na nich, przyjęto ograniczenia zakresu mantysy i eksponenty oraz wprowadzono inne założenia, które reguluje norma IEEE 754 (dla liczb zapisanych w kodzie dwójkowym).
Liczbę zapisuje się jako ciąg zer i jedynek przyjmując umowny podział na "pola":

			w jednostce zmiennoprzecinkowej		
S	**E**	**M**	**G**	**R**	**X/S**
znak	wykładnik w kodzie spolaryzowanym	ułamek	bity specjalne		

podział liczby zmiennoprzecinkowej na pola

S – znak, jest zawsze jedno bitowy i ma wartość 0 jeśli liczba jest dodatnia lub 1 jeśli jest ujemna
E – wykładnik (inaczej: eksponent, cecha), ma długość zależną od długości całej liczby i kodowane jest w kodzie $2^{k-1}-1$
M – moduł (inaczej mantysa) ułamka. Ma wartość z przedziału [1,2). Zapisuje się go bez poprzedzającej go jedynki z kropką (tzw. bit ukryty). Innymi słowy, zapisujemy jedynie część ułamkową modułu przyjmując, że zawsze (pomijając wyjątki, o których później) część całkowita wynosi 1.

Bity specjalne występują jedynie w wewnętrznej reprezentacji liczy w jednostce zmiennoprzecinkowej i wykorzystywane są do zniwelowania efektu utraty dokładności przy wykonywaniu działań, przez zaokrąglenie. Zarówno liczba wejściowa jak i wyjściowa takiego układu nie posiada tych bitów. Oznacza to, że liczby 32/64/128 bitowe, w jednostce zmiennoprzecinkowej, po uzupełnieniu bitami specjalnymi, maja długość odpowiednio 35/67/131 bitów.

G – bit ochrony (guard)
R – bit zaokrąglenia (round)
X -

rozmiar liczby	wykładnik E	moduł ułamka M	nazwa
32	8	23	single
64	11	52	double
128	16	111	extended

długości poszczególnych pól (w bitach)

Wartość reprezentowaną przez liczbę określa się wg wzoru:

$$x = S^{-1} \cdot M \cdot 2^{E}$$

PRZYKŁADY:
a) 0 0100 0100 111 1100 1010 0010 0111 1100 = $1{,}1111\,1001\,0100\,0100\,1111\,100 \cdot 2^{-59} = 1 \cdot 1{,}8168060 \cdot 2^{-59}$
DŁUGOŚĆ: 32 bity
s (1 bit) = 0 (liczba dodatnia)
E (8 bit) = 0100 0100 = 68-127= -59
M = 1,111 1100 1010 0010 0111 $1100_2 = 1{,}8168060_{10}$

b) 1 0100 0100 111 1100 1010 0010 0111 1100 = $-1{,}1111\,1001\,0100\,0100\,1111\,100 \cdot 2^{-59} = -1 \cdot 1{,}8168060 \cdot 2^{-59}$
DŁUGOŚĆ: 32 bity
s (1 bit) = 1 (liczba ujemna)
E (8 bit) = 0100 0100 = 68-127= -59
M = 1,111 1100 1010 0010 0111 $1100_2 = 1{,}8168060_{10}$

c) 0 0111 1111 000 0000 0000 0000 0000 0000 = $1 \cdot 1{,}0 \cdot 2^{0} =$ **1**
DŁUGOŚĆ: 32 bity
s (1 bit) = 0 (liczba dodatnia)
E (8 bit) = 0111 1111 = 127-127 = 0
M = 1,000 0000 0000 0000 0000 $0000_2 = 1{,}0_{10}$

TIPS & TRICKS

czyli jak ułatwić sobie życie (czytaj: obliczenia)

Przekodowanie liczb w systemie 2^{k-1}-1 na U2

Po co? Ano dla tego, że kochamy U2, w nim wszystko jest proste i traktujemy ten kod prawie tak samo naturalnie jak zwykły dziesiętny.
Co nam to ułatwi? Np. dzielenie przez dwa wykonywane przy pierwiastkowaniu.
Więc do rzeczy... Przekodowanie liczby z zachowaniem znaku jest trudniejsze a nam zachowanie znaku nie jest potrzebne (jak dodamy/odejmiemy dwie liczby ujemne lub podzielimy liczbę ujemna a potem zmienimy znów znak to co do modułu utrzymamy ten sam wynik) więc będziemy przekodowywać ze zmianą znaku.

