

Politechnika Wrocławska
Wydział Elektroniki
Dokumentacja końcowa

ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW: PROJEKT

Autor:

ALEKSANDER NAPOROWSKI 226229
TN 13:15 CZWARTEK

PROWADZĄCY:
dr inż. Tadeusz Tomczak

TEMAT: IMPLEMENTACJA PROCEDUR OBLICZEŃ NA LICZBACH
ZMIENNOPRZECINKOWYCH ZA POMOCĄ INSTRUKCJI STAŁOPRZECINKOWYCH

13 czerwca 2019

Spis treści

1	Cel projektu	2
2	Liczby zmiennoprzecinkowe	3
2.1	Co to jest?	3
2.2	Działania	4
2.2.1	Dodawanie i odejmowanie	4
2.2.2	Mnożenie	4
2.2.3	Dzielenie	5
2.2.4	Pierwiastek	5
3	Założenia projektu	6
3.1	Zmienna 80-bitowa	6
4	Środowisko i uruchomienie	7
5	Operacje bitowe	8
6	Testy	9
6.1	Testy jednostkowe	9
7	Podsumowanie	10

Rozdział 1

Cel projektu

Celem projektu była implementacja procedur obliczeń w języku C++ dla liczb zmiennoprzecinkowych, która za pomocą instrukcji stałoprzecinkowych pozwoli na wykonanie tych procedur.

Rozdział 2

Liczby zmiennoprzecinkowe

2.1 Co to jest?

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 IEEE754 (dla liczb zapisanych w kodzie dwójkowym). Liczbę zapisuje się jako ciąg zer i jedynek przyjmując umowny podział na “pola”:

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

Rysunek 2.1: Wzór liczby zmienoprzecinkowej

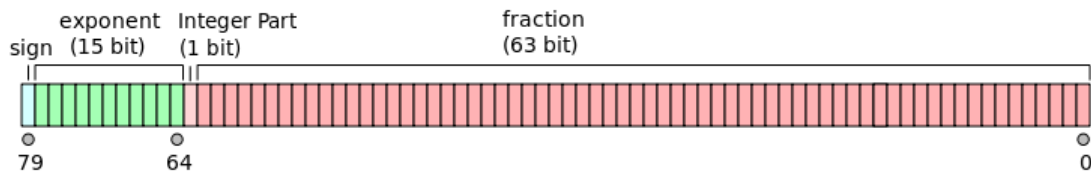
Na rysunku 2.1 przedstawiony jest wzór liczby zmiennoprzecinkowej, gdzie ‘S’ to znak, ‘M’ mantysa, a ‘E’ wykładnik. Liczby te zapisane są w postaci dziesiętnej, wzór pozwala obliczyć ich wartość.

W języku C++ jest kilka typów danych zmiennoprzecinkowych. Różnią się one precyzją, czyli dokładnością zapisu liczby. Jest to związane z ilością bitów, które w kodzie liczby zmiennoprzecinkowej przeznaczone są na zapis mantysy. Mantysa może być reprezentowana dokładniej im więcej posiada ona bitów. Liczby mogą mieć większy zakres gdy exponenta zapisana jest na większej ilości bitów.

Typy w C++:

- **float** 32bitowa,
- **double** 64 bitowa,
- **long double** 96 lub 128 bitów.

Do realizacji projektu była potrzebna zmienna, której wielkość wynosi 80 bitów. Na rysunku 2.2 przedstawiony jest podział bitów na znak posiadający jeden bit, cechę o długości 15 bitów i mantysę o długości 64 bitów.



Rysunek 2.2: Podział liczby 80 bitowej

2.2 Działania

Na liczbach zmiennoprzecinkowych wykonywane są działania arytmetyczne. Liczbę należy wyobrazić sobie w postaci wykładniczej, jak na rysunku 2.3.

$$\mathbf{x} = \mathbf{M} * \mathbf{B}^{\mathbf{E}}$$

Rysunek 2.3: Liczba zmiennoprzecinkowa w postaci wykładniczej

2.2.1 Dodawanie i odejmowanie

Przy pomocy wyłączenia części wspólnej obu liczb, dostajemy wzór dodawania (jak i odejmowania) przedstawiony na rysunku 2.4.

