



ELEMENTY ANALIZY NUMERYCZNEJ

Andrzej Marciniak

Politechnika Poznańska, Instytut Informatyki

WSTĘP DO PODSTAW ELEMENTÓW ANALIZY NUMERYCZNEJ

Andrzej Marciniak

Politechnika Poznańska, Instytut Informatyki

ELEMENTY ANALIZY NUMERYCZNEJ

- ★ arytmetyka przedziałowa i jej realizacja w języku Delphi Pascal
- ★ podstawowe pojęcia analizy numerycznej
- ★ realizacja obliczeń na wielomianach i funkcjach wymiernych
- ★ interpolacja
- ★ układy równań liniowych
- ★ równania i układy równań nieliniowych

ANALIZA NUMERYCZNA

ELEMENTY ANALIZY NUMERYCZNEJ +

- ★ obliczanie wyznacznika i odwracanie macierzy
- ★ aproksymacja
- ★ różniczkowanie numeryczne
- ★ całkowanie numeryczne
- ★ równania różniczkowe zwyczajne
- ★ równania różniczkowe cząstkowe

ELEMENTY ANALIZY NUMERYCZNEJ

Organizacja przedmiotu:

- wykład 24 godziny (12×2 godziny)
- ćwiczenia 24 godziny (12×2 godziny)

Zaliczenie przedmiotu:

- ćwiczenia – program (aplikacja) + sprawdzian
- wykład – egzamin

ELEMENTY ANALIZY NUMERYCZNEJ

Literatura:

- J. i M. Jankowscy, *Przegląd metod i algorytmów numerycznych*, Cz.1, WNT
- J. Stoer, R. Bulirsch, *Wstęp do analizy numerycznej*, PWN
- Z. Fortuna, B. Macukow, J. Wąsowski, *Metody numeryczne*, WNT
- A. Ralston, *Wstęp do analizy numerycznej*, PWN
- A. Marciniak, D. Gregulec, J. Kaczmarek, *Podstawowe procedury numeryczne w języku Turbo Pascal*, NAKOM
- D. Kincaid, W. Cheney, *Analiza numeryczna*, WNT

Na tym wykładzie ...

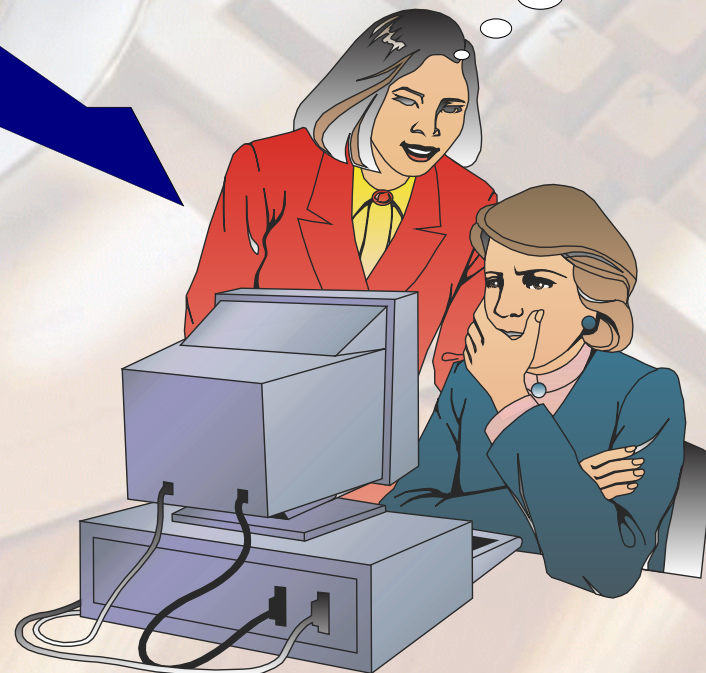
- ★ przykłady obliczeń w arytmetyce zmiennopozycyjnej
- ★ zmiennopozycyjne reprezentacje liczb
- ★ podstawy teoretyczne arytmetyki przedziałowej
- ★ realizacja zmiennopozycyjnej arytmetyki przedziałowej
 - ★ rodzaje zaokrągleń
 - ★ operacje zmiennopozycyjne na przedziałach
 - ★ przedziałowe reprezentacje liczb

Zadanie $y = f(x)$

?



„Otrzymałem wynik $y(?)$ ”



„Za pomocą metody M otrzymałem
wynik $y \pm \Delta y$. Metodę M wybrałem,
bo $a, b, c \dots$.”

Przykłady obliczeń

Przykład 1

$$\mathbf{A} = \left[\frac{1}{j + k - 1} \right]_{j, k = 1(1)40}$$

Teoretycznie: $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$

Obliczenia w typie *Extended*:

$$\mathbf{A}\mathbf{A}^{-1} = \begin{bmatrix} 0,99999999922874849 & 0,00000101141631603 & \dots & 1,7500000000000000 & -0,1250000000000000 \\ 0,000000000154250301 & 1,00000053085386753 & \dots & 3,7500000000000000 & 0,2500000000000000 \\ \dots & \dots & \dots & \dots & \dots \\ 0,00000000064028427 & 0,00000014435499907 & \dots & 3,2500000000000000 & 0,1250000000000000 \\ 0,00000000080763130 & 0,00000011082738638 & \dots & 1,7500000000000000 & 0,8125000000000000 \end{bmatrix}$$

Dlaczego taki wynik?

Wskaźnik uwarunkowania macierzy $\mathbf{A}$:

$$\|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \approx 2,717 \cdot 10^{11}$$

Przykłady obliczeń

Przykład 2

Dane są dwa wektory:

$$\mathbf{x} = [10^{20}, 1223, 10^{18}, 10^{15}, 3, -10^{12}]^T,$$

$$\mathbf{y} = [10^{20}, 2, -10^{22}, 10^{13}, 2111, 10^{16}]^T.$$

Ich iloczyn skalarny (dokładny) jest równy:

$$\mathbf{x} \cdot \mathbf{y} = 10^{40} + 2446 - 10^{40} + 10^{28} + 6333 - 10^{28} = 8779.$$

W arytmetyce zmiennopozycyjnej na każdym komputerze otrzymamy wartość 0 (zero)!

Dlaczego?

Przykłady obliczeń

Przykład 3

Rozważmy system zmiennopozycyjny o podstawie 10 i mantysie o długości 5. cyfr (dla uproszczenia) i obliczmy w nim różnicę dwu liczb

$$x = 0,10005 \cdot 10^5 \text{ i } y = 0,99973 \cdot 10^4.$$

Operandy są podobnego rzędu i wykonując obliczenia na komputerze otrzymujemy całkiem poprawny wynik:

$$x - y = \underline{0,77000 \cdot 10^1}.$$

Przypuśćmy teraz, że liczby x i y są wynikiem dwóch uprzednio wykonanych mnożeń, których niezaokrąglone wartości są równe

$$x = x_1 \cdot x_2 = 0,1000548241 \cdot 10^5 \text{ i } y = y_1 \cdot y_2 = 0,9997342213 \cdot 10^4.$$

Jeśli odejmiemy te dwie liczby, znormalizujemy wynik i zaokrąglimy go do 5. miejsc dziesiętnych, to otrzymamy $0,81402 \cdot 10^1$.

Standardowa arytmetyka zmiennopozycyjna z zaokrągleniem po każdym działaniu prowadzi do błędnych wyników!