Sprawa jest niezwykle banalna – należy zanegować wszystkie bity poza najstarszym (najmniej znaczącym albo, jak kto woli, tym najbardziej na lewo). Oto przykład:

NB	*2^{k-1}-1*	*U2 (liczba przeciwna)*	*k*
10	$10+127=137_{10}=$**1**0001001	**1**$1110110_{U2}=-10_{10}$	8
-25	$-25+127=102_{10}=$**0**1100110	**0**$0011001_{U2}=25_{10}$	8
0	$0+127=127_{10}=$**0**1111111	**0**$0000000_{U2}=0_{10}$	8

Zmiana znaku liczb w systemie 2^{k-1}-1:

Jak zostało pokazane powyżej, negacja wszystkich bitów poza najstarszym zmienia liczbę na U2 o przeciwnym znaku. Operacja odwrotna również zmienia znak więc możemy wykonać następujący algorytm:
- przekodowanie liczby na U2 (negacja wszystkich bitów poza pierwszym)
- zmiana znaku (negacja wszystkich bitów i dodanie 1)
- przekodowanie na 2^{k-1}-1 (negacja wszystkich bitów poza pierwszym)

Algorytm ten można jednak zoptymalizować. Pierwszy bit zanegowany jest raz a wszystkie pozostałe 3 krotnie. Podwójne zanegowanie nie daje żadnego rezultatu więc ostatecznie wszystkie bity są negowane tylko raz. Po przekodowaniu na U2 dodajemy jedynkę a później negujemy więc w ostatecznym rozrachunku jedynka ta jest odejmowana. Po uwzględnieniu tych faktów napisać można zoptymalizowany algorytm:
- zanegować wszystkie bity
- odjąć 1 od najmłodszego bitu

Przykład:

NB	*2^{k-1}-1*	*zanegowana*	*odjęte 1*	*k*
10	$10+127=137_{10}=10001001$	01110110	$01110101=117-127=-10_{10}$	8
-25	$-25+127=102_{10}=01100110$	10011001	$10011000=152-127=25_{10}$	8
0	$0+127=127_{10}=01111111$	10000000	$01111111=127-127=2_{10}$	8

sposób 2:

W systemie z obciążeniem wartość powiększona jest o liczbę ***O*** (będącą właśnie tym obciążeniem). W związku z tym nasza liczba ***x*** przedstawiona jest jako:

E = x + O

My chcemy znaleźć liczbę ***-x*** na podstawie danej nam liczby E i zapisać ją również w z obciążeniem. Szukana przez nas wartość powinna wynosić:

-x + O = -(x + O) + 2O = -E + 2O = 2O – E

Wynika z tego, że aby zmienić znak liczby w kodzie z obciążeniem musimy ją odjąć od podwojonego obciążenia. Brzmi to trudniej niż jest w rzeczywistości. Dla liczb zmiennoprzecinkowych używamy obciążenia 2^{k-1}-1 (gdzie *k* to liczba bitów przypadających na eksponentę. Dla standardu pojedynczej precyzji (8 bit na wykładnik) obciążenie $O=127_{10}=01111111_2$. Pomnożenie liczby przez 2 to przesunięcie jej o jedną pozycję w lewo więc $2O=11111110_2$. I od tej liczby należy odjąć naszą wartość obciążoną by dostać jej wartość przeciwną.

KODY SPECJALNE

Do pewnych specjalnych zastosowań zarezerwowano kody, w których wykładnik ma wartość minimalną (same zera) lub maksymalną (same jedynki).

