TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Đồ án 1
Kiến trúc máy tính và hợp ngữ

BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH

Học kỳ II
Năm học: 2017- 2018

LỜI CẢM ƠN

Trong quá trình thực hiện đồ án đề tài, nhóm em đã nhận được rất nhiều sự giúp đỡ, hỗ trợ từ các thầy cô Trường Đại học Khoa học Tự nhiên – ĐHQG TP.HCM và các bạn bè trong trường. Nhóm em xin bày tỏ lòng cảm ơn chân thành về sự hướng dẫn, chỉ bảo của mọi người.

Đặc biệt, nhóm em xin bày tỏ lòng biết ơn sâu sắc đến các thầy cô khoa Công nghệ thông tin, cụ thể hơn chúng em xin cảm ơn thầy Phạm Tuấn Sơn đã hỗ trợ, giảng dạy rất kĩ lưỡng từng phần nhỏ để chúng em có một đồ án thật hoàn chỉnh và phù hợp nhất. Chính những thứ tưởng chừng như nhỏ nhặt này đã góp phần to lớn giúp chúng em hoàn thành đồ án và bảng báo cáo này thật hoàn chỉnh nhất. Một lần nữa, chúng em xin bày tỏ lòng biết ơn sâu sắc đến với các thầy cô và bạn bè.

Ngoài ra, nhóm chúng em còn nhận được rất nhiều sự khích lệ tinh thần, động viên cổ vũ không chỉ từ các bạn đồng trang lứa mà còn đến từ các anh chị khoa Công nghệ thông tin khóa trước và các thầy cô.

Chúng em xin chân thành cảm ơn!

Mục lục

1 CHƯƠNG I: MỞ ĐẦU

1.1 Giới thiệu nhóm và phân công công việc

- Số thành viên: 3 người

STT	MSSV	Họ và tên
1	1612840	Dương Nguyễn Thái Bảo
2		
3	1612899	Hoàng Xuân Trường

1.2 Mô tả đồ án

- Biểu diễn và tính toán số học trên máy tính.
  - Thiết kế kiểu dữ liệu biểu diễn có số nguyên lớn có dấu 16 bytes (128bit).
  - Thiết kế kiểu dữ liệu biểu diễn số chấm động có độ chính xác Quadruple-precision (độ chính xác gấp 4 lần) độ lớn 128bit.