Przykłady obliczeń

Przykład 4

Teoretycznie: $(a + b) + c = a + (b + c) = b + (a + c)$

Wykonując obliczenia w arytmetyce zmiennopozycyjnej może okazać się, że $\text{fl}((a + b) + c) \neq \text{fl}(a + (b + c)) \neq \text{fl}(b + (a + c))$.

W jakiej kolejności należy dodać trzy liczby, aby zminimalizować błędy zaokrągleń?

Przykład 5

Teoretycznie: $a^2 - b^2 = a \cdot a - b \cdot b = (a + b)(a - b)$

W arytmetyce zmiennopozycyjnej może być $\text{fl}(a \cdot a - b \cdot b) \neq \text{fl}((a + b)(a - b))$.

Który sposób obliczania różnicy kwadratów jest lepszy?

Zmiennopozycyjne (maszynowe) reprezentacje liczb

Reprezentacja maszynowa liczby rzeczywistej x może być różna od tej liczby, tj.

$$x \neq \text{rd}(x) = x(1 + \varepsilon), \text{ gdzie } |\varepsilon| \leq \text{eps} = 2^{-t}.$$

Liczbę eps nazywa się **dokładnością maszynową**, przy czym t oznacza liczbę bitów przeznaczoną do zapisu mantysy (tutaj: w systemie dwójkowym).

Zmiennopozycyjne (maszynowe) reprezentacje liczb

W języku Delphi Pascal w postaci zmiennopozycyjnej zapisywane są wartości pięciu typów rzeczywistych:

- ★ ***Real48*** (na 6. bajtach, mantysa ze znakiem zajmuje 40 bitów),
- ★ ***Single*** (na 4. bajtach, mantysa ze znakiem zajmuje 25 bitów),
- ★ ***Real*** (w 32-bitowej implementacji języka) = ***Double*** (na 8. bajtach, mantysa ze znakiem zajmuje 53 bity),
- ★ ***Extended*** (na 10. bajtach, mantysa ze znakiem zajmuje 65 bitów).

Typy rzeczywiste *Comp* i *Currency* są typami stałopozycyjnymi.

Zmiennopozycyjne (maszynowe) reprezentacje liczb

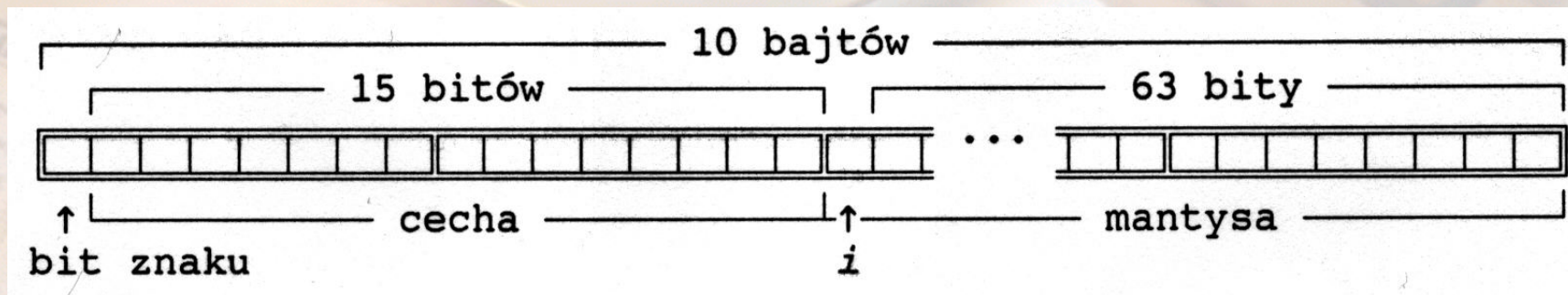
Ze standardem IEEE (skr. ang. *Institute of Electrical and Electronics Engineers*) są zgodne typy **Single** (typ rzeczywisty o pojedynczej precyzji) i **Double** (typ rzeczywisty o podwójnej precyzji).

Typ **Extended** można nazwać **typem rzeczywistym o poszerzonej precyzji** (z uwagi na liczbę bitów przeznaczoną do zapisu mantysy). Typ ten ma też większy zakres niż typ **Double** (z uwagi na większą liczbę bitów przeznaczoną do zapisu cechy).

W programach numerycznych pisanych w języku Delphi Pascal należy stosować typ *Extended*.

Zmiennopozycyjne (maszynowe) reprezentacje liczb

Zapis wartości typu *Extended*:



Wartość w liczby jest określona następująco:

$$w = \begin{cases} (-1)^z 2^{(c-16383)} (i.m), & 0 \leq c < 32767, \\ (-1)^z Inf, & c = 32767, \quad m = 0, \\ NaN, & c = 32767, \quad m \neq 0, \end{cases}$$

gdzie i oznacza cyfrę przed kropką w mantysie (0 lub 1).

Zmiennopozycyjne (maszynowe) reprezentacje liczb

Przykład (cd.)

Wewnętrzna (znormalizowana) wartość dwójkowa cechy:

011111111111011

$$= 0 \cdot 2^{14} + 1 \cdot 2^{13} + 1 \cdot 2^{12} + 1 \cdot 2^{11} + 1 \cdot 2^{10} + 1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 \\ + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Wewnętrzna (znormalizowana) wartość dziesiętna cechy: **16379**

Wartość dziesiętna cechy po zdenormalizowaniu: **-4**

Mantysa w zapisie dwójkowym:

1,100110011001100110011001100110011001100110011001100110011001101

Liczba w zapisie dwójkowym (po pomnożeniu przez 2^{-4} i uwzględnieniu znaku +, bo bit znaku jest równy 0):

0,000110011001100110011001100110011001100110011001100110011001101

=



Zmiennopozycyjne (maszynowe) reprezentacje liczb

Przykład (cd.)

Następne liczby rzeczywiste reprezentowane dokładnie:

0,100000000000000000000081315162936412832550558960065245628356933593750

Write (liczba:27) : 1.0000000000000000000000E-0001

0,1000000000000000000000149077798716756859676024760119616985321044921875

Write (liczba:27) : 1.0000000000000000000000E-0001

...

0,10000000000000000000004960224939121182785584096563979983329772949218750

Write (liczba:27) : 1.0000000000000000000000E-0001

0,10000000000000000000005027987574901526812709562364034354686737060546875

Write (liczba:27) : 1.0000000000000000000000E-0001

...

0,10000000000000000000005434563389583590975462357164360582828521728515625

Write (liczba:27) : 1.0000000000000000000000E-0001

EAN(20 z 74)

7 liczb błędnie zaokrąglanych?

Podstawy teoretyczne arytmetyki przedziałowej

Przedziałem rzeczywistym nazywamy domknięty i ograniczony zbiór liczb rzeczywistych $\mathbf{R}$

$$[x] = [\underline{x}, \bar{x}] = \{x \in \mathbf{R}: \underline{x} \leq x \leq \bar{x}\}.$$

Średnica (szerokość), promień i punkt środkowy przedziału $[x]$ są zdefiniowane następująco:

$$\begin{aligned}d([x]) &= \bar{x} - \underline{x}, \\r([x]) &= \frac{\bar{x} - \underline{x}}{2}, \\m([x]) &= \frac{\underline{x} + \bar{x}}{2}.\end{aligned}$$

Podstawy teoretyczne arytmetyki przedziałowej

Odległość $q([x], [y])$ pomiędzy przedziałami $[x]$ i $[y]$ jest zdefiniowana wzorem

$$q([x], [y]) = \max\left\{\left|\underline{x} - \underline{y}\right|, \left|\bar{x} - \bar{y}\right|\right\}.$$

Odległość jest nieujemna, a równa 0 wtedy i tylko wtedy, gdy $[x] = [y]$. Nie zależy ona od kolejności argumentów i spełnia nierówność trójkąta. Oznacza to, że jest ona metryką, a zbiór **IR** (zbiór przedziałów rzeczywistych) z tą metryką jest **przestrzenią metryczną**. Można udowodnić, że jest to **przestrzeń metryczna zupełna**.

