RAPORT TECHNICZNY

Implementacja biblioteki arytmetyki liczb stałoprzecinkowych dowolnej precyzji z wykorzystaniem wewnętrznej reprezentacji U2.

Kamil Pawelski

Adam Troszczyński

Spis treści

1. Cel Projektu
2. Założenia i ograniczenia
3. Technologie  
   3.1. Zastosowane języki  
   3.2. Oprogramowanie
4. Realizacja projektu
5. Opis kodu  
   5.1. Reprezentacja liczby  
   5.2. Konstruktor  
   5.3. Wypisywanie liczby  
   5.4. Ogólne zasady operacji
6. Testy poprawności
7. Badania wydajnościowe
8. Napotkane problemy

1. Cel Projektu

Celem projektu była implementacja biblioteki arytmetyki liczb stałoprzecinkowych dowolnej precyzji z wykorzystaniem wewnętrznej reprezentacji U2. Biblioteka miała za zadanie wykonywać podstawowe operacje arytmetyczne jak dodawanie, odejmowanie, mnożenie i dzielenie. Operacje miały wykorzystywać wspomaganie procesora. Następnie przeprowadzone zostały testy, aby sprawdzić wydajność naszego rozwiązania.

2. Założenia i ograniczenia

Ograniczeniami jakie zostały nam narzucone był wybór języka oraz, aby nasze operacje korzystały z instrukcji procesora. Z języków programowania jakie mogliśmy wybrać aby wykonać projekt były:  
- Język asemblera  
- C  
- C++  
Założyliśmy, że projekt wykonamy w C++ jak i w języku asemblera, który zapewni nam od razu wsparcie w wykonaniu danych operacji arytmetycznych.

3. Technologie

3.1. Zastosowane języki

Nasza biblioteka jak zostało napisane w założeniach została napisana w C++ z połączeniem języka asemblera (powód wyboru podany w założeniach). Zdecydowaliśmy się C++, ponieważ był nam już znany język z poprzednich kursów odbywanych na studiach. W porównaniu do potencjalnego innego wyboru, czyli C byśmy musieli sami zarządzać dynamicznie pamięcią, natomiast w C++ mogliśmy skorzystać z gotowych szablonów z biblioteki STL więc wiele potencjalnych problemów, które mogły by wystąpić w C zostało w ten sposób automatycznie zażegnane. Biblioteka została napisana na architekturę 32 bitową.

3.2. Oprogramowania

Nasza biblioteka została napisana na dystrybucji linuxa Ubuntu. Testy jednostkowe zostały napisane własnoręcznie. Kompilacja do programu wykonywalnego odbywa się poprzez kompilator GCC.

4. Realizacja projektu

Na początku mieliśmy duże problemy. Źle rozumieliśmy jak dokładnie miała działać nasza biblioteka przez co nasze początkowe próby pisania programu stały w miejscu. Zaczęliśmy od implementacji wyglądu naszej liczby (reprezentacja naszej liczby zostanie opisana w następnym punkcie). Skupiliśmy się początkowo na konstruktorze, aby nasza reprezentacja liczby była dla nas na tyle prosta, aby potem nie było problemu stosowaniem jej w operacjach. Kolejne było metoda wypisująca naszą liczbę w U2, aby umożliwić późniejsze testowanie. Po implementacji wypisania zaczęliśmy wykonywać działania arytmetycznie kolejno dodawanie, odejmowanie, mnożenie i dzielenie co sprawiło wiele problemów (opisane zostanie w późniejszych punktach). Po realizacji wszystkich operacji mogliśmy już się skupić na wykonywaniu badań wydajności.

5. Opis kodu

5.1 Reprezentacja liczby

Została utworzona klasa o nazwie TC w której znajdują się dwa pola, które tworzą całość naszej liczb.

**vector<uint8\_t> \_number** - Kontener przechowujący liczbę reprezentowaną w systemie U2. Pod każdym indeksem w kontenerze znajduję się 8-bitowa część liczby.

**int \_position** – Określa pozycję najmniej znaczącego bitu liczby.

5.2. Konstruktor

Konstruktor, nie tylko zapisuje podaną przez użytkownika liczbę i pozycję, ale również przekształca liczbę w kontenerze. Przyjęliśmy sobie konkretne przedziały, które mogą się znajdować pod indeksem w kontenerze. Dodatkowe operacje przesunięcia są wykonywane gdy podana przez użytkownika jest inna od ostatniej w danym przedziale.