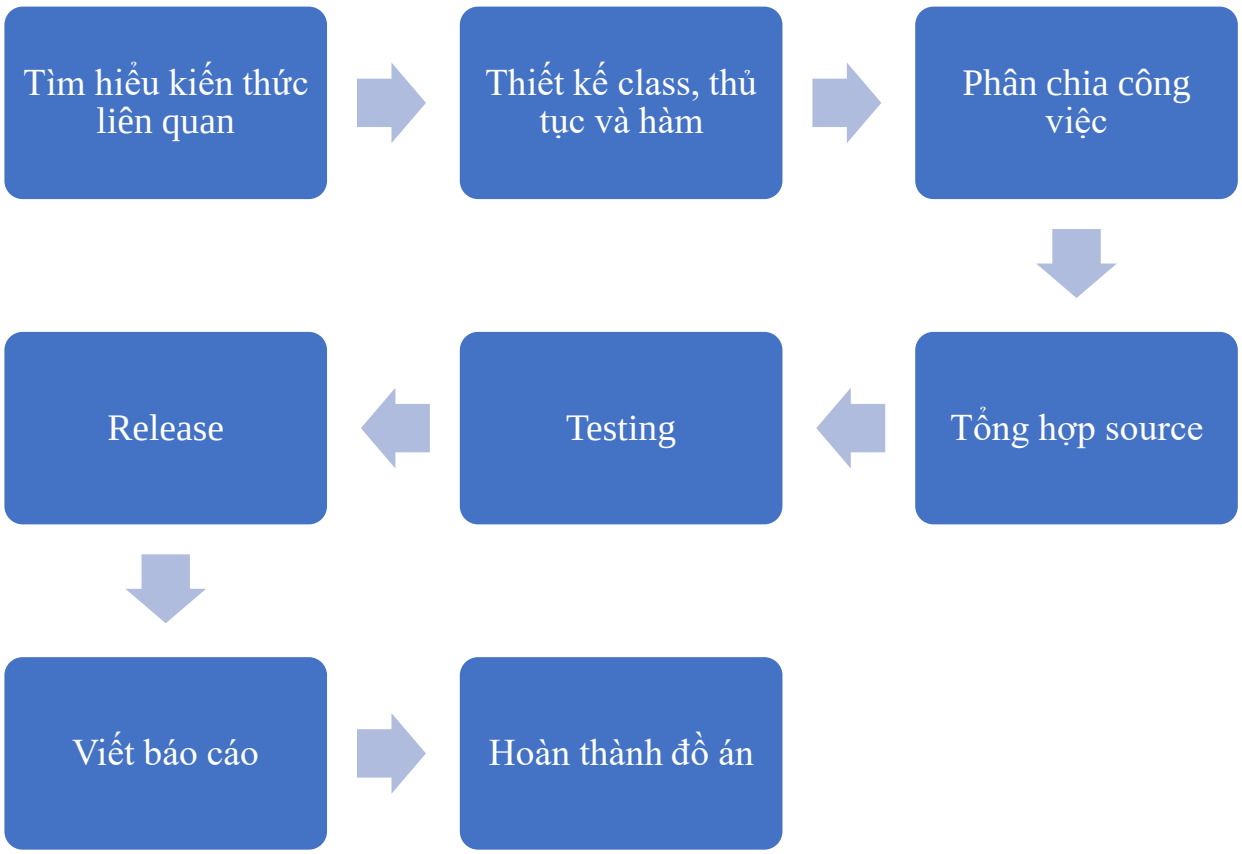
- Viết chương trình calculator đơn giản gồm các chức năng sau:
  - Chuyển đổi giá trị qua lại giữa các hệ 2,10,16 (số chấm động không chuyển đổi qua hệ 16)
  - Tính toán cộng, trừ, nhân, chia giữa 2 số bất kỳ.