ZERO:
Ponieważ przy założeniu, że mantysa jest znormalizowana (ma wartość z przedziału [1,2)), nie da się zapisać zera, na tą liczbę zarezerwowano specjalne kody, w których wszystkie bity poza pierwszym (S) są zerami.
Wyróżnia się więc **zero dodatnie** (przykład dla liczb 32-bitowych):
+0 = 0 0000 0000 000 0000 0000 0000 0000 0000

I **zero ujemne** (przykład dla liczb 32-bitowych):
-0 = 1 0000 0000 000 0000 0000 0000 0000 0000

LICZBY BLISKIE ZERU:
Kolejną konsekwencją normalizacji mantysy jest to, że nie da się zapisać liczb z przedziału $(-2^{Emin}, +2^{Emin})$. Na potrzeby tych liczb, zarezerwowano kody, w których wykładnik jest równy zero ale mantysa jest różna od zera. W przypadku tych liczb, przyjmuje się, że mantysa ma bit ukryty równy 0 a nie 1 ($0<M<1$), wykładnik jest natomiast równy $-2^{k-1}+2$.

Przykłady:
a) 0 0000 0000 111 1100 1010 0010 0111 1100 = $1 \cdot 0{,}8168060 \cdot 2^{-126}$
M = 0,111 1100 1010 0010 0111 1100_2 = **0**,8168060_{10}
E (8 bit) = 0000 0000 = -126 !!

a) 1 0000 0000 111 1100 1010 0010 0111 1100 = $-1 \cdot 0{,}8168060 \cdot 2^{-126}$
M = 0,111 1100 1010 0010 0111 1100_2 = **0**,8168060_{10}
E (8 bit) = 0000 0000 = -126

NIE-LICZBY:
Nie-liczby (NaN – Not A Number) to kody reprezentujące wyniki niedające się zakodować w postaci liczby zmiennoprzecinkowej (np. wynik pierwiastkowania kwadratowego liczby ujemnej).
Nie-liczbę kodujemy za pomocą liczby z wykładnikiem składającym się z samych jedynek i modułem różnym od zera.
Wyróżnia się nie-liczby ciche (QNaN) wytwarzane sprzętowo oraz nie-liczby sygnalizacyjne (SNaN) generowane w programie obsługi wyjątku. Pierwsze charakteryzują się kodem ułamka zaczynającym się od 1xxxx a drugie kodem ułamka zaczynającym się od 0xxxx.

Przykład kodów NaN (dla liczb 32-bitowych):
1 1111 1111 111 1111 1111 1111 1111 1111 (QNaN)
0 1111 1111 000 0000 0000 0000 0000 0001 (SNaN)

NIESKOŃCZONOŚCI:
Za pomocą liczb zmiennoprzecinkowych można również zakodować nieskończoności. Kodem zarezerwowanym dla tych wartości jest wykładnik składający się z samych jedynek i moduł z samych zer.

Wyróżnia się **minus nieskończoność** (przykład dla kodu 32-bitowego):
$-\infty$ = 1 1111 1111 000 0000 0000 0000 0000 0000

Plus nieskończoność (przykład dla kodu 32-bitowego):
∞ = 0 1111 1111 000 0000 0000 0000 0000 0000

PODSUMOWANIE:

Wykładnik	*Mantysa*	*Rodzaj liczby*
00....00	00....00	zero
00....00	>0	Liczby bliskie zeru
11....11	1x....xx	QNaN
11....11	0x....xx	SNaN
11....11	00....00	Nieskończoność

DZIAŁANIA ARYTMETYCZNE NA LICZBACH ZMIENNOPRZECINKOWYCH

Aby zrozumieć algorytmy działań arytmetycznych na liczbach zmiennoprzecinkowych należy wyobrazić je sobie w postaci wykładniczej:

$$x=M*B^E$$

i odnieść się do praw arytmetyki działających na liczbach takiej postaci.

DODAWANIE/ODEJMOWANIE:

Z postaci liczb zmiennoprzecinkowych wynika lekka komplikacja przy operacjach dodawania/odejmowania. Aby je wykonać należy najpierw sprowadzi składniki do wspólnego wykładnika. Weźmy dwie liczby:

$$x_1=M_1*B^{E1}$$
$$x_2=M_2*B^{E2}$$

Jako, że dodawanie jest przemienne (a odejmowanie to też dodawanie) przyjmijmy, że E1>E2. Wynik otrzymujemy wg wzoru:

$$x_1 \pm x_2 = M_1*B^{E1} + M_2*B^{E2} = (M_1*B^{E1-E2} + M_2)*B^{E2}$$

Po wykonaniu tych działań może okazać się konieczna normalizacja wyniku.