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{M}_1 * \mathbf{B}^{\mathbf{E}_1} \\ \mathbf{x}_2 &= \mathbf{M}_2 * \mathbf{B}^{\mathbf{E}_2}\end{aligned}$$

Rysunek 2.4: Liczby zmiennoprzecinkowe w notacji naukowej

$$\mathbf{x}_1 \pm \mathbf{x}_2 = \mathbf{M}_1 * \mathbf{B}^{\mathbf{E}_1} \pm \mathbf{M}_2 * \mathbf{B}^{\mathbf{E}_2} = (\mathbf{M}_1 * \mathbf{B}^{\mathbf{E}_1 - \mathbf{E}_2} \pm \mathbf{M}_2) * \mathbf{B}^{\mathbf{E}_2}$$

2.2.2 Mnożenie

W przeciwieństwie do dodawania/odejmowania, mnożenie liczb w postaci zmiennoprzecinkowej jest dużo prostsze rysunek 2.6.

$$\mathbf{\hat{x}}_1 * \mathbf{\hat{x}}_2 = (\mathbf{M}_1 * \mathbf{B}^{\mathbf{E}_1}) * (\mathbf{M}_2 * \mathbf{B}^{\mathbf{E}_2}) = (\mathbf{M}_1 * \mathbf{M}_2) * (\mathbf{B}^{\mathbf{E}_1} * \mathbf{B}^{\mathbf{E}_2}) = (\mathbf{M}_1 * \mathbf{M}_2) * \mathbf{B}^{\mathbf{E}_1 + \mathbf{E}_2}$$

2.2.3 Dzielenie

W dzieleniu przekształcenie jest bardzo podobne do tego przy mnożeniu.

$$\mathbf{x}_1/\mathbf{x}_2 = (\hat{\mathbf{M}}_1 * \mathbf{B}^{E1}) / (\hat{\mathbf{M}}_2 * \mathbf{B}^{E2}) = (\mathbf{M}_1/\mathbf{M}_2) * (\mathbf{B}^{E1} / \mathbf{B}^{E2}) = (\mathbf{M}_1 / \mathbf{M}_2) * \mathbf{B}^{E1-E2}$$

Rysunek 2.5: Wzór dzielenia

2.2.4 Pierwiastek

Ostatnim działaniem w projekcie, które miało zostać zaimplementowane jest pierwiastek. Liczby zmiennoprzecinkowe mają za zadanie przechowywać liczby rzeczywiste, więc pierwiastkowanie liczb ujemnych jest niemożliwe, tak więc znak zawsze musi wynosić '0', w innym przypadku będzie wynik NaN. Wzór jest na rysunku.

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

Rysunek 2.6: Wzór pierwiastka liczby zmiennoprzecinkowej

Rozdział 3

Założenia projektu

Podczas realizacji projektu z prowadzącym zostały ustalone następujące założenia:

- Implementacja w C++ (nowsze standardy min cpp11)
- Cztery podstawowe działania: dodawanie, odejmowanie, mnożenie, dzielenie oraz jedno dodatkowe: pierwiastek
- Program musi działać pod systemem operacyjnym Linux
- Zmienna musi mieć wielkość 80-bitów
- Użycie instrukcji stałoprzecinkowych (przesunięcia bitowe)

3.1 Zmienna 80-bitowa

Jednym z większych problemów jakie napotkałem podczas realizacji projektu było znalezienie zmiennej o odpowiedniej wielkości. Standardowe zmienne zaimplementowane w C++ były za małe albo za duże. W celu rozwiązania problemu zaimplementowałem własną zmienną 'float80', która jest unią. Posiada ona strukturę 'parts' z trzema polami 'mantissa' typu 'uint64_t', który jest 64bitowy, 'exponent' typu 'uint16_t' oraz 'sign' typu 'uint16_t'. Oprócz struktury unia posiada jeszcze zmienną long double 'number'. Kod stworzonej unii znajduje się na rysunku 3.1.

```
typedef union
{
    long double number;
    struct
    {
        uint64_t mantissa : 64;
        uint16_t exponent : 15;
        uint16_t sign : 1;
    }
    parts;
}
float80;
```

Rysunek 3.1: Kod unii float80

Rozdział 4

Środowisko i uruchomienie

Projekt został stworzony z myślą o systemach operacyjnych opartych o jądro Linux'a, było to jedno z założeń projektu. Działanie projektu było testowane też na komputerach używających systemu operacyjnego Windows, jednak kod nie działał prawidłowo, ponieważ kompilatory wbudowane w system operacyjny firmy Microsoft obcinają zakres zmiennej long double do wielkości zmiennej double. Systemem operacyjnym pod którym tworzony był projekt to Ubuntu w wersji 16.04 LTS 32 bity. Do kompilacji użyto GNU Compiler Collection, domyślnego kompilatora dla systemu Linux. IDE używany to CodeBlocks wersja 13.04. W ustawieniach kompilatora została dodana flaga `'-std=gnu++11'`.

Rozdział 5

Operacje bitowe

Operacja bitowa operuje na jednym lub więcej ciągu bitów lub liczbie w zapisie dwójkowym na poziomie poszczególnych bitów. Takie operacje na starszych urządzeniach pozawalały na szybsze dodawanie, odejmowanie, mnożenie i dzielenie. Na obecnych urządzeniach operacje bitowe są tak samo szybkie jak dodawanie arytmetyczne i o wiele szybsze niż mnożenia arytmetyczne.