2 CHƯƠNG II: GIỚI THIỆU ĐỒ ÁN

2.1 Thời gian thực hiện

- Đồ án bắt đầu từ ngày 10/3 đến ngày 31/3

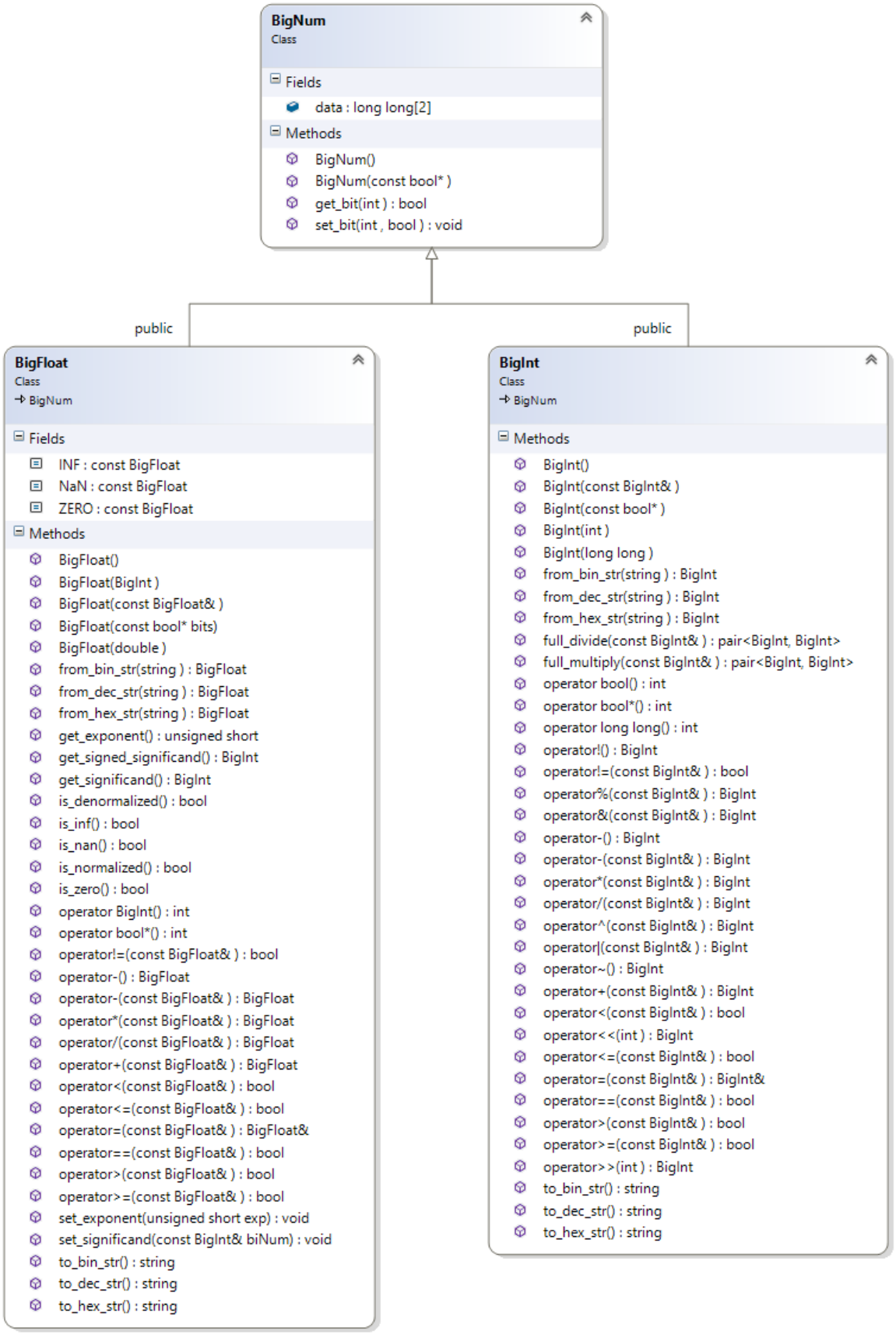
2.2 Các bước thực hiện đồ án



3 CHƯƠNG III: NỘI DUNG ĐỒ ÁN

3.1 Sơ đồ UML

- Sơ đồ UML thể hiện thiết kế class


BigNum
Class
Fields
data : long long[2]
Methods
BigNum()
BigNum(const bool*)
get_bit(int) : bool
set_bit(int , bool) : void
public
public
BigFloat
Class
BigNum
Fields
INF : const BigFloat
NaN : const BigFloat
ZERO : const BigFloat
Methods
BigFloat()
BigFloat(BigInt)
BigFloat(const BigFloat&)
BigFloat(const bool* bits)
BigFloat(double)
from_bin_str(string) : BigFloat
from_dec_str(string) : BigFloat
from_hex_str(string) : BigFloat
get_exponent() : unsigned short
get_signed_significand() : BigInt
get_significand() : BigInt
is_denormalized() : bool
is_inf() : bool
is_nan() : bool
is_normalized() : bool
is_zero() : bool
operator BigInt() : int
operator bool*() : int
operator!=(const BigFloat&) : bool
operator-() : BigFloat
operator-(const BigFloat&) : BigFloat
operator*(const BigFloat&) : BigFloat
operator/(const BigFloat&) : BigFloat
operator+(const BigFloat&) : BigFloat
operator<(const BigFloat&) : bool
operator<=(const BigFloat&) : bool
operator=(const BigFloat&) : BigFloat&
operator==(const BigFloat&) : bool
operator>(const BigFloat&) : bool
operator>=(const BigFloat&) : bool
set_exponent(unsigned short exp) : void
set_significand(const BigInt& biNum) : void
to_bin_str() : string
to_dec_str() : string
to_hex_str() : string
BigInt
Class
BigNum
Methods
BigInt()
BigInt(const BigInt&)
BigInt(const bool*)
BigInt(int)
BigInt(long long)
from_bin_str(string) : BigInt
from_dec_str(string) : BigInt
from_hex_str(string) : BigInt
full_divide(const BigInt&) : pair<BigInt, BigInt>
full_multiply(const BigInt&) : pair<BigInt, BigInt>
operator bool() : int
operator bool*() : int
operator long long() : int
operator!() : BigInt
operator!=(const BigInt&) : bool
operator%(const BigInt&) : BigInt
operator&(const BigInt&) : BigInt
operator-() : BigInt
operator-(const BigInt&) : BigInt
operator*(const BigInt&) : BigInt
operator/(const BigInt&) : BigInt
operator^(const BigInt&) : BigInt
operator|(const BigInt&) : BigInt
operator~() : BigInt
operator+(const BigInt&) : BigInt
operator<(const BigInt&) : bool
operator<<(int) : BigInt
operator<=(const BigInt&) : bool
operator=(const BigInt&) : BigInt&
operator==(const BigInt&) : bool
operator>(const BigInt&) : bool
operator>=(const BigInt&) : bool
operator>>(int) : BigInt
to_bin_str() : string
to_dec_str() : string
to_hex_str() : string

