Matematyka dyskretna

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytet Gdański

sem. zimowy 2025/26

inf.ug.edu.pl/~amb/MaD

Reprezentacja liczb całkowitych w komputerze

Liczby całkowite

- `unsigned integer` – dawniej dwa bajty, dzisiaj cztery
  32 bity interpretowane jako współczynniki wielomianu
  $[[w]] = \Sigma_{i=0}^{31} w_i \cdot 2^i$, $0 \leqslant [[w]] < 2^{32}$
- uwaga: stosowane są dwie architektury, „low endian" i „big endian"
  w zależności od kolejności odczytywania bitów

Stała liczba bitów

zamiana na układ dwójkowy
(schemat Hornera)

```
zapisz=a; j=0;
while (zapisz>0) {
    a[j]=zapisz%2;
    zapisz=(zapisz)/2;
    j=j+1; }
```

zamiana na układ dwójkowy
32-bitowy

```
zapisz=a; j=0;
while (j<32) {
    a[j]=zapisz%2;
    zapisz=(zapisz)/2;
    j=j+1; }
```

dzielenie jest całkowitoliczbowe (bez reszty)

- schemat Hornera jest tylko dla liczb dodatnich
- może się okazać, że wyprodukujemy wiele zer na początku
  może się okazać, że po zakończeniu $zapisz \neq 0$
- np. $a = -1$ spowoduje, że zawsze $zapisz = -1$ i $a[j] = 1$
  również $a = 2^{32} - 1$ wyprodukuje 32 jedynki, aż $zapisz = 0$
- dwie liczby różniące się o wielokrotność 2^{32} wyprodukują ten sam ciąg 32 bitów

Kodowanie liczb ujemnych

- kod uzupełnień do dwóch (U2): używany w implementacji `integer`
  - $-x$ jest reprezentowany przez $2^N - x$
  - zakres trochę niesymetryczny $[-2^{N-1}, 2^{N-1} - 1]$
  - zero jest liczbą nieujemną
  - łatwa implementacja działań arytmetycznych
  - najczęściej stosowany sposób zapisu liczb całkowitych
- `integer` – najbardziej znaczący bit jest liczbą ujemną
  tzn. $[[w]] = \Sigma_{i=0}^{30} w_i \cdot 2^i - w_{31} \cdot 2^{31}$, $\quad -2^{31} \leqslant [[w]] < 2^{31}$

01111111111111111111111111111111	$2^{31} - 1$
...	
00000000000000000000000000000001	1
00000000000000000000000000000000	0
11111111111111111111111111111111	-1
...	
10000000000000000000000000000000	-2^{31}

Inne rozwiązania

- kod uzupełnień do jedności (U1):
  - liczba $-x$ ma bity odwrócone z liczby x
  - zakres symetryczny $[-2^{N-1} - 1, 2^{N-1} - 1]$
  - są dwa zera, dodatnie i ujemne
  - dla liczb ujemnych różnica obu reprezentacji jest 1, dla dodatnich nie ma różnicy.
- kod znak-moduł (ZM): dokładnie tak jak w życiu codziennym, bit na znak i osobno wartość bezwzględna
  - zakres symetryczny $[-2^{N-1} - 1, 2^{N-1} - 1]$
  - są dwa zera, dodatnie i ujemne
  - skomplikowane algorytmy dla operacji arytmetycznych
  - dla liczb rzeczywistych jedyna możliwa idea
  - stosowana w pierwszych komputerach, potem rzadziej
- kod z przesunięciem: $[[x]] = x - C$, gdzie C jest ustaloną stałą, np. $C = 2^{N-1} - 1$ lub $C = 2^{N-1}$

Inne i jeszcze inne (teoretyczne) rozwiązania

ciąg bitów	unsigned	$U2$	$U1$	ZM	*shift* 4	*shift* 3
111	7	-1	0	-3	3	4
110	6	-2	-1	-2	2	3
101	5	-3	-2	-1	1	2
100	4	-4	-3	0	0	1
011	3	3	3	3	-1	0
010	2	2	2	2	-2	-1
001	1	1	1	1	-3	-2
000	0	0	0	0	-4	-3

- ujemna podstawa liczenia, tzn. -2
  - zakres asymetryczny: $-\frac{2}{3}(2^{N-1} - 1), \frac{1}{3}(2^{N-1} - 1)$
  - jedno zero
- ujemne cyfry: możliwe w kontekście, gdy cyfry są wielobitowe