Podstawy teoretyczne arytmetyki przedziałowej

Elementarne operacje arytmetyczne $\circ \in \{+, -, \cdot, /\}$ na liczbach rzeczywistych rozszerza się na argumenty przedziałowe $[x]$ i $[y]$ przez zdefiniowanie wyniku takiej operacji jako zbioru liczb rzeczywistych powstałego przez wykonanie operacji $\circ$ na dowolnych dwu liczbach zawartych w przedziałach $[x]$ i $[y]$, tj.

$$[x] \circ [y] = \{x \circ y: x \in [x], y \in [y]\},$$

czyli



Podstawy teoretyczne arytmetyki przedziałowej

czyli

$$[x] + [y] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}],$$

$$[x] - [y] = [\underline{x} - \bar{y}, \bar{x} - \underline{y}],$$

$$[x] \cdot [y] = \left[\min\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\} \right],$$

$$[x] / [y] = [x] \cdot [1 / \bar{y}, 1 / \underline{y}], \quad 0 \notin [y].$$

Podstawy teoretyczne arytmetyki przedziałowej

Elementarne operacje arytmetyczne na przedziałach są *izotoniczne ze względu na zawieranie*, tzn.

$$[x] \subseteq [x'] \wedge [y] \subseteq [y'] \Rightarrow [x] \circ [y] \subseteq [x'] \circ [y'],$$
$$\circ \in \{+, -, \cdot, /\}$$

Przynależność liczby x do przedziału $[x]$, $x \in [x]$, można przedstawić zależnością

$$|x - m([x])| \leq r([x]).$$

Promień przedziału $[x]$ jest zatem ograniczeniem górnym błędu bezwzględnego punktu środkowego $m([x])$ rozważanego jako przybliżenie nieznanej liczby $x \in [x]$.

Podstawy teoretyczne arytmetyki przedziałowej

Dla przedziałów wprowadza się pojęcie **najmniejszej** i **największej wartości bezwzględnej**:

$$\langle [x] \rangle = \min\{|x| : x \in [x]\},$$

$$|[x]| = \max\{|x| : x \in [x]\} = \max\{|\underline{x}|, |\overline{x}|\}.$$

Zauważmy, że jeśli $0 \in [x]$, to $\langle [x] \rangle = 0$.

Najmniejsza i największa wartość bezwzględna przedziału są **liczbami**. Wartość bezwzględna przedziału jest jednak przedziałem i jest oznaczana $\text{abs}([x])$ (abs oznacza tu elementarną funkcję przedziałową).

Podstawy teoretyczne arytmetyki przedziałowej

Niech $\varphi: D \subset \mathbf{R} \rightarrow \mathbf{R}$ oznacza elementarną funkcję rzeczywistą ciągłą na każdym domkniętym przedziale zawartym w D . Funkcję φ rozszerzamy na argumenty przedziałowe $[x] \in D$ (definicja **funkcji przedziałowej**):

$$\varphi([x]) = \{\varphi(x): x \in [x]\}.$$

Dla monotonicznych funkcji elementarnych można podać ich odpowiedniki przedziałowe:

$$\mathbf{abs}([x]) = [\langle [x] \rangle, |[x]|],$$

$$\varphi([x]) = [\varphi(\underline{x}), \varphi(\bar{x})], \quad \varphi \in \{\mathbf{arctg}, \mathbf{arcsinh}, \mathbf{ln}, \mathbf{sinh}\},$$

$$\varphi([x]) = [\varphi(\bar{x}), \varphi(\underline{x})], \quad \varphi \in \{\mathbf{arcctg}, \mathbf{arctgh}\},$$

Podstawy teoretyczne arytmetyki przedziałowej

$$\begin{aligned}[x]^2 &= \left[\langle [x] \rangle^2, |[x]|^2 \right], \\ \sqrt{[x]} &= \left[\sqrt{\underline{x}}, \sqrt{\bar{x}} \right], \\ e^{[x]} &= \left[e^{\underline{x}}, e^{\bar{x}} \right].\end{aligned}$$

Zauważmy, że $[-1, 2]^2 = [0, 4] \neq [-1, 2] \cdot [-1, 2] = [-2, 4]$.
Jeśli $0 \in [x]$, to mamy $[x]^2 \subset [x] \cdot [x]$.

Podstawy teoretyczne arytmetyki przedziałowej

W celu otrzymania otoczenia funkcji $f([x])$ podstawiamy przedział $[x]$ w miejsce argumentu x , po czym obliczamy f stosując arytmetykę przedziałową. Taki sposób wyznaczania funkcji przedziałowej nazywamy **rozszerzeniem przedziałowym** funkcji f i oznaczamy $f_{[\cdot]}([x])$. Na podstawie definicji funkcji przedziałowej mamy

$$f([x]) \subseteq f_{[\cdot]}([x]),$$

przy czym równość zachodzi raczej rzadko.

Uwaga: Rozszerzenia przedziałowe dwóch równoważnych matematycznie zapisów funkcji zmiennej rzeczywistej mogą być różne.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Rodzaje zaokrągleń

Rodzaje zaokrągleń:

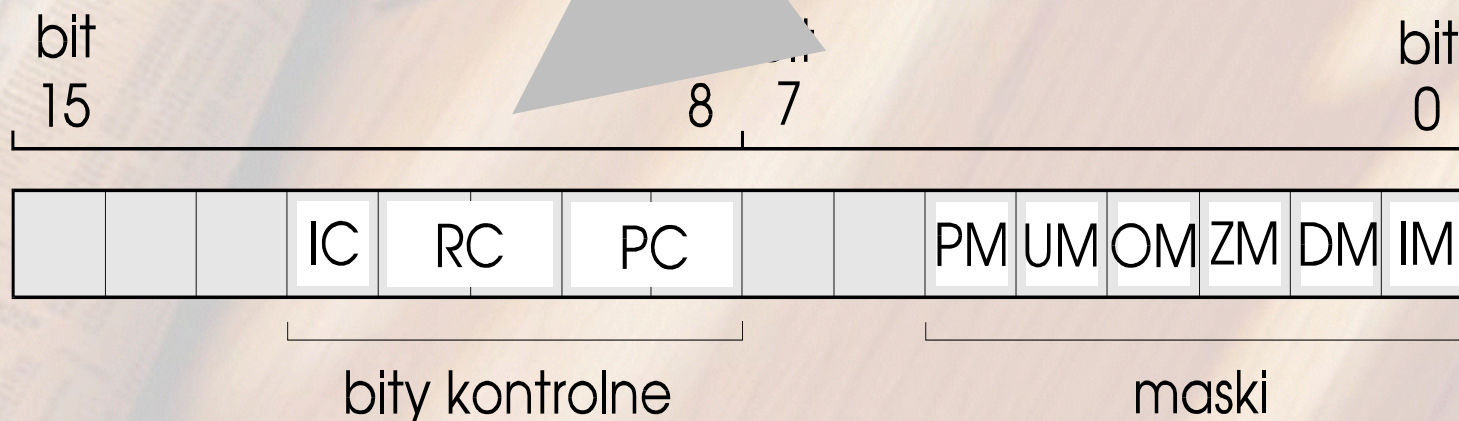
- ★ do najbliższej liczby maszynowej, a w przypadku równych odległości do liczby parzystej (jest to zaokrąglenie standardowe, przyjmowane domyślnie),
- ★ w dół, tj. w kierunku $-\infty$,
- ★ w górę, tj. w kierunku $+\infty$,
- ★ ucięcie w kierunku zera, tzn. liczby dodatnie są zaokrąglane w dół, a liczby ujemne – w górę.