3.2 Lớp BigNum

- Là lớp số cơ sở của 2 lớp số nguyên lớn và số thực lớn.
- BigNum gồm 2 phần tử kiểu long long để tạo thành 128 bit dữ liệu (kích thước long long là 64 bit).
- Lớp BigNum hỗ trợ 2 phương thức quan trọng là get_bit() - dùng để lấy giá trị 1 trong 128 bit, set_bit() - dùng để sửa giá trị của 1 bit.

3.2.1 Hàm get_bit(int index, bool value)

Thuật toán:

- Xác định được phần tử data cần thay đổi là index / 64, tức là index >> 6.
- Xác định bit cần thay đổi của data[index >> 6] là index % 64, tức là index & 63.
- Như vậy, nếu value = true thì ta sẽ áp dụng phép or giữa data[index >> 6] với dãy bit toàn 0, chỉ trừ bit thứ index & 63 là bật, tức là 1 << (index & 63).
- Ngược lại nếu value = false thì or giữa data[index >> 6] với dãy bit toàn 1, chỉ trừ bit thứ (index & 63) là 0, tức là phần bù của 1 << (index & 63), hay ~(1 << (index & 63)).

3.2.2 Hàm get_bit(int)

Thuật toán:

- Tương tự như get_bit, ta xác định được data cần xét là index >> 6, và vị trí bit cần lấy là (index & 63).
- Để lấy được giá trị bit này, ta lấy data[index >> 6] dịch phải (index & 63) bit, lúc này bit cần xét nằm ở vị trí 0 nên chỉ cần and với 1 là xác định được kết quả.

3.3 Lớp BigInt

- Lớp BigInt dùng để lưu giá trị của một số nguyên lớn có dấu có 128 bit.
- Lớp BigInt được kế thừa từ lớp cha BigNum với mục đích sử dụng lại các hàm ví dụ như get_bit(), set_bit() để dễ dàng hơn trong các thao tác xử lý.
- Phạm vi biểu diễn số trong lớp BigInt:
  - Từ -2^{127} đến $2^{127} - 1$.
  - Có tối đa 39 chữ số thập phân.

3.3.1 Chuyển giữa hệ thập phân và nhị phân

Từ hệ nhị phân sang chuỗi thập phân: Dựa vào nhận xét số BigInt có tối đa 39 chữ số thập phân nên chia 39 chữ số này thành 3 phần 13 chữ số và lưu

bằng 3 biến long long, sử dụng các phép / và %: Gọi p là số cần chuyển (kiểu BigInt):

- 13 chữ số nhỏ nhất là p % 10^{13}.
- 13 chữ số tiếp theo là (p / 10^{13}) % 10^{13}.
- 13 chữ số lớn nhất là p / 10^{13} / 10^{13}.

Từ chuỗi thập phân sang nhị phân: Ta làm ngược lại ở trên, chuyển từng 13 chữ số thập phân thành 3 số long long rồi nối lại bằng phép * và +.

3.3.2 Chuyển giữa hệ nhị phân sang hệ thập lục phân

Gom từng nhóm 4 bit của biểu diễn nhị phân để biểu diễn một chữ số thập lục phân, và ngược lại.