MNOŻENIE:

W przeciwieństwie do dodawania/odejmowania mnożenie liczb w postaci zmiennoprzecinkowej jest banalne. Dla liczb:

$$x_1=M_1*B^{E1}$$
$$x_2=M_2*B^{E2}$$

wynik mnożenia wynosi:

$$x_1*x_2 = (M_1*B^{E1}) * (M_2*B^{E2}) = (M_1*M_2)*(B^{E1}*B^{E2}) = (M_1*M_2)*B^{E1+E2}$$

Po wykonaniu tych działań może okazać się konieczna normalizacja wyniku

DZIELENIE:

Dzielenie jest bardzo podobne do mnożenia:

$$x_1/x_2 = (M_1*B^{E1}) / (M_2*B^{E2}) = (M_1/M_2)*(B^{E1} / B^{E2}) = (M_1 / M_2)*B^{E1-E2}$$

Po wykonaniu tych działań może okazać się konieczna normalizacja wyniku.

OBLICZANIE ODWROTNOŚCI:

Weźmy liczbę:

$$x=M*B^E$$

jej odwrotność wynosi:

$$1/x = x^{-1} = (M*B^E)^{-1} = M^{-1} * B^{-E} = 1/M * B^{-E}$$

Wynika z tego, że musimy osobno obliczyć odwrotność M i B^E. Policzenie odwrotności ostatniej nie powinno stanowić problemu ponieważ polega jedynie na zmianie znaku wykładnika (opisane wcześniej). Trochę gorzej jest z odwrotnością mantysy.

Aby obliczyć odwrotność mantysy wykorzystamy własność: $(1+x)^{-1} \approx 1-x \quad dla \quad x \approx 0$

Nasza mantysa jest liczbą w postaci 1+X (gdzie X to odczytana wprost wartość mantysy bez uwzględnienia ukrytej jedynki). Możemy więc rozpatrzyć z osobna dwa przypadki:

- $X \approx 0$:

Jeśli tak to możemy bezpośrednio skorzystać z podanego wyżej prawa i zapisać

$$M^{-1}=(1+X)^{-1}=1-X$$

Obliczenie odwrotności możemy więc obliczyć w 3 krokach:

- negacja wszystkich bitów mantysy
- dodanie 1 na najmłodszym bicie
- normalizacja otrzymanej odwrotności (otrzymamy liczbę w postaci 0,xxxx więc trzeba przeskalować ją do postaci 1,xxxx jednocześnie odpowiednio zmniejszając wykładnik)

- $X \approx 1$

W takim wypadku możemy mantysę zapisać w postaci ***2-X*** zamiast ***1+X***, w związku z tym nasza liczba ma wartość:

$$(2-X)\cdot 2^{E} = 2\cdot(1-X)\cdot 2^{E} = (1-X)\cdot 2^{E+1}$$

A jej odwrotność:

$$[(1-X)\cdot 2^{E+1}]^{-1} = (1+X)\cdot 2^{-(E+1)}$$

Wynika z tego również, że liczba wynikowa będzie już znormalizowana. Dla nas oznacza to jednak, że aby obliczyć odwrotność mantysy musimy jedynie:

– podzielić mantysę przez 2 (przesunąć o jedną pozycję w prawo) uwzględniając bit ukryty
– dodaj 2 do mantysy i zaneguj ją
– w wyniku otrzymujemy liczbę postaci 1,xxxx więc nie potrzebna jest normalizacja.

Poznane metody są przybliżonymi i będą sprawdzać się jedynie przy liczbach bliskich minimum i maksimum wartości mantysy. W innych nie możemy ich stosować ale na dziś nie chce mi się więcej pisać.

OBLICZANIE PIERWIASTKA KWADRATOWEGO:

Liczby zmiennoprzecinkowe mają za zadanie przechowywać liczby rzeczywiste więc pierwiastkowanie liczb ujemnych jest niewykonalne. W związku z tym, pierwszy bit (bit znaku S) powinien być równy 0. W przeciwnym wypadku wynikiem działania będzie zawsze NaN. Jego postać zależy od implementacji.