Czy może być jeszcze inny sposób zaokrąglania?

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Rodzaje zaokrągleń

W jednostce arytmetyki zmiennopozycyjnej (FPU) procesora Intel aktualnie realizowany sposób zaokrąglania określa zawartość dwubitowego pola RC, które jest 10. i 11. bitem jej słowa sterującego.



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Rodzaje zaokrągleń

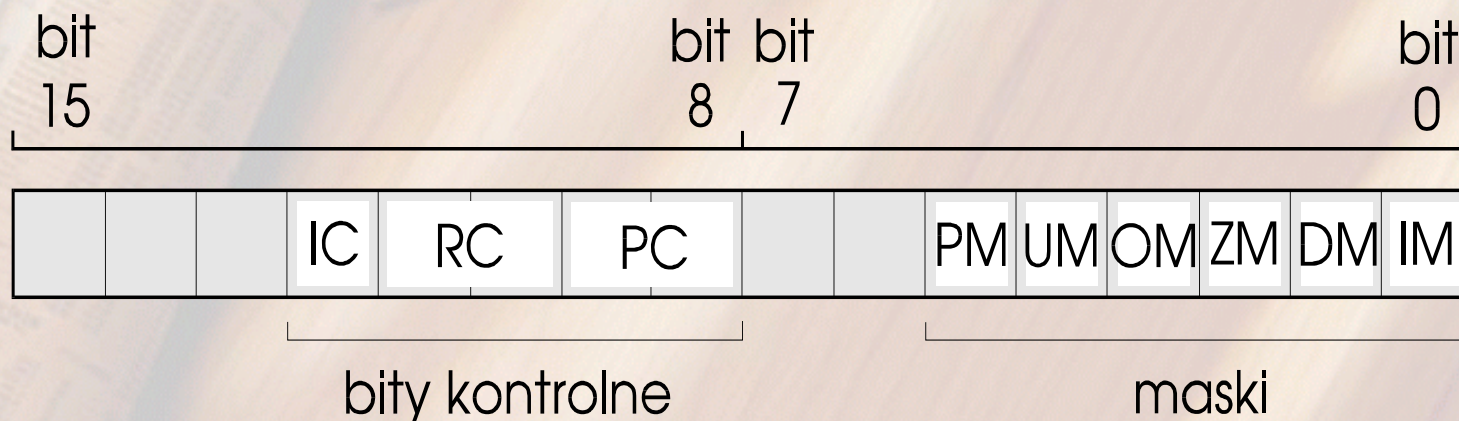
Zawartość pola **RC** – zaokrąglenie:

00 – standardowe,

01 – w dół,

10 – w górę,

11 – ucięcie w kierunku zera.



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Rodzaje zaokrągleń

IC – bit sterowania nieskończonością, który określa typ stosowanej arytmetyki nieskończoności:

0 – rzutowa (nierozróżnianie znaku nieskończoności),

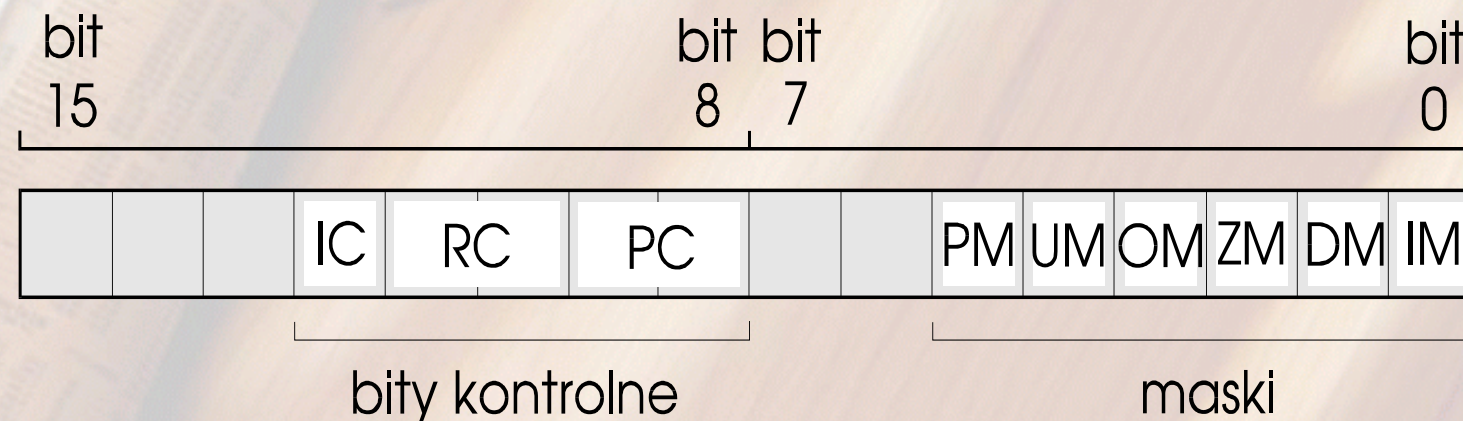
1 – afiniczna $(-\infty$ i $+\infty)$,

PC – pole sterowania precyzją, które określa jedną z trzech możliwych precyzji:

00 – krótka rzeczywista (wynik 24 bitowy),

10 – długa rzeczywista (wynik 53-bitowy),

11 – tymczasowa rzeczywista (wynik 64-bitowy).



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Rodzaje zaokrągleń

Maski:

PM – precyzji,

UM – niedomiaru,

OM – nadmiaru,

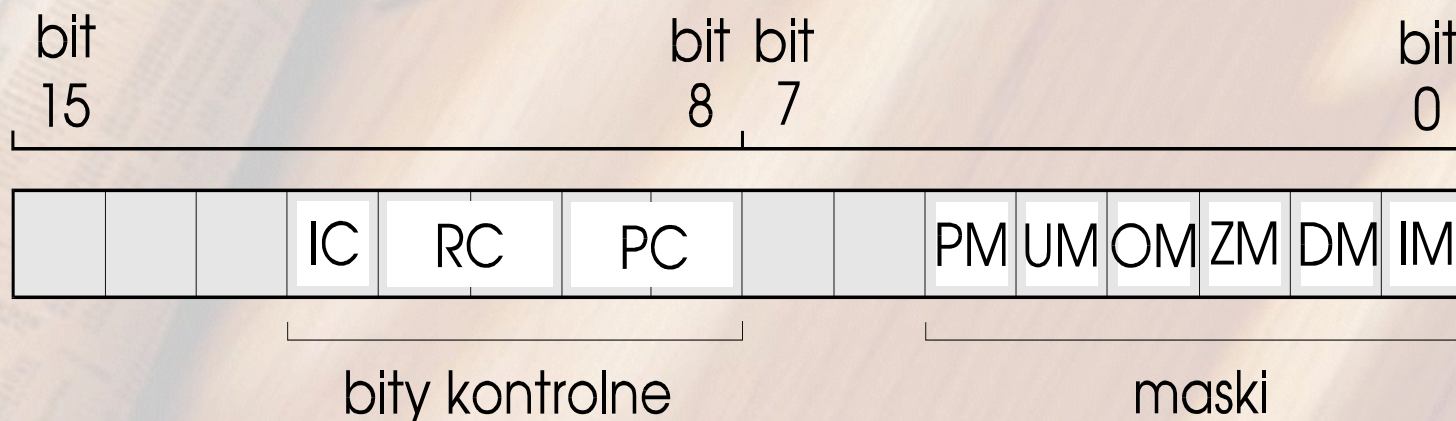
ZM – dzielenia przez zero,

DM – zdenormalizowanego operandu,

IM – niedozwolonej operacji.

Jeżeli bit jakiegś maski jest równy 1, oznacza to, że po wystąpieniu warunku określonego daną maską zostaną podjęte standardowe działania korygujące.

W przypadku, gdy bit maski jest równy 0, po wystąpieniu warunku zostanie wygenerowany sygnał przerwania.



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Niech $[x]$ i $[y]$ oznaczają przedziały maszynowe, tj. przedziały, których końce są liczbami maszynowymi.

Elementarne operacje zmiennopozycyjne na przedziałach maszynowych są określone następująco:

$$[x] + [y] = \left[\nabla(\underline{x} + \underline{y}), \Delta(\bar{x} + \bar{y}) \right],$$

$$[x] - [y] = \left[\nabla(\underline{x} - \bar{y}), \Delta(\bar{x} - \underline{y}) \right],$$

$$[x] \cdot [y] = \left[\min\left\{ \nabla(\underline{x}\underline{y}), \nabla(\underline{x}\bar{y}), \nabla(\bar{x}\underline{y}), \nabla(\bar{x}\bar{y}) \right\}, \max\left\{ \Delta(\underline{x}\underline{y}), \Delta(\underline{x}\bar{y}), \Delta(\bar{x}\underline{y}), \Delta(\bar{x}\bar{y}) \right\} \right],$$

$$[x] / [y] = [x] \cdot \left[\nabla(1/\bar{y}), \Delta(1/\underline{y}) \right], \quad 0 \notin [y],$$

gdzie ∇ oznacza zaokrąglenie w dół, a Δ – w górę.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Z podanych wzorów wynika, że wykonanie operacji na przedziałach wymaga włączania odpowiednich sposobów zaokrąglania, a więc sterowania słowem sterującym jednostki arytmetyki zmiennopozycyjnej (FPU).

Do tego celu służą:

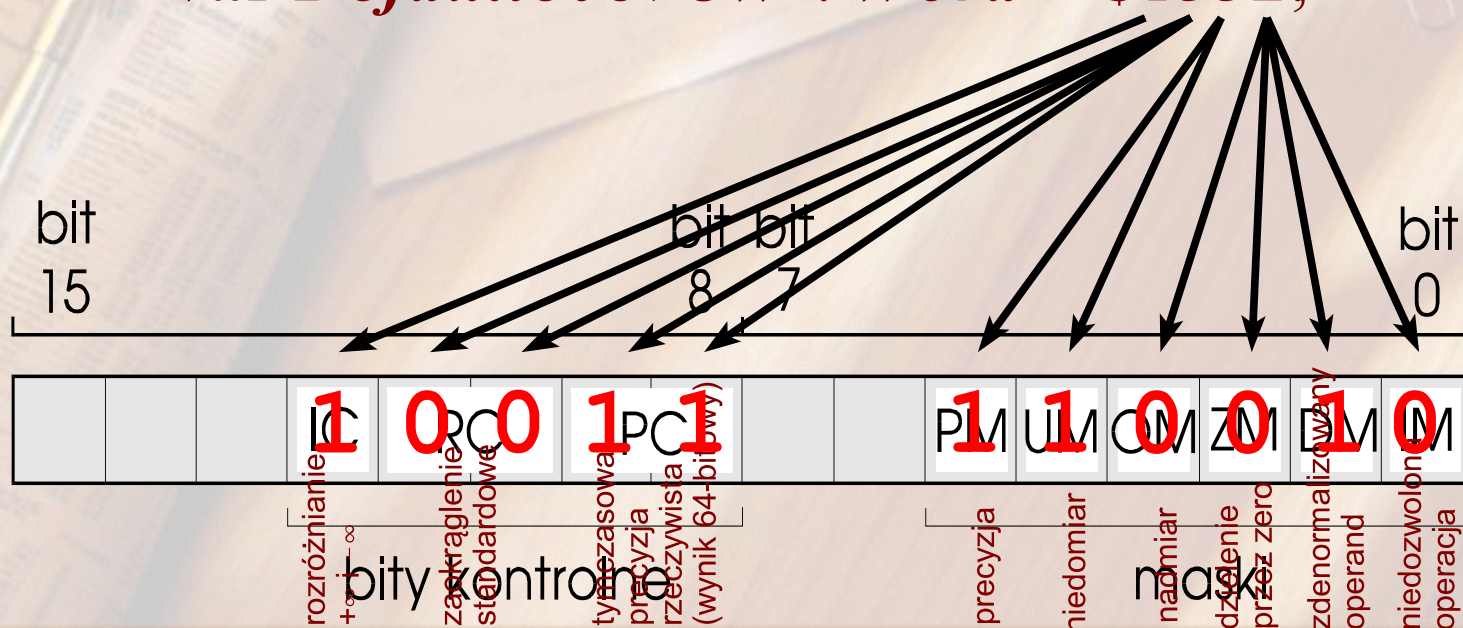
- ★instrukcje asemblera FSTCW i FLDCW,
- ★predefiniowana (w module *System*) zmienna *Default8087CW*,
- ★procedura *Set8087CW* (jest ona zdefiniowana w module *System*),
- ★funkcje *SetRoundMode* i *GetRoundMode* (zdefiniowane w module *Math*).

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Zmienna *Default8087CW* określa standardową zawartość słowa sterującego FPU. Jej deklaracja ma postać

var Default8087CW : Word = \$1332;



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Procedura *Set8087CW* służy do zmiany zawartości słowa sterującego FPU. Jej wywołanie ma postać

Set8087CW (wyrażenie)

gdzie wyrażenie (typu Word) określa jego nową zawartość.

Uwagi:

- ★ Nieodpowiednia zmiana zawartości słowa sterującego FPU (przez zmianę wartości zmiennej *Default8087CW* lub wywołanie procedury *Set8087CW*) może doprowadzić do konsekwencji trudnych do przewidzenia.
- ★ Zmiana zawartości słowa sterującego FPU za pomocą procedury *Set8087CW* powoduje zmianę wartości zmiennej *Default8087CW*.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Przykład

Jeśli zamierzamy zmienić standardową zawartość słowa sterującego FPU, to należy zdefiniować jakąś zmienną typu *Word*, np.

var stan : Word;

przypisać jej pierwotną (standardową) wartość zmiennej *Default8087CW*:

stan:=Default8087CW;

i dopiero po wykonaniu tej operacji wywołać procedurę *Set8087CW*, np.