3.3.3 Toán tử cộng

- Dựa trên Full adder, ta chạy từng bit từ trái sang phải trên data[2]. Trong mỗi lần lặp, ta cần tính tổng của 3 bit: 2bit của hai số BigInt cho trước và 1bit của *Carry*. Biến *Carry* được tính bằng cách OR của tất cả các cặp. Thuật toán cụ thể:

```
1. Sum = a XOR b XOR carry; //a và b là hai bit đang xét của hai
   số BigInt
2. carry = (a & b) | (a & c) | (b & c);
```

3.3.4 Toán tử trừ

- Duyệt từng bit của số BigInt, ta thực hiện phép tính trừ trên hai bit của hai số BigInt và số mượn *borrow*. Mô tả cách thực hiện:

```
1. Kết quả = (a - b - borrow) and 1;
2. If(kết quả < 0) thì borrow = 1 ngược lại borrow = 0;
```

3.3.5 Toán tử nhân

Sử dụng thuật toán Booth.

Trong cài đặt có 2 hàm nhân:

- Hàm full_multiply (nhân đầy đủ): trả về đủ 256 bit của kết quả, biểu diễn bằng 2 biến BigInt (1 biến lưu 128 bit cao và 1 biến lưu 128 bit thấp của kết quả). Cài đặt hàm này để ứng dụng trong thuật toán nhân của số thực lớn.
- Toán tử nhân: là 128 byte thấp từ hàm full_multiply.

3.3.6 Toán tử chia

Sử dụng thuật toán chia không dấu trong slide bài giảng Số chấm động - thầy Phạm Tuấn Sơn - Khoa Công nghệ Thông tin trường Đại học Khoa học Tự nhiên TP.HCM.
Trước khi nhân, đưa 2 toán hạng về số dương.
Trong cài đặt cũng có 2 phép chia:

- Hàm full_divide (chia đầy đủ): trả về phần dư và phần nguyên của kết quả chia.
- Toán tử chia: là phần nguyên từ hàm full_divide.
- Ngoài ra còn có toán tử chia lấy phần dư, đây là phần dư từ hàm full_divide.

3.3.7 Phép xử lý bit

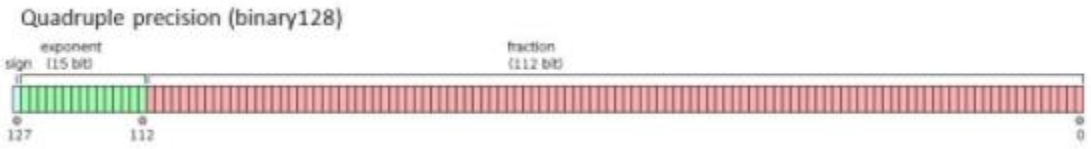
Các phép xử lý bit sử dụng các toán tử xử lý bit có sẵn &, |, ^, ~ trên từng phần tử data.

3.3.8 Phép so sánh

- So sánh bằng: 2 số BigInt bằng nhau khi data tương ứng bằng nhau.
- So sánh khác: dựa trên so sánh bằng.
- So sánh bé hơn: xét bit đầu tiên khác nhau (từ cao xuống). Nếu bit đó là cao nhất (127) thì số nào có bit 127 là 1 thì nhỏ hơn. Ngược lại 0 thì nhỏ hơn.
- So sánh lớn hơn: tương tự so sánh bé hơn.
- So sánh bé hơn hoặc bằng và lớn hơn hoặc bằng: tận dụng so sánh lớn hơn và bé hơn.

3.4 Lớp BigFloat

- Lớp BigFloat là kiểu dữ liệu biểu diễn số chấm động có độ chính xác Quadruple-precision độ lớn 128bit.
- Lớp BigFloat được kế thừa từ lớp BigNum, có quy ước cấu trúc như sau (chuẩn IEEE 754-2008): bit 127 là bit dấu, bit 112 tới 126 là 15 bit mũ (số bias), bit 0 tới 111 là 112 bit phần trị.



- Phạm vi biểu diễn của lớp BigFloat:

Số	Dấu	Mũ	Trị
Chuẩn max (+)	0	111111111111110 (= 16383)	1111...111 (= $1 - 2^{-112}$)

Chuẩn min (+)	0	000000000000001 (=-16384)	0000...0000 (= 0)
Chuẩn min (-)	1	111111111111110 (= 16383)	1111...111 (= $1 - 2^{-112}$)
Chuẩn max (-)	1	000000000000001 (=-16384)	0000...0000 (= 0)
Không chuẩn max (+)	0	000000000000000 (quy ước là -16384)	11111...1111 (1 - 2^{-112})
Không chuẩn min (-)	0	000000000000000 (quy ước là -16384)	0000….0001 (= 2^{-112})
Không chuẩn min (-)	1	000000000000000 (quy ước là -16384)	11111...1111 (1 - 2^{-112})
Không chuẩn max (-)	1	000000000000000 (quy ước là -16384)	0000….0001 (= 2^{-112})
Số vô cực	0	111111111111111	0
Số 0	0	000000000000000	0
Số NaN		111111111111111	Khác 0