W przypadku, gdy wykładnik jest parzysty sprawa jest prosta:

$$\sqrt{M\cdot B^{2n}} = \sqrt{M}\ \cdot\ B^{n}$$

Gorzej jest gdy wykładnik nie jest parzysty ponieważ nie możemy łatwo spierwiastkować B^{E}. W takim wypadku możemy przeskalować mantysę zmniejszając lub zwiększając wykładnik o jeden (dzięki czemu będzie on parzysty). Lepszym pomysłem jest zmniejszenie wykładnika dzięki czemu w wyniku pierwiastkowania otrzymamy już liczbę znormalizowaną i nie trzeba będzie jej znów skalować.

Przykłady:

a) $\sqrt{\mathbf{0\ 0100\ 0100\ 111\ 1100\ 1010\ 0010\ 0111\ 1100}} = \sqrt{1{,}111\ \ 1100\ 1010\ 0010\ 0111\ 1100\cdot 2^{-59}}$

Wartość liczby zmiennoprzecinkowej:
E (8 bit) = 0100 0100 = 68-127= -59
M = 1,111 1100 1010 0010 0111 1100
Wykładnik jest nieparzysty co oznacza, że nie możemy z niego łatwo wyciągnąć pierwiastka więc przeskalujemy mantysę. Zwiększy to wykładnik nie zmieniając wartości liczbywięc możemy w ten sposób doprowadzić wykładnik do liczby parzystej.

$$\sqrt{1{,}111\,1100\,1010\,0010\,0111\,1100\ \cdot\ 2^{-59}} = \sqrt{11{,}1111\,0010\,1000\,1001\,1111\,00\ \cdot\ 2^{-60}} =$$
$$= \sqrt{11{,}1111\,0010\,1000\,1001\,1111\,00}\ \cdot\ \sqrt{2^{-60}} = 1{,}1111\,100\ \cdot\ 2^{-30} =$$
$$= \mathbf{0\,0110\,0001\,1111\,1000\,0000\,0000\,0000\,0000}$$

Pierwiastek modułu przeskalowanego o 1 w prawo:

```
1  1, 1  1  1  1  0  0  1  0  1  0  0  0  1  0  0  1  1  1  1  1  =  1, 1  1  1  1  1  0  0
-  1
1  0  1  1
-  1  0  1
   1  1  0  1  1
   -  1  1  0  1
      1  1  1  0  0  0
      -  1  1  1  0  1
         1  1  0  1  1  1  0
         -  1  1  1  1  0  1
            1  1  0  0  0  1  1  0
            -  1  1  1  1  0  0  1
               1  0  0  1  1  0  1  0  0
               -  1  1  1  1  1  1  0  1
                  0  0  1  1  0  1  1  1  1  0  0  0  1  1  1
```

nowy wykładnik:
$-30+127 = 97 = 0110\ 0001_2$

b) $\sqrt{\mathbf{0\ 0100\ 0101\ 111\ 1100\ 1010\ 0010\ 0111\ 1100}} = \sqrt{1{,}111\ 1100\ 1010\ 0010\ 0111\ 1100 \cdot 2^{-58}}$

Wartość liczby zmiennoprzecinkowej:
E (8 bit) = 0100 0101 = 69-127= -58
M = 1,111 1100 1010 0010 0111 1100

Tutaj wykładnik jest parzysty więc po prostu wykonamy zwykłe pierwiastkowanie na module i na spotęgowanej podstawie systemu (czyli dwójce):

$\sqrt{1{,}111\ 1100\ 1010\ 0010\ 0111\ 1100 \cdot 2^{-58}} = 1{,}011001 \cdot 2^{-29} =$
$= \mathbf{0\ 0110\ 0010\ 0110\ 0100\ 0000\ 0000\ 0000}$

Wynik jest już znormalizowany więc nie musimy go skalować.

Pierwiastek z modułu:

1, 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 = 1, 0 1 1 0 0 1

1

0 **1 1 1 1**

\- 1 0 *0* *1*

0 1 1 0 **1 0**

\- 1 0 1 *0* *1*

0 0 1 0 1 **0 1 0 1 0 0**

\- 1 0 1 1 0 0 *0* *1*

1 0 1 0 0 0 1 1

nowy wykładnik:
$-29+127 = 98 = 0110\ 0010_2$