Set8087CW (\$133F);

(wywołanie to wyłącza wszystkie maski FPU). Po wykonaniu określonych operacji przy niestandardowej zawartości słowa sterującego FPU, należy ustalić jego standardową zawartość za pomocą instrukcji

Set8087CW (stan);

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Jeśli podejrzewamy, że bieżąca zawartość słowa sterującego FPU nie jest standardowa, a więc jest różna od pierwotnej wartości zmiennej *Default8087CW*, to do jego odczytania można posłużyć się instrukcją asemblera *FSTCW*.

Odpowiednia procedura w języku Delphi Pascal może mieć postać:

```
procedure ustalenie_stanu_pierwotnego (var stan : Word);  
var odczyt : Word;  
begin  
    asm  
        FSTCW odczyt  
    end;  
    stan:=odczyt  
end {ustalenie_stanu_pierwotnego};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Jeśli okaże się, że w wyniku wywołania tej procedury:

ustalenie_stanu_pierwotnego (stan);

gdzie stan oznacza zmienną typu *Word*, wartość zmiennej podanej w wywołaniu jest różna od \$1332, to w celu przywrócenia standardowej wartości słowa sterującego koprocatora należy zastosować instrukcję

Set8087CW (\$1332);

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

W celu włączenia zaokrąglenia w dół wywołanie procedury *Set8087CW* powinno mieć postać

***Set8087CW* ((stan or \$0400) and not \$0800);**

a w celu włączenia zaokrąglenia w górę – następującą postać:

***Set8087CW* ((stan and not \$0400) or \$0800);**

Zauważmy, że w powyższych wywołaniach zmienna *stan* wcale nie musi być równa standardowej zawartości słowa sterującego FPU (wywołania te powodują wyłącznie zmianę zawartości pola RC, które steruje zaokrągleniami).

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Do włączenia zaokrągleń w dół i w górę można też użyć [instrukcji asemblera FLDCW](#), a odpowiednie procedury paskalowe mogą mieć postacie:

```
procedure zaokraglenia_w_dol (const stan : Word);  
var nowy_stan : Word;  
begin  
    nowy_stan:=stan or $0400;                // włączenie bitu 10  
    nowy_stan:=nowy_stan and not $0800; // wyłączenie bitu 11  
    asm  
        FLDCW nowy_stan                      // ustalenie nowego stanu  
    end                                       // słowa sterującego FPU  
end {zaokraglenia_w_dol};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Do włączenia zaokrągleń w dół i w górę można też użyć [instrukcji asemblera FLDCW](#), a odpowiednie procedury paskalowe mogą mieć postacie:

```
procedure zaokraglenia_w_gore (const stan : Word);  
var nowy_stan : Word;  
begin  
    nowy_stan:=stan and not $0400;           // wyłączenie bitu 10  
    nowy_stan:=nowy_stan or $0800;          // włączenie bitu 11  
    asm  
        FLDCW nowy_stan                     // ustalenie nowego stanu  
    end                                     // słowa sterującego FPU  
end {zaokraglenia_w_gore};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Od wersji 7 języka Delphi Pascal w module *Math* (w module *System.Math* w pakiecie Delphi XE) są dostępne standardowe funkcje służące do ustalenia i określenia aktualnego rodzaju zaokrąglenia.

GetRoundMode – uzyskanie informacji o bieżącym rodzaju zaokrąglania realizowanym przez FPU; wartość funkcji jest typu *TFPURoundingMode*:

type TFPURoundingMode = (rmNearest, rmDown, rmUp, rmTruncate);

- rmNearest*** – zaokrąglenie standardowe, tj. do najbliższej liczby, a w przypadku równych odległości do liczby parzystej
- rmDown*** – zaokrąglenie w dół
- rmUp*** – zaokrąglenie w górę
- rmTruncate*** – obcięcie w kierunku zera

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

SetRoundMode (*zaokrąglenie*) – określenie rodzaju zaokrąglenia realizowanego przez FPU (wartością funkcji jest poprzedni rodzaj zaokrąglenia); parametr *zaokrąglenie* i wartość funkcji są typu ***TFPURoundingMode***

W realizacji zmiennopozycyjnej arytmetyki przedziałowej stosujemy następujące wywołania:

- ★ ***SetRoundMode***(*rmDown*) – włączenie zaokrąglenia w dół,
- ★ ***SetRoundMode***(*rmUp*) – włączenie zaokrąglenia w górę,
- ★ ***SetRoundMode***(*rmNearest*) – powrót do standardowego zaokrąglania.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Realizację czterech podstawowych działań arytmetycznych, tj. dodawania, odejmowania, mnożenia i dzielenia, w arytmetyce przedziałowej zapewniają w języku Delphi Pascal następujące funkcje:

```
function iadd (const x, y : interval) : interval;  
begin  
    SetRoundMode (rmDown);  
    Result.a:=x.a+y.a;  
    SetRoundMode (rmUp);  
    Result.b:=x.b+y.b;  
    SetRoundMode (rmNearest)  
end {iadd};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
function isub (const x, y : interval) : interval;  
begin  
    SetRoundMode (rmDown);  
    Result.a:=x.a-y.b;  
    SetRoundMode (rmUp);  
    Result.b:=x.b-y.a;  
    SetRoundMode (rmNearest)  
end {isub};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
function imul (const x, y : interval) : interval;  
var x1y1, x1y2, x2y1 : Extended;  
begin  
    SetRoundMode (rmDown);  
    x1y1:=x.a*y.a;  x1y2:=x.a*y.b;  x2y1:=x.b*y.a;  
    with Result do  
        begin  
            a:=x.b*y.b;  
            if x2y1<a then a:=x2y1;  
            if x1y2<a then a:=x1y2;  
            if x1y1<a then a:=x1y1  
        end;
```



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
SetRoundMode (rmUp);  
x1y1:=x.a*y.a;  x1y2:=x.a*y.b;  x2y1:=x.b*y.a;  
with Result do  
  begin  
    b:=x.b*y.b;  
    if x2y1>b then b:=x2y1;  
    if x1y2>b then b:=x1y2;  
    if x1y1>b then b:=x1y1  
  end;  
  SetRoundMode (rmNearest);  
end {imul};
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
function idiv (const x, y : interval) : interval;  
var x1y1, x1y2, x2y1 : Extended;  
begin  
  if (y.a<=0) and (y.b>=0)  
    then raise EZeroDivide.Create ('Dzielenie przez '  
      +'przedział zawierający 0.')  
  else begin  
    SetRoundMode (rmDown);  
    x1y1:=x.a/y.a; x1y2:=x.a/y.b; x2y1:=x.b/y.a;  
    with Result do  
      begin  
        a:=x.b/y.b;
```



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

if $x_2y_1 < a$ **then** $a := x_2y_1$;

if $x_1y_2 < a$ **then** $a := x_1y_2$;

if $x_1y_1 < a$ **then** $a := x_1y_1$

end;

SetRoundMode (*rmUp*);

$x_1y_1 := x.a/y.a$; $x_1y_2 := x.a/y.b$; $x_2y_1 := x.b/y.a$;

with *Result* **do**

begin

$b := x.b/y.b$;

if $x_2y_1 > b$ **then** $b := x_2y_1$;

if $x_1y_2 > b$ **then** $b := x_1y_2$;