3.4.1 Chuyển giữa hệ thập phân và nhị phân

Từ nhị phân sang thập phân:

- Cài đặt phương thức ép kiểu BigFloat sang BigInt để làm tròn.
- Liên tục nhân số cần chuyển với 10, lấy phần nguyên đưa vào kết quả và loại bỏ phần nguyên này, cho đến khi số này trở thành 0.

Từ thập phân sang nhị phân: Tách phần trị ra riêng, chuyển sang nhị phân rồi nhân 10 hoặc chia 10 cho đến khi phần mũ thập phân về 0.

3.4.2 Hàm chuyển từ hệ nhị phân sang hệ thập lục phân

Gom 4 bit nhị phân thành 1 chữ số thập lục phân, và ngược lại.

3.4.3 Toán tử cộng

- Cài đặt hàm get_signed_significand: dùng để lấy phần trị, kết hợp với số 1 (nếu là số chuẩn), lưu ở dạng số có dấu BigInt. Ví dụ: 1.11 => 111000...000 (113 bit).

- Lấy signed_significand của 2 số, đưa mũ về bằng nhau, đồng thời dịch trái phải tương ứng cho phần signed significand.
- Sau đó cộng 2 phần đó lại với nhau.
- Chuẩn hóa kết quả bằng cách đưa mũ về khoảng cho phép, kết hợp dịch trái dịch phải tương ứng với phần signed significand.
  - Dùng hàm set_exponent và set_significand để khởi tạo kết quả.

3.4.4 Toán tử trừ

- Gọi lại thuật toán cộng.

3.4.5 Toán tử nhân

- Lấy phần signed_significand, cộng 2 phần mũ với nhau.
- Nhân 2 phần signed_significand bằng hàm full_multiply của lớp BigInt, sau đó lấy các bit cao nhất (kể từ bit 1 đầu tiên của kết quả này).

3.4.6 Toán tử chia

- Lấy phần signed_significand, cộng 2 phần mũ với nhau.
- Liên tục dịch trái signed_significand của số bị chia rồi so sánh và trừ signed_significand của số chia. Kết quả lấy những bit cao nhất (kể từ lần đầu signed_significand của số bị chia lớn hơn) của kết quả.

3.4.7 Phép so sánh

Lấy phần signed_significand, đưa 2 số về cùng số mũ rồi so sánh trên phần siged_significand sau khi chuẩn hóa.

4 CHƯƠNG V: ĐÁNH GIÁ VÀ TỔNG KẾT QUÁ TRÌNH

4.1 Đánh giá hoàn thành đồ án

Tên lớp	Yêu cầu	Mức độ hoàn thành
BigInt	Hàm nhập	100%
	Hàm xuất	100%
	Chuyển đổi từ hệ thập phân sang hệ nhị phân	100%
	Chuyển đổi từ hệ nhị phân sang hệ thập phân	100%

	Chuyển đổi từ nhị phân sang hệ thập lục phân			100%
	Chuyển đổi từ thập phân sang thập lục phân			100%
	Các phép tính	Cộng		100%
		Trừ		100%
		Nhân		100%
		Chia	Lấy phần nguyên	100%
			Lấy phần dư	100%
BigFloat	Hàm nhập			100%
	Hàm xuất			100%
	Chuyển đổi từ hệ thập phân sang hệ nhị phân			100%
	Chuyển đổi từ hệ nhị phân sang hệ thập phân			100%
	Các phép tính	Cộng		100%
		Trừ		100%
		Nhân		100%
		Chia		100%
				Đánh giá tổng quát: 100%

4.2 Giao diện chương trình với testcase

4.3 Testcase BigInt

4.4 Testcase BigFloat

5 TÀI LIỆU THAM KHẢO

https://en.wikipedia.org/wiki/Adder_%28electronics%29#Full_adder