(Pierwsza pozycja w danym przedziale, Ostatnia pozycja w danym przedziale)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| … | (15, 8) | (7, 0) | (-1, -8) | … |

Czyli gdy użytkownik poda nam jakąś liczbę 8-bitową i poda pozycję 0 konstruktor nie wykonana dodatkowych operacji i zapiszę dokładnie tak samo jak zostało podane, będzie ona zapisana w jednym indeksie kontenera.

|  |  |  |
| --- | --- | --- |
|  | Pozycja | Liczba w kontenerze |
| Użytkownik | 0 | 10001001 |
| Konstruktor | 0 | 10001001 |

Natomiast gdy użytkownik dla takiej samej liczby 8-bitowej poda pozycję 3 wtedy dla takiej liczby przedział by wyglądał (10,3) co nie pasuje dla naszego zamysłu, więc konstruktor zacznie wykonywać przesunięcia, aby nasza liczba zaczynała się od zadanych przedziałów. W tym wypadku liczba zostanie wypisana na dwa indeksy w naszym kontenerze (15,8) i (7,0).

|  |  |  |
| --- | --- | --- |
|  | Pozycja | Liczba w kontenerze |
| Użytkownik | 3 | 10001001 (czyli tak naprawdę 10001001000) |
| Konstruktor | 0 | 11111000 1001000 (zapis w \_number po operacjach w konstruktorze) |

5.3. Wypisywanie liczby

Liczba jest zapisywana bit po bicie przy pomocy operacji logicznych do stringa. Dla konkretnych warunków wykonuje ona dodatkowe operacje (głównie dopisanie 0 lub przecinka).

5.4. Ogólne zasady operacji

Podczas wykonywania operacji arytmetycznych napotkaliśmy wiele problemów (zostanie opisane w późniejszych punktach). Początek każdego z działań jest praktycznie identyczny. Zaczynamy od pozbycia się przecinka jeśli on występuję, poprzez przesunięcie obydwu liczb.

5.5. Dodawanie i Odejmowanie

W dodawaniu i odejmowaniu zaczynamy od utworzenia nowego kontenera, w którym znajdzie się wynik naszego działania. Pozycja w wyniku będzie przyjmować mniejszą pozycję z dwóch podanych liczb. Dodatkowa zmienna oznaczająca pozycję bitu na najstarszej pozycji będzie przyjmować większą pozycję najstarszego bitu obu liczb i mając dwie te zmienne wyliczamy rozmiar dla kontenera wynikowego. Wyliczane są indeksy, aby wiedzieć od którego miejsca w kontenerze mamy dodawać/odejmować. Działania wykonują się na liczbach dodatnich, w wypadku gdy mamy ujemną zamieniamy ją sobie na dodatnią. Następnie te liczby są przekazywane w zależności jakiego znaku były do funkcji dodawania/odejmowania napisanych w asemblerze. Wynik zapisywany jest następnie w U2.

5.6. Mnożenie

W mnożeniu rozmiar wyliczany jest na podstawie dodania rozmiaru dwóch liczb. Przy pomocy dwóch pętli obliczamy kolejne iloczyny częściowe, które następnie dodajmy do naszego kontenera wynikowego.

5.7. Dzielenie

Dzielenie zostało wykonane algorytmem dzielenia nieodtwarzającego. Rozszerzamy tak aby dzielna była na pewno o 8 bitów większa od dzielnika. I rozpoczynamy algorytm dzielenia nieodtwarzającego. Bierzemy do nowego kontenera część dzielnej o rozmiarze dzielnika, a następnie po wykonaniu potencjalnego dodawania/odejmowania będziemy przesuwać nasz nowy kontener o 1 bit w lewo i tak, aż do ostatniego bitu dzielnej. Obecnie tylko dla dodatnich i podzielnych przez siebie.

6. Testy poprawności

7. Badania wydajnościowe

8. Napotkane problemy

Pierwszym problemem był problem z pozycją, jeśli chodzi o przekazywanie dalej naszej liczby wykonywaliśmy za każdym razem specjalnie przesunięcia, aby uzyskać łatwiejszy zapis do działania już na konkretnej liczbie. Problem ten naprawiliśmy zmieniając koncept zapisywanie liczby, który został opisany w konstruktorze. Kolejnym problemem było wykonywanie działań, które często w wypadku ujemnych liczb dawało nieprawidłowe wyniki. Postanowiliśmy przekazywać ich dodatnie odpowiedniki i wykonywać konkretnie odejmowanie/dodawanie w zależności jakich znaków były liczby. Poprawiło to znacznie nasze rezultaty. Ostatnim z takich większych problemów było wykonywanie działań liczb z przecinkiem. Po raz kolejny wyniki wychodziły inne od oczekiwanych. Postanowiliśmy wykonać operację przesunięcia, aby pozbyć się przecinka co spowodowało, że nasza liczba wychodzi zgodnie z oczekiwaną. Problem mieliśmy również z gotowymi bibliotekami do testów. Te, które chcieliśmy stosować wymagały kompilacji 64-bitowej, a nasz kod jest dla architektury   
32-bitowej. Postanowiliśmy napisać własną bibliotekę testów dla naszej biblioteki. Główne problemy naprawiliśmy, ale zwykle naprawa ich kosztuje nas wydajność w wykonywaniu metod. Natomiast dzięki temu znacznie poprawiliśmy ich skuteczność. Duży problem mieliśmy z dzieleniem. Nie potrafiliśmy nawet dobrze do tego podejść. Dzięki pomocy na zajęciach, że możemy bezproblemowo rozszerzyć naszą liczbę, udało się zaimplementować dzielenie algorytmem nieodtwarzającym.

9. Podsumowanie

10. Wnioski

11. Nasze założenia i potencjalne zmiany

12. Kompilacja

13. Podział Pracy