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
    if  $x_1 y_1 > b$  then  $b := x_1 y_1$   
  end  
end;  
SetRoundMode (rmNearest)  
end {idiv};
```

Typ *interval* należy zdefiniować następująco:

```
type interval = record  
    a, b : Extended  
end;
```


Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

Typ *interval* można zdefiniować wraz z deklaracjami operatorów przeciążonych, co ułatwia programowanie obliczeń na przedziałach:

type interval = record

a, b : *Extended*;

class operator *Implicit* (x : *Extended*) : interval;

class operator *Add* (x, y : interval) : interval;

class operator *Subtract* (x, y : interval) : interval;

class operator *Multiply* (x, y : interval) : interval;

class operator *Divide* (x, y : interval) : interval;

end;

przy czym definicje operatorów przeciążonych są następujące:



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
class operator interval.Add (x, y : interval) : interval;  
begin  
    Result:=iadd(x,y)  
end {Add};
```

```
class operator interval.Subtract (x, y : interval) : interval;  
begin  
    Result:=isub(x,y)  
end {Subtract};
```

```
class operator interval.Multiply (x, y : interval) : interval;  
begin  
    Result:=imul(x,y)  
end {Multiply};
```



Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Operacje zmiennopozycyjne na przedziałach

```
class operator interval.Divide (x, y : interval) : interval;  
begin  
    Result:=idiv(x,y)  
end {Divide};
```

```
class operator interval.Implicit (x : Extended) : interval;  
var s : string;  
begin  
    Str (x:26, s);  
    Result.a:=left_read(s);  
    Result.b:=right_read(s)  
end {Implicit};
```

Operator *Implicit* jest operatorem niejawnej zmiany typu. Funkcje *left_read* i *right_read* wyznaczają odpowiednio największą liczbę maszynową mniejszą lub równą liczbie reprezentowanej przez łańcuch *s* oraz najmniejszą liczbę maszynową większą lub równą tej liczbie.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Jeśli liczba rzeczywista nie jest liczbą maszynową, to może być reprezentowana w postaci przedziału o końcach będących dwiema sąsiednimi liczbami maszynowymi.

Proponowany sposób otrzymywania takiego przedziału opiera się na wykorzystaniu informacji o reprezentacji maszynowej danej (wprowadzonej liczby).

Ograniczamy się do liczb typu *Extended* (typu rzeczywistego języka Delphi Pascal o największej precyzji i największym zakresie).

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Oznaczmy:

- a – liczba wprowadzona do programu,
- x – zmienna, w której jest zapamiętana reprezentacja maszynowa (typu Extended) liczby a ,
- sa – zmienna typu łańcucha długiego (*AnsiString*), w której jest przechowywany ciąg wprowadzonych znaków liczby a ,
- sx – zmienna typu łańcucha długiego, w której zapamiętujemy stałoprzecinkową, dziesiętną wartość reprezentacji maszynowej liczby a , czyli wartość zmiennej x odczytaną z jej wewnętrznej postaci.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Jeśli liczbę wprowadzono w zapisie zmiennopozycyjnym, to w łańcuchu *sa* dokonujemy konwersji do zapisu stałopozycyjnego. Z uwagi na zakres liczb typu *Extended* łańcuch ten będzie zawierał mniej niż 5000 znaków, co wobec dozwolonych rozmiarów łańcuchów długich, ograniczonych praktycznie tylko dostępną pamięcią, nie jest dużo.

Przypadek 1: $sa = sx$

Liczba *a* jest reprezentowana dokładnie w komputerze, a więc przedział jej maszynowej reprezentacji ma szerokość 0 (przedziałem tym jest $[x, x]$).

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Przypadek 2: $sa \neq sx$

Analiza wewnętrznej reprezentacji liczb typu *Extended* prowadzi do wniosku, że przy równych cechach mantysy dwóch sąsiednich liczb maszynowych w tym typie różnią się o 2^{-63} .

Zatem dodając jedynkę dwójkową na ostatnim bicie wewnętrznej reprezentacji mantysy zmiennej x oraz wykonując wynikające stąd ewentualne modyfikacje cechy otrzymamy następną liczbę maszynową w stosunku do wartości zmiennej x .

Z kolei, gdy odejmiemy jedynkę dwójkową na ostatnim bicie, to otrzymamy poprzednią liczbę maszynową.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Zamiast tych operacji możemy do zmiennej x dodać i odjąć zmienną eps , której wartość wyznacza wewnętrzna (znormalizowana) postać mantysy równa 1 i cecha mniejsza o 63 od wartości cechy w wewnętrznej reprezentacji wartości zmiennej x .

Wyznaczenie takiej wartości zmiennej eps jest możliwe dla każdej zmiennej x typu **Extended**, której bezwzględna wartość jest o 2^{63} (około $5 \cdot 10^{20}$) większa od minimalnej dodatniej liczby w tym typie. Wobec dolnego ograniczenia zakresu liczb typu **Extended** do około $3,6 \cdot 10^{-4951}$ wystarczy przyjąć za dolną granicę zakresu rozważanych wielkości wartość 10^{-4930} .

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Otrzymane wartości przypisujemy zmiennym:

nx – w przypadku liczby maszynowej następnej,

px – w przypadku liczby maszynowej poprzedniej,

srx – łańcuch zawierający odpowiednik dziesiętny
(w postaci stałopozycyjnej) wartości zmiennej nx ,

spx – łańcuch zawierający odpowiednik dziesiętny
(w postaci stałopozycyjnej) wartości zmiennej px .

Po wykonaniu tych operacji dysponujemy czterema łańcuchami długimi: sa , sx , spx oraz srx .

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Jeśli wprowadziliśmy liczbę dodatnią, to wystarczy teraz sprawdzić, która z następujących par relacji łańcuchowych jest prawdziwa:

$$spx < sa < sx \quad \text{czy też} \quad sx < sa < snx.$$

Jeśli prawdziwa jest pierwsza para nierówności, to przedziałem zmiennopozycyjnym reprezentującym liczbę rzeczywistą (niemaszynową) a jest przedział $[px, x]$, a w drugim przypadku – przedział $[x, nx]$.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Zauważmy, że dla liczby dodatniej nierówności

$$spx < sa \quad \text{ i } \quad sa < snx$$

są zawsze spełnione (jeśli wprowadzona liczba jest reprezentowana dokładnie w komputerze, to $sa = sx$, a ponadto $spx < sx < snx$). Dlatego w praktyce wystarczy wyznaczyć tylko łańuchy sa i sx oraz ograniczyć się do sprawdzenia poprawności jednej z relacji

$$sa < sx \quad \text{ lub } \quad sx < sa.$$

W przypadku liczby ujemnej, gdy prawdziwa jest pierwsza nierówność, to poszukiwanym przedziałem jest $[x, nx]$, a gdy druga – przedział $[px, x]$.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przedziałowe reprezentacje liczb

Realizacja przedstawionego algorytmu w programie wcale nie jest łatwa. Główna trudność polega na konstrukcji łańcuchów długich z dziesiętną, stałopozycyjną postacią wewnętrznych reprezentacji liczb.