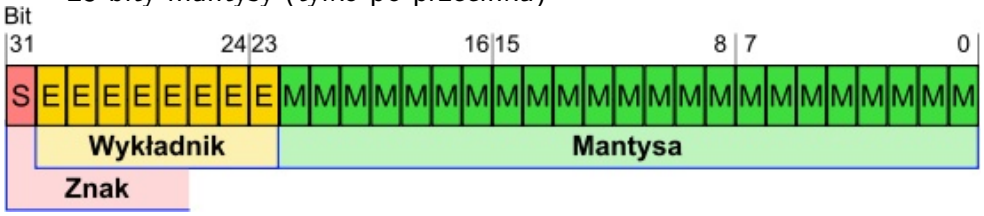
Reprezentacja liczb rzeczywistych w komputerze

Liczby rzeczywiste, standard IEEE 754

- `real` (`float`) – reprezentacja zmiennoprzecinkowa
- każda liczba rzeczywista $\neq 0$ da się przedstawić jako

$$s \cdot m \cdot 2^e$$

  $s \in \{1, -1\}$, $1 \leqslant m < 2$, e jest liczbą całkowitą
- osobno zapisujemy znak s, mantysę m oraz wykładnik e
- standard IEEE 754:
  najbardziej znaczący bit: znak, $0 = +$, $1 = -$
  8 bitów wykładnika: $-128 < e \leqslant 128$, przesunięcie o 127
  23 bity mantysy (tylko po przecinku)



http://pl.iiuj.wikia.com/wiki/Plik:550px-IEEE-754-single1.jpg

standard IEEE 754

- wykładnik jest zapisywany z przesunięciem 127, tzn.
  $00000000 = -127$, $01111111 = 0$, $10000000 = 1$, $11111111 = 128$
- skrajne wykładniki mają specjalne znaczenie
  $2^{-127} = 0$, o ile mantysa też zero, jest zero dodatnie i ujemne
  $2^{128} = \pm\infty$
  mantysa w tych przypadkach powinna być zerowa, inna oznacza, że nie jest to liczba rzeczywista, lecz błąd (NaN)
- 23 bity dają 7 miejsc znaczących dziesiętnych
  zakres liczb $\pm 10^{-38} \leqslant x \leqslant \pm 3.4 \cdot 10^{38}$
- podwójna precyzja (`double`) – 1+11+52=64 bity
  zakres $\pm 2 \cdot 10^{-308} \leqslant x \leqslant \pm 2 \cdot 10^{308}$
  dokładność – 16 cyfr znaczących
- albo 1+15+64=80 bitów (`long double`/`extended`)
- albo 1+15+112=128 bitów
- albo tylko 16 bitów

Przykład

- $(0.00727)_{10} = 1.86112 \cdot 2^{-8} \quad \Rightarrow \quad + \quad -8 \quad 0.86112$
  $+1 = (-1)^0 \quad -8 + 127 = 119 = (01110111)_2$
  0 01110111 11011100011100100101101

```
x=0.00727; e=0;
if (x<1) while (x<1) {x*=2;e--;}
else while (x>=2) {x/=2;e++;}
// jeszcze zamiana e+127 na bity
x-=1;
for (i=0;i<23;i++) {
   x*=2;
   if (x>1) {printf "1";x-=1;}
   else printf "0";}
```

- $(-727)_{10} = -1.419922 * 2^9 \quad \Rightarrow \quad - \quad 9 \quad 0.419922$
  1 10001000 01101011011111111111111

Problemy reprezentacji zmiennoprzecinkowej

- błędy zaokrągleń
  - jest kilka standardów zaokrąglania:
    w kierunku do zera, od zera, do najbliższej parzystej
- kolejność wykonywania obliczeń gra rolę!
- przepełnienie – wynik ma wykładnik większy niż np. 127
- niedomiar – wynik ma wykładnik mniejszy niż np. -126
- ale dzielenie przez ± 0 daje $\pm\infty$
- odejmowanie dwóch bliskich liczb zmniejsza liczbę cyfr znaczących
- analiza numeryczna zajmuje się m.in. problemem tych błędów

Błędy zaokrągleń

- liczba a ma reprezentację $\bar{a}$ z błędem ε, czyli $\bar{a} = a(1 + \varepsilon_a)$
- działania arytmetyczne powielają błędy argumentów i wprowadzają nowe
- $a(1 + \varepsilon_a) - b(1 + \varepsilon_b) = (a - b)(1 + \frac{a\varepsilon_a - b\varepsilon_b}{a-b})$
  $\overline{a - b} = (a - b)(1 + \varepsilon_{a-b})$, $\varepsilon_{a-b} = \frac{a}{a-b}\varepsilon_a - \frac{b}{a-b}\varepsilon_b + \varepsilon_-$
- $\varepsilon_{ab} = \varepsilon_a + \varepsilon_b + \varepsilon_*$
- błędy dla dodawania/odejmowania mogą być groźniejsze niż dla mnożenia/dzielenia