Operatory bitowe:

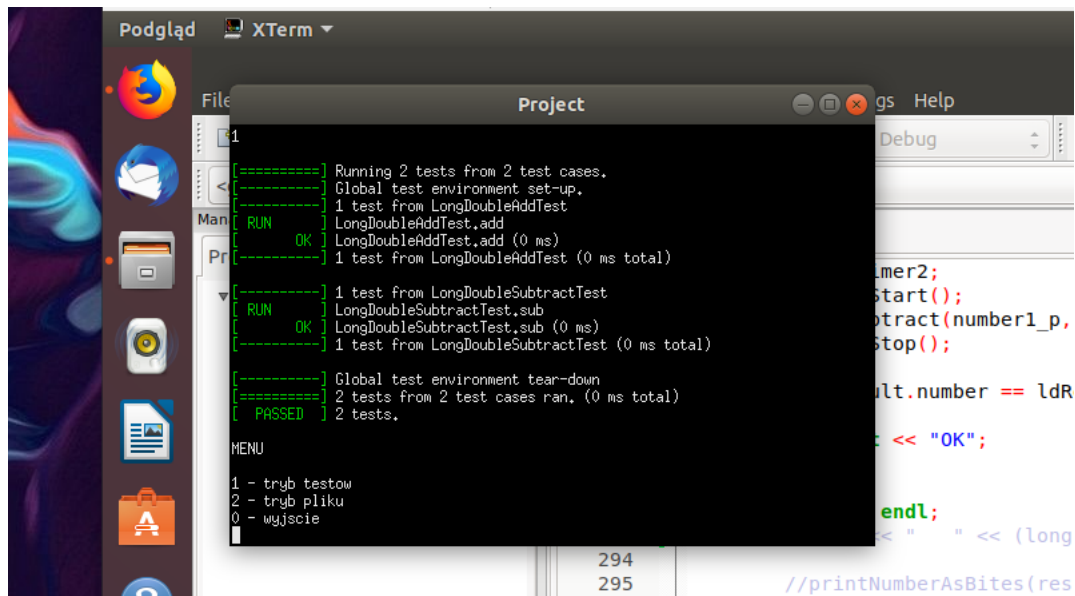
- `|` Operacja bitowa lub
- `&` Operacja bitowa i
- `^` Operacja bitowa albo
- `«` Przesunięcie bitowe w lewo
- `»` Przesunięcie bitowe w prawo
- `~` Zaprzeczenie bitowe, dopełnienie bitowe

Rozdział 6

Testy

6.1 Testy jednostkowe

W celu sprawdzenia, czy zaimplementowane procedury na liczbach zmiennoprzecinkowych działają w sposób poprawny, należało napisać testy jednostkowe. W tym celu zostało użyte open-source'owe narzędzie firmy Google - Google Test.

A screenshot of a Linux desktop environment. In the foreground, a terminal window titled 'Project' is open, displaying the output of a Google Test run. The output shows two test cases: 'LongDoubleAddTest' and 'LongDoubleSubtractTest', each with one test passing successfully. The total result is '2 tests from 2 test cases ran. (0 ms total)' and 'PASSED 2 tests.'. Below the output is a menu with options: '1 - tryb testow', '2 - tryb pliku', and '0 - wyjscie'. In the background, a file manager window is partially visible, showing a directory structure. The desktop background is a dark, abstract image.

Rysunek 6.1: Wynik testów Google Test

Rozdział 7

Podsumowanie

Temat projektu, mimo że podczas wyboru wydawał się być łatwy okazał się bardzo trudny. Poszukiwanie zmiennej, która posiada określoną ilość bitów spowodowała, że dość długo męczyłem się jak spełnić ten warunek. Kolejną przeszkodą w realizacji projektu było zrozumienie, o co chodzi z instrukcjami stałoprzecinkowymi. Początkowo myślałem, że trzeba użyć instrukcji SIMD, co okazało się błędem. Podczas zaawansowanych prac nad projektem znalazłem informacje o operacjach bitowych, które to powinny zostać użyte w projekcie od początku.

Wyniki testów czasowych, które objęły porównanie obliczeń arytmetycznych i bitowych, okazały się niemiarodajne ze względu na zbyt dobrą specyfikację urządzenia, na którym owe testy były uruchamiane. Operacji mnożenia i dzielenia nie udało mi się już zaimplementować, mimo początkowej pracy przy projekcie dwóch osób, projekt musiałem zakończyć sam.

Bibliografia

- [1] Operacje bitowe https://en.cppreference.com/w/cpp/language/operators?fbclid=IwAR2YRoWjFwVSoQHkL1aVkynzJT4rzN_iPoSs2gGwnvqy5RvL1MsiRzKPe8M.
- [2] Operacje bitowe w C++ https://edufinf.waw.pl/inf/alg/002_struct/0007.php
- [3] Operacje bitowe <http://www.algorytm.org/kurs-algorytmiki/operacje-bitowe.html>
- [4] Extended precision https://en.wikipedia.org/wiki/Extended_precision
- [5] IEEE https://edufinf.waw.pl/inf/alg/006_bin/0022.php
- [6] Basic Computer Architecture