Uwaga: Jeśli dana jest przedziałem rzeczywistym (w sensie matematycznym), to za lewy koniec przedziału maszynowego przyjmujemy największą liczbę maszynową $\leq$ lewemu końcowi przedziału rzeczywistego, a za prawy koniec – najmniejszą liczbę maszynową $\geq$ prawemu końcowi tego przedziału.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład

Rozważmy układ równań liniowych Boothroyda-Dekkera $\mathbf{Ax} = \mathbf{b}$, gdzie $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, $\mathbf{A}$ oznacza macierz kwadratową stopnia n , której współczynniki są określone wzorem

$$a_{ij} = \binom{n+i-1}{i-1} \cdot \binom{n-1}{n-j} \cdot \frac{n}{i+j-1}; \quad i, j = 1, 2, \dots, n;$$

a wielkości $b_i = i$ ($i = 1, 2, \dots, n$). Rozwiązanie dokładne tego układu ma postać

$$x_i = (-1)^i (i-1), \quad i = 1, 2, \dots, n.$$

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

Do rozwiązywania tego układu zastosowano metodę eliminacji Gaussa z pełnym wyborem elementu podstawowego.

Przy zaokrągleniach standardowym, w górę i w dół otrzymano następujące wyniki:

Rozwiązanie "normalne"

x[1]	=	-0.000000000004743192
x[2]	=	1.000000000045157710
x[3]	=	-2.000000000238984643
x[4]	=	3.000000000926638707
x[5]	=	-4.000000002935040749
x[6]	=	5.000000008041954110
x[7]	=	-6.000000019736256904
x[8]	=	7.000000044395588679
x[9]	=	-8.000000093031922800
x[10]	=	9.000000183787835168

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

Rozwiązanie przy zaokrągleniach		Szerokości
w górę	w dół	"przedziałów"
x[1] = 0.00000000004669545	-0.00000000001606113	0.00000000006275657
x[2] = 0.99999999952417554	1.00000000017043271	0.00000000064625717
x[3] = -1.99999999734731506	-2.00000000097625318	0.00000000362893812
x[4] = 2.99999998928228114	3.00000000402198646	0.00000001473970533
x[5] = -3.99999996489990830	-4.00000001336713452	0.00000004846722622
x[6] = 4.99999990113041707	5.00000003808894987	0.00000013695853281
x[7] = -5.99999975164410683	-6.00000009656985761	0.00000034492575078
x[8] = 6.99999943012533486	7.00000022328708134	0.00000079316174648
x[9] = -7.99999878513889253	-8.00000047905955295	0.00000169392066042
x[10] = 8.99999756377489755	9.00000096592257091	0.00000340214767336
Środek		Różnica w stosunku
"przedziału"		do rozw. "normalnego"
x[1] = 0.00000000001531716		0.00000000006274908
x[2] = 0.99999999984730412		-0.00000000060427298
x[3] = -1.99999999916178412		0.00000000322806232
x[4] = 2.99999999665213380		-0.00000001261425327
x[5] = -3.99999998913352141		0.00000004021688608
x[6] = 4.99999996960968347		-0.00000011080985763
x[7] = -5.99999992410698222		0.00000027325558682
x[8] = 6.99999982670620810		-0.00000061724967869
x[9] = -7.99999963209922274		0.00000129822000526
x[10] = 8.99999926484873423		-0.00000257302961745

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

*Analiza przedstawionych wyników prowadzi do wniosku, że przy każdym rodzaju zaokrąglania FPU **otrzymane rozwiązania są obarczone dość dużym błędem**, przy czym rozwiązanie uzyskane przy standardowym zaokrągleniu wcale nie musi być dokładniejsze (porównaj wartości $x[1]$).*

*Ponadto **rozwiązanie „normalne” niekoniecznie musi leżeć wewnątrz przedziału wyznaczonego przez dwa pozostałe rozwiązania** (zobacz, na przykład, wartości $x[6]$). Przedziału tego nie należy też uważać za rozwiązanie przedziałowe w sensie arytmetyki przedziałowej.*

*Przy okazji zwróćmy także uwagę na **zaokrąglenia wykonywane przez procedurę Writeln** – wyświetlone różnice wartości nie są równe różnicy wyświetlonych wartości.*

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

Zastosowanie zmiennopozycyjnej arytmetyki przedziałowej:

	Rozwiązanie przedziałowe	Szerokości przedziałów
x[1]=	[-0.000000000000000000, -0.000000000000000000]	0.0000000000000000E+0000
x[2]=	[1.000000000000000000, 1.000000000000000000]	0.0000000000000000E+0000
x[3]=	[-2.000000000000000000, -2.000000000000000000]	0.0000000000000000E+0000
x[4]=	[3.000000000000000000, 3.000000000000000000]	0.0000000000000000E+0000
x[5]=	[-4.000000000000000000, -4.000000000000000000]	0.0000000000000000E+0000
x[6]=	[5.000000000000000000, 5.000000000000000000]	0.0000000000000000E+0000
x[7]=	[-6.000000000000000000, -6.000000000000000000]	0.0000000000000000E+0000
x[8]=	[7.000000000000000000, 7.000000000000000000]	0.0000000000000000E+0000
x[9]=	[-8.000000000000000000, -8.000000000000000000]	0.0000000000000000E+0000
x[10]=	[9.000000000000000000, 9.000000000000000000]	0.0000000000000000E+0000

Przez zastosowanie arytmetyki przedziałowej w języku Delphi Pascal otrzymaliśmy zatem rozwiązanie dokładne!

Oczywiście w ogólności, tj. dla dowolnego układu równań liniowych, tak nie musi być.

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

Dla układu równań postaci

$$\begin{aligned}1,0x_1 + 4,5x_2 + 1,2x_3 + 2,1x_4 + 2,52x_5 &= 2,1; \\5,5x_1 + 3,3x_2 + 9,9x_3 + 1,848x_4 + 2,31x_5 &= 1,98; \\2,2x_1 + 1,485x_2 + 4,752x_3 + 9,24x_4 + 1,188x_5 &= 1,0395; \\7,15x_1 + 5,148x_2 + 1,716x_3 + 3,432x_4 + 4,5045x_5 &= 4,004; \\2,002x_1 + 1,5015x_2 + 5,148x_3 + 1,05105x_4 + 1,4014x_5 &= 1,26126;\end{aligned}$$

otrzymuje się następujące rozwiązanie przedziałowe:

	Rozwiązanie przedziałowe	Szerokości przedziałów
x[1]	[-0.00771054612831261, -0.00771054612831209]	5.16238205747763E-0016
x[2]	[-0.10075588680675287, -0.10075588680675171]	1.16191236319840E-0015
x[3]	[0.00081505200045489, 0.00081505200045508]	1.94743038969131E-0016
x[4]	[-0.00057063465615305, -0.00057063465615292]	1.37354068633369E-0016
x[5]	[1.01640170918098016, 1.01640170918098248]	2.32008422890173E-0015

Realizacja zmiennopozycyjnej arytmetyki przedziałowej

Przykład (cd.)

Na podstawie otrzymanego rozwiązania przedziałowego można jednak określić rozwiązanie danego układu z dokładnością do 14. miejsca po przecinku:

$$\begin{aligned}x_1 &= -0,00771054612831, & x_2 &= -0,10075588680675, \\x_3 &= 0,00081505200045, & x_4 &= -0,00057063465615, \\x_5 &= 1,01640170918098.\end{aligned}$$

Czy moglibyśmy podać rozwiązanie podanego układu z taką samą dokładnością przy zastosowaniu zwykłej arytmetyki zmiennopozycyjnej?



**Koniec
wykładu!**